

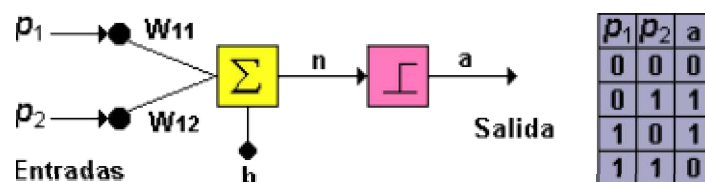
Redes Multicapa

LIMITACIÓN DEL PERCEPTRÓN

El perceptrón 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

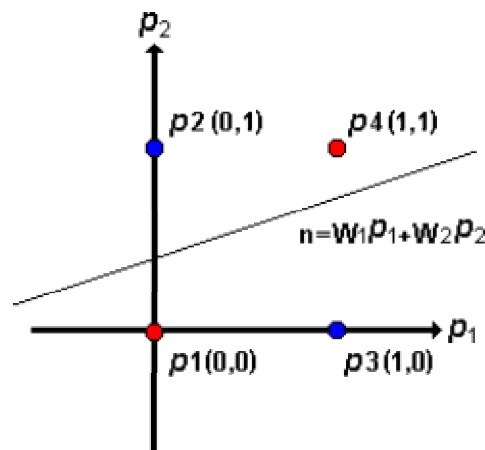
$$wp + b = 0$$

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.



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



Plano formado por el problema de la XOR

Los cuatro puntos en la figura son las posibles entradas de la red; la línea divide el plano en dos regiones, por lo que se podría clasificar los puntos de una región como pertenecientes a la clase que posee salida 1 (puntos azules) y los de la otra región como pertenecientes a la clase que posee salida 0 (puntos rojos), sin

embargo no hay ninguna forma de posicionar la línea para que los puntos correctos para cada clase se encuentren en la misma región. El problema de la compuerta XOR no es linealmente separable y una red tipo Perceptrón no está en capacidad de clasificar correctamente los patrones de esta función, debido a esta limitación del Perceptrón y a su amplia publicación en el libro de Minsky y Papert, el estudio de las redes neuronales se estancó durante casi 20 años.

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

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

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

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

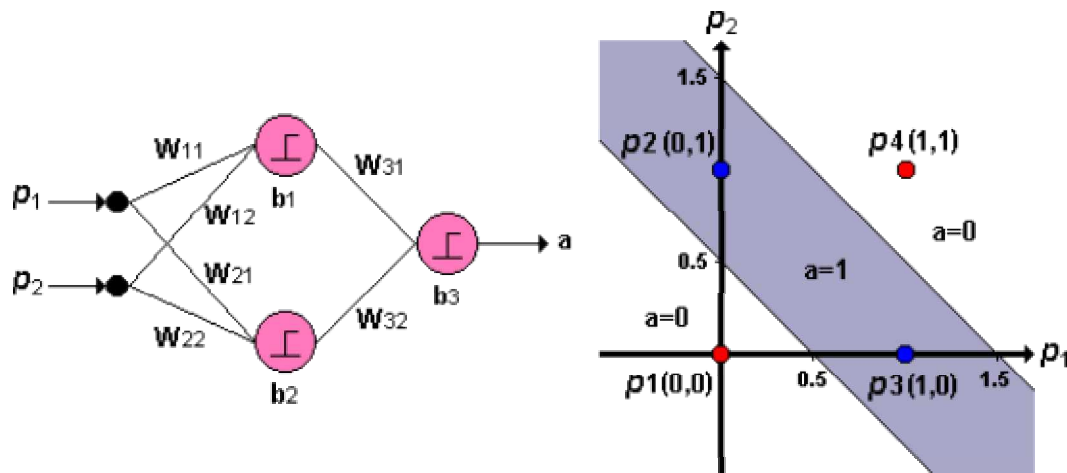
$$0 * W_{1,1} + 0 * W_{2,1} + b < 0 \quad (p_1) \quad 1 * W_{1,1} + 0 * W_{2,1} + b \geq 0 \quad (p_3)$$

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

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.

Perceptrón multicapa, con los valores de pesos y ganancias que clasifican correctamente los patrones de la compuerta XOR



Perceptrón multicapa para la XOR

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

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

REDES MULTICAPA DE PROPAGACIÓN HACIA DELANTE

Hemos visto antes, redes de una capa de PE de umbral sin interconexión. Ahora consideramos redes de propagación hacia delante (feedforward), estructuradas en capas sucesivas de neuronas artificiales (NA).

Las redes se definirán de forma precisa en términos de su arquitectura. Los elementos de cada topología son las NA y sus interconexiones. Cada NA recoge la información de n entradas con una *función de agregación* $g: \mathbb{R}^n \rightarrow \mathbb{R}$. La excitación total, calculada con esta función, se evalúa usando una función de activación $f: \mathbb{R} \rightarrow \mathbb{R}$. En los perceptrones, la función de agregación es la suma ponderada de las entradas. La activación, también llamada función de salida, compara la suma con un umbral. Más tarde generalizaremos f para que produzca todos los valores entre 0 y 1. En el caso de g también pueden considerarse otras funciones distintas de la adición, lo que permitiría calcular algunas funciones difíciles con menos PE.

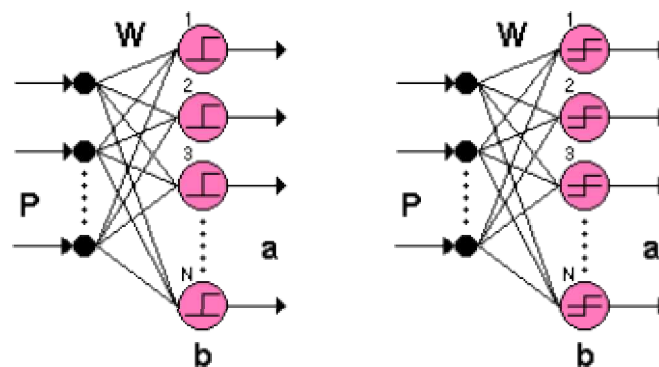
Los nodos de entrada son puntos de entrada de información a la red y no realizan ningún tipo de cálculo. Los resultados de la red se transmiten a los nodos de salida. El conjunto N comprende todos los elementos de computación de la red. Los arcos entre todas las unidades de computación tienen peso y también tienen peso los arcos entre los nodos de entrada y las unidades de computación, y entre estas y las salidas.

En la literatura de redes neuronales hay una inconsistencia en la notación que, desgraciadamente, ha llegado a convertirse en tradición. Los nodos de entrada suelen ser llamados *unidades de entrada*, aunque no se computa nada allí. Las unidades de computación de las que se lee la respuesta son llamadas *unidades de salida*.

Las arquitecturas *en capas* son aquéllas en las que el conjunto de unidades de computación N se subdivide en l subconjuntos N_1, \dots, N_l de forma que todas las conexiones van desde un elemento contenido en N_i a otro contenido en N_{i+1} , $i = 1, \dots, l - 1$. Las unidades contenidas en N_1 son las únicas conectadas a los nodos de entrada y las unidades en N_l son las de salida. Los conjuntos N_i se denominan *capas* de la red neuronal. (podemos llamar N_0 a la *capa de entrada*). Todas las capas sin entradas ni NA de salida, se llaman *capas ocultas*.

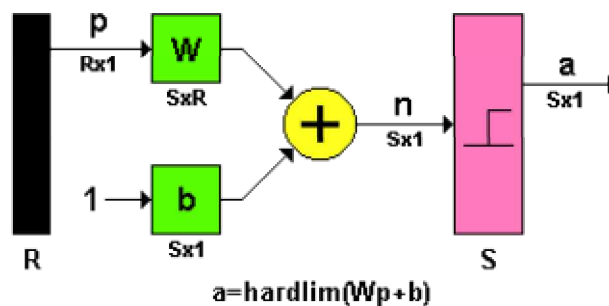
Una red neuronal con una arquitectura en capas no contiene ciclos. La entrada se procesa y se transmite de una capa a otra hasta que se calcula el resultado final. Normalmente las unidades de una capa no están conectadas entre sí (aunque hay modelos de redes neuronales en que sí lo están). En las topologías en capas, normalmente todas las unidades de una capa se conectan a todas las unidades de la capa siguiente. Las conexiones con peso 0 se entienden necesarias. Si hay m unidades en una capa y n unidades en la siguiente esto hace un total de mn arcos con peso. El número de pesos puede ser bastante grande y uno de los problemas interesantes es el de reducir el número de conexiones, o *purgar* la red.

PERCEPTRÓN MULTICAPA



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.



Notación compacta para la red tipo Perceptrón

La salida de la red está dada por:

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

Donde:


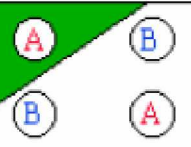
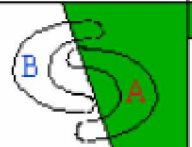





W : Matriz de pesos asignada a cada una de las entradas de la red de dimensiones $S \times R$, con S igual al número de neuronas, y R la dimensión del vector de entrada

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

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

Capacidades del Perceptrón multicapa con dos y tres capas

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.

<i>Estructura</i>	<i>Tipo de región de decisión</i>	<i>Problema del OR-Exclusivo</i>	<i>Clases lin.no separables</i>	<i>Formas más generales</i>
<i>Una capa</i> 	<i>Zonas separadas por hiperplanos</i>			
<i>Dos capas</i> 	<i>Zonas convexas</i>			
<i>Tres capas</i> 	<i>Zonas de complejidad arbitraria</i>			

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

de la intersección será una región convexa con un número de lados a lo sumo igual al número de neuronas de la segunda capa.


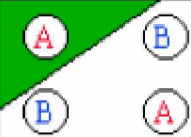


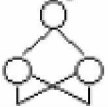
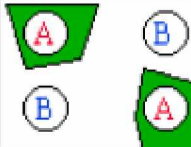
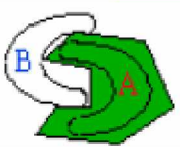
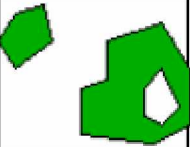
A partir de este análisis surge el interrogante respecto a los criterios de selección para las neuronas de las capas ocultas de una red multicapa, este número en general debe ser lo suficientemente grande como para que se forme una región compleja que pueda resolver el problema, sin embargo no debe ser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

La regla de aprendizaje del Perceptrón para una red multicapa es una generalización de:

$${}_1W^{nuevo} = {}_1W^{anterior} + ep^T$$

$$b^{nueva} = b^{anterior} + e$$

La importancia de la función de activación (continua).

Estructura	Tipo de región de decisión	Problema del OR-Exclusivo	Clases linno separables	Formas más generales
Una capa 	Zonas separadas por hiperplanos			
Dos capas 	Zonas de complejidad arbitraria			

Algoritmo BackPropagation

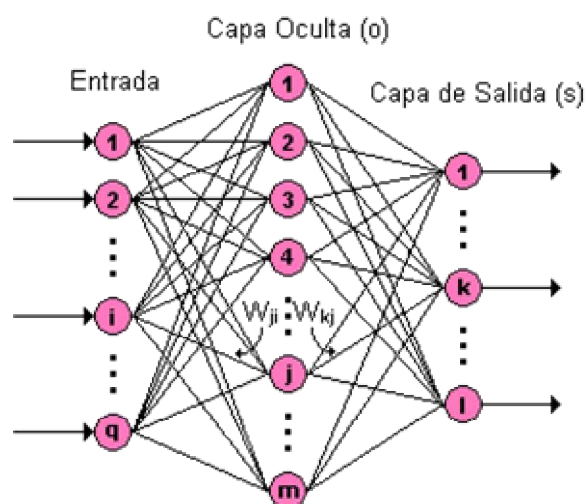
INTRODUCCIÓN

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

El primer algoritmo de entrenamiento para redes multicapa fue desarrollado por Paul Werbos en 1974, este se desarrolló en un contexto general, para cualquier tipo de redes, siendo las redes neuronales una aplicación especial, razón por la cual el algoritmo no fue aceptado dentro de la comunidad de desarrolladores de redes neuronales. Fue solo hasta mediados de los años 80 cuando el algoritmo Backpropagation o algoritmo de propagación inversa fue redescubierto al mismo tiempo por varios investigadores, David Rumelhart, Geoffrey Hinton y Ronald Williams, David Parker y Yann Le Cun. El algoritmo se popularizó cuando fue incluido en el libro "Parallel Distributed Processing Group" por los psicólogos David Rumelhart y James McClelland. La publicación de este libro trajo consigo un auge en las investigaciones con redes neuronales, siendo la Backpropagation una de las redes más ampliamente empleadas, aun en nuestros días.

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

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



explorar un gran espacio de entrada o que intentan correlacionar todas las permutaciones posibles de un conjunto de patrones muy complejo, el tiempo de computación necesario se hace bastante grande.

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

El algoritmo Backpropagation 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, se 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 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.

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.

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

Es importante recalcar que no existe una técnica para determinar el número de capas ocultas, ni el número de neuronas que debe contener cada una de ellas para un problema específico, esta elección es determinada por la experiencia del diseñador, el cual debe cumplir con las limitaciones de tipo computacional.

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

Veamos primero una forma más directa de obtener la regla delta (madaline), cuando hay varias neuronas de salida y suponemos activación lineal. El error cuadrático cometido al presentar el ejemplo p se define como indica la expresión, donde y indica el cálculo de la RNA y d el valor esperado. Para el error total se suma para todos los ejemplos de entrenamiento

$$E_p = \frac{1}{2} \sum_j (d_{pj} - y_{pj})^2$$

Si derivamos en esta expresión respecto de los parámetros de la red (pesos), aplicando la regla de la cadena. La primera parte dice como cambia el error con la salida j y la segunda, como cambia esa salida, al cambiar el peso W_{ij}

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial y_{pj}} \frac{\partial y_{pj}}{\partial w_{ji}} \quad \text{y será} \quad \frac{\partial E_p}{\partial y_{pj}} = -(d_{pj} - y_{pj}) = -\delta_{pj}$$

No sorprende pues, que la contribución de la neurona j al error sea proporcional a δ_{pj} . Además, es inmediato que

$$y_{pj} = \sum w_{ji} x_{pi} \Rightarrow \frac{\partial y_{pj}}{\partial w_{ji}} = x_{pi} \text{ y por tanto será } -\frac{\partial E_p}{\partial w_{ji}} = \delta_{pj} x_{pi}$$

En definitiva, el ajuste de pesos se hace con la siguiente expresión, conocida como regla delta

$$\Delta_p w_{ij} = \eta (d_{pj} - y_{pj}) x_{pi} = \eta \delta_{pj} x_{pi} \quad ; \quad \delta_{pj} = (d_{pj} - y_{pj})$$

Entonces, la variación del error global con los pesos, después de cada iteración completa (procesamiento de todo el conjunto de entrenamiento), es proporcional a la derivada, ya que esta es la suma de derivadas sobre todos los ejemplos de entrenamiento. Podemos afirmar que la regla delta se realiza de acuerdo con el gradiente descendente; realmente esto es cierto si los pesos no se modifican durante la iteración. No obstante, se puede aceptar también la modificación de pesos después de procesar cada ejemplo, si las modificaciones son muy pequeñas (para ello se utiliza el factor de aprendizaje η). En este caso, en que no hay neuronas ocultas, está garantizado que se puede encontrar el mínimo del error (el mejor conjunto de pesos) ya que este es único (si la función a aprender es representable).

Cuando hay capas ocultas esto no es así, y además, obtener el mínimo para valores discretos se hace muy complejo. Sin embargo en el caso continuo, el cálculo de derivadas nos da métodos adecuados para ello. Por ello vamos a considerar que la función de activación es una función no decreciente y diferenciable. En este caso, z va a representar la excitación de la neurona y tendremos (suponemos que f es diferenciable y no decreciente):

$$z_{pj} = \sum w_{ji} x_{pi} \Rightarrow y_{pj} = f_j(z_{pj})$$

Suponemos la misma función de error que antes, y aplicando de nuevo la regla de la cadena:

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial z_{pj}} \frac{\partial z_{pj}}{\partial w_{ji}} \quad \text{Tenemos que} \quad \frac{\partial z_{pj}}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \sum w_{ji} x_{pi} = x_{pi}$$

Obtenemos entonces las siguientes expresiones, y ahora solo tendríamos que obtener las δ_{pj}

$$-\frac{\partial E_p}{\partial w_{ji}} = \delta_{pj} x_{pi} \quad \text{si definimos} \quad \delta_{pj} = -\frac{\partial E_p}{\partial z_{pj}} \quad \Delta_p w_{ji} = \eta \delta_{pj} x_{pi}$$

Las δ_{pj} se pueden obtener de forma recursiva, a partir de los valores obtenidos en la capa de salida, para todas las capas ocultas, retropropagando el error en la salida por la red hacia las capas interiores. Para su cálculo, aplicamos de nuevo la regla de la cadena, poniéndolas como producto de dos derivadas parciales, la primera del error respecto de la salida en la neurona y la segunda de la salida de la neurona respecto de la excitación de la misma. Este segundo factor es claramente la derivada de la función de activación usada en la unidad correspondiente.

$$\delta_{pj} = -\frac{\partial E_p}{\partial z_{pj}} = -\frac{\partial E_p}{\partial y_{pj}} \frac{\partial y_{pj}}{\partial z_{pj}} = -\frac{\partial E_p}{\partial y_{pj}} f'_j(z_{pj})$$

Para calcular el primer factor, hay que distinguir que la neurona sea de la capa de salida o de una capa anterior.

Si es de la capa de salida, es inmediato su cálculo (recordar el madaline). Así la expresión para los δ_{pj} es inmediata:

$$\frac{\partial E_p}{\partial y_{pj}} = -(d_{pj} - y_{pj}) \quad \Rightarrow \Rightarrow \quad \delta_{pj} = (d_{pj} - y_{pj}) f'_j(z_{pj})$$

Para las neuronas de las capas ocultas, como no tenemos una expresión para el error, el cálculo no es tan inmediato. Aplicaremos de nuevo la regla de la cadena en la siguiente forma

$$\sum_p \frac{\partial E_p}{\partial z_{pk}} \frac{\partial z_{pk}}{\partial y_{pj}} = \sum_p \frac{\partial E_p}{\partial z_{pk}} \frac{\partial}{\partial y_{pj}} \sum_k w_{ji} x_{pi} = \sum_p \frac{\partial E_p}{\partial z_{pk}} w_{kj} = -\sum_k \delta_{pk} w_{kj}$$

por tanto, tenemos definitivamente para las neuronas en las capas ocultas.

$$\delta_{pj} = f'_j(z_{pj}) \sum_k \delta_{pk} w_{kj}$$

En definitiva, el proceso se resume en tres ecuaciones, que indican como se hace la adaptación de los pesos y los valores de las δ_{pj} , distinguiendo que la neurona sea de la capa de salida o no:

$$\Delta_p w_{ji} = \eta \delta_{pj} x_{pi} \quad \text{con} \quad \delta_{pj} = (d_{pj} - y_{pj}) f'_j(z_{pj}) \quad \text{o} \quad \delta_{pj} = f'_j(z_{pj}) \sum_k \delta_{pk} w_{kj}$$

Si la función de activación es la sigmoide, tenemos las siguientes expresiones, que son las utilizadas en el algoritmo Backpropagation. La función sigmoide y su derivada son:

$$y_{pj} = \frac{1}{1 + e^{-z_{pj}}} \quad \Rightarrow \quad f'_j(z_{pj}) = y_{pj}(1 - y_{pj})$$

Entonces, los valores para las δ_{pj} son en este caso:

$$\delta_{pj} = (d_{pj} - y_{pj}) y_{pj} (1 - y_{pj}) \quad \text{si } j \text{ es de salida} \quad y$$

$$\delta_{pj} = y_{pj} (1 - y_{pj}) \sum_k \delta_{pk} w_{kj} \quad \text{si no lo es}$$

En la expresión para las neuronas ocultas las δ_{pk} que hay en el sumatorio, se refieren a las calculadas en la iteración anterior, y corresponden a la capa siguiente (el cálculo se realiza de atrás hacia adelante, es decir empezando por la capa de salida).

TÉRMINO DE INERCIA. MOMENTUM

Se suele usar también un término momentum (inercia) en el ajuste de pesos, que acelera la convergencia del algoritmo. Lo que se hace es tener en cuenta el ajuste realizado en la iteración n , para hacer el ajuste en la iteración $n+1$, con un coeficiente del momentum. Pretende obviar salto bruscos en las direcciones de optimización que marca el gradiente descendente. El ajuste de los pesos se hace con

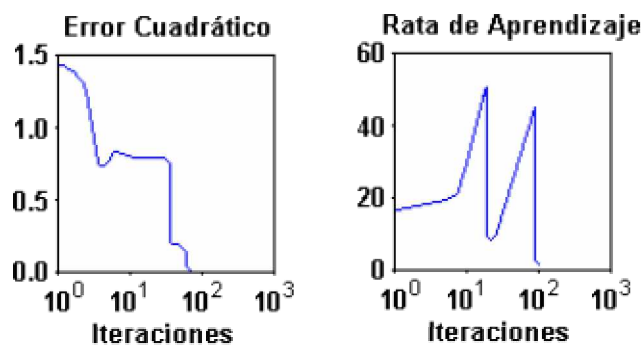
$$\Delta_p w_{ji}(n+1) = \eta \delta_{pj} x_{pi} + \alpha \Delta_p w_{ji}(n)$$

TASA DE APRENDIZAJE

Para una red multicapa la superficie del error no es una función cuadrática, su forma es diferente para diferentes regiones del espacio, la velocidad de convergencia puede incrementarse por la variación de la tasa de aprendizaje en cada parte de la superficie del error, sin sobrepasar el valor máximo para aprendizaje estable definido anteriormente.

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

- § Si el error cuadrático de todos los parámetros del set de entrenamiento se incrementa en un porcentaje ζ típicamente entre 1% y 5%, después de la actualización de los pesos, esa actualización es descartada, la tasa de aprendizaje se multiplica por un factor $0 < \rho < 1$, y el coeficiente de momentum es fijado en cero.
- § Si el error cuadrático se decrementa después de la actualización de los pesos, esa actualización es aceptada y la tasa de aprendizaje es multiplicada por un factor $\eta > 1$. Si el momentum había sido previamente puesto en cero, se retorna a su valor original.
- § Si el error cuadrático se incrementa en un valor menor a ζ , los pesos actualizados son aceptados, pero la tasa de aprendizaje y el coeficiente de momentum no son cambiados.



Característica de convergencia para una tasa de aprendizaje variable

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

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

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



Superficie típica de error

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

OTRAS ALTERNATIVAS PARA ACELERAR EL APRENDIZAJE.

Han surgido muchas variantes del algoritmo de Backpropagation, en general intentando disminuir el tiempo de computación, o para resolver alguno de los problemas del algoritmo. En las lecturas sobre el tema se incluyen diversas variantes del algoritmo de entrenamiento (en modo uso, se hace siempre igual y la única diferencia puede ser el uso de diferentes funciones de activación) de las que solo se indica su denominación: Algoritmo alfa-LMS ; Algoritmo de Mays ; Regla II y III del Madaline ; Algoritmo μ .LMS ;

Cálculo del gradiente hacia delante y hacia atrás ; Retropropagación en el tiempo ; Algoritmo FullPropagation ; Regla Delta-Bar-Delta ; Algoritmo QuickPropagation ; Método de direcciones conjugadas ; Método de métrica variable (quasi-Newton) ; Método de direcciones aleatorias. Método de Levenberg-Marquardt.

Para mejorar el aprendizaje y la rapidez, se han considerado procedimientos de eliminar neuronas cuyos pesos no se modifican o lo hacen muy poco (la neurona no parece tener relevancia en el problema). También se han considerado procesos constructivos de entrenamiento (se empieza con muy pocas neuronas y se van añadiendo conforme el aprendizaje avanza y mejora).

CUESTIONES A TENER EN CUENTA:

Número de capas y de neuronas. No hay procedimientos claros para decidir el mejor número de capas y/o neuronas en la RNA. Hay publicaciones sobre cotas mínimas y máximas, generalmente para problemas de clasificación, en función del número de ejemplos de entrenamiento, pero en la práctica casi nunca funcionan. Generalmente se determina de forma intuitiva y/o experimental, pero sin duda la experiencia del usuario sirve de gran ayuda.

¿Cómo elegir el número de capas?

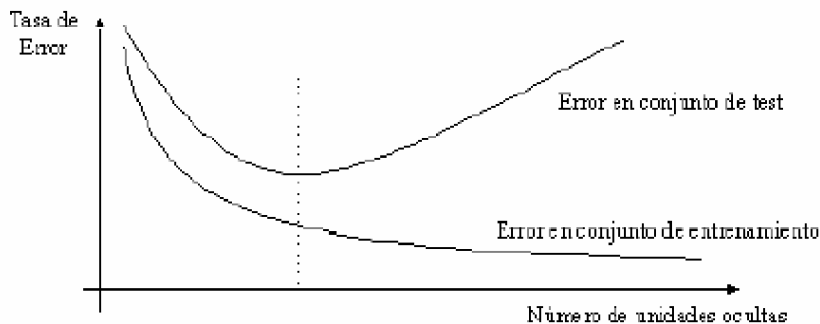


Conjunto de entrenamiento. Podemos decir casi lo mismo que en el apartado anterior. Lo que está claro, es que si se pretende un buen aprendizaje del dominio del problema, o una buena generalización de los ejemplos, estos deben cubrir todo el espectro del dominio. El número de ejemplos y los ejemplos concretos dependen del problema y la experiencia del usuario ayuda. Se suele determinar casi siempre (a veces solo se dispone de ejemplos concretos) de forma experimental.

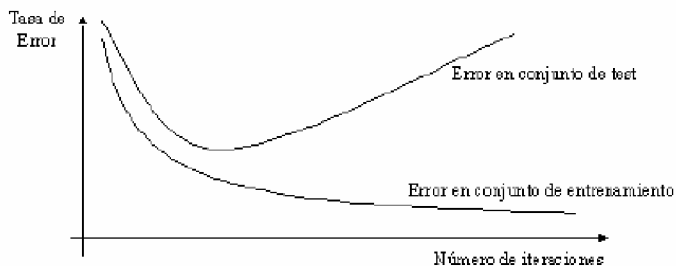
Preparación de los Datos. En general, los datos no se utilizan directamente en la forma en que se obtienen o los proporciona el problema. Casi siempre hay que hacer un pre-procesamiento, para adaptar los datos reales al modelo usado. Por ejemplo, las salidas hay que normalizarlas en $[0,1]$ si se usa activación sigmoide, porque sino, el error nunca podrá reducirse de forma satisfactoria. También es habitual normalizar las entradas, aún cuando no sea necesario. Obviamente, habrá un post-procesamiento de resultados, para hacerlos válidos en la realidad del problema.

FACTORES QUE INFLUYEN EN EL RENDIMIENTO DE LA RED

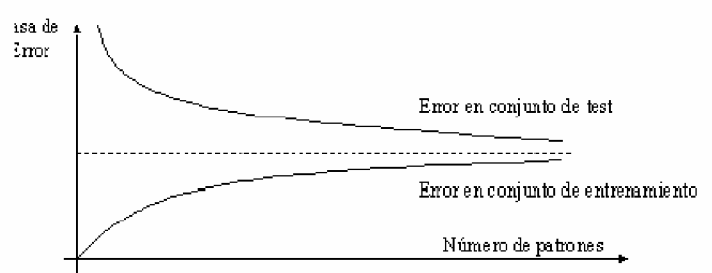
- *Efecto del número de unidades ocultas*



- *Efecto del número de iteraciones del alg. de entrenamiento - sobreentrenamiento*



- *Efecto del número de muestras de aprendizaje*



APLICACIONES

El algoritmo de retropropagación de errores se ha aplicado a una cantidad ingente de problemas de laboratorio (para verificar las prestaciones) y de la vida real. Su fama se debe sin duda al éxito de gran parte de estas aplicaciones.

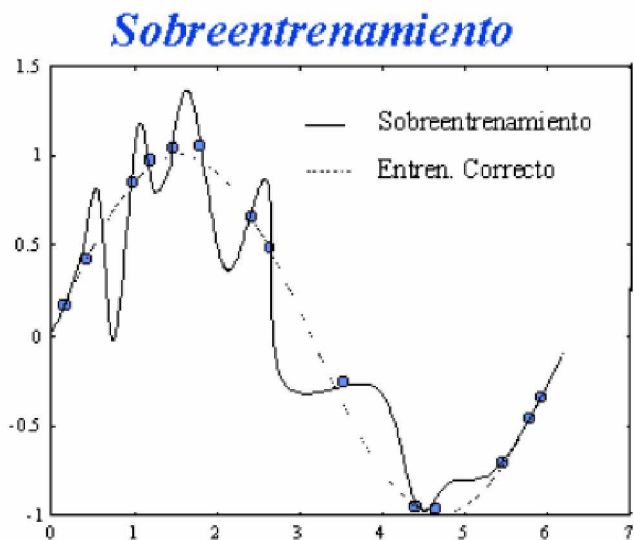
Codificación de patrones para clasificación	Tratamiento de señales digitales
Estructura secundaria de proteínas.	Predicción y previsión (de señales, series temporales, ...)
Corte adecuado de palabras con guiones.	Reconocimiento de códigos postales escritos a mano
Reconocimiento de tarjetas de sonar	Reconocimiento de sonido
Conducción de un coche	Comparación y ordenación de números difusos
Compresión de datos o imágenes.	Segmentación de clientes
Juego del Backgammon	Obtención de reglas difusas para describir sistemas difusos
Texto hablado (NETtalk: recibe texto escrito y lo transforma en sonido con un sintetizador de voz)	

PROBLEMAS Y DEFICIENCIAS:

- § Velocidad de convergencia. El algoritmo Backpropagation es lento y se han propuesto muchas modificaciones y variantes para mejorar la velocidad del entrenamiento. Algunas se reflejan en el apartado de variantes, y otras se incluyen en las lecturas sobre el tema.

§ Mínimos Locales. El método del gradiente descendente a veces queda "atrapado" en un mínimo local, del que no puede salir, y por tanto el aprendizaje no se hace bien. Una forma de obviar esto es realizar el aprendizaje varias veces, partiendo de pesos diferentes cada vez, y seleccionar la ejecución que mejor resuelve el problema. Algunas de las variantes buscan usar otros procesos de optimización no lineal más seguros que el gradiente descendente.

§ Sobreentrenamiento. Es el problema de que se aprende muy bien los ejemplos de entrenamiento, pero cuando se usa la RNA entrenada, con muestras que no ha aprendido, no es capaz de dar buenas respuestas. Se puede evitar seleccionando muy bien el conjunto de entrenamiento y parando el entrenamiento cuando el error es suficientemente pequeño pero no excesivamente pequeño. Se dice entonces, que la RNA no generaliza bien.



§ Saturación. Se puede producir cuando las salidas esperadas en cada neurona de salida son 0 o 1. Al pretender acercar las salidas de la Red a estos valores, nos ponemos en zonas donde la función de activación tiene tangente de pendiente casi cero y entonces no se produce apenas modificación de los pesos y por tanto del error. La forma de resolverlo es cambiar los valores de salidas esperadas a 0.1 y 0.9. Esto puede requerir adaptación de los datos reales. También se produce si los pesos se hacen muy grandes, por que los resultados de la Red ya no varían.

§ Estabilidad o robustez. Un tema interesante y actual, sobre todo para implementaciones hardware de la RNA entrenada, es la estabilidad del aprendizaje sobre variaciones en los pesos (y/o en las entradas). Hay medidas de estabilidad estadística, que se pueden usar para seleccionar la RNA entrenada más estable de entre varias ejecuciones. También hay métodos que tienen en cuenta estas medidas en el entrenamiento de la Red, aunque son más lentos.

