Computación neuronal. Universidade de Santiago de Compostela, 1995 (425-454)

XIV

MODELOS INCREMENTALES PARA REDES NEURONALES ARTIFICIALES

F. Castillo; J. M. Moreno; J. Cabestany
Universidad Politécnica de Cataluña

INTRODUCCIÓN

En otros capítulos de este mismo libro se pone de manifiesto como las redes neuronales existen como una alternativa en el procesado de la información. Si bien éstas tienen sus ventajas sobre métodos más convencionales, existen ciertas dificultades a la hora de escoger la red apropiada.

Uno de los actuales problemas en la utilización de las redes neuronales es escoger que tipo de red, asi como que estructura utilizar para un cierto problema. Existen ciertos resultados teóricos [Cybenko, 1989; Lapedes, 1988] que demuestran las limitaciones y la versatilidad de las redes neuronales artificiales (RNA) pero estos resultados sólo muestran límites superiores e inferiores en el número de neuronas necesarias para casos muy específicos y difíciles de interpretar para casos prácticos, lo que hace que en la práctica se tenga que adivinar la estructura adecuada mediante prueba y error.

Una alternativa es utilizar algoritmos que añaden o bien que eliminan neuronas a medida que se desarrolla el aprendizaje, como método para resolver el problema. Estas son las llamadas redes neuronales incrementales y decrementales, respectivamente, y en conjunto se las denomina «evolutivas», ya que su estructura no es fija. En este capítulo se centrará la discusión sobre las redes evolutivas de tipo incremental únicamente.

Las RNA incrementales pueden tener ciertas ventajas adicionales sobre las no evolutivas. Primero, al añadir demasiadas neuronas a una red se puede dar lugar al

sobre entrenamiento (*overfitting*) y por tanto, una pobre generalización, cosa que se puede evitar fácilmente en las redes evolutivas utilizando un criterio adecuado en la selección de nuevas neuronas. Segundo, el tiempo necesario para que la red converja suele ser mucho menor en sistemas incrementales, ya que la agregación de neuronas al sistema normalmente implica una reducción en la complejidad del problema a resolver. Tercero, la adición de nuevos datos al conjunto de entrenamiento implica en muchos casos tener que volver a entrenar la red neuronal. Esto se puede evitar en algunas redes evolutivas añadiendo neuronas adicionales encargadas de procesar estos nuevos datos.

Una de las aplicaciones más usuales de las RNA incrementales es en sistemas de clasificación, cuyas bases teóricas residen en la teoría clásica de la decisión. Estos sistemas tienen muchas aplicaciones incluyendo el reconocimiento de imágenes (escritura, objetos, etc.), reconocimiento de sonido y voz, codificación, sistemas expertos, y memorias asociativas. En las siguientes secciones se estudiarán primero las bases teóricas utilizadas y se tratará de identificar los diferentes tipos de RNA incrementales dando ejemplos concretos para cada tipo de red. En las secciones sucesivas se detallarán las diferentes variedades de éstas.

2. BASES TEÓRICAS

Un sistema clasificador cualquiera, sea basado en una red neuronal o en algún otro método, forma parte de un proceso más general que es el de reconocimiento de patrones, tal y como se muestra en la figura 1. Se trata de reconocer un objeto del cual se obtienen varias características mediante transductores especializados, que se encargan de traducir la cantidad existente de una característica particular en una señal eléctrica. El pre-procesado puede ser un filtro o una transformación aplicada sobre las características extraídas, lo que convierte la señal obtenida a un formato más apropiado para el clasificador. Finalmente, el clasificador se encarga de indicar a qué clase el patrón (que es el conjunto de características medidas) pertenece.

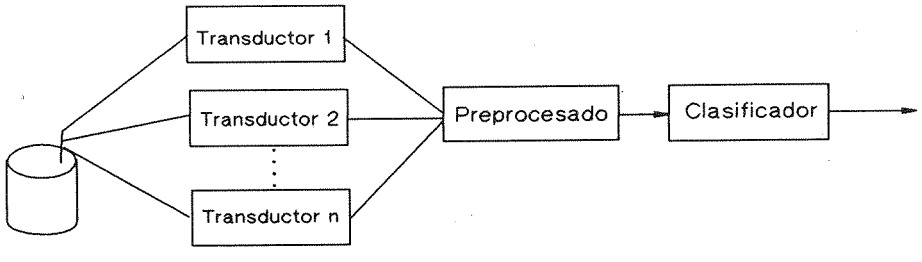


Figura 1. Proceso general de reconocimiento de patrones.

El clasificador óptimo se puede construir partiendo de la teoría estadística Bayesiana. Supongamos que se posee un conjunto de patrones (el conjunto de entrenamiento) para los cuales se conoce su pertenencia a las diferentes clases. Representemos con X^i el i-ésimo patrón. Supongamos también que se conoce la densidad de probabilidad, o sea, la probabilidad de que una observación de la clase w_i tome un valor cualquiera X, $p(X \mid w_i)$. Entonces se puede calcular la probabilidad a posteriori $P(w_i \mid X)$, que indica la probabilidad de que la clase w_i ocurra dado un patrón X. La regla de Bayes indica que se ha de calcular como:

$$P(w_i|X) = \frac{p(X|w_i)P(w_i)}{\sum_{j=1}^{n} p(X|w_j)P(w_j)} \qquad (1)$$

donde n es el número de clases y $P(w_i)$ es la probabilidad a priori para la clase w_i, o sea, la probabilidad de que escogiendo un patrón al azar, éste pertenezca a la clase w_i.

Es entonces lógico escoger siempre la clase con la mayor probabilidad a posteriori. Para un sistema con dos clases:

$$\begin{aligned} \text{escoger} \quad & w_1 \text{ si } P(w_1 \mid X) > P(w_2 \mid X) \\ & w_2 \text{ si } P(w_2 \mid X) > P(w_1 \mid X) \end{aligned} \qquad (2)$$

Éste, de hecho, es el criterio que nos da el **mínimo** error, cualquier otro criterio sólo dará un error mayor. Tomando como ejemplo las densidades de probabilidad de la figura 2 y calculando las probabilidades a posteriori con $P(w_1)=2/3$ y $p(w_2)=1/3$, se obtiene la representación de la figura 3. De aquí se puede apreciar que el clasificador de Bayes crea una frontera de decisión justo en el punto $P(w_1 \mid X)=P(w_2 \mid X)$. Se dice que la expresión (2) es el discriminante, ya que en función de esta desigualdad el clasificador determina si el patrón es de clase 1 ó 2. Cabe decir que el discriminante no es único, ya que existen múltiples formas de definir el punto $P(w_1 \mid X)=P(w_2 \mid X)$, considérese por ejemplo el discriminante:

$$\text{asignar X a } w_i \text{ si } p(X \mid w_i)P(w_i) > p(X \mid w_j)P(w_j), \quad \forall j \neq i \qquad (3)$$

Por tanto, una forma de aproximar el clasificador de Bayes es encontrar un discriminante que separe las dos clases, minimizando el número de patrones mal clasificados.

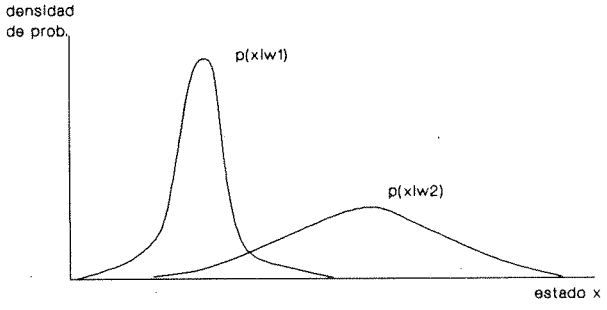


Figura 2. Ejemplo de densidad de probabilidad para 2 clases.

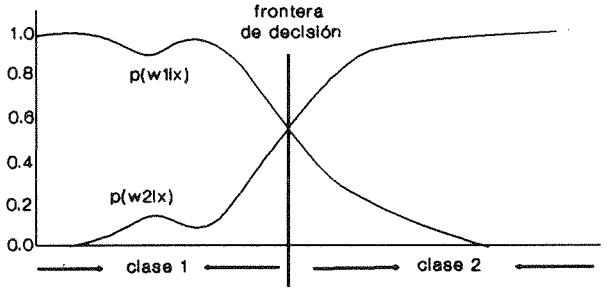


Figura 3. Cálculo de las probabilidades a posteriori a partir de la figura 2.

Los diferentes tipos de clasificadores neuronales fueron identificados por primera vez en [Castillo, 1992], basándose en las técnicas que éstas emplean para asociar un patrón a una determinada clase. Si la construcción de la red neuronal se basa en la determinación de un(os) discriminante(s) se le llama clasificador de separación lineal a tramos[1] (*Piecewise linear separation*, PLS). En este tipo de redes, representado en la figura 4, se trata de obtener un discriminante mediante la utilización de secciones de varias funciones lineales hasta conseguir separar los patrones de clases diferentes.

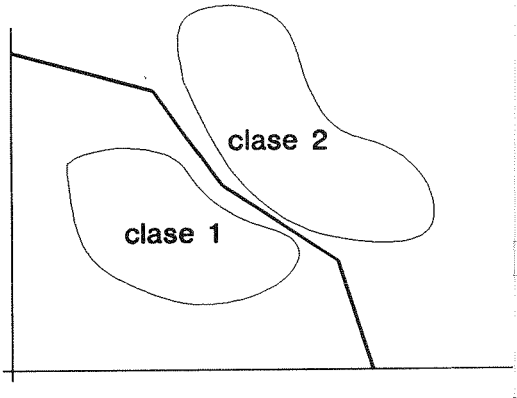


Figura 4. Clasificador que estima la frontera de decisión.

[1] Esto es debido a que normalmente son funciones de primer orden.

Un segundo método consiste en la creación de regiones en el espacio de entrada que vienen asociadas a las diferentes clases existentes, evitando la definición de un discriminante. Cualquier patrón que caiga en una región asociada a la clase w_i será asociada a dicha clase. La idea detrás de este segundo método es cubrir todo el espacio en la cual una clase es predominante con zonas etiquetadas, asociando las zonas a las diferentes clases. A este segundo tipo de red neuronal se le llama de región de influencia (*Region of Influence*, ROI, figura 5).

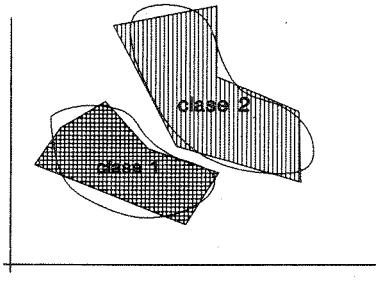


Figura 5. Clasificador que estima regiones de decisión.

En ambos métodos se evita calcular la densidad de probabilidad, que normalmente es desconocida, y se procede a la estimación o uso de la probabilidad a posteriori a través de las muestras. Esto tiene sus ventajas y desventajas tal y como se mencionan en [Castillo, 1993b] y se explica brevemente en la sección 5.1. Si por el contrario se estima la densidad de probabilidad, ésto da lugar a otra familia de redes basadas en los métodos probabilísticos (*Probabilistic Neural Networks*, PNN).

En las siguientes secciones se pasará revista a estos diferentes tipos de redes y al final se harán algunas conclusiones al respecto.

3. ALGORITMOS INCREMENTALES DE SEPARACIÓN LINEAL A TRAMOS

Como se ha indicado anteriormente, este tipo de algoritmos incrementales se dedican fundamentalmente a la resolución de tareas de clasificación de patrones, o, lo que es lo mismo, a la discriminación de vectores de características agrupados bajo distintas categorías o clases. En los siguientes apartados nos ocuparemos de exponer en primer lugar los modelos según los cuales estos algoritmos son capaces de generar la estructura de red más adecuada para resolver el problema planteado. A continuación se explicarán los métodos disponibles para llevar a cabo el entrenamiento de las unidades que componen la estructura generada. Finalmente, se presentará una taxonomía comparativa para este tipo de algoritmos, agrupándolos por características afines.

3.1. Principio de organización

El objetivo final de un sistema clasificador consiste en determinar la(s) función(es) discriminante(s) que permite(n) resolver el problema planteado. En el

caso de los algoritmos incrementales de separación lineal a tramos (*Piecewise Linear Separation* - PLS), las funciones discriminantes requeridas para resolver el problema de clasificación se obtienen mediante una aproximación lineal. Dicha aproximación consiste en la combinación de las funciones discriminantes asociadas a cada una de las unidades generadas por el algoritmo incremental. Dado que estas unidades son de tipo perceptron, cada una de ellas dará lugar, para un espacio de entrada compuesto por vectores de dimensión n, a un discriminante lineal que será un hiperplano de dimensión n-1 (por ejemplo, para un espacio de entrada de dos dimensiones, el discriminante lineal asociado a estas unidades será una línea recta).

Los algoritmos PLS suelen comenzar pues a partir de una estructura de red muy sencilla, compuesta normalmente por una unidad de tipo perceptron. Esta unidad es entrenada con todo el conjunto de vectores de entrada para resolver el problema de clasificación planteado. En el caso de que el problema de clasificación tenga como solución un discriminante lineal (es decir, las clases o categorías definidas en el espacio de entrada son linealmente separables), esta unidad será capaz de encontrar la solución al problema, y por tanto el algoritmo PLS no generará nuevas unidades. En caso contrario, la solución proporcionada por esta unidad será considerada tan sólo como una primera aproximación a la solución deseada. A continuación, el algoritmo continuará generando y entrenando nuevas unidades hasta que la aproximación obtenida mediante la combinación de las soluciones lineales de éstas sea capaz de resolver el problema planteado.

Estudiaremos el principio general de organización de estos algoritmos a partir de un ejemplo, que consistirá en la resolución del problema planteado en la figura 6. El problema consiste en determinar, para un vector de entrada de dos componentes, x_1 y x_2, si este vector pertenece a la clase 0, constituida por los patrones cuya posición está marcada con «o», o bien si pertenece a la clase 1, constituida por los patrones cuya situación viene marcada con «x».

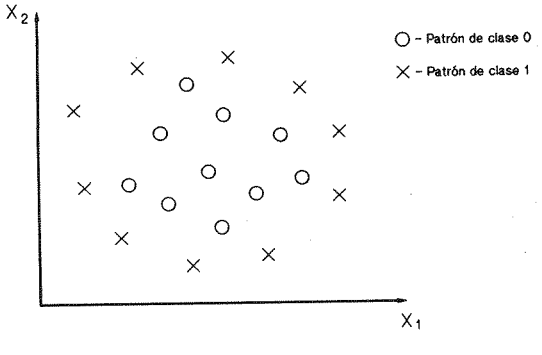


Figura 6. Problema de clasificación planteado.

El algoritmo que se usará para tratar de resolver este problema es el algoritmo *Neural Trees*, [Sirat y col., 1990]. Este algoritmo parte de una estructura de red muy simple, constituida por una sóla unidad, a la que denominaremos unidad 0. Esta unidad se entrena, mediante la regla delta, [Rosenblatt, 1962], o mediante el *algoritmo Pocket*, [Gallant, 1986], para resolver todo el problema de clasificación. Como en este caso el problema no admite separación lineal, será preciso que el algoritmo incremental genere nuevas unidades con el fin de refinar la solución al problema propuesto. En el caso del *algoritmo Neural Trees*, a partir de cada unidad se generan dos nuevas unidades. A cada una de estas dos nuevas unidades se les asignan nuevos conjuntos de entrenamiento, que serán versiones reducidas del problema original. En concreto, la nueva unidad 1 recibirá como conjunto de entrenamiento el formado por todos los patrones del problema original que la unidad 0 clasificaba como pertenecientes a la clase 0 (es decir, todos aquellos para los cuales la unidad 0 produce una salida 0). De la misma forma, la nueva unidad 2 recibirá como problema a resolver el constituido por todos los patrones del conjunto original que la unidad 0 clasificaba como pertenecientes a la clase 1. La figura 7 muestra la solución obtenida por la unidad 0 para el problema original planteado y cómo se obtienen los conjuntos de entrenamiento para las nuevas unidades 1 y 2 generadas a partir de ella.

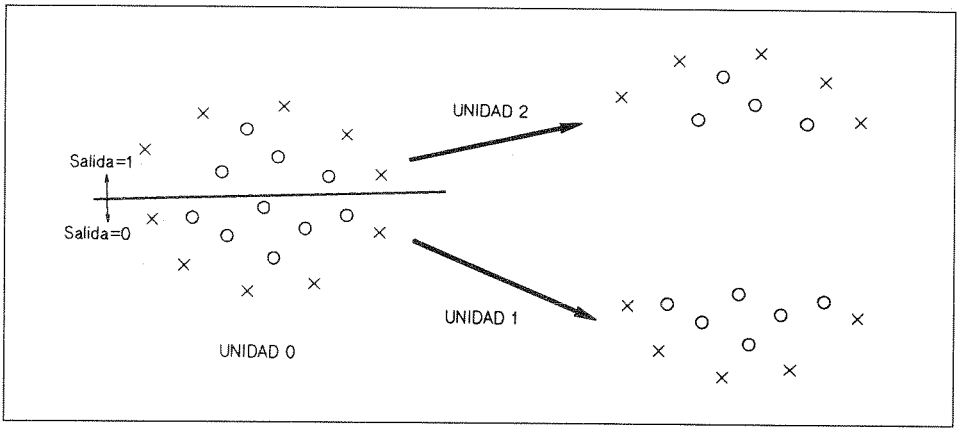


Figura 7. División del conjunto de entrenamiento para las unidades 1 y 2.

Las unidades 1 y 2 estarán, pues, como la unidad 0, conectadas a las unidades de entrada. Además, la unidad 1 se conecta a la unidad 0 mediante un enlace de tipo 0, mientras que la unidad 2 se conecta a la unidad 0 mediante un enlace de tipo 1. Como los problemas obtenidos para estas unidades tampoco admiten una solución lineal, se generarán dos nuevas unidades a partir de cada una de ellas, tal y como sucedió para la unidad 0.

Este proceso de subdivisión del problema de entrada original mediante la generación de nuevas unidades continúa hasta que todos los nuevos subproblemas generados admiten una solución lineal. Por tanto, en este momento la aproximación lineal obtenida por combinación de los discriminantes lineales asociados a cada una de las unidades generadas es lo suficientemente precisa como para resolver el problema planteado inicialmente. La figura 8 muestra la organización final de la red generada para el problema propuesto, así como los discriminantes lineales obtenidos para cada una de las unidades.

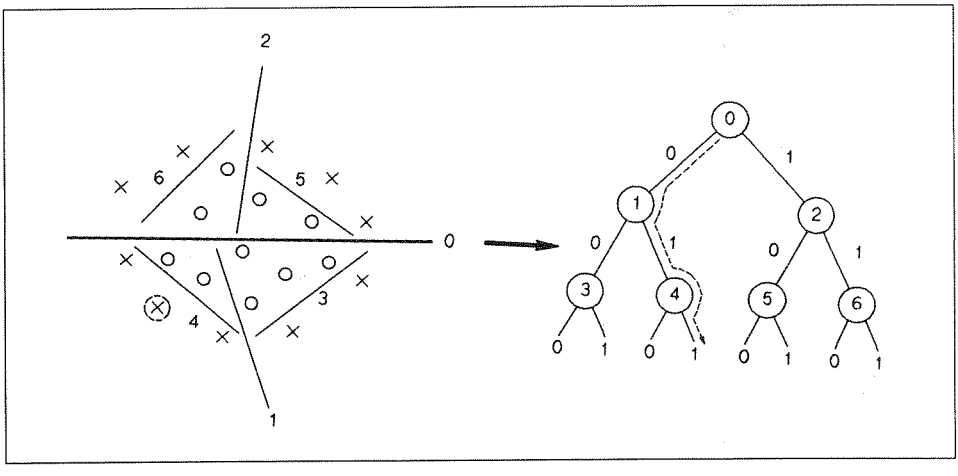


Figura 8. Estructura generada para el problema propuesto.

Como puede apreciarse, este algoritmo da lugar a una estructura similar a la que presentan los árboles de decisión. De hecho, una vez que la red está generada y entrenada, el sistema utilizado para determinar a qué clase pertenece un determinado vector de entrada consiste en una progresión a través del árbol de decisión. Así, una vez aplicado el patrón de entrada, y partiendo de la unidad 0, se comprobará la salida de la unidades que dependen de ella en función de la salida que produzca. Es decir, si la salida de la unidad 0 es «0», se comprobará la salida de la unidad conectada a ella a través del enlace de tipo 0, y si la salida es «1» se comprobará la salida de la unidad conectada a ella a través del enlace de tipo 1. Esta comprobación continuará en sentido descendente a lo largo del árbol hasta que se alcance una de las unidades terminales, cuya salida indicará directamente la clase a la que se asocia el vector de entrada. Debe tenerse en cuenta que todas las unidades generadas están conectadas a las unidades de entrada, y que los enlaces entre unidades son meros enlaces simbólicos, de manera que no existe una transferencia física de activaciones de salida entre las unidades. La línea de puntos en la figura 8 muestra el camino seguido a lo largo de la red generada para determinar que el patrón encerrado dentro de un círculo punteado pertenece a la clase 1.

3.2. Entrenamiento de unidades

Como se ha mencionado anteriormente, las unidades de las que se compone la estructura generada por los algoritmos PLS son en general del tipo perceptron y cada una de ellas es sometida a un proceso de entrenamiento sobre un conjunto de patrones generalmente distinto de las restantes.

Este entrenamiento es llevado a cabo normalmente mediante la regla delta. Según esta regla, una vez presentado a la unidad sometida a entrenamiento un vector de entrada $\mathbf{x}^p$, para el cual se espera obtener una salida t^p, el cambio que se debe efectuar sobre el vector de pesos asociado a dicha unidad viene dado por la siguiente ecuación:

$$\Delta\omega_i = \eta(t^p - o^p)x_i^p \qquad (4)$$

donde:

* ω_i : Componente i del vector de pesos de la unidad,
* η: Tasa de aprendizaje. Suele tomar valores comprendidos entre 0 y 1,
* o^p : Salida proporcionada por la unidad para el vector $\mathbf{x}^p$,
* x_i^p : Componente i del vector de entrada $\mathbf{x}^p$.

Uno de los algoritmos de aprendizaje más extendidos para el entrenamiento de las unidades generadas por los algoritmos PLS es el *algoritmo Pocket*. Este algoritmo consiste básicamente en la ejecución iterada de la regla delta sobre el conjunto de vectores de entrada que definen el problema a resolver, pero manteniendo almacenado como solución el vector de pesos que da lugar al máximo número de vectores de entrada clasificados correctamente.

Se ha demostrado que, en el caso de que el problema a resolver sea linealmente separable, ambos algoritmos, perceptron y *Pocket*, son capaces de llegar a esta solución en un número finito de iteraciones. Así mismo, en el caso de que el problema planteado no sea linealmente separable, se demuestra igualmente que ambos algoritmos llegan finalmente a la solución que proporciona el máximo número de vectores de entrada clasificados correctamente.

Sin embargo, en el caso de que el problema a resolver no sea linealmente separable, la solución proporcionada por estos algoritmos de aprendizaje puede no ser útil para el proceso de construcción de red que llevan a cabo los algoritmos incrementales de tipo PLS.

Consideremos por ejemplo el problema representado en la figura 9. En este caso se trata también de resolver un problema de clasificación en un espacio de

entrada bidimensional y con dos posibles categorías o clases a las que asignar los vectores de entrada.

En esta figura, la línea de puntos representa el discriminante lineal obtenido como solución al problema por una unidad entrenada mediante la regla delta o el *algoritmo Pocket*. Como puede fácilmente apreciarse, esta solución es efectivamente la que da lugar a un mayor número de vectores de entrada clasificados correctamente.

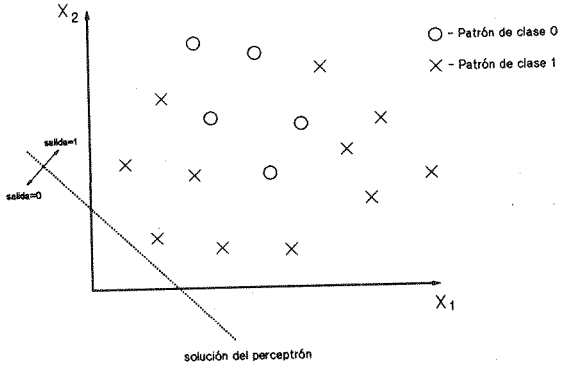


Figura 9. Solución de máxima tasa de clasificación.

No obstante, una solución como la anterior no es de utilidad para llevar a cabo un proceso de construcción de red como el que se explicó en el apartado anterior. Efectivamente, si recordamos la forma en la que el *algoritmo Neural Trees* asignaba los conjuntos de entrenamiento a las nuevas unidades generadas, podremos deducir que en este caso sólo se generará a partir de esta unidad una nueva unidad (sólo hay vectores de entrada a un lado de la función discriminante). Además, esta nueva unidad recibirá como conjunto de entrenamiento el formado por todos los vectores que constituían el problema original. Una vez entrenada, esta nueva unidad dará lugar a una solución como la anterior y transferirá de nuevo a su unidad descendiente el mismo problema a resolver, proceso que se repetirá de forma indefinida. La figura 10 muestra la estructura de red a que daría lugar este proceso de construcción.

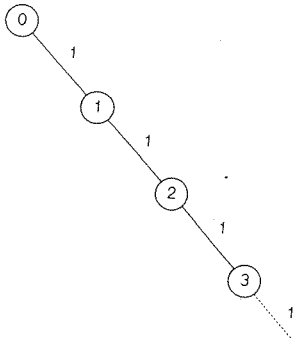


Figura 10. Estructura generada para la solución de máxima tasa de clasificación.

En consecuencia, la selección de la solución que genera el máximo número de vectores clasificados correctamente da lugar en este caso a una estructura de red redundante, compuesta por un número infinito de unidades e incapaz de resolver el problema planteado.

Con el fin de evitar este tipo de comportamiento indeseado, se han propuesto una serie de soluciones en [Moreno y col., 1993a, 1993b, 1994] encaminadas a mejorar la evolución de las redes generadas por los algoritmos incrementales PLS. El objetivo final de estas propuestas consiste en forzar que cada una de las unidades que genera el algoritmo sea capaz de realizar una división útil del problema para el cual es entrenada. En el contexto del cual nos ocupamos, el concepto de «división útil» significa que los problemas que se generan a partir de cada unidad para sus unidades descendientes deben ser una versión reducida (y por tanto, simplificada) del problema con el cual fue entrenada aquélla. De esta forma, asegurando que en cada paso de división se consigue una reducción en la complejidad del problema, se garantizará la construcción de una red de dimensión finita capaz de resolver el problema de clasificación planteado.

Básicamente, los métodos apuntados consisten en modificar la condición que utiliza el *algoritmo Pocket* para seleccionar el mejor vector de pesos para la unidad sometida a entrenamiento. De esta forma, el método propuesto en [Moreno y col., 1993a, 1993b], en lugar de seleccionar como mejor vector actual aquel que da lugar a un mayor número de patrones clasificados correctamente, impone que se cumplan simultáneamente las siguientes condiciones:

* A un lado del discriminante lineal determinado por la unidad deben haber únicamente patrones de la misma clase, y todos ellos deben estar clasificados correctamente.

* El número de vectores clasificados correctamente debe ser mayor que el que producía el vector de pesos anterior.

La figura 11 muestra una posible evolución temporal del vector de pesos de la unidad entrenada cuando el *algoritmo Pocket* se modifica mediante el criterio anterior.

Como puede deducirse de la anterior figura, en este caso la solución final aportada por la unidad ya entrenada sí que reduce o simplifica el problema que se transfiere a las unidades que se deriven de ella, dando lugar de esta forma a una estructura de red útil para resolver el problema inicial planteado.

En el caso de la modificación propuesta en [Moreno y col., 1994], cada vez que el vector de pesos debe ser comparado para determinar si se guardará como mejor

solución definitiva, se exige que este vector reduzca el valor de una cierta función de distribución definida sobre el conjunto de vectores de entrada. Esta función tiene en cuenta el grado de imbricación mutua de las distintas clases o categorías definidas en el espacio de entrada, de forma que su valor es nulo cuando se consigue determinar la función discriminante que separa totalmente dichas categorías.

De todo lo anterior se deduce por tanto que el proceso de entrenamiento de las unidades generadas por los algoritmos incrementales PLS determinará en gran medida la calidad de las soluciones aportadas por éstos. Por consiguiente, deberá ponerse especial cuidado a la hora de seleccionar el tipo de algoritmos utilizados para llevar a término dicho entrenamiento.

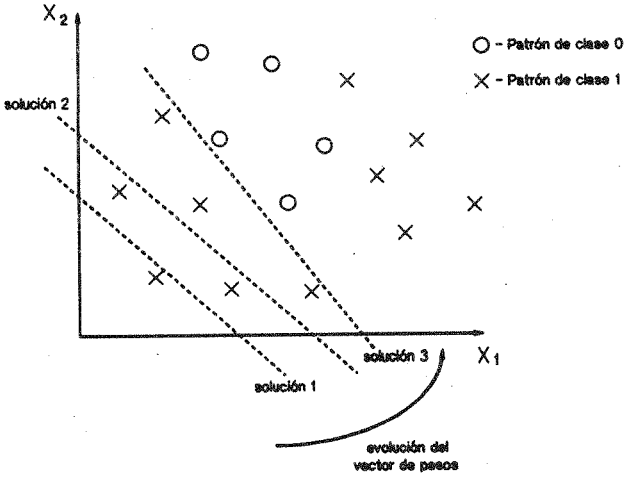


Figura 11. Evolución de las soluciones para los algoritmos modificados.

3.3. Taxonomía de los algoritmos incrementales PLS

Conocido el comportamiento y características más relevantes de los algoritmos incrementales PLS, resulta interesante de cara a seleccionar el algoritmo más adecuado para una aplicación determinada conocer qué algoritmos presentan características comunes. Es por ello que a continuación trataremos de agruparlos siguiendo dos criterios básicos: estrategia para la evolución de la red y tipo de arquitectura final generada. Finalmente, se presentarán en una tabla, a modo de resumen, las características más sobresalientes de los algoritmos presentados.

Desde el punto de vista del principio seguido para llevar a cabo la organización de la red, podemos distinguir dos tipos básicos de algoritmos incrementales PLS:

1) Aquellos que, como el *algoritmo Neural Trees* mostrado anteriormente, hacen que la estructura de la red evolucione mediante sucesivas particiones en el espacio de entrada en el cual se ha definido el problema. Cada una de estas particiones conferirá una mayor precisión a la aproximación lineal de la función discriminante que se desea obtener para resolver el problema de clasificación. A este tipo de algoritmos pertenecen, por ejemplo el *algoritmo Tiling*, [Mezard y col., 1989], el algoritmo de Marchand, [Marchand y col., 1990], el algoritmo PSIN, [Sun y col., 1988], el algoritmo de Knerr, [Knerr y col., 1991], el algoritmo CID3, [Cios y col., 1992] o el algoritmo de Nadal, [Nadal, 1989].
2) Aquellos algoritmos en los cuales las nuevas unidades añadidas a la red tienen como misión la corrección de los errores de clasificación asociados a las unidades generadas con anterioridad. En definitiva, esta estrategia queda reducida también a un proceso de partición iterada del conjunto de vectores de entrada con el fin de obtener la mejor aproximación lineal a la función discriminante que se desea conseguir. Pertenecen a esta categoría, entre otros, el *algoritmo Upstart*, [Frean, 1990], el algoritmo de Sato, [Sato y col., 1991], el algoritmo PSOHNN, [Ersoy y col., 1990], el algoritmo *Cascade Correlation*, [Fahlman y col., 1990] o el algoritmo de Hiroshe, [Hiroshe y col., 1991].

Por lo que respecta a la topología de red a que dan lugar los algoritmos incrementales PLS, podemos diferenciar dos grandes grupos de algoritmos:

1) Algoritmos PLS que generan una estructura de red del tipo árbol de decisión, como los algoritmos *Upstart* y *Neural Trees*.
2) Algoritmos que generan una arquitectura de red del tipo perceptron multinivel, [Rumelhart y col., 1986], como, por ejemplo, los algoritmos *Cascade Correlation*, *Tiling*, CID3, algoritmo de Nadal, etc.

No obstante, si bien existen dos tipos distintos de topologías de red generadas, las estructuras de tipo árbol de decisión se pueden convertir fácilmente, tal y como se demostró en [Bigot y col., 1993], en estructuras de tipo perceptron multinivel. Por tanto, se puede considerar que los algoritmos incrementales PLS dan lugar a lo que podríamos denominar arquitecturas de perceptron multinivel incrementales, para las cuales ya se han efectuado algunas aproximaciones, [Moreno y col., 1993c], de cara a llevar a cabo su implementación física en hardware.

La tabla I recoge a modo de resumen las características más destacadas de los algoritmos incrementales PLS mencionados anteriormente. Además del tipo de topología generada y del criterio utilizado para generar la red, se incluyen como características a tener en cuenta la posibilidad de generar un número variable de niveles ocultos y la posibilidad de que el nivel de salida pueda contener más de una unidad, es decir, que el algoritmo básico sea capaz de discriminar entre más de dos clases o categorías.

TABLA I. Tabla comparativa para varios algoritmos incrementales PLS.

	Arbol decisión	Perceptrón multinivel	Partición iterada	Corrección unidades	Niveles variables	Salida variable
Marchand		X	X			
Upstart	X			X		
Tiling		X	X		X	
Neural Trees	X		X			
PSIN	X		X			
Sato		X		X	X	X
Knerr		X	X			X
PSOHNN		X		X	X	X
CID3		X	X		X	X
Cascade Correlation		X		X	X	X
Nadal		X	X	X		
Hiroshe		X		X		

4. REDES DE REGIÓN DE INFLUENCIA (ROI)

4.1. Bases de funcionamiento

Un clasificador de este tipo muy utilizado es el clasificador por vecino más próximo. Este clasificador, a diferencia de los vistos en la sección anterior, no almacena parámetros que definen una función discriminante, sino que almacena patrones alrededor de los cuales se define una región que corresponde a una clase (la región de influencia). La formulación para el clasificador de vecino más próximo es:

$$\text{para} \quad d(X, X^p) < d(X, X^m), \quad \forall m \neq p \tag{5}$$

asociar X a la clase representada por θ^p

donde: X^i representa un patrón i almacenado por el clasificador,
θ^p es la etiqueta que indica la pertenencia de X^p a una clase,
$d(X, X^n)$ es la distancia entre el patrón presentado, X, y X^n.

La razón por la cual este clasificador funciona bien es la siguiente. Si X^m representa el patrón almacenado por la red más cercano a X, y θ^m es una etiqueta que asocia X^m a una clase particular, entonces la probabilidad de que $\theta^m=w_i$ es simplemente la probabilidad a posteriori, $P(w_i \mid X^m)$. Si el clasificador almacena muchos patrones, entonces:

$$P(w_i \mid X^m) \approx P(w_i \mid X) \tag{6}$$

ya que el patrón de entrada estará muy próximo a algún patrón almacenado por el clasificador, en este caso X^m es la más próxima. Nótese que el clasificador por vecino más próximo funciona como una regla de decisión aleatoria que clasifica X seleccionando la clase w_i con probabilidad $P(w_i \mid X)$ [Duda, 1973]. En otras palabras, la clase asociada al patrón más cercano al de entrada es la adjudicada a ella. Con un gran número de muestras es evidente que la proporción de muestras que pertenecen a clase w_i en el espacio alrededor de X es aproximadamente $P(w_i \mid X)$, razón por la cual la clase w_i se escoge con esta misma probabilidad. El único cálculo que se ha de efectuar es por tanto la evaluación de la distancia entre el patrón de entrada y cada una de las muestras.

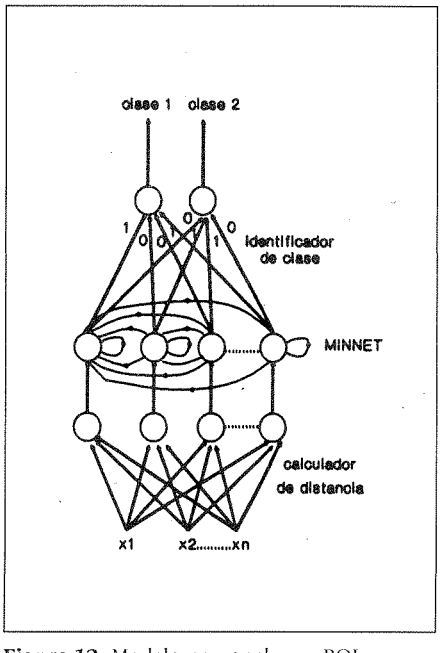


Figura 12. Modelo neuronal para ROIs.

Para entender el funcionamiento de este tipo de redes, no es necesario tener que acudir a las redes neuronales, sino que es perfectamente posible explicar su funcionamiento desde otra perspectiva. En cambio, la perspectiva de las redes neuronales muestra la inherente paralelización del proceso, que hace posible su realización en hardware. El proceso explicado en la expresión (5) se podría representar con la red neuronal de la figura 12. Esta red neuronal comprende de tres capas distintas. La capa inferior es la que calcula la distancia entre el patrón de entrada y cada uno de los ejemplares almacenados por la red. Cada una de las neuronas de esta capa inferior está calculando una de las distancias de la expresión (5). La siguiente capa es una MINNET que escoge la neurona que presenta la mínima distancia. Sólo una de estas neuronas estará activa en régimen estacionario. Esta capa se puede realizar utilizando el complemento de la MAXNET para el cual existen varias posibilidades (ver [Pao, 1989; Robinson, 1992]). La última capa únicamente

indica la pertenencia del ejemplar (neurona activada) a una clase, representado mediante un peso +1. En esta última capa hay tantas neuronas como clases.

Esta misma estructura no sólo sirve para crear clasificadores, sino también cuantificadores vectoriales y memorias asociativas [Castillo, 1992]. Para entender cómo, en la figura 13 se representa el espacio de entrada y se indican las diferentes regiones asociadas a cada ejemplar (marcado con una x). Si se asocia un peso a cada componente utilizada en el cálculo de la distancia:

$$d^j(X,X^j) = a_1|x_1 - x_1^j| + a_2|x_2 - x_2^j| + a_3|x_3 - x_3^j| + \ldots \qquad (7)$$

se pueden modificar los pesos, a_i, en función de la importancia que tiene una característica en la búsqueda global. Así por ejemplo, si se dispone de las regiones mostradas en la figura 14 cuando los coeficientes de la variable representada en el eje-x y el eje-y son iguales, $a_1=a_2$, y se desea buscar el patrón más parecido a X teniendo en cuenta sólo la característica x_2, entonces se escoge $a_2=0$ y las regiones que se forman a partir de esta búsqueda son las mostradas en la figura 15. Jugando con estos coeficientes es posible traducir una serie de directrices simbólicas o borrosas a un equivalente numérico. Estos cálculos efectuados en soft consumen mucho tiempo, pero con unas estructuras de hardware adecuadas, es posible acelerar la obtención de resultados [Castillo, 1993a].

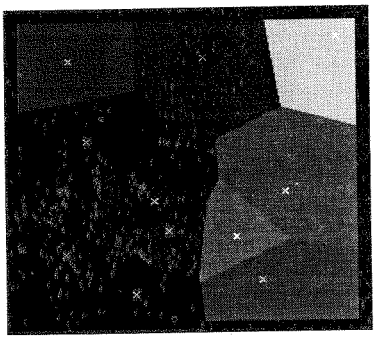


Figura 13. Teselación de Voronoi.

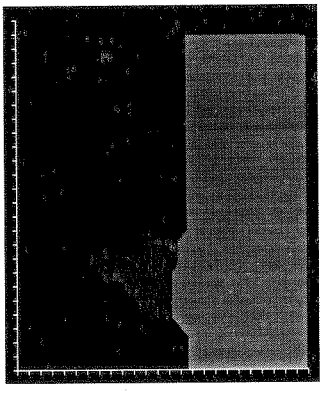


Figura 14. Ejemplo de regiones para el cual se modifican los coeficientes a_i.

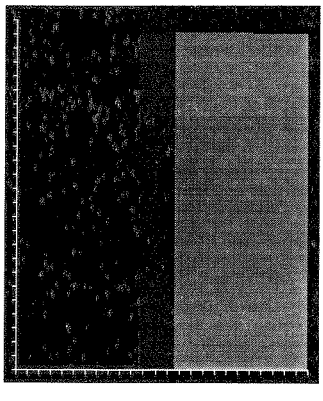


Figura 15. Regiones obtenidas tras poner el valor del coeficiente a_2 correspondiente al eje-y, a cero.

4.2. RCE, GAL

El algoritmo *Reduced Coulomb Energy* (RCE) [Reilly, 1982] consiste en el almacenamiento de patrones y asociar un radio a cada uno de éstos, que será el parámetro que define el volumen de la región asociada al ejemplar. El tipo de distancia empleado define la forma de la región.

En su versión más simple el algoritmo es tal y como se detalla a continuación:

1. si $d(X, X^n) < r^n$, $\Theta^n = \Theta'$
   y $d(X, X^i) > r^i$, $\Theta^i \neq \Theta'$ $\forall i \neq n$

   entonces X ha sido clasificado correctamente y ninguna modificación se efectúa sobre el clasificador

2. si $d(X, X^i) > r^i$, $\forall i$

   entonces un nuevo patrón es almacenado y etiquetado a su clase correspondiente
   $(N(t+1) = N(t) + 1, X^N = X, \Theta^N = \Theta')$

3. si $d(X, X^n) < r^n$ y $\Theta^n \neq \Theta'$

   entonces existe un conflicto, ya que el patrón de entrada ha sido clasificado erróneamente. Para corregir esto, r^n se reduce por debajo de $d(X, X^n)$, y X se almacena como un nuevo ejemplar. ($N(t+1) = N(t) + 1, X^N = X, \Theta^N = \Theta'$)

donde
- r^i radio del ejemplar X^i,
- N número total de ejemplares,
- Θ' etiqueta de pertenencia del patrón de entrada.

El aprendizaje se efectúa presentando patrones de forma secuencial y calculando si caen dentro de la región de influencia de algún ejemplar (patrón almacenado previamente). Si el ejemplar pertenece a la misma clase que el patrón de entrada (1er paso) entonces no se realizan cambios sobre el clasificador. Si por el contrario no cae dentro de la región de influencia de ningún ejemplar, entonces el patrón se almacena en el clasificador como un nuevo ejemplar y se le asigna un radio inicial (2º paso y figuras 16-18). Por último, si existe un conflicto, es decir, si el patrón de entrada cae dentro de una región de una clase distinta, entonces el radio del ejemplar correspondiente se reduce para excluir el patrón de entrada, el patrón de entrada se toma como nuevo ejemplar y se le asigna un radio inicial pequeño (3er paso y figura 19).

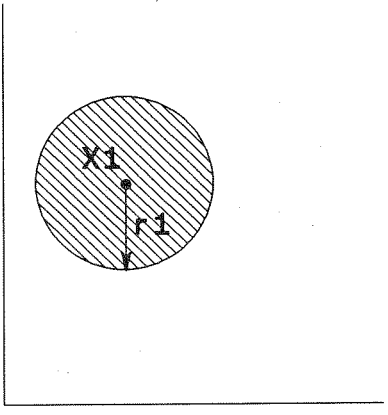


Figura 16. X^1 es almacenado.

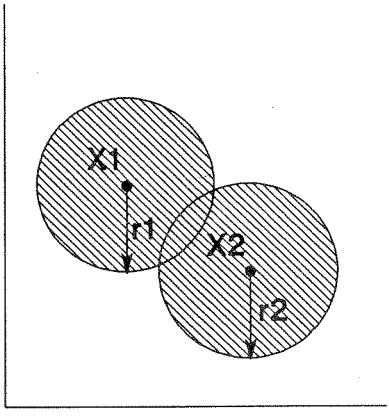


Figura 17. 2º patrón, X^2 se almacena.

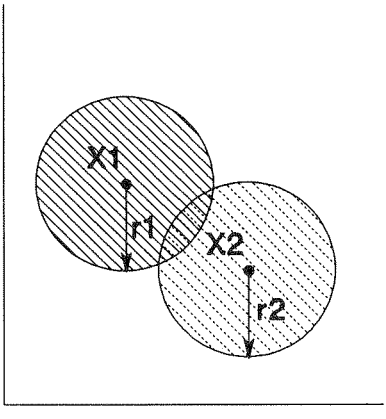


Figura 18. Patrón X^2 se almacena y nueva clase es creada.

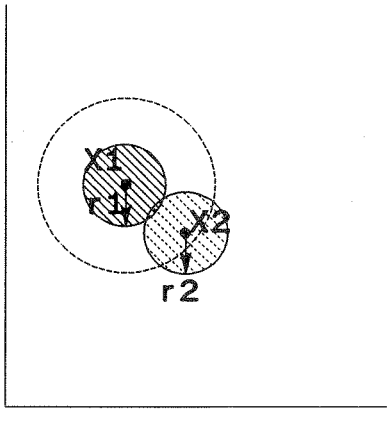


Figura 19. El 2º patrón tiene un conflicto con el 1º. Se reduce el radio de X^1 y X^2 es creado.

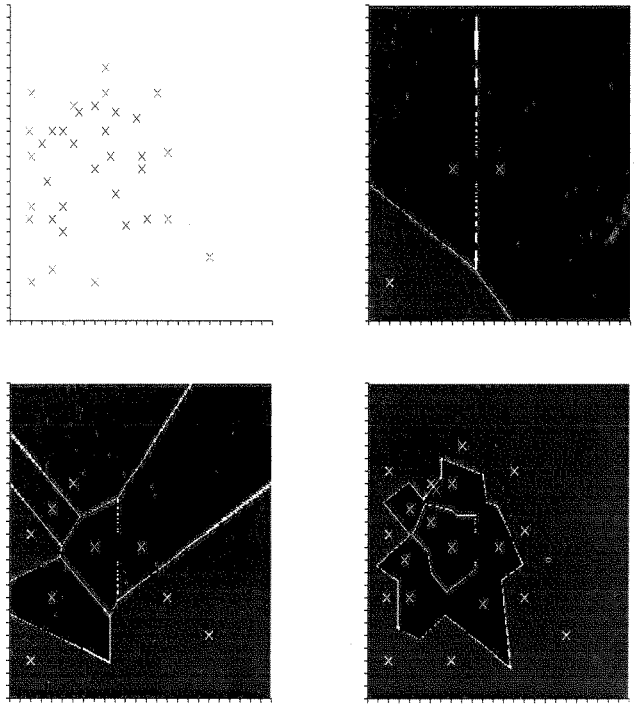
Otro algoritmo ROI propuesto en [Alpaydin, 1990] se llama el *Grow and Learn* (GAL). La red construida por este algoritmo es un clasificador de vecino más próximo, pero la diferencia reside en el entrenamiento. A la red se le presentan patrones y sólo los mal clasificados se almacenan como nuevos ejemplares. Si un patrón es clasificado correctamente, a éste se le ignora y se toma el siguiente patrón. De esta forma se construye un clasificador por vecino más próximo sin necesidad de almacenar todos los patrones del conjunto de entrenamiento:

$$\text{si} \qquad d(X, X^k) = \min_i d(X, X^i) \tag{8}$$

entonces: si $\Theta^k = \Theta'$, rechazar X,
sino $\Theta^k \neq \Theta'$, almacenar X como nuevo ejemplar
($N(t+1) = N(t) + 1$, $X^N = X$, $\Theta^N = \Theta'$)

Las figuras 20-23 muestran un ejemplo de entrenamiento utilizando el GAL.

Estos dos algoritmos pueden producir redes aceptables tras haberse presentado los patrones una sóla vez. Aún así, normalmente es necesario presentar los patrones varias veces a la red ya que la red resultante depende del orden de presentación. También es posible utilizar criterios para eliminar ejemplares después de algunas iteraciones y de esta forma evitar muestras poco representativas (ver [Alpaydin, 1990]).



Figuras 20-23. La figura superior de la izquierda muestra todos los patrones utilizados en el entrenamiento. Las otras muestran el estado del espacio de entrada en instantes diferentes del entrenamiento. Los datos son concéntricos y de 3 clases.

4.3. Redes de función de base radial (*Radial Basis Functions -RBF- networks*)

Los principios que definen el comportamiento de las redes *RBF* se derivan de los modelos clásicos de la teoría de decisión y clasificación, [Duda y Hart, 1973]. Sin embargo, el hecho de que se hayan incluido dentro del grupo de redes neuronales incrementales es debido a que recientemente se han propuesto ciertos métodos, [Lee y col., 1991; Platt, 1991; Musavi y col., 1992] que, actuando básicamente sobre los algoritmos de agrupación (*clustering*) de patrones utilizados durante el entrenamiento de las redes RBF, permiten que la red adquiera la estructura más apropiada dependiendo del problema que se pretenda resolver.

A diferencia del resto de algoritmos incrementales de región de influencia presentados anteriormente (*RCE* y *GAL*), este tipo de modelos neuronales no sólo se utilizan para llevar a cabo funciones de clasificación, sino que también pueden resolver funciones de aproximación o interpolación de funciones, [Broomhead y col., 1988].

La estructura básica de una red *RBF* es la que se muestra en la figura 24.

Como puede observarse, la red está compuesta por dos niveles. El primer nivel está constituido por unidades que ofrecen una salida cuyo valor depende de la proximidad del vector de entrada al vector que representa dicha unidad. Es por ello que este tipo de redes se denominan también redes de campo receptivo localizado (*localized receptive field networks*). Este es el motivo por el cual hemos incluido este tipo de modelos dentro de los algoritmos incrementales de región de influencia. El valor de salida suministrado por estas unidades se obtiene por evaluación de una función que depende de la distancia del vector de entrada al vector que representa la unidad (de ahí el nombre de funciones radiales). Las funciones utilizadas más habitualmente son funciones gausianas normalizadas, es decir:

$$o_i = e^{-\left(\frac{(x-w_i)^T (x-w_i)}{2\sigma_i^2}\right)} \qquad i = 1 \ldots N \tag{9}$$

donde:

* o_i: Salida de la unidad i del primer nivel,
* x : Vector de entrada,
* w_i: Vector que define a la unidad i,
* σ_i: Parámetro de normalización para la unidad i,
* N: Número de unidades en el primer nivel.

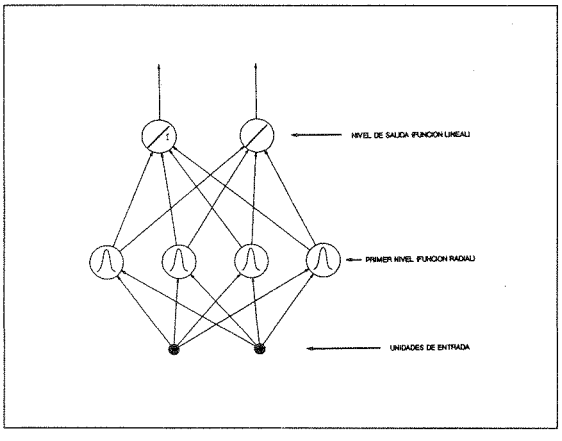


Figura 24. Estructura de una red RBF.

Existen otros tipos de funciones radiales susceptibles de ser empleadas por las unidades del primer nivel para determinar el grado de proximidad del vector de entrada al vector que define la unidad. En [Carlin, 1992] puede encontrarse una amplia revisión sobre la aplicación de distintos tipos de funciones radiales en este tipo de redes.

Por su parte, las unidades de salida realizan simplemente una combinación lineal de las salidas obtenidas por la unidades del primer nivel, las cuales son ponderadas mediante una serie de pesos que las conectan con las unidades de salida. El hecho de que las unidades de salida sean lineales es lo que permite que este tipo de redes sean utilizadas tanto para resolver problemas de clasificación como de interpolación de funciones.

Por lo que respecta al entrenamiento de este tipo de redes, generalmente se suele dividir en dos clases claramente diferenciadas: entrenamiento de las unidades del primer nivel y entrenamiento de las unidades del nivel de salida.

Como se ha mencionado anteriormente, el entrenamiento de las unidades del primer nivel se suele llevar a cabo habitualmente mediante algoritmos de agrupación (*clustering*), los cuales se encargan de determinar, normalmente mediante un proceso de agrupación por similitud de distancia, los vectores que definen a las unidades del primer nivel. Los algoritmos más habituales para llevar a cabo este proceso de agrupamiento son el algoritmo de las K medias (*K-means algorithm*), [Duda y Hart, 1973] o el algoritmo *ISODATA*, [Ball y Hall, 1966]. La figura 25 muestra el diagrama de flujo correspondiente al algoritmo *K-means*. En dicha figura no se muestra el cálculo del parámetro de normalización de cada agrupación, σ_i, que suele determinarse habitualmente como la desviación típica de los vectores asociados a cada agrupación.

El principal problema que presentan estos algoritmos es debido a que el número de agrupaciones que deben ser determinadas en el conjunto de vectores de entrada es un parámetro que debe ser fijado externamente, y que además determinará en buena medida el correcto funcionamiento de la red como sistema clasificador o interpolador. Es por ello que, como se indicó con anterioridad, se han

efectuado propuestas para hacer que los algoritmos de agrupación sean capaces de determinar por si mismos el número de agrupaciones que deben ser creadas para cada problema.

En el caso del algoritmo propuesto en [Lee y col., 1991], el proceso de generación de agrupaciones no sólo se encarga de añadir nuevas unidades, sino que también la forma de las funciones radiales es modificada durante el entrenamiento. El objetivo del algoritmo consiste en comenzar con un número reducido de unidades con un amplio área de influencia (valor elevado en el parámetro de normalización σ_i), para posteriormente ir añadiendo unidades con menor área de influencia, a fin de obtener una aproximación sucesivamente más precisa de la relación entrada/salida que se desea conseguir. El algoritmo de agrupación presentado en [Platt, 1991] también trata de conseguir que la relación entrada/salida que obtiene la red se ajuste lo mejor posible a la relación deseada mediante la adición de nuevas unidades. Sin embargo, en este caso sólo se añaden nuevas unidades cuando el error cometido por la red excede un cierto valor. En caso contrario, se lleva a cabo un proceso de adaptación sobre los vectores que definen dichas unidades, haciendo que dichos vectores se muevan en la dirección que permite reducir la tasa de error. Por lo que respecta al algoritmo de agrupación propuesto en [Musavi y col., 1992], inicialmente comienza con un número de agrupaciones igual al de patrones de entrenamiento, para sucesivamente llevar a cabo un proceso de mezcla de agrupaciones dependiendo del grado de solapamiento existente entre sus áreas de influencia.

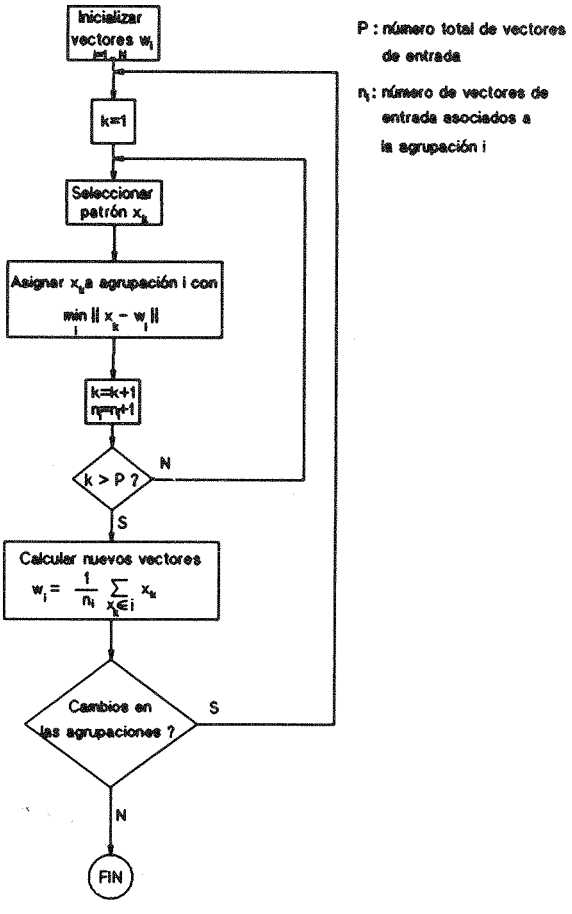


Figura 25. Algoritmo K-means.

Finalmente, el entrenamiento de las conexiones entre las unidades de salida y las unidades del primer nivel se suele llevar a cabo mediante algoritmos del tipo gradiente descendiente, como el algoritmo LMS [Widrow y Hoff, 1960].

5. REDES NEURONALES PROBABILISTAS (PNN)

5.1. Ventajas y limitaciones de los métodos probabilistas

Aunque las redes neuronales que estiman directamente la probabilidad a posteriori o que se basan en aproximar regiones o discriminantes obtenidos directamente de ella son mucho más simples de entrenar, también tienen ciertas desventajas. Una de las desventajas más grandes es que al operar directamente sobre las muestras, se puede dar un sobre entrenamiento si **todas** las muestras del conjunto de aprendizaje se aprenden, incluyendo muestras no representativas.

La figura 26 muestra como un PLS entrenado sobre unas muestras producidas por unas distribuciones solapadas resulta en un discriminante bastante complicado, cuando en este caso, debido al tipo de distribución, una recta vertical entre las dos medias de las distribuciones daría menor probabilidad de error.

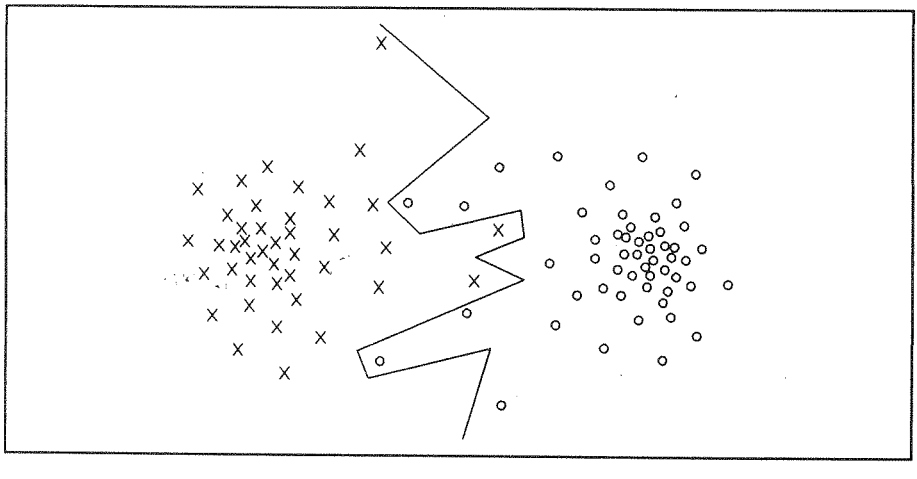


Figura 26. Ejemplo de sobre entrenamiento para un problema de 2 categorías.

Otro posible problema es que al evitar la estimación de la densidad de probabilidad y la probabilidad a priori, un cambio en la concentración de patrones de las diferentes clases producirá un aumento correspondiente en el error, siempre de acuerdo con (1). Los métodos estadísticos evitan estos problemas ya que estiman la densidad de probabilidad, y por tanto, están más cerca al óptimo de Bayes. Casi todas estas técnicas probabilistas utilizan métodos que dividen el espacio de entrada en diferentes regiones y tratan de estimar la densidad de probabilidad dentro de cada una de estas regiones.

5.2. Kernels y Ventanas de Parzen

Calculando la probabilidad P de que un patrón X caiga dentro de la región $\Re$:

$$P = \int_{\Re} p(X')dX' \tag{10}$$

si n muestras son obtenidas de forma independiente, k de las cuales caen dentro de la región $\Re$, una posible estimación de P es:

$$P \approx k/n \tag{11}$$

Suponiendo p(X) continuo y la región $\Re$ suficientemente pequeña como para que p no varíe demasiado dentro de esta región:

$$\int_{\Re} p(X')dX' \approx p(X)V \tag{12}$$

donde X es un punto en $\Re$ y V es el volumen de $\Re$. De las expresiones (10), (11) y (12):

$$\hat{p}(X) \approx p(X) = \frac{k/n}{V} \tag{13}$$

Si V se fija y se toman más y más muestras, lo que se calcula es una media espacial de p(x):

$$\frac{P}{V} = \frac{\int_{\Re} p(X')dX'}{\int_{\Re} dX'} \tag{14}$$

Generalmente, lo que se desea obtener son estimaciones de p(X) en regiones pequeñas y no únicamente su valor medio, ya que p(X) puede variar considerablemente de la media, dependiendo de la región. Esto implica que a medida que se

toman más muestras, se aminoran los volúmenes de las regiones para que todo el espacio de entrada esté cubierto por regiones pequeñas, dentro de cada cual se obtiene una buena estimación de p(X). Por tanto, vamos a representar con V_n el volumen de la región $\Re_n$ y k_n representará el número de patrones que caen dentro de esta región. $\hat{p}_n$ será la n-ésima estimación de p(X):

$$\hat{p}_n(X) = \frac{k_n / n}{V_n} \tag{15}$$

En el caso más simple, cada región $\Re_n$ será un hipercubo de d dimensiones donde h_n es la longitud de uno de sus bordes. Su volumen es por tanto:

$$V_n = (h_n)^d \tag{16}$$

Ahora definamos una función q(X):

$$q(X) = \begin{cases} 1 & |x_j| \leq 1/2 \qquad j=1,...,d \\ 0 & \text{si no} \end{cases} \tag{17}$$

que indica que q(X) es un hipercubo centrado sobre el origen. Entonces $q((X-X^i)/h_n)$ es igual a 1 cuando X^i cae dentro de este hipercubo de volumen V_n centrado sobre X. El número de muestras k_n en región $\Re_n$ es entonces:

$$k_n = \sum_{i=1}^{n} q\left(\frac{X - X^i}{h_n}\right) \tag{18}$$

y obtenemos una estimación para p(X) sustituyendo esta última expresión en (15):

$$\hat{p}_n(X) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{V_n} q\left(\frac{X - X^i}{h_n}\right) \tag{19}$$

Esta expresión es la llamada función kernel de la cual se pueden utilizar otros tipos de funciones tales como gaussiana o epanechnikov [Epanechnikov, 1969]. Si la h_n es la misma en todas las regiones, entonces nos resulta el algoritmo de las ventanas de Parzen [Parzen, 1962; Cacoullos, 1966; Specht, 1990]. Si al contrario, el tamaño de la región cambia dependiendo del kernel, entonces se dice que son adaptativas. La problemática asociada a este método es, primero, escoger los volúmenes iniciales y, segundo, definir una buena regla para disminuir el tamaño de las regiones a medida que se toman muestras.

Estos métodos son incrementales ya que almacenan todos los patrones en la red, y es fácil añadir más patrones. En cambio, si no se desea almacenar todos los patrones es necesario efectuar algún tipo de agrupación de patrones antes del aprendizaje, y la red deja de ser incremental.

5.3. Otros métodos

Otra posibilidad al estimar la densidad de probabilidad es utilizar la regla de vecino más próximo, parecido al presentado en la sección 4.1, con la diferencia de que aquí estimaremos $p(X \mid w_j)$. Utilizando este método, escogemos el volumen de la región que viene definido por los m vecinos más próximos. Cuando se toman más muestras, el volumen se reduce automáticamente. Si $d_m(X)$ representa la distancia entre X y su m-ésimo vecino en el espacio de entrada, escogemos el volumen de acuerdo con:

$$V_m(X) = c_d \; d_m(X)^d \tag{20}$$

y sustituyendo en (15) nos da una estimación de la densidad:

$$\hat{p}(X) = \frac{k}{n} \frac{1}{V_m(X)} \tag{21}$$

Una vez más, existe una problemática, en este caso al escoger m. Si m es demasiado grande, entonces la expresión (21) nos dará una media espacial de p(X). Si por el contrario m es demasiado pequeño, las estimaciones de p(X) serán muy diferenciadas al pasar de una región a otra.

CONCLUSIONES

En este capítulo se ha presentado una panorámica general sobre los modelos incrementales para redes neuronales artificiales. En primer lugar, se han destacado las limitaciones inherentes a los modelos clásicos de redes neuronales artificiales, indicando cómo estas limitaciones pueden ser superadas mediante los nuevos modelos neuronales evolutivos.

Tras describir los principios generales de organización que utilizan este tipo de modelos evolutivos, se ha presentado una taxonomía de los modelos evolutivos incrementales, los cuales pueden ser agrupados, tal y como se indicó, en tres grupos básicos dependiendo de la estrategia que emplean para resolver el problema propuesto.

Teniendo en cuenta que la aplicación más frecuente en la que se emplean los modelos neuronales evolutivos consiste en la resolución de problemas de clasificación, se ha presentado una breve introducción a la teoría clásica de decisión y clasificación. Con ello se ha pretendido fijar los objetivos y características del proceso de decisión, así como resaltar las características afines que presentan los modelos neuronales evolutivos con respecto a la teoría clásica de clasificación.

Finalmente, se ha ido pasando revista en sucesivas secciones a los diferentes modelos evolutivos incrementales que se señalaron al comienzo, resaltando especialmente sus principios de funcionamiento y organización y sus posibles dominios de aplicación.

REFERENCIAS

Alpaydin, A.I., *Neural Models of Incremental Supervised and Unsupervised Learning*, Tesis doctoral, EPFL Lausanne, Suiza, 1990.

Ball, G.H. y Hall, D.J., «ISODATA: An Iterative Method of Multivariate Data Analysis and Pattern Classification», *Proceedings of the IEEE International Communications Conference*, (1966), 116-117.

Bigot, P. y Cosnard, M., «Probabilistic Decision Trees and Multilayered Perceptrons», *European Symposium on Artificial Neural Networks*, (1993), 91-96.

Broomhead, D.S. y Lowe, D., «Multivariable Functional Interpolation and Adaptive Networks», *Complex Systems*, 2, (1988), 321-355.

Cacoullos, «Estimation of a multivariate density», *Annals of Inst. of Stat. Math.*, 18, (1966), 178-189.

Carlin, M., «Radial Basis Function Networks and Nonlinear Data Modelling», *Proceedings of Neuronimes'92*, (1992), 623-631.

Castillo, F., Cabestany, J. y Moreno, J.M., «Hardware Realizeable Associative Memories, Vector Quantizers, and Classifiers based on Competitive Neural Networks», *Proc. of the First IFIP WG-10.6 Workshop*, Grenoble, Francia, (1992), 57-61.

Castillo, F., Cabestany, J. y Moreno, J.M., «Region of Influence (ROI) Networks. Model and Implementation», en Mira, J., Cabestany, J. y Prieto, A. (Eds.), *New Trends in Neural Computation*, Springer-Verlag, 1993a, 96-101.

Castillo, F., «Statistics and Neural Network classifiers: A review from Multilayered Perceptrons to incremental Neural Networks», en Huning, H., Neuhauser, S., Raus, M. y Ritschel, W. (Eds.), *Workshop on Neural Networks at RWTH Aachen*, Aachen, Augustinus Verlag, 1993b.

Cios, K.J. y Liu, N., «A Machine Learning Method for Generation of a Network Architecture: A Continuous ID3 Algorithm», *IEEE Transactions on Neural Networks*, 3-2, (1992), 280-291.

Cybenko, G., «Approximation by Superpositions of Sigmoidal Function», *Mathematics of Control, Signals, and Systems*, 2, (1989), 303-314.

Duda, R. y Hart, P., *Pattern Classification and Scene Analysis*, J.Wiley & Sons, 1973.

Ersoy, O.K. y Hong, D., «Parallel, Self-Organizing, Hierarchical Neural Networks», *IEEE Transactions on Neural Networks*, 1-2, (1990), 167-178.

Epanechnikov, V.A., «Non-parametric estimation of a multivariate probability density», *Theory Proba. Appl.*, Vol. 14, (1969), 153-158.

Fahlman, S.E. y Lebiere, C., «The Cascade Correlation Learning Architecture», *Report de la universidad Carnegie Mellon*, N. CMU-CS-90-100, (1990).

Frean, M., «The Upstart Algorithm: A Method for Constructing and Training Feedforward Neural Networks», *Neural Computation*, 2, (1990), 198-209.

Gallant, S.I., «Optimal Linear Discriminants», *Proceedings of the 8th International Conference on Pattern Recognition*, 2, (1986), 849-854.

Hiroshe, Y, Yamushita, K. y Hijiya, S, «Back Propagation Algorithm which varies the number of Hidden Units», *Neural Networks*, 2, (1991), 61-66.

Jutten, C. y Comon, P., «Neural Bayesian Classifier», en Mira, J., Cabestany, J. y Prieto, A. (Eds.), *New Trends in Neural Computation*, Springer-Verlag, 1993, 119-124.

Knerr, S., Personnaz, L. y Dreyfus, G., «A New Approach to the Design of Neural Network Classifiers and its Application to the Automatic Recognition of Handwritten Digits», *International Joint Conference on Neural Networks*, (1991), 91-96.

Lapedes, A. y Farber, R., «How Neural Nets Work», en Anderson, D.Z. (Ed.), *Neural Information Processing Systems*, Nueva York: American Institute of Physics, 1988, 442-456.

Lee, S. y Rhee, M.K, «A Gaussian Potential Function Network with Hierarchically Self-Organizing Learning», *Neural Networks*, 4, (1991), 207-224.

Marchand, M., Golea, M. y Ruján, P., «A Convergence Theorem for Sequential Learning in Two-Layer Perceptrons», *Europhysics Letters*, 11, (1990), 487-492.

Mezard, M. y Nadal, J.P., «Learning in Feedforward Layered Networks: The Tiling Algorithm», *Journal of Physics*, A22, (1989), 2191-2203.

Moreno, J.M., Castillo, F. y Cabestany, J., «Enhanced Unit Training for Piecewise Linear Separation Incremental Algorithms», *European Symposium on Artificial Neural Networks*, (1993a), 33-38.

Moreno, J.M., Castillo, F. y Cabestany, J., «Optimized Learning for Improving the Evolution of Piecewise Linear Separation Incremental Algorithms», en Mira, J., Cabestany, J. y Prieto, A. (Eds.), *New Trends in Neural Computation*, Springer-Verlag, Berlin, 1993b.

Moreno, J.M., Castillo, F. y Cabestany, J., «Hardware Implementation of Piecewise Linear Separation Incremental Algorithms», *Proceedings of Neuronimes'93*, (1993c), 199-208.

Moreno, J.M., Castillo, F. y Cabestany, J., «Improving Piecewise Linear Separation Incremental Algorithms using Complexity Reduction Methods», a publicarse en European Symposium on Artificial Neural Networks, (1994).

Musavi, M.T., Ahmed, W., Chan, K.H., Faris, K.B. y Hummels, D.M., «On the Training of Radial Basis Function Classifiers», *Neural Networks*, 5, (1992), 595-603.

Nadal, J.P., «Study of a Growth Algorithm for a Feedforward Network», *International Journal of Neural Systems*, 1-1, (1989), 55-59.

Parzen, E., «On estimation of a probability density function and mode», *Ann. Math. Stat.*, 33, septiembre, (1962), 1065-1076.

Platt, J., «A Resource Allocating Network for Function Interpolation», *Neural Computation*, 3, (1991), 213-225.

Reilly, D.L., Cooper, L.N. y Elbaum, C., «A Neural Model for Category Learning», *Biological Cybernetics*, 45, (1982), 35-41.

Robinson, M.E., Yoneda, H. y Sánchez-Sinencio, E., «A Modular CMOS Design of a Hamming Network», *IEEE Trans. on Neural Networks*, mayo, 1992, 444-456.

Rosenblatt, F., *Principles of Neurodynamics*, New York: Spartan, 1962.

Rumelhart, D.E. y McClelland, D.E., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, 1986.

Sato, A., Yamada, K., Tsukumo, J. y Temma, T., «Neural Network Models for Incremental Learning», en Kohonen, T., Makisara, K. y Simula, O. (Eds.), *Artificial Neural Networks*, Elsevier Science Publishers, 1991.

Sirat, J.A. y Nadal, J.P., «Neural Trees: A New Tool for Classification», *Technical Report*, Laboratoires d'Electronique Philips, (1990).

Specht, D., «Probabilistic Neural Networks», *Neural Networks*, Vol. 3, N. 1, (1990).

Sun, G.Z., Chun, H.H. y Lee, Y.C., «Parallel Sequential Induction Network: A New Paradigm of Neural Network Architecture», *IEEE International Conference on Neural Networks*, (1988), 489-496.

Widrow, B. y Hoff, M., «Adaptive switching circuits», *1960 IRE WESCON Convention Record*, (1960), 96-104.