UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
ARAGÓN

"APLICACIÓN DE REDES NEURONALES PARA EL RECONOCIMIENTO DE FIGURAS GEOMÉTRICAS"

T E S I S

QUE PARA OBTENER EL TÍTULO DE :

INGENIERO EN COMPUTACIÓN

P R E S E N T A :

JOSÉ ARTURO SAENZ RODRÍGUEZ

ASESOR:

ING. ARTURO OCAMPO ALVÁREZ

MÉXICO 2004

Agradecimientos:

Quiero agradecer, a mis padres que aportaron en mi vida la base, sin la cual ninguna persona podría avanzar por el camino de la vida, a mis profesores que con sus enseñanzas, lograron establecer las dudas necesarias para despertar en mi el hambre por el conocimiento; a mis hermanos, que sin su apoyo y respaldo, tal vez este trabajo no habría concluido; a los amigos, que en los momentos difíciles, nunca se ausentaron de mi vida y con sus consejos me ayudaron a salvar los tropiezos que se me presentaban; pero por sobre todas las cosas, a mi esposa Angélica y mi hija Daniela, las cuales me infunden todo el coraje y la fuerza necesaria para avanzar día con día, ya que en ellas encuentro la confianza, el apoyo y la comprensión que, como seres humanos, sentimos que nos hace falta en los momentos difíciles, y que aportan la sabiduría necesaria; que me ayuda a esclarecer las dudas, que me generan dichas situaciones, en otras palabras, proporcionan lo mas importante y valioso que puede tener un hombre en esta vida, el amor puro y sincero que solo una familia puede brindar.

Gracias por estar aquí en este momento tan importante en mi vida.

"Conocer para pensar, pensar para dudar, dudar para saber" (Juanan Urkijo)

INTRODUCCIÓN.

La necesidad de buscar alternativas más confiables en el proceso de reconocimiento de patrones, nos conduce a la investigación de nuevas tecnologías enfocadas a este fin, una de ellas es la que actualmente estamos analizando. El proceso de reconocimiento de patrones visuales con redes neuronales, nos muestra un enfoque diferente al actual, ya que este modelo es mas parecido al que por millones de años a utilizado la naturaleza en todos los seres existentes, y en especial, en el hombre marca uno de los principales detonantes en el proceso evolutivo con respecto al resto de las especies, ya que mediante la forma que el hombre interpreta el mundo que lo rodea, piensa en la manera de adaptarse o adaptarlo a sus necesidades, este proceso es largo y detallado, ya que abarca desde el estudio del funcionamiento del sistema nervioso central, principalmente la neurona, la manera en que llegan las señales y son interpretadas por ella, hasta el proceso de adaptación de las nuevas tecnologías a la forma de trabajo de estos sistemas biológicos, tomando en cuenta la manera de introducir los patrones a los sistemas desarrollados, ya sea mediante reconocedores visuales, analizadores de sonidos, etc., utilizando para este fin a la robótica y a la ingeniería, de ello dependerá el tipo de red neuronal que queramos implementar, tomando en cuenta que dentro de un organismo biológico, existen diferentes tipos de redes neuronales dedicadas cada una a una tarea especifica, ya sea en software, utilizando un modelo dividido en tres módulos; uno encargado de procesar el patrón, el segundo que se encargara de procesar dicho patrón para su posterior análisis con un software especializado y por ultimo el de interpretación de los datos obtenidos y la entrega final de resultados, o en un modelo realizado en hardware en el cual omitiremos la parte de pre-procesamiento, realizándolo de manera automática, esto implica la utilización de hardware mas costoso que el anterior y una especialización de los dispositivos generados.

Observando estas últimas características se decidió trabajar bajo el concepto de Red neuronal en Software, ya que este es muy útil para fines de investigación y pruebas de desempeño, además de ofrecer la posibilidad de mejorar dichas tecnologías y una vez mejoradas su posterior programación en el modelo de Red neuronal en hardware.

El presente trabajo se desglosara en tres capítulos, en el primero de ellos hablaremos del funcionamiento de los componentes de las redes neuronales biológicas, su forma de comunicación, el modelo de aprendizaje biológico y posteriormente, esto lo llevaremos al mundo artificial, tratando de explicar la manera de emular dicho sistemas dentro del mundo computacional, explicaremos las principales teorías de redes neuronales artificiales, así como su definición matemática. Dentro del segundo capitulo abordaremos el tema de la robótica, esto con el fin de sentar la bases que nos llevaran al diseño de un prototipo que mejor se adecue a las necesidades del experimento, hablaremos de la teoría de visón de maquina, la cual nos brindara los fundamentos en los cuales nos apoyaremos en la adquisición de las imágenes necesarias en un robot polar construido, todo esto basado en los conceptos anteriores, por ultimo en el tercer

capitulo se hará la unión de los conceptos asimilados en los capítulos anteriores para la generación de un análisis de resultados para verificar que el prototipo realizado funcione y entregue resultados que se puedan garantizarse como confiables.

Finalmente encontraremos una serie de anexos dentro de los cuales hablaremos de diferentes tipos de Redes Neuronales y sus principales aplicaciones ofreciendo para ello una serie de ejemplos, así como su descripción teórica.

OBJETIVOS.

OBJETIVO GENERAL.

Diseñar un prototipo capaz de adquirir una imagen y procesarla para su reconocimiento utilizando redes neuronales.

OBJETIVOS ESPECIFICOS.

- Establecer las bases teóricas del funcionamiento de una red neuronal biológica para aplicarlo en un ambiente artificial.
- Fundamentar los principios básicos de la robótica para el diseño y construcción de un prototipo capaz de adquirir un patrón visual.
- Diseñar una interfaz la cual se encargue de procesar los datos obtenidos por el prototipo y enviarlos a la computadora.
- Diseñar el software encargado de manejar la información recibida de la interfaz para generar la base de conocimientos de la red neuronal.
- Implementar la red neuronal que se encargara de interpretar y reconocer el patrón visual basado en la información almacenada.
- Interpretación de los resultados validando el grado de aprendizaje del sistema propuesto.

Planteamiento del problema y delimitación del tema.

De acuerdo con lo expuesto teóricamente en las materias de inteligencia artificial, procesamiento digital de señales y reconocimiento de patrones, se ve la necesidad de implementar un sistema en el cual se apliquen estas teorías para comprobar y entender estas nuevas técnicas de reconocimiento de patrones.

Se propone el diseño e implementación de un prototipo capaz de captar la intensidad luminosa de una figura realizando un barrido mediante un robot de tipo polar, enviando la información recolectada a una tarjeta de adquisición de datos la cual cumple la función de interfaz entre las señales del mundo real y el mundo artificial, generando los archivos necesarios para el entrenamiento de una red neuronal.

Tomando en cuenta en factor costo beneficio se decidió utilizar materiales de reciclaje (flopy de 5 ¼, tarjeta controladora HC11, PC estándar, un fotodiodo, una lente de aumento, etc) y software de uso educativo (Matlab, NeuralWork). Considerando que los conocimientos adquiridos pueden sentar las bases para un desarrollo a nivel industrial enfocado al control de calidad.

CAPITULO I

FUNDAMENTOS DE REDES NEURONALES Y ROBÓTICA

1.1 CARACTERÍSTICAS PRINCIPALES DE LAS REDES NEURONALES

1.1.1 INTRODUCCIÓN A LAS REDES NEURONALES

Resulta irónico pensar que máquinas de computo capaces de realizar 100 millones de operaciones en coma flotante por segundo, no sean capaces de entender el significado de las formas visuales o de distinguir entre distintas clases de objetos. Los sistemas de computación secuencial, son exitosos en la resolución de problemas matemáticos o científicos, en la creación, manipulación y mantenimiento de bases de datos, en comunicaciones electrónicas, en el procesamiento de textos, gráficos y auto edición, incluso en funciones de control de electrodomésticos, haciéndolos más eficientes y fáciles de usar, pero definitivamente tienen una gran incapacidad para interpretar el mundo.

Esta dificultad de los sistemas de computo que trabajan bajo la filosofía de los sistemas secuenciales, desarrollados por Von Neuman, ha hecho que un gran número de investigadores centre su atención en el desarrollo de nuevos sistemas de tratamiento de la información, que permitan solucionar problemas cotidianos, tal como lo hace el cerebro humano; este órgano biológico cuenta con varias características deseables para cualquier sistema de procesamiento digital, tales como:

1. Es robusto y tolerante a fallas, diariamente mueren neuronas sin afectar su desempeño.
2. Es flexible, se ajusta a nuevos ambientes por aprendizaje, no hay que programarlo.
3. Puede manejar información difusa, con ruido o inconsistente.
4. Es altamente paralelo
5. Es pequeño, compacto y consume poca energía.

El cerebro humano constituye una computadora muy notable, es capaz de interpretar información imprecisa suministrada por los sentidos a un ritmo increíblemente veloz. Logra discernir un susurro en una sala ruidosa, un rostro en un callejón mal iluminado y leer entre líneas un discurso; lo más impresionante de todo, es que el cerebro aprende sin instrucciones explícitas de ninguna clase, a crear las representaciones internas que hacen posibles estas habilidades.
Basados en la eficiencia de los procesos llevados a cabo por el cerebro, e inspirados en su funcionamiento, varios investigadores han desarrollado desde hace más de 30 años la teoría de las Redes Neuronales Artificiales (RNA), las cuales emulan las redes neuronales biológicas, y que se han utilizado para aprender estrategias de solución basadas en ejemplos de comportamiento típico de patrones; estos sistemas no requieren que la tarea a ejecutar se programe, ellos generalizan y aprenden de la experiencia.

La teoría de las RNA ha brindado una alternativa a la computación clásica, para aquellos problemas, en los cuales los métodos tradicionales no han entregado resultados muy convincentes, o poco convenientes. Las aplicaciones más exitosas de las RNA son:

1. Procesamiento de imágenes y de voz
2. Reconocimiento de patrones
3. Planeamiento
4. Interfaces adaptivas para sistemas Hombre/máquina
5. Predicción
6. Control y optimización
7. Filtrado de señales

Los sistemas de computo tradicional procesan la información en forma secuencial; un computador serial 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 serial 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, este es el estado de la red para el cual se logra equilibrio. El conocimiento de una red neuronal no se almacena en instrucciones, el poder de la red está en su topología y en los valores de las conexiones (pesos) entre neuronas.
Las RNA son una teoría que aún esta en proceso de desarrollo, su verdadera potencialidad no se ha alcanzado todavía; aunque los investigadores han desarrollado potentes algoritmos de aprendizaje de gran valor práctico, las representaciones y procedimientos de que se sirve el cerebro, son aún desconocidas. Tarde o temprano los estudios computacionales del aprendizaje con RNA acabarán por converger a los métodos descubiertos por evolución, cuando eso suceda, muchos datos empíricos concernientes al cerebro comenzarán súbitamente a adquirir sentido y se tornarán factibles muchas aplicaciones desconocidas de las redes neuronales.

1.1.2 FUNCIONAMIENTO DE UNA NEURONA BIOLOGICA

El cerebro consta de un gran número (aproximadamente 10^{11}) de elementos altamente interconectados (aproximadamente 10^4 conexiones por elemento), llamados neuronas. Estas neuronas tienen tres componentes principales, las dendritas, el cuerpo de la célula o soma, y el axón. Las dendritas, son el árbol receptor de la red, son como fibras nerviosas que cargan de señales eléctricas el cuerpo de la célula. El cuerpo de la célula, realiza la suma de esas señales de entrada. El axón es una fibra larga que lleva la señal desde el cuerpo de la

célula hacia otras neuronas. El punto de contacto entre un axón de una célula y una dendrita de otra célula es llamado sinápsis, la longitud de la sinápsis es determinada por la complejidad del proceso químico que estabiliza la función de la red neuronal. Un esquema simplificado de la interconexión de dos neuronas biológicas se observa en la figura 1.2.1

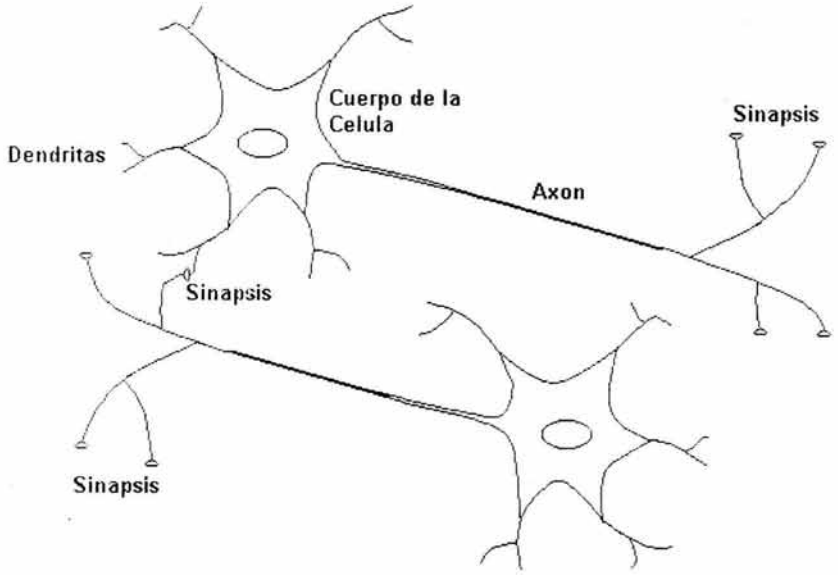


Figura 1.2.1 Neuronas Biológicas

Algunas de las estructuras neuronales son determinadas en el nacimiento, otra parte es desarrollada a través del aprendizaje, proceso en que nuevas conexiones neuronales son realizadas y otras se pierden por completo. El desarrollo neurológico se hace crítico durante los primeros años de vida, por ejemplo está demostrado que si a un cachorro de gato, se le impide usar uno de sus ojos durante un periodo corto de tiempo, el nunca desarrollara una visión normal en ese ojo.

Las estructuras neuronales continúan cambiando durante toda la vida, estos cambios consisten en el refuerzo o debilitamiento de las uniones sinápticas; por ejemplo se cree que nuevas memorias son formadas por la modificación de esta intensidad entre sinápsis, así el proceso de recordar el rostro de un nuevo amigo, consiste en alterar varias sinápsis.

Como consecuencia de los primeros estudios sobre la base neural de los sistemas mnémicos (relacionados con la memoria), se creía que el almacenamiento de la memoria asociativa, tanto implícita como explícita, requerían de un circuito neuronal muy complejo. Entre quienes comenzaron a oponerse a este enfoque se hallaba Donald O. Hebb, profesor de la universidad de Milner; Hebb sugirió que el aprendizaje asociativo podría ser producido por

un mecanismo celular sencillo y propuso que las asociaciones podrían formarse por una actividad neuronal coincidente: "Cuando un axón de la célula A excita la célula B y participa en su activación, se produce algún proceso de desarrollo o cambio metabólico en una o en ambas células, de suerte que la eficacia de A, como célula excitadora de B, se intensifica". Según la regla Hebbiana de aprendizaje, el que coincida la actividad de las neuronas presinápticas (suministran el impulso de entrada) con la de las postsinápticas (reciben el impulso) es muy importante para que se refuerce la conexión entre ellas, este mecanismo es llamado pre-postasociativo, del cual puede observarse un ejemplo en la figura 1.2.2

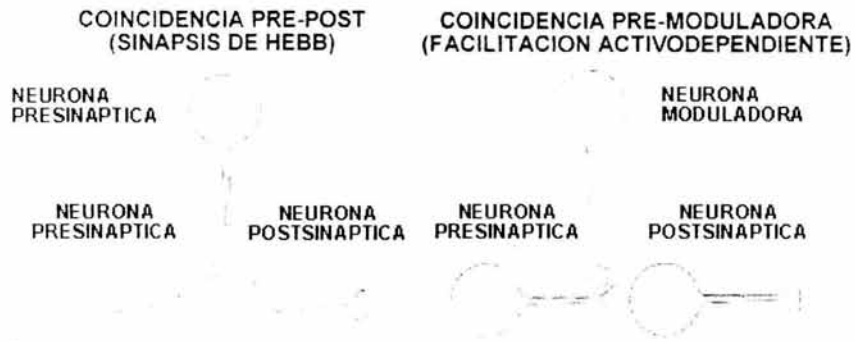


Figura 1.2.2 Cambios asociativos de las fuerzas sinápticas durante el aprendizaje

Todas las neuronas conducen la información de forma similar, esta viaja a lo largo de axones en breves impulsos eléctricos, denominados potenciales de acción; los potenciales de acción que alcanzan una amplitud máxima de unos 100 mV y duran 1 ms, son resultado del desplazamiento a través de la membrana celular de iones de sodio dotados de carga positiva, que pasan desde el fluido extracelular hasta el citoplasma intracelular; la concentración extracelular de sodio supera enormemente la concentración intracelular.

La membrana en reposo mantiene un gradiente de potencial eléctrico de -70mv, el signo negativo se debe a que el citoplasma intracelular está cargado negativamente con respecto al exterior; los iones de sodio no atraviesan con facilidad la membrana en reposo, los estímulos físicos o químicos que reducen el gradiente de potencial, o que despolaricen la membrana, aumentan su permeabilidad al sodio y el flujo de este ion hacia el exterior acentúa la despolarización de la membrana, con lo que la permeabilidad al sodio se incrementa más aún.

Alcanzado un potencial crítico denominado "umbral", la realimentación positiva produce un efecto regenerativo que obliga al potencial de membrana a cambiar de signo. Es decir, el interior de la célula se torna positivo con respecto al exterior, al cabo de 1 ms, la permeabilidad del sodio decae y el potencial de membrana retorna a -70mv, su valor de reposo. Tras cada explosión de

actividad iónica, el mecanismo de permeabilidad del sodio se mantiene refractario durante algunos milisegundos; la tasa de generación de potenciales de acción queda así limitada a unos 200 impulsos por segundo, o menos.

Aunque los axones puedan parecer hilos conductores aislados, no conducen los impulsos eléctricos de igual forma, como hilos eléctricos no serían muy valiosos, pues su resistencia a lo largo del eje es demasiado grande y a resistencia de la membrana demasiado baja; la carga positiva inyectada en el axón durante el potencial de acción queda disipada uno o dos milímetros más adelante, para que la señal recorra varios centímetros es preciso regenerar frecuentemente el potencial de acción a lo largo del camino la necesidad de reforzar repetidamente esta corriente eléctrica limita a unos 100 metros por segundo la velocidad máxima de viaje de los impulsos, tal velocidad es inferior a la millonésima de la velocidad de una señal eléctrica por un hilo de cobre.

Los potenciales de acción, son señales de baja frecuencia conducidas en forma muy lenta, estos no pueden saltar de una célula a otra, la comunicación entre neuronas viene siempre mediada por transmisores químicos que son liberados en las sinápsis. Un ejemplo de comunicación entre neuronas y del proceso químico de la liberación de neurotransmisores se ilustra en la figura 1.2.3.

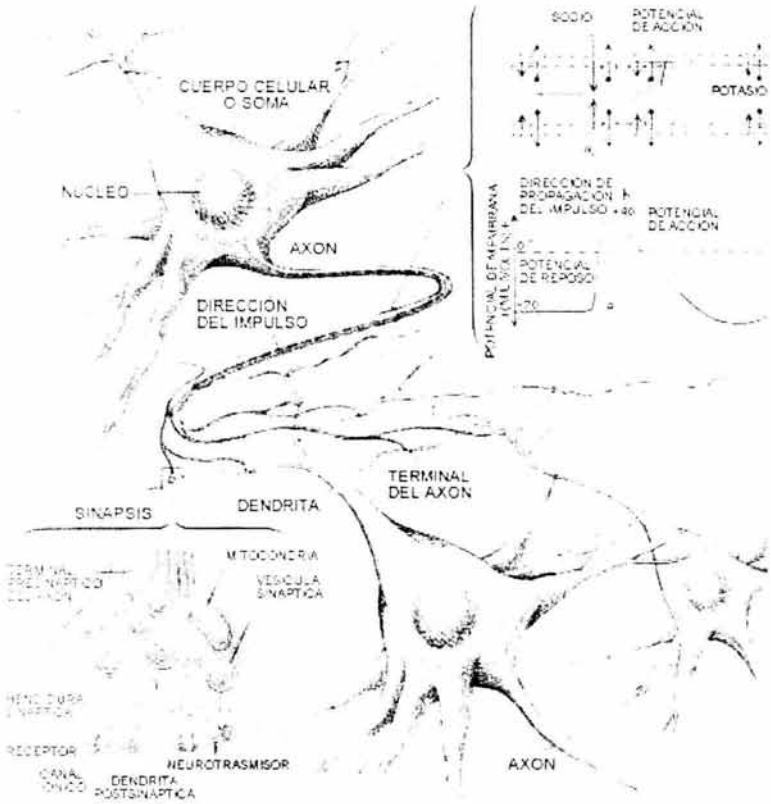


Figura 1.2.3 Comunicación entre neuronas

Cuando un potencial de acción llega al terminal de un axón son liberados transmisores alojados en diminutas vesículas, que después son vertidos en una hendidura de unos 20 nanómetros de anchura que separa la membrana presináptica de la postsináptica; durante el apogeo del potencial de acción, penetran iones de calcio en el terminal nervioso, su movimiento constituye la señal determinante de la exocitosis sincronizada, esto es la liberación coordinada de moléculas neurotransmisoras. En cuanto son liberados, los neurotransmisores se enlazan con receptores postsinápticos, instando el cambio de la permeabilidad de la membrana.

Cuando el desplazamiento de carga hace que la membrana se aproxime al umbral de generación de potenciales de acción, se produce un efecto excitador y cuando la membrana resulta estabilizada en la vecindad el valor de reposo se produce un efecto inhibidor. Cada sinápsis produce sólo un pequeño efecto, para determinar la intensidad (frecuencia de los potenciales de acción) de la respuesta cada neurona ha de integrar continuamente hasta unas 1000 señales sinápticas, que se suman en el soma o cuerpo de la célula.

En algunas neuronas los impulsos se inician en la unión entre el axón y el soma, y luego se transmiten a lo largo del axón a otras células nerviosas. Cuando el axón está cerca de sus células destino, se divide en muchas ramificaciones que forman sinápsis con el soma o axones de otras células. Las sinápsis pueden ser excitatorias o inhibitorias según el neurotransmisor que se libere, cada neurona recibe de 10.000 a 100.000 sinápsis y su axón realiza una cantidad similar de sinápsis.

Las sinápsis se clasifican según su posición en la superficie de la neurona receptora en tres tipos: axo-somática, axo-dendrítica, axo-axónica. Los fenómenos que ocurren en la sinápsis son de naturaleza química, pero tienen efectos eléctricos laterales que se pueden medir.

En la figura 1.2.4 se visualiza el proceso químico de una sinápsis y los diferentes elementos que hacen parte del proceso tanto en la neurona presináptica, como en la postsináptica.

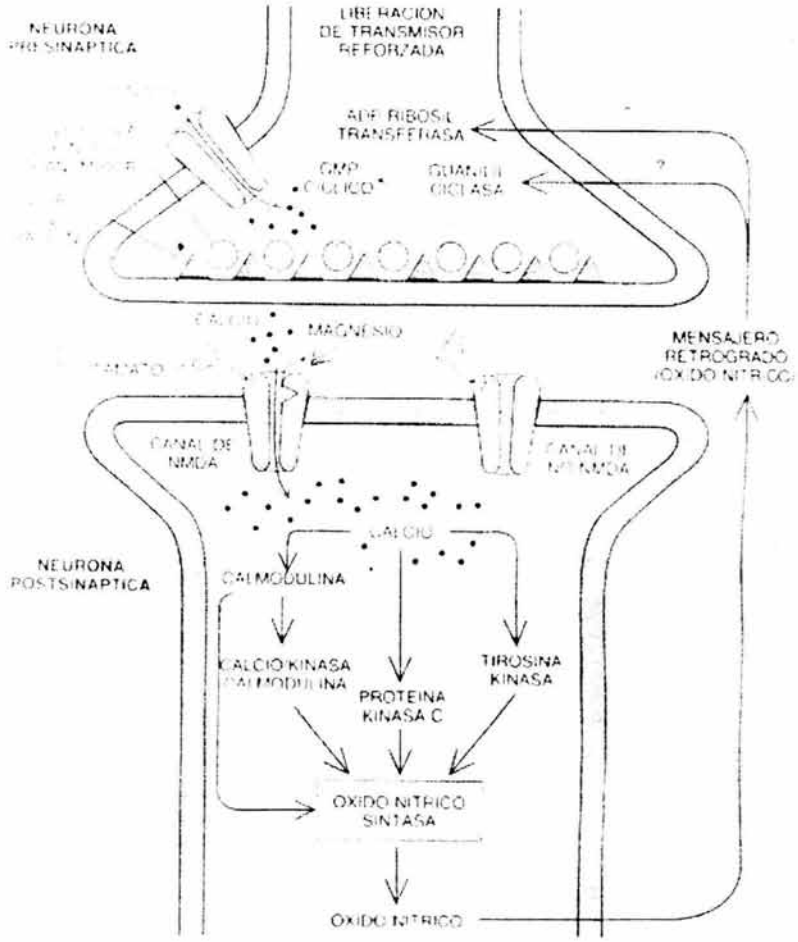
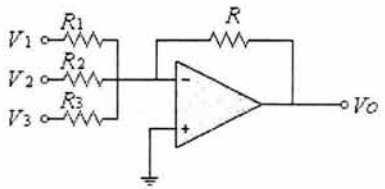


Figura 1.2.4 Proceso químico de una sinápsis

Las RNA no alcanzan la complejidad del cerebro, sin embargo hay dos aspectos similares entre redes biológicas y artificiales, primero los bloques de construcción de ambas redes son sencillos elementos computacionales (aunque las RNA son mucho más simples que las biológicas) altamente interconectados; segundo, las conexiones entre neuronas determinan la función de la red.

1.1.3 CARACTERISTICAS DE UNA RED NEURONAL ARTIFICIAL

El modelo de una neurona artificial es una imitación del proceso de una neurona biológica, puede también asemejarse a un sumador hecho con un amplificador operacional tal como se ve en la figura 1.3.1.



$$Vo = -\sum_{i=1}^{n} \frac{R}{Ri} Vi$$

Figura 1.3.1 Neurona Artificial

Existen varias formas de nombrar una neurona artificial, es conocida como nodo, neuronodo, celda, unidad o elemento de procesamiento (PE); En la figura 1.3.1 se observa un PE en forma general y su similitud con una neurona biológica

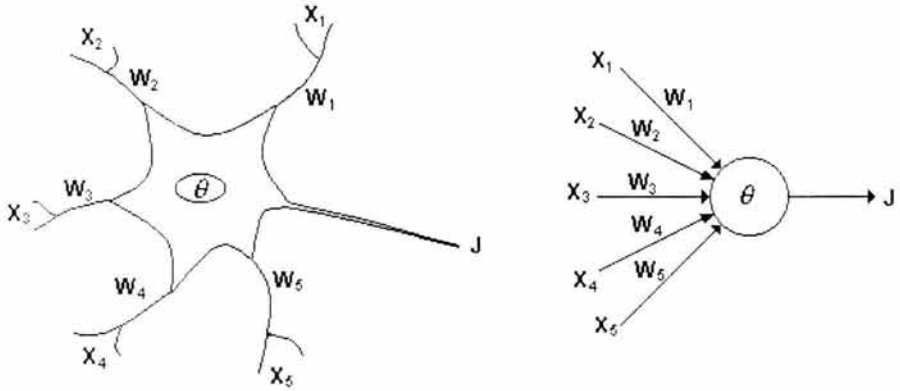


Figura 1.3.2 De la neurona biológica a la neurona artificial

De la observación detallada del proceso biológico se han hallado los siguientes análogos con el sistema artificial:

- Las entradas X_i representan las señales que provienen de otras neuronas y que son capturadas por las dendritas.
- Los pesos W_i son la intensidad de la sinápsis que conecta dos neuronas; tanto X_i como W_i son valores reales.

- f es la función umbral que la neurona debe sobrepasar para activarse; este proceso ocurre biológicamente en el cuerpo de la célula.

Las señales de entrada a una neurona artificial X_1, X_2,.., X_n son variables continuas en lugar de pulsos discretos, como se presentan en una neurona biológica. Cada señal de entrada pasa a través de una ganancia o peso, llamado peso sináptico o fortaleza de la conexión cuya función es análoga a la de la función sináptica de la neurona biológica. Los pesos pueden ser positivos (excitatorios), o negativos (inhibitorios), el nodo sumatorio acumula todas las señales de entradas multiplicadas por los pesos o ponderadas y las pasa a la salida a través de una función umbral o función de transferencia. La entrada neta a cada unidad puede escribirse de la siguiente manera

$$neta_i = \sum_{i=1}^{n} W_i X_i = \vec{X}\,\vec{W} \qquad (1.3.1)$$

Una idea clara de este proceso se muestra en la figura 1.3.3, en donde puede observarse el recorrido de un conjunto de señales que entran a la red.

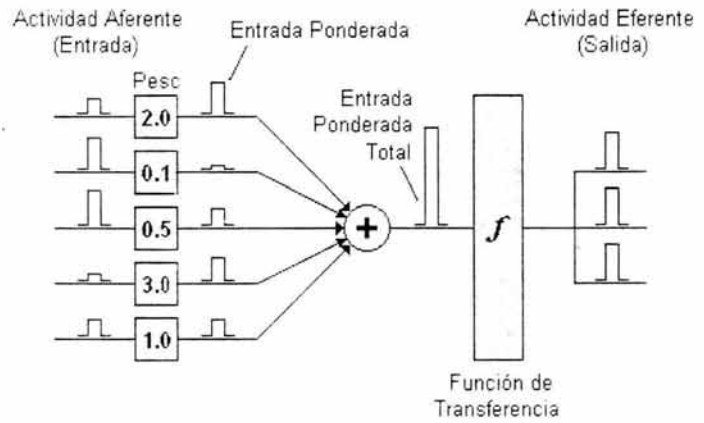


Figura 1.3.3 Proceso de una red neuronal

Una vez que se ha calculado la activación del nodo, el valor de salida equivale a

$$x_i = f_i(neta_i) \qquad (1.3.2)$$

Donde f_i representa la función de activación para esa unidad, que corresponde a la función escogida para transformar la entrada neta$_i$ en el valor de salida x_i y que depende de las características específicas de cada red.

1.1.3.1 Notación: Una notación matemática estándar no ha sido aún establecida para las redes neuronales, ya que sus aplicaciones son útiles en muchos campos, Ingeniería, Física, Sicología y Matemáticas. En este trabajo se adoptó

la siguiente convención para identificar las variables, de manera que fuera compatibles con las diferentes áreas, siendo lo más sencilla posible:

- Valores escalares: se representarán por medio de letra minúscula itálica
- Vectores: se representarán con letra itálica minúscula en negrilla.
- Matrices: se representarán con letra mayúscula itálica en negrilla.

Para redes multicapa, los parámetros adoptaran la siguiente forma:

$W^{c}_{s^c,s^c}$

Donde c, es el número de la capa a la que corresponde dicho peso, y s representa las neuronas que participan en proceso.

Así $W^{2}_{1,1}$representa el peso de la segunda capa que comunica la primera neurona de dicha capa con la primera neurona de la primera capa. De igual manera el peso que representa la conexión desde la última neurona de la capa dos a la última neurona de la capa uno estará representado por:

$W^{2}_{s^2,s^1}$

Esta convención es adoptada para todos los parámetros de la red.

1.1.3.2 Funciones de Transferencia: Un modelo más académico que facilita el estudio de una neurona, puede visualizarse en la figura 1.3.4

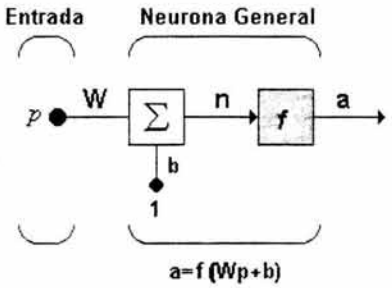


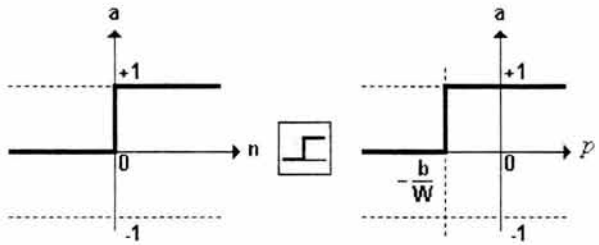
Figura 1.3.4 Neurona de una sola entrada

Las entradas a la red serán ahora presentadas en el vector p, que para el caso de una sola neurona contiene solo un elemento, w sigue representando los pesos y la nueva entrada b es una ganancia que refuerza la salida del sumador n, la cual es la salida neta de la red; la salida total está determinada por la función de transferencia , la cual puede ser una función lineal o no lineal de n, y que es escogida dependiendo de las especificaciones del problema que la neurona tenga que resolver; aunque las RNA se inspiren en modelos biológicos no existe ninguna limitación para realizar modificaciones en las funciones de salida, así que se encontrarán modelos artificiales que nada tienen que ver con las características del sistema biológico.

1.1.3.2.1 Función de Transferencia Limitador Fuerte (*hardlim*):

La figura 1.3.5, muestra como esta función de transferencia acerca la salida de la red a cero, si el argumento de la función es menor que cero y la lleva a uno si este argumento es mayor que uno. Esta función crea neuronas que clasifican las entradas en dos categorias diferentes, caracteristica que le permite ser empleada en la red tipo Perceptrón

$$a = \begin{Bmatrix} 1 \ si \ n \geq 0 \\ 0 \ si \ n < 0 \end{Bmatrix} \quad (1.3.3)$$



a= *hardlim* (n) — Función de Transferencia Limitador Fuerte

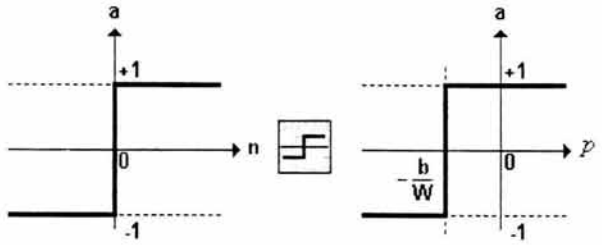
a= *hardlim* (Wp+b) — Entrada a una Neurona Limitador Fuerte

Figura 1.3.5 Función de transferencia Hardlim

El icono para la función Hardlim reemplazara a la letra *f* en la expresión general, cuando se utilice la función Hardlim.

Una modificación de esta función puede verse en la figura 1.3.6, la que representa la función de transferencia Hardlims que restringe el espacio de salida a valores entre 1 y –1.

$$a = \begin{Bmatrix} 1 \ si \ n \geq 0 \\ -1 \ si \ n < 0 \end{Bmatrix} \quad (1.3.4)$$



a= *hardlims* (n)

Función de Transferencia
Limitador Fuerte Simetrica

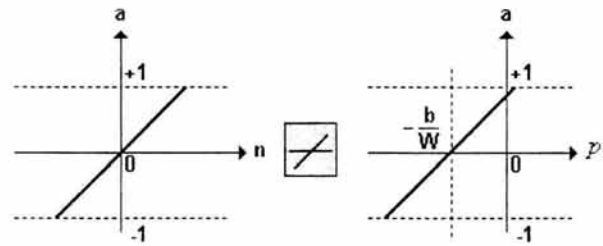
a= *hardlims*(Wp+b)

Entrada a una Neurona
Limitador Fuerte Simetrica

Figura 1.3.6 Función de transferencia Hardlims

1.1.3.2.2 Función de transferencia lineal (purelin): La salida de una función de transferencia lineal es igual a su entrada,

$$a = n \quad (1.3.5)$$



a= *purelin* (n)

Función de Transferencia
Lineal

a= *purelin* (Wp+b)

Entrada a una Neurona
Lineal

Figura 1.3.7 Función de transferencia lineal

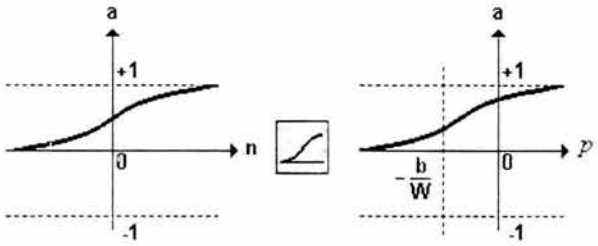
En la gráfica del lado derecho de la figura 1.3.6, puede verse la característica de la salida *a* de la red, comparada con la entrada *p*, más un valor de ganancia *b*, neuronas que emplean esta función de transferencia son utilizadas en la red tipo Adaline.

1.1.3.2.3 Función de transferencia sigmoidal (logsig): Esta función toma los valores de entrada, los cuales pueden oscilar entre mas y menos infinito, y restringe la salida a valores entre cero y uno, de acuerdo a la expresión

$$a = \frac{1}{1+e^{-n}} \quad (1.3.6)$$

Esta función es comúnmente usada en redes multicapa, como la Backpropagation, en parte porque la función logsig es diferenciable.

Figura 1.3.8 Función de transferencia sigmoidal



a= *logsig* (n)

Función de Transferencia Logaritmica Sigmoidal

a=*logsig*(Wp+b)

Entrada a una Neurona Logaritmica Sigmoidal

La tabla 1.3.1 hace una relación de las principales funciones de transferencia empleadas en el entrenamiento de redes neuronales.

Nombre	Relación Entrada /Salida	Icono	Función
Limitador Fuerte	$a = 0\ n < 0$ $a = 1\ n \geq 0$		*hardlim*
Limitador Fuerte Simétrico	$a = -1\ n < 0$ $a = +1\ n \geq 0$		*hardlims*
Lineal Positiva	$a = 0\ n < 0$ $a = n\ 0 \leq n$		*poslin*
Lineal	$a = n$		*purelin*
Lineal Saturado	$a = 0\ n < 0$ $a = n\ 0 \leq n \leq 1$ $a = 1\ n > 1$		*satlin*

Lineal Saturado Simétrico	$a = -1 \; n < -1$ $a = n \quad -1 \le n \le 1$ $a = +1 \; n > 1$	⧸	*satlins*
Sigmoidal Logaritmico	$a = \frac{1}{1+e^{-n}}$	∠	*logsig*
Tangente Sigmoidal Hiperbólica	$a = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}}$	⧸	*tansig*
Competitiva	$a = 1$ *Neurona con n max* $a = 0$ *El resto de neuronas*	C	*compet*

Tabla 1.3.1 Funciones de Transferencia

1.1.1.3.3 Topología de una Red: Típicamente una neurona tiene más de una entrada; en la figura 1.3.9 se observa una neurona con R entradas; las entradas individuales $p_1, p_2, \ldots, p_R$ son multiplicadas por los pesos correspondientes $w_{1,1}$, $w_{1,2}, \ldots w_{1,R}$ pertenecientes a la matriz de pesos ***W***.

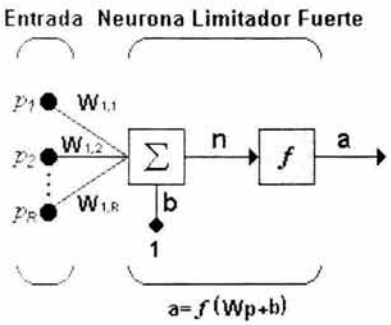


Figura 1.3.9 Neurona con múltiples entradas

La neurona tiene una ganancia *b*, la cual llega al mismo sumador al que llegan las entradas multiplicadas por los pesos, para formar la salida *n*,

$$n = w_{1,1}p_1 + w_{1,2}p_2 + \ldots + w_{1,R}p_R + b \quad (1.3.7)$$

Esta expresión puede ser escrita en forma matricial

$$n = \boldsymbol{Wp} + b \quad (1.3.8)$$

Los subíndices de la matriz de pesos representan los términos involucrados en la conexión, el primer subíndice representa la neurona destino y el segundo, representa la fuente de la señal que alimenta a la neurona. Por ejemplo, los índices de $w_{1,2}$ indican que este peso es la conexión desde la segunda entrada a la primera neurona. Esta convención se hace más útil cuando hay más de una neurona, o cuando se tiene una neurona con demasiados parámetros; en este caso la notación de la figura 1.3.9, puede resultar inapropiada y se prefiere emplear la notación abreviada representada en la figura 1.3.10

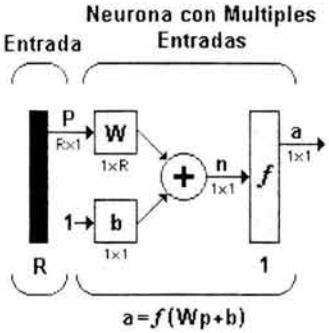


Figura 1.3.10 Neurona con múltiples entradas, notación abreviada

El vector de entrada ***p*** es representado por la barra sólida vertical a la izquierda. Las dimensiones de ***p*** son mostradas en la parte inferior de la variable como Rx1, indicando que el vector de entrada es un vector fila de R elementos. Las entradas van a la matriz de pesos ***W***, la cual tiene R columnas y solo una fila para el caso de una sola neurona. Una constante 1 entra a la neurona multiplicada por la ganancia escalar *b*. La salida de la red *a*, es en este caso un escalar, si la red tuviera más de una neurona ***a*** sería un vector.

Dentro de una red neuronal, los elementos de procesamiento se encuentran agrupados por capas, una capa es una colección de neuronas; de acuerdo a la ubicación de la capa en la RNA, esta recibe diferentes nombres

Capa de entrada: Recibe las señales de la entrada de la red, algunos autores no consideran el vector de entrada como una capa pues allí no se lleva a cabo ningún proceso.

Capas ocultas: Estas capas son aquellas que no tienen contacto con el medio exterior, sus elementos pueden tener diferentes conexiones y son estas las que determinan las diferentes topologías de la red

Capa de salida: Recibe la información de la capa oculta y transmite la respuesta al medio externo.

Una red de una sola capa con un número *S* de neuronas, se observa en la figura 1.3.11 en la cual, cada una de las R entradas es conectada a cada una de las neuronas, la matriz de pesos tiene ahora *S* filas.

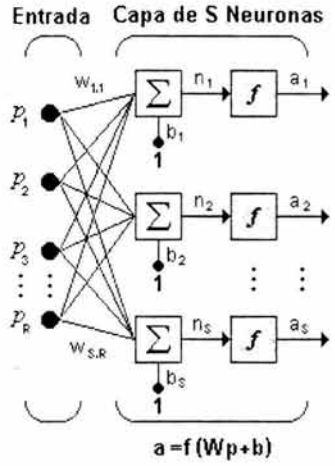


Figura 1.3.11 Capa de *S* neuronas

La capa incluye la matriz de pesos, los sumadores, el vector de ganancias, la función de transferencia y el vector de salida. Esta misma capa se observa en notación abreviada en la figura 1.3.12

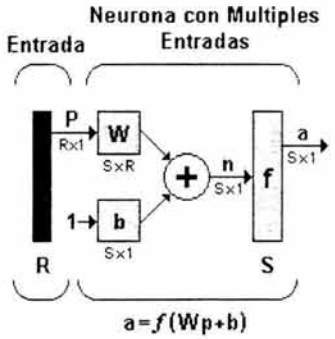


Figura 1.3.12 Capa de S neuronas con notación abreviada

En la figura 1.3.12 se han dispuesto los símbolos de las variables de tal manera que describan las características de cada una de ellas, por ejemplo la entrada a la red es el vector **p** cuya longitud R aparece en su parte inferior, **W** es la matriz de pesos con dimensiones SxR expresadas debajo del símbolo que la representa dentro de la red, **a** y **b** son vectores de longitud S el cual, como se ha dicho anteriormente representa el número de neuronas de la red.

Ahora, si se considera una red con varias capas, o red multicapa, cada capa tendrá su propia matriz de peso **W**, su propio vector de ganancias **b**, un vector de entradas netas **n**, y un vector de salida **a**. La versión completa y la versión en notación abreviada de una red de tres capas, pueden ser visualizadas en las figuras 1.3.13 y 1.3.14, respectivamente.

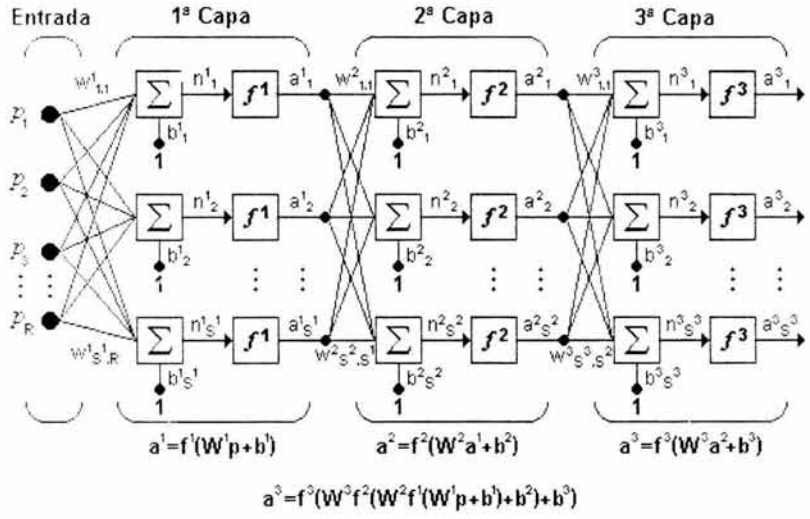


Figura 1.3.13 Red de tres capas

Para esta red se tienen R entradas, S^1 neuronas en la primera capa, S^2 neuronas en la segunda capa, las cuales pueden ser diferentes; las salidas de las capas 1 y 2 son las entradas a las capas 2 y 3 respectivamente, así la capa 2 puede ser vista como una red de una capa con $R=S^1$ entradas, $S^1=S^2$ neuronas y una matriz de pesos W^2 de dimensiones S^1xS^2

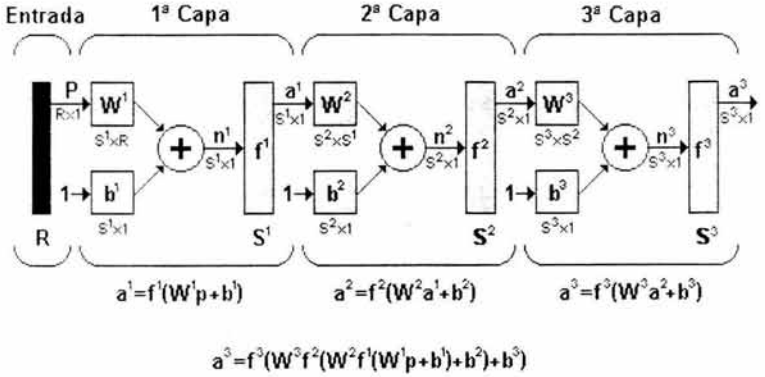


Figura 1.3.14 Red de tres capas con notación abreviada

Las redes multicapa son más poderosas que las redes de una sola capa, por ejemplo, una red de dos capas que tenga una función sigmoidal en la primera capa y una función lineal en la segunda, puede ser entrenada para aproximar muchas funciones de forma aceptable, una red de una sola capa no podría hacer esto como se verá en capítulos posteriores.

Un tipo de redes, un poco diferente a las que se han estudiado hasta el momento, son las redes recurrentes, estas contienen una realimentación hacia atrás o retroalimentación, es decir algunas de sus salidas son conectadas a sus entradas. Un tipo de red recurrente de tiempo discreto es mostrado en la figura 1.3.15.

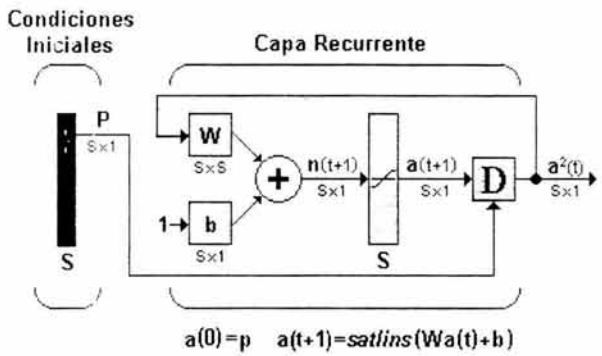


Figura 1.3.15 Redes Recurrentes

Para este tipo particular de red el vector ***p*** suple las condiciones iniciales (***a***(0)= ***p***), y la salida está determinada por:

$$\boldsymbol{a}(1) = satlins(\boldsymbol{W}\boldsymbol{a}(0) + \boldsymbol{b}), \quad \boldsymbol{a}(2) = satlins(\boldsymbol{W}\boldsymbol{a}(1) + \boldsymbol{b}) \tag{1.3.9}$$

Donde a(1) y a(2), corresponden a la salida de la red para el primer y segundo intervalo de tiempo, respectivamente. La red alcanzará su estado estable cuando la salida para un instante de tiempo sea la misma salida del instante de tiempo anterior.

Las redes recurrentes son potencialmente más poderosas que las redes con realimentación hacia delante. En este tipo de redes se introducen también dos nuevos conceptos, el bloque de retardo de la figura 1.3.16 y el bloque integrador de la figura 1.3.17

- Retardo

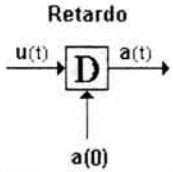


Figura 1.3.16 Bloque de retardo

$$\boldsymbol{a}(t) = \boldsymbol{u}(t-1) \quad (1.3.10)$$

La salida del bloque de retardo es el valor de entrada retrasado en un paso de tiempo, este bloque requiere que la salida sea inicializada con el valor a(0) para el tiempo t=0; a(0) se convierte en la salida de la red para el instante de tiempo inicial.

- Integrador

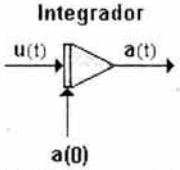


Figura 1.3.17 Bloque integrador

La salida del integrador es calculada de acuerdo a la expresión

$$\boldsymbol{a}(t) = \int_0^t \boldsymbol{u}(\tau)d\tau + \boldsymbol{a}(0) \quad (1.3.11)$$

Una red recurrente cuya implementación necesita un bloque integrador se ilustra en la figura 2.6.4.

En general las redes neuronales se pueden clasificar de diversas maneras, según su topologia, forma de aprendizaje (supervisado o no supervisado), tipos de funciones de activación, valores de entrada (binarios o continuos); un resumen de esta clasificación se observa en la figura 1.3.18

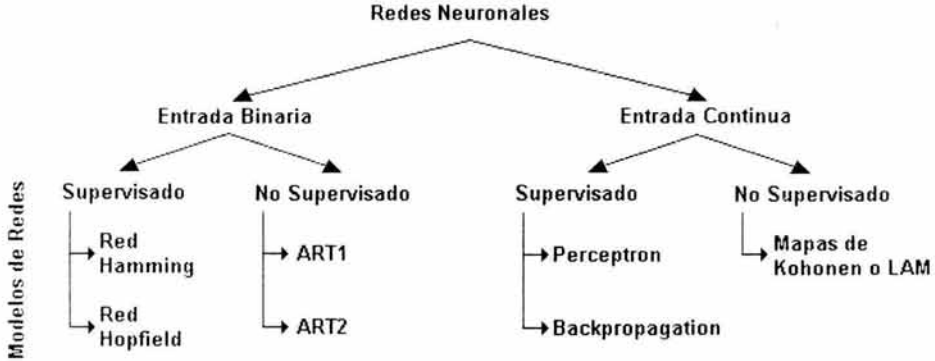


Figura 1.3.18 Clasificación de las Redes Neuronales

1.2. REDES NEURONALES

1.2.1 BACKPROPAGATION

1.2.1.1 Antecedentes: La regla de aprendizaje del Perceptrón de Rosenblatt y el algoritmo LMS de Widrow y Hoff fueron diseñados para entrenar redes de una sola capa. Como se discutió anteriormente, estas redes tienen la desventaja que solo pueden resolver problemas linealmente separables, fue esto lo que llevo al surgimiento de las redes multicapa para sobrepasar esta dificultad en las redes hasta entonces conocidas.

El primer algoritmo de entrenamiento para redes multicapa fue desarrollado por Paul Werbos en 1974, este 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 dentro de la comunidad de desarrolladores de redes neuronales. Fue solo hasta mediados de los años 80 cuando el algoritmo Backpropagation o algoritmo de propagación inversa fue redescubierto al mismo tiempo por varios investigadores, David Rumelhart, Geoffrey Hinton y Ronal Williams, David Parker y Yann Le Cun. El algoritmo se popularizó cuando fue incluido en el libro "Parallel Distributed Processing Group" por los sicólogos David Rumelhart y James McClelland. La publicación de este libro trajo consigo un auge en las investigaciones con redes neuronales, siendo la Backpropagation una de las redes más ampliamente empleadas, aun en nuestros días.

Uno de los grandes avances logrados con la Backpropagation es que esta red aprovecha la naturaleza paralela de las redes neuronales para reducir el tiempo requerido por un procesador secuencial para determinar la correspondencia entre unos patrones dados. Además el tiempo de desarrollo de cualquier sistema que se este tratando de analizar se puede reducir como consecuencia de que la red puede aprender el algoritmo correcto sin que alguien tenga que deducir por anticipado el algoritmo en cuestión.

La mayoría de los sistemas actuales de computo se han diseñado para llevar a cabo funciones matemáticas y lógicas a una velocidad que resulta asombrosamente alta para el ser humano. Sin embargo la destreza matemática no es lo que se necesita para solucionar problemas de reconocimiento de patrones en entornos ruidosos, característica que incluso dentro de un espacio de entrada relativamente pequeño, puede llegar a consumir mucho tiempo. El problema es la naturaleza secuencial del propio computador; el ciclo tomar – ejecutar de la naturaleza Von Neumann solo permite que la máquina realice una operación a la vez. En la mayoría de los casos, el tiempo que necesita la máquina para llevar a cabo cada instrucción es tan breve (típicamente una millonésima de segundo) que el tiempo necesario para un programa, así sea muy grande, es insignificante para los usuarios. Sin embargo, para aquellas aplicaciones que deban explorar un gran espacio de entrada o que intentan correlacionar todas las permutaciones posibles de un conjunto de patrones muy complejo, el tiempo de computación necesario se hace bastante grande.

Lo que se necesita es un nuevo sistema de procesamiento que sea capaz de examinar todos los patrones en paralelo. Idealmente ese sistema no tendria que ser programado explícitamente, lo que haría es adaptarse a sí mismo para aprender la relación entre un conjunto de patrones dado como ejemplo y ser capaz de aplicar la misma relación a nuevos patrones de entrada. Este sistema debe estar en capacidad de concentrarse en las características de una entrada arbitraria que se asemeje a otros patrones vistos previamente, sin que ninguna señal de ruido lo afecte. Este sistema fue el gran aporte de la red de propagación inversa, Backpropagation.

La Backpropagation es un tipo de red de aprendizaje supervisado, que emplea un ciclo propagación – adaptación de dos fases. Una vez que se ha aplicado un patrón a la entrada de la red como estímulo, este se propaga desde la primera capa a través de las capas superiores de la red, hasta generar una salida. La señal de salida se compara con la salida deseada y se calcula una señal de error para cada una de las salidas.

Las salidas de error se propagan hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de la capa oculta que contribuyen directamente a la salida. Sin embargo las neuronas de la capa oculta solo reciben una fracción de la señal total del error, basándose aproximadamente en la contribución relativa que haya aportado cada neurona a la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido una señal de error que describa su contribución relativa al error total. Basándose en la señal de error percibida, se actualizan los pesos de conexión de cada neurona, para hacer que la red converja hacia un estado que permita clasificar correctamente todos los patrones de entrenamiento.

La importancia de este proceso consiste en que, a medida que se entrena la red, las neuronas de las capas intermedias se organizan a sí mismas de tal modo que las distintas neuronas aprenden a reconocer distintas características del espacio total de entrada. Después del entrenamiento, cuando se les presente un patrón arbitrario de entrada que contenga ruido o que esté incompleto, las neuronas de la capa oculta de la red responderán con una salida activa si la nueva entrada contiene un patrón que se asemeje a aquella característica que las neuronas individuales hayan aprendido a reconocer durante su entrenamiento. Y a la inversa, las unidades de las capas ocultas tienen una tendencia a inhibir su salida si el patrón de entrada no contiene la característica para reconocer, para la cual han sido entrenadas.

Varias investigaciones han demostrado que, durante el proceso de entrenamiento, la red Backpropagation tiende a desarrollar relaciones internas entre neuronas con el fin de organizar los datos de entrenamiento en clases. Esta tendencia se puede extrapolar, para llegar a la hipótesis consistente en que todas las unidades de la capa oculta de una Backpropagation son asociadas de alguna manera a características específicas del patrón de entrada como consecuencia del entrenamiento. Lo que sea o no exactamente la asociación puede no resultar evidente para el observador humano, lo importante es que la

red ha encontrado una representación interna que le permite generar las salidas deseadas cuando se le dan las entradas, en el proceso de entrenamiento. Esta misma representación interna se puede aplicar a entradas que la red no haya visto antes, y la red clasificará estas entradas según las características que compartan con los ejemplos de entrenamiento.

1.2.1.2 Estructura de la Red: La estructura típica de una red multicapa se observa en la figura 2.3.1

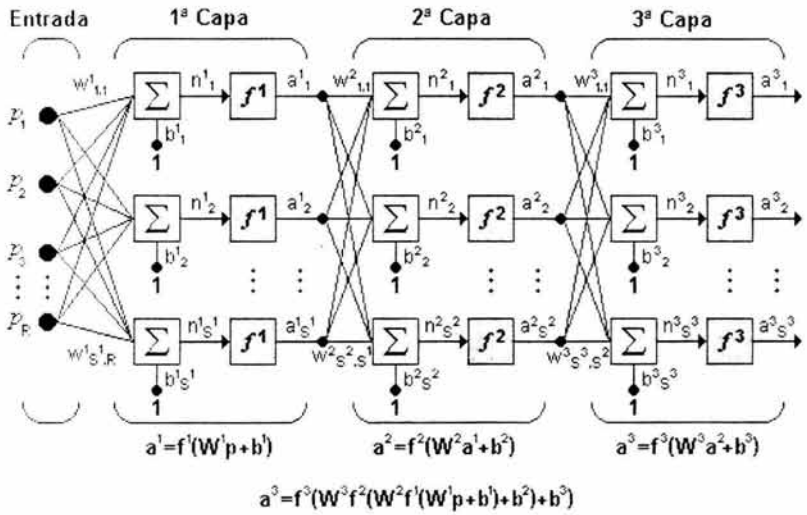


Figura 2.3.1 Red de tres capas

Puede notarse que esta red de tres capas equivale a tener tres redes tipo Perceptrón en cascada; la salida de la primera red, es la entrada a la segunda y la salida de la segunda red es la entrada a la tercera. Cada capa puede tener diferente número de neuronas, e incluso distinta función de transferencia.

En la figura 2.3.1, W^1 representa la matriz de pesos para la primera capa, W^2 los pesos de la segunda y así similarmente para todas las capas que incluya una red. Para identificar la estructura de una red multicapa, se empleará una notación abreviada, donde el número de entradas va seguido del número de neuronas en cada capa:

$$R : S^1 : S^2 : S^3 \quad (2.3.1)$$

Donde S representa el número de neuronas y el exponente representa la capa a la cual la neurona corresponde.

La notación de la figura 2.3.1 es bastante clara cuando se desea conocer la estructura detallada de la red, e identificar cada una de las conexiones, pero cuando la red es muy grande, el proceso de conexión se torna muy complejo y es bastante útil utilizar el esquema de la figura 2.3.2

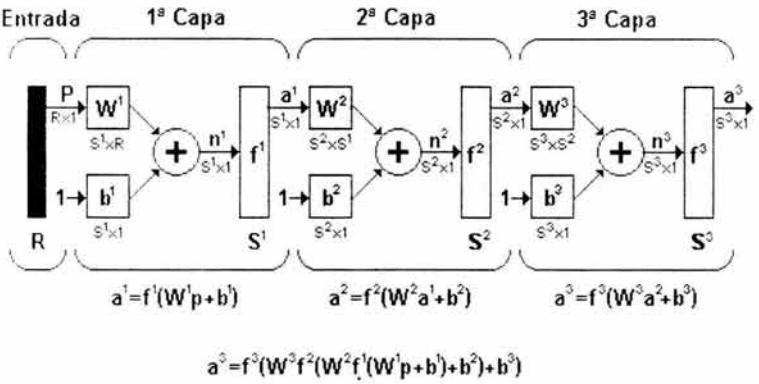


Figura 2.3.2 Notación compacta de una red de tres capas

1.2.1.3 Regla de Aprendizaje El algoritmo Backpropagation para redes multicapa es una generalización del algoritmo LMS, ambos algoritmos realizan su labor de actualización de pesos y ganancias con base en el error medio cuadrático. La red Backpropagation trabaja bajo aprendizaje supervisado y por tanto necesita un set de entrenamiento que le describa cada salida y su valor de salida esperado de la siguiente forma:

$$\{p_1,t_1\}, \{p_2,t_2\}, \ldots, \{p_Q, t_Q\} \quad (2.3.2)$$

Donde p_Q es una entrada a la red y t_Q es la correspondiente salida deseada para el patrón q-ésimo. El algoritmo debe ajustar los parámetros de la red para minimizar el error medio cuadrático.

El entrenamiento de una red neuronal multicapa se realiza mediante un proceso de aprendizaje, para realizar este proceso se debe inicialmente tener definida la topología de la red esto es: número de neuronas en la capa de entrada el cual depende del número de componentes del vector de entrada, cantidad de capas ocultas y número de neuronas de cada una de ellas, número de neuronas en la capa de la salida el cual depende del número de componentes del vector de salida o patrones objetivo y funciones de transferencia requeridas en cada capa, con base en la topología escogida se asignan valores iniciales a cada uno de los parámetros que conforma la red.

Es importante recalcar que no existe una técnica para determinar el número de capas ocultas, ni el número de neuronas que debe contener cada una de ellas

para un problema especifico, esta elección es determinada por la experiencia del diseñador, el cual debe cumplir con las limitaciones de tipo computacional.

Cada patrón de entrenamiento se propaga a través de la red y sus parámetros para producir una respuesta en la capa de salida, la cual se compara con los patrones objetivo o salidas deseadas para calcular el error en el aprendizaje, este error marca el camino mas adecuado para la actualización de los pesos y ganancias que al final del entrenamiento producirán una respuesta satisfactoria a todos los patrones de entrenamiento, esto se logra minimizando el error medio cuadrático en cada iteración del proceso de aprendizaje.

La deducción matemática de este procedimiento se realizará para una red con una capa de entrada, una capa oculta y una capa de salida y luego se generalizará para redes que tengan más de una capa oculta.

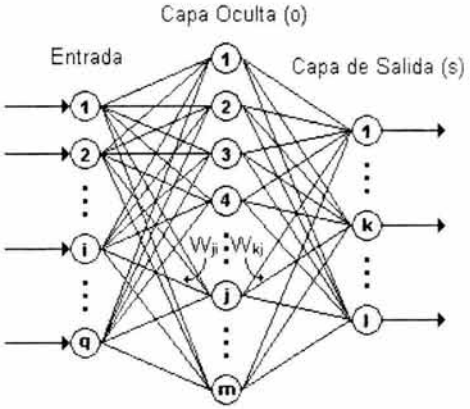


Figura 2.3.3 Disposición de una red sencilla de 3 capas

Es importante aclarar que en la figura 2.3.3

q: Equivale al número de componentes el vector de entrada.
m: Número de neuronas de la capa oculta
l: Número de neuronas de la capa de salida

Para iniciar el entrenamiento se le presenta a la red un patrón de entrenamiento, el cual tiene *q* componentes como se describe en la ecuación (2.3.3)

$$P = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_i \\ \vdots \\ p_q \end{bmatrix} \quad (2.3.3)$$

Cuando se le presenta a la red una patrón de entrenamiento, este se propaga a través de las conexiones existentes produciendo una entrada neta *n* en cada una las neuronas de la siguiente capa, la entrada neta a la neurona *j* de la siguiente capa debido a la presencia de un patrón de entrenamiento en la entrada esta dada por la ecuación (2.3.4), nótese que la entrada neta es el valor justo antes de pasar por la función de transferencia

$$n_j^o = \sum_{i=1}^{q} W_{ji}^o p_i + b_j^o \quad (2.3.4)$$

W^o_{ji}: Peso que une la componente *i* de la entrada con la neurona *j* de la capa oculta
p_i: *Componente i* del vector *p* que contiene el patrón de entrenamiento de *q* componentes
b^o_j: Ganancia de la neurona *j* de la capa oculta

Donde el superíndice (^{o}) representa la capa a la que pertenece cada parámetro, es este caso la capa oculta.

Cada una de las neuronas de la capa oculta tiene como salida a^o_j que esta dada por la ecuación (2.3.5)

$$a_j^o = f^o\left(\sum_{i=1}^{q} W_{ji}^o p_i + b_j^o\right) \quad (2.3.5)$$

f^o: Función de transferencia de las neuronas de la capa oculta

Las salidas a^o_j de las neuronas de la capa oculta (de *l* componentes) son las entradas a los pesos de conexión de la capa de salida, este comportamiento esta descrito por la ecuación (2.3.6)

$$n_k^s = \sum_{j=1}^{m} W_{kj}^s a_j^o + b_k^s \quad (2.3.6)$$

W^s_{kj}: Peso que une la neurona *j* de la capa oculta con la neurona *k* de la capa de salida, la cual cuenta con *s* neuronas
a^o_j: Salida de la neurona *j* de la capa oculta, la cual cuenta con *m* neuronas.
b^s_k: Ganancia de la neurona *k* de la capa de salida.
n^s_k: Entrada neta a la neurona *k* de la capa de salida

La red produce una salida final descrita por la ecuación (2.3.7)

$$a^s_k = f^s\left(n^s_k\right) \quad (2.3.7)$$

f^s: Función de transferencia de las neuronas de la capa de salida

Reemplazando (2.3.6) en (2.3.7) se obtiene la salida de la red en función de la entrada neta y de los pesos de conexión con la ultima capa oculta

$$a^s_k = f^s\left(\sum_{j=1}^{m} W^s_{kj} a^o_j + b^s_k\right) \quad (2.3.8)$$

La salida de la red de cada neurona a^s_k se compara con la salida deseada t_k para calcular el error en cada unidad de salida (2.3.9)

$$\delta^s_k = \left(t_k - a^s_k\right) \quad (2.3.9)$$

El error debido a cada patrón *p* propagado esta dado por (2.3.11)

$$ep^2 = \frac{1}{2}\sum_{k=1}^{s}\left(\delta_k\right)^2 \quad (2.3.10)$$

ep^2: Error medio cuadrático para cada patrón de entrada *p*
δ^s_k: Error en la neurona *k* de la capa de salida con *l* neuronas

Este proceso se repite para el número total de patrones de entrenamiento (*r*), para un proceso de aprendizaje exitoso el objetivo del algoritmo es actualizar todos los pesos y ganancias de la red minimizando el error medio cuadrático total descrito en (2.3.11)

$$e^2 = \sum_{p=1}^{r} ep^2 \quad (2.3.11)$$

e^2: Error total en el proceso de aprendizaje en una iteración luego de haber presentado a la red los *r* patrones de entrenamiento

El error que genera una red neuronal en función de sus pesos, genera un espacio de n dimensiones, donde n es el número de pesos de conexión de la red, al evaluar el gradiente del error en un punto de esta superficie se obtendrá

la dirección en la cual la función del error tendrá un mayor crecimiento, como el objetivo del proceso de aprendizaje es minimizar el error debe tomarse la dirección negativa del gradiente para obtener el mayor decremento del error y de esta forma su minimización, condición requerida para realizar la actualización de la matriz de pesos en el algoritmo Backpropagation:

$$W_{k+1} = W_k - \alpha \nabla ep^2 \quad (2.3.12)$$

El gradiente negativo de ep^2 se denotara como $-\nabla ep^2$ y se calcula como la derivada del error respecto a todos los pesos de la red

En la capa de salida el gradiente negativo del error con respecto a los pesos es:

$$-\frac{\partial ep^2}{\partial W_{kj}^s} = -\frac{\partial}{\partial W_{kj}^s}\left(\frac{1}{2}\sum_{k=1}^{l}\left(t_k - a_k^s\right)^2\right) = \left(t_k - a_k^s\right) \times \frac{\partial a_k^s}{\partial W_{kj}^s} \quad (2.3.13)$$

$-\frac{\partial ep^2}{\partial W_{kj}^s}$: Componente del gradiente $-\nabla ep^2$ respecto al peso de la conexión de la neurona de la capa de salida y la neurona j de la capa oculta W_{kj}^s

$\frac{\partial a_k^s}{\partial W_{kj}^s}$: Derivada de la salida de la neurona *k* de la capa de salida respecto, al peso W^s_{kj}

Para calcular $\frac{\partial a_k^s}{\partial W_{kj}^s}$ se debe utilizar la regla de la cadena, pues el error no es una función explícita de los pesos de la red, de la ecuación (2.3.7) puede verse que la salida de la red a^s_k esta explícitamente en función de n^s_k y de la ecuación (2.3.6) puede verse que n^s_k esta explícitamente en función de W^s_{kj}, considerando esto se genera la ecuación (2.3.13)

$$\frac{\partial a_k^s}{\partial W_{kj}^s} = \frac{\partial a_k^s}{\partial n_k^s} \times \frac{\partial n_k^s}{\partial W_{kj}^s} \quad (2.3.14)$$

Tomando la ecuación (2.3.14) y reemplazándola en la ecuación (2.3.13) se obtiene.

$$-\frac{\partial ep^2}{\partial W_{kj}^s} = \left(t_k - a_k^s\right) \times \frac{\partial a_k^s}{\partial n_k^s} \times \frac{\partial n_k^s}{\partial W_{kj}^s} \quad (2.3.15)$$

$\frac{\partial n_k^s}{\partial W_{kj}^s}$: Derivada de la entrada neta a la neurona *k* de la capa de salida respecto a los pesos de la conexión entre las neuronas de la capa oculta y la capa de salida

$\frac{\partial a_k^s}{\partial n_k^s}$: Derivada de la salida de la neurona *k* de la capa de salida respecto a su entrada neta.

Reemplazando en la ecuación (2.3.15) las derivadas de las ecuaciones (2.3.6) y (2.3.7) se obtiene

$$-\frac{\partial ep^2}{\partial W_{kj}^s} = \left(t_k - a_k^s\right) \times f'^s\left(n_k^s\right) \times a_j^o \quad (2.3.16)$$

Como se observa en la ecuación (2.3.16) las funciones de transferencia utilizadas en este tipo de red deben ser continuas para que su derivada exista en todo el intervalo, ya que el término $f^s(n^s_k)$ es requerido para el cálculo del error.

Las funciones de transferencia f más utilizadas y sus respectivas derivadas son las siguientes:

logsig: $f(n) = \frac{1}{1+e^{-n}}$ $f'(n) = f(n)(1 - f(n))$ $f'(n) = a(1-a)$ (2.3.17)

tansig: $f(n) = \frac{e^n - e^{-n}}{e^n + e^{-n}}$ $f'(n) = 1 - (f(n))^2$ $f'(n) = (1 - a^2)$ (2.3.18)

purelin: $f(n) = n$ $f'(n) = 1$ (2.3.19)

De la ecuación (2.3.16), los términos del error para las neuronas de la capa de salida están dados por la ecuación (2.3.20), la cual se le denomina comúnmente sensitividad de la capa de salida.

$$\delta_k^s = \left(t_k - a_k^s\right) f'^s\left(n_k^s\right) \quad (2.3.20)$$

Este algoritmo se denomina Backpropagation o de propagación inversa debido a que el error se propaga de manera inversa al funcionamiento normal de la red, de esta forma, el algoritmo encuentra el error en el proceso de aprendizaje desde las capas más internas hasta llegar a la entrada; con base en el cálculo de este error se actualizan los pesos y ganancias de cada capa.

Después de conocer (2.3.20) se procede a encontrar el error en la capa oculta el cual esta dado por:

$$-\frac{\partial ep^2}{\partial W_{ji}^o} = -\frac{\partial}{\partial W_{ji}^o}\left(\frac{1}{2}\sum_{k=1}^{l}\left(t_k - a_k^s\right)^2\right) = \sum_{k=1}^{l}\left(t_k - a_k^s\right)\times\frac{\partial a_k^s}{\partial W_{ji}^o} \quad (2.3.21)$$

Para calcular el último término de la ecuación (2.3.21) se debe aplicar la regla de la cadena en varias ocasiones como se observa en la ecuación (2.3.22) puesto que la salida de la red no es una función explícita de los pesos de la conexión entre la capa de entrada y la capa oculta

$$\frac{\partial a_k^s}{\partial W_{ji}^o} = \frac{\partial a_k^s}{\partial n_k^s}\times\frac{\partial n_k^s}{\partial a_k^o}\times\frac{\partial a_k^o}{\partial n_j^o}\times\frac{\partial n_j^o}{\partial W_{ji}^o} \quad (2.3.22)$$

Todos los términos de la ecuación (2.3.23) son derivados respecto a variables de las que dependan explícitamente, reemplazando (2.3.22) en (2.3.21) tenemos:

$$-\frac{\partial ep^2}{\partial W_{ji}^o} = \sum_{k=1}^{l}\left(t_k - a_k^s\right)\times\frac{\partial a_k^s}{\partial n_k^s}\times\frac{\partial n_k^s}{\partial a_k^o}\times\frac{\partial a_k^o}{\partial n_j^o}\times\frac{\partial n_j^o}{\partial W_{ji}^o} \quad (2.3.23)$$

Tomando las derivas de las ecuaciones (2.3.4) (2.3.5) (2.3.6) (2.3.7) y reemplazándolas en la ecuación (2.3.23) se obtiene la expresión del gradiente del error en la capa oculta

$$-\frac{\partial ep^2}{\partial W_{ji}^o} = \sum_{k=1}^{l}\left(t_k - a_k^s\right)\times f'^s\left(n_k^s\right)\times \boldsymbol{W}_{kj}^s \times f'^o\left(n_j^o\right)\times \boldsymbol{p}_i \quad (2.3.24)$$

Reemplazando la ecuación (2.3.20) en la ecuación (2.3.24) se tiene:

$$-\frac{\partial ep^2}{\partial W_{ji}^o} = \sum_{k=1}^{l} \delta_k^s \times \boldsymbol{W}_{kj}^s \times f'^o\left(n_j^o\right)\times \boldsymbol{p}_i \quad (2.3.25)$$

Los términos del error para cada neurona de la capa oculta esta dado por la ecuación (2.3.26), este término también se denomina sensitividad de la capa oculta

$$\delta_j^o = f'^o\left(n_j^o\right)\times\sum_{k=1}^{l}\delta_k^s \boldsymbol{W}_{kj}^s \quad (2.3.26)$$

Luego de encontrar el valor del gradiente del error se procede a actualizar los pesos de todas las capas empezando por la de salida, para la capa de salida la actualización de pesos y ganancias esta dada por (2.3.27) y (2.3.28).

$$W_{kj}(t+1) = W_{kj}(t) - 2\alpha\delta_k^s \quad (2.3.27)$$

$$b_k(t+1) = b_k(t) - 2\alpha\delta_k^s \quad (2.3.28)$$

α: Rata de aprendizaje que varía entre 0 y 1 dependiendo de las características del problema a solucionar.

Luego de actualizar los pesos y ganancias de al capa de salida se procede a actualizar los pesos y ganancias de la capa oculta mediante las ecuaciones (2.3.29) y (2.3.30)

$$W_{ji}(t+1) = W_{ji}(t) - 2\alpha\delta_j^o p_i \quad (2.3.29)$$

$$b_j(t+1) = b_j(t) - 2\alpha\delta_j^o \quad (2.3.30)$$

Esta deducción fue realizada para una red de tres capas, si se requiere realizar el análisis para una red con dos o más capas ocultas, las expresiones pueden derivarse de la ecuación (2.3.26) donde los términos que se encuentran dentro de la sumatoria pertenecen a la capa inmediatamente superior, este algoritmo es conocido como la regla Delta Generalizada desarrollada por Rumelhart D [], la cual es una extensión de la regla delta desarrollada por Widrow [] en 1930

Para algunos autores las sensitividades de las capas están detonadas por la letra **S**, reescribiendo las ecuaciones (2.3.20) y (2.3.26) con esta notación se obtienen las ecuaciones (2.3.31) y (2.3.32)

$$\boldsymbol{S}^M = -2\dot{f}^M(\boldsymbol{n}^M)(\boldsymbol{t} - \boldsymbol{a}) \quad (2.3.31)$$

$$\boldsymbol{s}^m = \dot{f}^m(\boldsymbol{n}^m)(\boldsymbol{W}^{m+1})^T \boldsymbol{s}^{m+1}, \text{ para } m = M-1,\ldots,2,1. \quad (2.3.32)$$

En la ecuación (2.3.31) M representa la última capa y S^M la sensitividad para esta capa, la ecuación (2.3.32) expresa el cálculo de la sensitividad capa por capa comenzando desde la última capa oculta, cada uno de estos términos involucra que el término para la sensitividad de la capa siguiente ya este calculado.

Como se ve el algoritmo Backpropagation utiliza la misma técnica de aproximación en pasos descendientes que emplea el algoritmo LMS, la única complicación está en el cálculo del gradiente, el cual es un término indispensable para realizar la propagación de la sensitividad.

En las técnicas de gradiente descendiente es conveniente avanzar por la superficie de error con incrementos pequeños de los pesos; esto se debe a que

tenemos una información local de la superficie y no se sabe lo lejos o lo cerca que se está del punto mínimo, con incrementos grandes, se corre el riesgo de pasar por encima del punto mínimo, con incrementos pequeños, aunque se tarde más en llegar, se evita que esto ocurra. El elegir un incremento adecuado influye en la velocidad de convergencia del algoritmo, esta velocidad se controla a través de la rata de aprendizaje α, la que por lo general se escoge como un número pequeño, para asegurar que la red encuentre una solución. Un valor pequeño de αsignifica que la red tendrá que hacer un gran número de iteraciones, si se toma un valor muy grande, los cambios en los pesos serán muy grandes, avanzando muy rápidamente por la superficie de error, con el riesgo de saltar el valor mínimo del error y estar oscilando alrededor de él, pero sin poder alcanzarlo.

Es recomendable aumentar el valor de αa medida que disminuye el error de la red durante la fase de entrenamiento, para garantizar así una rápida convergencia, teniendo la precaución de no tomar valores demasiado grandes que hagan que la red oscile alejándose demasiado del valor mínimo. Algo importante que debe tenerse en cuenta, es la posibilidad de convergencia hacia alguno de los mínimos locales que pueden existir en la superficie del error del espacio de pesos como se ve en la figura 2.3.4.

Figura 2.3.4 Superficie típica de error

En el desarrollo matemático que se ha realizado para llegar al algoritmo Backpropagation, no se asegura en ningún momento que el mínimo que se encuentre sea global, una vez la red se asiente en un mínimo sea local o global cesa el aprendizaje, aunque el error siga siendo alto. En todo caso, si la solución es admisible desde el punto de vista del error, no importa si el mínimo es local o global o si se ha detenido en algún momento previo a alcanzar un verdadero mínimo.

Para ilustrar el cálculo de cada uno de los términos del algoritmo Backpropagation, este se utilizara se para aproximar la siguiente función:

$$t = \sin\frac{\pi}{4}p \quad para\ el\ intervalo\ -2 \le p \le 2 \tag{2.3.33}$$

La función se ha restringido al intervalo entre –2 y 2 para conservarla dentro de límites observables, como se observa en la figura 2.3.5

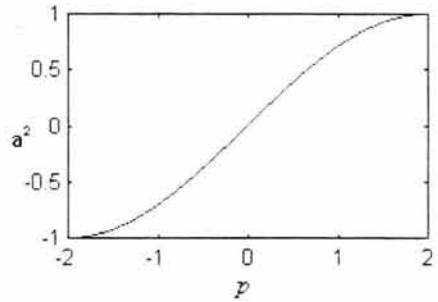


Figura 2.3.5 Intervalo de la función t

La configuración escogida para la red corresponde a una red 1:2:1 según la notación definida con anterioridad, es decir una entrada, dos neuronas en la capa oculta y una salida; esta estructura se visualiza en la figura 2.3.6

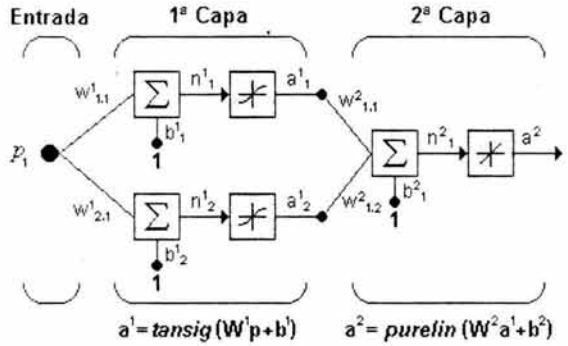


Figura 2.3.6 Red utilizada para aproximar la función

Como se observa la salida de la red para la primera capa está dada por

$$a^1 = tansig(W^1 p^T + b) \quad (2.3.34)$$

Las redes tipo Backpropagation utilizan principalmente dos funciones de transferencia en la primera capa: *logsig*, cuando el rango de la función es siempre positivo y *tansig* como en este caso, cuando se le permite a la función oscilar entre valores positivos y negativos limitados en el intervalo –1, 1.

La salida de la segunda capa está determinada generalmente por la función de transferencia *purelin*, la cual reproduce exactamente el valor resultante después de la sumatoria.

$$a^2 = purelin(W^2 * a^1 + b^2) \quad (2.3.35)$$

Al evaluar la ecuación (2.3.33) en los diferentes patrones de entrenamiento, se obtienen los valores de las entradas y sus salidas asociadas, ya que como se dijo antes la red Backpropagation es una red de aprendizaje supervisado. Es importante destacar, que no es estrictamente necesario el conocimiento de la función a aproximar, basta con conocer la respuesta a una entrada dada, o un registro estadístico de salidas para modelar el comportamiento del sistema, limitando el problema a la realización de pruebas a una caja negra.

Los parámetros de entrada y sus valores de salida asociados, se observan en la tabla 2.3.1

	1	2	3	4	5	6
p	-2	-1,2	0,4	0,4	1,2	2
t	-1	-0,81	-0,31	0,309	0,809	1

Tabla 2.3.1 Set de entrenamiento de la red

Los valores iniciales para la matriz de pesos y el vector de ganancias de la red se escogieron en forma aleatoria así:

$$W^1 = \begin{bmatrix} -0.2 \\ 0.5 \end{bmatrix}, \quad b^1 = \begin{bmatrix} 0.7 \\ -0.2 \end{bmatrix}, \quad W^2 = \begin{bmatrix} 0.1 & 0.3 \end{bmatrix}, \quad b^2 = \begin{bmatrix} 0.8 \end{bmatrix}, \quad \alpha = 0.1$$

Para el proceso de cálculo, se le presenta a la red el patron de entrenamiento *p1* de esta forma la primera iteración es como sigue

$$a^1 = tansig\left(\begin{bmatrix} -0.2 \\ 0.5 \end{bmatrix}(-0.2) + \begin{bmatrix} 0.7 \\ -0.2 \end{bmatrix}\right) = \begin{bmatrix} 0.8 \\ -0.83 \end{bmatrix}$$

$$a^2 = \begin{bmatrix} 0.1 & 0.3 \end{bmatrix}\begin{bmatrix} 0.8 \\ -0.83 \end{bmatrix} + \begin{bmatrix} 0.8 \end{bmatrix} = 0.63$$

$$e = t - a = -1 - (0.63) = -1.63$$

Como se esperaba la primera iteración no ha sido suficiente, para aproximar la función correctamente, así que se calculará la sensitividad para iniciar el proceso de actualización de los valores de los pesos y las ganancias de la red.

Los valores de las derivadas del error medio cuadrático son:

$$\dot{f}^1(n) = 1 - a_1^2$$

$$\dot{f}^2(n) = 1$$

Y las sensitividades, empezando desde la última hasta la primera capa,

s^2=-2(1) (-1.63) = 3 26

$$s^1 = \begin{bmatrix} (1-0.8^2) & 0 \\ 0 & (1-0.83^2) \end{bmatrix} \begin{bmatrix} 0.1 \\ 0.3 \end{bmatrix} (3.26) = \begin{bmatrix} 0.1171 \\ 0.2983 \end{bmatrix}$$

Con estos valores, y de acuerdo a la regla de actualización descrita anteriormente, los nuevos parámetros de la red son:

$$W^2(1) = [0.1 \quad 0.3] - 0.1(3.26)[0.8 \quad -0.83] = [-0.161 \quad 0.5718]$$

$$b^2(1) = [0.8] - 0.1(3.26) = 0.474$$

$$W^1(1) = \begin{bmatrix} -0.2 \\ 0.5 \end{bmatrix} - 0.1 \begin{bmatrix} 0.1171 \\ 0.2983 \end{bmatrix} (-2) = \begin{bmatrix} -0.1766 \\ 0.5957 \end{bmatrix}$$

$$b^1(1) = \begin{bmatrix} 0.7 \\ -0.2 \end{bmatrix} - 0.1 \begin{bmatrix} 0.1171 \\ 0.2983 \end{bmatrix} = \begin{bmatrix} 0.688 \\ -0.2298 \end{bmatrix}$$

Con esto se completa la primera iteración, y el algoritmo queda listo para presentar a la red el siguiente patrón y continuar el proceso iterativo hasta obtener un valor de tolerancia aceptable para el error.

En 1989 Funahashi [] demostró matemáticamente que una red neuronal multicapa puede aproximar cualquier función no lineal o mapa lineal multivariable, $f(x) = R^n \rightarrow R$ Este teorema es de existencia, pues prueba que la red existe pero no indica como construirla y tampoco garantiza que la red aprenderá función.

El algoritmo Backpropagation es fácil de implementar, y tiene la flexibilidad de adaptarse para aproximar cualquier función, siendo una de las redes multicapa más potentes; esta característica ha convertido a esta red en una de las más ampliamente utilizadas y ha llevado al desarrollo de nuevas técnicas que permitan su mejoramiento. Dentro de estas técnicas encontramos dos métodos heurísticos y dos métodos basados en algoritmos de optimización numérica.

1.2.1.3.1 Red Backpropagation con momentum: Esta modificación está basada en la observación de la última sección de la gráfica del error medio cuadrático en el proceso de convergencia típico para una red Backpropagation; este proceso puede verse en la figura 2.3.7 en la cual se nota la caída brusca del error en la iteración para la cual alcanza convergencia

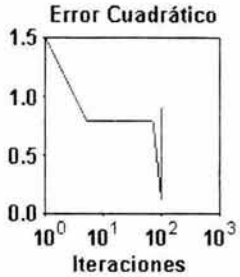


Figura 2.3.7 Comportamiento típico del proceso de convergencia para una red Backpropagation

Este comportamiento puede causar oscilaciones no deseadas, por lo que es conveniente suavizar esta sección de la gráfica incorporando un filtro pasa-bajo al sistema. Para ilustrar el efecto positivo del filtro en el proceso de convergencia, se analizará el siguiente filtro de primer orden:

$$y(k) = \gamma y(k-1) + (1-\gamma)w(k) \quad (2.3.36)$$

Donde *w(k)* es la entrada al filtro, *y(k)* su salida y γ es el coeficiente de momentum que está en el intervalo $0 \leq \gamma \leq 1$

El efecto del filtro puede observase en la figura 2.3.8, en la cual se tomó como entrada al filtro la función:

$$w(k) = 1 + \text{sen}\left(\frac{2\pi k}{16}\right) \quad (2.3.37)$$

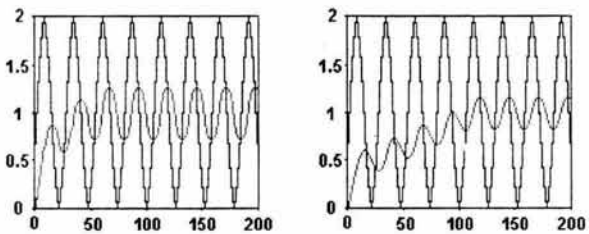


Figura 2.3.8 Efecto del coeficiente de momentum

El coeficiente de momentum se asumió $\gamma = 0.9$ para la gráfica de la izquierda y $\gamma = 0.98$ para la gráfica de la derecha. De esta figura puede notarse como la oscilación es menor a la salida del filtro, la oscilación se reduce a medida que γ se decrementa, el promedio de la salida del filtro es el mismo que el promedio de entrada al filtro aunque mientras γ sea incrementado la salida del filtro será más lenta.

Recordando los parámetros de actualización empleados por el algoritmo Backpropagation tradicional:

$$\Delta W^m(k) = -\alpha s^m (a^{m-1})^T \quad (2.3.38)$$

$$\Delta b^m(k) = -\alpha s^m \quad (2.3.39)$$

Al adicionar el filtro con momentum a este algoritmo de actualización, se obtienen las siguientes ecuaciones que representan el algoritmo Backpropagation con momentum:

$$\Delta W^m(k) = \gamma \Delta W^m(k-1) - (1-\gamma)\alpha s^m (a^{m-1})^T \quad (2.3.40)$$

$$\Delta b^m(k) = \gamma \Delta b^m(k-1) - (1-\gamma) - \alpha s^m \quad (2.3.41)$$

Este algoritmo, hace que la convergencia sea estable e incluso más rápida, además permite utilizar una rata de aprendizaje alta.

La figura 2.3.9 referencia el comportamiento del algoritmo con momentum en el punto de convergencia:

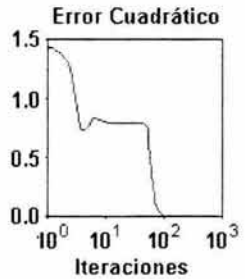


Figura 2.3.9 Trayectoria de convergencia con momentum

1.2.1.3.2 Red Backpropagation con rata de aprendizaje variable: Del análisis de la sección 2.3.3 se vio que $\nabla e(x)$ es el gradiente del error, de igual forma se definirá $\nabla e^2(x)$ como la Hessiana de la función de error, donde *x* representa las variables de las cuales depende el error (pesos y ganancias), esta matriz es siempre de la forma:

$$\nabla^2 e(x) = \begin{bmatrix} \frac{\partial^2 e(x)}{\partial x_1^2} & \frac{\partial^2 e(x)}{\partial x_1 \partial x_2} & \cdots \frac{\partial^2 e(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 e(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 e(x)}{\partial x_2^2} & \cdots \frac{\partial^2 e(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \cdots \vdots \\ \frac{\partial^2 e(x)}{\partial x_n \partial x_1} & \frac{\partial^2 e(x)}{\partial x_n \partial x_2} & \cdots \frac{\partial^2 e(x)}{\partial x_n^2} \end{bmatrix} \quad (2.3.42)$$

La superficie del error medio cuadrático para redes de una sola capa es siempre una función cuadrática y la matriz Hessiana es por tanto constante, esto lleva a que la máxima rata de aprendizaje estable para el algoritmo de pasos descendientes sea el máximo valor propio de la matriz Hessiana dividido 2.HBD[]. Para una red multicapa la superficie del error no es una función cuadrática, su forma es diferente para diferentes regiones del espacio, la velocidad de convergencia puede incrementarse por la variación de la rata de aprendizaje en cada parte de la superficie del error, sin sobrepasar el valor máximo para aprendizaje estable definido anteriormente.

Existen varias técnicas para modificar la rata de aprendizaje; este algoritmo emplea un procedimiento mediante el cual la rata de aprendizaje varia de acuerdo al rendimiento que va presentando el algoritmo en cada punto; si el error disminuye vamos por el camino correcto y se puede ir más rápido incrementando la rata de aprendizaje, si el error aumenta, es necesario decrementar la rata de aprendizaje; el criterio de variación de αdebe estar en concordancia con las siguientes reglas heurísticas:

1. Si el error cuadrático de todos los parámetros del set de entrenamiento se incrementa en un porcentaje ζ típicamente entre 1% y 5%, después de la actualización de los pesos, esa actualización es descartada, la rata de aprendizaje se multiplica por un factor $0 < \rho < 1$, y el coeficiente de momentum es fijado en cero.

2. Si el error cuadrático se decrementa después de la actualización de los pesos, esa actualización es aceptada y la rata de aprendizaje es multiplicada por un factor $\eta > 1$. Si γ había sido previamente puesto en cero, se retorna a su valor original.

3. Si el error cuadrático se incrementa en un valor menor a ζ, los pesos actualizados son aceptados, pero la rata de aprendizaje y el coeficiente de momentum no son cambiados.

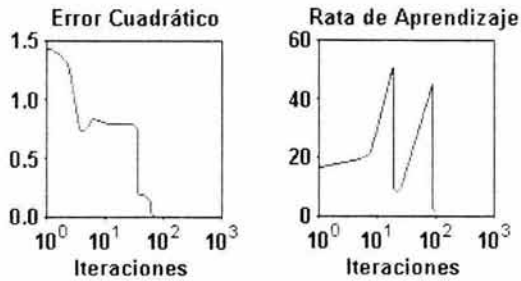


Figura 2.3.10 Característica de convergencia para una rata de aprendizaje variable

La figura 2.3.10, muestra la trayectoria de la rata de aprendizaje para este algoritmo en comparación con la característica de convergencia

Existen muchas variaciones de este algoritmo, por ejemplo Jacobs[] propuso la regla delta-bar-delta, en la cual cada uno de los parámetros de la red, (pesos y ganancias) tenían su propia rata de aprendizaje. El algoritmo incrementa la rata de aprendizaje para un parámetro de la red si el parámetro escogido, ha estado en la misma dirección para varias iteraciones; si la dirección del parámetro escogido cambia, entonces la rata de aprendizaje es reducida.

Los algoritmos Backpropagation con momentum y con rata de aprendizaje variable son los dos métodos heurísticos más utilizados para modificar el algoritmo Backpropagation tradicional. Estas modificaciones garantizan rápida convergencia para algunos problemas, sin embargo presentan dos problemas principales: primero, requieren de un gran número de parámetros (ζ, ρ, γ), los que la mayoría de las veces se definen por un método de ensayo y error de acuerdo a la experiencia del investigador, mientras que el algoritmo tradicional, solo requiere definir la rata de aprendizaje; segundo, estas modificaciones pueden llevar a que el algoritmo nunca converja y se torne oscilante para problemas muy complejos.

Como se menciono antes, existen también métodos de modificación basados en técnicas de optimización numérica, de esta clase de modificaciones se destacaran las más sobresalientes; es importante recalcar que estos métodos requieren una matemática más exigente, que el simple del dominio de cálculo diferencial.

1.2.1.3.3 Método del Gradiente Conjugado []: Este algoritmo no involucra el cálculo de las segundas derivadas de las variables y converge al mínimo de la función cuadrática en un número finito de iteraciones. El algoritmo del gradiente conjugado, sin aplicarlo aún al algoritmo de propagación inversa consiste en:

1. Seleccionar la dirección de p_0, la condición inicial, en el sentido negativo del gradiente:

$$p_0 = -g_0 \quad (2.3.43)$$

Donde

$$g(k) \equiv \nabla e(x)\big|_{x=x_k} \text{ (2.3.44)}$$

2. Seleccionar la rata de aprendizaje α_k para minimizar la función a lo largo de la dirección

$$x_{k+1} = x_k + \alpha_k \boldsymbol{p}_k \text{ (2.3.45)}$$

3. Seleccionar la dirección siguiente de acuerdo a la ecuación

$$\boldsymbol{p}_k = -\boldsymbol{g}_k + \beta_k \boldsymbol{p}_{k-1} \text{ (2.3.46)}$$

con

$$\beta_k = \frac{\Delta \boldsymbol{g}_{k-1}^T \boldsymbol{g}_k}{\Delta \boldsymbol{g}_{k-1}^T \boldsymbol{p}_{k-1}} \text{ o } \beta_k = \frac{\boldsymbol{g}_k^T \boldsymbol{g}_k}{\boldsymbol{g}_{k-1}^T \boldsymbol{g}_{k-1}} \text{ (2.3.47)}$$

4. Si el algoritmo en este punto aún no ha convergido, regresamos al numeral 2

Este algoritmo no puede ser aplicado directamente a una red neural porque el error no es una función cuadrática; lo que afecta al algoritmo en dos formas, primero no es hábil para minimizar la función a lo largo de una línea como es requerido en el paso 2; segundo, el error mínimo no será alcanzado normalmente en un número finito de pasos y por esto el algoritmo necesitará ser inicializado después de un número determinado de iteraciones.
A pesar de estas complicaciones, esta modificación del algoritmo Backpropagation converge en muy pocas iteraciones, y es incluso uno de los algoritmos más rápidos para redes multicapa, como puede notarse en la figura 2.3.11

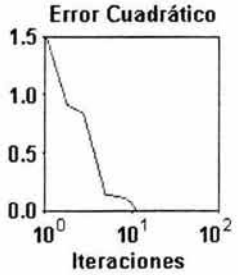


Figura 2.3.11 Trayectoria del Gradiente Conjugado

1.2.1.3.4 Algoritmo de Levenberg - Marquardt:[] Este algoritmo es una modificación del método de Newton, el que fue diseñado para minimizar funciones que sean la suma de los cuadrados de otras funciones no lineales; es por ello que el algoritmo de Levenberg - Marquardt, tiene un excelente desempeño en el entrenamiento de redes neuronales donde el rendimiento de la red esté determinado por el error medio cuadrático.

El método de Newton para optimizar el rendimiento *e(x)* es:

$$X_{k+1} = X_k - A_k^{-1} g_k \quad (2.3.48)$$

$$A_k \equiv \nabla^2 e(x)\big|_{x=x_k} \qquad g_k \equiv \nabla e(x)\big|_{x=x_k} \quad (2.3.49)$$

Si asumimos que *e(x)* es una suma de funciones cuadráticas:

$$e(x) = \sum_{i=1}^{n} v_i^2(x) = v^T(x)v(x) \quad (2.3.50)$$

El gradiente puede ser escrito entonces en forma matricial:

$$\nabla e(x) = 2J^T(x)v(x) \quad (2.3.51)$$

Donde *J(x)* es la matriz Jacobiana.

Ajustando el método de Newton, obtenemos el algoritmo de Levenberg Marquardt

$$x_{k+1} = x_k - \left[J^T(x_k)J(x_k) + \mu_k I\right]^{-1} J^T(x_k)v(x_k) \quad (2.3.52)$$

o

$$\Delta x_k = -\left[J^T(x_k)J(x_k) + \mu_k I\right]^{-1} J^T(x_k)v(x_k) \quad (2.3.53)$$

La nueva constante μ determina la tendencia el algoritmo, cuando μ_k se incrementa, este algoritmo se aproxima al algoritmo de pasos descendientes para ratas de aprendizaje muy pequeñas; cuando μ_k se decrementa este algoritmo se convierte en el método de Gauss - Newton

El algoritmo comienza con un valor pequeño para μ_k, por lo general 0.01, si en ese paso no se alcanza el valor para *e(x)* entonces el paso es repetido con μ_k multiplicado por un factor $\vartheta > 1$. Si se ha escogido un valor pequeño de paso en la dirección de paso descendiente, *e(x)* debería decrecer. Si un paso produce un pequeño valor para *e(x)*, entonces el algoritmo tiende al método de Gauss - Newton, el que se supone garantiza una rápida convergencia. Este algoritmo genera un compromiso entre la velocidad del método de Gauss-Newton y la garantía de convergencia del método de paso descendiente.

Los elementos de la matriz Jacobiana necesarios en el algoritmo de Levenberg-Marquardt son de este estilo:

$$[J]_{h,l} = \frac{\partial e_{k,q}}{\partial x_1} \quad (2.3.54)$$

Donde x es el vector de parámetros de la red, que tiene la siguiente forma:

$$x^T = [x_1, x_2, ..x_n] = \left[w^1_{1,1}, w^1_{1,2}, \ldots, w^1_{S^1,R}, b^1_1, \ldots, b^1_{S^1}\right] \quad (2.3.55)$$

Para utilizar este algoritmo en las aplicaciones para redes multicapa, se redefinirá el término sensitividad de forma que sea más simple hallarlo en cada iteración.

$$s^m_{i,h} \equiv \frac{\partial e_{k,q}}{\partial n^m_{i,q}} \quad (2.3.56)$$

Donde $h=(q-1)S^M + k$

Con la sensitividad definida de esta manera, los términos de la matriz Jacobiana pueden ser calculados más fácilmente:

$$[J]_{h,l} = \frac{\partial e_{k,q}}{\partial w^m_{i,j}} = \frac{\partial e_{k,q}}{\partial n^m_{i,q}} * \frac{\partial n_{i,q}}{\partial w^m_{i,j}} = s^m_{i,h} * \frac{\partial n_{i,q}}{\partial w^m_{i,j}} = s^m_{i,h} * a^{m-1}_{j,q} \quad (2.3.57)$$

y para las ganancias:

$$[J]_{h,l} = \frac{\partial e_{k,q}}{\partial b^m_i} = \frac{\partial e_{k,q}}{\partial n^m_{i,q}} * \frac{\partial n_{i,q}}{\partial b^m_i} = s^m_{i,h} * \frac{\partial n_{i,q}}{\partial b^m_i} = s^m_{i,h} \quad (2.3.58)$$

De esta forma, cuando la entrada p_Q ha sido aplicada a la red y su correspondiente salida a^M_Q ha sido computada, el algoritmo Backpropagation de Levenberg-Marquardt es inicializado con:

$$S^M_q = -f^M(n^M_q) \quad (2.3.59)$$

Cada columna de la matriz S^M_Q debe ser propagada inversamente a través de la red para producir una fila de la matriz Jacobiana. Las columnas pueden también ser propagadas conjuntamente de la siguiente manera:

$$S^m_q = f^m(n^m_q)(W^{m+1})^T S^{m+1}_q \quad (2.3.60)$$

La matrices sensitividad total para cada capa en el algoritmo de Levenberg-Marquardt son formadas por la extensión de las matrices computadas para cada entrada:

$$S^m = \left[S_1^m\right]\left[S_2^m\right] \cdots \left[S_Q^m\right] \quad (2.3.61)$$

Para cada nueva entrada que es presentada a la red, los vectores de sensitividad son propagados hacia atrás, esto se debe a que se ha calculado cada error en forma individual, en lugar de derivar la suma al cuadrado de los errores. Para cada entrada aplicada a la red habrá S^M errores, uno por cada elemento de salida de la red y por cada error se generara una fila de la matriz Jacobiana.

Este algoritmo puede resumirse de la siguiente manera:

1. Se presentan todas las entradas a la red, se calculan las correspondientes salidas y cada uno de los errores según

$$\boldsymbol{e}_q = \boldsymbol{t}_q - \boldsymbol{a}_q^M \quad (2.3.62)$$

   se calcula después, la suma de los errores cuadrados para cada entrada $e(x)$

2. Se calculan las sensitividades individuales y la matriz sensitividad total y con estas, se calculan los elementos de la matriz Jacobiana.

3. Se obtiene Δx_k

4. Se recalcula la suma de los errores cuadrados usando $x_k + \Delta x_k$. Si esta nueva suma es más pequeña que el valor calculado en el paso 1 entonces se divide μpor ϑ, se calcula $x_{k+1} = x_k + \Delta x_k$y se regresa al paso 1. Si la suma no se reduce entonces se multiplica μpor ϑ

y se regresa al paso 3.
El algoritmo debe alcanzar convergencia cuando la norma del gradiente de

$$\nabla \boldsymbol{e}(\boldsymbol{x}) = 2\boldsymbol{J}^T(\boldsymbol{x})\boldsymbol{v}(\boldsymbol{x}) \quad (2.3.63)$$

sea menor que algún valor predeterminado, o cuando la suma de los errores cuadrados ha sido reducida a un error que se haya trazado como meta.
El comportamiento de este algoritmo se visualiza en la figura 2.3.12, la cual muestra la trayectoria de convergencia con $\mu = 0.01$y $\vartheta = 5$

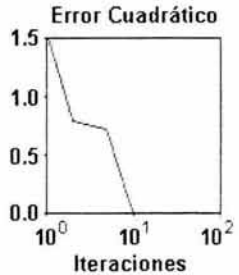


Figura 2.3.12 Trayectoria del algoritmo Levenberg-Marquardt

Como puede verse, este algoritmo converge en menos iteraciones que cualquier método discutido anteriormente, por supuesto requiere mucha más computación por iteración, debido a que implica el cálculo de matrices inversas. A pesar de su gran esfuerzo computacional sigue siendo el algoritmo de entrenamiento más rápido para redes neuronales cuando se trabaja con un moderado número de parámetros en la red, si el número de parámetros es muy grande utilizarlo resulta poco práctico.

1.3 Robótica

La robótica es una ciencia aplicada que ha sido considerada como una combinación de tecnología de las maquinas-herramientas y de la informática. Para comprender el empleo de sensores en robótica hay que estar familiarizado con la forma en que se programan los robots. Para comprender el uso de un efector final debe conocerse que una función fundamental de un robot es manipular piezas y herramientas. Para describir la tecnología de un robot tenemos que definir una diversidad de características técnicas relativas a la forma en que esta construido el robot y a la manera en que opera. Los robots trabajan con sensores, herramientas y pinzas y deberán definirse estos términos.

1.3.1 ANATOMÍA DEL ROBOT

La anatomía del robot se refiere a la construcción física del cuerpo, brazo y muñeca de la maquina en cuestión. La muñeca está constituida por varios componentes que le permiten orientarse en una diversidad de posiciones. Los movimientos relativos entre los diversos componentes del cuerpo, brazo y muñeca son proporcionados por una serie de articulaciones. Estos movimientos de las articulaciones suelen implicar deslizamientos o giros. El cuerpo, el brazo y el conjunto de la muñeca se denomina el manipulador.

La mayoría de los robots en la actualidad tiene una de estas cuatro configuraciones básicas:

Configuración Polar

Configuraron cilíndrica

Configuración de coordenadas cartesianas

configuración de brazo articulado.

las cuatro configuraciones básicas se ilustran en las representaciones esquemáticas de la siguiente figura:

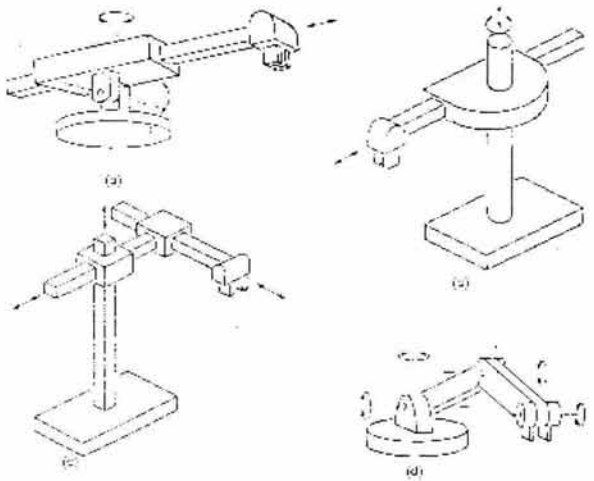


Figura 1 configuraciones básicas de los robots

la configuración polar se ilustra en la parte *(a)* de la figura 1 . Generalmente utiliza un brazo telescópico que puede elevarse o bajar alrededor de un pivote horizontal. Este pivote está montado sobre una base giratoria. Estas diversas articulaciones proporcionan al robot la capacidad para desplazar su brazo dentro de un espacio esférico, y de aquí la denominación de robot de "coordenadas esféricas" que suele aplicase a este tipo de mecanismos.

La configuración cilíndrica, según se muestra en la figura 1 *(b)*, utiliza una columna vertical y un dispositivo de deslizamiento que puede moverse hacia arriba o abajo a lo largo de la columna. El brazo del robot está unido al dispositivo deslizante de modo que puede moverse en sentido radial con respecto a la columna. Haciendo girar la columna, el robot es capaz de conseguir un espacio de trabajo que se aproxima a un cilindro.

El robot de coordenadas cartesianas, ilustrado en la parte (c) de la figura 1, utiliza tres dispositivos deslizantes perpendiculares para construir los ejes *x, y, z*. Otros nombres se aplican a este tipo de configuraciones, incluyendo las denominaciones de robot *xyz*, y robot rectilíneo. Desplazando los tres dispositivos deslizantes entre si, el robot es capaz de operar dentro de una envolvente rectangular de trabajo.

Dadas las características de esta configuración , se toma este tipo de robots para lograr la autonomía de funcionamiento para la captura de las imágenes para su procesamiento e interpretación con la utilidad de las redes neuronales.

El robot de brazo articulado se ilustra en la figura 1 *(d)*. Su configuración es similar a la de un brazo humano. está constituido por dos componentes rectos

que corresponden al antebrazo y al brazo humanos, montados sobre un pedestal vertical. Estos componentes están conectados por dos articulaciones giratorias que corresponden al hombro y al codo. Una muñeca está unida al extremo del antebrazo, con lo que se proporcionan varias articulaciones suplementarias.

1.3.2 Movimientos del robot.

Los robots están diseñados para realizar un trabajo. El trabajo se realiza permitiendo que el robot desplace su cuerpo, brazo y muñeca mediante una serie de movimientos y posiciones. Unido a la muñeca está el efector final, que se utiliza por el robot para realizar un tarea especifica. Los movimientos de las articulaciones individuales asociados con estas dos categorías se denominan, a veces, por el termino de "grados de libertad" y un robot está dotado de cuatro a seis grados de libertad.

Los movimientos del robot se realizan por medio de articulaciones accionadas. Tres articulaciones suelen estar asociadas con la acción del brazo y el cuerpo y dos o tres articulaciones se suelen emplear para accionar la muñeca. Para la conexión de las diversas articulaciones del manipulador se emplean unos elementos rígidos denominados uniones.

Las articulaciones utilizadas en el diseño de robots suelen implicar un movimiento relativo de las uniones contiguas, movimientos que es lineal o rotacional. Las articulaciones lineales implican un movimiento deslizante o de traslación de las uniones de conexión. Este movimiento puede conseguirse de varias formas. El término de articulación prismática se utiliza, en la documentación sobre robótica en lugar de articulación lineal.

Hay tres tipos de articulaciones giratorias que pueden distinguirse en los manipuladores de robots. Los tres tipos se ilustran en la figura-2. Denominaremos al primero como una articulación de tipo *R* (como rotacional). En la articulación de tipo *R* el eje de rotación es perpendicular a los ejes de las dos uniones. El segundo tipo de articulaciones giratorias implica un movimiento de torsión entre las uniones de entrada salida. El eje de rotación de la articulación de torsión es paralelo a los ejes de ambas uniones. Llamaremos a esta articulación de tipo *T* (*T* es la abreviatura de torsión). El tercer tipo de articulación giratoria es una articulación de revolución en la que la unión de entrada es paralela al eje de rotación y la de salida es perpendicular a dicho eje. Esencialmente, la unión de salida gira alrededor de la de entrada como si estuviera en órbita. Esta articulación se designara como tipo *V* (*V* procede de revolución).

Las articulaciones de brazo y del cuerpo están diseñadas para permitir al robot desplazar su efecto final a una posición deseada dentro de los limites del tamaño del robot y de los movimientos de las articulaciones. Para los robots de configuración polar, cilíndrica o de brazo articulado, los tres grados de libertad asociados con los movimientos del brazo y del cuerpo son:

Transversal vertical: es la capacidad de desplazar la muñeca hacia arriba o abajo para proporcionar la postura vertical deseada.

Transversal radial: Implica la extensión o retracción (movimientos hacia adentro o afuera) del brazo desde el centro vertical del robot.

Transversal rotacional: Es la rotación del brazo alrededor del eje vertical.

Los grados de libertad asociados con el brazo y el cuerpo del robot, se indican en la figura 2 para un robot de configuración polar. Grados de libertad similares están asociados con la configuración cilíndrica y el robot de brazo articulado. Para un robot de coordinadas cartesianas los tres grados de libertad son: movimiento vertical (movimiento eje Z), movimiento hacia adentro y afuera (movimiento eje Y), y movimiento derecha izquierda (movimiento eje X). Estos se consiguen por los movimientos correspondientes de los tres dispositivos de deslizamiento ortogonales del brazo del robot.

El movimiento de la muñeca está diseñado para permitir al robot orientar adecuadamente el efector final con respecto a la tarea a realizar.

La muñeca suele disponer de hasta tres grados de libertad:

Giro de la muñeca: También denominado oscilación de la muñeca,. Que implica la rotación del mecanisrno alrededor del eje del brazo.

Elevación de la muñeca: Habida cuenta de que el giro de la muñeca está en su posición central, la elevación implicaría la rotación arriba o abajo de la misma. La elevación de la muñeca se denomina a veces, flexión de la muñeca.

Desviación de la muñeca: De nuevo, considerando que el giro de la muñeca está en la posición central, la rotación implicaría la rotación a derecha o izquierda de la muñeca.

1.3.3 SISTEMAS DE IMPULSIÓN DEL ROBOT

La capacidad del robot para desplazar su cuerpo, brazo y muñeca se proporciona por el sistema de impulsión utilizado para accionar el robot. El sistema impulsor determina la velocidad de los movimientos del brazo, la resistencia mecánica del robot y su rendimiento dinámico.

Los robots están accionados por uno de los tres tipos de sistemas de impulsión que son:

Hidráulica

Eléctrica

Neumática

Las impulsiones hidráulica y eléctrica son los dos tipos utilizados principalmente en los robots más sofisticados.

La impulsión hidráulica suele estar asociada con los robots más grandes. La ventaja habitual del sistema de impulsión hidráulica es proporcionar al robot una mayor velocidad y resistencia mecánica. Los inconvenientes del sistema de impulsión hidráulica radican en que suelen añadir más necesidades de espacio y son propensos a fugas de aceite.

Los sistemas de impulsión eléctrica no suelen proporcionar tanta velocidad o potencia como los sistemas hidráulicos, pero la exactitud y repetibilidad de los robots de este tipo suelen ser mejores. En consecuencia, los robots de impulsión eléctricos tienden a ser mas pequeños, con menos exigencias de espacio y sus aplicaciones tienden hacia un trabajo más preciso, estos argumentos son los que nos permitieron elegir este tipo de impulsión para la realización de nuestro prototipo que presenta el tipo de impulsión eléctrico mediante el uso de baterías y por que este era necesario para la comunicación con la PC, ya que los robots de este tipo suelen utilizar la impulsión eléctrica accionada con motores paso a paso o servomotores de corriente continua; que se ven en nuestro robot en la impulsión de los motores de desplazamiento hacia adelanta y el motor que realiza la impulsión de la muñeca para la obtención de imágenes.

La economía de los dos tipos de sistemas es también un factor en la decisión adecuada de que sistemas debemos seleccionar.

La impulsión hidráulica suele reservarse para los robots más pequeños que tienen menos grados de libertad. Estos robots suelen estar limitados a simples operaciones de "coger y situar" con ciclos rápidos.

1.3.3.1 Velocidad de movimientos

Las capacidades de velocidad de los robots actuales llegan hasta un máximo de aproximadamente 1.7 m/s. En consecuencia, las velocidades pueden obtenerse por robots grandes con brazo extendidos a su distancia máxima del eje vertical del robot. Pro su puesto, la velocidad determina la rapidez con la que el robot puede realizar un ciclo de trabajo determinado.

Suele existir un a relación inversa entre la exactitud y la velocidad de los movimientos. Cuando se incrementa la exactitud requerida, el robot necesita más tiempo con el fin de reducir errores de localización en sus diversas articulaciones para conseguir la posición final deseada. El peso del objeto desplazado influye también en la velocidad operativa. Objetos más pesados significan mayor inercia y cantidad de movimiento y el robot debe accionar con más lentitud para tratar con seguridad estos factores.

Las distancias cortas pueden no permitir al robot alcanzar la velocidad operativa programada.

1.3.3.2 Sistemas de control y rendimiento dinámico

Para poder función robot debe tener un medio de controlar su sistema impulsor para la regularización adecuada d e sus movimientos.

Los robots pueden clasificarse en cuatro categorías, según sus sistemas de control, estas son:

De secuencia limitada

De reproducción con control punto a punto

De reproducción con control de recorrido continuo

Inteligentes

De las cuatro categorías, los robots de secuencia limitada representan el control de nivel más bajo y los robots inteligentes el mas sofisticado.

Los robots de secuencia limitada no utilizan servocontrol para indicar las posiciones relativas de las articulaciones. En cambio, se controlan por el posicionamiento de los interruptores de fin de carrera y/o topes mecánicos para establecer los puntos finales de desplazamiento para cada una de las articulaciones. El establecimiento de las posiciones y la secuencias de estos topes implican una puesta a punto mecánica del manipulador en lugar de una programación del robot en el sentido habitual del término.

Los robots de reproducción utilizan una unidad de control más sofisticada, en la que una serie de posiciones o movimientos son "enseñados" al robot, registrados en memoria y luego repetidos por el robot bajo su propio control. El termino "reproducción" es descriptivo de este modo operativo general. El procedimiento de enseñar y registrar en memoria se conoce como programación del robot. Los robots de reproducción suelen tener alguna forma de servocontrol para asegurar que las posiciones conseguidas por el robot son las posiciones que se le "enseñaron",

Los robots de reproducción pueden clasificare en dos categorías: robot punto a punto (PTP) y robot de trayectoria continua (CP). Los robots punto a punto son capaces de realizar ciclos de movimientos que consisten en una serie de localizaciones de puntos deseados y acciones afines. Al robot se le enseña cada punto, y estos puntos se registran en la unidad de control del robot. Durante la reproducción, el robot se controla para desplazarse desde un punto a otro en la secuencia adecuada. Los robots punto a punto no controlan la trayectoria tomada por el robot para pasar de un punto al siguiente.

Los robots de trayectoria continua son capaces de realizar ciclos de movimientos, en los que se controla la trayectoria seguida por el robot. El movimiento en línea recta es una forma común de control de trayectoria continua. El programador especifica el punto inicial y el punto final de la trayectoria y la unidad de control calcula la secuencia de los puntos individuales que permiten al robot seguir una trayectoria de línea recta.

Los **robots inteligentes** constituyen una clase cada vez más numerosa de los robots y capacidad no solo para reproducir un ciclo de movimiento programado,

sino para interaccionar con su entorno de una manera que parece inteligente. Los robots inteligentes pueden modificar su ciclo programado en respuesta a las condiciones particulares que se produzcan en el lugar de trabajo. Pueden tomar decisiones lógicas basadas en los datos del sensor recibidos desde la operación.

1.3.3.3 Velocidad de respuesta y estabilidad

La velocidad de respuesta y la estabilidad son dos características importantes del rendimiento dinámico en relación con el diseño de los sistemas de control. La velocidad de respuesta se refiere a la capacidad del robot para desplazarse a la siguiente posición en un breve periodo de tiempo. Este tiempo de respuesta está evidentemente relacionado con la velocidad de movimiento del robot. En robótica la estabilidad se suele definir como medida de las oscilaciones que se reproducen en el brazo durante el movimiento de una posición a la siguiente. En el diseño de los sistemas de control suele ser deseable que el robot tenga una buena estabilidad y un tiempo de respuesta corto.

1.3.4 PRECISIÓN DE MOVIMIENTO

Otra medida del rendimiento es la precisión del movimiento del robot. Definiremos la precisión como una función de tres características:

Resolución espacial.

Exactitud.

Repetibilidad.

1.3.4.1 Resolución espacial

La resolución espacial de un robot se defino como el más pequeño incremento de movimientos en el que el robot puede dividir el volumen de trabajo.

La resolución de control viene determinada por el sistema de posición del robot y su sistema de medida de realimentación. Es la capacidad del controlador para dividir el margen total de movimiento para la articulación particular en incrementos individuales, que pueden regularse en el controlador. Los incrementos se denominan, como "puntos direccionables". La capacidad para dividir el margen de las articulaciones en incrementos depende de la capacidad de almacenamiento de bits en la memoria de control. El número de incrementos identificables separados (puntos direccionables) para un eje en particular viene dado por:

Numero de incrementos = 2^n

En donde n = número de bits en la memoria del control.

Por ejemplo un robot de 8 bits de almacenamiento puede dividir el margen en 256 posiciones distintas. La resolución de control se definirá como el margen de movimientos total dividido por el número de incrementos.

1.3.4.2 Exactitud

La exactitud se refiere a la capacidad de un robot para situar el extremo de su muñeca en un punto de destino deseado dentro del volumen de trabajo. La exactitud de un robot puede definirse en términos de resolución espacial, porque la capacidad para alcanzar un punto de destino determinado depende de cuán próximos pueda el robot definir los incrementos de control para cada uno de sus movimientos de las articulaciones. Podríamos definir inicialmente la exactitud, como una mitad de la resolución de control.

Nuestra definición de exactitud se aplica al caso mas desfavorable, en donde el punto destino está directamente entre dos puntos de control. En segundo lugar, la exactitud se mejora si el ciclo de movimiento está restringido a un margen de trabajo limitado. Los errores mecánicos tenderán a reducirse cuando el robot esté sometido a u rango restringido de movimientos.

1.3.4.3 Repetibilidad.

La repetibilidad está relacionada con la capacidad del robot para situar su muñeca, o un efector final unido a su muñeca, en un punto en el espacio que se hubiera "enseñado" con anterioridad al robot. La repetibilidad y la exactitud se refieren a dos aspectos diferentes de la precisión del robot. La exactitud se relaciona con la capacidad del robot para conseguir un punto destino deseado. La repetibilidad se refiere a la capacidad del robot para volver al punto programado cuando se la ordena que lo haga.

1.3.4.4 Control coordinado de fuerza y posición

Una característica del robot que está relacionada con nuestra exposición anteriror es el control coordinado de fuerza y posición. Dicho control del manipulador de¡ robot se refiere al desplazamiento del extremo (le lit muñeca en respuesta a una fuerza o tesión que se ejerza sobre él. A veces se utiliza el término <<elástico>> para describir un robot con un alto valor de esta característica.

1.3.4.5 Efectores finales

Podríamos denominar a estos dispositivos como los periféricos del robot.

En robótica, el término de efector final se utiliza para describir la mano o herramienta que está unida a la muñeca. El efector final representa el herramental especial que permite al robot de uso general realizar una aplicación particular. Este heramiental especial debe diseñarse específicamente para la aplicación.

1.4 SENSORES ROBÓTICOS

Transductores y sensores.

Un transductor es un dispositivo que transforma un tipo de variable física (Ej: fuerza, presión, temperatura, velocidad, etc.) en otro. Un sensor es un transductor que se utiliza para medir una variable física de interés.

Los transductores y los sensores pueden clasificarse en dos tipos básicos, dependiendo la forma de la señal convertida. Los tipos son:

Transductores analógicos.

Transductores digitales.

Los transductores analógicos proporcionan una señal analógica continua, como por ejemplo voltaje o corriente eléctrica. Esta señal puede ser tomada como el valor de la variable física que se mide. Los transductores digitales producen una señal de salida digital, en la forma de un conjunto de bits de estado en paralelo o formando una serie de pulsaciones que, pueden ser contadas. -En una u otra forma, las señales digitales representan el valor de la variable medida. Los transductores digitales han llegado a ser más populares a causa de la facilidad con la que se pueden emplear como instrumentos de medición independientes. Además, suelen ofrecer la, ventaja de ser más compatibles con las computadoras digitales que los sensores analógicos en la automatización y en el control de procesos. ,

1.4.1 Características deseables de los sensores

Exactitud. La exactitud de la medición debe ser tan alta corno fuese posible. Se entiende por exactitud que el valor verdadero de la variable se pueda detectar sin errores sistemáticos positivos o negativos en la medición. Sobre varias mediciones de la variable, el promedio de error entre el valor real y el valor detectado tenderá a ser cero.

Precisión. La precisión de la medición debe ser tan alta como fuese posible. La precisión significa que existe o no una pequeña variación aleatoria en la medición de la variable. La dispersión en los valores de una serie de mediciones será mínima.

Rango de funcionamiento. El sensor debe tener un amplio rango de funcionamiento y debe ser exacto y preciso en todo el rango.

Velocidad de respuesta. El transductor debe ser capaz de responder a los cambios de la variable detectada en un tiempo mínimo. Lo ideal sería una respuesta instantánea.

Calibración. El sensor debe ser fácil de calibrar. El tiempo y los procedimientos necesarios para llevar a cabo el proceso de calibración deben ser mínimos. Además, el sensor no debe necesitar una recalibración frecuente. El término

«desviación» se aplica con frecuencia para indicar la pérdida gradual de exactitud del sensor que se produce con el tiempo y el uso, lo cual hace necesaria su recalibración.

Fiabilidad. El sensor debe tener una alta fiabilidad. No debe estar sujeto a fallos frecuentes durante el funcionamiento.

Coste y facilidad de funcionamiento. El coste para comprar, instalar y manejar el sensor debe ser tan bajo como sea posible, Además, lo ideal sería que la instalación y manejo del dispositivo no necesite de ningún operador altamente cualificado.

1.4.2 Sensores en Robótica.

Los sensores utilizados en robótica incluyen una amplia gama de dispositivos que se pueden dividir en las categorías siguientes:

Sensores táctiles.

Sensores de proximidad y alcance.

Sensores diversos y sistemas basados en sensores.

Sistemas de visión de máquina.

1.4.2.1 Sensores táctiles. Se trata de sensores que responden a fuerzas de contacto con otro objeto. Algunos de estos dispositivos son capaces de medir el nivel de fuerza implicada.

1.4.2.2 Sensores de proximidad y alcance. Un sensor de proximidad es un dispositivo que indica cuándo un objeto está próximo a otro objeto, pero antes de que se produzca el contacto. Cuando puede detectarse la distancia entre los objetos, el dispositivo se denomina un sensor de alcance

1.4.2.3 Tipos diversos. La categoría de diversos incluye las clases restantes de sensores que se utilizan en robótica. Se incluye los sensores para temperatura, presión y otras variables.

Visión de máquina. Un sistema de máquina es capaz de «visionar» el espacio de trabajo e interpretar lo que ve. Estos sistemas se emplean en robótica para realizar tareas de inspección, reconocimiento de piezas y otras similares.

CAPITULO II

IMPLEMENTACIÓN DEL ROBOT PARA EL RECONOCIMIENTO VISUAL

2.1 Una Intrcducción a la Visión de Máquina

Los sensores proporcionan al robot información sobre su entorno, que él puede usar para guiar sus acciones. La visión es un sentido muy importante porque puede dar información de una resolución relativamente alta a distancias relativamente grandes. Se han desarrollado muchos tipos de sistemas de visión para robots, los cuales entran en uso práctico cada vez con más frecuencia.

Los sistemas de visión para máquinas comenzaron a aparecer en el decenio iniciado en 1950, asociados con aplicaciones no robóticas, como lectura de documentos, conteo de especimenes biológicos en la sangre, y reconocimiento aerofotográfico de aplicación militar. En el campo propiamente robótico el desarrolló comenzo a mediados del decenio iniciado en 1960, con el establecimiento de laboratorios de inteligencia artificial en instituciones como el MIT, la U. de Stanford y el Stanford Research Institute. Para 1980 muchas compañías estaban desarrollando sistemas de visión para robots y para otras aplicaciones industriales.

2.1.1.1 VISION BIDIMENSIONAL

El objetivo general de una máquina vidente es derivar una descripción de una escena, analizando una o mas imágenes de dicha escena. En algunas situaciones la escena misma es básicamente bidimensional. Por ejemplo si el robot trabaja con piezas planas sobre una superficie plana, o si busca perforaciones en tal superficie. La visión para situaciones bidimensionales es más fácil que para las tridimensionales, y, como era natural, los primeros trabajos sobre máquinas videntes se hicieron en esa modalidad.

2.1.1.1.1 El proceso de visión bidimensional

Para que un robot reconozca partes, perforaciones, etc. en una superficie, y en general objetos, Primero debe distinguir los objetos de interés del resto de la superficie. En otras palabras, debe ser capaz de "destacar" partes de la imagen que corresponden a esos objetos. Este proceso de extraer subconjuntos de una imagen que corresponden a partes relevantes de la escena se denomina SEGMENTACION.

Cuando se ha extraído un subconjunto de una imagen, generalmente es necesario medir varias propiedades geométricas de tal subconjunto (tamaño, forma, etc.) tales medidas pueden ser la base para reconocer si el subconjunto representa un objeto dado, así como para determinar la posición y orientación de tal objeto. También pueden servir de base para una posterior segmentación dentro del subconjunto; por ejemplo, si dos objetos se tocan o superponen, puede ocurrir que hayan sido extraídos como un solo subconjunto, y puede ser necesario dividir este subconjunto en dos partes basándose en criterios geométricos, por ejemplo descomponiéndolo en partes convexas. Esta etapa del proceso de visión se denomina ANALISIS GEOMETRICO. Se pueden diseñar diferentes algoritmos para Análisis Geométrico, dependiendo de la manera como

estén representados los subconjuntos de imagen dentro del computador, por esto el tema de la representación geométrica de subconjuntos de la imagen está estrechamente relacionado con el Análisis Geométrico.

El reconocer objetos por medio del análisis de subconjuntos de una imagen puede variar mucho en dificultad, dependiendo de la complejidad de los objetos. Si los objetos que pueden estar presentes en la escena difieren mucho entre sí, se puede usar una comparación relativamente sencilla contra patrones o plantillas; en esta situación, puede incluso ser innecesario extraer explícitamente los objetos del resto de la imagen. Más a menudo, los objetos pueden reconocerse porque cumplen un conjunto característico de valores de sus parámetros geométricos. Este método de comparación de rasgos geométricos se usó, por ejemplo en un módulo de visión desarrollado en el SRI Internacional a mediados de los años 70. Si los objetos a reconocer son complejos, puede ser necesario descomponer en etapas el proceso de reconocimiento: Detectar primero subobjetos y reconocer sus propiedades, y luego reconocer los objetos como combinaciones de esos subobjetos en relaciones específicas. este procedimiento se conoce como el método de comparación de estructuras.

2.1.1.1.2 Segmentación

Una imagen digital es una matriz de números que representan valores de iluminación en puntos regularmente espaciados de la imagen de una escena. Los elementos de más bajo nivel de tal imagen se llaman PIXELs (contracción de "Picture Element"), y sus valores se denominan NIVELES DE GRIS (el color casi no ha sido utilizado aún en los sistemas de visión robóticos). La efectividad de una técnica de Segmentación depende de las propiedades de la clase de imágenes a que se aplique.

2.1.1.1.3 Establecimiento de umbrales

Si el brillo de un objeto difiere significativamente del de su entorno, origina en la imagen un conjunto de pixels con Niveles de Gris muy diferentes de los Niveles de los pixels circundantes. (A menudo pueden producirse grandes diferencias de brillo entre un objeto y su entorno controlando el ángulo en que incide la luz). Los subconjuntos de imagen con esa característica pueden extraerse de la imagen estableciendo un UMBRAL para los Niveles de Gris, o sea clasificando cada pixel como "claro" u "oscuro" dependiendo de si su Nivel de Gris es inferior o superior al umbral.

Si la iluminación de la escena está bajo nuestro control, y podemos también calibrar el sensor, puede ser posible establecer un umbral permanente para segmentar correctamente escenas de determinada clase, pero, en general, será necesario determinar el umbral óptimo para cada imagen individual. Si los objetos ocupan una fracción significativa de la escena, esa determinación se puede hacer analizando el HISTOGRAMA DE LA IMAGEN, el cual es un gráfico que nos muestra la frecuencia con que ocurre en la imagen cada Nivel de Gris. Este histograma debe tener dos picos, uno que representa el Nivel de Gris

predominante en el fondo, y otro el predominante en el objeto. Los Niveles intermedios deben ser relativamente infrecuentes, correspondiendo en el histograma a valles entre los picos. Evidentemente, en estos casos un buen Nivel en el cual establecer el Umbral es el más infrecuente entre los Niveles de los dos picos (o sea correspondiente al punto más bajo del valle entre picos), ya que casi todos los pixels del objeto caen a un lado de tal umbral, y casi todos los pixels del fondo caen al otro lado.

Si la iluminación de la escena no es uniforme, puede suceder que objetos intrinsecamente oscuros a un lado de la escena resulten más brillantes que un fondo intrinsecamente claro en otro lado, de tal manera que los objetos no pueden ser separados del fondo por la simple comparación con un umbral. Una manera de atacar esta situación es dividir la imagen en secciones y escoger un umbral adecuado para cada sección analizando su propio histograma. Posteriormente esos histogramas pueden ser interpolados para obtener un umbral variable que segmente convenientemente la imagen completa.

2.1.1.1.4 Bases de las visión en color para máquinas

Como se mencionó antes, la visión robótica a color a sido poco estudiada en general, aunque tiene un papel muy importante en aplicaciones específicas. El color en un punto de una imagen puede representarse por tres números que representen, por ejemplo, los niveles de las componentes roja, verde y azul. Entonces una Imagen Digital a Color es una matriz de tripletas de valores. Si esos niveles de los pixels se grafican como puntos de un "espacio cromático", el objeto y el fondo originan cúmulos de puntos. Tales cúmulos son análogos a los picos del histograma y la imagen se puede segmentar en regiones de diferente color parcelando el espacio cromático de tal manera que se separen los cúmulos. Este ha sido el método clásico utilizado para segmentar las imágenes obtenidas por los sensores remotos multiespectrales, pero aún no tiene utilización extensiva en visión robótica.

2.1.1.1.5 Detección de bordes

Los objetos pequeños no se extraen fácilmente del fondo en que se ven, utilizando el método del umbral, porque los picos que producen en el histograma son demasiado pequeños para detectarlos con confiabilidad. Similarmente, si una escena contiene muchos objetos de diferentes brillos, tampoco es fácil extraerlos por el método del umbral, porque sus picos respectivos en el histograma se superponen. En tales casos se puede utilizar otros métodos para segmentación, con tal de que los objetos posean un brillo relativamente uniforme y que contrasten fuertemente con su inmediato entorno. Esto implica que la variación de nivel de gris ocurre muy gradualmente dentro de cada objeto, y rápidamente en los bordes de objetos. En tal caso los objetos se pueden extraer por Detección de Bordes, o sea detectando los pixels en los cuales la rata de cambio del Nivel de Gris respecto al espacio es alta.

El método clásico de detectar bordes en una imagen es aplicar un operador derivada isotrópico, tal como el Gradiente. Tal operador dará valores altos en los bordes, sin importar sus orientaciones. Para este fin se han utilizado muchas aproximaciones discretas al gradiente, como el operdor cruzado de Roberts, y el operador de Sobel usado en los primeros sistemas de visión de Stanford.

Otros métodos básicos de detección de bordes incluyen la comparación de la vecindad de cada pixel con patrones o plantillas de Funciones Paso en diferentes orientaciones; si se detecta un buen ajuste, es muy probable que haya un borde en esa orientación. Otro método consiste en ajustar una superficie polinomial a los Niveles de Gris en la vecindad de cada pixel; si el gradiente de la superficie ajustada es alto, es probable que haya un borde.

2.1.1.1.6 Análisis de textura

Si un objeto no presenta un brillo uniforme, sino que muestra cierta "trama", ni el umbral ni la detección de bordes son útiles para extraerlo, ya que sus pixels no poseen Niveles de Gris en un rango estrecho y presenta muchos "bordes" internos. No obstante, tal objeto puede ser distinguible de su vecindad basándonos en su trama o patrón característico de Niveles de Gris o TEXTURA VISUAL.

Las Texturas Visuales se pueden caracterizar por conjuntos de propiedades locales de sus pixels, o sea por el hecho de que en una región texturada, tienden a presentarse ciertos patrones locales de Niveles de Gris en la vecindad de cada pixel. Cada pixel puede caracterizarse por un conjunto de números calculando para él un conjunto de propiedades relacionadas con la textura, para segmentar la imagen en regiones de texturas diferentes. Este procedimiento tiene cierta analogía al tratamiento de imágenes a color que se mencionó antes, pero como las propiedades texturales tienden a ser más variables que los colores, debe obtenerse primero cierto promedio local para hacer más compactos los cúmulos correspondientes a cada región. Similarmente, calculando los valores promedios de las propiedades locales y tomando luego diferencias de esos promedios se puede calcular un "gradiente de textura" en cada pixel y usarlo para detectar bordes entre regiones de diferentes texturas.

Un método potente para caracterizar texturas consiste en realizar varios tipos de operaciones de adelgazamiento y expansión de regiones y analizar sus resultados; por ejemplo, las tramas visuales delgadas desaparecen al aplicar un poco de adelgazamiento, mientras las tramas con espaciamientos pequeños se funden al aplicar un poco de expansión. Este método de análisis de imagen ha sido usado en gran variedad de aplicaciones durante unos 20 años.

2.1.1.1.7 Acreción de regiones

Los métodos de segemtación discutidos hasta ahora tratan cada pixel o su vecindad independientemente; no les concierne si los pixels resultantes constituyen una región conectada o si los segmentos de bordes resultantes

constituyen una frontera suave de alto contraste. Se pueden obtener regiones o bordes de mejor calidad condicionando los resultados a que sean localmente consistentes, o sea que las regiones estén conectadas o que los bordes se encadenen suavemente. Para asegurar continuidad se puede utilizar métodos de seguimiento secuencial de los bordes, o acreción de regiones. Un método más poderoso, pero computacionalmente más costoso es buscar consistencia global por partes, o sea buscar regiones que sean óptimas con respecto a la consistencia o suavidad del Nivel de Gris, o bordes que sean óptimos respecto a contraste y a suavidad de la dirección. Un método útil para hallar regiones globalmente consistentes, es un proceso de "dividir y unir", en el cual las regiones se dividen si son inconsistentes, y se unen pares de regiones adyacentes si su unión es consistente.

2.1.1.1.8 Análisis Geométrico

Una vez que en una imagen se ha segmentado una región , esta se puede representar por una imagen binaria, en la cual los pixels que pertenecen a la región tienen valor 1, y los del fondo tienen valor 0. De esta imagen binaria se pueden calcular varias propiedades geométricas de la región. Este proceso se denomina a veces como visión binaria.

2.1.1.1.9 Conectividad y bordes

Si una escena contiene varios objetos sobre un fondo, la segmentación de su imagen dá el conjunto de pixels pertenecientes a todos los objetos; la segmentación no ha distinguido los objetos entre si. Para trabajar con cada objeto es necesario "etiquetar" o marcar los pixels de los objetos, de tal manera que los que pertenecen a un mismo objeto tengan la misma marca, y viceversa. Este proceso se denomina MARCACION DE COMPONENTES CONECTADOS, y asigna una marca distintiva única a cada conjunto de pixels que están interconectados. El hecho de estar mutuamente conectados es el principio básico del proceso de contar objetos (o sea regiones internamente conectadas) en una imagen.

El borde de una región es el conjunto de pixels adyacentes a pixels no pertenecientes a la región. Tales pixels de frontera caen en un conjunto de curvas, correspondiendo una de ellas al borde externo de la región, y los demás a agujeros, si los hay. Para marcar esos bordes individualmente se puede utilizar un método de Seguimiento de Bordes, que comienza en un pixel del borde y visita sucesivamente todos los pixels que pertenecen al mismo borde, hasta retornar al pixel inicial.

2.1.1.1.10 Propiedades de tamaño y forma

El área de una región se puede medir aproximadamente en función del número de sus pixels, o sea el número de pixels que tienen una marca particular. A su vez, el perímetro es función del número de pixels del borde externo, o (para un borde específico) el número de movimientos necesarios para dar la vuelta por el

borde pixel por pixel. Un parámetro geométrico usado frecuentemente y que mide lo compacto de una región es: C= Area/(perímetro)2. La elongación de una región se puede definir usando un proceso de adelgazamiento y medición de área: una región es elongada si aunque tenga gran área desaparece al aplicarle un ligero adelgazamiento. Las medidas de distancia son otra fuente de información útil sobre las formas.

Muchas propiedades formales de una región se pueden derivar midiendo la curvatura (o sea la tasa de cambio de dirección) de su borde externo. Son concavidades las partes del borde con curvatura negativa, o sea aquellas donde la dirección cambia el sentido opuesto a las agujas del reloj cuando el borde se sigue en el sentido de dichas agujas. Las esquinas son puntos del borde donde la curvatura pose un valor absoluto muy alto. Tales propiedades formales son útiles para segmentar una región en caso de ser necesario, por ejemplo cuando dos objetos de la escena se tocan o se superponen parcialmente.

2.1.2 Representaciones geométricas

Una región no tiene que ser representada por el conjunto de unos en su imagen binaria, se pueden usar otras representaciones más compactas si la forma de la región es simple. Las siguientes son algunas de las más conocidas:

2.1.2.1 Código de Longitud de Secuencia.

Cada fila de la imagen binaria consiste en secuencias de unos alternadas con secuencias de ceros. Así que una fila se determina completamente con especificar el valor inicial 0 o 1 y las longitudes de las cadenas. La mayoría de las propiedades geométricas de una región pueden medirse directamente de su código de Longitud de Secuencias. Este método de codificar una imagen ha sido muy utilizado en compresión de datos.

2.1.2.2 Código de Encadenamiento.

La secuencia de movimientos hechos al seguir un borde, junto con las coordenadas del punto de arranque, determinan completamente dicho borde; esta secuencia de movimientos se denomina el código de encadenamiento del borde. Una región queda determinada al especificar sus bordes de este modo, y muchas propiedades de la región se pueden calcular directamente a partir de esta codificación de los bordes. El mismo proceso se puede utilizar para codificar y procesar curvas digitalizadas.

2.1.2.3 Árbol Tetrafurcado ("Quadtree")

Cualquier región es la unión de los bloques máximos de pixels (ej: cuadrados) que están contenidos en la región; así que la región queda determinada al especificar la lista de centros y tamaños de esos bloques (cuadrados). Tal representación es muy compacta pero algo difícil de trabajar puesto que es una lista no ordenada. Una representación menos compacta pero más tratable es la denominada Árbol Tetrafurcado ("quadtree") que se obtiene al subdividir

recursivamente la imagen binaria en cuadrantes, deteniéndose la subdivisión de un bloque cuando todos sus pixels son iguales. El proceso de subdivisión recursiva define una estructura de árbol que se va ramificando en de a cuatro ramas; el árbol facilita el procesamiento para cálculos geométricos.

2.1.2.4 Otras Representaciones

También son útiles varios métodos para aproximar la forma de una región. La frontera de una región puede ser aproximada por un polígono. Una región similar a una cinta posee un eje medio en el cual los centros de los cuadrados máximos caen aproximadamente a lo largo de una curva (el esqueleto de la cinta); entonces la región se puede aproximar especificando esa curva junto con una función de amplitud que especifica cómo varían los tamaños de los cuadrados al movernos sobre la curva. Las aproximaciones poligonales sucesivas a una curva se pueden organizar en una estructura de árbol, en la cual un nodo representa el lado de un polígono que aproxima un arco dado de la curva y los hijos de ese nodo representan los lados de un polígono refinado que aproximan subarcos del arco.

2.1.3 Reconocimiento en escenas bidimensionales.

En los ambientes de trabajo en que nos podemos limitar a dos dimensiones, asumimos que los objetos a reconocer son relativamente planos y que yacen sobre una superficie.

2.1.4 Ajuste a plantilla y transformada de Hough

Si la forma del objeto deseado se conoce de antemano con precisión, este es posible que se reconozca por medio de un proceso de comparación contra patrones o plantillas. Un método primitivo sería comparar la plantilla contra la imagen, pixel por pixel, en todas las posiciones y orientaciones bidimensionales posibles. La comparación se puede basar en diferencias a nivel de pixel, o calculando alguna función de correlación. Este es un método computacionalmente costoso que se usa sólo para detectar partes pequeñas de objetos, tales como bordes rectos o esquinas, usando plantillas que constan de pocos pixels. Si de este modo podemos detectar un conjunto de rasgos distintivos del objeto, luego se pueden buscar combinaciones de esos rasgos en posiciones relativas adecuadas para determinar el objeto en sí.

Un método menos costoso computacionalmente, conocido como la transformada de Hough, se puede utilizar para objetos que aunque son extensos, contienen relativamente pocos pixels, por ejemplo una curva delgada, o el borde de un objeto. La idea básica es mapear esos pixels hacia un espacio transformado definido de tal manera que los pixels pertenecientes a la curva o borde de interés se proyecten sobre un mismo punto en ese espacio. Si en la imagen hay presentes muchos pixels que pertenecen todos a la misma curva, ellos darán origen a un pico en el espacio de Hough.

2.1.4.1 Ajuste a características

Si los objetos que se espera encontrar en la escena son suficientemente diferentes entre sí, a menudo es posible distinguirlos midiendo los valores de un conjunto de propiedades, sin requerir una confrontación contra plantillas. Las propiedades usadas pueden ser las de índole geométrica mencionadas antes, o también relacionadas con el color o la textura. Para cada región candidata en la imagen se le mide el conjunto de propiedades y se compara con los valores ideales que caracterizan a los varios tipos de objetos. Se ha reconocido que una región R es un objeto X cuando las medidas para R coinciden aproximadamente con los valores típicos de X.

2.1.4.2 Reconocimiento Estructural

Si los objetos son relativamente complejos puede no ser posible distinguirlos entre sí con base en conjuntos de valores de ciertas características, ya que los rangos de valores que caracterizan diferentes objetos pueden no ser suficientemente distintos. En tal caso se puede intentar un método estructural. Se extraen de la imagen partes de los objetos y se caracterizan por valores de parámetros. Luego se hallan combinaciones de esas partes, en relaciones determinadas, las cuales representan partes más complejas u objetos completos. Este proceso se puede repetir en varias etapas, si fuere necesario, hasta que se identifican los objetos deseados. Este método se conoce como RECONOCIMIENTO ESTRUCTURAL, o RECONOCIMIENTO SINTACTICO ya que tiene analogía con el análisis sintáctico del lenguaje.

2.1.5 VISION TRIDIMENSIONAL

La visión es mucho más compleja si requerimos tratarla tridimensionalmente, por ejemplo si el robot debe trabajar con una pila de objetos, o tiene que determinar la orientación de un objeto en el espacio. En tales escenas las superficies de los objetos pueden tener orientaciones variables y, por ejemplo, un objeto uniformemente reflectante puede no aparecer con brillo uniforme.

También es difícil obtener información uniforme; las sombras son inevitables. Si se pueden calcular los efectos de la iluminación y de la orientación de las superficies, se puede corregir la información de brillo de la imagen, de manera que esta presente la reflectividad intrínseca de las superficies para luego segmentar la imagen en regiones que correspondan a superficies uniformemente reflectivas. Además, si se conoce la orientación de una superficie en todos los puntos, se puede segmentar la imagen en regiones dentro de las cuales no hay cambios abruptos en la orientación y que entonces probablemente pertenecen a las superficies de un mismo objeto. Pero aún si se logra un segmentación correcta, el reconocimiento del objeto tridimensional aún es difícil porque tal objeto puede originar en la imagen regiones de formas (bidimensionales) muy variadas según su orientación. Además, sólo vemos un lado del objeto desde un punto de vista dado, y los objetos pueden ocultarse

parcialmente entre sí, lo cual significa que el reconocimiento se debe basar en datos incompletos.

La investigación a fondo de la visión tridimensional artificial comenzó a mediados de los 60 en los principales laboratorios de inteligencia artificial. Se desarrollaron muchas técnicas para recuperar información sobre la orientación y las formas tridimensionales de las superficies visibles en una escena. Los método utilizados incluyen medición de la distancia, mapeado estéreo, y la inferencia sobre la forma de la superficie a partir de indicios presentes en una sola imagen. Adicionalmente, se desarrollaron métodos para la información morfológica tridimensional y se realizó un trabajo básico sobre el reconocimiento de objetos tridimensionales parcialmente visibles y con orientación desconocida.

Las técnicas de visión tridimensional tienden a ser computacionalmente costosas, y pocas han sido aplicadas fuera de ambientes experimentales.

2.1.5.1 Recuperación de forma superficial a partir de una sola imagen

El nivel de gris de un pixel en una imagen digital representa la cantidad de luz recibida por el sensor desde una dirección dada, o sea el brillo aparente de un punto P dado en la superficie de algún objeto en la escena. Ese valor de brillo es el resultante de, al menos, tres factores: Nivel de iluminación en P, reflectividad de la superficie en P y la orientación espacial de tal superficie relativa a la dirección de la mirada. No es posible separar los efectos de esos factores si solamente conocemos el brillo del punto P. Sin embargo, conociendo toda la imagen, y bajo ciertas condiciones de la escena y su iluminación, se pueden obtener algunas conclusiones razonables.

2.1.5.2 Forma a partir de sombreado

En la vecindad inmediata de un punto P, la iluminación tiende a ser aproximadamente constante, excepto en el borde de una sombra. Entonces, si P está en una superficie uniformemente reflectiva, las variaciones del brillo (sombreado) en la vecindad de P, se deben principalmente a variación en la orientación de la superficie. Las variaciones del brillo no determinan completamente la orientación de la superficie, pero sí la restringen, y si se conocen en ciertos puntos de una región esto puede permitir el determinar la orientación en el resto de la región

2.1.5.3 Forma a partir de textura

Si una superficie tiene una textura o trama visual uniforme, dará origen a una región de la imagen en la cual dicha trama no es uniforme, debido a efectos de la orientación de la superficie. La dirección de la superficie, en principio, se puede deducir a partir de mediciones locales de la anisotropía de la trama visual de la imagen. Una técnica relacionada con este principio consiste en iluminar la escena con un patrón o trama regular de luz y sombras (ejemplo: rejilla uniforme de huecos cuadrados). Las distorsiones de tal trama, como aparecen en la

imagen, proporcionan información a cerca de las orientaciones superficiales. Tal tipo de iluminación recibe el nombre de "estructurada".

2.1.5.4 Forma tridimensional a partir de forma bidimensional

También se pueden obtener indicios acerca de las formas tridimensionales de las superficies de la escena a partir de las formas bidimensionales de los rasgos. Un ejemplo primitivo de este concepto, usando aristas lineales, demostró que las formas de las esquinas donde se unen las aristas dan información útil sobre cuales superficies de las que se encuentran en una de esas uniones pertenecen a un mismo objeto. Posteriormente se generalizó este trabajo basándose en que un borde en una imagen puede originarse por varios tipos de cambios abruptos en la escena, incluyendo cambios en la iluminación (bordes por sombra), ocultamiento parcial, o marcas en una misma superficie. Cuando las aristas se reúnen en un punto (vértice) no son posibles todas las combinaciones de esos casos, y cuando se consideran todas esas uniones en la imagen, se reduce mucho el numero de posibilidades. En principio es posible distinguir entre varios tipos de aristas analizando cuidadosamente las variaciones de brillo en su vecindad. Cuando superficies planas se encuentran en un vértice, las orientaciones espaciales de las superficies están restringidas, y esto da información útil acerca de la forma de poliedros.

Las formas de los bordes o regiones en una imagen, también pueden dar información cuantitativa sobre la forma de la superficie si se hacen ciertas asumsiones respecto a los extremos. Por ejemplo, una curva en una imagen puede ser producida por muy diferentes curvas en el espacio, pero muchas veces se puede asumir que la curva real en el espacio es la que tiene la menor curvatura posible. Similarmente, una región en una imagen puede originarse por diversos trozos de superficie en el espacio, pero se puede asumir que el trozo real de superficie es el que tiene la forma más simple, o sea la más simétrica y compacta.

2.1.5.5 Detección de distancia y visión en estéreo

Si disponemos de dos imágenes de una misma escena tomadas de dos posiciones diferentes conocidas, y si podemos identificar en ambas los dos puntos correspondientes a un mismo punto P de la escena, entonces se puede calcular por triangulación la posición del punto P en el espacio. La principal dificultad es el identificar los pares correspondientes de puntos de imagen.

También se puede derivar la forma de la superficie a partir de múltiples imágenes tomadas desde una misma posición, pero bajo diferentes condiciones de iluminación. El sombreado en cada imagen impone restricciones en las orientaciones de las superficies en cada punto, y la intersección de esas restricciones determina la orientación sin ambigüedad. En esta técnica, conocida como "estéreo fotométrico" no hay problema en identificar un par de puntos correspondientes, puesto que las imágenes están en perfecto registro.

La información sobre la forma de las superficies estaría inmediatamente disponible si se pudiera medir directamente la distancia a cada punto superficial visible. Se han desarrollado varios tipos de sensores de distancia. Un método es iluminar una parte de la escena cada vez, por ejemplo con un plano de luz; la posición espacial de un punto iluminado puede quedar completamente determinado por su posición en la imagen, ya que debe quedar a lo largo de una dirección espacial dada (la dirección de la visión) y también sobre el plano de luz. Otro método consiste en iluminar la escena punto por punto, con un pulso de luz. y medir la distancia hasta el punto iluminado por medio del desplazamiento de fase o retardo entre el envió del pulso y el retorno del reflejo.

2.1.5.6 Reconocimiento de objetos tridimensionales

En la imagen segmentada de una escena tridimensional las regiones representan parches de superficie. El reconocimiento de objetos tridimensionales a partir de una imagen bidimensional es difícil porque solamente se ve un lado de los objetos, aún ignorando la posible ocultación de un objeto por otro, y las formas de las partes visibles dependen de la orientación del objeto. Si se sabe de antemano qué posibles objetos pueden aparecer en la escena, una técnica sería almacenar descripciones de esos objetos como se verían desde varias decenas de puntos de vista, para hacer una comparación de formas, pero esto es computacionalmente costoso.

Se pueden definir descripciones tridimensionales de objetos de varias maneras. Un objeto se puede determinar especificando las formas de sus superficies, las cuales se pueden aproximar por "parches" de varios tipos estándar, (Como en el caso bidimensional, las aproximaciones sucesivas se pueden organizar en un árbol). Un objeto también se puede aproximar por la unión de "cubos máximos" de varios tipos. Los cubos máximos se pueden organizar en una estructura de árbol ramificado de a 8 ramas ("octree"), definido subdividiendo recursivamente el espacio en octantes.

Dado un conjunto de tales descripciones tridimensionales de objetos, estos se pueden reconocer en la imagen por un proceso de análisis de restricciones. Una región o borde en la imagen no puede ser proyección de cada posible borde o superficie de un objeto; sólo se puede originar de un subconjunto de los objetos posibles, y estos tiene que estar en un subconjunto de las posibles orientaciones espaciales. Si un conjunto de bordes o regiones es consistente, en este sentido, con la presencia de cierto objeto en cierta orientación, hay evidencia fuerte de que este objeto está presente en la escena

2.1.6 OTROS TOPICOS

2.1.6.1 Análisis de imagen variable en el tiempo

Hasta aquí nos hemos referido solamente a imágenes estáticas, pero cada vez se investiga más la dinámica de las escenas, usando secuencias de imágenes.

2.1.6.2 Métodos de multiresolución

Muchos tipos de operaciones sobre imágenes, tales como la detección de bordes, la acreción de regiones, la comparación de formas, etc. se pueden realizar eficientemente usando un método grueso-fino, en el cual la imagen se procesa primero a baja resolución, y los resultados se usan luego para guiar el proceso a resolución más alta. También muchas operaciones se pueden realizar con una estrategia de "dividir y conquistar", en la cual los resultados obtenidos localmente a alta resolución se pueden combinar recursivamente hacia resultados globales

2.1.6.3 Computación paralela

Muchos pasos de los procesos de visión se pueden implementar como procesos cooperativos, el los cuales se realizan en paralelo las operaciones locales, por ejemplo en cada pixel o en cada nodo de una estructura de datos.

2.1.6.4 Restricciones computacionales

Muchas de las operaciones para la visión involucran masivas cantidades de cómputo. En la actualidad algunas operaciones no son comercialmente viables. Por esta razón, por ejemplo, los sistemas de visión comerciales son binarios, ya que aún no se justifica económicamente procesar datos de niveles de gris o de color a las velocidades requeridas. Se espera que estas limitaciones vayan desapareciendo en el futuro.

2.2 ROBOT DE VISION RUDIMENTARIO PARA RECONOCIMIENTO DE FIGURAS GEOMETRICAS

2.2.1INTRODUCCIÓN.

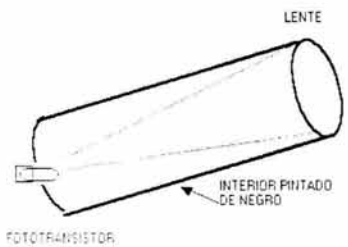
En la actualidad la habilidad de obtener imágenes para los sistemas de visión se ha incrementado y el procesamiento de estas ha favorecido la toma de decisiones, desde un diagnostico médico hasta operaciones militares, estos requerimientos son el origen del Reconocimiento Automático de Objetos (ATR) [1]. Para la maquina de visión prototipo que diseñamos nos basamos en el modelo biológico del procesamiento de la retina, el cual consiste básicamente en enfocar la imagen para que incida en un solo punto, y utilizamos un solo fotoreceptor alineado a una lente de aumento. Con este sencillo sistema nos ahorramos el gasto de una cámara de vídeo y un sofisticado sistema de digitalización de la imagen. Sin embargo, la desventaja de la máquina de visión propuesta es la manipulación de la frecuencia espacial, por lo que se requiere de un servomecanismo con movimiento finos bien controlados que barra toda la superficie del área que se desea digitalizar, lo cual lleva a un mayor tiempo de espera. Por otro lado, las Redes Neuronales Artificiales (RNA) han sido usadas en una gran variedad de aplicaciones de reconocimiento de patrones y se ha demostrado que los datos que reciben para su entrenamiento, no necesariamente deben ser perceptibles al ojo humano. Para este trabajo se utiliza el prototipo de la máquina de visión para hacer un barrido de varias

figuras geométricas en diferentes posiciones y entrenar una Red Neuronal de Retropropagación para hacer una clasificación de dichas figuras. La utilización de esta máquina de visión muy simple y el software desarrollado, se puede usar en el control de calidad de productos que requieran una forma geométrica uniforme.

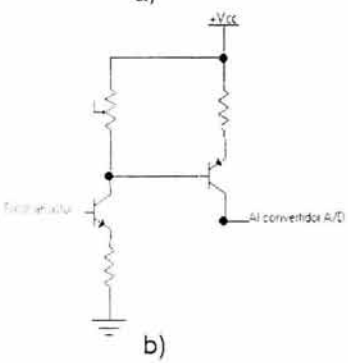
Este robot rudimentario de visión se utiliza para realizar practicas de robótica en la carrera de Ingeniería en Computación, donde los alumnos programan la velocidad de barrido, la resolución espacial, y diseñan algoritmos para seguimiento de contornos. El robot sirve también como instrumento para recolectar datos, y probar algoritmos de redes neuronales básicos como el perceptron, adaline, madaline y retropropagación, recogiendo datos de las figuras geométricas: con diferentes intensidades de luz, en diferentes posiciones y diferentes tamaños.

2.2.2. DISEÑO DE LA MAQUINA DE VISIÓN.

Es importante tener un entendimiento básico del proceso de percepción visual antes describir el procedimiento de captura de imágenes para este prototipo. El mecanismo visual es uno de los más perfectos, nos proporciona información acerca de los objetos por medio del órgano sensorial conocido como ojo. Los rayos atraviesan una parte transparente del globo ocular, penetran en el interior del mismo e inciden sobre el tejido sensible. Como resultado surgen impulsos que son transmitidos por fibras nerviosas del cerebro donde originan la sensación visual.



a)



b)

La luz nos llega en una serie de ondas, cuyas diferentes longitudes excitan distintas sensaciones de color cuando inciden en la retina. El ojo no presenta la misma sensibilidad a los rayos luminosos de todas las longitudes de onda. Algunos, como los rayos ultravioleta e infrarrojos, entre otros, no producirían ninguna sensación visual. Vemos la imágenes de los objetos porque las ondas luminosas se reflejan en ellos y entran en el ojo, que a menudo ha sido comparado con una cámara fotográfica. En el ojo la luz que se refleja atraviesa el cristalino y es enfocada sobre la retina sensible a la luz. El ajuste adecuado se produce gracias a la modificación de la forma del cristalino que es un cuerpo elástico, acción que se ejecuta por medio de los pequeñisimos músculos situados dentro del globo ocular.

2.2.2.1 Acondicionamiento de la señal luminosa.

El ojo del robot no es mas que una lente, un

tubo, una célula fotoeléctrica (fototransistor NT3032) y un cable de conexión. La lente debe ser de 10 dioptrías o un poco más potente, es decir, la distancia focal debe ser de 10 cm. o menos. Una simple lupa de plástico puede ser más que suficiente. El soporte de la lente es un tubo de cartón acoplado con cinta adhesiva. Una vez fijado el fototransistor en el punto focal se pueden hacer pruebas, midiendo el voltaje generado de acuerdo con la intensidad de luz registrada, directamente en un multímetro (figura 1a).

El sistema de visión tipo cíclope, denominado así por que solo tiene una lente, dota al robot de una señal de referencia situada en el foco de una lente: es como el bastón de un hombre ciego que va ejecutando pequeños golpes sobre la banqueta, y va percibiendo en cada golpe si hay un cambio de fondo o un obstáculo. Este único punto de visión con el que cuenta el cíclope es desplazado en todas direcciones por el propio robot, haciendo posible la construcción de una representación de los niveles de luz que inciden en él. Lo difícil es activar al robot para que explore adecuadamente, permitiendo que los niveles de gris se presenten en la pantalla de la computadora confeccionando una imagen convencional. Esta tarea se realiza primero ajustando la intensidad de luz, por medio del circuito de la figura 1b, que sirve para captar un bajo nivel de iluminación, de tal manera que a menor intensidad de luz mayor voltaje de salida.

Figura 1. Un ojo de bajo costo.

2.2.2.1.2 Anatomía del Robot.

El diseño anatómico del cíclope se debe al material de desecho con el cual se implemento. Al desarmar un drive de 5¼ de una computadora se acoplo un montaje mecánico como el que se muestra en la figura 2. Que dio como resultado una configuración polar, con una articulación rotacional en la base con giro a lo largo del eje perpendicular, y una articulación rotacional con una acción de torsión sobre el cuerpo del robot. Y como efector final el ojo del cíclope.

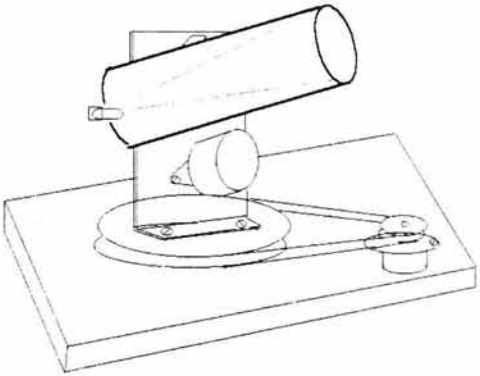


Figura 2. Montaje mecánico del cíclope.

2.2.2.1.3 Control del Robot.

El sistema impulsor para la regulación adecuada de los movimientos del cíclope, se implementa con un motor paso a paso, que se encuentra montado sobre el cuerpo del robot y un motor de corriente directa que mueve la polea del movimiento horizontal sobre la base del cíclope. Por desgracia los pasos son bastante grandes, y con tan solo 24 medios pasos se abarcan 90 grados totalmente, por lo que se agregan poleas para desmultiplicar el movimiento. Como la capacidad para dividir el margen de la articulación en incrementos depende de la capacidad de almacenamiento en bits de la memoria de control, seleccionamos un microcontrolador de 8 bits y por tanto tenemos un margen de movimiento total de 256 posiciones distintas. Las inexactitudes mecánicas en los componentes de las uniones y articulaciones del robot afectan la resolución espacial de la imagen que se pretende digitalizar. Las inexactitudes mecánicas proceden de la desviación elástica en los miembros estructurales, tensión de los cordones de las poleas y la carga que se manipula, así como la velocidad con que se programe el desplazamiento del robot

La resolución espacial del robot se controla a través de la tarjeta SIMMP-2 que tiene una arquitectura basada en el microcontrolador 68HC11F1 fabricado por Motorola®, esta tarjeta permite que el usuario tenga acceso a casi todas las líneas asociadas al microcontrolador ya sea con periféricos propios o con líneas de control del mismo. La idea de controlar los movimientos del robot con un microcontrolador y no directamente desde la PC, se debe a que en el futuro se pretende implementar algoritmos inteligentes directamente en el microcontrolador ya sean con Redes Neuronales o Fuzzy Logic. Por el momento, es indispensable la PC para ver la representación gráfica de la imagen, y la ejecución fuera de línea de los algoritmos para el reconocimiento de patrones. El diagrama a bloques del sistema de control se puede observar en la figura 3. La tarjeta SIMMP-2 se configura para trabajar en modo expandido,

grabando el programa de comunicación en la memoria RAM, el cual consiste básicamente en: configurar el puerto de comunicación serie (8 bits de datos, sin paridad, un bit de paro y velocidad de 9600 baudios); espera un dato de la PC y verificar que el carácter recibido sea diferente de 'F'(fin); mover el datos recibido a uno de los puertos de salida del micro; ejecutar una subrutina de espera en lo que se estabiliza la parte mecánica; comenzar la conversión A/D y almacenar el dato en memoria; enviar el dato convertido a la PC, y repetir la secuencia. En la figura 4a, se muestra el diagrama de flujo del programa contenido en el microcontrolador.

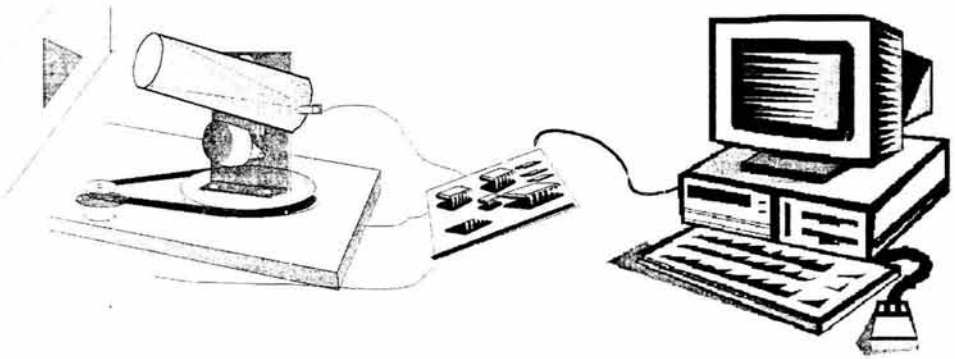


Figura 3. Sistema de control para manipular el robot cíclope.

CAPITULO III

ADQUISICIÓN DE IMÁGENES, ANÁLISIS Y RESULTADOS CON MATLAB

3.1 RECONOCIMIENTO DE FIGURAS GEOMETRICAS.

El ser humano percibe los objetos que lo rodean a través de un proceso de abstracción simbólica, para extraer sus propiedades básicas. Al identificar y reconocer los objetos de una escena, se dice que ésta ha sido analizada. La Inteligencia Artificial (IA), entendida muy ampliamente como el modelado y simulación de las actividades cognitivas complejas (percepción, memoria, solución de problemas, etc.) que caracterizan a los organismos avanzados, y en particular a los seres humanos, tiene dos ramas bien diferenciadas:

Por un lado se trata de modelar la actividad racional mediante sistemas formales de reglas y manipulación simbólica (generalmente mediante sistemas lógicos), que podríamos denominar simbólico – deductiva. Se postulan una serie de reglas y el sistema resuelve los problemas realizando deducciones sobre las reglas existentes.

Por otro lado se desarrollan modelos computacionales inspirados en las redes neuronales biológicas denominados inductivos o subsimbólicos, ya que extraen la información necesaria para resolver un problema de un conjunto de ejemplos, sin necesidad de indicarle las reglas necesarias para resolverlo.

3.1.1 Adquisición de la Imagen.

Una imagen puede obtenerse por medios diversos; el digital por ejemplo requiere de una adecuada utilización de convertidores analógico digital y dependiendo de este, se podrá determinar la calidad y definición de la imagen. Sin embargo, hay que considerar que el ojo humano tiene la capacidad de distinguir diferentes niveles de brillantes, el rango de niveles de intensidad de luz que el sistema visual humano puede captar es del orden de 10^{10}, desde el umbral de ausencia de luz hasta el límite del destello. La psicovisión es la relación de los colores que percibimos, los cuales son básicamente una combinación de longitud de onda o energía de la radiación electromagnética. La tecnología ha permitido extender la Psicovisión a nuevas regiones como si nuestros ojos se hubiesen hecho más poderosos. A través de las imágenes acústicas, magnéticas, de radar, etc.

La comunicación entre la computadora y el microcontrolador se realiza a través de la transmisión de datos por el puerto serie, esta interfase se programa en MATLAB utilizando el minitoolbox "CPORT", que se puede bajar del servidor web de Mathworks. El diagrama de flujo que describe el programa que manipula al robot y procesa la imagen, se muestra en la figura 4b. Lo que el programa realiza básicamente es: Asignar las constantes de retardo (velocidad del robot) y 8 caracteres a un arreglo, que son el equivalente en código binario (8 bits) para accionar la secuencia del motor de pasos; se asigna un handle (numero de proceso de windows) para configurar el puerto serial; y se realiza un ciclo, donde se escribe al puerto los dato del arreglo que hace que mueve verticalmente el ojo del ciclope, a la vez que lee el puerto para tomar el dato convertido equivalente a la intensidad de luz donde apunta el robot. Este ciclo se realiza 8

veces y al terminar envía una secuencia para mover el motor de corriente directa (CD). simulando un tren de pulsos, de tal manera que cada que llega el ángulo del ojo, al tope, se mueve el robot de manera horizontal (figura 5a).

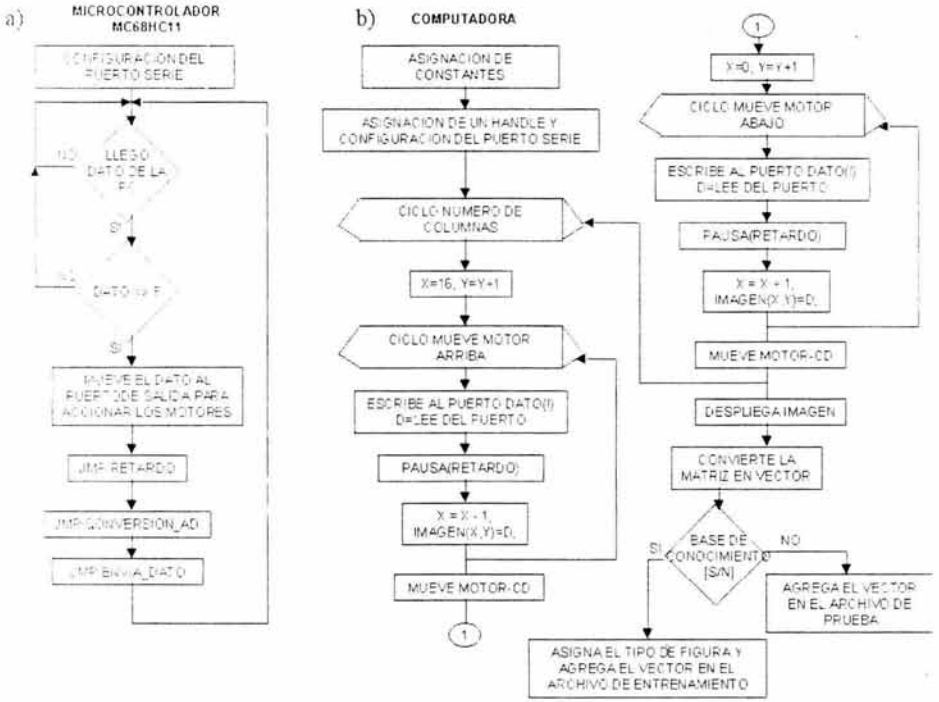


Figura 4. Diagrama de flujo, a) del programa en el microcontrolador, b) del programa en la PC.

Cuando termina de barrer la figura y la muestra en pantalla, en este momento se detiene el proceso y le pregunta al usuario si desea incluir la imagen en la base de conocimiento. En caso de contestar que si, entonces pregunta que tipo de figura es: c(u)adrado,(t)riangulo, (r)ectangulo, (c)irculo]?. Esto se hace con el fin de adquirir datos para entrenar la RNA, la figura 5b muestra la forma en que se descompone la matriz que forma la imagen, en un solo vector que se introduce en el archivo de entrenamiento "figuras.nna". En caso contrario quiere decir que el usuario quiere reconocer la figura, para lo cual, el programa escribe en forma de vector los datos de la figura en el archivo "test.nna". En este momento el usuario tiene que ejecutar desde windows al software de NeuralWorks, para hacer el reconocimiento con la red neuronal previamente diseñada y entrenada. En cuanto termina el proceso de reconocimiento los resultados se almacenan el archivo "test.nnr", y el usuario debe regresar a la ventana de MATLAB, para

continuar con el programa. El resto del programa simplemente lo que hace es leer el archivo de resultados, y compara que valor es el mas cercano de acuerdo con la tabla de asignación de figuras (ver figura 5b).

3.2 Diseño de la Red Neuronal de Retropropagación.

Son muchas las aplicaciones que resultan difíciles de realizar porque hay muchos problemas cuya solución no es adecuada mediante procesos secuenciales. Por ejemplo, las aplicaciones que deben realizar complejas traducciones de datos que no poseen una función de correspondencia predefinida que describa el proceso de traducción o aquellas que deben proporcionar una mejor aproximación como salida cuando se les presentan unos datos de entrada ruidosos. Las Redes Neuronales se emplean debido a su capacidad para adaptarse o aprender, generalizar u organizar datos cuyas operaciones están basadas en procesos paralelos.

Un algoritmo que ha resultado útil para la solución de estos y muchos otros problemas que requieren el reconocimiento de tramas complejas y la realización de funciones de correspondencia no triviales es el algoritmo de retropropagación.

Una red trabajando bajo el algoritmo de Retropropagación del error, está diseñada para que funcione como una red multicapa con propagación hacia adelante, empleando un aprendizaje supervisado.

Una red de Retropropagación es entrenada con una serie de entradas y sus correspondientes salidas, motivo por el cual, se dijo que el algoritmo de retropropagación del error trabaja bajo un modo de aprendizaje supervisado.

Una red de Retropropagación funciona de la manera siguiente: Durante la sesión de entrenamiento, los pesos de la red convergen en un punto en el espacio de pesos de la red, en el que el problema es conocido por la red; cuando la red ha alcanzado este punto, puede dar la salida correcta para cada entrada dada. Sin embargo, debido a la naturaleza de la red, no siempre se puede aprender el problema exactamente. La salida producida por la red cuando se presenta una entrada, puede ser una aproximación de la salida correcta, motivo por el cual es difícil decir cuando la red ha aprendido el problema en cuestión.

Las redes de retropropagación son en realidad redes de propagación hacia adelante, cuyo algoritmo de aprendizaje es el algoritmo de Retropropagación del error.

En general, una red neuronal consiste en una serie de elementos procesadores con conexiones directas entre ellos. Es posible asociar un peso a cada una de estas conexiones. Para una red de Retropropagación, estos pesos pueden ser actualizados durante el entrenamiento con la ayuda de una regla de aprendizaje.

Desde el punto de vista de las aplicaciones prácticas de las Redes neuronales Artificiales (RNA), su principal ventaja frente a otras técnicas reside en el

proceso paralelo, adaptativo y no lineal. Se han desarrollado aplicaciones de RNA para fines tan variados como Visión Artificial, Procesamiento de Señales e Imágenes, reconocimiento del habla y de caracteres, sistemas expertos, Análisis de Imágenes Medicas, Control de Robots, etc. La mayoría de las aplicaciones comentadas se han desarrollado mediante el algoritmo de retropropagación. Al hablar de redes de retropropagación hacemos referencia a un algoritmo de aprendizaje más que ha una arquitectura determinada.

Por las razones antes citadas se opto por la utilización de una red neuronal de Retropropagación utilizando una función de activación sigmoidal, sobre las demás arquitecturas existentes, de las cuales hacemos mención en el anexo correspondiente.

Para este primer trabajo el diseño de la Red Neuronal se realiza con el software NeuralWorks.

El diseño de la red neuronal se llevo a cabo tomando en cuenta el numero de entradas a la RNA, contenidas en el archivo "figuras.nna", donde cada vector representa una imagen que contiene 32 X 32 píxeles, lo que da un total de 32 elementos de entrada y cuatro de salida, con la siguiente secuencia: 0001 – representa un cuadrado, 0010 – representa un triangulo, 0100 – representa un rectángulo y 1000 – representa un circulo. (Fig.-5)

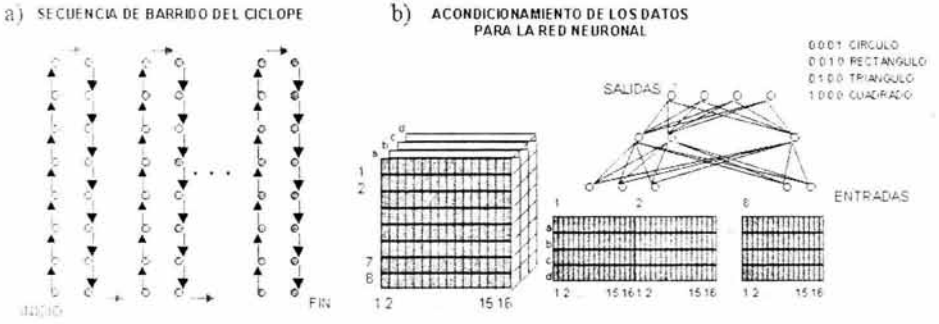


Figura 5. Construcción de la imagen.

Estos valores fueron obtenidos después del proceso de conversión del archivo de la imagen a un vector con el software MATLAB , mediante una programa el cual es el encargado de mostrar el valor binario de cada uno de los píxeles que conforman la imagen obtenida con el robot diseñado para este fin.

3.3 Programación

El programa fuente en matlab, el cual nos permite transformar la imagen obtenida mediante el robot, en una matriz de 32 x 32 y posteriormente en un vector de 32 elementos que serán nuestros valores de entrada se muestra a continuación:

```
respc('Quieres guardar la figura?(s/n): ','s');
if strcmp(resp,'s')
  tipo= input('Quieres guardar la figura en la Base de conocimiento o en test?(b/t): ','s');
  if strcmp(tipo,'t')
    disp('Archivo test: test.nna');

    A=imread('figura.gif');
    a=sum(A);
    fid=fopen('test.nna','a+');
  else
    if strcmp(tipo,'b')
    disp('Base de Conocimiento: base.nna ');

    A=imread('figura.gif');
    a=sum(A);
    fid=fopen('base.nna','a+');

    end
  end
  disp('Tipo de Figura:')
  disp('(t)riangulo')
  disp('(c)irculo')
  disp('c(u)adrado')
```

```
disp('(r)ectangulo')
fig=input('==> ','s');
switch fig

case 't', disp('selecciono un triangulo')

   for i= 1:length(a)
      fprintf(fid,'%f ',a(i));
   end
   fprintf(fid,'\t 0100\r\n');
   fclose(fid);
case 'c', disp('selecciono un circulo')

   for i= 1:length(a)
      fprintf(fid,'%f ',a(i));
   end
   fprintf(fid,'\t 0001\r\n');
   fclose(fid);
case 'u', disp('selecciono cuadrado')

   for i= 1:length(a)
      fprintf(fid,'%f ',a(i));
   end
   fprintf(fid,'\t 1000\r\n');
   fclose(fid);
case 'r', disp('selecciono un rectangulo')
```

```
    for i= 1:length(a)
        fprintf(fid,'%f ',a(i));
    end
    fprintf(fid,'\t 0010\r\n');
    fclose(fid);
  otherwise, disp('Seleccion no valida')
    fclose(fid);
  end
else
disp('fin de programa');
end
```

3.3.1 funciones Utilizadas

Para ello se utilizaron ciertas funciones de Matlab que a continuación describiremos:

Input

La función *input* permite imprimir un mensaje en la línea de comandos de MATLAB y recuperar como valor de retorno un valor numérico o el resultado de una expresión tecleada por el usuario.

Después de imprimir el mensaje, el programa espera que el usuario teclee el valor numérico o la expresión. Cualquier expresión válida de MATLAB es aceptada por este comando. El usuario puede teclear simplemente un vector o una matriz. En cualquier caso, la expresión introducida es evaluada con los valores actuales de las variables de MATLAB y el resultado se devuelve como valor de retorno. }

Un ejemplo de uso de esta función:

```
» n = input('Teclee el número de ecuaciones')
```

strcmp(c1,c2)

comparación de cadenas. Si las cadenas son iguales devuelve un uno, y si no lo son, devuelve un cero (funciona de modo diferente que la correspondiente función de C)

disp(c)

imprime el texto contenido en la variable c

imread('')

permite leer un archivo en formato gif, jpg, BMP, png, etc. Y tranformarlo en una matriz.

sum(A):

Realiza una sumatoria de las Columnas de una Matriz tranformandola en un vector.

FUNCIONES *FOPEN* Y *FCLOSE*

Estas funciones sirven para abrir y cerrar ficheros, respectivamente. La función *fopen* tiene la forma siguiente:

[fi,texto] = fopen('filename','c')

donde fi es un valor de retorno que sirve como identificador del fichero, texto es un mensaje para caso de que se produzca un error, y c es un carácter (o dos) que indica el tipo de operación que se desea realizar. Las opciones más importantes son las siguientes: 'r' lectura (de *read*) 'w' escritura reemplazando (de *write*) 'a' escritura a continuación (de *append*)

'r+' lectura y escritura

Cuando por alguna razón el fichero no puede ser abierto, se devuelve un (-1). En este caso el valor de retorno texto puede proporcionar información sobre el tipo de error que se ha producido (también existe una función llamada *ferror* que ermite obtener información sobre los errores. En el *Help* del programa se puede ver cómo utilizar esta función).

Después de realizar las operaciones de lectura y escritura deseadas, el fichero se puede cerrar con la función *close* en la forma siguiente:

st = fclose(fi)

Donde st es un valor de retorno para posibles condiciones de error. Si se quieren cerrar a la vez todos los ficheros abiertos puede utilizarse el comando: st = close('all')

switch

La sentencia *switch* realiza una función análoga a un conjunto de *if...elseif* concatenados.

Su forma general es la siguiente:

switch switch_expresión case case_expr1, bloque1 case {case_expr2, case_expr3, case_expr4,...}

bloque2

..

.

otherwise, % opción por defecto

bloque3

end

Al principio se evalúa la *switch_expresion*, cuyo resultado debe ser un número escalar o una cadena de caracteres. Este resultado se compara con las *ase_expr*, y se ejecuta el bloque de sentencias que corresponda con ese resultado. Si ninguno es igual a *switch_expresion* se ejecutan las sentencias correspondientes a *otherwise*. Según puede verse en el ejemplo anterior, es posible agrupar varias condiciones dentro de unas llaves (constituyendo lo que se llama un *cell array* o vector de celdas, explicado en el Apartado 4.4); basta la igualdad con cualquier elemento del cell array para que se ejecute ese bloque de sentencias. La "igualdad" debe entenderse en el sentido del operador de igualdad (==) para escalares y la función *strcmp()* para cadenas de caracteres). A diferencia de C/C++/Java13, en MATLAB sólo se ejecuta uno de los bloques relacionado con un

case.

Fprintf

Finalmente, la función *fprintf* dirige su salida formateada hacia el fichero indicado por el identificador. Su forma general es:

fprintf(fi,'cadena de control',var1,var2,...)

Esta es la función más parecida a su homóloga de C. La cadena de control contiene los formatos de escritura, que son similares a los de C, como muestran los ejemplos siguientes:

fprintf(fi,'El número de ecuaciones es: %d\n',n)

fprintf(fi,'El determinante es: %lf10.4\n',n)

De forma análoga, la función *sprintf* convierte su resultado en una cadena de caracteres que devuelve como valor de retorno, en vez de enviarlo a un fichero.

resultado = sprintf('El cuadrado de %f es %12.4f\n',n,n*n) donde resultado es una cadena de caracteres. Esta función constituye el método más general de convertir números en cadenas de caracteres, por ejemplo para ponerlos como titulos de figuras.

3.4 Procesamiento de Imagen

Se realizo el barrido de 4 figuras geométricas (circulo, triángulo, cuadrado y rectángulo) a la misma distancia. Cada barrido se realizo 4 veces, girando la posición de la figura, logrando así, 16 registros. La tarjeta SIMMP-2 contiene varios puertos de entrada/salida digitales por los cuales se controlan los motores de paso del servomecanismo como ya se explico en el apartado 2.3 y la señal del fototransistor es acoplada al circuito de la figura 1b, que entrega un voltaje de 5 a 3 volts; esta señal es digitalizada a través del convertidor A/D interno del microcontrolador. Por cada barrido en la figura geométrica se forma una matriz de 8 x 16 que es almacenada en la memoria RAM de la tarjeta SIMMP-2 y posteriormente se transmite a través del puerto serial a la PC, mostrando una imagen muy pobre como la que se muestra en la figura 7.

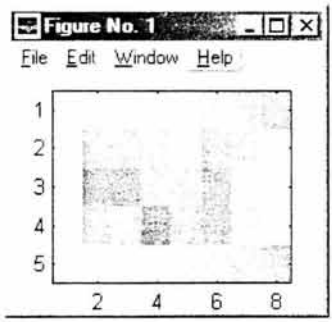


Figura 7. Imagen de baja resolución

de un rectángulo generada por el ciclope.

La Red Neuronal de Retropropagación se entreno en base a los archivos generados por el barrido de las figuras geométricas, ocupando el 75 % de los datos seleccionados al azar y el restante 25 % se utilizo para probar la Red Neuronal. Para generar esta Red se utilizo el software NeuralWorks, convirtiendo la matriz a un vector, el cual sirve de entrada con 32 elementos, una capa oculta con 50 elementos y 1 elemento de proceso como salida, uno por cada figura geométrica y utilizando una función de activación sigmoidal.

Los resultados arrojados por la Red Neuronal, para clasificar el tipo de figura geométrica que el prototipo de visión capturo se muestran en la tabla 1. Estos resultados fueron procesados en Excel haciendo una comparación lógica (falso, verdadero) entre las entradas del archivo de prueba y los resultados arrojados por NeuralWorks, considerando una tolerancia de acierto de ±0.3, tomando en cuenta que se activa en 1 la clasificación de la figura geométrica correspondiente, esto quiere decir un valor $0.7 < x < 1.3$, se considera como verdadero.

Tabla 1. Resultados obtenidos en el reconocimiento de figuras geométricas.

Tabla 1

Tipo de figura	Circulo	Triángulo	Cuadrado	Rectángulo
Porcentaje de Clasificación	72 %	84 %	92 %	94 %

3.5 Análisis de Resultados

Los resultados obtenidos hasta el momento con este robot son muy alentadores, por que con pocos recursos se construye un sistema de visión muy sencillo, que apoya la investigación aplicando conceptos básicos de óptica, procesamiento digital de señales, control y reconocimiento de patrones. Esto nos puede arrojar la necesidad de trabajar en el perfeccionamiento de la red neuronal de retropropagación para así poder dar el siguiente paso hacia el almacenamiento de la misma directamente en el microcontrolador y poder así comenzar a trabajar con una red neuronal implementada en hardware que pueda entregar mejores resultados y sea mas atractiva para la industria y así fomentar el interés sobre este campo de investigación.

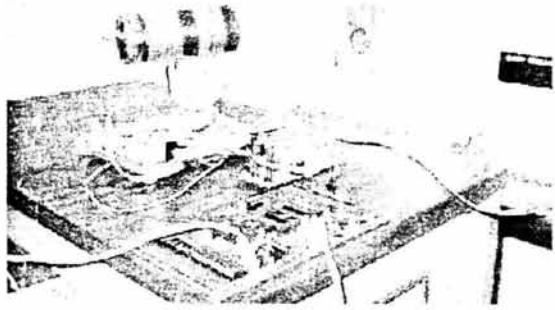


Figura 6. Robot para reconocimiento de figuras geométricas

3.6 Conclusiones:

Tomando en cuenta los Resultados obtenidos después del proceso de la digitalización de la imagen, transformación de la misma en un vector de 32 posiciones, el procesamiento dentro de la base de conocimiento de la red neuronal, y su posterior análisis, podemos darnos cuenta de que el prototipo funciona correctamente, ya que entrega resultados alentadores, además de que observamos que conforme mas se incremente el proceso de aprendizaje, la red neuronal podrá mostrarnos resultados cada vez mas confiables, ya que se estará incrementando su base de conocimientos, y con ello reforzamos el aprendizaje obtenido.

En general diremos que el proyecto fue exitoso basado en las características de costo/beneficio con las cuales se trabajaron durante el transcurso del mismo.

CONCLUSIONES FINALES

Una vez realizado el trabajo de tesis, podemos concluir lo siguiente:

- Se logro establece las bases para la generación de una red neuronal de retorpropagación, de acuerdo a las necesidades del proyecto.
- Se construyo un prototipo robótico, mediante el cual capturamos una serie de imágenes, que nos sirvieran como base de conocimientos para nuestra red neuronal.
- Se implemento la tarjeta SIMMP-2 que utiliza el microprocesador HC-11, como interfaz entre el robot y la PC, con el fin de generar el archivo con la imagen capturada por el robot.
- Se desarrollo un programa en MatLab, el cual realizo la función de transformar el archivo que contenía la imagen en un vector, el cual se iba almacenando ya sea en la base de conocimiento o en la base de prueba con el formato requerido por el software de redes neuronales (NeuralWork)
- Se configuro la red neuronal de retropropagación en el software especializado Neuralwork, para realizar el proceso de reconocimiento de las imágenes procesadas.
- Se genero la tabla de análisis de resultados del experimento, en base a los datos obtenidos.

Después de verificar los resultados obtenido dentro del experimento, observamos que, a medida de que aumenta nuestra base de conocimientos, es decir, aumentamos el numero de muestras realizadas por el prototipo; la eficiencia del experimento aumentaba, sin embargo, recomendamos experimentar con otras topologías de redes neuronales a las cuales se hace referencia en los anexos de este trabajo.

A demás, una vez concluido el trabajo, observamos la necesidad de realizar pruebas con diferentes arquitecturas de redes neuronales, que pudieran entrgar mejores resultados, así como la modificación del diseño del prototipo, esto debido a que observamos que un robot de tipo cartesiano ofrece una ventaja sobre el de tipo polar, esto en el momento de la capturar de la imagen.

ANEXO A
TIPOS DE REDES NEURONALES

PERCEPTRÓN

Antecedentes: La primera red neuronal conocida, fue desarrollada en 1943 por Warren McCulloch y Walter Pitts; esta consistía en una suma de las señales de entrada, multiplicadas por unos valores de pesos escogidos aleatoriamente. La entrada es comparada con un patrón preestablecido para determinar la salida de la red. Si en la comparación, la suma de las entradas multiplicadas por los pesos es mayor o igual que el patrón preestablecido la salida de la red es uno (1), en caso contrario la salida es cero (0). Al inicio del desarrollo de los sistemas de inteligencia artificial, se encontró gran similitud entre su comportamiento y el de los sistemas biológicos y en principio se creyó que este modelo podía computar cualquier función aritmética o lógica.

La red tipo Perceptrón fue inventada por el sicólogo 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 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, y esto lo llevó a una teoría de *separabilidad estadística* que utilizaba para caracterizar las propiedades más visibles de estas redes de interconexión ligeramente aleatorias.

El primer modelo de Perceptrón fue desarrollado en un ambiente biológico imitando el funcionamiento del ojo humano, el fotoperceptrón como se le llamo era un dispositivo que respondía a señales ópticas; como se muestra en el figura 2.1.1 la luz incide en los puntos sensibles (**S**) de la estructura de la retina, cada punto S responde en forma todo-nada a la luz entrante, los impulsos generados por los puntos S se transmiten a las unidades de asociación (**A**) de la capa de asociación; cada unidad A está conectada a un conjunto aleatorio de puntos S, denominados conjunto fuente de la unidad A, y las conexiones pueden ser tanto excitatorias como inhibitorias. Las conexiones tienen los valores posibles +1, -1 y 0, cuando aparece un conjunto de estímulos en la retina, una unidad A se activa si la suma de sus entradas sobrepasa algún valor umbral; si la unidad esta activada, A produce una salida que se envía a la siguiente capa de unidades.

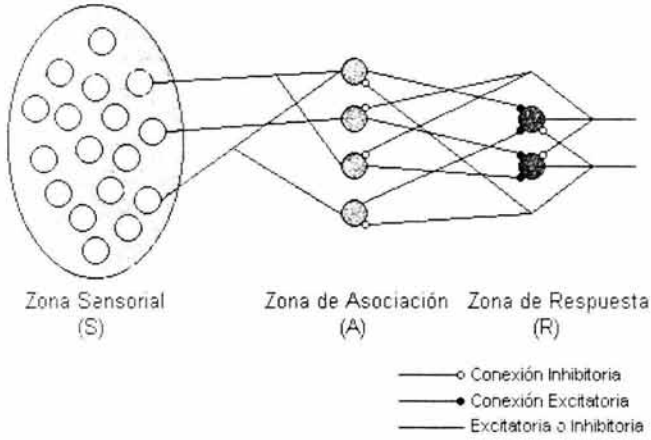


Figura 2 1.1 Modelo del Fotoperceptrón de Rosenblatt

De forma similar, las unidades A están conectadas a unidades de respuesta (**R**) dentro de la capa de respuesta y la conectividad vuelve a ser aleatorio entre capas, pero se añaden conexiones inhibitorias de realimentación procedentes de la capa de respuesta y que llegan a la capa de asociación, también hay conexiones inhibitorias entre las unidades R. Todo el esquema de conexiones se describe en forma general en un diagrama de Venn, para un Perceptrón sencillo con dos unidades de respuesta como el de la figura 2.1.2.

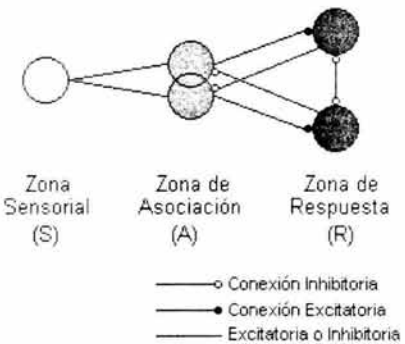


Figura 2.1.2 Esquema de conexiones de un Perceptrón sencillo

El Perceptrón era inicialmente un dispositivo de aprendizaje, en su configuración inicial no estaba en capacidad de distinguir patrones de entrada muy complejos, sin embargo mediante un proceso de aprendizaje era capaz de adquirir esta capacidad. En esencia, el entrenamiento 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 Perceptrón para una entrada dada. Se aplicaba una entrada a la retina, y 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. Si se activaba una unidad **R** incorrecta, se hacía disminuir la salida de las unidades **A** que hubiesen contribuido.

Mediante estas investigaciones se pudo demostrar que el Perceptrón era capaz de clasificar patrones correctamente, en lo que Rosenblatt denominaba un entorno diferenciado, en el cual cada clase estaba formada por patrones similares. El Perceptrón también era capaz de responder de manera congruente frente a patrones aleatorios, pero su precisión iba disminuyendo a medida que aumentaba el número de patrones que intentaba aprender.

En 1969 Marvin Minsky y Seymour Papert publicaron su libro: "Perceptrons: An introduction to Computational Geometry"[], el cual para muchos significó el final de las redes neuronales. En el se presentaba un análisis detallado del Perceptrón, en términos de sus capacidades y limitaciones, en especial en cuanto a las restricciones que existen para los problemas que una red tipo Perceptrón puede resolver; la mayor desventaja de este tipo de redes es su incapacidad para solucionar problemas que no sean linealmente separables.

Minsky y Papert se apartaban de la aproximación probabilistica de Rosenblatt y volvían a las ideas de cálculo de predicados en el análisis del Perceptrón. Su idea de Perceptrón aparece en la figura 2.1.3

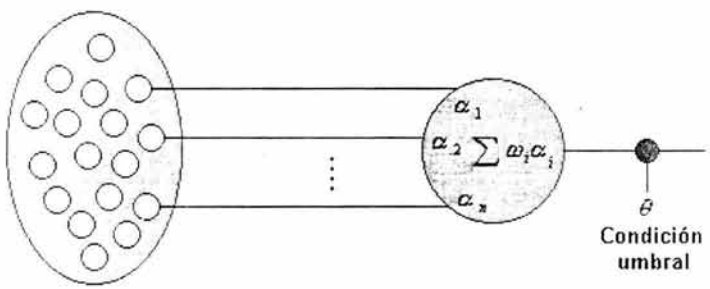


Figura 2.1.3 Perceptrón según Minsky y Papert

La estructura de un Perceptrón sencillo es similar a la del elemento general de procesamiento que se muestra en la figura 2.1.3; en la que se observa la adición de una condición umbral en la salida. Si la entrada neta, a esta condición es mayor que el valor umbral, la salida de la red es 1, en caso contrario es 0.

La función de salida de la red en la figura 2.1.3 es llamada función umbral o función de transferencia

$$f(salida) = \begin{cases} 1 \; si \; salida \geq \theta \\ 0 \; si \; salida < \theta \end{cases} \quad (2.1.1)$$

A pesar de esta limitación, el Perceptrón es aún hoy una red de gran importancia, pues con base en su estructura se han desarrollado otros modelos de red neuronal como la red Adaline y las redes multicapa.

Estructura de la red:

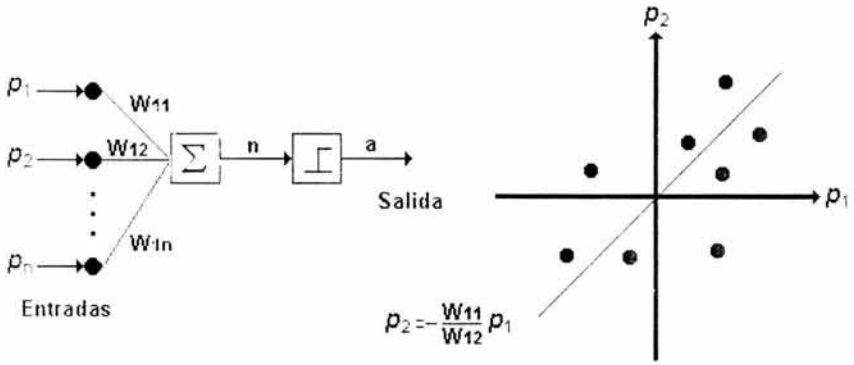


Fig. 2.1.4 Perceptrón

La única neurona de salida del Perceptrón realiza la suma ponderada de las entradas, resta el umbral y pasa el resultado a una función de transferencia de tipo escalón. La regla e decisión es responder +1 si el patrón presentado pertenece a la clase A, o –1 si el patrón pertenece a la clase B (figura 2.1.4), la salida depende de la entrada neta (n = suma de las entradas p_i ponderadas).

La red tipo Perceptrón emplea principalmente dos funciones de transferencia, *hardlim* con salidas 1, 0 o *hardlims* con salidas 1, -1; su uso depende del valor de salida que se espera para la red, es decir si la salida de la red es unipolar o bipolar; sin embargo la función *hardlims* es preferida sobre la *hardlim*, ya que el tener un cero multiplicando algunas de los valores resultantes del producto de las entradas por el vector de pesos, ocasiona que estos no se actualicen y que el aprendizaje sea más lento.

Una técnica utilizada para analizar el comportamiento de redes como el Perceptrón es presentar en un mapa las regiones de decisión creadas en el espacio multidimensional de entradas de la red, en estas regiones se visualiza qué patrones pertenecen a una clase y cuáles a otra, el Perceptrón separa las regiones por un hiperplano cuya ecuación queda determinada por los pesos de las conexiones y el valor umbral de la función de activación de la neurona, en este caso los valores de los pesos pueden fijarse o adaptarse empleando diferentes algoritmos de entrenamiento.

Para ilustrar el proceso computacional del Perceptrón consideremos la matriz de pesos en forma general.

$$W = \begin{bmatrix} W_{1,1} & W_{1,2} & \cdots & W_{1,R} \\ W_{2,1} & W_{2,2} & \cdots & W_{2,R} \\ W_{S,1} & W_{S,2} & \cdots & W_{S,R} \end{bmatrix} \quad (2.1.2)$$

Los pesos para una neurona están representados por un vector compuesto de los elementos de la *i*-ésima fila de ***W***

$$w = \begin{bmatrix} w_{i,1} \\ w_{i,2} \\ \vdots \\ w_{i,R} \end{bmatrix} \quad (2.1.3)$$

De esta forma y empleando la función de transferencia *hardlim* la salida de la neurona *i* de la capa de salida

$$a_i = hardlim\left(n_i\right) = hardlim\left(w_i^T p_i\right) \quad (2.1.4)$$

El Perceptrón, al constar de una sola capa de entrada y otra de salida con una única neurona, tiene una capacidad de representación bastante limitada, este modelo sólo es capaz de discriminar patrones muy sencillos, patrones linealmente separables (concepto que se estudiará en la sección 2.1.4), el caso más conocido es la imposibilidad del Perceptrón de representar la función OR EXCLUSIVA.

Regla de aprendizaje: El Perceptrón es un tipo de red de aprendizaje supervisado, es decir necesita conocer los valores esperados para cada una de las entradas presentadas; su comportamiento está definido por pares de esta forma:

$$\{p_1, t_1\}, \{p_2, t_2\}, \ldots, \{p_Q, t_Q\} \quad (2.1.5)$$

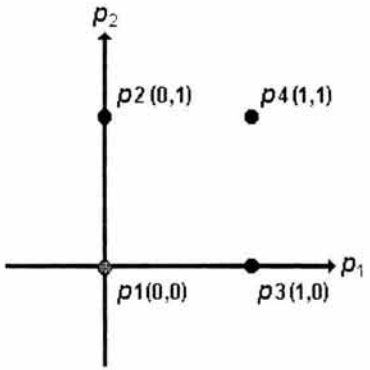
Cuando *p* es aplicado a la red, la salida de la red es comparada con el valor esperado *t*, y la salida de la red esta determinada por:

$$a = f\left(\sum_i w_i p_i\right) = hardlims\left(\sum_i w_i p_i\right) \quad (2.1.6)$$

Los valores de los pesos determinan el funcionamiento de la red, estos valores se pueden fijar o adoptar utilizando diferentes algoritmos de entrenamiento de la red.

Como ejemplo de funcionamiento de una red neuronal tipo Perceptrón, se solucionará el problema de la función OR, para esta función la red debe ser

capaz de devolver a partir de los cuatro patrones de entrada, a qué clase pertenece cada uno; es decir para el patrón 00 debe devolver la clase cero y para los restantes la clase 1, según la gráfica 2.1.5



p_1	p_2	a
0	0	0
0	1	1
1	0	1
1	1	1

Figura 2.1.5 Función OR

Para este caso las entradas a la red serán valores binarios, la salida de la red esta determinada por

$$a = hardlims\left(\sum_i w_i p_i\right) = hardlims(w_1 p_1 + w_2 p_2) \tag{2.1.7}$$

Si $w_1p_1+w_2p_2$ es mayor que 0 la salida será 1, en caso contrario la salida será –1 (función escalón unitario). Como puede verse la sumatoria que se le pasa a cada parámetro (entrada total) a la función *hardlim* (función de salida o de transferencia) es la expresión matemática de una recta, donde w_1 y w_2 son variables y p_1 y p_2 son constantes. En la etapa de aprendizaje se irán variando los valores de los pesos obteniendo distintas rectas, lo que se pretende al modificar los pesos de las conexiones es encontrar una recta que divida el plano en dos espacios de las dos clases de valores de entrada, concretamente para la función OR se deben separar los valores 01, 10, y 11 del valor 00; la red Perceptrón que realiza esta tarea y la gráfica característica pueden observarse en la figura 2.1.6 allí puede verse como las posibles rectas pasarán por el origen de coordenadas, por lo que la entrada 00 quedará sobre la propia recta.

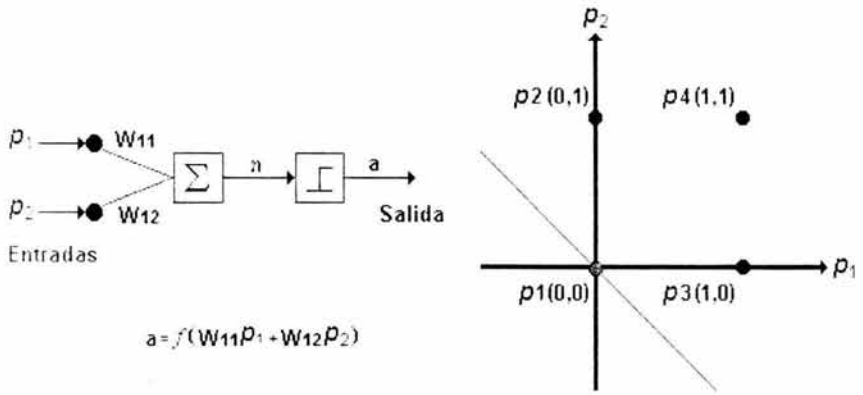


Figura 2.1.6 Perceptrón aplicado a la función OR

Se aplicará este método para resolver también el problema de la función AND, el cual se describe en la siguiente figura

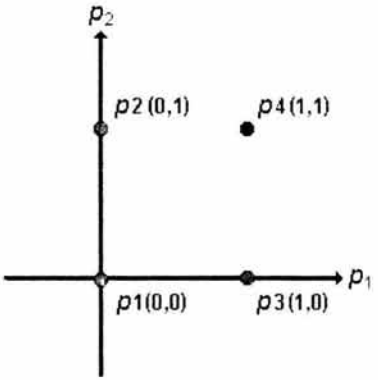


Figura 2.1.7 Espacio de salida de una compuerta AND

Analizando el comportamiento de la AND se llega a la conclusión de que es imposible que una recta que pase por el origen, separe los valores 00,01 y 10 del valor 11, por lo que se hace necesario introducir un término independiente para realizar esta tarea, a este término se le da el nombre de ganancia y se representa por la letra *b*, al cual por lo general se le asigna un valor inicial de 1 y se ajusta durante la etapa de aprendizaje de la red; este nuevo término permite desplazar la recta del origen de coordenadas dando una solución para el caso de la función AND y ampliando el número de soluciones de la función OR

Ahora la salida de la neurona esta dada por

$$a = hardlims(w_1 p_1 + w_2 p_2 + b) \quad (2.1.8)$$

Las soluciones obtenidas para las funciones AND y OR se observan en la figura 2.1.8

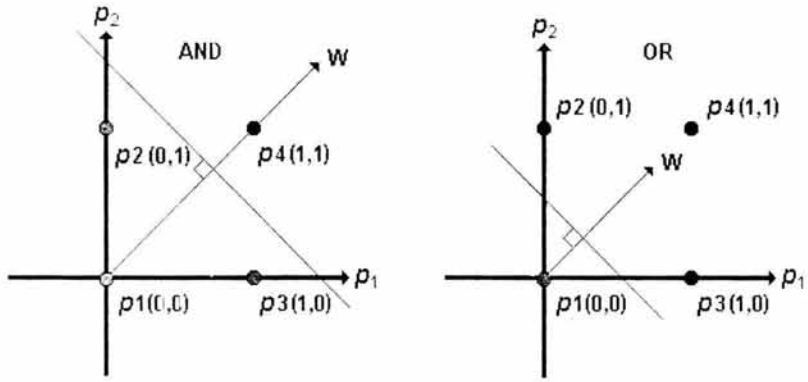


Figura 2.1.8 Solución para una función AND y una OR

En el proceso de entrenamiento el Perceptrón se expone a un conjunto de patrones de entrada y los pesos de la red son ajustados de forma que al final de entrenamiento se obtengan las salidas esperadas para cada unos de esos patrones de entrada.

El algoritmo de entrenamiento del Perceptrón puede resumirse en los siguientes pasos:

1. Se inicializa la matriz de pesos y el valor de la ganancia, por lo general se asignan valores aleatorios a cada uno de los pesos w_i y al valor b

2. Se presenta el primer patrón a la red, junto con la salida esperada en forma de pares entrada/salida

3. Se calcula la salida de la red por medio de

$$a = f\left(w_1 p_1 + w_2 p_2 + b\right) \quad (2.1.9)$$

donde f puede ser la función *hardlim* o *hardlims*

4. Cuando la red no retorna la salida correcta, es necesario alterar el valor de los pesos, tratando de llevarlo hasta p y así aumentar las posibilidades de que la clasificación sea correcta, una posibilidad es adicionar p a w haciendo que el vector w apunte en la dirección de p, y de esta forma después de repetidas presentaciones de p a la red, w se aproximará asintoticamente a p; este es el procedimiento adoptado para la regla de aprendizaje del Perceptrón.

El proceso de aprendizaje del Perceptrón puede definirse en tres reglas, las cuales cubren la totalidad de combinaciones de salidas y sus correspondientes valores esperados. Estas reglas utilizando la función de transferencia *hardlim*, se expresan como sigue:

$$Si\ \ t=1\ \ y\ \ a=0,\ entonces\ \ {}_1w^{nuevo} = {}_1w^{anterior} + p \quad (2.1.10)$$

$$Si\ \ t=0\ \ y\ \ a=1,\ entonces\ \ {}_1w^{nuevo} = {}_1w^{anterior} - p \quad (2.1.11)$$

$$Si\ \ t=a,\ entonces\ \ {}_1w^{nuevo} = {}_1w^{anterior} \quad (2.1.12)$$

Las tres condiciones anteriores pueden ser escritas en forma compacta y generalizarse para la utilización de las funciones de transferencia *hardlim* o *hardlims*, generalización que es posible introduciendo el error en las reglas de aprendizaje del Perceptrón:

$$e = t - a \quad (2.1.13)$$

Por lo tanto:

$$Si\ \ e=1,\ entonces\ \ {}_1w^{nuevo} = {}_1w^{viejo} + p \quad (2.1.14)$$

$$Si\ \ e=-1,\ entonces\ \ {}_1w^{nuevo} = {}_1w^{anterior} - p \quad (2.1.15)$$

$$Si\ \ e=0,\ entonces\ \ {}_1w^{nuevo} = {}_1w^{anterior} \quad (2.1.16)$$

En una sola expresión la ley puede resumirse así:

$${}_1w^{nuevo} = {}_1w^{anterior} + ep = {}_1w^{anterior} + (t-a)p \quad (2.1.17)$$

Y extendiendo la ley a las ganancias

$$b^{nueva} = b^{anterior} + e \quad (2.1.18)$$

Para ilustrar la regla de aprendizaje del Perceptrón, se dará solución al problema de clasificación de patrones ilustrado en la figura 2.1.9

$$P_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \quad t_1 = 1 \quad P_2 = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \quad t_2 = 1 \quad P_3 = \begin{bmatrix} -2 \\ 1 \end{bmatrix} \quad t_3 = -1 \quad P_4 = \begin{bmatrix} 0 \\ 2 \end{bmatrix} \quad t_4 = -1$$

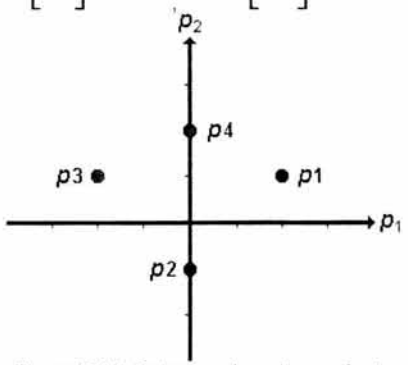


Figura 2.1.9 Patrones de entrenamiento

En este caso las salidas toman valores bipolares de 1 o –1, por lo tanto la función de transferencia a utilizar será *hardlims*. Según la dimensiones de los patrones de entrenamiento la red debe contener dos entradas y una salida.

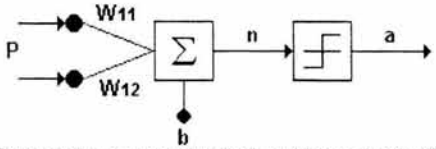


Figura 2.1.10 Red Perceptrón que resolverá el problema de clasificación de patrones

Para decidir si una red tipo Perceptrón puede aplicarse al problema de interés, se debe comprobar si el problema es linealmente separable, esto puede determinarse gráficamente de la figura 2.1.9, en donde se observa que existe un gran número de líneas rectas que pueden separar los patrones de una categoría de los patrones de la otra, el siguiente paso es asumir arbitrariamente los valores para los pesos y ganancias iniciales de entrada a la red; el proceso terminará cuando se hayan obtenido los pesos y ganancias finales que permitan a la red clasificar correctamente todos los patrones presentados.

Los valores iniciales asignados aleatoriamente a los parámetros de la red son:

$$W = [-0.7 \quad 0.2] \; b = [0.5]$$

Con base en el procedimiento descrito anteriormente, el proceso de aprendizaje de la red es el siguiente:

- Iteración 0

La red clasificara los patrones de entrenamiento según la característica de decisión mostrada en la figura 2.1.11, la cual depende de los valores de los pesos y ganancias iniciales.

Interceptos con los ejes: $\frac{-b}{W_{11}} = -2.5 \; \frac{-b}{W_{21}} = 0.71$

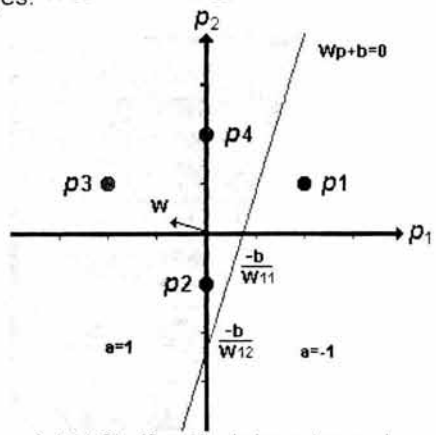


Figura 2.1.11 Clasificación de los patrones de acuerdo a la iteración 0

Como puede verse, la característica de decisión es ortogonal al vector de pesos ***W***. La red clasifica incorrectamente los patrones p_1, p_3 y p_4; en esta iteración; a continuación presentamos a la red el patrón de entrenamiento p_1.

- Iteración 1

$$W^0 = [-0.7 \quad 0.2] \; b^0 = [0.5]$$

$$a = hardlims\left(\begin{bmatrix}-0.7 & 0.2\end{bmatrix}\begin{bmatrix}2\\1\end{bmatrix}+[0.5]\right)$$

$$a = hardlims(-0.7) = -1$$

$$e = t - a = 1 - (-1) = 2$$

De la iteración 0 p_1 estaba mal clasificado, la actualización de pesos permite que este patrón sea clasificado correctamente.

$$W^1 = W^0 + ep^T \quad W^1 = [-0.7 \quad 0.2] + 2[2 \quad 1] = [3.3 \quad 2.2]$$

$$b^1 = b^0 + e \quad b^1 = 0.5 + 2 = 2.5$$

La iteración 1 lleva a la característica de decisión de la figura 2.1.12

Interceptos con los ejes: $\frac{-b}{W_{11}} = -0.75 \quad \frac{-b}{W_{21}} = -1.13$

Como se observa el patrón de entrenamiento p_1 ha sido clasificado correctamente, y casualmente los patrones p_2 y p_3 fueron correctamente ubicados, pues aun no han sido presentados a la red.

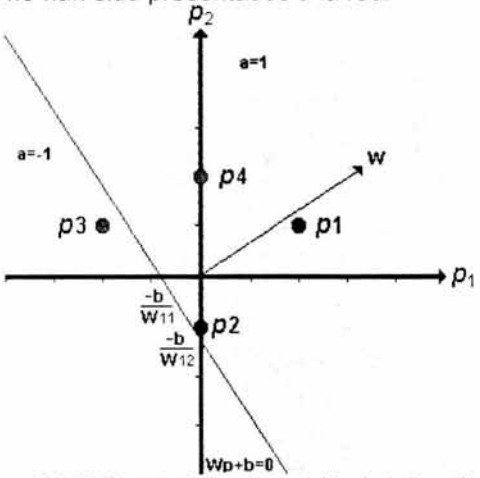


Figura 2.1.12 Característica de decisión de la iteración 1

- Iteración 2

Se presenta p_2 a la red, y es clasificado correctamente, como se observo gráficamente

$$W^1 = [3.3 \quad 2.2] \quad b^1 = [2.5]$$

$$a = hardlims\left(\begin{bmatrix}3.3 & 2.2\end{bmatrix}\begin{bmatrix}0\\-1\end{bmatrix}+[2.5]\right)$$

$$a = hardlims(0.3) = 1$$

$$e = t - a = 1 - (1) = 0$$

Este patrón ha sido clasificado correctamente y por lo tanto no hay actualización del set de entrenamiento

$$W^2 = W^1 + eP^T W^2 = [3.3 \quad 2.2] + 0[0 \quad -1] = [3.3 \quad 2.2]$$

$$b^2 = b^1 + e\,b^2 = 2.5 + 0 = 2.5$$

- Iteración 3

Se presenta p_3 a la red y es clasificado correctamente, como se observó gráficamente

$$W^2 = [3.3 \quad 2.2]\; b^2 = [2.5]$$

$$a = hardlims\left([3.3 \quad 2.2]\begin{bmatrix}-2\\1\end{bmatrix} + [2.5]\right)$$

$$a = hardlims(-1.9) = -1$$

$$e = t - a = -1 - (-1) = 0$$

Como se esperaba, no hubo error en la clasificación de este patrón, y esto lleva a que no haya actualización de los pesos de la red

$$W^3 = W^2 + eP^T W^3 = [3.3 \quad 2.2] + 0[-2 \quad 1] = [3.3 \quad 2.2]$$

$$b^3 = b^2 + e\,b^3 = 2.5 + 0 = 2.5$$

- Iteración 4

Se presenta a la red p_4,

$$W^3 = [3.3 \quad 2.2]\; b^3 = [2.5]$$

$$a = hardlims\left([3.3 \quad 2.2]\begin{bmatrix}0\\2\end{bmatrix} + [2.5]\right)$$

$$a = hardlims(6.9) = 1$$

$$e = t - a = -1 - (1) = -2$$

La red ha clasificado incorrectamente este patrón y por lo tanto deben modificarse pesos y ganancias

$$W^4 = W^3 + eP^T W^4 = [3.3 \quad 2.2] - 2[0 \quad 2] = [3.3 \quad -1.8]$$

$$b^4 = b^3 + e\,b^4 = 2.5 - 2 = 0.5$$

En esta iteración la red se comportara de acuerdo a la característica de decisión de la figura 2.1.13

Interceptos con los ejes: $\dfrac{-b}{W_{11}} = -0.15$ $\dfrac{-b}{W_{21}} = 0.27$

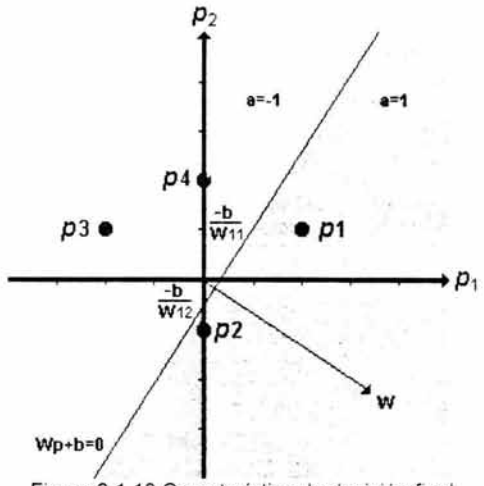


Figura 2.1.13 Característica de decisión final

De la figura 2.1.13 se observa que la red ha clasificado correctamente los patrones de entrenamiento, después de entrenada la red con los pesos y ganancias finales, cualquier otro valor de entrada será clasificado según la característica de decisión mostrada.

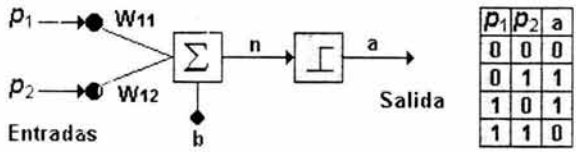
Es de importancia notar que en este caso los patrones de entrada se encuentran en dos dimensiones y por lo tanto es fácil determinar gráficamente cuando han sido clasificados correctamente, en el caso que los patrones se encuentren en tres dimensiones esta visualización se dificulta y en el caso de que los patrones sean de orden superior la visualización resulta imposible; para estos casos se debe comprobar matemáticamente que el error correspondiente a cada patrón de entrenamiento para los pesos finales es nulo.

Limitación de la red Perceptrón

En la sección 2.1.1, se planteó la restricción que existe para los tipos de problemas que una red Perceptrón puede solucionar, como se dijo esta red puede resolver solamente problemas que sean linealmente separables, esto es problemas cuyas salidas estén clasificadas en dos categorías diferentes y que permitan que su espacio de entrada sea divido en estas dos regiones por medio de un hiperplano de características similares a la ecuación del Perceptrón, es decir

$$wp + b = 0 \quad (2.1.19)$$

Ejemplos de problemas de este tipo son las funciones lógicas OR y AND estudiadas anteriormente; para ilustrar más claramente que significa que un problema sea linealmente separable se analizará un caso que no lo sea, el caso de la compuerta XOR, el cual se visualiza en la figura 2.1.14



p1	p2	a
0	0	0
0	1	1
1	0	1
1	1	0

Figura 2.1.14 Compuerta XOR

Se pretende que para los valores de entrada 00 y 11 se devuelva la clase 0 y para los patrones 01 y 10 la clase 1. Como puede verse de la figura 2.1.15 el problema radica en que no existe ninguna línea recta que separe los patrones de una clase de los de la otra

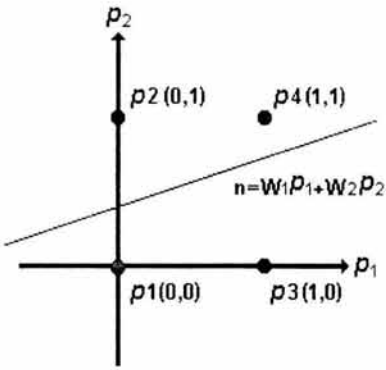


Figura 2.1.15 Plano formado por el problema de la XOR

Los cuatro puntos en la figura son las posibles entradas de la red; la línea divide el plano en dos regiones, por lo que se podría clasificar los puntos de una región como pertenecientes a la clase que posee salida 1 (puntos azules) y los de la otra región como pertenecientes a la clase que posee salida 0 (puntos rojos), sin embargo no hay ninguna forma de posicionar la línea para que los puntos correctos para cada clase se encuentren en la misma región. El problema de la compuerta XOR no es linealmente separable y una red tipo Perceptrón no esta en capacidad de clasificar correctamente los patrones de esta función, debido a esta limitación del Perceptrón y a su amplia publicación en el libro de Minsky y Papert, el estudio de las redes neuronales se estanco durante casi 20 años.

El proceso para determinar si un problema es linealmente separable o no, se realiza gráficamente sin problema, cuando los patrones de entrada generan un espacio de dos dimensiones, como en el caso de las funciones AND, OR o de la XOR; sin embargo, esta visualización se dificulta cuando el conjunto de patrones de entrada es de tres dimensiones, y resulta imposible de observar gráficamente cuando los patrones de entrada son de dimensiones superiores; en este caso se requiere plantear condiciones de desigualdad que permitan comprobar la separabilidad lineal de los patrones, esto se realiza con base en la ecuación de salida del Perceptrón

$Wp+b \geq 0$, para aquellos patrones cuya salida deseada sea 1

$Wp+b < 0$, para aquellos patrones cuya salida deseada sea 0

En el caso de la XOR, teniendo en cuenta los valores de la tabla al lado derecho de la figura 2.1.14, estas desigualdades se expresan así:

$$0*W_{1,1}+0*W_{2,1}+b<0_{(p_1)} \quad 1*W_{1,1}+0*W_{2,1}+b\geq 0_{(p_3)}$$

$$0*W_{1,1}+1*W_{2,1}+b\geq 0_{(p_2)} \quad 1*W_{1,1}+1*W_{2,1}+b<0_{(p_4)}$$

Si no hay contradicción en las desigualdades anteriores, el problema es linealmente separable. Como se observa de las desigualdades 2, 3 y 4, es imposible que $W_{2,1}\geq 0$, $W_{1,1}\geq 0$ y que su suma sea menor que cero, esta es una forma alternativa de comprobar que el problema de la XOR no es linealmente separable. El aporte de esta técnica se aprecia mejor para problemas cuyo espacio de entrada sea de dimensiones mayores.

La solución al problema de clasificación de patrones de la función XOR se encontraría fácilmente si se descompone el espacio en tres regiones: una región pertenecería a una de las clases de salida y las otras dos pertenecen a la segunda clase, así que si en lugar de utilizar únicamente una neurona de salida se utilizaran dos, se obtendrían dos rectas por lo que podrían delimitarse tres zonas; para poder elegir entre una zona u otra de las tres, es necesario utilizar otra capa con una neurona cuyas entradas serán las salidas de las neuronas anteriores; las dos zonas o regiones que contienen los puntos (0,0) y (1,1) se asocian a una salida nula de la red y la zona central se asocia a la salida con valor 1, de esta forma es posible encontrar una solución al problema de la función XOR, por tanto se ha de utilizar una red de tres neuronas, distribuidas en dos capas para solucionar este problema.

En la figura 2.1.16 se observa un esquema de lo que sería una red Perceptrón multicapa, con los valores de pesos y ganancias que clasifican correctamente los patrones de la compuerta XOR

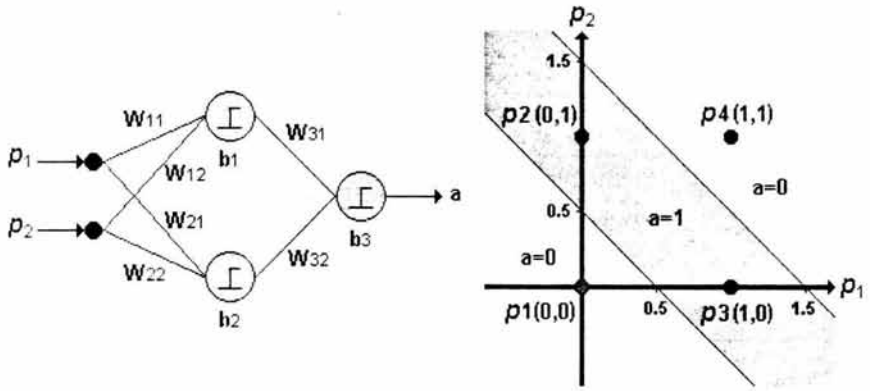


Figura 2.1.16 Perceptrón multicapa para la XOR

Los valores de la matriz de pesos y del vector de ganancias son:

$$w_{11}=1\ w_{12}=1$$
$$w_{21}=1\ w_{22}=1$$
$$w_{31}=1\ w_{32}=-1.5$$
$$b_1=0.5\ b_2=1.5\ b_3=0.5$$

Perceptron Multicapa: En el problema de la función XOR se explicó como un Perceptrón multicapa había sido implementado para hallar una solución, el esquema general de un Perceptrón multicapa puede encontrarse generalizando la figura 2.4.1 a una red con múltiples entradas y que incluya una entrada adicional representada por la ganancia *b*, este esquema general se ve en la figura 2.1.17 en donde se notan las conexiones entre sus nodos de entrada y las neuronas de salida.

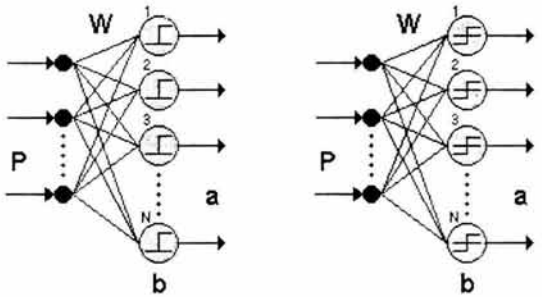


Figura 2.1.17 Conexiones del Perceptrón

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

Un esquema simplificado del modelo del Perceptrón de la figura 2.1.17 se observa en la figura 2.1.18

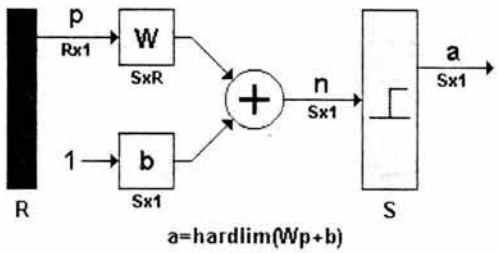


Figura 2.1.18 Notación compacta para la red tipo Perceptrón

La salida de la red está dada por:

$$a = hardlim(W * p + b) \quad (2.1.20)$$

Donde

W: Matriz de pesos asignada a cada una de las entradas de la red de dimensiones SxR, con S igual al número de neuronas, y R la dimensión del vector de entrada

p: Vector de entradas a la red de dimensiones Rx1

b: Vector de ganancias de la red de dimensiones Sx1

Las capacidades del Perceptrón multicapa con dos y tres capas y con una única neurona en la capa de salida se muestran en la figura 2.1.19 extraída del libro de Hilera J y Martínez V []. En la segunda columna se muestra el tipo de región de decisión que se puede formar con cada una de las configuraciones, en la siguiente se indica el tipo de región que se formaría para el problema de la XOR, en las dos últimas columnas se muestran las regiones formadas para resolver el problema de clases mezcladas y las formas más generales para cada uno de los casos.

Estructura	Regiones de Decisión	Problema de la XOR	Clases con Regiones Mezcladas	Formas de Regiones más Generales

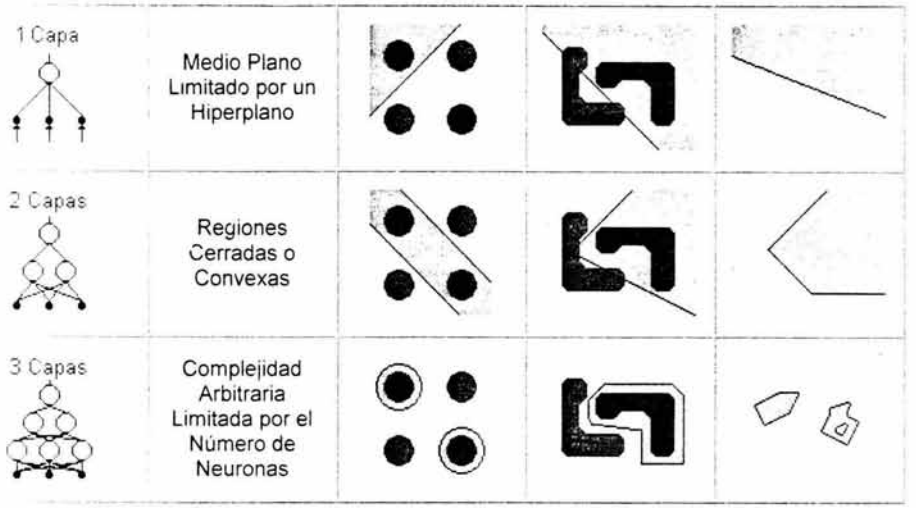


Figura 2.1.19 Distintas formas de las regiones generadas por un Perceptrón multicapa

El Perceptrón básico sólo puede establecer dos regiones separadas por una frontera lineal en el espacio de entrada de los patrones; un Perceptrón con dos capas, puede formar cualquier región convexa en este espacio. Las regiones convexas se forman mediante la intersección de regiones formadas por cada neurona de la segunda capa, cada uno de estos elementos se comporta como un Perceptrón simple, activándose su salida para los patrones de un lado del hiperplano, si el valor de los pesos de las conexiones entre las neuronas de la segunda capa y una neurona del nivel de salida son todos igual a 1, y la función de salida es de tipo *hardlim*, la salida de la red se activará sólo si las salidas de todos los nodos de la segunda capa están activos, esto equivale a ejecutar la función lógica AND en el nodo de salida, resultando una región de decisión intersección de todos los semiplanos formados en el nivel anterior. La región de decisión resultante de la intersección será una región convexa con un número de lados a lo sumo igual al número de neuronas de la segunda capa.

A partir de este análisis surge el interrogante respecto a los criterios de selección para las neuronas de las capas ocultas de una red multicapa, este número en general 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 la estimación de los pesos puede ser no confiable para el conjunto de los patrones de entrada disponibles. Hasta el momento no hay un criterio establecido para determinar la configuración de la red y esto depende más bien de la experiencia del diseñador.

La regla de aprendizaje del Perceptrón para una red multicapa es una generalización de las ecuaciones (2.1.17) y (2.1.18)

$$ {}_1\boldsymbol{W}^{nuevo} = {}_1\boldsymbol{W}^{anterior} + \boldsymbol{e}\boldsymbol{p}^{T} \quad (2.1.21) $$

$$ \boldsymbol{b}^{nueva} = \boldsymbol{b}^{anterior} + \boldsymbol{e} \quad (2.1.22) $$

ADALINE

Antecedentes: Al mismo tiempo que Frank Rosenblatt trabajaba en el modelo del Perceptrón Bernard Widrow y su estudiante Marcian Hoff introdujeron el modelo de la red Adaline y su regla de aprendizaje llamada algoritmo LMS (Least Mean Square).

La red Adaline es similar al Perceptrón, excepto en su función de transferencia, la cual es una función de tipo lineal en lugar de un limitador fuerte como en el caso del Perceptrón. La red Adaline presenta la misma limitación del Perceptrón en cuanto al tipo de problemas que pueden resolver, ambas redes pueden solo resolver problemas linealmente separables, sin embargo el algoritmo LMS es más potente que la regla de aprendizaje del Perceptrón ya que minimiza el error medio cuadrático, la regla sirvió de inspiración para el desarrollo de otros algoritmos, este es el gran aporte de esta red.

El término Adaline es una sigla, sin embargo su significado cambió ligeramente a finales de los años sesenta cuando decayó el estudio de las redes neuronales, inicialmente se llamaba ADAptive LInear NEuron (Neurona Lineal Adaptiva), para pasar después a ser Adaptive LInear Element (Elemento Lineal Adaptivo), este cambio se debió a que la Adaline es un dispositivo que consta de un único elemento de procesamiento, como tal no es técnicamente una red neuronal.

El elemento de procesamiento realiza la suma de los productos de los vectores de entrada y de pesos, y aplica una función de salida para obtener un único valor de salida, el cual debido a su función de transferencia lineal será +1 si la sumatoria es positiva o –1 si la salida de la sumatoria es negativa. En términos generales la salida de la red está dada por

$$\boldsymbol{a} = \boldsymbol{W}^T \boldsymbol{p} \quad (2.2.1)$$

En este caso, la salida es la función unidad al igual que la función de activación; el uso de la función identidad como función de salida y como función de activación significa que la salida es igual a la activación, que es la misma entrada neta al elemento.

El Adaline es **AD**aptivo en el sentido de que existe un procedimiento bien definido para modificar los pesos con objeto de hacer posible que el dispositivo proporcione el valor de salida correcto para la entrada dada; el significado de correcto para efectos del valor de salida depende de la función de tratamiento de señales que esté siendo llevada a cabo por el dispositivo. El Adaline es **LI**neal porque la salida es una función lineal sencilla de los valores de la entrada. Es una **NE**urona tan solo en el sentido (muy limitado) del PE. También se podría decir que el Adaline es un Elemento Lineal, evitando por completo la definición como **Ne**urona.

Estructura de la red: La estructura general de la red tipo Adaline puede visualizarse en la figura 2.2.1

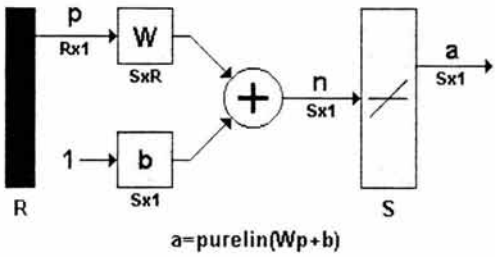


Figura 2.2.1 Estructura de una red Adaline

La salida de la red está dada por:

$$a = purelin(Wp + b) = Wp + b \quad (2.2.2)$$

Para una red Adaline de una sola neurona con dos entradas el diagrama corresponde a la figura 2.2.2

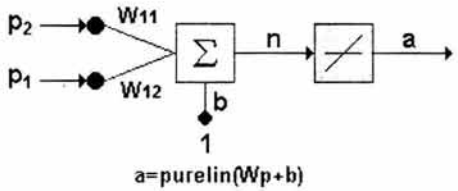


Figura 2.2.2 Adaline de una neurona y dos entradas

En similitud con el Perceptrón, el límite de la característica de decisión para la red Adaline se presenta cuando n = 0, por lo tanto:

$$\boldsymbol{w}^T \boldsymbol{p} + b = 0 \quad (2.2.3)$$

especifica la línea que separa en dos regiones el espacio de entrada, como se muestra en la figura 2.2.3

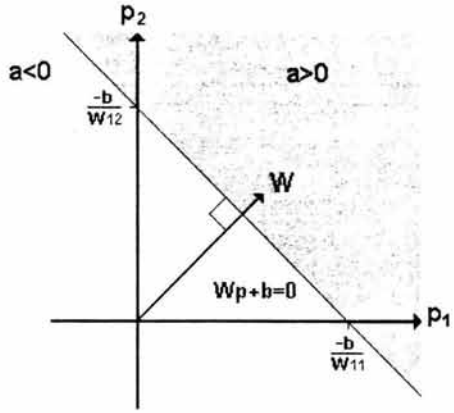


Figura 2.2.3. Característica de decisión de una red tipo Adaline

La salida de la neurona es mayor que cero en el área gris, en el área blanca la salida es menor que cero. Como se mencionó anteriormente, la red Adaline puede clasificar correctamente patrones linealmente separables en dos categorías.

Regla de aprendizaje: Al igual que el Perceptrón, la red Adaline es una red de aprendizaje supervisado que necesita conocer de antemano los valores asociados a cada entrada. Los pares de entrada/salida tienen la siguiente forma:

$$\{p_1,t_1\},\{p_2,t_2\},\ldots,\{p_Q,t_Q\} \quad (2.2.4)$$

Donde $p_Q$es la entrada a la red y $t_Q$es su correspondiente salida deseada, cuando una entrada *p* es presentada a la red, la salida de la red es comparada con el valor de *t* que le es asociado.

El algoritmo LMS se deriva de la regla Widrow-Hoff Delta, la que en términos generales para un proceso de actualización de los pesos de una red Adaline, se deduce de la siguiente manera, de acuerdo al procedimiento descrito en Widrow[]

$$\boldsymbol{W}(k+1) = \boldsymbol{W}(k) + \alpha \frac{e(k)\boldsymbol{p}(k)}{|\boldsymbol{p}(k)|^2} \quad (2.2.5)$$

En el cual *k* representa la iteración actual del proceso de actualización, $\boldsymbol{W}(k+1)$ es el siguiente valor que tomará el vector de pesos y $\boldsymbol{W}(k)$es el valor actual del vector de pesos. El error actual *e(k)* es definido como la diferencia entre la respuesta deseada $\boldsymbol{t}(k)$y la salida de la red $\boldsymbol{a}(k) = \boldsymbol{W}^T(k)\boldsymbol{p}(k)$antes de la actualización:

$$e(k) = \mathbf{t}(k) - \mathbf{W}^T(k)\mathbf{p}(k) \quad (2.2.6)$$

La variación del error en cada iteración es representada por

$$\Delta e(k) = \Delta\left(\mathbf{t}(k) - \mathbf{W}^T(k)\mathbf{p}(k)\right) = -\mathbf{p}^T(k) * \mathbf{W}(k) \quad (2.2.7)$$

En concordancia con la ecuación (2.2.5) la actualización de los pesos, teniendo en cuenta el error es:

$$\Delta W(k) = \mathbf{W}(k+1) - \mathbf{W}(k) = \alpha \frac{e(k)\mathbf{p}(k)}{|\mathbf{p}(k)|^2} \quad (2.2.8)$$

Combinando las ecuaciones (2.2.8) y (2.2.7), se obtiene:

$$\Delta e(k) = -\alpha \frac{e(k)\mathbf{p}^T(k)\mathbf{p}(k)}{|\mathbf{p}(k)|^2} = -\alpha\ e(k) \quad (2.2.9)$$

De esta forma, el error es reducido por un factor αmientras los pesos van cambiando a medida que se presenta un valor de entrada. Cada vez que se presenta un nuevo patrón el ciclo de actualización inicia nuevamente; el siguiente error es reducido por un factor α, y el proceso continua. Los valores iniciales del vector de pesos son usualmente escogidos como cero y se actualizan hasta que el algoritmo alcance convergencia.

La elección de αcontrola la estabilidad y velocidad de la convergencia del proceso de entrenamiento como puede verse en la ecuación (2.2.9); si se escoge un valor muy pequeño de α,el algoritmo pierde velocidad y tarda mucho en alcanzar convergencia, si por el contrario se toma un valor muy grande, el algoritmo pierde estabilidad y se torna oscilante alrededor del valor de convergencia. Para patrones de entrada independientes en el tiempo, la estabilidad es garantizada para valores de αque varíen entre

$$0 < \alpha < 2 \quad (2.2.10)$$

Si se fija αen un valor mayor a 1 el error es innecesariamente sobre-corregido, por lo tanto un rango de valores prácticos para la rata de aprendizaje es:

$$0.1 < \alpha < 1 \quad (2.2.11)$$

Este algoritmo es auto-normalizado en el sentido que la elección de αno depende de la magnitud de las señales de entrada; cada peso actualizado es colineal con los parámetros de entrada y su magnitud es inversamente proporcional a $|p(k)|^2$. Si se emplea como entradas binarias 1 y 0, la actualización no ocurre para pesos cuya entrada sea cero, mientras con entradas binarias ± 1todos los pesos son actualizados en cada iteración y la

convergencia es más rápida. Por esta razón, las entradas simétricas +1 y –1 son generalmente preferidas.

Una descripción geométrica del proceso de actualización de pesos en la regla Widrow-Hoff delta o algoritmo LMS, se describe en la figura 2.2.4

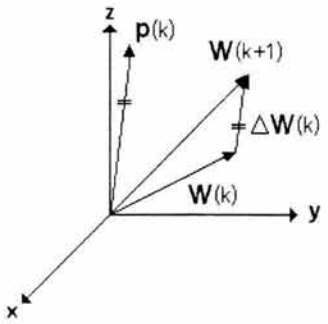


Figura 2.2.4 Actualización de pesos del algoritmo LMS

De acuerdo a la ecuación (2.2.8), $\boldsymbol{W}(k+1)$equivale la suma de $\boldsymbol{W}(k)$y $\Delta\boldsymbol{W}(k)$, además $\Delta\boldsymbol{W}(k)$es paralelo con el vector de entrada $\boldsymbol{p}(k)$. De la ecuación (2.2.7), el cambio en el error es igual al producto negativo de $\boldsymbol{p}(k)$y $\Delta\boldsymbol{W}(k)$, como el algoritmo LMS selecciona a $\Delta\boldsymbol{W}(k)$de tal forma que sea colineal con $\boldsymbol{p}(k)$, el cambio en el error deseado se calcula con la menor magnitud de $\Delta\boldsymbol{W}(k)$ posible, empleando el principio de mínima perturbación [].

Extendiendo el algoritmo a la actualización de las ganancias, se tiene:

$$b(k+1) = b(k) + \alpha e(k) \quad (2.2.12)$$

El algoritmo LMS corrige el error y si todos los patrones de entrada son de igual longitud, la actualización de pesos y ganancias tiende a minimizar el error medio cuadrático, esta es la principal propiedad de este algoritmo.

En el algoritmo LMS, los valores de los incrementos $\Delta\boldsymbol{W}(k)$y $\Delta\boldsymbol{b}(k)$se calculan con base en las derivadas parciales de la función del error medio cuadrático con respecto a pesos y ganancias respectivamente.

Para explicar el cálculo del error medio cuadrático se considerará una red Adaline y se empleará un algoritmo de pasos descendientes aproximado, como el que utilizaron Widrow y Hoff; con este algoritmo calculando el gradiente en cada iteración (gradiente instantáneo) y no el gradiente verdadero, la función para el error medio cuadrático es:

$$e^2(k) = (t(k) - a(k))^2 \quad (2.2.13)$$

En la ecuación 2.2.13 $t(k)$ representa la salida esperada en la iteración *k* y $a(k)$ representa la salida de la red; el error cuadrático esperado ha sido reemplazado por el error cuadrático en la iteración *k*, por lo tanto en cada iteración se tiene un gradiente del error de la siguiente forma:

$$\left[\nabla e^2(k)\right]_j = \frac{\partial e^2(k)}{\partial w_{i,j}} = 2e(k)\frac{\partial e(k)}{\partial w_{1,j}} \; para \; j = 1,2,...,R \qquad (2.2.14)$$

y

$$\left[\nabla e^2(k)\right]_{R+1} = \frac{\partial e^2(k)}{\partial b} = 2e(k)\frac{\partial e(k)}{\partial b} \qquad (2.2.15)$$

Los primeros R elementos del error son derivadas parciales con respecto a los pesos de la red, mientras que los elementos restantes son derivadas parciales con respecto a las ganancias

Se evaluará primero la derivada parcial de $e(k)$ con respecto a $w_{i,j}$:

$$\frac{\partial e(k)}{\partial w_{i,j}} = \frac{\partial\left[t(k) - (_1\mathbf{w}^T * \mathbf{p}(k) + b)\right]}{\partial w_{i,j}}$$

$$= \frac{\partial\left[t(k) - \left[\sum_{i=1}^{R} w_{1,i}\, p_i(k) + b\right]\right]}{\partial w_{i,j}} \qquad (2.2.16)$$

Donde $p_i(k)$ es el i-ésimo elemento del vector de entrada en la k-ésima iteración, esto puede simplificarse así:

$$\frac{\partial e(k)}{\partial w_{i,j}} = -p_j(k) \qquad (2.2.17)$$

De manera similar se obtiene el elemento final del gradiente, correspondiente a la derivada parcial del error con respecto a la ganancia:

$$\frac{\partial e(k)}{\partial b} = -1 \qquad (2.2.18)$$

En esta ecuación pueden verse las ventajas de la simplificación del error medio cuadrático al poder ser calculado por medio del error en la iteración *k*, y así para calcular el error se necesita solo multiplicar el error por el número de entradas.

La aproximación de $\nabla e(k)$ encontrada en la ecuación (2.2.14) es reemplazada en la ecuación (2.2.5) que define el proceso de actualización de pesos para el algoritmo LMS; después de haber evaluado las derivadas parciales el proceso de actualización puede expresarse como sigue:

$$\boldsymbol{W}(k+1) = \boldsymbol{W}(k) + 2\alpha e(k)\,\boldsymbol{p}(k) \quad (2.2.19)$$

$$b(k+1) = b(k) + 2\alpha e(k) \quad (2.2.20)$$

Ahora t(k) y $\boldsymbol{W}$(k) son términos independientes. Las ecuaciones (2.2.19) y (2.2.20) conforman la regla de actualización de parámetros empleada por una red Adaline, la rata de aprendizaje α se tomó constante durante el proceso de deducción del algoritmo.

En forma matricial el algoritmo de actualización para pesos y ganancias para la red Adaline, se expresa como:

$$\boldsymbol{W}(k+1) = \boldsymbol{W}(k) + 2\alpha \mathbf{e}(k)\,\boldsymbol{p}^T(k) \quad (2.2.21)$$

$$\boldsymbol{b}(k+1) = \boldsymbol{b}(\mathrm{k}) + 2\alpha \mathbf{e}(k) \quad (2.2.22)$$

Nótese que ahora el error ***e*** y la ganancia ***b*** son vectores.

Principal aplicación de la red Adaline:

La red Adaline ha sido ampliamente utilizada en el procesamiento de señales; para valorar el real aporte de esta red en ese campo, se detallarán un poco las herramientas hasta ahora empleadas en los procesos de filtrado.

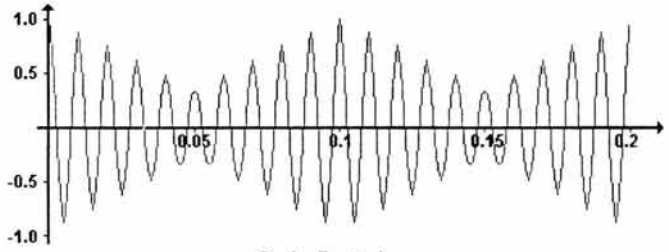
A comienzos del estudio de las comunicaciones electrónicas, se diseñaban filtros analógicos empleando circuitos RLC (Resistencias, Inductores, Condensadores) para eliminar el ruido en las señales empleadas de comunicaciones; este procesamiento se ha transformado en una técnica de múltiples facetas, destacándose en la actualidad el uso de procesadores digitales de señales (DSP), que pueden llevar a cabo los mismos tipos de aplicaciones de filtrado ejecutando filtros de convolución realizados mediante programación convencional, en cualquier lenguaje de programación conocido.

El proceso de filtrado sigue ocupando un lugar muy importante en la industria, pues siempre será necesario eliminar el ruido en señales portadoras de información. Considérese una transmisión de radio en AM, las técnicas electrónicas de comunicación, bien sean para señales de audio o de datos constan de una codificación y una modulación de la señal. La información que hay que transmitir, se puede codificar en forma de una señal analógica que reproduce exactamente las frecuencias y las amplitudes del sonido original. Dado que los sonidos que se están codificando representan un valor continuo que va desde el silencio, pasando por la voz, hasta la música, la frecuencia

instantánea de la señal variará con el tiempo, oscilando entre 0 y 10.000 Hz aproximadamente.

En lugar de intentar transmitir directamente esta señal codificada, se transmite la señal en forma más adecuada para la transmisión por radio; esto se logra modulando la amplitud de una señal portadora de alta frecuencia con la señal de información analógica. Para la radio AM, la frecuencia portadora estará en el intervalo de los 550 a los 1650 kHz , dado que la frecuencia de la portadora es muy superior a la frecuencia máxima de la señal de información, se pierde muy poca información como consecuencia de la modulación; la señal modulada puede ser transmitida después a una estación receptora (o se puede retransmitir a cualquiera que tenga un receptor de radio), en la cual la señal se demodula y se reproduce en forma de sonido.

La razón más evidente para utilizar un filtro en una radio de AM es que cada persona tiene sus preferencias de música y diversión y dado que hay tantas emisoras de radio diferentes es necesario permitir que cada usuario sintonice su receptor a una cierta frecuencia seleccionable. Al sintonizar la radio, lo que se esta haciendo es, modificar las caracteristicas de respuesta en frecuencia de un filtro *pasa banda* que está dentro de la radio, este filtro solo deja pasar las señales procedentes de la emisora en la que se este interesado y elimina todas las demás señales que estén siendo transmitidas dentro del espectro AM.



Onda Portadora

Onda de Información

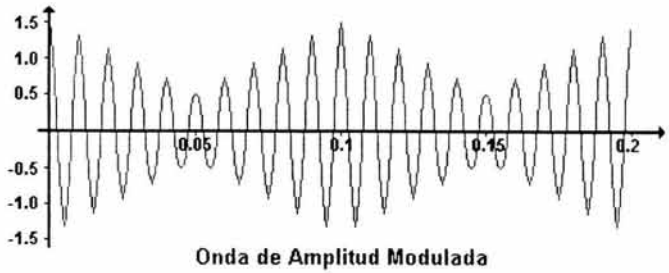


Figura 2.2.5 Técnicas de codificación de información y modulación en amplitud

La herramienta matemática para el diseño de filtros más utilizada es la Serie de Fourier, que describe la naturaleza de las señales periódicas en el dominio frecuencial y viene dada por:

$$x(t) = \sum_{n=0}^{\infty} a_n Cos(2\pi n f_0 t) + \sum_{n=1}^{\infty} b_n Sen(2\pi n f_0 t) \qquad (2.2.23)$$

En donde

f_0 Frecuencia fundamental de la señal en el dominio del tiempo
a_n y b_n Coeficientes necesarios para modular la amplitud de los términos individuales de la serie.

Las primeras realizaciones de los cuatro filtros básicos de la figura 2.2.6 poseían una gran limitación: Solo eran ajustables en un pequeño intervalo

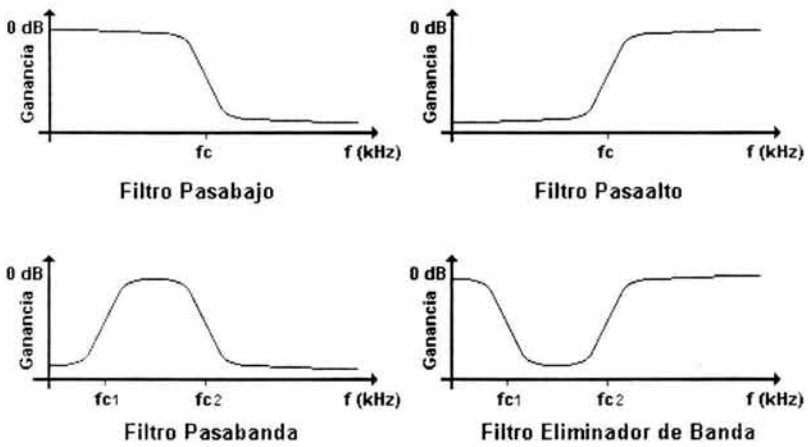


Figura 2.2.6 Características de los cuatro filtros básicos

Todos los filtros se pueden caracterizar a partir de su respuesta $h(n)$ a la función de impulso unitario, que se representa por $\delta(n)$ en la forma:

$$h(n) = R[\delta(n)] \quad (2.2.24)$$

La ventaja de esta formulación es que una vez se conoce la respuesta del sistema para el impulso unitario, la salida del sistema para cualquier entrada está dada por

$$y(n) = R[x(n)] = \sum_{i=-\infty}^{\infty} h(i)\, x(n-i) \quad (2.2.25)$$

Donde $x(n)$ es la entrada al sistema

Esta ecuación describe una convolución entre la señal de entrada y la respuesta del sistema al impulso unitario. Para este caso, basta tener en cuenta que la convolución es una operación de suma entre productos, similar al tipo de operación que realiza un Perceptrón cuando calcula su señal de activación. La red Adaline emplea este mismo calculo para determinar cuanta estimulación de entrada recibe a partir de una señal instantánea de entrada; esta red tiene diseñado en su interior una forma de adaptar los coeficientes ponderables (pesos de la red) para hacer aumentar o disminuir la estimulación que recibirá la próxima vez que se le presente la misma señal. La utilidad de esta capacidad se pone de manifiesto cuando se diseña un filtro digital por medio de software; con un programa normal, el programador debe saber exactamente como se especifica el algoritmo de filtrado y cuales son los detalles de las características de las señales; si se necesitaran modificaciones, o si cambian las características de la señal, es necesario reprogramar; cuando se emplea una red tipo Adaline, el problema se convierte, en que la red sea capaz de especificar la señal de salida deseada, dada una señal de entrada específica.

La red Adaline toma la entrada y la salida deseada, y se ajusta a sí misma para ser capaz de llevar a cabo la transformación deseada. Además, si cambian las características de la señal, la red Adaline puede adaptarse automáticamente.

En orden a usar la red tipo Adaline para implementar un filtro adaptivo, se debe incorporar el concepto de retardos en línea, el cual se visualiza en la figura 2.2.7

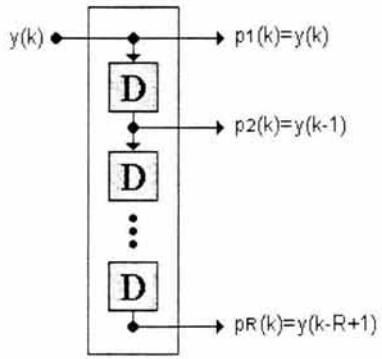


Figura 2.2.7 Retardos en linea

Si se combina la red Adaline con un bloque de retardos en línea, se ha creado un filtro adaptivo como el de la figura 2.2.8

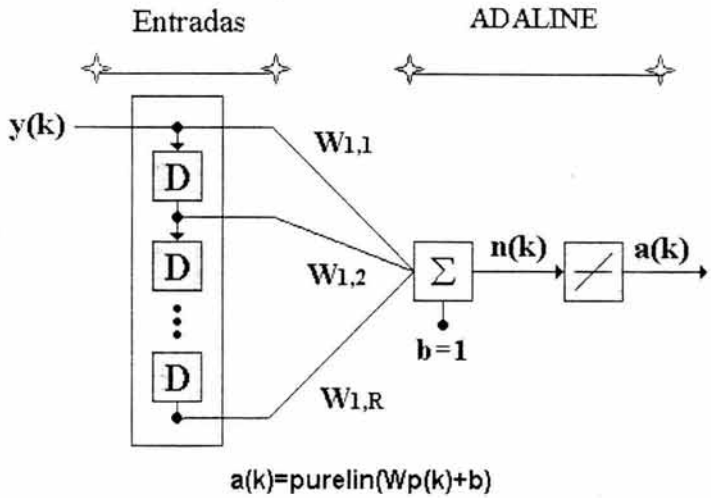


Figura 2.2.8 Filtro adaptivo

Cuya salida está dada por:

$$a(k) = purelin = (\boldsymbol{Wp} + b) = \sum_{i=1}^{R} w_{1,i}\, y(k-i+1) + b \qquad (2.2.26)$$

APRENDIZAJE ASOCIATIVO

Antecedentes: Las redes con aprendizaje no supervisado (también conocido como auto-supervisado) no requieren influencia externa para ajustar los pesos de las conexiones entre sus neuronas, la red no recibe ninguna información por parte del entorno que le indique si la salida generada en respuesta a una determinada entrada es o no correcta, por ello suele decirse que estas redes son capaces de auto organizarse.

Estas redes deben encontrar las características, regularidades, correlaciones o categorías que se puedan establecer entre los datos que se presenten en su entrada; puesto que no hay supervisor que indique a la red la respuesta que debe generar ante una entrada concreta, cabría preguntarse precisamente por lo que la red genera en estos casos, existen varias posibilidades en cuanto a la interpretación de la salida de estas redes que dependen de su estructura y del algoritmo de aprendizaje empleado.

En algunos casos, la salida representa el grado de familiaridad o similitud entre la información que se le está presentando en la entrada de la red y las informaciones que se le han mostrado hasta entonces, en otro caso la red podría realizar una clusterización (clustering) o establecimiento de categorías, indicando la salida de la red a que categoría pertenece la información presentada a la entrada, siendo la propia red quien deba encontrar las categorías apropiadas a partir de correlaciones entre las informaciones presentadas. Una variación de esta categorización es el prototipado, en este caso la red obtiene ejemplares o prototipos representantes de las clases a las que pertenecen las informaciones de entrada.

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

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

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

- Aprendizaje asociativo
- Aprendizaje competitivo

En el primer caso normalmente se pretende medir la familiaridad o extraer características de los datos de entrada, mientras que el segundo suele orientarse hacia la clusterización o clasificación de dichos datos. En esta sección se profundizará en el estudio del primero de estos algoritmos, el correspondiente al aprendizaje asociativo.
Una asociación es cualquier vínculo entre la entrada de un sistema y su correspondiente salida. Cuando dos patrones son vinculados por una asociación, el patrón de entrada es a menudo referido como el estímulo, y la salida es referida como la respuesta.
El aprendizaje asociativo fue inicialmente estudiado por escuelas de Sicología, las cuales se dedicaron a estudiar las relaciones entre el comportamiento humano y el comportamiento animal. Una de las primeras influencias en este campo fue el experimento clásico de Pavlov, en el cual se entrenó a un perro para salivar al escuchar el sonido de una campana si le era presentado un plato de comida, este es un ejemplo del llamado Condicionamiento Clásico. Otro de los principales exponentes de esta escuela fue B.F. Skinner, su experimento involucró el entrenamiento de ratas, las cuales debían presionar un botón para obtener comida, a este tipo de entrenamiento se le llamo Condicionamiento Instrumental.

Basado en este tipo de comportamiento, Donald Hebb postuló el siguiente principio conocido como la regla de Hebb:
" Cuando un axón de una celda A está lo suficientemente cerca de otra celda B como para excitarla y repetidamente ocasiona su activación, un cambio metabólico se presenta en una o ambas celdas, tal que la eficiencia de A, como celda excitadora de B, se incrementa". Con el término celda, Hebb se refería a un conjunto de neuronas fuertemente conexionadas a través de una estructura compleja, la eficiencia podría identificarse con la intensidad o magnitud de la conexión, es decir el peso.
Este postulado aplicado a redes asociativas, marcó el inicio del aprendizaje no supervisado. Un gran número de investigadores ha contribuido al aprendizaje asociativo, en particular Tuevo Kohonen, James Anderson y Stephen Grossberg. Anderson y Kohonen desarrollaron independientemente el asociador lineal a finales de los años 60's y Grossberg introdujo la red asociativa no lineal durante este mismo periodo.
Según la regla de aprendizaje de Hebb, la actividad coincidente en las neuronas présináptica y postsináptica es crítica para fortalecer la conexión entre ellas, a esto se denomina mecanismo asociativo pre-post.

Estructura de la red: La red más sencilla capaz de realizar una asociación se presenta en la figura 2.4.1, esta es una red de una sola neurona con una función de transferencia limitador fuerte

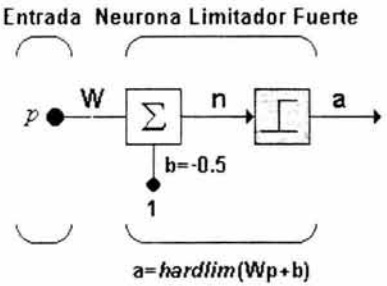


Figura 2.4.1 Asociador lineal con un limitador fuerte

La salida *a* de la neurona está determinada por su entrada *p*, de acuerdo a:

$$a=hardlim\ (wp+b) \quad (2.4.1)$$

Por simplicidad se tomará el valor de *p* como cero o uno, indicando presencia o ausencia de estímulo. El valor de *a* esta limitado por la función de transferencia con salida cero o uno.

$$p=\begin{cases}1, & presencia\ de\ estimulo\\ 0, & ausencia\ estimulo\end{cases} \qquad a=\begin{cases}1, & hay\ respuesta\ por\ parte\ de\ la\ red\\ 0, & no\ hay\ respuesta\ por\ parte\ de\ la\ red\end{cases}$$

La presencia de una asociación entre el estímulo p=1 y la respuesta a=1, es indicada por el valor de w. La red responderá al estímulo, solamente si wp es mayor que $-b$.

El estudio de redes asociativas ha evitado el uso de redes complejas, por tanto se han definido dos tipos de estímulos: un conjunto de entradas llamado *estímulo no condicionado*, análogo a la comida presentada al perro en el experimento de Pavlov y otro conjunto de entradas llamado *estímulo condicionado*, análogo a la campana en el experimento. Inicialmente el perro saliva solamente cuando la comida es presentada, esta característica innata hace que el perro aprenda. Sin embargo, cuando la campana ha acompañado la comida repetidas veces, el perro es *condicionado* a salivar con el sonido de la campana aún cuando la comida no haya sido presentada.

Definiendo las clases de entradas a una red asociativa, se tiene:

Estímulo no condicionado: Corresponde a la entrada, que pudiendo ser de carácter escalar o vectorial, refuerza el aprendizaje y ayuda a hacer la asociación con la salida deseada, este estímulo se presenta intermitentemente para simular un real proceso de aprendizaje y memorización de la red; la mayoría de las veces el estímulo no condicionado se convierte en la salida deseada de la red.

Estímulo condicionado: Es el objeto de la asociación, debe ser *siempre* presentado a la red y ésta debe asociarlo con la salida deseada; al final del proceso de aprendizaje la red debe ser capaz de entregar la respuesta correcta con la presentación de este único estímulo a su entrada, sin importar si el estímulo no condicionado ha sido presentado o no, pues la asociación ya ha sido realizada.

En este caso representaremos el estímulo no condicionado por p^0 y el estímulo condicionado simplemente por p. Los pesos w^0, asociados con p^0 se tomarán fijos y los pesos w asociados a p serán actualizados en cada iteración.

La figura 2.4.2 representa la red correspondiente al asociador lineal para una fruta, la red tiene ambos estímulos, no condicionado (forma de la fruta) y condicionado (olor de la fruta), escogidos aleatoriamente para este caso, en el cual se tratará simplemente de ilustrar el objeto de una asociación. Según la elección de los estímulos se desea que la red asocie la forma de la fruta pero no su olor, es decir el sensor de olor trabajará siempre correctamente, de tal manera que la red lo tendrá siempre presente, pero el sensor de forma trabajará intermitentemente, cuando la forma sea detectada (sensor de forma p^0=1), la red responderá correctamente identificando la fruta.

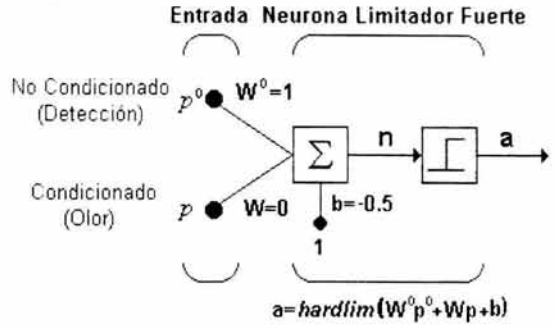


Figura 2.4.2 Asociador para una fruta

La definición de los estímulos estará dada por:

$$p^0 = \begin{cases} 1 & si\ la\ forma\ es\ detectada \\ 0 & si\ la\ forma\ no\ es\ detectada \end{cases} \qquad p = \begin{cases} 1\ si\ el\ olor\ detectado \\ 0\ \ si\ el\ olor\ no\ es\ detectado \end{cases}$$

Con el propósito de cumplir las condiciones matemáticas del ejemplo, se ha escogido b = -0.5. Para iniciar con el asociador se asignará a w^0 un valor mayor a $-b$ y a w un valor menor que $-b$. Los siguientes valores satisfacen estos requerimientos:

$$w^0=1,\ w=0 \quad (2.4.2)$$

La función de entrada/salida del asociador para una fruta, puede simplificarse a:

$$a= hardlim\ (p^0 - 0.5) \quad (2.4.3)$$

La red responderá solo si p^0=1, sin importar si p=1, o p=0, es decir la red responderá independientemente del estímulo condicionado.

Llegará un momento en que el sensor de forma no trabajará más y se espera que para ese momento la red haya realizado una asociación correcta para identificar la fruta con la sola presencia del olor, sin necesidad de que su forma tenga que ser detectada, esto se logrará variando los valores para los pesos de conexión de la red para el estímulo condicionado.

Regla de Hebb: Esta regla puede interpretarse matemáticamente teniendo en cuenta que si dos neuronas en cualquier lado de la sinápsis son activadas simultáneamente, la longitud de la sinápsis se incrementará. Si se revisa la figura 2.4.3 correspondiente a un asociador lineal, se ve como la salida a, es determinada por el vector de entrada $\boldsymbol{p}$.

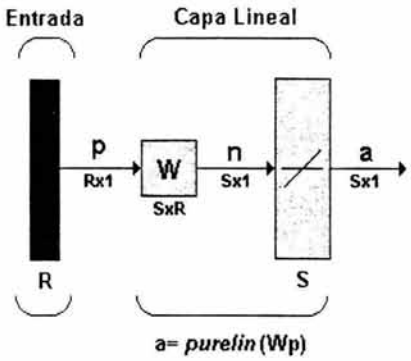


Figura 2.4.3 Asociador Lineal

$$a_i = \sum_{j=1}^{R} w_{ij} p_j \qquad (2.4.4)$$

Puede notarse como la conexión (sinápsis) entre la entrada p_j y la salida a_i es el peso w_{ij}. De esta forma el postulado de Hebb implica que si un valor positivo p_j produce un valor positivo a_i, el valor de w_{ij} debe incrementarse,

$$w_{ij}^{nuevo} = w_{ij}^{anterior} + \alpha(a_{iq})(p_{jq}) \qquad (2.4.5)$$

Donde :

p_{jq} :j-ésimo elemento del q-ésimo vector de entrada p_q
a_{iq} :i-ésimo elemento de salida de la red, cuando el q-ésimo vector de entrada es presentado
α :es la rata de aprendizaje, la cual es un valor positivo constante

La regla de Hebb dice que el cambio en el peso w_{ij} es proporcional al producto de las funciones de activación en cualquier lado de la sinápsis. Así, los pesos serán incrementados cuando p_j y a_i sean positivos, pero también lo harán cuando ambos parámetros sean negativos, en contraposición los pesos se decrementarán cuando p_j y $a_i$tengan signos contrarios.

Si se retorna a la discusión de los estímulos en animales y seres humanos, debe decirse que ambos tienden a asociar eventos que ocurren simultáneamente. Parafraseando el postulado de Hebb: "Si el estímulo del olor de la fruta, ocurre simultáneamente con la respuesta del concepto de esa fruta, (activada por algún otro estímulo como la forma de la fruta), la red debe alargar la conexión entre ellos para que después, la red active el concepto de esa fruta en respuesta a su olor solamente."

La regla de aprendizaje de Hebb determina que el incremento del peso w_{ij} entre la entrada p_j de una neurona y su salida a_i en la q-ésima iteración es:

$$w_{ij}(q) = w_{ij}(q-1) + \alpha\, a_i(q) p_j(q) \quad (2.4.6)$$

La rata de aprendizaje α determina cuantas veces un estímulo y su respuesta deben ocurrir juntos antes de que la asociación sea hecha. En la red de la figura 2.4.2, una asociación será hecha cuando $w > -b = 0.5$, entonces para una entrada $p=1$ se producirá una salida $a=1$, sin importar el valor de p^0

Para comprender el funcionamiento de la regla de Hebb, ésta se aplicará a la solución del asociador de la fruta resuelto en el numeral anterior. El asociador será inicializado con los valores determinados anteriormente

$$w^0=1,\ w(0) = 0 \quad (2.4.7)$$

El asociador será repetidamente expuesto a la fruta; sin embargo mientras el sensor de olor trabajará en forma siempre confiable (estímulo condicionado), el sensor de la forma operará intermitentemente (estímulo no condicionado). Así la secuencia de entrenamiento consiste en la repetición de la siguiente secuencia de valores de entrada:

$\{p^0(1) = 0,\ p(1) = 1\},\ \{p^0(2) = 1,\ p(2) = 1\}\ldots.$ (2.4.8)

Usando una rata de aprendizaje $\alpha = 1$, y empleando la regla de Hebb, serán actualizados los pesos w correspondientes al estímulo condicionado, ya que como se dijo anteriormente, los pesos correspondientes al estímulo no condicionado se mantendrán constantes.

La salida para la primera iteración (q=1) es:

$a(1) = hardlim(w^0 p^0(1) + w(0)\, p(1) - 0.5)$

$= hardlim\ (1*0 + 0*1 - 0.5) = 0$ *No hay respuesta* (2.4.9)

El olor solamente no ha generado una respuesta esto es, no hubo una asociación entre el olor de la fruta y el concepto de la fruta como tal, sin una respuesta la regla de Hebb, no altera el valor de w

$w(1)=w(0)+a(1)\, p(1) = 0 + 0*1=0$ (2.4.10)

En la segunda iteración, son detectados tanto la forma como el olor de la fruta y la red responderá correctamente identificando la fruta

$a(2) = hardlim\ (w^0 p^0(2) + w(1)\, p(2) - 0.5)$ (2.4.11)

$= hardlim(1*1 + 0*1 - 0.5) = 1$ *La fruta ha sido detectada*

Como el estímulo del olor y la respuesta de la red ocurrieron simultáneamente la regla de Hebb, incrementa los pesos entre ellos.

$w(2)= w(1) + a(2)\, p(2) = 0 + 1*1 = 1$ (2.4.12)

En la tercera iteración a pesar de que el sensor de la forma falla nuevamente, la red responde correctamente. La red ha realizado una asociación útil entre el olor de la fruta y su respuesta.

$a(3)=hardlim(w\,^0p^0(3)+w(2)\,p(3) - 0.5)$ (2.4.13)

$=hardlim(1*0+1*1-0.5) =1$ *La fruta ha sido detectada*

$w(3) = w(2)+a(3)\,p(3) = 1+1*1 = 2$ (2.4.14)

Ahora la red es capaz de identificar la fruta por medio de su olor o de su forma; incluso si los dos sensores tienen fallas intermitentes, la red responderá correctamente la mayoría de las veces.

Una forma de mejorar la regla de Hebb, es adicionar un término que controle el crecimiento de la matriz de peso, a esta modificación se le da el nombre de regla de Hebb con rata de olvido.

$$\begin{aligned} \boldsymbol{W}(q) &= \boldsymbol{W}(q-1) + \alpha\,\boldsymbol{a}(q)\boldsymbol{p}^T(q) - \gamma\boldsymbol{W}(q\text{-}1) \\ &= (1-\gamma)\boldsymbol{W}(q\text{-}1) + \alpha\boldsymbol{a}(q)\boldsymbol{p}^T(q) \end{aligned} \qquad (2.4.15)$$

Donde γ es la rata de olvido, la cual es una constante positiva menor que 1; cuando γ se aproxima a cero la ley de aprendizaje se convierte en la ley de Hebb estándar; cuando γ se aproxima a 1, la rata de aprendizaje olvida rápidamente las entradas anteriores y recuerda solamente los patrones más recientes. El efecto de esta nueva constante, es controlar que el crecimiento de la matriz de pesos no se realice sin límites y así darle un mejor aprovechamiento a la capacidad de memoria de la red.

Red Instar: Hasta ahora se han considerado solamente reglas de asociación entre entradas y salidas escalares. Si se examina la red de la figura 2.4.4, se nota como esta neurona esta enfrentada a un problema de reconocimiento de patrones cuya entrada es de tipo vectorial; esta neurona es el tipo de red más simple capaz de resolver esta clase de problemas y es llamada red Instar.

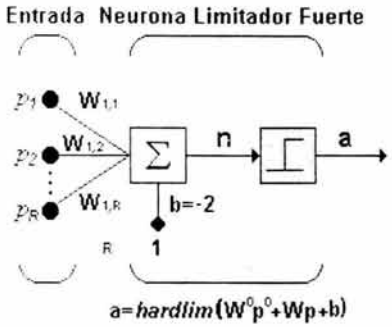


Figura 2.4.4 Red Instar

Puede notarse la similitud entre la red Instar y la red Perceptrón de la figura 2.1.6, o incluso a la red Adaline de la figura 2.2 3. Estas redes han tenido diferentes nombres, debido a razones históricas y a que su desempeño ha sido analizado en diferentes ambientes. Para la Instar no se considerará directamente su característica de decisión, concepto que fue bastante importante para el Perceptrón, en lugar de ello se analizará la capacidad de la Instar para reconocimiento de patrones a través de asociaciones y aprendizaje no supervisado.

La ecuación para determinar la entrada/salida de la Instar es:

$a = hardlims\ (\boldsymbol{w}^T\boldsymbol{p}+b)$ (2.4.16)

La red Instar se activará si el producto punto entre el vector de pesos (fila de la matriz de pesos) y la entrada sea mayor o igual a $-b$

$$\boldsymbol{w}^T\boldsymbol{p} \geq -b \quad (2.4.17)$$

Los vectores $\boldsymbol{w}$ y $\boldsymbol{p}$ son de longitud constante, por lo tanto el mayor producto punto se presentará cuando los dos vectores apunten en la misma dirección; dicho de otra forma cuando el ángulo entre $\boldsymbol{w}$ y $\boldsymbol{p}$ sea $\theta = 0$, esto permite observar que la red instar de la figura 2.4.4 se activará cuando $\boldsymbol{p}$ y $\boldsymbol{w}$ estén muy cercanos, escogiendo un apropiado valor para la ganancia b se puede determinar que tan cerca deben estar $\boldsymbol{p}$ y $\boldsymbol{w}$ para que la instar se active, si se fija

$$b = -\|\boldsymbol{w}\| \cdot \|\boldsymbol{p}\| \quad (2.4.18)$$

la instar se activará solamente cuando $\boldsymbol{p}$ apunte exactamente en la misma dirección de $\boldsymbol{w}$, de esta forma b se puede incrementar a valores ligeramente mayores a $-\|\boldsymbol{w}\| \cdot \|\boldsymbol{p}\|$, el mayor valor de b se presentará cuando la Instar esté activa. Es importante recalcar que este análisis asume que todos los vectores tienen la misma longitud.

Uno de los inconvenientes de la regla de Hebb con rata de olvido, es que requiere que los estímulos se presenten de forma repetitiva o de lo contrario se perderá la asociación, se desea encontrar una regla alternativa que habilite el término con olvido solo cuando la Instar es activa $a \neq 0$, de esta forma los valores de los pesos seguirán siendo limitados, pero el porcentaje de olvido será minimizado. Para obtener los beneficios del término de peso con rata de olvido, se adiciona un nuevo término proporcional a $a_i(q)$.

$$w_{ij}(q) = w_{ij}(q-1) + \alpha a_i(q) p_j(q) - \gamma a_i(q) w_{ij}^{anterior} \quad (2.4.19)$$

El nuevo término de peso se hace proporcional a la salida escalar $a_i(q)$, ya que se desea controlar esta salida para que reproduzca el estímulo no condicionado; si se considera que la rata a la cual la red aprende nuevos pesos es igual a la rata de olvido $\alpha = \gamma$, la ecuación (2.4.18) puede simplificarse a:

$$w_{ij}(q) = w_{ij}(q-1) + \alpha a_i(q)\left(p_j(q) - w_{ij}^{anterior}\right) \quad (2.4.20)$$

Esta ecuación es la llamada regla de Instar, que en forma vectorial teniendo en cuenta el caso en que la instar esta activa (a_i=1), se convierte en:

$\boldsymbol{w}(q) = \boldsymbol{w}(q-1) + \alpha (\boldsymbol{p}(q) - \boldsymbol{w}(q-1))$

$= (1-\alpha)\, \boldsymbol{w}(q-1) + \alpha\, \boldsymbol{p}(q)$ (2.4.21)

Esta operación se muestra en la figura 2.4.5

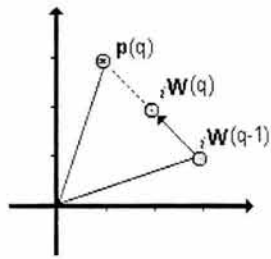


Figura 2.4.5 Representación gráfica de la regla de Instar

Cuando la instar es activa, el vector de pesos se mueve hacia el vector de entrada a lo largo de una línea entre el vector de pesos anterior y el vector de entrada. La distancia a la que se mueve el vector depende del valor de la rata de aprendizaje α. Cuando α=0, el nuevo vector de pesos es igual al vector de pesos anterior. Cuando α=1, el nuevo vector de pesos es igual al vector de entrada. Si α=0.5 el nuevo vector de pesos será la mitad entre el vector de pesos anterior y el vector de entrada.

Una característica útil de la regla Instar es que si los vectores de entrada son normalizados, entonces $\boldsymbol{w}$ será también normalizado una vez la red haya

aprendido un vector particular $\boldsymbol{p}$, esta regla no solamente minimiza la rata de olvido, también normaliza los vectores de peso si el vector de entrada es normalizado.

Se aplicará la regla de Instar para solucionar el problema de la figura 2.4.6, similar al problema del asociador para una fruta; este nuevo caso cuenta con dos entradas, una indicando si la fruta ha sido visualizada o no (estímulo no condicionado) y otra consistente en un vector de tres medidas pertenecientes a la fruta (estímulo condicionado).

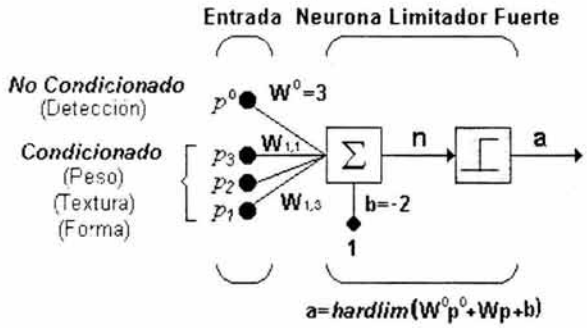


Figura 2.4.6 Reconocimiento de una fruta por medio de una Instar

La salida de la red, está determinada por

$a = hardlim(w^0p^0+\boldsymbol{W}\boldsymbol{p}+b)$

Los elementos de entrada a la red serán valores de 1 o –1, las tres propiedades que se medirán de la fruta son: forma, textura y peso, de esta manera la salida del sensor de forma será 1 si la fruta es aproximadamente redonda o –1 si la fruta es elíptica, la salida del sensor de textura será 1 si la superficie de la fruta es suave y será –1 si es rugosa y la salida del sensor de peso será 1 si la fruta pesa más de una libra o –1 si el peso de la fruta es menor de esta medida.

En este caso la elección del estímulo condicionado y el no condicionado ya no es aleatoria, pues como se dijo en análisis anteriores, el estímulo no condicionado se convierte la mayoría de las veces en la salida deseada de la red que es tipo de escalar para una red Instar, por lo tanto el sensor que representa la visualización de la red será el estímulo no condicionado y el vector de medidas de la fruta será el estímulo condicionado.

Con las dimensiones consideradas $\boldsymbol{p}$ es un vector normalizado con $\|p\| = \sqrt{3}$. La definición de p^0 y $\boldsymbol{p}$ es:

$$p^0 = \begin{cases} 1 \ \textit{fruta detectada visualmente} \\ 0 \ \textit{fruta no detectada} \end{cases} \quad \boldsymbol{p} = \begin{bmatrix} \textit{forma} \\ \textit{textura} \\ \textit{peso} \end{bmatrix}$$

El valor de la ganancia se asumirá como b = -2, un valor ligeramente más positivo que $-\|p\|_2$ = -3. Lo ideal es que la red tenga una asociación constante, entre la visualización de la fruta y su respuesta, para que w^0 sea mayor que – b.

Inicialmente la red no responderá a ninguna combinación de medidas de la fruta, puesto que la fruta no ha sido detectada visualmente, así que los pesos iniciales serán cero

w^0=3. $\boldsymbol{W}(0) = {}_1\boldsymbol{w}^T(0) = [0\ 0\ 0]$ (2.4.22)

Usando la regla Instar con una rata de aprendizaje $\alpha = 1$, los pesos actualizados se encontrarán de la siguiente forma:

$w(q) = w(q-1) + a(q)\,(p(q) - w(q-1))$ (2.4.23)

La secuencia de entrenamiento consistirá en repetidas presentaciones de la fruta, los sensores estarán actuando todo el tiempo sin embargo, en orden a observar la operación de la regla Instar se asumirá que el sensor que visualiza la fruta actuará intermitentemente, simulando así una falla en su construcción

$$\left\{ p^0(1) = 0, \boldsymbol{p}(1) = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \right\}, \left\{ p^0(2) = 1, \boldsymbol{p}(2) = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \right\} \quad (2.4.24)$$

Como la matriz $\boldsymbol{W}$ inicialmente contiene ceros, la Instar no responderá a los sensores de la fruta en la primera iteración

$a(1)=hardlim(w^0 p^0(1) + \boldsymbol{W}\boldsymbol{p}(1) - 2)$

$$a(1) = hardlim\left(3*0 + \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} - 2 \right) = 0 \quad No\ hay\ respuesta \quad (2.4.25)$$

Como la neurona no respondió, sus pesos no serán actualizados por la regla Instar

$w(0) = w(0) + a(1)(p(1) - w(0))$

$$= \begin{bmatrix}0\\0\\0\end{bmatrix} + 0\left(\begin{bmatrix}1\\-1\\1\end{bmatrix} - \begin{bmatrix}0\\0\\0\end{bmatrix} = \begin{bmatrix}0\\0\\0\end{bmatrix}\right) \quad (2.4.26)$$

En la segunda iteración, cuando la fruta haya sido detectada visualmente, la neurona responderá

$a(2) = hardlim\ (w^0p^0(2)+\mathbf{Wp}\ (2)-2)$

$$a(2) = hardlim\left(3*1 + \begin{bmatrix}0 & 0 & 0\end{bmatrix}\begin{bmatrix}1\\-1\\-1\end{bmatrix} - 2\right) = 1 \quad fruta\ detectada \quad (2.4.27)$$

El resultado es que la red aprendió a asociar el vector de medidas de la fruta con su respuesta. El vector de pesos de la red, se convierte en una copia del vector de medidas de la fruta.

$w(2) = w(1) + a\ (2)(p(2)-w\ (1))$

$$= \begin{bmatrix}0\\0\\0\end{bmatrix} + 1\left(\begin{bmatrix}1\\-1\\1\end{bmatrix} - \begin{bmatrix}0\\0\\0\end{bmatrix} = \begin{bmatrix}1\\-1\\-1\end{bmatrix}\right) \quad (2.4.28)$$

La red puede ahora reconocer la fruta por sus medidas; la neurona respondió en la tercera iteración, aún cuando el sistema de detección visual falló, por lo tanto la red realizará una asociación entre la presencia de la fruta y el vector de estímulos condicionados, sin importar si el sensor de visualización (estímulo no condiconado) opera adecuadamente.

$a(3) = hardlim\ (w^0p^0(3)+\mathbf{Wp}\ (3) - 2)$

$$a(3) = hardlim\left(3*0 + \begin{bmatrix}1 & -1 & -1\end{bmatrix}\begin{bmatrix}1\\-1\\-1\end{bmatrix} - 2\right) = 1 \quad fruta\ detectada \quad (2.4.29)$$

Cuando las medidas de la fruta han sido detectadas completamente, los pesos dejan de cambiar y se estabilizan.

$w(3) = w(2) + a\ (3)(p(3)-w\ (2))$

$$= \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} + 1 \left(\left(\begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} - \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \right) \right) \quad (2.4.30)$$

Red Outstar: Ya se ha visto como la red Instar (con una entrada tipo vector y una salida tipo escalar) puede resolver problemas de reconocimiento de patrones por asociación de un vector particular de estímulo, con su respuesta. La red Outstar, mostrada en la figura 2.4.7 tiene una entrada tipo escalar y una salida tipo vectorial y puede recordar patrones por asociación de un estímulo con un vector de respuesta.

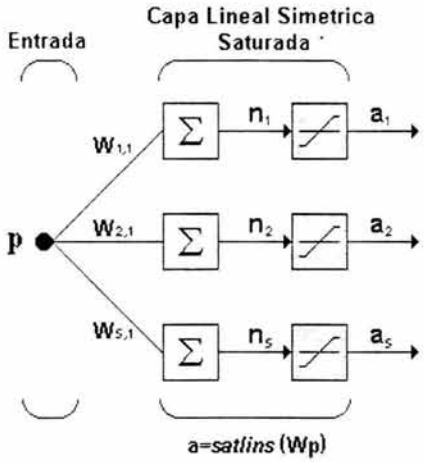


Figura 2.4.7 Red Outstar

La expresión de salida de esta red es:

$\boldsymbol{a}=satlins(\boldsymbol{W}p)$ (2.4.30)

Se desea recordar un vector con valores entre –1 y 1, para lo cual se utilizará la función de saturación simétrica *satlins*, aunque pueden usarse otras funciones como por ejemplo *hardlims*.

Para derivar la regla Instar, el problema del olvido presentado por la regla de aprendizaje de Hebb fue limitado por el nuevo término de peso, el cual era proporcional a la salida de la red a_i De manera similar, para obtener la regla de aprendizaje Outstar el término con olvido se hará proporcional a la entrada de la red p_j ya que la salida de esta red es un vector, con el cual se espera simular el estimulo no condicionado

$$w_{ij}(q) = w_{ij}(q-1) + \alpha a_i(q) p_j(q) - \gamma p_i(q) w_{ij}(q-1) \quad (2.4.31)$$

Si se hace la rata de olvido γ igual a la rata de aprendizaje α se obtiene

$w_{ij}(q) = w_{ij}(q-1) + \alpha (a_i(q) - w_{ij}(q-1)) p_j(q)$ (2.4.32)

La regla Outstar tiene propiedades complementarias a la regla Instar; el aprendizaje ocurre cuando una entrada p_j tiene un valor diferente a cero (en lugar de a_i). Cuando el aprendizaje ocurre, la columna w_j se acerca al vector de salida.

Se entrenará la red de la figura 2.4.8, para observar el funcionamiento del algoritmo

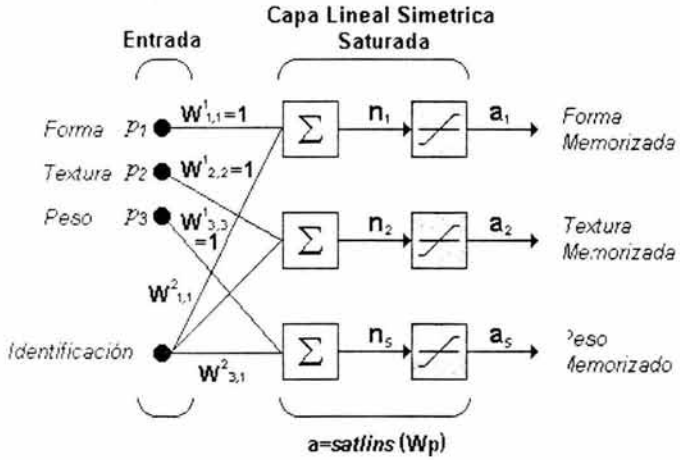


Figura 2.4.8 Reconocimiento de una fruta mediante una Outstar

La salida de la red será calculada como:

$\boldsymbol{a}=satlins(\boldsymbol{W}^0\boldsymbol{p}^0+\boldsymbol{W}p)$ (2.4.33)

Donde

$$\boldsymbol{W}^0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.4.34)$$

Continuando con el reconocimiento de frutas, los estímulos condicionado y no condicionado son:

$$\boldsymbol{p}^0 = \begin{bmatrix} forma \\ textura \\ peso \end{bmatrix} \quad p = \begin{cases} 1, & la\ fruta\ es\ visualizada \\ 0, & la\ fruta\ no\ es\ visualizada \end{cases}$$

Como puede verse el estímulo no condicionado para una red Outstar tiene forma vectorial y el estímulo no condicionado forma escalar, en forma opuesta a la red de Instar; la salida esperada de la red, es el vector de medidas de la fruta para cualquier entrada disponible.

La matriz de pesos para el estímulo no condicionado $\boldsymbol{W}^0$ es la matriz identidad, así cualquier conjunto de medidas $\boldsymbol{p}^0$ (con valores entre 1 y -1) será reproducido a la salida de la red. La matriz de pesos para el estímulo condicionado $\boldsymbol{W}$, es inicializada en ceros para que un 1 en p no genere respuesta. $\boldsymbol{W}$ será actualizada con la regla Outstar, usando una rata de aprendizaje de 1.

La secuencia de entrenamiento consiste en repetidas presentaciones de la visualización de la fruta y de sus medidas, las cuales se escogieron de la siguiente forma:

$$p^0 = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} \quad (2.4.35)$$

Para probar la red, el sistema de medidas de la red será presentado intermitentemente

$$\left\{ \boldsymbol{p}^0(1) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix},\ p(1) = 1 \right\}, \left\{ \boldsymbol{p}^0(2) = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix},\ p(2) = 1 \right\} \quad (2.4.36)$$

En la primera iteración la fruta es vista pero sus medidas no están disponibles, y como el vector de medidas es en este caso el estímulo no condicionado la red no estregará una respuesta.

$$\boldsymbol{a} = satlins(\boldsymbol{W}^0\boldsymbol{p}^0(1) + \boldsymbol{W}p(1)) \quad (2.4.37)$$

$$\boldsymbol{a}(1) = satlins\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} 1 \right) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}\ no\ hay\ respuesta$$

La red ve la fruta, pero no puede determinar sus medidas porque aún no las ha aprendido; por lo tanto los pesos no son alterados

$$\boldsymbol{w}_1(1) = \boldsymbol{w}_1(0)+(\boldsymbol{a}\,(1) - \boldsymbol{w}_1\,(2))\,\boldsymbol{p}(1) \quad (2.4.38)$$

$$= \begin{bmatrix}0\\0\\0\end{bmatrix} + \left(\begin{bmatrix}0\\0\\0\end{bmatrix} - \begin{bmatrix}0\\0\\0\end{bmatrix}\right)1 = \begin{bmatrix}0\\0\\0\end{bmatrix}$$

En la segunda iteración, tanto la fruta como sus medidas son presentadas a la red

$$\boldsymbol{a}(2) = satlins\left(\begin{bmatrix}-1\\-1\\1\end{bmatrix} + \begin{bmatrix}0\\0\\0\end{bmatrix}1\right) = \begin{bmatrix}-1\\-1\\1\end{bmatrix} \; medidas\ correctas \quad (2.4.39)$$

La red entregó las medidas de la fruta a la salida, es decir realizó la primera asociación entre la fruta y sus medidas, por lo tanto los pesos son actualizados

$$\boldsymbol{w}_1(2) = \boldsymbol{w}_1(1)+(\boldsymbol{a}\,(2) - \boldsymbol{w}_1\,(1))\,\boldsymbol{p}(2) \quad (2.4.40)$$

$$= \begin{bmatrix}0\\0\\0\end{bmatrix} + \left(\begin{bmatrix}-1\\-1\\1\end{bmatrix} - \begin{bmatrix}0\\0\\0\end{bmatrix}\right)1 = \begin{bmatrix}-1\\-1\\1\end{bmatrix}$$

Cuando la fruta fue visualizada y sus medidas presentadas, la red forma una asociación entre ellas, la matriz de pesos es ahora una copia de las medidas de la fruta y de esa forma podrá recordarlas más adelante.

En la tercera iteración, las medidas no son presentadas a la red, y aún así la red las reproduce porque las recuerda por medio de la asociación que realizó

$$\boldsymbol{a}(3) = satlins\left(\begin{bmatrix}0\\0\\0\end{bmatrix} + \begin{bmatrix}-1\\-1\\1\end{bmatrix}1\right) = \begin{bmatrix}-1\\-1\\1\end{bmatrix} \; medidas\ recordadas \quad (2.4.41)$$

Desde este momento, los pesos no sufrirán grandes cambios, a menos que la fruta sea vista con medidas diferentes

$$\boldsymbol{w}_1(3) = \boldsymbol{w}_1(2)+(\boldsymbol{a}\,(3) - \boldsymbol{w}_1\,(2))\,\boldsymbol{p}(3) \quad (2.4.42)$$

$$= \begin{bmatrix}-1\\-1\\1\end{bmatrix} + \left(\begin{bmatrix}-1\\-1\\1\end{bmatrix} - \begin{bmatrix}-1\\-1\\1\end{bmatrix}\right)1 = \begin{bmatrix}-1\\-1\\1\end{bmatrix}$$

Las redes de Instar y Outstar son empleadas conjuntamente en la red ART [], y cada una en forma independiente son utilizadas en gran cantidad de aplicaciones debido a su fácil implementación y al funcionamiento casi intuitivo de su regla de aprendizaje; las redes asociativas se utilizan principalmente para filtrado de información en la reconstrucción de datos, eliminando distorsiones o ruido, también se emplean para explorar relaciones entre informaciones similares, para facilitar la búsqueda por contenido en bases de datos y para resolver problemas de optimización.

REDES COMPETITIVAS

Antecedentes: En las redes con aprendizaje competitivo (y cooperativo), suele decirse que las neuronas compiten (y cooperan) unas con otras con el fin de llevar a cabo una tarea dada. Con este tipo de aprendizaje se pretende que cuando se presente a la red cierta información de entrada, sólo una de las neuronas de salida de la red, o una por cierto grupo de neuronas, se active (alcance su valor de respuesta máximo). Por tanto las neuronas compiten para activarse quedando finalmente una, o una por grupo, como neurona vencedora y el resto quedan anuladas y siendo forzadas a sus valores de respuesta mínimos.

La competición entre neuronas se realiza en todas las capas de la red, existiendo en estas redes neuronas con conexiones de autoexitación (signo positivo) y conexiones de inhibición (signo negativo) por parte de neuronas vecinas.

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

A principios de 1959, Frank Rosenblatt creó su simple clasificador espontáneo, una red de aprendizaje no supervisado basado en el Perceptrón, el cual aprendía a clasificar vectores de entrada en dos clases con igual número de términos.

A finales de los años 60's y principios de los 70's, Stephen Grossberg introdujo muchas redes competitivas que usaban inhibición lateral obteniendo buenos resultados. Algunos de los comportamientos útiles obtenidos por él, fueron la supresión del ruido, aumento del contraste y normalización de vectores.

En 1973, Christoph Von Der Malsburg introduce la regla del mapa de organización propia, que permitía a la red clasificar entradas en las cuales las neuronas que estuviesen en un vecindario cercano a la neurona ganadora, respondieran a entradas similares. La topología de esta red imitaba de alguna forma las estructuras encontradas en la corteza visual de los gatos, estudiada por David Hubel y Torten Wiesel. Su regla de aprendizaje generó gran interés,

pero esta utilizaba un cálculo no local para garantizar que los pesos fueran normalizados, este hecho hacía este modelo biológicamente poco posible.

Grossberg extendió el trabajo de Von Der Malsburg, redescubriendo la regla Instar. Grossberg mostró que la regla Instar removió la necesidad de renormalizar los pesos, porque los vectores de pesos que aprendían a reconocer vectores de entrada normalizados, automáticamente se normalizarán ellos mismos.

El trabajo de Grossberg y Von Der Malsburg enfatizó la posibilidad biológica de sus redes. Otro exitoso investigador, Tuevo Kohonen ha sido también un fuerte proponente de las redes competitivas; sin embargo, su énfasis ha sido en aplicaciones para ingeniería y en descripciones de eficiencia matemática de las redes. Durante la década de los 70 Kohonen desarrolló una versión simplificada de la regla Instar, inspirada también en la red de Von Der Malsburg y Grossberg, de esta forma encontró una manera muy eficiente de incorporar topología a una red competitiva.

Otra forma de aplicar este tipo de aprendizaje fue propuesta por Rumelhart y Zisper [32] en 1985, quienes utilizaban redes multicapa dividiendo cada capa en grupos de neuronas, de tal forma que éstas disponían de conexiones inhibitorias con otras neuronas de su mismo grupo y conexiones excitadoras con las neuronas de la siguiente capa; en una red de este tipo, después de recibir diferentes informaciones de entrada, cada neurona en cada grupo se especializa en la respuesta a determinadas características de los datos de entrada.

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

Una variación del aprendizaje supervisado aplicado a redes multicapa consiste en imponer una inhibición mutua entre neuronas únicamente cuando están a cierta distancia unas de otras (suponiendo que las neuronas se han dispuesto geométricamente, por ejemplo formando capas bidimendisionales), existe entonces un área o región de vecindad alrededor de las neuronas que constituye un grupo local.

Fukushima [11] empleó esta idea en 1975 para una red multicapa llamada Cognitron, fuertemente inspirada en la anatomía y fisiología del sistema visual humano y en 1980 el mismo Fukushima [12] en una versión mejorada de la anterior a la que llamó Necognitron, presentó una variación de esta red

utilizando aprendizaje supervisado. El Necognitrron disponía de un gran número de capas con arquitectura muy específica de interconexiones entre ellas y era capaz de aprender a diferenciar caracteres, aunque estos se presentasen a diferente escala, en diferente posición o distorsionados.

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

Red de Kohonen: Existen evidencias que demuestran que en el cerebro hay neuronas que se organizan en muchas zonas, de forma que las informaciones captadas del entorno a través de los órganos sensoriales se representan internamente en forma de mapas bidimensionales. Por ejemplo, en el sistema visual se han detectado mapas del espacio visual en zonas del córtex (capa externa del cerebro), también en el sistema auditivo se detecta una organización según la frecuencia a la que cada neurona alcanza mayor repuesta (organización tonotópica).

Aunque en gran medida esta organización neuronal está predeterminada genéticamente, es probable que parte de ella se origine mediante el aprendizaje, esto sugiere que el cerebro podría poseer la capacidad inherente de formar mapas topológicos de las informaciones recibidas del exterior, de hecho esta teoría podría explicar su poder de operar con elementos semánticos: algunas áreas del cerebro simplemente podrían crear y ordenar neuronas especializadas o grupos con características de alto nivel y sus combinaciones, en definitiva se construirían mapas especiales para atributos y características.

A partir de estas ideas Tuevo Kohonen [24] presentó en 1982 un sistema con un comportamiento semejante, se trataba de un modelo de red neuronal con capacidad para formar mapas de características de manera similar a como ocurre en el cerebro; el objetivo de Kohonen era demostrar que un estímulo externo (información de entrada) por sí solo, suponiendo una estructura propia y una descripción funcional del comportamiento de la red, era suficiente para forzar la formación de los mapas.

Este modelo tiene dos variantes denominadas LVQ (Learning Vector Quantization) y TPM (Topology Preserving Map) o SOM (Self Organizing Map), ambas se basan en el principio de formación de mapas topológicos para establecer características comunes entre las informaciones (vectores) de entrada a la red, aunque difieren en las dimensiones de éstos, siendo de una

sola dimensión en el caso de LVQ y bidimensional o tridimensional en la red SOM. Estas redes se tratarán con mayor profundidad en secciones posteriores.

El aprendizaje en el modelo de Kohonen es de tipo Off-line, por lo que se distingue una etapa de aprendizaje y otra de funcionamiento. En la etapa de aprendizaje se fijan los valores de las conexiones (feedforward) entre la capa de entrada y la salida. Esta red utiliza un aprendizaje no supervisado de tipo competitivo, las neuronas de la capa de salida compiten por activarse y sólo una de ellas permanece activa ante una determinada información de entrada a la red, los pesos de las conexiones se ajustan en función de la neurona que haya resultado vencedora.

Durante la etapa de entrenamiento, se presenta a la red un conjunto de informaciones de entrada (vectores de entrenamiento) para que ésta establezca en función de la semejanza entre los datos las diferentes categorías (una por neurona de salida), que servirían durante la fase de funcionamiento para realizar clasificaciones de nuevos datos que se presenten a la red. Los valores finales de los pesos de las conexiones entre cada neurona de la capa de salida con las de entrada se corresponderán con los valores de los componentes del vector de aprendizaje que consigue activar la neurona correspondiente. En el caso de existir más patrones de entrenamiento que neuronas de salida, más de uno deberá asociarse con la misma neurona, es decir pertenecerán a la misma clase.

En este modelo el aprendizaje no concluye después de presentarle una vez todos los patrones de entrada, sino que habrá que repetir el proceso varías veces para refinar el mapa topológico de salida, de tal forma que cuantas más veces se presenten los datos, tanto más se reducirán las zonas de neuronas que se deben activar ante entradas parecidas, consiguiendo que la red pueda realizar una clasificación mas selectiva.

Un concepto muy importante en la red de Kohonen es la zona de vecindad, o vecindario alrededor de la neurona vencedora i^*, los pesos de las neuronas que se encuentren en esta zona a la que se le dará el nombre de $\boldsymbol{X}(q)$, serán actualizados junto con el peso de la neurona ganadora, en un ejemplo de aprendizaje cooperativo.

El algoritmo de aprendizaje utilizado para establecer los valores de los pesos de las conexiones entre las N neuronas de entrada y las M de salida es el siguiente:

1. En primer lugar se inicializan los pesos (w_{ij}) con valores aleatorios pequeños y se fija la zona inicial de vecindad entre las neuronas de salida.

2. A continuación se presenta a la red una información de entrada (la que debe aprender) en forma de vector $\boldsymbol{p} = (p_1, p_2, \ldots, p_n)$, cuyas componentes p_i serán valores continuos.

3. Puesto que se trata de un aprendizaje competitivo, se determina la neurona vencedora de la capa de salida, esta será aquella i cuyo vector de pesos $\boldsymbol{w}_i$ (vector cuyas

componentes son los valores de los pesos de las conexiones entre esa neurona y cada una de las neuronas de la capa de entrada) sea el más parecido a la información de entrada ***p*** (patrón o vector de entrada). Para ello se calculan las distancias o diferencias entre ambos vectores, considerando una por una todas las neuronas de salida, suele utilizarse la distancia euclídea o la siguiente expresión que es similar a aquella, pero eliminando la raíz cuadrada:

$$d_i = \sum_{j=1}^{N} (p_j - w_{ij})^2 \qquad 1 \le i \le M \tag{2.5.1}$$

p_j: Componente *i*-ésimo del vector de entrada
w_{ij}: Peso de la conexión entre la neurona *j* de la capa de entrada y la neurona *i* de la capa de salida.

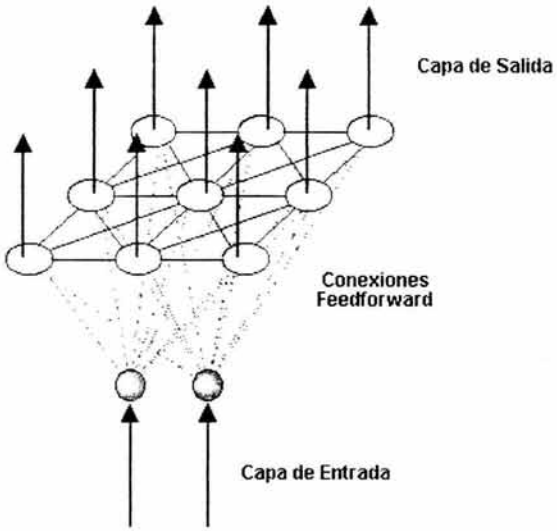


Figura 2.5.1 Conexiones de una red de Kohonen

4. Una vez localizada la neurona vencedora (*i**), se actualizan los pesos de las conexiones entre las neuronas de entrada y dicha neurona, así como los de las conexiones entre las de entrada y las neuronas vecinas de la vencedora, en realidad lo que se consigue con esto es asociar la información de entrada con una cierta zona de la capa de salida. Esto se realiza mediante la siguiente ecuación

$$w(q) = w(q-1) + a(q)(p(q) - w(q-1)) \text{ para } i \in \boldsymbol{X}(q) \tag{2.5.2}$$

El tamaño de $\boldsymbol{X}(q)$ se puede reducir en cada iteración del proceso de ajuste de los pesos, con lo que el conjunto de neuronas que pueden considerarse vecinas cada vez es menor como se observa en la figura 2.5.2, sin embargo en la práctica es habitual considerar una zona fija en todo el proceso de entrenamiento de la red.

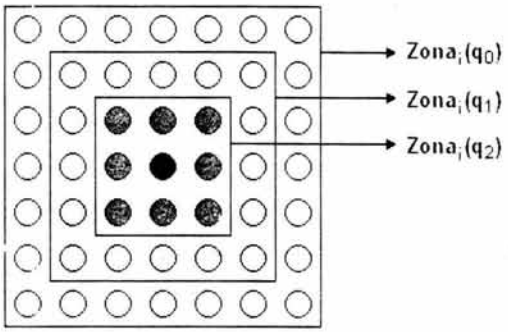


Figura 2.5.2 Posible evolución de la vecindad en una red de Kohonen

El término $\alpha(q)$ es el coeficiente de aprendizaje o parámetro de ganancia, con un valor entre 0 y 1 el cual decrece con el número de iteraciones (q) del proceso de entrenamiento, de tal forma que cuando se ha presentado un gran número de veces todo el juego de patrones de aprendizaje su valor es prácticamente nulo, con lo que la modificación de los pesos es insignificante.

Para hallar α suele utilizarse una de las siguientes expresiones [20]:

$$\alpha(q) = \frac{1}{q} \qquad \alpha(q) = \alpha_1\left(1 - \frac{q}{\alpha_2}\right) \quad (2.5.3)$$

Siendo α_1 un valor de 0.1 ó 0.2 y α_2 un valor próximo al número total de iteraciones del aprendizaje, que por lo general se toma como 10000 para esta red.

5. El proceso debe repetirse, volviendo a presentar todo el juego de patrones de aprendizaje $p_1, p_2..,p_n$ hasta obtener la salida deseada.

Como la regla Instar, la regla de Kohonen habilita a los pesos de una neurona a aprender un vector de entrada y de esta forma resolver aplicaciones de reconocimiento de patrones. A diferencia de la regla Instar, el aprendizaje no es proporcional a la salida de la neurona $a_i(q)$, en lugar de ello el aprendizaje ocurre cuando la neurona i sea miembro del conjunto $\boldsymbol{X}(q)$, si la regla Instar es aplicada a una capa de neuronas cuya función de transferencia solamente retorna valores de 0 o 1 (por ejemplo hardlim), la regla de Kohonen es equivalente a la regla Instar.

En definitiva lo que hace una red de Kohonen es realizar una tarea de clasificación, puesto que la neurona de salida activada ante una entrada representa la clase a la que pertenece dicha información de entrada, además ante otra entrada parecida se activa la misma neurona de salida, u otra cercana a la anterior debido a la semejanza entre las clases, así se garantiza que las

neuronas topológicamente próximas sean sensibles a entradas físicamente similares; por esta causa la red es especialmente útil para establecer relaciones desconocidas previamente entre conjuntos de datos.

Red de Hamming: La red de Hamming ilustrada en la figura 2.5.3 es uno de los ejemplo más simples de aprendizaje competitivo, a pesar de ello su estructura es un poco compleja ya que emplea el concepto de capas recurrentes en su segunda capa y aunque hoy en día en redes de aprendizaje competitivo se ha simplificado este concepto con el uso de funciones de activación más sencillas, la red de Hamming representa uno de los primeros avances en este tipo de aprendizaje, convirtiéndola en un modelo obligado de referencia dentro de las redes de aprendizaje competitivo. Las neuronas en la capa de salida de esta red compiten unas con otras para determinar la ganadora, la cual indica el patrón prototipo más representativo en la entrada de la red, la competición es implementada por inhibición lateral (un conjunto de conexiones negativas entre las neuronas en la capa de salida).

Esta red consiste en dos capas; la primera capa, la cual es una red Instar, realiza la correlación entre el vector de entrada y los vectores prototipo, la segunda capa realiza la competición para determinar cual de los vectores prototipo está más cercano al vector de entrada.

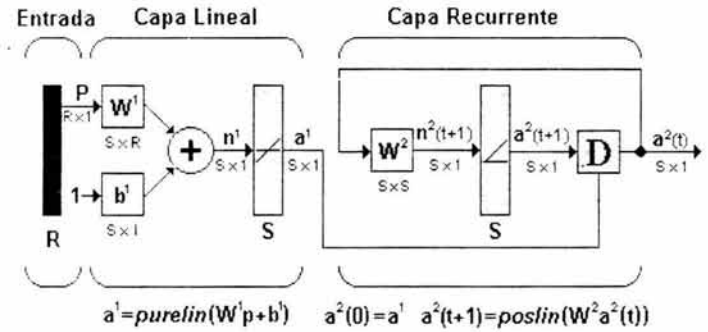


Figura 2.5.3 Red de Hamming

Capa 1:

La red Instar es capaz de clasificar solo un patrón; para que múltiples patrones sean reconocidos se necesitan múltiples Instar y es precisamente de esa forma como está compuesta la primera capa de la red de Hamming. Para una mejor comprensión de su funcionamiento se partirá de unos vectores prototipo que la red debe clasificar

$\{p_1, p_2, ..., p_Q\}$ (2.5.4)

La matriz de pesos $\boldsymbol{W}^1$ y el vector de ganancias $\boldsymbol{b}^1$ para la capa uno serán:

$$W^1 = \begin{bmatrix} {}_1W^T \\ {}_2W^T \\ \vdots \\ {}_SW^T \end{bmatrix} = \begin{bmatrix} p_1^T \\ p_2^T \\ \vdots \\ p_Q^T \end{bmatrix}, \quad b^1 = \begin{bmatrix} R \\ R \\ R \\ R \end{bmatrix} \quad (2.5.5)$$

Donde cada fila de W^1 representa un vector prototipo, el cual deseamos reconocer y cada elemento b^1 es igual al número de elementos en cada vector de entrada (R) (el número de neuronas S es igual al número de vectores prototipo Q). Así la salida de la primera capa será:

$$a^1 = W^1 p + b^1 = \begin{bmatrix} p_1^T p + R \\ p_2^T p + R \\ \vdots \\ p_Q^T 1 + R \end{bmatrix} \quad (2.5.6)$$

La salida de la capa 1 es igual al producto punto de los vectores prototipo con la entrada más el vector R; este producto indica cuan cercano está cada vector de entrada a los patrones prototipo.

Capa 2:

La red Instar emplea una función de transferencia *poslin* para decidir si el vector de entrada estaba lo suficientemente cerca al vector prototipo. En la capa 2 de la red de Hamming se utilizan múltiples Instar, así se determinara por medio de una capa competitiva el patrón prototipo más cercano. Las neuronas en esta capa son inicializadas con la salida de la capa en realimentación, la cual indica la correlación entre los patrones prototipo y el vector de entrada. Las neuronas compiten unas con otras para determinar una ganadora; después de la competición solo una neurona tendrá una salida no cero. La neurona ganadora indica cual categoría de entrada fue presentada a la red (cada vector prototipo representa una categoría).

La salida de la primera capa a^1 es usada para inicializar la segunda capa:

$a^2(0) = a^1$ (2.5.7)

La salida de la segunda capa está determinada de acuerdo a la siguiente relación recurrente:

$a^2(t+1) = poslin\ (W^2 a^2(t))$ (2.5.8)

Los pesos de la segunda capa W^2 son fijados de tal forma que los elementos de la diagonal sean 1, y los elementos por fuera de la diagonal tengan pequeños valores negativos.

$$w_{ij}^2 = \begin{Bmatrix} 1, si \quad i = j \\ -\varepsilon \, de \, otra \, forma \end{Bmatrix} \quad Donde \; 0 < \varepsilon < \frac{1}{s-1} \quad (2.5.9)$$

Esta matriz produce un efecto inhibitorio, en el cual la salida de cada neurona tiene un efecto inhibitorio sobre todas las otras neuronas. Para ilustrar este efecto, sustituimos los valores de peros de 1 y $-\varepsilon$ por los apropiados elementos de W^2. Reescribiendo la ecuación de salida de la red para una sola neurona se tiene:

$$a^2_i(t+1) = poslin\left(a_i^2(t) - \varepsilon \sum_{j \neq i} a_j^2(t)\right) \quad (2.5.10)$$

En cada iteración, cada salida de la neurona se decrementará en proporción a la suma de las salidas de las otras neuronas. La salida de la neurona con la condición inicial más grande se decrementará más despacio que las salidas de otras neuronas; eventualmente cada neurona tendrá una salida positiva y en ese punto la red habrá alcanzado el estado estable.

Estructura general de una red competitiva: En las redes asociativas, se vio como la regla instar puede aprender a responder a un cierto grupo de vectores de entrada que están concentrados en una región del espacio. Supóngase que se tienen varias instar agrupadas en una capa, tal como se muestra en la figura 2.5.4, cada una de las cuales responde en forma máxima a un cierto grupo de vectores de entrada de una región distinta del espacio.

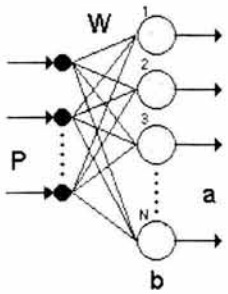


Figura 2.5.4 Instar agrupadas en una capa

Se puede decir que esta capa de Instars clasifica cualquier vector de entrada, porque la Instar con la mayor respuesta para alguna entrada dada es la que identifica a la región del espacio en la cual yace el vector de entrada. En lugar de examinar la respuesta de cada instar para determinar cual es la mayor, la labor de clasificación sería más fácil si la Instar de mayor respuesta fuera la única unidad que tuviese una salida no nula; esto se puede conseguir si las instar compiten unas con otras por el privilegio de la activación, este es el principio de las redes competitivas.

Las neuronas de la segunda capa de la red de Hamming, están en competición porque cada neurona se excita a sí misma e inhibe a todas las otras neuronas, para simplificar la discusión se definirá una nueva función de transferencia que hace el trabajo de una capa recurrente competitiva

***a* = *compet* (*n*)** (2.5.11)

Donde ***a*** es la salida total de la red y ***n*** es la entrada neta a la función de transferencia, *compet* es una función de transferencia que encuentra el índice i^* de la neurona con la entrada neta más grande y fija su salida en uno, todas las otras neuronas tienen salida 0.

$$a_i = \begin{cases} 1, & i = i^* \\ 0, & i \neq i^* \end{cases} \quad Donde\ n_{i^*} \geq n_i, \forall i, \qquad i^* \leq i,\ \forall n_i = n_{i^*} \tag{2.5.12}$$

Reemplazando la capa recurrente de la red de Hamming, con una función de transferencia competitiva, la presentación de una capa competitiva se simplifica de la siguiente manera.

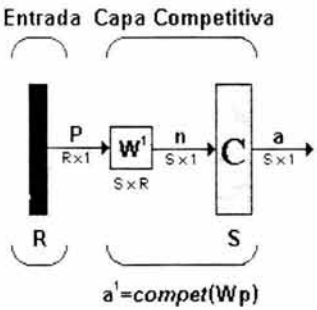


Figura 2.5.5 Capa Competitiva

Como con la red de Hamming, los vectores prototipo son almacenados en las filas de ***W***. La entrada neta ***n*** calcula la distancia entre el vector de entrada ***p*** y cada prototipo $\boldsymbol{w}_i$ (asumiendo que los vectores tiene longitudes normalizadas ***L***).

La entrada neta n_i de cada neurona es proporcional al ángulo θ_i entre ***p*** y el vector prototipo $\boldsymbol{w}_i$.

$$\boldsymbol{n} = \boldsymbol{W}\boldsymbol{p} = \begin{bmatrix} {}_1\boldsymbol{w}^T \\ {}_2\boldsymbol{w}^T \\ {}_S\boldsymbol{w}^T \end{bmatrix} \boldsymbol{p} = \begin{bmatrix} {}_1\boldsymbol{w}^T\boldsymbol{p} \\ {}_2\boldsymbol{w}^T\boldsymbol{p} \\ {}_S\boldsymbol{w}^T\boldsymbol{p} \end{bmatrix} = \begin{bmatrix} \boldsymbol{L}^2\boldsymbol{\cos}\ {}_1 \\ \boldsymbol{L}^2\boldsymbol{\cos}\ {}_2 \\ \boldsymbol{L}^2\boldsymbol{\cos}\ {}_S \end{bmatrix} \tag{2.5.13}$$

La función de transferencia competitiva asigna una salida de 1 a la neurona cuyo vector de pesos apunte en la dirección más cercana al vector de entrada

$a = compet(n)$ (2.5.14)

Regla de aprendizaje: En este punto es posible diseñar una red competitiva que realice clasificaciones correctas fijando el valor de las filas de $\boldsymbol{W}$ en los valores del vector prototipo esperado, sin embargo es deseable tener una regla de aprendizaje que pueda entrenar los pesos en una red competitiva sin conocer los vectores prototipo, una de estas reglas es la Instar estudiada es el numeral 2.4.3

$${}_1w(q) = {}_1w(q\text{-}1) + a(q)(p(q) - {}_1w(q\text{-}1)) \quad (2.5.15)$$

Para redes competitivas, a tiene un valor diferente de cero solamente para la neurona ganadora ($i=i^*$), de esta forma los mismos resultados serán obtenidos utilizando la regla de Kohonen

$${}_i\boldsymbol{w}(q) = {}_i\boldsymbol{w}(q\text{-}1) + \alpha\,(\boldsymbol{p}(q) - {}_i\boldsymbol{w}(q\text{-}1)) = (1-\alpha)\,{}_i\boldsymbol{w}(q\text{-}1) + \alpha\,\boldsymbol{p}(q) \quad (2.5.16)$$

y

$$w(q) = w(q\text{-}1) \quad i \neq i^* \quad (2.5.17)$$

Así, la fila de la matriz de pesos que este más cerca al vector de entrada (o tenga el producto punto más grande con el vector de entrada) se moverá hacia el vector de entrada. Este se mueve a lo largo de la línea entre la fila anterior del vector de pesos y el vector de entrada, como puede verse en la figura 2.5.6

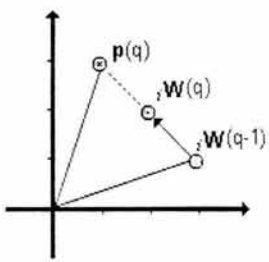


Figura 2.5.6 Representación gráfica de la regla de Kohonen

Para demostrar como trabaja una red competitiva, se creará una red que clasifique los siguientes vectores:

$$p1 = \begin{bmatrix} -0.216 \\ 0.993 \end{bmatrix},\ p2 = \begin{bmatrix} 0.216 \\ 0.993 \end{bmatrix},\ p3 = \begin{bmatrix} 0.993 \\ 0.216 \end{bmatrix}$$

$$p4 = \begin{bmatrix} 0.993 \\ -0.216 \end{bmatrix},\ p5 = \begin{bmatrix} -0.622 \\ -0.873 \end{bmatrix},\ p6 = \begin{bmatrix} -0.873 \\ -0.622 \end{bmatrix}$$

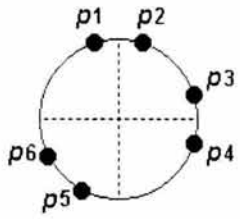


Figura 2.5.7 Vectores de entrada

La red tendrá tres neuronas, por lo tanto los vectores serán clasificados en tres clases o grupos, esta es una de las principales características de las redes competitivas, ellas pueden agrupar los patrones de entrada en clases que no se conocen. Los pesos normalizados escogidos aleatoriamente son:

$$ {}_1w = \begin{bmatrix} 0.7071 \\ -0.7071 \end{bmatrix},\ {}_2w = \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix},\ {}_3w = \begin{bmatrix} -1.000 \\ 0.000 \end{bmatrix},\ \boldsymbol{W} = \begin{bmatrix} {}_1\boldsymbol{w}^T \\ {}_2\boldsymbol{w}^T \\ {}_3\boldsymbol{w}^T \end{bmatrix} $$

Los vectores de datos y los pesos asignados pueden visualizarse en la figura 2.5.8

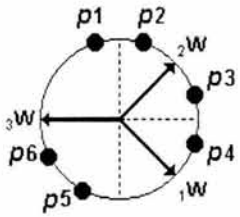


Figura 2.5.8 Vectores de entrada y vector de pesos

Se presenta a la red el vector $p2$

$$ \boldsymbol{a} = compet(\boldsymbol{Wp2}) = compet\left(\begin{bmatrix} 0.7071 & -0.7071 \\ 0.7071 & 0.7071 \\ -1.000 & 0.000 \end{bmatrix}\begin{bmatrix} 0.216 \\ 0.993 \end{bmatrix}\right) $$

$$ \boldsymbol{a} = compet\left(\begin{bmatrix} -0.5494 \\ 0.8549 \\ -0.2160 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} $$

El vector de peso de la segunda neurona estaba más cercano a $p2$, por lo tanto ganó la competición (i=2) y su salida es 1. Ahora se aplicará la regla de Kohonen a la neurona ganadora con una rata de aprendizaje $\alpha = 0.5$

$$ {}_2w^{nuevo} = {}_2w^{anterior} + \alpha\,(p2 - {}_2w^{anterior}) $$

$$ {}_2w^{nuevo} = \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix} + 0.5\left(\begin{bmatrix} 0.216 \\ 0.993 \end{bmatrix} - \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix} \right) = \begin{bmatrix} 0.9527 \\ 0.5641 \end{bmatrix} $$

La regla de Kohonen hace que ${}_2w$ tienda hacia $p2$ como puede verse en la figura 2.5.9, si continuamos escogiendo vectores de entrada aleatoriamente y presentándoselos a la red, en cada iteración el vector de pesos se acercará más al vector de entrada.

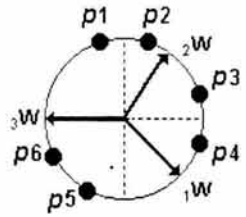


Figura 2.5.9 Proceso de entrenamiento

Cada vector de pesos apuntará hacia una clase diferente del vector de entrada, convirtiéndose en un prototipo para esa clase. Cuando termine el proceso de entrenamiento, los pesos finales se verán como aparece en la figura 2.5.10

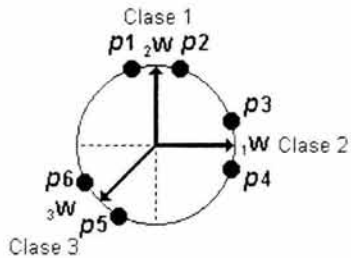


Figura 2.5.10 Pesos Finales

Problemas de las redes Competitivas: Las redes competitivas, son bastante eficientes para resolver problemas de clasificación, sin embargo presentan algunos problemas. El primero es la elección de una rata de aprendizaje que permita hallar un punto de equilibrio entre velocidad de convergencia y la estabilidad final de los vectores de peso. Una rata de aprendizaje cercana a cero, torna el aprendizaje muy lento pero garantiza que cuando un vector haya alcanzado el centro de la clase objetivo, se mantendrá allí indefinidamente. En contraste, una rata de aprendizaje cercana a uno genera un aprendizaje muy

rápido, pero los vectores de peso continuarán oscilando aún después de que se haya alcanzado convergencia. La indecisión que se presenta al escoger la rata de aprendizaje puede ser empleada como una ventaja si se inicia el entrenamiento con una rata de aprendizaje alta y se decrementa en el transcurso del proceso de entrenamiento cuando sea necesario, desafortunadamente esta técnica no funciona si la red necesita continuamente ser adaptada a nuevos argumentos de los vectores de entrada (caso en que la red se trabaje On-line). Un ejemplo de este problema se visualiza en la figura 2.5.11

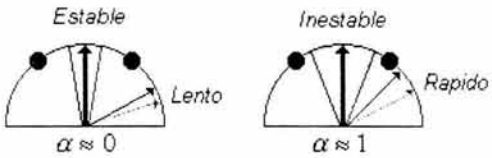


Figura 2.5.11 Variación de la rata de aprendizaje

Un problema de estabilidad más serio, ocurre cuando las clases están muy juntas; en ciertos casos, un vector de pesos tratando de apuntar hacia una clase determinada, puede entrar al territorio de otro vector de pesos. En la figura 2.5.12, pueden observarse con círculos azules, como dos vectores de entrada son presentados repetidas veces a la red; el resultado, es que los vectores de pesos que representan las clases de la mitad y de la derecha se encuentran a la derecha. Con seguridad, se presentará el caso en que una de las clases de la derecha será clasificada por el vector de pesos del centro

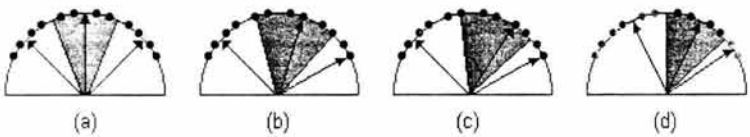


Figura 2.5.12 Aprendizaje Inestable

Un tercer problema con redes competitivas, es que es posible que el vector de pesos inicial de una neurona se encuentre muy lejos de cualquiera de los vectores de entrada y por lo tanto nunca gane la competición. La consecuencia será, la "muerte" de la neurona, lo que por supuesto no es recomendable.

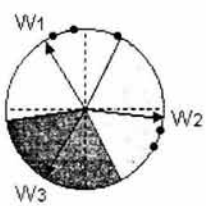


Figura 2.5.13 Causa de la muerte de una neurona

En la figura 2.5.13 el vector de peso w_3 nunca ganará la competición, sin importar cual sea el orden en que se le presenten los vectores de entrada. Una solución a este problema, consiste en adicionar una ganancia negativa a la entrada neta de cada neurona y decrementar así la ganancia total cada vez que la neurona gane la competición; esto hará que difícilmente una neurona gane varias veces la competición, a este mecanismo se le llama "conciencia".

Una capa competitiva tiene tantas clases como neuronas, lo que podría complicar algunas aplicaciones, especialmente cuando el número de clases no se conoce de antemano. En capas competitivas, cada clase consiste de una región convexa del espacio de entrada, las capas competitivas no pueden formar clases con regiones no convexas o clases que sean la unión de regiones no conectadas.

Mapas de auto organización (SOM): Se cree que algunos sistemas biológicos realizan sus operaciones siguiendo un método de trabajo que algunos investigadores han llamado, *on-center/off-surround*; este término describe un patrón de conexión entre neuronas, cada neurona se refuerza a ella misma (center) mientras inhibe a todas las neuronas a su alrededor (surround). En las redes competitivas biológicas, lo que sucede realmente es que cuando una neurona se refuerza a ella misma, refuerza también las neuronas que están cerca; la transición entre reforzar las neuronas "vecinas" o inhibirlas, se realiza suavemente a medida que la distancia entre las neuronas aumenta. De esta forma el proceso *on-center/off-surround*; para redes biológicas sigue el comportamiento señalado en la figura 2.5.14, función que habitualmente es referida como sombrero mejicano debido a su forma.

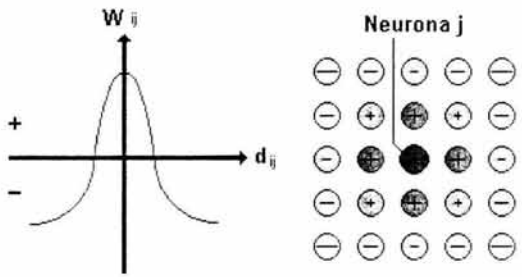


Figura 2.5.14 *on-center/off-surround*; para capas biológicas

Tratando de emular la actividad biológica, sin tener que implementar conexiones on-center/off-surround; de realimentación no lineal, Kohonen diseñó la red conocida como mapa de auto organización (SOM). Esta red determina primero la neurona ganadora i^* usando el mismo procedimiento que las redes competitivas, luego los vectores de pesos de todas las neuronas que se encuentren en una región cercana "vecindario", serán actualizados mediante la regla de Kohonen

$_i\boldsymbol{w}(q) = {}_i\boldsymbol{w}(q\text{-}1) + \alpha\,(\boldsymbol{p}(q) - {}_i\boldsymbol{w}(q\text{-}1))$ *para* $i \in \boldsymbol{N}_{i^*}(d)$ (2.5.18)

Donde el vecindario N_i^* contiene el índice para todas las neuronas que se encuentren a un radio "*d*" de la neurona ganadora i^*

$$N_{i^*}(d) = \{j, d_{ij} \le d\} \quad (2.5.19)$$

Cuando un vector ***p*** es presentado, los pesos de la neurona ganadora y de sus vecinas tenderán hacia ***p***, el resultado es que después de muchas presentaciones las neuronas vecinas habrán aprendido vectores similares que cada una de las otras.

El concepto de vecindario es ilustrado en la figura 2.5.15; para la primera figura se ha tomado un vecindario de radio d =1 alrededor de la neurona 13; para la segunda figura se ha tomado un vecindario de radio d =2.

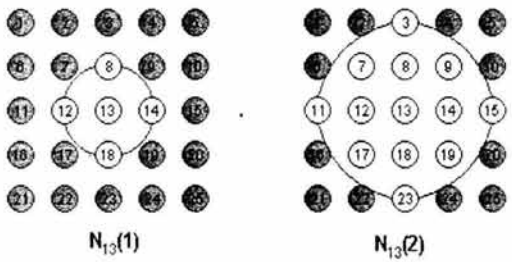


Figura 2.5.15 Vecindarios

Estos vecindarios pueden definirse como sigue:

$$\boldsymbol{N}_{13}(1) = \{8,12,13,14,18\} \quad (2.5.20)$$

$$\boldsymbol{N}_{13}(2) = \{3,7,8,9,11,12,13,14,15,17,18,19,23\}$$

El vecindario puede determinarse en diferentes formas; Kohonen, por ejemplo ha sugerido vecindarios rectangulares o hexagonales para lograr alta eficiencia; es importante destacar que el rendimiento de la red no es realmente sensitivo a la forma exacta del vecindario.
La figura 2.5.16 ilustra un mapa de auto organización de dos dimensiones

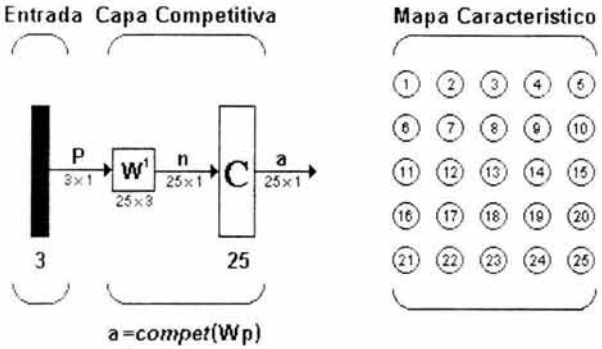


Figura 2.5.16 Mapa de auto organización

Learning Vector Quantization (LVQ): Esta red es un híbrido que emplea tanto aprendizaje no supervisado, como aprendizaje supervisado para clasificación de patrones

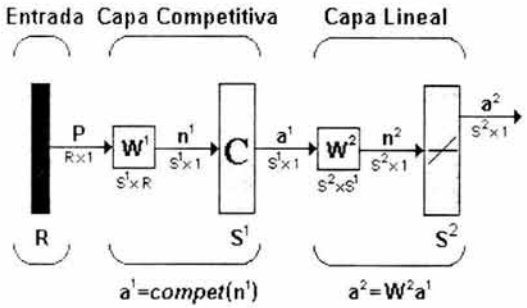


Figura 2.5.17 Red LVQ

En la red LVQ, cada neurona de la primera capa es asignada a una clase, después cada clase es asignada a una neurona en la segunda capa. El número de neuronas en la primera capa, S^1 debe ser mayor o al menos igual que el número de neuronas en la segunda capa, S^2.

Al igual que con redes competitivas, cada neurona en la primera capa de la red LVQ aprende un vector prototipo, el cual permite a la neurona clasificar una región del espacio de entrada, sin embargo en lugar de calcular la distancia entre la entrada y el vector de pesos por medio del producto punto, la red LVQ calcula la distancia directamente. Una ventaja de hacer el cálculo de la distancia directamente, es que los vectores no necesitan ser normalizados, cuando los vectores son normalizados la respuesta de la red será la misma sin importar la técnica que se utilice.

La entrada neta a la primera capa de la red LVQ es entonces,

$$n_i^1 = -\begin{bmatrix} \|{}_1\mathbf{w}^1 - \mathbf{p}\| \\ \|{}_2\mathbf{w}^1 - \mathbf{p}\| \\ \vdots \\ \|{}_{S^1}\mathbf{w}^1 - \mathbf{p}\| \end{bmatrix} \quad (2.5.21)$$

La salida de la primera capa de la red LVQ es,

$\boldsymbol{a^1 = compet\ (n^1)}$ (2.5.22)

Así, la neurona cuyo vector de pesos este cercano al vector de entrada tendrá salida 1 y las otras neuronas, tendrán salida 0; en este aspecto la red LVQ se comporta igual a las redes competitivas, la única diferencia consiste en la interpretación, mientras que en las redes competitivas la salida no cero representa una *clase* del vector de entrada, para el algoritmo LVQ, indica mas bien una *sub- clase*, y de esta forma muchas neuronas (subclases), conforman una clase.

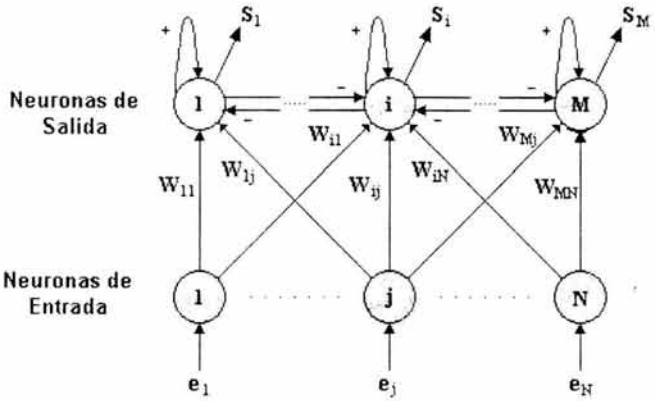


Figura 2.5.18 Comportamiento de las neuronas en una red LVQ

La segunda capa de la red LVQ es usada para combinar subclases dentro de una sola clase, esto es realizado por la matriz de pesos $\boldsymbol{W^2}$ Las columnas de $\boldsymbol{W^2}$ representan las subclases y las filas representan las clases, $\boldsymbol{W^2}$ tiene un solo 1 en cada columna, todos los demás elementos son cero, la fila en la cual se presenta el 1 indica cual es la clase a la que la subclase pertenece.

$W^2{}_{ki} = 1 \Rightarrow$ *la subclase i pertenece a la clase k* (2.5.23)

Una propiedad importante de esta red, es que el proceso de combinar subclases para formar clases, permite a la red LVQ crear clases más complejas. Una capa competitiva estándar tiene la limitación de que puede crear solo regiones de decisión convexas; la red LVQ soluciona esta limitación.

La red LVQ combina aprendizaje competitivo con aprendizaje supervisado, razón por lo cual necesita un set de entrenamiento que describa el comportamiento propio de la red

$\{p_1, t_1\}, \{p_2, t_2\}, \ldots, \{p_Q, t_Q\}$ (2.5.24)

Para ilustrar el desempeño de la red LVQ, se considerará la clasificación de un vector particular de tres elementos dentro de otro de cuatro clases, de esta forma:

$$\left\{ \boldsymbol{p}_1 = \begin{bmatrix} \sqrt{0.74} \\ 0 \\ \sqrt{0.74} \end{bmatrix}, \quad \boldsymbol{t}_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \right\} \quad (2.5.25)$$

Antes de que suceda el aprendizaje, cada neurona en la segunda capa es asignada a una neurona de salida, así se genera la matriz $\boldsymbol{W}^2$; por lo general, igual número de neuronas ocultas son conectadas a cada neurona de salida, para que cada clase pueda ser conformada por el mismo número de regiones convexas. Todos los elementos de $\boldsymbol{W}^2$ son cero excepto los que cumplan la siguiente condición:

Si la neurona i es asignada a la clase k $\Rightarrow w^2_{ki}=1$ (2.5.26)

Una vez $\boldsymbol{W}^2$ ha sido definida, nunca será alterada. Los pesos ocultos $\boldsymbol{W}^1$ son actualizados por medio de la regla de Kohonen.

La regla de aprendizaje del algoritmo LVQ, trabaja de la siguiente manera:

1. En cada iteración, un vector de entrada $\boldsymbol{p}$ es presentado a la red y se calcula la distancia a cada vector prototipo.
2. Las neuronas ocultas compiten, la neurona i^* gana la competición y el i^*-ésimo elemento de $\boldsymbol{a}^1$ se fija en 1
3. $\boldsymbol{a}^1$ es multiplicada por $\boldsymbol{W}^2$ para obtener la salida final $\boldsymbol{a}^2$, la cual tiene solamente un elemento no cero, k^*, indicando que el patrón $\boldsymbol{p}$ está siendo asignado a la clase k^*

La regla de Kohonen es empleada para mejorar la capa oculta de la red LVQ, en dos formas:
Primero, si $\boldsymbol{p}$ es clasificado correctamente los pesos de la neurona ganadora ${}_{i^*}\boldsymbol{w}^1$ se hacen tender hacia $\boldsymbol{p}$.

${}_{i^*}\boldsymbol{w}(q) = {}_{i^*}\boldsymbol{w}(q-1) - a(q)\,(\boldsymbol{p}\,(q) - {}_{i^*}\boldsymbol{w}(q-1))$ *si* $a^2_{k^*} = t_{k^*} = 1$ (2.5.27)

Segundo, si $\boldsymbol{p}$ es clasificado incorrectamente una neurona equivocada ganó la competición y por lo tanto sus pesos ${}_{i^*}\boldsymbol{w}^1$ se alejan de $\boldsymbol{p}$.

${}_{i^*}\boldsymbol{w}(q) = {}_{i^*}\boldsymbol{w}(q-1) - a(q)\,(\boldsymbol{p}\,(q) - {}_{i^*}\boldsymbol{w}(q-1))$ *si* $a^2_{k^*} = 1 \neq t_{k^*} = 0$ (2.5.28)

El resultado será que cada neurona se moverá hacia los vectores que cayeron dentro de la clase, para la cual ellos forman una subclase y lejos de los vectores que cayeron en otras clases.

Se ilustrará el funcionamiento de la red LVQ, buscando que clasifique correctamente los siguientes patrones, cuyas clases se han definido arbitrariamente:

$$clase\ 1:\ \left\{\boldsymbol{p}_1=\begin{bmatrix}-1\\-1\end{bmatrix},\ \boldsymbol{p}_2=\begin{bmatrix}1\\1\end{bmatrix}\right\},\ clase\ 2:\ \left\{\boldsymbol{p}_3=\begin{bmatrix}1\\-1\end{bmatrix},\ \boldsymbol{p}_4=\begin{bmatrix}-1\\1\end{bmatrix}\right\}$$

Los vectores esperados asociados a cada una de las entradas son:

$$\left\{\boldsymbol{p}_1=\begin{bmatrix}-1\\-1\end{bmatrix},\ ,\ \boldsymbol{t}_1=\begin{bmatrix}1\\0\end{bmatrix}\right\},\left\{\boldsymbol{p}_2=\begin{bmatrix}1\\1\end{bmatrix},\ \boldsymbol{t}_2=\begin{bmatrix}1\\0\end{bmatrix}\right\},\ \left\{\boldsymbol{p}_3=\begin{bmatrix}1\\-1\end{bmatrix},\ \boldsymbol{t}_3=\begin{bmatrix}0\\1\end{bmatrix}\right\},\left\{\boldsymbol{p}_4=\begin{bmatrix}-1\\1\end{bmatrix},\ \boldsymbol{t}_4=\begin{bmatrix}0\\1\end{bmatrix}\right.$$

La posición inicial de los patrones de entrada es la siguiente:

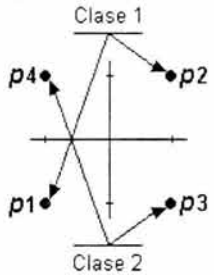


Figura 2.5.19 Posición de los patrones de entrada

Si se escogen dos subclases para cada una de las dos clases existentes, tendremos entonces cuatro subclases, lo que determina que deben haber cuatro neuronas en la capa oculta. La matriz de pesos para la capa de salida es:

$$W^2=\begin{bmatrix}1 & 1 & 0 & 0\\ 0 & 0 & 1 & 1\end{bmatrix}$$

W^2 conecta las neuronas ocultas 1 y 2 a la neurona de salida 1 y las neuronas ocultas 3 y 4 a la neurona de salida 2. Cada clase será formada por dos regiones convexas.

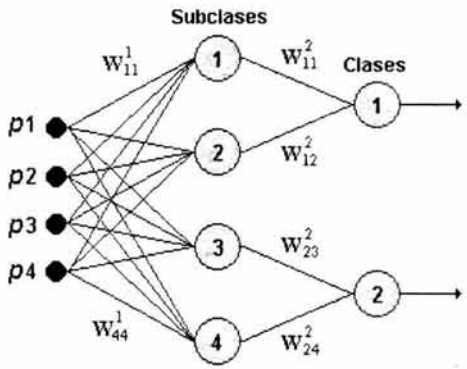


Figura 2.5.20 Esquema de la red LVQ que solucionará el ejemplo

W^1 será inicializada con valores aleatorios, de la siguiente forma:

$$ {}_1w^1 = \begin{bmatrix} -0.673 \\ 0.970 \end{bmatrix}, \; {}_2w^1 = \begin{bmatrix} -0.969 \\ -0.379 \end{bmatrix}, \; {}_3w^1 = \begin{bmatrix} 1.002 \\ 0.140 \end{bmatrix}, \; {}_4w^1 = \begin{bmatrix} 0.785 \\ 0.955 \end{bmatrix} $$

La posición inicial de estos vectores de pesos, se observa en la figura 2.5.21

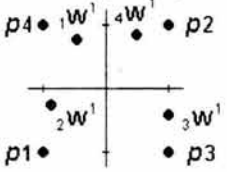


Figura 2.5.21 Estado inicial del vector de peso

Un vector de entrada diferente será presentado en cada iteración, se encontrará su respuesta y luego se actualizaran los pesos correspondientes. Se presentará inicialmente $p3$ a la red:

$$ a^1 = compet(n^1) = compet\left(\begin{bmatrix} -\|{}_1w^1 - p_3\| \\ -\|{}_2w^1 - p_3\| \\ -\|{}_3w^1 - p\| \\ -\|{}_4w^1 - p\| \end{bmatrix}\right) $$

$$ a^1 = compet\left(\begin{bmatrix} -\|[-0.673 \quad 0.970]^T - [1 \quad -1]^T\| \\ -\|[-0.969 \quad -0.379]^T - [1 \quad -1]^T\| \\ -\|[1.002 \quad 0.140]^T - [1 \quad -1]^T\| \\ -\|[0.7805 \quad 0.955]^T - [1 \quad -1]^T\| \end{bmatrix}\right) = compet\left(\begin{bmatrix} -2.5845 \\ -2.0646 \\ -1.1400 \\ -1.9668 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} $$

La tercera neurona oculta ha estado más cerca del vector $p3$ y de esta forma ya se determino la subclase, ahora determinamos la clase a la cual pertenece multiplicando a^1 por W^2

$$ a^2 = W^2 a^1 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} $$

La salida de la red indica que p_3 es un miembro de la clase 2, lo cual es correcto por lo tanto ${}_3w^1$ es desplazado en la dirección de $p3$.

${}_3w^1(1) = {}_3w^1(0) + \alpha\,(p_3 - {}_3w^1(0))$

$$ {}_3w^1(1) = \begin{bmatrix} 1.002 \\ 0.140 \end{bmatrix} + 0.5\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} - \begin{bmatrix} 1.002 \\ 0.140 \end{bmatrix}\right) = \begin{bmatrix} 1.001 \\ -0.430 \end{bmatrix} $$

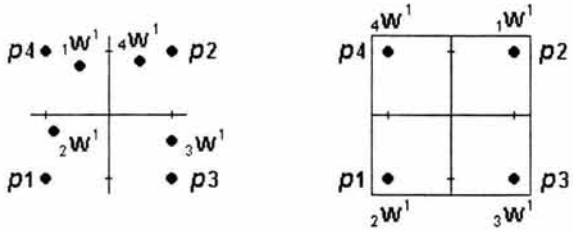


Figura 2.5.22 Resultado después de la primera y después de muchas iteraciones

El diagrama al lado izquierdo de la figura 2.5.22, muestra como el vector peso ${}_3w^1$ es actualizado después de la primera iteración; el diagrama de la derecha, muestra la localización de los pesos después de que el algoritmo ha alcanzado convergencia, además en esta parte de la gráfica puede verse como las regiones del espacio de entrada son clasificadas. Los vectores de entrada $\boldsymbol{p}_1$ y $\boldsymbol{p}2$ perteneciente a la clase uno son visualizadas en azul y los vectores $\boldsymbol{p}3$ y $\boldsymbol{p}4$ pertenecientes a la clase dos pueden verse en blanco.

REDES RECURRENTES

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

Red de Hopfield:

Antecedentes: En la década de los 80´s con el fin de estudiar procesos que involucran sistemas gobernados por ecuaciones diferenciales no lineales surge la teoría clásica de control geométrico basada en la geometría diferencial; simultáneamente renace el estudio de las Redes Neuronales debido al redescubrimiento del algoritmo Backpropagation, este hecho sumado al fracaso de las metodologías tradicionales aplicadas a la inteligencia artificial y a la disponibilidad de herramientas computacionales de bajo costo permitieron el desarrollo las redes neuronales recurrentes cuya principal aplicación es el control e identificación de sistemas no lineales. Este desarrollo es posible debido a que las propiedades matemáticas de las redes recurrentes están enmarcadas en las mismas propiedades que fundamentan el control geométrico, la primera red neuronal recurrente de naturaleza dinámica fue propuesta por Hopfield en 1984 bajo el contexto de las memorias asociativas.

Estructura de la red: En búsqueda de una implementación practica, Hopfield presentó su modelo básico como un circuito eléctrico, el cual se muestra en la figura 2.6.1, donde cada neurona se representa por un amplificador operacional y una red asociada formada por una capacitancia y una resistencia, la entrada a cada amplificador es la suma de las corrientes I_i mas las realimentaciones provenientes de otros amplificadores, por ejemplo el segundo amplificador realimenta al amplificador S a través de la resistencia R_{S2}, en caso de necesitarse realimentaciones con signo negativo, estas se hacen por medio de la salida inversora de cada amplificador; la ecuación para el modelo de Hopfield basado en las leyes de Kirchhoff se muestra en la ecuación (2.6.1).

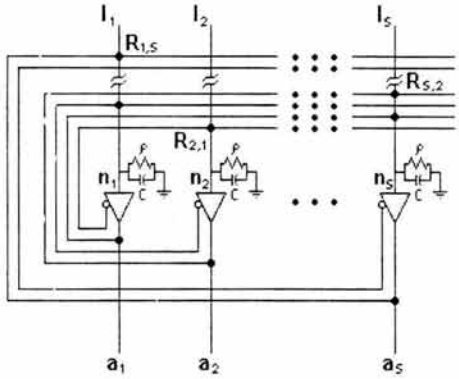


Figura 2.6.1 Circuito Eléctrico red Hopfield

$$C\frac{dn_i(t)}{dt} = \sum_{j=1}^{S} T_{ij}\,a_j(t) - \frac{n_i(t)}{R_i} + I_i \qquad (2.6.1)$$

Donde n_i es el voltaje de entrada a cada amplificador y $a_i = f(n_i)$ su salida, con característica de amplificación f la cual es generalmente de tipo sigmoidal,

$$|T_{ij}| = \frac{1}{R_{i,j}} \quad \text{y} \quad \frac{1}{R_i} = \frac{1}{\rho} + \sum_{j=1}^{S} \frac{1}{R_{ij}}$$

Multiplicando a ambos lados de la ecuación (2.6.1) por R_i y definiendo $\tau = R_iC$, $w_{ij} = R_iT_{ij}$ y $b_i = R_iI_i$, esta puede reescribirse en la ecuación (2.6.2) la cual describe el comportamiento de cada una de las neuronas dinámicas que componen el circuito eléctrico de la red de Hopfield.

$$\tau\frac{dn_i(t)}{dt} = -n_i(t) + \sum_{j=1}^{S} w_{ij}\,a_j + b_i \qquad (2.6.2)$$

Utilizando la ecuación (2.6.2) y escribiéndola en su forma matricial con $\mathbf{a}(t) = \mathbf{f}(\mathbf{n}(t))$, se obtiene (2.6.3), en esta ecuación se describe el comportamiento de la red de Hopfield

$$\tau\frac{d\mathbf{n}(t)}{dt} = -\mathbf{n}(t) + W\mathbf{a}(t) + \mathbf{b} \qquad (2.6.3)$$

La red de Hopfield en notación compacta se muestra en la figura 2.6.2, en donde el vector de $\mathbf{p}$ no se considera como la entrada a la red sino como la condición inicial de la red

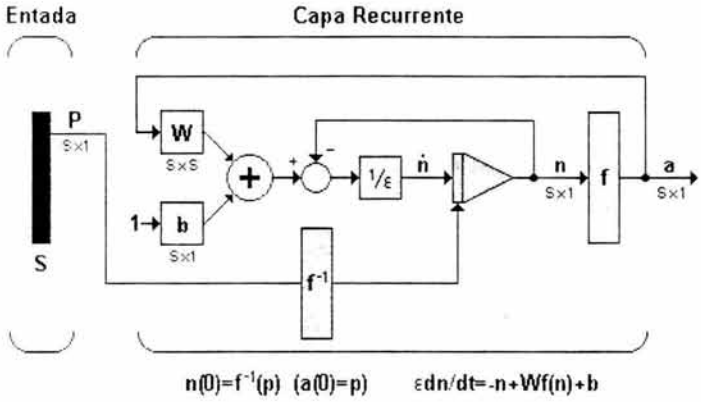


Figura 2.6.2 Notación compacta red de Hopfield

Como se observa, la red de Hopfield esta compuesta de neuronas dinámicas altamente interconectadas gobernadas por ecuaciones diferenciales no lineales, esta red funciona como una memoria asociativa no lineal que puede procesar patrones presentados de forma incompleta o con ruido, siendo útil como una poderosa herramienta de optimización

En el libro "Neural Network Design" [23], se muestra que una de las principales contribuciones de Hopfield fue la aplicación de la teoría de estabilidad de Lyapunov al análisis de las redes recurrentes, la teoría de estabilidad de Lyapunov se aplica a través del teorema de LaSalle y para su utilización el primer paso es escoger una función de Lyapunov, para lo cual Hopfield sugirió la siguiente función:

$$V(\boldsymbol{a}) = -\frac{1}{2}\boldsymbol{a}^T\boldsymbol{W}\boldsymbol{a} + \sum_{i=1}^{S}\left\{\int_0^{a_i} f^{-1}(u)du\right\} - \boldsymbol{b}^T\boldsymbol{a} \qquad (2.6.4)$$

Donde ***a*** es la salida de la red, ***W*** es la matriz de pesos y ***b*** es el vector de ganancias.

La escogencia de esta particular función, fue clave en el desarrollo de Hopfield, pues el primer y el tércer termino de esta ecuación conforman una función cuadrática, las cuales pueden aproximar gran cantidad de funciones en un pequeño intervalo, especialmente cerca de puntos donde se encuentre un mínimo local.

Para usar el teorema de LaSalle se necesita evaluar la derivada de la ecuación 2.6.4, por claridad se evaluará cada uno de los tres términos de forma independiente, tomando la derivada del primer término de la ecuación 2.6.4 se obtiene:

$$\frac{d}{dt}\left(-\frac{1}{2}\boldsymbol{a}^T\boldsymbol{W}\boldsymbol{a}\right) = -\frac{1}{2}\nabla\left[\boldsymbol{a}^T\boldsymbol{W}\boldsymbol{a}\right]^T\frac{d\boldsymbol{a}}{dt} = -\left[\boldsymbol{W}\boldsymbol{a}\right]^T\frac{d\boldsymbol{a}}{dt} = -\boldsymbol{a}^T\boldsymbol{W}\frac{d\boldsymbol{a}}{dt} \quad (2.6.5)$$

Derivando el segundo termino de la ecuación 2.6.4, el cual consiste de una sumatoria de integrales y considerando una de estas integrales se obtiene:

$$\frac{d}{dt}\left\{\int_0^{a_i} f^{-1}(u)du\right\} = \frac{d}{da_i}\left\{\int_0^{a_i} f^{-1}(u)du\right\}\frac{da_i}{dt} = f^{-1}(a_i)\frac{da_i}{dt} = n_i\frac{da_i}{dt} \quad (2.6.6)$$

Tomando en consideración todas las integrales, en forma matricial la derivada del segundo termino es:

$$\frac{d}{dt}\left[\sum_{i=1}^{S}\left\{\int_0^{a_i} f^{-1}(u)du\right\}\right] = \boldsymbol{n}^T\frac{d\boldsymbol{a}}{dt} \quad (2.6.7)$$

Derivando el tercer término de la ecuación 2.6.4 y apoyándose en las propiedades de las funciones cuadráticas se obtiene la ecuación 2.6.8

$$\frac{d}{dt}\left\{-\boldsymbol{b}^T\boldsymbol{a}\right\} = -\nabla\left[\boldsymbol{b}^T\boldsymbol{a}\right]^T\frac{d\boldsymbol{a}}{dt} = -\boldsymbol{b}^T\frac{d\boldsymbol{a}}{dt} \quad (2.6.8)$$

La derivada total de la ecuación 2.6.8 se obtiene al unir los resultados de las ecuaciones 2.6.5, 2.6.7 y 2.6.8

$$\frac{d}{dt}V(\boldsymbol{a}) = -\boldsymbol{a}^T\boldsymbol{W}\frac{d\boldsymbol{a}}{dt} + \boldsymbol{n}^T\frac{d\boldsymbol{a}}{dt} - \boldsymbol{b}^T\frac{d\boldsymbol{a}}{dt} = \left[-\boldsymbol{a}^T\boldsymbol{W} + \boldsymbol{n}^T - \boldsymbol{b}^T\right]\frac{d\boldsymbol{a}}{dt} \quad (2.6.9)$$

comparando con la ecuación (2.6.3) del modelo eléctrico de Hopfield, se tiene que:

$$\left[-\boldsymbol{a}^T\boldsymbol{W} + \boldsymbol{n}^T - \boldsymbol{b}^T\right]\frac{d\boldsymbol{a}}{dt} = -\varepsilon\left[\frac{d\boldsymbol{n}(t)}{dt}\right]^T \quad (2.6.10)$$

Esto permite reescribir la ecuación 2.6.9 así como sigue:

$$\frac{d}{dt}V(\boldsymbol{a}) = -\varepsilon\left[\frac{d\boldsymbol{n}(t)}{dt}\right]^T\frac{d\boldsymbol{a}}{dt} = -\varepsilon\sum_{i=1}^{S}\left(\frac{dn_i}{dt}\right)\left(\frac{da_i}{dt}\right) \quad (2.6.11)$$

ya que $n_i = f^{-1}(a_i)$, es posible expandir la derivada de n_i de la siguiente forma:

$$\frac{dn_i}{dt} = \frac{d}{dt}\left[f^{-1}(a_i)\right] = \frac{d}{da_i}\left[f^{-1}(a_i)\right]\frac{da_i}{dt} \quad (2.6.12)$$

Con esto la ecuación (2.6.11) puede ser reescrita como:

$$\frac{d}{dt}V(\boldsymbol{a}) = -\varepsilon\sum_{i=1}^{S}\left(\frac{dn_i}{dt}\right)\left(\frac{da_i}{dt}\right) = -\varepsilon\sum_{i=1}^{S}\left(\frac{d}{da_i}\left[f^{-1}(a_i)\right]\right)\left(\frac{da_i}{dt}\right)^2 \quad (2.6.13)$$

si se asume que $f^{-1}(a_i)$ es una función incremental, como sucede en los amplificadores operacionales, entonces:

$$\frac{d}{da_i}\left[f^{-1}(a_i)\right] > 0 \quad (2.6.14)$$

Este resultado implica en la ecuación 2.6.12 que:

$$\frac{d}{dt}V(\boldsymbol{a}) \le 0 \quad (2.6.15)$$

De esta manera, si $f^{-1}(a_i)$ es una función incremental, todos los valores propios de la función $dV(\boldsymbol{a})/dt$ son no positivos lo cual implica que la red sea estable, entonces $V(\boldsymbol{a})$ es una función de Lyapunov valida

Los atractores de Hopfield son puntos estacionarios de la función de Lyapunov que satisfacen la ecuación (2.6.16)

$$\frac{d\boldsymbol{a}}{dt} = 0 \quad (2.6.16)$$

Estos puntos estacionarios son puntos donde se encuentra un mínimo de la función $V(\boldsymbol{a})$ descrita en la ecuación (2.6.4), en estos puntos el gradiente de la función $V(\boldsymbol{a})$ igual a cero [21].

$$\nabla V(\boldsymbol{a}) = \left[\frac{\partial V}{\partial a_1} \quad \frac{\partial V}{\partial a_2} \quad \cdots \quad \frac{\partial V}{\partial a_S}\right]^T = 0 \quad (2.6.17)$$

La función de Lyapunov descrita por la ecuación (2.6.4) puede simplificarse si se considera que la ganancia γ es grande, como sucede en los amplificadores con los que se implementa la red, una función de transferencia típica para estos amplificadores no lineales se muestra a continuación:

$$a = f(n) = \frac{2}{\pi}tan^{-1}\left(\frac{\gamma\pi n}{2}\right) \quad (2.6.18)$$

Para evaluar el segundo termino de la función de Lyapunov se requiere el calculo de $f^{-1}(u)$.

$$f^{-1}(u) = \frac{2}{\gamma\pi} tan\left(\frac{\pi u}{2}\right) \quad (2.6.19)$$

Si la ganancia γ es muy grande y la salida de la red se mantiene en el rango $1>a>-1$, el segundo termino de la función de Lyapunov tiende a cero y puede definirse la función de alta ganancia de Lyapunov como:

$$V(\boldsymbol{a}) = -\frac{1}{2}\boldsymbol{a}^T\boldsymbol{W}\boldsymbol{a} - \boldsymbol{b}^T\boldsymbol{a} \quad (2.6.20)$$

Regla de Aprendizaje

La red de Hopfield no tiene una ley de aprendizaje asociada, esto significa que la red no es entrenada ni realiza un proceso de aprendizaje, sin embargo es posible determinar la matriz de pesos por medio de un procedimiento basado en la función de alta ganancia de Lyapunov descrita por la ecuación 2.6.20.

$$V(\boldsymbol{a}) = -\frac{1}{2}\boldsymbol{a}^T\boldsymbol{W}\boldsymbol{a} - \boldsymbol{b}^T\boldsymbol{a} \quad (2.6.21)$$

El procedimiento consiste en escoger la matriz de pesos $\boldsymbol{W}$ y el vector de ganancias $\boldsymbol{b}$ tal que V toma la forma de la función que se quiere minimizar, convirtiendo el problema que se quiere resolver, en un problema de minimización cuadrática, puesto que la red de Hopfield minimizará a V

Una red de Hopfield puede diseñarse como una memoria asociativa, en este caso es llamada memoria de contenido direccionable, porque la memoria recupera la información almacenada con base en parte de su contenido, en contraste con las memorias estándar de computo, donde la información se recupera con base en sus direcciones, por ejemplo si se tiene una base de datos de contenido direccionable que contiene nombres y direcciones de los empleados de una empresa, la información completa se recupera por ejemplo suministrando el nombre (o parte de él), este tipo de memoria es la misma memoria autoasociativa excepto que en este caso se utilizará la red recurrente de Hopfield en vez del asociador lineal estudiado en la sección 2.4.

Cuando se le presenta un patrón de entrada a la red de Hopfield, el estado inicial de la salida será el mismo patrón de entrada y luego la red convergerá al patrón prototipo almacenado que se encuentre más cercano (o que más se parezca) al patrón de entrada, para que la red memorice un patrón prototipo, este debe ser un mínimo de la función de Lyapunov

Asumiremos que los patrones prototipo son $\{p_1, p_2, \ldots, p_Q\}$ y que cada uno de estos vectores se compone de S elementos, al asumir que $Q<<S$, el espacio de estado es amplio y los patrones prototipo se encuentran bien distribuidos y por lo tanto no están cercanos uno de otro.

Para garantizar que los patrones prototipo a almacenar son mínimos de la función de Lyapunov, se propone la siguiente función para evaluar el error en la aproximación.

$$J(\boldsymbol{a}) = -\frac{1}{2}\sum_{q=1}^{Q}\left(\left[\boldsymbol{p}_Q\right]^T \boldsymbol{a}\right)^2 \qquad (2.6.22)$$

Si los elementos de $\boldsymbol{a}$ son restringidos a valores de □ 1, la función es minimizada en los patrones prototipo como se mostrara a continuación:

Asumiendo que los patrones prototipo son ortogonales, y evaluando el error en uno de ellos, se tendrá que.

$$J(\boldsymbol{a}) = -\frac{1}{2}\sum_{q=1}^{Q}\left(\left[\boldsymbol{p}_Q\right]^T \boldsymbol{p}_j\right)^2 = -\frac{1}{2}\left(\left[\boldsymbol{p}_Q\right]^T \boldsymbol{p}_j\right)^2 = -\frac{S}{2} \qquad (2.6.23)$$

La segunda igualdad de la ecuación 2.6.23 se debe a la ortogonalidad de los patrones prototipo y la ultima igualdad a que todos los elementos de $\boldsymbol{p}_j$ son □ 1, evaluando el error del patrón aleatorio de entrada, el cual presumiblemente no esta cercano a ningún patrón prototipo, cada elemento de la sumatoria en la ecuación (2.6.22) es el producto punto entre un patrón prototipo y la entrada, el producto punto se incrementara cuando la entrada se mueva cerca del patrón prototipo, sin embargo, si la entrada no se encuentra cerca de algún patrón prototipo, todos los términos de la sumatoria serán pequeños y por lo tanto $J(\boldsymbol{a})$ será la mayor (menos negativa) y cuando $\boldsymbol{a}$ sea igual a alguno de los patrones prototipo $J(\boldsymbol{a})$ será mas pequeña (mas negativa).

La ecuación (2.6.22) es una función cuadrática que indica con precisión el desempeño del contenido de la memoria direccionable, el próximo paso es escoger la matriz de pesos $\boldsymbol{W}$ y ganancias $\boldsymbol{b}$, tal que la función de Lyapunov de Hopfield V sea equivalente al desempeño de la función cuadrática J.

Si se utiliza la regla de aprendizaje supervisado de Hebb para calcular la matriz de pesos (con patrones objetivo iguales a los patrones de entrada)

$$\mathbf{W} = \sum_{q=1}^{Q} p_q \left(p_q\right)^T \quad \text{y } \boldsymbol{b}=0 \qquad (2.6.24)$$

entonces la función de Lyapunov será:

$$V(\boldsymbol{a}) = -\frac{1}{2}\boldsymbol{a}^T\left[\sum_{q=1}^{Q} p_q \left(p_q\right)^T\right]\boldsymbol{a} = -\frac{1}{2}\sum_{q=1}^{Q}\boldsymbol{a}^T p_q \left(p_q\right)^T \boldsymbol{a} \qquad (2.6.25)$$

y puede ser reescrita como:

$$V(\boldsymbol{a}) = -\frac{1}{2}\sum_{q=1}^{Q}\left[\left(p_q\right)^T \boldsymbol{a}\right]^2 = J(\boldsymbol{a}) \quad (2.6.26)$$

Podemos observar que la función de Lyapunov es igual al desempeño del error del contenido de la memoria direccionable, la salida de la red de Hopfield tendera a converger a los patrones prototipo almacenados, en el caso que todos los patrones prototipo sean ortogonales, cada uno será un punto de equilibrio de la red; la red puede tener muchos otros puntos de equilibrio indeseables, una regla práctica para evitarlos consiste en que cuando se utilice la regla de Hebb, el número de patrones almacenados no debe superar el 15% del número de neuronas de la red.

Identificación de Sistemas No Lineales: El comportamiento dinámico de las redes recurrentes hace que sean una poderosa herramienta en la identificación de sistemas dinámicos no lineales.

En la forma estándar una neurona dinámica esta regida por la siguiente ecuación y se muestra en la figura 2.6.3

$$\dot{x}_i = -x_i + \sum_{j=1}^{N} \omega_{ij}\, \sigma(x_j) + \gamma_i u \qquad i = 1,\ldots,N \quad (2.6.27)$$

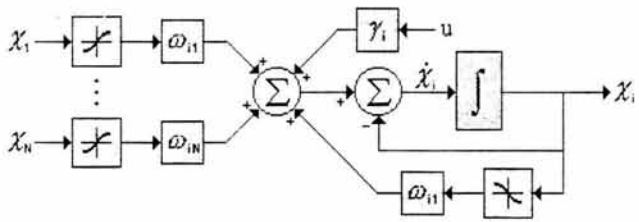


Figura 2.6.3 Neurona dinámica

o en forma matricial:

$$\dot{x} = A x + W \sigma(x) + \tilde{g} u \quad (2.6.28)$$

donde $A = -I$, $W = [\omega_{ij}]$, $\sigma(x) = [\sigma(x_1) \quad \cdots \quad \sigma(x_N)]^T$, y $\tilde{g} = [\gamma_i]$

En la figura 2.6.4 se observa una red neuronal dinámica recurrente, donde cada unidad de procesamiento es una neurona dinámica y cada punto es un peso.

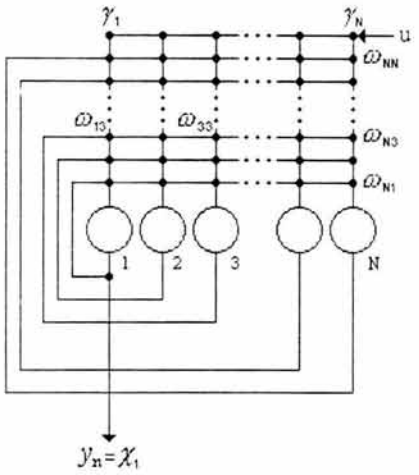


Figura 2.6.4 Red neuronal dinámica recurrente

Para garantizar la estabilidad de las redes dinámicas recurrentes en el proceso de identificación de sistemas no lineales, Delgado[9] formuló condiciones estrictas para los pesos la red y su desarrollo se basa en la función de Lyapunov.

Para el entrenamiento de la red de Hopfield en identificación de sistemas, se utiliza el algoritmo de Chemotaxis, el cual permite entrenar redes neuronales de cualquier tipo sin calcular el gradiente del error, este algoritmo fue formulado considerando el movimiento de una bacteria en un medio donde hay un gradiente de solución alimenticia; la bacteria se mueve inicialmente al azar hasta detectar un aumento en la concentración de la solución y luego continua en esa dirección.

El algoritmo de Chemotaxis toma los pesos iniciales al azar con distribución Gaussinana, cuando una iteración es exitosa (disminuye el valor de la función de error) el algoritmo continua en esta dirección hasta que la función de error J no muestra cambios

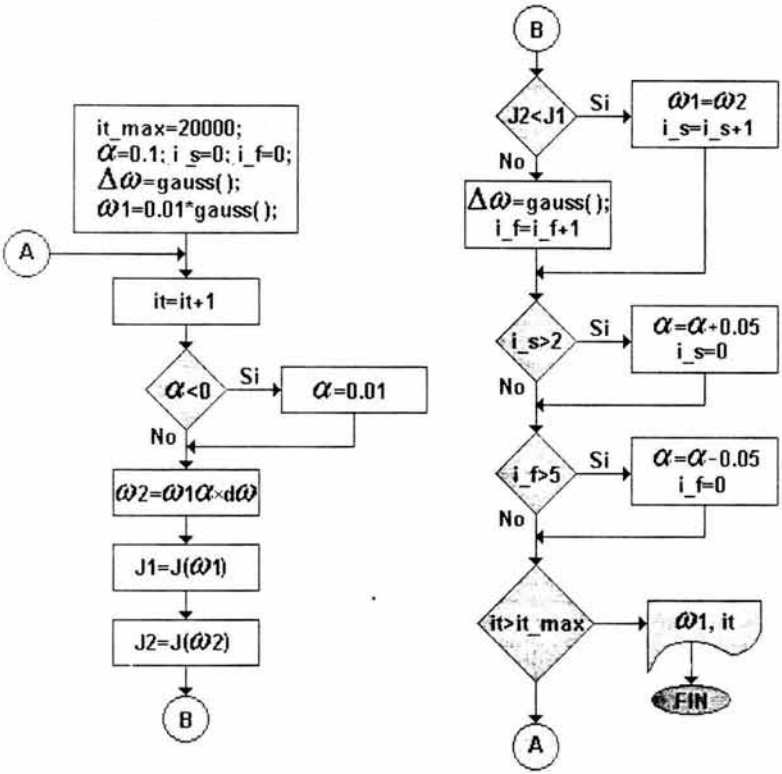


Figura 2.6.5 Algoritmo de Chemostaxis

it_max: Numero máximo de iteraciones
it: Contador de iteraciones
i_s: Contador de iteraciones exitosas
i_f: Contador de iteraciones no exitosas
□ : Rata de aprendizaje
□ 1: Antigua matriz de pesos
□ 2: Antigua matriz de pesos
□ □ : Perturbacion en la matriz de pesos
gauss( ): Generador de números aleatorios con distribución Gaussiana
Ji: Indice de la función de error correspondiente a la matriz de pesos □ i.

La función de error *Ji* relaciona la salida del sistema a aproximar con la salida de la red dinámica entrenada con *NP* patrones de entrenamiento.

$$J = \sum_{k=1}^{NP} (d_k - y_k)^2 \qquad (2.6.29)$$

d_k: Salida deseada para el patrón de entrenamiento *k*.

y_k: Salida actual de la red ante el patrón de entrenamiento *k*.

Redes Multicapa

Estructura de la red: Las redes multicapa son de naturaleza estática, o sea su salida no evoluciona con el tiempo (para un patrón de entrada existe una salida asociada), pero pueden adquirir un comportamiento dinámico (para un patrón entrada la salida posee un estado transitorio y converge a un valor en el estado estacionario) realimentando sus entradas con estados anteriores de sus salidas.

La red esta compuesta de una capa estática la cual generalmente posee un número de neuronas superior al número de variables de estado del sistema a identificar, la salida de la capa estática va a un sumador donde se le resta el valor anterior de la variable de estado Z_i identificada por el sistema, de esta operación se genera la derivada de cada una de las *i* variables de estado identificadas por el sistema.

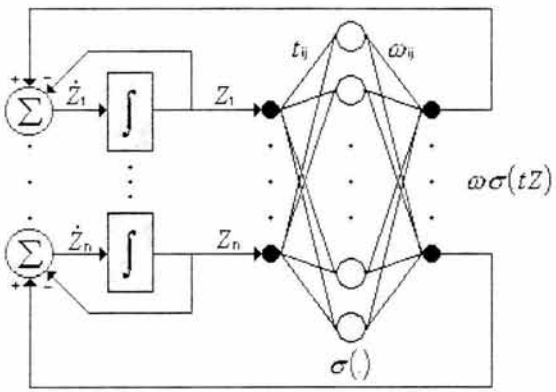


Figura 2.6.6 Red Dinámica Multicapa

La red recurrente dinámica multicapa cuyo comportamiento lo describe la ecuación (2.6.30) puede identificar el comportamiento de un sistema autónomo (u=0) descrito por la ecuación (2.6.31)

$$\frac{d}{dt}z = \bar{f}(z) = Ax + \omega\sigma(Tz) \qquad (2.6.30)$$

$$\frac{d}{dt}x = f(x) = Ax + f_0(x) \qquad (2.6.31)$$

donde $x, z \in \Re^n$, $A \in \Re^{nxn}$, $f(x): \Re^n \to \Re^n$, $\bar{f}(z): \Re^n \to \Re^n$, $W \in \Re^{nxN}$, $T \in \Re^{nxN}$, $\sigma(z) = [\sigma(z_1), \sigma(z_2), \ldots, \sigma(z_n)]$ y función de transferencia □ (□

)=*tansig*( ), n es el numero de variables de estado del sistema y N el numero de neuronas en la capa oculta.

Según Delgado[9], sin perdida de generalidad si el origen se asume como punto de equilibrio, el sistema (2.6.31) será identificado con la red (2.6.30) alrededor de su región de atracción y se garantiza que el error en la aproximación $e(t)$ es limitado.

Regla de Aprendizaje: La etapa estática que hace parte de la red multicapa dinámica recurrente generalmente es entrenada con el algoritmo de Chemotaxis o cualquier algoritmo de propagación inversa (Backpropagation), estos algoritmos fueron descritos en la sección 2.3, el algoritmo de Chemotaxis fue explicado en el contexto de la identificación de sistemas dinámicos por medio de la red de Hopfield donde es realmente indispensable, pues para redes dinámicas multicapa los algoritmos de propagación inversa son más eficientes y rápidos.

Los patrones de entrenamiento de la capa estática de la figura (2.6.6) son diferentes combinaciones de valores de las variables de estado y los patrones objetivo están dados por la suma de cada variable de estado con su correspondiente derivada como se muestra en la figura 2.6.7

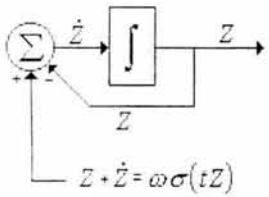


Figura 2.6.7 Patrones de entrenamiento de la red multicapa

La red después de entrenada tiene la estructura de la ecuación (2.6.32)

$$\frac{d}{dt}\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} -z_1 \\ -z_2 \\ \vdots \\ -z_n \end{bmatrix} + \begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1n} \\ W_{21} & W_{22} & \cdots & W_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ W_{n1} & W_{n2} & \cdots & W_{nn} \end{bmatrix} \begin{bmatrix} \sigma(t_{11}z_1 + t_{12}z_2 + \ldots + t_{1n}z_n) \\ \sigma(t_{21}z_2 + t_{22}z_2 + \ldots + t_{2n}z_n) \\ \vdots \\ \sigma(t_{n1}z_1 + t_{n2}z_2 + \ldots + t_{nn}z_n) \end{bmatrix} \quad (2.6.32)$$

Para garantizar que la red a identificado la dinámica del sistema, el Jacobiano de la red en el origen (2.6.33) debe tener valores propios muy cercanos a los del sistema que ha sido aproximado.

$$J_M = -I_n + WT \quad (2.6.33)$$

- Transformado una red dinámica multicapa en una red dinámica recurrente tipo Hopfield

La red dinámica multicapa de la figura (2.6.6), puede transformarse en una red dinámica tipo Hopfield por medio de la siguiente transformación lineal descrita en la ecuación (2.6.34)

$$\chi = Tz \qquad \frac{d\chi}{dt} = T\frac{dz}{dt} \quad (2.6.34)$$

Generalmente la matriz T es cuadrada, pero en caso no ser cuadrada la transformación se realiza por medio de la inversa generalizada; la red transformada tendrá la estructura (2.6.35)

$$\frac{d}{dt}\chi = -I_N\chi + TW\sigma(\chi) \quad (2.6.35)$$

donde el nuevo vector de estado $\chi \in \Re^N$, $TW \in \Re^{NxN}$, I_N es la matriz identidad de dimensión N, la transformación (2.6.34) extiende la red dinámica multicapa (2.6.32) en la red dinámica recurrente de Hopfield (2.6.35), aunque en la red de Hopfield no existen neuronas en la capa oculta el numero de estados es mayor o igual al numero de estados de la red multicapa N□ n

Después de realizar la transformación, la red tiene la estructura (2.6.36)

$$\frac{d}{dt}\begin{bmatrix}\chi_1\\ \chi_2\\ \vdots\\ \chi_N\end{bmatrix} = \begin{bmatrix}-\chi_1\\ -\chi_2\\ \vdots\\ -\chi_N\end{bmatrix} + [TW]\begin{bmatrix}\sigma(\chi_1)\\ \sigma(\chi_2)\\ \vdots\\ \sigma(\chi_N)\end{bmatrix} \quad (2.6.36)$$

El Jacobiano de la red descrito en la ecuación 2.6.37 debe tener valores propios muy cercanos a los del sistema que ha sido aproximado e iguales a los de la red multicapa.

$$J_H = -I_N + TW \quad (2.6.37)$$

Red de Elman

Estructura de la Red: La red de Elman típicamente posee dos capas, cada una compuesta de una red tipo Backpropagation, con la adición de una conexión de realimentación desde la salida de la capa oculta hacia la entrada de la misma capa oculta, esta realimentación permite a la red de Elman aprender a reconocer y generar patrones temporales o variantes con el tiempo.

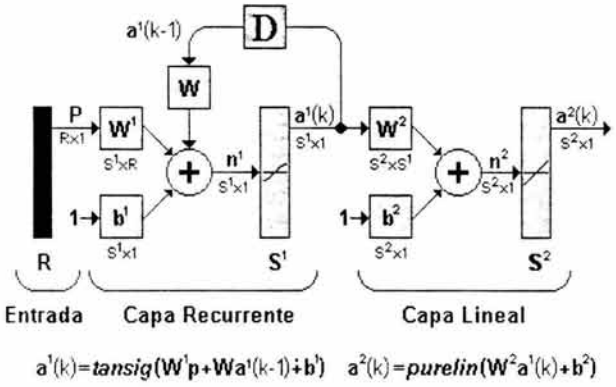


Figura 2.6.8 Red de Elman

La red de Elman generalmente posee neuronas con función transferencia sigmoidal en su capa oculta, en este caso *tansig* y neuronas con función de transferencia tipo lineal en la capa de salida, en esta caso *purelin*, la ventaja de la configuración de esta red de dos capas con este tipo de funciones de trasferencia, es que según lo demostrado por Funahashi [16], puede aproximar cualquier función con la precisión deseada mientras que esta posea un numero finito de discontinuidades, para lo cual la precisión de la aproximación depende de la selección del numero adecuado de neuronas en la capa oculta.

Para la red de Elman la capa oculta es la capa recurrente y el retardo en la conexión de realimentación almacena los valores de la iteración previa, los cuales serán usados en la siguiente iteración; dos redes de Elman con los mismos parámetros y entradas idénticas en las mismas iteraciones podrían producir salidas diferentes debido a que pueden presentar diferentes estados de realimentación.

Entrenamiento de la red: Debido a la estructura similar de la red de Elman con una red tipo Backpropagation, esta red puede entrenarse con cualquier algoritmo de propagación inversa como los explicados en la sección 2.3 de este capitulo, entre los cuales se destacan los algoritmos basados en técnicas de optimización como el del gradiente conjugado o el algoritmo de Levemberg Marquard.

El entrenamiento de la red puede resumirse en los siguientes pasos:

- Presentar a la red, los patrones de entrenamiento y calcular la salida de la red con los pesos iniciales, comparar la salida de la red con los patrones objetivo y generar la secuencia de error.
- Propagar inversamente el error para encontrar el gradiente del error para cada conjunto de pesos y ganancias,
- Actualizar todos los pesos y ganancias con el gradiente encontrado con base en el algoritmo de propagación inversa.

La red de Elman no es tan confiable como otros tipos de redes porque el gradiente se calcula con base en una aproximación del error, para solucionar un problema con este tipo de red se necesitan más neuronas en la capa oculta que si se solucionara el mismo problema con otro tipo de red.

ANEXO B

EJEMPLO DE APLICACIONES DE REDES NEURONALES

DETECCIÓN DE OBSTÁCULOS POR MEDIO DE UN ROBOT

Descripción del problema:

Un robot es un dispositivo automático que realiza acciones especificas, que dependen de las necesidades del proceso en que se encuentre involucrado, en este caso se tiene un robot que cuenta con cuatro sensores de proximidad en distintas ubicaciones que permanentemente detectan si hay objetos que se encuentren a una distancia superior o inferior a la preestablecida, con base en esto se decide si dar marcha adelante o atrás a cada uno de los dos motores que posee; en las lecturas de los sensores podrían darse 16 posibles combinaciones ($16=2^4$) y para cada combinación cada uno de los dos motores podría dar marcha adelante o marcha atrás.

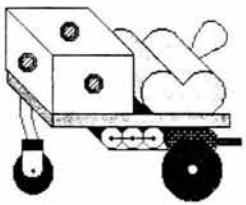


Figura 3.1.1 Robot

El comportamiento del robot lo describe la tabla 3.1.2, cuando los sensores detecten un objeto que se encuentra a una distancia inferior a la predeterminada se dirá que el objeto se encuentra cerca y esto se representa por medio de un 1 y cuando se detecte un objeto que se encuentra a una distancia mayor que la predeterminada se dirá que el objeto esta lejos lo cual se indica con un –1; dependiendo de estas lecturas los motores podrán dar marcha adelante, lo que se representara por un 1 o dar marcha atrás con un –1.

S1	S2	S3	S4	M1	M2
1	1	1	1	-1	-1
-1	1	1	1	-1	1
1	1	-1	-1	1	-1
-1	-1	-1	-1	1	1
1	-1	1	1	1	-1
1	1	-1	1	-1	1

1	1	1	-1	1	-1

Tabla 3.1.1 Comportamiento del robot

Justificación del tipo de red

Este tipo de problema generalmente es resuelto suministrándole al robot una base de datos que contiene todas las posibles situaciones que se podrían presentarse y sus respectivas soluciones, en este caso se necesitaría almacenar las respuestas para ambos motores ante las 16 posibles combinaciones en las lecturas de los sensores, cuando el número de variables de entrada y el número de salidas es mucho mayor, la cantidad de datos necesarios para especificar cada posible situación crece indefinidamente, debido a esto se requerirían dispositivos con gran capacidad de almacenamiento; *en contraste una red neuronal puede entrenarse con un número representativo de patrones y aprender el comportamiento del sistema utilizando dispositivos de menos capacidad de almacenamiento y costo.*

Una red tipo Perceptrón puede ser entrenada con patrones de cualquier dimensión en la entrada y en la salida con datos binarios, por la simplicidad del problema este tipo de red es la mas adecuada.

Para garantizar que el problema puede ser resuelto por una red neuronal tipo Perceptrón se debe comprobar que los patrones de entrenamiento son linealmente separables, para esto se deben plantear las desigualdades generadas por cada patrón de entrenamiento, en este caso cada patrón de cuatro dimensiones generara dos desigualdades (una por cada salida), estas desigualdades no deben contradecirse, si esto ocurriera el problema no podría ser resuelto por una red tipo Perceptrón de una sola capa y deberá buscarse otro tipo de solución.

Debido a la naturaleza bipolar de la salida, la función de transferencia a utilizar es *hardlims* (ver sección 1.3.2), la cual se rige por las siguientes condiciones.

$$\left\{\begin{matrix} n \geq 0 & a = 1 \\ n < 0 & a = -1 \end{matrix}\right\} \quad (3.1.1)$$

La salida de la red está dada por la ecuación 3.1.2, (ver sección 2.1)

$$n = (\boldsymbol{W}\boldsymbol{p} + b) \quad (3.1.2)$$

Aplicando esta ecuación a cada patrón de entrenamiento se tienen las desigualdades de la tabla 3.1.2, las cuales se satisfacen plenamente, lo que implica que el problema es linealmente separable y puede ser resuelto por una red tipo Perceptron

$$p_1\begin{cases} W_{11}+W_{12}+W_{13}+W_{14}+b_1<0 \\ W_{21}+W_{22}+W_{23}+W_{24}+b_2<0 \end{cases} \quad p_2\begin{cases} -W_{11}+W_{12}+W_{13}+W_{14}+b_1<0 \\ -W_{21}+W_{22}+W_{23}+W_{24}+b_2\geq 0 \end{cases}$$

$$p_3\begin{cases} W_{11}+W_{12}-W_{13}-W_{14}+b_1\geq 0 \\ W_{21}+W_{22}-W_{23}-W_{24}+b_2<0 \end{cases} \quad p_4\begin{cases} -W_{11}-W_{12}-W_{13}-W_{14}+b_1\geq 0 \\ -W_{21}-W_{22}-W_{23}-W_{24}+b_2\geq 0 \end{cases}$$

$$p_5\begin{cases} W_{11}-W_{12}+W_{13}+W_{14}+b_1\geq 0 \\ W_{21}-W_{22}+W_{23}+W_{24}+b_2<0 \end{cases} \quad p_6\begin{cases} W_{11}+W_{12}-W_{13}+W_{14}+b_1<0 \\ W_{21}+W_{22}-W_{23}+W_{24}+b_2\geq 0 \end{cases}$$

$$p_7\begin{cases} W_{11}+W_{12}+W_{13}-W_{14}+b_1\geq 0 \\ W_{21}+W_{22}+W_{23}-W_{24}+b_2<0 \end{cases}$$

Tabla 3.1.2 Desigualdades que garantizan que el problema sea linealmente separable

Entrenamiento de la red:

A la red se le presentaran 7 patrones de la tabla 3.1.1, para los cuales dependiendo de las lecturas de los sensores se le dirá al robot que hacer especificamente y luego se probará la red con los casos restantes para comprobar la capacidad de generalización de la red neuronal ante un patrón nunca antes visto.

Los estados de lecturas de los sensores y de operación de los motores fueron designados con 1 y –1, puesto que para la convergencia del proceso de entrenamiento resulta más ventajosos propagar valores de 1 y –1 que de 1 y 0.

Debido a la naturaleza de la salida y de la entrada de la red, la función de transferencia apropiada es *hardlims*, la cual trabaja con valores bipolares.

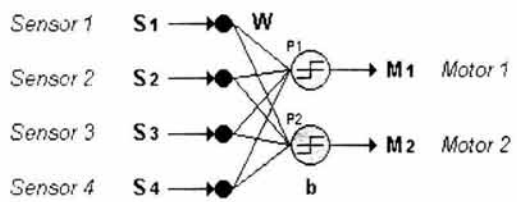


Figura 3.1.2 Red tipo Perceptrón

Se creará una red de 4 entradas con una neurona tipo Perceptrón para cada salida, teniendo así una salida bidimensional, los pesos iniciales aleatorios de la red se muestran en la tabla 3.1.3

Wi=net.IW{1,1}						bi=net.b{1}	
	S1	S2	S3	S4			
M1	0.8636	-0.1627	0.0503	0.3443		M1	1
M2	-0.0680	0.6924	-0.5947	0.6762		M2	1

Tabla 3.1.3 Pesos Iniciales

El siguiente código crea una red tipo Perceptrón con función de transferencia *hardlims*, dos neuronas en la salida, utiliza como patrones de entrenamiento las lecturas de los cuatro sensores almacenados en ***p*** y como patrones objetivo o salidas deseadas las acciones de ambos motores almacenados en el vector ***t***.

```
net=newp([-1 1;-1 1;-1 1;-1 1],2,'hardlims');
net.adaptParam.passes=200;
Wi;
[net,a,e]=adapt(net,P,t);
Wf=net.IW{1,1};
bf=net.b{1};
```

Los pesos finales de la red entrada que satisface todos los patrones de entrada y salida son:

Wf=net.IW{1,1}						bf=net.b{1}	
	S1	S2	S3	S4			
M1	0.8636	-4.1627	0.0503	-3.6557		M1	1
M2	-4.0680	0.6924	-4.5947	4.6762		M2	1

Tabla 3.1.4 Pesos finales red entrenada

La red fue simulada para la totalidad de combinaciones posibles de entrada para comprobar que no exista error en el aprendizaje de los patrones de entrenamiento y para observar su capacidad de generalización en los casos restantes

```
S1=[1 -1]; S2=[1 -1]; S3=[1 -1]; S4=[1 -1];
P=combvec(S1,S2,S3,S4);
net=newp([-1 1;-1 1;-1 1;-1 1],2,'hardlims');
```

Wf:
t=sim(net,P);

La respuesta de la red a todos los patrones de entrenamiento fue exitosa, como puede observarse en la tabla 3.1.5

	S1	S2	S3	S4	M1	M2
P1	1	1	1	1	-1	-1
P2	-1	1	1	1	-1	1
P3	1	1	-1	-1	1	-1
P4	-1	-1	-1	-1	1	1
P5	1	-1	1	1	1	-1
P6	1	1	-1	1	-1	1
P7	1	1	1	-1	1	-1

Tabla 3.1.5 Simulación de la red para los Patrones de Entrenamiento

La red fue simulada para las posibles combinaciones restantes obteniéndose los siguientes resultados:

	S1	S2	S3	S4	M1	M2
C1	-1	-1	1	1	1	1
C2	-1	1	-1	1	-1	1
C3	1	-1	-1	1	1	1
C4	-1	-1	-1	1	1	1
C5	-1	1	1	-1	-1	-1
C6	1	-1	1	-1	1	-1
C7	-1	-1	1	-1	1	-1
C8	-1	1	-1	-1	-1	1
C9	1	-1	-1	-1	1	-1

Tabla 3.1.6 Simulación de la red para las nuevas combinaciones

Las combinaciones que no hacían parte del set de entrenamiento, al ser presentadas a la red fueron aproximadas al patrón del set de entrenamiento aprendido con menor distancia euclidiana.

Para las combinaciones C1, C2 y C4 la red decidió dar marcha adelante a ambos motores; para la combinación C5 la red decidió dar marcha atrás a ambos motores, para las combinaciones C6, C7 y C9 la red decidió dar marcha adelante a M1 y marcha atrás a M2 y para las combinaciones C2 y C8 la red decidió dar marcha atrás a M1 y marcha adelante a M2

Una red tipo Perceptrón de una sola capa es una buena solución a un problema que involucre patrones linealmente separables, en el caso de contar con patrones que no son linealmente separables se tiene la alternativa de utilizar una red Perceptrón multicapa o cambiar definitivamente de red, nótese que una red Perceptrón multicapa puede solucionar el problema de separabilidad lineal a medida que aumenta el numero de capas de la red.

La capacidad de generalización de las redes neuronales juega un papel importante cuando las posibles combinaciones de patrones de entrada son tantas que resultaría imposible especificarle a un dispositivo que hacer en cada caso, puesto que la red se entrena con un número de patrones representativo y no con la totalidad de ellos, ahorrando tiempo de computo en la solución del problema.

En las aplicaciones desarrolladas con redes neuronales juega un papel importante la tolerancia a fallas que las caracteriza, pues en caso de fallar uno o varios sensores (como en este caso) la red siempre producirá una salida que en la mayoría de los casos es la mas acertada, debido a que la red después de un proceso de aprendizaje exitoso esta en capacidad de generalizar el comportamiento del sistema.

FILTRO ADAPTIVO

Descripción del problema:

Sin duda la principal aplicación de la red Adaline está en el campo del procesamiento de señales, en concreto para el diseño y realización de filtros que eliminen el ruido de señales portadoras de información. Como filtro adaptivo se ha utilizado ésta red en numerosas aplicaciones, destacándose su uso en filtros de ecualización adaptivos en modems de alta velocidad y canceladores adaptivos del eco para filtrado de señales en comunicaciones telefónicas de larga distancia y comunicaciones vía satélite. Una de las primeras aplicaciones de redes neuronales que tuvo éxito en la industria fue el Adaline para la eliminación de ecos en circuitos telefónicos.

Por otro lado, los filtros adaptivos se pueden usar para predecir el valor futuro de una señal a partir de su valor actual basándose en un aprendizaje en el que se emplea como entrada el valor retardado de la señal actual y como salida esperada, el valor actual de la señal, el filtro intentará minimizar el error entre su salida y la señal anterior. Una vez el filtro predice correctamente la señal actual, basándose en la señal anterior, se puede utilizar directamente la actual como entrada sin el retardo, el filtro realizará una predicción del valor futuro de la señal.

Otro ejemplo es el filtro adaptivo utilizado para modelar las respuestas de un sistema basándose en las señales de entrada, en este caso las entradas al filtro son las mismas que las del sistema, ajustando los pesos durante el aprendizaje en función de la diferencia entre su salida y la del sistema, éste será el modelo de filtro adaptivo utilizado en esta aplicación.

Existen sistemas en los cuales es posible determinar la fuente de ruido, por ejemplo cuando son los mismos elementos de medición los que introducen señales de ruido a la señal original; este ruido es ocasionado por los niveles de voltaje y de frecuencia del sistema alimentador de los aparatos, el cual es imprescindible en el proceso.

En la presente aplicación, una señal de ruido ha sido introducida por los cables de conexión de los aparatos y se ha mezclado con la señal que se desea medir, lo que en este caso es crítico, ya que no se conoce la forma exacta de la señal de interés y por tanto no es posible determinar cual debe ser la señal correcta de salida.

Aprovechando que la fuente de ruido ha sido completamente determinada, se utilizará este conocimiento para implementar un procedimiento de filtrado que por medio de un filtro adaptivo permita recuperar la señal original.

La figura 3.3.1, es un esquema de la disposición que se dará al filtro dentro del contexto general que se ha dispuesto para solucionar el problema. Las variables son:

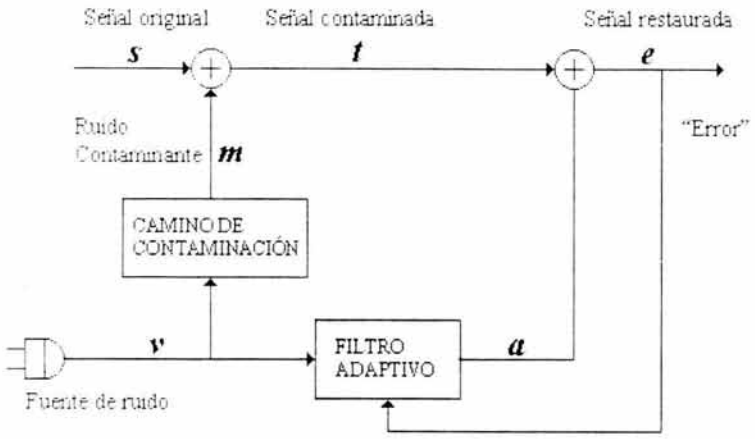


Figura 3.3.1 Diagrama de Bloques de un filtro adaptivo

s: Señal de interés, es decir es la señal que se desea medir, aunque se conoce su naturaleza, su forma es indeterminada.

v: Fuente de poder de los instrumentos de medida, por sus características de potencia y frecuencia es una fuente de ruido.

m: La señal *v*, es indispensable en el proceso de medición ya que sin ella los aparatos de medida no funcionarían; *v* es la fuente detectada de ruido, pero su contribución se limita a la parte inducida en los cables de conexión, que se convierten en el camino de contaminación; de esta forma la señal *m* es la fuente de ruido que afecta realmente la señal original *s*.

t: Señal resultante de la suma de la señal de interés *s* y de *m* que es el porcentaje efectivo de ruido que afecta la medición.

a: Salida del filtro.

e: Diferencia entre la señal contaminada *t* y la señal que entrega el filtro adaptivo, si el filtro realiza un buen trabajo, esta señal debe ser la señal restaurada sin ruido. Durante el proceso de aprendizaje la señal *e* servirá como referencia para determinar el desempeño del filtro.

La señal ***v*** es una señal por lo general de 60Hz, que en este caso alimenta el filtro adaptivo, los parámetros del filtro deben ajustarse para minimizar la señal de error ***e***, de esta forma y de acuerdo con el esquema de la figura 3.3.1 podria pensarse que la salida esperada del filtro ***a*** es la señal contaminada ***t***, pero debe recordarse que la única fuente efectiva de ruido es la señal ***m*** por lo tanto, lo que realmente se espera del filtro es que reproduzca sólo la parte de ***t*** que afecta la medición (*t*= *s*+*m*), o sea ***m***. Por consiguiente la salida del filtro ***a*** debe ser cada

vez más cercana a ***m*** para que la señal de error ***e***, se aproxime a la señal no contaminada ***s***.

La entrada al filtro es el valor de la fuente de ruido, que en este caso se asumirá como una señal senoidal uniformemente distribuida entre -0.2 y 0.2 para conservarla dentro de límites observables; la frecuencia de la onda es de 60 Hz, y la frecuencia de muestreo es de 180 Hz.

$$v(t) = 1.2\sin\left[\frac{2\pi t}{3}\right] \quad (3.3.1)$$

El estudio de los procesos de filtrado se ha basado en un análisis en el dominio de la frecuencia, determinado por las series de Fourier, así para un sistema de filtrado dado, una señal de entrada $f(t)$ que produce una salida $r(t)$ característica del sistema en el dominio del tiempo, se obtendrá un análogo en el dominio de la frecuencia tal que, $F(w)$ producirá una señal de salida $F(w)H(w)$.

En la figura 3.3.2, se observa como el sistema actúa como un filtro de las diferentes componentes de frecuencia. Para garantizar que el filtrado sea exitoso, el filtro debe atenuar igualmente todas las componentes de frecuencia, la respuesta debe ser una réplica de la señal de entrada no en magnitud, pero si en forma; en general puede haber un retraso de tiempo asociado con esta réplica, por lo tanto, se habrá filtrado exitosamente una señal $f(t)$, si la respuesta es $k * f(t - t_0)$; la respuesta es una réplica de la entrada con magnitud k veces la señal original y un retraso de t_0 segundos.

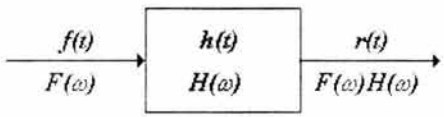


Figura 3.3.2 Representación de un sistema de filtrado en el dominio del tiempo y en el dominio frecuencial

Para el filtro adaptivo $v(t)$ es la señal de entrada al filtro y la respuesta debe ser $k * v(t - t_0)$ que en el dominio de la frecuencia, equivale a desfasar la función original 90^0 y multiplicarla por un factor k, el cual se escogió como 1/10 para simplificar el proceso de visualización.

Como se indicó anteriormente, la salida del filtro debe ser la señal m para que al final del proceso pueda obtenerse la señal restaurada, por lo tanto la forma de la señal m será:

$$m(k) = 0.12 sin\left[\frac{2\pi k}{3} + \frac{\pi}{2}\right] \quad (3.3.2)$$

Para tener una idea clara de la relación entre *m* y *v*, la figura 3.3.3 ilustra la proporción de la señal original de ruido y de la señal efectiva de ruido que afecta la medición.

En la gráfica de la parte superior de la figura 3.3.3, se ve como *v* es una señal senoidal de magnitud 1.2, mientras que la gráfica inferior muestra la señal *m* que alcanza solo la décima parte de *v* con una magnitud de 0.12 y desfasada 90^0 con respecto a la señal *v*, éstas dos ondas representan los patrones de entrenamiento de la red, siendo *v* la entrada y *m* la salida deseada.

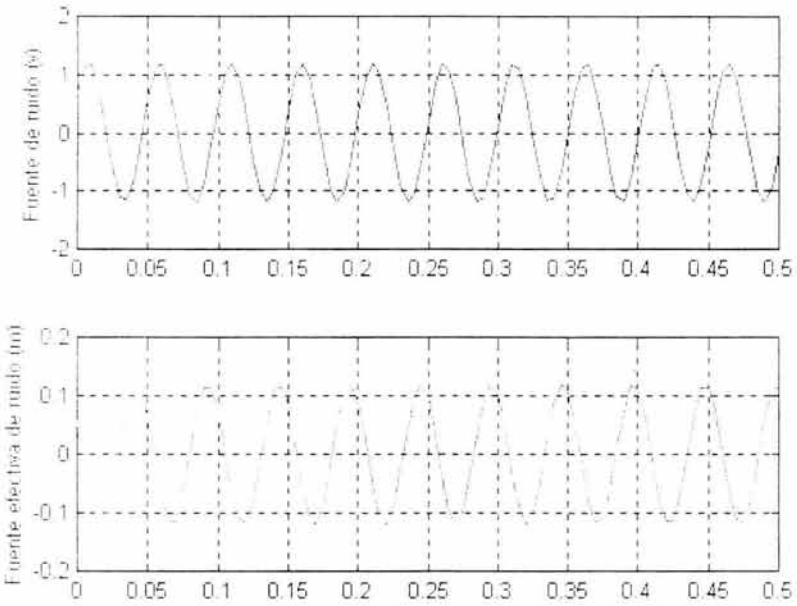


Figura 3.3.3 Señal original de ruido y señal que afecta el proceso de medición

Justificación del tipo de red

Como puede notarse, la red Adaline tiene la estructura básica del Perceptrón, la diferencia es que su función de transferencia es del tipo lineal, pero por poseer una estructura similar presenta la misma limitación de la red tipo Perceptrón de poder resolver solo problemas linealmente separables.

Falta página

N° 183

```
net.IW{1,1}=rands(1,5);
net.b{1}=[0];
p=[1 2 3 4];
```

Los valores correspondientes a la entrada y a la salida de la red se obtuvieron evaluando la ecuación (3.3.1) para valores desde $2\pi/3$ hasta 4π. La red se entrenó para 5000 iteraciones, mediante las cuales el filtro logró una excelente aproximación de la función original.

```
net.adaptParam.passes=5000;
[net,y,E,pf,af]=adapt(net,p,T,pi);
```

Para entrenar la red se generaron 101 puntos, algunos de los cuales pueden verse en la siguiente tabla:

	1	2		23		51	52		78		101
Entrada a la red (v)	0.6967	1.1345	....	1.1926	....	0.2119	0.8583	....	-1.1165	....	-0.3137
Valor esperado (m)	0.0561	-0.0159	....	-0.0653	....	0.0961	0.0365	....	0.0107	....	0.1176
Valor entregado (a)	-0.2592	-0.2379	....	-0.0385	....	0.0766	0.0293	....	0.0163	....	0.1098

W1=net.IW{1,1}

	I1	I2	I3	I4	I5
N1	-0.0405	-0.0127	-0.0319	-0.0389	-0.0097

b1=net.b{1}

N1	0.0032

Mse 0.0022

Tabla 3.3.1 Resultados del proceso de entrenamiento

Para poder juzgar el trabajo realizado por el filtro, la figura 3.3.5 muestra la señal original y la señal restaurada, que permiten comprobar las bondades del filtro adaptivo.

En la figura 3.3.5 se observa como la señal *e* (verde) reproduce la señal original *s* (naranja) logrando una excelente aproximación, a partir de la cual puede retomarse el proceso de medición, pues en este punto cualquier tratamiento que

se le haga a la señal de interés es altamente confiable, ya que ésta se encuentra libre de ruido.

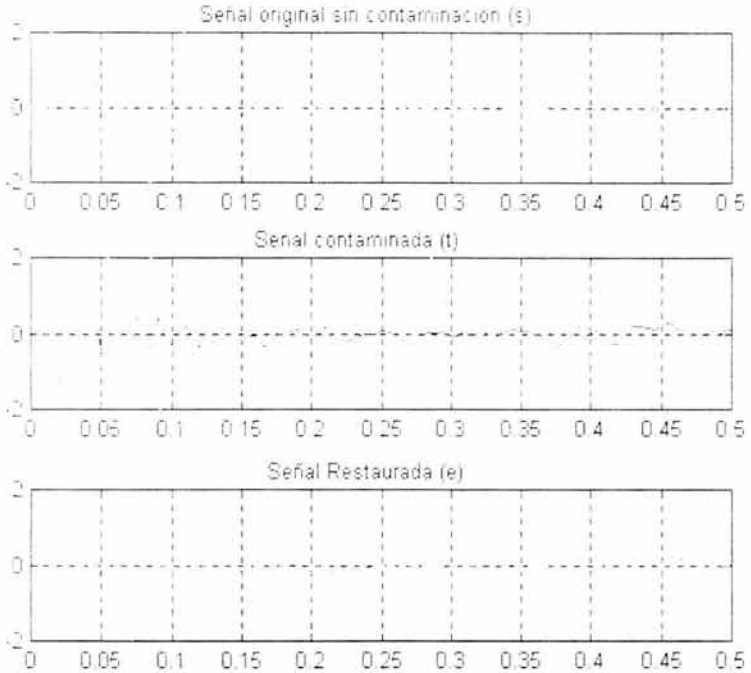


Figura 3.3.5 Señal recuperada por el filtro

Al final del proceso iterativo el máximo error entregado por la red equivale a 1.41e-07, este es un valor bastante aceptable teniendo en cuenta que el error difícilmente será cero, puesto que el algoritmo LMS emplea un valor aproximado para el gradiente, que como se dijo en el capítulo 2 recibe el nombre de gradiente instantáneo, en lugar del valor real para realizar la actualización de los pesos: este valor del gradiente es una versión distorsionada del verdadero gradiente que ocasionará que los pesos sigan variando suavemente, a pesar de que el error medio cuadrático haya alcanzado el mínimo valor posible.

Es importante diferenciar entre el valor del error del filtro adaptivo, el cual mide su desempeño y corresponde al error medio cuadrático descrito anteriormente, y el la señal de error del sistema en general, con la cual se espera reproducir la señal original sin contaminación *s*, trabajo que depende a su vez del desempeño del filtro.

En esta red todos los parámetros están muy ligados, por ejemplo en el esquema general de la figura 3.3.1 se observa como la salida *e* (correspondiente a la diferencia entre *a* la salida del filtro y *m* la señal que se esperaba que este entregará para poder reproducir la señal original *s* a la salida del sistema), realimenta el filtro adaptivo convirtiéndose en un factor decisivo en el proceso de actualización de los pesos de la red, por otro lado la dimensión del vector de

pesos tiene una influencia directa en el tiempo necesario de entrenamiento, por lo que generalmente se debe tomar un compromiso entre este aspecto y la aceptabilidad de la solución (normalmente se mejora el error aumentando el número de pesos).
El valor del parámetro α tiene una gran influencia sobre el entrenamiento. Si α es demasiado grande, es posible que la convergencia no se produzca debido a que se darán altos en torno al mínimo sin alcanzarlo. Si α es demasiado pequeño, se alcanzara convergencia pero a costa de una etapa de aprendizaje más larga.

DESCRIPCIÓN DE LAS FUNCIONES UTILIZADAS EN MATLAB

1. **Red tipo Perceptrón:** Las siguientes son las herramientas de redes neuronales del Matlab 5.3: utilizadas en el entrenamiento de las redes neuronales correspondientes a las aplicaciones del proceso de detección de obstáculos de un robot y Control de cambio de giro de un motor trifásico.

- *newp*: Crea una red tipo Perceptrón, que requiere las siguientes entradas:

  NET = NEWP(PR,S,TF,LF)
  PR : Rx2 matriz de valores máximos y mínimos para los R elementos de entrada.
  S : Número de neuronas.
  TF : Función de Transferencia, en este caso 'hardlims'.
  LF : Función de aprendizaje, para este caso 'learnp'.

- *rands*: Función simétrica que inicializa aleatoriamente los valores de pesos y ganancias de una red con valores entre -1 y 1; requiere de la estructura

  rands(S,PR), generando una matriz de dimensiones S x PR.

- *adapt:* Permite a una red neuronal adaptarse a los patrones de entrada, esta función tiene la siguiente sintaxis:

  [net,Y,E,Pf,Af] = adapt(NET,P,T,Pi,Ai)

  net : Red que va a crearse
  P : Entradas a la red; deben aparecer en forma de un arreglo de matrices.
  T : Salidas esperadas de la red, si no se especifican son ceros por defecto.
  Pi : Condiciones de retardo para la entrada inicial, por defecto son ceros.
  Ai : Condiciones de retardo para la capa inicial, por defecto ceros

- *net.adaptParam.passes*: Número de iteraciones que utiliza el programa.

2. **Red tipo Adaline:** Las siguientes son las funciones de las herramientas de Redes Neuronales del Matlab utilizadas en el entrenamiento del filtro adaptivo diseñado con base en una red Adaline.

- *newlin:* Función para crea una red tipo Adaline, que requiere las siguientes entradas:

  NEWLIN(PR,S,ID,LR)

  R: Matriz de Rx2 que contiene los valores máximos y mínimos de cada uno de los R elementos de entrada.
  S : Número de neuronas
  ID : Arreglo que contiene los valores de los retardos, por defecto

todos sus valores son cero.
LR : Rata de aprendizaje, por defecto = 0.01

- *net.inputWeights{1,1}.delays:* Especifica los retardos iniciales
- *net.adaptParam.passes*: Número máximo de iteraciones
- *[net,y,E,pf,af]=adapt(net,p,T,pi):* Comando de entrenamiento de la red; requiere como entradas la red creada anteriormente, los patrones de entrada, las salidas esperadas y los retardos iniciales, retorna el estado final de la red, los valores obtenidos para cada patrón de entrada con sus correspondientes errores así como los valores finales de los retardos.

3. **Red tipo Backpropagation:** La red neuronal Backpropagation presenta una gran variedad de opciones de configuración, dependiendo de la necesidad de aprendizaje y de la aplicación que se este desarrollando.

- *newff:* Crea una red tipo Backpropagation, requiere que le sean especificados los siguientes parámetros

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

  PR : Rx2 Matriz de valores máximos y mínimos de cada uno de las R neuronas de entrada.
  Si : Número de neuronas para cada una de las capas.
  TFi : Función de transferencia a utilizar en cada una de las capas, por defecto utiliza *tansig*
  BTF : Algoritmo de entrenamiento a utilizar, por defecto utiliza *trainlm*
  BLF : Función de actualización de los pesos, por defecto utiliza *learngdm.*
  PF : Función para evaluar el desempeño de la red, por defecto utiliza *mse.*

Los siguientes fueron los algoritmos de entrenamiento que se utilizaron en el ejemplo de control de voltaje por inyección de reactivos en una barra remota y en la aplicación de predicción de consumo de carga durante sus respectivos procesos de aprendizaje hasta que se encontró uno que brindara un aprendizaje óptimo, para cada uno de ellos utilizando la red Backpropagation:

Traingd: Algoritmo de pasos descendientes, que actualiza pesos y ganancias variándolos en la dirección negativa del gradiente de la función del error. Es un algoritmo de aprendizaje muy lento, que requiere de la siguiente sintaxis:

- *net.trainParam.epochs:* Máximo número de iteraciones para obtener convergencia
- *net.trainParam.goal*: Error máximo permitido
- *net.trainParam.lr*: Rata de aprendizaje
- *net.trainParam.max_fail:* Máximo número de fallas
- *net.trainParam.min_grad:* Mínimo rendimiento del gradiente

- *net.trainParam.show:* Intervalo de visualización de los resultados

- *net.trainParam.time:* Máximo tiempo de entrenamiento en segundos

Con este algoritmo el aprendizaje de la red se detendrá si el número de iteraciones excede el comando *net.trainParam.epochs*, si se alcanzó el valor del error propuesto como meta, si la magnitud del gradiente es menor que *net.trainParam.min_grad*, o si el tiempo d entrenamiento supera el valor de *net.trainParam.time*.

Traingdm: Equivale al algoritmo tradicional, más un nuevo coeficiente de momentum, que interviene en el proceso de actualización de los pesos. Si el error de la red en una iteración dada, excede el valor del error en la iteración anterior, en un valor mayor al definido por un radio de cobertura dado el que puede determinarse por medio de la función *max_perf_inc* y que está tipicamente alrededor de 1.04, los nuevos pesos y ganancias son descartados y el coeficiente de momentum *mc* es fijado en cero.

La sintaxis de este algoritmo es igual a la utilizada para el algoritmo traingd, más un nuevo comando que permite modificar el coeficiente de momentum

- *net.trainParam.mc:* Valor fijado para el coeficiente de momentum

Traingda: Algoritmo de Gradiente Descendiente, que emplea una rata de aprendizaje adaptiva durante el proceso de entrenamiento. La rata de aprendizaje varía entre 0.01 y 1, una rata de aprendizaje muy pequeña torna lento el aprendizaje, pero si se incrementa demasiado el aprendizaje puede tornarse inestable y crear divergencia, por esto la función *traingda* varía la rata de aprendizaje tratando de sacar provecho de la inclinación del gradiente en cada momento; su gran desventaja es que los pesos iniciales varían muy poco así se encuentren distantes de los valores de convergencia. La sintaxis de este el algoritmo es la siguiente:

- *net.trainParam.epochs*: Máximo número de iteraciones para obtener convergencia

- *net.trainParam.goal*: Error máximo permitido

- *net.trainParam.lr:* Rata de aprendizaje inicial

- *net.trainParam.lr_inc:* Porcentaje que incrementa la rata de aprendizaje cuando el error disminuye

- *net.trainParam.lr_dec:* Porcentaje en que es decrementada la rata de aprendizaje cuando el error aumenta

- *net.trainParam.max_fail:* Máximo número de fallas

- *net.trainParam.max_perf_inc:* Máximo incremento del rendimiento

- *net.trainParam.min_grad*: Mínimo rendimiento del gradiente

- *net.trainParam.show*: Los resultados son visualizados siempre que transcurre este número de iteraciones.
- *net.trainParam.time*: Máximo tiempo de entrenamiento en segundos

Trainrp: Las redes multicapa, utilizan tipicamente una función de transferencia sigmoidal (ver capítulo 1) en las capas ocultas, estas funciones comprimen un infinito rango de entradas, dentro de un finito rango de salidas, además se caracterizan porque su pendiente tendera cada vez más a cero, mientras más grande sea la entrada que se le presenta a la red, esto ocasiona problemas cuando se usa un algoritmo de entrenamiento de pasos descendientes, porque el gradiente empieza a tomar valores muy pequeños y por lo tanto no habrán cambios representativos en los pesos y las ganancias, así se encuentren bastante lejos de sus valores óptimos. El propósito del algoritmo Backpropagation Resileint (RPROP) es eliminar este efecto en la magnitud de las derivadas parciales. En este algoritmo solamente el signo de la derivada es utilizado para determinar la dirección de actualización de los parámetros, la magnitud de las derivadas no tiene efecto en la actualización. La magnitud en el cambio de cada peso es determinada por separado; el valor del incremento de pesos y ganancias es determinado por el factor *delt_inc*, así la derivada parcial del error con respecto a los pesos tenga el mismo signo durante dos iteraciones sucesivas; el valor de decremento está determinado por el factor *delt_dec* así la derivada del error con respecto a los pesos haya cambiado de signo con respecto a la anterior iteración; si la derivada es cero, entonces el valor actualizado se conserva; si los pesos continúan cambiando en la misma dirección durante varias iteraciones, la magnitud de cambios de los pesos se decrementa.
La sintaxis de este algoritmo se resume a continuación:

- *net.trainParam.epochs:* Máximo número de iteraciones del entrenamiento
- *net.trainParam.show:* Intervalo de visualización de los resultados
- *net.trainParam.goal:* Error deseado
- *net.trainParam.time=inf:* Máximo tiempo de entrenamiento en segundos
- *net.trainParam.min_grad:* Mínimo rendimiento del gradiente
- *net.trainParam.max_fail:* Máximo número de fallas
- *net.trainParam.lr*: Rata de aprendizaje
- *net.trainParam.delt_inc:* Incremento en la actualización de pesos
- *net.trainParam.delt_dec:* Decremento en la actualización de pesos
- *net.trainParam.delta0:* Incremento inicial en la actualización de pesos
- *net.trainParam.deltamax:* Máximo cambio en los pesos

Trainbfg: Algoritmo alternativo que emplea la técnica del gradiente conjugado, su expresión matemática se deriva del método de Newton, con la ventaja de que no es necesario computar las segundas derivadas; este algoritmo requiere mas capacidad de almacenamiento que el algoritmo tradional, pero generalmente converge en menos iteraciones. Requiere de un cálculo aproximado de la matriz Hessiana, la cual es de dimensiones $n^2 \times n^2$, donde n la cantidad de pesos y ganancias de la red; para redes que involucren una gran cantidad de parámetros es preferible emplear el algoritmo trainrp.

- *net.trainParam.epochs*: Máximo número de iteraciones del entrenamiento
- *net.trainParam.show:* Número de iteraciones entre las cuales se muestran resultados
- *net.trainParam.goal:* Error deseado
- *net.trainParam.time=inf:* Máximo tiempo de entrenamiento en segundos
- net.trainParam.min_grad: Mínimo rendimiento del gradiente
- net.trainParam.max_fail=5: Máximo número de fallas
- *net.trainParam.searchFcn* 'srchcha' Nombre de la rutina de búsqueda lineal a utilizar.
- *net.trainParam.scal_tol*: Se divide entre el valor de Delta para determinar la tolerancia para la búsqueda lineal.
- *net.trainParam.alpha*: Factor de escala que determina una reducción suficiente en el desempeño.
- *net.trainParam.beta*: Factor de escala que determina un tamaño de paso suficientemente grande.
- *net.trainParam.delta:* Tamaño de paso inicial en el intervalo de localización de paso.
- *net.trainParam.gama*: Parámetro para evitar pequeñas reducciones en el desempeño.
- *net.trainParam.low_lim*: Límite inferior en el cambio del tamaño del paso.
- *net.trainParam.up_lim*: Límite superior en el cambio del tamaño del paso.
- *net.trainParam.maxstep:* Máximo longitud de paso.
- *net.trainParam.minstep:* Mínima longitud de paso; por defecto es 1.0e-6
- *net.trainParam.bmax:* Máximo tamaño de paso.

Trainlm: Algoritmo que actualiza los pesos y las ganancias de acuerdo a la optimización de Levenberg-Marquardt. Es el algoritmo más rápido para redes Backpropagation; tiene la desventaja de requerir de un set de entrenamiento lo más estándar posible, pues de otra forma solo aproximará correctamente valores que se encuentren dentro de los patrones de aprendizaje. Si el set de entrenamiento es muy extenso, se recomienda reducir el Jacobiano.
La sintaxis de este algoritmo es la siguiente:

- *net.trainParam.epochs*: Máximo número de iteraciones del entrenamiento
- *net.trainParam.goal:* Error deseado
- *net.trainParam.lr:* Rata de aprendizaje
- *net.trainParam.max_fail:* Máximo número de veces que falla el valor de Mu
- *net.trainParam.mem_reduc*Factor de fraccionamiento de Jacobiano para ahorrar memoria
- *net.trainParam.min_grad:* Mínimo rendimiento del gradiente
- *net.trainParam.show:*Intervalo de visualización de los resultados.
- *net.trainParam.time:* Máximo tiempo de entrenamiento en segundos
- *tr.mu:* Valor del Mu adaptivo

Algunos **Enlaces Interesantes...**

Los siguientes son algunos enlaces interesantes en el contexto del desarrollo de las redes neuronaeles

- Tutorial de Redes Neuronales desarrollado en la Universidad Politécnica de Madrid UPM (España), bajo la dirección del Dr. Diego Andina de la Fuente
  Contiene una explicación de los distintos tipos de redes segun su forma de aprendizaje y permite descargar un simulador para entrenar una red tipo backpropagation
  http://www.gc.ssr.upm.es/inves/neural/ann2/anntutor.htm

- Applets en Java que permiten de forma interactiva entrenar una Red Neuronal Artificial, una Red tipo Perceptron de una sola y bvarias capas; el codigo fuente es abierto con fines academicos
  Proyecto de grado de Fred Corbett de la universidad de Manitoba, Canada
  http://home.cc.umanitoba.ca/~umcorbe9/mlp.html

- Aplicación en Java de una Red Neuronal para aproximar distintas funciones
  http://neuron.eng.wayne.edu/bpFunctionApprox/bpFunctionApprox.html

- Visión Artificial y Visión Humana, aplicación a la percepción Visual en Robotica
  Grupo de investigacion dirigido por el profesor Luis Jañez Esacalada's
  http://sirio.psi.ucm.es/PROYECTOS/VISIONROBOT/vavh.html

- Redes Neuronales aplicadas al reconocimiento de voz. Proyecto desarrollado por José María Garcia Jiménez en Madrid, España
  http://electronica.com.mx/neural/aplicaciones/index.html

- Control de procesos mediante Redes Neuronales
  Presentado por Cristina Garrido de la Universidad de Concepción para optar al título de Ingeniero Civil Químico
  http://www.diq.udec.cl/~cgarrido/
  http://melquiades.diq.udec.cl/~cgarrido/

- Introducción a las Redes Neuronales Artificiales.

- Desarrollado por Alfredo Catalina Gallego
  http://www.gui.uva.es/login/13/redesn.html

- Sistemas Neurodifusos: Proyectos finales curso Redes Neuronales y Logica Difusa, Universidad del Valle, Colombia.
  http://maxwell.univalle.edu.co/proyectos/rna/neurofuzzy1/
  http://maxwell.univalle.edu.co/proyectos/rna/neurofuzzy2/

- Redes de Resonancia Adaptiva (ART), arquitectura, entrenamiento y aplicaciones de esta red

- Proyectos finales curso Redes Neuronales, Universidad del Valle, Colombia
  http://maxwell.univalle.edu.co/proyectos/rna/ART1/
  http://maxwell.univalle.edu.co/proyectos/rna/ART2/
  http://maxwell.univalle.edu.co/proyectos/rna/ART4/

- Pagina de la Universidad de Chile contiene manual sobre el Snns
  http://cipres.cec.uchile.cl/~em753/

- Tutorial con teoria detallada
  http://www.paisvirtual.com/informatica/navegadores/toniomos/rn/temas-

rn.htm

- Tutorial Redes Neuronales contiene Teoria General
  http://www.gc.ssr.upm.es/inves/neural/ann2/anntutorial.html

- Red de Hopfield como reconoimiento de patrones
  http://www.eeng.dcu.ie/~annet/intro/demo.html

Enlace Principal http://ohm.utp.edu.co/neuronales/

BIBLIOGRAFÍA

1. BARAN M.E, WU F.F. "Network reconfiguration in distribution sistems for loss reduction and load balancing". IEEE. Transaction on power delivery, Vol 4, # 2, April 1989.
2. CHAPMAN Stephen J. "Máquinas Eléctricas". McGraw Hill. Santafé de Bogotá Colombia. 1997
3. DELGADO Alberto. "Inteligencia Artificial y Minirobots". Bogota D.C. Colombia. Ecoe Ediciones. Julio 1998.
4. DELGADO Alberto. "Propiedades Matemáticas y Aplicaciones de las Redes Neuronales Dinámicas Recurrentes". Santafé de Bogotá Colombia.
5. FISCHBACH Gerald D.. "Mente y cerebro". Investigación y Ciencia # 194, Noviembre de 1992
6. FLOREZ Oscar y SALAZAR Harold. Proyecto de grado "Utilización de Redes Neuronales Artificiales en la Reconfiguración de Alimentadores Primarios". Pereira Colombia. Universidad Tecnológica. 1998.
7. FREEMAN James y SKAPURA David. "Redes Neuronales: Algoritmos, aplicaciones y técnicas de programación". Delaware E.U.A. Addison Wesley Iberoamericana S.A. 1993
8. GALLEGO Ramón y Ruben. "Flujo de Carga". Pereira Colombia. Universidad Tecnológica de Pereira. 1999
9. GIRALDO Didier, TABARES Ivan. "Programa que entrena neuronas para implementar funciones lógicas". Scientia et technica. Año II, #5, Junio de 1997.
10. HAGAN Martin, DEMUTH Howard y BEALE Mark. "Neural Network Design". PWS Publishing Company. Boston U.S.A. 1996.
11. HILERA José R., MARTINEZ Víctor J. "Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones ". Ra-ma Editorial. Madrid. 1995
12. HINTON Geoffrey E. "Redes neuronales que aprenden de la experiencia". Investigación y ciencia. #194, Noviembre de 1992.
13. KANDEL Eric R., ROBERT Hawkins. "Bases biológicas del aprendizaje y de la individualidad". Investigación y ciencia #194, Noviembre de 1992.
14. KOHONEN Tuevo. "Associative memory. A system theoretical approach". Springer-Verlag, 1977
15. KOHONEN Tuevo. "Self-organized formaion of topologically correct feature maps". Biological Cybernetics #43 , 1982. Reimpreso en el texto "Neurocomputing" (J.Anderson y E. Rosenfeld ed.), MIT press,1988.
16. KOHONEN Tuevo. "Learning Vector Quantization". Abstracts of the first annual INNS Meeting, #308, 1988.
17. KONONEN Tuevo. "Self-organization and associative memory (3ª ed.)". Springer-Verlag, 1989.
18. KOSKO. "Bidirectional Associative Memories". IEEE Transactions on system, Man & Cybernetics, #18, pags 42-60,1988
19. LATHI B.P. "Introducción a la teoría y sistemas de comunicación". Limusa editores. Méjico. 1995
20. M. Minsky y S. Papert. "Perceptrons". ED. MIT Press,1969

21. MURILLO José Joaquín. "Sistemas de distribución de energía eléctrica". Pereira, Colombia. 1997.
22. SIMPSON P.K. "Foundations of neural Networks". Artificial Neural Networks. IEEE PRESS. New York. 1992
23. WIDROW Bernard, LEHR Michael A. "30 years of adaptive neural networks: Perceptron, Madaline, and Backpropagation". Proceedings of the IEEE, vol 78 #9, September 1990, pp 1415-1442.

BIBLIOGRAFIA (robot)

1. [1] AAAI Presidential Adress, August 1988 Turing Award Presentation, March 1995.
2. The challenge of Artificial Intelligence pp 86-91.

3. [2] IEEE Transaction on patter Analysis and Machine Intelligence. Vol. 19, num 10 Octuber 1997. Computer Society.

4. [3] Artificial Neural Networks, Paradigms Applications, and hardware Implementations.
5. J. G. Taylor. A Silicon Model of vertebrate Retinal processing, pp. 436-443.

6. [4] Artificial Neural Networks, Paradigms Applications, and hardware Implementations.
7. Gail A. Carpenter. Neural Network Models for Pattern Recognition and Associetive Memory. pp 138-152.

8. AMAT, BASAÑEZ y TORRAS: Sistemas de Visión Tridimensional en los robots industriales, revista Mundo Electrónico, Número 130.
9. ANGULO, J. M.: Robótica Práctica, tecnología y aplicaciones, Editorial Paraninfo, Madrid, 1986.
10. DORF, R. C. (editor en jefe): International Encyclopedia of Robotics, John Wiley & Sons, 1988
11. DOUGHERTY, E. R. y Ch. R. GIARDINA: Mathematical Methods for Artificial Intelligence and Autonomous Systems, Prentice-Hall International Editions, Englewood Cliffs, 1988
12. ESCUDERO, L. F.: Reconocimiento de Patrones, editorial Paraninfo, Madrid, 1977
13. FU, K. S.: Syntactic Pattern Recognition and Applications, Prentice-Hall, Englewood Cliffs, 1982.
14. FU, K. S., R. C. GONZALEZ y C. S. G. LEE: Robótica: Detección, Visión e Inteligencia, McGraw-Hill/Interamericana de España S. A., Madrid, 1988.
15. IÑIGO, R. y J. M. ANGULO: Visión por computador y su aplicación en la robótica, revista Mundo Electrónico, octubre 1984.

16. KANADE, T.: Geometrical Aspects of Interpreting Images as a Three-Dimensional Scene, revista Proceedings of the IEEE, julio 1983, pags. 789-802.
17. KELLEY, R. B. y otros: Three Vision Algorithms for Acquiring Workpieces from Bins, revista Proceedings of the IEEE, julio 1983, pags. 803-820.
18. McPHERSON, C. y E. HALL: Three-Dimensional Robot Vision Techniques, en Recent Advances in Robotics, G. Beni (editor), pag. 263-312, Wiley-Interscience, John Wiley & Sons, 1985
19. SERRA, J.: Image Analisys and Mathematical Morphology, Academic Press, 1982
20. WINSTON, P. H. (editor) y otros: The Psychology of computar vision, McGraw-Hill Computer Science Series, 1975
21. ZUCKER, S. W.: Computer Vision for Robotics, a tutorial introduction. Conferencia tutorial del congreso AAAI-82. American Association for Artificial Intelligence. 1982.
22. ttp://luisguillermo.com/visiona.htm