

EL PERCEPTRON

REDES NEURONALES ARTIFICIALES

ABSTRACT

Neural networks are composed of simple elements operating in parallel. These elements are inspired by biological nervous systems. As in nature, the network function is determined largely by the connections between elements. We can train a neural network to perform a particular function by adjusting the values of the connections (weights) between elements. Rosenblatt created many variations of the perceptron. One of the simplest was a single-layer network whose weights and biases could be trained to produce a correct target vector when presented with the corresponding input vector. The training technique used is called the perceptron learning rule. The perceptron generated great interest due to its ability to generalize from its training vectors and learn from initially randomly distributed connections. Perceptrons are especially suited for simple problems in pattern classification. They are fast and reliable networks for the problems they can solve. In addition, an understanding of the operations of the perceptron provides a good basis for understanding more complex networks.

1. CARACTERÍSTICAS PRINCIPALES DE LAS REDES NEURONALES

1.1 INTRODUCCIÓN A LAS REDES NEURONALES

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

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

Es robusto y tolerante a fallas, diariamente mueren neuronas sin afectar su desempeño.

Es flexible, se ajusta a nuevos ambientes por aprendizaje, no hay que programarlo.

Puede manejar información difusa, con ruido o inconsistente.

Es altamente paralelo

Es pequeño, compacto y consume poca energía.

El cerebro humano constituye una computadora muy notable, es capaz de interpretar información imprecisa suministrada por los sentidos a un ritmo increíblemente veloz. Logra discernir un susurro en una sala ruidosa, un rostro en un callejón mal iluminado y leer entre líneas un discurso; lo más impresionante de todo, es que el cerebro aprende sin instrucciones explícitas de ninguna clase, a crear las representaciones internas que hacen posibles estas habilidades.

Basados en la eficiencia de los procesos llevados a cabo por el cerebro, e inspirados en su funcionamiento, varios investigadores han desarrollado desde hace más de 30 años la teoría de las

Redes Neuronales Artificiales (RNA), las cuales emulan las redes neuronales biológicas, y que se han utilizado para aprender estrategias de solución basadas en ejemplos de comportamiento típico de patrones; estos sistemas no requieren que la tarea a ejecutar se programe, ellos generalizan y aprenden de la experiencia.

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

Procesamiento de imágenes y de voz

Reconocimiento de patrones

Planeamiento

Interfaces adaptivas para sistemas Hombre/máquina

Predicción

Control y optimización

Filtrado de señales

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

Las RNA son una teoría que aún esta en proceso de desarrollo, su verdadera potencialidad no se ha alcanzado todavía; aunque los investigadores han desarrollado potentes algoritmos de aprendizaje de gran valor práctico, las representaciones y procedimientos de que se sirve el cerebro, son aún desconocidas. Tarde o temprano los estudios computacionales del aprendizaje con RNA acabarán por converger a los métodos descubiertos por evolución, cuando eso suceda, muchos datos empíricos concernientes al cerebro comenzarán súbitamente a adquirir sentido y se tornarán factibles muchas aplicaciones desconocidas de las redes neuronales.

1.2 FUNCIONAMIENTO DE UNA NEURONA BIOLÓGICA

El cerebro consta de un gran número (aproximadamente 10^{11}) de elementos altamente interconectados (aproximadamente 10^4 conexiones por elemento), llamados neuronas. Estas neuronas tienen tres componentes principales, las dendritas, el cuerpo de la célula o soma, y el axón. Las dendritas, son el árbol receptor de la red, son como fibras nerviosas que cargan de señales eléctricas el cuerpo de la célula. El cuerpo de la célula, realiza la suma de esas señales de entrada. El axón es una fibra larga que lleva la señal desde el cuerpo de la célula hacia otras neuronas. El punto de contacto entre un axón de una célula y una dendrita de otra célula es llamado sinápsis, la longitud de la sinápsis es determinada por la complejidad del proceso químico que estabiliza la función de la red neuronal. Un esquema simplificado de la interconexión de dos neuronas biológicas se observa en la figura 1.2.1.

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

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

Como consecuencia de los primeros estudios sobre la base neural de los sistemas mnémicos

(relacionados con la memoria), se creía que el almacenamiento de la memoria asociativa, tanto implícita como explícita, requerían de un circuito neuronal muy complejo. Entre quienes comenzaron a oponerse a este enfoque se hallaba Donald O. Hebb, profesor de la universidad de Milner; Hebb sugirió que el aprendizaje asociativo podría ser producido por un mecanismo celular sencillo y propuso que las asociaciones podrían formarse por una actividad neuronal coincidente: "Cuando un axón de la célula A excita la célula B y participa en su activación, se produce algún proceso de desarrollo o cambio metabólico en una o en ambas células, de suerte que la eficacia de A, como célula excitadora de B, se intensifica". Según la regla Hebbiana de aprendizaje, el que coincida la actividad de las neuronas presinápticas (suministran el impulso de entrada) con la de las postsinápticas (reciben el impulso) es muy importante para que se refuerce la conexión entre ellas, este mecanismo es llamado pre-postasociativo, del cual puede observarse un ejemplo en la figura 1.2.2.

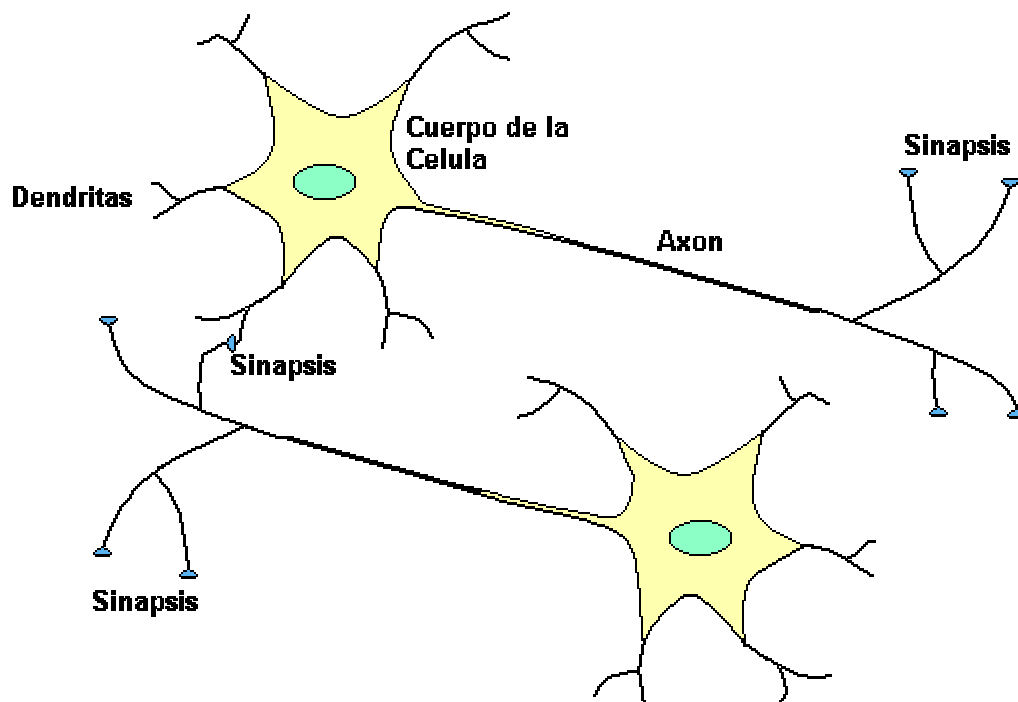


Figura 1.2.1 Neuronas Biológicas

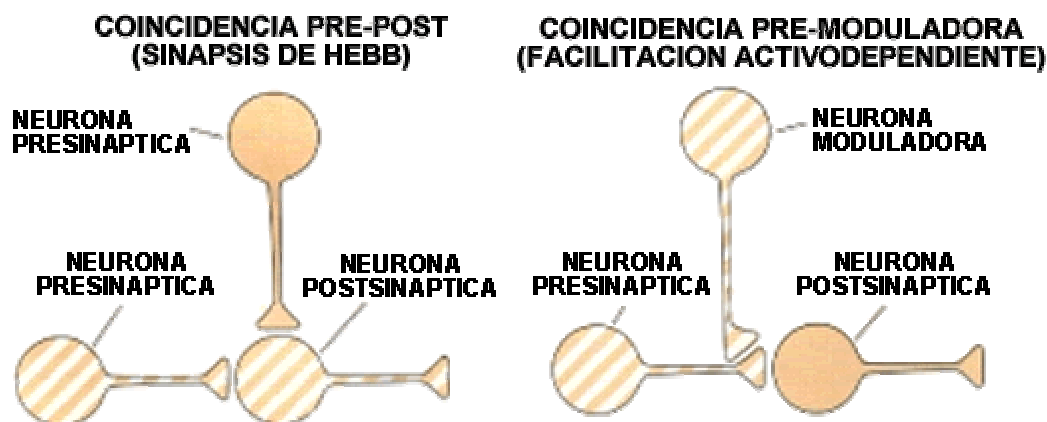


Figura 1.2.2 Cambios asociativos de las fuerzas sinápticas durante el aprendizaje

Las RNA no alcanzan la complejidad del cerebro, sin embargo hay dos aspectos similares entre redes biológicas y artificiales, primero los bloques de construcción de ambas redes son sencillos

elementos computacionales (aunque las RNA son mucho más simples que las biológicas) altamente interconectados; segundo, las conexiones entre neuronas determinan la función de la red.

1.3 CARACTERISTICAS DE UNA RED NEURONAL ARTIFICIAL

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

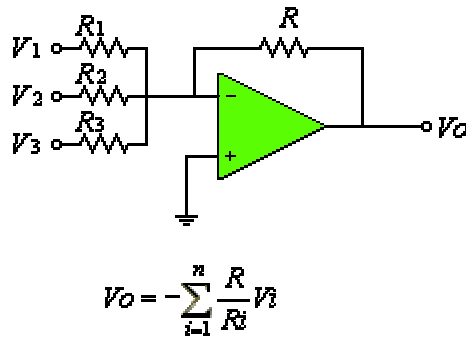


Figura 1.3.1 Neurona Artificial

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

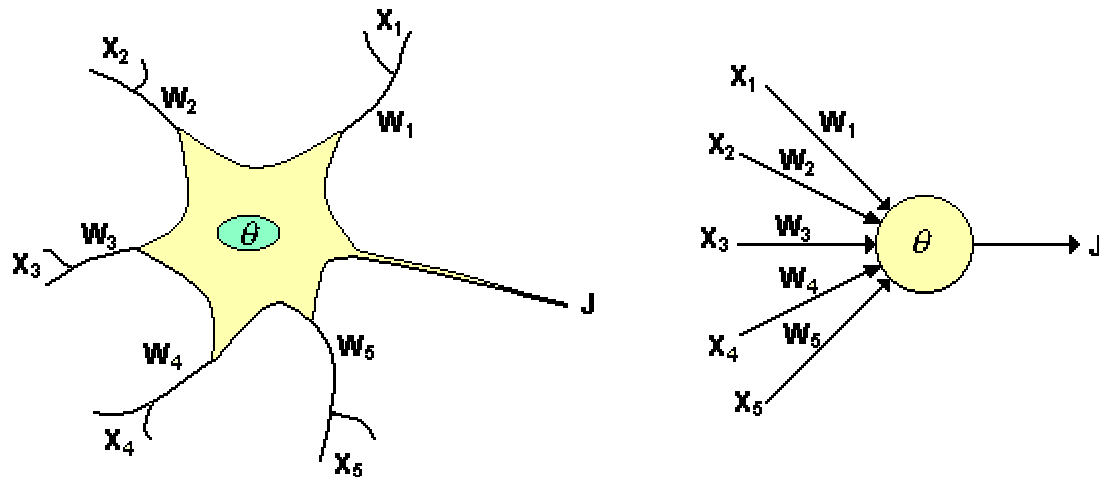


Figura 1.3.2 De la neurona biológica a la neurona artificial

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

Las entradas X_i representan las señales que provienen de otras neuronas y que son capturadas por las dendritas.

Los pesos W_i son la intensidad de la sinápsis que conecta dos neuronas; tanto X_i como W_i son valores reales.

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

Las señales de entrada a una neurona artificial X_1, X_2, \dots, X_n son variables continuas en lugar de pulsos discretos, como se presentan en una neurona biológica. Cada señal de entrada pasa a través de una ganancia o peso, llamado peso sináptico o fortaleza de la conexión cuya función es análoga a

la de la función sináptica de la neurona biológica. Los pesos pueden ser positivos (excitatorios), o negativos (inhibitorios), el nodo sumatorio acumula todas las señales de entradas multiplicadas por los pesos o ponderadas y las pasa a la salida a través de una función umbral o función de transferencia. La entrada neta a cada unidad puede escribirse de la siguiente manera.

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

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

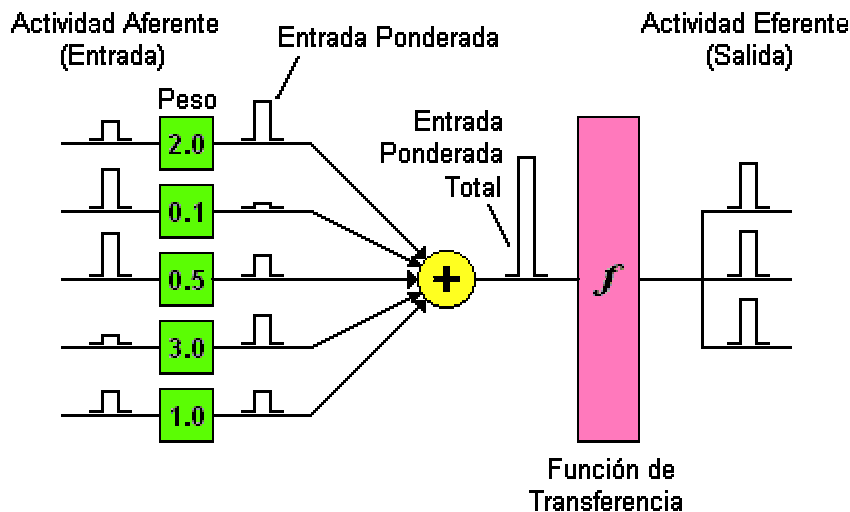


Figura 1.3.3 Proceso de una red neuronal

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

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

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

1.3.1 Notación. Una notación matemática estándar no ha sido aún establecida para las redes neuronales, ya que sus aplicaciones son útiles en muchos campos, Ingeniería, Física, Psicología y Matemáticas. En este trabajo se adoptará la siguiente convención para identificar las variables, de manera que fuera compatibles con las diferentes áreas, siendo lo más sencilla posible:

Valores escalares: se representarán por medio de letra minúscula itálica

Vectores: se representarán con letra itálica minúscula en negrilla.

Matrices: se representarán con letra mayúscula itálica en negrilla.

Para redes multicapa, los parámetros adoptaran la siguiente forma:

$$W_{sc,sc}^c$$

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

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

$$W_{s^2, s^1}^2$$

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

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

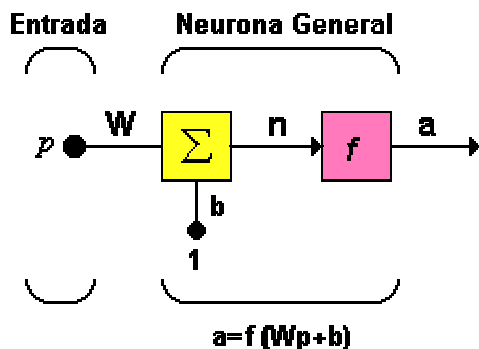


Figura 1.3.4 Neurona de una sola entrada

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

1.3.2.1. Limitador fuerte (Hardlim): La figura 1.3.5, muestra como esta función de transferencia acerca la salida de la red a cero, si el argumento de la función es menor que cero y la lleva a uno si este argumento es mayor que uno. Esta función crea neuronas que clasifican las entradas en dos categorías diferentes, característica que le permite ser empleada en la red tipo Perceptrón

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

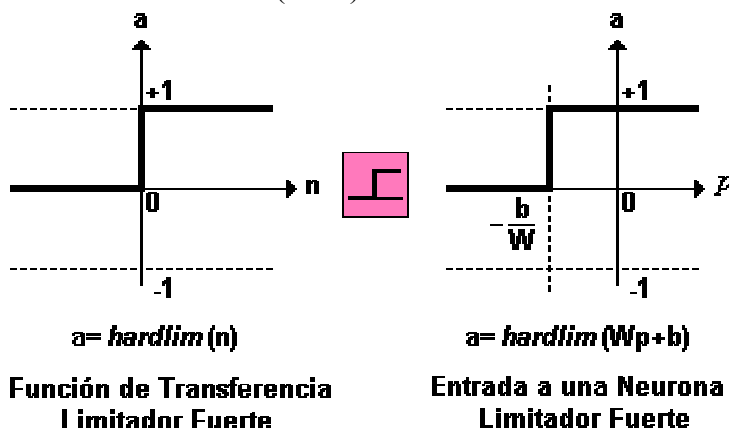


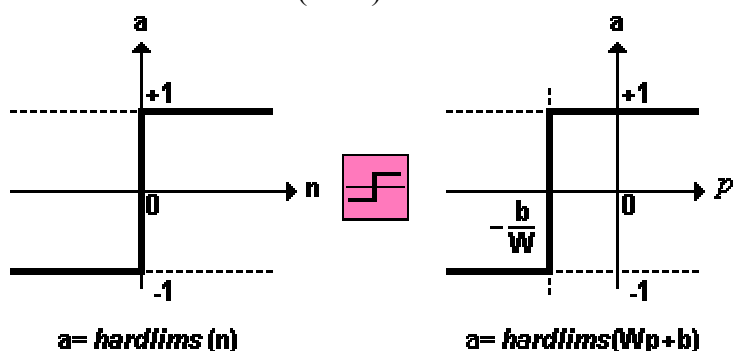
Figura 1.3.5 Función de transferencia Hardlim

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

utilice la función Hardlim.

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

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



**Función de Transferencia
Limitador Fuerte Simetrica**

**Entrada a una Neurona
Limitador Fuerte Simetrica**

Figura 1.3.6 Función de transferencia Hardlims

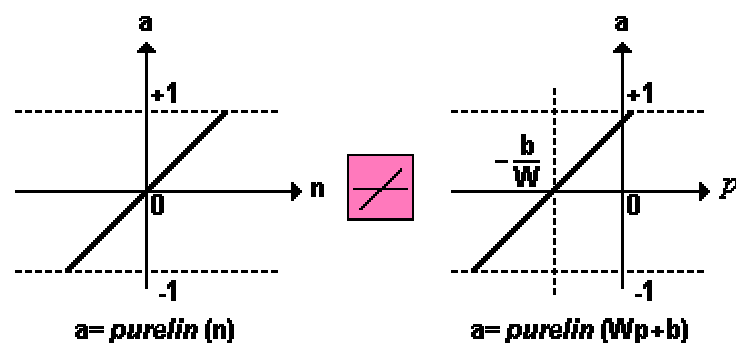
1.3.2.2 Función de transferencia lineal (purelin): La salida de una función de transferencia lineal es igual a su entrada,

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

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

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

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



**Función de Transferencia
Lineal**

**Entrada a una Neurona
Lineal**

Figura 1.3.7 Función de transferencia lineal

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

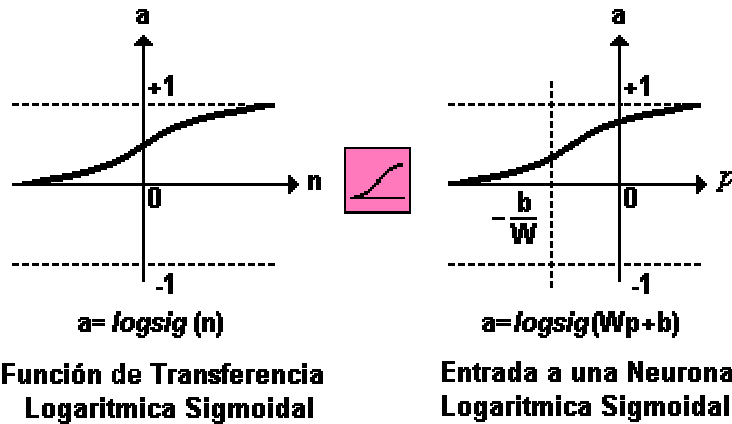


Figura 1.3.8 Función de transferencia sigmoidal

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

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

Competitiva	$a = 1$ Neurona con n max $a = 0$ El resto de neuronas	C	compet
-------------	---	----------	--------

Tabla 1.3.1 Funciones de Transferencia

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

Entrada Neurona Limitador Fuerte

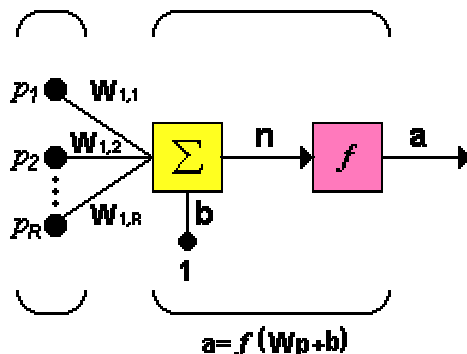


Figura 1.3.9 Neurona con múltiples entradas

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

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

Esta expresión puede ser escrita en forma matricial

$$n = Wp + b \quad (1.3.8)$$

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

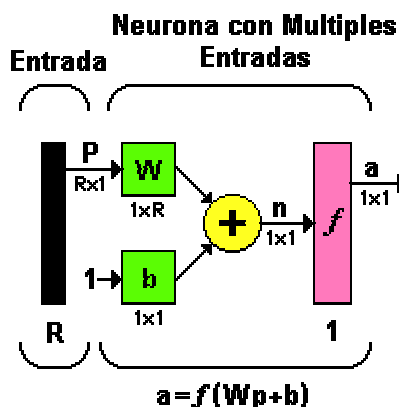


Figura 1.3.10 Neurona con múltiples entradas, notación abreviada

El vector de entrada p es representado por la barra sólida vertical a la izquierda. Las dimensiones de

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

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

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

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

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

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

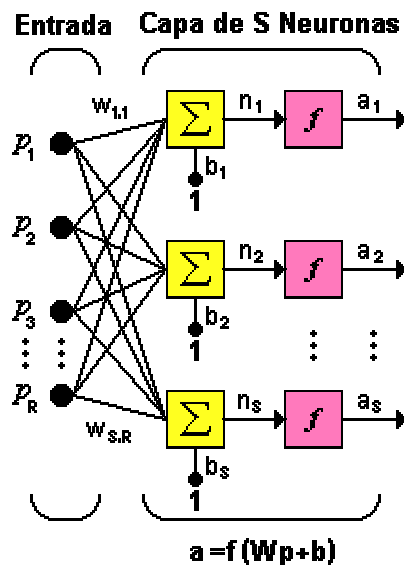


Figura 1.3.11 Capa de S neuronas

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

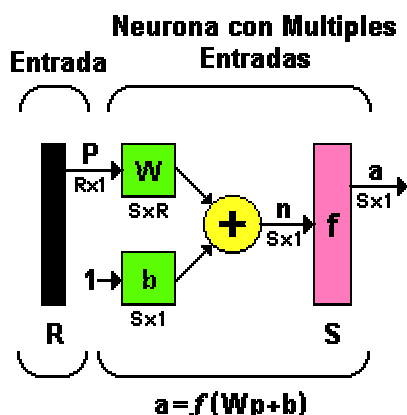


Figura 1.3.12 Capa de S neuronas con notación abreviada

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

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

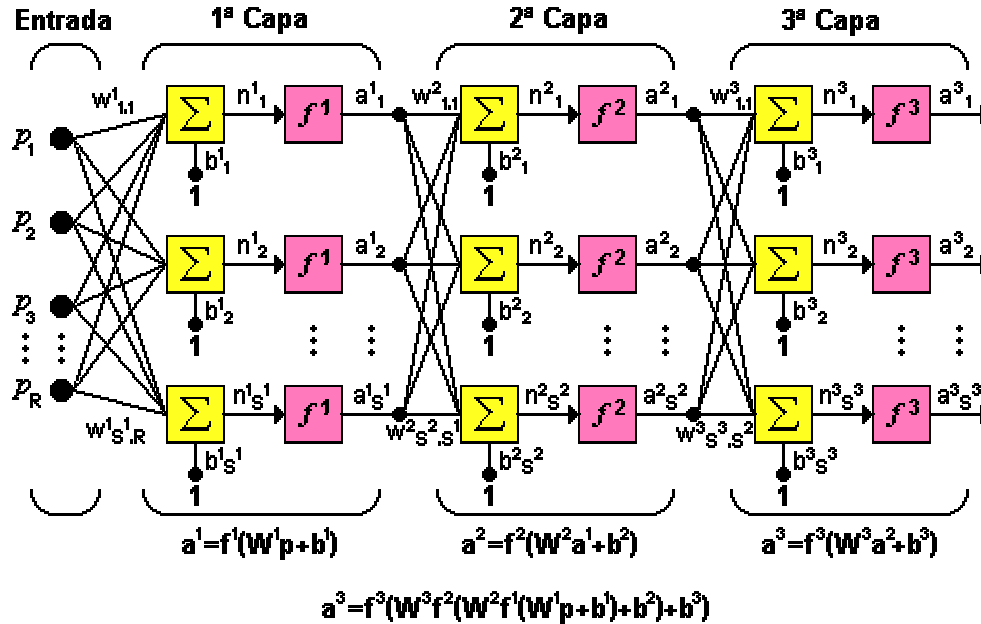


Figura 1.3.13 Red de tres capas

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

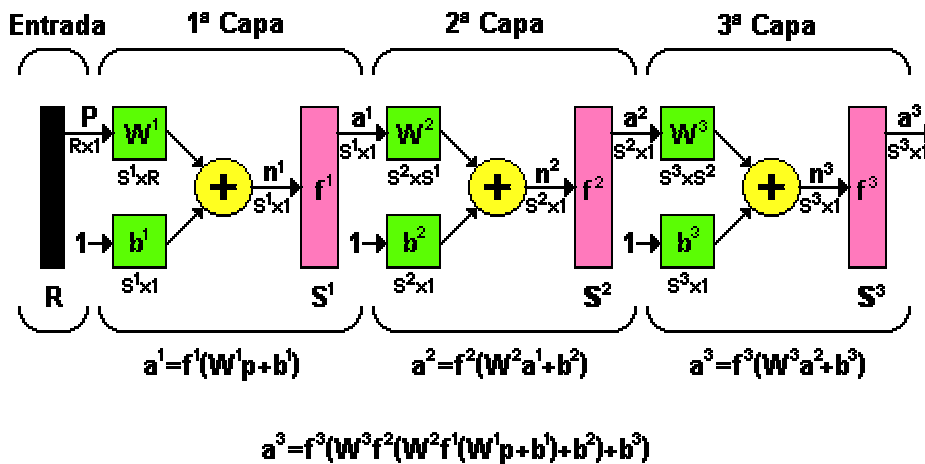


Figura 1.3.14 Red de tres capas con notación abreviada

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

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

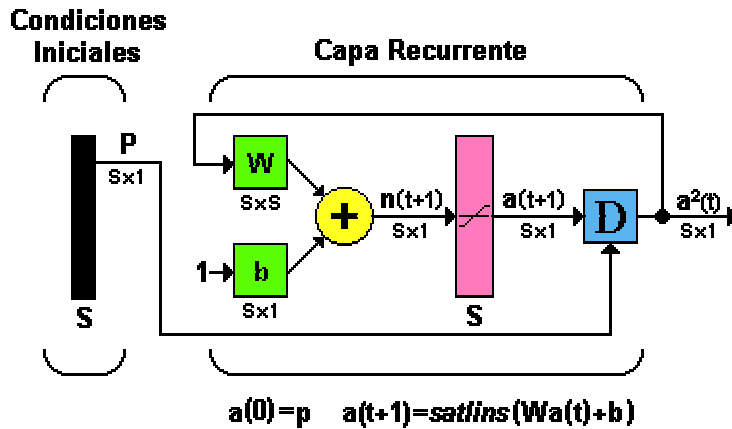


Figura 1.3.15 Redes Recurrentes

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

$$a(1) = satlins(Wa(0) + b), \quad a(2) = satlins(Wa(1) + b) \quad (1.3.9)$$

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

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

Retardo

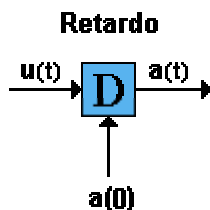


Figura 1.3.16 Bloque de retardo

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

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

Integrador

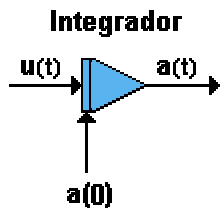


Figura 1.3.17 Bloque integrador

La salida del integrador es calculada de acuerdo a la expresión

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

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

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

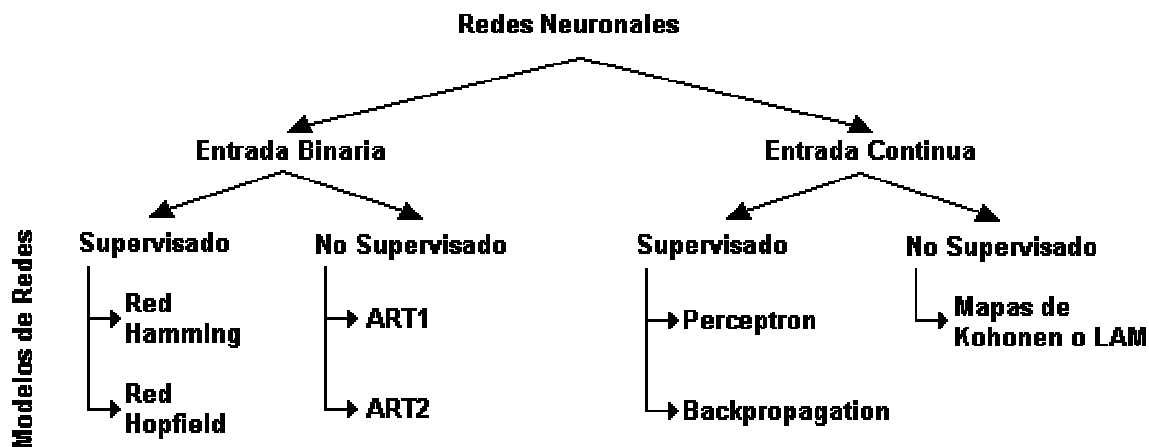


Figura 1.3.18 Clasificación de las Redes Neuronales

ESTADO DEL ARTE

Las RNA no son capaces de resolver todo tipo de problema y por tanto no son el sustituto de las computadoras convencionales; por ejemplo no serían adecuadas para escribir un sistema de nómina. Pero hay algunas áreas en las que se han aplicado exitosamente como son: en el Análisis financiero, el Procesamiento de señales, en la Mercadotecnia, en la Automatización y robótica, en los Diagnósticos médicos, en problemas de Clasificación, en el Reconocimiento de patrones, en el Control de procesos, trabajos de Optimización, etc.

Nace entonces la inquietud de saber: ¿las RNA, en qué tipo de aplicaciones trabajan adecuadamente y en dónde su aplicación es potencial?, y por ende ¿cuáles son los alcances y limitaciones de las RNA?, la respuesta es que trabajan exitosamente en tareas donde no hay reglas bien definidas las cuales parecen fáciles para los humanos y difíciles para las computadoras. Este campo es tan amplio como el de las computadoras convencionales. La Inteligencia Artificial ha estado generalmente dominada por las áreas de manipulación lógica y simbólica, pero algunos piensan que las RNA reemplazarán la Inteligencia Artificial tradicional actual. Mas bien parece que se combinarán en sistemas apoyándose

mutuamente, como sucede en los seres humanos. Otros sistemas realizan actividades que requieren más tiempo cuando el reconocimiento falla o cuando se requieren niveles superiores de decisión.

Otra pregunta interesante es ¿qué tan confiables son las RNA?, a lo que se puede responder que las RNA son hasta cierto grado impredecibles (como los humanos). No hay manera de garantizar la salida de una red para una entrada a menos que se prevean todas las posibilidades en las entradas, y se entrene la red suficientemente bien. Sin embargo esto puede resultar impráctico en muchos casos y verdaderamente imposible en otros, lo que ha originado muchas críticas a las RNA. Parte de esta crítica se debe al hecho de que esperamos que las computadoras sean perfectas, pero los humanos no son perfectos... Otro problema relacionado y criticado en las RNA es su inhabilidad de "explicar" como resuelven el problema. Esta inhabilidad se debe a que la representación interna generada en la red puede ser demasiado compleja aún en los casos más sencillos.

ANÁLISIS Y DISEÑO DEL PERCEPTRON Y SU IMPLEMENTACION

2.1 PERCEPTRÓN

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

La red tipo Perceptrón fue inventada por el psicólogo Frank Rosenblatt en el año 1957. Su intención era ilustrar algunas propiedades fundamentales de los sistemas inteligentes en general, sin entrar en mayores detalles con respecto a condiciones específicas y desconocidas para organismos biológicos concretos. Rosenblatt opinaba que la herramienta de análisis más apropiada era la teoría de probabilidades, y esto lo llevó a una teoría de *separabilidad estadística* que utilizaba para caracterizar las propiedades más visibles de estas redes de interconexión ligeramente aleatorias.

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

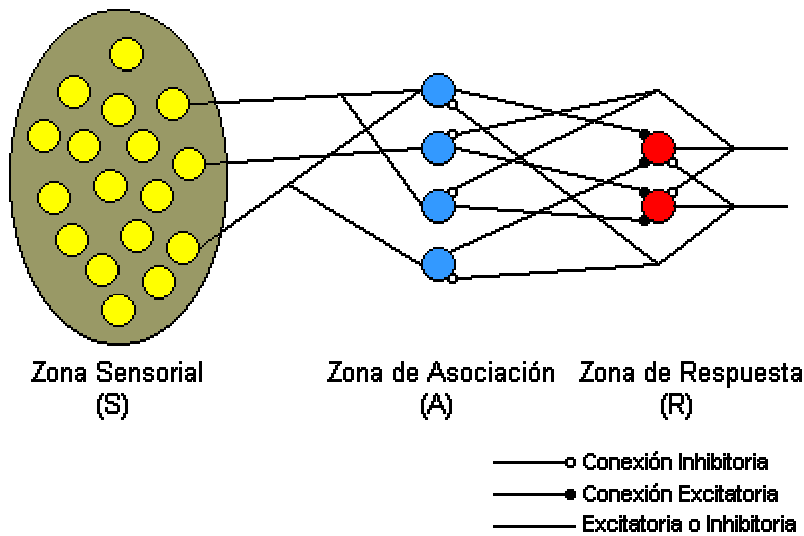


Figura 2.1.1 Modelo del Fotoperceptrón de Rosenblatt

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

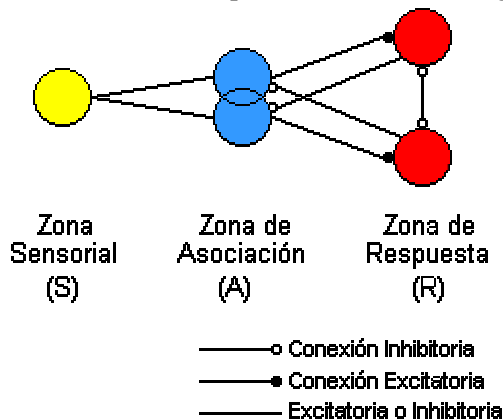


Figura 2.1.2 Esquema de conexiones de un Perceptrón sencillo

El Perceptrón era inicialmente un dispositivo de aprendizaje, en su configuración inicial no estaba en capacidad de distinguir patrones de entrada muy complejos, sin embargo mediante un proceso de aprendizaje era capaz de adquirir esta capacidad. En esencia, el entrenamiento implicaba un proceso de refuerzo mediante el cual la salida de las unidades A se incrementaba o se decrementaba dependiendo de si las unidades A contribuían o no a las respuestas correctas del Perceptrón para una entrada dada. Se aplicaba una entrada a la retina, y el estímulo se propagaba a través de las capas hasta que se activase una unidad de respuesta. Si se había activado la unidad de respuesta correcta, se incrementaba la salida de las unidades A que hubieran contribuido. Si se activaba una unidad **R** incorrecta, se hacía disminuir la salida de las unidades A que hubiesen contribuido.

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

aumentaba el número de patrones que intentaba aprender.

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

Minsky y Papert se apartaban de la aproximación 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

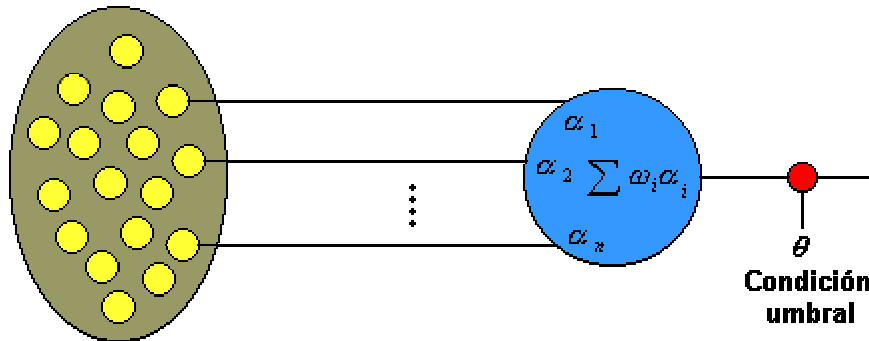


Figura 2.1.3 Perceptrón según Minsky y Papert

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.

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

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:

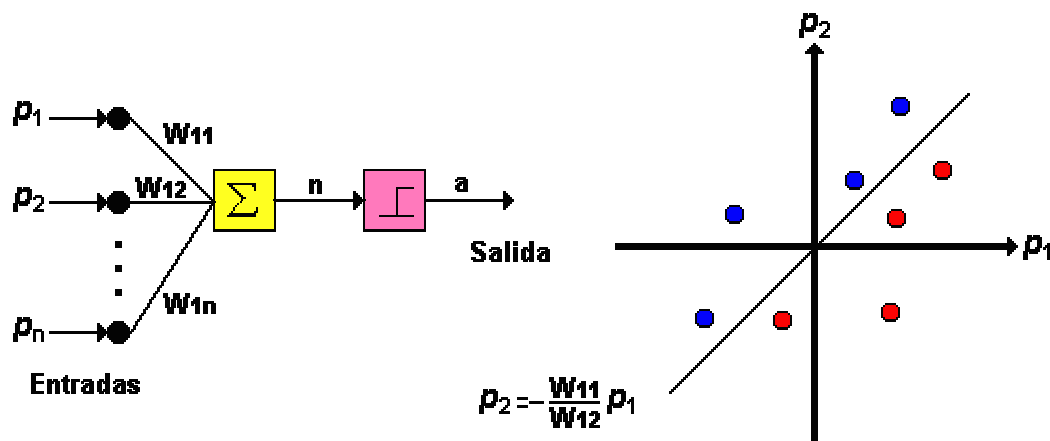


Fig. 2.1.4 Perceptrón

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 (este concepto se enunciará 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.

Para observar entonces el funcionamiento de una red neuronal tipo Perceptrón, solucionaremos a continuación 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

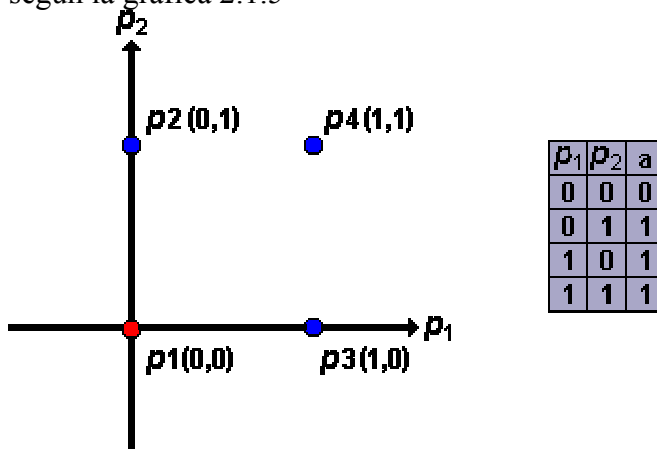


Figura 2.1.5 Función OR

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

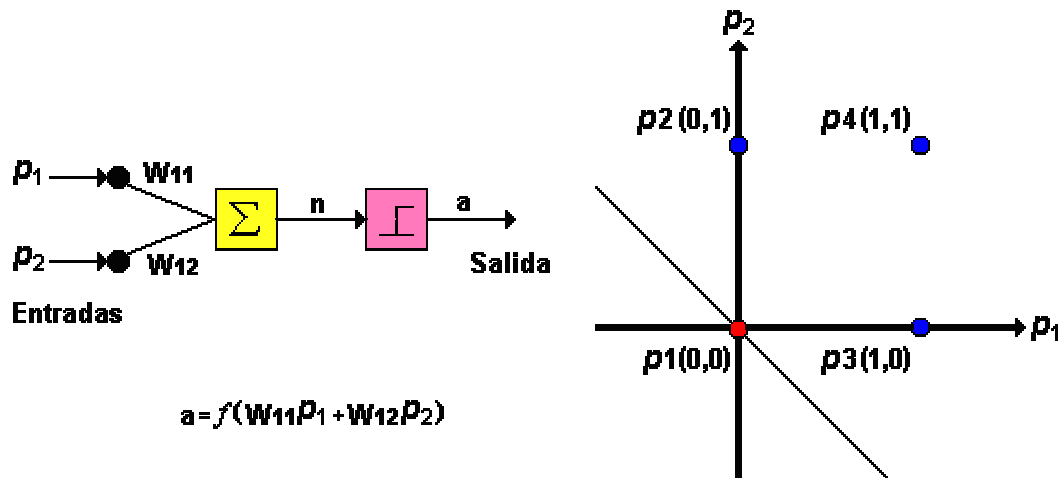


Figura 2.1.6 Perceptrón aplicado a la función OR

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

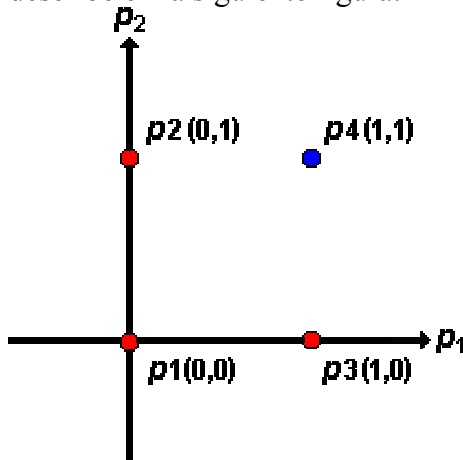


Figura 2.1.7 Espacio de salida de una compuerta AND

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:

$$a = \text{hardlims}(w_1p_1 + w_2p_2 + b) \quad (2.1.8)$$

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

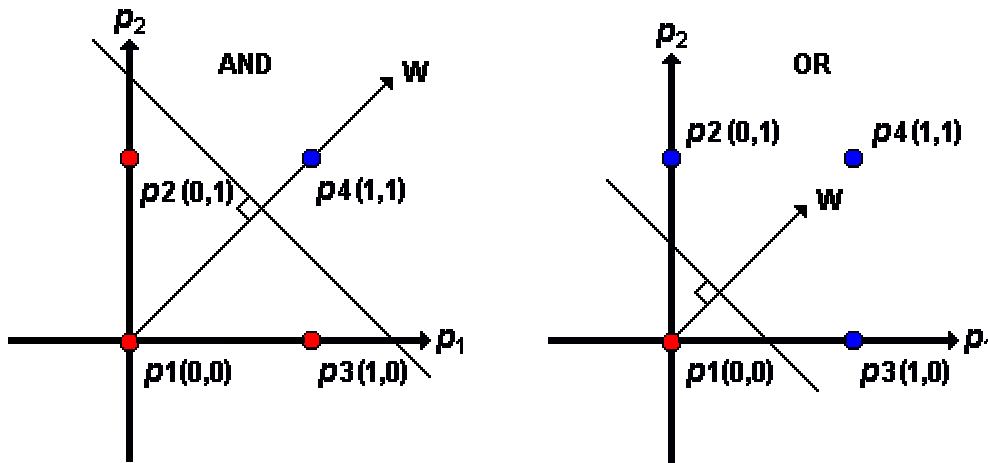


Figura 2.1.8 Solución para una función AND y una OR

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

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

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

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

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*

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:

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

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

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

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:

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

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

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

En una sola expresión la ley puede resumirse así:

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

Y extendiendo la ley a las ganancias

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

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

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

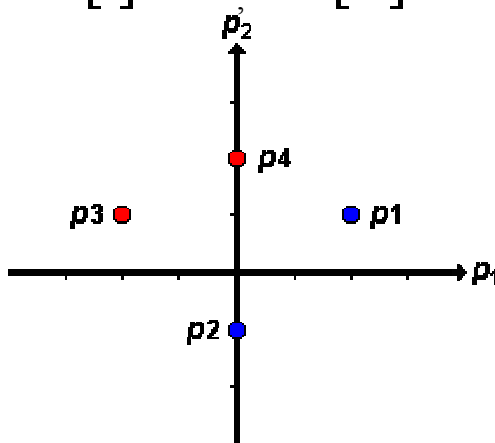


Figura 2.1.9 Patrones de entrenamiento

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

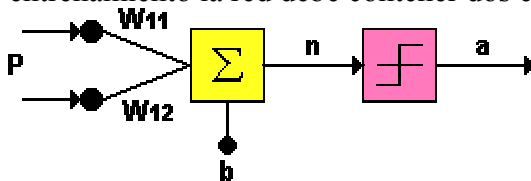


Figura 2.1.10 Red Perceptrón que resolverá el problema de clasificación de patrones

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

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

$$W = \begin{bmatrix} -0.7 & 0.2 \end{bmatrix} \quad b = \begin{bmatrix} 0.5 \end{bmatrix}$$

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

el siguiente:

Iteración 0

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

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

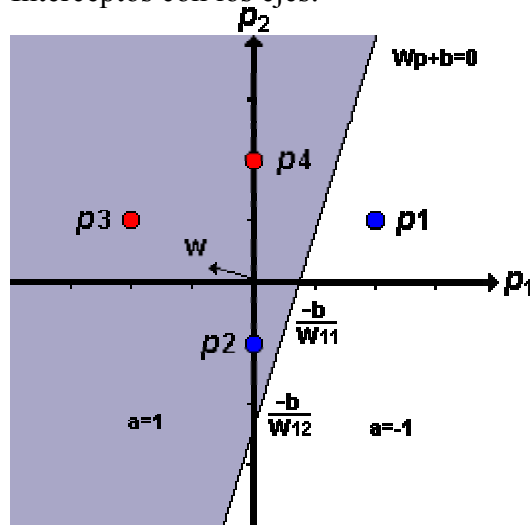


Figura 2.1.11 Clasificación de los patrones de acuerdo a la iteración 0

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

Iteración 1

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

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

$$a = \text{hardlims}(-0.7) = -1$$

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

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

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

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

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

$$\frac{-b}{W_{11}} = -0.75 \quad \frac{-b}{W_{21}} = -1.13$$

Interceptos con los ejes:

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

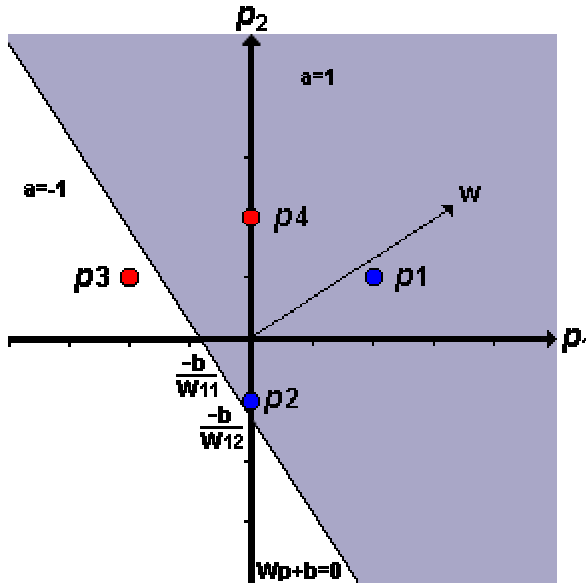


Figura 2.1.12 Característica de decisión de la iteración 1

Iteración 2

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

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

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

$$a = \text{hardlims}(0.3) = 1$$

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

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

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

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

Iteración 3

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

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

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

$$a = \text{hardlims}(-1.9) = -1$$

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

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

$$W^3 = W^2 + eP^T W^3 = \begin{bmatrix} 3.3 & 2.2 \end{bmatrix} + 0 \begin{bmatrix} -2 & 1 \end{bmatrix} = \begin{bmatrix} 3.3 & 2.2 \end{bmatrix}$$

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

Iteración 4

Se presenta a la red p_4 ,

$$W^3 = \begin{bmatrix} 3.3 & 2.2 \end{bmatrix} \quad b^3 = \begin{bmatrix} 2.5 \end{bmatrix}$$

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

$$a = \text{hardlims}(6.9) = 1$$

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

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

$$W^4 = W^3 + e P^T \quad W^4 = \begin{bmatrix} 3.3 & 2.2 \end{bmatrix} - 2 \begin{bmatrix} 0 & 2 \end{bmatrix} = \begin{bmatrix} 3.3 & -1.8 \end{bmatrix}$$

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

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

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

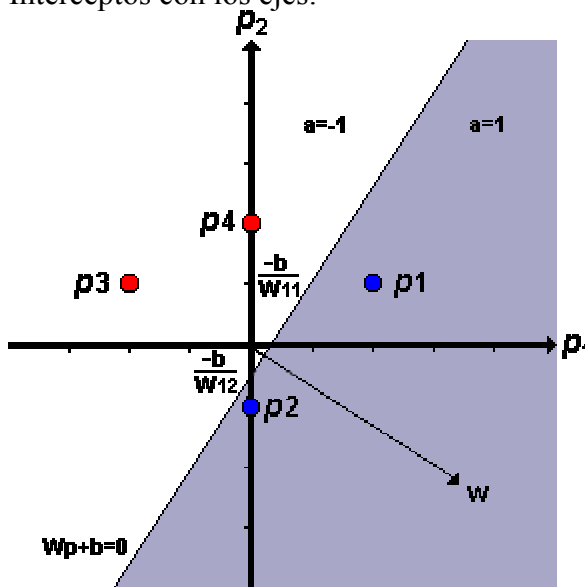


Figura 2.1.13 Característica de decisión final

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, analizaremos un caso en el que no lo sea, el caso de la compuerta XOR (nuestro tercer caso a implementar con el Perceptrón), el cual se visualiza en la figura 2.1.14

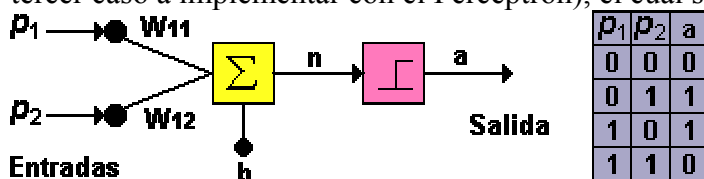


Figura 2.1.14 Compuerta XOR

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

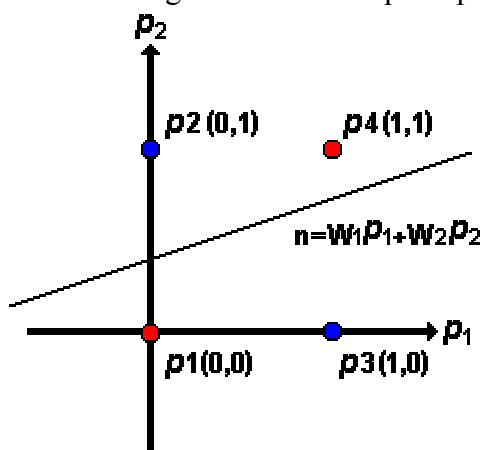


Figura 2.1.15 Plano formado por el problema de la XOR

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 esta en capacidad de clasificar correctamente los patrones de esta función, debido a esta limitación del Perceptrón y a su amplia publicación en el libro de Minsky y Papert, el estudio de las redes neuronales se estanco durante casi 20 años.

El proceso para determinar si un problema es linealmente separable o no, se realiza gráficamente sin problema, cuando los patrones de entrada generan un espacio de dos dimensiones, como en el caso de las funciones AND, OR o de la XOR; sin embargo, esta visualización se dificulta cuando el conjunto de patrones de entrada es de tres dimensiones,

y resulta imposible de observar gráficamente cuando los patrones de entrada son de dimensiones superiores; en este caso se requiere plantear condiciones de desigualdad que permitan comprobar la separabilidad lineal de los patrones, esto se realiza con base en la ecuación de salida del Perceptrón

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

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

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

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

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

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

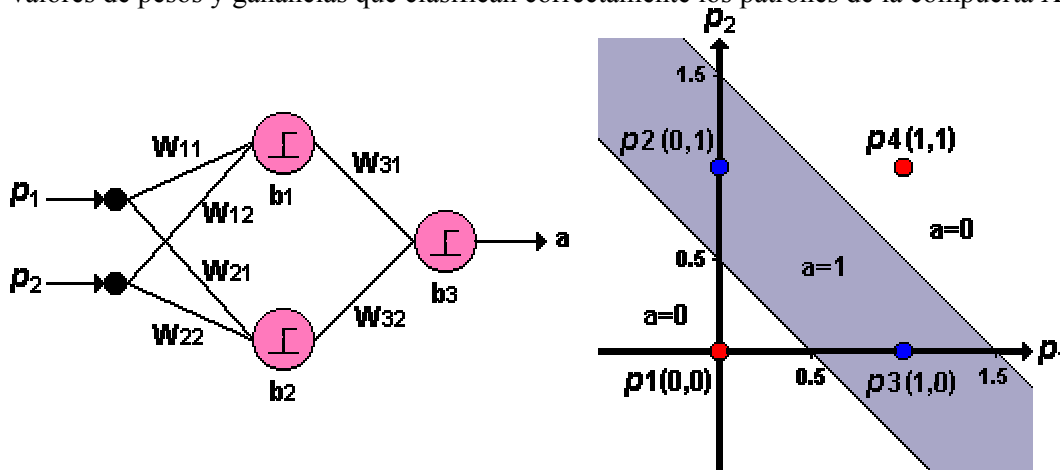


Figura 2.1.16 Perceptrón multicapa para la XOR

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

$$\begin{aligned} w_{11} &= 1 \\ w_{21} &= 1 \\ w_{31} &= 1 \end{aligned}$$

$$\begin{aligned} w_{12} &= 1 \\ w_{22} &= 1 \\ w_{32} &= -1.5 \end{aligned}$$

$$b_1=0.5 \quad b_2=1.5 \quad b_3=0.5$$

2.1.5 Perceptrón multicapa: En el problema de la función XOR explicamos como un Perceptrón multicapa había sido implementado para hallar una solución, el esquema general de un Perceptrón multicapa puede generalizarse 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.

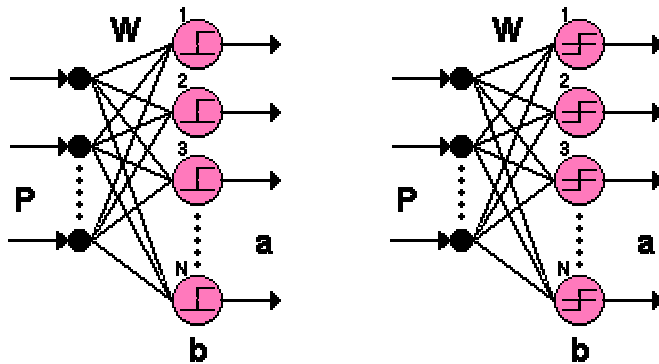


Figura 2.1.17 Conexiones del Perceptrón

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

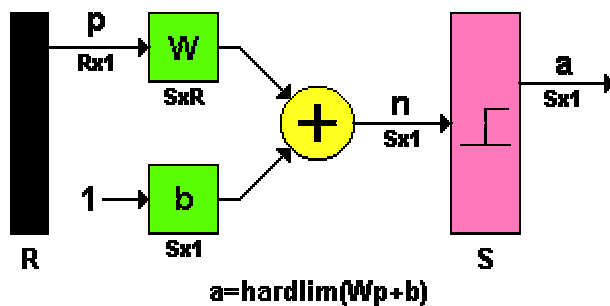


Figura 2.1.18 Notación compacta para la red tipo Perceptrón

La salida de la red está dada por:

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

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. 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.

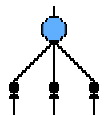
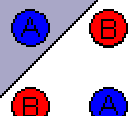
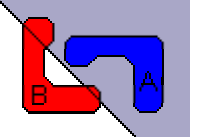

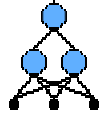
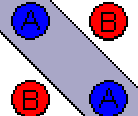
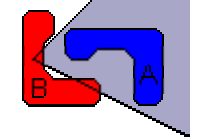
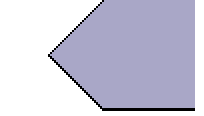
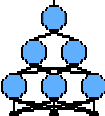
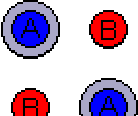
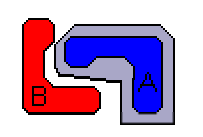
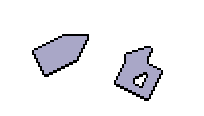
Estructura	Regiones de Decisión	Problema de la XOR	Clases con Regiones Mezcladas	Formas de Regiones más Generales
1 Capa 	Medio Plano Limitado por un Hiperplano			
2 Capas 	Regiones Cerradas o Convexas			
3 Capas 	Complejidad Arbitraria Limitada por el Número de Neuronas			

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

A partir de este análisis surge el interrogante respecto a los criterios de selección para las neuronas de las capas ocultas de una red multicapa, este número en general debe ser lo suficientemente grande como para que se forme una región compleja que pueda resolver el problema, sin embargo no debe ser muy grande pues la estimación de los pesos puede ser no confiable para el conjunto de los patrones de entrada disponibles. Hasta el momento no hay un criterio establecido para determinar la configuración de la red y esto depende más bien de la experiencia del diseñador de la misma.

DETALLES DE IMPLEMENTACION

El programa fue desarrollado en el Lenguaje de Programación Visual C++ de Microsoft utilizando técnicas de Programación Orientada a Objetos, con el fin de aplicar y promover el principio de modularidad. El proyecto de la aplicación se deriva de la implementación de tres archivos cuyas funciones se describen en el encabezado de cada uno; también tendremos a lo largo del código, diferentes enunciados que nos irán guiando hacia el fin que cumplen las diferentes variables y funciones que se tienen dentro del programa.

CODIGO FUENTE DEL PROGRAMA: RNA.h, Main.cpp, Lectura.h

```
//*****//
/*
RNA.h:
-----
Archivo de Cabecera que contiene las definiciones básicas para
el funcionamiento de RNA's basadas en perceptron y adalaine.

Materia:
-----
Inteligencia Artificial, Luis Carlos Torres. Fac. Ing. Sis, UNAL.

*/
//*****//

#define POR_LOTES 1
#define PAT_A_PAT 2
#define MIN_PESO -0.3
#define MAX_PESO 0.3
#define PERCEPTRON 1

//*****//
int Heavyside(double x)
{
    int y;
    y = x >= 0.0 ? 1 : 0;
    return y;
}

//*****//
double Aleatorio(double a, double b)
{
    double y, z;
    y = rand();
    z = y / RAND_MAX;

    return (a + ((b - a) * z));
}

//*****//
void entrada_per(int X[5][4]) /* Patrones de entrada para el perceptron */
{
    int i;

    for(i=1; i<=4; i++)
    {
        X[i][0] = -1;
    }
    X[1][1]=0;
    X[1][2]=0;
    X[2][1]=0;
    X[2][2]=1;
    X[3][1]=1;
    X[3][2]=0;
    X[4][1]=1;
    X[4][2]=1;
}

//*****//
void OR(int Y[5][2]) /* Salida deseada perceptron */
{

```

```

Y[1][1]=0;
Y[2][1]=1;
Y[3][1]=1;
Y[4][1]=1;
}

//*****//
void XOR(int Y[5][2])    /* Salida deseada perceptron */
{
    Y[1][1]=0;
    Y[2][1]=1;
    Y[3][1]=1;
    Y[4][1]=0;
}

//*****//
void AND(int Y[5][2])    /* Salida deseada perceptron */
{
    Y[1][1]=0;
    Y[2][1]=0;
    Y[3][1]=0;
    Y[4][1]=1;
}

//*****//

class PerceptronSimple
{
private:
    double Lambda;
    int niter;
    int m,n,p;
    int UP[5];
    // int X[5][4];
    // int Y[5][2];

public:
    //*****
    PerceptronSimple(int m, int n, int p, double Lambda,
        int TipoEntre, int niter, int X[5][4], int Y[5][2])
    //*****
    {
        int iter,i,j,r,k;
        int erro;
        double upr,sigmaxwri;
        bool continuar;
        double sigmaW[4][3];
        double W[4][3];

        this->m = m;
        this->n = n;
        this->p = p;
        this->Lambda = Lambda;
        this->niter = niter;

        for (i = 0; i <= m; i++)
            for (j = 1; j <= n; j++)
                W[i][j] = Aleatorio(MIN_PESO,MAX_PESO);

        iter = 0;

        do {
            printf("\nIteracion No.%3d\n",iter+1);

```

```

printf(" X1\t X2\tDeseada\tCalculada\tError\n");
continuar = false;

if (TipoEntre == POR_LOTES) {
    for (i = 0; i <= this->m; i++)
        for (j = 1; j <= this->n; j++)
            sigmaW[i][j] = 0.0;
}

for (k = 1; k <= this->p; k++) {
    for (r = 1; r <= this->n; r++) {
        // calcular la actividad de la neurona r
        // componente presináptica: Multiplicar fila k de
        // la matriz X (patrón actual: k) por columna r de
        // la matriz W (pesos de neurona actual: r)
        upr = 0.0;
        for (i = 0; i <= m; i++)
        {
            upr += ((X[k][i]) * (W[i][r]));
        }

        this->UP[r] = Heavyside(upr);
        erro = Y[k][r] - this->UP[r];

        if (erro != 0) continuar = true;

        for (i = 0; i <= m; i++) {
            sigmawri = this->Lambda * erro * (double)X[k][i];
            if (TipoEntre == PAT_A_PAT) {
                W[i][r] += sigmawri;          // Actualiza peso actual: wri
            } else {
                sigmaW[i][r] += sigmawri;    // Acumula modificación de los
            }
        }
        // de dendritas
        printf("%2d\t", X[k][1]); // Visualizar x1
        printf("%2d\t", X[k][2]); // Visualizar x2
        printf("%5d\t", Y[k][1]); // Visualizar salida deseada
        printf("%5d\t", UP[r]);   // Visualizar salida calculada
        printf("%3d ", abs(erro)); // Visualiza el error
        printf("\n");
    } // de neuronas
} // de patrones

if (TipoEntre == POR_LOTES) {
    // Actualiza todos los pesos
    for (i = 1; i <= n; i++)
        for (j = 0; j <= p; j++) {
            W[j][i] += sigmaW[j][i];
        }
}

iter++;
} while (iter < this->niter && continuar);

printf("\n");
if (!continuar) {
    printf("Aprendimos en %d iteraciones!\n", iter);
} else {
    printf("Hicimos %d iteraciones, pero no aprendimos!\n", iter);
}
}

```

pesos

```

};

//*****//
/*
lectura.h:
-----
Archivo de Cabecera que contiene funciones propias de lectura por
teclado, para fines de validación de entradas.

Materia:
-----
Inteligencia Artificial, Luis Carlos Torres. Fac. Ing. Sis, UNAL.

*/
//*****//

#include <conio.h>
#include <ctype.h>

// *****
double leerRealPosi(const char *label,double Iz,double De)
{
    double t;
    char b[80];
    int ch;
    int i;
    int punto;

    if (Iz < 0) {
        printf("Cota inferior negativa!\n");
        exit (1);
    }

    if (De < Iz) {
        printf("Cota superior debe ser mayor que inferior!\n");
        exit (1);
    }

    do {
        cputs( label );
        punto = 0;
        i = 0;
        b[0] = '\0';
        do {
            do {
                ch = _getch();
                if (ch == '.') punto++;
            } while( (ch < '0' || ch > '9') &&
                (ch != '.' || (ch == '.' && punto > 1))&& ch != '\r' && ch != '\b');

            if (ch == '\b') {
                if (i > 0) {
                    _putch('\b');
                    _putch(' ');
                    _putch('\b');
                    punto -= b[i] == '.' ? 1 : 0;
                    b[i] = '\0';
                    i--;
                }
            } else {
                _putch( ch );
                if ( ch == '\r' ) {
                    _putch('\n');
                }
            }
        } while (1);
    } while (1);
}

```



```

        b[i] = '\0';
    } else {
        b[i++] = ch;
        b[i] = '\0';
    }
}
t = (double)atof(b);
} while ( ch != '\r' || i >= 80);
if (t < Iz || t > De)
    printf("Rango debe estar entre %f y %f\n",Iz,De);
} while (t < Iz || t > De);

return t;
}

// *****
int leerEntPosi(const char *label,int Iz,int De)
{
    int t;
    char b[80];
    int ch;

    int i;

    if (Iz < 0) {
        printf("Cota inferior negativa!\n");
        exit (1);
    }

    if (De < Iz) {
        printf("Cota superior debe ser mayor que inferior!\n");
        exit (1);
    }

    do {
        cputs( label );
        i = 0;
        b[0] = '\0';
        do {
            do {
                ch = _getch();
            } while( (ch < '0' || ch > '9') &&
                ch != '\r' && ch != '\b');

            if (ch == '\b') {
                if (i > 0) {
                    _putch('\b');
                    _putch(' ');
                    _putch('\b');
                    b[i] = '\0';
                    i--;
                }
            } else {
                _putch( ch );
                if ( ch == '\r' ) {
                    _putch('\n');
                    b[i] = '\0';
                } else {
                    b[i++] = ch;
                    b[i] = '\0';
                }
            }
        }
        t = (int)atoi(b);
    } while ( ch != '\r' || i >= 80);
}

```

```

        if (t < Iz || t > De) {
            // printf("%d\n",t);
            printf("Rango debe estar entre %d y %d\n",Iz,De);
        }
    } while (t < Iz || t > De);

    return t;
}

//*****//
/* main.cpp
Programa:
-----
Implementación de un PERCEPTRON para resolver una
función lógica dada (and, or o xor).

Materia:
-----
Inteligencia Artificial, Luis Carlos Torres. Fac. Ing. Sis, UNAL.
*/
//*****//

#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <math.h>
#include "lectura.h"
#include "RNA.h"

//*****//
void main(void)
{
    srand((unsigned)time(NULL));
    int TipoEntre;
    double Lambda;
    int niter;
    int n,p;
    int X[5][4];
    int Y[5][2];
    int op;

    int seguir;
    do {

        printf("\nImplementacion del algoritmo Perceptron en una RNA\n");
        printf("Juan Pablo Sanchez Castellanos, Codigo: 255749\n\n");
        printf("1. Perceptron Simple\n");
        printf("2. Salir\n\n");

        if ((op = leerEntPosi("Que Opcion Desea? ",1,2)) == 2) exit(0);

        niter = leerEntPosi("Numero de Iteraciones Deseadas: ",1,10000000);
        Lambda = leerRealPosi("Parametro de Aprendizaje (Lambda): ",0,1);

        // Numero de estados del perceptron
        int m = 2;

        if (op == PERCEPTRON) {
            // En caso de que sea un perceptron, he aqui la jugada...

            n = 1;
            p = 4;
            entrada_per(X);

```

```

// Determina la función a evaluar
printf("\nFuncion Logica a entrenar:\n\n");
printf("1. AND.\n");
printf("2. OR.\n");
printf("3. XOR.\n\n");
switch(leerEntPosi("Que Opcion Desea? ",1,3)) {
case 1:
    AND(Y);
    break;
case 2:
    OR(Y);
    break;
case 3:
    XOR(Y);
    break;
}

// Determina tipo de entrenamiento
printf("\nTipo de Entrenamiento:\n\n");
printf("1. POR LOTES\n");
printf("2. PATRON A PATRON\n\n");
TipoEntre = leerEntPosi("Que Opcion Desea? ",1,2);

class PerceptronSimple *P = new PerceptronSimple(m, n, p, Lambda, TipoEntre, niter, X, Y);
// exit(1);

} else {
}

printf("\n\n");
seguir = leerEntPosi("Desea seguir probando las RNA? (1: Si, 2: No) ",1,2);
} while (seguir == 1);
}

```

INSTRUCCIONES DE OPERACIÓN

Ejecutar el archivo Main.exe, presionar la tecla ENTER para ir siguiendo las instrucciones de la pantalla, escoger las opciones deseadas hasta que sea mostrada la solución del problema escogido.

CONCLUSIONES

Es un modelo de red neuronal muy simple, con lo que no se pueden llevar a cabo de una manera correcta los objetivos planteados por Rosenblatt (reconocimiento de patrones visuales) u otros de los problemas planteados dentro del entorno que cubre las RNA. Sin embargo, creó los principios sobre los que se fundamentan los modelos de redes neuronales posteriores, especialmente en lo que concierne a la estructura y funcionamiento de los elementos de proceso.

El Perceptrón es un modelo de red neuronal de aprendizaje por refuerzo (se fija sólo en si la salida está bien o mal, pero no en el valor de la discrepancia entre ambas), síncrono y sin realimentación.

Las aplicaciones del Perceptrón se limitan a los problemas separables linealmente como puede ser la separación de elementos en varias clases (siempre que sean linealmente separables una de otra). En esta ocasión, tendremos tantos elementos Perceptrón en la capa de salida como clases existan. La entrada presentada pertenecerá a aquella clase cuya salida sea 1, lo que la diferenciará de las demás que serán 0.

RECOMENDACIONES

El algoritmo de Adaline utiliza una regla conocida como regla Delta creada por Widrow y Hoff, esta regla en tiempo de entrenamiento actualiza o cambia los pesos de las conexiones neuronales para minimizar la diferencia entre la entrada de la red a la unidad de salida y el valor deseado, es decir, aproxima los valores obtenidos hacia los valores deseados. Esto se logra reduciendo el error en cada patrón a la vez. También se pueden realizar actualizaciones de pesos de patrones por lotes, es decir, corregir el peso de las conexiones de un conjunto de patrones en vez de uno por uno.

El Adaline calcula el error a partir de la entrada total, y el perceptrón utiliza la salida.

Yo recomendaría implementar Adaline en vez del Perceptrón, ya que este no solo garantiza una solución válida para problemas linealmente separables, sino que produce un error mínimo, según el criterio utilizado LMS (Least Means Square error). Es claro entonces que Adaline en la mayoría de los casos es más eficiente que el Perceptrón, claro está que este se basó en los principios de su antecesor (el Perceptrón), así que no debemos menospreciar éste.

BIBLIOGRAFÍA

Internet:

http://www.infor.uva.es/biometria/Documentos/informes_uva/EstadoArte/EstadoArte/node12.html

http://www.mathworks.com/access/helpdesk/help/toolbox/nnet/nnet_tocframe.shtml

<http://hn.ruv.itesm.mx/hn/pgit/get/ago02/cs5012/foros/moderado/52.html?inline=1&nogifs>