

Autores | Guillermo Choque Aspiazu
Ramiro Loza
Rances Mendez

Redes Neuronales Artificiales con MatLab



**LA PAZ, BOLIVIA
DICIEMBRE DE 2002**

Redes Neuronales Artificiales con MatLab©

MSc. Guillermo Choque Aspiazu.

Docente de la Carrera de Informática de la Universidad Mayor de San
Andrés.

Presidente de la Asociación de Investigación en Software Inteligente.

Lic. Ramiro Loza Herrera.

Responsable de Redes Neuronales y Computación Masivamente
Paralela de la Asociación de Investigación en Software Inteligente.

Lic. Rances Méndez Quintanilla.

Vicepresidente del área de Inteligencia Artificial de la Asociación de
Investigación en Software Inteligente.

© A.I.S.I. 2006.

Asociación de Investigación en Software Inteligente.

La Paz, Bolivia.

Datos de catalogación bibliográfica

MSc. Guillermo Choque Aspiazu
Lic. Ramiro Loza Herrera
Lic. Rances Méndez Quintanilla

Redes Neuronales Artificiales con MatLab
Asociación de Investigación en Software
Inteligente
La Paz, Bolivia; 2006

ISBN: xx-xxxx-xxx-x

Área: Inteligencia Artificial

Formato: ???x???

Páginas: ???

Redes Neuronales Artificiales con MatLab

© Guillermo Choque Aspiazu, Ramiro Loza Herrera y Rances Méndez Quintanilla

© Asociación de Investigación en Software Inteligente, A.I.S.I.

ISBN xx-xxxx-xxx-x, primera edición

La Paz, Bolivia. Derechos reservados © A.I.S.I., 2006

Marcas Comerciales: Se ha intentado a lo largo de este libro distinguir las marcas registradas de los términos descriptivos, siguiendo el estilo de mayúsculas que usa el fabricante, sin intención de infringir la marca y sólo en beneficio del propietario de la misma.

Derechos Reservados: No se permite la reproducción total o parcial de este libro, ni su tratamiento informático, ni la transmisión en cualquier forma o cualquier medio, electrónico, mecánico, fotocopia, registro u otros métodos sin el permiso previo y por escrito de los autores.

Impreso por: Imprenta ABC XYZ

IMPRESO EN BOLIVIA – PRINTED IN BOLIVIA

Estimado lector:

El libro que usted tiene en sus manos posee un gran valor, ya que sus autores, han vertido conocimientos, experiencia y mucho trabajo, además han procurado una presentación digna de su contenido.

Usted puede obtener fotocopias de las páginas del libro para su uso personal, pero desconfíe y rehusé cualquier ejemplar “pirata” o fotocopia ilegal del mismo porque, de lo contrario, contribuirá al lucro de quienes, conscientemente o inconscientemente, se aprovechan de manera ilegítima del esfuerzo de los autores.

La reprografía indiscriminada y la piratería, no solamente son prácticas ilegales, sino que atentan contra la creatividad y contra la difusión de la cultura.

**PROMUEVA LA CREATIVIDAD
RESPETE LOS DERECHOS DE AUTOR**

DEDICATORIA

Este esfuerzo está dedicado a todos los socios de la Asociación de Investigación en Software Inteligente, para mostrar a la comunidad científica y académica que es posible unir conocimiento teórico y el ejercicio práctico en aras del avance del software inteligente y la ciencia de las computadoras.

*Guillermo Choque Aspiazu,
Ramiro Loza Herrera,
Rances Méndez Quintanilla.*

Para toda mi querida familia y en especial para mis hermanas Susi y Mayra, quienes me brindan mucha alegría y apoyo para seguir surgiendo en el camino de la investigación.

Ramiro Loza Herrera

Dedicada a mí querida madre Janeth

Rances Méndez Quintanilla

PREFACIO

Hace más de una década que el Grupo de Inteligencia Artificial y Sistemas Expertos (GIASE) proporcionaba las bases de la investigación en inteligencia artificial en la Carrera de Informática, esfuerzo que aguantó algunos meses y languideciente cerró su paso a los investigadores para replegarse al limbo junto con sus creadores. Quizá lo que faltó en su momento fue la voluntad y dinamismo de sus integrantes para llevar a mejores derroteros a esta sobresaliente agrupación.

Enrumbado en los primeros años del siglo XXI, espacio en el cual las nuevas tecnologías y los proyectos computacionales asociados con la inteligencia artificial aparentan plantear un cambio de rumbo en la investigación de base enfatizándola más hacia la sociedad y sus inevitables mutaciones sociales emerge, como respuesta natural a la falta de incentivos a la investigación, una asociación de estudiantes y docentes agrupadas bajo un común denominador: la investigación productiva. El 5 de mayo de 2006 se funda la Asociación de Investigación en Software Inteligente con el auspicio de los esfuerzos individuales de 6 licenciados en informática y aproximadamente 24 estudiantes.

La Asociación de Investigación en Software Inteligente organizó con bastante éxito los primeros simposios en inteligencia artificial e ingeniería del software, durante los meses de junio y septiembre del año 2006. El taller destacado en el simposio sobre inteligencia artificial fue el referido a las redes neuronales. El paradigma conexionista, de proceso distribuido paralelo o simplemente el identificado como redes neuronales, constituye precisamente uno de los pilares de la escuela conexionista de la inteligencia artificial, muchísimas aplicaciones actuales utilizan este paradigma como elemento central en sus propuestas de solución, especialmente en los problemas que precisan elementos de clasificación, selección, reconocimiento de patrones y predicción.

La propuesta teórica normalmente queda pequeña si no es complementada con su referente práctico que, a manera de producto software, constituye el insumo requerido para mostrar la eficiente aplicación de los principios y fundamentos de la escuela conexionista. Es así que se presenta el laboratorio matricial denominado MatLab para convertir la abundante y rica teoría de las redes neuronales en productos palpables a través de la aplicación de un modelo de proceso adecuado.

El texto comprende los siguientes capítulos: el capítulo 1 contiene una introducción detallada de las bases fundamentales asociadas a las redes neuronales; el capítulo 2 es una vista de la metodología y las principales topologías utilizadas al interior del estudio sobre redes neuronales; el capítulo 3 presenta el aprendizaje en las redes neuronales; el capítulo 4 comprende uno de los modelos más interesantes de las redes neuronales como es el perceptrón; el capítulo 5 presenta el algoritmo de retropropagación; el capítulo 6 está referido a los filtros adaptativos lineales como son el adaline y el madaline; el capítulo 7 abarca los modelos de redes neuronales recurrentes; el capítulo 8 se refiere a los mapas autoorganizados de Kohonen y finalmente el capítulo 9 involucra un estudio completo de caso, específicamente el reconocimiento de patrones a través de una red neuronal de Hopfield.

Agradezco nuevamente la confianza que depositan los integrantes de la Asociación de Investigación en Software Inteligente para hacer que los esfuerzos que se emprenden, en pos de mejorar la difícil situación académica y económica en la que se encuentran muchas personas del medio, constituyan incentivos para no desmayar en el camino y comprender que aún resta mucho por hacer.

Guillermo Choque Aspiazu
Octubre 2006

AGRADECIMIENTOS

Queremos agradecer a la cuna formadora de nuestros conocimientos que es la magnífica Universidad Mayor de San Andrés, a los docentes y estudiantes de la Carrera de Informática que en una acción digna de resaltar conformaron la Asociación de Investigación en Software Inteligente para la investigación productiva en inteligencia artificial e ingeniería del software de las siguientes generaciones.

*Guillermo Choque Aspiazu,
Ramiro Loza Herrera,
Rances Méndez Quintanilla.*

Deseo agradecer la ayuda y apoyo que recibí de mi familia, en especial de mi madre, y de cada uno de los socios de la A.I.S.I. El contenido de varios capítulos son la recopilación de información tomada de artículos, libros y otro tipo de documentos escritos por otros autores sin cuyo talento y conocimiento en el área de la Inteligencia Artificial no hubiese sido posible publicar este libro. Para terminar, quiero agradecer profundamente a MSc. Guillermo Choque Aspiazu y a Lic. Ramiro Loza por todo el esfuerzo y cúmulo de conocimiento vertido en esta publicación.

Rances Méndez Quintanilla

RESUMEN

Las redes neuronales artificiales son redes interconectadas masivamente en paralelo de elementos simples denominados neuronas, las cuales cuentan con organización jerárquica, intentando interactuar con los objetos del mundo real del mismo modo que lo hace el sistema nervioso biológico en tareas cognoscitivas como reconocer un rostro familiar, hablar, comprender el lenguaje y recuperar contextualmente información apropiada desde la memoria. Como en la naturaleza, la función de la red está determinada principalmente por la conexión entre elementos, se puede entrenar una red neuronal para que realice una función particular mediante el ajuste de los valores de las conexiones, denominadas pesos, entre dichos elementos y la elección de una función de transferencia que produzca el resultado deseado. Dichas redes son aplicadas principalmente a problemas de clasificación, reconocimiento de patrones y predicción. El presente libro trata del paradigma conexionista de la inteligencia artificial, llamado redes neuronales artificiales, la principal característica de una red neuronal es la capacidad de aprender, realizando tareas para las cuales todavía no existen algoritmos, o para las cuales es virtualmente imposible escribir una serie de pasos lógicos o aritméticos que proporcionen la solución adecuada a una tarea determinada.

En este libro encontrara la base necesaria para emprender proyectos relacionados con la Inteligencia Artificial, específicamente con la fascinante área de las redes neuronales, y su aplicación en el reconocimiento de patrones. En los capítulos que componen la presente publicación, se intenta que el lector a través de cada uno de los mismos vaya adquiriendo las bases teóricas y practicas necesarias. El presente libro realiza una introducción a las redes neuronales, para luego ir profundizando en temas tales como el aprendizaje, modelos de redes neuronales artificiales, filtros adaptativos lineales, los mapas auto-organizados de Kohonen y finalmente presenta un caso de estudio de reconocimiento de patrones a través de la red neuronal de Hopfield; en cada uno de estos capítulos, luego de haber desarrollado la teoría necesaria, se complementa con un ejemplo practico y ejercicios que el lector debe realizar para comprender de mejor manera el tema o

modelo estudiado, cabe hacer notar que cada uno de los ejemplos desarrollados fueron previamente analizados siguiendo el método propuesto en el segundo capítulo, para posteriormente implementarlos en programas de MatLab.

ÍNDICE DEL CONTENIDO

Página.

1	Introducción a las redes neuronales	
1.1.	Modelo biológico	
1.2.	Modelo idealizado.....	
1.3.	Definición de red neuronal	
1.4.	Elementos de una red neuronal artificial	
	Taller 1: Familiarización con MatLab	
	Ejercicios propuestos.....	
2	Método y topologías en las redes neuronales	
2.1.	Método en las redes neuronales	
2.2.	Estructura de una red neuronal	
2.3.	Características de las redes neuronales	
2.4.	Topología de las redes neuronales	
	Taller 2: Caja de herramientas para redes neuronales	
	Ejercicios propuestos.....	
3	Aprendizaje en las redes neuronales	
3.1.	Redes con aprendizaje supervisado	
3.2.	Redes con aprendizaje no supervisado	
	Taller 3: Manejo del algoritmo de aprendizaje Widrow-Hoff	
	Ejercicios propuestos.....	
4	Perceptrón	
4.1.	Modelo neuronal	
4.2.	Arquitectura	
4.3.	Regla de aprendizaje del perceptrón	
4.4.	Perceptrón multinivel	
4.5.	Limitaciones	
	Taller 4: Perceptrones	
	Ejercicios propuestos.....	
5	Retropropagación (backpropagation)	
5.1.	Aproximación al algoritmo de retropropagación	
5.2.	Operación	
5.3.	Arquitectura	

5.4. Aprendizaje por retropropagación	
5.5. Mejora de la generalización	
5.6. Consideraciones prácticas	
5.7. Limitaciones	
Taller 5: Manejo de redes de retropropagación	
Ejercicios propuestos.....	
6 Filtros adaptativos lineales, adaline y madaline	
6.1. Filtros y procesamiento digital de señales	
6.2. Modelo neuronal	
6.3. Arquitectura	
	Página.
6.4. Error cuadrático medio	
6.5. Algoritmo de aprendizaje Widrow-Hoff	
6.6. Filtrado adaptativo	
6.7. Filtros adaptativos de múltiples neuronas	
6.8. Algoritmo de entrenamiento regla madaline II	
6.9. Limitaciones	
Taller 6: Manejo de redes adaline.....	
Ejercicios propuestos.....	
7 Redes neuronales recurrentes.....	
7.1. Arquitecturas	
7.2. Aprendizaje	
7.3. Red de elman	
7.4. Red de hopfield	
7.4.5. Limitaciones	
Taller 7: Detección de amplitud	
Ejercicios propuestos.....	
8 Mapas autoorganizados de kohonen	
8.1. Arquitectura	
8.2. Funcionamiento	
8.3. Aprendizaje	
8.4. Fases en la aplicación de los mapas autoorganizativos	
Taller 8: Agrupación y clasificación de datos aleatorios	
Ejercicios propuestos.....	

9 Reconocimiento de patrones a través de la red neuronal de
hopfield.....

9.1. Descripción del sistema

9.2. Especificación del diseño

9.3. Implementación

Bibliografía

1 INTRODUCCIÓN A LAS REDES NEURONALES

Existen tareas para las cuales todavía no existen algoritmos, o para las cuales es virtualmente imposible escribir una serie de pasos lógicos o aritméticos que proporcionen la solución adecuada a una tarea determinada. Estas tareas tienen características importantes en común: (a) los humanos saben como realizarlas; (b) se puede generar grandes cantidades de ejemplos de ellas y (c) cada tarea requiere realizar una asociación entre objetos de dos conjuntos.

Los ejemplos de este tipo de tareas son las facultades cognoscitivas como reconocer un rostro familiar, hablar, comprender el lenguaje y recuperar contextualmente información apropiada desde la memoria. Estas tareas están más allá del alcance de las computadoras programadas de manera convencional, así como de los sistemas expertos tradicionales.

Como alternativa para el procesamiento automatizado de la información, se establece la neurocomputación, de manera acelerada, como una disciplina. Formalmente la neurocomputación es la disciplina relacionada con los sistemas de procesamiento de la información adaptativos no programados, que desarrollan asociaciones o transformaciones entre objetos. En lugar de un procedimiento paso a paso para ejecutar la transformación de la información, la red neuronal genera sus propias reglas internas que gobiernan la asociación y refina estas reglas comparando sus resultados con los ejemplos.

La neurocomputación es fundamentalmente un paradigma nuevo y diferente para el procesamiento de la información, la alternativa más radical de la programación algorítmica. Pero la neurocomputación aún no puede reemplazar a la programación algorítmica, por el momento la neurocomputación se complementa con la programación algorítmica [1].

Las Redes Neuronales Artificiales (RNA) o sistemas conexionistas son sistemas de procesamiento de la información cuya estructura y funcionamiento están inspirados en las redes neuronales biológicas. Consisten en un conjunto de elementos simples de procesamiento llamados nodos o neuronas unidas entre sí por conexiones que tienen un valor numérico modificable llamado peso.

La actividad que la unidad de procesamiento o neurona artificial realiza en un sistema de este tipo es simple. Normalmente, consiste en sumar los valores de las entradas (*inputs*) que recibe de otras unidades conectadas a ella, comparar esta cantidad con el valor umbral y, si lo iguala o supera, enviar activación o salida (*output*) a las unidades a las que esté conectada. Tanto las entradas que la unidad recibe como las salidas que envía dependen a su vez del peso o fuerza de las conexiones por las cuales se realizan dichas operaciones [2].

1.1. MODELO BIOLÓGICO

Una neurona es una célula viva y, como tal, contiene los mismos elementos que forman parte de todas las células biológicas. Además, contienen elementos característicos que las diferencian [2]. Una neurona consta de un cuerpo celular más o menos esférico de 5 a 10 micras de diámetro del que se desprende una rama principal o axón y varias ramas más cortas llamadas dendritas (Figura 1.1.). A su vez el axón presenta ramas en torno a su punto de arranque, y con frecuencia se ramifica extensamente cerca de su extremo.

Una de las características que diferencian a las neuronas del resto de las células vivas, es la capacidad que tienen éstas de comunicarse. En términos generales, las dendritas y el cuerpo celular reciben señales de entrada, el cuerpo celular las combina e integra y emite señales de salida. El axón transporta estas señales a los terminales axónicos, que se encargan de distribuir información a un nuevo conjunto de neuronas. Por lo general, una neurona recibe información de miles de otras neuronas y envía información a miles de otras más. Se calcula que en el cerebro humano existen del orden de 10^{15} conexiones.

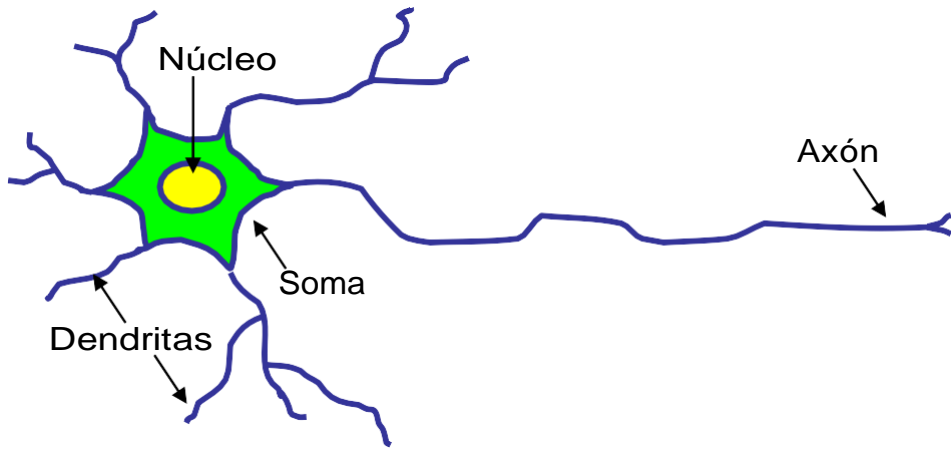


Figura 1.1. Estructura de una neurona biológica
Fuente: Tomado de [1]

Las señales a las que se hace referencia son de dos tipos de naturaleza: eléctrica y química. La señal generada por la neurona y transportada a lo largo del axón es un impulso eléctrico, mientras que las señales que se transmiten entre los terminales axónicos de una neurona y las dendritas de otra es de origen químico; concretamente, se realiza mediante moléculas de sustancias transmisoras (neurotransmisores) que fluyen a través de unos contactos especiales, llamados sinapsis, que tienen la función de receptor y están localizados entre los terminales axónicos y las dendritas de las neuronas siguientes (espacio sináptico entre 50 y 200 Å).

La generación de las señales eléctricas está íntimamente relacionada con la composición de la membrana celular. El proceso de generación de las señales eléctricas de una neurona se puede simplificar del siguiente modo [3]: la neurona, como todas las células, es capaz de mantener en su interior un líquido cuya composición difiere marcadamente de la composición del líquido del exterior. La diferencia más notable se da con relación a la concentración de iones sodio y potasio. El medio externo es unas 10 veces más rico en iones sodio que el interno, mientras que el interno tiene una concentración de iones potasio 10 veces mayor que el externo. Esta diferencia de

concentraciones de iones sodio y potasio genera una diferencia de potencial entre el interior y el exterior de la membrana celular del orden de los 70 mV (negativa en el interior de la membrana). Esto es lo que se conoce como el potencial de reposo de la célula nerviosa.

La llegada de señales procedentes de otras neuronas a través de las dendritas actúa acumulativamente, bajando ligeramente el valor del potencial de reposo. Dicho potencial modifica la permeabilidad de la membrana de manera de que cuando llega a cierto valor crítico comienza una entrada masiva de iones sodio que invierten la polaridad de la membrana.

La inversión del voltaje de la cara interior de la membrana cierra el paso a los iones potasio hasta que se restablece el equilibrio en reposo. La inversión del voltaje, conocida como potencial de acción, se propaga a lo largo del axón y, a su vez, provoca la emisión de los neurotransmisores en los terminales axónicos.

Después de un período refractario, puede seguir un segundo impulso. El resultado de todo esto es la emisión por parte de la neurona de trenes de impulsos, cuya frecuencia varía en función de la cantidad de neurotransmisores recibidos.

Existen dos tipos de sinapsis: (a) la sinapsis activadora, cuyos neurotransmisores provocan disminuciones de potencial en la membrana de la célula postsináptica, facilitando la generación de impulsos a mayor velocidad, y (b) la sinapsis inhibidora, cuyos neurotransmisores tienden a estabilizar el potencial de la membrana, dificultando la emisión de impulsos. Casi todas las neuronas reciben entradas procedentes de sinapsis activadoras o inhibidoras. En cada instante algunas de ellas estarán activas y otras se hallarán en reposo; la suma de los efectos activadores e inhibidores determina si la célula será o no estimulada: es decir, si emitirá o no un tren de impulsos y a qué velocidad [2].

1.2. MODELO IDEALIZADO

La neurona que se presenta en la figura 1.2., es utilizada en muchas redes neuronales artificiales actuales. De manera similar al modelo de neurona funcional básico consta de tres partes básicas: Dendritas, Soma con núcleo y Axón.

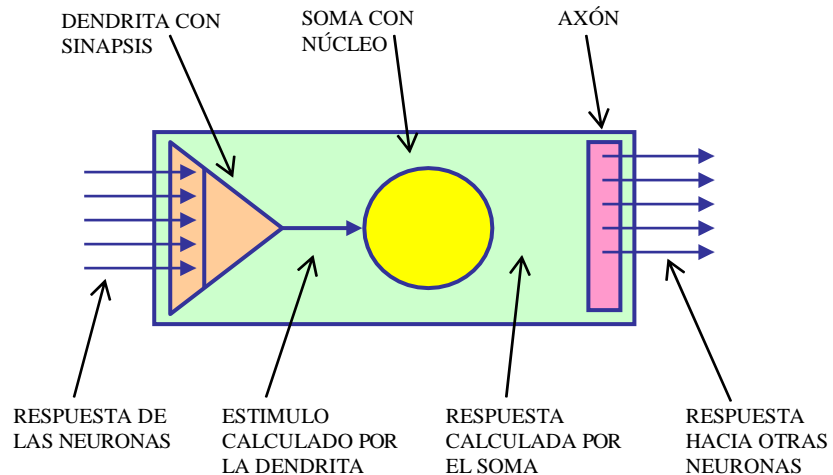


Figura 1.2. Modelo idealizado de una neurona
Fuente: Tomado de [1]

La neurona tiene una sola dendrita. La dendrita acumula el estímulo enviado a la neurona desde las otras neuronas. El soma procesa los estímulos recibidos a través de su dendrita y decide la respuesta de la neurona. El axón se encarga de distribuir esta respuesta a las otras neuronas.

Los modelos de redes neuronales artificiales, o simplemente redes neuronales, se conocen por diversos nombres como modelos conexionistas o modelos de procesamiento distribuido paralelo. En lugar de ejecutar un programa secuencialmente como en una arquitectura Von Neumann, la red neuronal explora muchas hipótesis de manera simultánea utilizando redes masivamente paralelas compuestas de muchos elementos de procesamiento conectados por enlaces con pesos.

1.3. DEFINICIÓN DE RED NEURONAL

Las redes neuronales son sistemas compuestos de elementos simples (neuronas) operando en paralelo, inspirados en el sistema nervioso biológico. Como en la naturaleza, la función de la red está determinada principalmente por la conexión entre elementos. Se puede entrenar una red neuronal para que realice una función particular mediante el ajuste de los valores de conexión, denominados pesos, entre dichos elementos y la elección de una función de transferencia que produzca el resultado deseado [4].

1.4. ELEMENTOS DE UNA RED NEURONAL ARTIFICIAL

Las redes neuronales son modelos que intentan reproducir el comportamiento del cerebro. Por lo tanto, dicho modelo realiza una simplificación, averiguando cuáles son los elementos relevantes del sistema. Una elección adecuada de sus características, más una estructura conveniente, es el procedimiento convencional utilizado para construir redes capaces de realizar una determinada tarea.

Cualquier modelo de red neuronal consta de dispositivos elementales de proceso: las neuronas. A partir de ellas, se pueden generar representaciones específicas, de forma tal que un estado conjunto de ellas puede significar una letra, un número o cualquier otro objeto. Generalmente se pueden encontrar tres tipos de neuronas:

- 1) Aquellas que reciben estímulos externos, relacionadas con el aparato sensorial, que tomarán la información de entrada.
- 2) Dicha información se transmite a ciertos elementos internos que se ocupan de su procesamiento. Es en las sinapsis y neuronas correspondientes a este segundo nivel donde se genera cualquier tipo de representación interna de la información. Puesto que no tienen relación directa con la información de entrada ni con la de salida, estos elementos se denominan unidades ocultas.
- 3) Una vez finalizado el período de procesamiento, la información llega a las unidades de salida, cuya misión es dar la respuesta del sistema.

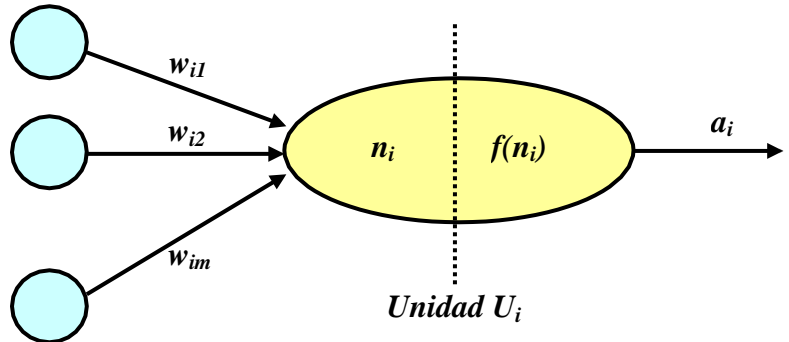


Figura 1.3. Estructura de una neurona artificial
Fuente: Tomado de [2]

La neurona artificial (Figura 1.3) pretende mimetizar las características más importantes de las neuronas biológicas. Cada neurona i -ésima está caracterizada en cualquier instante por un valor numérico denominado valor o estado de activación, una función de activación o de transferencia que transforma el estado actual de activación en una señal de salida a_i . Dicha señal de salida es enviada a través de canales de comunicación unidireccionales a otras unidades de la red; en estos canales, la señal se modifica de acuerdo con la sinapsis (el peso w_{ji}) asociada a cada uno de ellos según una determinada regla. Las señales moduladas que han llegado a la unidad j -ésima se combinan entre ellas, generando así la entrada total n_j :

$$n_j = \sum w_{ji} a_i \quad (1)$$

La dinámica que rige la actualización de los estados de las unidades (evolución de la red neuronal) puede ser de dos tipos: modo asincrónico y modo sincrónico. En el primer caso, las neuronas evalúan su estado continuamente según les va llegando información y lo hacen de forma independiente. En el caso sincrónico la información también llega de forma continua pero los cambios se realizan simultáneamente, como si existiera un reloj interno que decidiera cuándo deben cambiar su estado. Los sistemas biológicos quedan probablemente entre ambas posibilidades.

1.4.1. Neurona artificial

Si se tienen R unidades (neuronas), se puede ordenar las mismas de manera arbitraria y designar a la j -ésima unidad como U_j . Su trabajo es simple y único, consiste en recibir las entradas de las células vecinas y calcular un valor de salida el cual es enviado todas las células conectadas con U_j . En cualquier sistema que se esté modelando es útil caracterizar tres tipos de unidades: entradas, salidas y ocultas. Las unidades de entrada reciben señales desde el entorno; estas entradas (que son a la vez entradas a la red) pueden ser señales provenientes de sensores o de otros sectores del sistema (salidas de la red); estas señales pueden controlar directamente otros sistemas. Las unidades ocultas son aquellas cuyas entradas y salidas se encuentran dentro del sistema; es decir, no tienen contacto con el exterior. Se conoce como **capa** o **nivel** a un conjunto de neuronas cuyas entradas provienen de la misma fuente (que puede ser otra capa de neuronas).

1.4.2. Estado de activación

Adicionalmente al conjunto de unidades, la representación necesita los estados del sistema en un tiempo k . Esto se especifica por un vector de números reales $A(k)$, que representa el estado de activación del conjunto de unidades de procesamiento [2]. Cada elemento del vector representa la activación de una unidad en el tiempo k . La activación de una unidad U_j en el tiempo k se designa por $a_j(k)$, es decir:

$$A_i(k) = (a_1(k), a_2(k), \dots, a_N(k))$$

(2)

El procesamiento que realiza la red se considera como la evolución de un patrón de activación en el conjunto de unidades que la componen a través del tiempo.

Todas las neuronas que componen la red se hallan en cierto estado. Los valores que puede tomar un estado pueden ser continuos o discretos.

Además pueden ser limitados o ilimitados. Si son discretos, suelen tomar un conjunto pequeño de valores o bien valores binarios.

Finalmente es necesario saber qué criterios o reglas siguen las neuronas para alcanzar tales estados de activación. En principio esto depende de dos factores: por un lado, es necesario tener idea del mecanismo de interacción entre las neuronas. El estado de activación estará fuertemente influenciado por tales interacciones, ya que el efecto que producirá una neurona sobre otra será proporcional a la fuerza, peso o magnitud de la conexión entre ambas. Por otro lado, la señal que envía cada neurona a sus vecinas dependerá de su propio estado de activación.

1.4.3. Conexiones entre neuronas

Las conexiones que unen a las neuronas que forman una RNA tienen asociado un peso, que es el que hace que la red adquiera conocimiento. Considere a como el valor de salida de una neurona i en un instante dado. Una neurona recibe un conjunto de señales que proporcionan información del estado de activación de todas las neuronas con las que se encuentra conectada. Cada conexión (sinapsis) entre la neurona i y la j está ponderada por un peso w_{ji} . Normalmente, como simplificación, se considera que el efecto de cada señal es aditivo, de tal forma que la entrada neta que recibe una neurona (potencial postsináptico) n_j es la suma del producto de cada señal individual por el valor de la sinapsis que conecta ambas neuronas:

$$n_j = \sum_{i=1}^N w_{ji} a_i \quad (3)$$

Esta regla muestra el procedimiento a seguir para combinar los valores de entrada a una unidad con los pesos de las conexiones que llegan a esa unidad y es conocida como regla de propagación.

Suele utilizarse una matriz W con todos los pesos w_{ij} que reflejan la influencia que sobre la neurona j tiene la neurona i . W es un conjunto de elementos positivos, negativos o nulos. Si w_{ji} es positivo indica que

la interacción entre las neuronas i y j es activadora; es decir, siempre que la neurona i esté activada, la neurona j recibirá una señal de i que tenderá a activarla. Si w_{ji} es negativo la sinapsis será inhibitoria. En este caso si i está activada, enviará una señal j que tenderá a desactivar a esta. Finalmente, si w_{ji} es 0 se supone que no hay conexión entre ambas.

1.4.4. Función de transferencia o activación

Entre las unidades o neuronas que forman una red neuronal artificial existe un conjunto de conexiones que unen unas con otras. Cada unidad transmite señales a aquellas que están conectadas con su salida. Asociada con cada unidad U_i hay una función de activación o de transferencia $f(n_i)$ que transforma la entrada neta de la neurona en una señal de salida $a_i(k)=f(n_i(k))$ □. El vector que contiene las salidas de todas las neuronas en un instante k es $a(k)=(f_1(n_1(k)), f_2(n_2(k)), ..., f_N(n_N(k)))$.

Además, normalmente la función de activación no está centrada en el origen del eje de abscisas, sino que existe cierto desplazamiento debido a las características internas de la neurona y que no es igual en todas ellas. Este valor se denota como θ y representa el umbral de activación de la neurona i .

$$a_i(k+1) = f(n_i - \theta) = f\left(\sum_{j=1}^N w_{ji} a_j(k) - \theta_i\right) \quad (4)$$

Existen cuatro funciones de activación típicas (Figura 1.4) que determinan distintos tipos de neuronas [5]:

- a) Función escalón
- b) Función lineal y mixta
- c) Función sigmoideal
- d) Función gaussiana.

La función escalón o umbral sólo se utiliza cuando las salidas de la red son binarias, que representa dos valores posibles. La salida de una neurona se activa sólo cuando la entrada neta es mayor o igual que

cierto valor umbral, es decir que la función puede estar desplazada sobre los ejes. La función mixta y sigmoideal son las más apropiadas cuando se quiere como salida la información analógica. A continuación se verá con más detalle las distintas funciones:

- a) **Neurona de función escalón.** La forma más fácil de definir la activación de una neurona es considerar que ésta es binaria. La función de transferencia escalón se asocia a neuronas binarias en las cuales cuando la suma de las entradas es mayor o igual que el umbral de la neurona, la activación es 1; si es menor, la activación es 0 o -1 . Las redes formadas por este tipo de neuronas son fáciles de implementar en hardware, pero a menudo sus capacidades están limitadas.
- b) **Neurona de función lineal y mixta.** La función lineal o identidad responde a la expresión $f(x)=x$. En las neuronas con función mixta, si la suma de las señales de entrada es menor que un límite inferior, la activación se define como 0 o -1 . Si dicha suma es mayor que un límite superior, entonces la activación es 1. Si la suma de entradas está entre ambos límites entonces la activación se define como una función lineal de la suma de las entradas.
- c) **Neurona de función continua (sigmoideal).** Cualquier función definida simplemente en un intervalo de posibles valores de entrada, con un incremento monótonico y que tenga ambos límites superiores e inferiores, por ejemplo las funciones sigmoideal o arcotangente, podrá realizar la función de activación de forma satisfactoria. Con la función sigmoideal, para la mayoría de los valores del estímulo de entrada (variable independiente), el valor dado por la función es cercano a uno de los valores asintóticos. Esto hace que en la mayoría de los casos, el valor de salida este comprendido en la zona alta o baja del sigmoide. De hecho, cuando la pendiente es elevada, esta función tiende a la función escalón. Sin embargo, la importancia de la función sigmoideal es que su derivada es siempre positiva y cercana a cero para los valores grandes positivos o negativos; además, toma su valor máximo cuando x es 0. Esto hace que se puedan utilizar las reglas de

aprendizaje definidas para las funciones escalón, con la ventaja de que la derivada está definida en todo el intervalo. La función escalón no tiene derivada definida en el punto de transición y esto no ayuda a los métodos de aprendizaje en los que se usan derivadas.

- d) **Neurona con función de transferencia gaussiana.** Los centros y la amplitud de estas funciones pueden ser adaptadas lo cual las hace más atractivas que las funciones sigmoidales. Las correspondencias que suelen requerir dos niveles ocultos utilizando neuronas con funciones de transferencia sigmoidales, algunas veces se pueden realizar con un solo nivel en redes con funciones gaussianas. Para simplificar la expresión de la salida de una neurona i , es habitual considerar la existencia de una neurona ficticia con valor de salida unidad asociada a la entrada de cada neurona i mediante una conexión con peso de valor $-\theta_i$. De esta forma la expresión de salida quedará:

$$a_i(k+1) = f(n_i - \theta_i) = f\left(\sum_{j=1}^N w_{ji} a_j(k) - \theta_i\right) = f\left(\sum_{j=0}^N w_{ji} a_j(k)\right) \quad (5)$$

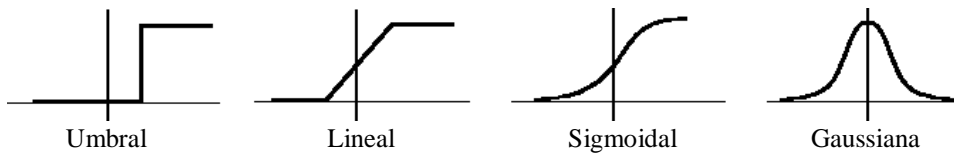


Figura 1.4. Funciones de activación
Fuente: Tomado de [5]

1.4.5. Regla de aprendizaje

Existen muchas definiciones de concepto general de aprendizaje, una de ellas podría ser: *La modificación del comportamiento inducido por la interacción con el entorno y cómo resultado de experiencias conducente al establecimiento de nuevos modelos de respuesta a estímulos externos.* Esta definición fue enunciada muchos años antes

de que surgieran las redes neuronales, sin embargo puede ser aplicada también a los procesos de aprendizaje de estos sistemas. De manera biológica, se suele aceptar que la información memorizada en el cerebro está más relacionada con los valores sinápticos de las conexiones entre las neuronas que con ellas mismas: es decir, el conocimiento se encuentra en las sinapsis. En el caso de las redes neuronales artificiales, se puede considerar que el conocimiento se encuentra representado en los pesos de las conexiones entre neuronas. Todo proceso de aprendizaje implica cierto número de cambios en estas conexiones. En realidad, puede decirse que se aprende modificando los valores de los pesos de la red.

Al igual que el funcionamiento de una red depende del número de neuronas de las que disponga y de cómo estén conectadas entre sí, cada modelo dispone de su o sus propias técnicas de aprendizaje.

1.4.6. Representación vectorial

En ciertos modelos de redes neuronales, se utiliza la forma vectorial como herramienta de representación de algunas magnitudes. Si se considera una red formada por varias capas de neuronas idénticas, es posible considerar las salidas de cierta capa de n unidades como un vector n -dimensional $A=(a_{1j}, a_{2j}, ..., a_{nj})$. La entrada neta de la j -ésima unidad se puede escribir en forma de producto escalar del vector de entradas por el vector de pesos. Cuando los vectores tienen igual dimensión, este producto se define como la suma de los productos de los componentes de ambos vectores.

$$n_j = \sum_{i=1}^N w_{ji} a_i \quad (6)$$

Donde n representa el número de conexiones de la j -ésima unidad. La ventaja de la notación vectorial es que la ecuación anterior se puede escribir de la forma:

$${}_j n = W \cdot A \quad (7)$$

Taller 1

FAMILIARIZACIÓN CON MATLAB

MatLab es un programa interactivo para computación numérica y visualización de datos. Es ampliamente usado por Ingenieros de Control en el análisis y diseño, posee además una extraordinaria versatilidad y capacidad para resolver problemas en matemática aplicada, física, química, ingeniería, finanzas y muchas otras aplicaciones. Está basado en un sofisticado software de matrices para el análisis de sistemas de ecuaciones. Permite resolver complicados problemas numéricos sin necesidad de escribir un programa. MatLab es un entorno de computación y desarrollo de aplicaciones, es totalmente integrado y orientado para llevar a cabo proyectos en donde se encuentren implicados elevados cálculos matemáticos y la visualización gráfica de los mismos. MatLab integra análisis numérico, cálculo matricial, proceso de señal y visualización gráfica en un entorno completo donde los problemas y sus soluciones son expresados del mismo modo en que se escribirían tradicionalmente, sin necesidad de hacer uso de la programación tradicional. El nombre de MatLab proviene de la contracción de los términos MATrix LABoratory y fue inicialmente concebido para proporcionar fácil acceso a las librerías LINPACK y EISPACK, las cuales representan dos de las librerías más importantes en computación y cálculo matricial. MatLab es un sistema de trabajo interactivo cuyo elemento básico de trabajo son las matrices. El programa permite realizar de un modo rápido la resolución numérica de problemas en un tiempo mucho menor que si se quisiesen resolver estos mismos problemas con lenguajes de programación tradicionales como pueden ser los lenguajes Fortran, Basic o C [3].

1. Inicio de MatLab

Después de ejecutar el programa MatLab desde el sistema operativo empleado, haciendo doble clic sobre el icono de MatLab, aparece el indicador de comandos el cual está listo para recibir instrucciones en lenguaje MatLab. Este indicador es de la siguiente forma:

>>

Al iniciar el uso de MatLab están disponibles dos comandos de ayuda y demostración. Para ejecutarlos se escribe el comando en la línea de comandos después del símbolo >> y se presiona la tecla Enter. Por ejemplo:

>>help

permite obtener una ayuda sobre los diferentes comandos de MatLab.

>>demo

hace una demostración de las diferentes aplicaciones de MatLab.
Para cerrar o finalizar el uso de MatLab se usa el comando quit.
>>quit

2. Uso de comandos

La primera forma de interactuar con MatLab es a través de la línea de comandos. Puede ejecutarse un comando si este está escrito después del símbolo >> y se presiona la tecla Enter. MatLab trabaja esencialmente con matrices numéricas rectangulares. La manera más fácil de entrar matrices pequeñas es enumerando los elementos de ésta de tal manera que: (a) los elementos estén separados por blancos ó comas, (b) los elementos estén cerrados entre corchetes, [] y (c) muestre el final de cada fila con “;” (punto y coma).

Ejemplo:

```
>>A = [ 1 2 3; 4 5 6; 7 8 9 ]
```

resultaría en la matriz

A =

1 2 3

4 5 6

7 8 9

MatLab guarda esta matriz para utilizarla luego bajo el nombre de A. Si la matriz a introducir es muy grande se puede utilizar el siguiente formato:

```
A = [1 2 3
```

```
4 5 6
```

```
7 8 9]
```

El comando **load** y la función **fread** pueden leer matrices generadas en sesiones anteriores ó generadas por otros programas. Ya que MatLab se basa en el álgebra de matrices como ejemplo se crea una matriz. Estas pueden estar formadas por un sólo elementos (escalar), por una fila o una columna (vector) o por una serie de filas y columnas (matriz propiamente dicha).

```
>>A=1
```

define A como un escalar de valor 1. Al definir A automáticamente MatLab presenta en pantalla su valor.

A =

1

Para no presentar el valor de la variable creada, debe agregarse punto y coma (;) al final del comando. Después de crear una variable, puede presentar su valor en pantalla escribiendo la variable después del prompt (>>).

```
>>A
```

Se pueden redefinir variables, por ejemplo:

```
>>A=[1 2 3]
```

define A como un vector de tres elementos, A(1)=1, A(2)=2 y A(3)=3. Estos elementos deben separarse con espacios en blanco o comas (,). Para definir una matriz se deben separar las filas con punto y coma (;) o con retorno (Enter).

```
>>A=[1 2 3; 4 5 6]
```

o

```
>>A=[1 2 3  
4 5 6]
```

ambos comandos producen el mismo efecto:

```
A =
```

```
1 2 3
```

```
4 5 6
```

2.1. Elementos de matrices

Los elementos de una matriz pueden ser cualquier expresión de MatLab.

Ejemplo:

```
>>x = [-1.3,sqrt(3),(1+2+3) *4/5]
```

resultaría en

```
x =
```

```
-1.3000 1.7321 4.8000
```

Se puede hacer referencia a los elementos individuales de la matriz con índices entre paréntesis.

Ejemplo: En el ejemplo anterior si se tiene

```
>>x(4) = abs(x(1))
```

resultaría

```
x =
```

```
-1.3000 1.7321 4.8000 0 1.3000
```

Para añadir otra fila a la matriz A de arriba se puede hacer lo siguiente:

```
>>r = [10 11 12];
```

```
>>A = [A; r]
```

y resultaría

```
A =
```

```
1 2 3
```

```
4 5 6
```

```
7 8 9
```

```
10 11 12
```

2.2. Instrucciones de MatLab y variables

Si se omite el nombre de la variable y el signo "=", MatLab automáticamente crea la variable **ans** para guardar el resultado. También distingue las letras mayúsculas de las minúsculas. Todos los nombres de funciones deben estar en letras minúsculas.

2.3. Información del espacio de trabajo

Los ejemplos que se han presentado se guardan en variables que están en el espacio de trabajo de MatLab. Para listar las variables en el espacio de trabajo se utiliza el comando **who** . Para ver información adicional acerca de estas variables se utiliza el comando **whos**.

2.4. Variables permanentes

Las variables permanentes son aquellas con significado especial, y que no se pueden eliminar. Estas son por ejemplo las variables **ans** y **eps**. La variable **eps** es una tolerancia para determinar. Por ejemplo la singularidad y el rango. Su valor inicial es la distancia de 1.0 al próximo número de punto flotante mayor.

2.5. Funciones

Las funciones que utiliza MatLab son intrínsecas al procesador de éste. Otras funciones están disponibles en la librería externa de archivos-M. Además de éstas funciones todo usuario también puede crear otras funciones. Se puede combinar las funciones de acuerdo a las necesidades de uso.

Ejemplo:

```
>>x = sqrt(log(z))
```

2.6. Almacenamiento del espacio de trabajo

Para salir de MatLab se escribe quit ó exit. Al terminar una sesión de MatLab, las variables en el espacio de trabajo se borran. Si se desea guardar el espacio de trabajo se escribe **save**. El comando **save** guarda todas las variables en un archivo llamado matlab.mat. Se puede utilizar **save** y **load** con otros nombres de archivos, ó para guardar solo variables seleccionadas

Ejemplo:

```
>> save temp X Y Z
```

Este ejemplo guarda las variables X, Y, Z en el archivo temp.mat. Usando el comando load temp se obtiene nuevamente del archivo temp.mat. Los comandos **load** y **save** también pueden importar y exportar información de archivos ASCII.

2.7. Manejo de vectores y matrices

Los dos puntos (:), son importantes en MatLab. Por ejemplo

```
>>x = 1:5
```

genera un vector fila que contiene los números enteros del 1 al 5:

```
x =
```

```
1 2 3 4 5
```

Para el incremento no necesariamente se tiene que incrementar por números enteros, pueden ser decimales, números negativos ó constantes. Para los índices se puede hacer referencia a los elementos individuales de matrices encerrando sus índices entre paréntesis.

Ejemplo:

```
A =
```

```
1 2 3
```

```
4 5 6
```

```
7 8 9
```

```
>>A(3, 3) = A(1, 3) + A(3, 1)
```

resultaría

```
A =
```

```
1 2 3
```

```
4 5 6
```

```
7 8 10
```

Un índice puede ser un vector. Si x y v son vectores, entonces $x(v)$ es $[x(v(1)), x(v(2)), ..., x(v(n))]$. Para matrices, los índices de vectores permiten acceso a submatrices contiguas y no contiguas.

Por ejemplo, suponga que A es una matriz 10 por 10. Entonces

```
A(1:5, 3)
```

especifica la submatriz 5 x 1, ó vector columna, que consiste de los primeros cinco elementos en la tercera columna de A . También

```
A(1:5, 7:10)
```

es la submatriz 5 x 4 de las primeras cinco filas y las últimas cuatro columnas. Utilizando solo los dos puntos denota todo lo correspondiente a la fila ó columna. Se podría tener una instrucción como:

```
A(:, [3 5 10]) = B(:, 1:3)
```

que reemplaza la tercera, quinta y décima columna de A con las primeras tres columnas de B .

3. Programación con MatLab

Programar en MatLab es usar una serie de comandos que permitan realizar una tarea o función específica. Dichos comandos pueden ser escritos uno por uno a través de la línea de comandos:

```
>>A=[1 2 3;4 5 6;7 8 9]
```



```

A =
1 2 3
4 5 6
7 8 9
>>A'
ans =
1 4 7
2 5 8
3 6 9

```

El primer comando `A=[1 2 3;4 5 6;7 8 9]` define la matriz A y el siguiente comando `A'` calcula y presenta en pantalla la transpuesta de A.

3.1. Archivos-M: comandos y funciones

Los archivos de disco que contienen instrucciones de MatLab se llaman archivos-M. Esto es así porque siempre tienen una extensión de ".m" como la última parte de su nombre de archivo. Un archivo -M consiste de una secuencia de instrucciones normales de MatLab, que probablemente incluyen referencias a otros archivos-M. Un archivo-M se puede llamar a sí mismo de manera recursiva. Es posible crear archivos-M utilizando un editor de texto ó procesador de palabras. Hay dos tipos de archivos -M: los de comandos y las funciones. Los archivos de comandos, automatizan secuencias largas de comandos. Los archivos de funciones, permiten añadir a MatLab funciones adicionales expandiendo así la capacidad de este programa. Ambos, comandos y funciones, son archivos ordinarios de texto ASCII.

Cuando un archivo de comandos es invocado, MatLab simplemente ejecuta los comandos encontrados en dicho archivo. Las instrucciones en un archivo de comando operan globalmente en los datos en el espacio de trabajo. Los comandos son utilizados para hacer análisis, resolver problemas, ó diseñar secuencias largas de comandos que se conviertan en interactivas. Por ejemplo, suponga que el archivo **fib.m** contiene los siguientes comandos de MatLab:

```

% Un archivo-M para calcular los elementos de la serie de Fibonacci
f = [1 1]; i = 1;
while f(i) + f(i+1) < 1000
f(i+2) = f(i) + f(i+1);
i = i + 1;
end
plot(f)

```

Si se escribe **fibo** en una ventana de MatLab seguido de "enter" se observa que MatLab calcula los primeros 16 números de Fibonacci, y luego grafica estos. Luego que la ejecución del archivo es completada, las variables **f** e **i** permanecen en el espacio de trabajo. Los programas de demostración incluidos en MatLab son ejemplos de cómo usar comandos para hacer tareas más complicadas. Para utilizar las demostraciones escriba la instrucción *demos* en el "prompt" de MatLab.

>> demos

Un archivo-M que contiene la palabra **function** al principio de la primera línea, es un archivo de función. En una función, a diferencia de un comando, se deben de pasar los argumentos. Las variables definidas y manipuladas dentro de la función son locales a esta y no operan globalmente en el espacio de trabajo. Los archivos de funciones se utilizan para extender a MatLab, es decir, crear nuevas funciones para MatLab utilizando el lenguaje propio de MatLab.

El archivo **mean.m** contiene las instrucciones:

```
function y = mean(x)
% Valor medio.
% Para vectores, mean(x) retorna el valor medio de los elementos del vector
x.
% Para matrices, mean(x) es un vector fila que contiene el valor medio de
cada columna.
[m, n] = size(x);
if m == 1
m = n;
end
y = sum(x)/m;
```

Las líneas que comienzan con "%" son interpretadas como comentarios por MatLab. La existencia de este archivo en el disco duro define una nueva función en MatLab llamada **mean**. Si **z** es un vector de los enteros desde 1 a 99,

por ejemplo: **z = 1:99**; entonces, el valor promedio es encontrado escribiendo **mean(z)** que resultaría
ans =
50

En **mean.m**, la primera línea declara el nombre de la función, los argumentos de entrada, y los argumentos de salida. Sin esta línea sería un archivo de comando. El símbolo % indica que el resto de la línea es un comentario. Las primeras líneas documentan el archivo-M y aparecen en la pantalla cuando se escribe **help mean**. Las variables **m**, **n**, e **y** son locales a mean y no existen en el espacio de trabajo (o si existen, permanecen sin cambios.) No es necesario asignar los enteros de 1 al 99 en la variable **x**. Se utiliza **mean** con una variable llamada z. Este vector que contenía los enteros de 1 a 99 fue pasado ó copiado a **mean** donde se convirtió en una variable local llamada **x**.

Ejemplo

```
% Ejemplo de un archivo-m
% Creación del vector x usando el comando for
n=5;
for i=1:n
x(i)=i^2;
end
x
% Fin del archivo-m
```

Este ejemplo es un archivo-m tipo comando. Para ejecutarlo, en la línea de comandos se debe escribir el nombre del archivo:

```
>>ejemplo
```

```
x =
1 4 9 16 25
```

Ejemplo

```
% Calcula el promedio de los elementos de un vector y dibuja dicho vector
% Sintaxis : promedio(x) donde x es el vector a promediar
```

```
function p = promedio(x)
```

```
n=length(x);
```

```
p=0;
```

```
for i=1:n
```

```
p=p+x(i);
```

```
end
```

```
p=p/n;
```

```
plot(x);
```

Para ejecutar la función, se hace la llamada en la línea de comandos incluyendo el parámetro. La función promedio usa por parámetro un vector. Este vector debe ser definido previamente.

```
>>A=[1 2 4 3 7 5 6 1 2 0 8 5];
```

```
>>promedio(A)
```

```
ans =
```

```
3.6667
```

MatLab presenta las imágenes en una ventana de figuras. Esta imagen es el resultado del comando **plot(x)** al ejecutar la función promedio. MatLab posee un conjunto de archivos-m incorporados (built-in). Puede agregársele archivos -m definidos por el usuario almacenando los mismos en el directorio principal de MatLab. Los comentarios incluidos en estos scripts y funciones se visualizan al usar el comando **help** seguido del nombre del archivo.

```
>>help promedio
```

Calcula el promedio de los elementos de un vector y dibuja dicho vector

Sintaxis: promedio(x) donde x es el vector a promediar

Para ver el contenido de un archivo-m se usa el comando type seguido del nombre del archivo.

```
>>type promedio
```

Ejercicios propuestos 1

1. Defina dos matrices de números enteros **A** y **B**, luego realice las siguientes operaciones:
 - a) Obtenga la transpuesta de la matriz **A**.
 - b) Obtenga la transpuesta de la matriz **B**.
 - c) Obtenga **A+B** y **A-B**, comente las restricciones para la realización de dichas operaciones.
 - d) Obtenga **A*B**.
 - e) Obtenga **A/B** y **B/A**, comente los resultados.
2. Construya archivos-m de MatLab para obtener lo siguiente:
 - a) Los primeros **n** números naturales de Fibonacci.
 - b) La suma de los primeros **n** números naturales pares.
 - c) La suma de los primeros **n** números naturales impares.
 - d) Determine si un número dado **x** es primo o no.
 - e) Calcule el máximo común divisor de dos números enteros (**m, n**).

2 MÉTODO Y TOPOLOGÍAS EN LAS REDES NEURONALES

2.1. MÉTODO EN LAS REDES NEURONALES

Se presenta una metodología para el desarrollo de sistemas neurobiológicos que tiene una incidencia directa sobre aquellos relacionados con la neurocomputación y descritos de manera inicial en el capítulo 1. La metodología tiene tres etapas diferenciadas que son: descripción del sistema, especificación del diseño e implantación del sistema [6].

La meta principal de la descripción del sistema consiste en el relevamiento relacionado con las características centrales del sistema neurobiológico a desarrollar, los problemas que resuelven las redes neuronales suelen estar centrados con preferencia en tres áreas diferenciadas: tareas de clasificación y categorización, reconocimiento de patrones y labores de predicción [5]. Los datos neuronales disponibles y cualquier conocimiento funcional, relativo al problema a resolver, se utiliza para describir la arquitectura, la funcionalidad, y la representación del sistema neuronal. Esta descripción debe incluir al menos: (1) la conexión interna básica entre los subsistemas, (2) las funciones de respuesta de las neuronas, (3) las curvas de afinado de la neurona, (4) las relaciones funcionales de los subsistemas y (5) el comportamiento completo del sistema [6].

El propósito de la especificación del diseño consiste en delinear las limitaciones conocidas del mundo real para el sistema neuronal. Como resultado de su aplicación se debe ser capaz de explicitar acerca de las condiciones operativas, tales como el ruido, a las cuales esta sujeta el sistema. La meta principal de este paso consiste en especificar las restricciones de implementación en el modelo para cada variable representada identificada en el paso previo [6].

El tercer paso de la metodología involucra el diseño procedimental, la generación de código y la ejecución del modelo propiamente dicho. Dadas la descripción del sistema y las especificaciones de diseño, este paso combina los mismos para determinar las reglas apropiadas de decodificado, y por consiguiente los pesos sinápticos, necesarios para determinar el comportamiento deseado [6].

Tabla 2.1. Resumen de la metodología para generar modelos neurobiológicos

Fuente: Tomado de [6]

Pasos	Detalle
Paso 1	Descripción del sistema. <ul style="list-style-type: none"> - Identificar las propiedades neurobiológicas relevantes (curvas de afinado, conectividad, etc.). - Especificar las representaciones como variables (escalares, vectores, funciones, etc.). - Proporcionar una descripción funcional, especificación de los subsistemas y la arquitectura completa del sistema.
Paso 2	Especificación de diseño. <ul style="list-style-type: none"> - Especificar el rango, precisión y el radio de las señales al ruido para cada variable. - Especificar las características dinámicas y temporales para cada variable.
Paso 3	Implementación <ul style="list-style-type: none"> - Determinar las reglas de decodificación para implementar las transformaciones especificadas. - Determinar las partes del modelo que serán simulados y el grado de detalle. - Ejecutar experimentos numéricos utilizando la simulación resultante.

2.2. ESTRUCTURA DE UNA RED NEURONAL

2.2.1. Niveles o capas de neuronas

La distribución de neuronas dentro de la red se realiza formando niveles o capas de un número determinado de neuronas cada una. A partir de su situación dentro de la red, se pueden distinguir tres tipos de capas:

- De entrada:** Es la capa que recibe directamente la información proveniente de las fuentes externas de la red.

- b) **Ocultas:** Son internas a la red y no tienen contacto directo con el exterior. El número de niveles ocultos puede estar entre 0 y un número elevado. Las neuronas de las capas ocultas pueden estar interconectadas de distintas maneras, lo que determina, junto a su número, las distintas topologías de redes neuronales.
- c) **De salida:** Transfieren información de la red hacia el exterior.

Se dice que una red está totalmente conectada si todas las salidas desde un nivel llegan a todos y cada uno de los nodos del nivel siguiente.

2.2.2. Formas de conexión entre neuronas.

La conectividad entre los nodos de una red neuronal está relacionada con la forma en que las salidas de las neuronas están canalizadas para convertirse en entradas de otras neuronas (Figura 2.1.). La señal de salida de un nodo puede ser una entrada de otro elemento de proceso, o incluso ser una entrada de sí mismo (conexión autorrecurrente).

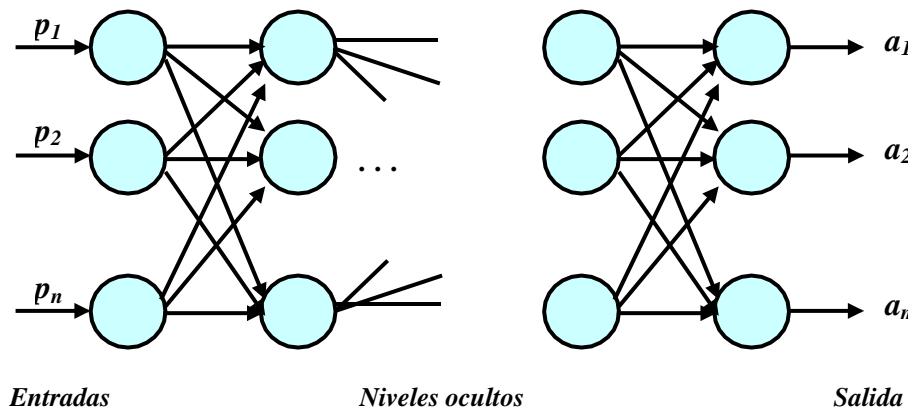


Figura 2.1. Estructura de red multinivel con conexiones hacia adelante
Fuente: [2]

Cuando ninguna salida de las neuronas es entrada de neuronas del mismo nivel o de niveles precedentes, la red se escribe como de propagación hacia adelante, cuando las salidas pueden ser conectadas como entradas de neuronas de niveles previos o del mismo nivel,

incluyéndose ellas mismas, la red es de propagación hacia atrás. Las redes de propagación hacia atrás que tienen lazos cerrados, son sistemas recurrentes.

2.3. CARACTERÍSTICAS DE LAS REDES NEURONALES

Existen cuatro aspectos que caracterizan una red neuronal: su topología, el mecanismo de aprendizaje, el tipo de asociación realizada entre la información de entrada y de salida y la forma de representación de esta información [2]. En este apartado se verá lo relacionado con la topología dejando para capítulos posteriores los otros puntos señalados.

2.4. TOPOLOGÍA DE LAS REDES NEURONALES

La topología o arquitectura de las redes neuronales consiste en la organización y disposición de las neuronas en la red formando capas o agrupaciones de neuronas más o menos alejadas de la entrada y salida de la red. En este sentido, los parámetros fundamentales de red son: el número de capas, el número de neuronas por capa, el grado de conectividad y el tipo de conexiones entre neuronas.

Cuando se realiza una clasificación de las redes en términos topológicos, se suele distinguir entre las redes con una sola capa o nivel de neuronas y las redes con múltiples capas, es decir con dos, tres, cuatro o más capas.

2.4.1. Redes Monocapa

En las redes monocapa o de una capa (Tabla 2.2), como la red de Hopfield y la red Brain State In-a-Box, se establece conexiones laterales entre las neuronas que pertenecen a la única capa que constituye la red. También pueden existir conexiones autorrecurrentes

(salida de una neurona conectada a la entrada), aunque en algún modelo, como en el modelo de Hopfield, esta recurrencia no se utiliza.

Tabla 2.2. Redes neuronales monocapa

Fuente: Modificado de [2]

Tipos de conexiones			Modelo de red
Conexiones explícitas	laterales	Conexiones autorrecurrentes	Brain State-In-aBox
			Additive Grossberg (AG) Shunting Grossberg (SG) Optimal Linear Associative Memory
		No autorrecurrentes	Hopfield Boltzman Machine Cauchy Machine
Crossbar			Crossbar Learning Matrix (LM)

Una topología equivalente a las de las redes de una capa es la denominada topología crossbar o de barras cruzadas. Una red de este tipo, por ejemplo la red Learning Matriz, consiste en una matriz de terminales de entrada y salida o barras que se cruzan en unos puntos a los que se les asocia un determinado peso. Esta representación de barra cruzada suele utilizarse como etapa de transición cuando se pretende implementar físicamente una red monocapa, puesto que es relativamente sencillo desarrollar en términos de hardware una estructura como la indicada, por ejemplo, las barras cruzadas serían cables y los puntos de conexión resistencias cuyos valores representarían los pesos de la red.

Finalmente hay que indicar que las redes monocapa se utilizan típicamente en tareas relacionadas conocidas como autoasociación; por ejemplo, para regenerar la información de entrada que se presentan a la red incompleta o distorsionada.

2.4.2. Redes Multicapa

Las redes multicapa (Tabla 2.3.) son aquellas que disponen de conjuntos de neuronas agrupadas en varios niveles o varias capas, normalmente mayores o iguales a dos. En estos casos, una forma para distinguir la capa a la que pertenece una neurona, consistiría en fijarse

en el origen de las señales que recibe la entrada y el destino de la señal de salida. Normalmente, todas las neuronas de una capa reciben señales de entrada de otra capa anterior, más cercana a las entradas de la red y envían las señales de salida a una capa posterior, más cercana a las salidas de la red. A estas conexiones se las denomina conexiones hacia delante o feedforward.

Sin embargo, en un gran número de estas redes también existe la posibilidad de conectar las salidas de las neuronas de capas posteriores a las entradas de las capas anteriores, a estas conexiones se las denomina conexiones hacia atrás o feedback. Estas dos posibilidades permiten distinguir dos tipos de redes con múltiples capas: las redes con conexiones hacia adelante o redes feedforward y las redes que disponen de conexiones tanto hacia adelante como hacia atrás o redes feedforward/feedback.

2.4.2.1. Redes con conexiones hacia adelante

En las redes feedforward todas las señales neuronales se propagan hacia delante a través de las capas de la red. No existen conexiones hacia atrás, es decir ninguna salida de neuronas de la capa i se aplica a la entrada de neuronas de capas $i-1$, $i-2$, ..., etc. y normalmente tampoco autorrecurrentes, es decir la salida de una neurona conectada a su propia entrada, ni laterales, entendidas como la salida de una neurona conectada a entradas de neuronas de la misma capa, excepto en el caso de los modelos de red propuestos por Kohonen denominados Learning Vector Quantizer (LVQ) y Topology Preserving Map (TPM), en las que existen unas conexiones implícitas muy particulares entre las neuronas de la capa de salida.

Las redes feedforward más conocidas son: Perceptrón, Adaline, Madaline, Linear Adaptive Memory (LAM), Drive Reinforcement, Backpropagation. Todas ellas son especialmente útiles en aplicaciones de reconocimiento o clasificación de patrones.

2.4.2.2. Redes con conexiones hacia adelante y hacia atrás

En este tipo de redes circula información tanto hacia adelante como hacia atrás durante el funcionamiento de la red. Para que esto sea posible existen conexiones feedforward y conexiones feedback entre las neuronas. En general, excepto el caso particular de las redes Cognitron y Neocognitron, suelen ser bicapa existiendo por lo tanto dos conjuntos de pesos: los correspondientes a las conexiones feedforward de la primera capa (capa de entrada) hacia la segunda (capa de salida) y los de las conexiones feedback de la segunda a la primera. Los valores de los pesos de estos tipos de conexiones no tienen porque coincidir, siendo diferentes en la mayor parte de los casos.

Este tipo de estructura bicapa es particularmente adecuada para realizar una asociación de una información o patrón de entrada, en la primera capa, con otra información o patrón de salida en la segunda capa, denominada heteroasociación, aunque también pueden ser utilizadas para la clasificación de patrones.

Algunas redes de ese tipo tienen un funcionamiento basado en lo que se conoce como resonancia, de tal forma que las informaciones en la primera y segunda capas interactúan entre sí hasta que alcanzan un estado estable. Este funcionamiento permite un mejor acceso a la información almacenada en la red. Los dos modelos de red feedforward/feedback de dos capas más conocidos son la red ART (Adaptive Resonance Theory) y la red BAM (Bidirectional Associative Memory).

También en este grupo de redes existen algunas que tienen conexiones laterales entre neuronas de la misma capa. Estas conexiones se diseñan como conexiones activadoras, con peso positivo, que permiten la cooperación entre neuronas, o como inhibidoras con peso negativo, estableciéndose una competencia entre las neuronas correspondientes.

Una red de este tipo que, además, dispone de conexiones autorrecurrentes es la denominada CABAM. Finalmente, hay que comentar la existencia de un tipo de red feedforward/feedback multicapa muy particular denominada Neocognitron, en la que las

neuronas se disponen en planos superpuestos o capas bidimensionales, lo cual permite que puedan eliminarse las variaciones geométricas, en tamaños, giros, desplazamientos o distorsiones que presenten la información o patrones de entrada a la red.

Tabla 2.3. Redes neuronales multicapa

Fuente: Modificado de [2]

No de capas	Tipos de conexiones			Modelo de red
2 capas	Conexiones hacia adelante			Adaline/Madaline Perceptrón Linear Assoc. Rew. Penalty Linear Assoc. Memory Optimal LAM Drive Reinforcement
			Conexiones laterales implícitas y auto-rrecurrentes	LVQ TPM
	Conexiones hacia adelante/atrás	Sin laterales	BAM Adaptive BAM Temporal Assoc. Mem. Fuzzy Assoc. Memory	
		Con laterales y auto-recurrente	Compet. Adaptive BAM ART	
	3 capas	Conexiones hacia adelante	Sin laterales	Adaptive Heur. Critic. Boltzmann/Cauchy Machine
Con laterales			Counterpropagation Bol./Cau. Machina	
Conexión adelante, atrás y laterales		Bol./Cau. Machina		
N capas	Conexión hacia delante		Backpropagation	
	Adelante/atrás (capas bidimensionales)		Cognitron/Neocognitron	

Taller 2

CAJA DE HERRAMIENTAS PARA REDES NEURONALES

Las redes neuronales están compuestas de elementos simples operando en paralelo. Dichos elementos se encuentran inspirados por los sistemas nerviosos biológicos. Como se observa en la naturaleza, la función de la red está ampliamente determinada por las conexiones entre los elementos simples. Se puede entrenar una red neuronal para ejecutar una función particular ajustando los valores de las conexiones (pesos) entre los elementos. De manera común las redes neuronales son ajustadas, o entrenadas, de manera tal que una entrada particular conduzca a una salida específica.

El modelo de neurona y la arquitectura de una red neuronal describen la forma en la que una red transforma sus entradas en salidas validas. Esta transformación puede ser considerada como un proceso de cómputo. El modelo y la arquitectura colocan las limitaciones sobre las cuales puede ser procesada una red neuronal. La forma en la que una red calcula su salida debe ser entendida antes de que los métodos para el entrenamiento de la red sean explicados de manera práctica.

1. NOTACIÓN MATEMÁTICA PARA ECUACIONES Y FIGURAS

Para representar escalares se utiliza letras minúsculas itálicas (*a*, *b*, *c*, ...), para representar vectores se utiliza letras minúsculas en negrilla (**a**, **b**, **c**,...), para designar matrices letras mayúsculas en negrilla (**A**, **B**, **C**,...).

Las matrices con peso se representan en la tabla T2.1.

Tabla T2.1. Matrices con peso en MatLab

Fuente: Tomado de [3]

Formula MatLab	Descripción
$w_{i,j}(t)$	Elemento escalar, <i>i</i> fila, <i>j</i> columna, <i>t</i> tiempo o iteración.
$W(t)$	Matriz <i>W</i> .
$w_j(t)$	Vector columna.
$w_i(t)$	Vector fila.
$b_i(t)$	Elemento escalar.
$b(t)$	Vector <i>b</i> .
$IW^{k,l}$	Matriz de pesos de entrada
$LW^{k,l}$	Matriz de pesos de las capas

Un superíndice simple es utilizado para identificar los elementos de una capa. Por ejemplo, la entrada a la red de la capa 3 puede ser considerada como n^3 . Los superíndices son utilizados para identificar la conexión fuente (I) y la conexión destino (k) de las capas asociadas a las matrices con peso y de las matrices de entrada. Por ejemplo, la capa de la matriz de pesos de la capa 2 a la capa 4 puede ser representada como $LW^{4,2}$.

Un ejemplo para el manejo de las ecuaciones y la forma de referencia se observa en la figura T2.1.

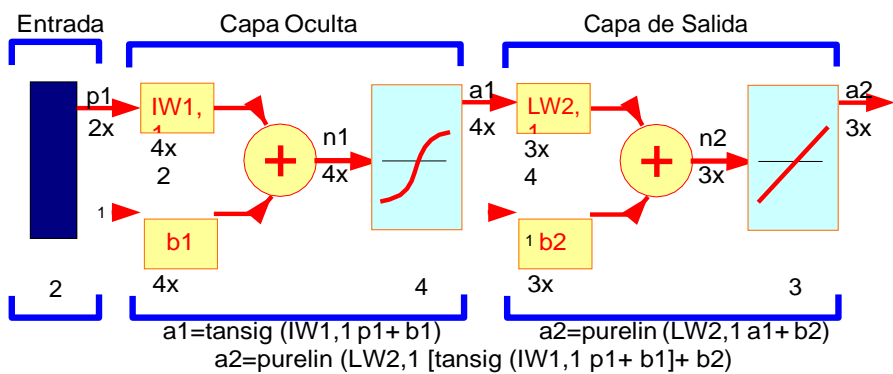


Figura T2.1. Manejo de ecuaciones y su notación
Fuente: Modificado de [3]

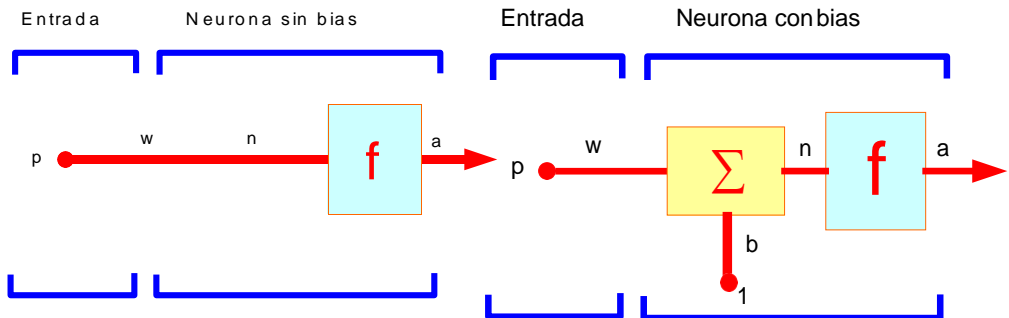
2. EQUIVALENCIA ENTRE FÓRMULAS Y CÓDIGO MATLAB

La transición de las formulas matemáticas a código MatLab y viceversa puede ser realizada con la ayuda de unas cuantas reglas. Las mismas se encuentran listadas en la tabla T2.2.

Tabla T2.2. Transición de formulas matemáticas a MatLab
Fuente: Tomado de [3]

Formula matemática	Equivalente MatLab	Ejemplo
Superíndice	Índice de arreglos	$p^1 \rightarrow p\{1\}$
Subíndice	Índice de paréntesis	$P_2 \rightarrow p(2)$
Superíndice, subíndice	Índice de arreglo, paréntesis	$p^1_2 \rightarrow p\{1\}(2)$
Paréntesis	Índice de arreglo secundario	$p^1(k-1) \rightarrow p\{1,k-1\}$
Operador matemático	Operador MatLab	$ab \rightarrow a*b$

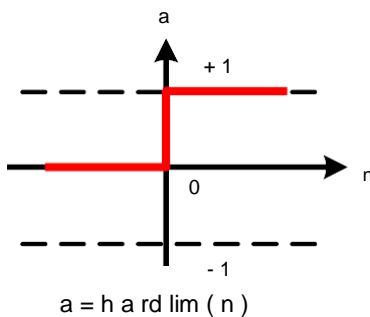
3. REPRESENTACIÓN DE MODELOS DE NEURONA SIMPLE



4. FUNCIONES DE TRANSFERENCIA

Se presenta las tres funciones de transferencia mas utilizadas, estas son: la función hard-limit (signo), la función lineal y la función log-sigmoidal (sigmoide).

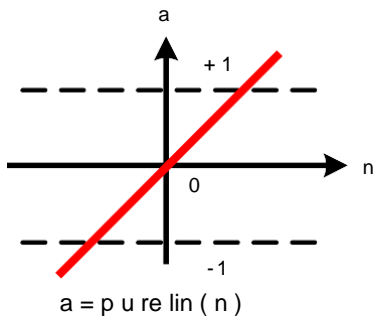
La función de transferencia hard-limit tiene asociada la siguiente representación y su procedimiento en MatLab.



```
n = -5:0.1:5;  
plot(n,hardlim(n),'c+:');
```

Figura T2.3. Función escalón
Fuente: Modificado de [3]

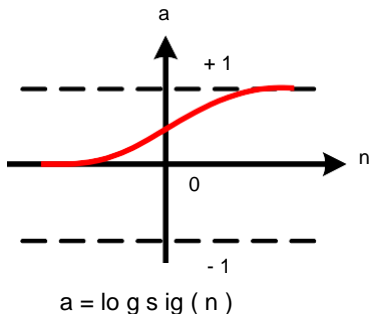
La función de transferencia lineal tiene asociada la siguiente representación y su procedimiento en MatLab.



```
n = -5:0.1:5;  
plot(n,purelin(n),'c+:');
```

Figura T2.4. Función lineal
Fuente: Modificado de [3]

La función de transferencia log-sigmoidal tiene asociada la siguiente representación y su procedimiento en MatLab.



```
n = -5:0.1:5;  
plot(n,logsig(n),'c+:');
```

Figura T2.5. Función sigmoide
Fuente: Modificado de [3]

5. NEURONAS CON VECTORES DE ENTRADA

Una neurona con un vector de entrada de un R -elemento simple es mostrada a continuación. Aquí las entradas con elementos individuales p_1, p_1, \dots, p_R son multiplicadas por los pesos $w_{1,1}, w_{1,2}, \dots, w_{1,R}$, y dichos valores pesados son agrupados en una suma. Dicha suma es simplemente Wp , el producto de la matriz W y el vector p .

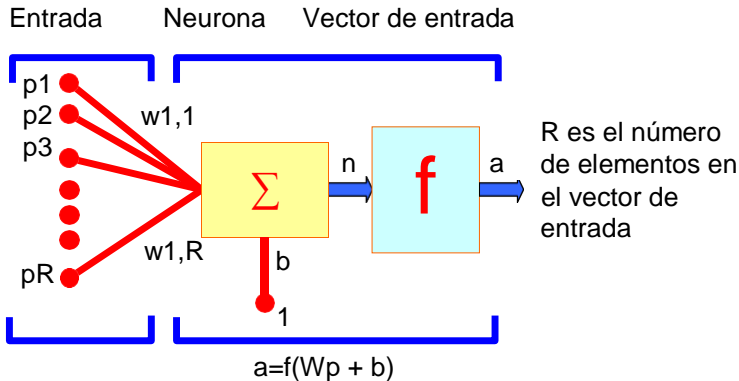


Figura T2.6. Neurona con vectores de entrada

Fuente: Modificado de [3]

La neurona tiene un bias b , el cual es sumado de manera conjunta a las entradas con peso para formar la red de entrada n . Esta suma, n , es el argumento de la función de transferencia f .

$$n = w_{1,1}p_1 + w_{1,2}p_2 + \dots + w_{1,R}p_R + b \quad (8)$$

Esta expresión puede, por supuesto, ser escrita en código MatLab como sigue:

$$\mathbf{n} = \mathbf{W} * \mathbf{p} + b \quad (9)$$

Ejercicios propuestos 2

1. Realice la representación grafica y el archivo-m en MatLab de las siguientes funciones de transferencia.
 - a) Compet Transfer Function
 - b) Symmetric Hard-Limit Transfer Function.
 - c) Positive Linear Transfer Function.
 - d) Radial Basis Function.
 - e) Satlin Transfer Function.
 - f) Satlins Transfer Function.
 - g) Softmax Transfer Function.
 - h) Triangular Basis Function.

3 APRENDIZAJE EN LAS REDES NEURONALES

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 proceso de aprendizaje se reducen a la destrucción, modificación y creación de conexiones entre las neuronas. En los sistemas biológicos existe una continua creación y destrucción de conexiones. 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 0. De la misma forma, una conexión se destruye cuando su peso pasa a ser 0.

Durante el proceso de aprendizaje, los pesos de las conexiones de la red sufren modificaciones, por tanto se puede afirmar que este proceso ha terminado, o que la red ha aprendido, cuando los valores de los pesos permanecen estables $dwi/dk=0$.

Un aspecto importante respecto al aprendizaje en 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. Estos criterios determinan lo que se conoce como la regla de aprendizaje de la red.

De forma general se suelen considerar dos tipos de reglas: las que responden al aprendizaje supervisado, y las correspondientes al aprendizaje no supervisado. La diferencia fundamental entre ambos tipos estriba en la existencia de un agente externo o supervisor, que controle el proceso de aprendizaje de la red.

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 deshabilitación hasta que el proceso finalice. En el primer caso se trataría de un aprendizaje **on line**, mientras que el segundo se conoce como aprendizaje **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 otro de test o prueba que serán utilizados en la fase correspondiente. En las redes con aprendizaje **off line**, los pesos de las conexiones permanecen fijos después de 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. En las redes con aprendizaje **on line** no se distingue entre fase de entrenamiento y de operación, de tal forma que los pesos varían dinámicamente siempre que se presente una nueva información al sistema. En este tipo de redes, debido al carácter dinámico de las mismas, el estudio de la estabilidad suele ser un aspecto fundamental.

3.1. REDES CON APRENDIZAJE SUPERVISADO

El aprendizaje supervisado (Tabla 3.1.) se caracteriza por un entrenamiento controlado por un agente externo, supervisor o maestro, que determina la respuesta que debería generar la red a partir de una entrada determinada. El supervisor comprueba la salida de la red y en el 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:

- a) Aprendizaje por corrección de error.
- b) Aprendizaje por refuerzo.
- c) Aprendizaje estocástico.

3.1.1. Aprendizaje por corrección de error

Este aprendizaje consiste en ajustar los pesos de las conexiones de la red en función de la diferencia entre los valores deseados y los obtenidos en la salida de la red; es decir, en función del error cometido en la salida (Figura 3.1). Una regla o algoritmo simple de aprendizaje por corrección de error está asociada al siguiente formalismo:

$$\Delta w_{ij} = \alpha a_j (t - a_j) \quad (10)$$

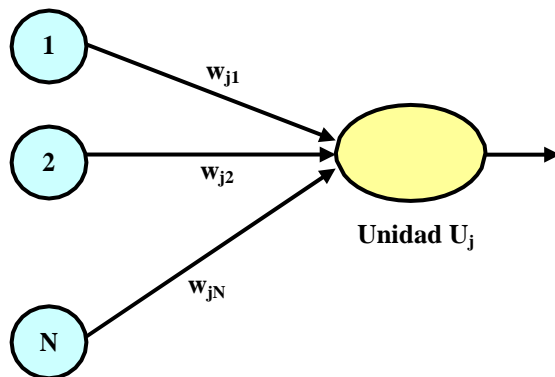


Figura 3.1. Aprendizaje corrección de error
Fuente: Según [2]

Donde:

Δw_{ji} : Variación en el peso de la conexión entre las neuronas i y j ($\Delta w_{ji} = w_{ji}^{actual} - w_{ji}^{anterior}$).

a_i : Valor de salida de la neurona i .

t_j : Valor de salida deseado para la neurona j .

a_j : Valor de salida obtenido en la neurona j .

α : Factor de aprendizaje ($0 < \alpha \leq 1$) que regula la velocidad del aprendizaje.

Un ejemplo de este tipo de algoritmo lo constituye la **regla de aprendizaje del Perceptrón**, utilizada en el entrenamiento de la red del mismo nombre que desarrolló Rosenblatt en 1958. Sin embargo, existen otros algoritmos más evolucionados que éste, que presenta

limitaciones, como el no considerar la magnitud del error global cometido durante el proceso completo de aprendizaje de la red, considerando únicamente los errores individuales o locales correspondientes al aprendizaje de cada información por separado.

Un algoritmo muy conocido que mejora el algoritmo del perceptrón y permite un aprendizaje más rápido y un campo de aplicación más amplio es el propuesto por Widrow y Hoff en 1960 denominado regla delta o regla del error cuadrado mínimo (LMS Error: Least Mean Squared Error), que se aplicó en las redes desarrolladas por los mismos autores, conocidas como Adaline (Adaptative Linear Element), con una única neurona de salida y Madaline con varias neuronas de salida.

Widrow y Hoff definieron una función que permitía cuantificar el error global cometido en cualquier momento durante el proceso de entrenamiento de la red. Este error medio se expresa de la siguiente forma:

$$Error_{global} = \frac{1}{2P} \sum_{k=1}^P \sum_{j=1}^N (a_j^{(k)} - t_j^{(k)})^2 \quad (11)$$

Donde:

N: Número de neuronas de salida (en el caso de Adaline N=1).

P: Cantidad de información que debe aprender la red.

$\frac{1}{2} \sum_{j=1}^N (a_j^{(k)} - t_j^{(k)})^2$: Error cometido en el aprendizaje de la información

k-ésima.

Por tanto, de lo que se trata es de encontrar unos pesos para las conexiones de la red que minimicen esta función de error. Para ello, el ajuste de los pesos de las conexiones de la red se puede hacer de forma proporcional a la variación relativa del error que se obtiene al variar el peso correspondiente.

$$\Delta w_{ij} = k \frac{\partial Error_{global}}{\partial w_{ij}}$$

(12)

Mediante este procedimiento, se llegan a obtener un conjunto de pesos con los que se consigue minimizar el error medio. Otro algoritmo de aprendizaje por corrección de error lo constituye el denominado regla delta generalizada o algoritmo de propagación de errores (error backpropagation) también conocida como regla LMS (Least mean square error) multicapa. Se trata de una generalización de la regla delta para aplicarla a redes con conexiones hacia adelante con capas ocultas.

Estas redes multicapa pueden utilizarse en muchas más aplicaciones que las ya conocidas Perceptrón, Adaline y Madaline, pero su proceso es mucho más lento, debido a que durante el mismo se debe explorar el espacio de posibles formas de utilización de las neuronas de las capas ocultas; es decir, se debe establecer cuál va a ser su papel en el funcionamiento de la red.

Existe también una versión recurrente del algoritmo backpropagation que se suele utilizar en redes multicapa que presentan conexiones recurrentes con el fin de que estas redes aprendan la naturaleza temporal de algunos datos. Para concluir con los algoritmos por corrección de error, hay que mencionar que también se utilizan en algunas redes monocapa con conexiones laterales y autorrecurrentes como la red Brain State-in -a-Box (BSB) introducida por Anderson, Ritz y Jones en 1977. Aunque en una primera etapa el aprendizaje de esta red es sin supervisión, se suelen refinar los valores de los pesos de las conexiones mediante un aprendizaje por corrección de error basado en una adaptación de la regla de Widrow-Hoff.

3.1.2. 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 basándose en un mecanismo de probabilidades. Se podría decir que en este tipo de aprendizaje la función del supervisor se asemeja más a la de un crítico (que opina sobre la respuesta de la red) que a la de un maestro (que indica a la red la respuesta concreta de que debe generar), como ocurriría en el caso de supervisión por corrección de error.

Un ejemplo de algoritmo por refuerzo lo constituye el denominado Linear Reward-Penalty o L_{R-P} (algoritmo lineal con recompensa y penalización) presentado por Narendra y Thathacher en 1974. Este algoritmo ha sido ampliado por Barto y Anand, quienes en 1985 desarrollaron el denominado Associative Reward Penalty que se aplica en redes con conexiones hacia delante de dos capas cuyas neuronas de salida presentan una función de activación estocástica.

Otro algoritmo por refuerzo es el conocido como Adaptive Heuristic Critic, introducido por Barto, Sutton y Anderson en 1983 que se utiliza en redes feedforward de tres capas especialmente diseñadas para que una parte de la red sea capaz de generar un valor interno de refuerzo que es aplicado a las neuronas de salida de la red.

3.1.3. Aprendizaje estocástico

Este tipo de aprendizaje 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 la distribución de probabilidades.

En el aprendizaje estocástico se suele hacer una analogía con la termodinámica. Se asocia a la red neuronal con un sólido físico que tiene un cierto estado energético. En el caso de la red, la energía representa el grado de estabilidad de la misma, de tal forma que el estado de mínima energía correspondería a una situación en la que los pesos de las conexiones consiguen que su funcionamiento sea el que más se ajuste al comportamiento deseado.

Según lo anterior, el aprendizaje consistiría en realizar un cambio aleatorio de los valores de los pesos y determinar la energía de la red (habitualmente la función energía es una función denominada de Lyapunov). Si la energía es menor después del cambio: es decir, si el comportamiento de la red se acerca al deseado, se acepta el cambio. Si, por el contrario, la energía no es menor, se acepta el cambio según una distribución de probabilidades. Una red que utiliza este tipo de aprendizaje es la conocida como Boltzmann Machine ideada por Hinton, Ackley y Sejnowsky en 1984 que lo combina con el aprendizaje Hebbiano o con el aprendizaje por corrección de error (como la regla delta).

La máquina de Boltzman es una red con diferentes topologías alternativas pero siempre con unas neuronas ocultas que permiten, mediante un ajuste probabilístico, introducir un ruido que va decreciendo durante el proceso de aprendizaje para escapar de los mínimos relativos de la función de energía favoreciendo la búsqueda del mínimo global. El procedimiento de utilizar para escapar de mínimos locales suele denominarse simulated annealing y su combinación con la asignación probabilística mediante la capa oculta es lo que se conoce como aprendizaje estocástico. La idea de todo esto es asemejar la red con un sólido físico que inicialmente presenta una alta temperatura (ruido) y que se va enfriando gradualmente hasta alcanzar el equilibrio térmico (mínima energía).

Existe otra red basada en este tipo de aprendizaje denominado Cauchy Machine desarrollada por Szu en 1986 que es un refinamiento de la anterior y que emplea un procedimiento más rápido de búsqueda del

mínimo global y una función de probabilidad diferente (la distribución de probabilidad de Cauchy frente a la de Boltzman).

Tabla 3.1. Tipos de redes con aprendizaje supervisado

Fuente: Modificado de [2]

Tipo			Modelo de Red
Aprendizaje por corrección de Error	por	OFF Line	Perceptrón Adaline/Madaline Backpropagation Brain State-In-a-Box Counterpropagation
Aprendizaje Refuerzo	por	ON Line	Linear Reward Penalty Associative Rew. Penalty Adaptive Heuristic Critic
Aprendizaje Estocástico		OFF Line	Boltzmann Machine Cauchy Machine

3.2. REDES CON APRENDIZAJE NO SUPERVISADO

Las redes con aprendizaje no supervisado (Tabla 3.2.) también conocido como aprendizaje 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; por ello, suele decirse que estas redes son capaces de autoorganizarse.

Estas redes deben encontrar las características, regularidades, correlaciones o categorías que se puedan establecer entre los datos que se presenten en su entrada. Puesto que no hay un supervisor que indique a la red la respuesta que debe generar ante una entrada concreta, cabría preguntarse qué es lo que la red genera en estos casos. 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 entre la información que se le está presentando a la entrada y la información que se le han mostrado hasta entonces. En otro caso, podría realizar

una categorización (clustering) o establecimiento de categorías, donde la salida de la red indica a qué categoría pertenece la información presentada a la entrada, siendo la propia red la que debe encontrar las categorías apropiadas a partir de correlaciones entre las informaciones presentadas. Una variación de esta categorización es el prototipado. En este caso la red obtiene ejemplares o prototipos representantes de las clases a las que pertenecen las informaciones de entrada.

También el aprendizaje sin supervisión permite realizar una codificación de los datos de entrada, generando a la salida una versión codificada de la entrada, con menos bits, pero manteniendo la información relevante de los datos.

Finalmente, algunas redes con aprendizaje no supervisado realizan una correspondencia de características (feature mapping), obteniéndose en las neuronas de salida una disposición geométrica que representa un mapa topográfico de las características de los datos de entrada, de tal forma que si se presentan a la red informaciones similares siempre sean afectadas neuronas de salida próximas entre sí, en la misma zona del mapa.

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

- a) Aprendizaje Hebbiano.
- b) Aprendizaje Competitivo y Cooperativo.

En el primer caso, normalmente se pretende medir la familiaridad o extraer características de los datos de entrada, mientras que el segundo suele orientarse hacia la clasificación de dichos datos.

3.2.1. Aprendizaje hebbiano

Este tipo de aprendizaje se basa en el siguiente postulado formulado por Donald O. Hebb en 1949: *“Cuando un axón de una celda A está suficientemente cerca como para conseguir activar una celda B y repetida o persistentemente toma parte en su activación, algún*

proceso de crecimiento o cambio metabólico tiene lugar en una o ambas celdas, de tal forma que la eficiencia de A, cuando la celda a activar es B, aumenta”. Por celda, Hebb entiende un conjunto de neuronas fuertemente conexas a través de una estructura compleja. La eficiencia podría identificarse con la intensidad o magnitud de la conexión: es decir, con el peso.

Se puede decir, por tanto, que el aprendizaje hebbiano consiste básicamente en el ajuste de los pesos de las conexiones de acuerdo con la correlación (multiplicación en el caso de valores binarios +1 y -1) de los valores de activación (salidas) de las dos neuronas conectadas:

$$\Delta w_{ij} = a_i a_j \quad (13)$$

Esta expresión responde a la idea de Hebb, puesto que si las dos unidades son activas (positivas), se produce un refuerzo de la conexión. Se trata de una regla de aprendizaje no supervisado, pues la modificación de los pesos se realiza en función de los estados (salidas) de las neuronas obtenidos tras la presentación de cierto estímulo (entrada), sin tener en cuenta si se deseaba obtener o no esos estados de activación.

Este tipo de aprendizaje fue empleado por Hopfield en la conocida red que lleve su nombre (Red Hopfield), introducida en 1982 y muy extendida en la actualidad debido principalmente a la relativa facilidad en su implementación en circuitos integrados VLSI.

Grossberg lo utilizó en 1968 en la red denominada Additive Grossberg y en 1973 en la red Shunting Grossberg. En ambos casos se trataba de redes de una capa con conexiones laterales y autorrecurrentes.

La red feedforward llamada Learning Matrix (LM), desarrollada por Steinbuch en 1961 también aprendía mediante correlación hebbiana. En esta red, las neuronas se conectaban en forma matricial asignándose un peso a cada punto de conexión, peso que se obtenía mediante una regla tipo hebbiana.

Otras redes que utilizan el aprendizaje hebbiano son: la red feedforward/feedback de 2 capas denominada Bidirectional Associative Memory (BAM), desarrollada por Kosko en 1988, la red desarrollada por Amari en 1972 llamada Temporal Associative Memory (TAM), con la misma topología que la anterior, pero ideada para aprender la naturaleza temporal de las informaciones que se le muestran (para recordar valores anteriores en el tiempo): la red feedforward de dos capas conocida Linear Associative Memory (LAM) introducida por Anderson en 1968 y refinada por Kohonen, y la red Optimal Linear Associative Memory (OLAM) desarrollada de forma independiente por Wee en 1968 y por Kohonen y Ruohonen en 1973.

Existen muchas variaciones del aprendizaje hebbiano: por ejemplo, Sejnowski en 1977 utilizó la correlación de la covarianza de los valores de activación de las neuronas, Sutton y Barto en 1981 utilizaron la correlación del valor medio de una neurona con la varianza de la otra. Klopff en 1986 propuso una correlación entre las variaciones de los valores de activación en dos instantes de tiempo sucesivos, aprendizaje el que denominó drive-reinforcement y que utilizó en redes del mismo nombre con topología feedforward de dos capas.

Otra versión de este aprendizaje es el denominado hebbiano diferencial que utiliza la correlación de las derivadas en el tiempo de las funciones de activación de las neuronas. El aprendizaje hebbiano diferencial es utilizado en la red feedforward/feedback de dos capas denominada ABAM (Adaptive Bidirectional Associative Memory) introducida por Kosko en 1987. También este autor en 1987 presentó una red con la misma topología que la anterior, pero utilizando otra versión de este aprendizaje, el denominado aprendizaje hebbiano difuso; esta red tenía por nombre Fuzzy Associative Memory (FAM), y se basaba en la representación de las informaciones que debía aprender la red en forma de conjuntos difusos.

Finalmente, hay que comentar la existencia de las redes basadas en mecanismos de aprendizaje que resultan de la combinación de la correlación hebbiana con algún otro método, como es el caso de las redes Boltzmann Machine y Cauchy Machine (mencionadas en el apartado 1.3), que los combinan con el ya comentado simulated annealing. También la red feedforward de tres capas llamada counterpropagation (CPN), desarrollada por Hecht-Nielsen en 1987, utiliza un aprendizaje que es combinación del hebbiano y de un tipo de aprendizaje competitivo introducido por Kohonen denominado learning vector quantization (LVQ).

3.2.2. Aprendizaje competitivo y cooperativo

En las redes con aprendizaje competitivo (y cooperativo), suele decirse que las neuronas compiten (y cooperan) unas con otras con el fin de llevar a cabo una tarea dada. Con este tipo de aprendizaje, se pretende que cuando se presente a la red cierta información de entrada, sólo una de las neuronas de salida de la red, o una por cierto grupo de neuronas, se active (alcance su valor de respuesta máximo). Por tanto, las neuronas compiten por activarse, quedando finalmente una, o una por grupo, como neurona vencedora (winner take all unit), quedando anulado el resto de las neuronas, las que son forzadas al valor de respuesta mínimo.

La competencia entre neuronas se realiza en todas las capas de la red, existiendo en estas neuronas conexiones recurrentes de autoactivación y conexiones de inhibición (signo negativo) por parte de neuronas vecinas. Si el aprendizaje es cooperativo, estas conexiones con las vecinas serán de activación (signo positivo).

El objetivo de este aprendizaje es categorizar los datos que se introducen en la red. De esta forma, las informaciones similares son clasificadas formando parte de la misma categoría, y por tanto deben activar la misma neurona de salida. Las clases o categorías deben ser creadas por la propia red, puesto que se trata de un aprendizaje no supervisado, a través de las correlaciones entre los datos de entrada.

Una forma de aplicar este tipo de aprendizaje fue propuesta por Rumelhart y Zisper en 1985 quienes utilizaban redes multicapa dividiendo cada capa en grupos de neuronas, de tal forma que éstas disponían de conexiones inhibitoras con otras neuronas de su mismo grupo, y conexiones activadoras con las neuronas de la siguiente capa. En una red de este tipo, después de recibir diferentes informaciones de entrada, cada neurona en cada grupo se especializa en la respuesta a determinadas características de los datos de entrada.

En este tipo de redes, cada neurona tiene asignado un peso total, suma de todos los pesos de las conexiones que tiene a su entrada. El aprendizaje afecta sólo a las neuronas ganadoras (activas), redistribuyendo este peso total entre sus conexiones, sustrayendo una porción a los pesos de todas las conexiones que llegan a la neurona vencedora y repartiendo esta cantidad por igual entre todas las conexiones procedentes de unidades activas. Por tanto la variación del peso de una conexión entre una unidad i y otra j será nula si la neurona j no recibe activación por parte de la neurona i (no vence en presencia de un estímulo por parte de i , y se modificará (se reforzará) si es activada por dicha neurona i .

Un ejemplo de este tipo de aprendizaje es el desarrollado por kohonen conocido como Learning Vector Quantization (LVQ), aplicado a redes feedforward de dos capas. El LVQ puede aplicarse de forma diferente, según se precise obtener una o varias unidades vencedoras en la capa de salida. Existe también una extensión supervisada del LVQ basada en un mecanismo por corrección de error.

Una variación del aprendizaje supervisado aplicado a redes multicapa consiste en imponer una inhibición mutua entre neuronas únicamente cuando están a cierta distancia unas de otras (suponiendo que las neuronas se han dispuesto geométricamente, por ejemplo formando capas bidimensionales).

Existe entonces un área o región de vecindad alrededor de las neuronas que constituye su grupo local. Fukushima empleó esta idea en 1975 en una red multicapa llamada Cognitron, fuertemente inspirada en la

anatomía y fisiología del sistema visual humano, y en 1980 en una versión mejorada de la anterior, denominada Neocognitron. Esta red disponía de un gran número de capas con una arquitectura muy específica de interconexiones entre ellas, y era capaz de aprender a diferenciar caracteres, aunque estos se presentasen a diferente escala, en diferente posición o distorsionados.

El aspecto geométrico de la disposición de las neuronas de una red también es la base de un caso particular de aprendizaje competitivo introducido por Kohonen en 1982 conocido como feature mapping aplicado en redes con una disposición bidimensional de las neuronas de salida, que permiten obtener mapas topológicos o topográficos en los que, de algún modo, estarían representadas las características principales de las informaciones presentadas a la red. De esta forma, si la red recibe informaciones con características similares, se generarán mapas parecidos, puesto que serían afectadas neuronas de salida próximas entre sí.

Para concluir este apartado, se debe comentar la existencia de otro caso particular del aprendizaje competitivo, denominada teoría de la resonancia adaptativa desarrollado por Carpenter y Grossberg en 1986 y utilizado en la red feedforward de dos capas conocida como ART (en sus dos variantes: ART1, que trabaja con información binaria, y ART2, que maneja información analógica). Esta red realiza un prototipado de las informaciones que recibe a la entrada, generando como salida un ejemplar o prototipo que representa a todas las informaciones que podrían considerarse pertenecientes a la misma clase o categoría.

La teoría de la resonancia adaptativa se basa en la idea de hacer resonar la información de entrada con los prototipos de las categorías que reconoce la red; si entra en resonancia con alguno (es suficientemente similar), la red considera que pertenece a dicha categoría y únicamente realiza una pequeña adaptación del prototipo (para que se parezca algo más al dato presentado). Cuando no resuena con ningún prototipo, no se parece a ninguno de los existentes (recordados por la red) hasta ese momento, la red se encarga de crear

una nueva categoría con el dato de entrada como prototipo de la misma.

Tabla 3.2. Redes con aprendizaje no supervisado

Fuente: Modificado de [2]

Tipo		Modelo de Red
Aprendizaje Hebbiano	OFF LINE	Hopfield Learning Matrix Temporal Associative Memory (LAM) Optimal LAM Drive Reinforcement Fuzzy Associative Memory
	ON LINE	Additive Grossberg Shunting Grossberg Bidirectional Associative Memory (BAM) Adaptive BAM
Aprendizaje competitivo/cooperativo	OFF LINE	Learning Vector Quantizer Cognitron/Neocognitron Topology Preserving Map
	ON LINE	Adaptive Resonante Theory

3.2.3. Representación de la información de entrada y salida

Las redes neuronales pueden también clasificarse en función de la forma en que se representan las informaciones de entrada y las respuestas o datos de salida (Tabla 3.3.). Así, en un gran número de redes, tanto los datos de entrada como de salida son de naturaleza analógica: es decir, son valores reales continuos, normalmente estarán normalizados y su valor absoluto será menor que la unidad. Cuando esto ocurre, las funciones de activación de las neuronas serán también continuas, del tipo lineal o sigmoidal.

Otras redes, por el contrario, sólo admiten valores discretos o binarios a su entrada generando también salidas de este tipo. En este caso, la función de activación será del tipo escalón. Debido a su mayor sencillez, es habitual encontrar algunos modelos de redes reales que aunque inicialmente habían sido desarrollados como discretos por sus

autores, posteriormente se ha realizado una versión continua de los mismos, como el caso del modelo de Hopfield (Discrete Hopfield, Continuous Hopfield) y el denominado Adaptive Resonance Theory (ART1, ART2). Existe también un tipo de redes (que podrían denominarse híbridas) en las que las informaciones de entrada pueden ser valores continuos, aunque las salidas de la red son discretas.

Tabla 3.3. Clasificación de las redes neuronales en función al tipo de información de E/S

Fuente: Modificado de [2]

Redes Continuas E: Analógica S: Analógica	Redes Híbridas E: Analógica S: Binária	Redes Discretas E: Binária S: Binária
Backpropagation	Perceptrón	Discrete Hopfield
Brain State in a Box	Adaline/Madaline	Learning Matrix
Continuous Hopfield	Linear Associative	Temporal Associative
	Reward Penalty	Memory
LAM	Adaptive Heuristic Critic	Directional Ass. Memory
Optimal LAM		Cognitron/Neocognitron
Drive Reinforcement		Adaptive Resonante Theory I
Counter Propagation		Boltzmann Machine
Additive Grossberg		Cauchy Machine
Shunting Grossberg		
Adaptive BAM		
Learning Vector		
Quantizer		
TPM		
ART2		
Continuous Hopfield		

Taller 3

MANEJO DEL ALGORITMO DE APRENDIZAJE WIDROW-HOFF

El algoritmo LMS o algoritmo de aprendizaje Widrow-Hoff, está basado en un procedimiento descendente paso a paso. Aquí nuevamente, las redes lineales son entrenadas con ejemplos de un comportamiento correcto.

Widrow y Hoff plantearon la estimación del error cuadrático medio a través del uso del error cuadrático en cada iteración. Si se toma las derivadas parciales del error cuadrático con respecto a los pesos y biases en la k-ésima iteración se tiene:

$$\frac{\partial e^2(k)}{\partial w_{1,j}} = 2e(k) \frac{\partial e(k)}{\partial w_{1,j}}$$

para $j = 1, 2, \dots, R$ y

$$\frac{\partial e^2(k)}{\partial b} = 2e(k) \frac{\partial e(k)}{\partial b}$$

Luego se considera la derivada parcial con respecto al error.

$$\frac{\partial e(k)}{\partial w_{1,j}} = \frac{\partial [t(k) - a(k)]}{\partial w_{1,j}} = \frac{\partial}{\partial w_{1,j}} [t(k) - (Wp(k) + b)] \quad \text{o de manera equivalente}$$

$$\frac{\partial e(k)}{\partial w_{1,j}} = \frac{\partial}{\partial w_{1,j}} \left[t(k) - \left(\sum_{i=1}^R w_{1,i} p_i(k) + b \right) \right]$$

Aquí $p_i(k)$ representa el i-ésimo elemento del vector de entrada en la k-ésima iteración. De manera similar,

$$\frac{\partial e(k)}{\partial w_{1,j}} = -p_j(k)$$

Esto puede ser simplificado como:

$$\frac{\partial e(k)}{\partial w_{1,j}} = -p_j(k) \quad \text{además de}$$

$$\frac{\partial e(k)}{\partial b} = -1$$

Finalmente, el cambio en la matriz de pesos y el bias es representado como: $2\alpha e(k)p(k)$ y $2\alpha e(k)$. Estas dos ecuaciones forman la base del algoritmo de aprendizaje Widrow-Hoff (LMS)

Esos resultados pueden ser extendidos al caso de neuronas múltiples, y escritos en forma matricial de la siguiente manera:

$$\begin{aligned} W(k+1) &= W(k) + 2\alpha e(k)p^T(k) \\ b(k+1) &= b(k) + 2\alpha e(k) \end{aligned}$$

Aquí el error e y el bias b son vectores y α es una tasa de aprendizaje. Si α es grande, el aprendizaje ocurre de manera rápida, pero si es demasiado grande, puede conducir a inestabilidad y al posible incremento de errores. Para asegurar algoritmos estables, la tasa de aprendizaje debe ser menor que el recíproco del más grande eigenvalor de la matriz de correlación $p^T p$ de los vectores de entrada.

Afortunadamente se cuenta con una función en el toolbox denominada **learnwh** que realiza todos los cálculos precedentes mostrados. Dicha función calcula el cambio en los pesos como:

$$dw = lr * e * p'$$

y el cambio en el bias de la siguiente manera:

$$db = lr * e$$

Ejercicios propuestos 3

1. Analice la función **learnwh** ejecutando `>>help learnwh`. Comente los resultados.
2. Analice la función **maxlinlr** ejecutando `>>help maxlinlr`. Comente los resultados.

4 PERCEPTRONES

El dispositivo conocido como perceptrón fue inventado por el fisiólogo Frank Rosenblatt a finales de 1950. Este modelo intenta “ilustrar parte de las propiedades fundamentales de los sistemas inteligentes en general, sin convertirse en el engrane profundamente especial y frecuentemente desconocido, los cuales son condiciones sostenibles para los organismos biológicos particulares”. Rosenblatt creyó que la conectividad que desarrollan las redes biológicas contiene un elemento aleatorio grande. Así, tomó investigaciones anteriores, como de McCulloch-Pitts, dónde la lógica simbólica fue empleada para analizar mejor las estructuras idealizadas. Más bien, Rosenblatt pensó que la herramienta del análisis más apropiada era la teoría de probabilidad. Él desarrolló una teoría de separabilidad estadística que utilizó para caracterizar las propiedades de los elementos interconectados aleatoriamente formando redes [5].

El fotoperceptrón es un dispositivo que responde a los modelos ópticos. La figura 4.1. muestra un ejemplo. En este dispositivo, la luz choca con en el sensor (S) que son los puntos de la estructura de la retina. Cada punto de S responde a todos o a ninguno en la manera que entra la luz. Pulsos generados por los puntos de S se transmiten a unidades en la capa de la asociación (A). Cada unidad A se conecta al conjunto aleatorio de S puntos, llamado unidad origen del conjunto A , las conexiones pueden ser activadoras o inhibitorias. Las conexiones tienen tres posibles valores, +1, -1 y 0. Cuando un modelo del estímulo aparece en la retina, una unidad A se vuelve activa si la suma de sus entradas excede algún valor umbral. Si se activa, la unidad A produce una salida que se envía a la próxima capa de unidades. De manera similar, las unidades de A son conectadas a las unidades de respuesta (R) [5]. En la figura 4.1. un fotoperceptrón simple tiene un área sensoria, un área asociadora y un área de respuesta, las conexiones son entre unidades de varias áreas.

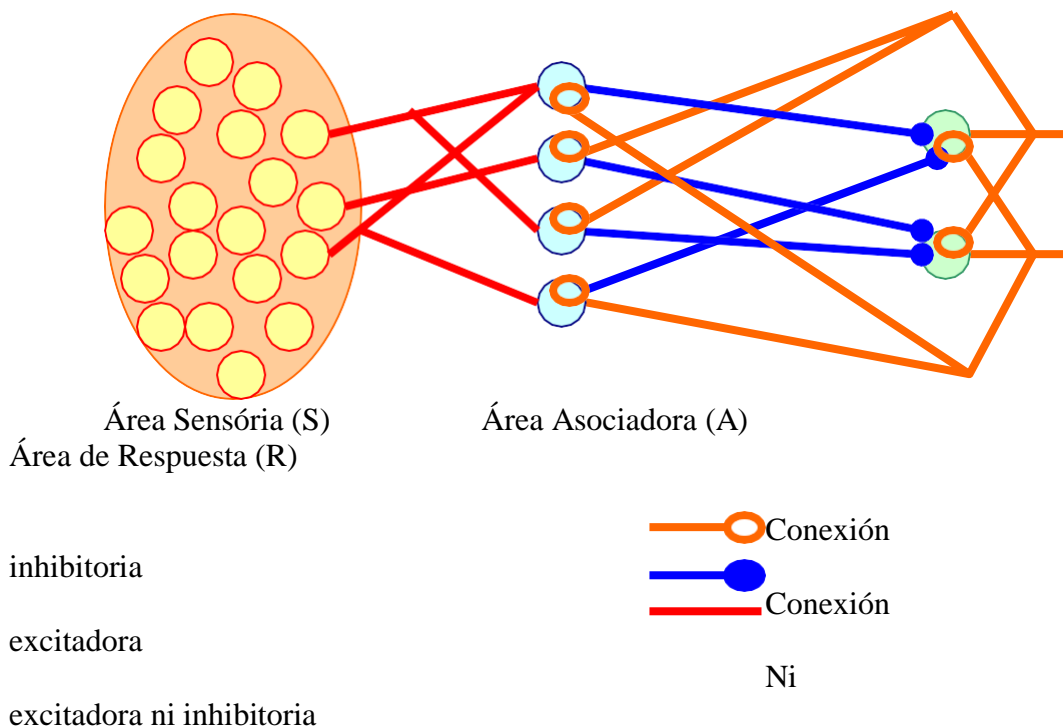


Figura 4.1. Fotoperceptrón

Fuente: Modificado de [5]

Rosenblatt creó muchas variaciones del perceptrón. Una de las más simples es la red en la cual los pesos y el bias son ajustados para producir un vector de salidas deseadas cuando se le presente un correspondiente vector de entrada. La técnica de entrenamiento usada es llamada regla de aprendizaje del perceptrón. Los perceptrones generaron un gran interés debido a su habilidad de generalizar los vectores de entrenamiento y aprender de conexiones iniciales distribuidas aleatoriamente [3].

Los perceptrones están especialmente preparados para resolver el problema de clasificación de patrones. Son redes rápidas y confiables para problemas que pueden resolver. Son base para el entendimiento de la operación de redes más complejas [3].

4.1. MODELO NEURONAL

Una neurona perceptrón, que usa una función de transferencia escalón (*hardlim*) que se muestra en la figura 4.2.

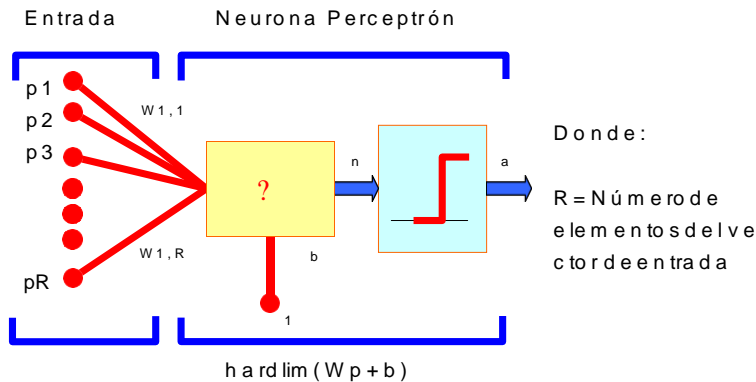


Figura 4.2. Neurona perceptrón

Fuente: Tomado de [3]

Cada entrada externa tiene asociado un peso denominado w_{ji} , y la suma del producto de los pesos con las entradas que pasan por la función de transferencia escalón (*hardlim*), que también cuenta con una entrada de 1, representando el bias. La función de transferencia escalón, tal como se puede observar en la figura 4.3., retorna un 0 ó 1 [3].

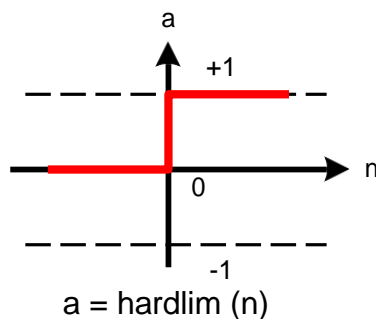


Figura 4.3. Función de transferencia escalón

Fuente: Tomado de [3]

En la figura 4.4. dos regiones de la clasificación son formadas por la decisión límite de la línea L a $Wp + b = 0$. Esta línea es perpendicular a la matriz de pesos W y al cambio según el bias b . Los vectores de la entrada encima y a la izquierda de la línea L resultaran en un neto, con un valor mayor que 0 y la neurona con una función de transferencia límite produce una salida de 1. Los vectores de la entrada debajo y en el lado derecho de la línea L causan una salida de la neurona igual a 0. La línea divisoria puede orientarse y moverse para clasificar el espacio de la entrada deseado de acuerdo a los valores de los pesos y el bias [3].

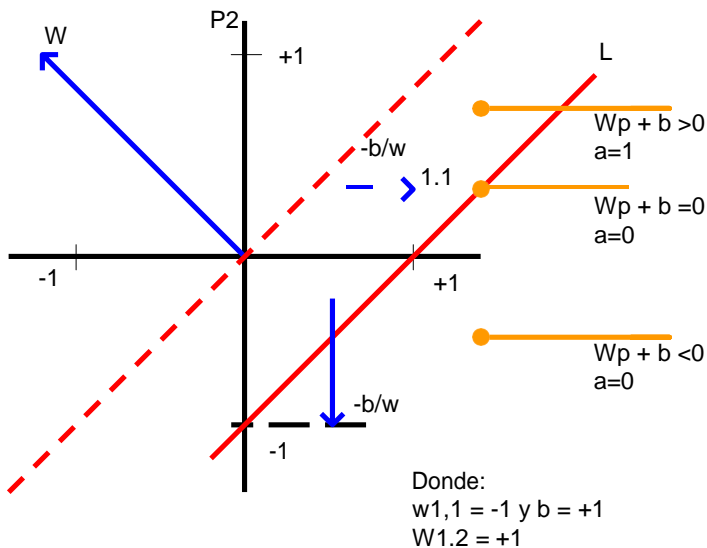


Figura 4.4. Regiones de clasificación
Fuente: Tomado de [3]

Las neuronas sin el bias siempre tendrán una línea de clasificación que pasa por el origen. El bias permite a la neurona para resolver problemas dónde dos conjuntos de vectores de la entrada están en lados diferentes del origen [3].

4.2. ARQUITECTURA

La red de perceptrón consiste en una sola capa de S neuronas de perceptrón conectadas a las R entradas a través de un conjunto de pesos w_{ji} . El índice i y j indican que el peso w_{ji} es la conexión de la entrada i a la neurona perceptrón j [3].

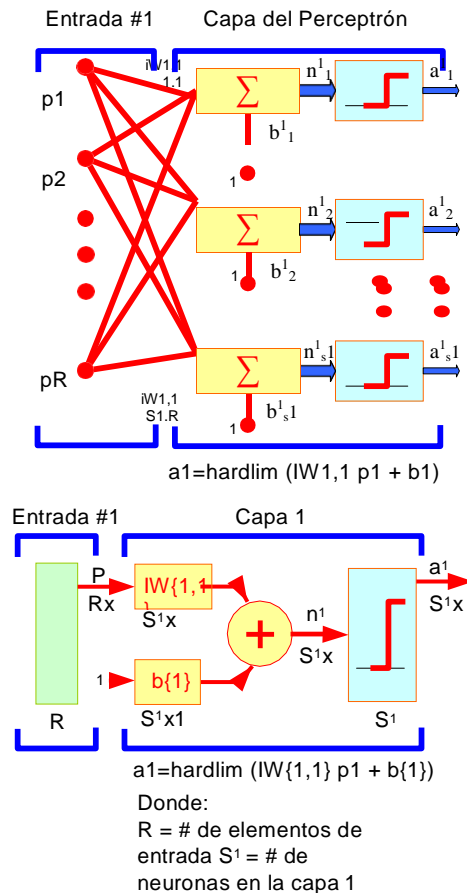


Figura 4.5. Arquitectura del perceptrón

Fuente: Tomado de [3]

La regla de aprendizaje del perceptrón describe la capacidad de entrenamiento de una capa, esta restricción da limitaciones de cálculo al perceptrón [3].

4.3. REGLA DE APRENDIZAJE DEL PERCEPTRÓN

Los perceptrones son entrenados con ejemplos de comportamiento deseado. El comportamiento deseado se resume en un conjunto de pares entrada-salida, $p_1t_1, p_2t_2, \dots, p_Qt_Q$ donde p es una entrada a la red y t es su correspondiente salida deseada (objetivo). El objetivo es reducir el error e , que es la diferencia entre la respuesta de la neurona a y el vector t de salidas deseadas [3].

Hay tres condiciones que pueden ocurrir en una neurona de acuerdo al vector de entrada p que es presentado a la red [3]:

- CASO 1:** Si un vector de entrada es presentado y la salida es correcta ($a = t$, y $e = t - a = 0$), la matriz de pesos w no es alterada.
- CASO 2:** Si la salida de la neurona es 0 y se desea 1 ($a = 0$ y $t = 1$, y $e = t - a = 1$) el vector de entrada p es añadido a la matriz de pesos w . Esto hace que la matriz de pesos se aproxime al vector de entrada, incrementando la oportunidad de que en el futuro la red clasifique como 1.
- CASO 3:** Si la salida de la neurona es 1 y se desea 0 ($a = 1$ y $t = 0$, y $e = t - a = -1$) el vector de entrada p sustraído de la matriz de pesos w . Esto hace que la matriz de pesos se aproxime al vector de entrada, incrementando la oportunidad de que en el futuro la red clasifique como 0.

La regla de aprendizaje del perceptrón puede ser escrita brevemente en términos de error $e = t - a$, y el cambio hecho en la matriz de pesos es Δw :

- CASO 1:** Si $e = 0$, hacer un cambio Δw es igual a 0
- CASO 2:** Si $e = 1$, hacer un cambio Δw es igual a p^T
- CASO 3:** Si $e = -1$, hacer un cambio Δw es igual a $-p^T$

Los tres cambios son escritos en la siguiente expresión:

$$\Delta w = (t-a)p^T = ep^T \quad (14)$$

El cambio expresado en una neurona con bias es simplificado a un peso con una entrada igual a 1.

$$\Delta b = (t - a)I = e \quad (15)$$

Para el caso de una capa de neuronas se tiene:

$$\Delta W = (t - a)(p)^T = e(p)^T \text{ y } \Delta b = (t - a) = e \quad (16)$$

La regla de aprendizaje del perceptrón es resumida como sigue:

$$W^{new} = W^{old} + ep^T \text{ y } b^{new} = b^{old} + e \quad (17)$$

donde $e = t - a$

El proceso de hallar nuevos pesos (y bias) se repite hasta que no exista error. Se debe notar que la regla de aprendizaje garantiza una convergencia en un número finito de pasos para todos los problemas que puede resolver el perceptrón. Esto incluye todos los problemas que son “linealmente separables”. Los objetos a ser clasificados deben ser separados con una línea.

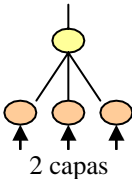
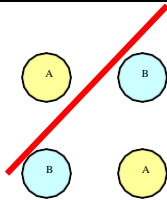
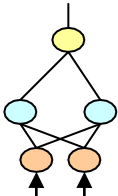
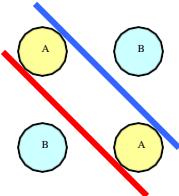
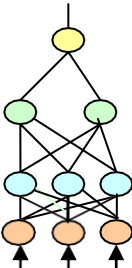
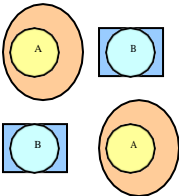
4.4. PERCEPTRÓN MULTINIVEL

Un perceptrón multinivel o multicapa es una red con conexiones hacia delante, compuesta de varias capas de neuronas entre la entrada y la salida de la misma, esta red permite establecer regiones de decisión mucho más complejas que las de dos semiplanos, como lo hace el perceptrón de un solo nivel [6].

La estructura de la conexión del perceptrón es de alimentación hacia delante y con tres capas. La primera capa es una memoria, en la que se almacena los datos sensoriales. Los elementos de la primera capa son conectadas totalmente arbitrariamente a los elementos de la segunda capa, llamada la “capa oculta”. Cada neurona de esta capa combina la información que viene de diferentes elementos sensores y representa un posible patrón. Las neuronas de esta capa son totalmente conectadas

a las neuronas de la capa de salida llamada “capa perceptrón”. Los pesos entre la capa de entrada y la capa de salida son hallados; esto porque usualmente solo dos capas son presentadas gráficamente. Esta es la razón de porque los perceptrones a veces son llamados “redes de capa singular” [6].

Tabla 4.1. Distintas formas de las regiones generadas por un Perceptrón Multinivel
Fuente: Modificado de [6]

Estructura	Regiones de decisión	Problema de XOR
 <p>2 capas</p>	Medio plano limitado por un hiperplano.	
 <p>3 capas</p>	Regiones cerradas o convexas	
 <p>4 capas</p>	Arbitraria complejidad limitada por el número de neuronas	

4.5. LIMITACIONES

Las redes perceptrón tienen serias limitaciones. Primero, las salidas solo toman dos valores (0 ó 1) debido a la función de transferencia. Segundo, los perceptrones sólo clasifican problemas linealmente separables [3].

Taller 4

PERCEPTRONES

1. Creación de un perceptrón

Un perceptrón se crea con la función *newp*

net = newp(PR,S)

Donde los argumentos de entrada son:

PR: Matriz de $R \times 2$ de valores mínimos y máximos del vector de entrada R

S: Número de neuronas

Comúnmente la función de transferencia *hardlims* es usada en los perceptrones, por lo que ésta función se asume por defecto. La regla de aprendizaje es *learnp* ó *learnpn*

2. Simulación

La simulación del funcionamiento de la red neuronal se ejecuta con *sim*

a = sim (net, P)

Donde los argumentos son:

net: Red perceptrón creada

P: Patrones de prueba para la simulación

a: Resultado de la simulación

3. Entrenamiento adaptativo

El entrenamiento adaptativo se realiza a través de la siguiente orden

[newnet,a,e] = adapt(net,P,T);

Donde los argumentos son:

net: Red perceptrón creada anteriormente

P: Vector de entrada a la red

T: Vector de salida asociado a la entrada **P**

newnet: Red entrenada adaptativamente

a: Salida de la red

e: Error

4. Entrenamiento por lotes

El entrenamiento por lotes se realiza a través de la orden *train*

newnet = train(net,P,T);

Donde los argumentos son:

net: Red perceptrón creada

P: Vector de entrada a la red

T: Vector de de salida asociado a la entrada **P**

newnet: Red entrenada por lotes

5. Gráficas del perceptrón

Para obtener gráficas del perceptrón se utilizan las siguientes opciones:

plotpv grafica un perceptrón con los vectores de entradas y de salidas, las entradas deben ser menores o iguales a 3 patrones.

plotpv(P,T)

plotpc grafica una línea de clasificación tomando en cuenta los pesos y el bias



plotpc(W,b)

6. Ejemplo de perceptrón con la función OR

1. Descripción del sistema

Asociado al problema de la función OR se tiene la tabla de verdad, donde

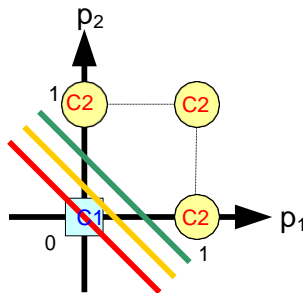
$1=V, 0=F$

<u>p₁</u>	<u>p₂</u>	<u>OR=t</u>		
0	0	0	}	C1 
0	1	1		
			}	C2 

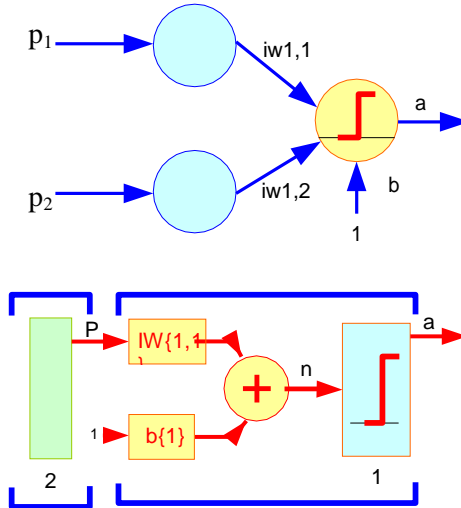
1	0	1
1	1	1

En la tabla de verdad se observa dos clases C1 (cuadrado) y C2 (círculo) que corresponden a las salidas de la función OR para 4 patrones, también se distingue un aprendizaje supervisado, ya que se tiene la tupla (P,T) que son los patrones de entrada y las salidas deseadas.

Teorema de separabilidad lineal.- El teorema propuesto por Rosenblatt es utilizado para ver la separabilidad lineal de las clases del problema.



Debido a que la función **OR** presenta una separabilidad lineal clara, ya que las clases pueden ser separadas por una línea ó por un hiperplano. Entonces el modelo a utilizar sería un perceptrón simple con dos entradas (p_1 , p_2) y una salida (a).



La función de transferencia por la capa de entrada es lineal (*purelin*) y por la capa de salida es límite ó escalón (*harlim*).

donde $a = \text{harlim}(IW\{1,1\}P + b\{1\}) = \text{hardlim}(iw_{1,1}p_1 + iw_{1,2}p_2 + b)$

2. Especificación del diseño

Funciones

$$a = \begin{cases} 0 & \text{si } p_1 = 0 \text{ y } p_2 = 0 \\ 1 & \text{e.o.c.} \end{cases}$$

a) Abstracción de datos

A continuación se definen los datos usados en la red neuronal:

Atributo	Valor	Tipo	Tamaño
p_1	$\{0,1\}$	$N Z^+$	9 N(1)
p_2	$\{0,1\}$	$N Z^+$	9 N(1)
$iw_{1,1}$	$[-1,1]$	R^+	9.99 N(1,2)
$iw_{1,2}$	$[-1,1]$	R^+	9.99 N(1,2)
t	$\{0,1\}$	$N Z^+$	9 N(1)
a	$\{0,1\}$	$N Z^+$	9 N(1)

b	$[-1,1]$	R^+	9.99	$N(1,2)$
---	----------	-------	------	----------

b) Abstracción procedimental

Para la abstracción procedimental se utiliza el pseudocódigo.

Algoritmo_Perceptron_Simple ()

P1: Introducción (leer) de p_1, p_2 y la función lógica (t =salida deseada)

P2: Categorizar de acuerdo a la función lógica

P3: Verificar la Separabilidad Lineal

P4: Si no es separable, entonces imprimir “No es perceptrón simple”: FIN

P5: Definir la topología del perceptrón simple

P6: Definir los pesos

P7: Definir la función de activación

P8: Mostrar la arquitectura y función de activación

P9: Entrenar la red

P10: Simular la red

FIN_Algoritmo_Perceptron_Simple

3. Implementación

%Describir un conjunto de cuatro patrones, con dos elementos para el vector de entradas y su %correspondiente salida deseada.

P = [0 0 1 1; 0 1 0 1];

T = [0 1 1 1];

plotpv(P,T)

%Primeramente se crea la red neuronal con una neurona de salida

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

%Posteriormente se simula la red para ver el estado de los pesos

a = sim(net,P)

%El entrenaminto modifica los pesos

net = train(net,P,T);

plotpc(net.IW{1,1},net.b{1})

%Se simula la red para ver el resultado del entrenamiento
a = sim(net,P)

Ejercicios propuestos 4

1. Construir una red neuronal artificial de tipo perceptrón para la función **AND**.
2. Construir una red neuronal artificial de tipo perceptrón con tres entradas, para la función **X AND Y OR Z**.
3. Construir una red neuronal artificial de tipo perceptrón para la función **XOR** y evaluar los resultados obtenidos.

5 RETROPROPAGACIÓN

Hay muchas aplicaciones potenciales que son difíciles de implementar por que hay muchos problemas inadecuados a la solución que siguen un proceso secuencial. Las aplicaciones deben tener un rendimiento complejo de la interpretación de los datos, todavía no se tiene una función de mapeo para describir el proceso de interpretación, o ellos deben proveer una “mejor conjetura” de la salida cuando se presenta una entrada de datos ruidosos [5].

Una red neuronal halla útil enviar problemas que requieren el reconocimiento de patrones complejos y rendimiento de mapeo no trivial a la red de retropropagación (backpropagation (BPN)). Formalizada por Werbos, Parker y posteriormente por Rumelhart y McClelland [5].

La red de retropropagación fue creada a través de la generalización de la regla de aprendizaje para redes multicapa Widrow-Hoff y funciones de transferencias no lineales y diferenciables. Vectores de entrada y su correspondiente vector de salida son usados en el entrenamiento de la red hasta que pueda aproximar una función, asociar los vectores de entrada con los vectores de salida o clasificar los vectores de entrada de manera apropiada con la definición del supervisor. Redes con bias, capa sigmoideal y una capa de salida lineal son capaces de aproximar cualquier función con un número finito de discontinuidades. El algoritmo de retropropagación normal es un algoritmo del gradiente descendente como la regla de aprendizaje de Widrow-Hoff. El término retropropagación se refiere a la manera en que el gradiente es calculado para redes no lineales multicapa. Propiamente el entrenamiento en las redes retropropagación tiende a dar respuestas razonables cuando se le presentan entradas que no vio anteriormente. Típicamente, una nueva entrada puede llevar a una salida similar, siendo una salida correcta para el vector de entradas usado en el entrenamiento que es similar a las entradas presentadas [3].

5.1. APROXIMACIÓN AL ALGORITMO DE RETROPROPAGACIÓN

De hecho, un algoritmo de búsqueda incluso con una entrada relativamente pequeña puede probar ser de un alto consumo de tiempo. El problema es la naturaleza secuencial de la computadora; el ciclo de la arquitectura de Von Neumann permite que la máquina tenga un rendimiento de operación en un tiempo. En muchos casos, el tiempo requerido por una computadora para el rendimiento de cada instrucción es muy pequeña (típicamente alrededor de un millonésimo de segundo) el tiempo agregado que requiere para un programa grande es insignificante para los usuarios humanos. Sin embargo, para las aplicaciones esta búsqueda debe ser extensa a través del espacio de entrada, o intentar correlacionar todas las permutaciones posibles de un patrón complejo, el tiempo requerido por las máquinas rápidas pueden volverse intolerables. Las redes multicapa aproximan a la función deseada por el proceso paralelo y que es altamente conexionista [5].

5.2. OPERACIÓN

Para empezar, la red aprende un conjunto de ejemplos de pares entrada-salida que son usadas en dos fases, ciclos de propagar y adaptar. Después de que un patrón de entrada es aplicado como un estímulo a la primera capa de unidades de neuronas, es propagada a cada capa superior hasta que la salida es generada. Este patrón de salida luego es comparado con la salida deseada, y una señal de error es calculada por cada unidad de salida. Las señales de error son transmitidas hacia atrás, de la capa de salida a cada nodo intermedio que contribuya directamente con la salida. No obstante, cada unidad de las capas intermedias recibe sólo una porción del error total, basados en la aproximación de la contribución relativa de la unidad que hace la salida original. Este proceso se repite, capa por capa, hasta que cada nodo en la red recibe una señal de error que describe su contribución relativa al error total [5].

Basados en la señal de error recibida, los pesos de las conexiones son modificados para todas las unidades que causan que la red converja hacia un estado que permite que se codifiquen todos los patrones de entrenamiento. La significancia de este proceso es el entrenamiento, los nodos en las capas intermedias se organizan como distintos nodos para aprender a reconocer diferentes características del espacio total de entradas. Después del entrenamiento, cuando son presentados nuevos patrones arbitrarios que son ruidosos o incompletos, las unidades de la capa oculta de la red pueden responder con la activación de la salida si el nuevo patrón de entrada contiene un patrón que se parece a las características particulares aprendidas para ser reconocidas en el entrenamiento. Recíprocamente, las unidades de las capas ocultas tienden a inhibir las salidas si los patrones de entrada no contienen las características que fueron entrenadas a reconocer [5].

5.3. ARQUITECTURA

La arquitectura que es comúnmente usada para el algoritmo retropropagación son las redes multicapa. Una neurona con R entradas, cada entrada es pesada con un w apropiado. La suma de las entradas pesadas y el bias forman la entrada a la función de transferencia f . En las neuronas se utilizan cualquier función de transferencia diferenciable para generar la salida [3].

Las redes multicapa a menudo usan una función de transferencia *Sigmoide*.

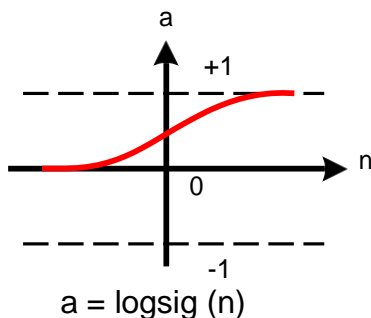


Figura 5.1. Función de transferencia sigmoide

Fuente: Tomada de [3]

La función *logsig* genera salidas entre 0 y 1, ya que la neurona de entrada va de una infinidad de valores negativos a positivos. Alternativamente, las redes multicapa pueden usar la función de transferencia tangente sigmoideal (*tansig*) [3].

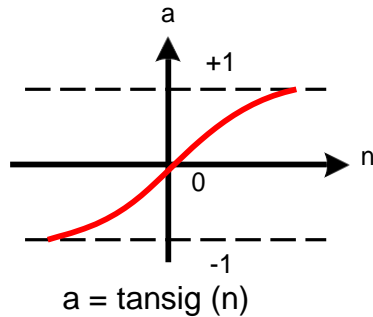


Figura 5.2. Función de transferencia tangente sigmoideal
Fuente: Tomada de [3]

Ocasionalmente, la función de transferencia lineal es usada en las redes retropropagación (*purelin*).

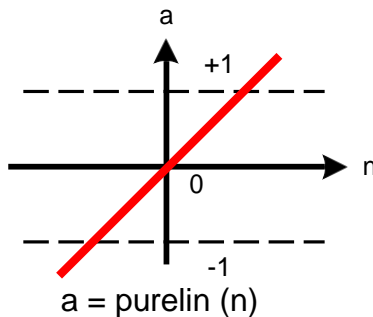


Figura 5.3. Función de transferencia lineal
Fuente: Tomada de [3]

Si la última capa de la red tiene neuronas sigmoideales, entonces la salida de la red está limitada en un rango pequeño. Si son neuronas de

salidas lineales las salidas de la red pueden tomar cualquier valor. En el algoritmo retropropagación es importante calcular las derivadas de cualquier función de transferencia usada. Cada función de transferencia *tansig*, *logsig* y *purelin* tienen una correspondiente función derivada: *dtansig*, *dlogsig* y *dpurelin* [3].

Redes de conexión hacia adelante a menudo tienen muchas capas ocultas de neuronas sigmoidales finalizando con una capa de salida de neuronas lineales. Múltiples capas de neuronas con una función de transferencia no lineal permite a la red aprender relaciones lineales o no lineales entre los vectores de entrada y salida. La salida de la capa lineal toma valores entre -1 a +1. Por otra parte, si es deseable restringir las salidas de la red (valores entre 0 y 1) entonces la capa de salida debe usar un número determinado de capas que expongan las matrices de pesos. La notación apropiada usada en dos capas *tansig/purelin* se muestra en la figura 5.4 [3].

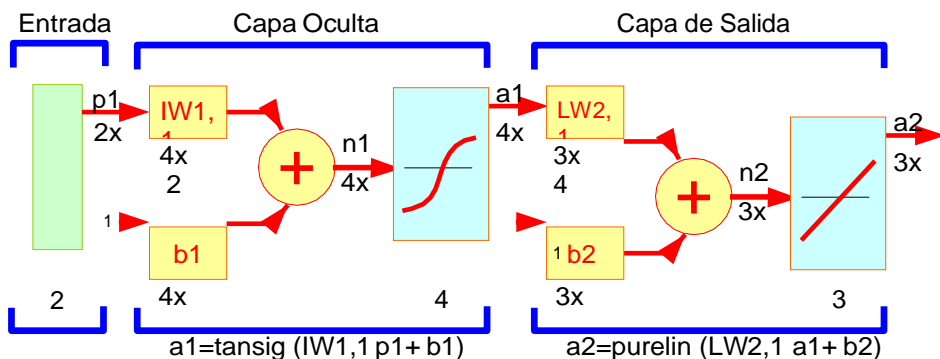


Figura 5.4. Red multicapa
Fuente: Tomada de [3].

La red neuronal de la figura 5.4. puede ser usada para aproximar cualquier función con un número finito de discontinuidades, las neuronas de la capa oculta son colocadas arbitrariamente [3].

5.4. APRENDIZAJE POR RETROPROPAGACIÓN

El algoritmo de retropropagación tiene dos fases, una hacia adelante y otra hacia atrás. Durante la primera fase el patrón de entrada es presentado a la red y propagado a través de las capas hasta llegar a la capa de salida. Obtenidos los valores de salida de la red, se inicia la segunda fase, comparándose estos valores con la salida esperada para obtener el error. Se pasa a la capa anterior con una retropropagación del error, ajustando convenientemente los pesos y continuando con este proceso hasta llegar a la primera capa. De esta manera se han modificado los pesos de las conexiones de la red para cada ejemplo o patrón de aprendizaje del problema, se conoce el valor de entrada y salida deseada que debería generar la red ante dicho patrón. Para aplicar este entrenamiento se siguen los siguientes pasos y fórmulas [6]:

Paso 1

Inicializar los pesos de la red con valores pequeños aleatorios

Paso 2

Presentar un patrón de entrada, $P_p = p_{p1}, p_{p2}, \dots, p_{pN}$, y especificar la salida deseada que debe generar la red: t_1, t_2, \dots, t_m (si la red se utiliza como un clasificador, todas las salidas deseadas serán cero, salvo una, que será la de la clase a la que pertenece el patrón de entrada).

Paso 3

Calcular la salida actual de la red, para ello se presenta las entradas a la red y se calcula la salida que presenta cada capa hasta llegar a la capa de salida, ésta será la salida de la red a_1, a_2, \dots, a_M .

Los pasos son los siguientes:

1. Se calculan las entradas netas para las neuronas ocultas procedentes de las neuronas de entrada.
2. Para una neurona j oculta:

$$n_{pj}^h = \sum_{i=1}^N w_{ji}^h p_i + b_j \quad (18)$$

en donde el índice h se refiere a magnitudes de la capa oculta (hidden); el subíndice p , al p -ésimo vector de entrenamiento, y j a la j -ésima neurona oculta. El término b (bias) puede ser opcional, pues actúa como una entrada más.

3. Se calculan las salidas de las neuronas ocultas:

$$a_{pj} = f^h_i (n^h_{pj}) \quad (19)$$

4. Se realizan los mismos cálculos para obtener las salidas de las neuronas de salida (capa o: output)

$$n^o_{pj} = \sum_{k=1}^L w^o_{kj} a_{pj} + b^o_k \quad (20)$$

$$a_{pk} = f^o_k (n^o_{pk}) \quad (21)$$

Paso 4

Calcular los términos de error para todas las neuronas.

Si la neurona k es una neurona de la capa de salida, el valor de delta es:

$$\sigma^o_{pk} = (t_{pk} - a_{pk}) f'^o_k (n^o_{pk}) \quad (22)$$

La función f debe cumplir el requisito de ser derivable, lo que implica la imposibilidad de utilizar una función escalón. En general, se dispone de dos formas de función de salida que sirven: la función lineal de salida ($f_k(n_{jk}) = n_{jk}$) y la función sigmoideal definida por la expresión:

$$f_k(n_{jk}) = 1 / (1 + e^{-n_{jk}}) \quad (23)$$

La selección de la función de salida depende de la forma en que se decida representar los datos de salida: si se desea que las neuronas de salida sean binarias, se utiliza la función sigmoideal, puesto que esta función es casi bienestable y, además derivable. En otros casos es tan aplicable una función como otra.

Para la función lineal, se tiene: $f'_{pk} = 1$, mientras que la derivada de una función f sigmoidal es:

$$f'_{pk} = f_{pk}(1 - f_{pk}) = a_{pk}(1 - a_{pk}) \quad (24)$$

por lo que los términos de error para las neuronas de salida quedan:

$$\sigma^o_{pk} = (t_{pk} - a_{pk}) \quad (25)$$

para la salida lineal, y

$$\sigma^o_{pk} = (t_{pk} - a_{pk})a_{pk}(1 - a_{pk}) \quad (26)$$

para la salida sigmoidal.

Si la neurona j no es de salida, entonces la derivada parcial del error no puede ser evaluada directamente. Por tanto, se obtiene el desarrollo a partir de valores que son conocidos y otros que pueden ser evaluados.

La expresión obtenida en este caso es:

$$\sigma^h_{pk} = f'^h_j(n^h_{pj}) \sum_k \sigma^o_{pk} w^o_{kj} \quad (27)$$

donde se observa que el error en las capas ocultas depende de todos los términos de error de la capa de salida. De aquí surge el término de propagación hacia atrás. En particular para la función sigmoidal:

$$\sigma^h_{pj} = p_{pj}(1 - p_{pj}) \sum_k \sigma^o_{pk} w^o_{kj} \quad (28)$$

donde k se refiere a todas las neuronas de la capa superior a la de la neurona j . Así, el error que se produce en una neurona oculta es proporcional a la suma de los errores conocidos que se producen en las neuronas a las que está conectada la salida de ésta, multiplicando cada uno de ellos por el peso de la conexión. Los umbrales internos de las neuronas se adaptan de forma similar, considerando que están conectados con los pesos desde entradas auxiliares de valor constante.

Paso 5

Actualización de los pesos

Para ello, se utiliza el algoritmo recursivo, comenzando por las neuronas de salida y trabajando hacia atrás hasta llegar a la capa de entrada, ajustando los pesos de la forma siguiente:

Para los pesos de las neuronas de la capa de salida:

$$w_{kj}^o(k+1) = w_{kj}^o(k) + \Delta w_{kj}^o(k+1); \quad (29)$$

$$\Delta w_{kj}^o(k+1) = \alpha \sigma_{pj}^o a_{pj} \quad (30)$$

Y para los pesos de las neuronas de la capa oculta:

$$w_{ji}^h(k+1) = w_{ji}^h(k) + \Delta w_{ji}^h(k+1); \quad (31)$$

$$\Delta w_{ji}^h(k+1) = \alpha \sigma_{pj}^h a_{pj} a_{pi} \quad (32)$$

Paso 6

El proceso se repite hasta que el término de error

$$E_p = (1/2) \sum_{k=1}^M \sigma_{pk}^2 \quad (33)$$

resulta aceptablemente pequeño para cada uno de los patrones aprendidos.

5.5. MEJORA DE LA GENERALIZACIÓN

Uno de los problemas que ocurre en el entrenamiento de las redes neuronales es el llamado sobreajustamiento. El error del conjunto de entrenamiento es conducido a valores muy pequeños, pero cuando nuevos datos son presentados a la red el error es grande. La red memoriza los ejemplos de entrenamiento, pero no aprende a generalizar nuevas situaciones [3].

Un método para mejorar la generalización es usar una red que tenga un gran ajuste para proveer una adecuada adaptación. Redes grandes, tienen más complejidad de la que ellas pueden crear. Si se usa una red bastante pequeña, no tiene tanto poder para ajustar los datos. Hay dos métodos para mejorar la generalización que son la regularización y la parada temprana [3].

- a) **Regularización.** Este involucra la modificación de la función de rendimiento, que es normalmente escogida para la suma de errores cuadráticos de la red del conjunto de entrenamiento [3].

Modificación de la función de rendimiento. La típica función de rendimiento usada en redes de alimentación hacia delante es la media de la suma de los errores cuadráticos [3]:

$$F = mse = \frac{1}{N} \sum_{i=1}^N (e_i)^2 = \frac{1}{N} \sum_{i=1}^N (t_i - a_i)^2 \quad (34)$$

Es posible mejorar la generalización si se modifica la función de rendimiento con la adición de un término que consiste de la media de la suma de los errores cuadráticos de los pesos y el bias de la red [3]:

$$msereg = \gamma mse + (1 - \gamma) msw, \quad (35)$$

donde γ es la proporción de rendimiento, y

$$msw = \frac{1}{N_j} \sum_j^n w_j^2 \quad (36)$$

El problema con regularización es que es dificultoso determinar un valor óptimo para la proporción de rendimiento γ . Si este parámetro es muy grande, se obtiene un sobreajuste. Si la proporción es muy pequeña, la red no se ajusta adecuadamente a los datos de entrenamiento [3].

- b) **Parada temprana.** Otro método para mejorar la generalización es llamado la parada temprana. Esta técnica divide los datos disponibles en tres subconjuntos. El primer subconjunto es el conjunto de entrenamiento que es usado para calcular el gradiente y modificar los pesos y los bias. El segundo

subconjunto es el conjunto de validación, el error de la validación es monitoreado mientras dura el proceso de entrenamiento. El error de validación normalmente decrementa mientras dura la fase inicial del entrenamiento. Sin embargo, cuando la red empieza a sobre ajustar los datos, el error de la validación típicamente se eleva. Cuando el error de la validación se incrementa por un número específico de iteraciones, el entrenamiento se detiene, los pesos y el bias con el mínimo error de validación son retornados. El conjunto de errores de prueba no es usado mientras dura el entrenamiento, pero es usado para comparar los diferentes modelos. Es muy útil graficar el conjunto de errores de prueba en el proceso de entrenamiento. Si el error del subconjunto de prueba cambia al mínimo número de iteraciones significativo al conjunto de validación, esto indica una pobre división del conjunto de datos [3].

5.6. CONSIDERACIONES PRÁCTICAS

Algunas consideraciones prácticas para el diseño de la red neuronal de retropropagación de errores son anotadas a continuación.

- a) **Datos de entrenamiento.-** Una consideración práctica es la definición de términos que son suficientemente y propiamente considerados en la selección del entrenamiento de los pares para el BNP. Desafortunadamente, no existe una única definición que se aplique a todos los casos. Como muchos aspectos de los sistemas de redes neuronales, la experiencia es a menudo el mejor maestro, se obtiene facilidad con el uso de redes neuronales en la manera de apreciar como seleccionar y preparar un conjunto de entrenamiento. En general, se utilizan muchos datos disponibles para el entrenamiento de la red, aunque se pueden necesitar usarlos todos. De los datos disponibles para el entrenamiento, un pequeño subconjunto es frecuentemente todo lo que se necesita para entrenar satisfactoriamente una red. El resto de los datos puede ser usado para las pruebas de la red verificando el rendimiento del mapeo deseado sobre los vectores de entrada que nunca se enfrentaron en el entrenamiento [5].

b) Tamaño de la red.- El problema es saber ¿cuántos nodos se necesitan para resolver un problema en particular? ¿tres capas son siempre suficientes? Estas preguntas son acerca de los datos de entrenamiento, no hay respuestas estrictas para responder a estas preguntas. Generalmente, tres capas son suficientes. A veces, un problema parece fácil de resolver con más de una capa oculta. En este caso “más fácil” significa que la red aprende rápidamente. El tamaño de la red es usualmente dictado por la naturaleza de la aplicación. Se pueden determinar un número de nodos de salida para decidir si se busca valores analógicos o valores binarios en las unidades de salida. Determinar el número de unidades en la capa oculta no es proporcional a las capas de entrada y salida. El objetivo principal es usar pocas unidades en la capa oculta como sea posible, por que cada unidad añade peso a la computadora mientras dura la simulación. Por supuesto, en un sistema que es totalmente implementado en hardware (un procesador por elemento de proceso), adicionalmente el peso de la computadora no necesita mucha consideración (la comunicación del interprocesador puede ser un problema). Para un tamaño de red razonable (cientos o miles de entradas), el tamaño que necesita la capa oculta es una fracción relativamente pequeña de la capa de entrada. Si la red falla en converger a una solución puede ser que necesite más nodos ocultos. Si esto no converge, probar con menos nodos ocultos y fijar un tamaño que sea la base global del rendimiento del sistema [5].

5.7. LIMITACIONES

El algoritmo del gradiente descendiente generalmente es muy lento, por que requiere pequeños ratos de aprendizaje para un aprendizaje estable. La variación del momento es usualmente rápida y más simple que el gradiente descendiente, ya que permiten mayor aprendizaje mientras se mantenga la estabilidad, pero todavía son muy lentas para muchas aplicaciones prácticas [3].

Las redes multicapa son capaces de ajustarse al rendimiento acerca de cualquier cálculo lineal o no lineal, y pueden aproximar cualquier función arbitrariamente razonablemente bien. Sin embargo, mientras la red es entrenada es teóricamente capaz de tener un rendimiento correcto, el algoritmo de retropropagación y sus variaciones no siempre hallan una solución [3].

Taller 5

MANEJO DE REDES DE RETROPROGACIÓN

1. Creación de una red backpropagation

La red perceptrón multicapa de tipo backpropagation se crea con la función *newff*

net = newff (PR,[S1 S2 ...SN],{TF1 TF2 ...TFN},BTF)

Donde los parámetros son:

Net: Nueva red de alimentación hacia adelante

PR: Matriz ($R \times 2$) donde aparecen los máximos y mínimos de los R elementos de entrada.

Si: Número de neuronas la capa i ésima,

TFi: Función de transferencia de la capa i ésima. Por defecto = *tansig* (otras posibilidades: *logsig* ó *purelin*).

BTF: Algoritmo de retropropagación de error. Por defecto = *traingdx* (otras posibilidades: *traingd*, *traingdm*, *traingda*).

El subíndice $1 \leq i \leq N$ indica el número de capas

2. Simulación

La simulación del funcionamiento de la red neuronal se ejecuta con *sim*

a = sim (net, P)

Donde los argumentos son:

net: Red perceptrón creada

P: Patrones de prueba para la simulación

a: Resultado de la simulación

3. Inicialización

Cabe hacer notar que los pesos de red son inicializadas automáticamente, pero se puede establecer valores propios, con la función *init*.

net = init(net)

Existen tres métodos de inicialización

initlay: inicialización personalizada, “default”

initwb: Inicializa randomicamente “rands”

initnw: Inicialización en base a la técnica de “Nguyen y Widrow”

Ej: net.layers{1}.initFcn = 'initwb';

4. Aprendizaje

Es el proceso en el cual los pesos y el bias son ajustados de manera tal que minimice la función de rendimiento (error).

Existen dos maneras deferentes manera de aprendizaje:

- a) Entrenamiento incremental
- b) Entrenamiento por lotes

a) Entrenamiento incremental

Los pesos y el bias son ajustados después de cada patrón presentado a la red

La función empleada es **adapt** (permite que una red adapte sus pesos).

[net,a,e] = adapt (net,P,T)

Los parámetros utilizados son:

net: La red neuronal
a: Salidas obtenida por la red
e: Error
P: Patrones de entrada
T: Valores de salida deseados

Para el ajuste de los pesos y el bias se emplea el algoritmo gradiente descendente. Existen dos funciones de aprendizaje:

learngd: Algoritmo de aprendizaje sin momento

Parámetro **lr** = tasa de aprendizaje

learngh: Algoritmo de aprendizaje con momento

Parametro *lr* = tasa de aprendizaje
 mc = momento

Ej: net.adaptParam.lr=0.05;

Para la función *adapt* la única forma de terminar el aprendizaje es estableciendo el número de pasos o ciclos.

b) Entrenamiento por lotes

Los pesos se ajustan después de haber presentado todos los patrones, por lo que el error promedio por lotes considera la suma de los errores promedios secuenciales. El entrenamiento por lotes emplea el algoritmo gradiente descendente para ajustar los pesos.

Existen muchas variaciones de este algoritmo, una iteración de este algoritmo se escribe así:

$$x_{k+1} = x_k - \alpha_k g_k$$

Donde x_k es un vector de pesos y bias actuales, g_k es el gradiente actual, α_k es la tasa de aprendizaje.

La función que se emplea para un entrenamiento por lotes es *train*.

[net,TR, a, e]= train(net,P,T)

Los parámetros utilizados son:

net: Red neuronal multicapa
P: RxN matriz de entradas.
T: SxN matriz de salidas objetivos

y devuelve los siguientes parámetros:

net: Red neuronal entrenada
TR: Valores de entrenamiento: número de iteraciones realizadas,
 rendimiento en cada época.
a: Salidas de la red
e: Error de cada neurona de salida

Los algoritmos de aprendizaje se dividen en dos categorías

Técnicas heurísticas

Técnicas de optimización numérica

Técnicas heurísticas

1. Gradiente descendente (*traingd*)
2. Gradiente descendente con momento (*traingdm*)
3. Gradiente descendente con tasa de aprendizaje variable (*traingda*, *traingdx*)
4. Gradiente descendente elástico (*trainrp*)

Técnicas de optimización numérica

1. Gradiente conjugada

- a) Fletcher-Reeves Update (*traincgf*)
- b) Polak-Ribière Update (*traincgp*)
- c) Powell-Beale Restarts (*traincgb*)
- d) Scaled Conjugate Gradient (*traincsg*)

2. Quasi – Newton

- a) BFGS Algorithm (*trainbfg*)
- b) One Step Secant Algorithm (*trainoss*)

3. Levenberg – Marquardt

- a) Levenberg-Marquardt (*trainlm*)

Todos los anteriores algoritmos de aprendizaje están definidos como parámetro **BTF** en *newff*.

net=newff ([],[],{ },BTF)

5. Parámetros

Antes de entrenar a la red se deben establecer ciertos parámetros indicados en *net.trainParam*. Existen diferentes parámetros de acuerdo al algoritmo de aprendizaje, por ejemplo los parámetros que se utilizan para *traingd* son:

net.trainParam.lr = 0.05;

Tasa de aprendizaje

net.trainParam.epochs = 6000;

Número de ciclos

net.trainParam.goal = 0.0001;

Precisión máxima deseada

net.trainParam.min_grad = 0;
net.trainParam.show = 100;

Rendimiento del gradiente
Indica cada cuantos bucles de
entrenamiento se presentarán
resultados en pantalla.

6. Generalización

MatLab implementa dos métodos para mejorar la generalización:

- a) Regularización
- b) Parada temprana

a) Regularización

Modifica la función de rendimiento, es decir el error cuadrático medio (*mse*).

$$mse = \frac{1}{N} \sum_{j=1}^N (e_j)^2 = \frac{1}{N} \sum_{j=1}^N (t_j - a)^2$$

Aumenta un término al *mse*.

$$msereg = \gamma mse + (1 - \gamma) msw$$

Donde γ es la tasa de rendimiento $[0, 1]$, y

$$msw = \frac{1}{n} \sum_{j=1}^n w_j^2$$

msw es la suma de al cuadrado de los pesos y el bias de la red. Esta función de rendimiento obliga que los pesos y el bias sean más pequeños y que la respuesta de la red sea suave, uniforme y menos probable a un sobreajuste.

En MatLab esta función de rendimiento es asignada:

net.performFcn = 'msereg';

El problema de la regularización es determinar el valor óptimo de la tasa de rendimiento. MatLab regulariza este parámetro de manera automática, cuya función es “*trainbr*” y esta implementada como un algoritmo de aprendizaje.

b) Parada temprana

El entrenamiento se detiene cuando el error en el conjunto de validación aumenta. Los datos son divididos en tres subconjuntos

- Datos de entrenamiento
- Datos de validación
- Datos de prueba

La red se entrena con los datos de entrenamiento y su capacidad de generalización se evalúa monitoreando el error en el conjunto de validación.

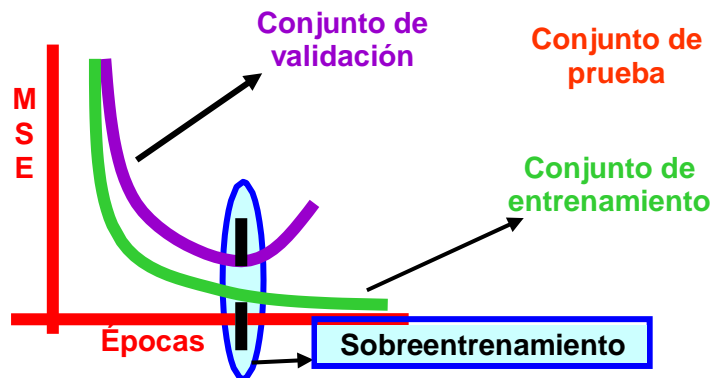


Figura T5.1. Monitoreo de parada temprana
Fuente: Modificado de [3]

El criterio de parada temprana puede ser usado con cualquiera de las funciones de aprendizaje que fueron descritas. La función que emplea este criterio es:

```
[net,tr]=train(net,p,t,[],[],v);
```

Donde:

- p:** Es el conjunto de entrenamiento
t: Es la salida deseada
v: Es el conjunto de validación (definir **v.P**, **v.T**)

Alternativamente también se puede trabajar con el conjunto de prueba, esto sería:

```
[net,tr]=train(net,p,t,[],[],v, pr);
```

Donde:

pr: Es el conjunto de prueba (definir **pr.P**, **pr.T**)

7. Preprocesamiento y postprocesamiento

Preprocesamiento.- La red neuronal puede ser más eficiente si se realiza un preprocesamiento a los datos de entrada y sus salidas deseadas. MatLab implementa tres rutinas de preprocesamiento:

- a) Normalizar entre un mínimo y un máximo
- b) Normalizar con una media y una desviación standart
- c) Análisis de componentes principales

a) Normalizar entre un mínimo y un máximo.- Emplea tres funciones: *premnmx*, *postmnmx*, *tramnmx*.

La función *premnmx* se utiliza para escalar las entradas y las salidas deseadas en un rango [-1, 1].

```
[pn,minp,maxp,tn,mint,maxt] = premnmx(p,t);  
net=train(net,pn,tn);
```

Las salidas obtenidas después del entrenamiento estarán en el rango [-1, 1].

La función *postmnmx* se usa para convertir las salidas obtenidas a sus valores verdaderos.

```
an = sim(net,pn);  
a = postmnmx(an,mint,maxt);
```

La función *tramnmx* se usa para normalizar los datos nuevos y así poder ser empleadas en la red.

```
pnewn = tramnmx(pnew,minp,maxp);  
anewn = sim(net,pnewn)
```

b) Normalizar con una media y una desviación standart.- Emplea tres funciones: *prestd*, *poststd*, *trastd*.

La función ***prestd*** normaliza las entradas y las salidas deseadas con una media igual a cero y una desviación igual a uno.

[pn,meanp,stdp,tn,meant,stdt] = prestd(p,t);

Las salidas obtenidas luego del proceso tendrán una media igual a cero y una desviación iguala uno.

La función ***poststd*** se usa para convertir las salidas obtenidas a sus valores verdaderos.

**an = sim(net,pn);
a = poststd(an,meant,stdt);**

La función ***trastd*** se usa para normalizar los datos nuevos y así poder ser empleadas en la red

**pnewn = trastd(pnew,meanp,stdp);
anewn = sim(net,pnewn);**

c) Análisis de componentes principales.- Este método reduce la dimensión de los datos de entrada, proporciona nuevos datos ortogonales, elimina datos redundantes. Emplea dos funciones: ***prepca***, ***trapca***.

La función ***prepca*** se emplea para el análisis de componentes principales. Antes de emplear estas funciones normaliza los datos de entrada con una media igual a cero y una desviación igual a uno.

**[pn,meanp,stdp] = prestd(p);
[ptrans,transMat] = prepca(pn,0.02);**

El segundo argumento de ***prepca*** quiere decir que eliminará los componentes principales que no contribuyan con al menos 2 % a la diferencia total del conjunto de datos. La función ***trapca*** se usa para los datos nuevos y así poder ser empleadas en la red

**pnewn = trastd(pnew,meanp,stdp);
pnewtrans = trapca(pnewn,transMat);
a = sim(net,pnewtrans);**

Postprocesamiento.- Es útil investigar la respuesta de la red en mas detalle. Una opción es el análisis de regresión entre la respuesta de la red (salidas obtenidas) y las salidas deseadas. La función empleada es *postreg* (postproceso de la respuesta de la red entrenada con regresión lineal)

$$[M,B,R] = \text{postreg}(A,T)$$

Donde:

A: Salidas obtenidas

T: Salidas deseadas

M: Pendiente





B: Intersección entre las salidas deseadas y obtenidas

R: Coeficiente de correlación

8. Ejemplo del problema de la función XOR

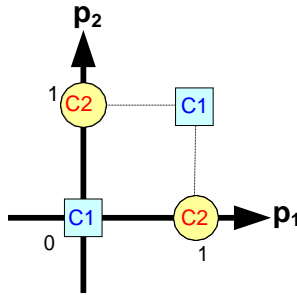
1. Descripción del Sistema

Asociado al problema de la función XOR se tiene la tabla de verdad, donde
 $1=V, 0=F$

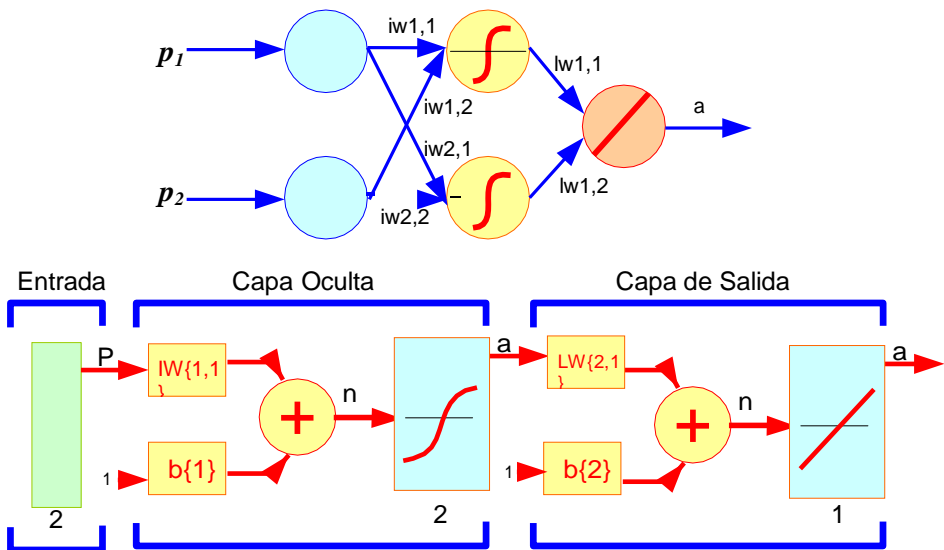
<u>p₁</u>	<u>p₂</u>	<u>XOR=t</u>		
0	0	0	C1	
0	1	1	C2	
1	0	1	C2	
1	1	0	C1	

En la tabla de verdad se observa dos clases C1 (cuadrado) y C2 (círculo) que corresponden a las salidas de la función XOR para 4 patrones, también se distingue un aprendizaje supervisado, ya que se tiene la tupla (P,T) que son los patrones de entrada y las salidas deseadas.

Teorema de separabilidad lineal.- El teorema propuesto por Rosenblatt es utilizado para ver la separabilidad lineal de las clases del problema.



Debido a que la función **XOR** no presenta una separabilidad lineal, ya que las clases no pueden ser separadas por una línea ó por un hiperplano. Entonces el modelo a utilizar sería un perceptrón multinivel con dos entradas (p_1 , p_2), una salida (a) y una capa oculta (H) con dos neuronas.



La función de transferencia por la capa de entrada y la capa de salida es lineal (*purelin*), por la capa de oculta es tangente sigmoidal (*tansig*).

Donde:

$$\begin{aligned}
 a1 &= \text{tansig} (IW\{1,1\} P + b\{1\}) \\
 a2 &= \text{purelin} (LW\{2,1\} a1 + b\{2\}) \\
 a2 &= \text{purelin} (LW\{2,1\} * [\text{tansig} (IW\{1,1\} P + b\{1\})] + b\{2\})
 \end{aligned}$$

2. Especificación del Diseño

Funciones

$$a2 = \begin{cases} \approx 0 & \text{si } p_1 = p_2 \\ \approx 1 & \text{e.o.c.} \end{cases}$$

a) Abstracción de datos

A continuación se definen los datos usados en la red neuronal:

Atributo	Valor	Tipo	Tamaño
p_1	{0,1}	$N Z^+$	9 N(1)
p_2	{0,1}	$N Z^+$	9 N(1)
$IW\{1,1\}$	[-1,1]	R^+	9.99 N(1,2)
$LW\{2,1\}$	[-1,1]	R^+	9.99 N(1,2)
t	[0,1]	$N Z^+$	9 N(1)
$a1$	[-1,1]	R^+	9.99 N(1,2)
$a2$	[0,1]	R^+	9.99 N(1,2)
$b\{1\}$	[-1,1]	R^+	9.99 N(1,2)
$b\{2\}$	[-1,1]	R^+	9.99 N(1,2)

b) Abstracción procedimental

Para la abstracción procedimental se utiliza el pseudocódigo.

Algoritmo_Backpropagation_XOR ()

P1: Introducción (leer) de p_1 , p_2 y la función lógica XOR(t , salida deseada)

P2: Categorizar de acuerdo a la función lógica

P3: Verificar la Separabilidad Lineal

P4: Si es separable, entonces imprimir “Es perceptrón simple”:

FIN

P5: Definir la topología de la red perceptrón multicapa

P6: Definir los pesos

P7: Definir la función de activación por capa

P8: Mostrar la arquitectura y función de activación

P9: Definir los parámetros para el entrenamiento

P10: Entrenar la red

P11: Validar el funcionamiento de la red

FIN_Algoritmo_Backpropagation_XOR

3. Implementación

%Definición de los patrones (entrada-salida)

P = [0 0 1 1 ;0 1 0 1];

T = [0 1 1 0];

plotpv(P, T);

%Creación de una nueva red neuronal de tipo Backpropagation

net = newff(minmax(P),[2,1],{'tansig','purelin'},'traingd');

a = sim(net,P);

linea = plotpc(net.IW{1,1},net.b{1});

pause

%Definición de los parámetros

net.trainParam.show = 50;

net.trainParam.lr = 0.05;

net.trainParam.epochs = 3000;

net.trainParam.goal = 0.001;

%Entrenamiento de la red neuronal

net=train(net,P,T);

linea = plotpc(net.IW{1,1},net.b{1});

% Validación para ver el estado de los pesos

a = sim(net,Patrones);

Ejercicios propuestos 5

1. Construir un archivo de datos que contenga tres columnas de datos (**x**; **sin(x)**; **cos(x)**) e intentar aproximarlos con una red Backpropagation. Probar varios parámetros de los valores de aprendizaje y momento, luego comparar la convergencia en los distintos casos. ¿Qué valores se recomienda para este problema?

2. Considerar la función no lineal $y(x) = 20e^{-8.5x}(\ln(0.9x + 0.2) + 1.5)$. Luego generar un fichero con 50 pares $(x, y(x))$ en el intervalo de (0,1) para entrenar un perceptron multicapa. Generar también un fichero con otros 50 puntos distintos para comprobar la validez de la aproximación.
3. El objetivo del negocio es predecir si la empresa quebrará en base a los siguientes datos financieros:
 - Flujo de Caja / Deuda Total
 - Ingreso Neto / Activo Total
 - Activo Corriente / Pasivo Corriente
 - Activo Corriente / Netas

46 empresas con la misma situación de (quiebre o no quiebre).

Desarrollar una red neuronal para este problema, utilizar el método propuesto para el desarrollo de redes neuronales. El archivo de los datos se encuentra en *datos_emp.dat*.

6

FILTROS ADAPTATIVOS LINEALES, ADALINE Y MADALINE

El ADALINE (Red Neuronal Adaptativa Lineal, del inglés *Adaptive Linear Networks*) es una red similar al perceptrón, pero la función de transferencia es lineal en lugar de la función de transferencia límite. Esto permite salidas de cualquier valor, superando la limitación del perceptrón en sus salidas (0 ó 1). Las redes ADALINE y perceptrón sólo pueden resolver problemas linealmente separables. Sin embargo, ADALINE utiliza la regla de aprendizaje LMS (Least Mean Square Error) o también conocido como regla del mínimo error cuadrático medio que es más poderosa que la regla utilizada por las redes perceptrón. El LMS o regla de Widrow-Hoff minimiza el error cuadrático medio y así mueve las decisiones límite lejos de los patrones de entrenamiento [3].

Primero se diseña una red lineal que cuando se le presente un conjunto de vectores de entrada, produzca una salida correspondiente al vector objetivo. Para cada vector de entrada se calcula un vector de salida. La diferencia entre la salida obtenida y la salida deseada es el error. Posteriormente se hallan valores de los pesos y el bias de tal manera que la suma de los errores cuadráticos es minimizada de acuerdo a un valor específico. Este problema es manejable por que los sistemas lineales tienen un único error que es el mínimo. En muchos casos se puede calcular una red lineal directamente, de tal manera que el error es mínimo al dar el vector de entradas y salidas deseadas. En otros casos los problemas numéricos prohíben el cálculo directo, afortunadamente, siempre se puede entrenar la red para tener un mínimo error utilizando la regla de aprendizaje de Widrow y Hoff.

Redes como ésta son usadas en la cancelación del error, procesamiento de señales y sistemas de control [3].

La habilidad de ADALINE para adaptar los coeficientes de sus pesos es extremadamente útil. Cuando se escribe un filtro digital en un programa de computadora, el programador debe conocer exactamente como especificar el algoritmo de filtrado y buscar detalles de las características de la señal. Si se desean modificaciones, o si las características de la señal cambian, se requiere una reprogramación. Cuando el programador utiliza una ADALINE, el problema cambia a ser capaz de especificar la señal de salida deseada de acuerdo a una señal particular de entrada. El ADALINE toma una entrada y una salida deseada, se ajusta a la transformación de rendimiento deseado. Además, si las características de la señal cambian, el ADALINE se adapta automáticamente [5].

6.1. FILTROS Y PROCESAMIENTO DIGITAL DE SEÑALES

El siguiente paso en la evolución del diseño de los filtros viene con computadoras de sistemas digitales, y más recientemente con la habilidad de los microcomputadores que son chips con una arquitectura a medida-entallada para aplicaciones de procesamiento de señales [5].

Esta computadora ejecuta un programa a la aplicación de operaciones de transformación discretas R , digitalizando aproximaciones de señales continuas $x(n)$, produce un valor de salida $y(n)$ para cada ejemplo de entrada cuando n es discreto en los pasos variables del tiempo. En este aspecto el rendimiento de la transformación es un filtro digital. Además, cualquier filtro puede ser completamente caracterizado si su respuesta $h(n)$ se aproxima a la función de unidad de impulso, representado por $\delta(n)$. De mejor manera [5],

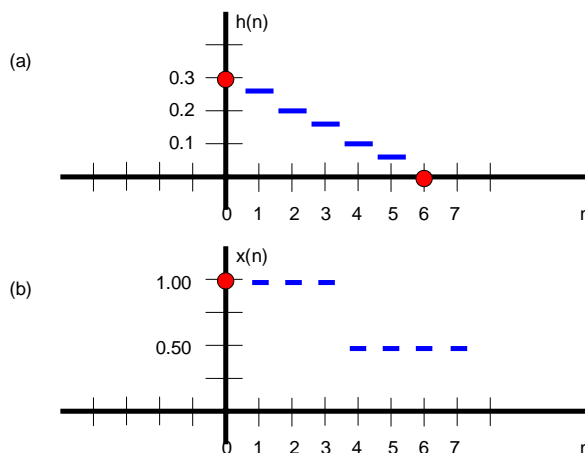
$$h(n) = R [\delta(n)] \quad (37)$$

El beneficio de esta formulación es que la respuesta del sistema a la unidad de impulso es conocida, la salida del sistema para cualquier entrada es [5]:

$$\begin{aligned} y(n) &= \mathbf{R} [x(n)] \\ &= \sum_{i=-\infty}^{\infty} \mathbf{h}(i) x(n-i) \end{aligned} \quad (38)$$

donde $x(n)$ es la entrada al sistema.

Es suficiente proponer que la sumatoria es una operación de suma de productos, similar a la de una red neuronal tipo perceptrón, cuando el rendimiento es encontrado por el cálculo de la señal de activación de entrada. Específicamente, la red ADALINE usa exactamente este cálculo de suma de productos, al menos en el ejemplo de retardo y el cambio de operadores para determinar como la simulación de una entrada es recibida por una señal de entrada instantánea. La figura 6.1 muestra a) el proceso que viene determinado por la respuesta deseada del filtro en una función de impulso de ocho pasos del tiempo discretos, b) la señal de entrada es probada y es cuantificado en ocho tiempos, c) el filtro de salida es producido por cada paso del tiempo por la multiplicación por cada término de (a) que corresponde a un valor de (b) [5].



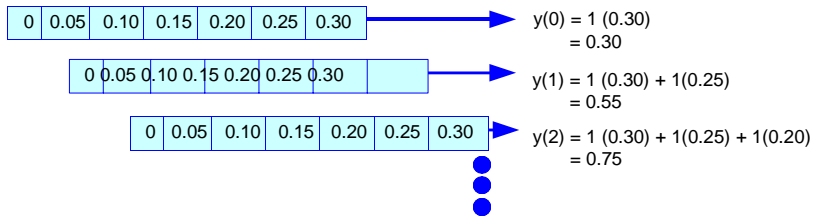


Figura 6.1. Circunvolución del cálculo de la suma
Fuente: Tomado de [5]

6.2. MODELO NEURONAL

Una neurona lineal con R entradas se muestra a continuación:

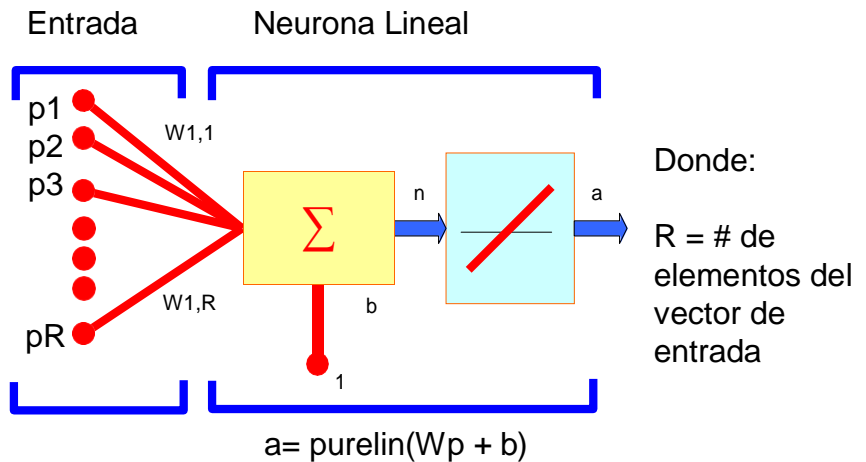


Figura 6.2. Neurona lineal
Fuente: Tomado de [3]

Esta red tiene una estructura base igual al perceptrón. La única diferencia es que usa una función de transferencia lineal que se llama *purelin* [3].

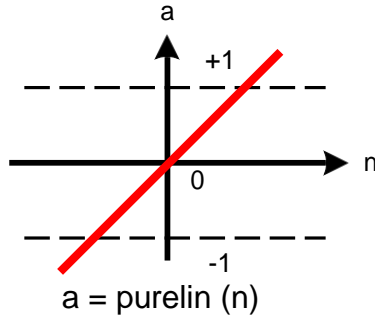


Figura 6.3. Función de transferencia lineal
Fuente: Tomado de [3]

La función de transferencia lineal calcula la salida de una neurona por una respuesta simple de los valores que pasan por la red [3].

$$a = \text{purelin}(n) = \text{purelin}(Wp + b) = (Wp + b) \quad (39)$$

Esta neurona puede ser entrenada para aprender una afinidad de funciones de entrada, o hallar una aproximación lineal a funciones no lineales. Una red lineal no realiza un cálculo no lineal [3].

6.3. ARQUITECTURA

Las redes lineales muestran una capa con S neuronas conectadas a R entradas a través de una matriz de pesos W [3].

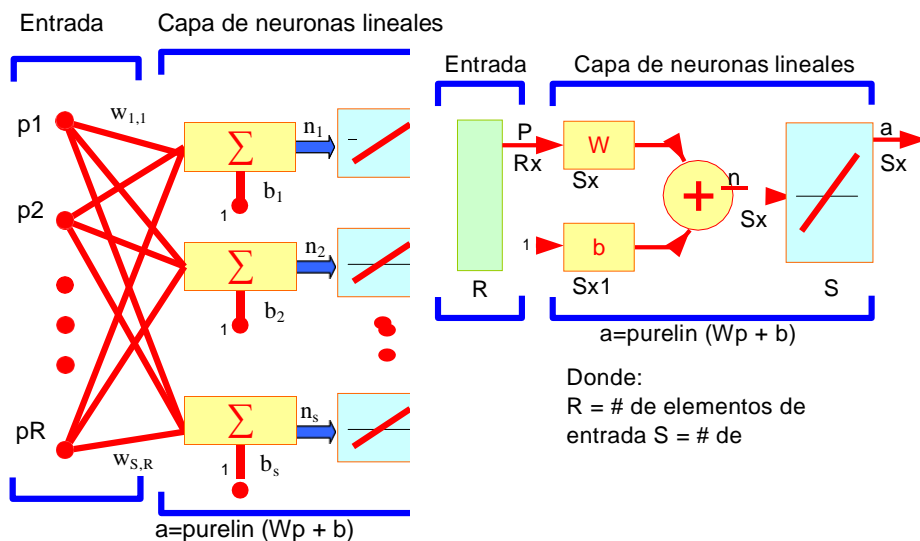


Figura 6.4. Arquitectura ADALINE

Fuente: Tomado de [3]

Esta red neuronal es llamada MADALINE debido a que esta conformado por muchos ADALINES. Notar que en la figura a la izquierda se define un vector de salida a de tamaño S . La regla de aprendizaje Widrow-Hoff sólo puede entrenar redes lineales de una sola capa. Esto no es una gran desventaja, sin embargo, una simple capa de redes lineales tiene una capacidad semejante a una red lineal multicapa. Por cada red lineal multicapa, hay una red lineal unicapa equivalente [3].

6.4. ERROR CUADRÁTICO MEDIO

Como la regla de aprendizaje del perceptrón, el algoritmo del menor error cuadrático medio (LMS) es un ejemplo de entrenamiento supervisado, en el cual la regla de aprendizaje es provista por un conjunto de ejemplos $\{p1, t1\}, \{p2, t2\}, \dots, \{pQ, tQ\}$, donde p es la entrada a la red y t es su correspondiente salida deseada. El error es calculado como la diferencia entre la salida deseada y la salida

obtenida. Se busca minimizar el promedio de la suma de los errores [3].

$$mse = \frac{1}{Q} \sum_{k=1}^Q e(k)^2 = \frac{1}{Q} \sum_{k=1}^Q (t(k) - a(k))^2 \quad (40)$$

El algoritmo LMS ajusta los pesos y el bias de la red ADALINE, para minimizar el error cuadrático medio. Afortunadamente, el error cuadrático medio indica el rendimiento de la red que es una función cuadrática. Así, el rendimiento indica que se tiene un mínimo global, un imponente mínimo ó un no – mínimo, dependiendo de las características del vector de entrada. Específicamente, las características del vector de entrada determinan si existe solo una solución [3].

6.5. ALGORITMO DE APRENDIZAJE WIDROW-HOFF

El algoritmo de aprendizaje por error cuadrado mínimo (del ingles Least Mean Square Error LMS) o regla de aprendizaje Widrow-Hoff, es basado en un proceso de aproximación de manera descendiente. Las redes lineales son entrenadas con ejemplos de comportamiento correcto. Widrow y Hoff tenían la visión de poder estimar el error cuadrático medio con la utilización del error cuadrático en cada iteración. Si se toma la derivada parcial del error cuadrático con respecto a los pesos y el bias en la iteración k se tiene [3]:

$$\frac{\partial e^2(k)}{\partial w_{1,j}} = 2e(k) \frac{\partial e(k)}{\partial w_{1,j}}$$

Para $j=1, 2, \dots, R$ y

$$\frac{\partial e^2(k)}{\partial b} = 2e(k) \frac{\partial e(k)}{\partial b} \quad (41)$$

Luego la derivada parcial con respecto del error

$$\begin{aligned} \frac{\partial e(k)}{\partial w_{1,j}} &= \frac{\partial [t(k) - a(k)]}{\partial w_{1,j}} = \frac{\partial}{\partial w_{1,j}} [t(k) - (Wp(k) + b)] \\ \frac{\partial e(k)}{\partial w_{1,j}} &= \frac{\partial}{\partial w_{1,j}} \left[t(k) - \left(\sum_{i=1}^R w_{1,i} p_i(k) + b \right) \right] \end{aligned} \quad (42)$$

Donde $p_i(k)$ es el elemento i del vector de entrada en la iteración k .
Similarmente,

$$\frac{\partial e(k)}{\partial w_{1,j}} = -p_j(k) \quad (43)$$

Esto se simplifica a:

$$\begin{aligned} \frac{\partial e(k)}{\partial w_{1,j}} &= -p_j(k) \quad y \\ \frac{\partial e(k)}{\partial b} &= -1 \end{aligned} \quad (44)$$

Por último el cambio de la matriz de pesos y el bias puede ser: $2\alpha e(k)p^T(k)$ y $2\alpha e(k)$. Estas dos ecuaciones forman la base del algoritmo de aprendizaje LMS. Estos resultados pueden ser extendidos al caso de múltiples neuronas, y escritos en forma matricial de la forma [3]:

$$W(k+1) = W(k) + 2\alpha e(k)p^T(k) \quad (45)$$

$$b(k+1) = b(k) + 2\alpha e(k) \quad (46)$$

Donde el error e y el bias b son vectores y α es la tasa de aprendizaje. Si α es grande, el aprendizaje sucede rápido, pero si es exageradamente grande puede llevar a la inestabilidad y el error puede incrementarse.

Para garantizar un aprendizaje estable, la tasa de aprendizaje debe ser menor que el valor mayor del vector de la matriz de correlación $\mathbf{p}^T \mathbf{p}$ del vector de entrada [3].

6.6. FILTRADO ADAPTATIVO

La red ADALINE, como el perceptrón, sólo puede resolver problemas linealmente separables. No obstante, el ADALINE es usado ampliamente en aplicaciones prácticas. El filtrado adaptativo es una de las mayores áreas de aplicación [3].

a) Línea de la cola de retardo. Se necesita un nuevo componente, la línea de la cola de retardo, para usar totalmente la red ADALINE. Hay señales de entrada por la izquierda, y pasos a través de $N-1$ retardos. La salida de la línea de la cola de retardo (TLD) es un vector de dimensión N , este produce la señal de entrada en el tiempo actual, la señal previa de entrada, etc. [3].

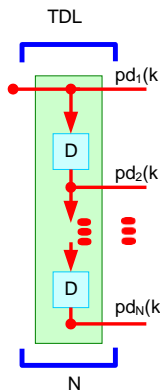


Figura 6.5. Línea de cola de retardo

Fuente: Tomada de [3]

b) Filtro adaptativo. Se combina la línea de la cola de retardo con la red ADALINE para crear un filtro adaptativo como se muestra en la figura 6.6 [3].

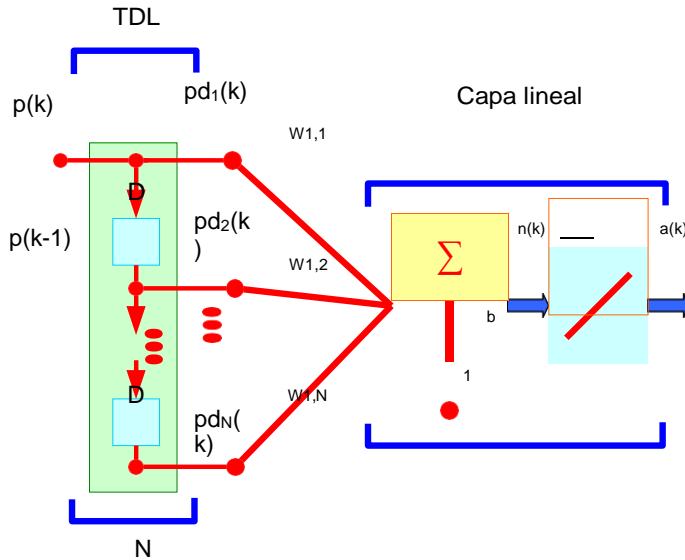


Figura 6.6. Filtro adaptativo
Fuente: Tomada de [3]

La salida del filtro esta dada por

$$a(k) = \text{purelin}(Wp + b) = \sum_{i=1}^R w_{1,i} a(k-i+1) + b \quad (47)$$

La red muestra a que se refiere el procesamiento de señales digitales en el campo de un filtro que proporciona una respuesta de impulso finita (FIR).

6.7. FILTROS ADAPTATIVOS DE MÚLTIPLES NEURONAS

Madaline es el acrónimo de muchos Adalines. Los Adalines son ordenados en una arquitectura multicapa, el Madaline puede presentarse con un vector de entrada multidimensional [5].

Se puede buscar la manera para utilizar más de una neurona en un sistema adaptativo, así que se necesita alguna notación adicional. La

línea de la cola de retardo es usada por S neuronas lineales como se muestra en la figura 6.7 [3].

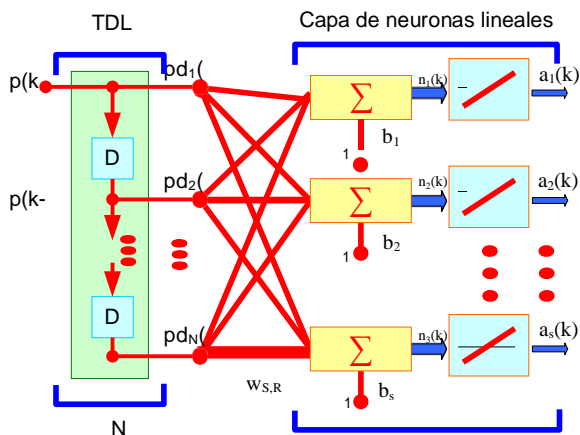


Figura 6.7. Filtro Adaptativo para múltiples neuronas
Fuente: Tomada de [3]

Alternativamente, se muestra la red de una forma abreviada

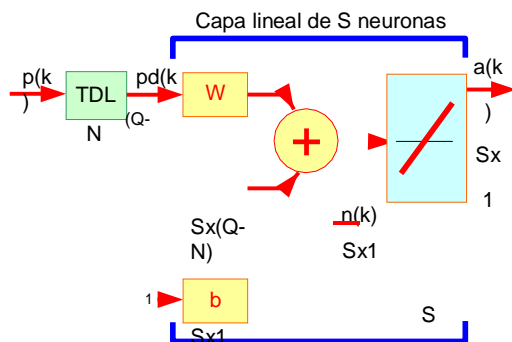


Figura 6.8. Filtro Adaptativo en forma abreviada
Fuente: Tomada de [3]

Si se busca más detalle acerca de la línea de la cola de retardo y no existen demasiados retardos se utiliza la siguiente notación

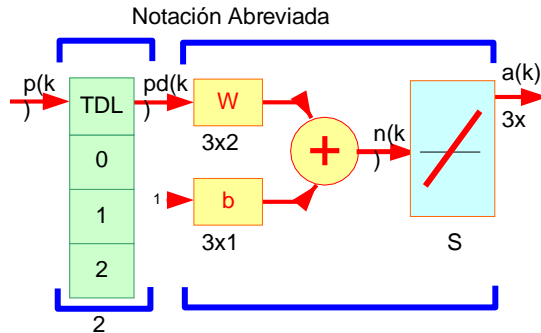


Figura 6.9. Línea de cola de retardo

Fuente: [3]

En la figura 6.9 se tiene una línea de la cola de retardo que envía la señal actual, la señal y la señal con retardo están antes de la matriz de pesos. Se puede tener una lista extensa y valores de retardo que se omiten si se lo desea. El único requerimiento que el retardo tiene es incrementar el orden que va de la cabeza al fondo de la cola [3].

6.8. ALGORITMO DE ENTRENAMIENTO REGLA MADALINE II

Es posible un método de entrenamiento para la estructura de una red Madaline basada en el algoritmo LMS; Sin embargo, el método cuenta con el reemplazo de la función umbral de salida con una función continua diferenciable. El método es conocido como Regla de Madaline II (MRII del inglés Madaline Rule II) [5].

MRII se asemeja a procesos de prueba y error con la adición de “inteligencia” en la forma del principio de mínima alteración. Ya que la salida de la red es una serie de unidades bipolar, la cantidad de entrenamiento es reducida al número de nodos de salidas incorrectas que afectan al error de salida, mientras se incurre en el menor cambio en los pesos que tienen que tener como precedente un procedimiento de aprendizaje. Este principio es visto en el siguiente algoritmo [5]:

1. *Aplicar un vector de entradas para el entrenamiento de Madaline y propagar hacia las neuronas de salida.*

2. *Contar el número de valores incorrectos por la capa de salida, llamar a este el error.*
3. *Para todas las unidades de la capa de salida,*
 - a. *Seleccionar previamente el primer nodo cuya salida es cercana a cero. (Este nodo puede cambiar la salida bipolar con el cambio de los pesos del término de mínima alteración).*
 - b. *Cambiar los pesos de la unidad seleccionada tal como la salida bipolar de la unidad cambiada.*
 - c. *Propagar el vector de entrada hacia delante de las entradas a las salidas.*
 - d. *Si el cambio del peso da una reducción del error, aceptar el cambio del peso; de otra manera, restablecer los pesos originales.*
4. *Repetir el paso 3 para todas las capas, excepto la capa de entrada*
5. *Para todas las unidades de la capa de salida,*
 - a. *Seleccionar el par de unidades no seleccionadas anteriormente, cuya salida sea cercana a cero.*
 - b. *Aplicar una corrección del peso en ambas unidades, con un orden de cambio por cada salida.*
 - c. *Propagar el vector de entrada hacia delante, de las entradas a las salidas*
 - d. *Si el resultado del error cambia por los pesos, aceptar el cambio; de otra manera, reestablecer los pesos originales.*
6. *Repetir el paso 5 por todas las capas, excepto la capa de entrada.*

6.9. LIMITACIONES

Las redes ADALINE solo aprenden relaciones entre vectores de entrada y salida. No halla soluciones para algunos problemas. Sin embargo, incluso una perfecta solución no existe, el ADALINE puede minimizar la suma de los errores cuadráticos si la tasa de aprendizaje es suficientemente pequeña. Es posible que la red halle una solución cerrada de acuerdo a la naturaleza lineal de su arquitectura. Esto

influye porque el error que aparece de una red lineal es una parábola multidimensional. Entonces las parábolas sólo tienen un mínimo [3].

Taller 6

MANEJO DE REDES ADALINE

1. Creación de un ADALINE

Una red de capa lineal se crea con la función *newlin*

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

Las redes ADALINE son usadas en filtros adaptativos para el procesamiento de señales y la predicción. En este tipo de redes los argumentos de entrada son:

PR: Matriz de $R \times 2$ de valores mínimos y máximos del vector de entrada

R

S: Número de neuronas

ID: Vector de retraso, por defecto es [0]

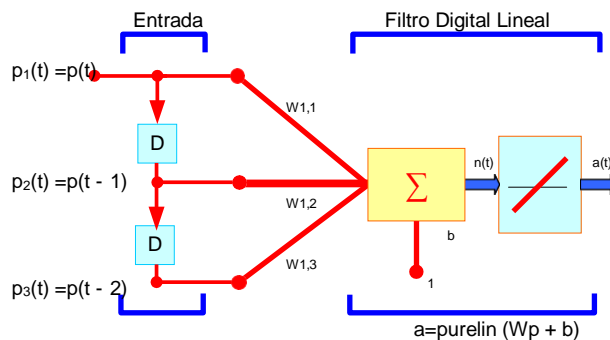
LR: Tasa de aprendizaje, por defecto es 0.01

net: Nueva red de capa lineal

2. Ejemplo de filtro adaptativo

1. Descripción del sistema

Para el resolver el problema se necesita el uso de un filtro, por lo cual se emplea una red neuronal de tipo Adaline, además se utiliza una línea de la cola de retardo. De acuerdo a la descripción del sistema, se constan de 3 entradas ($p1$, $p2$, $p3$), un bias (b) y una salida de tipo lineal (a).



La función de transferencia utilizada es lineal (*purelin*), esto porque para este tipo de redes (Adaline) tienen ya definidos ciertos parámetros, como la función de transferencia.

Donde: $a = \text{purelin}(w_{1,1}p_1 + w_{1,2}p_2 + w_{1,3}p_2 + b)$

2. Especificación del Diseño

a) Abstracción de Datos

A continuación se definen los datos usados en la red neuronal:

Atributo	Valor	Tipo	Tamaño	
p ₁	[0,10]	N Z ⁺	9	N(1)
p ₂	[0,10]	N Z ⁺	9	N(1)
p ₂	[0,10]	N Z ⁺	9	N(1)
w _{1,1}	[0,5]	R ⁺	9.99	N(12)
w _{1,2}	[0,5]	R ⁺	9.99	N(12)
w _{1,3}	[0,5]	R ⁺	9.99	N(12)
a	[0,60]	R ⁺	9.99	N(12)

b) Abstracción Procedimental

Para la abstracción procedimental se utiliza el pseudocódigo.

Algoritmo *Filtro Adaptativo()*

- P1: *Introducción (leer) de p₁, p₂, p₃*
- P2: *Verificar si el problema puede resolverse con una red ADALINE*
- P3: *Si el problema no se puede resolver, entonces imprimir “No se puede resolver con una red ADALINE”: FIN*
- P4: *Definir la topología de la red ADALINE*
- P5: *Definir los pesos*
- P6: *Definir la función de activación por capa*
- P7: *Mostrar la arquitectura y función de activación*
- P8: *Definir los parámetros para el entrenamiento*
- P9: *Entrenar la red*

FIN *Filtro Adaptativo*

3. Implementación

Asumir que los valores de entrada están en un rango de 0 a 10. Entonces se pueden definir la red con una salida.

```
net = newlin([0,10],1);
```

Se puede especificar el retardo de la línea de la cola de retardos con

```
net.inputWeights{1,1}.delays = [0 1 2];
```

Esto indica que la línea de retardo está conectada a la matriz de pesos de la red a través de los retardos de unidades de tiempo 0,1 y 2 (la especificación de los retardos deben estar en orden ascendente), también se dan los pesos y el bias se la siguiente forma:

```
net.IW{1,1} = [7 8 9];
```

```
net.b{1} = [0];
```

Finalmente se definen los valores iniciales de las salidas de retraso (valores secuenciales).

```
pi = {1 2}
```

Notar que las mismas están ordenadas de izquierda a derecha para corresponder a los retardos tomados de la cima al fondo en la figura. Esto concluye la organización de la red. Ahora las entradas son escalares en la secuencia de 3, 4, 5 y 6.

```
p = {3 4 5 6}
```

Ahora se tiene una red y su secuencia de entradas. Se simula la red para ver que salida es una función del tiempo.

```
[a,pf] = sim(net,p,pi);
```

La salida de la red es la siguiente

```
a =
```

```
[46] [70] [94] [118]
```

Y los valores finales para el retardo de salida

```
pf =
```

```
[5] [6].
```

Este ejemplo es muy sencillo para que el lector pueda verificarlo de manera práctica. La red definida debe ser entrenada con la función **adapt** para

producir una secuencia particular de salida. Suponer, para esta instancia, que la red produzca la secuencia de valores 10, 20, 30 y 40.

T = {10 20 30 40}

Se entrena la red para la salida **T**, se empieza por las condiciones del retardo inicial que se usaron. Se especifican el número pasos igual a 10 con la secuencia

net.adaptParam.passes = 10;

Entonces se entrena la red

[net,y,E pf,af] = adapt(net,p,T,pi);

El código retorna los pesos y el bias encontrados.

wts = net.IW{1,1}

wts =

0.5059 3.1053 5.7046

bias = net.b{1}

bias =

-1.5993

y =

[11.8558] [20.7735] [29.6679] [39.0036]

Probablemente si se tienen pasos adicionales la secuencia de salida estaría con valores más cercanos a los valores deseados de 10, 20, 30 y 40. De esta manera se especifican las redes adaptativas, simulándolas y finalmente entrenándolas con **adapt**.

Ejercicios propuestos 6

1. Construir una red neuronal artificial de ADALINE para la función **AND** y **OR**, comparar los resultados obtenidos con la red perceptrón.
2. Simular la cancelación de ruido en una señal. Una neurona lineal es usada para adaptarse a una señal dada, esta puede predecir una señal de segundo. **TIME** se define como los pasos del tiempo de la simulación. **P** es la señal de estos pasos del tiempo. **T** es la señal derivada de **P** por el cambio a la derecha, multiplicando por 2 y añadiéndola.

```
time = 1:0.01:2.5;  
X = sin(sin(time).*time*10);  
P = con2seq(X);  
T = con2seq(2*[0 X(1:(end-1))] + X);
```


7

REDES NEURONALES RECURRENTES

Las redes neuronales artificiales o modelos conexionistas son la base de potentes técnicas en reconocimiento de formas, especialmente en el campo del reconocimiento de imágenes y del habla, ya que permiten construir modelos que representan a una función no lineal arbitraria; por otra parte, son capaces de modelar una tarea determinada adaptando los parámetros internos de la red mediante un proceso de aprendizaje. Sin embargo, la propiedad de representación es muy diferente a la capacidad de aprender a partir de ejemplos, ya que, como en cualquier técnica de aproximación de funciones, la red podría estancarse en un mínimo local, o si la topología no es adecuada para la tarea que se desea abordar, se genera un modelo inapropiado [11].

Las redes neuronales generalmente son divididas en dos categorías muy amplias: redes neuronales unidireccionales o de alimentación hacia adelante (feedforward en inglés), que se caracterizan por trabajar sin ningún tipo de ciclos y las redes recurrentes, que trabajan con uno o más ciclos. La presencia de ciclos en una red le permite realizar un análisis natural de sistemas dinámicos, ya que el estado de la red en un momento del tiempo depende del estado previo, en algunos casos, es más natural ver a los ciclos como una especificación que provee múltiples restricciones que los nodos de la red deben satisfacer, estas restricciones se interpretan como especificaciones de estado de equilibrio del sistema dinámico [12].

Las redes neuronales recurrentes fueron ampliamente estudiadas y desarrolladas en la década de los 90, se las diseñó para aprender secuencias o patrones que varían en el tiempo. Una red recurrente posee conexiones de retroalimentación (ciclo cerrado) que le permiten aprender y reconocer patrones a partir de estados anteriores, entre las redes recurrentes más sobresalientes se tiene: Memoria Asociativa Bidireccional (Bidirectional Associative Memory, BAM), Hopfield, Boltzmann Machine, y Redes Backpropagation Recurrentes (Jordan, Elman, etc.) [13].

En el contexto de las redes recurrentes existen redes dinámicas por naturaleza como lo son la red de Hopfield, la red de Jordan, la red de Elman y redes dinámicas que siendo de naturaleza estática como lo son las redes multicapa logran el comportamiento dinámico realimentando sus entradas con muestras anteriores de las salidas, el comportamiento dinámico de las redes recurrentes hace que sean una poderosa herramienta para simular e identificar sistemas dinámicos no lineales [14].

Las redes neuronales parcialmente recurrentes fueron presentadas al finalizar el año de 1980 por muchos investigadores tales como, Rumelhart, Hinton y Williams para aprender cadenas de caracteres. Este tipo de redes neuronales han sido empleadas en la solución de un gran número de problemas, como por ejemplo, para el seguimiento del movimiento de la cabeza en sistemas de realidad virtual, pronóstico de datos financieros, demanda de energía eléctrica, control de calidad del agua, reconocimiento del habla y generación de notas musicales [13,14].

Según Kremer (1995), una red neuronal recurrente frecuentemente es usada para identificar o generar salidas temporales de un sistema no lineal, este tipo de red es capaz de aproximarse a una máquina de estado finito y gracias a esta característica se simula cualquier serie de tiempo.

7.1. ARQUITECTURAS

Sobre un modelo de neurona clásico, modelo de McCulloch-Pitt, la recurrencia se establece en tres diferentes puntos:

1. En las conexiones de entrada a la neurona: Se establece una realimentación en cada una de las conexiones sinápticas.
2. Sobre la salida de activación $a(t)$: A las entradas de la neurona se añaden valores anteriores de salidas de activación de dicha neurona, provocando una realimentación (con los retardos correspondientes) de la activación de la neurona de salida a la entrada.
3. Sobre la salida de la neurona $y(t)$: es similar al caso anterior, pero ahora la realimentación se realiza después de haber aplicado la función no lineal sobre la salida de activación [15].

Las arquitecturas de estas redes varían desde aquellas que están totalmente interconectadas (figura 7.1) hasta aquellas redes parcialmente conectadas (figura 7.2), que incluyen a las redes multicapa feedforward que tienen distintas capas de entrada y salida. En las redes totalmente interconectadas no se distinguen capas de entrada de los nodos, además, cada nodo posee una entrada de todos los demás nodos y la retroalimentación hacia el mismo nodo es permitida [14].

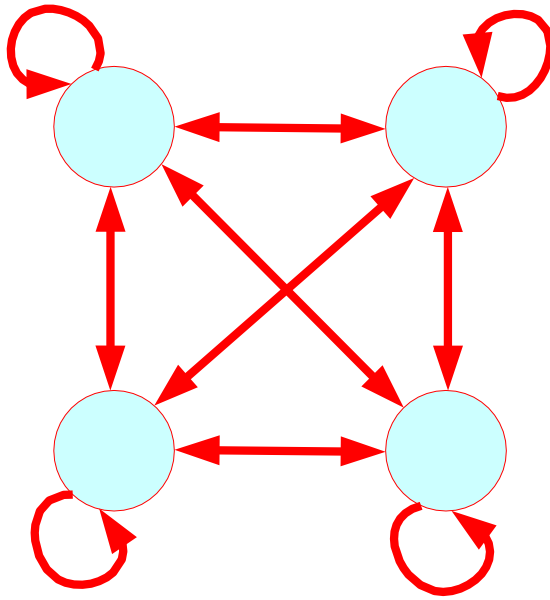


Figura 7.1. Red recurrente totalmente interconectada
Fuente: Modificado de [14], p. 2

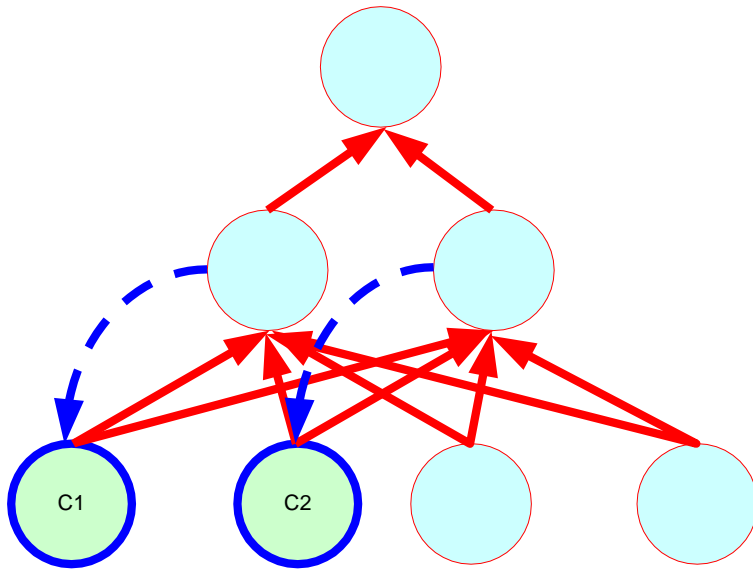


Figura 7.2. Red recurrente parcialmente interconectada
Fuente: Modificado de [14], p. 2

Las redes parcialmente interconectadas han sido empleadas para aprender cadenas de caracteres, aunque algunos nodos tienen la estructura feedforward, otros nodos proveen secuencias a las unidades de contexto que realimentan a otros nodos. Los pesos de las unidades de contexto (C_1 y C_2) son consideradas como si se tratasen de unidades de entrada [14].

Considérese a una red neuronal recurrente, la cual admite un análisis en términos de los estados de equilibrio. Estas redes, en las que se encuentra la red de Hopfield y las Máquinas de Boltzmann, son generalmente representadas mediante grafos no dirigidos, es decir, grafos en los que la presencia de una conexión de un nodo S_i al nodo S_j implica una conexión nodo S_j al nodo S_i , tal y como se observa en la figura 7.3, en esta figura se presume que el valor del peso W_{ij} es igual al peso W_{ji} . Este grafo posiblemente este parcialmente o totalmente conectado [12].

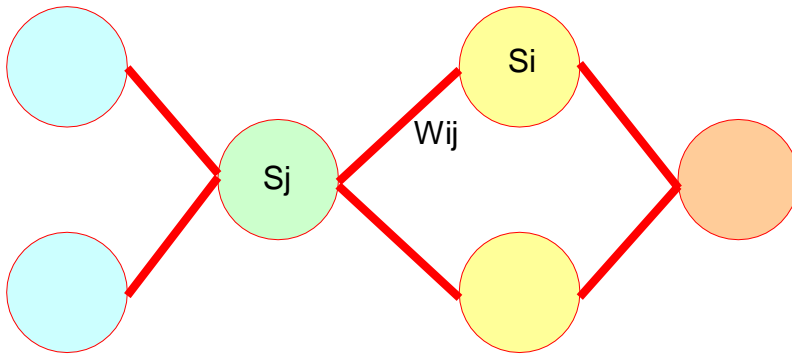


Figura 7.3. Red recurrente genérica sin dirección
Fuente: Modificado de [12], p. 705

También existen redes neuronales recurrentes (por ejemplo la red de Elman) cuya representación se la realiza mediante grafos dirigidos, ver figura 7.4. En estos grafos es permitida la interconexión arbitraria entre nodos, es decir, existen nodos cuya conexión no es reciproca. Se asocia un valor real al peso W_{ij} que se enlaza desde el nodo j al nodo i , tolerando el valor de cero en peso W_{ij} si no existe conexión. En el instante t , el i -ésimo nodo de la red tiene un valor de activación $S_i[t]$, cuyo valor es discreto ó continuo, generalmente son valores discretos en los cuales t es un índice discreto [12].

La presencia de ciclos en el grafo caracterizan a estas redes y las diferencia de redes feedforward. Nótese que no se requieren de conexiones reciprocas como en la red de Hopfield o las Máquinas de Boltzmann [12].

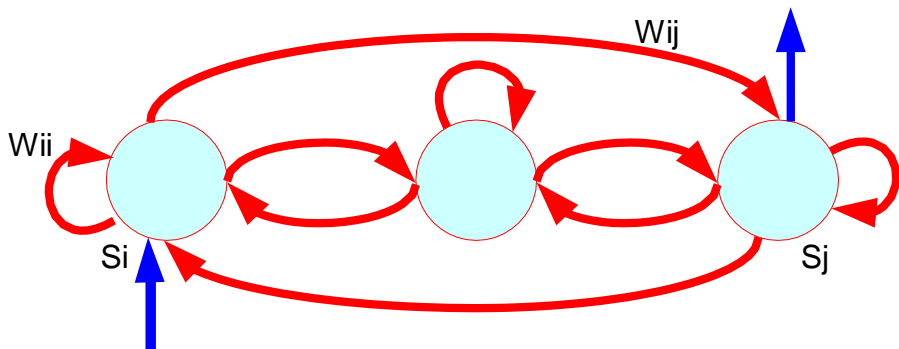


Figura 7.4. Red recurrente genérica
 Fuente: Modificado de [12], p. 706

7.2. APRENDIZAJE

El aprendizaje es un aspecto fundamental de las redes neuronales y debido a esta característica es que son empleadas en diferentes aplicaciones. Las redes neuronales recurrentes modelan patrones de comportamiento complejo (incluye ciclos limitados y patrones caóticos), y resulta algo complicado el colocar condiciones sobre los pesos W_{ij} que garanticen una clase de comportamiento en particular, es por esto que, los investigadores interesados en el comportamiento variable a través del tiempo de este tipo de redes neuronales utilizan algoritmos de entrenamiento como un método de programación de la red al proveerle ejemplos del comportamiento deseado [12].

Los algoritmos de aprendizaje han sido tema de discusión de muchos investigadores (por ejemplo, Nilsson (1965) y Mendel (1970)). El aprendizaje Hebbiano y el aprendizaje por el descenso del gradiente son los pilares del aprendizaje de las distintas técnicas desarrolladas hasta ahora. Una versión muy popular que surgió a partir del aprendizaje por el descenso del gradiente es la retro-propagación del error presentada por Rumelhart (1986) y Werbos (1993), el algoritmo de retro-propagación es relativamente sencillo de implementar, y muchos problemas pueden ser resueltos a través de su uso, pero debido a la complejidad de los sistemas dinámicos procesados por las redes neuronales recurrentes surgió la necesidad de algoritmos complejos que representen el aprendizaje [14].

Los científicos desarrollaron una gran variedad de esquemas basados en el aprendizaje de descenso del gradiente, en particular basados en el aprendizaje por retro-propagación. El año de 1990, Werbos presento el algoritmo de Backpropagation Through Time (retro-propagación a través del tiempo), que permite a las redes neuronales recurrentes evolucionar en el tiempo como una secuencia de redes estáticas que usan la retro-propagación del error [14].

Las redes neuronales recurrentes poseen las siguientes ventajas:

1. Habilidad inherente para simular autómatas de estado finito, su comportamiento es el de un sistema dinámico.
2. Representación en forma de estados de información sobre la historia de la secuencia.
3. Toda máquina de Turing es simulada por una red neuronal recurrente.
4. Computacionalmente más poderosas que las redes con alimentación hacia adelante.
5. Por lo menos equivalentes a una máquina de Turing [12].

El principal problema de la aplicación de las redes neuronales recurrentes radica en la selección de la arquitectura de la red neuronal, es decir, el número y tipo de neuronas, la localización de los ciclos de retroalimentación y la implementación de un algoritmo de entrenamiento adecuado. Por tanto, considerando las diferentes topologías y algoritmos de aprendizaje disponibles para las redes neuronales recurrentes, la selección apropiada del par arquitectura-aprendizaje depende directamente del problema que se vaya a resolver y esta selección determinara el éxito o fracaso del trabajo emprendido [13].

7.3. RED DE ELMAN

Esta red típicamente posee dos capas, cada una compuesta de una red tipo Backpropagation, con la adición de una conexión de realimentación desde la salida de la capa oculta hacia la entrada de la misma capa oculta, esta realimentación permite a la red aprender a reconocer y generar patrones temporales o variantes con el tiempo. La red neuronal de Elman tiene una capa oculta, la cual esta totalmente conectada a otras neuronas, denominadas unidades contextuales que permiten a esta red predecir comportamientos futuros a partir de comportamientos pasados, estas unidades de contexto son las encargadas de proporcionarle un comportamiento dinámico y de realimentar a la capa oculta [15].

Los valores de activación de la capa oculta son copiados a las unidades de contexto durante cada ciclo de entrenamiento, y son utilizadas como parte de las entradas a la capa oculta en el siguiente ciclo. De esta manera, en el siguiente ciclo se tomara información del ciclo actual, por tal motivo, las unidades de la capa oculta son capaces de producir salidas basadas no solamente en las entradas actuales, sino que también lo hace de la información precedente a estas entradas. Elman sostiene que las unidades de la capa oculta mejoran la representación de la estructura temporal de la entrada [16].

En la figura 7.5, se muestran las conexiones tipo feedforward entre todas las unidades de las distintas capas (capa origen a capa de destino), en cada ciclo de entrenamiento las activaciones de las unidades de la capa oculta son copiadas una a una a las correspondientes unidades de la capa de contexto. En el siguiente ciclo la capa de contexto se combina con las nuevas entradas y constituyen la entrada para la capa oculta de la red. De esta manera, la información del estado anterior esta disponible para la red [16].

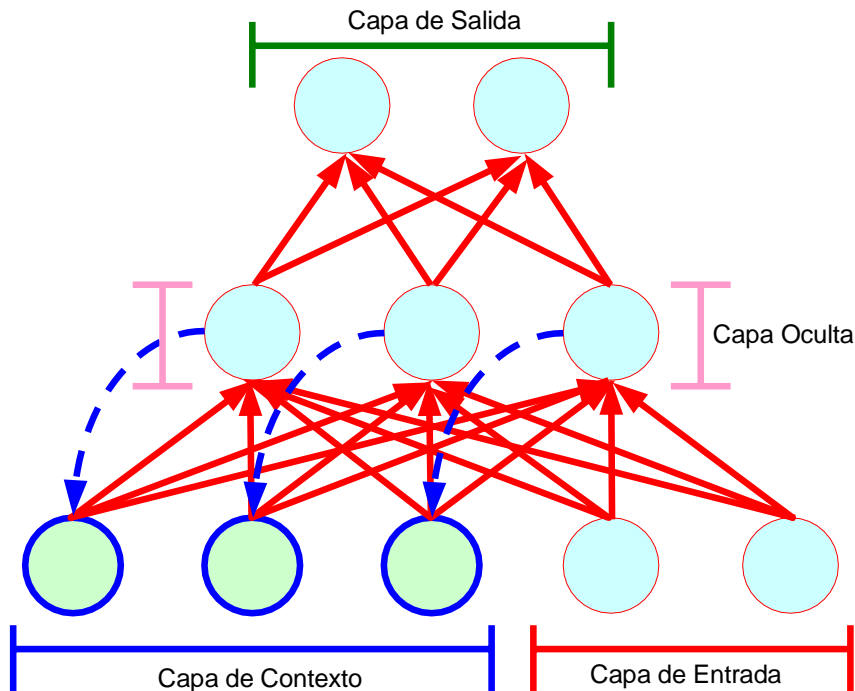


Figura 7.5. Red neuronal recurrente de Elman
Fuente: Modificado de [16], p. 18

7.3.1. Arquitectura

La red neuronal recurrente de Elman, es una red de dos capas con una retroalimentación de las unidades de contexto a las unidades de la capa oculta, esta conexión permite que la red de Elman detecte y genere patrones que varían en el tiempo [5].

En la red neuronal de Elman, las neuronas de la capa oculta tienen como función de activación a la función tangente hiperbólica (*tansig*) y las neuronas de la capa de salida poseen como función de activación a la función identidad (*purelin*). La combinación de estas dos funciones de transferencia en las capas de la red (capa oculta y capa de salida) permiten que la red se aproxime a cualquier función (función con un número finito de discontinuidades) con gran precisión, el único requerimiento es que la capa oculta tenga un número suficiente de neuronas [5].

En la figura 7.6, se muestra la arquitectura de la red neuronal recurrente de Elman (la red por motivos didácticos esta compuesta por R entradas, una neurona en la capa oculta, una unidad de contexto y una neurona en la capa de salida); las entradas están representadas por el vector fila $P_{1 \times R}$; los pesos de las conexiones están representados por el vector columna $IW_{n \times I}$; b_1 y b_2 representan el bias de las respectivas neuronas de la capa oculta y la de la capa de salida; el peso de la unidad de contexto (D) esta representado por el vector LW .

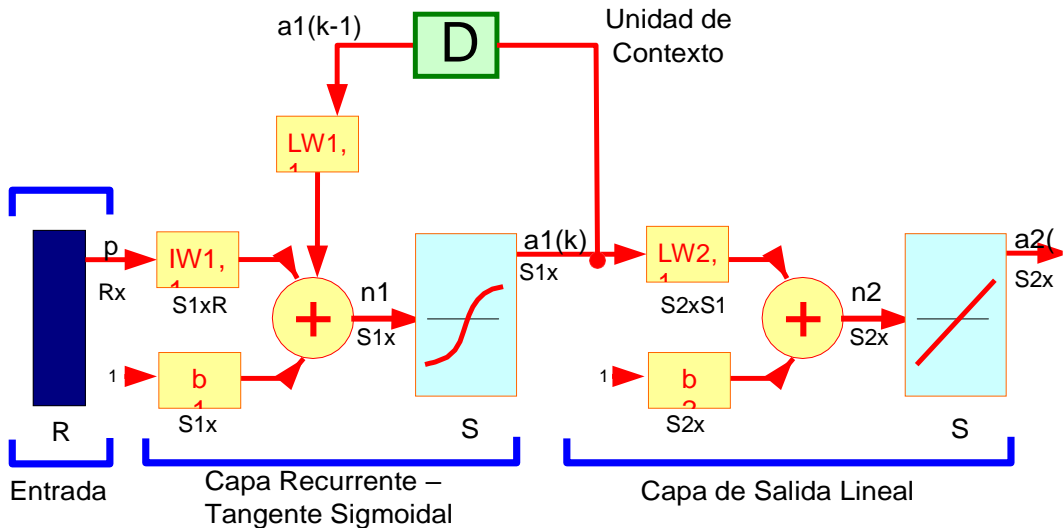


Figura 7.6. Arquitectura de la red de Elman
Fuente: Modificado de [3]

Donde:

$$a1(k)=tansig(IW\{1,1\}p + LW\{1,1\}a1(k-1) + b\{1\})$$

$$a2(k)=purelin(LW\{2,1\}a1(k) + b\{2\})$$

7.3.2. Funciones de transferencia

A continuación se desarrollan las funciones de transferencia o activación de la capa oculta y de la capa de salida de la red neuronal recurrente de Elman.

1. Función tangente hiperbólica: Esta función de transferencia en la red neuronal recurrente de Elman, calcula la salida de la capa oculta a partir de los valores de las unidades de entrada y contexto. Esta función toma los valores de la entrada (un vector) y devuelve como resultado a cada elemento del vector con valores que se encuentran entre ± 1 [5]. Estos valores de salida son calculados a partir de la formula:

$$a = \frac{2}{1 + e^{-2n}} - 1 \quad (48)$$

Que matemáticamente representa a la *tansig(n)*.

En la figura 7.7, se muestra la representación gráfica de esta función de transferencia.

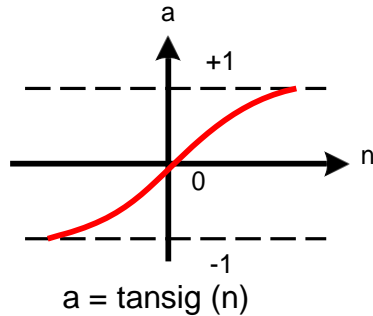


Figura 7.7. Función de transferencia tangente hiperbólica
Fuente: Basado en [5]

2. Función identidad: También conocida como función ***purelin***, es una función lineal, calcula los valores de la capa de salida en la red neuronal recurrente de Elman, toma los valores procesados por la capa oculta y devuelve estos mismos valores [5].

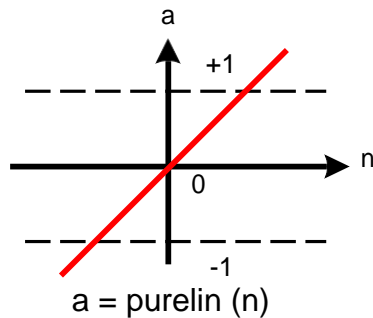


Figura 7.8. Función de transferencia lineal
Fuente: Basado en [5]

Es importante en una red neuronal artificial mencionar el tipo de entradas (estructura de datos) que maneja, los datos que van a ser procesados, generalmente son elementos de un vector, y existen básicamente dos tipos de vectores de entradas, y estos son:

1. Vectores concurrentes, son aquellos que ocurren de manera simultánea o al mismo tiempo, para estos vectores el orden no es importante, es decir, que no importa el orden en el cual aparecen, este tipo de entrada es usada por una red neuronal artificial estática (esta red no tiene ningún tipo de retroalimentación).
2. Vectores secuenciales, son vectores secuenciales dependientes del tiempo, es decir, que son secuencias de vectores que ocurren en un determinado periodo de tiempo, por tal motivo el orden de aparición de estos vectores es importante, generalmente estos vectores son usados como entradas de las redes neuronales artificiales dinámicas (son aquellas redes neuronales artificiales que poseen retroalimentación o demoras) [5].

7.3.3. Entrenamiento

La red neuronal de Elman para la etapa de entrenamiento emplea el algoritmo de retro-propagación a través del tiempo (Backpropagation Through the Time, BPTT). Este algoritmo es usado para el entrenamiento de redes neuronales recurrentes y es una extensión del algoritmo Backpropagation, se deriva de desplegar la operación temporal de la red en una red con alimentación hacia adelante, y su topología crece en una capa a cada paso de tiempo, manteniéndose los pesos sinápticos [17].

Sí $T(n)$ el conjunto de índices j de neuronas de la red neuronal para las que existe una respuesta deseada $d_j(n)$ que la salida de la j -ésima neurona debería tener en el tiempo n . Entonces, se define la señal de error así:

$$e_j(n) = \begin{cases} d_j(n) - y_j & \text{si } j \in T(n) \\ 0 & \text{en otro caso} \end{cases} \quad (49)$$

Sea el valor instantáneo del error total de la red:

$$E(n) = \frac{1}{2} \sum_j e_j^2(n) \quad (50)$$

El objetivo del aprendizaje es minimizar el error total sobre algún periodo de tiempo apropiado $[n_0, n_1]$, esto es:

$$E_{total}(n_0, n_1) = \sum_{n=n_0}^{n_1} E(n) = \frac{1}{2} \sum_{n=n_0}^{n_1} \sum_j e_j^2(n) \quad (51)$$

Para esta tarea particular el periodo de tiempo está representado por un patrón temporal, por ejemplo, el ingreso de una palabra completa, donde n_0 es el tiempo inicial en el cual se empieza a procesar una determinada palabra (primeras palabras) y n_1 el tiempo final (últimas palabras). El objetivo entonces es minimizar la función de costo $E_{total}(n_0, n_1)$, calculando el gradiente de esta función con respecto a los pesos sinápticos de la red.

Haykin en 1999, indicó que para realizar esto, se usa el algoritmo BPTT, que de manera general, realiza el procesamiento hacia adelante de los datos a través de la red para el intervalo $[n_0, n_1]$, guardando la historia de entradas, estado y respuestas deseadas de la red en ese intervalo. Posteriormente se realiza el paso hacia atrás sobre esta historia para obtener los valores de los gradientes locales

$\delta_j(n) = - \left(\frac{\partial E_{total}(n_0, n_1)}{\partial v_j(n)} \right)$ para $n_0 \leq n \leq n_1$, cuyo cálculo se deriva de la definición de la regla delta para aplicar el ajuste o corrección de pesos:

$$\Delta w_{ji} = -\eta \frac{\partial E_{total}(n_0, n_1)}{\partial w_{ji}} = \eta \sum_{n=n_0+1}^{n_1} \delta_j(n) y_i(n-1) \quad (52)$$

En este punto es necesario notar que para el problema en cuestión existe una capa de entrada, la cual no posee ciclos de retroalimentación, por lo tanto los gradientes locales para el tiempo n_0 corresponden a la primera capa oculta desplegada (esto es, la segunda capa de la red neuronal) y esta es la razón para que en este caso se tenga en cuenta el tiempo n_0 en el cálculo de los gradientes locales. Este proceso se describe detalladamente a continuación.

El cálculo de los gradientes locales está dado por la siguiente ecuación:

$$\delta_j(n) = \begin{cases} \phi_j(v_j(n)) e_j(n) & \text{si } n = n_1 \\ \phi_j(v_j(n)) \left[e_j(n) + \sum_k w_{kj} \delta_k(n+1) \right] & \text{si } n_0 \leq n \leq n_1 \end{cases} \quad (53)$$

Las ecuaciones 49), 51) y 53) ofrecen la posibilidad de la existencia de diferentes valores de respuestas deseadas para diversas neuronas en distintos tiempos. Esto es válido cuando existen ciclos de retroalimentación en las neuronas para las que se especifica una respuesta deseada. Sin embargo, dado que se hace uso en este caso de una red neuronal de Elman, donde únicamente existe recurrencia entre las neuronas de la capa oculta, es innecesario especificar la señal de error para los tiempos anteriores al tiempo actual de entrenamiento como se muestra en la ecuación 53) [17].

Además, para el caso específico de la red neuronal de Elman, la capa de salida es la única que posee respuestas deseadas y se sobreentiende que ésta no tiene ciclos de retroalimentación, por lo tanto, para el tiempo final n_1 el cálculo de los gradientes locales para la capa de salida depende de la señal de error y el cálculo de los gradientes locales para la última capa oculta desplegada depende de los gradientes locales de la capa de salida. La topología de la red se visualiza como una simple red neuronal multicapa con una capa de entrada, tantas capas ocultas desplegadas como la magnitud del intervalo $[n_0, n_1]$ y una capa de salida [17].

Sea $T(n)$ el conjunto de índices de neuronas en la capa de salida, $H(n_1)$ el de índices de neuronas de la última capa oculta desplegada y $H(n)$ el de índices de neuronas en una capa oculta desplegada en el tiempo n . Por consiguiente, las ecuaciones deducidas y usadas para los gradientes locales son las siguientes:

$$\delta_j(n) = \begin{cases} \phi_j'(v_j(n)) e_j(n) & \text{si } n = n_0 \text{ y } j \in T(n) \\ \phi_j(v_j(n)) \sum_k w_{kj}^j \delta_k(n) & \text{si } n = n_0, j \in H(n) \text{ y } k \in T(n) \\ \phi_j(v_j(n)) \sum_k w_{kj}^j \delta_k(n+1) & \text{si } n_0 \leq n \leq n_1, j \in H(n) \text{ y } k \in H(n+1) \end{cases} \quad (54)$$

Esta formulación es más cercana a la especificada en el algoritmo BPTT truncado pero existe la gran diferencia con respecto al entrenamiento en que éste se realiza de una manera continua, esto es, una vez se presenta una entrada a la red inmediatamente se hace la retro-propagación desde el tiempo actual n hasta un tiempo $n - h$ determinado, mientras que en este caso el entrenamiento es por periodos, y se le denomina algoritmo BPTT por épocas. Para este caso particular, la retro-propagación se realiza sólo hasta que se procesa totalmente la palabra, que ingresa en el tiempo n_0 y termina en el tiempo n_1 , y este paso hacia atrás se ejecuta desde el tiempo n_1 hasta el tiempo n_0 como lo describe la ecuación 7), es decir, que se define como $h = n_1 - n_0$ [17].

7.4. RED DE HOPFIELD

Jhon Hopfield en abril de 1982 publicó en Estados Unidos el libro "Neural Network and Physical Systems with Emergent Collective Computational Abilities", gracias al trabajo sobre neurofisiología en invertebrados, desarrolló un tipo de red neuronal auto-asociativa. Esta red cuenta con el número suficiente de simplificaciones como para poder extraer información sobre las características relevantes del sistema. Las redes Hopfield son auto-asociadores en los cuales los valores de los nodos son actualizados de manera iterativa basados en un principio de cómputo local; el nuevo estado de cada nodo depende únicamente de sus entradas ponderadas en un tiempo dado. Cada neurona de la capa de entrada esta conectada con una neurona de la capa media, y cada neurona de la capa media emite una sola conexión hacia la capa de salida. Estas conexiones (capa de entrada - capa media, y capa media - capa de salida) no implican calculo de pesos sinápticos ni de valores umbral [18].

7.4.1. Arquitectura

Este modelo consiste en una red monocapa con N neuronas cuyos valores de salida son binarios: $\{0,1\}$ ó $\{-1,1\}$. Las funciones de activación son del tipo escalón. Se trata, por tanto, de una red discreta con entradas y salidas binarias, posteriormente Hopfield desarrolló una versión continua con entradas y salidas analógicas utilizando neuronas con funciones de activación de tipo sigmoidal. Existen conexiones laterales (cada neurona se encuentra conectada a todas las demás) pero no autorrecurrentes (no consigo misma). Los pesos asociados a las conexiones entre pares de neuronas son simétricos ($w_{ij} = w_{ji}$) [19].

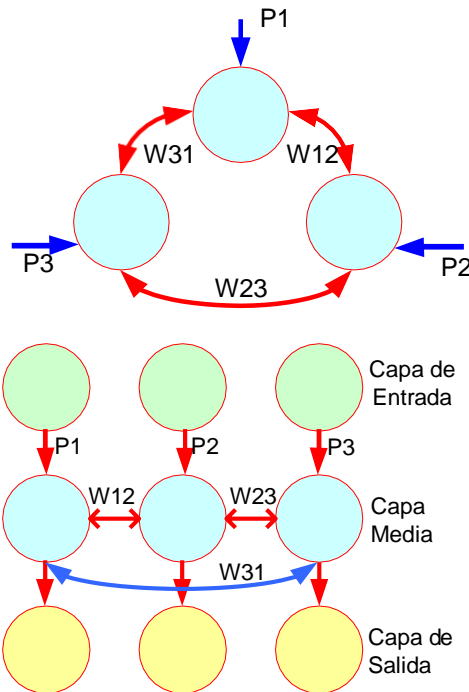


Figura 7.9. Arquitectura de la red de Hopfield
Fuente: Modificado de [5], p. 150

El hecho de que todas las neuronas de la capa media se encuentren interconectadas, hace que en esta capa se dé una retroalimentación entre sus neuronas, de forma que al activarse una neurona de esta capa hace que las otras neuronas cambien su estado de activación, que a la vez harán cambiar el suyo propio.

Esta característica de las redes de Hopfield hace que funcionen de manera diferente al Perceptrón. Así, en el Perceptrón todas las neuronas de una misma capa transmiten el patrón de activación inmediatamente hacia las neuronas de la siguiente capa, (el patrón de activación sería un vector formado por los valores de las neuronas de una capa, por ejemplo: (0, 1, 0, 1, 1)). Mientras que en una red de Hopfield las neuronas de la capa de entrada si transmiten inmediatamente su patrón de activación hacia las neuronas de la capa media (y además lo hacen sin variación debida a pesos sinápticos),

pero las neuronas de la capa media no transmitirán ningún patrón de activación hasta que hayan llegado a un estado de equilibrio, en el cual el patrón de activación de la capa media se mantiene estable [20].

7.4.2. Funcionamiento

Se trata de una red autoasociativa, por tanto, patrones diferentes pueden ser almacenados en la red, como si de una memoria se tratase, durante la etapa de aprendizaje. Posteriormente, cuando se presenta una entrada a la red, esta evoluciona hasta generar una salida que coincidirá con la que corresponde a esa entrada, o bien la más parecida si la entrada está distorsionada o incompleta. La información que recibe la red debe haber sido previamente codificada y representada en forma de vector (como una configuración binaria o como un conjunto de valores reales dependiendo de si la red es discreta o continua) con tantas componentes como neuronas (N) tenga la red. Cada neurona recibe un elemento del vector [20].

Al centrarse en una sola neurona el funcionamiento sería el siguiente:

- I. Recibe como entrada la salida de cada una de las otras neuronas (por las conexiones laterales). Estos valores de salida, inicialmente coinciden con las entradas del vector, multiplicadas por los pesos de las conexiones correspondientes. La suma de todos estos valores constituirá el valor de entrada neta de la neurona a la que se debe aplicar la función de transferencia obteniéndose el valor de salida correspondiente. En el instante inicial ($k=0$) la información de entrada es (p_1, p_2, \dots, p_N) .

$$a_i(k=0) = e_i; \quad 1 \leq i \leq N$$

(55)

$$a_i(k+1) = f \left[\sum_{j=1}^N w_{ij} a_j(k) - b_i \right]; \quad 1 \leq i \leq N$$

(56)

2. Este proceso continúa hasta que las salidas de las neuronas se estabilizan y alcanzan la convergencia durante algunas iteraciones.

$$a_j(k+1) = a_j(k) \quad (57)$$

7.4.3. Aprendizaje

El mecanismo de aprendizaje utilizado es de tipo *OFF LINE*, por lo que existe una etapa de aprendizaje y otra de funcionamiento de la red. Para la primera etapa utiliza un aprendizaje *no supervisado* de tipo hebbiano, que básicamente indica:

“Cuando una célula A está suficientemente cerca de otra B como para tomar parte en excitarla, y lo hace persistentemente, suceden cambios metabólicos en una o ambas células de tal forma que se incrementa la eficiencia de A en excitar B”

De esta manera el peso de una conexión entre una neurona i y otra j se obtiene mediante el producto de los componentes i -ésimo y j -ésimo del vector que representa la información o patrón que debe almacenar. Utilizando una notación matricial para representar los pesos de la red, se puede utilizar una matriz de dimensión $N \times N$ (recordar que N es el número de neuronas de la red y por tanto de componentes del vector de entrada). Esta matriz es simétrica ($w_{ij} = w_{ji}$) y con la diagonal con valores nulos ($w_{ii} = 0$) al no haber conexiones autorecurrentes [18,20].

También se tienen N entradas que la red debe aprender, expresadas igualmente en forma matricial: P_1, P_2, \dots, P_N ; Utilizando esta notación, el aprendizaje consistiría en la creación de la matriz de pesos W a partir de los N vectores de entrada que se enseñan a la red.

$$W = \sum_{i=1}^N (P_i^* P_i - I)$$

(58)

Donde: P_i^* es la transpuesta de la matriz P_i , y para anular los pesos de las conexiones autorecurrentes w_{ii} se usa la matriz identidad (I).

7.4.4. Función de transferencia

La versión discreta de esta red fue ideada para trabajar con valores binarios -1 y +1, por tanto, la función de activación de cada neurona de la red es de tipo escalón, cuya representación gráfica se la observa en la figura 7.10.

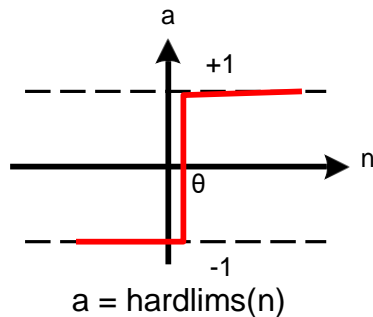


Figura 7.10. Función de transferencia escalón
Fuente: Modificado de [20]

Donde, θ_i es el umbral de disparo de la neurona i , que representa el desplazamiento de la función de transferencia a lo largo del eje de las ordenadas (n).

Para la etapa de funcionamiento, se debe simplemente multiplicar el patrón de entrada que se desea reconocer por la matriz de pesos W generada en la etapa de entrenamiento, a este resultado se le aplica la función de activación que posee cada neurona (función escalón con desplazamiento sobre el origen), la salida de la red después de la primera iteración debe nuevamente ser multiplicada por la matriz W y al resultado aplicarle nuevamente la función de activación, este proceso se lo realiza con el propósito de estabilizar la red, ya que este modelo suele presentar falsas memorias entregando resultados errados [18].

7.4.5. Limitaciones

Existen varios problemas asociados a la red Hopfield. Los dos más importantes se refieren a la cantidad limitada de datos que se pueden almacenar y la necesidad de que estos datos sean ortogonales entre sí.

1. Espacio de memoria necesario para implementar la red: Suponer que se tiene una imagen de $N \times N$ píxeles. Una red de Hopfield que pueda procesarla requerirá $N \times N$ neuronas, y debido al tipo de interconexiones, un total de $(N \times N) \times 2$ conexiones para su realización [20].
2. Número limitado de entradas en la etapa de aprendizaje: Si se almacenan demasiados patrones, durante su funcionamiento la red puede converger a valores de salida diferentes de los aprendidos, con lo que la tarea de asociación entre la información presentada y la almacenada se realiza incorrectamente. Esta situación no se produce nunca si el número de patrones almacenados es menor o igual que $N \cdot (4 \cdot \ln(N))^{-1}$, siendo N el número de neuronas de la red. Si se permite la posibilidad de un mínimo error en la recuperación de los patrones almacenados, suficientemente pequeño para poder identificar dicho patrón, el número de patrones almacenados puede ascender por debajo de un 13,8 % del número de neuronas de la red.
3. Ortogonalidad de los patrones aprendidos: Si los patrones almacenados no son suficientemente diferentes entre sí (no son ortogonales) puede ocurrir que ante una entrada la red no haga

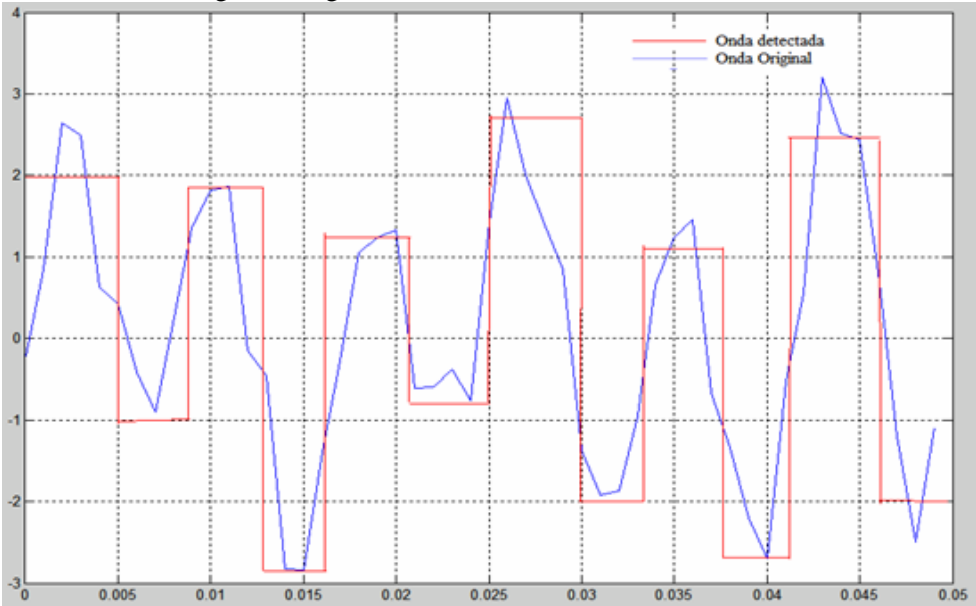
una asociación correcta y genere una salida errónea, tal vez la salida correspondiente a otra entrada aprendida que fuese muy parecida. Lo que se debe conseguir es que los patrones que se dan a la red durante la etapa de aprendizaje sean ortogonales, lo cual ocurre si se cumple que cada par de patrones de entrada difieren en, al menos $N/2$ componentes, siendo N el número total de componentes por patrón. Esta condición de ortogonalidad establece que dados dos patrones de entrada, estos deben diferir en al menos la mitad de sus componentes (distancia Hamming), o estableciendo una distancia mínima del 30% para que se garantice todavía un funcionamiento aceptable [18].

Taller 7

DETECCIÓN DE AMPLITUD

1. Descripción del sistema

Se desea calcular la amplitud de una onda que varia con el tiempo, tal y como se observa en la siguiente figura.



2. Especificación del diseño

a) Abstracción de datos

Nombre	Descripción	Tipo	Valor
Amplitud	Extensión de una onda o señal (alto).	Escalar de tipo real.	$[-2,+2]$
Seno	Función a partir de la cual se genera la onda.	Función trigonométrica continua de valor real.	$[-1,+1]$
Tiempo	Duración de una señal.	Escalar de tipo real que es generalmente medido en segundos.	$[0,\infty]$
Tansig	Función de activación empleada en la capa	Función trigonométrica continua de valor real.	$[-1,1]$

	oculta.	
Activación	Función que inhibe o excita a la neurona.	Función de activación de la red.
Identidad	Función de activación empleada en la capa de salida.	Función trigonométrica [-1,+1] de valor real.
Neurona	Unidad mínima de procesamiento.	Escalar, contiene datos discretos. [-2,+2]

La red neuronal recurrente de Elman tendrá una capa de entrada con una neurona, y otra neurona en la capa de salida, esta unidad de procesamiento se excita o inhibe mediante la función identidad (*purelin*), mientras que en la capa oculta contará con 12 unidades que se excitan o inhiben mediante la función tangente hiperbólica (*tansig*).

b) Abstracción procedimental

Algoritmo_Deteccion_Amplitud()

- P1: Leer la onda sinusoidal*
- P2: Convertir el vector concurrente en un vector de secuencias dependiente del tiempo*
- P3: Definir la topología de la red*
- P4: Presentar el vector de secuencias a la red de Elman*
- P5: Entrenar a la red*
- P6: Probar el entrenamiento*
- P7: Probar la generalización de la red*

FIN_Deteccion_Amplitud

3. Implementación

```
%
=====
=
% Mark Beale, 12-15-93
% Copyright 1992-2005 The MathWorks, Inc.
% $Revision: 1.18.2.1 $ $Date: 2005/11/15 01:14:57
$
%
=====
=
```

```

echo on % Despliega notas en pantalla.
% NEWELM - Inicializa la red recurrente de Elman.
% SIM - Simulación de la red recurrente de
Elman.
% TRAIN - Entrenamiento de la red recurrente de
Elman.

% DETECCIÓN DE AMPLITUD:
% Usando las funciones mencionadas se entrenara
una red recurrente
% con el objetivo de detectar la amplitud de una
onda.
pause % Presione una tecla para continuar...

% DEFINICIÓN DEL PROBLEMA
% La primera onda es generada con una amplitud
de 1.
p1 = sin(1:20); % Vector de 20 posiciones con
valores seno (Entrada1)
t1 = ones(1,20); % Vector de salida con elementos
iguales a 1 (Salida1)

% La segunda onda es generada con una amplitud
de 2.
p2 = sin(1:20)*2; % Vector de 20 posiciones con
valores seno (Entrada2)
t2 = ones(1,20)*2; % Vector de salida con elementos
iguales a 2 (Salida2)

% La red será entrenada por la secuencia formada
por
% la repetición de las ondas dos veces.
p = [p1 p2 p1 p2]; % Se genera una matriz p de
entradas a partir de los
% vectores generados p1 y p2.
t = [t1 t2 t1 t2]; % Se genera una matriz t de
salida a partir de los
% vectores generados t1 y t2.
Pseq = con2seq(p); % Convierte una concurrencia
en una secuencia

```

```

Tseq = con2seq(t);
pause % Presione cualquier tecla para continuar...

% DEFINICIÓN DE LA RED DE ELMAN:
% NEWELM crea una red de Elman cuyas entradas
varían de -2 hasta 2
% tiene 10 neuronas ocultas y una neurona de
salida.
% net = newelm(PR,[S1 S2...SN1],[TF1
TF2...TFN1],BTF,BLF,PF)
% Descripción de los argumentos:
% PR - Matriz de Rx2 del valor mínimo y máximo
permitido en
%      las R entradas de las neuronas de la capa
de entrada.
% Si - Tamaño de la i-ésima capa, para N1 capas.
% TFi - Función de activación de la i-ésima capa,
por defecto='tansig'.
% BTF - Función de entrenamiento, por defecto =
'traingdx'.
% BLF - Función de aprendizaje (pesos/bias), por
defecto = 'learngdm'.
% PF - Función de rendimiento, por defecto =
'mse'.
net = newelm([-2 2],[10
1],[ 'tansig','purelin'],'traingdx');
net.trainParam.epochs = 500; % Número de ciclos
para entrenar
net.trainParam.show = 10; % Muestra la evolución
del entrenamiento
net.trainParam.goal = 0.01; % Meta del aprendizaje
pause % Presione cualquier tecla para continuar...

% ENTRENAMIENTO:
% TRAIN entrena a la red. Sin parámetro se emplea
el predeterminado
% [net,tr,Y,E] = train(NET,P,T,Pi,Ai)
% Descripción de los argumentos:
% NET - Red.
% P - Entradas de la red.

```

```

% T - Objetivo de la red (salida), por defecto =
0.
% Pi - Condición inicial de la demora de la
entrada, por defecto = 0.
% Ai - Condición inicial de demora de la capa,
por defecto = 0.
% Retorna:
% NET - Red nueva.
% TR - Registro de entrenamiento (ciclos y
rendimiento).
% Y - Salidas de la red.
% E - Errores de la red.

% Inicio del entrenamiento...
tic
[net,tr] = train(net,Pseq,Tseq);
toc
% ...fin del entrenamiento.

% SEMILOGY()gráfica (ciclos,rendimiento) en base
logarítmica para Y
semilogy(tr.epoch,tr.perf)
title('ERROR CUADRADO MEDIO DE LA RED DE ELMAN')
xlabel('CICLOS')
ylabel('ERROR CUADRADO MEDIO')
pause % Presione cualquier tecla para continuar...
% PRUEBA DE LA RED DE ELMAN
a = sim(net,Pseq);

% Se gráfica la salida de la red y los objetivos
time = 1:length(p);
plot(time,t,'--',time,cat(2,a{:}))
title('PRUEBA DE LA DETECCIÓN DE AMPLITUD DE UNA
ONDA')
xlabel('Tiempo')
ylabel('Objetivo - - Salida ---')
pause % Presione cualquier tecla para continuar...

% REVISIÓN DE LA GENERALIZACIÓN
% Se crearan ondas con amplitudes de 1.6 y 1.2.

```

```

p3 = sin(1:20)*1.6;      % Onda de entrada con
amplitud de 1.6
t3 = ones(1,20)*1.6;    % Se desea que las salidas
sean de 1.6.
p4 = sin(1:20)*1.2;      % Onda de entrada con
amplitud de 1.2
t4 = ones(1,20)*1.2;    % Se desea que las salidas
sean de 1.2.

% Se repiten dos veces las series generadas para
probar la red.
pg = [p3 p4 p3 p4];
tg = [t3 t4 t3 t4];
pgseq = con2seq(pg);
pause % Presione cualquier tecla para continuar...

%      RESULTADOS DE LA GENERALIZACIÓN:
a = sim(net,pgseq);

%      Se gráfica la salida y los objetivos de la red
time = 1:length(pg);
plot(time,tg,'--',time,cat(2,a{:}))
title('Prueba de la Generalización')
xlabel('TIEMPO')
ylabel('OBJETIVO - - SALIDA ---')

% La red no reconoce aquellas amplitudes en las que
no fue entrenada
pause % Presione cualquier tecla para continuar...
echo off

```

Ejercicios propuestos 7

1. Analice el algoritmo de aprendizaje BackPropagation Through the Time y diseñe un diagrama de flujo, someta el diseño a pruebas de escritorio.
2. Escriba el pseudocódigo que transforme una matriz $A_{N \times N}$ en un vector fila $B_{1 \times (N \times N)}$ e implémtelo en un script.

3. Diseñe e implemente un algoritmo que permita verificar la ortogonalidad de dos vectores.
4. Lea acerca del tratamiento de memorias falsas (memorias espurias) en las redes neuronales de Hopfield y comente en clases las conclusiones de su lectura.

8 MAPAS AUTOORGANIZADOS DE KOHONEN

En 1982 Teuvo Kohonen presentó un modelo de red neuronal denominado mapas autoorganizados o SOM (Self-Organizing Maps), basado en ciertas evidencias descubiertas a nivel cerebral y con un gran potencial de aplicabilidad práctica. Este tipo de red se caracteriza por poseer un aprendizaje no supervisado competitivo, que se basa en el funcionamiento del cerebro humano que presenta regiones en las cuales vecindarios de neuronas responden de manera similar ante algún estímulo en una secuencia ordenada [5,21].

En el aprendizaje no supervisado (o autoorganizado) no existe ningún experto que le indique a la red neuronal si está operando correcta o incorrectamente, pues no se dispone de ninguna salida objetivo hacia la cual la red neuronal deba tender. Así, durante el proceso de aprendizaje la red autoorganizada debe descubrir por sí misma rasgos comunes, regularidades, correlaciones o categorías en los datos de entrada, e incorporarlos a su estructura interna de conexiones. Se dice, por tanto, que las neuronas deben autoorganizarse en función de los estímulos (datos) procedentes del exterior [21].

Dentro del aprendizaje no supervisado existe un grupo de modelos de red caracterizados por poseer un aprendizaje competitivo. En el aprendizaje competitivo las neuronas compiten unas con otras con el fin de llevar a cabo una tarea dada. Con este tipo de aprendizaje, se pretende que cuando a la red se le presente un patrón de entrada, sólo una de las neuronas de salida (o un grupo de vecinas) se active. Por tanto, las neuronas compiten por activarse, quedando finalmente una neurona como vencedora y anuladas el resto, que son forzadas a sus valores de respuesta mínimos [21].

Los mapas autoorganizados de Kohonen mediante un aprendizaje no supervisado, son de gran utilidad en el campo del análisis exploratorio de datos, debido a que son sistemas capaces de realizar análisis de clusters (grupos con características similares), representar densidades de probabilidad y proyectar un espacio de alta dimensión sobre otro de dimensión menor [21].

8.1. ARQUITECTURA

Un modelo SOM está compuesto por dos capas de neuronas. La capa de entrada (formada por N neuronas, una por cada variable de entrada) se encarga de recibir y transmitir a la capa de salida la información procedente del exterior. La capa de salida (formada por M neuronas) es la encargada de procesar la información y formar el mapa de rasgos. Normalmente, las neuronas de la capa de salida se organizan en forma de mapa bidimensional como se muestra en la figura 8.1.

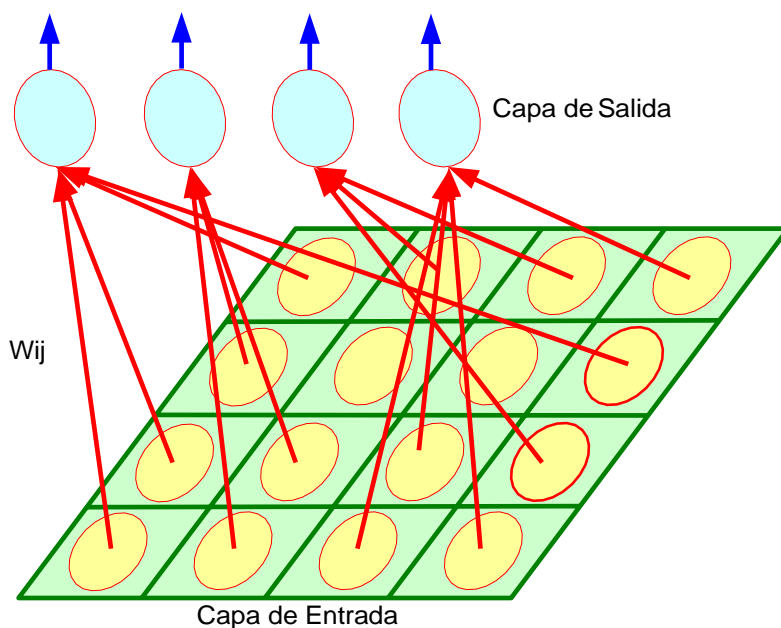


Figura 8.1. Arquitectura de la red de Kohonen

Las conexiones entre las dos capas que forman la red son hacia adelante, es decir, la información se propaga desde la capa de entrada hacia la capa de salida. Cada neurona de entrada i está conectada con cada una de las neuronas de salida j mediante un peso w_{ji} . De esta forma, las neuronas de salida tienen asociado un vector de pesos \mathbf{W} llamado vector de referencia, debido a que constituye el vector prototipo (o promedio) de la categoría representada por la neurona de salida j . Entre las neuronas de la capa de salida, existen conexiones laterales de excitación e inhibición implícitas, pues aunque no estén conectadas, cada una de estas neuronas va a tener cierta influencia sobre sus vecinas. Esto se consigue a través de un proceso competitivo entre las neuronas y la aplicación de una función denominada de vecindad [21].

8.2. FUNCIONAMIENTO

Cuando se presenta un patrón \mathbf{P} de entrada $\mathbf{P}_p: p_{p1}, \dots, p_{pi}, \dots, p_{pN}$, éste se transmite directamente desde la capa de entrada hacia la capa de salida. En esta capa, cada neurona calcula la similitud entre el vector de entrada \mathbf{P}_p y su propio vector de pesos \mathbf{W}_j o vector de referencia según una cierta medida de distancia o criterio de similitud establecido. A continuación, simulando un proceso competitivo, se declara vencedora la neurona cuyo vector de pesos sea más similar al de entrada [21].

La siguiente ecuación representa cuál de las M neuronas se activará al presentar el patrón de entrada \mathbf{P}_p :

$$a_{pj} = \begin{cases} 1 & \min_i \| \mathbf{P}_p - \mathbf{W}_i \| \\ 0 & \text{resto} \end{cases} \quad (59)$$

Donde, y_{pj} representa la salida o el grado de activación de las neuronas de salida en función del resultado de la competición (neurona vencedora = 1), $\|P_p - W_j\|$ representa una medida de similitud entre el vector o patrón de entrada $P_p: p_{p1}, \dots, p_{pi}, \dots, p_{pN}$, y el vector de pesos $W_j: w_{j1}, \dots, w_{ji}, \dots, w_{jN}$, de las conexiones entre cada una de las neuronas de entrada y la neurona de salida j .

En esta etapa se pretende encontrar el vector de referencia más parecido al vector de entrada para averiguar qué neurona es la vencedora y, sobre todo, en virtud de las interacciones excitatorias e inhibitorias que existen entre las neuronas para averiguar en qué zona del espacio bidimensional de salida se encuentra tal neurona. Por tanto, lo que hace esta red es realizar una tarea de clasificación, ya que la neurona de salida activada ante una entrada representa la clase a la que pertenece dicha información de entrada. Además, como ante una nueva entrada parecida se activa la misma neurona de salida, u otra cercana a la anterior, debido a la semejanza entre las clases, se garantiza que las neuronas topológicamente próximas sean sensibles a entradas físicamente similares [21].

8.3. APRENDIZAJE

Se debe advertir, en primer lugar, que no existe un algoritmo de aprendizaje totalmente estándar para esta red. Sin embargo, se trata de un procedimiento bastante robusto ya que el resultado final es en gran medida independiente de los detalles de su realización concreta. En consecuencia, se expondrá el algoritmo más habitual asociado a este modelo [21].

El algoritmo de aprendizaje trata de establecer mediante la presentación de un conjunto de patrones de entrenamiento, las diferentes categorías (una por neurona de salida) que servirán durante la etapa de funcionamiento para realizar clasificaciones de nuevos patrones de entrada. El proceso de aprendizaje se desarrolla de la siguiente manera, una vez presentado y procesado un vector de entrada, se establece a partir de una medida de similitud, la neurona vencedora, esto es, la neurona de salida cuyo vector de pesos es el más parecido respecto al vector de entrada.

A continuación, el vector de pesos asociado a la neurona vencedora se modifica de manera que se asemeje más al vector de entrada. De este modo, ante el mismo patrón de entrada, dicha neurona responderá en el futuro todavía con más intensidad. El proceso se repite para un conjunto de patrones de entrada los cuales son presentados repetidamente a la red, de tal forma que al final los diferentes vectores de pesos sintonizan con uno o varios patrones de entrada y, por tanto, se relacionan con dominios específicos del espacio de entrada. Si dicho espacio está dividido en grupos, cada neurona se especializará en uno de ellos, y la operación esencial de la red se podrá interpretar como un análisis de clusters [21].

Masters en 1993, da una interpretación geométrica del proceso de aprendizaje que resulta interesante para comprender la operación de la red SOM. El efecto de la regla de aprendizaje no es otro que acercar de forma iterativa el vector de pesos de la neurona de mayor actividad (ganadora) al vector de entrada. Así, en cada iteración el vector de pesos de la neurona vencedora rota hacia el de entrada, y se aproxima a él en una cantidad que depende del tamaño de la tasa de aprendizaje.

En la figura 8.2, se muestra cómo opera la regla de aprendizaje para el caso de patrones pertenecientes a un espacio de entrada de dos dimensiones, representados en la figura por los vectores de color celeste. Suponer que los vectores del espacio de entrada se agrupan en tres clusters, y que el número de neuronas de la red es también tres. Al principio del entrenamiento los vectores de pesos de las tres neuronas (representados por vectores de color rojo) son aleatorios y se distribuyen por la circunferencia. Conforme avanza el aprendizaje, éstos se van acercando progresivamente a las muestras procedentes del espacio de entrada, para quedar finalmente estabilizados como centroides de los tres clusters [21].

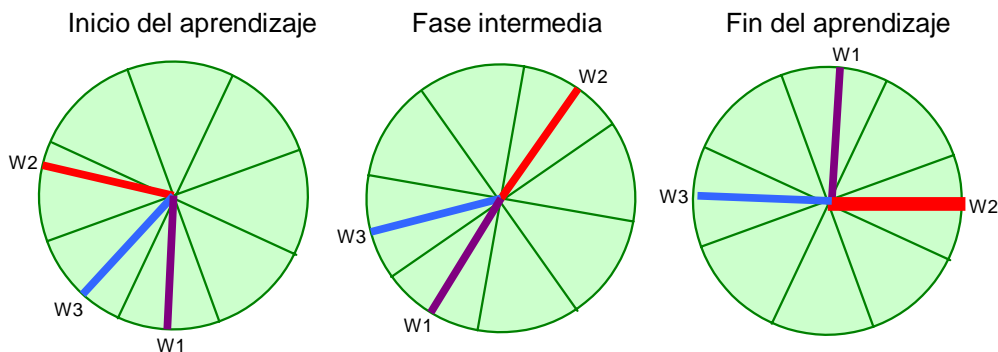


Figura 8.2. Proceso de aprendizaje en 2 dimensiones
Fuente: Modificado de [21]

Al finalizar el aprendizaje, el vector de referencia de cada neurona de salida se corresponderá con el vector de entrada que consigue activar la neurona correspondiente. En el caso de existir más patrones de entrenamiento que neuronas de salida, como en el ejemplo expuesto, más de un patrón deberá asociarse con la misma neurona, es decir, pertenecerán a la misma clase. En tal caso, los pesos que componen el vector de referencia se obtienen como un promedio (centroide) de dichos patrones.

Además de este esquema de aprendizaje competitivo, el modelo SOM aporta una importante novedad, ya que incorpora relaciones entre las neuronas próximas en el mapa. Para ello, introduce una función denominada zona de vecindad que define un entorno alrededor de la actual neurona ganadora (vecindad); su efecto es que durante el aprendizaje se actualizan tanto los pesos de la vencedora como los de las neuronas pertenecientes a su vecindad. De esta manera, en el modelo SOM se logra que neuronas próximas sintonicen con patrones similares, quedando de esta manera reflejada sobre el mapa una cierta imagen del orden topológico presente en el espacio de entrada [21].

Una vez entendida la forma general de aprendizaje del modelo, se va a expresar este proceso de forma matemática. Recordar que cuando se presenta un patrón de entrenamiento, se debe identificar la neurona de salida vencedora, es decir, la neurona cuyo vector de pesos sea el más parecido al patrón presentado. Un criterio de similitud muy utilizado es la distancia euclídea:

$$\min \|P_p - W_j\| = \min \sum_{i=1}^N (p_{pi} - w_{ji})^2 \quad (60)$$

De acuerdo con este criterio, dos vectores serán más similares cuanto menor sea su distancia.

Una medida de similitud alternativa más simple que la euclídea, es la correlación o producto escalar, según el cual, dos vectores serán más similares cuanto mayor sea su correlación.

$$\min \|P_p - W_j\| = \max \sum_{i=1}^N (p_{pi} * w_{ji}) \quad (61)$$

Identificada la neurona vencedora mediante el criterio de similitud, se pasa a modificar su vector de pesos asociado y el de sus neuronas vecinas, según la regla de aprendizaje:

$$\Delta w_{ji}(k+1) = \alpha(k)(p_{pi} - w_{ji}^*(k)), \quad j \in Zona_{j^*}(k) \quad (62)$$

Donde n hace referencia al número de ciclos o iteraciones, es decir, el número de veces que ha sido presentado y procesado todo el juego de patrones de entrenamiento; $\alpha(k)$ es la tasa de aprendizaje que con un valor inicial entre 0 y 1, decrece con el número de iteraciones (k) del proceso de aprendizaje; $Zona_{j^*}(k)$ es la zona de vecindad alrededor de la neurona vencedora j^* en la que se encuentran las neuronas cuyos pesos son actualizados. Al igual que la tasa de aprendizaje, el tamaño de esta zona normalmente se va reduciendo paulatinamente en cada iteración, con lo que el conjunto de neuronas que pueden considerarse vecinas cada vez es menor.

En el proceso general de aprendizaje suelen considerarse dos fases. En la primera fase, se pretende organizar los vectores de pesos en el mapa. Para ello, se comienza con una tasa de aprendizaje y un tamaño de vecindad grandes, para luego ir reduciendo su valor a medida que avanza el aprendizaje. En la segunda fase, se persigue el ajuste fino del mapa, de modo que los vectores de pesos se ajusten más a los vectores de entrenamiento. El proceso es similar al anterior aunque suele ser más largo, tomando la tasa de aprendizaje constante e igual a un pequeño valor (por ejemplo, 0.01) y un radio de vecindad constante e igual a 1.

Kohonen en 1990, indicó, que no existe un criterio objetivo acerca del número total de iteraciones necesarias para realizar un buen entrenamiento del modelo. Sin embargo, el número de iteraciones debería ser proporcional al número de neuronas del mapa (a más neuronas, son necesarias más iteraciones) e independiente del número de variables de entrada. Aunque 500 iteraciones por neurona es una cifra adecuada, de 50 a 100 suelen ser suficientes [21].

8.4. FASES EN LA APLICACIÓN DE LOS MAPAS AUTOROGANIZATIVOS

A continuación se pasa a describir las diferentes fases necesarias para la aplicación de los mapas autoorganizados a un problema típico de agrupamiento de patrones.

8.4.1. Inicialización de los pesos

Para diseñar una red de Kohonen, se deben asignar valores a los pesos a partir de los cuales comenzar la etapa de entrenamiento. En general, no existe discusión en este punto y los pesos se inicializan con pequeños valores aleatorios, por ejemplo, entre -1 y 1 ó entre 0 y 1, aunque también se pueden inicializar con valores nulos o a partir de una selección aleatoria de patrones de entrenamiento [21].

8.4.2. Entrenamiento de la red

Vista la manera de modificar los vectores de pesos de las neuronas a partir del conjunto de entrenamiento, se van a proporcionar una serie de consejos prácticos acerca de tres parámetros relacionados con el aprendizaje cuyos valores óptimos no pueden conocerse a priori dado un problema.

8.4.2.1. Medida de similitud

La distancia euclídea y la regla de aprendizaje presentada son métricamente compatibles, por tanto, no hay problema. Sin embargo, la correlación o producto escalar y la regla de aprendizaje presentada no son compatibles, ya que dicha regla procede de la métrica euclídea, y la correlación solamente es compatible con esta métrica si se utilizan vectores normalizados (en cuyo caso la distancia euclídea y correlación coinciden). Por tanto, si se usa la correlación como criterio de similitud, se debería utilizar vectores normalizados; mientras que si se usa la distancia euclídea, ésto no será necesario. Finalmente, independientemente del criterio de similitud utilizado, se recomienda que el rango de posibles valores de las variables de entrada sea el mismo, por ejemplo, entre -1 y 1 ó entre 0 y 1 [5].

8.4.2.2. Tasa de aprendizaje

Recordar que $\alpha(n)$ es la tasa de aprendizaje que determina la magnitud del cambio en los pesos ante la presentación de un patrón de entrada. La tasa de aprendizaje, con un valor inicial entre 0 y 1, por ejemplo, 0.8, decrece con el número de iteraciones (n), de forma que cuando se ha presentado un gran número de veces todo el juego de patrones de aprendizaje, su valor es prácticamente nulo, con lo que la modificación de los pesos es insignificante. Normalmente, la actualización de este parámetro se realiza mediante una de las siguientes funciones [21]:

$$\alpha(k) = k^{-1}$$

$$\alpha(k) = \alpha_1 \left(1 - \frac{k}{\alpha_2} \right)$$

(63)

Donde, α_1 toma un valor de 0.1 ó 0.2 y α_2 es un valor próximo al número total de iteraciones del aprendizaje. Suele tomarse un valor de $\alpha_2 = 10000$. El empleo de una u otra función no influye en exceso en el resultado final.

8.4.2.3. Zona de vecindad

La zona de vecindad ($Zona_{j^*}(k)$) es una función que define en cada iteración, k , si una neurona de salida pertenece o no a la vecindad de la vencedora j^* . La vecindad es simétrica y centrada en j^* , y puede adoptar una forma circular, cuadrada, hexagonal o cualquier otro polígono regular [21].

En general, $Zona_{j^*}(k)$ decrece a medida que avanza el aprendizaje y depende de un parámetro denominado radio de vecindad $R(k)$, que representa el tamaño de la vecindad actual. La función de vecindad más simple y utilizada es la de tipo escalón. En este caso, una neurona j pertenece a la vecindad de la ganadora j^* solamente si su distancia es inferior o igual a $R(k)$. Con este tipo de función, las vecindades adquieren una forma (cuadrada, circular, hexagonal, etc.) de bordes nítidos, en torno a la vencedora; por lo que en cada iteración únicamente se actualizan las neuronas que distan de la vencedora menos o igual a $R(k)$.

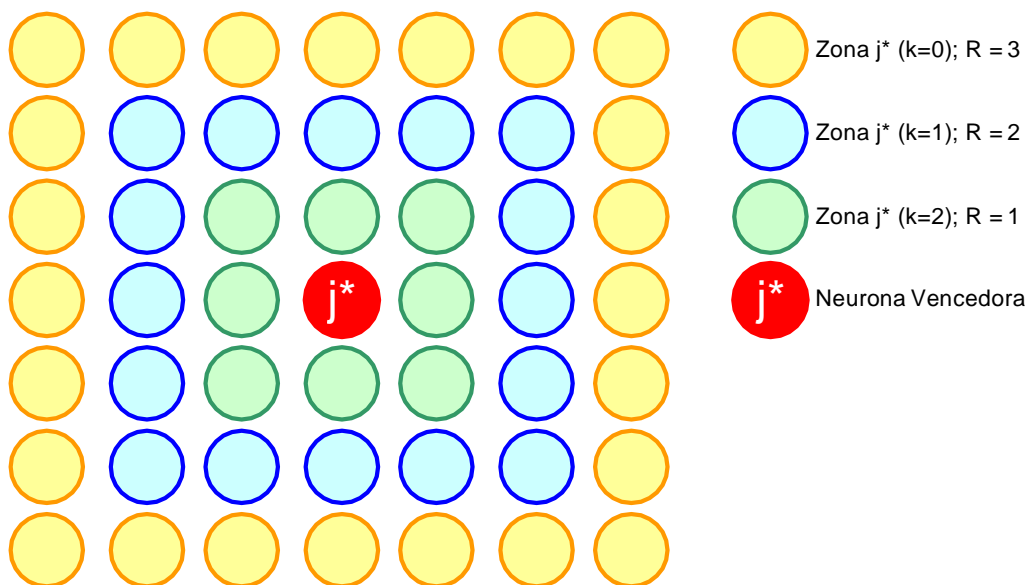


Figura 8.3. Posible evolución de la zona de vecindad

Fuente: Modificado de [21]

También se utiliza en ocasiones funciones gaussianas o en forma de sombrero mejicano, continuas y derivables en todos sus puntos, que al delimitar vecindades decrecientes en el dominio espacial establecen niveles de pertenencia en lugar de fronteras nítidas [5].

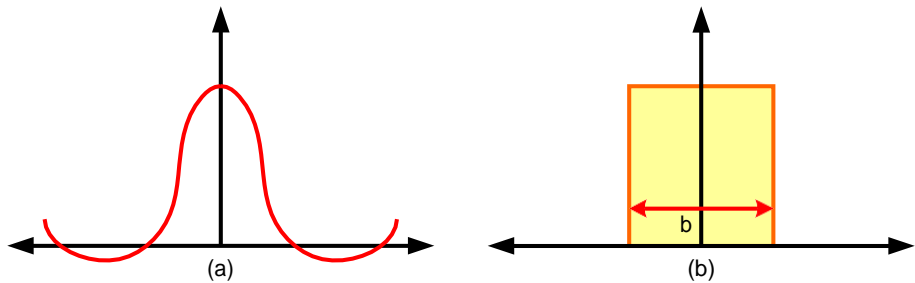


Figura 8.4. Formas de la función de vecindad

Fuente: Modificado de [5], p. 267

En la figura 8.4, se observan dos gráficos que ilustran dos diferentes modelos interconexiones laterales a través de todas las unidades de la red. La parte (a) caracteriza a las interconexiones entre determinadas neuronas, más conocida como la función del sombrero mejicano, en la cual una neurona central excitada, excita a un pequeño vecindario a su alrededor, esta señal a medida que se aleja se convierte en una señal inhibidora. La parte (b) muestra una función simple que debe ser usada como una primera aproximación a la función del sombrero mejicano. La distancia, b , define un vecindario alrededor de una neurona central que participa en el aprendizaje [5].

La zona de vecindad posee una forma definida, pero como se ha visto, su radio varía con el tiempo. Se parte de un valor inicial grande, R_0 , por ejemplo, igual al diámetro total del mapa que determina vecindades amplias, con el fin de lograr la ordenación global del mapa. $R(k)$ disminuye monótonamente con el tiempo, hasta alcanzar un valor final de $R_f=1$, por el que se actualizan los pesos de la neurona vencedora y las adyacentes. Una posible función de actualización de $R(k)$ descrita por Martín del Brío y Sanz en 1997 es la siguiente [21]:

$$R(k) = R_0 + (R_f - R_0) \frac{k}{k_R}$$

(64)

Donde k es la iteración y k_R el número de iteraciones para alcanzar R_f .

Taller 8

AGRUPAMIENTO DE DATOS

1. Descripción del sistema

Se desea realizar el agrupamiento de datos generados aleatoriamente, con el propósito de comprender y observar la manera en la que realiza este trabajo la red neuronal de Kohonen al aproximarse a las distribuciones de probabilidad subyacentes del conjunto de datos con un número reducido de unidades.

2. Especificación del diseño

a) Abstracción de datos

Nombre	Descripción	Tipo	Valor
Datos	Información dispersa con características similares.	Escalar de tipo real.	[0, 1]
Patrón	Modelo o representación de un objeto.	Vector de datos.	[0, 1]
Vecindario	Grupo de neuronas con información similar.	Clúster	
Pesos	Representación numérica del aprendizaje.	Matriz de valores reales.	[-1, 1]
Activación	Función que inhibe o excita a la neurona.	Función de activación.	[0, 1]
Neurona vencedora	Neurona que percibe de mejor manera la información de entrada.	Neurona	1
Neurona	Unidad mínima de procesamiento.	Escalar, contiene datos discretos.	[-1, +1]

En la presente red se mide la similitud entre neuronas con la función del sombrero mejicano que ayuda a definir la vecindad de la neurona vencedora.

b) Abstracción de procedimental

Algoritmo_Agrupamiento_Datos ()

P1: Leer o generar nube de datos

P2: Generar aleatoriamente los pesos iniciales

P3: Ordenar la nube de datos
P4: Obtener la neurona vencedora
P5: Obtener tasa de aprendizaje
P6: Obtener la vecindad de la neurona vencedora
P7: Obtener la convergencia de la red

FIN_Agrupamiento_Datos ()

4. Implementación

Las siguientes rutinas fueron escritas por Hugh Pasika.

```

function [W_out, p_out]=som_2d(P, nsofmx, nsofmy,
epochs, phas, W)
% Usa un vecindario Gaussiano
%      P          - patrones
%      nsofmx/y    - número de neuronas
%      epochs      - número de ciclos para el
entrenamiento
%      phas        - Fase de agrupamiento entre dos
elementos
%      W           - Valores iniciales de los pesos
(opcional)
%      W_out       - Pesos finales
%      p_out       - Valores finales de la tasa de
aprendizaje y el tamaño del vecindario.

% Valores iniciales
r=nsofmx; c=nsofmy; cr=r+1; cc=c+1;
[pats dimz]=size(P);

% Si no existen pesos iniciales los crea.
if nargin < 6, W=randn(nsofmx,nsofmy,dimz)*.1; end

% Obtiene los valores máximos y mínimos de los
patrones
mx=min(P(:,1));mxx=max(P(:,1));
my=min(P(:,2));myx=max(P(:,2));

% Inicia el agrupamiento
n = 1:epochs;
if length(phas) == 2,

```

```

so = phas(2);
t1 = epochs/(log(so+1));
sigmas = so*exp(-n/t1);
lrs = phas(1)*exp(-n/epochs);
else
    sigmas = (phas(3)-
phas(4))*fliplr(n)/epochs+phas(4);
    lrs = (phas(1)-
phas(2))*fliplr(n)/epochs+phas(2);
end

mask=zeros(2*r+1,2*c+1);
x=[-c:c]; y=[r:-1:-r];
for i=1:length(x), for j=1:length(y),
    mask(j,i)=sqrt(x(i)^2+y(j)^2);
end
end

```

```

sigma=sigmas(1)
m = (1/(2*pi*sigma^2)*exp((-
mask.^2/(2*sigma^2))));
m = m/max(max(m));
lbr =cr-floor(r/2);
ubr =cr+floor(r/2);
lbc =cc-floor(r/2);
ubc =cc+floor(r/2);

```

```

sigma=sigmas(length(sigmas));
m = (1/(2*pi*sigma^2)*exp((-
mask.^2/(2*sigma^2))));
m = m/max(max(m));

```

```

ax=(max(abs([max(P) min(P)]))); % Limites externos
del gráfico
ax=([min(P(:,1)) max(P(:,1)) min(P(:,2))
max(P(:,2)) ]);

```

```

% Aprendizaje
for epoch=1:epochs,
    t=clock;
    [neword]=shuffle(P);

```

```

sigma=sigmas(epoch);
for pat=1:pats,in=newword(pat,:)' ;
    Dup=in(:,ones(r,1),ones(c,1));
    Dup=permute(Dup,[2,3,1]);
    Dif= W - Dup;
    Sse=sum(Dif.^2,3);
    [val1 win_rows]=min(Sse);
    [val2 wc]=min(val1);
    wr=win_rows(wc);
    WN(pat,1:2)=[wr wc];
    M1 = mask(cr-wr+1:cr+(r-wr),cc-wc+1:cc+(c-
wc));
    M1 = (1/(2*pi*sigma^2)*exp((-
M1.^2/(2*sigma^2))));
    M1 = M1/max(max(M1)); %
Gausanización
    M1=M1(:, :, ones(1,dimz));
    W = W + lrs(epoch)*(Dup-W).*M1; % Nuevo peso
end

secs=etime(clock,t);
fprintf(1,'En el ciclo %i de %i próxima
actualización en aproximadamente %g segundos a
partir de ahora.\n\n',epoch,epochs,secs)
t=clock;
p_out = [lrs(length(lrs)) sigma];
W_out = W;
som_pl_map(W,1,2); drawnow
end

function som_map(A,p1,p2, P1, P2)
% function som_map(A,p1,p2)
%     Dibuja la grilla
%     A - Pesos de la RED
%     p1, p2 - Planos para el graficado

% Generación de valores iniciales de la grilla
if nargin == 3, P1=A(:, :,p1); P2=A(:, :,p2); end

% Dibuja el plano
[r c]=size(P1);

```



```

cla
hold on

%plot horizontal
for j=1:c-1,
    plot([P1(:,j),P1(:,j+1)], [P2(:,j),P2(:,j+1)], '-b')
end

%plot vertical
for i=1:c-1,

plot([P1(i,:),P1(i+1,:)], [P2(i,:),P2(i+1,:)], '-b')
end
hold off

```

```

function [o1,o2,o3,o4,o5] = shuffle(n1,n2,n3,n4,n5)
%   mezcla filas de la matriz manteniendo el orden
[r c]=size(n1);
[y,idx] = sort(rand(1,r));

for i=1:nargin;
    eval(['o' num2str(i) '=n' num2str(i)
'(idx,:);']);
end

```

SOM

```

clf % Limpia figuras
P=rand(500,2); % Genera una distribución aleatoria
subplot(2,2,1)
plot(P(:,1),P(:,2),'.') % Gráfica la distribución
title('Distribución de Entrada')
drawnow

subplot(2,2,2)
W=(rand(10,10,2)*.2)-.1; % Genera los pesos
aleatoriamente
som_map(W,1,2) % Gráfica los pesos
title('Pesos Iniciales')
set(gca, 'Box', 'On')

```

```
drawnow
```

```
% Gráfica paso a paso la evolución del agrupamiento  
subplot(2,2,3)  
[W1 p1]=som_2d(P,10,10,10,[.01,8],W);  
som_map(W1,1,2)  
title('Fase de ordenamiento')  
set(gca,'Box','On')  
drawnow
```

```
% Gráfica del mapa organizado  
subplot(2,2,4)  
W=som_2d(P,10,10,200,[p1(1) .001 p1(2) 0],W1);  
som_map(W,1,2)  
title('Fase de convergencia')  
set(gca,'Box','On')  
drawnow
```

Ejercicios propuestos 8

1. Analice el algoritmo de aprendizaje de la red neuronal de Kohonen, propuesto en esta sección y diseñe un diagrama de flujo del mismo, someta el diseño a pruebas de escritorio.
2. Lea acerca de otros métodos y técnicas empleadas en la etapa de aprendizaje de la red neuronal de Kohonen.

9 RECONOCIMIENTO DE PATRONES A TRAVÉS DE LA RED NEURONAL DE HOPFIELD

EJEMPLO PRÁCTICO

Las redes neuronales artificiales, por su constitución y fundamentos, presentan un gran número de características semejantes a las del cerebro humano. Son capaces de aprender de la experiencia, de generalizar de casos anteriores a nuevos casos, o de abstraer características esenciales a partir de entradas que representan información aparentemente irrelevante. Las redes neuronales artificiales ofrecen, por ello, numerosas ventajas y justifican la aplicación de este tipo de tecnología en múltiples áreas. Como lo señalan Maren y colaboradores (1990), las características de las redes neuronales incluyen:

1. **Aprendizaje adaptivo:** Es la capacidad de aprender a realizar tareas basadas en un entrenamiento o en una experiencia inicial.
2. **Auto-organización:** Una red neuronal es capaz de crear su propia organización, o representación de la información que recibe en la etapa de aprendizaje.
3. **Tolerancia a fallas:** Aunque la destrucción parcial de una red conduce a una degradación de su estructura, algunas capacidades de la red se pueden retener, incluso ante deterioros estructurales.
4. **Operación en tiempo real:** Los sistemas basados en arquitectura neuronal pueden ser implementados en paralelo. Con esta modalidad operativa, se diseñan y fabrican equipos con hardware específico para procesamiento rápido en paralelo.

5. Disponibilidad tecnológica: Actualmente existen circuitos integrados con topologías de redes neuronales, lo que mejora la capacidad operativa de las redes y facilita su incorporación modular a sistemas más complejos.

Los diferentes tipos de redes neuronales obedecen a la necesidad de contar con estructuras de procesamiento adecuadas para una aplicación particular.

Las redes de Hopfield por ejemplo, pueden usarse como un modelo sencillo para explicar la manera en la que ocurren las asociaciones entre ideas o recuerdos en el cerebro. Así, una idea parcial sería un estado de activación que formaría parte del área de atracción de una idea más general, que actuaría como un punto de equilibrio del área de atracción, de manera que al introducir la idea parcial en la red, se pueda alcanzar la idea general (el equilibrio).

Por estos motivos, esta red resulta especialmente útil para el área de reconocimiento de patrones (letras, números, mapas geográficos, etc.). A continuación se desarrollara un ejemplo de reconocimiento de patrones empleando una red de Hopfield.

9.1. DESCRIPCIÓN DEL SISTEMA

Una empresa dedicada a la actividad de envío de correspondencia tiene una máquina que clasifica paquetes por medio de etiquetas que indican el medio de transporte a usar para despachar los paquetes, vía aérea, terrestre o acuática. El problema consiste en que dichas etiquetas se mancharon debido al derrame de aceite que provoco la distorsión de las mismas y por tanto, la mala clasificación de los distintos paquetes.

Por lo expuesto, se le pide a usted como encargado del área de Informática diseñar e implementar una red neuronal artificial que sea capaz de realizar la clasificación de los paquetes de manera correcta a pesar de las distorsiones que se presentan en las etiquetas.

9.1.1. Materiales

- 1) Etiquetas: Existen etiquetas de dos tamaños, la primera tiene un tamaño de 35x35 pixeles y la segunda es de 50x50 pixeles, ambas etiquetas son de color negro y blanco. A continuación se muestran las etiquetas de 50x50 pixeles.

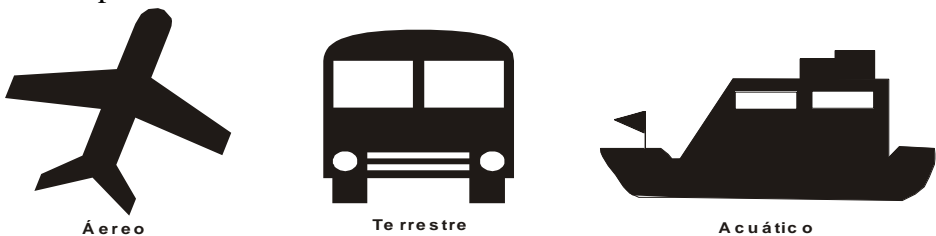


Figura 9.1. Medios de Transporte

- 2) Máquina clasificadora: Máquina con cámara fotográfica y un brazo robot capaz girar 360 grados y alzar paquetes hasta de 300 Kg., que es operada desde una computadora.

9.2 ESPECIFICACIÓN DEL DISEÑO

a) Abstracción de datos

Tabla 9.1. Estructura de datos

Nombre	Descripción	Tipo	Valor
Número de patrones	Cantidad máxima de objetos a reconocer.	Escalar de tipo entero.	$[1, \infty)$
Patrón	Modelo o representación de un objeto.	Matriz cuadrada de 35x35 o 50x50.	$[0, 255]$
Codifica	Matriz que tiene los valores enteros de la imagen	Matriz cuadrada de 35x35 o 50x50.	$[-1, 1]$

	codificada según el área de interés.		
Pesos	Matriz en la que se almacena el resultado de la etapa de aprendizaje.	Matriz cuadrada de tamaño 35x35 o 50x50, con valores enteros.	$(-\infty, \infty)$
Activación	Función que inhibe o excita a la neurona.	Función de activación, Escalón.	$[-1, 1]$
Identidad	Matriz con la diagonal principal igual a cero y los elementos de las triangulares iguales a uno.	Matriz cuadrada	$[0, 1]$
Transpuesta	Función matricial que convierte las columnas en filas y viceversa.	Función matricial	$(-\infty, \infty)$
Neurona	Unidad mínima de procesamiento.	Escalar, contiene datos discretos.	$[-1, +1]$

La red neuronal de Hopfield tendrá $N \times N$ neuronas, donde N representa el número de filas o columnas que tiene el patrón de entrada. Cada una de estas neuronas esta conectada a todas las demás pero no consigo misma, y por tanto tendrá $2 \times (N \times N)$ conexiones, la función de activación usada por cada una de las neuronas es de tipo escalón que solo genera valores de 1 ó -1.

b) Abstracción procedimental

Agoritmo_Reconocimiento_Patrones

P1: Leer patrón

P2: Codificar al patrón con los valores de +1 o -1.

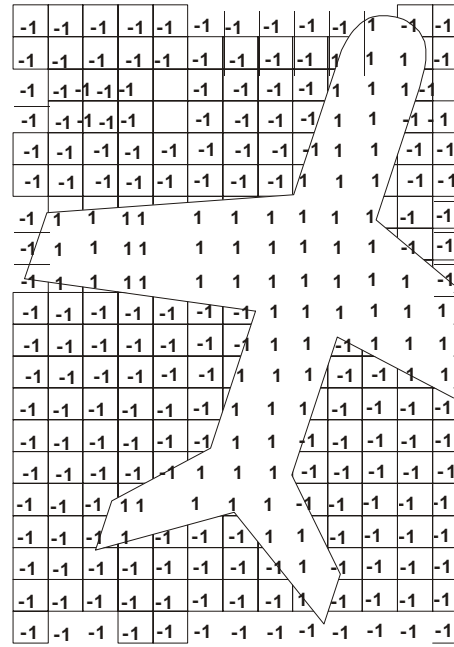
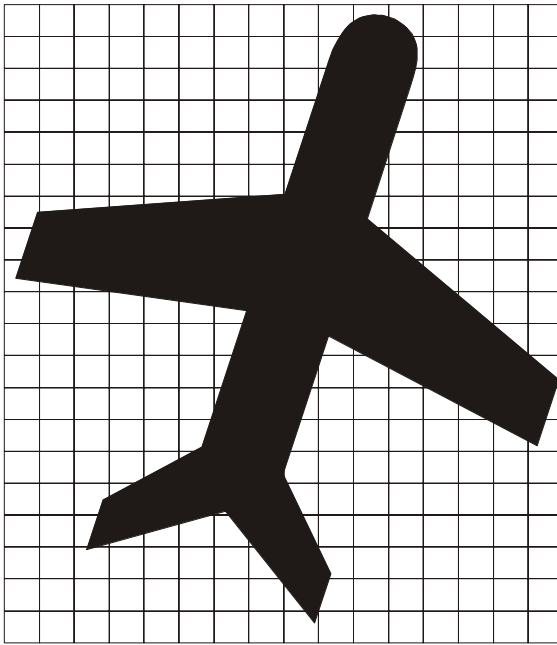


Figura 9.2. Patrón codificado y presentado en forma de matriz cuadrada.

P3: Convertir la matriz resultante de la codificación en un vector fila.

P4: Obtener la transpuesta del vector generado en el paso anterior.

P5: Multiplicar el vector columna (vector del paso P4) por el vector fila (vector del paso P3 y luego restarle la matriz identidad.

P6: Almacenar la matriz resultante del paso P5).

P7: Regresar a P1 si se tienen más patrones que presentarle a la red, de lo contrario ir a P8

P8: Sumar todas las matrices almacenadas en el paso 6) y almacenar el resultado, esta sumatoria representa la matriz de pesos W.

P9: Probar la red, repetir los pasos P1, P2 y P3; multiplicar el vector fila obtenido por la matriz de pesos W.

P10: Al resultado aplicarle la función escalón (hardlims).

9.3. IMPLEMENTACIÓN

A continuación se presenta la codificación de las rutinas más importantes descritas en el algoritmo.

```
% LeerPatron.
% =====
% Rances Méndez Quintanilla, 16-07-2006
% Copyright 2006 A.I.S.I.
clear all; %Limpia variables de memoria
close all; %Cierra ventanas gráficas

% Muestra menu
disp('  SELECCIONE EL TAMAÑO DEL PATRON  ');
disp('-----');
disp('    1. IMAGENES DE 35 x 35 PÍXELES');
disp('    2. IMAGENES DE 50 x 50 PÍXELES');
opcion = input('Presione la tecla 1 ó 2: '); %Espera por
la selección

disp('    SELECCIONE EL TIPO DE PATRON  ');
disp('-----');
disp('    A. IMAGENES CON FONDO NEGRO');
disp('    B. IMAGENES CON FONDO BLANCO');
opcion1 = input('Presione la tecla A ó B: ','s');
SW=0;
opcion1=upper(opcion1);

% Abre las imagenes (Patrones)
if (opcion==1) & (opcion1=='A')
    V1 = imread('Patron\35x35\A\avion.bmp');
    V3 = imread('Patron\35x35\A\bus.bmp');
    V4 = imread('Patron\35x35\A\barco.bmp');
    SW=1;
    Nota='Imagen de 35x35 con fondo negro';
    Dir='Patron\35x35\A\ruido\';
end

if (opcion==1) & (opcion1=='B')
```



```

V1 = imread('Patron\35x35\B\avion.bmp');
V3 = imread('Patron\35x35\B\bus.bmp');
V4 = imread('Patron\35x35\B\barco.bmp');
SW=1;
Nota='Imagen de 35x35 con fondo blanco';
Dir='Patron\35x35\B\ruido\';
end

if (opcion==2) & (opcion1=='A')
    V1 = imread('Patron\50x50\A\avion.bmp');
    V3 = imread('Patron\50x50\A\bus.bmp');
    V4 = imread('Patron\50x50\A\barco.bmp');
    SW=1;
    Nota='Imagen de 50x50 con fondo negro';
    Dir='Patron\50x50\A\ruido\';
end

if (opcion==2) & (opcion1=='B')
    V1 = imread('Patron\50x50\B\avion.bmp');
    V3 = imread('Patron\50x50\B\bus.bmp');
    V4 = imread('Patron\50x50\B\barco.bmp');
    SW=1;
    Nota='Imagen de 50x50 con fondo blanco';
    Dir='Patron\50x50\B\ruido\';
end

disp(Nota);
if SW==1
    % Grafica los patrones de entrada
    subplot(1,3,1), imshow(V1); title('AVION');
    subplot(1,3,2), imshow(V3); title('BUS');
    subplot(1,3,3), imshow(V4); title('BARCO');

    % Guarda las variables que contienen a las imagenes
    save datos0 V1 V3 V4 Nota Dir;

else
    disp('Seleccion invalida, Presione un tecla') % saca
mensaje
    pause(); %Detiene la ejecución hasta que se presione
una tecla
    return
end

```

```

% Codifica.
% =====
% Rances Méndez Quintanilla, 16-07-2006
% Copyright 2006 A.I.S.I.
% Codifica las matrices (-1, 1)
for col=1:Info(2);
    for fila=1:Info(1);
        if V1(fila,col)==0;
            A(fila,col)=1;
        else
            A(fila,col)=-1;
        end

        if V3(fila,col)==0;
            I(fila,col)=1;
        else
            I(fila,col)=-1;
        end

        if V4(fila,col)==0;
            O(fila,col)=1;
        else
            O(fila,col)=-1;
        end
    end
end

% Transforma la Matriz X de tamaño NxM en un vector
% fila de NxM columnas
cont=1;
for fila=1:Info(1)
    for col=1:Info(2)
        A1(1,cont)=A(fila,col);
        I1(1,cont)=I(fila,col);
        O1(1,cont)=O(fila,col);
        cont=cont+1;
    end
end

% Entrena.
% =====
% Rances Méndez Quintanilla, 16-07-2006
% Copyright 2006 A.I.S.I.
% Entrenamiento

```

```
% Wj = (PatronJ' * PatronJ) - I
% W = W1 + W2 +...

% Ajuste de pesos individuales
Wa = (A1'*A1) - eye(Info(1)*Info(2));
.
.
.
% Matriz de Pesos de la red
W = Wa +...
%Estos scripts fueron realizados en MatLab® 7.2.
```

Ejercicios propuestos 9

1. Entrene a la red con más patrones y pruebe la red, comente los resultados.
2. Haga rotar las imágenes distorsionadas y preséntelas a la red, comente los resultados.

BIBLIOGRAFÍA

- [1] Choque Aspiazu G. Inteligencia Artificial: Perspectivas y Realizaciones. La Paz, Bolivia: UMSA 2002.
- [2] Hilera J, Martínez V. Redes Neuronales Artificiales: Fundamentos, modelos y aplicaciones. Madrid, España: Ra-Ma 1995.
- [3] Barron A. Neural net approximation. Adaptive and Learning Systems. Yale University 1992.
- [4] Demuth H, Beale M. User Guide of Neural Network Toolbox: Computation, Visualization and Programming. NY, USA: The MathWorks Inc. 2004.
- [5] Freeman JA, Skapura DM. Neural networks: Algorithms, Applications and Programming Techniques. Texas, United States of America: Addison-Wesley Publishing Company, Inc 1991.
- [6] Eliasmith C, Anderson C. Neural Engineering: Computation, Representation and Dynamics in Neurobiological Systems. Cambridge, Massachusetts, London, England: Bradford Book, The MIT Press 2003.
- [7] Sjoberg A, Melin B, Isaksson P. The MatLab Handbook. New York, United States of America: Addison-Wesley Publishing Company, Inc 1996.
- [8] Rumelhart D, McClelland J. Parallel Distributed Processing. United States of America: The MIT Press 1986.
- [9] Rumelhart D, McClelland J. Parallel Distributed Processing. United States of America: The MIT Press 1986.
- [10] Hagan M, Demuth H, Beale M. Neural Network Design. [cited: 01/01/2006]; Available from: <http://ee.okstate.edu/mhagan/nnd.html>
- [11] Aibar P, Benedí M, Casacuberta F, Castro M, Marzal A, Prat F. Metodologías. 26/09/2005 [cited 26/09/2005]; Available from: http://www.dsic.upv.es/docs/bib-dig/documentacion/etd-10202003-483737/dsic_dd_9716.ps
- [12] Jordan M. Recurrent Network. 23/04/1998 [cited : 05/09/2005]; Available from: <http://www.pubserv.com/mitecs/pdf/jordan2.pdf>
- [13] Medsker L, ed. Recurrent Neural Networks - Design and Applications. Washinton, USA: CRC Press 2001.
- [14] Mandic P, Chambers A. Recurrent Neural Networks for Prediction. England: Wiley&Sons Ltd. 2001.

- [15] Vivaracho PC, Romero L. Redes Neuronales en reconocimiento de locutor. 15/01/2001 [cited 10/10/2004]; Available from: <http://lisisu02.fis.usal.es/~airene/capit2.pdf>
- [16] Chin K. A connectionist approach to acquiring semantic knowledge using competitive learning. England: Simon Fraser University; 1993.
- [17] Roa S. Clasificación de Frases del Lenguaje Natural usando Redes Neuronales Recurrentes. Bogotá, Colombia: Universidad Nacional de Colombia; 2002.
- [18] Muruzabal J, Velez O. El modelo de Hopfield. SF [cited : 17/05/2006]; Available from: <http://thales.cica.es/rd/Recursos/rd98/TecInfo/07/capitulo5.html>
- [19] Muñoz J. Redes Recurrentes y Autónomas. SF [cited : 17/05/2006]; Available from: <http://www.lcc.uma.es/~munozp/Tema3MC-06.pdf>
- [20] Redes de Hopfield. SF [cited : 17/05/2006]; Available from: <http://club.telepolis.com/alimanya/hopfield.htm>
- [21] Palmer A, Montaña J, Jiménez R. Los Mapas Autoorganizados de Kohonen. 1/01/2002 [cited : 06/06/2006]; Available from: <http://www.psiquiatria.com/psicologia/revista/67/3301>