

2. PRINCIPALES TIPOS DE REDES NEURONALES

2.1 PERCEPTRÓN

2.1.1 Antecedentes. La primera red neuronal conocida, fue desarrollada en 1943 por Warren McCulloch y Walter Pitts; ésta 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 llamó, era un dispositivo que respondía a señales ópticas; como se muestra en la 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.



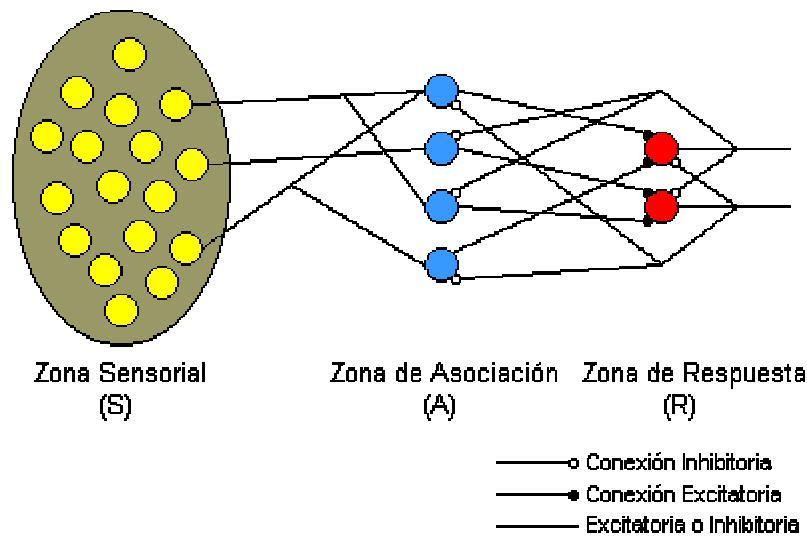


Figura 2.1.1 Modelo del Fotoperceptrón de Rosenblatt

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.

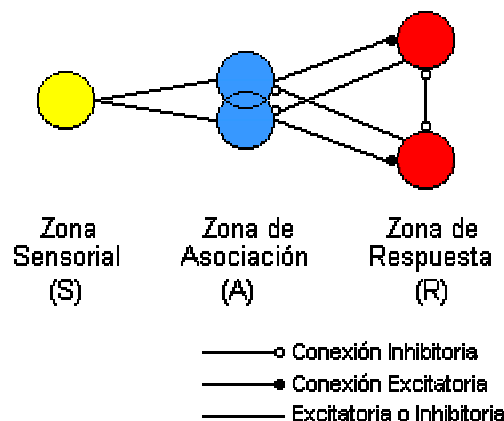


Figura 2.1.2 Esquema de conexiones de un Perceptrón sencillo



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"[20], el cual para muchos significó el final de las redes neuronales. En él 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 probabilística 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

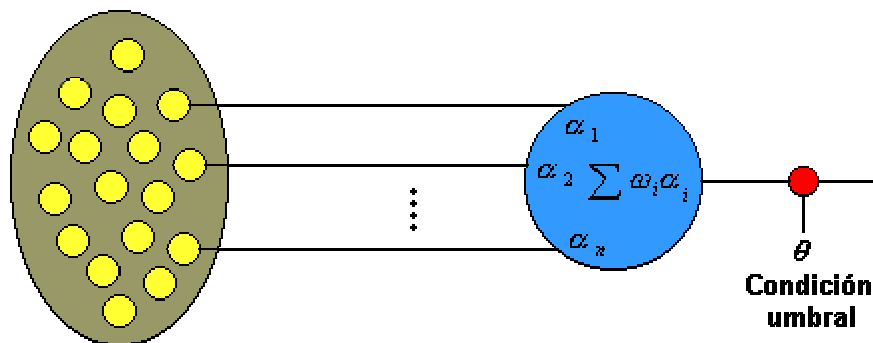


Figura 2.1.3 Perceptrón según Minsky y Papert

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



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

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.

2.1.2 Estructura de la red.

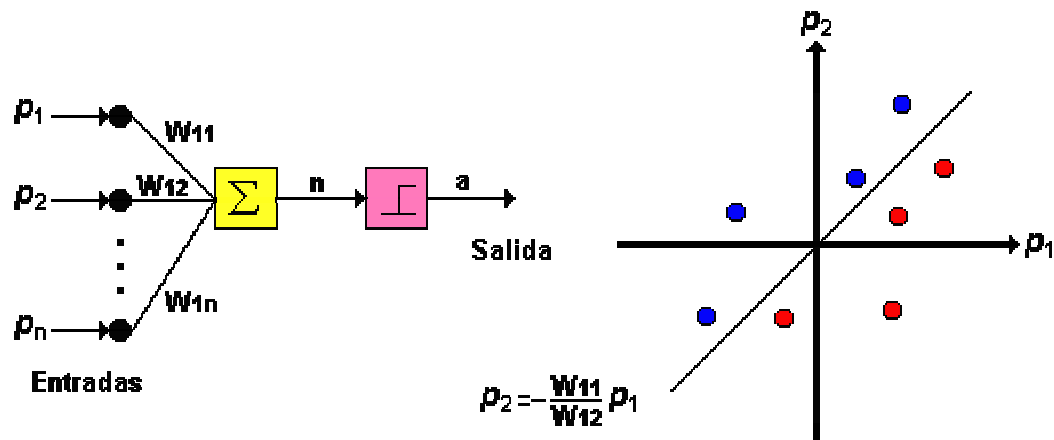


Fig. 2.1.4 Perceptrón

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 de 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.



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

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

$$a_i = \text{hardlim}(n_i) = \text{hardlim}(w_i^T p_i) \quad (2.1.4)$$

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.

2.1.3 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:

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

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:

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

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



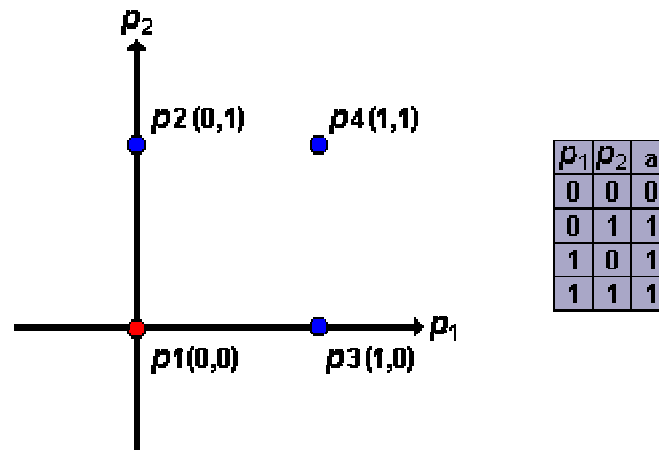


Figura 2.1.5 Función OR

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

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

Si $w_1 p_1 + w_2 p_2$ 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.

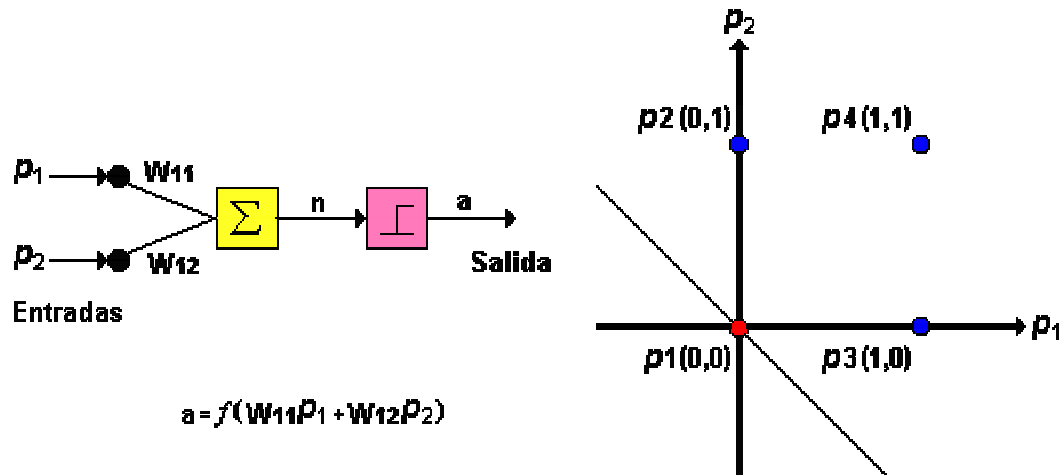


Figura 2.1.6 Perceptrón aplicado a la función OR

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

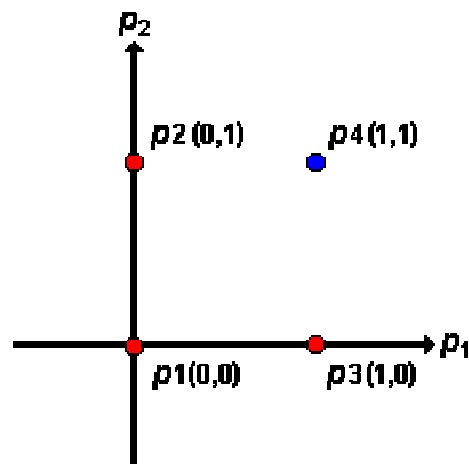


Figura 2.1.7 Espacio de salida de una compuerta AND

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

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

Las soluciones obtenidas para la función AND y la OR, se ven en la figura 2.1.8

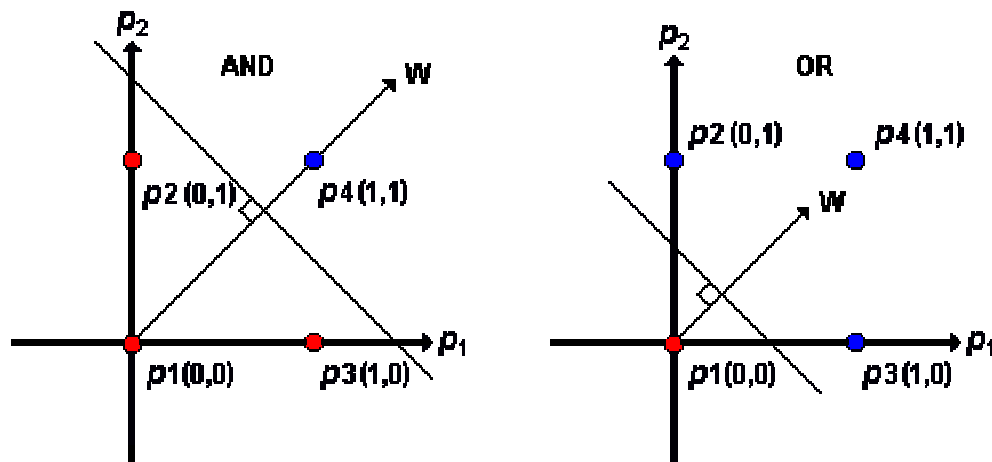


Figura 2.1.8 Solución para una función AND y una OR

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 salidas esperadas para cada uno 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

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

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 asintóticamente 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:

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

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

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

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:

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

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

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

En una sola expresión la ley puede resumirse así:

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



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

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

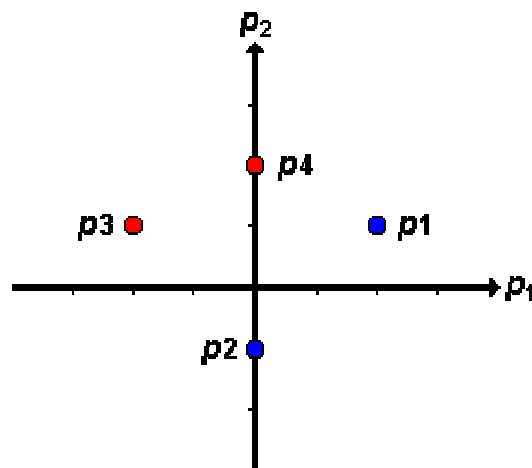


Figura 2.1.9 Patrones de entrenamiento

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.



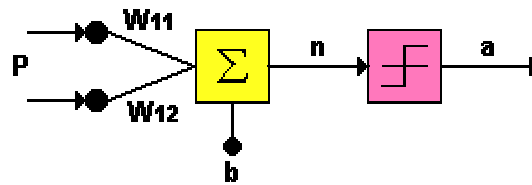


Figura 2.1.10 Red Perceptrón que resolverá el problema de clasificación de patrones

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:

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

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

- Iteración 0



La red clasificará 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: $\frac{-b}{W_{11}} = -2.5$ $\frac{-b}{W_{21}} = 0.71$

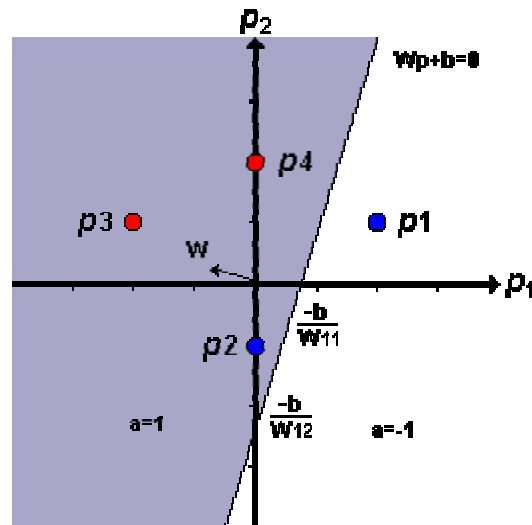


Figura 2.1.11 Clasificación de los patrones de acuerdo a la iteración 0

Como puede verse, la característica de decisión es ortogonal al vector de pesos W . La red clasifica incorrectamente los patrones $p1$, $p3$ y $p4$; en esta iteración; a continuación presentamos a la red el patrón de entrenamiento $p1$.

- Iteración 1

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

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

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



De la iteración 0 $p1$ 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: $\frac{-b}{W_{11}} = -0.75$ $\frac{-b}{W_{21}} = -1.13$

Como se observa el patrón de entrenamiento $p1$ ha sido clasificado correctamente, y casualmente los patrones $p2$ y $p3$ fueron correctamente ubicados, pues aún no han sido presentados a la red.

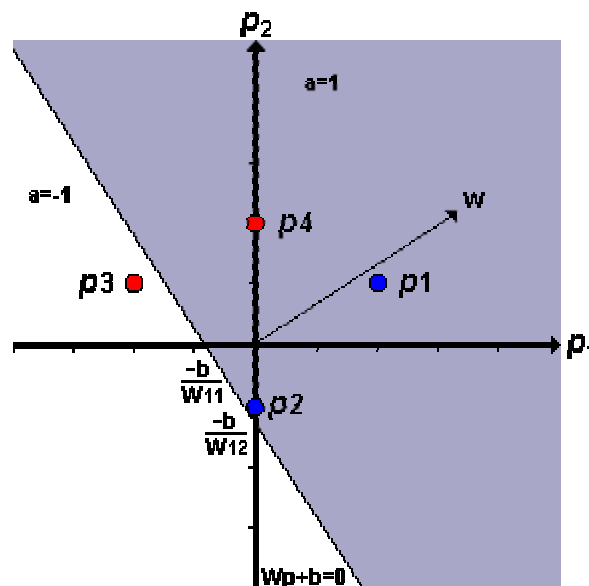


Figura 2.1.12 Característica de decisión de la iteración 1



- Iteración 2

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

$$W^1 = \begin{bmatrix} 3.3 & 2.2 \end{bmatrix} \quad b^1 = [2.5]$$

$$a = \text{hardlims} \left(\begin{bmatrix} 3.3 & 2.2 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \end{bmatrix} + [2.5] \right) \quad a = \text{hardlims}(0.3) = 1$$

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

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

$$W^2 = W^1 + ep^T \quad W^2 = \begin{bmatrix} 3.3 & 2.2 \end{bmatrix} + 0 \begin{bmatrix} 0 & -1 \end{bmatrix} = \begin{bmatrix} 3.3 & 2.2 \end{bmatrix}$$

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

- Iteración 3

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

$$W^2 = \begin{bmatrix} 3.3 & 2.2 \end{bmatrix} \quad b^2 = [2.5]$$



$$a = \text{hardlims}\left([3.3 \quad 2.2] \begin{bmatrix} -2 \\ 1 \end{bmatrix} + [2.5]\right) \quad a = \text{hardlims}(-1.9) = -1$$

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

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

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

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

- Iteración 4

Se presenta a la red $p4$,

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

$$a = \text{hardlims}\left([3.3 \quad 2.2] \begin{bmatrix} 0 \\ 2 \end{bmatrix} + [2.5]\right) \quad a = \text{hardlims}(6.9) = 1$$

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

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

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

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



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

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

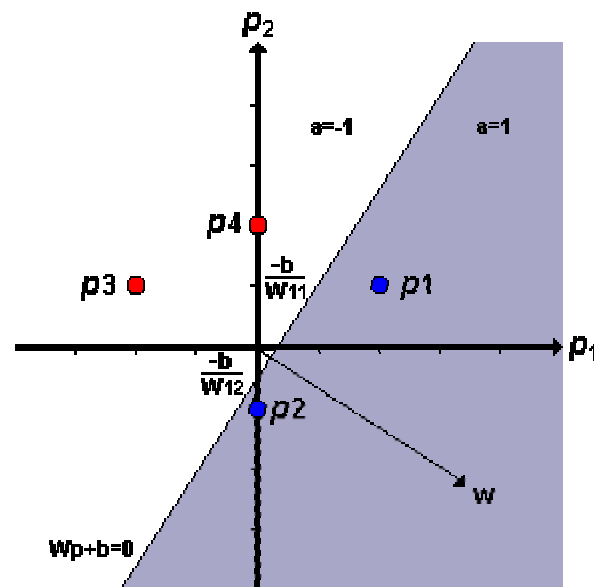


Figura 2.1.13 Característica de decisión final

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.

2.1.4 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 dividido 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

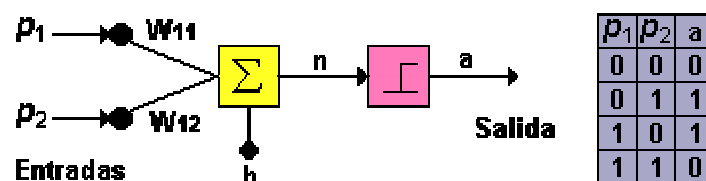


Figura 2.1.14 Compuerta XOR



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

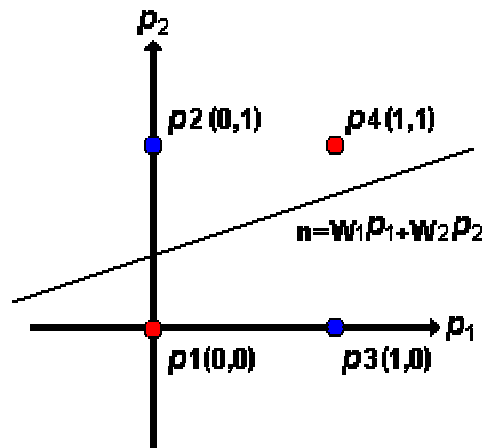


Figura 2.1.15 Plano formado por el problema de la XOR

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 está 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 estancó 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:

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

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

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



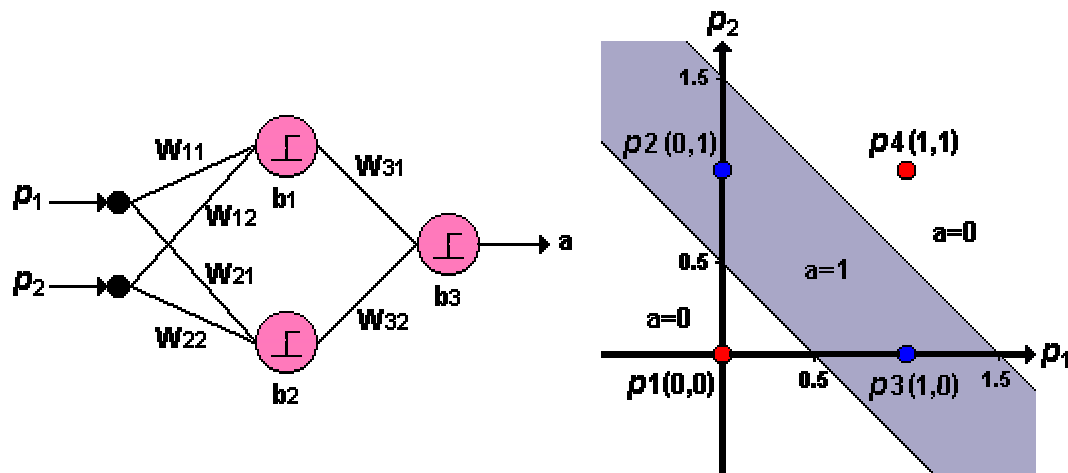


Figura 2.1.16 Perceptrón multicapa para la XOR

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

$$\begin{array}{ll} w_{11}=1 & w_{12}=1 \\ w_{21}=1 & w_{22}=1 \\ w_{31}=1 & w_{32}=-1.5 \\ b_1=0.5 & b_2=1.5 & b_3=0.5 \end{array}$$

2.1.5 Perceptrón 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.

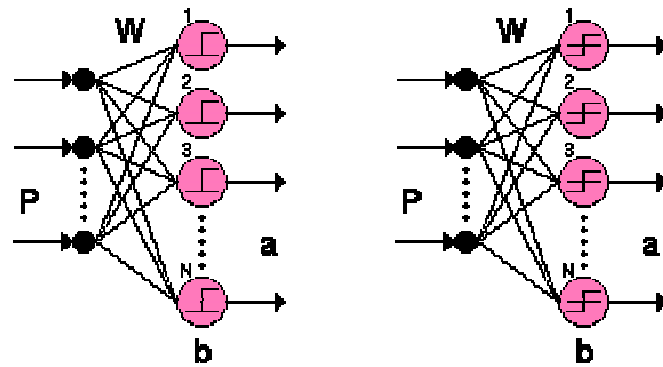


Figura 2.1.17 Conexiones del Perceptrón

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

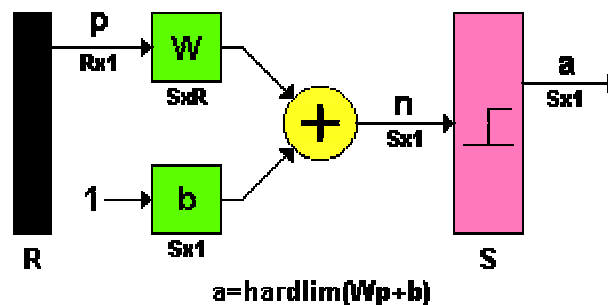


Figura 2.1.18 Notación compacta para la red tipo Perceptrón



La salida de la red está dada por:

$$a = \text{hardlim}(\mathbf{W} * \mathbf{p} + \mathbf{b}) \quad (2.1.20)$$

Donde

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

p: Vector de entradas a la red de dimensiones $R \times 1$

b: Vector de ganancias de la red de dimensiones $S \times 1$

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 [11]. 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.



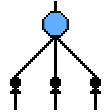
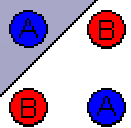
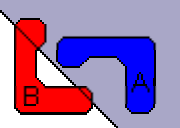

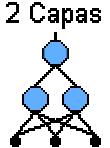
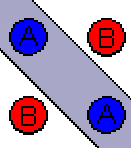
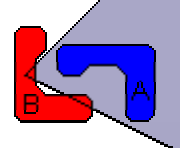
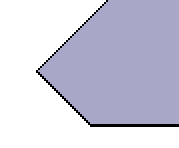
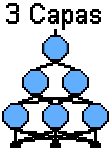
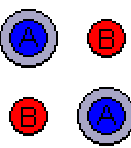
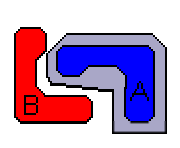
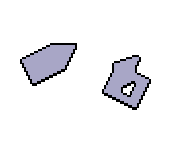
Estructura	Regiones de Decisión	Problema de la XOR	Clases con Regiones Mezcladas	Formas de Regiones más Generales
1 Capa 	Medio Plano Limitado por un Hiperplano			
2 Capas 	Regiones Cerradas o Convexas			
3 Capas 	Complejidad Arbitraria Limitada por el Número de Neuronas			

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 compuestas 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)

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

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



2.2 ADALINE

2.2.1 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 sólo 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, característica que lo hace bastante práctico en las aplicaciones de procesamiento de señales digitales, por ejemplo las líneas telefónicas de gran distancia utilizan la red Adaline para cancelar el ruido inherente a su recorrido.

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

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

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 **L**ineal 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



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

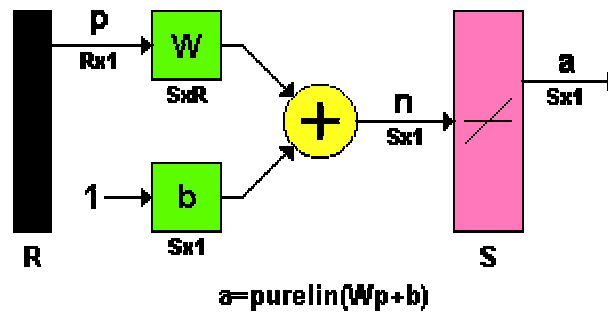


Figura 2.2.1 Estructura de una red Adaline

La salida de la red está dada por:

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

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

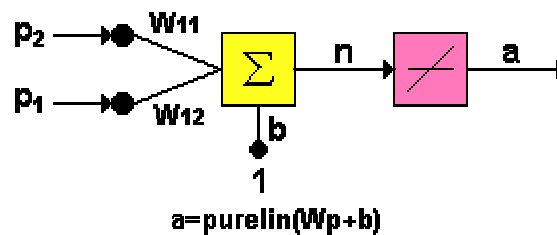


Figura 2.2.2 Adaline de una neurona y dos entradas



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

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

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

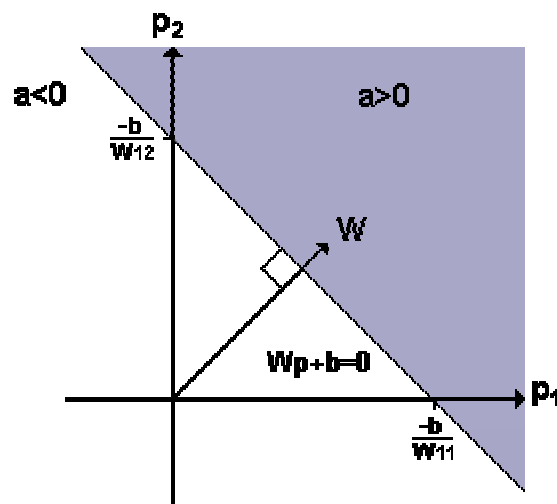


Figura 2.2.3. Característica de decisión de una red tipo Adaline

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.

2.2.3 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:



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

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:

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

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

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

La variación del error en cada iteración es representada por



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

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

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

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 α mientras 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 α controla la estabilidad y velocidad de la convergencia del proceso de entrenamiento; 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 α que varíen entre

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

Si se fija α en 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 α no 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 ± 1 todos 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



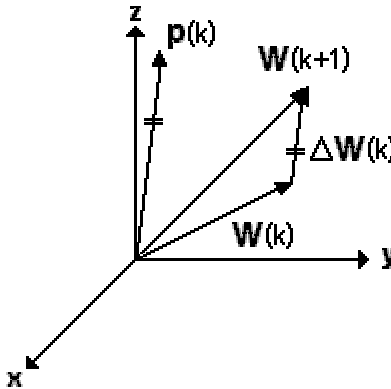


Figura 2.2.4 Actualización de pesos del algoritmo LMS

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

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 $\Delta \mathbf{W}(k)$ y $\Delta \mathbf{b}(k)$ 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 de una sola neurona y se empleará un algoritmo de pasos descendientes aproximado, como el que utilizaron Widrow y Hoff. La función de error es una función matemática definida en el espacio de pesos multidimensional para un conjunto de patrones dados, es una superficie que tendrá muchos mínimos (globales y locales) y la regla de aprendizaje va a buscar el punto en el espacio de pesos donde se encuentra el mínimo global de esa superficie; aunque la superficie de error es desconocida, el método de gradiente descendiente consigue obtener información local de dicha superficie a través del gradiente, con esa información se decide qué dirección tomar para llegar hasta el mínimo global de dicha superficie.

Con este algoritmo calculando el gradiente en cada iteración (gradiente instantáneo) y no el gradiente sobre el error total después de haber presentado todos los patrones, 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:

$$[\nabla e^2(k)]_j = \frac{\partial e^2(k)}{\partial w_{i,j}} = 2e(k) \frac{\partial e(k)}{\partial w_{1,j}} \text{ para } j=1,2,\dots,R \quad (2.2.14)$$

y

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

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 $w_{i,j}$:

$$\begin{aligned} \frac{\partial e(k)}{\partial w_{i,j}} &= \frac{\partial [t(k) - (\mathbf{w}^T * \mathbf{p}(k) + b)]}{\partial w_{i,j}} \\ &= \frac{\partial \left[t(k) - \left[\sum_{i=1}^R w_{1,i} p_i(k) + b \right] \right]}{\partial w_{i,j}} \end{aligned} \quad (2.2.16)$$



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

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

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

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

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.

Esta aproximación de $\nabla e(k)$, puede ser usada en el algoritmo de pasos descendientes tal como aparece en la ecuación (2.2.5) para darle forma final a la actualización de pesos y ganancias del algoritmo LMS de las ecuaciones (2.2.14) y (2.2.15)

$$\mathbf{w}(k+1) = \mathbf{w}(k) + 2\alpha e(k) \mathbf{p}(k) \quad (2.2.19)$$

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



La tasa 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:

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

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

Nótese que ahora el error e y la ganancia \mathbf{b} son vectores

2.2.4 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 está haciendo es, modificar las características de respuesta en frecuencia de un filtro *pasa banda* que está dentro de la radio, este filtro sólo deja pasar las señales procedentes de la emisora en la que se esté interesado y elimina todas las demás señales que estén siendo transmitidas dentro del espectro AM.

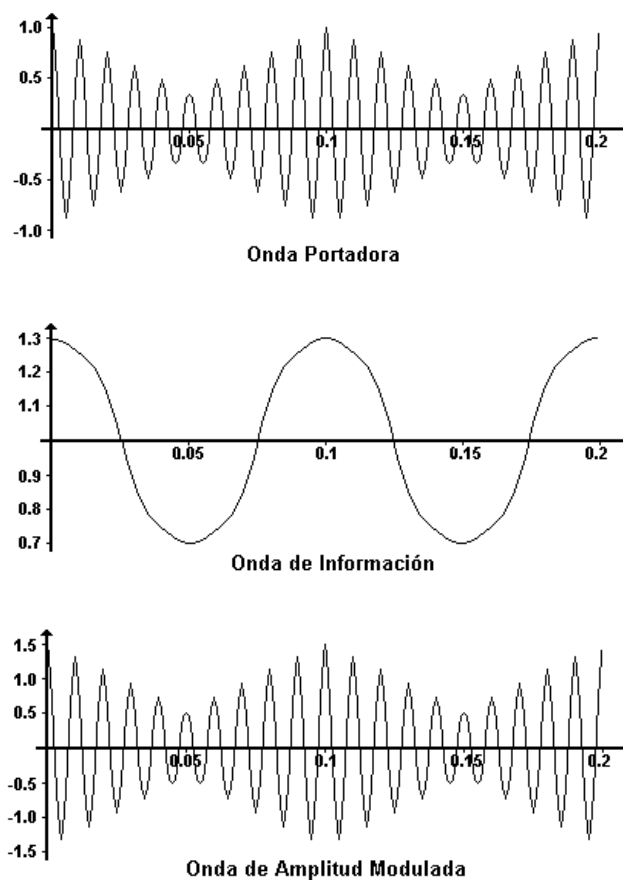


Figura 2.2.5 Técnicas de codificación de información y modulación en amplitud



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:

$$x(t) = \sum_{n=0}^{\infty} a_n \cos(2\pi n f_0 t) + \sum_{n=1}^{\infty} b_n \sin(2\pi n f_0 t) \quad (2.2.23)$$

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

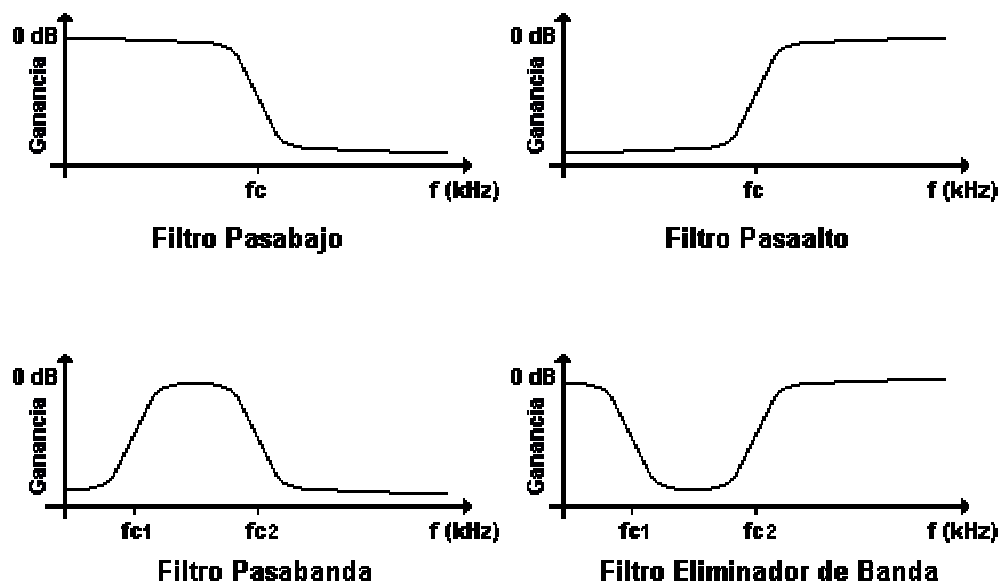


Figura 2.2.6 Características de los cuatro filtros básicos



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 cálculo para determinar cuánta 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 cuáles son los detalles de las características de las señales; si se necesitarán 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

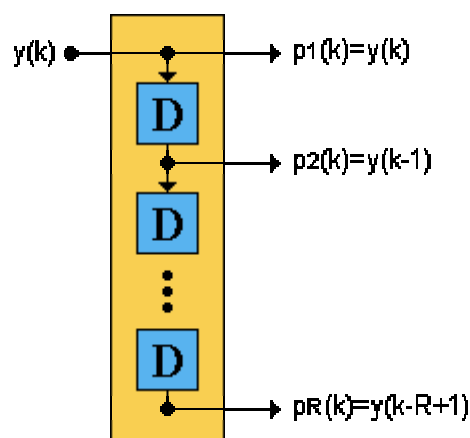


Figura 2.2.7 Retardos en línea



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

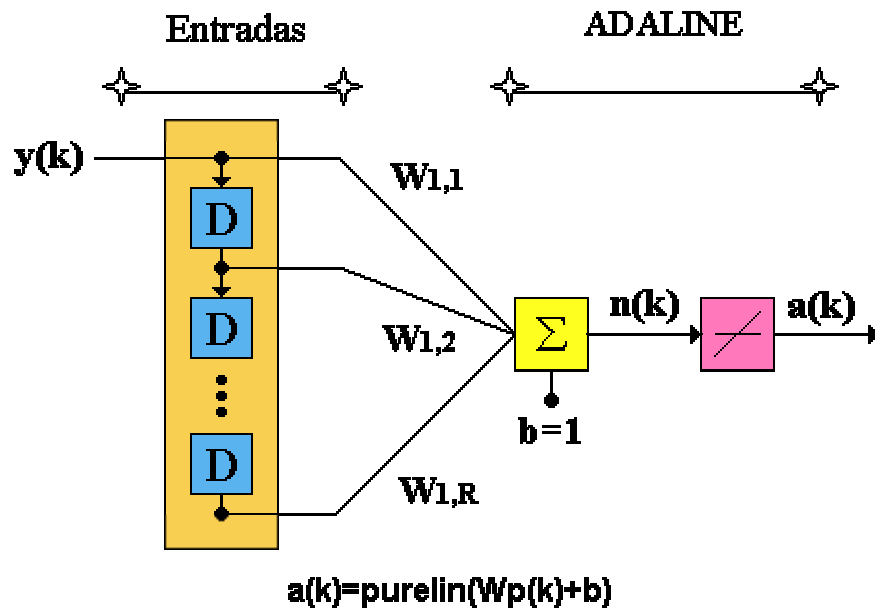


Figura 2.2.8 Filtro adaptivo

Cuya salida está dada por:

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



2.3 BACKPROPAGATION

2.3.1 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 sólo pueden resolver problemas linealmente separables, fue esto lo que llevó 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, éste se desarrolló en un contexto general, para cualquier tipo de redes, siendo las redes neuronales una aplicación especial, razón por la cual el algoritmo no fue aceptado dentro de la comunidad de desarrolladores de redes neuronales. Fue sólo 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 psicólogos David Rumelhart y James McClelland. La publicación de éste 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 esté 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 cómputo 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 sólo 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 tendría 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, éste 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 sólo 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.

2.3.2 Estructura de la Red. La estructura típica de una red multicapa se observa en la figura 2.3.1

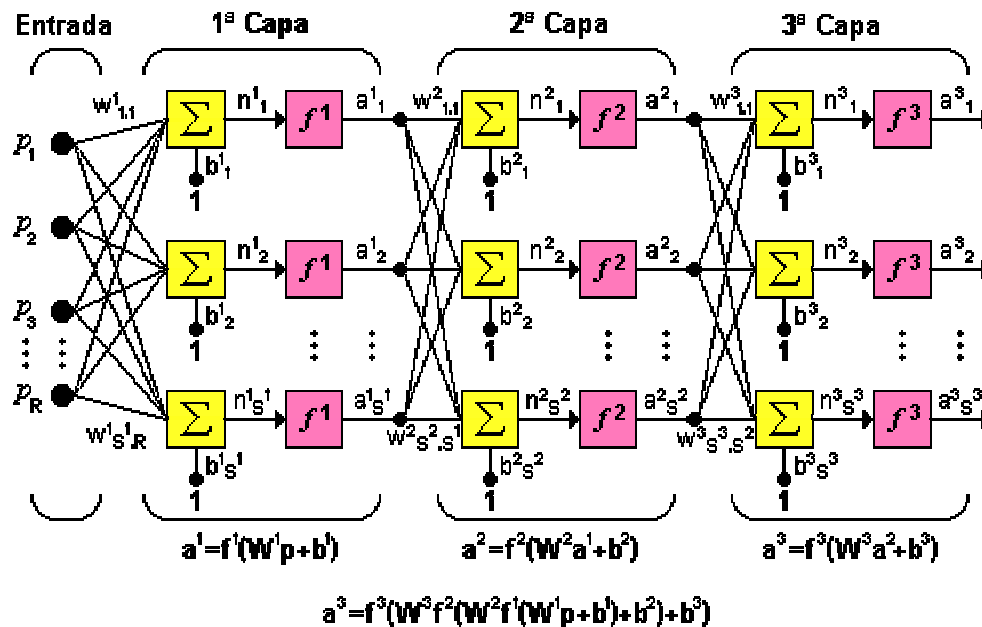


Figura 2.3.1 Red de tres capas

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

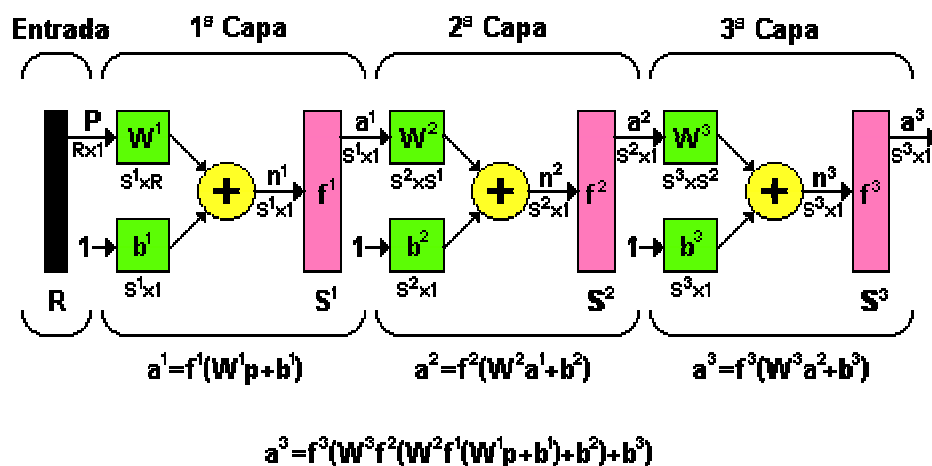


Figura 2.3.2 Notación compacta de una red de tres capas



2.3.3 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:

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

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 específico, 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.

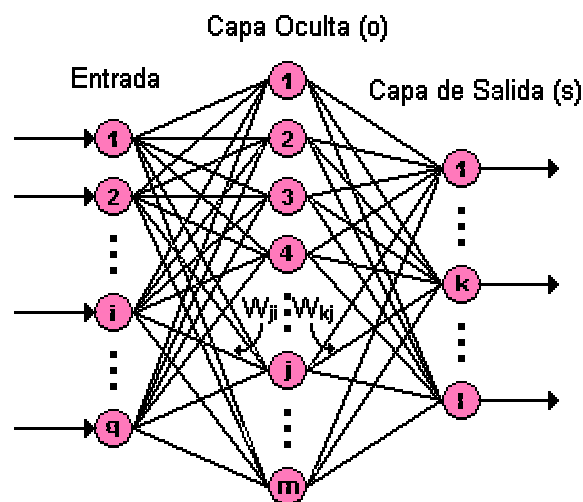


Figura 2.3.3 Disposición de una red sencilla de 3 capas



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_{ji}^o : Peso que une la componente i de la entrada con la neurona j de primera capa oculta

p_i : Componente i del vector p que contiene el patrón de entrenamiento de q componentes

b_j^o : 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_j^o que está 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_j^o de las neuronas de la capa oculta (de l componentes) son las entradas a los pesos de conexión de la capa de salida, $a_k^o \Rightarrow n_k^s$ 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_{kj}^s : 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_j^o : Salida de la neurona j de la capa oculta, la cual cuenta con m neuronas.

b_k^s : Ganancia de la neurona k de la capa de salida.

n_k^s : Entrada neta a la neurona k de la capa de salida

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

$$a_k^s = f^s(n_k^s) \quad (2.3.7)$$

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

$$a_k^s = f^s\left(\sum_{j=1}^m W_{kj}^s a_j^o + b_k^s\right) \quad (2.3.8)$$

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

$$\delta_k = (t_k - a_k^s) \quad (2.3.9)$$



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

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

ep^2 : Error medio cuadrático para cada patrón de entrada p

δ_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 denotará 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:

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

$-\frac{\partial ep^2}{\partial W_{kj}^s}$: 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

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

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_k^s está explícitamente en función de n_k^s y de la ecuación (2.3.6)



puede verse que n_k^s esta explícitamente en función de W_{kj}^s , 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,

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

$\frac{\partial n_k^s}{\partial W_{kj}^s}$: 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 última 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



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

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 $f'^s(n_k^s)$ es requerido para el cálculo del error.

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

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

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

$$\text{purelin: } f(n) = n \quad f'(n) = 1 \quad (2.3.19)$$

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 sensibilidad de la capa de salida.

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



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:

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

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:



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

Tomando las derivadas 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

$$-\frac{\partial ep^2}{\partial W_{ji}^o} = \sum_{k=1}^l (t_k - a_k^s) \times f'^s(n_k^s) \times W_{kj}^s \times f'^o(n_j^o) \times p_i \quad (2.3.24)$$

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

$$-\frac{\partial ep^2}{\partial W_{ji}^o} = \sum_{k=1}^l \delta_k^s \times W_{kj}^s \times f'^o(n_j^o) \times p_i \quad (2.3.25)$$

Los términos del error para cada neurona de la capa oculta están dados por la ecuación (2.3.26), este término también se denomina sensibilidad de la capa oculta

$$\delta_j^o = f'^o(n_j^o) \times \sum_{k=1}^l \delta_k^s W_{kj}^s \quad (2.3.26)$$



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 está 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 la 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 [32], la cual es una extensión de la regla delta desarrollada por Widrow [34] en 1930

Algunos autores denotan las sensibilidades de las capas 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)

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

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

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

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 tasa 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 α significa 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 α a 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.

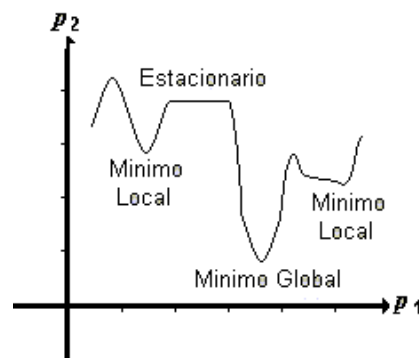


Figura 2.3.4 Superficie típica de error



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 estos términos, utilizamos el algoritmo Backpropagation, para aproximar la siguiente función:

$$t = \sin \frac{\pi}{4} p \quad \text{para el intervalo } -2 \leq p \leq 2 \quad (2.3.33)$$

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

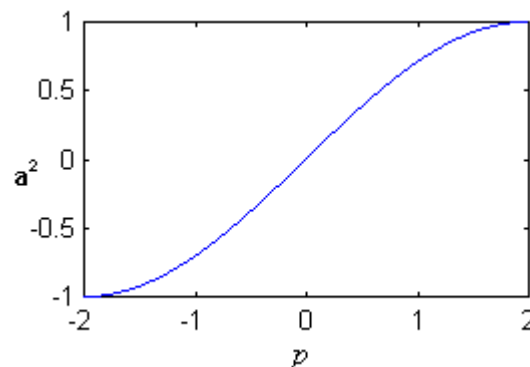


Figura 2.3.5 Intervalo de la función t



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

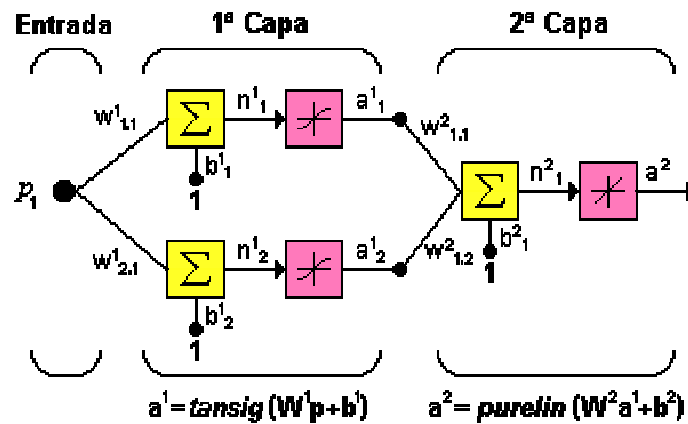


Figura 2.3.6 Red utilizada para aproximar la función

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

$$a^l = \text{tansig}(W^l p^T + b) \quad (2.3.34)$$

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 siempre por la función de transferencia *purelin*, la cual reproduce exactamente el valor resultante después de la sumatoria.

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

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:



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

Para el proceso de cálculo, se le presenta a la red el parámetro p_1 , de esta forma la primera iteración es como sigue

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

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

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

Como se esperaba la primera iteración no ha sido suficiente, para aproximar la función correctamente, así que se calculará la sensibilidad 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:

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

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

Y las sensibilidades, 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 [15] 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 se encuentran dos métodos heurísticos y dos métodos basados en algoritmos de optimización numérica.



Dentro de los métodos heurísticos tenemos:

2.3.3.1 Red Backpropagation con momentum [30]. 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

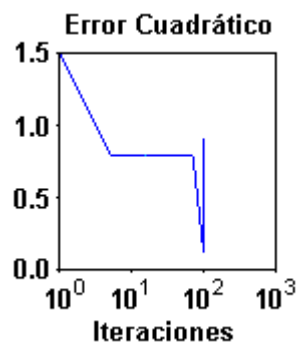


Figura 2.3.7 Comportamiento típico del proceso de convergencia para una red Backpropagation

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 observarse en la figura 2.3.8, en la cual se tomó como entrada al filtro la función:

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

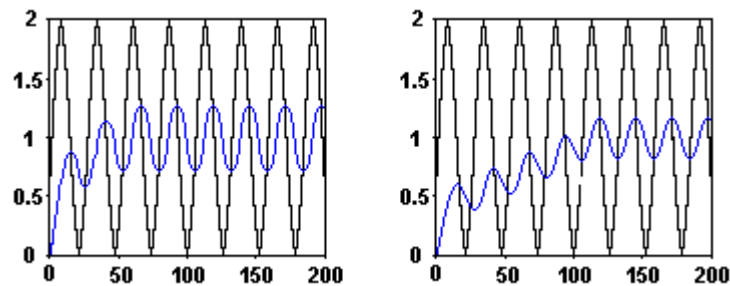


Figura 2.3.8 Efecto del coeficiente de momentum

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 la red Backpropagation tradicional:

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



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

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

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

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

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

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

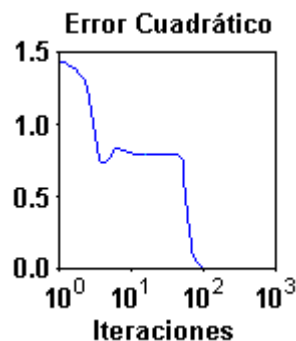


Figura 2.3.9 Trayectoria de convergencia con momentum

2.3.3.2 Red Backpropagation con tasa de aprendizaje variable [30]: Del análisis de la sección 2.3.3 se vio que $\nabla e(\mathbf{x})$ es el gradiente del error, de igual



forma se definirá $\nabla^2 \mathbf{e}(\mathbf{x})$ como la Hessiana de la función de error, donde \mathbf{x} representa las variables de las cuales depende el error (pesos y ganancias), esta matriz es siempre de la forma:

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

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, ésto 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 α debe 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.

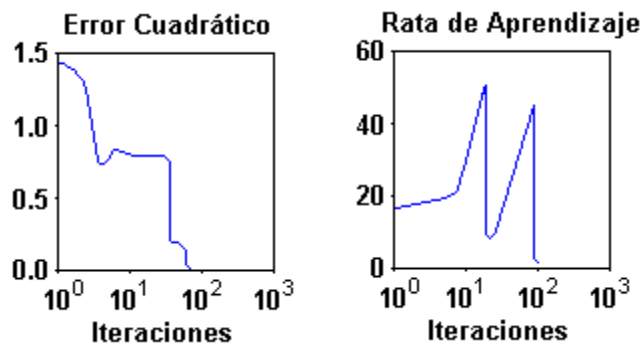


Figura 2.3.10 Característica de convergencia para una rata de aprendizaje variable



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



2.3.3.3 Método del Gradiente Conjugado [30]. 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

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

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

$$x_{k+1} = x_k + \alpha_k p_k \quad (2.3.45)$$

3. Seleccionar la dirección siguiente de acuerdo a la ecuación

$$p_k = -g_k + \beta_k p_{k-1} \quad (2.3.46)$$

con

$$\beta_k = \frac{\Delta g_{k-1}^T g_k}{\Delta g_{k-1}^T p_{k-1}} \text{ o } \beta_k = \frac{g_k^T g_k}{g_{k-1}^T g_{k-1}} \quad (2.3.47)$$

4. Si el algoritmo en este punto aún no ha convergido, se regresa 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

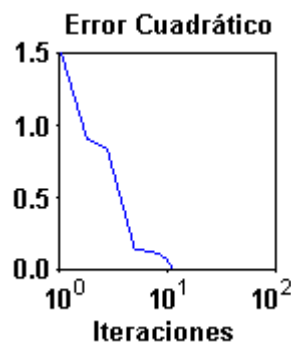


Figura 2.3.11 Trayectoria del Gradiente Conjugado

2.3.3.4 Algoritmo de Levenberg – Marquardt [30]. 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:

$$\mathbf{X}_{k+1} = \mathbf{X}_k - \mathbf{A}_k^{-1} \mathbf{g}_k \quad (2.3.48)$$

$$\mathbf{A}_k \equiv \nabla^2 e(\mathbf{x}) \Big|_{\mathbf{x}=\mathbf{x}_k} \quad \mathbf{g}_k \equiv \nabla e(\mathbf{x}) \Big|_{\mathbf{x}=\mathbf{x}_k} \quad (2.3.49)$$

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

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

El gradiente puede ser escrito entonces en forma matricial:

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

Donde $\mathbf{J}(\mathbf{x})$ es la matriz Jacobiana.

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

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \left[\mathbf{J}^T(\mathbf{x}_k) \mathbf{J}(\mathbf{x}_k) + \mu_k \mathbf{I} \right]^{-1} \mathbf{J}^T(\mathbf{x}_k) \mathbf{v}(\mathbf{x}_k) \quad (2.3.52)$$



o determinando directamente el incremento:

$$\Delta \mathbf{x}_k = -[\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k) + \mu_k \mathbf{I}]^{-1} \mathbf{J}^T(\mathbf{x}_k) \mathbf{v}(\mathbf{x}_k) \quad (2.3.53)$$

La nueva constante μ determina la tendencia del algoritmo, cuando μ_k se incrementa, este algoritmo se aproxima al algoritmo de pasos descendientes para tasas 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:



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

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

$$x^T = [x_1, x_2, \dots, x_n] = [w_{1,1}^1, w_{1,2}^1, \dots, w_{S^1,R}^1, b_1^1, \dots, b_{S^1}^1] \quad (2.3.55)$$

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

$$s_{i,h}^m \equiv \frac{\partial e_{k,q}}{\partial n_{i,q}^m} \quad (2.3.56)$$

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

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

$$[J]_{h,l} = \frac{\partial e_{k,q}}{\partial w_{i,j}^m} = \frac{\partial e_{k,q}}{\partial n_{i,q}^m} * \frac{\partial n_{i,q}^m}{\partial w_{i,j}^m} = s_{i,h}^m * \frac{\partial n_{i,q}^m}{\partial w_{i,j}^m} = s_{i,h}^m * a_{j,q}^{m-1} \quad (2.3.57)$$

o para las ganancias:



$$[J]_{h,l} = \frac{\partial e_{k,q}}{\partial b_i^m} = \frac{\partial e_{k,q}}{\partial n_{i,q}^m} * \frac{\partial n_{i,q}}{\partial b_i^m} = s_{i,h}^m * \frac{\partial n_{i,q}}{\partial b_i^m} = s_{i,h}^m \quad (2.3.58)$$

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

$$S_q^M = -f^M(n_q^M) \quad (2.3.59)$$

Cada columna de la matriz S_Q^M 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:

$$S_q^m = f^m(n_q^m)(W^{m+1})^T S_q^{m+1} \quad (2.3.60)$$

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

$$S^m = [S_1^m \ S_2^m] \dots [S_Q^m] \quad (2.3.61)$$

Para cada nueva entrada que es presentada a la red, los vectores de sensibilidad 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

$$e_q = t_q - a_q^M \quad (2.3.62)$$

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

2. Se calculan las sensitividades individuales y la matriz sensibilidad 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 μ por 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 μ por ϑ y se regresa al paso 3.

El algoritmo debe alcanzar convergencia cuando la norma del gradiente de



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

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 $\vartheta = 5$

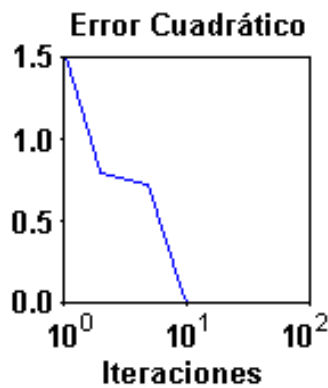


Figura 2.3.12 Trayectoria del algoritmo Levenberg-Marquardt

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.



2.4 APRENDIZAJE ASOCIATIVO

2.4.1 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 autoorganizarse.

Estas redes deben encontrar las características, regularidades, correlaciones o categorías que se puedan establecer entre los datos que se presenten en su entrada; puesto que no hay 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 (feature 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 Psicología, 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 conexas 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 período.

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.

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

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

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



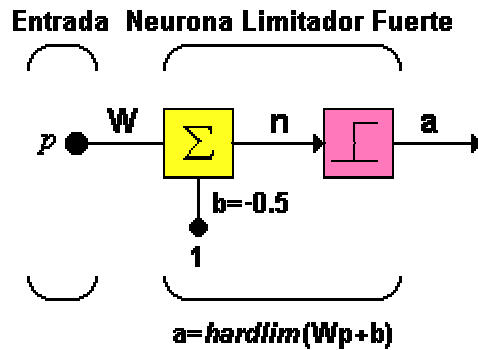


Figura 2.4.1 Asociador lineal con un limitador fuerte

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.

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

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 se representará 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.

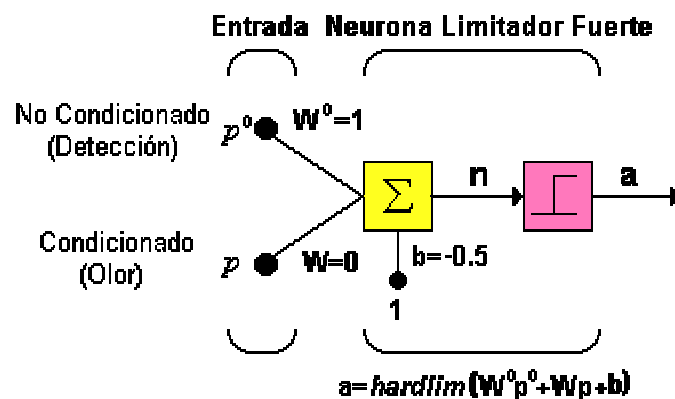


Figura 2.4.2 Asociador para una fruta

La definición de los estímulos estará dada por:

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



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:

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

La red responderá sólo 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.

2.4.3 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 p .

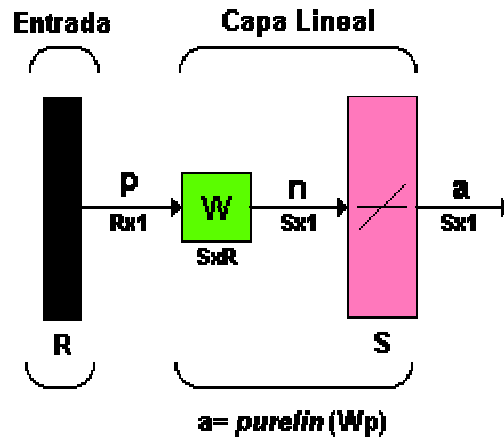


Figura 2.4.3 Asociador Lineal

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

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,

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

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 sinapsis. 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

$$w^0=1, \quad w(0) = 0 \quad (2.4.6)$$

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:

$$\{p^0(1) = 0, p(1) = 1\}, \{p^0(2) = 1, p(2) = 1\}.... \quad (2.4.7)$$

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:

$$\begin{aligned} a(1) &= \text{hardlim}(w^0 p^0(1) + w(0) p(1) - 0.5) \\ &= \text{hardlim}(1*0 + 0*1 - 0.5) = 0 \text{ No hay respuesta} \\ (2.4.8) \end{aligned}$$

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

$$w(1) = w(0) + a(1) p(1) = 0 + 0*1 = 0 \quad (2.4.9)$$

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

$$\begin{aligned} a(2) &= \text{hardlim}(w^0 p^0(2) + w(1) p(2) - 0.5) \\ &= \text{hardlim}(1*1 + 0*1 - 0.5) = 1 \text{ La fruta ha sido detectada} \end{aligned} \quad (2.4.10)$$

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

$$w(2) = w(1) + a(2) p(2) = 0 + 1*1 = 1 \quad (2.4.11)$$



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.

$$a(3) = \text{hardlim}(w^0 p^0(3) + w(2) p(3) - 0.5) \quad (2.4.12)$$

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

$$w(3) = w(2) + a(3) p(3) = 1 + 1*1 = 2 \quad (2.4.13)$$

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.

$$\begin{aligned} W(q) &= W(q-1) + \alpha a(q) p^T(q) - \gamma W(q-1) \\ &= (1 - \gamma) W(q-1) + \alpha a(q) p^T(q) \end{aligned} \quad (2.4.14)$$

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.

2.4.4 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 está 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.

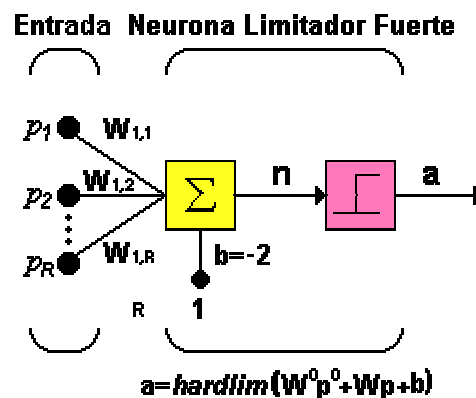


Figura 2.4.4 Red Instar

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:

$$a = \text{hardlims}(\mathbf{w}^T \mathbf{p} + b) \quad (2.4.15)$$

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 $-b$

$$\mathbf{w}^T \mathbf{p} \geq -b \quad (2.4.16)$$

Los vectores \mathbf{w} y \mathbf{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 \mathbf{w} y \mathbf{p} sea $\theta = 0$, esto permite observar que la red instar de la figura 2.4.4 se activará cuando \mathbf{p} y \mathbf{w} estén muy cercanos, escogiendo un apropiado valor para la ganancia b se puede determinar que tan cerca deben estar \mathbf{p} y \mathbf{w} para que la instar se active, si se fija



$$b = -\|\mathbf{w}\| \cdot \|\mathbf{p}\| \quad (2.4.17)$$

la instar se activará solamente cuando \mathbf{p} apunte exactamente en la misma dirección de \mathbf{w} , de esta forma b se puede incrementar a valores ligeramente mayores a $-\|\mathbf{w}\| \cdot \|\mathbf{p}\|$, 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 sólo 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 $a_i(q)$.

$$w_{ij}(q) = w_{ij}(q-1) + \alpha a_i(q) p_j(q) - \gamma a_i(q) w_{ij}^{anterior} \quad (2.4.18)$$

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:

$$w_{ij}(q) = w_{ij}(q-1) + \alpha a_i(q) (p_j(q) - w_{ij}^{anterior}) \quad (2.4.19)$$

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:

$$\begin{aligned} \mathbf{w}(q) &= \mathbf{w}(q-1) + \alpha (\mathbf{p}(q) - \mathbf{w}(q-1)) \\ &= (1-\alpha) \mathbf{w}(q-1) + \alpha \mathbf{p}(q) \end{aligned} \quad (2.4.20)$$

Esta operación se muestra en la figura 2.4.5

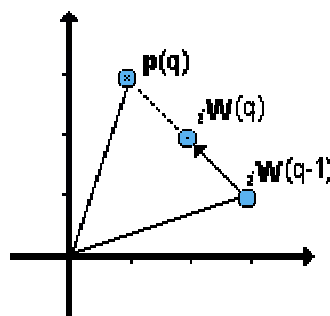


Figura 2.4.5 Representación gráfica de la regla de Instar



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 tasa de aprendizaje α . Cuando $\alpha = 0$, el nuevo vector de pesos es igual al vector de pesos anterior. Cuando $\alpha = 1$, el nuevo vector de pesos es igual al vector de entrada. Si $\alpha = 0.5$ 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 \mathbf{w} será también normalizado una vez la red haya aprendido un vector particular \mathbf{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).



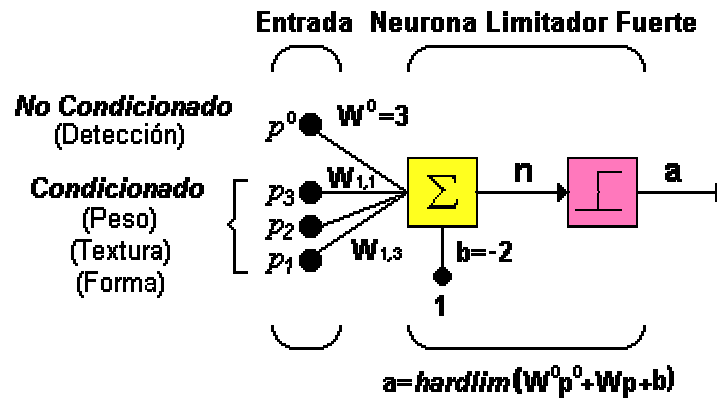


Figura 2.4.6 Reconocimiento de una fruta por medio de una Instar

La salida de la red está determinada por

$$a = \text{hardlim}(w^0 p^0 + W p + b)$$

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 \mathbf{p} es un vector normalizado con $\|\mathbf{p}\| = \sqrt{3}$. La definición de p^0 y \mathbf{p} es:

$$p^0 = \begin{cases} 1 & \text{fruta detectada visualmente} \\ 0 & \text{fruta no detectada} \end{cases} \quad \mathbf{p} = \begin{bmatrix} \text{forma} \\ \text{textura} \\ \text{peso} \end{bmatrix}$$

El valor de la ganancia se asumirá como $b = -2$, un valor ligeramente más positivo que $-\|\mathbf{p}\|^2 = -3$. 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

$$w^0=3, \quad \mathbf{W}(0) = {}_1\mathbf{w}^T(0) = [0 \ 0 \ 0] \quad (2.4.21)$$

Usando la regla Instar con una rata de aprendizaje $\alpha = 1$, los pesos actualizados se encontrarán de la siguiente forma:



$$w(q) = w(q-1) + a(q) (p(q) - w(q-1)) \quad (2.4.22)$$

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

$$\left\{ p^0(1) = 0, \mathbf{p}(1) = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \right\}, \left\{ p^0(2) = 1, \mathbf{p}(2) = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \right\} \quad (2.4.23)$$

Como la matriz **W** inicialmente contiene ceros, la Instar no responderá a los sensores de la fruta en la primera iteración

$$a(1) = \text{hardlim}(w^0 p^0(1) + \mathbf{Wp}(1) - 2) \quad (2.4.24)$$

$$a(1) = \text{hardlim} \left(3 * 0 + \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} - 2 \right) = 0 \quad \text{No hay respuesta}$$

Como la neurona no respondió, sus pesos no serán actualizados por la regla Instar

$$w(0) = w(0) + a(1)(p(1) - w(0))$$



$$= \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + 0 \left(\begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad (2.4.25)$$

En la segunda iteración, cuando la fruta haya sido detectada visualmente, la neurona responderá

$$a(2) = \text{hardlim} (w^0 p^0(2) + Wp(2) - 2)$$

$$a(2) = \text{hardlim} \left(3 * 1 + \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} - 2 \right) = 1 \text{ fruta detectada} \quad (2.4.26)$$

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.

$$w(2) = w(1) + a(2)(p(2) - w(1))$$

$$= \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + 1 \left(\begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} \quad (2.4.27)$$

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 condicionado) opera adecuadamente.

$$a(3) = \text{hardlim} (w^0 p^0(3) + \mathbf{W} \mathbf{p}(3) - 2)$$

$$a(3) = \text{hardlim} \left(3 * 0 + \begin{bmatrix} 1 & -1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} - 2 \right) = 1 \text{ fruta detectada} \quad (2.4.28)$$

Cuando las medidas de la fruta han sido detectadas completamente, los pesos dejan de cambiar y se estabilizan.

$$w(3) = w(2) + a(3)(p(3) - w(2))$$

$$= \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} + 1 \left(\begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} - \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \right) = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \quad (2.4.29)$$

2.4.5 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.



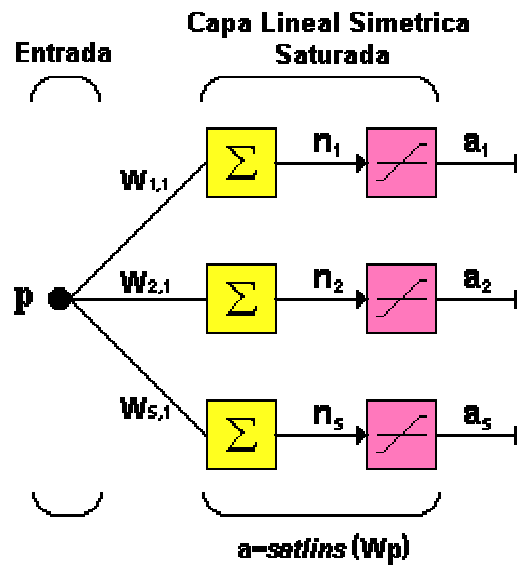


Figura 2.4.7 Red Outstar

La expresión de salida de esta red es:

$$a = \text{satlins}(Wp) \quad (2.4.30)$$

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 estímulo no condicionado.



$$\mathbf{a} = \text{satlins}(\mathbf{W}^0 \mathbf{p}^0 + \mathbf{W} \mathbf{p}) \quad (2.4.33)$$

Donde

$$\mathbf{W}^0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.4.34)$$

Continuando con el reconocimiento de frutas, los estímulos condicionado y no condicionado son:

$$\mathbf{p}^0 = \begin{bmatrix} \text{forma} \\ \text{textura} \\ \text{peso} \end{bmatrix} \quad p = \begin{cases} 1, & \text{la fruta es visualizada} \\ 0, & \text{la fruta no es visualizada} \end{cases}$$

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 \mathbf{W}^0 es la matriz identidad, así cualquier conjunto de medidas \mathbf{p}^0 (con valores entre 1 y -1) será reproducido a la salida de la red. La matriz de pesos para el estímulo condicionado \mathbf{W} , es inicializada en ceros para que un 1 en \mathbf{p} no genere respuesta. \mathbf{W} será actualizada con la regla Outstar, usando una tasa 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

$$\left\{ p^0(1) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, p(1) = 1 \right\}, \left\{ p^0(2) = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}, p(2) = 1 \right\} \quad (2.4.36)$$

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 entregará una respuesta.

$$a = \text{satlins}(W^0 p^0(1) + Wp(1)) \quad (2.4.37)$$

$$a(1) = \text{satlins} \left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} 1 \right) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \text{ no hay respuesta}$$

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



$$\mathbf{w}_1(1) = \mathbf{w}_1(0) + (\mathbf{a}(1) - \mathbf{w}_1(0)) \mathbf{p}(1) \quad (2.4.38)$$

$$= \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right) \mathbf{1} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

En la segunda iteración, tanto la fruta como sus medidas son presentadas a la red

$$\mathbf{a}(2) = \text{satlins} \left(\begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \mathbf{1} \right) = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} \text{ medidas correctas} \quad (2.4.39)$$

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

$$\mathbf{w}_1(2) = \mathbf{w}_1(1) + (\mathbf{a}(2) - \mathbf{w}_1(1)) \mathbf{p}(2)$$

$$= \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \left(\begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right) \mathbf{1} = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} \quad (2.4.40)$$

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

$$\mathbf{a}(3) = \text{satlins} \left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} 1 \right) = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} \text{ medidas recordadas} \quad (2.4.41)$$

Desde este momento, los pesos no sufrirán grandes cambios, a menos que la fruta sea vista con medidas diferentes

$$\mathbf{w}_1(3) = \mathbf{w}_1(2) + (\mathbf{a}(3) - \mathbf{w}_1(2)) \mathbf{p}(3) \quad (2.4.42)$$

$$= \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} + \left(\begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} - \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} \right) 1 = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}$$

Las redes de Instar y Outstar son empleadas conjuntamente en la red ART [20], 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.



2.5 REDES COMPETITIVAS

2.5.1 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 autoexcitación (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 Torte Wiesel. Su regla de aprendizaje generó gran interés, pero ésta utilizaba un cálculo no local para garantizar que los pesos fueran normalizados, este hecho hacia este modelo biológicamente poco posible.

Grossberg extendió el trabajo de Von Der Malsburg, redescubriendo la regla Instar. Grossberg mostró que la regla Instar removi6 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 bidimensionales), 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 Necognitron 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.

2.5.2 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, ésto 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 cómo 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 varias 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 $X(q)$, 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 $\mathbf{p} = (p_1, p_2, \dots, p_n)$, 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, ésta será aquella i cuyo vector de pesos \mathbf{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 \mathbf{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:

$$d_i = \sum_{j=1}^N (p_j - w_{ij})^2 \quad 1 \leq i \leq M \quad (2.5.1)$$

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.

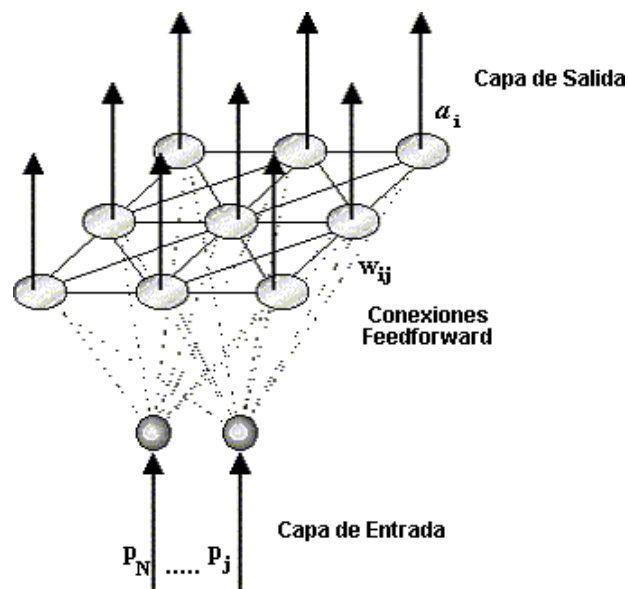


Figura 2.5.1 Conexiones de una red de Kohonen

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

$$w_i(q) = w_i(q-1) + \alpha(q)(p_i(q) - w_i(q-1)) \quad \text{para } i \in X(q) \quad (2.5.2)$$

El tamaño de $X(q)$ 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.

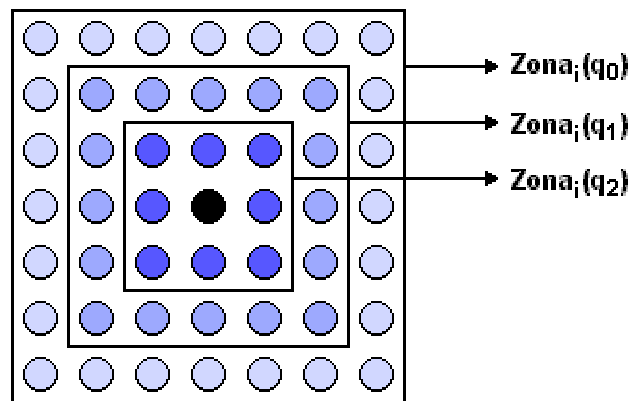


Figura 2.5.2 Posible evolución de la vecindad en una red de Kohonen

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]:

$$\alpha(q) = \frac{1}{q} \quad \alpha(q) = \alpha_1 \left(1 - \frac{q}{\alpha_2} \right) \quad (2.5.3)$$

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, \dots, 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 $X(q)$, 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.

2.5.3 Red de Hamming. La red de Hamming ilustrada en la figura 2.5.3 es uno de los ejemplos 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 cuál de los vectores prototipo está más cercano al vector de entrada.

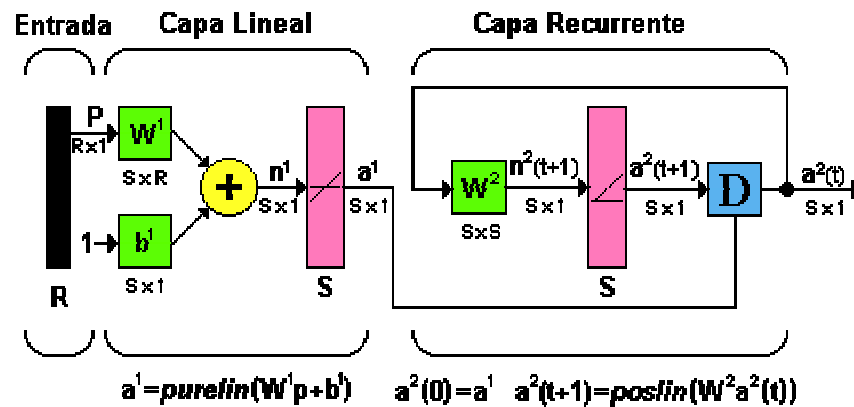


Figura 2.5.3 Red de Hamming

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

$$\{p_1, p_2, \dots, p_Q\} \quad (2.5.4)$$

La matriz de pesos W^1 y el vector de ganancias b^1 para la capa uno serán:



$$\mathbf{W}^I = \begin{bmatrix} {}_1\mathbf{W}^T \\ {}_2\mathbf{W}^T \\ \vdots \\ {}_s\mathbf{W}^T \end{bmatrix} = \begin{bmatrix} p_1^T \\ p_2^T \\ \vdots \\ p_Q^T \end{bmatrix}, \mathbf{b}^1 = \begin{bmatrix} R \\ R \\ R \\ R \end{bmatrix} \quad (2.5.5)$$

Donde cada fila de \mathbf{W}^1 representa un vector prototipo, el cual deseamos reconocer y cada elemento \mathbf{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:

$$\mathbf{a}^1 = \mathbf{W}^I \mathbf{p} + \mathbf{b}^1 = \begin{bmatrix} p_1^T \mathbf{p} + R \\ p_2^T \mathbf{p} + R \\ \vdots \\ p_Q^T \mathbf{p} + R \end{bmatrix} \quad (2.5.6)$$

La salida de la capa 1 es igual al producto punto de los vectores prototipo con la entrada más el vector \mathbf{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 determinará 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 sólo 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:

$$a^2(0) = a^1 \quad (2.5.7)$$

La salida de la segunda capa está determinada de acuerdo a la siguiente relación recurrente:

$$a^2(t+1) = \text{poslin}(W^2 a^2(t)) \quad (2.5.8)$$

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.



$$w_{ij}^2 = \begin{cases} 1, & \text{si } i = j \\ -\varepsilon & \text{de otra forma} \end{cases} \quad \text{Donde } 0 < \varepsilon < \frac{1}{s-1} \quad (2.5.9)$$

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 pesos de 1 y $-\varepsilon$ por los apropiados elementos de \mathbf{W}^2 ; reescribiendo la ecuación de salida de la red para una sola neurona se tiene :

$$a_i^2(t+1) = \text{poslin} \left(a_i^2(t) - \varepsilon \sum_{j \neq i} a_j^2(t) \right) \quad (2.5.10)$$

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.

2.5.4 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.



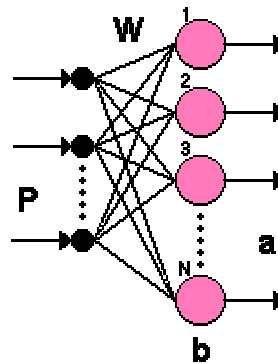


Figura 2.5.4 Instar agrupadas en una capa

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 cuál 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

$$a = \text{compet}(n) \quad (2.5.11)$$



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.

$$a_i = \begin{cases} 1, & i = i^* \\ 0, & i \neq i^* \end{cases} \quad \text{Donde } n_{i^*} \geq n_i, \forall i, \quad i^* \leq i, \forall n_i = n_{i^*} \quad (2.5.12)$$

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.

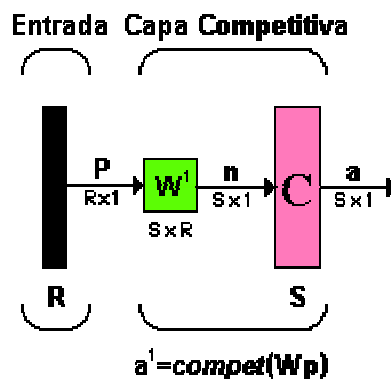


Figura 2.5.5 Capa Competitiva

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 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 \mathbf{p} y el vector prototipo \mathbf{w}_i :

$$\mathbf{n} = \mathbf{W}\mathbf{p} = \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \\ \vdots \\ \mathbf{w}_s^T \end{bmatrix} \mathbf{p} = \begin{bmatrix} \mathbf{w}_1^T \mathbf{p} \\ \mathbf{w}_2^T \mathbf{p} \\ \vdots \\ \mathbf{w}_s^T \mathbf{p} \end{bmatrix} = \begin{bmatrix} L^2 \cos \theta_1 \\ L^2 \cos \theta_2 \\ \vdots \\ L^2 \cos \theta_s \end{bmatrix} \quad (2.5.13)$$

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

$$\mathbf{a} = \text{compet}(\mathbf{n}) \quad (2.5.14)$$

2.5.5 Regla de aprendizaje. En este punto es posible diseñar una red competitiva que realice clasificaciones correctas fijando el valor de las filas de \mathbf{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 en el numeral 2.4.3

$${}_1w(q) = {}_1w(q-1) + a(q)(p(q) - {}_1w(q-1)) \quad (2.5.15)$$



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,

$${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha (\mathbf{p}(q) - {}_i\mathbf{w}(q-1)) = (1 - \alpha) {}_i\mathbf{w}(q-1) + \alpha \mathbf{p}(q) \quad (2.5.16)$$

y

$$\mathbf{w}(q) = \mathbf{w}(q-1) \quad i \neq i^* \quad (2.5.17)$$

Así, la fila de la matriz de pesos que esté 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

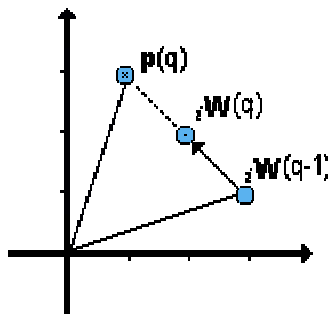


Figura 2.5.6 Representación gráfica de la regla de Kohonen

Para demostrar como trabaja una red competitiva, se creará una red que clasifique los siguientes vectores:



$$p_1 = \begin{bmatrix} -0.216 \\ 0.993 \end{bmatrix}, p_2 = \begin{bmatrix} 0.216 \\ 0.993 \end{bmatrix}, p_3 = \begin{bmatrix} 0.993 \\ 0.216 \end{bmatrix}$$

$$p_4 = \begin{bmatrix} 0.993 \\ -0.216 \end{bmatrix}, p_5 = \begin{bmatrix} -0.622 \\ -0.873 \end{bmatrix}, p_6 = \begin{bmatrix} -0.873 \\ -0.622 \end{bmatrix}$$

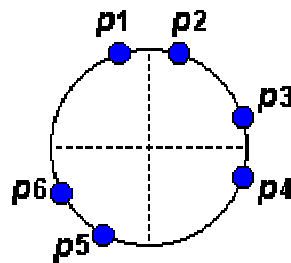


Figura 2.5.7 Vectores de entrada

La red tendrá tres neuronas, por lo tanto los vectores serán clasificados en tres clases o grupos, ésta 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:

$${}_1w = \begin{bmatrix} 0.7071 \\ -0.7071 \end{bmatrix}, {}_2w = \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix}, {}_3w = \begin{bmatrix} -1.000 \\ 0.000 \end{bmatrix}, W = \begin{bmatrix} {}_1w^T \\ {}_2w^T \\ {}_3w^T \end{bmatrix}$$

Los vectores de datos y los pesos asignados pueden visualizarse en la figura 2.5.8



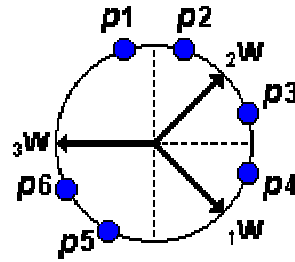


Figura 2.5.8 Vectores de entrada y vector de pesos

Se presenta a la red el vector p_2

$$a = \text{compet}(\mathbf{W}p_2) = \text{compet}\left(\begin{bmatrix} 0.7071 & -0.7071 \\ 0.7071 & 0.7071 \\ -1.000 & 0.000 \end{bmatrix} \begin{bmatrix} 0.216 \\ 0.993 \end{bmatrix}\right)$$

$$a = \text{compet}\left(\begin{bmatrix} -0.5494 \\ 0.8549 \\ -0.2160 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

El vector de peso de la segunda neurona estaba más cercano a p_2 , 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 tasa de aprendizaje $\alpha = 0.5$

$${}_2\mathbf{w}^{\text{nuevo}} = {}_2\mathbf{w}^{\text{anterior}} + \alpha (p_2 - {}_2\mathbf{w}^{\text{anterior}})$$

$${}_2\mathbf{w}^{\text{nuevo}} = \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix} + 0.5 \left(\begin{bmatrix} 0.216 \\ 0.993 \end{bmatrix} - \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix} \right) = \begin{bmatrix} 0.9527 \\ 0.5641 \end{bmatrix}$$



La regla de Kohonen hace que ${}_2w$ tienda hacia p_2 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.

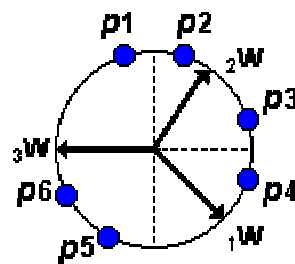


Figura 2.5.9 Proceso de entrenamiento

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

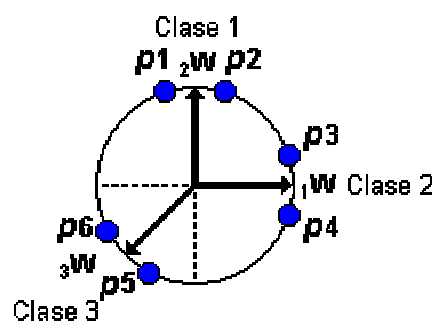


Figura 2.5.10 Pesos Finales

2.5.6 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 tasa de aprendizaje que permita hallar un punto de equilibrio entre velocidad de convergencia y la estabilidad final de los vectores de peso. Una tasa 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 tasa 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 tasa de aprendizaje puede ser empleada como una ventaja si se inicia el entrenamiento con una tasa 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

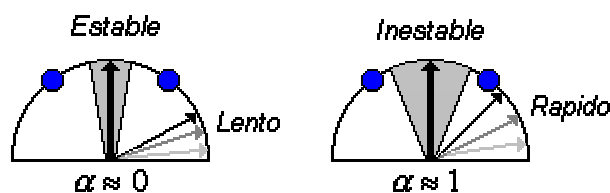


Figura 2.5.11 Variación de la tasa de aprendizaje

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

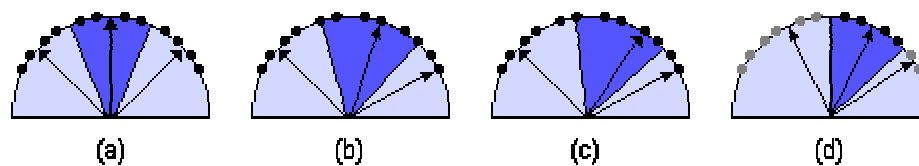


Figura 2.5.12 Aprendizaje Inestable

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.

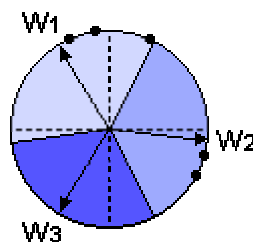


Figura 2.5.13 Causa de la muerte de una neurona

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.

2.5.7 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.



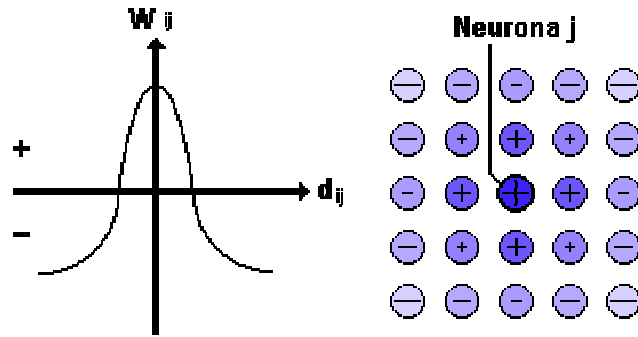


Figura 2.5.14 on-center/off-surround; para capas biológicas

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

$${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha (\mathbf{p}(q) - {}_i\mathbf{w}(q-1)) \quad \text{para } i \in N_{i^*}(d) \quad (2.5.18)$$

Donde el vecindario N_{i^*} contiene el índice para todas las neuronas que se encuentren a un radio “ d ” de la neurona ganadora i^*

$$N_i(d) = \{j, d_{ij} \leq d\} \quad (2.5.19)$$



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 $d = 2$.

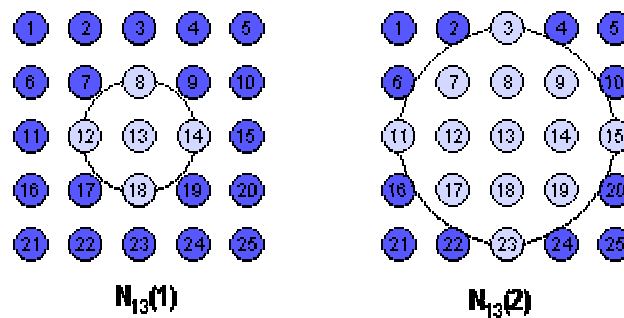


Figura 2.5.15 Vecindarios

Estos vecindarios pueden definirse como sigue:

$$N_{13}(1) = \{8, 12, 13, 14, 18\} \quad (2.5.20)$$

$$N_{13}(2) = \{3, 7, 8, 9, 11, 12, 13, 14, 15, 17, 18, 19, 23\}$$

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

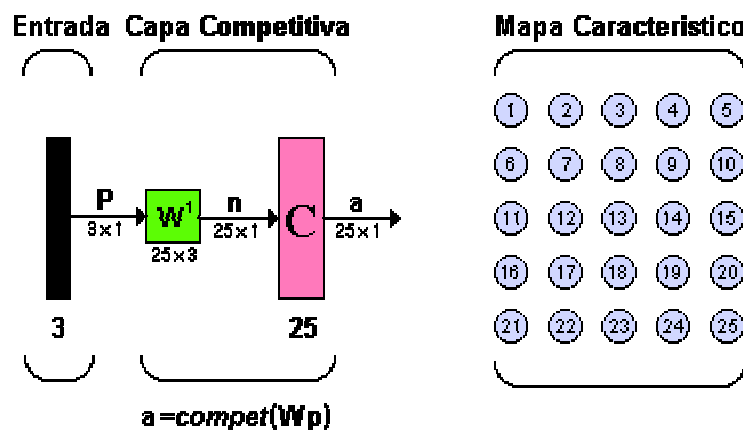


Figura 2.5.16 Mapa de auto organización

2.5.8 Learning Vector Quantization (LVQ). Esta red es un híbrido que emplea tanto aprendizaje no supervisado, como aprendizaje supervisado para clasificación de patrones

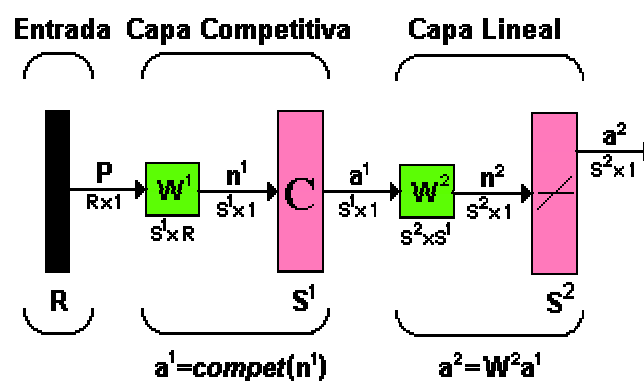


Figura 2.5.17 Red LVQ



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,

$$n_i^1 = - \begin{bmatrix} \|w^1_1 - p\| \\ \|w^1_2 - p\| \\ \vdots \\ \|w^1_{S^1} - p\| \end{bmatrix} \quad (2.5.21)$$

La salida de la primera capa de la red LVQ es,

$$a^1 = \text{compet}(n^1) \quad (2.5.22)$$



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.

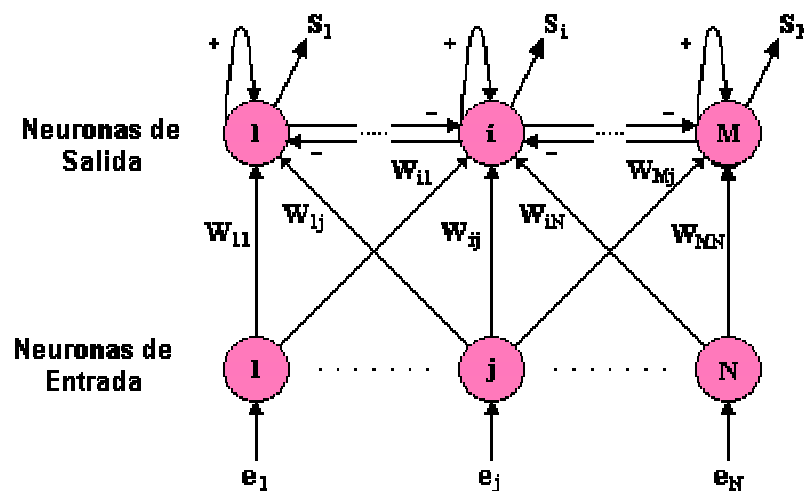


Figura 2.5.18 Comportamiento de las neuronas en una red LVQ

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 W^2 . Las columnas de W^2 representan las subclases y las filas representan las clases, 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.

$$W^2_{ki} = 1 \Rightarrow \text{la subclase } i \text{ pertenece a la clase } k \quad (2.5.23)$$



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 sólo 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

$$\{p_1, t_1\}, \{p_2, t_2\}, \dots, \{p_Q, t_Q\} \quad (2.5.24)$$

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:

$$\left\{ p_1 = \begin{bmatrix} \sqrt{0.74} \\ 0 \\ \sqrt{0.74} \end{bmatrix}, t_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \right\} \quad (2.5.25)$$

Antes de que suceda el aprendizaje, cada neurona en la segunda capa es asignada a una neurona de salida, así se genera la matriz 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 \mathbf{W}^2 son cero excepto los que cumplan la siguiente condición:

$$\text{Si la neurona } i \text{ es asignada a la clase } k \Rightarrow w_{ki}^2 = 1 \quad (2.5.26)$$

Una vez \mathbf{W}^2 ha sido definida, nunca será alterada. Los pesos ocultos \mathbf{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 \mathbf{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 \mathbf{a}^1 se fija en 1.
3. \mathbf{a}^1 es multiplicada por \mathbf{W}^2 para obtener la salida final \mathbf{a}^2 , la cual tiene solamente un elemento no cero, k^* , indicando que el patrón \mathbf{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 \mathbf{p} es clasificado correctamente los pesos de la neurona ganadora $i^* w^1$ se hacen tender hacia \mathbf{p} .

$$i^* w(q) = i^* w(q-1) - a(q) (\mathbf{p}(q) - i^* w(q-1)) \text{ si } a_{k^*}^2 = t_{k^*} = 1 \quad (2.5.27)$$

Segundo, si \mathbf{p} es clasificado incorrectamente una neurona equivocada ganó la competición y por lo tanto sus pesos $i^* w^1$ se alejan de \mathbf{p} .

$$i^* w(q) = i^* w(q-1) - a(q) (\mathbf{p}(q) - i^* w(q-1)) \text{ si } a_{k^*}^2 = 1 \neq t_{k^*} = 0 \quad (2.5.28)$$

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:

$$\text{clase 1: } \left\{ \mathbf{p}_1 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}, \text{ clase 2: } \left\{ \mathbf{p}_3 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \mathbf{p}_4 = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \right\}$$

Los vectores esperados asociados a cada una de las entradas son:



$$\left\{ \mathbf{p}_1 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\} \left\{ \mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\} \left\{ \mathbf{p}_3 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \mathbf{t}_3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\} \left\{ \mathbf{p}_4 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \mathbf{t}_4 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\}$$

La posición inicial de los patrones de entrada es la siguiente:

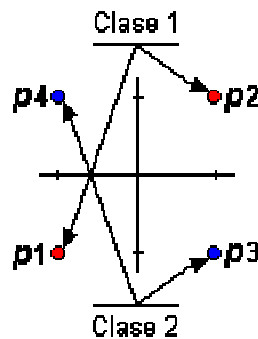


Figura 2.5.19 Posición de los patrones de entrada

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:

$$\mathbf{W}^2 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

\mathbf{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.



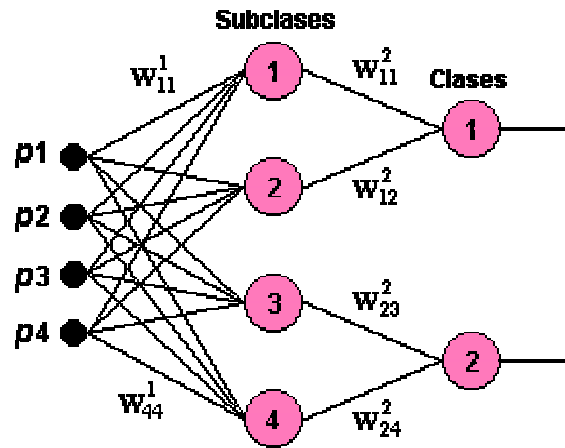


Figura 2.5.20 Esquema de la red LVQ que solucionará el ejemplo

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

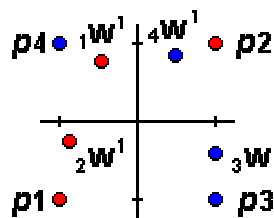


Figura 2.5.21 Estado inicial del vector de peso

Un vector de entrada diferente será presentado en cada iteración, se encontrará su respuesta y luego se actualizarán los pesos correspondientes. Se presentará inicialmente p_3 a la red:



$$a^1 = \text{compet}(n^1) = \text{compet} \left(\begin{bmatrix} -\|{}_1 w^1 - p_3\| \\ -\|{}_2 w^1 - p_3\| \\ -\|{}_3 w^1 - p_3\| \\ -\|{}_4 w^1 - p_3\| \end{bmatrix} \right)$$

$$a^1 = \text{compet} \left(\begin{bmatrix} -\|[-0.673 \ 0.970]^T - [1 \ -1]^T\| \\ -\|[-0.969 \ -0.379]^T - [1 \ -1]^T\| \\ -\|[1.002 \ 0.140]^T - [1 \ -1]^T\| \\ -\|[0.7805 \ 0.955]^T - [1 \ -1]^T\| \end{bmatrix} \right) = \text{compet} \left(\begin{bmatrix} -2.5845 \\ -2.0646 \\ -1.1400 \\ -1.9668 \end{bmatrix} \right) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

La tercera neurona oculta ha estado más cerca del vector p_3 y de esta forma ya se determinó 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 ${}_3 w^1$ es desplazado en la dirección de p_3 .

$${}_3 w^1(1) = {}_3 w^1(0) + \alpha (p_3 - {}_3 w^1(0))$$



$${}_3w^1(1) = \begin{bmatrix} 1.002 \\ 0.140 \end{bmatrix} + 0.5 \left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} - \begin{bmatrix} 1.002 \\ 0.140 \end{bmatrix} \right) = \begin{bmatrix} 1.001 \\ -0.430 \end{bmatrix}$$

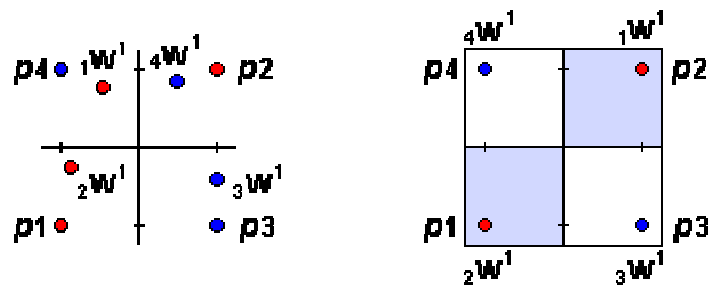


Figura 2.5.22 Resultado después de la primera y después de muchas iteraciones

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 p_1 y p_2 perteneciente a la clase uno son visualizadas en azul y los vectores p_3 y p_4 pertenecientes a la clase dos pueden verse en blanco.



2.6 REDES RECURRENTE

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.

2.6.1 Red de Hopfield.

2.6.1.1 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.

2.6.1.2 Estructura de la red. En búsqueda de una implementación práctica, 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 más 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, éstas 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 (2.6.1).

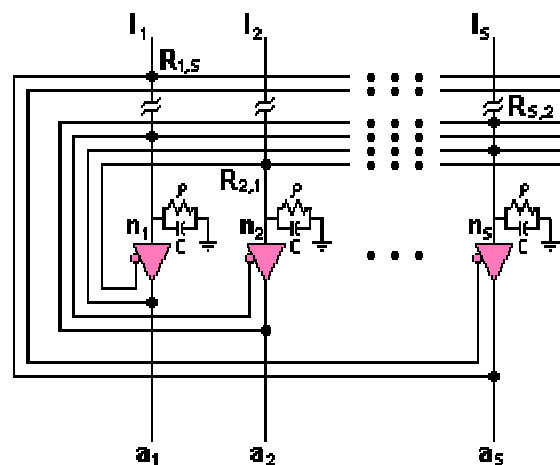


Figura 2.6.1 Circuito Eléctrico red Hopfield



$$C \frac{dn_i(t)}{dt} = \sum_{j=1}^S T_{ij} a_j(t) - \frac{n_i(t)}{R_i} + I_i \quad (2.6.1)$$

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 sigmoideal,

$$|T_{ij}| = \frac{1}{R_{i,j}} \quad \text{y} \quad \frac{1}{R_i} = \frac{1}{\rho} + \sum_{j=1}^S \frac{1}{R_{ij}}.$$

Multiplicando a ambos lados de la ecuación (2.6.1) por R_i y definiendo $\epsilon = R_i C$, $\omega_{ij} = R_i T_{ij}$ y $b_i = R_i I_i$, ésta 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.

$$\epsilon \frac{dn_i(t)}{dt} = -n_i(t) + \sum_{j=1}^S \omega_{ij} a_j + b_i \quad (2.6.2)$$

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

$$\epsilon \frac{d\mathbf{n}(t)}{dt} = -\mathbf{n}(t) + \mathbf{W}\mathbf{a}(t) + \mathbf{b} \quad (2.6.3)$$



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

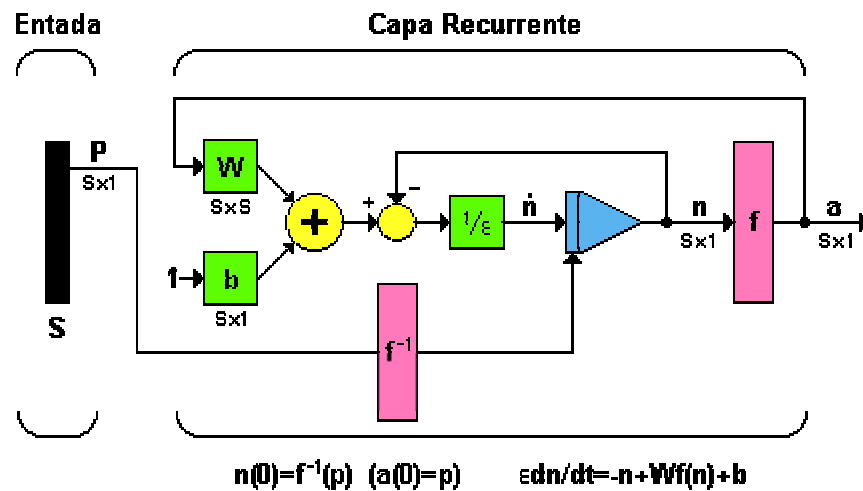


Figura 2.6.2 Notación compacta red de Hopfield

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:

$$V(\mathbf{a}) = -\frac{1}{2} \mathbf{a}^T \mathbf{W} \mathbf{a} + \sum_{i=1}^S \left\{ \int_0^{a_i} f^{-1}(u) du \right\} - \mathbf{b}^T \mathbf{a} \quad (2.6.4)$$

Donde \mathbf{a} es la salida de la red, \mathbf{W} es la matriz de pesos y \mathbf{b} es el vector de ganancias.

La escogencia de esta particular función, fue clave en el desarrollo de Hopfield, pues el primer y el tercer término 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:

$$\frac{d}{dt} \left(-\frac{1}{2} \mathbf{a}^T \mathbf{W} \mathbf{a} \right) = -\frac{1}{2} \nabla [\mathbf{a}^T \mathbf{W} \mathbf{a}]^T \frac{d\mathbf{a}}{dt} = -[\mathbf{W} \mathbf{a}]^T \frac{d\mathbf{a}}{dt} = -\mathbf{a}^T \mathbf{W} \frac{d\mathbf{a}}{dt} \quad (2.6.5)$$



Derivando el segundo término 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 término es:

$$\frac{d}{dt} \left[\sum_{i=1}^S \left\{ \int_0^{a_i} f^{-1}(u) du \right\} \right] = \mathbf{n}^T \frac{d\mathbf{a}}{dt} \quad (2.6.7)$$

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

$$\frac{d}{dt} \left\{ -\mathbf{b}^T \mathbf{a} \right\} = -\nabla \left[\mathbf{b}^T \mathbf{a} \right]^T \frac{d\mathbf{a}}{dt} = -\mathbf{b}^T \frac{d\mathbf{a}}{dt} \quad (2.6.8)$$

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

$$\frac{d}{dt} V(\mathbf{a}) = -\mathbf{a}^T \mathbf{W} \frac{d\mathbf{a}}{dt} + \mathbf{n}^T \frac{d\mathbf{a}}{dt} - \mathbf{b}^T \frac{d\mathbf{a}}{dt} = \left[-\mathbf{a}^T \mathbf{W} + \mathbf{n}^T - \mathbf{b}^T \right] \frac{d\mathbf{a}}{dt} \quad (2.6.9)$$



comparando con la ecuación (2.6.3) del modelo eléctrico de Hopfield, se tiene que:

$$\left[-\mathbf{a}^T \mathbf{W} + \mathbf{n}^T - \mathbf{b}^T \right] \frac{d\mathbf{a}}{dt} = -\epsilon \left[\frac{d\mathbf{n}(t)}{dt} \right]^T \quad (2.6.10)$$

Esto permite reescribir la ecuación 2.6.9 así como sigue:

$$\frac{d}{dt} V(\mathbf{a}) = -\epsilon \left[\frac{d\mathbf{n}(t)}{dt} \right]^T \frac{d\mathbf{a}}{dt} = -\epsilon \sum_{i=1}^S \left(\frac{dn_i}{dt} \right) \left(\frac{da_i}{dt} \right) \quad (2.6.11)$$

ya que $n_i = f^{-1}(a_i)$, es posible expandir la derivada de n_i de la siguiente forma:

$$\frac{dn_i}{dt} = \frac{d}{dt} [f^{-1}(a_i)] = \frac{d}{da_i} [f^{-1}(a_i)] \frac{da_i}{dt} \quad (2.6.12)$$

Con esto la ecuación (2.6.11) puede ser reescrita como:

$$\frac{d}{dt} V(\mathbf{a}) = -\epsilon \sum_{i=1}^S \left(\frac{dn_i}{dt} \right) \left(\frac{da_i}{dt} \right) = -\epsilon \sum_{i=1}^S \left(\frac{d}{da_i} [f^{-1}(a_i)] \right) \left(\frac{da_i}{dt} \right)^2 \quad (2.6.13)$$

si se asume que $f^{-1}(a_i)$ es una función incremental, como sucede en los amplificadores operacionales, entonces:



$$\frac{d}{da_i} [f^{-1}(a_i)] > 0 \quad (2.6.14)$$

Este resultado implica en la ecuación 2.6.12 que:

$$\frac{d}{dt} V(\mathbf{a}) \leq 0 \quad (2.6.15)$$

De esta manera, si $f^{-1}(a_i)$ es una función incremental, todos los valores propios de la función $dV(\mathbf{a})/dt$ son no positivos lo cual implica que la red sea estable, entonces $V(\mathbf{a})$ es una función de Lyapunov válida

Los atractores de Hopfield son puntos estacionarios de la función de Lyapunov que satisfacen la ecuación (2.6.16)

$$\frac{d\mathbf{a}}{dt} = 0 \quad (2.6.16)$$

Estos puntos estacionarios son puntos donde se encuentra un mínimo de la función $V(\mathbf{a})$ descrita en la ecuación (2.6.4), en estos puntos el gradiente de la función $V(\mathbf{a})$ igual a cero [21].



$$\nabla V(\mathbf{a}) = \left[\frac{\partial V}{\partial a_1} \quad \frac{\partial V}{\partial a_2} \quad \dots \quad \frac{\partial V}{\partial a_s} \right]^T = 0 \quad (2.6.17)$$

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 término de la función de Lyapunov se requiere el cálculo 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 término de la función de Lyapunov tiende a cero y puede definirse la función de alta ganancia de Lyapunov como:

$$V(\mathbf{a}) = -\frac{1}{2} \mathbf{a}^T \mathbf{W} \mathbf{a} - \mathbf{b}^T \mathbf{a} \quad (2.6.20)$$



2.6.1.3 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.

$$V(\mathbf{a}) = -\frac{1}{2} \mathbf{a}^T \mathbf{W} \mathbf{a} - \mathbf{b}^T \mathbf{a} \quad (2.6.21)$$

El procedimiento consiste en escoger la matriz de pesos \mathbf{W} y el vector de ganancias \mathbf{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 $\{p_1, p_2, \dots, p_Q\}$ y que cada uno de estos vectores se compone de S elementos, al asumir que $Q \ll S$, 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.

$$J(a) = -\frac{1}{2} \sum_{q=1}^Q \left([p_q]^T a \right)^2 \quad (2.6.22)$$

Si los elementos de 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.

$$J(\mathbf{a}) = -\frac{1}{2} \sum_{q=1}^Q ([\mathbf{p}_q]^T \mathbf{p}_j)^2 = -\frac{1}{2} ([\mathbf{p}_q]^T \mathbf{p}_j)^2 = -\frac{S}{2} \quad (2.6.23)$$

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 \mathbf{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 incrementará 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 $J(\mathbf{a})$ será la mayor (menos negativa) y cuando \mathbf{a} sea igual a alguno de los patrones prototipo $J(\mathbf{a})$ 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 \mathbf{W} y ganancias \mathbf{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)

$$\mathbf{W} = \sum_{q=1}^Q p_q (p_q)^T \text{ y } \mathbf{b}=0 \quad (2.6.24)$$

entonces la función de Lyapunov será:

$$V(\mathbf{a}) = -\frac{1}{2} \mathbf{a}^T \left[\sum_{q=1}^Q p_q (p_q)^T \right] \mathbf{a} = -\frac{1}{2} \sum_{q=1}^Q \mathbf{a}^T p_q (p_q)^T \mathbf{a} \quad (2.6.25)$$

y puede ser reescrita como:

$$V(\mathbf{a}) = -\frac{1}{2} \sum_{q=1}^Q \left[(p_q)^T \mathbf{a} \right]^2 = J(\mathbf{a}) \quad (2.6.26)$$

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 tenderá 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.

2.6.1.4 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

$$\dot{\chi}_i = -\chi_i + \sum_{j=1}^N \omega_{ij} \sigma(\chi_j) + \gamma_i u \quad i = 1, \dots, N \quad (2.6.27)$$

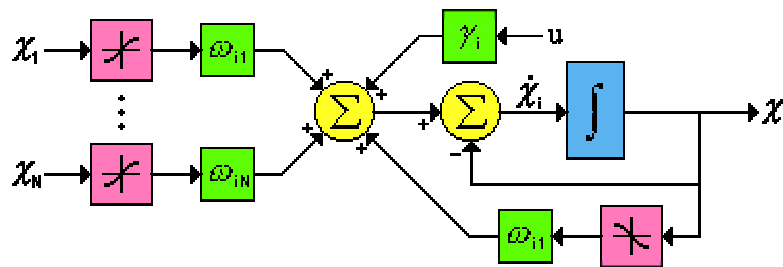


Figura 2.6.3 Neurona dinámica

o en forma matricial:

$$\dot{\chi} = A\chi + W\sigma(\chi) + \tilde{g}u \quad (2.6.28)$$



donde $A = -I$, $W = [\omega_{ij}]$, $\sigma(\chi) = [\sigma(\chi_1) \cdots \sigma(\chi_N)]^T$ 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.

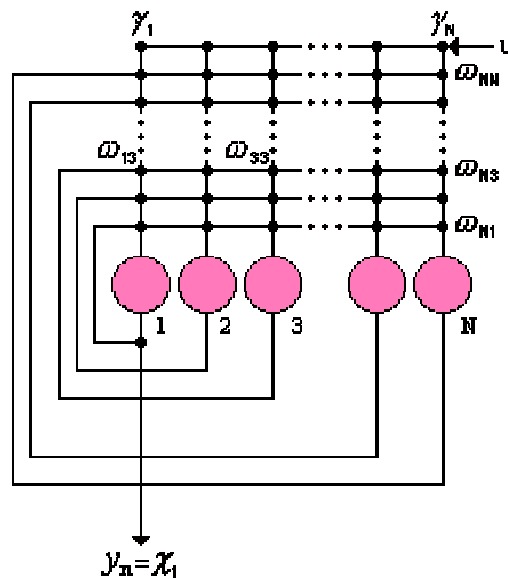


Figura 2.6.4 Red neuronal dinámica recurrente

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 continúa 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 continúa en esta dirección hasta que la función de error J no muestra cambios

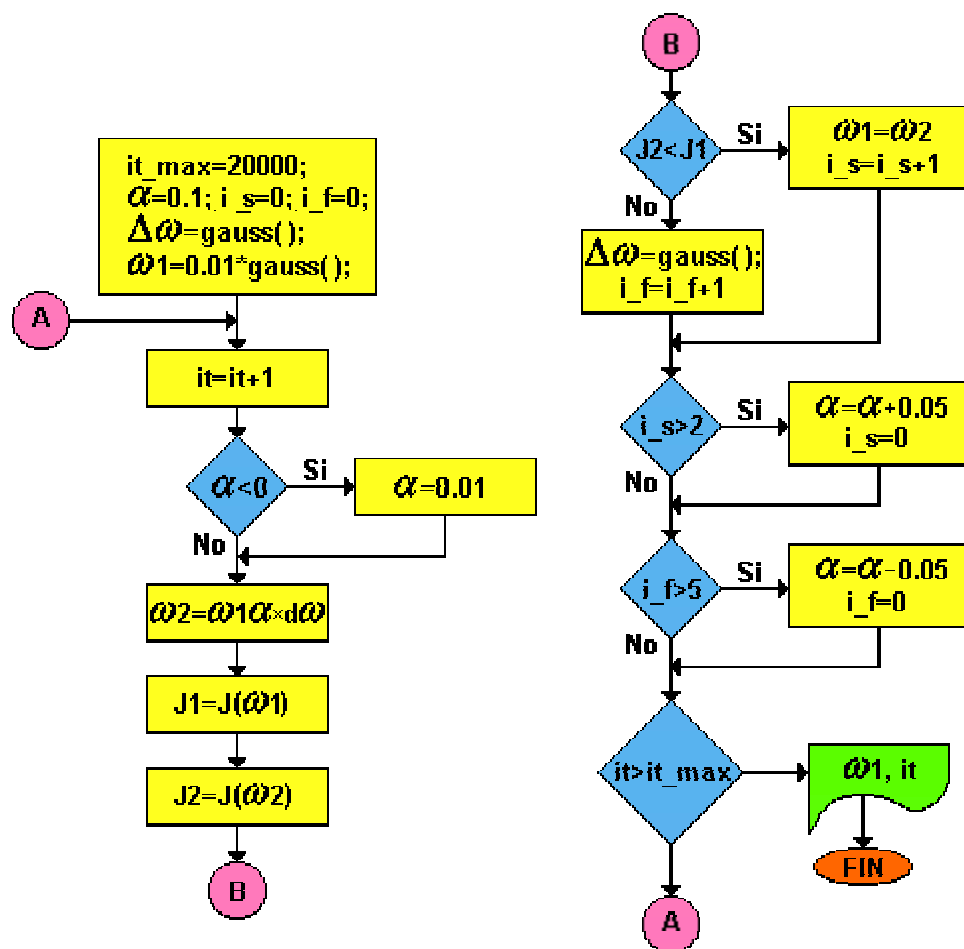


Figura 2.6.5 Algoritmo de Chemotaxis



it_max: Número máximo de iteraciones	it: Contador de iteraciones
i_s: Contador de iteraciones exitosas	α : Rata de aprendizaje
i_f: Contador de iteraciones no exitosas	ω_1 : Antigua matriz de pesos
$\Delta\omega$: Perturbación en la matriz de pesos	ω_2 : Antigua matriz de pesos
gauss(): Generador de números aleatorios con distribución Gaussiana	
Ji: Índice de la función de error correspondiente a la matriz de pesos ω_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.

$$J = \sum_{k=1}^{NP} (d_k - y_k)^2 \quad (2.6.29)$$

d_k : Salida deseada para el patrón de entrenamiento k .

y_k : Salida actual de la red ante el patrón de entrenamiento k .

2.6.2 Redes Multicapa

2.6.2.1 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.

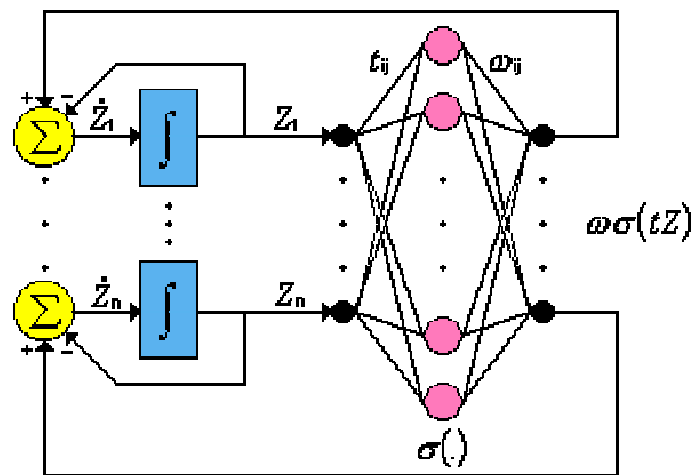


Figura 2.6.6 Red Dinámica Multicapa

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)



$$\frac{d}{dt} z = \bar{f}(z) = Ax + \omega \sigma(Tz) \quad (2.6.30)$$

$$\frac{d}{dt} x = f(x) = Ax + f_o(x) \quad (2.6.31)$$

donde $x, z \in \mathbb{R}^n$, $A \in \mathbb{R}^{n \times n}$, $f(x): \mathbb{R}^n \rightarrow \mathbb{R}^n$, $\bar{f}(z): \mathbb{R}^n \rightarrow \mathbb{R}^n$, $W \in \mathbb{R}^{n \times N}$, $T \in \mathbb{R}^{n \times N}$, $\sigma(z) = [\sigma(z_1), \sigma(z_2), \dots, \sigma(z_n)]$ y función de transferencia $\sigma(\theta) = \text{tansig}(\theta)$, n es el número de variables de estado del sistema y N el número de neuronas en la capa oculta.

Según Delgado[9], sin pérdida 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.

2.6.2.2 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

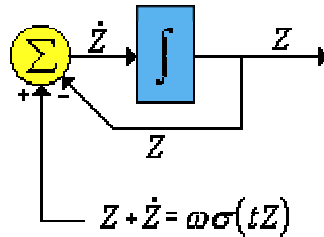


Figura 2.6.7 Patrones de entrenamiento de la red multicapa

La red después de entrenada tiene la estructura de la ecuación (2.6.32)

$$\frac{d}{dt} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} -z_1 \\ -z_2 \\ \vdots \\ -z_n \end{bmatrix} + \begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1n} \\ W_{21} & W_{22} & \cdots & W_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ W_{n1} & W_{n2} & \cdots & W_{nn} \end{bmatrix} \begin{bmatrix} \sigma(t_{11}z_1 + t_{12}z_2 + \dots + t_{1n}z_n) \\ \sigma(t_{21}z_1 + t_{22}z_2 + \dots + t_{2n}z_n) \\ \vdots \\ \sigma(t_{n1}z_1 + t_{n2}z_2 + \dots + t_{nn}z_n) \end{bmatrix} \quad (2.6.32)$$

Para garantizar que la red ha 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)

$$\chi = Tz \quad \frac{d\chi}{dt} = T \frac{dz}{dt} \quad (2.6.34)$$

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 $\chi \in \Re^N$, $TW \in \Re^{N \times N}$, 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 número de estados es mayor o igual al número de estados de la red multicapa $N \geq n$



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.

$$J_H = -I_N + TW \quad (2.6.37)$$

2.6.3 Red de Elman

2.6.3.1 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.



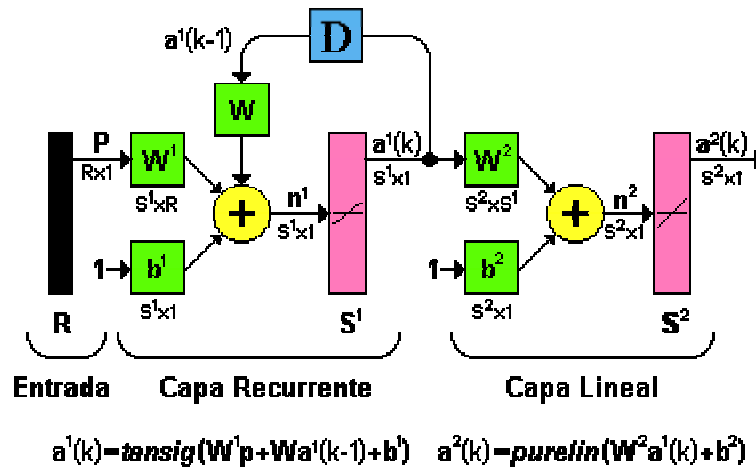


Figura 2.6.8 Red de Elman

La red de Elman generalmente posee neuronas con función transferencia sigmoideal 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 transferencia, es que según lo demostrado por Funahashi [16], puede aproximar cualquier función con la precisión deseada mientras que ésta posea un número finito de discontinuidades, para lo cual la precisión de la aproximación depende de la selección del número 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.



2.6.3.2 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 capítulo, entre los cuales se destacan los algoritmos basados en técnicas de optimización como el del gradiente conjugado o el algoritmo de Levenberg 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.

