

**UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES**

Grupo de Inteligencia Artificial (GIA)

INTRODUCCION A LAS REDES NEURONALES ARTIFICIALES

Fernando Tanco

Redes Neuronales Artificiales

1. Introducción

Resulta irónico pensar que máquinas de cómputo 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, sino que 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 ventajas de las redes neuronales son:

- *Aprendizaje adaptativo*. Capacidad de aprender a realizar tareas basadas en un entrenamiento o una experiencia inicial.
- *Autoorganización*. Una red neuronal puede crear su propia organización o representación de la información que recibe mediante una etapa de aprendizaje.
- *Generalización*. Facultad de las redes neuronales de responder apropiadamente cuando se les presentan datos o situaciones a los que no habían sido expuestas anteriormente.
- *Tolerancia a fallos*. La destrucción parcial de una red conduce a una degradación de su estructura; sin embargo, algunas capacidades de la red se pueden retener, incluso sufriendo gran daño. Con respecto a los datos, las redes neuronales pueden aprender a reconocer patrones con ruido, distorsionados o incompletos.
- *Operación en tiempo real*. Los computadores neuronales pueden ser realizados en paralelo, y se diseñan y fabrican máquinas con hardware especial para obtener esta capacidad.
- *Fácil inserción dentro de la tecnología existente*. Se pueden obtener chips especializados para redes neuronales que mejoran su capacidad en ciertas tareas. Ello facilita la integración modular en los sistemas existentes.

2. Panorama histórico

Alan Turing, en 1936, fue el primero en estudiar el cerebro como una forma de ver el mundo de la computación; sin embargo, los primeros teóricos que concibieron los fundamentos de la computación neuronal fueron Warren McCulloch, un neurofisiólogo, y Walter Pitts, un matemático, quienes, en 1943, lanzaron una teoría acerca de la forma de trabajar de las neuronas. Ellos modelaron una red neuronal simple mediante circuitos eléctricos. Otro importante libro en el inicio de las teorías de las redes neuronales fue escrito en 1949 por Donald Hebb, *La organización del comportamiento*, en el que establece una conexión entre psicología y fisiología.

En 1957, Frank Rosenblatt comenzó el desarrollo del Perceptrón, es el modelo más antiguo de red neuronal, y se usa hoy en día de varias formas para la aplicación de reconocedor de patrones. Este modelo era capaz de generalizar; es decir, después de haber aprendido una serie de patrones era capaz de reconocer otros similares, aunque no se le hubieran presentado anteriormente. Sin embargo, tenía una serie de limitaciones, quizás la más conocida era la incapacidad para resolver el problema de la función OR-exclusiva y, en general, no era capaz de clasificar clases no separables linealmente.

En 1959, Bernard Widrow y Marcial Hoff, de Stanford, desarrollaron el modelo ADALINE (ADaptive LINEar Elements). Esta fue la primera red neuronal aplicada a un problema real (filtros adaptativos para eliminar ecos en las líneas telefónicas) y se ha usado comercialmente durante varias décadas.

Uno de los mayores investigadores de las redes neuronales desde los años 60 hasta nuestros días es Stephen Grossberg (Universidad de Boston). A partir de su extenso conocimiento fisiológico, ha escrito numerosos libros y desarrollado modelos de redes neuronales. Estudió los mecanismos de la percepción y la memoria. Grossberg realizó en 1967 una red, *Avalancha*, que consistía en elementos discretos con actividad que varía con el tiempo que satisface ecuaciones diferenciales continuas, para resolver actividades tales como reconocimiento continuo del habla y aprendizaje del movimiento de los brazos de un robot.

En 1969 surgieron numerosas críticas que frenaron, hasta 1982, el crecimiento que estaban experimentando las investigaciones sobre redes neuronales. Marvin Minsky y Seymour Papert, del MIT, publicaron un libro, *Perceptrons*, que además de contener un análisis matemático detallado del Perceptrón, consideraba que la extensión a Perceptrones multinivel (el Perceptrón original solo poseía una capa) era completamente estéril. Las limitaciones del Perceptrón eran importantes, sobre todo su incapacidad para resolver muchos problemas interesantes. Esta fue una de las razones por la cual la investigación en redes neuronales quedó rezagada por más de 10 años.

A pesar del libro *Perceptrons*, algunos investigadores continuaron con su trabajo. Tal fue el caso de James Anderson, que desarrolló un modelo lineal llamado Asociador Lineal, que consistía en

elementos integradores lineales (neuronas) que sumaban sus entradas. También desarrolló una potente extensión del Asociador Lineal llamada Brain-State-in-a-Box (BSB) en 1977.

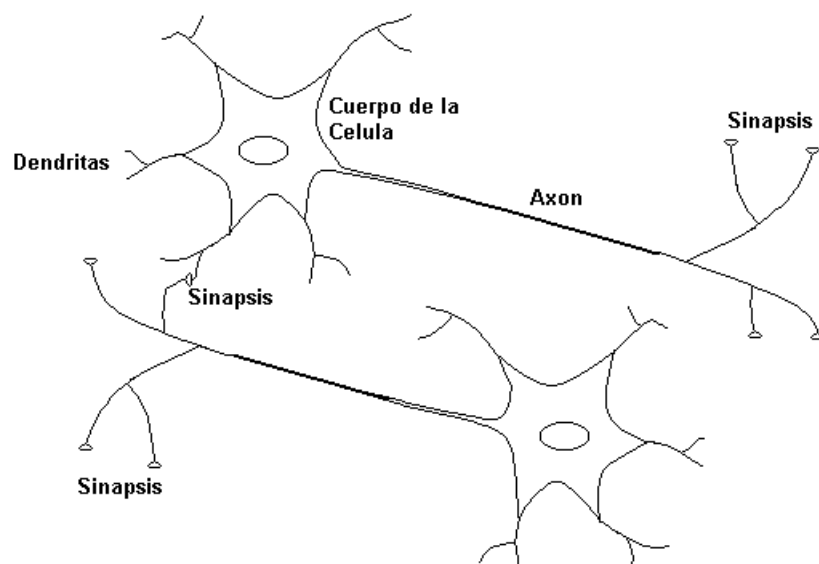
En Europa y Japón, las investigaciones también continuaron. Kunihiro Fukushima desarrolló en 1980 el Neocognitrón, un modelo de red neuronal para el reconocimiento de patrones visuales. Teuvo Kohonen, un ingeniero electrónico de la universidad de Helsinki, desarrolló un modelo similar al de Anderson pero independientemente.

En 1982 comenzó a resurgir el interés por las redes neuronales gracias a dos trabajos importantes. John Hopfield presentó su trabajo (basado en la física estadística), en el cual describe con claridad y rigor matemático una red a la que ha dado su nombre, que es una variación del Asociador Lineal, pero, además, mostró cómo tales redes pueden trabajar y qué pueden hacer. El otro trabajo pertenece a Teuvo Kohonen, con un modelo con capacidad para formar mapas de características de manera similar a como ocurre en el cerebro, este modelo tenía dos variantes denominadas LVQ (Learning Vector Quantization) y SOM (Self-Organizing Map).

En 1986, el desarrollo del algoritmo back-propagation fue dado a conocer por Rumelhart, Hinton y Williams. Ese mismo año, el libro *Parallel Distributed Processing*, fue publicado por Rumelhart y McClelland, siendo este libro la mayor influencia para la aplicación generalizada del algoritmo back-propagation.

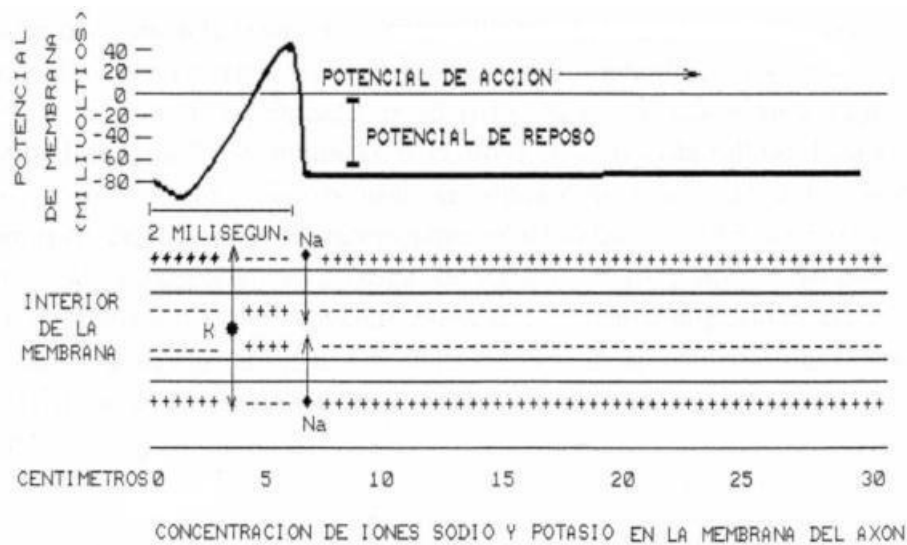
3. El modelo biológico

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 siguiente figura:

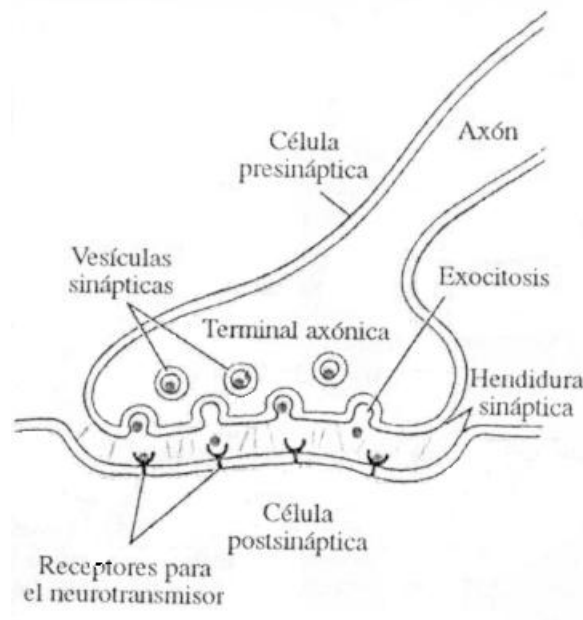


Todas las neuronas conducen la información de forma similar, esta viaja a lo largo de axones en breves impulsos eléctricos, denominados potenciales de acción; los potenciales de acción que alcanzan una amplitud máxima de unos 100 mV y duran un par de ms, son resultado del desplazamiento a través de la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

potasio (carga negativa) que se desplazan desde el fluido intracelular al extracelular. En la siguiente figura se observa la propagación del impulso eléctrico a lo largo del axón.



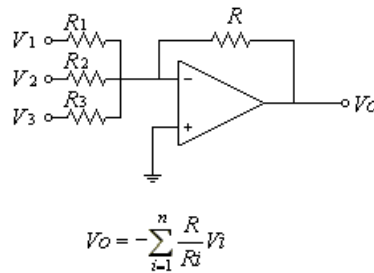
Los potenciales de acción, son señales de baja frecuencia conducidas en forma muy lenta, estos no pueden saltar de una célula a otra, la comunicación entre neuronas viene siempre mediada por transmisores químicos que son liberados en las sinápsis, son los llamados neurotransmisores. Existen dos tipos de sinapsis: a) *las sinapsis excitadoras*, cuyos neurotransmisores provocan disminuciones de potencial en la membrana de la célula postsináptica, facilitando la generación de impulsos a mayor velocidad, y b) *las sinapsis inhibidoras*, cuyos neurotransmisores tienden a estabilizar el potencial de la membrana, dificultando la emisión de impulsos.



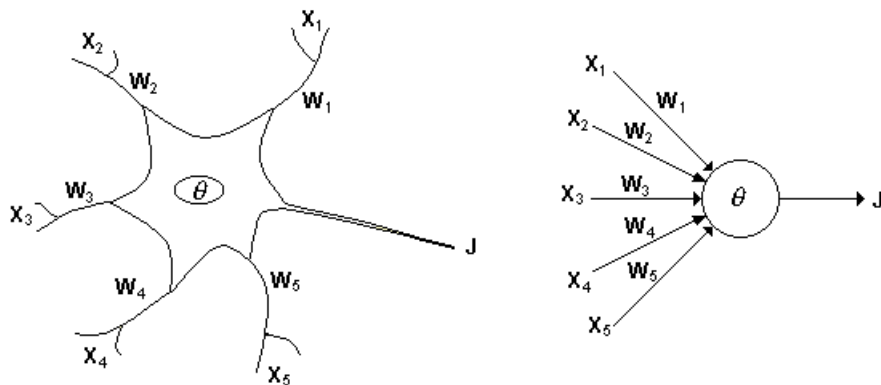
La neurona recibe las entradas procedentes de sinapsis excitadoras e inhibidoras, o sea recibe impulsos amplificados o atenuados. La suma de éstos impulsos en el cuerpo de la célula determinará si será o no estimulada, dependiendo de que si el valor de dicha suma es superior al valor de umbral de generación del potencial de acción.

4. La neurona 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:



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



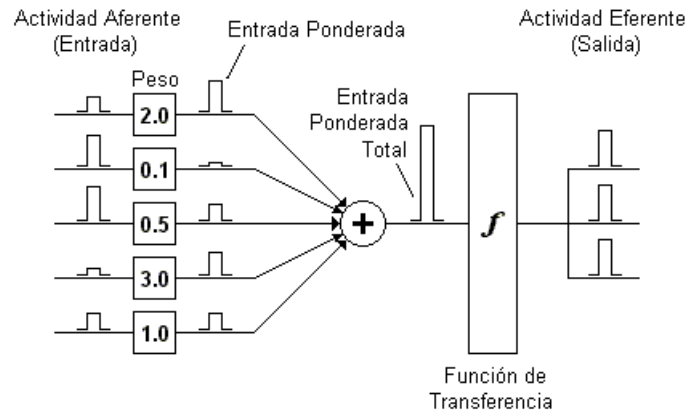
De la observación detallada del proceso biológico se han hallado los siguientes análogos con el sistema artificial:

- 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:

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

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



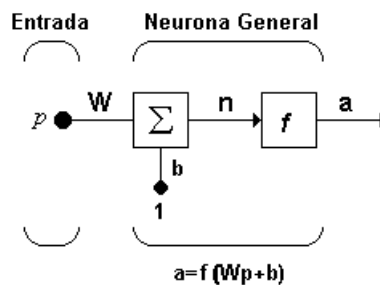
Una vez que se ha calculado la activación del nodo, el valor de salida equivale a

$$x_i = f_i(neta_i)$$

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.

5. Funciones de transferencia

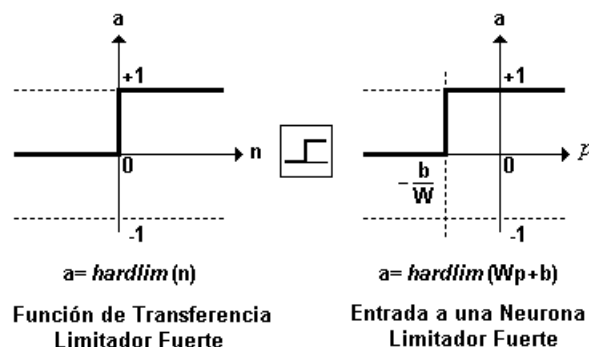
Un modelo más académico que facilita el estudio de una neurona, puede visualizarse en la figura:



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.

Limitador fuerte (Hardlim): la siguiente figura 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

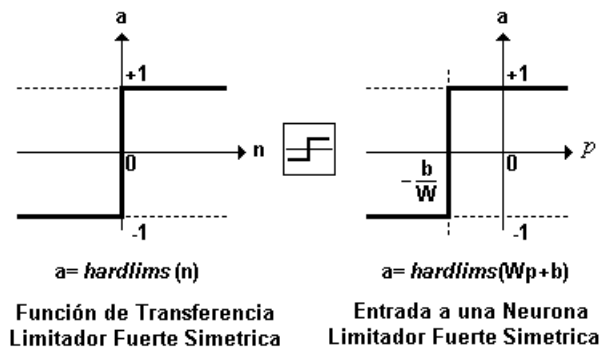
$$a = \begin{cases} 1 & \text{si } n \geq 0 \\ 0 & \text{si } n < 0 \end{cases}$$



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

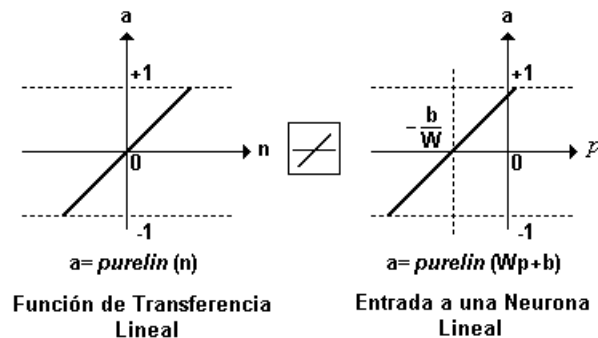
Una modificación de esta función puede verse en la siguiente figura, la que representa la función de transferencia Hardlims que restringe el espacio de salida a valores entre 1 y -1.

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



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

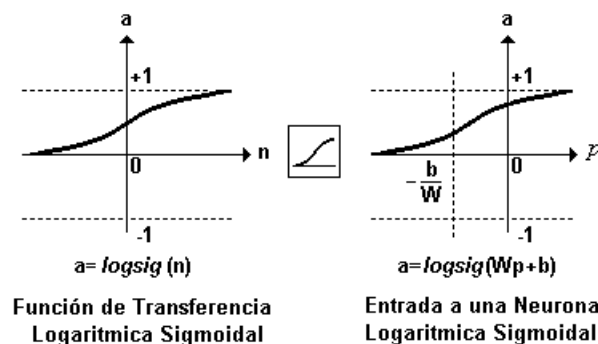
$$a = n$$



En la gráfica del lado derecho 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.









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

$$a = \frac{1}{1 + e^{-n}}$$



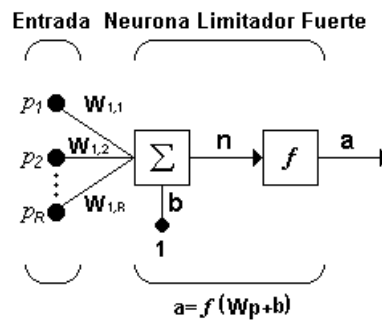
Esta función es comúnmente usada en redes multicapa, como la Backpropagation, en parte porque la función logsig es diferenciable.

La siguiente tabla 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>

6. Topología de una red

Típicamente una neurona tiene más de una entrada; en la siguiente figura 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 .



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 ,

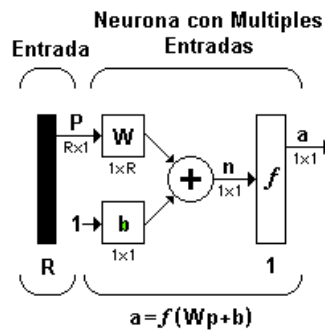
$$n = w_{1,1}p_1 + w_{1,2}p_2 + \dots + w_{1,R}p_R + b$$

Esta expresión puede ser escrita en forma matricial

$$n = Wp + b$$

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 anterior, puede resultar inapropiada y se prefiere emplear la notación abreviada representada en la siguiente figura



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.

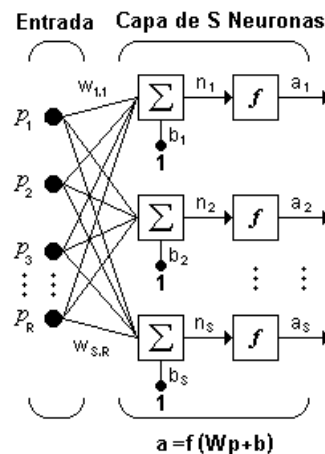
Dentro de una red neuronal, los elementos de procesamiento se encuentran agrupados por capas, una capa es una colección de neuronas; de acuerdo a la ubicación de la capa en la RNA, esta recibe diferentes nombres:

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.

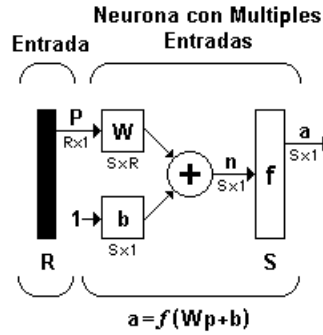
Capas ocultas: Estas capas son aquellas que no tienen contacto con el medio exterior, sus elementos pueden tener diferentes conexiones y son estas las que determinan las diferentes topologías de la red

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 siguiente figura 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.

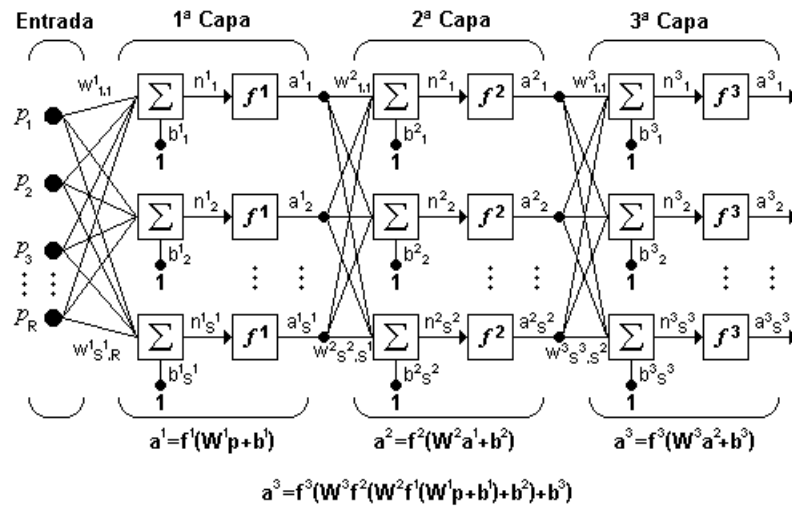


La capa incluye la matriz de pesos, los sumadores, el vector de ganancias, la función de transferencia y el vector de salida. Esta misma capa se observa en notación abreviada en la siguiente figura.

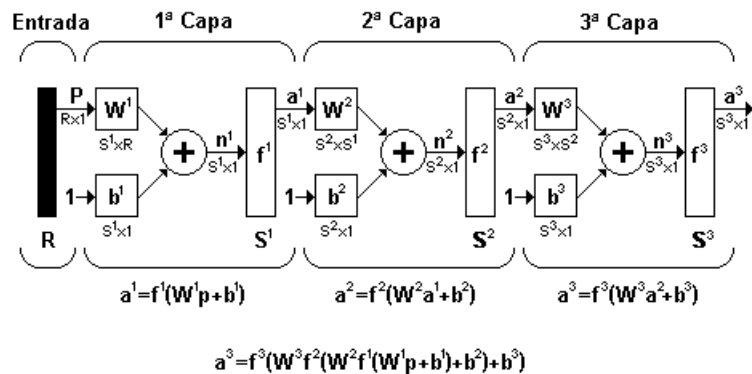


En la figura anterior se han dispuesto los símbolos de las variables de tal manera que describan las características de cada una de ellas, por ejemplo la entrada a la red es el vector p cuya longitud R aparece en su parte inferior, W es la matriz de pesos con dimensiones $S \times R$ expresadas debajo 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 siguientes dos figuras respectivamente.

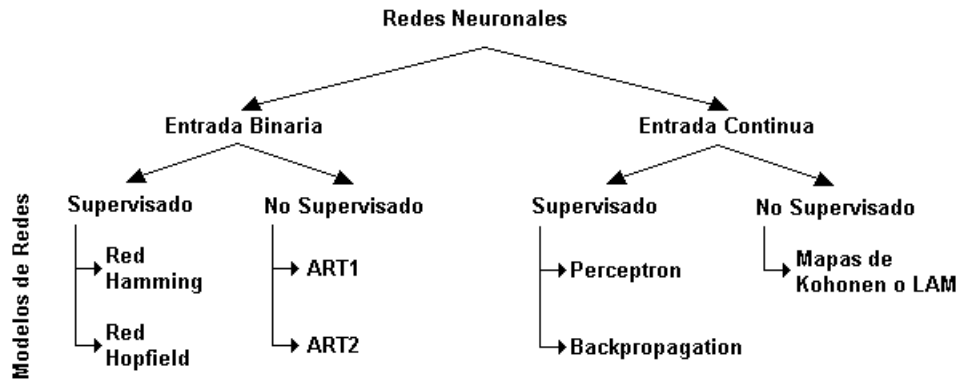


Para esta red se tienen R entradas, S^1 neuronas en la primera capa, S^2 neuronas en la segunda capa, las cuales pueden ser diferentes; las salidas de las capas 1 y 2 son las entradas a las capas 2 y 3 respectivamente, así la capa 2 puede ser vista como una red de una capa con $R = S^1$ entradas, $S^1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



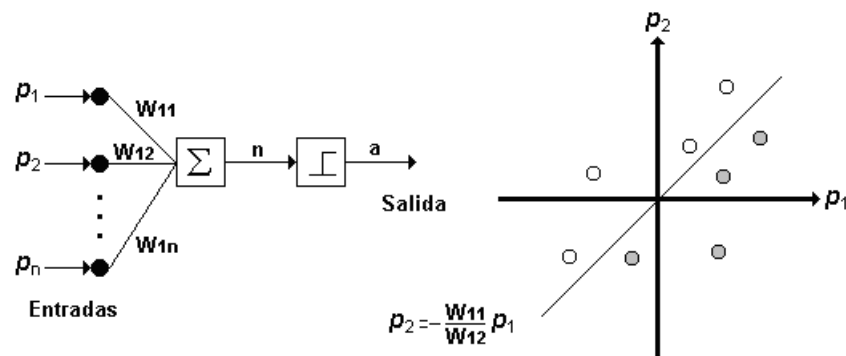
Las redes multicapa son más poderosas que las redes de una sola capa, por ejemplo, una red de dos capas que tenga una función sigmoideal en la primera capa y una función lineal en la segunda, puede ser entrenada para aproximar muchas funciones de forma aceptable, una red de una sola capa no podría hacer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

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 observa en la siguiente figura.



7. Perceptrón

7.1. Estructura de la red



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, 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:

$$W = \begin{bmatrix} W_{1,1} & W_{1,2} & \dots & W_{1,R} \\ W_{2,1} & W_{2,2} & \dots & W_{2,R} \\ \vdots & \vdots & \ddots & \vdots \\ W_{S,1} & W_{S,2} & \dots & W_{S,R} \end{bmatrix}$$

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

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

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

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

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, el caso más conocido es la imposibilidad del Perceptrón de representar la función OR EXCLUSIVA.

7.2. 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:

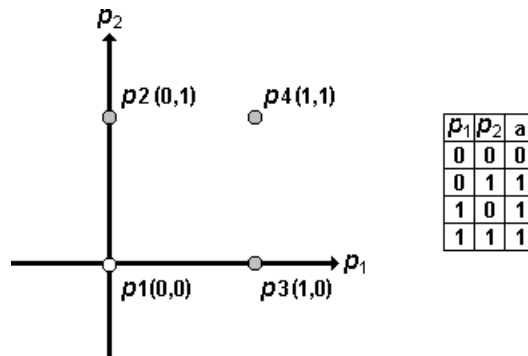
$$\{p_1, t_1\}, \{p_2, t_2\}, \dots, \{p_Q, t_Q\}$$

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:

$$a = f\left(\sum_i w_i p_i\right) = \text{hardlim}\left(\sum_i w_i p_i\right)$$

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.

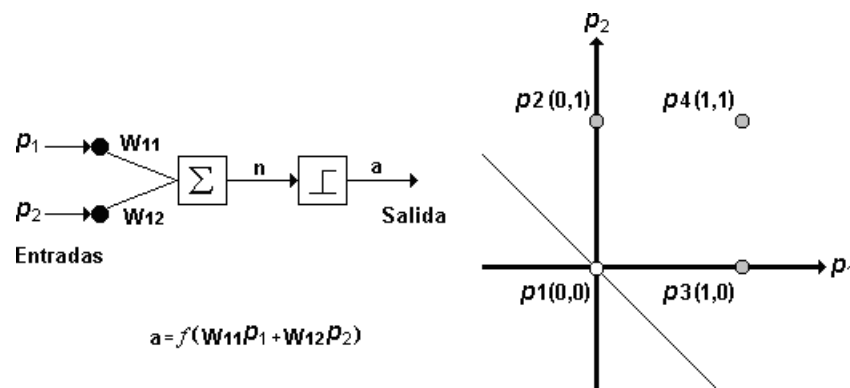
Como ejemplo de funcionamiento de una red neuronal tipo Perceptrón, se solucionará el problema de la función OR, para esta función la red debe ser capaz de devolver a partir de los cuatro patrones de entrada, a qué clase pertenece cada uno; es decir para el patrón 00 debe devolver la clase cero y para los restantes la clase 1, según la siguiente gráfica:



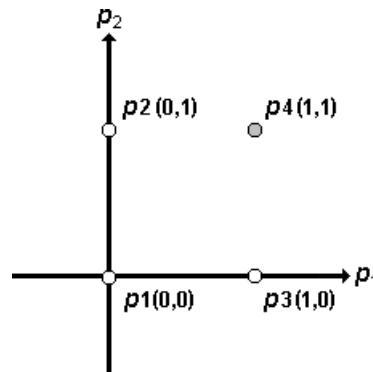
Para este caso las entradas a la red serán valores binarios, la salida de la red esta determinada por:

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

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 *hardlims* (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 siguiente figura, allí puede verse como las posibles rectas pasarán por el origen de coordenadas, por lo que la entrada 00 quedará sobre la propia recta.



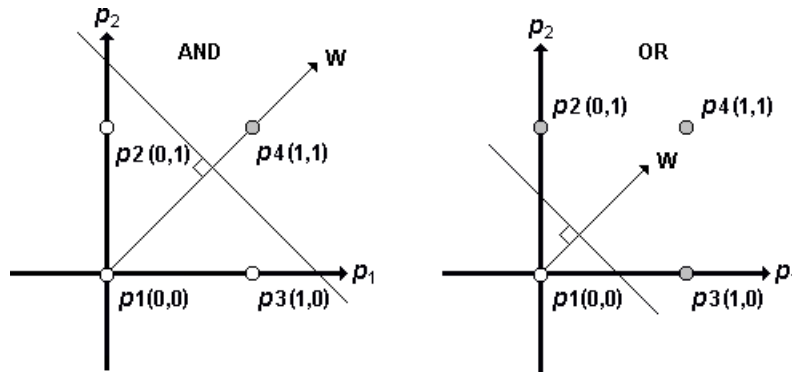
Se aplicará este método para resolver también el problema de la función AND, el cual se describe en la siguiente figura:



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, umbral o bias 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:

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

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



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 para N elementos procesales de entrada y un único elemento procesal de salida 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 pequeños a cada uno de los pesos w_i y al valor b . El valor de bias b puede tomarse como el valor de un peso adicional w_0 asociado con una entrada adicional siempre en 1.
- 2) Se presenta un nuevo patrón de entrada a la red, junto con la salida esperada (t) en forma de pares entrada/salida.
- 3) Se calcula la salida de la red por medio de:

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

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, entonces introduciendo un término error $e = t - a$, la modificación de los pesos será:

$$w_1^{\text{nuevo}} = w_1^{\text{anterior}} + \eta e p = w_1^{\text{anterior}} + \eta (t - a) p$$

y extendiendo la ley a las ganancias:

$$b^{\text{nueva}} = b^{\text{anterior}} + \eta e$$

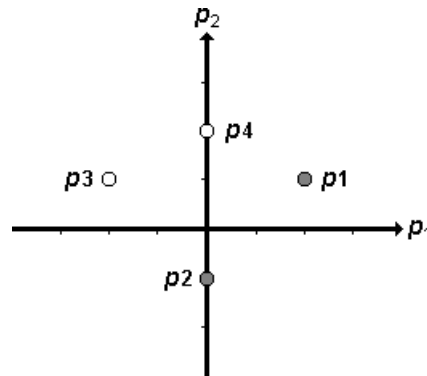
siendo η un factor que define la velocidad de aprendizaje y varía en el rango de 0.0 a 1.0. Este factor debe ser ajustado de forma que satisfaga tanto los requerimientos de aprendizaje rápido como la estabilidad de las estimaciones de los pesos.

- 5) Volver al paso 2. Este proceso se repite hasta que el error e que se produce para cada uno de los patrones es cero o bien menor que un valor preestablecido.

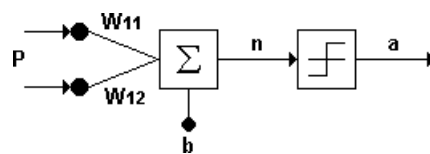
Este algoritmo es extensible al caso de múltiples neuronas en la capa de salida. El Perceptrón será capaz de aprender a clasificar todas sus entradas, en un número finito de pasos, siempre y cuando el conjunto de los patrones de entrada sea linealmente separable.

Para ilustrar la regla de aprendizaje del Perceptrón, se dará solución al problema de clasificación de patrones ilustrado en la siguiente figura con los patrones de entrenamiento:

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



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.



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 con los patrones de entrenamiento, 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.

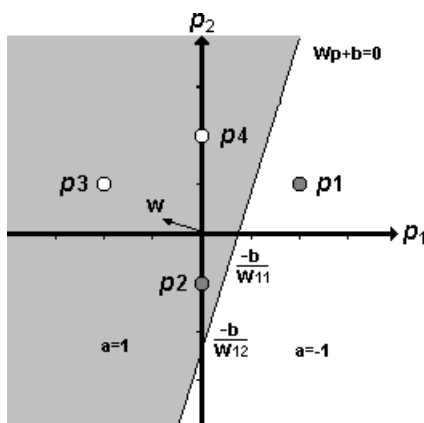
Los valores iniciales asignados aleatoriamente a los parámetros de la red son:

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

Con base en el procedimiento descrito anteriormente, y considerando al factor η igual a 1, 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 siguiente figura, la cual depende de los valores de los pesos y ganancias iniciales.



Intersección con los ejes:

$$\frac{-b}{W_{11}} = 0.71 \quad \frac{-b}{W_{12}} = -2.5$$

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

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

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

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

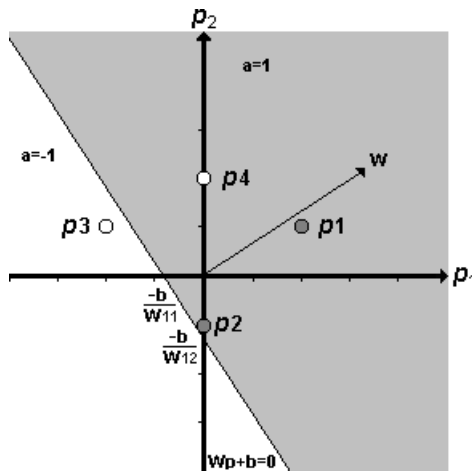
$$e = t - a = 1 - (-1) = 2$$

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

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

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

La iteración 1 lleva a la característica de decisión de la siguiente figura:



Intersección con los ejes:

$$\frac{-b}{W_{11}} = -0.75 \quad \frac{-b}{W_{12}} = -1.13$$

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

- Iteración 2

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

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

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

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

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

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

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

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

- Iteración 3

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

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

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

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

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

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.

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

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

- Iteración 4

Se presenta a la red p_4 ,

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

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

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

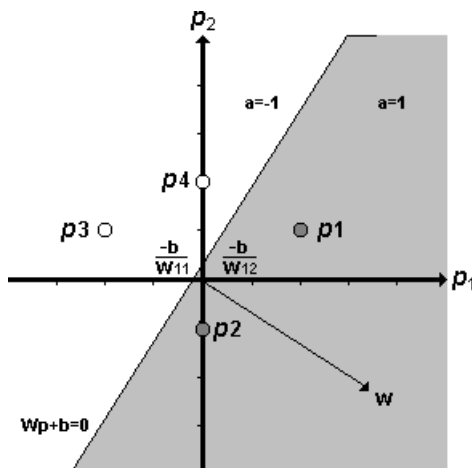
$$e = t - a = -1 - (1) = -2$$

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

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

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

En esta iteración la red se comportará de acuerdo a la característica de decisión de la siguiente figura:



Intersección con los ejes:

$$\frac{-b}{W_{11}} = -0.15 \quad \frac{-b}{W_{12}} = 0.27$$

De esta figura 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 (generalización).

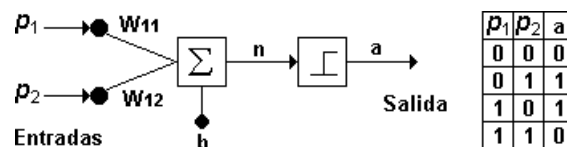
Es de importancia notar que en este caso los patrones de entrada se encuentran en dos dimensiones y por lo tanto es fácil determinar gráficamente cuando han sido clasificados correctamente, en el caso que los patrones se encuentren en tres dimensiones esta visualización se dificulta y en el caso de que 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.

7.3. Limitaciones del Perceptrón

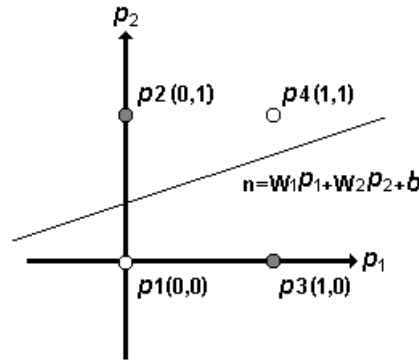
Ya anteriormente se planteó la restricción que existe para los tipos de problemas que una red Perceptrón puede solucionar, como se dijo, esta red puede resolver solamente problemas que sean linealmente separables, esto es problemas cuyas salidas estén clasificadas en dos categorías diferentes y que permitan 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

$$wp + b = 0$$

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 siguiente figura:



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 en la siguiente figura, el problema radica en que no existe ninguna línea recta que separe los patrones de una clase de los de la otra:



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 negros) y los de la otra región como pertenecientes a la clase que posee salida 0 (puntos blancos), 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 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.

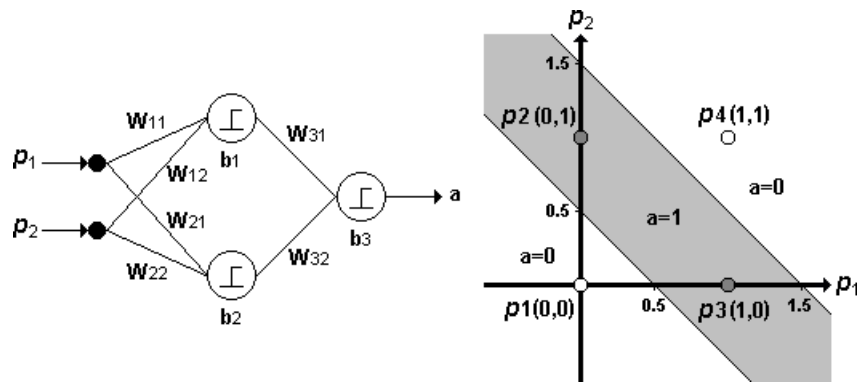
En el caso de la XOR, teniendo en cuenta los valores de la tabla al lado derecho de la figura de la compuera, estas desigualdades se expresan así:

$$\begin{array}{ll} 0 * W_{1,1} + 0 * W_{2,1} + b < 0 & (p_1) \\ 0 * W_{1,1} + 1 * W_{2,1} + b \geq 0 & (p_2) \\ 1 * W_{1,1} + 0 * W_{2,1} + b \geq 0 & (p_3) \\ 1 * W_{1,1} + 1 * W_{2,1} + b < 0 & (p_4) \end{array}$$

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 siguiente figura 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:



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

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

7.4. Perceptr3n multicapa

Un Perceptr3n multicapa es una red con alimentaci3n hacia delante, compuesta de varias capas de neuronas entre la entrada y la salida de la misma, esta red permite establecer regiones de decisi3n mucho m3s complejas que las de dos semiplanos, como lo hace el Perceptr3n de un solo nivel. En el problema de la funci3n XOR se explic3 como un Perceptr3n multicapa hab3a sido implementado para hallar una soluci3n.

Las capacidades del Perceptr3n multicapa con dos y tres capas y con una 3nica neurona en la capa de salida se muestran en la siguiente figura. En la segunda columna se muestra el tipo de regi3n de decisi3n que se puede formar con cada una de las configuraciones, en la siguiente se indica el tipo de regi3n que se formar3a para el problema de la XOR, en las dos 3ltimas columnas se muestran las regiones formadas para resolver el problema de clases mezcladas y las formas m3s generales para cada uno de los casos:

Estructura	Regiones de Decisi3n	Problema de la XOR	Clases con Regiones Mezcladas	Formas de Regiones m3s Generales
1 Capa 	Medio Plano Limitado por un Hiperplano			
2 Capas 	Regiones Cerradas o Convexas			
3 Capas 	Complejidad Arbitraria Limitada por el N3mero de Neuronas			

El Perceptr3n b3sico s3lo puede establecer dos regiones separadas por una frontera lineal en el espacio de entrada de los patrones; un Perceptr3n con dos capas, puede formar cualquier regi3n convexa en este espacio. Las regiones convexas se forman mediante la intersecci3n 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.

8. ADALINE

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

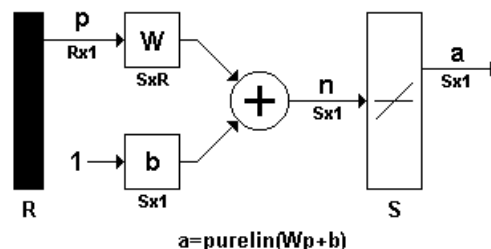
La red Adaline es similar al Perceptrón, excepto en su función de transferencia, la cual es una función de tipo lineal en lugar de un limitador fuerte como en el caso del Perceptrón. La red Adaline presenta la misma limitación del Perceptrón en cuanto al tipo de problemas que pueden resolver, ambas redes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

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

El 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 NEurona.

8.1. Estructura de la red

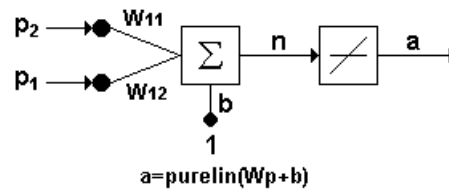
La estructura general de la red tipo Adaline puede visualizarse en la siguiente figura:



La salida de la red está dada por:

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

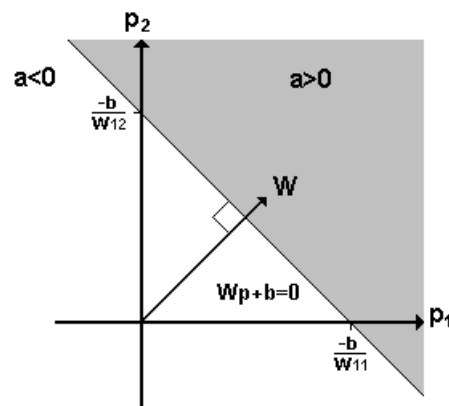
Para una red Adaline de una sola neurona con dos entradas el diagrama corresponde a la figura siguiente:



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

$$\mathbf{w}^T \mathbf{p} + b = 0$$

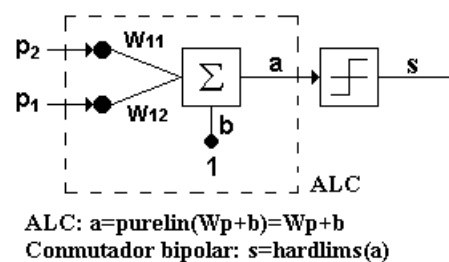
específica la línea que separa en dos regiones el espacio de entrada, como se muestra en la siguiente figura:



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.

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 elemento de procesamiento que realiza la suma de los productos de los vectores de entrada y de pesos se denomina combinador adaptivo lineal (ALC). Una variante consiste en agregar un conmutador bipolar para obtener un único valor de salida, el cual debido a su función de transferencia será +1 si la salida del ALC es positiva o -1 si la salida del ALC es negativa.



8.2. Regla de aprendizaje

El ALC realiza el cálculo de la suma ponderada de las N entradas:

$$a = b + \sum_{j=1}^N w_j p_j$$

Para realizar una simplificación en la función de salida, vamos a considerar el valor umbral b como una conexión ficticia de peso w_0 . Si tenemos en cuenta que para esta entrada se toma el valor de $p_0=1$, la ecuación anterior se puede escribir de la forma:

$$a = w_0 p_0 + \sum_{j=1}^N w_j p_j = \sum_{j=0}^N w_j p_j$$

La regla de aprendizaje de mínimo error cuadrado medio LMS (*Least Mean Squared*) es conocida también como *regla delta* porque trata de minimizar un delta o diferencia entre valor observado y deseado en la salida de la red. Entonces, la regla delta, es un método para hallar el vector de pesos W deseado, el cual deberá ser único y asociar con éxito cada vector del conjunto de vectores o patrones de entrada $\{X^1, X^2, X^3, \dots, X^Q\}$ con su correspondiente valor de salida correcto o deseado $\{t^1, t^2, t^3, \dots, t^Q\}$.

Concretamente, la regla de aprendizaje LMS minimiza el error cuadrado medio, definido como:

$$\langle \varepsilon_k^2 \rangle = \frac{1}{2} \sum_{k=1}^Q \varepsilon_k^2$$

donde Q es el número de vectores de entrada (patrones) que forman el conjunto de entrenamiento, y ε_k la diferencia entre la salida deseada y la obtenida cuando se introduce el patrón k -ésimo, que en el caso del Adaline, se expresa como $\varepsilon_k = (t_k - a_k)$, siendo a_k la salida del ALC; es decir:

$$a_k = P_k \cdot W^T = \sum_{j=0}^N w_j p_{kj}$$

La función error es una función matemática definida en el espacio de pesos multidimensionales para un conjunto de patrones dados. Es una superficie que tendrá muchos mínimos (global y locales), y la regla de aprendizaje va en busca del punto en el espacio de pesos donde se encuentra el mínimo global de esta superficie. Aunque la superficie de error es desconocida, el método de gradiente decreciente consigue obtener información local de dicha superficie a través del gradiente. Con esta información se decide qué dirección tomar para llegar al mínimo global de dicha superficie. Entonces las modificaciones de los pesos son proporcionales al gradiente decreciente de la función error $\Delta w_j = -\alpha (\partial \langle \varepsilon_k^2 \rangle / \partial w_j)$. Por lo tanto, se deriva la función error con respecto a los pesos para ver cómo varía el error con el cambio de los pesos.

Aplicamos la regla de la cadena para el cálculo de dicha derivada:

$$\Delta w_i = -\alpha \frac{\partial \langle \varepsilon_k^2 \rangle}{\partial w_i} = -\alpha \frac{\partial \langle \varepsilon_k^2 \rangle}{\partial a_k} \cdot \frac{\partial a_k}{\partial w_i}$$

Se calcula la primera derivada:

$$\frac{\partial \langle \varepsilon_k^2 \rangle}{\partial a_k} = \frac{\partial [\frac{1}{2}(t_k - a_k)^2]}{\partial a_k} = \frac{1}{2} [2(t_k - a_k)(-1)] = -(t_k - a_k) = -\varepsilon_k$$

Calculamos la segunda derivada de la expresión de Δw_i :

$$\frac{\partial a_k}{\partial w_i} = \frac{\partial [\sum_{j=0}^N (w_j p_{k_j})]}{\partial w_i} = \frac{\partial (w_i p_{k_i})}{\partial w_i} = p_{k_i}$$

Así pues el valor del gradiente del error producido por un patron dado k es:

$$\frac{\partial \langle \varepsilon_k^2 \rangle}{\partial w_i} = -\varepsilon_k p_{k_i}$$

Las modificaciones en los pesos son proporcionales al gradiente descendente de la función error:

$$\begin{aligned} \Delta w_i &= -\alpha(-\varepsilon_k p_{k_i}) = \alpha \varepsilon_k p_{k_i} = \alpha(t_k - a_k) p_{k_i} \\ w_i(t+1) &= w_i(t) + \alpha(t_k - a_k) p_{k_i} \end{aligned}$$

siendo α la constante de proporcionalidad o tasa de aprendizaje.

En notación matricial quedaría:

$$W(t+1) = W(t) + \alpha \varepsilon_k P_k = W(t) + \alpha(t_k - a_k) P_k$$

Esta expresión representa la modificación de los pesos obtenida a partir del algoritmo LMS, y es parecida a la obtenida anteriormente para el caso del Perceptrón. α es el parámetro que determina la estabilidad y la velocidad de convergencia del vector de pesos hacia el valor de error mínimo. Los cambios en dicho vector deben hacerse relativamente pequeños en cada iteración, sino podría ocurrir que no se encontrase nunca un mínimo, o se encontrase solo por accidente, en lugar de ser el resultado de una convergencia sostenida hacia él.

Aunque a simple vista no parece que exista gran diferencia entre los mecanismos de aprendizaje del Perceptrón y del Adaline, este último mejora al del Perceptrón ya que va a ser más sencillo alcanzar el mínimo error, facilitando la convergencia del proceso de entrenamiento.

Resumiendo, el algoritmo de aprendizaje puede expresarse como el siguiente proceso iterativo:

- 1) Se inicializa la matriz de pesos y el valor de la ganancia, por lo general se asignan valores aleatorios pequeños a cada uno de los pesos w_i y al valor b . El valor de bias b puede tomarse como el valor de un peso adicional w_0 asociado con una entrada adicional siempre en 1.
- 2) Se aplica un vector o patrón de entrada, p_k , en las entradas del Adaline.
- 3) Se obtiene la salida lineal $a_k = P_k \cdot W^T = \sum_{j=0}^N w_j p_{k_j}$ y se calcula la diferenciación respecto a la deseada $\varepsilon_k = (t_k - a_k)$.
- 4) Se actualizan los pesos: $W(t+1) = W(t) + \alpha \varepsilon_k P_k = W(t) + \alpha(t_k - a_k) P_k$
- 5) Se repiten los pasos 2 al 4 con todos los vectores de entrada (Q).

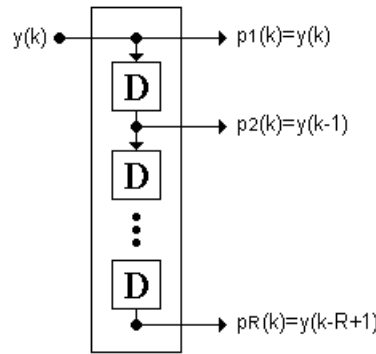
- 6) Si el error cuadrático medio $\langle \varepsilon_k^2 \rangle = \frac{1}{2} \sum_{k=1}^Q \varepsilon_k^2$ es un valor reducido aceptable, termina el proceso de aprendizaje; sino, se repite otra vez desde el paso 2 con todos los patrones.

8.3. Aplicación

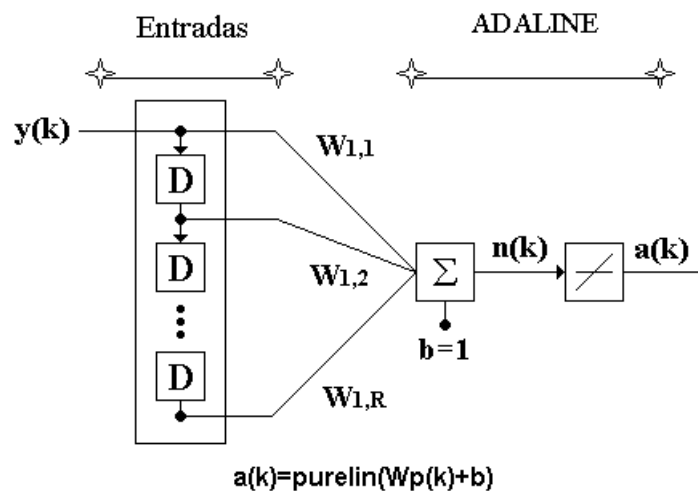
La aplicación más común de la red Adaline es el *filtro adaptativo*, para eliminar el ruido de una señal de entrada. Cuando se diseña un filtro digital por medio de software; con un programa normal, el programador debe saber exactamente como se especifica el algoritmo de filtrado y cuales son los detalles de las características de las señales; si se necesitaran modificaciones, o si cambian las características de la señal, es necesario reprogramar; cuando se emplea una red tipo Adaline, el problema se convierte, en que la red sea capaz de especificar la señal de salida deseada, dada una señal de entrada específica.

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

Para implementar un filtro adaptativo, se debe incorporar el concepto de retardos en línea, el cual se visualiza en la siguiente figura:



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



Cuya salida está dada por:

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

9. Backpropagation

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

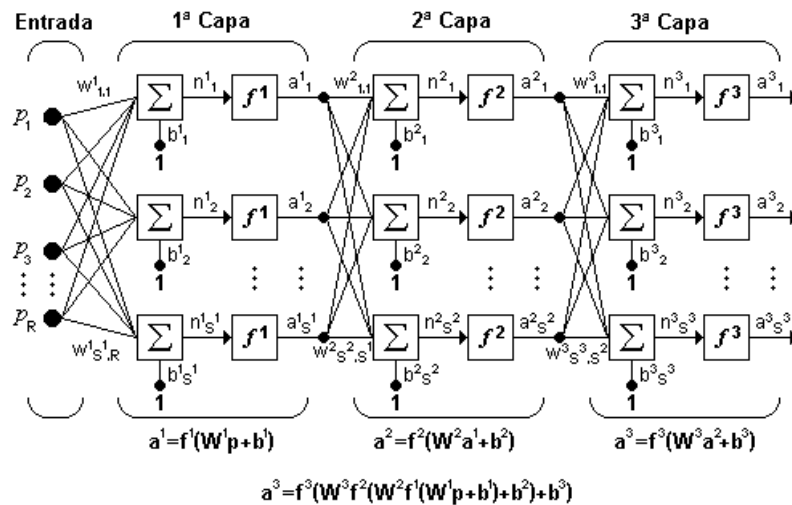
Las salidas de error se propagan hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de la capa oculta que contribuyen directamente a la salida. Sin embargo las neuronas de la capa oculta solo reciben una fracción de la señal total del error, basándose aproximadamente en la contribución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

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

Varias investigaciones han demostrado que, durante el proceso de entrenamiento, la red Backpropagation tiende a desarrollar relaciones internas entre neuronas con el fin de organizar los datos de entrenamiento en clases. Esta tendencia se puede extrapolar, para llegar a la hipótesis consistente en que todas las unidades de la capa oculta de una Backpropagation son asociadas de alguna manera a características específicas del patrón de entrada como consecuencia del entrenamiento. Lo que sea o no exactamente la asociación puede no resultar evidente para el observador humano, lo importante es que la red ha encontrado una representación interna que le permite generar las salidas deseadas cuando se le dan las entradas, en el proceso de entrenamiento. Esta misma representación interna se puede aplicar a entradas que la red no haya visto antes, y la red clasificará estas entradas según las características que compartan con los ejemplos de entrenamiento.

1.1. Estructura de la red

La estructura típica de una red multicapa se observa en la siguiente figura:

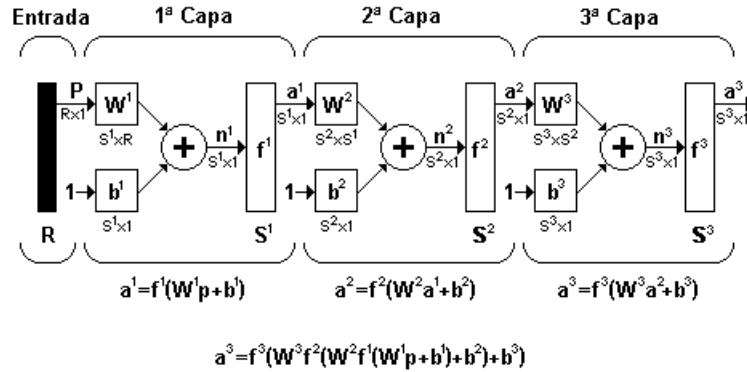


Puede notarse que esta red de tres capas equivale a tener tres redes tipo Perceptrón en cascada; la salida de la primera red, es la entrada a la segunda y la salida de la segunda red es la entrada a la

tercera. Cada capa puede tener diferente número de neuronas, e incluso distinta función de transferencia.

En esta figura, 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: $R : S^1 : S^2 : S^3$, donde S representa el número de neuronas y el exponente representa la capa a la cual la neurona corresponde.

La notación de la figura anterior 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 siguiente figura:



1.2. 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:

$$\{p_1, t_1\}, \{p_2, t_2\}, \dots, \{p_Q, t_Q\} \quad (1)$$

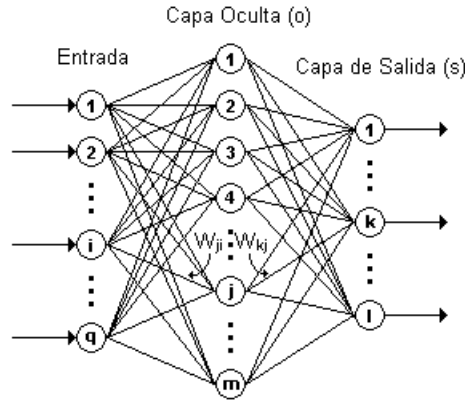
Donde p_Q es una entrada a la red y t_Q es la correspondiente salida deseada para el patrón q-ésimo. El algoritmo debe ajustar los parámetros de la red para minimizar el error cuadrático medio.

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 cuadrático medio 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.



Es importante aclarar que en esta figura:

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):

$$P = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_i \\ \vdots \\ p_q \end{bmatrix} \quad (2)$$

Cuando se le presenta a la red un patrón de entrenamiento, este se propaga a través de las conexiones existentes produciendo una entrada neta n en cada una de 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 (3), nótese que la entrada neta es el valor justo antes de pasar por la función de transferencia:

$$n_j^o = \sum_{i=1}^q W_{ji}^o p_i + b_j^o \quad (3)$$

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

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

b_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 , dada por la ecuación (4):

$$a_j^o = f^o \left(\sum_{i=1}^q W_{ji}^o p_i + b_j^o \right) \quad (4)$$

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 m componentes) son las entradas a los pesos de conexión de la capa de salida, este comportamiento esta descrito por la ecuación (5):

$$n_k^s = \sum_{j=1}^m W_{kj}^s a_j^o + b_k^s \quad (5)$$

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 (6):

$$a_k^s = f^s(n_k^s) \quad (6)$$

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

Reemplazando (5) en (6) se obtiene la salida de la red en función de la entrada neta y de los pesos de conexión con la última capa oculta:

$$a_k^s = f^s\left(\sum_{j=1}^m W_{kj}^s a_j^o + b_k^s\right) \quad (7)$$

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 (8):

$$\delta_k = (t_k - a_k^s) \quad (8)$$

El error debido a cada patrón p propagado esta dado por (9):

$$ep^2 = \frac{1}{2} \sum_{k=1}^s (\delta_k)^2 \quad (9)$$

ep^2 : Error cuadrático medio 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 cuadrático medio total descrito en (10):

$$e^2 = \sum_{p=1}^r ep^2 \quad (10)$$

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:

$$W_{k+1} = W_k - \alpha \nabla ep^2 \quad (11)$$

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

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

$$-\frac{\partial ep^2}{\partial W_{kj}^s} = -\frac{\partial}{\partial W_{kj}^s} \left(\frac{1}{2} \sum_{k=1}^l (t_k - a_k^s)^2 \right) = (t_k - a_k^s) \times \frac{\partial a_k^s}{\partial W_{kj}^s} \quad (12)$$

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

$\frac{\partial a_k^s}{\partial W_{kj}^s}$: Derivada de la salida de la neurona k de la capa de salida respecto, al peso W_{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 (6) puede verse que la salida de la red a_k^s esta explícitamente en función de n_k^s y de la ecuación (5) puede verse que n_k^s esta explícitamente en función de W_{kj}^s , considerando esto se genera la ecuación (13):

$$\frac{\partial a_k^s}{\partial W_{kj}^s} = \frac{\partial a_k^s}{\partial n_k^s} \times \frac{\partial n_k^s}{\partial W_{kj}^s} \quad (13)$$

Tomando la ecuación (13) y reemplazándola en la ecuación (12) se obtiene:

$$-\frac{\partial ep^2}{\partial W_{kj}^s} = (t_k - a_k^s) \times \frac{\partial a_k^s}{\partial n_k^s} \times \frac{\partial n_k^s}{\partial W_{kj}^s} \quad (14)$$

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

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

Reemplazando en la ecuación (14) las derivadas de las ecuaciones (5) y (6) se obtiene:

$$-\frac{\partial ep^2}{\partial W_{kj}^s} = (t_k - a_k^s) \times f'^s(n_k^s) \times a_j^o \quad (15)$$

Como se observa en la ecuación (15) 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:

logsig: $f(n) = \frac{1}{1 + e^{-n}}$ $f'(n) = f(n)(1 - f(n))$ $f''(n) = a(1 - a)$ (16)

$$\text{tansig: } f(n) = \frac{e^n - e^{-n}}{e^n + e^{-n}} \quad f'(n) = 1 - (f(n))^2 \quad f''(n) = (1 - a^2) \quad (17)$$

$$\text{purelin: } f(n) = n \quad f'(n) = 1 \quad (18)$$

De la ecuación (15), los términos del error para las neuronas de la capa de salida están dados por la ecuación (19), la cual se le denomina comúnmente sensibilidad de la capa de salida.

$$\delta_k^s = (t_k - a_k^s) f'^s(n_k^s) \quad (19)$$

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 (19) se procede a encontrar el error en la capa oculta el cual esta dado por:

$$-\frac{\partial ep^2}{\partial W_{ji}^o} = -\frac{\partial}{\partial W_{ji}^o} \left(\frac{1}{2} \sum_{k=1}^l (t_k - a_k^s)^2 \right) = \sum_{k=1}^l (t_k - a_k^s) \times \frac{\partial a_k^s}{\partial W_{ji}^o} \quad (20)$$

Para calcular el último término de la ecuación (20) se debe aplicar la regla de la cadena en varias ocasiones como se observa en la ecuación (21) 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:

$$\frac{\partial a_k^s}{\partial W_{ji}^o} = \frac{\partial a_k^s}{\partial n_k^s} \times \frac{\partial n_k^s}{\partial a_k^o} \times \frac{\partial a_k^o}{\partial n_j^o} \times \frac{\partial n_j^o}{\partial W_{ji}^o} \quad (21)$$

Todos los términos de la ecuación (21) son derivados respecto a variables de las que dependan explícitamente, reemplazando (21) en (20) tenemos:

$$-\frac{\partial ep^2}{\partial W_{ji}^o} = \sum_{k=1}^l (t_k - a_k^s) \times \frac{\partial a_k^s}{\partial n_k^s} \times \frac{\partial n_k^s}{\partial a_k^o} \times \frac{\partial a_k^o}{\partial n_j^o} \times \frac{\partial n_j^o}{\partial W_{ji}^o} \quad (22)$$

Tomando las derivas de las ecuaciones (3) (4) (5) (6) y reemplazándolas en la ecuación (22) se obtiene la expresión del gradiente del error en la capa oculta:

$$-\frac{\partial ep^2}{\partial W_{ji}^o} = \sum_{k=1}^l (t_k - a_k^s) \times f'^s(n_k^s) \times W_{kj}^s \times f'^o(n_j^o) \times p_i \quad (23)$$

Reemplazando la ecuación (19) en la ecuación (23) se tiene:

$$-\frac{\partial ep^2}{\partial W_{ji}^o} = \sum_{k=1}^l \delta_k^s \times W_{kj}^s \times f'^o(n_j^o) \times p_i \quad (24)$$

Los términos del error para cada neurona de la capa oculta esta dado por la ecuación (25), este término también se denomina sensibilidad de la capa oculta:

$$\delta_j^o = f'^o(n_j^o) \times \sum_{k=1}^l \delta_k^s W_{kj}^s \quad (25)$$

Luego de encontrar el valor del gradiente del error se procede a actualizar los pesos de todas las capas empezando por la de salida, para la capa de salida la actualización de pesos y ganancias esta dada por (26) y (27).

$$W_{kj}(t+1) = W_{kj}(t) + \alpha \delta_k^s a_j^o \quad (26)$$

$$b_k(t+1) = b_k(t) + \alpha \delta_k^s \quad (27)$$

α : tasa (o velocidad) 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 (28) y (29):

$$W_{ji}(t+1) = W_{ji}(t) + \alpha \delta_j^o p_i \quad (28)$$

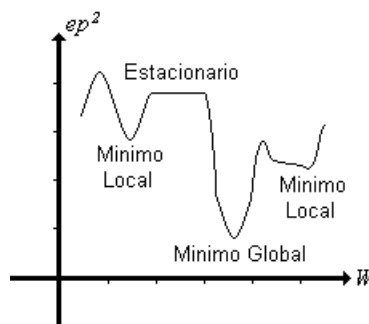
$$b_j(t+1) = b_j(t) + \alpha \delta_j^o \quad (29)$$

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 (25) donde los términos que se encuentran dentro de la sumatoria pertenecen a la capa inmediatamente superior.

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 (0.05 a 0.25), 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 siguiente figura.



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.

Otra forma de incrementar la velocidad de convergencia consiste en utilizar una técnica llamada **momento**. Este término adicional tiende a mantener los cambios de peso en la misma dirección. Las ecuaciones de cambio de pesos pasan entonces a ser:

$$W_{kj}(t+1) = W_{kj}(t) + \alpha \delta_k^s a_j^o + \beta (W_{kj}(t) - W_{kj}(t-1)) \quad (30) \quad \text{para la capa de salida.}$$

$$W_{ji}(t+1) = W_{ji}(t) + \alpha \delta_j^o p_i + \beta (W_{ji}(t) - W_{ji}(t-1)) \quad (31) \quad \text{para la capa oculta.}$$

Siendo β el parámetro de momento, el cual toma valores positivos menores a 1 y determina el efecto en $t+1$ del cambio de los pesos en el instante t . Con este momento se consigue la convergencia de la red en menor número de iteraciones, ya que si en t el incremento de un peso era positivo y en $t+1$ también, entonces el descenso por la superficie de error en $t+1$ es mayor. Sin embargo, en t el incremento era positivo y en $t+1$ es negativo, el paso que se da en $t+1$ es más pequeño, lo cual es adecuado, ya que eso significa que se ha pasado por un mínimo y los pasos deben ser menores para poder alcanzarlo.

Resumiendo, los pasos para el entrenamiento de la red Backpropagation son:

- 1) Se inicializa la matriz de pesos y el valor de la ganancia, por lo general se asignan valores aleatorios pequeños a cada uno de los pesos w_i y al valor b . El valor de bias b puede tomarse como el valor de un peso adicional w_0 asociado con una entrada adicional siempre en 1.
- 2) Se aplica un vector o patrón de entrada p_k en las entradas de la red, y se especifica la salida deseada t_k .
- 3) Se calcula las salidas actuales de la red con las ecuaciones (3), (4), (5) y (6).
- 4) Se calculan los términos error para todas las neuronas, empezando con las de la capa de salida y siguiendo con las de la capa oculta, mediante las ecuaciones (19) y (25).
- 5) Se actualizan los pesos de las neuronas de la capa de salida y las de la capa oculta, mediante las ecuaciones (26), (27), (28) y (29). Opcional: se puede añadir un término *momento*.
- 6) El proceso se repite a partir del punto 2) hasta que el término error expresado mediante la ec.(9) resulta aceptablemente pequeño para cada uno de los patrones aprendidos.

10. Bibliografía

- Redes Neuronales Artificiales. Fundamentos, Modelos y Aplicaciones. José Ramón Hilera Gonzalez y Victor José Martínez Hernando.
- Tutorial Redes Neuronales. Universidad Tecnológica de Pereira – Facultad de Ingeniería Eléctrica.
- Redes Neuronales. Algoritmos, Aplicaciones y Técnicas de Programación. James A. Freeman y David M. Skapura.
- Neural Networks. A Comprehensive Foundation. Simon Haykin.
- Pattern Recognition Using Neural Networks. Theory and Algorithms for Engineers and Scientists. Carl G. Looney.
- Neural Network Toolbox. For Use with MATLAB®. User's Guide Version 4. Howard Demuth and Mark Beale. (PDF).