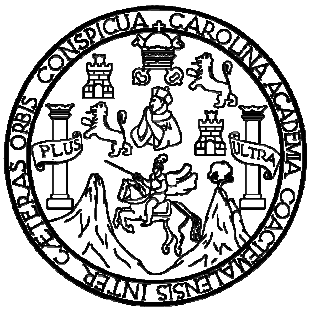
Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Eléctrica

FUNDAMENTOS PARA LA IMPLEMENTACIÓN DE RED NEURONAL PERCEPTRÓN MULTICAPA MEDIANTE SOFTWARE

José Francisco Castro García
Asesorado por el Ing. Enrique Ruiz Carballo

Guatemala, noviembre de 2006

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA



FACULTAD DE INGENIERÍA

FUNDAMENTOS PARA LA IMPLEMENTACIÓN DE RED NEURONAL PERCEPTRÓN MULTICAPA MEDIANTE SOFTWARE

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

JOSÉ FRANCISCO CASTRO GARCÍA

ASESORADO POR EL ING. ENRIQUE RUIZ CARBALLO

AL CONFERÍRSELE EL TÍTULO DE
INGENIERO ELECTRÓNICO

GUATEMALA, NOVIEMBRE DE 2006

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Murphy Olympo Paiz Recinos
VOCAL I	Inga. Glenda Patricia García Soria
VOCAL II	Lic. Amahán Sánchez Álvarez
VOCAL III	Ing. Miguel Ángel Dávila Calderón
VOCAL IV	Br. Kenneth Issur Estrada Ruiz
VOCAL V	Br. Elisa Yazminda Vides Leiva
SECRETARIA	Inga. Marcia Ivonne Véliz Vargas

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Murphy Olympo Paiz Recinos
EXAMINADOR	Ing. Francisco Javier González
EXAMINADOR	Dr. Juan Carlos Córdova
EXAMINADOR	Ing. Byron Odilio Arrivillaga
SECRETARIA	Inga. Marcia Ivonne Véliz Vargas

HONORABLE TRIBUNAL EXAMINADOR

Cumpliendo con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

Fundamentos para la implementación de red neuronal perceptrón multicapa mediante software,

tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, con fecha 21 de agosto de 2006.

José Francisco Castro García

Guatemala, 24 de octubre de 2006.

Ingeniero
Coordinador del Área de Electrónica
Escuela de Ingeniería Mecánica Eléctrica
Universidad de San Carlos de Guatemala

Estimado Ingeniero:

Por este medio le informo que he revisado el trabajo de graduación titulado: "**Fundamentos para la implementación de red neuronal Perceptrón multicapa mediante software**", elaborado por el estudiante **José Francisco Castro García**.

El mencionado trabajo llena los requisitos para dar mi aprobación, e indicarle que el autor y mi persona somos responsables por el contenido y conclusiones del mismo.

Atentamente,

Ing. Enrique Edmundo Ruiz Carballo
Colegiado 2225
ASESOR

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

Guatemala, 02 de octubre 2006.

Señor Director
Ing. Mario Renato Escobedo Martìnez
Escuela de Ingenieria Mecànica Elèctrica
Facultad de Ingenieria, USAC.

Señor Director:

Me permito dar aprobación al trabajo de Graduación titulado: **Fundamentos para la implementaciòn de red neuronal Perceptròn multicapa mediante software.** desarrollado por el estudiante; Josè Francisco Castro Garcìa, por considerar que cumple con los requisitos establecidos para tal fin.

Sin otro particular, aprovecho la oportunidad para saludarle.

Atentamente,

ID Y ENSEÑAD A TODOS

Ing. Julio Cesar Solares Peñate
Coordinador Area de Electrònica

UNIVERSIDAD DE SAN CARLOS
FACULTAD DE INGENIERIA
DIRECCION ESCUELA DE INGENIERIA
MECANICA ELECTRICA
DE GUATEMALA

JCSP/sro

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

El Director de la Escuela de Ingenieria Mecànica Elèctrica, después de conocer el dictamen del Asesor, con el Visto Bueno del Coordinador de Area, al trabajo de Graduación del estudiante; Josè Francisco Castro Garcia titulado: **Fundamentos para la implementaciòn de red neuronal Perceptròn multicapa mediante software**, procede a la autorización del mismo.

Ing. Mario Renato Escobedo Martínez
DIRECTOR

UNIVERSIDAD DE SAN CARLOS
FACULTAD DE INGENIERIA
DIRECCION ESCUELA DE INGENIERIA
MECANICA ELECTRICA
DE GUATEMALA

GUATEMALA, 3 DE NOVIEMBRE 2,006.

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

Ref. DTG. 485.2006

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Mecánica Eléctrica, al trabajo de graduación titulado: **FUNDAMENTOS PARA LA IMPLEMENTACIÓN DE RED NEURONAL PERCEPTRÓN MULTICAPA MEDIANTE SOFTWARE,** presentado por el estudiante universitario **José Francisco Castro García,** procede a la autorización para la impresión del mismo.

IMPRÍMASE.

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

Ing. Murphy Olympo Paiz Recinos
DECANO

Guatemala, noviembre 9 de 2006

/gdech

Todo por ti, Carolingia Mía
Dr. Carlos Martínez Durán
2006: Centenario de su Nacimiento

AGRADECIMIENTOS A:

Dios	Por darme la vida, guiar mi camino y ayudarme a alcanzar este triunfo.
Mis padres	Francisco de Jesús Castro Palma y Dilia Amparo García de Castro, por su cariño y apoyo para alcanzar este momento en mi vida.
Mis hermanos	Luis Rodolfo, Digby, Marlon, Guisela, Erika y Conchita, por el apoyo y cariño de siempre.
Mis amigos	Por su amistad sincera y los momentos que hemos compartido.
Mi asesor	Por su colaboración en la realización de este trabajo de graduación.

A todos aquellos que, de alguna forma, me han ayudado a lo largo de toda mi vida.

Dedicado a mis queridos y excelentes padres
Francisco de Jesús y Dilia Amparo,
como muestra de agradecimiento a su amor, esfuerzo y dedicación…

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES……..……..…V
GLOSARIO……..……..…IX
RESUMEN……XIII
OBJETIVOS……..XV
INTRODUCCIÓN……XII

1. **INTRODUCCIÓN AL PROCESAMIENTO DE INFORMACIÓN**…….....…1
   1.1. Sistemas clásicos de control……3
   1.2. Sistemas de procesamiento secuencial……5
   1.3. Principios biológicos neuronales……7
      1.3.1. Elementos de una neurona……...9
         1.3.1.1. El soma……..……10
         1.3.1.2. El axón……..……..10
         1.3.1.3. Las dendritas……11
         1.3.1.4. Las sinapsis……..11
      1.3.2. Procesamiento de información en el cerebro……..12
      1.3.3. Teoría básica del aprendizaje……15

2. **REDES NEURONALES ARTIFICIALES**……..……17
   2.1. Introducción a las de redes neuronales artificiales……19
   2.2. Elementos de una red neuronal artificial……..21
      2.2.1. Estructura de una neurona artificial……22
         2.2.1.1. Entradas……....22
         2.2.1.2. Pesos sinápticos……23
         2.2.1.3. Funciones de activación……..23
            2.2.1.3.1. Limitador fuerte……24

2.2.1.3.2. Limitador fuerte simétrico……..25
2.2.1.3.3. Función lineal……26
2.2.1.3.4. Función sigmoidal logarítmica……27
2.2.1.3.5. Función sigmoidal tangente hiperbólica……28
2.2.1.4. Salida……..28
2.3. Formas de interconexión y topologías……..29
2.4. Mecanismo de aprendizaje……30
2.4.1. Aprendizaje supervisado……31
2.4.2. Aprendizaje no supervisado……..32
2.5. Principales tipos de redes neuronales……34
2.5.1. Redes de propagación hacia delante……..34
2.5.2. Redes de propagación retroalimentadas……..35
2.5.3. Redes de propagación con retardos de tiempo……..36

3. **EL PERCEPTRÓN**……..37
3.1. Perceptrón simple……41
3.1.1. Estructura de la red……42
3.1.2. Análisis del perceptrón simple……43
3.1.3. Regla de aprendizaje……48
3.1.4. Limitaciones de la red……55
3.2. Perceptrón multicapa……59
3.2.1. Estructura de la red……..62
3.2.2. Reglas de aprendizajes……63
3.2.2.1. Regla delta generalizada……..64
3.2.2.2. Método del gradiente descendiente con velocidad de aprendizaje variable……70
3.2.2.3. Método del gradiente descendiente con momento……71
3.2.2.4. Algoritmo de Levenberg Marquardt……..72
3.2.3. Descripción de aplicaciones……73

4. **IMPLEMENTACIÓN UTILIZANDO MATLAB**……..…..77
4.1. El entorno de Matlab……78
4.2. Implementación de arreglos vectoriales y matriciales en Matlab…….80
4.3. Herramientas para redes neuronales……..83
4.3.1. Creación de red perceptrón multicapa……..…….85
4.3.2. Entrenamiento de la red……..88
4.3.3. Simulación de la red……92
4.4. Formas de aplicación……..……94
4.4.1. Ejemplo de reconocimiento de caracteres numéricos en imágenes……..……96

CONCLUSIONES……..101
RECOMENDACIONES……..103
REFERENCIAS……..……..105
BIBLIOGRAFÍA……..……..107
APÉNDICE A……..……..109
APÉNDICE B……..……..113

ÍNDICE DE ILUSTRACIONES

FIGURAS

1. Elementos en el procesamiento de información……..……2
2. Respuesta de un sistema lineal……..4
3. Principales componentes de una neurona……..10
4. Unión sináptica……..12
5. Potencial excitador e inhibidor en el interior de la dendrita……14
6. Potencial de acción en el axón……..14
7. Estructura de una neurona artificial……..22
8. Función limitador fuerte……25
9. Función limitador fuerte simétrico……..26
10. Función lineal……..26
11. Función sigmoidal logarítmica……..27
12. Función sigmoidal tangente hiperbólica……28
13. Esquema de red neuronal artificial……..……29
14. Aprendizaje supervisado……..32
15. Aprendizaje no supervisado……..……33
16. Red de propagación hacia delante……..34
17. Red de propagación retroalimentada……..35
18. Red de propagación con retardos de tiempo……..36
19. Fotoperceptrón de Rosenblatt……..……38
20. Perceptrón de Minsky y Papert……..……..40
21. Estructura del perceptrón simple……..……42
22. Perceptrón de una neurona con dos entradas……..……43

23. Gráfica x_2 vrs. x_1……..44
24. Perceptrón simple de una neurona……..45
25. Regiones de salida del perceptrón de una neurona……..46
26. Plano dividiendo un espacio tridimensional……..47
27. Comportamiento del perceptrón del ejemplo……54
28. Puntos de la función XOR……56
29. Red neuronal que realiza la función XOR……57
30. Regiones de decisión de la función XOR……..58
31. Región formada por 5 neuronas de capa oculta……..61
32. Estructura del perceptrón multicapa……62
33. Codificador de información……74
34. Clasificador de imágenes visuales……..75
35. Control automático de proceso industrial……76
36. Entorno de trabajo de Matlab……78
37. Ventana principal de la interfaz gráfica de redes neuronales……83
38. Ventana para crear un arreglo en la interfaz gráfica……..84
39. Ventana para crear una red neuronal en la interfaz gráfica……86
40. Ventana para entrenar una red neuronal en la interfaz gráfica……89
41. Ventana para simular una red neuronal en la interfaz gráfica……92
42. Posicionamiento de los píxeles en el arreglo de entrada……..97
43. Imágenes usadas en el entrenamiento de la red……98
44. Gráfica del Error vrs. Iteraciones realizadas en el entrenamiento……99
45. Imágenes que no se usaron en el entrenamiento de la red……..100

TABLAS

I. Función lógica AND……51
II. Función lógica XOR……55
III. Salida para imágenes con el número uno……109
IV. Salida para imágenes con el número dos……109
V. Salida para imágenes con el número tres……110
VI. Salida para imágenes con el número cuatro……110
VII. Salida para imágenes con el número cinco……110
VIII. Salida para imágenes con el número seis……111
IX. Salida para imágenes con el número siete……111
X. Salida para imágenes con el número ocho……111
XI. Salida para imágenes con el número nueve……112
XII. Salida para imágenes con el número cero……112

GLOSARIO

Algoritmo de aprendizaje	Conjunto de pasos bien definidos para la solución de un problema de aprendizaje en una red neuronal.
Aprendizaje	Proceso en el cual una red neuronal modifica sus pesos sinápticos en respuesta a una entrada, para proporcionar la salida adecuada.
Arreglo	Estructura de dato que permite almacenar información en varias localidades o posiciones de memoria, dispuestas en una o mas dimensiones.
Axón	Prolongación de una neurona, por la que ésta transmite impulsos nerviosos hasta otras neuronas.
Capa	Agrupación de neuronas dispuestas en un nivel de una red neuronal.
Codificar	Transformar, mediante las reglas de un código, la formulación de un mensaje.
Constante de momento	Término agregado al método del gradiente descendiente para obtener una convergencia más rápida.
Converger	Aproximarse a un valor numérico.

Dendritas	Prolongación ramificada de una neurona, mediante la cual ésta recibe estímulos externos.
Entrada total	Suma de todas las entradas a una neurona multiplicadas por sus pesos sinápticos.
Entrenamiento	Proceso en el cual una red neuronal modifica sus pesos sinápticos en respuesta a una entrada, para proporcionar la salida adecuada.
Estímulo	Incitación para realizar una acción.
Fluido extracelular	Sustancia que rodea una célula nerviosa (neurona).
Función de activación	Función matemática que transforma la entrada total en la respuesta de una neurona artificial.
Función sigmoidal	Tipo de función de activación continua en todo su rango que describe la forma de la letra s.
Generalización	Habilidad de una red neuronal de almacenar en sus pesos sinápticos las características comunes de los ejemplos que fueron usados en el entrenamiento.
Gradiente	Razón entre la variación de una magnitud entre dos puntos próximos y la distancia que los separa.
Hiperespacio	Espacio formado por más de tres dimensiones.

Hiperplano	Plano de n-1 dimensiones formado en un espacio de n dimensiones -Hiperespacio.
Linealmente separable	Se dice de aquello puntos de un espacio de n dimensiones que pueden separarse por medio de un plano de n-1 dimensiones.
Membrana postsináptica	Tejido que rodea la dendrita de una neurona.
Membrana presináptica	Tejido que rodea el axón de una neurona.
Montículo del axón	Parte de una célula nerviosa -neurona- en la cual se une el axón con el cuerpo de la célula.
Neurobiología	Ciencia que estudia la biología del sistema nervioso.
Neurona	Célula nerviosa con características distintas a las demás células, consta de un cuerpo celular llamado soma del que parten diversas prolongaciones.
Neurotransmisor	Sustancia que permite la transmisión de señales entre el axón de una neurona y la dendrita de otra.
Peso sináptico	Valores numéricos constantes que ponderan los valores de entrada de una neurona artificial.

Potencial de acción	Potencial eléctrico que se forma en interior del axón de una neurona durante la transmisión de una señal.
Potencial excitador	Potencial eléctrico que se produce en el interior de una dendrita que contribuye a la formación de un potencial de acción.
Potencial inhibidor	Potencial eléctrico que se produce en el interior de una dendrita que se opone a la formación de un potencial de acción.
Procesamiento de información	Aplicación sistemática de una serie de operaciones sobre un conjunto de datos para abstraer la información importante.
Sinapsis	Punto de contacto entre el axón de una neurona y la dendrita de otra, el cual permite la transmisión de señales por medio de neurotransmisores.
Soma	Porción central de una neurona que contiene el núcleo y tiene unida una o más ramificaciones llamadas dendritas y axones.
Vector de entrada	Parte de la estructura de una red neuronal que contiene los valores de entrada a la red.
Velocidad de aprendizaje	Factor utilizado en el método del gradiente descendiente para asegurar que la red llegue a asentarse en una solución.

RESUMEN

Las redes neuronales artificiales son modelos de procesamiento de información, inspirados en el funcionamiento del cerebro; éstas han brindado una alternativa para aquellos problemas, en los cuales los métodos tradicionales no han tenido los resultados deseados.

Debido a su constitución, las redes neuronales presentan un gran número de características semejantes a las del cerebro; por ejemplo, son capaces de aprender de la experiencia, de generalizar de casos anteriores a nuevos casos, pueden procesar correctamente datos incompletos o distorsionados y ser capaces de seguir funcionando, adecuadamente, a pesar de sufrir lesiones, ya que, la información se halla distribuida por toda la red. Las redes neuronales no ejecutan una secuencia de pasos como los sistemas de cómputo tradicionales, sino que responden en forma paralela a las entradas que se les presenta.

El perceptrón multicapa es una red neuronal que puede implementarse, fácilmente, utilizando un computador convencional y el software adecuado, este tipo de red neuronal responde, rápidamente, a las entradas que se le presentan; su programación consiste en presentarle un grupo de ejemplos, sin necesidad de desarrollar complicados algoritmos; su salida se genera, rápidamente, por la propagación de las señales a través de sus capas.

OBJETIVOS

- **General**

  Proporcionar la información fundamental para implementar la red neuronal perceptrón multicapa en diversas aplicaciones, en las cuales no se han obtenido buenos resultados utilizando los métodos tradicionales.

- **Específicos**

1. Indicar las desventajas de los sistemas de control clásicos y sistemas de procesamiento secuencial en determinadas tareas.

2. Dar una descripción abstracta del funcionamiento de los sistemas biológicos neuronales, estableciendo los principios que dieron origen a las redes neuronales artificiales.

3. Proporcionar información acerca de la teoría de las redes neuronales artificiales.

4. Describir la estructura y la forma de aprendizaje de la red neuronal perceptrón multicapa.

5. Indicar una forma de implementar la red neuronal perceptrón multicapa utilizando software.

INTRODUCCIÓN

Las Redes neuronales artificiales son una teoría que aún esta en proceso de desarrollo; a pesar de que su verdadera potencialidad aún no se ha alcanzado, éstas han brindado una alternativa a la computación clásica, para aquellos problemas que resultan muy difíciles de resolver, por medio de algoritmos.

Las redes neuronales son sistemas de procesamiento que simulan muchas de las habilidades del cerebro; éstas son capaces de aprender de la experiencia, de generalizar, de casos anteriores a nuevos casos, pueden procesar, correctamente, datos incompletos o distorsionados y ser capaces de seguir funcionando, adecuadamente, a pesar de sufrir lesiones. Las redes neuronales no ejecutan una secuencia de operaciones, sino que responden en paralelo a las entradas que se les presenta.

Este trabajo presenta un estudio de las redes neuronales artificiales, en el cual se indican sus principales características y ventajas, frente a los sistemas de procesamiento tradicionales. Debido a la gran cantidad de tipos de redes neuronales existentes, el estudio se enfoca en el perceptrón multicapa que es el tipo de red neuronal de mayor uso, en la actualidad.

En el primer capítulo se describen las principales características de los sistemas tradicionales de procesamiento de información y se explican los principios biológicos neuronales que dieron origen a la teoría de las redes neuronales artificiales.

En el segundo capítulo se realiza una introducción a la teoría de las redes neuronales, describiendo los elementos que forman una red neuronal, los mecanismos de aprendizaje y los principales tipos de redes neuronales.

El tercer capítulo proporciona información respecto de las redes neuronales perceptrón simple y perceptrón multicapa, indicando sus estructuras, algoritmos de aprendizaje y capacidades, este capítulo finaliza con una breve descripción de algunas aplicaciones del perceptrón multicapa.

Finalmente, el cuarto capítulo explica cómo implementar la red neuronal perceptrón multicapa utilizando el software Matlab, al final del capítulo se realiza un ejemplo donde se utiliza este tipo de red neuronal para reconocer caracteres numéricos dibujados en imágenes de mapa de bits.

1. INTRODUCCIÓN AL PROCESAMIENTO DE INFORMACIÓN

Uno de los principales objetivos y preocupaciones científicas a lo largo de la historia, ha sido el poder diseñar y construir máquinas capaces de realizar diversas actividades con cierta inteligencia; de los intentos realizados en este sentido, se han llegado a definir métodos para diseñar sistemas, que tomen decisiones o realicen acciones, de acuerdo a los requerimientos de una actividad.

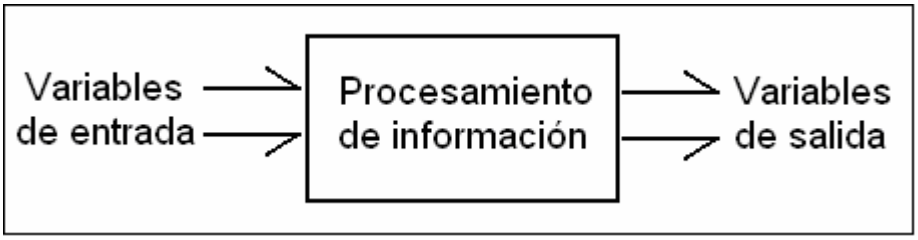
En un sentido general, procesar información se le llama a todas aquellas acciones encaminadas a abstraer de determinado grupo de información, los datos que nos interesan; también se puede decir, que el procesamiento de información, consiste en transformar información no utilizable en información útil para poder tomar una decisión.

En el campo de la ingeniería electrónica, el procesamiento de la información, es la transformación de señales tomadas de un medio físico, en señales útiles para realizar una tarea de forma automática. De acuerdo con la descripción anterior, los elementos que intervienen en el procesamiento de la información son: las variables de entrada, el sistema de procesamiento y las variables de salida.

Las variables de entrada contienen información sobre el medio físico, pero ésta, no puede ser usada directamente para tomar una decisión o realizar una acción, por lo que es necesario su procesamiento. Las variables de salida, son la abstracción de la información de interés en la entrada, éstas son usadas en la toma de decisiones o para realizar una acción en un sistema dado.

El sistema de procesamiento de datos, es el encargado de transformar las variables de entrada, en variables de salida; éste toma de las variables de entrada la información que es de interés, ordenándola y colocándola en las variables de salida. En la figura 1 puede observarse la interacción entre los elementos que intervienen en el procesamiento de información.

Figura 1. Elementos en el procesamiento de información



Los sistemas de procesamiento pueden ser agrupados básicamente en tres grupos, éstos son: los sistemas clásicos de control, los sistemas de procesamiento secuencial y las redes neuronales artificiales. Con el avance en la velocidad de los microprocesadores, los sistemas de procesamiento secuencial pueden utilizarse en muchas de las aplicaciones en las que se usaban sistemas de control clásicos; las redes neuronales artificiales han surgido como posibles soluciones a problemas que no pueden ser resueltos por sistemas de procesamiento secuencial.

No debe pensarse que un sistema de procesamiento puede usarse en todas las aplicaciones, por ejemplo, existen aplicaciones que no pueden realizarse usando sistemas de procesamiento secuencial, por la complejidad que implica la programación, siendo más adecuado realizarlas por medio de redes neuronales artificiales; también hay aplicaciones que por su naturaleza, resulta mas adecuado realizarlas por sistemas de procesamiento secuencial o sistemas de control lineales.

1.1. Sistemas clásicos de control

El término sistema de control es aplicable al conjunto de elementos que son colocados en un sistema fijo, para que éste se comporte de la manera deseada. De la definición anterior, un sistema de control está integrado por los elementos encargados de realizar las mediciones, el sistema de procesamiento de datos y los actuadores. Aunque un sistema de procesamiento de datos es sólo una parte de un sistema de control completo, ésta constituye la de mayor importancia. En el sistema de procesamiento, las variables de entrada son obtenidas de los elementos que hacen las mediciones y las variables de salida, son ingresadas a los actuadores para que realicen una acción en el sistema a controlar.

Es común utilizar el término señal para todas aquellas formas de ondas eléctricas que llevan información, por tanto, los términos procesamiento de señales y procesamiento de información o datos tienen asociado el mismo concepto. Un sistema de procesamiento de señales puede ser representado matemáticamente por medio de funciones, en donde, las señales de entrada son las variables independientes y las señales de salida son las variables dependientes; por ejemplo, un sistema de procesamiento con señal de entrada asociada a la variable **x** y señal de salida asociada a la variable **y**, puede ser representado mediante la formula 1-1, en donde la función $f(\,)$ representa el sistema de procesamiento.

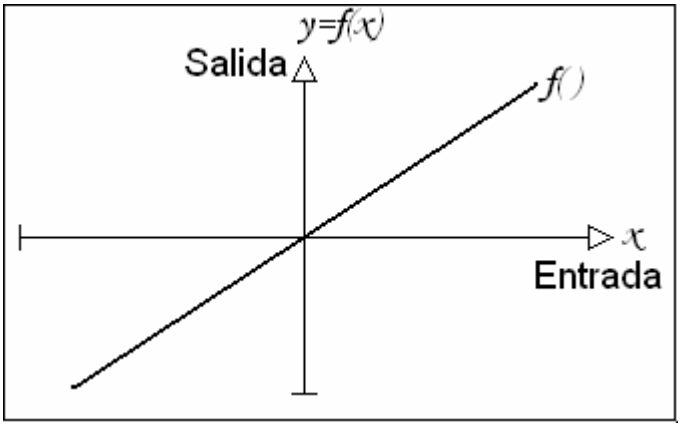
$$y = f(x) \qquad [1\text{-}1]$$

Los sistemas de control clásicos, se modelan como sistemas lineales e invariantes en el tiempo, éstos poseen muchas ventajas en lo referente a su análisis.

Un sistema lineal es aquel que cumple con el principio de superposición, el cual se define de la siguiente forma: si $y_1 = f(x_1)$ y $y_2 = f(x_2)$ entonces $ay_1 + by_2 = f(ax_1 + bx_2)$, siendo a y b constantes.

Se dice que un sistema es invariante en el tiempo, si un desplazamiento temporal o retardo en la señal de entrada, provoca el mismo desplazamiento o retardo en la señal de salida. Los sistemas de control clásicos son construidos con dispositivos eléctricos o electrónicos operando linealmente, así se logra una respuesta lineal del sistema; en la figura 2 se muestra la respuesta de un sistema lineal.

Figura 2. Respuesta de un sistema lineal



Para analizar sistemas de control clásicos, lo que se hace es modelar los sistemas según sus parámetros linealizados, para luego controlarlo, sin embargo en este modelado se desprecian muchos parámetros importantes a causa de las alinealidades de los mismos, pero, si se llegaran a tomar en cuenta todos estos parámetros no lineales, el análisis se volvería demasiado complicado; por esto, se hace necesario eliminar muchos parámetros en aplicaciones reales. Sin estos parámetros el modelo es funcional, pero en la vida real, se obtienen muchas inexactitudes.

Una descripción más detallada de los sistemas clásicos de control es desarrollada en el trabajo realizado por Benjamín Kuo[1], en éste trabajo también se explican los métodos usados para el diseño y análisis de sistemas de control lineales[2].

Para el modelado de sistemas lineales, existe una gran variedad de técnicas analíticas y gráficas, con fines de diseño y análisis; mientras que los sistemas no lineales son difíciles de tratar matemáticamente.

1.2. Sistemas de procesamiento secuencial

Los sistemas de procesamiento secuencial son aquellos en los cuales la información es transformada mediante la ejecución de una serie de operaciones lógicas o matemáticas. Estos sistemas son implementados mediante microprocesadores. Los microprocesadores actuales son capaces de realizar millones de operaciones por segundo, pudiendo resolver en poco tiempo problemas de gran dificultad para las personas, lo que permite procesar información rápidamente y con gran exactitud.

Las operaciones que debe realizar el microprocesador son indicadas mediante un programa. Un programa es una secuencia de instrucciones que transforman un grupo de variables de entrada en variables de salida para dar solución a un problema. Para elaborar un programa es necesario hacer uso de un lenguaje de programación, el cual es una notación para transmitir un grupo de instrucciones a un código manejado por el microprocesador. Cuando se programa, se busca modelar a través de datos numéricos o lógicos, los elementos que intervienen en un sistema fijo para poder controlarlo.

En los sistemas de procesamiento secuencial, se pueden realizar operaciones lógicas y aritméticas, ejecutar instrucciones si se cumple determinada condición, repetir un grupo de instrucciones un determinado número de veces o hasta que una condición se cumpla, comparar datos numéricos, almacenar información de eventos pasados y otras operaciones, según el tipo de microprocesador que se utilice.

En el trabajo presentado por Larry Long[3], se describen los métodos utilizados para el desarrollo de programas en sistemas de procesamiento secuencial.

Los sistemas de procesamiento secuencial se han implementado en muchas aplicaciones en las que eran utilizados los sistemas clásicos de control debido a las grandes ventajas que ofrecen.

Los sistemas clásicos de control no resultan adecuados cuando se necesita modelar parámetros no lineales; este problema no afecta a los sistemas de procesamiento secuencial. Se pueden hacer programas que efectúen operaciones no lineales, con esto se puede obtener mayor exactitud en aplicaciones que presenten alinealidades en sus parámetros.

A pesar de que actualmente los sistemas de procesamiento secuencial son capaces de realizar millones de operaciones por segundo, no son capaces de entender que significan las formas en las imágenes visuales o distinguir entre distintas clases de objetos. Muchas aplicaciones no se han automatizado debido a las complejidades que implica la programación de un computador para llevar a cabo estas tareas; cuando se trata de interpretar el mundo estos sistemas de procesamiento no son capaces de diferenciar entre objetos similares y objetos totalmente diferentes.

La razón de la incapacidad de distinguir entre diferentes objetos, seguramente es porque en estas aplicaciones la información que se maneja es imprecisa, dificultando su modelado mediante datos numéricos o lógicos. Cuando se intenta manejar información imprecisa con este tipo de sistemas de procesamiento, la programación resulta demasiado compleja y generalmente no se obtienen los resultados deseados.

La incapacidad de los computadores actuales para interpretar al mundo, no indica que estas máquinas sean completamente inadecuadas, hay muchas tareas que resultan especialmente adecuadas para ser resueltas mediante computadores convencionales, por ejemplo, la resolución de problemas matemáticos y científicos, creación, manipulación y mantenimiento de bases de datos, comunicaciones electrónicas, procesamientos de textos y muchas funciones de control.

1.3. Principios biológicos neuronales

El cerebro humano constituye un procesador de información muy notable, es capaz de procesar a un ritmo veloz gran cantidad de información imprecisa suministrada por los sentidos, puede dar respuestas adecuadas incluso en situaciones nuevas.

El cerebro forma parte del sistema nervioso central y se encuentra ubicado dentro del cráneo, es una masa de tejido gris-rosáceo compuesto de células nerviosas, se calcula que el cerebro humano posee más de cien mil millones de células nerviosas o neuronas, éstas están conectadas unas con otras y son responsables del control de todas las funciones mentales.

Asimismo, el cerebro es el centro de control del movimiento, del sueño, del hambre, de la sed y de casi todas las actividades vitales necesarias para la supervivencia. Todas las emociones humanas como el amor, el odio, el miedo, la ira, la alegría y la tristeza están controladas por el cerebro. También se encarga de recibir e interpretar las innumerables señales que se envían constantemente desde el organismo y el exterior.

Al comparar las capacidades de procesamiento de información de un computador electrónico con las de una persona, resulta irónico pensar que sistemas capaces de realizar millones de operaciones por segundo, que es de gran dificultad para las personas, no sean capaces de poder distinguir entre patrones de entrada, que es algo sencillo para una persona; esto puede entenderse si se compara la forma en que la información es procesada; la diferencia se hace visible al momento de saber que el computador hace sus operaciones por medio de procesos secuenciales y lógicos, mientras que el cerebro es multidireccional funcionando en una forma mucho más compleja ya que procesa la información sintetizando e integrando la misma a través de procesos paralelos y simultáneos.

También se sabe que el tiempo de conmutación de los componentes de un computador electrónico moderno es más de siete órdenes de magnitud más rápido que el de las células que constituyen nuestro sistema neurobiológico, pero aunque el tiempo de respuesta de una célula neuronal individual es mayor que el de los componentes de un computador, el paralelismo masivo y las interconexiones que se observan entre las células neuronales son la causa de la capacidad del cerebro para procesar información en poco tiempo. La gran cantidad de neuronas interconectadas que posee el cerebro humano y su complejo funcionamiento justifican el nivel superior de inteligencia del hombre.

Aunque aún se ignora mucho sobre la forma en que el cerebro humano procesa la información, se han deducido las características esenciales de las neuronas y sus conexiones, para construir modelos que puedan imitar su funcionamiento.

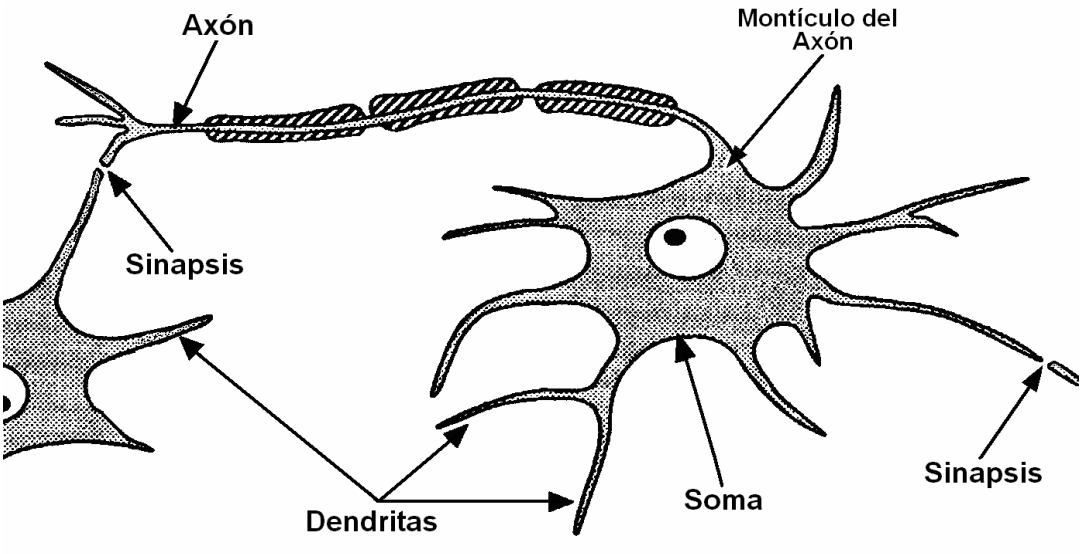
1.3.1. Elementos de una neurona

El cerebro consta de un gran número de elementos altamente interconectados llamados neuronas, cada neurona tiene aproximadamente diez mil conexiones con otras neuronas. A través de estas interconexiones una neurona recoge señales procedentes de otras neuronas, luego procesa estas señales y transmite otra señal hacia otras neuronas, de esta manera la información se transmite de unas neuronas a otras.

Las neuronas están cubiertas por una membrana que separa el interior de la célula del fluido que la rodea (fluido extracelular), esta membrana actúa de tal forma que se mantenga una diferencia de potencial entre el interior de la célula y el fluido extracelular.

El término neurona es usado para denominar a la célula nerviosa y todas sus prolongaciones; al contrario de otras células del organismo, éstas no se dividen ni se reproducen. El tamaño y forma de las neuronas es variable, pero todas tienen los mismos componentes; en la figura 3 se muestran los principales componentes de una célula neuronal, éstos son: el soma o cuerpo de la célula, el axón y las dendritas; las conexiones entre neuronas son materializadas por medio de la sinapsis.

Figura 3. Principales componentes de una neurona



Adaptado de: James Freeman. **Redes neuronales.** Pág. 9

1.3.1.1. El soma

Es la porción central o cuerpo celular, que contiene el núcleo y tiene unida una o más estructuras denominadas dendritas y axones, éstas son extensiones de la célula y están implicadas en la recepción y envío de las señales; en el soma se realiza la suma de todas las señales provenientes de otras neuronas.

1.3.1.2. El axón

El axón es una fibra larga y delgada que lleva la señal desde su montículo hasta otras neuronas; generalmente existe un gran número de ramificaciones en el extremo del axón, para permitir que las señales de una neurona sean transmitidas a muchas otras. En estos sistemas biológicos las señales consisten en diferencias de potencial que se transmiten sin atenuación.

1.3.1.3. Las dendritas

Las dendritas son extensiones de la célula, que se encargan de la recepción de señales provenientes de otras neuronas. Son un grupo de fibras nerviosas muy ramificadas y de forma irregular que están conectadas directamente al soma, cada neurona posee un gran número de dendritas que permiten recibir información de muchas otras neuronas. Las dendritas transmiten un potencial eléctrico hasta el soma, este potencial es de magnitud variable y puede ser excitador o inhibidor, según contribuya o no a la generación de un potencial de acción en el axón.

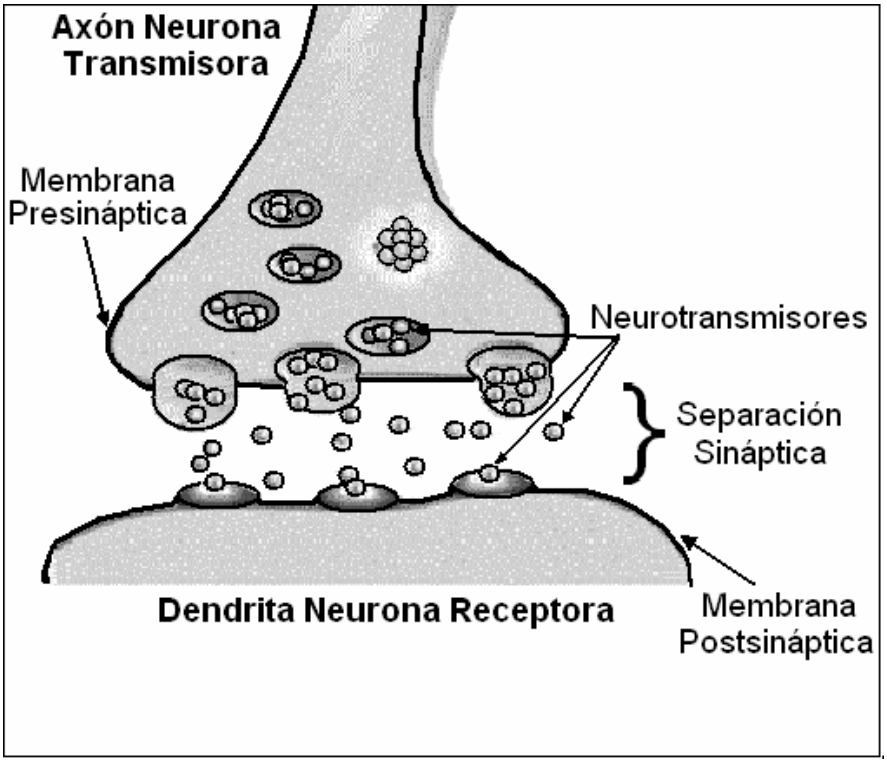
1.3.1.4. Las sinapsis

La unión existente entre dos neuronas se denomina unión sináptica o sinapsis, está unión consiste en un punto de contacto entre el axón de una neurona y la dendrita de otra neurona. En la unión sináptica se produce una conversión entre una señal eléctrica a una química. La señal eléctrica consiste en una diferencia de potencial entre el axón y el fluido extracelular, está diferencia de potencial llamado potencial de acción, hace que la membrana presináptica, que se encuentra en el extremo del axón, libere unas sustancias llamadas neurotransmisores.

Los neurotransmisores se difunden a través de la unión sináptica y se unen a la membrana postsináptica que se encuentra en el extremo de la dendrita. El efecto de los neurotransmisores sobre la membrana postsináptica, da lugar a un potencial eléctrico entre la dendrita y el fluido extracelular, este potencial eléctrico es variable y puede ser de naturaleza excitadora o inhibidora, según los neurotransmisores que se liberen; este potencial es transmitido por la dendrita hasta el soma, donde se combinará con otros potenciales.

Se puede decir que en la sinapsis se convierte una señal eléctrica (Potencial de acción) a una señal química (neurotransmisores) y luego ésta se convierte a otra señal eléctrica (Potencial excitador o inhibidor). La figura 4 muestra la unión sináptica cuando son liberados los neurotransmisores.

Figura 4. Unión sináptica



Fuente: Instituto de Investigaciones Biológicas Clemente Estable. **Neuronas y neurotransmisión.** <http://iibce.edu.uy/uas/neuronas/abc.html>

1.3.2. Procesamiento de información en el cerebro

El procesamiento de información en el cerebro es realizado por las neuronas y sus interconexiones, las señales en el cerebro son de naturaleza eléctrica y química. Las señales eléctricas consisten en diferencias de potencial eléctrico entre el interior de la célula y el fluido extracelular.

Las señales químicas consisten en neurotransmisores emitidos por la membrana presináptica del axón y absorbidas por la membrana postsináptica de la dendrita.

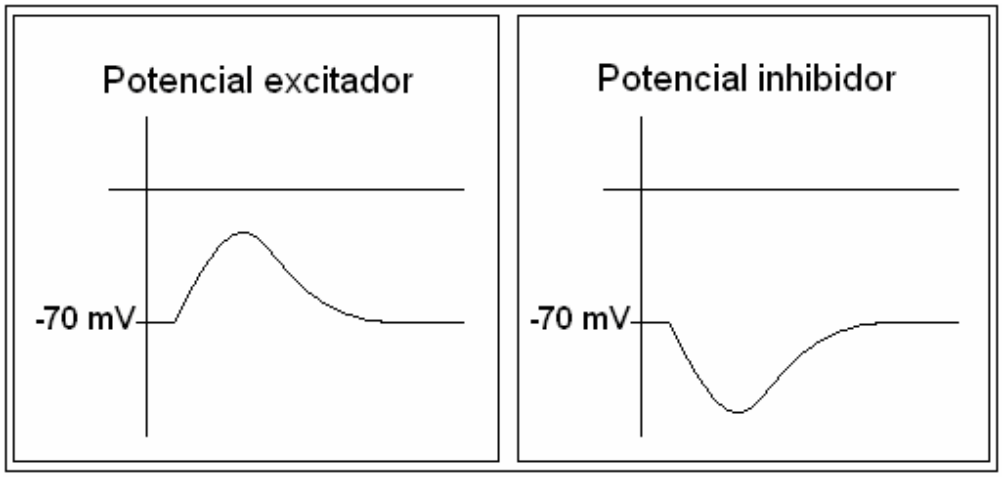
Una neurona en equilibrio químico posee un diferencia de potencial entre 70 a 100 milivoltios siendo el más negativo el interior de la célula, este potencial se denomina potencial de reposo de la neurona (- 70 mV). Una neurona emite una señal cuando su soma alcanza un potencial umbral, esto hace que su axón emita un potencial de valor positivo con respecto al fluido extracelular, a este se le llama potencial de acción.

Cuando el potencial de acción llega a la membrana presináptica se producen reacciones químicas que hacen que se liberen neurotransmisores en la separación sináptica, los neurotransmisores se difunden a través de la separación sináptica y se unen a la membrana postsináptica, la acción química que se produce da lugar a la generación de cargas eléctricas positivas o negativas en el interior de la dendrita de la neurona receptora, las cargas eléctricas producen potenciales que pueden ser excitadores o inhibidores.

Los potenciales excitadores son los que ayudan a que se forme un potencial de acción en el axón de la neurona receptora, los potenciales inhibidores cancelan el efecto de los anteriores, impidiendo la generación de un potencial de acción.

En la figura 5 se muestran las formas de onda de los potenciales excitadores y los potenciales inhibidores producidos en el interior de la dendrita, debe observarse que en estado de reposo el potencial es de -70 milivoltios.

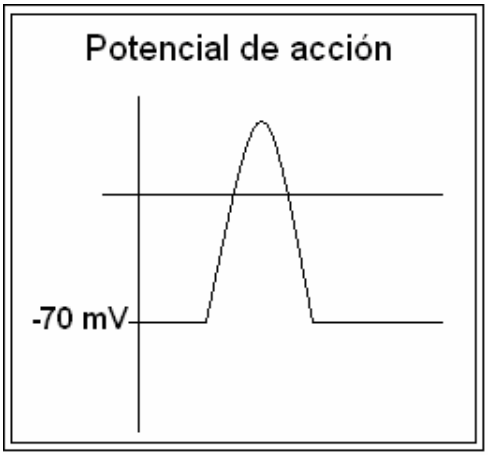
Figura 5. Potencial excitador e inhibidor en el interior de la dendrita



Los potenciales (excitadores o inhibidores) producidos en el interior de las dendritas se transmiten hasta el soma, donde son combinados (suma de potenciales eléctricos), si la combinación resultante supera un determinado potencial umbral se genera en el montículo del axón un potencial de acción.

La forma de onda del potencial de acción se muestra en la figura 6, se observa que éste potencial tiene un valor positivo con respecto al fluido extracelular.

Figura 6. Potencial de acción en el axón



Las fibras nerviosas en sí son malos conductores, el potencial de acción generado en el montículo del axón queda disipado uno o dos milímetros más adelante, para que la señal recorra varios centímetros, el potencial de acción es regenerado a lo largo del axón.

El potencial de acción se transmite por todo el axón hasta llegar a la membrana postsináptica y luego ésta libera neurotransmisores. El potencial de acción permanece aproximadamente durante 1 milisegundo, después de este tiempo la neurona regresa a su potencial de reposo, este período de tiempo limita la frecuencia de transmisión de los impulsos nerviosos a unos 1,000 por segundo.

Debe tenerse en cuenta que la descripción anterior es una visión general simplificada sobre la transmisión de señales entre neuronas en el cerebro, la neurobiología es un tema mucho mas complejo que lo descrito anteriormente.

1.3.3. Teoría básica del aprendizaje

Los sistemas biológicos no nacen preprogramados con todo el conocimiento y las capacidades que llegarán a tener eventualmente. Un proceso de aprendizaje que tiene lugar a lo largo de un período de tiempo, modifica de alguna forma la red neuronal para incluir la nueva información.

La teoría básica del aprendizaje procede de un libro escrito en 1949 por Donald O. Hebb[4], la cual proporciona una idea de cómo es guardada la información en el cerebro, ésta dice así:

Cuando un axón de una célula A está suficientemente próximo para excitar a una célula B o toma parte en su disparo de forma persistente, tiene lugar algún proceso de crecimiento o algún cambio metabólico en una de las células, o en las dos, de tal modo que la eficiencia de A, como una de las células que desencadena el disparo de B, se ve incrementada.

Dado que la conexión entre neuronas se hace a través de la sinapsis, es razonable suponer que cualesquiera cambios que puedan tener lugar durante el aprendizaje deberán producirse en ellas.

La teoría de Hebb suponía un aumento en el área de la unión sináptica, teorías mas recientes, afirman que el responsable es un incremento en la velocidad con que se liberan los neurotransmisores, en todo caso, durante el aprendizaje se producen cambios en la sinapsis.

En el cerebro el conocimiento se encuentra almacenado en las conexiones ponderadas entre las neuronas (sinapsis); en el proceso de aprendizaje, estas conexiones ponderadas se ajustan.

2. REDES NEURONALES ARTIFICIALES

Debido a la eficiencia de los procesos llevados a cabo por el cerebro, siendo inspirada en el funcionamiento de éste, se ha desarrollado la teoría de las Redes Neuronales Artificiales, las cuales emulan a las redes neuronales biológicas; éstas han brindado una alternativa a la computación clásica, para aquellos problemas, en los cuales los métodos tradicionales no han tenido los resultados deseados. Estos sistemas no requieren que la tarea a ejecutar se programe, ellos generalizan y aprenden mediante ejemplos.

En 1936, Alan Turing fue el primero en estudiar el cerebro como una forma de ver el mundo de la computación. Sin embargo, los primeros teóricos que concibieron los fundamentos de la computación neuronal fueron Warren McCulloch, un neurofisiólogo, y Walter Pitts, un matemático, quienes en 1943, lanzaron una teoría acerca de la forma de trabajar de las neuronas.

En 1949 Donald Hebb escribió un importante libro, en el que se establece una conexión entre psicología y fisiología. Fue el primero en explicar los procesos del aprendizaje (que es el elemento básico de la inteligencia humana) desde un punto de vista psicológico, desarrollando una regla de como el aprendizaje ocurría, éste es el fundamento de la mayoría de las funciones de aprendizaje que pueden hallarse en una red neuronal. Los trabajos de Hebb formaron las bases de la Teoría de las Redes Neuronales.

En 1950 Karl Lashley en sus series de ensayos, encontró que la información no era almacenada en forma centralizada en el cerebro sino que era distribuida.

En 1957 Frank Rosenblatt comenzó el desarrollo del Perceptrón, ésta es la red neuronal más antigua; éste modelo era capaz de generalizar, es decir, después de haber aprendido una serie de patrones podía reconocer otros similares, aunque no se le hubiesen presentado anteriormente. Sin embargo, tenía una serie de limitaciones, en general, era incapaz de clasificar clases no separables linealmente.

En 1960 Bernard Widrow y Marcial Hoff desarrollaron el modelo Adaline, ésta fue la primera red neuronal aplicada a un problema real; éste modelo se ha utilizado comercialmente durante varias décadas para eliminar ecos en las líneas telefónicas.

En 1969 surgieron críticas que frenaron, hasta 1982, el crecimiento que estaban experimentando las investigaciones sobre redes neuronales. En 1974 Paul Werbos desarrolló la idea básica del algoritmo de aprendizaje de propagación hacia atrás, cuyo significado quedó definitivamente aclarado en 1985.

En 1986 David Rumelhart y Geoffrey Hinton redescubrieron el algoritmo de aprendizaje de propagación hacia atrás. A partir de 1986, el panorama fue alentador con respecto a las investigaciones y el desarrollo de las redes neuronales.

Las Redes neuronales artificiales son una teoría que aún esta en proceso de desarrollo, su verdadera potencialidad aún no se ha alcanzado; aunque se han desarrollado potentes algoritmos de aprendizaje de gran valor práctico, las representaciones y procedimientos de que se sirve el cerebro son aún desconocidas.

2.1. Introducción a las de redes neuronales artificiales

Las Redes neuronales artificiales o RNA, son nuevos modelos de procesamiento de información, inspirados por la forma en que el cerebro procesa información. Aunque son una simplificación de las redes neuronales biológicas, las RNA son aptas para resolver problemas que la gente puede resolver, pero las computadoras no pueden.

Los sistemas de computo tradicional procesan la información en forma secuencial; un computador consiste por lo general de un solo procesador que puede manipular instrucciones y datos que se localizan en la memoria, el procesador lee, y ejecuta una a una las instrucciones en la memoria; este sistema es secuencial, todo sucede en una sola secuencia determinística de operaciones. Las RNA no ejecutan instrucciones, responden en paralelo a las entradas que se les presenta. El resultado no se almacena en una posición de memoria, éste es el estado de la red para el cual se logra equilibrio. El alcance de una red neuronal no consiste en instrucciones, el poder de la red está en su topología y en los valores de las conexiones entre neuronas.

Las neuronas biológicas son componentes relativamente simples del cerebro, pero cuando millares de ellas se conectan en forma conjunta se hacen muy poderosas, esto justifica la alta capacidad de cómputo del cerebro.

Una red neuronal artificial está constituida por muchos elementos simples de procesamiento, organizados en niveles y muy interconectados. El conocimiento es adquirido por la red a través de un proceso que se denomina aprendizaje; el conocimiento se almacena mediante la modificación de la fuerza o peso sináptico de las distintas uniones entre elementos de procesamiento.

Debido a su constitución, las redes neuronales artificiales presentan un gran número de características semejantes a las del cerebro. Por ejemplo, son capaces de aprender de la experiencia, de generalizar de casos anteriores a nuevos casos, de abstraer características esenciales a partir de entradas que representan información imprecisa, etc. Esto hace que ofrezcan numerosas ventajas, entre ellas se incluyen:

- Aprendizaje Adaptativo: son capaces de aprender en base a un entrenamiento o experiencia inicial.

- Generalización: una vez entrenada, a la red se le pueden presentar datos distintos a los usados durante el aprendizaje. La respuesta obtenida dependerá del parecido de los datos con los ejemplos de entrenamiento.

- Abstracción o tolerancia al ruido: son capaces de extraer o abstraer las características esenciales de las entradas aprendidas, de esta manera pueden procesar correctamente datos incompletos o distorsionados.

- Procesamiento paralelo: la información es procesada por las neuronas artificiales en forma paralela.

- Memoria distribuida: el conocimiento acumulado por la red se halla distribuido en numerosas conexiones.

- Tolerancia a fallos: una red neuronal es capaz de seguir funcionando adecuadamente a pesar de sufrir lesiones como la destrucción de neuronas o sus conexiones, ya que la información se halla distribuida por toda la red.

2.2. Elementos de una red neuronal artificial

Los elementos individuales de procesamiento en una red neuronal artificial, equivalentes a las células neuronales en una red neuronal biológica, son las neuronas artificiales. Las neuronas artificiales también son llamadas nodos, neuronodos, unidades o elementos de procesamiento.

Una neurona artificial es un modelo simplificado de una neurona biológica; en este modelo, los elementos de la célula neuronal son representados por bloques que indican operaciones matemáticas con las señales. Las señales que son representadas por flechas, son de tipo numérico real. Al igual que una neurona biológica, una neurona artificial posee muchas entradas y una sola salida, que se puede aplicar a muchas otras neuronas de la red.

En las redes neuronales artificiales, los valores de las conexiones sinápticas entre neuronas, son considerados como parte de la estructura de la neurona; a diferencia de las redes neuronales biológicas, en las cuales se considera a la sinapsis externa a la célula neuronal, constituyendo el medio de conexión entre neuronas.

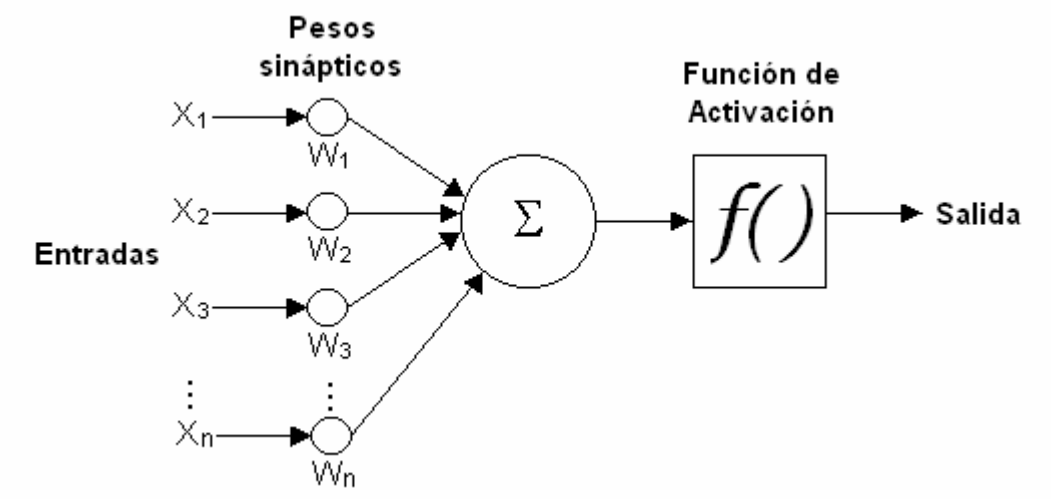
Comparando las neuronas artificiales con las biológicas, la entrada de una neurona artificial, sería el equivalente al potencial de acción en el axón de la neurona presináptica y la señal de salida sería equivalente al potencial de acción en el axón de la propia neurona.

Así, las señales en las entradas de una neurona artificial, son tomadas directamente de las salidas de otras neuronas, luego, dentro de la neurona artificial, estas señales son modificadas según sus valores sinápticos.

2.2.1. Estructura de una neurona artificial

Una neurona artificial posee diversas entradas ponderadas, un bloque sumador, una función de activación y su respectiva salida. En la figura 7, se muestra la estructura de una neurona artificial.

Figura 7. Estructura de una neurona artificial



2.2.1.1. Entradas

Las señales del entorno son suministradas a la neurona por medio de sus entradas; estas señales pueden ser externas a la red neuronal o ser la salida de otras neuronas. Las entradas en una neurona artificial son análogas a las sinapsis de una célula neuronal, debido a que éstas son el medio para el ingreso de información a la neurona.

En la figura 7 las entradas se muestran como X_1, X_2, X_3,…, X_n; las flechas indican la dirección del flujo de información.

2.2.1.2. Pesos sinápticos

Estos constituyen bloques de ganancia aplicados a las señales de entrada, es decir, son valores numéricos constantes por los cuales se multiplican las señales de entrada. Al igual que en las células neuronales, éstos pueden ser excitadores o inhibidores; un peso sináptico positivo produce una señal excitadora, ya que al multiplicarse por una entrada positiva y sumarse con las demás señales, ayuda a generar una señal de salida positiva; de la misma forma, un peso sináptico negativo produce una señal inhibidora, ya que cancela el efecto de una señal excitadora.

Los pesos sinápticos son análogos a las sinapsis entre las células neuronales y al igual que en las redes neuronales biológicas, todo el conocimiento de la red se encuentra distribuido en los valores de éstos. En la figura 7, los pesos sinápticos se muestran como W_1, W_2, W_3,…, W_n.

Los valores de los pesos sinápticos son asignados en la etapa de aprendizaje de la red, de acuerdo con el algoritmo de aprendizaje utilizado; algunos pesos sinápticos pueden tener un valor grande respecto a los demás y otros podrían anularse. Un peso sináptico de valor cero, indica que no hay conexión entre dos neuronas.

2.2.1.3. Funciones de activación

Asociada a cada neurona artificial hay una función que transforma la entrada total en la respuesta de la neurona, ésta es la función de activación. En cada neurona artificial, los productos de las entradas por sus pesos sinápticos, son sumados, ésta suma constituye la entrada total a la neurona.

La suma de las entradas ponderadas o entrada total, es transformada por la función de activación, para obtener la salida de la neurona. Esto se puede representar matemáticamente, mediante la siguiente expresión:

$$Salida = f(W_1X_1 + W_2X_2 + W_3X_3 + ... + W_nX_n) \qquad [2\text{-}1]$$

En donde, X_1, X_2, X_3,…, X_n, son las entradas; W_1, W_2, W_3,…, W_n, son los pesos sinápticos de las entradas y $f()$ es la función de activación de la neurona. En las RNA generalmente, son usadas cinco tipos de funciones de activación, éstas son: limitador fuerte, limitador fuerte simétrico, función lineal, función sigmoidal logarítmica y función sigmoidal tangente hiperbólica. A continuación se explica cada uno de estas.

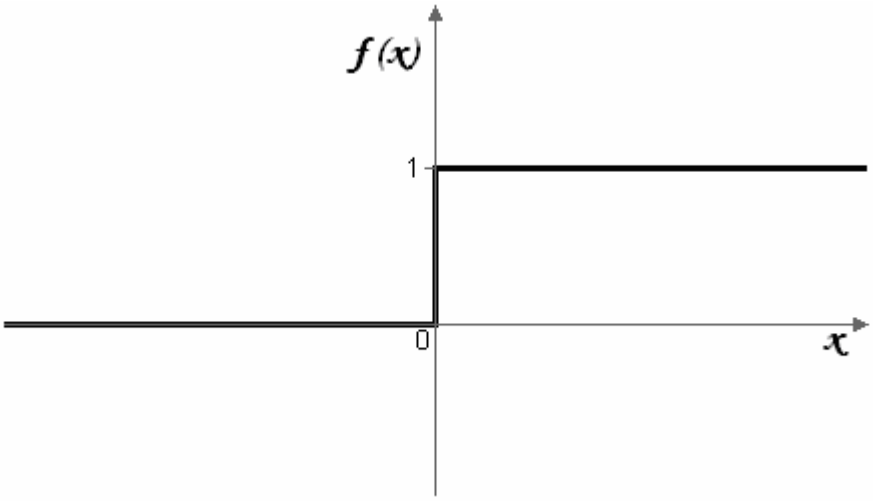
2.2.1.3.1. Limitador fuerte

En ésta se genera una salida si la entrada total supera un determinado umbral; las neuronas con éste tipo de función de activación, se denominan neuronas binarias, ya que su salida puede tomar uno de dos valores (0 o 1). La función limitador fuerte, puede representarse matemáticamente como la función escalón unitario, de la siguiente forma:

$$f(x) = u(x) \qquad [2\text{-}2]$$

La función limitador fuerte se muestra en la figura 8. Se observa que para un valor de entrada total mayor que cero, la salida es uno y para un valor menor que cero, su salida es cero.

Figura 8. Función limitador fuerte



2.2.1.3.2. Limitador fuerte simétrico

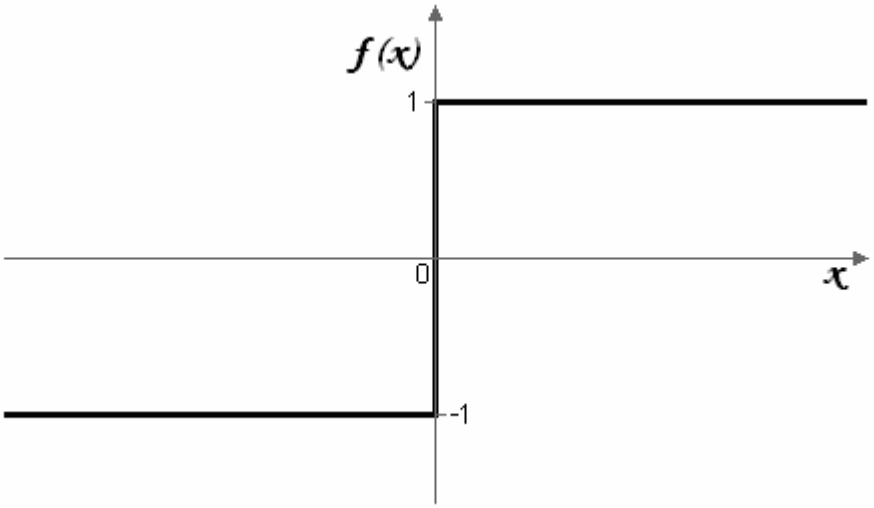
Ésta es similar a la función limitador fuerte, ya que es discontinua en cero, pero es simétrica con respecto al eje horizontal. Los valores de salida de la función limitador fuerte simétrico son 1 o -1 para todo su dominio, por tanto, las neuronas con esta función de activación son binarias, al igual que con la función limitador fuerte.

Esta función puede representarse mediante la siguiente expresión:

$$f(x) = u(x) - u(-x) \qquad [2\text{-}3]$$

La respuesta de la función limitador fuerte simétrico se muestra en la figura 9, se observa que esta no es diferenciable en cero, al igual que la función limitador fuerte.

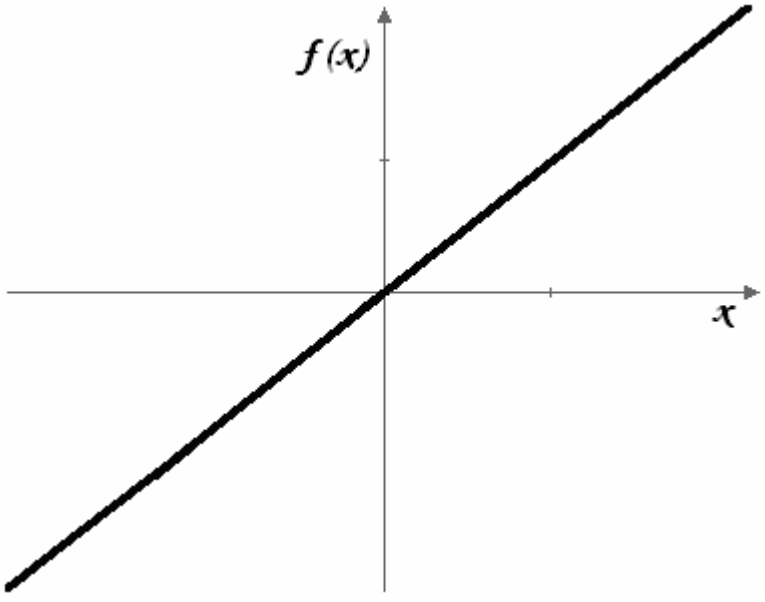
Figura 9. Función limitador fuerte simétrico



2.2.1.3.3. Función lineal

La función lineal corresponde a la función f(x)=x, esta función no tiene valores límites en su rango de salida, ésta se muestra en la figura 10.

Figura 10. Función lineal

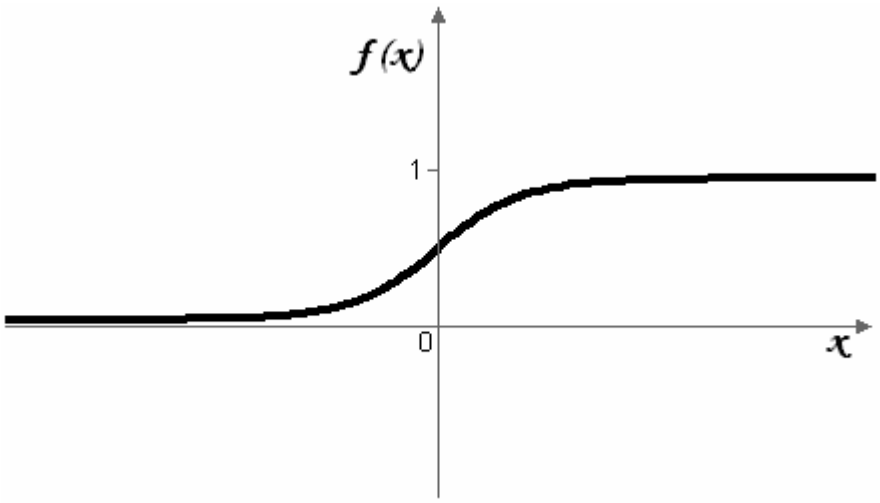


2.2.1.3.4. Función sigmoidal logarítmica

Esta función se asemeja a la función limitador fuerte, ya que su salida está comprendida entre cero y uno, pero a diferencia de ésta, es continua en todo su dominio y por tanto, es diferenciable. Debido a que muchos algoritmos de aprendizaje, necesitan hacer uso de la derivada de la función de activación, esta función tiene mayor utilidad que la función limitador fuerte, la cual no es diferenciable en cero.

En la función sigmoidal logarítmica, los valores de entrada total un poco mayores que cero, poseen una salida muy cercana a la unidad y entradas un poco menores que cero, tienen una salida cercana a cero. En la figura 11 se muestra la respuesta de esta función.

Figura 11. Función sigmoidal logarítmica



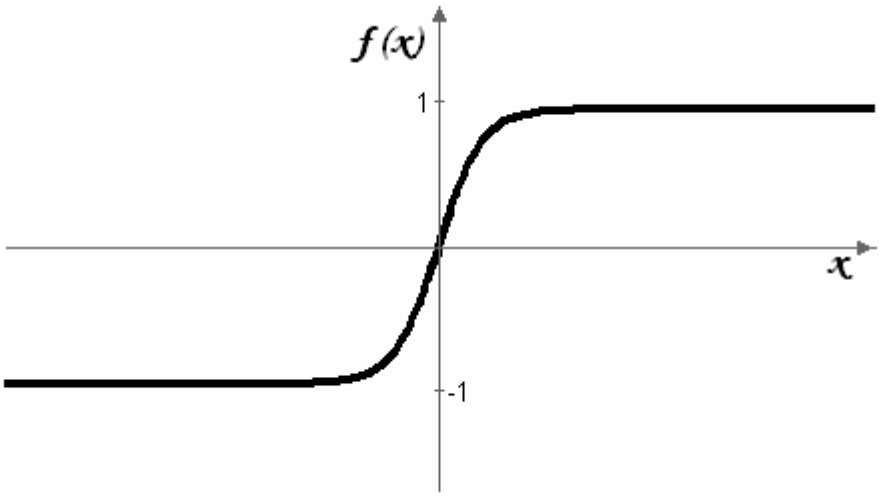
Esta función puede representarse mediante la siguiente expresión:

$$f(x) = \frac{1}{1 + e^{-x}} \qquad [2\text{-}4]$$

2.2.1.3.5. Función sigmoidal tangente hiperbólica

La función sigmoidal tangente hiperbólica es diferenciable en todo su dominio, pero a diferencia de la sigmoidal logarítmica, su salida es bipolar, ya que está comprendida entre -1 y 1. La respuesta de esta función se muestra en la figura 12.

Figura 12. Función sigmoidal tangente hiperbólica



Esta función puede representarse mediante la siguiente expresión:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \qquad [2\text{-}5]$$

2.2.1.4. Salida

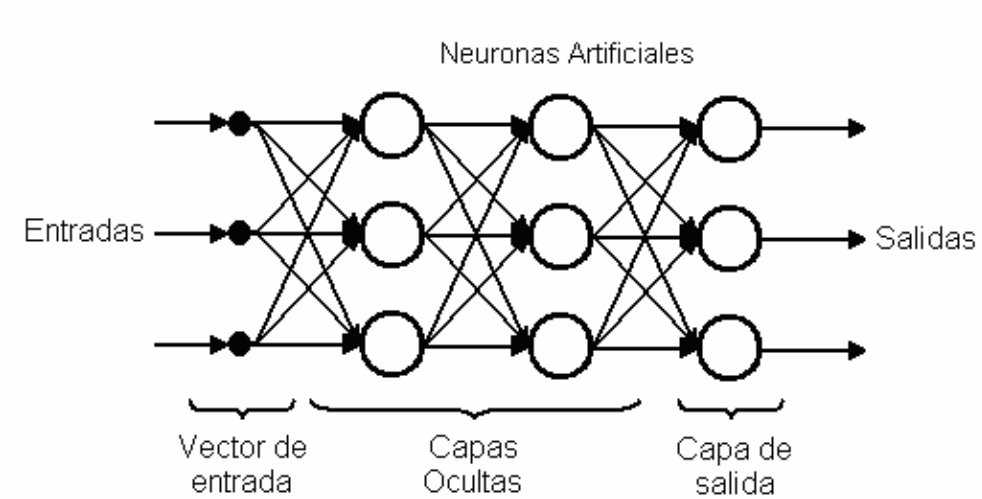
La salida de una neurona se obtiene de la función de activación, ésta es análoga al axón de las células neuronales. La salida de una neurona puede ser aplicada a una entrada de muchas otras neuronas o bien, puede ser tomada como salida de la red.

2.3. Formas de interconexión y topologías

Dentro de una red neuronal, los elementos de procesamiento se encuentran agrupados en capas; una capa es una colección de neuronas. De acuerdo a la ubicación de la capa dentro de la red, ésta puede ser oculta o de salida. Las entradas de la red se agrupan en un vector de entrada, éste no se considera una capa, pues no contiene neuronas.

Así, las redes neuronales están formadas por un vector de entrada, capas ocultas y capa de salida. El vector de entrada contiene las señales provenientes de las fuentes externas; las capas ocultas se encuentran entre el vector de entrada y la capa de salida, el número de capas ocultas puede ser desde cero hasta un número elevado; la capa de salida es la que transmite la respuesta de la red al medio externo. En la figura 13 se muestra el esquema de una red neuronal artificial formada por nueve neuronas, agrupadas en tres capas (vector de entrada, dos capas ocultas y capa de salida).

Figura 13. Esquema de red neuronal artificial



Los parámetros fundamentales de una red neuronal son: el número de entradas, el número de capas, el número de neuronas de cada capa y el grado de conectividad. El grado de conectividad, se refiere a que tantas conexiones existen, entre las neuronas de un nivel y nivel siguiente; generalmente, todas las salidas de un nivel, se conectan a todas las entradas de cada neurona del nivel posterior, así como se muestra en la figura 13.

Las redes neuronales pueden ser monocapa o multicapa, según el número de capas que contengan. Las redes monocapa, solamente están formadas por el vector de entrada y la capa de salida. Las redes multicapa además del vector de entrada y la capa de salida, poseen capas ocultas; éstas poseen mayor capacidad de procesamiento de información.

En general, un mayor número de neuronas le permite a la red neuronal, procesar información más compleja. Si la red posee muy pocas neuronas, ésta no puede aprender lo necesario para realizar una determinada aplicación. Desafortunadamente, no existe una fórmula para determinar en forma exacta el número ideal de neuronas y capas según la aplicación; aunque existen formas de determinar un número adecuado de neuronas, según el tipo de red neuronal y la aplicación; si éstas no son suficientes, deben agregarse más.

2.4. Mecanismo de aprendizaje

Todo conocimiento de una red neuronal se encuentra distribuido en los pesos sinápticos de las neuronas. Una red neuronal aprende a través de un proceso de ajuste de sus pesos sinápticos. El aprendizaje es el proceso por el cual una red neuronal modifica sus pesos sinápticos en respuesta a una entrada, para proporcionar la salida adecuada.

Los cambios que se producen durante el proceso de aprendizaje implican la destrucción, modificación y creación de conexiones entre las neuronas. La creación de una nueva conexión consiste en que el peso de la misma pasa a tener un valor distinto de cero, una conexión se destruye cuando su peso pasa a ser cero.

Se define como algoritmo de aprendizaje al conjunto de reglas bien definidas para la solución de un problema de aprendizaje. Existe gran variedad de algoritmos de aprendizaje teniendo cada uno sus propias ventajas.

Los algoritmos de aprendizaje difieren entre sí en el tipo de red para el cual están hechos y la forma como se formulan los cambios en los pesos sinápticos. El aprendizaje en una red neuronal puede ser supervisado o no supervisado. La diferencia fundamental entre ambos tipos, es la existencia o no, de un agente externo que controle el aprendizaje de la red.

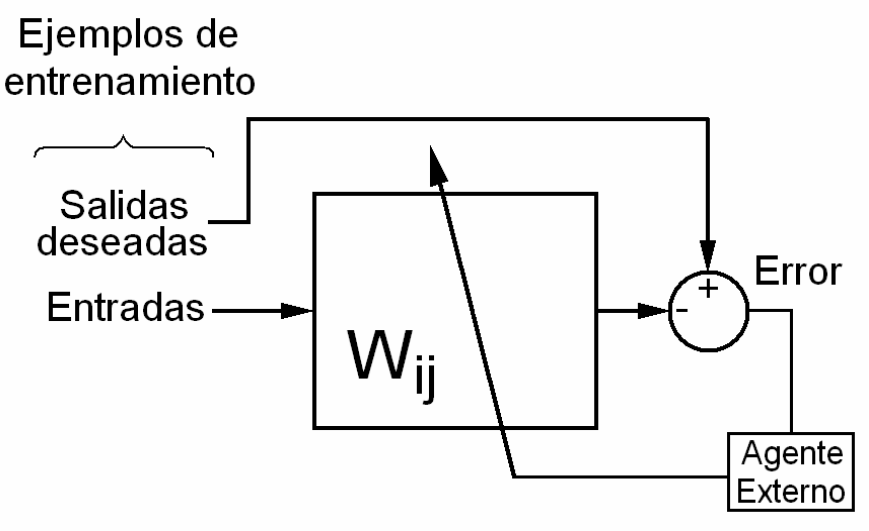
2.4.1. Aprendizaje supervisado

Los algoritmos de aprendizaje supervisado son los más populares, en éstos, el proceso de aprendizaje se realiza mediante el control de un agente externo, que determina la respuesta que debería generar la red a partir de una entrada determinada.

Un conjunto de ejemplos son presentados, uno a uno a la red; para cada entrada se calcula el valor de la salida, esta salida se compara con la salida deseada para dicha entrada y se evalúa una función de error global. El error se usa para modificar los pesos sinápticos de las neuronas, con el fin de acercar las salidas reales a las deseadas.

La figura 14 muestra gráficamente la forma en que se realiza el aprendizaje supervisado, se observa que un agente externo varía los pesos sinápticos de acuerdo al error en la salida.

Figura 14. Aprendizaje supervisado



2.4.2. Aprendizaje no supervisado

En los algoritmos de aprendizaje no supervisado, el conjunto de datos de entrenamiento consiste sólo en los patrones de entrada, por lo tanto, la red es entrenada sin el beneficio del control de un agente externo.

En el aprendizaje no supervisado la red aprende a adaptarse, basada en las experiencias recogidas, de entradas presentadas anteriormente. Las redes que implementan este tipo de aprendizaje no requieren de influencia externa para ajustar los pesos de las conexiones entre sus neuronas.

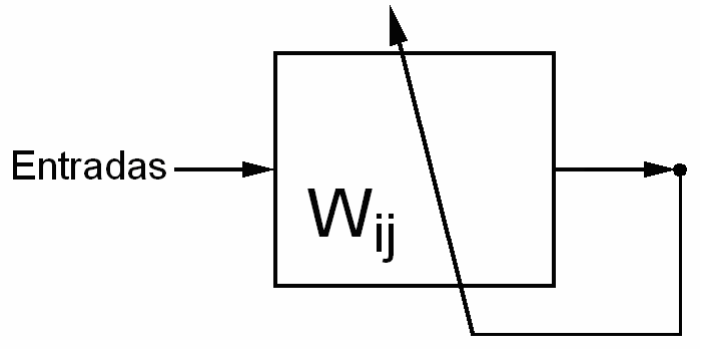
Este tipo de redes neuronales, no reciben ninguna información, por parte del entorno, que les indique si la salida generada en respuesta de una entrada, es o no correcta. Suele decirse que estas redes son capaces de auto organizarse.

Las redes neuronales que utilicen aprendizaje no supervisado deben ser capaces de encontrar las características, similitudes o categorías que se pueden establecer entre los datos que se presentan en su entrada.

En algunos casos, la salida de la red representa, el grado de similitud entre la información que se le está presentando en la entrada y la que se le ha presentado en el pasado.

En otros casos, la red podría indicar en su salida a qué categoría pertenece la información presentada, siendo la propia red quien debe encontrar las categorías apropiadas, a partir de las similitudes entre las entradas presentadas. La figura 15 describe la forma en que se realiza el aprendizaje no supervisado.

Figura 15. Aprendizaje no supervisado



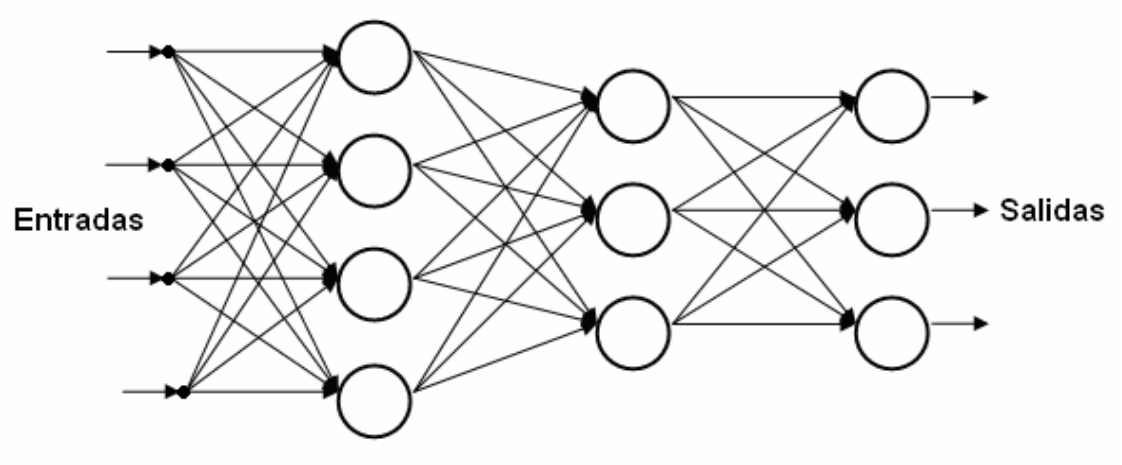
2.5. Principales tipos de redes neuronales

2.5.1. Redes de propagación hacia adelante

En estas redes, las señales se propagan hacia adelante a través de las capas de la red. En este tipo de redes las neuronas tienen sus salidas conectadas a las entradas de neuronas de capas posteriores, es decir, sus conexiones son hacia adelante. Así, las señales de las neuronas de una capa inferior son transmitidas hacia las neuronas de la capa superior.

Este tipo de redes son útiles en aplicaciones de reconocimiento o clasificación de patrones. En la figura 16 se muestran las conexiones en una red de propagación hacia adelante.

Figura 16. Red de propagación hacia adelante



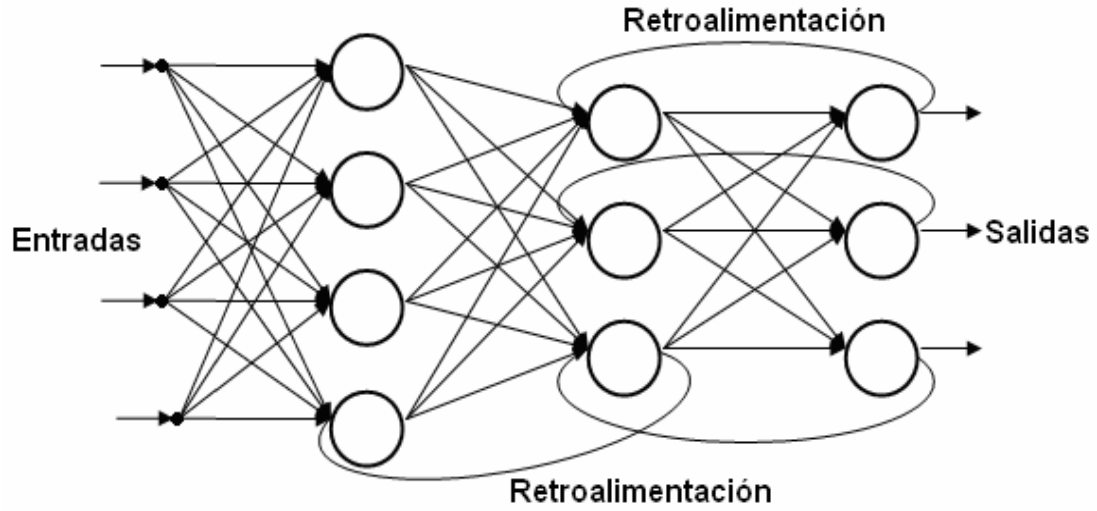
Las redes de propagación hacia adelante más conocidas son: Perceptrón, Adaline, Madaline, Asociador lineal adaptativo, Red Kohonen y Perceptrón multicapa (*Multilayer Perceptron* o *Feed Forward Backpropagation*).

2.5.2. Redes de propagación retroalimentadas

En las redes de propagación retroalimentadas, las señales se transmiten tanto hacia la capa de adelante como hacia la capa de atrás, durante el funcionamiento de la red. La retroalimentación se refiere a la forma en que las neuronas se conectan, ya que la salida de una neurona puede estar conectada a la entrada de neuronas de capas anteriores. También entran en ésta clasificación, las redes cuyas neuronas tengan conexiones con ella misma o con otras de su misma capa.

En la figura 17 se muestran las conexiones en una red de propagación retroalimentada.

Figura 17. Red de propagación retroalimentada

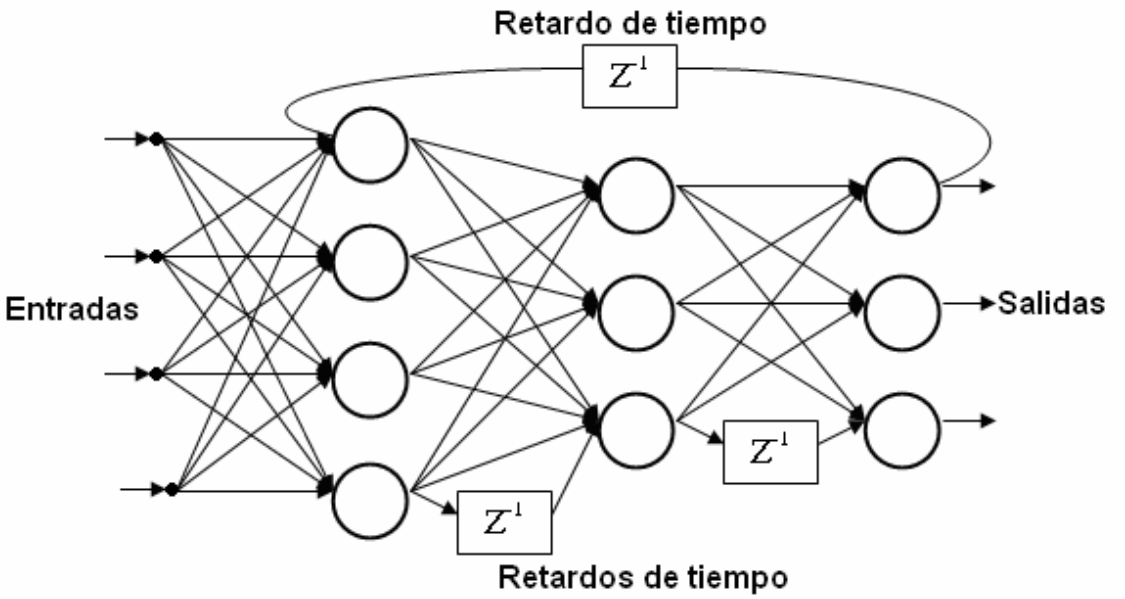


Las redes de propagación retroalimentadas más conocidas son: La red Teoría de la resonancia adaptativa (*ART, Adaptive Resonance Theory*) y la red Memoria asociativa bidireccional (*BAM, Bidirectional Associative Memory*).

2.5.3. Redes de propagación con retardos de tiempo

En este tipo de redes se incorporan elementos de memoria en las conexiones, éstos elementos de memoria almacenan las señales que se produjeron en estados anteriores. Estas redes no encuentran siempre la misma solución a un problema, a pesar que las entradas sean iguales, debido a que dependen de estados anteriores. En la figura 18 se muestra una red de propagación con retardos de tiempo.

Figura 18. Red de propagación con retardos de tiempo



Las redes de propagación con retardos de tiempo mas conocidas son la red Hopfield y la Red de propagación hacia atrás con retardos de tiempo (*Time delay backpropagation*).

3. EL PERCEPTRÓN

La primera red neuronal conocida, fue desarrollada en 1943 por el neurofisiólogo Warren McCulloch y el matemático Walter Pitts; ésta consistía en una suma de las señales de entrada, multiplicadas por unos valores de pesos escogidos aleatoriamente. La entrada total era comparada con un valor preestablecido para determinar la salida de la red. Si en la comparación, la suma de las entradas multiplicadas por los pesos era mayor o igual que el valor preestablecido, la salida de la red era uno; en caso contrario la salida era cero.

Al principio, se encontró gran similitud entre el comportamiento de éstas redes neuronales artificiales con el de los sistemas biológicos y se creyó que este modelo podía computar cualquier función aritmética o lógica.

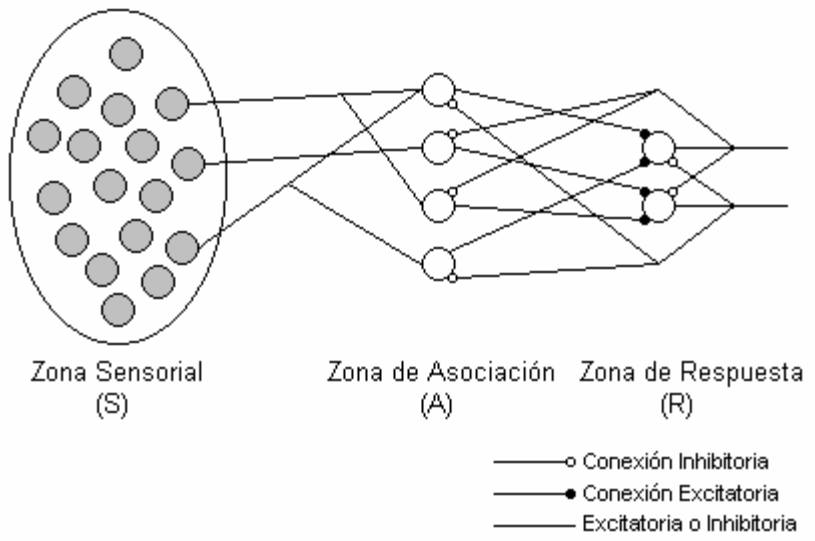
La red tipo Perceptrón fue inventada por el psicólogo Frank Rosenblatt en el año 1957. Su intención era ilustrar algunas propiedades fundamentales de los sistemas inteligentes en general, sin entrar en mayores detalles con respecto a condiciones específicas y desconocidas para organismos biológicos concretos.

Rosenblatt creía que la conectividad existente en las redes biológicas tiene un elevado porcentaje de aleatoriedad, por lo que se oponía al análisis de McCulloch y Pitts, en el cual se empleaba lógica simbólica para analizar estructuras bastante idealizadas. Rosenblatt opinaba que la herramienta de análisis más apropiada era la teoría de probabilidades, esto lo llevó a una teoría de separabilidad estadística, que utilizaba para caracterizar las propiedades más visibles de estas redes con interconexiones ligeramente aleatorias.

El primer modelo de perceptrón, desarrollado por Rosenblatt, fue el fotoperceptrón; éste era un dispositivo que respondía a señales ópticas, imitando el funcionamiento del ojo humano.

El fotoperceptrón se muestra en el figura 19; en este modelo la luz incide en los puntos sensibles S de la zona sensorial, cada punto S responde en forma todo-nada a la luz entrante (1 o 0).

Figura 19. Fotoperceptrón de Rosenblatt



Fuente: James Freeman. **Redes neuronales.** Pág. 24

Los impulsos generados por los puntos S, se transmiten a las unidades A de la capa de asociación, por medio de sus entradas. Cada unidad A recibe señales de un conjunto aleatorio de puntos S, denominados conjunto fuente de la unidad A, estas señales pueden ser excitatorias, inhibitorias o nulas (+1, -1 o 0 respectivamente). Si la suma de las entradas de la unidad A, sobrepasa algún valor umbral, ésta se activa y produce una salida que se envía a la siguiente capa de unidades.

De forma similar, las unidades A están conectadas a unidades R de la capa de respuesta y la conectividad es aleatoria entre capas; pero a diferencia de la capa de asociación, se añaden conexiones inhibitorias de realimentación desde las unidades R a las unidades A; también hay conexiones inhibitorias entre las unidades de respuesta R.

El fotoperceptrón era un dispositivo de aprendizaje; en su configuración inicial no era capaz de distinguir tramas visuales; sin embargo mediante un proceso de aprendizaje era capaz de adquirir esta capacidad. El aprendizaje implicaba un proceso de refuerzo, mediante el cual la salida de las unidades A se incrementaba o se decrementaba, dependiendo de si las unidades A contribuían o no a las respuestas correctas del fotoperceptrón para una entrada dada.

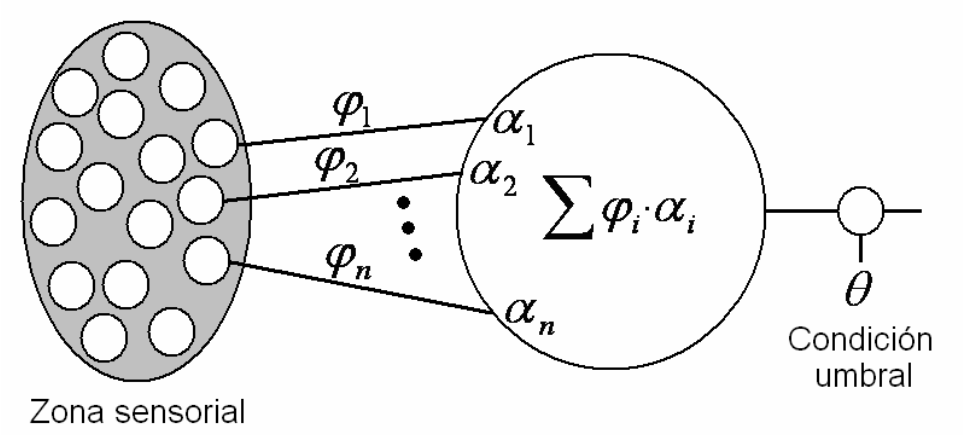
Cuando se aplicaba una trama visual a la zona sensorial, el estímulo se propagaba a través de las capas, hasta que se activase una unidad de respuesta; si se había activado la unidad de respuesta correcta, se incrementaba la salida de las unidades A que hubieran contribuido; pero si se activaba una unidad R incorrecta, se hacía disminuir la salida de las unidades A que hubiesen contribuido.

Mediante este modelo, Rosenblatt pudo demostrar que el perceptrón era capaz de clasificar tramas visuales correctamente, en lo que él denominaba un entorno diferenciado, en el cual cada clase estaba formada por tramas visuales que eran similares unas a otras en algún sentido. El fotoperceptrón también era capaz de responder de manera congruente frente a tramas aleatorias, pero su precisión iba disminuyendo a medida que aumentaba el número de tramas que intentaba aprender.

En 1969 Marvin Minsky y Seymour Papert publicaron el libro *Perceptrons: An introduction to Computational Geometry* (Perceptrones: Una introducción a geometría computacional), que para muchas personas significó el final de las redes neuronales. En este libro se presentaba un análisis detallado del Perceptrón, en términos de sus capacidades y limitaciones.

Minsky y Papert se apartaban de la aproximación probabilística de Rosenblatt y volvieron a las ideas de McCulloch y Pitts en el análisis del Perceptrón. Su modelo de Perceptrón se muestra en la figura 20, en donde φ_i es la i-ésima entrada, la cual es multiplicada por su peso asociado α_i.

Figura 20. Perceptrón de Minsky y Papert



Fuente: James Freeman. **Redes neuronales.** Pág. 27

En este modelo se observa una condición umbral en la salida; si la entrada total es mayor que el valor umbral, la salida es uno; en caso contrario, la salida es cero. Este modelo de una salida constituye una neurona artificial con función de activación limitador fuerte y entradas binarias. El modelo de Minsky y Papert del perceptrón, es el que actualmente se conoce como perceptrón simple o perceptrón.

3.1. Perceptrón simple

El modelo más sencillo de red neuronal es el perceptrón simple, se trata de una red de propagación hacia adelante, monocapa (no contiene capas ocultas) con función de activación limitador fuerte o limitador fuerte simétrico. Las entradas a la red pueden ser binarias o continuas, mientras que las salidas siempre son binarias debido a su función de activación; el valor de salida de una neurona es uno cuando la suma de sus entradas multiplicadas por sus pesos sinápticos es mayor que cero, en caso contrario su salida es cero.

Un perceptrón simple puede contener varias neuronas, pero éstas deben estar agrupadas en una sola capa y tener la misma función de activación, así las señales de entrada son aplicadas a todas las neuronas y las salidas son obtenidas de estas mismas neuronas.

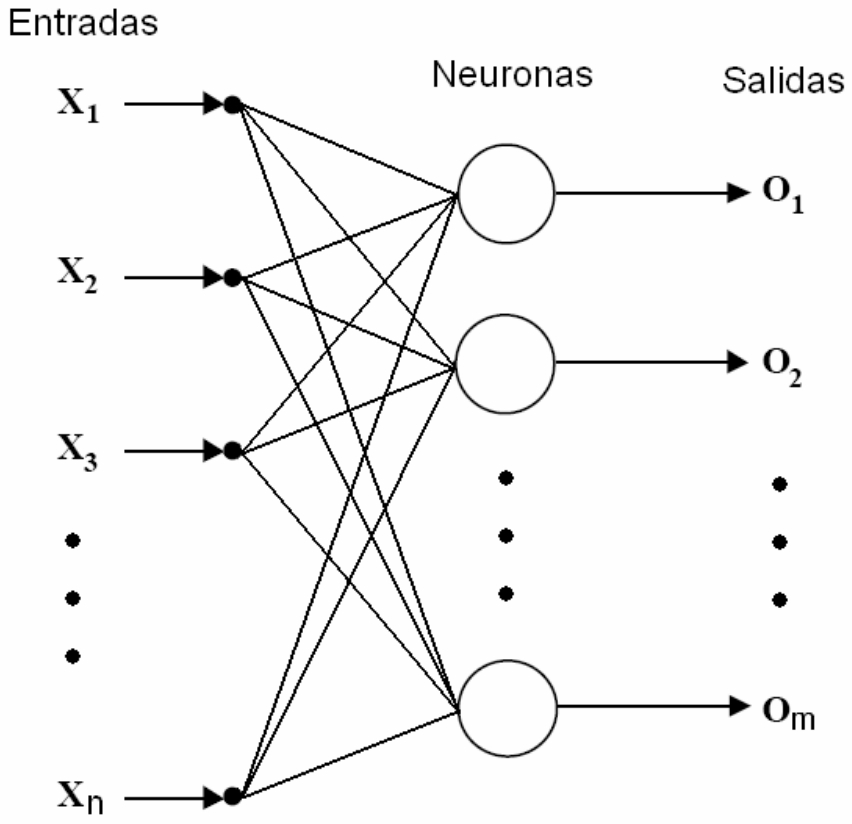
El perceptrón simple es un asociador de patrones; es capaz de asociar, durante el proceso de aprendizaje, patrones de entrada, representados por arreglos de valores numéricos continuos, con una salida binaria. Cuando a la red se le presenta un patrón de entrada utilizado en el aprendizaje, ésta responde con la salida asociada a este patrón de entrada. Si se le presenta una entrada diferente a las utilizadas durante el aprendizaje, la red responde con la salida asociada al patrón de entrada más parecido, siendo capaz de generalizar.

Al estar formado solamente por una capa, el perceptrón simple tiene una capacidad bastante limitada, ya que son necesarias mas capas para asociar y generalizar patrones de entrada complejos; a pesar de esto, el perceptrón es una red de gran importancia, pues en base a su estructura se han desarrollado otros modelos de redes neuronales.

3.1.1. Estructura de la red

La estructura del perceptrón simple se muestra en la figura 21; la red está constituida por una capa de salida de m neuronas, representadas mediante círculos y un vector de m entradas a la red, representadas por puntos. Todas las entradas X son aplicadas a cada una de las neuronas, éstas son ponderadas según su respectivo peso sináptico y luego son sumadas, la función de activación de la neurona, determina el valor de salida O_i correspondiente al resultado de ésta suma; en la red perceptrón esta función de activación es el limitador fuerte o el limitador fuerte simétrico.

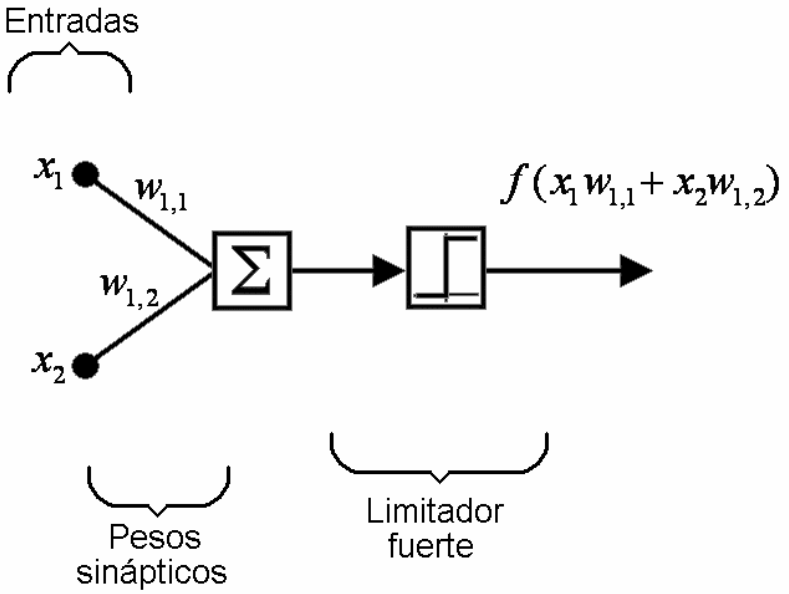
Figura 21. Estructura del perceptrón simple



3.1.2. Análisis del perceptrón simple

Para entender el comportamiento del perceptrón, se principia analizando una red constituida por una neurona con dos entradas, como la mostrada en la figura 22.

Figura 22. Perceptrón de una neurona con dos entradas



El bloque con el símbolo de sumatoria indica la suma de las entradas multiplicadas por sus pesos sinápticos y el último bloque representa la función de activación limitador fuerte. El valor de salida de la red está determinado por la siguiente función:

$$Salida = \begin{cases} 1 & si \quad (w_{1,1}x_1 + w_{1,2}x_2) \geq 0 \\ 0 & si \quad (w_{1,1}x_1 + w_{1,2}x_2) < 0 \end{cases} \qquad [3\text{-}1]$$

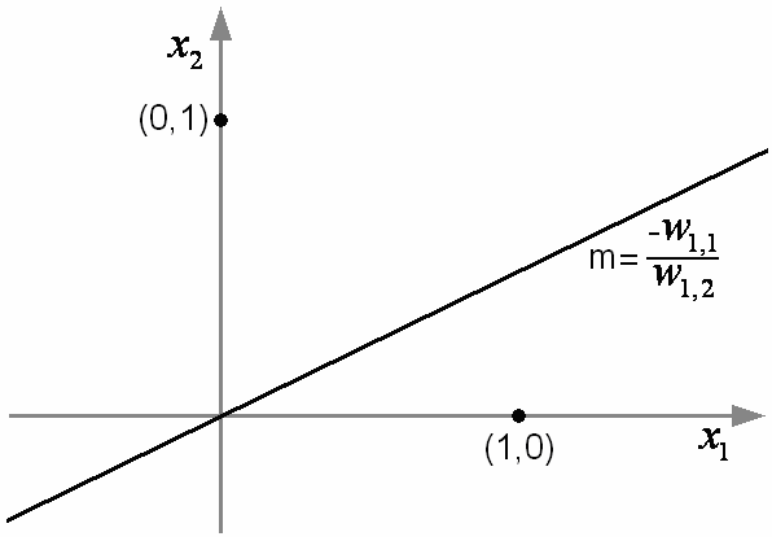
Donde x_1, x_2 son las entradas y $w_{1,1}$, $w_{1,2}$ son los pesos sinápticos correspondientes.

La ecuación del umbral de decisión es la siguiente:

$$w_{1,1}x_1 + w_{1,2}x_2 = 0 \qquad [3\text{-}2]$$

Al graficar esta ecuación en un plano coordenado donde los valores de x_1 se representan en el eje horizontal y los valores de x_2 en el eje vertical, se obtiene una línea recta con pendiente $m=-w_{1,1}/w_{1,2}$ que pasa por el origen como la mostrada en la figura 23.

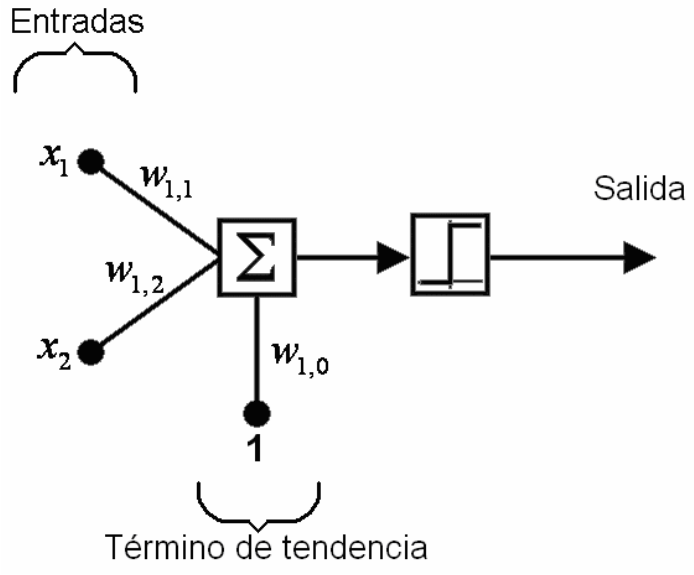
Figura 23. Gráfica x_2 vrs. x_1



En esta gráfica se observa que la línea divide al plano x_2-x_1 en dos regiones, los puntos (x_1, x_2) en la región sobre la línea darán una salida, mientras que los puntos bajo la línea darán otra salida; por ejemplo, los puntos (1,0) y (0,1), mostrados en la figura 23, darán una salida diferente. Los valores de los pesos sinápticos $w_{1,1}$ y $w_{1,2}$ se ajustan durante el aprendizaje para obtener la pendiente adecuada según las salidas deseadas.

En esta red se observa que los pesos sinápticos únicamente modificarán el valor de la pendiente de la línea; pero para poder separar los puntos (0,0.5) y (0,1) es necesario trasladar la línea con respecto al eje vertical; esto se soluciona agregando una entrada a la neurona de la red, llamada término de tendencia, que siempre tiene un valor de uno; esta entrada es multiplicada por un peso sináptico, al igual que las demás. En la figura 24 se muestra la red perceptrón con este término de tendencia.

Figura 24. Perceptrón simple de una neurona



Con la adición del término de tendencia el valor de salida es:

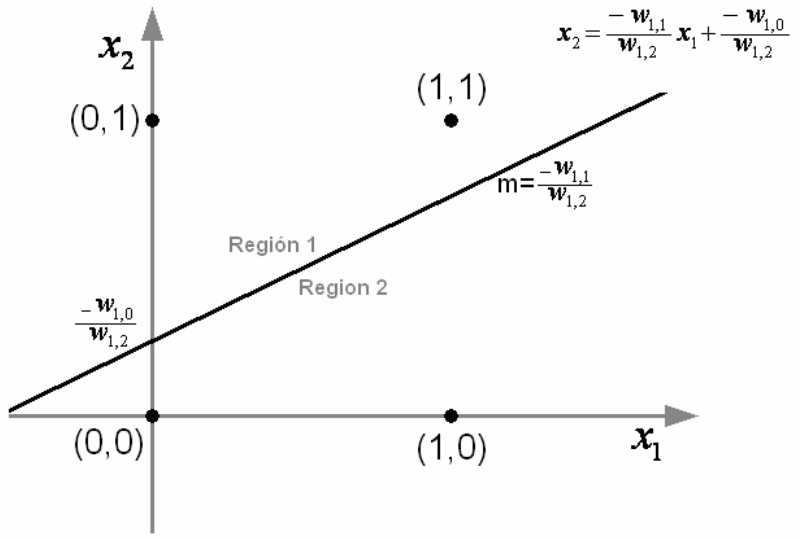
$$Salida = \begin{cases} 1 & si \quad (w_{1,1}x_1 + w_{1,2}x_2 + w_{1,0}) \geq 0 \\ 0 & si \quad (w_{1,1}x_1 + w_{1,2}x_2 + w_{1,0}) < 0 \end{cases} \qquad [3\text{-}3]$$

Siendo la ecuación de umbral de decisión la siguiente:

$$w_{1,1}x_1 + w_{1,2}x_2 + w_{1,0} = 0 \qquad [3\text{-}4]$$

Al graficar esta ecuación en el plano x_2 - x_1, se obtiene una línea recta que corta el eje vertical en el punto $-w_{1,0}/w_{1,2}$, con pendiente $m=-w_{1,1}/w_{1,2}$, como la mostrada en la figura 25.

Figura 25. Regiones de salida del perceptrón de una neurona

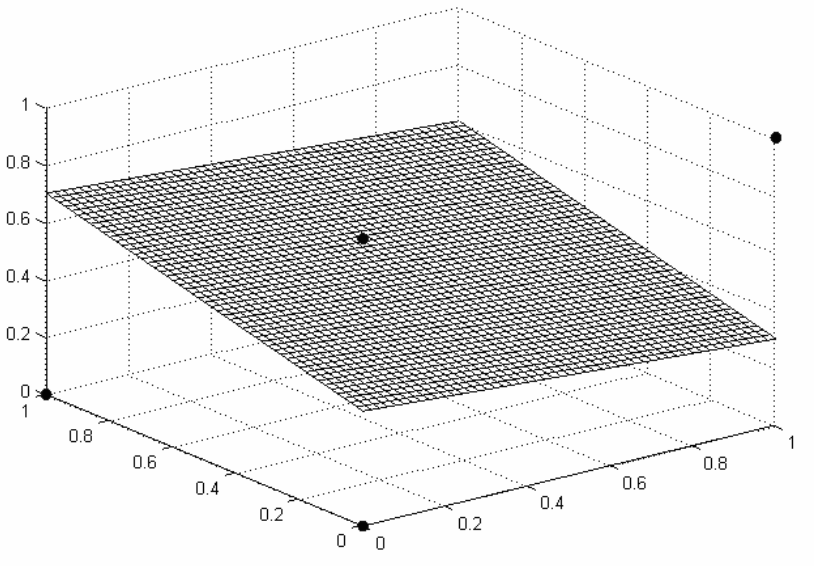


La línea parte el plano en dos regiones distintas; se pueden clasificar a los puntos de una región, como pertenecientes a la clase que posee una salida uno y a los de la otra región, pertenecientes a la clase que posee una salida cero.

En resumen, el comportamiento de una neurona de un perceptrón simple con dos entradas, puede visualizarse en un espacio de dos dimensiones, que representan estas entradas; para ello se grafica en este espacio bidimensional los puntos correspondientes a valores de entrada, luego se trata de trazar una línea recta que divida este espacio en dos regiones correspondientes a dos salidas distintas; esta línea recta corresponde a valores de pesos sinápticos que deberán obtenerse en el proceso de aprendizaje. El algoritmo de aprendizaje del perceptrón crea por sí solo la línea recta más adecuada, según las dos clases de puntos a reconocerse.

El análisis anterior puede aplicarse a un perceptrón simple con 3 entradas, graficando los puntos correspondientes a los valores de entrada, en un espacio de tres dimensiones; en este caso, el espacio tridimensional es dividido en dos regiones de decisión por medio de un plano (dos dimensiones) como se muestra en la figura 26.

Figura 26. Plano dividiendo un espacio tridimensional



El umbral de decisión de este perceptrón de tres entradas es el plano mostrado en la figura 26, que corresponde a la siguiente ecuación:

$$w_{1,1}x_1 + w_{1,2}x_2 + w_{1,3}x_3 + w_{1,0} = 0 \qquad [3\text{-}5]$$

Este análisis puede extenderse a un perceptrón simple con n entradas, colocando los puntos correspondientes a las entradas en un espacio de n dimensiones llamado hiperespacio; así como un plano de dos dimensiones divide un espacio de tres dimensiones, un plano de n-1 dimensiones, llamado hiperplano, divide en dos regiones un espacio de n dimensiones.

En general, el comportamiento de la k-ésima neurona de un perceptrón simple de n entradas, puede determinarse dividiendo un hiperespacio en dos regiones de decisión, mediante un hiperplano correspondiente a la siguiente ecuación:

$$\sum_{i=1}^{n} w_{k,i} x_i + w_{k,0} = 0 \qquad [3\text{-}6]$$

Donde x_i es el valor de la i-ésima entrada, $w_{k,i}$ es el peso sináptico correspondiente a la i-ésima entrada de la k-ésima neurona y $w_{k,0}$ es el peso sináptico del término de tendencia de la k-ésima neurona.

Todos los puntos dentro de una región limitada por el hiperplano tienen un mismo valor de salida. Durante el proceso de aprendizaje se ajustan los pesos sinápticos, para colocar el hiperplano en la posición que divida adecuadamente los puntos del hiperespacio, según los ejemplos presentados a la red.

3.1.3. Regla de aprendizaje

El Perceptrón es un tipo de red de aprendizaje supervisado; necesita conocer los valores esperados, para cada una de las entradas presentadas como ejemplo; su comportamiento está definido según los ejemplos presentados a la red.

En el aprendizaje o entrenamiento, el perceptrón se expone a un conjunto de pares entrada-salida y los pesos de la red son ajustados, de forma que al final del entrenamiento se obtengan las salidas esperadas para cada una de estas entradas de ejemplo.

El algoritmo de entrenamiento del Perceptrón se describe a continuación:

1. Se inicializan los valores de los pesos sinápticos $w_{k,i}$; por lo general se asignan valores aleatorios.

2. Se presenta el primer ejemplo de entrada a la red ($x_1,\ldots x_n$), junto con su salida esperada y.

3. Se realizan los pasos 4 al 6 para cada neurona k del perceptrón.

4. Se calcula la salida de la red, para la neurona k, por medio de la siguiente ecuación:

$$Salida_k = \begin{cases} 1 \quad si \quad \sum_{i=1}^{n} w_{k,i}x_i + w_{k,0} \geq 0 \\ 0 \quad si \quad \sum_{i=1}^{n} w_{k,i}x_i + w_{k,0} < 0 \end{cases} \qquad [3\text{-}7]$$

5. Se determina el error, comparando el valor de salida con el valor esperado de la siguiente forma:

$$Error = y_k - Salida_k \qquad [3\text{-}8]$$

6. Los pesos sinápticos se corrigen según el valor de error y el ejemplo de entrada, de la siguiente forma:

$$w_{k,i(nuevo)} = w_{k,i(anterior)} + Error \cdot x_i \quad \text{Para todas las i, de 1 a n} \qquad [3\text{-}9]$$

7. Se presenta el siguiente ejemplo de entrada a la red ($x_1,\ldots x_n$), junto con su salida esperada **y**, luego se vuelve al paso 3. Si no existen mas ejemplos se regresa al primer ejemplo, hasta que los errores de las neuronas, para todos los ejemplos, sean cero. El entrenamiento finaliza cuando se obtienen las salidas adecuadas para todos los ejemplos de entrada.

Para asegurar llegar a una solución en el entrenamiento del perceptrón, se puede hacer una modificación al paso 6, agregando el término μ llamado velocidad de aprendizaje; así los pesos sinápticos se corrigen de acuerdo a la siguiente expresión:

$$w_{k,i(nuevo)} = w_{k,i(anterior)} + \mu \cdot Error \cdot x_i \quad \text{Para todas las i, de 1 a n} \quad [3\text{-}10]$$

El término de velocidad de aprendizaje puede tener valores entre cero y uno ($0<\mu<1$); un valor pequeño de μ implica que se tendrá que hacer un gran número de iteraciones, pero este es el costo para asegurar que el entrenamiento llegue a asentarse en una solución.

Durante el aprendizaje se irán variando los valores de los pesos sinápticos, obteniendo distintos hiperplanos en el hiperespacio correspondiente a las entradas. El aprendizaje pretende encontrar un hiperplano que divida el hiperespacio en dos regiones de decisión adecuadas para separar las dos clases de valores de entrada.

Para ejemplificar el funcionamiento del perceptrón y su regla de aprendizaje, se usará un perceptrón simple de una neurona con función de activación limitador fuerte, para realizar la función lógica AND; para esto la red deberá ser capaz de proporcionar la salida adecuada, a partir de dos entradas binarias.

La tabla I muestra las cuatro combinaciones que se forman con dos entradas digitales y las salidas correspondientes a la función lógica AND; estos cuatro pares de entradas salidas, serán usadas como ejemplos en el entrenamiento de la red.

Tabla I. Función lógica AND

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

Utilizando la tabla I, se realiza el entrenamiento del perceptrón de la siguiente manera:

- Los valores iniciales asignados aleatoriamente a los pesos sinápticos son:
  w1 = 1.6 w2 = 2.5 w0 = 4.3

Iteración 1

- El primer ejemplo es x_1=0 x_2=0 (0 0), con una entrada total de:
  1.6x0 + 2.5x0 + 4.3 = 4.3 que corresponde a una salida 1 en la función limitador fuerte.
  Se esperaba una salida 0 y se obtuvo 1, por lo que el error es 0 - 1 = -1
  Los pesos sinápticos se actualizan de la siguiente manera:
  w_1 = 1.6 - 1x0 = 1.6
  w_2 = 2.5 - 1x0 = 2.5
  w_0 = 4.3 - 1x1 = 3.3

- El segundo ejemplo es x_1=0 x_2=1 (0 1), con una entrada total de:
  1.6x0 + 2.5x1 + 3.3 = 5.8 que corresponde a una salida 1 en la función limitador fuerte.
  Se esperaba una salida 0 y se obtuvo 1, por lo que el error es 0 - 1 = -1
  Los pesos sinápticos se actualizan de la siguiente manera:
  w_1 = 1.6 - 1x0 = 1.6
  w_2 = 2.5 - 1x1 = 1.5
  w_0 = 3.3 - 1x1 = 2.3
- El tercer ejemplo es x_1=1 x_2=0 (1 0), con una entrada total de:
  1.6x1 + 1.5x0 + 2.3 = 3.9 que corresponde a una salida 1
  Se esperaba una salida 0 y se obtuvo 1, por lo que el error es 0 - 1 = -1
  Los pesos sinápticos se actualizan de la siguiente manera:
  w_1 = 1.6 - 1x1 = 0.6
  w_2 = 1.5 - 1x0 = 1.5
  w_0 = 2.3 - 1x1 = 1.3
- El cuarto ejemplo es x_1=1 x_2=1 (1 1), con una entrada total de:
  0.6x1 + 1.5x1 + 1.3 = 3.4 que corresponde a una salida 1
  Se obtuvo la salida que se esperaba, por lo que el error es 0 y no se cambian los pesos sinápticos.

Iteración 2 Se presentan nuevamente los cuatro ejemplos

- El primer ejemplo es x_1=0 x_2=0 (0 0), con una entrada total de:
  0.6x0 + 1.5x0 + 1.3 = 1.3 que corresponde a una salida 1
  Se esperaba una salida 0 y se obtuvo 1, por lo que el error es 0 - 1 = -1
  Los pesos sinápticos se actualizan de la siguiente manera:
  w_1 = 0.6 - 1x0 = 0.6
  w_2 = 1.5 - 1x0 = 1.5
  w_0 = 1.3 - 1x1 = 0.3

- El segundo ejemplo es x_1=0 x_2=1 (0 1), con una entrada total de:
  0.6x0 + 1.5x1 + 0.3 = 1.8 que corresponde a una salida 1
  Se esperaba una salida 0 y se obtuvo 1, por lo que el error es 0 - 1 = -1
  Los pesos sinápticos se actualizan de la siguiente manera:
  w_1 = 0.6 - 1x0 = 0.6
  w_2 = 1.5 - 1x1 = 0.5
  w_0 = 0.3 - 1x1 = -0.7
- El tercer ejemplo es x_1=1 x_2=0 (1 0), con una entrada total de:
  0.6x1 + 0.5x0 - 0.7 = -0.1 que corresponde a una salida 0
  Se obtuvo la salida que se esperaba, por lo que el error es 0 y no se cambian los pesos sinápticos.
- El cuarto ejemplo es x_1=1 x_2=1 (1 1), con una entrada total de:
  0.6x1 + 0.5x1 - 0.7 = 0.4 que corresponde a una salida 1
  Se obtuvo la salida que se esperaba, por lo que el error es 0 y no se cambian los pesos sinápticos.

Iteración 3

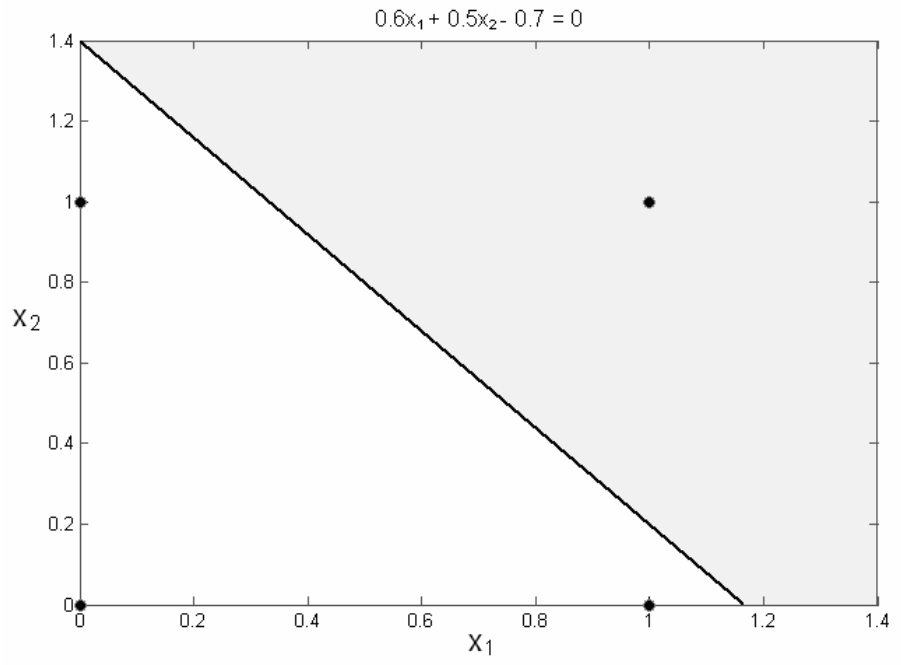
- El primer ejemplo es x_1=0 x_2=0 (0 0), con una entrada total de:
  0.6x0 + 0.5x0 - 0.7 = -0.7 que corresponde a una salida 0
  Se obtuvo la salida que se esperaba, por lo que el error es 0 y no se cambian los pesos sinápticos.
- El segundo ejemplo es x_1=0 x_2=1 (0 1), con una entrada total de:
  0.6x0 + 0.5x1 - 0.7 = -0.2 que corresponde a una salida 0
  Se obtuvo la salida que se esperaba, por lo que el error es 0 y no se cambian los pesos sinápticos.
- El tercer ejemplo es x_1=1 x_2=0 (1 0), con una entrada total de:
  0.6x1 + 0.5x0 - 0.7 = -0.1 que corresponde a una salida 0
  Se obtuvo la salida que se esperaba, por lo que el error es 0 y no se cambian los pesos sinápticos.

- El cuarto ejemplo es x_1=1 x_2=1 (1 1), con una entrada total de:
  0.6x1 + 0.5x1 - 0.7 = 0.4 que corresponde a una salida 1
  Se obtuvo la salida que se esperaba, por lo que el error es 0 y no se cambian los pesos sinápticos.

- Ya que en la última iteración se obtuvieron las salidas correctas para todos los ejemplos, se finaliza el entrenamiento.

El comportamiento de esta red puede visualizarse en la figura 27, donde se presenta la gráfica de la línea de umbral de decisión, correspondiente a estos pesos sinápticos. En esta gráfica se muestran los puntos (0,0), (0,1), (1,0) y (1,1), que corresponden a los ejemplos usados en el entrenamiento de la red; puede verse que la línea separa al plano en dos regiones de acuerdo al valor de salida de los puntos.

Figura 27. Comportamiento del perceptrón del ejemplo



3.1.4. Limitaciones de la red

Cuando se analiza un perceptrón de dos entradas mediante una gráfica, puede observase que la única forma de dividir el plano en las dos regiones de decisión, es por medio de una línea recta; de la misma forma, en el caso de que existan muchas entradas, un hiperespacio solo puede dividirse en dos regiones por medio de un hiperplano. A los puntos en un hiperespacio que pueden separarse en dos clases por medio de un hiperplano, se les conoce como linealmente separables.

Minsky y Papert, en su libro publicado en 1969, presentaron un análisis de las restricciones que existen para los problemas que una red tipo perceptrón puede resolver; la mayor desventaja de esta red es que no es capaz de distinguir dos clases que no son linealmente separables.

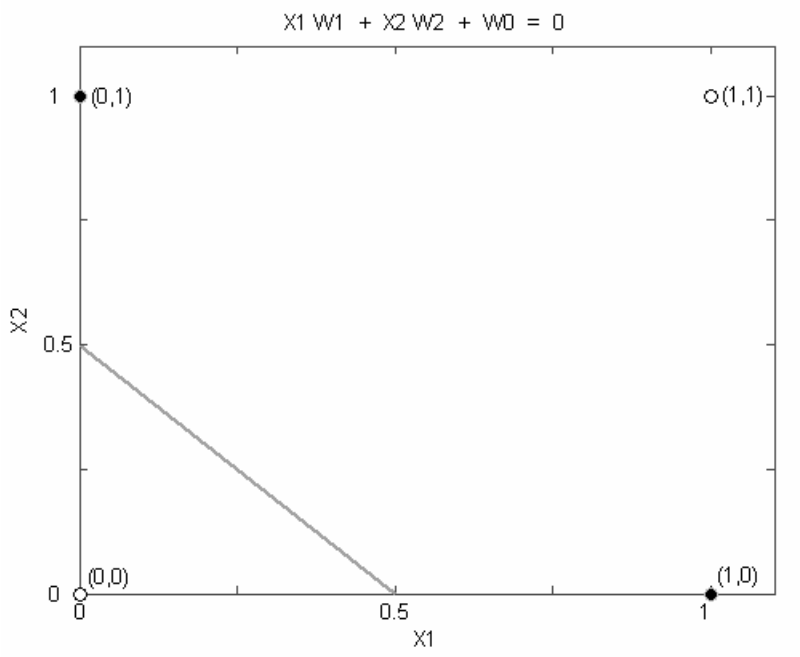
Un ejemplo de dos clases no linealmente separables es la función lógica XOR (OR exclusiva), la cual se muestra en la tabla II.

Tabla II. Función lógica XOR

x_1	x_2	y x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

En la figura 28 se muestran los 4 puntos de la tabla II, graficados en el plano x_2-x_1; se pretende separar los puntos (0,0), (1,1) en una región y los puntos (0,1), (1,0) en la otra región.

Figura 28. Puntos de la función XOR



En la figura 28 se observa que no hay forma de posicionar la línea recta para separar los puntos (0,1) y (1,0) de los demás, por lo que no es posible entrenar el perceptrón simple para efectuar correctamente la función XOR. Debido a esta limitación del perceptrón, el estudio de las redes neuronales artificiales se estanco durante casi 20 años.

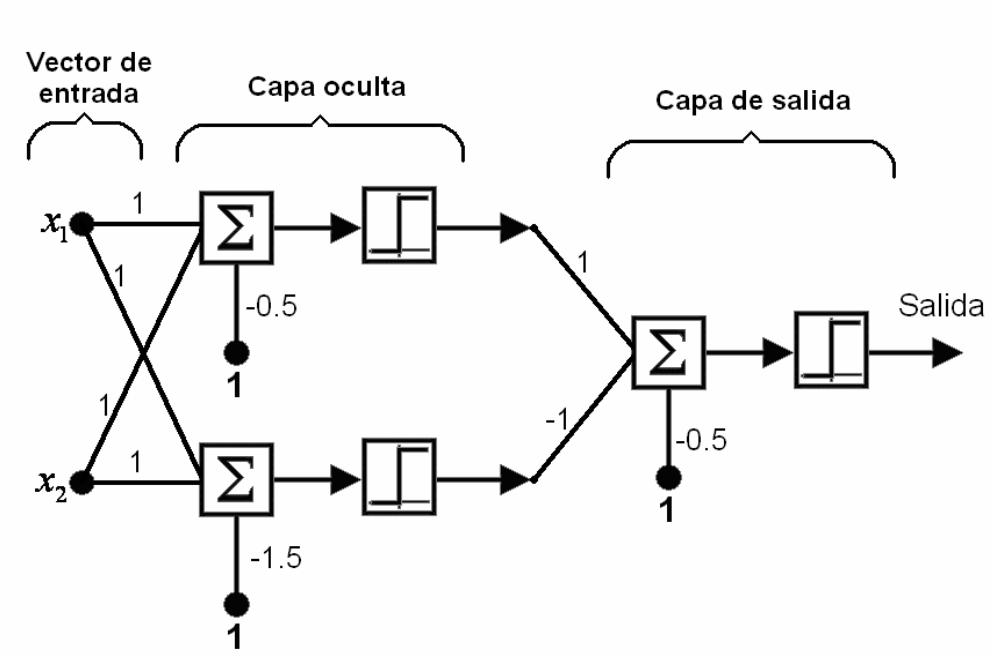
Para saber si una red perceptrón simple puede aplicarse a un problema de interés, se debe comprobar que éste sea linealmente separable. El proceso para determinar si dos clases son linealmente separable o no, se realiza gráficamente sin problema, cuando se tienen dos entradas, ya que el análisis se realiza en un espacio de dos dimensiones, como en el caso de las funciones AND, OR o XOR; sin embargo, esta visualización se dificulta cuando el análisis es de tres dimensiones y resulta imposible de observar gráficamente cuando los patrones de entrada son de dimensiones superiores.

En el caso de que se tengan un número de entradas mayor que tres, se debe plantear condiciones de desigualdad que permitan comprobar la separabilidad lineal de los valores de entrada, esto se realiza con base en la ecuación de salida del Perceptrón, esta es:

$$Salida = \begin{cases} 1 \quad si \quad \sum_{i=1}^{n} w_i x_i + w_0 \geq 0 \\ 0 \quad si \quad \sum_{i=1}^{n} w_i x_i + w_{k,0} < 0 \end{cases} \qquad [3\text{-}11]$$

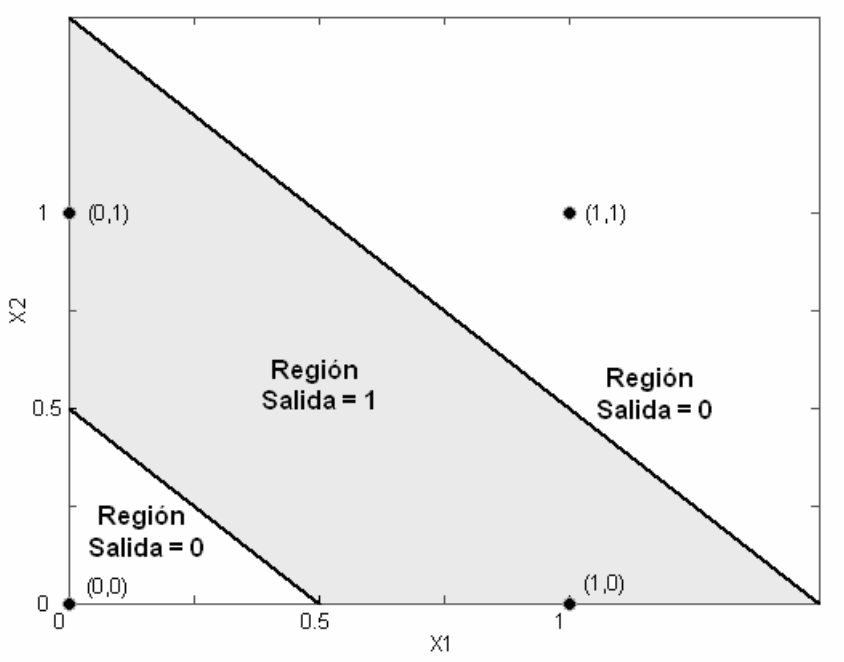
El problema de la función XOR puede solucionarse usando una red neuronal de dos capas (vector de entrada, capa oculta y capa de salida) con dos neuronas en la capa oculta y una neurona en la capa de salida, como la que se muestra en la figura 29.

Figura 29. Red neuronal que realiza la función XOR



Mediante el uso de dos neuronas en la capa oculta se dividió el plano x_2-x_1 en tres regiones de decisión; esto permite clasificar correctamente las entradas, como se muestra en la figura 30.

Figura 30. Regiones de decisión de la función XOR



Cada neurona de la capa oculta, separa el plano en dos regiones, delimitadas por las líneas que se muestran en la figura 30; así, entre ambas neuronas se diferencian tres regiones en el plano; luego, mediante la neurona de la capa de salida, se clasifican estas regiones para obtener un valor de salida uno, para la región central y un valor de salida cero, para las regiones exteriores.

Con el uso de dos líneas rectas, que representan dos neuronas de capa oculta, se puede dividir un plano en tres o cuatro regiones de decisión; de la misma forma, cuando se tienen muchas entradas, con dos hiperplanos se puede dividir un hiperespacio en tres o cuatro regiones.

Si se colocaran mas neuronas en la capa intermedia podría dividirse el hiperespacio en muchas regiones de decisión, esto muestra que agregando capas ocultas, se pueden distinguir clases que no son linealmente separables, lo que permite utilizar una red neuronal en diferentes aplicaciones de naturaleza compleja.

El perceptrón simple por estar formado por una capa, solamente es capaz de distinguir clases que sean linealmente separables, pero redes neuronales con capas ocultas, pueden distinguir entre diferentes clases de naturaleza compleja; esto muestra la importancia de las capas ocultas en las redes neuronales y nos proporciona una idea de la capacidad de las redes multicapa.

Por sus limitadas capacidades, el perceptrón simple es una red de poca aplicación, pero se considera de gran importancia, pues en base a su estructura y funcionamiento, se desarrollaron nuevos modelos de redes neuronales, como es el caso del perceptrón multicapa.

3.2. Perceptrón multicapa

Como se explicó anteriormente, las redes neuronales monocapa pueden asociar patrones de entrada con las salidas deseadas y generalizar; pero tienen la desventaja de solamente poder clasificar entradas linealmente separables; fue esta limitación la que dio surgimiento a las redes multicapa.

Aunque las redes neuronales multicapa eran capaces de clasificar entradas complejas, el algoritmo de aprendizaje del perceptrón, fue desarrollado únicamente para redes monocapa, siendo necesario un algoritmo de aprendizaje, para que las redes multicapa pudieran ser funcionales.

El primer algoritmo de aprendizaje para redes multicapa fue desarrollado por Paul Werbos en 1974, éste se desarrolló en un contexto general, para cualquier tipo de redes, siendo las redes neuronales una aplicación especial, razón por la cual el algoritmo no fue aceptado por los desarrolladores de redes neuronales artificiales.

Fue hasta mediados de los años 80, cuando el algoritmo de retropropagación (conocido también como delta generalizada o propagación hacia atrás) fue redescubierto al mismo tiempo por David Rumelhart, Geoffrey Hinton, Ronal Williams, David Parker y Yann Le Cun. El algoritmo se popularizó en 1986, cuando fue incluido en el libro *Parallel Distributed Processing Group* (Grupo de Procesamiento distribuido en Paralelo) por los psicólogos David Rumelhart y James McClelland. La publicación de este libro produjo un crecimiento en las investigaciones de redes neuronales, siendo el perceptrón multicapa una de las redes que más se utilizó.

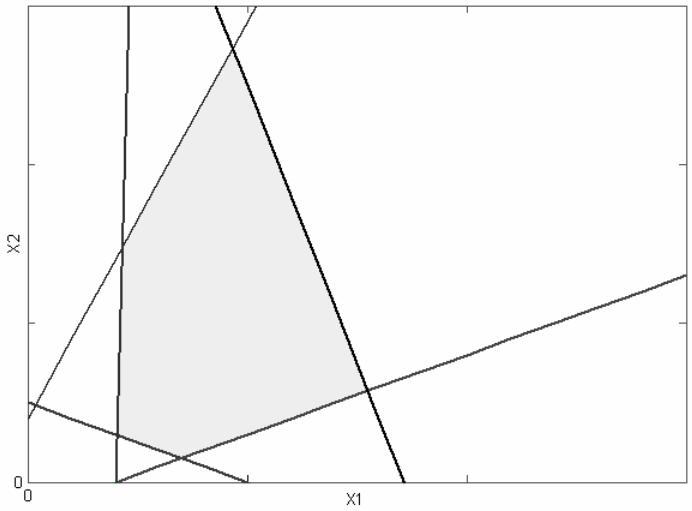
La red neuronal perceptrón multicapa es una red de propagación hacia adelante, compuesta por dos o más capas de neuronas con función de activación sigmoidal logarítmica, sigmoidal tangente hiperbólica o lineal.

La red perceptrón multicapa es entrenada de forma supervisada mediante el algoritmo de retropropagación, esta es la razón por la que también se conozca como red de retropropagación o red de propagación hacia atrás. Al hablar de red de retropropagación o red de propagación hacia atrás se hace referencia al algoritmo de aprendizaje y no a la propagación de las señales durante el funcionamiento de la red. La retropropagación consiste en propagar el error hacia atrás durante el entrenamiento, es decir, de la capa de salida hacia la capa de entrada, pasando por las capas ocultas.

Una de las grandes ventajas del perceptrón multicapa es que esta red aprovecha la naturaleza paralela de las redes neuronales para reducir el tiempo requerido por un procesador secuencial, para determinar la salida adecuada a partir de una entrada. Además el tiempo de desarrollo se reduce como consecuencia de que la red puede aprender el algoritmo correcto sin que alguien tenga que deducir por anticipado el algoritmo en cuestión.

Una red neuronal de dos capas, puede formar cualquier región convexa en un hiperespacio, mediante la intersección de las regiones formadas por cada neurona de la capa oculta. El resultado es una región de decisión de forma convexa con un número de lados igual o menor al número de neuronas de la segunda capa, como se muestra en la figura 31.

Figura 31. Región formada por 5 neuronas de capa oculta

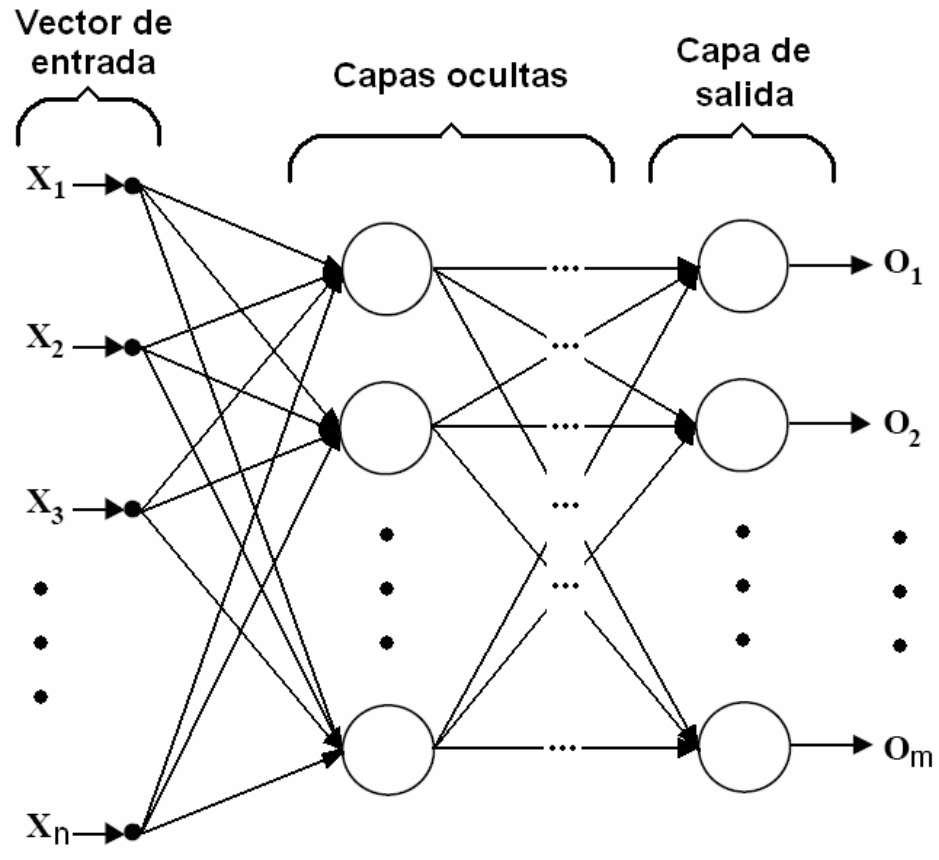


El número de neuronas de las capas ocultas debe ser lo suficientemente grande como para que se forme una región compleja que pueda resolver el problema, sin embargo no debe ser muy grande, pues las regiones de decisión se limitarían solo a los ejemplos de entrenamiento.

3.2.1. Estructura de la red

El perceptrón multicapa está formado por el vector de entrada, una o más capas ocultas y la capa de salida; las capas ocultas pueden tener cualquier número de neuronas, mientras que el tamaño del vector de entrada y el número de neuronas de la capa de salida se seleccionan de acuerdo a las entradas y salidas de la red respectivamente. Todas las neuronas de cada capa deben tener la misma función de activación, pudiendo ser esta función, diferente a la de otras capas. Las funciones de activación usadas por el perceptrón multicapa son la función sigmoidal logarítmica, sigmoidal tangente hiperbólica o lineal. En la figura 32 se muestra la estructura del perceptrón multicapa.

Figura 32. Estructura del perceptrón multicapa



3.2.2. Reglas de aprendizaje

El perceptrón multicapa es una red de aprendizaje supervisado; por tanto, para su entrenamiento necesita un conjunto de pares entrada-salida como ejemplos. Los pesos sinápticos son ajustados de forma que al final del entrenamiento se obtengan las salidas esperadas para cada una de estas entradas de ejemplo.

A medida que se entrena la red, las neuronas de las capas ocultas se organizan a sí mismas, de tal modo que éstas aprenden a reconocer distintas características de los ejemplos. Después del entrenamiento, cuando se les presenta una entrada que contiene ruido o está incompleta, las neuronas de las capas ocultas responden con una salida activa, si la entrada posee las características que estas neuronas aprendieron a reconocer durante el entrenamiento.

Varios investigadores han demostrado que durante el proceso de entrenamiento, el perceptrón multicapa tiende a desarrollar relaciones internas entre neuronas, con el fin de organizar los datos de entrenamiento en diferentes clases. Esta tendencia indica que todas las unidades de la capa oculta son asociadas, de alguna manera, a características específicas de los ejemplos de entrada, como consecuencia del entrenamiento. Aunque esta asociación puede no resultar evidente, lo importante es que la red ha encontrado una representación interna que le permite generar las salidas deseadas. Esta misma representación interna se puede aplicar a entradas que no fueron utilizadas durante el entrenamiento; así la red clasificará estas entradas según las características que compartan con los ejemplos de entrenamiento.

El aprendizaje de una red neuronal consiste en hallar el conjunto adecuado de pesos sinápticos, para que la red pueda resolver correctamente un problema, para esto también es necesario que el conjunto de ejemplos de entrenamiento sea suficiente, para que la red pueda encontrar las características relevantes de estos ejemplos y así poder generalizarlos.

El primer algoritmo de aprendizaje del perceptrón multicapa fue la regla delta generaliza; en base a éste, se han desarrollado otros algoritmos para mejorar o acelerar el entrenamiento de esta red.

3.2.2.1. Regla delta generalizada

Este algoritmo se basa en calcular el gradiente negativo o gradiente descendiente del error cuadrático medio en la salida, para obtener el error en los pesos sinápticos. El error se calcula primero para la capa de salida, en base a éste se calcula el error de la siguiente capa, continuando así hasta llegar a la entrada; por último se actualizan los pesos sinápticos de cada capa de acuerdo a sus valores de error. A diferencia de otros algoritmos de entrenamiento, la regla delta generalizada no necesita de hacer una buena aproximación de los pesos sinápticos iniciales.

La regla delta generalizada también se conoce como algoritmo de retropropagación o propagación hacia atrás (*Backpropagation*), debido a que el error se propaga de manera inversa al funcionamiento normal de la red; también se le llama método del gradiente descendiente, por la forma en que se obtiene el error. A continuación se deduce el error en los pesos sinápticos de las neuronas que forman la capa de salida de un perceptrón multicapa.

La entrada total de la k-ésima neurona de salida para el p-ésimo ejemplo está dado por la siguiente ecuación:

$$entrada_{pk}^{o} = \sum_{j=1}^{L} w_{kj}^{o} i_{pj} + w_{k0}^{o} \qquad [3\text{-}12]$$

Donde L es el número de entradas i de la neurona, ponderadas por sus pesos sinápticos w. La salida de esta neurona se calcula de acuerdo con su función de activación y su entrada total, ésta es:

$$o_{pk} = f^{o}(entrada_{pk}^{o}) \qquad [3\text{-}13]$$

Así, el error en la k-ésima salida obtenido en el p-ésimo ejemplo es:

$$Error_{pk} = (y_{pk} - o_{pk}) \qquad [3\text{-}12]$$

Donde y_{pk} es la k-ésima salida deseada para el p-ésimo ejemplo y o_{pk} es la salida obtenida con los pesos sinápticos actuales.

El error cuadrático medio en la salida de la red, para el p-ésimo ejemplo de entrada es el siguiente:

$$E_p = \frac{1}{2}\sum_{k=1}^{M} Error_{pk}^{\ 2} = \frac{1}{2}\sum_{k=1}^{M} (y_{pk} - o_{pk})^2 \qquad [3\text{-}13]$$

Donde M es el número total de salidas que posee la red.

El error en los pesos sinápticos es determinado por el gradiente negativo del error cuadrático medio a la salida de la red, este gradiente está dado por la siguiente ecuación:

$$\frac{\partial}{\partial w_{kj}^{o}}\left(E_p\right) = -(y_{pk} - o_{pk}) \cdot \frac{\partial}{\partial entrada_{pk}^{o}}(f^{o}) \cdot \frac{\partial}{\partial w_{kj}^{o}}(entrada_{pk}^{o}) \quad [3\text{-}14]$$

Dado que

$$\frac{\partial}{\partial w_{kj}^{o}}(entrada_{pk}^{o}) = \frac{\partial}{\partial w_{kj}^{o}}(\sum_{j=1}^{L} w_{kj}^{o} i_{pj} + w_{k0}^{o}) = i_{pj} \quad [3\text{-}15]$$

El gradiente negativo puede expresarse de la siguiente forma:

$$-\frac{\partial}{\partial w_{kj}^{o}}\left(E_p\right) = (y_{pk} - o_{pk}) \cdot f^{o}{}'(entrada_{pk}^{o}) \cdot i_{pj} \quad [3\text{-}16]$$

Así, el error en los pesos sinápticos de la capa de salida es:

$$\Delta_p w_{kj}^{o} = \eta \cdot (y_{pk} - o_{pk}) \cdot f^{o}{}'(entrada_{pk}^{o}) \cdot i_{pj} \quad [3\text{-}17]$$

El factor η se denomina parámetro de velocidad de aprendizaje y puede tener un valor entre cero y uno ($0<\eta<1$); η debe tener un valor pequeño para asegurar que la red llegue a asentarse en una solución, aunque esto significa que la red tendrá que hacer un gran número de iteraciones; si η tiene un valor alto, la red podría alejarse de la solución. Es posible aumentar el valor de η a medida que disminuye el error de la red para reducir el número de iteraciones.

Para resumir las ecuaciones, el término de error de la k-ésima neurona de salida en el p-ésimo ejemplo se define como:

$$\delta^{o}_{pk} = (y_{pk} - o_{pk}) \cdot f^{o}{}'(entrada^{o}_{pk}) \qquad [3\text{-}18]$$

Así, los pesos sinápticos de la capa de salida son modificados de la siguiente forma:

$$w^{o}_{kj\ nuevo} = w^{o}_{kj\ actual} + \eta \cdot \delta^{o}_{pk} \cdot i_{pj} \qquad [3\text{-}19]$$

Para obtener el error en los pesos sinápticos de la capa de salida, se calcula el gradiente del error cuadrático medio de salida, respecto a los pesos de la capa de salida. De la misma forma, para calcular el error en los pesos sinápticos de las capas ocultas, se calcula el gradiente negativo del error cuadrático medio de salida respecto a los pesos de las capas ocultas.

Para la capa oculta más cercana a la salida, este gradiente es:

$$-\frac{\partial}{\partial w^{h}_{ji}}\left(E_p\right) = f^{h}{}'(entrada^{h}_{pj}) \cdot x_{pi} \cdot \sum_{k=1}^{M} (y_{pk} - o_{pk}) \cdot f^{o}{}'(entrada^{o}_{pk}) \cdot w^{o}_{kj}$$

[3-20]

Donde $f^{h}( )$ es la función de activación de esta capa oculta, $entrada_{pj}$ es la entrada total a la j-ésima neurona de la capa oculta en el p-ésimo ejemplo, x_{pi} es la i-ésima entrada a la capa oculta en el p-ésimo ejemplo, w_{kj} es el j-ésimo peso sináptico de la k-ésima neurona de salida.

Así, el término de error de la j-ésima neurona de esta capa oculta en el p-ésimo ejemplo es:

$$\delta_{pj}^{h} = f^{h}{}'(entrada_{pj}^{h}) \cdot \sum_{k=1}^{M} \delta_{pk}^{o} \cdot w_{kj}^{o} \qquad [3\text{-}21]$$

Y los pesos sinápticos de esta capa oculta son modificados de la siguiente forma:

$$w_{ji_{nuevo}}^{h} = w_{ji_{actual}}^{h} + \eta \cdot \delta_{pj}^{h} \cdot x_{pi} \qquad [3\text{-}22]$$

Se observa que los términos de error de la capa oculta dependen de todos los términos de error de la capa de salida, por tanto, los errores se propagan hacia atrás durante el entrenamiento.

A continuación se describe el algoritmo de propagación hacia atrás o regla delta generalizada, para una red de dos capas con vector de entrada de tamaño N, L neuronas en la capa oculta y M neuronas en la capa de salida.

1. Se inicializan los valores de los pesos sinápticos de toda la red; por lo general se asignan valores aleatorios.

2. Se presenta el primer ejemplo de entrada a la red (xp = [xp1,…, xpN]) junto con su salida esperada (yp = [yp1,…, ypM]).

3. Se calcula la entrada total de cada neurona de la capa oculta.

$$entrada_{pj}^{h} = \sum_{i=1}^{N} w_{ji}^{h} \cdot x_{pi} + w_{j0}^{h} \qquad [3\text{-}23]$$

4. Se calculan las salidas de la capa oculta.

$$i_{pj} = f^h(entrada^h_{pj}) \qquad [3\text{-}24]$$

5. Se calcula la entrada total de cada neurona de la capa de salida.

$$entrada^o_{pk} = \sum_{j=1}^{L} w^o_{kj} i_{pj} + w^o_{k0} \qquad [3\text{-}25]$$

6. Se calculan las salidas de la red

$$o_{pk} = f^o(entrada^o_{pk}) \qquad [3\text{-}26]$$

7. Se calculan los términos de error para las neuronas de la capa de salida.

$$\delta^o_{pk} = (y_{pk} - o_{pk}) \cdot f^o{}'(entrada^o_{pk}) \qquad [3\text{-}27]$$

8. Se calculan los términos de error para las neuronas de la capa oculta.

$$\delta^h_{pj} = f^h{}'(entrada^h_{pj}) \cdot \sum_{k=1}^{M} \delta^o_{pk} \cdot w^o_{kj} \qquad [3\text{-}28]$$

9. Se actualizan los pesos sinápticos de las neuronas de la capa de salida.

$$w^o_{kj\ nuevo} = w^o_{kj\ actual} + \eta \cdot \delta^o_{pk} \cdot i_{pj} \qquad [3\text{-}29]$$

10. Se actualizan los pesos sinápticos de la capa oculta.

$$w^{h}_{ji\,nuevo} = w^{h}_{ji\,actual} + \eta \cdot \delta^{h}_{pj} \cdot x_{pi} \qquad [3\text{-}30]$$

11. Se presenta el siguiente ejemplo de entrada a la red, junto con su salida esperada y se regresa al paso 3. Si no existen mas ejemplos se calcula el error cuadrático medio de todos los ejemplos, como se muestra en la siguiente ecuación.

$$Error = \sum_{p} \frac{1}{2} \sum_{k=1}^{M} (y_{pk} - o_{pk})^2 \qquad [3\text{-}31]$$

El entrenamiento finaliza cuando el error cuadrático medio de todos los ejemplos, resulta aceptable para el funcionamiento de la red.

3.2.2.2. Método del gradiente descendiente con velocidad de aprendizaje variable

Este algoritmo es una variación de la regla delta generalizada, para disminuir el número iteraciones sin alejarse de la solución. En este caso se varia el factor de velocidad de aprendizaje η, con forme cambia el error durante las iteraciones. Si el error calculado en una iteración es menor que el error en la iteración anterior, se aumenta en un factor determinado el valor de la velocidad de aprendizaje; en el caso de que el error calculado sea igual o mayor al error en la iteración anterior la velocidad de aprendizaje disminuye y se vuelve a iterar.

Este algoritmo tiene más pasos por iteración que la regla delta generalizada, ya que también debe comparar los valores de error para variar el factor de velocidad de aprendizaje.

3.2.2.3. Método del gradiente descendiente con momento

Este algoritmo también es una variación de la regla delta generalizada; este método agrega un término de momento a la actualización de pesos sinápticos, para acelerar la convergencia. El nuevo término toma en cuenta el cambio que sufrió el error en las últimas dos iteraciones. El algoritmo es el mismo que la regla delta generalizada, pero la actualización de pesos es diferente.

La actualización de los pesos sinápticos de la capa de salida se realiza de la siguiente forma:

$$w^{o}_{kj\,nuevo} = w^{o}_{kj\,actual} + \eta \cdot \delta^{o}_{pk} \cdot i_{pj} + \alpha(w^{o}_{kj\,actual} - w^{o}_{kj\,anterior}) \qquad [3\text{-}32]$$

Donde el factor α se denomina constante de momento. El valor de α se fija inicialmente, permaneciendo inalterable durante todas las iteraciones.

De esta misma forma son actualizados los pesos de las capas ocultas.

$$w^{h}_{ji\,nuevo} = w^{h}_{ji\,actual} + \eta \cdot \delta^{h}_{pj} \cdot x_{pi} + \alpha(w^{h}_{ji\,actual} - w^{h}_{ji\,anterior}) \qquad [3\text{-}33]$$

Aunque se aumenta el tiempo que dura cada iteración, este método permite converger a una solución en menor cantidad de iteraciones.

3.2.2.4. Algoritmo de Levenberg Marquardt

Este algoritmo fue diseñado para encontrar las raíces de funciones formadas por la suma de los cuadrados de funciones no lineales, siendo el aprendizaje de redes neuronales, una aplicación especial de este algoritmo. El algoritmo de Levenberg Marquardt es una variación del método de iterativo de newton para encontrar las raíces de una función.

El algoritmo de Levenberg Marquardt puede aplicarse en cualquier problema donde se necesite encontrar los valores de las raíces de una función; en el caso de las redes neuronales artificiales, la función es el error cuadrático medio de las salidas de la red y las raíces de esta función son los valores correctos de los pesos sinápticos.

Para encontrar los valores de los pesos sinápticos utilizando éste algoritmo deben realizarse operaciones matemáticas con matrices; la matriz de error en los pesos sinápticos se calcula de la siguiente forma:

$$W_{nuevo} = W_{actual} - [J^T \cdot J + \mu \cdot I]^{-1} \cdot J^T \cdot E \qquad [3\text{-}34]$$

Donde W es una matriz constituida por valores de los pesos sinápticos, I es la matriz de identidad, μ es un valor escalar que varía de acuerdo al cambio en el error; E es la matriz de errores en la red y J es la matriz Jacobiana, ésta es calculada de la siguiente forma:

$$J = [\nabla E \cdot E^{-1}]^T \qquad [3\text{-}35]$$

Donde ∇E es la matriz de gradientes de error en valores numéricos.

El algoritmo de Levenberg Marquardt converge a la solución en menos iteraciones que la regla delta generaliza, pero cada iteración requiere mas tiempo, ya que se realizan mas operaciones. Cuando se entrena a la red utilizando una gran cantidad de ejemplos, es mejor utilizar el algoritmo de Levenberg Marquardt, ya que permite obtener la solución en un menor tiempo.

3.2.3. Descripción de aplicaciones

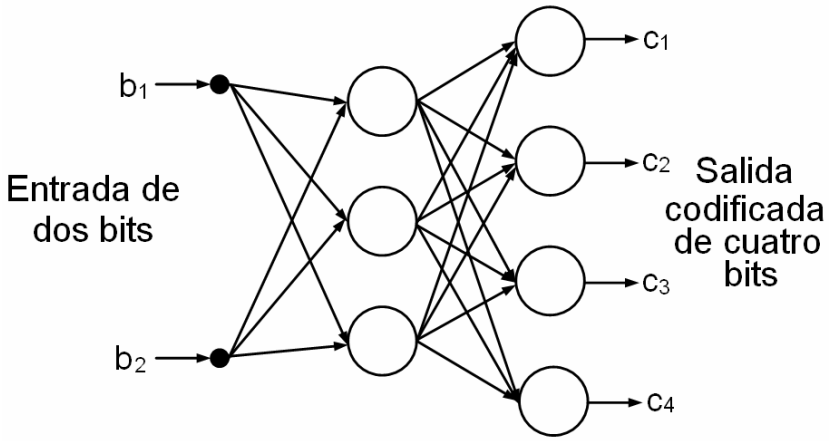
El perceptrón multicapa es la red neuronal mas utilizada actualmente; sus principales ventajas son el procesamiento en paralelo, la tolerancia al ruido, el aprendizaje mediante ejemplos y la generalización. Para implementar el perceptrón multicapa no se necesita construir algoritmos complejos, como en los sistemas secuenciales, solamente se debe entrenar la red, mostrándole un grupo de ejemplos de entrada con su salida.

El perceptrón multicapa puede ser utilizado en diversas aplicaciones en donde se necesite procesar información compleja, entre estas se puede mencionar codificación de datos, reconocimiento del habla, reconocimiento de imágenes visuales, control de robots, reconocimiento de caracteres, pronóstico de eventos y control automático de procesos.

Por ejemplo en la codificación de datos puede utilizarse un perceptrón multicapa con un número de entradas igual o menor al número de salidas. Para codificar información digital debe utilizarse una tabla con los diferentes valores de entrada con sus respectivos códigos de salida; los datos de la tabla deben utilizarse para el entrenamiento de la red, así cuando a la red se le presente una entrada, se obtendrá el correspondiente código salida.

En la figura 33 se muestra el diagrama de un perceptrón multicapa utilizado para generar un código de cuatro bits a partir de dos bits de información.

Figura 33. Codificador de información

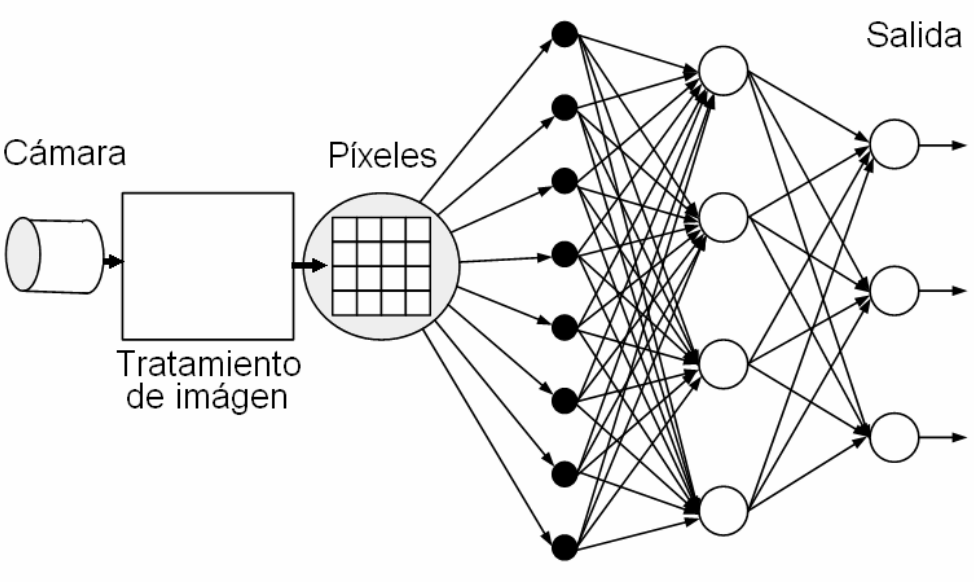


Otra ventaja de usar redes neuronales es que el tiempo requerido por una red ya entrenada en obtener la salida es bastante rápido, ya que las operaciones son realizadas en forma paralela; mientras que con un sistema secuencial el valor de entrada se compararía con todos los valores de la tabla hasta obtener la salida adecuada.

Una aplicación muy común del perceptrón multicapa es el reconocimiento de imágenes visuales. En esta aplicación, los píxeles que forman la imagen son usados como entradas a la red neuronal y los valores de salida deben corresponder con la clase a la que pertenece esta imagen. El entrenamiento debe realizarse con varios ejemplos de cada clase; estos ejemplos deben seleccionarse adecuadamente, para que la red pueda obtener las características que diferencian una clase de otra. Para mejorar la eficiencia de la red, generalmente se aplica algún tratamiento a la imagen antes de ingresarse a la red neuronal.

En la figura 34 se muestra un perceptrón multicapa usado para clasificar imágenes captadas por una cámara.

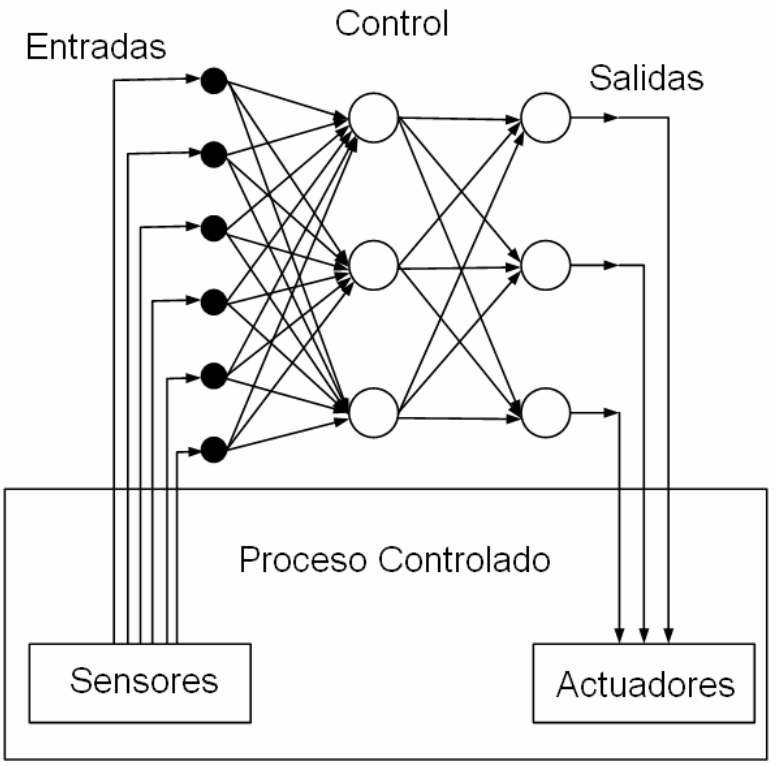
Figura 34. Clasificador de imágenes visuales



Otra aplicación del perceptrón multicapa es el control automático de un proceso industrial; en este caso, las salidas de los sensores son conectadas a la entrada de la red neuronal y las salidas de la red son conectadas a los actuadores. La tabla usada en el entrenamiento de la red debe contener todos los eventos posibles en el proceso, con su respectiva acción, de esta forma la red responderá con la acción adecuada a cada evento del proceso. La ventaja de usar redes neuronales en esta aplicación es que en caso de falla de algún sensor o ruido en la entrada, la red responderá con la acción adecuada, ya que la salida será la correspondiente al ejemplo que tenga la mayor similitud con esta entrada.

En la figura 35 se muestra un esquema del control automático de un proceso realizado por un perceptrón multicapa.

Figura 35. Control automático de proceso industrial



En muchas aplicaciones el proceso de entrenamiento podría ser tardado, según la complejidad de la información a clasificar, pero este entrenamiento es realizado solamente una vez; después de que la red haya sido entrenada, esta responde de forma rápida a las entradas que se le presentan.

El perceptrón multicapa puede ser utilizado en un gran número de aplicaciones, éste proporciona una solución aceptable en relativamente poco tiempo de desarrollo.

4. IMPLEMENTACIÓN UTILIZANDO MATLAB

La forma más práctica de implementar una red neuronal es utilizando un computador convencional, esto permite visualizar y modificar fácilmente diversas características de la red.

Un computador, a pesar de ser un sistema de procesamiento secuencial, puede ser utilizado para implementar redes neuronales artificiales, ya que éste se puede programar para realizar cualquier tipo de proceso algorítmico, incluyendo la simulación de un sistema de procesamiento en paralelo.

Al implementar redes neuronales utilizando un computador, es necesario disponer de algún software que permita su creación, entrenamiento y simulación. En este trabajo se utiliza el software Matlab versión 6.5 para implementar la red neuronal perceptrón multicapa.

Matlab es un entorno de computación que dispone de gran variedad de librerías (*Toolboxes*) que cubren muchas áreas de la ingeniería; una de éstas es la librería de redes neuronales (*Neural Network Toolbox*).

La librería de redes neuronales proporciona varias funciones para la creación, entrenamiento y simulación de los modelos de redes neuronales de mayor uso en la actualidad. En el caso del perceptrón multicapa, estas funciones permiten especificar el número de capas, la cantidad de neuronas de cada capa, las funciones de activación, el algoritmo de aprendizaje a utilizarse y otros parámetros.

4.1. El entorno de Matlab

El nombre Matlab proviene de "*Matrix Laboratory*" (laboratorio matricial), fue creado originalmente para resolver problemas de algebra matricial. Matlab es un entorno de computación que combina cálculo numérico, gráficos, simulación y un lenguaje de programación de alto nivel. Actualmente es usado en diversas áreas de aplicación.

En la figura 36 se muestra el entorno de trabajo de Matlab, el cual está constituido principalmente por la ventana de comandos (*Command Window*); todas las instrucciones deben escribirse en esta ventana después del símbolo "**>>**", conocido como indicador de comandos. Cada instrucción se ejecuta después de presionar la tecla Enter; en la mayoría de instrucciones los resultados aparecen en la misma ventana de comandos.

Figura 36. Entorno de trabajo de Matlab

Para evitar que los resultados arrojados por las instrucciones se desplieguen en la ventana de comandos, se finaliza cada instrucción con el caracter punto y coma (**;**).

En Matlab se utilizan variables en forma de arreglos para almacenar diferentes tipos de datos; estas variables pueden ser creadas, cambiadas o eliminadas en cualquier momento, por lo que no es necesario declararlas.

Para crear una variable solamente hay que asignarle un valor; en caso de que la variable ya esté definida, se sobrescribirá el valor anterior. Cualquier variable con un valor asignado estará disponible mientras no sea borrada; para ver el valor de una variable, hay que escribir su nombre en la ventana de comandos.

La forma general de las instrucciones es bastante similar a la notación matemática. Por ejemplo, la instrucción “a=5+10*2”, permite realizar una multiplicación y una suma, almacenando el resultado en la variable a; la instrucción “b=log(a)”, permite calcular el logaritmo del valor almacenado en la variable a, almacenando el resultado en la variable b.

Todas las instrucciones pueden escribirse directamente en la ventana de comandos de Matlab; sin embargo, al desarrollar un programa, es más ventajoso hacer un archivo que contenga las instrucciones, para poder revisarlo, modificarlo y ejecutarlo otra vez.

Matlab permite crear archivos de texto conteniendo instrucciones, las cuales serán ejecutadas una tras otra, al escribir el nombre del archivo en la ventana de comandos; estos archivos son llamados ficheros M (*M files*) y deben tener extensión m (***.m**).

Aunque los ficheros M se pueden crear con cualquier editor de texto, Matlab dispone de un editor que permite crear y modificar estos ficheros en un ambiente de programación. Para crear un fichero M se debe escribir ***edit*** en la ventana de comandos o seleccionar *New M-file* (crear nuevo fichero M), en el menú de Matlab.

En el trabajo presentado por Delores Etter[5], se realiza una exposición sobre las características del entorno de trabajo de Matlab, la representación de datos y los comandos básicos. El trabajo presentado por Shoichiro Nakamura[6] contiene un tutorial práctico para personas que empiezan a utilizar Matlab.

4.2. Implementación de arreglos vectoriales y matriciales en Matlab

Un arreglo es una variable que permite almacenar información en varias localidades o posiciones de memoria, dispuestas en una o mas dimensiones. Un arreglo vectorial es aquel, que sus posiciones de memoria están dispuestas en una dimensión; por ejemplo:

$$\begin{bmatrix} 4 & 2 & 6 & -10 & 8 & 0 & -2 & 10 \end{bmatrix}$$

Este arreglo contiene ocho posiciones de memoria dispuestas en una dimensión. Un arreglo matricial es aquel, que tiene sus posiciones de memoria dispuestas en dos dimensiones; por ejemplo:

$$\begin{bmatrix} 5 & 8 & 2 & 10 & 6 & 15 \\ 7 & 2 & 11 & 3 & 4 & 20 \\ 13 & 9 & 16 & -5 & 0 & -3 \\ 1 & -7 & 5 & 0 & 4 & 0 \end{bmatrix}$$

El arreglo anterior contiene veinticuatro posiciones de memoria dispuestas en dos dimensiones; ya que contiene cuatro filas y seis columnas, este arreglo es de 4x6 posiciones, mientras que el primero es de 1x8 posiciones.

En matlab todas las variables son arreglos, cuando un valor es asignado a una variable, esta variable constituye un arreglo de una posición; por ejemplo, la expresión valor1=8, crea un arreglo de 1x1 posiciones con el nombre valor1.

La asignación de valores a un arreglo vectorial se puede hacer directamente, mediante la siguiente forma:

A = [4 2 6 8 -10 5] o A = [4, 2, 6, 8, -10, 5]

Para separar las columnas se usa el caracter coma o simplemente un espacio. A los vectores matriciales se les asignan valores de la misma forma, pero usando el carácter punto y coma (**;**) para separar las filas; por ejemplo:

A = [0 5 10 5; 8 4 2 1; -10 7 9 1]

La expresión anterior permite asignar al arreglo A, la siguiente matriz de 3x4 posiciones:

$$\begin{bmatrix} 0 & 5 & 10 & 5 \\ 8 & 4 & 2 & 1 \\ -10 & 7 & 9 & 1 \end{bmatrix}$$

También se puede asignar valores a las posiciones de un arreglo, indicando éstas por medio de subíndices. Los subíndices se escriben entre paréntesis y el menor subíndice utilizado es 1.

Por ejemplo, para asignar un valor a la posición de la fila tres y columna cinco del arreglo matricial A, se escribe la siguiente expresión:

A(3,5) = 10

Cuando se trata de un arreglo vectorial solamente se escribe el número de la posición entre paréntesis, sin indicar que se trata de la fila uno.

Existen funciones especiales que permiten crear arreglos matriciales con valores predefinidos. La función zeros(m,n), crea un arreglo de mxn posiciones cuyos valores son cero; por ejemplo, la expresión C = zeros(5, 10), crea un arreglo de 5x10 y lo almacena en la variable C. Para crear un arreglo que en todas sus posiciones contenga unos, se usa la función ones(m,n). La función eye(m), permite crear un arreglo de mxm posiciones cuyos valores corresponden a una matriz de identidad.

Los arreglos matriciales permiten realizar operaciones matemáticas entre matrices, como suma, resta y multiplicación, utilizando la notación matemática normal (+ - *). Para realizar operaciones matemáticas con los elementos de un arreglo, excepto para la suma y resta, se antepone un punto al símbolo de la operación (**.* ./ .^**), para indicar que la operación es con cada elemento del arreglo y no se trata de una operación matricial; cuando la operación involucra dos arreglos, ambos deben tener las mismas dimensiones.

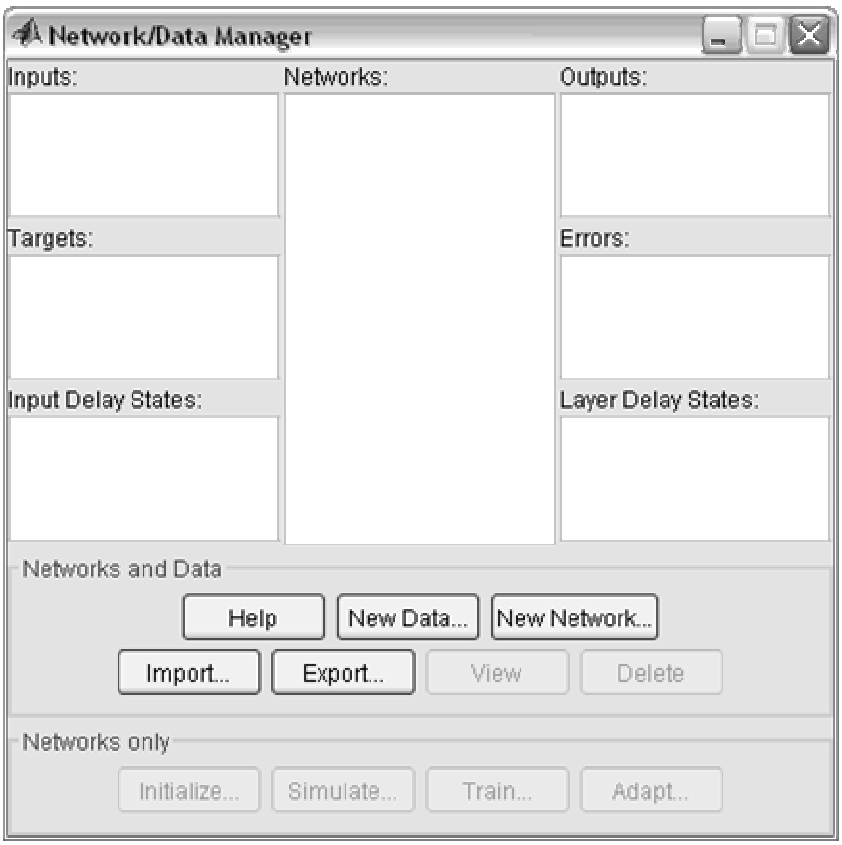
El trabajo realizado por Shoichiro Nakamura[7], explica el uso de arreglos en Matlab para realizar diversas operaciones con matrices. En el trabajo realizado por Delores Etter[8] se describen varias funciones de Matlab para realizar operaciones con arreglos.

4.3. Herramientas para redes neuronales

La librería de redes neuronales de Matlab provee diversas funciones para la implementación de varios tipos de redes neuronales artificiales, permitiendo definir la estructura de la red, funciones de activación, regla de aprendizaje y otros parámetros; además cuenta con una interfaz gráfica que facilita el desarrollo de aplicaciones con redes neuronales.

Para acceder a la interfaz gráfica de redes neuronales se escribe el comando **nntool** en la ventana de comandos. La figura 37 muestra la ventana principal de la interfaz gráfica de redes neuronales.

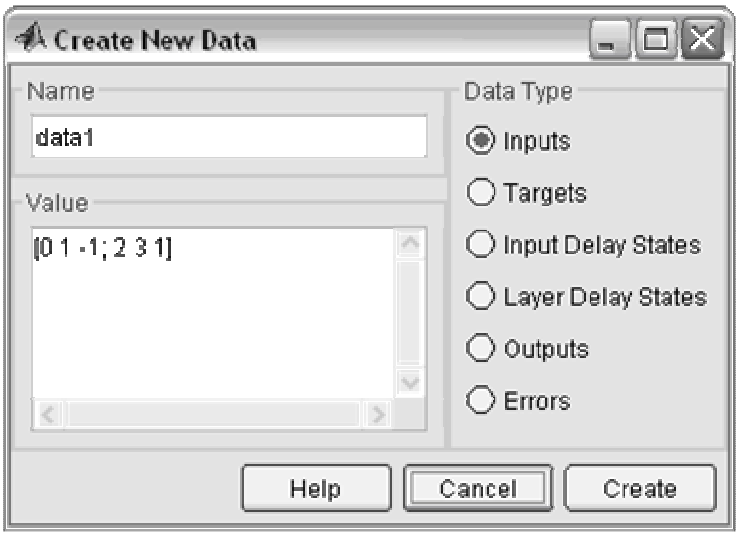
Figura 37. Ventana principal de la interfaz gráfica de redes neuronales



Con la interfaz gráfica de redes neuronales se puede crear, modificar, visualizar y eliminar arreglos utilizados como ejemplos de entrada-salida, arreglos usados como entrada en la simulación y redes neuronales; también permite importar y exportar datos al espacio de trabajo de Matlab.

Para crear un arreglo en la interfaz gráfica debe presionarse el botón *New Data* (Dato Nuevo) en la ventana principal; luego aparece la ventana mostrada en la figura 38.

Figura 38. Ventana para crear un arreglo en la interfaz gráfica



En esta ventana se coloca el nombre del arreglo así como su valor; como se puede observar, el valor se coloca en el mismo formato que se explicó anteriormente. En la parte derecha de la ventana se puede seleccionar el tipo de dato del arreglo; se selecciona "*Inputs*" (Entradas) para los arreglos que serán utilizados como entradas a la red neuronal, ya sea en el entrenamiento o en la simulación; se selecciona "*Targets*" (Objetivos) para los arreglos que representan las salidas deseadas, estos serán usados para el entrenamiento de la red.

4.3.1. Creación de red perceptrón multicapa

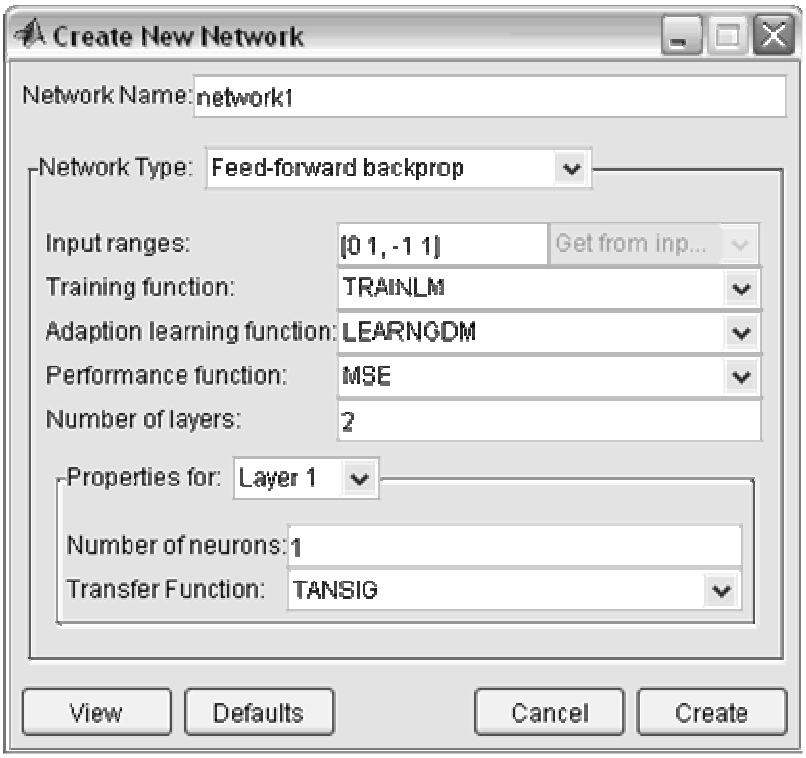
Antes de la creación de un perceptrón multicapa debe determinarse, de acuerdo a la aplicación, el número de capas, el número de neuronas en cada capa y las funciones de activación. Se puede obtener buenos resultados utilizando dos o tres capas de neuronas.

En la capa de salida se selecciona un número de neuronas igual al número de clases a diferenciar; el número de neuronas en las capas intermedias se selecciona de acuerdo al criterio del diseñador, generalmente se selecciona el doble de las neuronas de la capa de salida mas uno; otros criterios son, seleccionar la mitad de la suma del número de entradas con el número de neuronas en la capa de salida o también seleccionar el número de neuronas en la capa de salida multiplicado por diez.

Generalmente la función de activación usada en las capa de salida es la función sigmoidal logarítmica; en las capas intermedias se usa la función sigmoidal logarítmica o la función sigmoidal tangente hiperbólica. Cuando en la capa de salida se usa la función sigmoidal logarítmica, los valores de salida de los ejemplos deben estar entre 0.15 y 0.95, ya que esta función solamente alcanza el valor de cero y el valor de uno en el menos infinito e infinito respectivamente. Por esta misma razón, cuando se usa en la salida la función sigmoidal tangente hiperbólica deben usarse ejemplos con valores de salida entre -0.9 y 0.9.

Para crear una red perceptrón multicapa utilizando la interfaz gráfica de redes neuronales, debe presionarse el botón *New Network* (Nueva Red) en esta interfaz gráfica, luego aparecerá la ventana que se muestra en la figura 39.

Figura 39. Ventana para crear una red neuronal en la interfaz gráfica



Se le asigna un nombre a la red por medio del cuadro de texto *Network Name* (Nombre de red), que se encuentra en la parte superior de la ventana.

En el menú descendente *Network Type* (Tipo de red) se selecciona *Feed-forward backpropagation* (Alimentación hacia adelante, propagación hacia atrás), que es el nombre que Matlab le da al perceptrón multicapa.

En el cuadro de texto *Input ranges* (Rangos de entrada) se ingresa un arreglo nx2 que indica los rangos de valores del vector de entrada, donde n es el número de entradas; en la primera columna se escriben los valores mínimos de cada entrada y en la segunda columna se escriben los valores máximos de cada entrada.

También puede seleccionarse en el menú descendente *Get from input* (Obtener de la entrada) uno de los arreglos de entrada disponibles en la interfaz gráfica, para obtener los valores máximos y mínimos a partir de éste.

El algoritmo de entrenamiento que se utilizará se selecciona en el menú descendente *training function* (Función de entrenamiento). Matlab dispone de varios de algoritmos de entrenamiento, algunos de éstos son:

Traingd, es la regla delta generalizada o método del gradiente descendiente.
Traingda, es el método del gradiente descendiente con velocidad de aprendizaje variable.
Traingdm, es el método del gradiente descendiente con momento.
Traingdx, es el método del gradiente descendiente con velocidad de aprendizaje variable y momento; éste es una combinación de los dos anteriores.
Trainlm, es el algoritmo de Levenberg Marquardt.

En el cuadro de texto *Number of layers* (Número de capas) se escribe la cantidad de capas que tendrá la red. Mediante el menú descendente *properties for layer* (Propiedades de la capa) se indica el número de neuronas de cada capa, así como su función de activación; la función sigmoidal logarítmica aparece como LOGSIG; la función sigmoidal tangente hiperbólica aparece como TANSIG y la función lineal aparece como PURELIN.

Para crear un perceptrón multicapa en el espacio de trabajo de Matlab se usa la función **newff**, esta función permite asignar un nombre a la red, indicar los rangos de las entradas, declarar el número de neuronas de cada capa, declarar las funciones de activación en cada capa e indicar el algoritmo de entrenamiento que se utilizará.

La sintaxis de la función newff es la siguiente:

Net = newff(PR, [S_1 S_2 … S_N], {'TF_1' , 'TF_2' , … , 'TF_N'}, 'BTF')

Donde

Net, es el nombre asignado a la red.

R, es el número de entradas a la red.

PR, es la matriz de rangos de Rx2 posiciones donde aparecen los mínimos en la primera columna y máximos en la segunda columna.

S_i, es el número de neuronas en la i-ésima capa, para un total de N capas.

TF_i, es la función de activación de la i-ésima capa; ésta puede ser logsig, tansig o purelin; si no se indica se asume como tansig.

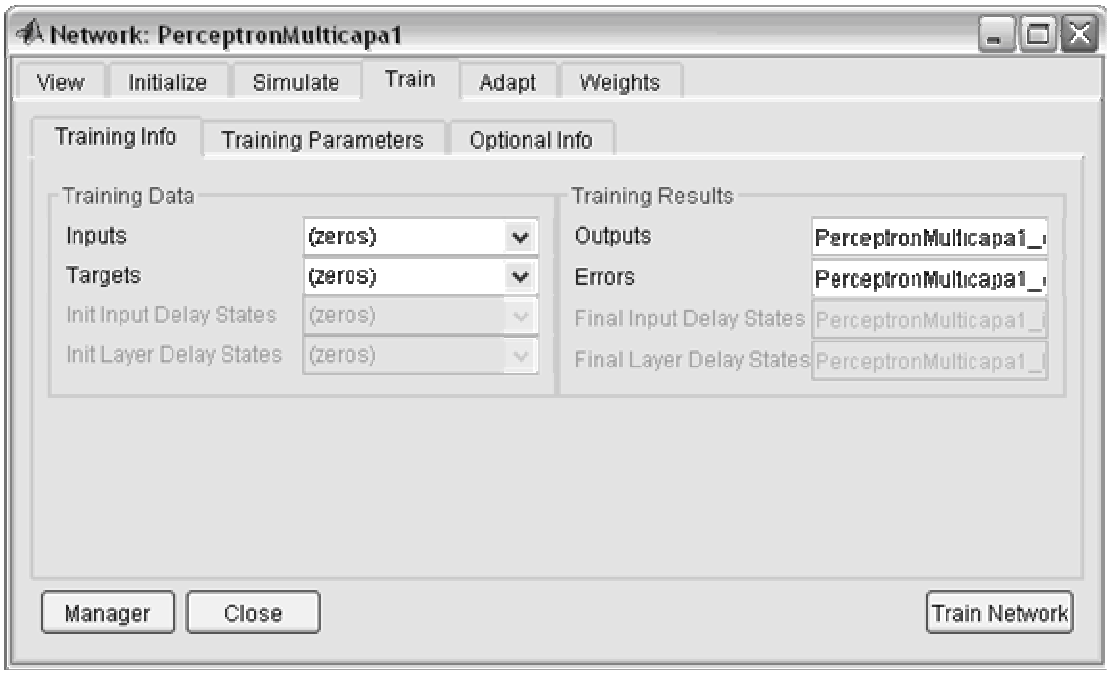
BTF, es el algoritmo de entrenamiento; existen muchas opciones, entre esas están traingd, traingda, traingdm, traingdx y trainlm; si no se indica se asume como traingdx.

Cuando se crea el perceptrón multicapa los valores de sus pesos sinápticos son asignados al azar, durante el entrenamiento éstos se ajustan de acuerdo a los ejemplos utilizados.

4.3.2. Entrenamiento de la red

Después de crear la red neuronal se procede a su entrenamiento; para entrenar un perceptrón multicapa utilizando la interfaz gráfica de la figura 37, debe seleccionarse la red en el cuadro *Networks* (Redes) y luego presionar el botón *train* (entrenar); al hacer esto aparecerá la ventana que se muestra en la figura 40.

Figura 40. Ventana para entrenar una red neuronal en la interfaz gráfica



Para el entrenamiento del perceptrón multicapa es necesario un grupo de entradas y salidas de ejemplos, éstos son arreglos que deben crearse en ésta interfaz gráfica de la forma como se explicó anteriormente; en el menú descendente *Inputs* (entradas), se selecciona el arreglo que contiene todas las entradas de ejemplo, cada columna en el arreglo representa una entrada; en el menú descendente *Targets* (objetivos), se selecciona el arreglo que contiene todas las salidas de ejemplo correspondientes a las entradas, también cada columna en el arreglo representa una salida, si en caso los arreglos de entrada y salida no poseen el mismo número de columnas se generará un mensaje de error al tratar de entrenar la red.

Después de seleccionar los arreglos que contienen las entradas y salidas de ejemplo, se presiona el botón *Train Network* (Entrenar Red) para iniciar el entrenamiento del perceptrón multicapa.

Durante el entrenamiento, se presenta una gráfica que muestra el error total en las salidas en función de las iteraciones realizadas.

Por medio de la pestaña *Training Parameters* (Parámetros de entrenamiento), se puede cambiar los parámetros del algoritmo de entrenamiento, entre estos están el parámetro *epochs* (épocas), que indica el número máximo de iteraciones, por defecto es cien iteraciones. También está el parámetro *goal* (meta), que indica el error mínimo aceptable para dar por finalizado el entrenamiento, por defecto es cero, pero generalmente es muy difícil llegar a éste valor de error, por lo que se recomienda cambiarlo.

Para entrenar a un perceptrón multicapa en el espacio de trabajo de Matlab, se usa la función **train**, ésta permite entrenar la red utilizando un arreglo que representa las entradas de ejemplo y otro que representa las salidas correspondientes a las entradas de ejemplo. La red es entrenada de acuerdo al algoritmo de entrenamiento definido en la creación de ésta.

La sintaxis de la función train es la siguiente:

[Net, TR] = train(Net, P, T)

Donde

Net, es el nombre de la red a entrenar.

TR, son los resultados devueltos después del entrenamiento: iteraciones realizadas, errores en la última iteración, etc.

R, **S**, es el número de entradas y salidas de la red respectivamente.

M, es el número de ejemplos con que se entrenará a la red.

P, es un arreglo de RxM, que contiene las entradas de ejemplo.

T, es un arreglo de SxM, que contiene las salidas de los ejemplos.

Antes de entrenar la red, pueden modificarse los parámetros de entrenamiento, esto se hace por medio del siguiente comando:

Net.trainparam.Parametro = Valor

Donde

Net, es el nombre de la red

Parametro, es el parámetro a modificar

Valor, es el valor que tomará ese parámetro

Los parámetros de aprendizaje varían de acuerdo al algoritmo de entrenamiento que se utilice; éstos se explican a continuación.

show, número de iteraciones entre dos mensajes de reporte, mostrados en la ventana de comandos.

epochs, número máximo de iteraciones.

goal, máximo error permitido

lr, factor de velocidad de aprendizaje η.

lr_inc, factor de incremento en la velocidad de aprendizaje.

lr_dec, factor de decremento en la velocidad de aprendizaje.

max_perf_inc, máximo incremento de error, admisible entre dos ciclos de entrenamiento consecutivos.

mc, constante de momento α.

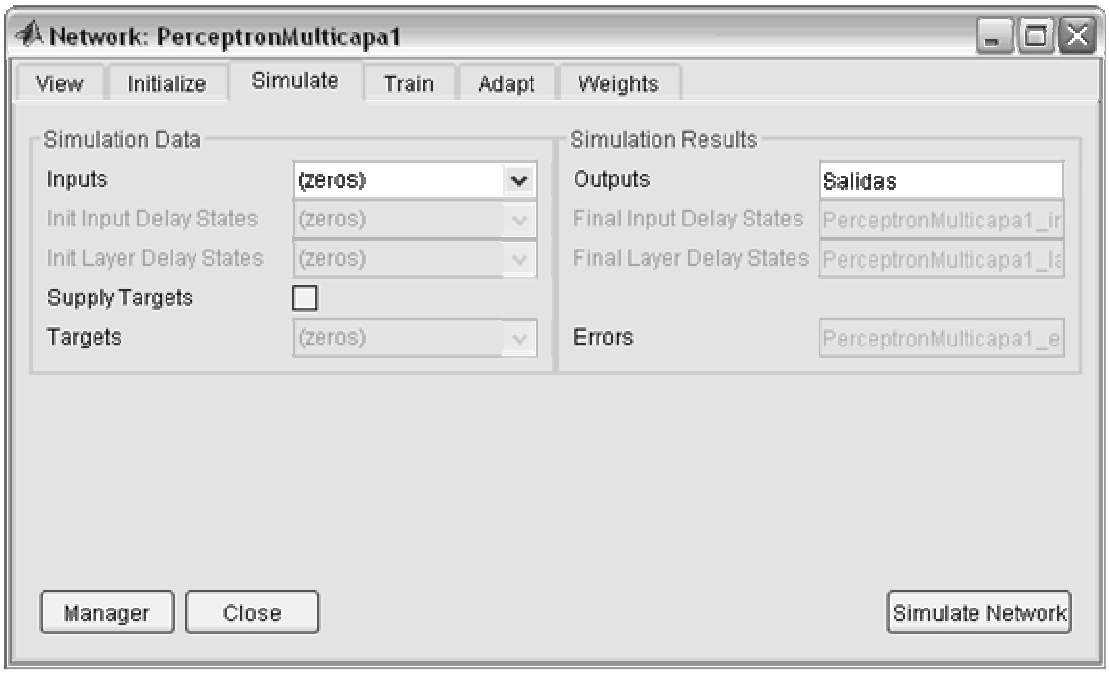
Los parámetros lr_inc, lr_dec y max_perf_inc se aplican a los algoritmos traingda y traingdx; el parámetro mc se aplica a los algoritmos traingdm y traingdx.

4.3.3. Simulación de la red

La simulación consiste aplicar una entrada a la red ya entrenada, para obtener una salida, poniendo en funcionamiento la red neuronal. Antes de simular la red debe observase que ésta haya sido entrenada exitosamente, de forma que el error promedio en las salidas sea tolerable, este valor es mostrado en la ventana de comandos al finalizar el entrenamiento de la red.

Para simular el perceptrón multicapa utilizando la interfaz gráfica de redes neuronales de la figura 37, debe seleccionarse la red ya entrenada en el cuadro *Networks* (Redes) y luego presionar el botón *simulate* (simular); al hacer esto aparecerá la ventana que se muestra en la figura 41.

Figura 41. Ventana para simular una red neuronal en la interfaz gráfica



En el menú descendente *Inputs* (entradas), se selecciona el arreglo que contiene todas las entradas, cada columna es una entrada diferente.

En el cuadro de texto Outputs (Salidas), se escribe el nombre del arreglo que contendrá las salidas de la red, correspondientes a las entradas en la simulación. El arreglo salida tendrá el mismo número de columnas que el arreglo de entrada, ya que cada columna en el arreglo de entrada genera una columna en el arreglo de salida.

Para generar el arreglo de salida, se presiona el botón *Simulate Network* (Simular Red); la simulación es bastante rápida, el arreglo de salida puede visualizarse en la interfaz gráfica de redes neuronales.

Para simular la red ya entrenada en el espacio de trabajo de Matlab, se usa la función **sim**, ésta permite generar un arreglo de salida a partir de un arreglo de entrada.

La sintaxis de la función sim es la siguiente:

A = sim(Net, P)

Donde

Net, es la red neuronal ya entrenada.

R, **S**, es el número de entradas y salidas de la red respectivamente.

M, es el número de patrones de entrada a ingresar a la red.

P, es el arreglo de entrada de RxM.

A, es el arreglo de salida de SxM.

Para ingresar un patrón de entrada a la red se ingresa un arreglo de Rx1 y se obtiene una salida de Sx1, si se desea se pueden ingresar varios patrones de entrada a la vez, colocándolos en cada columna del arreglos de entrada, así se tendrá un arreglo de RxM con M patrones de entrada.

La mejor forma de implementar redes neuronales es usando funciones en el espacio de trabajo de Matlab. La interfaz gráfica de redes neuronales es una buena herramienta para empezar a utilizar redes neuronales artificiales en Matlab, pero el tiempo necesario para crear, entrenar o simular una red neuronal, es mucho mayor que el requerido cuando se trabaja desde la ventana de comandos.

La interfaz gráfica de redes neuronales también permite importar o exportar redes neuronales o arreglos al espacio de trabajo de Matlab, para hacer esto, se presiona el botón *Import* (Importar) para llevar cualquier arreglo o red neuronal desde el espacio de trabajo de Matlab hasta la interfaz gráfica de redes neuronales y para llevar al espacio de trabajo de Matlab cualquier arreglo o red neuronal desde la interfaz gráfica de redes neuronales se presiona el botón *Export* (Exportar).

En la ayuda que provee Matlab[9], se explican todas las funciones de la librería de redes neuronales y se desarrollan ejemplos de diferentes tipos de redes.

4.4. Formas de aplicación

El perceptrón multicapa posee las ventajas de ser fácil de implementar y responder rápidamente a las entradas que se le presentan; su programación consiste en presentarle un grupo de ejemplos, sin necesidad de desarrollar complicados algoritmos; la salida se genera por la propagación de las señales a través de sus capas, no necesitando comparar la entrada con los valores de una tabla para poder decidir cual es la salida correcta.

Al desarrollar una aplicación del perceptrón multicapa utilizando Matlab, primero hay que llevar la información necesaria, al espacio de trabajo de Matlab; para esto, Matlab provee diversas funciones que permiten captar información de cualquier medio externo al espacio de trabajo. Luego de tener la información en el espacio de trabajo de Matlab, ésta deberá colocarse en un arreglo vectorial, expresada en forma numérica en el rango correspondiente a la función de activación de la capa oculta más cercana a la entrada.

Al disponer de varios ejemplos de entrada, expresados por medio de arreglos numéricos de una dimensión, se colocan en cada columna de un arreglo matricial, todos los ejemplos de entrada, y en otro arreglo matricial se indican las salidas deseadas de las columnas del arreglo de entrada.

Después se crea el perceptrón multicapa en Matlab, con la arquitectura y algoritmo de entrenamiento que se considere más conveniente para la aplicación; luego, utilizando el arreglo de entrada y salida, se entrena al perceptrón multicapa; debe observase que se llegue a un error suficientemente bajo para que el entrenamiento sea aceptable.

Finalmente, en la simulación, se ingresa al perceptrón multicapa cada entrada que desea reconocerse, obteniendo la salida correspondiente; ésta salida puede ser enviada a un medio externo para realizar una determinada acción.

Para ilustrar la implementación del perceptrón multicapa utilizando Matlab, se desarrollará un ejemplo, en el cual se utilizará este modelo de red neuronal para reconocer caracteres numéricos en imágenes de mapa de bits.

4.4.1. Ejemplo de reconocimiento de caracteres numéricos en imágenes

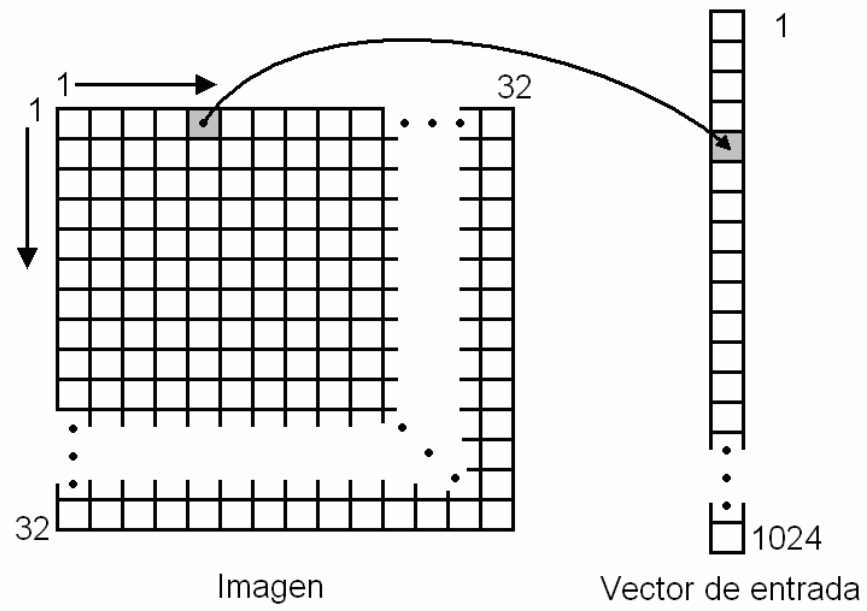
Para desarrollar este ejemplo se utilizó el software Matlab versión 6.5 de Mathworks, una computadora con 128 MB. de memoria RAM, Procesador AMD Athlon de 1.1 GHz. y sistema operativo Microsoft Windows Xp.

También fueron utilizadas cien imágenes de mapa de bits de 64x64 píxeles, que contenían los números del cero al nueve, dibujados en diez diferentes formas; estas imágenes estaban almacenadas en la carpeta "D:\Figuras" y sus nombres tenían un formato específico, en el cual se indicaba el tipo de letra por medio de un número al principio del nombre y separado por un guión, se indicaba el número que contenía la imagen, a excepción del diez, ya que estas imágenes contenían el número cero.

Se desarrollo un programa que implementaba el perceptrón multicapa para reconocer los caracteres numéricos dibujados en estas imágenes. Al principio se leyeron todas las imágenes que se utilizarían; todas éstas fueron redimensionadas a 32x32 píxeles, para disminuir el tamaño del vector de entrada; también las imágenes fueron convertidas a blanco y negro.

Cada imagen fue colocada en un arreglo unidimensional de 1024x1, cada posición del arreglo contenía un valor numérico en un rango de cero a uno, que representaba la intensidad de cada píxel, el cero representaba al color negro y el uno representaba al blanco. La lectura de los píxeles para su colocación en el arreglo vectorial, se realizó de izquierda a derecha y de arriba hacia abajo, como se muestra en la figura 42.

Figura 42. Posicionamiento de los píxeles en el arreglo de entrada

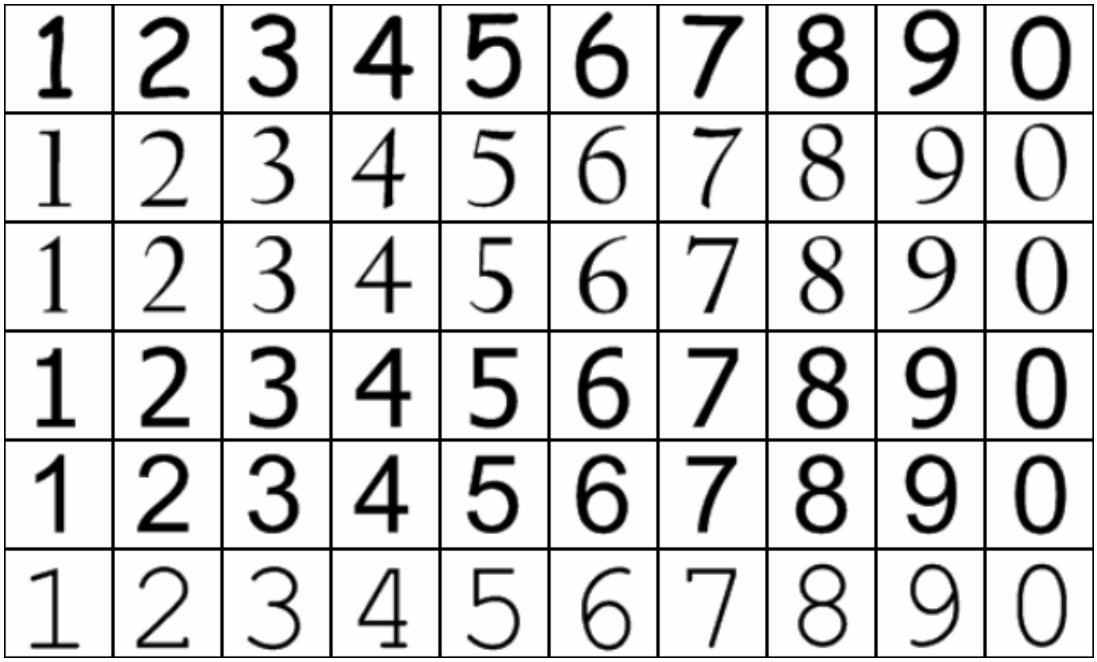


Se creo un perceptrón multicapa con un vector de entrada de tamaño 1024, dos capas ocultas y la capa de salida de 10 neuronas; la primera capa oculta tenía 100 neuronas y la segunda 50; para calcular el tamaño de la primera capa oculta se tomó el criterio de el número de clases multiplicado por diez; el tamaño de la segunda capa oculta se asumió por criterio propio. A todas las neuronas les fue asignada una función de activación sigmoidal logarítmica, ya que las entradas van de cero a uno y las salidas también se tomaron en ese rango.

El perceptrón multicapa fue entrenado por medio del método del gradiente descendiente con velocidad de aprendizaje variable y momento. Ya que se disponían de diez tipos diferentes de dibujos de cada número, se utilizaron seis tipos de cada número para entrenar la red, los cuatro tipos restantes fueron usados para probar la capacidad de generalización de la red.

Las sesenta imágenes utilizadas para entrenar la red se muestran en la figura 43.

Figura 43. Imágenes usadas en el entrenamiento de la red

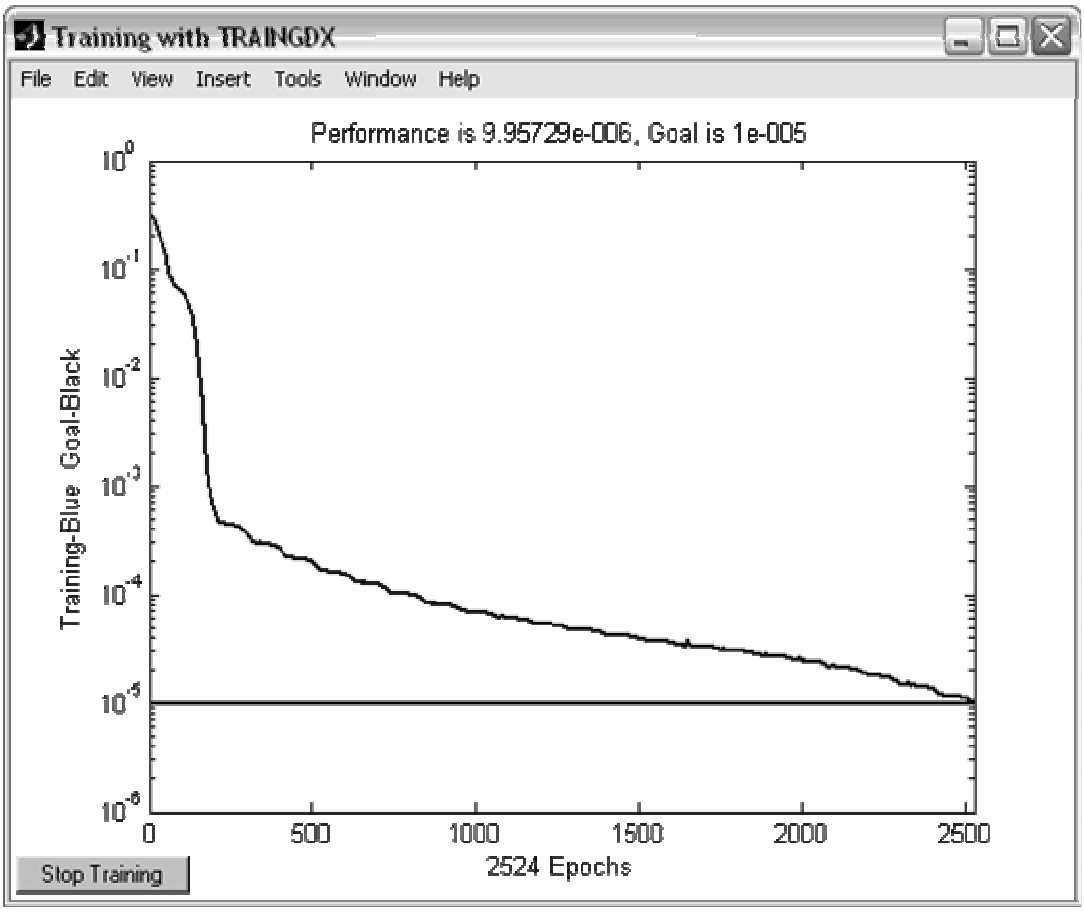


Los arreglos vectoriales que contenían las imágenes del entrenamiento fueron colocados en un arreglo matricial, donde cada columna contenía los píxeles de una imagen diferente. El arreglo matricial de salida usado en el entrenamiento contenía dos tipos de valores (0.05 y 0.95); los valores de éste arreglo se colocaron de acuerdo a los valores deseados de salida, éstos se describen de la forma siguiente: cuando la imagen contenía el número uno, la salida de la primera neurona debía ser 0.95, mientras todas las demás eran 0.05; si la imagen contenía el número dos, la salida de la segunda neurona debía ser 0.95, mientras las demás eran 0.05 y así sucesivamente.

La red fue entrenada después de 2,524 iteraciones, con un error máximo permisible de 9.957×10^{-6}.

Al finalizar el entrenamiento se obtuvo una gráfica del error máximo permisible en función de las iteraciones realizadas, esta gráfica se presenta en la figura 44.

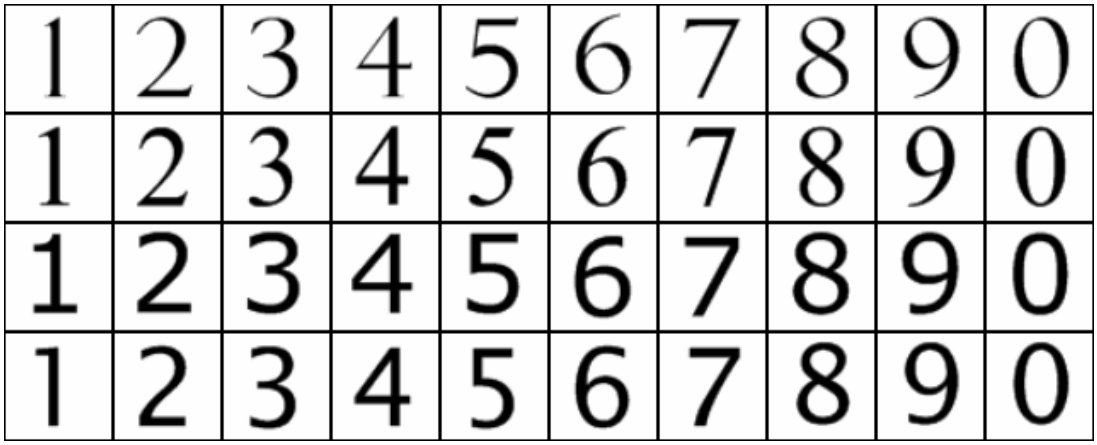
Figura 44. Gráfica del Error vrs. Iteraciones realizadas en el entrenamiento



Para probar la red se ingresaron todas las imágenes al perceptrón multicapa durante la simulación, obteniendo rápidamente diversos valores de salida; el número reconocido se tomó como el correspondiente a la neurona de salida que tenga el mayor valor; así, si el mayor valor se obtiene en la primera neurona, se reconoció el número uno, si se obtuvo en la última neurona de salida, se reconoció el número cero.

Las imágenes que no fueron utilizadas en el entrenamiento, pero se utilizaron en la simulación (para probar la capacidad de generalización del perceptrón) se muestran en la figura 45.

Figura 45. Imágenes que no se usaron en el entrenamiento de la red



Para poder visualizar los números reconocidos por el perceptrón multicapa durante la simulación con todas las imágenes, éstos se almacenaron en un arreglo vectorial.

Finalmente, al revisar el arreglo vectorial que contenía todos los números reconocidos, se observó que todas las imágenes fueron reconocidas correctamente, incluso aquellas imágenes que no fueron utilizadas durante el entrenamiento, quedando demostrada la capacidad de generalización del perceptrón multicapa. En el apéndice A, se muestran las salidas obtenidas para todas las imágenes utilizadas en la simulación.

El código fuente de este programa se presenta en el apéndice B.

CONCLUSIONES

1. El perceptrón multicapa es la red neuronal artificial más utilizada en la actualidad, ésta procesa la información en forma paralela y aprende mediante ejemplos presentados durante su entrenamiento, siendo adecuada para aplicaciones donde se necesite procesar información compleja; esta red neuronal es capaz de abstraer las características esenciales de los ejemplos aprendidos, para procesar, correctamente, entradas incompletas o distorsionadas.

2. La capacidad de procesamiento de la red neuronal perceptrón multicapa es adquirida durante el proceso de aprendizaje, el cual consiste en variar los valores de los pesos sinápticos de las neuronas de la red, de acuerdo con un algoritmo de entrenamiento, de forma que se obtengan las salidas deseadas para todos los ejemplos de entrada; de esta forma, todo el conocimiento del perceptrón multicapa se encuentra distribuido en los valores de los pesos sinápticos de sus neuronas.

3. La forma más práctica de implementar la red neuronal perceptrón multicapa es utilizando un computador convencional, esto permite visualizar y modificar, fácilmente, la estructura de la red, implementar un algoritmo de entrenamiento y variar los valores de los pesos sinápticos de sus neuronas; también, es necesario disponer de un software que permita implementar este tipo de red neuronal.

4. La librería de redes neuronales de Matlab provee diversas funciones para implementar la red neuronal perceptrón multicapa; permitiendo definir el número de capas, número de neuronas y función de activación en cada capa, rango de valores de sus entradas, regla de aprendizaje y sus parámetros; además cuenta con una interfaz gráfica que facilita el desarrollo de aplicaciones.

5. El perceptrón multicapa puede implementarse fácilmente y responde rápidamente a las entradas que se le presentan; su programación consiste en presentarle un grupo de ejemplos, sin necesidad de desarrollar complicados algoritmos; la salida se genera por la propagación de las señales a través de sus capas, no necesitando comparar la entrada con los valores de una tabla para poder decidir cual es la salida correcta.

6. Se demostró la capacidad de procesamiento de la red neuronal perceptrón multicapa mediante un ejemplo realizado en Matlab, en el cual fue creada y entrenada una red perceptrón multicapa para reconocer caracteres numéricos en imágenes de mapa de bits de 64x64 píxeles, observándose que la red reconoció, correctamente, todos los caracteres numéricos presentados, incluso en aquellas imágenes que no fueron utilizadas en el entrenamiento.

RECOMENDACIONES

1. Las redes neuronales artificiales deben considerarse como una alternativa en la solución de aquellos problemas en los cuales no se han obtenido buenos resultados utilizando los métodos tradicionales.

2. El perceptrón multicapa puede ser utilizado en aplicaciones en las cuales se necesite procesar información rápidamente.

3. Los ejemplos utilizados para el entrenamiento del perceptrón multicapa deben seleccionarse adecuadamente para que la red adquiera la capacidad de generalización.

4. El perceptrón multicapa puede ser una buena solución en aquellas aplicaciones en las cuales se maneja información incompleta o se necesita procesar información compleja.

REFERENCIAS

1. Benjamín C. Kuo, **Sistemas de control automático.** (7ª. Edición; México: Editorial Prentice Hall Hispanoamericana, 1996) pp. 2-19

2. Abid., pp. 77-117

3. Larry Long, **Introducción a las computadoras y al procesamiento de información.** (2ª. Edición; México: Editorial Prentice All Hispanoamericana, 1990) pp. 134-156

4. **Organization of Behaviour** citado por James A. Freeman, **Redes neuronales. Algoritmos, aplicaciones y técnicas de programación.** (Estados Unidos: Editorial Addison Wesley Iberoamericana, 1993) p. 16

5. Delores M. Etter, **Solución de problemas de ingeniería con Matlab.** (2ª. Edición; México: Editorial Prentice Hall Hispanoamericana, 1998) pp. 33-121

6. Shoichiro Nakamura, **Análisis numérico y visualización gráfica con Matlab.** (1ª. Edición; México: Editorial Prentice Hall Hispanoamericana, 1997) pp. 1-32

7. Abid., pp. 94-136

8. Delores M. Etter, Op. cit., pp. 124-142

9. **Ayuda de Matlab.** (Matlab versión 6.5; Estados Unidos: The Mathworks, 18 de Junio de 2002) Neural Network Toolbox

BIBLIOGRAFÍA

1. Freeman, James A. y David M. Skapura, **Redes Neuronales. Algoritmos, aplicaciones y técnicas de programación.** Estados Unidos: Editorial Addison Wesley Iberoamericana, 1993. 431pp.

Referencias electrónicas

2. Acosta, Maria Isabel y Alfonso Zuluaga. **Tutorial sobre redes neuronales artificiales aplicadas en ingeniería eléctrica.** http:// ohm.utp.edu.co/neuronales/Navigator.html. Colombia: Universidad Tecnológica de Pereira, Agosto 2006.

3. Bollilla, Ana. **Redes neuronales.** www.monografias.com. Uruguay, Agosto 2006.

4. Daza, Sandra Patricia. **Redes neuronales artificiales. Fundamentos, modelos y aplicaciones.** http://www.ilustrados.com/publicaciones/ EpyVZEuFAAboUblISx.php. Colombia: Universidad de Nueva Granada, Agosto 2006.

5. Díaz, Moisés Daniel. **Redes neuronales artificiales.** http:// www.cs.us.es/~delia/sia/html98-99/pag-alumnos/web9/indice.html. España, Julio 2006.

6. González Penedo, Manuel. **Curso de Redes Neuronales.** http:// carpanta.dc.fi.udc.es/~cipenedo/cursos/scx/scx.html. España, Agosto 2006.

7. **Librería de redes neuronales para usar con Matlab.** http:// www.mathworks.com/access/helpdesk/help/pdf_doc/nnet/nnet.pdf. Estados Unidos: The MathWorkrs, Septiembre 2006. 840pp.

8. **Redes neuronales artificiales**. http://www.iiia.csic.es/~mario/rna/tutorial/ index.html. España, Agosto 2006.

APÉNDICE A

A continuación, se presentan los valores obtenidos en las salidas del perceptrón multicapa del ejemplo del capitulo 4, para todas las imágenes mostradas en la simulación, ordenadas según el número dibujado en la imagen.

Tabla III. Salida para imágenes con el número uno

Salida	Valor deseado	Tipo de imagen									
		Usadas en el entrenamiento						No usadas anteriormente			
		1	2	3	4	5	6	7	8	9	10
1	**0.95**	**0.9486**	**0.9503**	**0.9501**	**0.9516**	**0.9461**	**0.95**	**0.9364**	**0.95**	**0.9676**	**0.934**
2	0.05	0.052	0.0506	0.0492	0.0508	0.0493	0.0495	0.078	0.0625	0.1167	0.0664
3	0.05	0.0448	0.0474	0.0538	0.0538	0.0489	0.0468	0.0696	0.0663	0.0482	0.0235
4	0.05	0.0502	0.0478	0.0504	0.0531	0.0516	0.0469	0.0279	0.0464	0.0348	0.0548
5	0.05	0.0525	0.0498	0.0447	0.0509	0.0483	0.0534	0.0714	0.0714	0.1111	0.0399
6	0.05	0.0513	0.0472	0.0504	0.0511	0.0502	0.0508	0.0333	0.0326	0.0221	0.0655
7	0.05	0.0502	0.0506	0.05	0.0513	0.0468	0.0496	0.0528	0.0441	0.0469	0.0476
8	0.05	0.0556	0.048	0.0497	0.0474	0.0522	0.0499	0.0664	0.0546	0.057	0.0416
9	0.05	0.0482	0.0533	0.0467	0.0514	0.0475	0.0498	0.0438	0.0618	0.0493	0.0903
10	0.05	0.0527	0.0502	0.0519	0.0487	0.0463	0.0487	0.0459	0.0591	0.0411	0.0595

Tabla IV. Salida para imágenes con el número dos

Salida	Valor deseado	Tipo de imagen									
		Usadas en el entrenamiento						No usadas anteriormente			
		1	2	3	4	5	6	7	8	9	10
1	0.05	0.048	0.0526	0.0503	0.048	0.0509	0.0512	0.0316	0.0523	0.0839	0.0689
2	**0.95**	**0.9485**	**0.9461**	**0.9501**	**0.9513**	**0.9502**	**0.9489**	**0.7446**	**0.9532**	**0.9234**	**0.9734**
3	0.05	0.05	0.0489	0.0516	0.0501	0.0499	0.0496	0.4403	0.0347	0.0779	0.034
4	0.05	0.052	0.0452	0.0526	0.0528	0.0477	0.0488	0.0357	0.0391	0.0555	0.0222
5	0.05	0.0496	0.0495	0.0495	0.05	0.0506	0.05	0.0809	0.094	0.0164	0.0251
6	0.05	0.0501	0.0533	0.0488	0.0487	0.0509	0.0511	0.0232	0.0199	0.0303	0.042
7	0.05	0.0493	0.0485	0.0502	0.0496	0.0507	0.049	0.0402	0.0775	0.0763	0.0363
8	0.05	0.0521	0.0489	0.0508	0.0514	0.0478	0.0493	0.1487	0.054	0.0644	0.0553
9	0.05	0.0508	0.0506	0.0514	0.0503	0.0495	0.0497	0.026	0.109	0.0243	0.0612
10	0.05	0.0502	0.0517	0.0466	0.0507	0.0502	0.0505	0.0427	0.0359	0.1075	0.0576

Tabla V. Salida para imágenes con el número tres

Salida	Valor deseado	Tipo de imagen									
		Usadas en el entrenamiento						No usadas anteriormente			
		1	2	3	4	5	6	7	8	9	10
1	0.05	0.0501	0.0491	0.0521	0.0537	0.0488	0.0465	0.0242	0.0855	0.0054	0.0139
2	0.05	0.0497	0.0509	0.0504	0.0526	0.0502	0.0521	0.061	0.0595	0.1419	0.0901
3	**0.95**	**0.9493**	**0.9503**	**0.951**	**0.9504**	**0.9494**	**0.9492**	**0.9202**	**0.696**	**0.7179**	**0.8172**
4	0.05	0.0498	0.0523	0.0472	0.0524	0.0489	0.0525	0.0293	0.4072	0.0498	0.0755
5	0.05	0.0499	0.052	0.0499	0.0492	0.0503	0.0501	0.0099	0.1982	0.0833	0.0433
6	0.05	0.0511	0.0456	0.0543	0.0393	0.0527	0.0469	0.0165	0.1348	0.0113	0.0151
7	0.05	0.0494	0.0521	0.0481	0.0544	0.0481	0.0512	0.0792	0.1746	0.2382	0.0321
8	0.05	0.05	0.0529	0.0462	0.0485	0.0505	0.0524	0.0756	0.016	0.0398	0.0355
9	0.05	0.0502	0.0469	0.0526	0.0485	0.0505	0.051	0.1319	0.0322	0.101	0.0875
10	0.05	0.0498	0.0511	0.0501	0.0493	0.0506	0.049	0.0476	0.0259	0.0352	0.1123

Tabla VI. Salida para imágenes con el número cuatro

Salida	Valor deseado	Tipo de imagen									
		Usadas en el entrenamiento						No usadas anteriormente			
		1	2	3	4	5	6	7	8	9	10
1	0.05	0.0517	0.0536	0.0527	0.0469	0.0471	0.0502	0.0211	0.0464	0.048	0.0399
2	0.05	0.0484	0.0475	0.0497	0.0513	0.0535	0.0501	0.2536	0.0989	0.021	0.0318
3	0.05	0.049	0.0538	0.0481	0.0518	0.0477	0.0503	0.2399	0.0789	0.217	0.0372
4	**0.95**	**0.9499**	**0.9469**	**0.9494**	**0.9484**	**0.9512**	**0.9497**	**0.7805**	**0.8994**	**0.9193**	**0.9694**
5	0.05	0.0489	0.0491	0.0491	0.0505	0.0528	0.0503	0.0094	0.0117	0.0592	0.0588
6	0.05	0.0493	0.0494	0.0503	0.049	0.0506	0.0501	0.1207	0.0923	0.0773	0.0888
7	0.05	0.0515	0.0507	0.0511	0.0497	0.0494	0.0499	0.0487	0.0664	0.021	0.0722
8	0.05	0.0504	0.0123	0.0102	0.0552	0.0512	0.0499	0.0242	0.0573	0.0517	0.0862
9	0.05	0.0509	0.0519	0.0524	0.0505	0.0476	0.0493	0.0501	0.0231	0.0213	0.0329
10	0.05	0.0504	0.0504	0.0506	0.0478	0.0511	0.0505	0.0432	0.0237	0.085	0.0335

Tabla VII. Salida para imágenes con el número cinco

Salida	Valor deseado	Tipo de imagen									
		Usadas en el entrenamiento						No usadas anteriormente			
		1	2	3	4	5	6	7	8	9	10
1	0.05	0.0504	0.0493	0.0468	0.0492	0.0501	0.0502	0.0774	0.0576	0.041	0.102
2	0.05	0.051	0.0508	0.0435	0.0501	0.0502	0.0498	0.0435	0.0811	0.2116	0.0369
3	0.05	0.0501	0.0497	0.0524	0.0508	0.0497	0.0505	0.0079	0.0886	0.0477	0.1364
4	0.05	0.0503	0.0513	0.0518	0.0502	0.0504	0.0494	0.0372	0.0877	0.018	0.1649
5	**0.95**	**0.9499**	**0.9493**	**0.9454**	**0.9503**	**0.95**	**0.9503**	**0.5636**	**0.4402**	**0.6171**	**0.5037**
6	0.05	0.0499	0.0496	0.0526	0.0498	0.05	0.0496	0.0263	0.132	0.0783	0.0328
7	0.05	0.0502	0.0501	0.0473	0.05	0.0503	0.0505	0.1234	0.079	0.1091	0.0298
8	0.05	0.0498	0.0505	0.0524	0.0499	0.0502	0.0496	0.0868	0.1217	0.158	0.0445
9	0.05	0.0501	0.0479	0.0519	0.0494	0.0512	0.0502	0.1397	0.016	0.0522	0.0522
10	0.05	0.0499	0.0509	0.0468	0.0501	0.0498	0.0502	0.0458	0.0216	0.0333	0.0365

Tabla VIII. Salida para imágenes con el número seis

Salida	Valor deseado	Tipo de imagen									
		Usadas en el entrenamiento						No usadas anteriormente			
		1	2	3	4	5	6	7	8	9	10
1	0.05	0.0504	0.0491	0.0475	0.0509	0.0503	0.0513	0.0806	0.0469	0.0505	0.0235
2	0.05	0.0493	0.0501	0.0511	0.0507	0.0496	0.0502	0.0194	0.1075	0.0901	0.065
3	0.05	0.0505	0.0505	0.0489	0.0498	0.0489	0.0503	0.1328	0.065	0.0265	0.0414
4	0.05	0.0499	0.0498	0.0535	0.0503	0.0502	0.0509	0.0456	0.1462	0.0558	0.0471
5	0.05	0.0504	0.0506	0.0502	0.0495	0.0501	0.0497	0.1077	0.0324	0.221	0.1129
6	**0.95**	**0.95**	**0.9481**	**0.9508**	**0.9495**	**0.9493**	**0.9493**	**0.3391**	**0.9329**	**0.6749**	**0.7153**
7	0.05	0.0494	0.0503	0.0512	0.0501	0.0503	0.0504	0.1539	0.0177	0.0775	0.0213
8	0.05	0.05	0.049	0.0514	0.0505	0.0504	0.0502	0.0388	0.0328	0.1672	0.0347
9	0.05	0.0499	0.0496	0.0511	0.0503	0.05	0.0491	0.039	0.0338	0.0576	0.0836
10	0.05	0.0502	0.0479	0.0519	0.0496	0.0506	0.0494	0.5546	0.0665	0.0714	0.4145

Tabla IX. Salida para imágenes con el número siete

Salida	Valor deseado	Tipo de imagen									
		Usadas en el entrenamiento						No usadas anteriormente			
		1	2	3	4	5	6	7	8	9	10
1	0.05	0.0481	0.0488	0.0507	0.0466	0.054	0.0507	0.0441	0.0375	0.0384	0.0862
2	0.05	0.048	0.05	0.05	0.05	0.0511	0.0505	0.0876	0.0318	0.1165	0.2675
3	0.05	0.0526	0.0492	0.0527	0.0501	0.0479	0.0486	0.0297	0.0444	0.031	0.0669
4	0.05	0.05	0.0506	0.0482	0.0505	0.0485	0.0516	0.035	0.0399	0.0556	0.084
5	0.05	0.0478	0.0533	0.0511	0.0461	0.0518	0.0495	0.2063	0.0361	0.0697	0.0266
6	0.05	0.0498	0.0524	0.0492	0.048	0.0494	0.0522	0.0301	0.0288	0.0311	0.0623
7	**0.95**	**0.9491**	**0.9479**	**0.9507**	**0.9535**	**0.9505**	**0.9471**	**0.8647**	**0.8662**	**0.9197**	**0.8398**
8	0.05	0.0504	0.0547	0.0495	0.0455	0.0492	0.052	0.0244	0.0695	0.0267	0.0891
9	0.05	0.0493	0.0481	0.0485	0.052	0.0512	0.0503	0.0578	0.1025	0.0512	0.1427
10	0.05	0.0487	0.0499	0.0476	0.0531	0.0505	0.051	0.0625	0.0446	0.0466	0.0989

Tabla X. Salida para imágenes con el número ocho

Salida	Valor deseado	Tipo de imagen									
		Usadas en el entrenamiento						No usadas anteriormente			
		1	2	3	4	5	6	7	8	9	10
1	0.05	0.0492	0.0504	0.0512	0.0516	0.0497	0.0501	0.0522	0.0767	0.0167	0.0374
2	0.05	0.0494	0.0495	0.0508	0.0495	0.0504	0.0505	0.0412	0.0648	0.0807	0.1353
3	0.05	0.0494	0.0497	0.0491	0.0497	0.0508	0.0496	0.1499	0.2784	0.0405	0.0265
4	0.05	0.0502	0.0526	0.0118	0.0511	0.0505	0.0468	0.0154	0.0114	0.026	0.0279
5	0.05	0.05	0.0508	0.0492	0.0505	0.0502	0.0494	0.0419	0.0722	0.083	0.0621
6	0.05	0.0502	0.0481	0.0526	0.0486	0.0511	0.0514	0.0263	0.0705	0.0442	0.0719
7	0.05	0.0503	0.0505	0.0477	0.0522	0.0489	0.0487	0.0082	0.0503	0.1017	0.151
8	**0.95**	**0.9501**	**0.9507**	**0.9481**	**0.9504**	**0.9488**	**0.9487**	**0.881**	**0.8733**	**0.9323**	**0.9631**
9	0.05	0.0506	0.0501	0.0476	0.051	0.0495	0.0482	0.0969	0.2611	0.1121	0.0807
10	0.05	0.0489	0.0506	0.0499	0.0504	0.0505	0.0498	0.0614	0.06	0.1604	0.0263

Tabla XI. Salida para imágenes con el número nueve

Salida	Valor deseado	Tipo de imagen									
		Usadas en el entrenamiento						No usadas anteriormente			
		1	2	3	4	5	6	7	8	9	10
1	0.05	0.0491	0.0504	0.0509	0.0496	0.0498	0.0504	0.0216	0.038	0.0108	0.0209
2	0.05	0.0503	0.0502	0.0502	0.0492	0.051	0.0495	0.1425	0.023	0.0803	0.061
3	0.05	0.05	0.0506	0.0499	0.0493	0.0509	0.0497	0.1135	0.2205	0.1962	0.1072
4	0.05	0.0491	0.0501	0.0504	0.0492	0.0513	0.0502	0.0843	0.0298	0.0051	0.0081
5	0.05	0.051	0.0503	0.0497	0.0497	0.0503	0.0505	0.01	0.0152	0.0352	0.0339
6	0.05	0.0503	0.0506	0.0495	0.0505	0.0492	0.0511	0.0371	0.0328	0.0046	0.0169
7	0.05	0.05	0.0493	0.0499	0.0485	0.0516	0.0499	0.0377	0.048	0.111	0.1079
8	0.05	0.0516	0.0504	0.0501	0.0496	0.0502	0.0506	0.0329	0.1983	0.0561	0.1215
9	**0.95**	**0.949**	**0.9493**	**0.9498**	**0.9505**	**0.9501**	**0.9494**	**0.9137**	**0.7413**	**0.6948**	**0.8535**
10	0.05	0.0496	0.0504	0.0505	0.0505	0.0496	0.05	0.0639	0.0447	0.0505	0.0149

Tabla XII. Salida para imágenes con el número cero

Salida	Valor deseado	Tipo de imagen									
		Usadas en el entrenamiento						No usadas anteriormente			
		1	2	3	4	5	6	7	8	9	10
1	0.05	0.0499	0.05	0.0515	0.0487	0.0504	0.0501	0.0521	0.0756	0.1051	0.1041
2	0.05	0.0498	0.0501	0.0489	0.0488	0.0517	0.0501	0.0821	0.0814	0.054	0.0415
3	0.05	0.0494	0.0491	0.0506	0.0518	0.0497	0.0495	0.052	0.1068	0.0236	0.0044
4	0.05	0.0494	0.0505	0.048	0.0511	0.0507	0.0501	0.0311	0.0307	0.0341	0.0655
5	0.05	0.05	0.0494	0.0482	0.0497	0.0521	0.0505	0.0724	0.0872	0.0273	0.0943
6	0.05	0.0496	0.0499	0.0505	0.0506	0.0492	0.0502	0.104	0.129	0.0862	0.048
7	0.05	0.05	0.0513	0.049	0.0497	0.05	0.051	0.0471	0.0477	0.0555	0.0116
8	0.05	0.0494	0.0503	0.0484	0.0494	0.0514	0.0503	0.0331	0.068	0.032	0.0585
9	0.05	0.0503	0.0503	0.0499	0.0499	0.049	0.0512	0.0519	0.1023	0.0581	0.0473
10	**0.95**	**0.9508**	**0.9504**	**0.9505**	**0.9501**	**0.9485**	**0.9496**	**0.9456**	**0.9063**	**0.8863**	**0.9365**

APÉNDICE B

A continuación se presenta el programa realizado en Matlab para el ejemplo del capítulo 4.

```
%EjemploPerceptronMulticapa.m
% Ejemplo de implementación Perceptrón Multicapa
% Tesis José Francisco Castro García

figuras{1,1}=0;    % esta estructura de datos contendrá todos las imágenes
                   % que serán usadas
lectura=0;         % almacena temporalmente cada imagen

for tipo=1:1:10    % Se leen todas las imágenes y se guardan en la
                   % estructura figuras
   for letra=1:1:10
      lectura=imread(strcat('D:\Figuras\',int2str(tipo),'-',int2str(letra),'.bmp'));
                                                          % Se lee la imagen
      imaglec=double(imresize(im2bw(lectura),0.5));
            % se convierte a blanco y negro, se cambian sus dimensiones a 32x32
      figuras{tipo,letra}=reshape(imaglec',1024,1);
             % se colocan todos los píxeles en un arreglo unidimensional
      letra=letra+1;
   end
end

ent=figuras{1,1};     % En el arreglo ent se colocan todas las imágenes
                      % usadas como ejemplos en el entrenamiento

for tipo=1:1:6
   for letra=1:1:10
      if((tipo~=1)||(letra~=1))
         ent=horzcat(ent,figuras{tipo,letra});
      end
   end
end

sal=[eye(10) eye(10) eye(10) eye(10) eye(10) eye(10)];

sal=0.9*sal+0.05;          % Se construye el arreglo usado como ejemplo
                           % de salida en el entrenamiento
```

```
rango=minmax(ent);        % Se determinan los rangos de cada entrada
                          % del Perceptrón multicapa

red1=newff(rango,[100 50 10],{'logsig','logsig','logsig'},'traingdx');
                                        % Se crea el Perceptrón multicapa

red1.trainParam.epochs=5000;   % Se asigna el numero máximo de iteraciones
red1.trainParam.goal=0.00001;  % Se asigna el error máximo permitido
red1.trainParam.show=100;      % En la pantalla se presentaran los resultados
                               % cada 100 iteraciones

red1=train(red1,ent,sal);           % Se entrena el Perceptrón multicapa

                    % Ahora se prueban todas las imágenes

prue=figuras{1,1};          % Se colocan todas las imágenes en el arreglo prue
for tipo=1:1:10
   for letra=1:1:10
      if((tipo~=1)||(letra~=1))
         prue=horzcat(prue,figuras{tipo,letra});
      end
   end
end

resultados=sim(red1,prue);   % Se simula la red con todas las imágenes

Reconocidos=[];      % Se escriben los números reconocidos en este arreglo
for i=1:1:100
   Reconocidos=[Reconocidos
find(resultados(1:10,i)==max(resultados(1:10,i)))];
end                       % El numero reconocido se asigna de acuerdo con la
                          % neurona de salida que tenga el valor máximo
```