1 El algoritmo constructivo C-Mantec

1.1. Introducción

Tal y como se comentó en el Capítulos anteriores, la elección de la arquitectura de red neuronal adecuada para un problema de clasificación es una tarea difícil y no hay un acuerdo general sobre la estrategia a seguir a fin de seleccionar una arquitectura de red neuronal óptima, siendo todavía el método computacionalmente ineficiente de "prueba y error" muy utilizado en las aplicaciones que utilizan redes neuronales artificiales.

En este capítulo se presenta el algoritmo C-Mantec (*Competitive MAjority Network Training Error Correcting*), que introduce la competición entre las neuronas en la fase de entrenamiento impidiendo la congelación de los pesos sinápticos y permitiendo que todas las neuronas puedan aprender en cualquier etapa del procedimiento. Este enfoque cooperativo del proceso de aprendizaje conduce a una mejora global en el rendimiento de la red.

La competencia entre las neuronas se utiliza mucho en el aprendizaje no supervisado (reducción de dimensiones, agrupación de patrones, etc.), tanto en el modelado artificial como en el biológico [**?**, **?**, **?**, **?**], pero ha sido menos aplicado en los sistemas de aprendizaje supervisado [**?**]. C-Mantec hace uso de la regla de aprendizaje del perceptrón termal [**?**] para que las neuronas de la capa oculta puedan competir en la fase de entrenamiento. El perceptrón termal es más estable que el perceptrón simple clásico, al ser capaz de mantener el conocimiento adquirido usando una tasa de aprendizaje dinámica que impide grandes modificaciones de los pesos sinápticos cuando se quiere aprender un ejemplo mal clasificado con un potencial sináptico muy alejado del plano separador generado por el perceptrón, evitando así olvidar en exceso lo que había aprendido hasta ese momento. Esta propiedad permite que el perceptrón termal genere muy buenas aproximaciones lineales a un problema no separable, lo que será utilizado por el algoritmo C-Mantec para la competición de las neuronas en la fase de entrenamiento de manera que la neurona que tenga que modificar en menor grado los pesos sinápticos para aprender un ejemplo, es decir, la que se encuentre más cerca de aprenderlo, será la neurona ganadora candidata a modificar sus pesos sinápticos. Si ninguna neurona desea aprender el ejemplo, esto es, si la neurona candidata seleccionada para aprender considera que el cambio de sus pesos sinápticos implica desaprender mucho de lo aprendido previamente, podrá negarse a realizar esa tarea, con lo que se añadirá una nueva unidad a la arquitectura cuya primera tarea será aprender ese ejemplo concreto.

Un grave problema que afecta a casi todos los algoritmos de aprendizaje, y en particular a los constructivos, es el problema del sobre-ajuste [**?**, **?**, **?**]. En aprendizaje automático el sobre-ajuste (también es frecuente emplear el término en inglés *"over-fitting"*) es el efecto de sobre-entrenar un algoritmo de aprendizaje con unos ciertos datos para los que se conoce el resultado deseado. Un algoritmo de aprendizaje debe alcanzar un estado en el cual sea capaz de predecir el resultado de patrones distintos a los utilizados en la fase de entrenamiento. En ese estado se dice que el algoritmo

puede resolver situaciones distintas a las acaecidas durante el entrenamiento al haber aprendido el problema. Sin embargo, cuando un sistema se entrena demasiado el algoritmo puede quedar ajustado a unas características muy específicas de los datos con los que se entrenó, que no tienen relación causal con la función objetivo tal, y como se observa en la Fig. 1.1. Cuando un algoritmo entra en la fase de sobre-ajuste, el error global al responder frente a los patrones entrenamiento sigue decreciendo mientras que su respuesta ante nuevos patrones va empeorando tal y como se muestra en la Fig. 1.2.

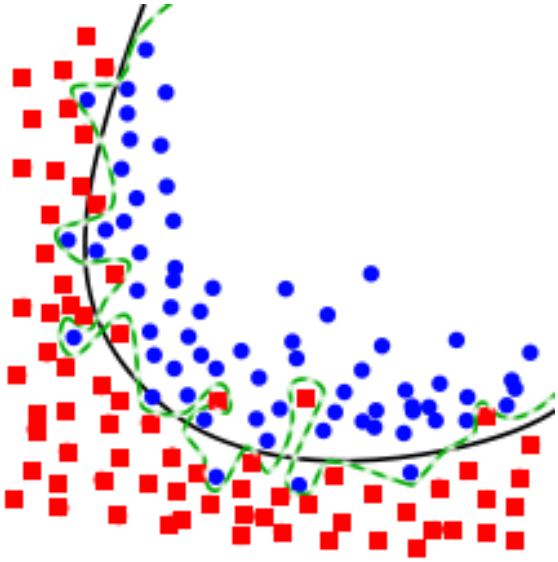


Figura 1.1: Efecto del sobre-ajuste.

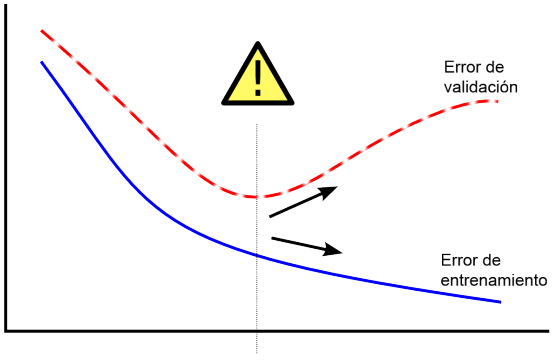


Figura 1.2: Zona de sobre-ajuste: Reducción del error en el conjunto de entrenamiento (azul) frente a la reducción del error en el conjunto de validación (rojo)

En el caso de los algoritmos constructivos este problema es crítico, ya que normalmente estos algoritmos aprenden el conjunto de aprendizaje hasta obtener error cero. En estas situaciones, las arquitecturas generadas contienen un gran número de

neuronas como resultado de un ajuste excesivo a los datos del problema. Denominaremos ejemplos ruidosos a todos aquellos patrones que obligan a la red a generar neuronas especificas con objeto de poder ser clasificados correctamente por la red. Estos patrones complican enormemente el ajuste del modelo y son los causantes de que la red se especialice en exceso para poder clasificarlos correctamente. En este sentido, la posible eliminación de patrones ruidosos del conjunto de entrenamiento podría conducir a la generación de una arquitectura con una cantidad significativamente menor de neuronas. Según la teoría de la navaja de Occam [**?**], se debería obtener mejores aciertos de generalización con arquitecturas más compactas, es decir, con menos neuronas en la capa oculta. Uno de los objetivos de este algoritmo es identificar estos ejemplos ruidosos en la fase de entrenamiento para poder así eliminarlos del conjunto de datos al mismo tiempo que está generando la arquitectura de la red. En la sec. 1.8 se analizará con mayor detalle como identificar estos ejemplos, mientras que en la sec. 1.10 se mostrará la aplicación exitosa del algoritmo C-Mantec a conjuntos de datos con entrada continua, problemas donde en general la información que contiene no es separable y son de naturaleza ruidosa.

La organización del presente capítulo es la siguiente: En la sec. 1.2 se presenta el perceptrón termal utilizado por este algoritmo; en la sec. 1.3 se explica con detalle la función de mayoría en la que se sustenta la arquitectura de la red. La red C-Mantec se propone en la sec. 1.4, mientras que en la sec. 1.5 se explica en detalle el algoritmo de entrenamiento y generación de la arquitectura de la red. A continuación, en la sec. 1.6 se estudia el efecto que tiene la competición a la hora de generar arquitecturas compactas con una alta capacidad de generalización. La convergencia del algoritmo se analiza en la sec. 1.7, y en la sec. 1.8 se estudia con mayor detalle la identificación y eliminación de los ejemplos ruidosos del conjunto de entrenamiento. La sec. 1.9 presenta una extensión a conjuntos de datos con más de dos clases a clasificar. En la sec. 1.10 se mostrarán los resultados obtenidos por C-Mantec con distintos conjuntos de datos y finalmente en la sec. 1.11 se comentaran las conclusiones y trabajos futuros.

1.2. Unidades de proceso

1.2.1. Perceptrón Simple

En 1958 el psicólogo Frank Ronsenblant desarrolló un modelo simple de neurona, basado en el modelo de McCulloch y Pitts, que utilizaba una regla de aprendizaje basada en la corrección del error, el perceptrón [**?**]. El perceptrón es un modelo matemático muy simple de una neurona biológica que puede ser visto como una neurona artificial, o una unidad básica de inferencia en forma de discriminador lineal que se usa como neurona dentro de otro tipo de redes neuronales artificiales más complejas. En la Fig. 1.3 se puede observar el esquema básico de un perceptrón,

donde **x** es el vector de entrada y **w** se conoce como el vector de pesos sinápticos que ponderan la entrada.

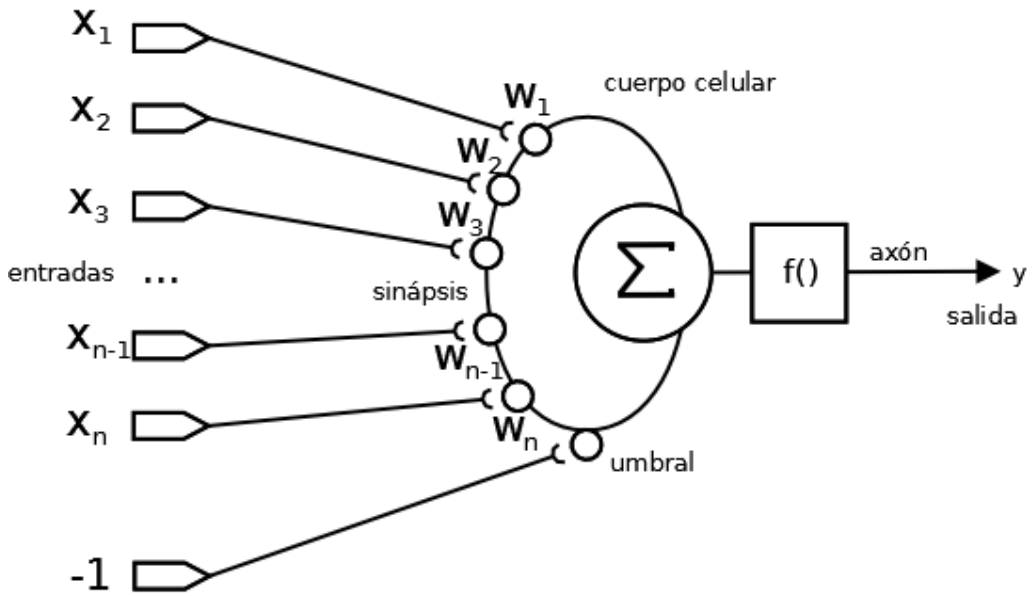


Figura 1.3: Esquema básico de un Perceptrón.

Llamaremos potencial sináptico $h(\mathbf{x})$ a la suma ponderada de todas las entradas por sus pesos sinápticos correspondientes:

$$h(\mathbf{x}) = \sum_{i=1}^{N} w_i x_i - \theta \tag{1.1}$$

donde θ es el valor umbral o valor de activación. El valor umbral es un peso sináptico más, pero con la diferencia de que posee una entrada fija con valor -1 y se usa para trasladar el punto de equilibrio del perceptrón a un punto distinto del origen. La salida del perceptrón (ω) podrá ser un valor discreto o continuo en base a la función de activación (S).

$$y = f\left(h\left(\mathbf{x}\right)\right) \tag{1.2}$$

Haykin [**?**] define la función de activación como una función que limita la amplitud de la salida neuronal, y define el teorema universal de aproximación de funciones dando las condiciones necesarias para determinar si una función $f(t)$ es una función de activación. Las funciones sigmoidea, tangente hiperbólica, saturación, signo o

escalón, que se muestran a continuación, cumplen las condiciones del teorema y suelen ser utilizadas como funciones de activación.

$$escalón(h) = \begin{cases} 1 & si\, h \geq 0 \\ 0 & en\, otro\, caso \end{cases} \tag{1.3}$$

$$signo(h) = \begin{cases} 1 & si\, h \geq 0 \\ -1 & en\, otro\, caso \end{cases} \tag{1.4}$$

$$saturación(h) = \begin{cases} c & si\, h \geq 1 \\ h & si\, 1 > h > -1 \\ -c & si\, h \leq -1 \end{cases} \tag{1.5}$$

donde el factor de ampliación c es generalmente tomado como 1, aunque su valor puede variar según sea necesario.

$$sigmoidea(h) = \frac{1}{1 + exp\,(-a\, h)} \tag{1.6}$$

donde a es un parámetro que define la pendiente de la función sigmoidea.

$$tanH(h) = \frac{1 - exp\,(a\, h)}{1 + exp\,(-a\, h)} \tag{1.7}$$

El aprendizaje es el proceso por el cual el perceptrón modifica sus pesos sinápticos en respuesta a una información de entrada. El proceso es adaptativo, es decir, no existe una forma de determinar los pesos de antemano, por lo que necesita un proceso iterativo. La regla de aprendizaje del perceptrón es supervisada por corrección del error, la cual por medio de un agente experto que determina la respuesta que debería generar el perceptrón a partir de una determinada entrada ajusta los pesos en función del error cometido en la salida tal y como muestra la ecuación 1.8.

$$\triangle w_i = \varepsilon(t - y)x_i \tag{1.8}$$

donde t es la respuesta deseada para la entrada $\mathbf{x}$, ω es la salida del perceptrón, x_i representa el valor de la entrada i-esima que se encuentra conectada al peso sináptico que se desea modificar (w_i), y ε es una tasa de aprendizaje comprendida en el intervalo $[0, 1]$ que indica como de rápido queremos que se modifiquen los pesos sinápticos en cada paso del aprendizaje.

1.2.2. Perceptrón Termal

El perceptrón termal fue introducido por Frean en 1992 [**?**] como una modificación del algoritmo del perceptrón simple de manera que puede obtener una aproximación estable ante problemas NLS. Consideremos un perceptrón modelado con una función de activación signo (1.4) con dos estados posibles de respuesta: $ON = TRUE = 1$ y $OFF = FALSO = -1$. La entrada del perceptrón será un conjunto de N señales de entrada en un rango de valores continuos, mientras que la salida se computa mediante la siguiente ecuación:

$$y = f(\mathbf{x}) = \begin{cases} ON & si\, h(\mathbf{x}) \geq 0 \\ OFF & en\, otro\, caso \end{cases} \tag{1.9}$$

donde $h(\mathbf{x})$ es el potencial sináptico del perceptrón que ha sido reflejado en la ecuación 1.1.

La regla de aprendizaje del perceptrón simple sólo converge cuando el problema es linealmente separable, siendo totalmente inestable en caso contrario. Cuando un problema es NLS, el perceptrón simple no converge porque intenta aprender todos los ejemplos que clasifica incorrectamente, desaprendiendo constantemente todo lo aprendido hasta ese momento, lo que provoca que nunca se estabilice con un mínimo aprendido. El perceptrón termal, por el contrario, es capaz de aprender sólo aquellos ejemplos que no sean difíciles de aprender por él y rechazar aquellos ejemplos que implicaran modificar sus pesos sinápticos en exceso.

La modificación de los pesos sinápticos en un perceptrón termal, ∇w_i, se realiza de la misma manera que en el perceptrón simple, por corrección del error, tal que cada vez que se le presenta un patrón de entrada que se desea clasificar los pesos sinápticos se modifican de acuerdo a la siguiente ecuación:

$$\triangle w_i = (t - S(\mathbf{x}))x_i T_{fact}(\mathbf{x}) \tag{1.10}$$

$$T_{fac}(\mathbf{x}) = \frac{T}{T_o}\, exp\left(\frac{-|h(\mathbf{x})|}{T}\right) \tag{1.11}$$

$$T = max\,(T_0 - it\, b, 0) \tag{1.12}$$

$$b = \frac{T_0}{itmax} \tag{1.13}$$

donde la única diferencia con el perceptrón simple viene dada por la incorporación de una tasa de aprendizaje dinámica $T_{fac}(\mathbf{x})$ que se encuentra descrita en la 1.11. Esta tasa de aprendizaje depende del valor del potencial sináptico (1.1), por lo que varía de forma dinámica en función de la entrada $\mathbf{x}$ con la que es estimulado el perceptrón. El objetivo es que esta tasa indique como de cerca está el perceptrón de cambiar de estado de activo a inactivo o viceversa. De esta forma, $T_{fac}(\mathbf{x})$ estará próxima a uno si el perceptrón está muy cerca de poder cambiar su salida $y = f(\mathbf{x})$, y se irá moviendo progresivamente a cero cuanto más alejado se encuentre de esta posibilidad.

Por otro lado, $T_{fac}(\mathbf{x})$ depende también de la temperatura T, variable introducida artificialmente y que decrece linealmente en cada iteración desde su temperatura inicial (T_0) hasta cero según la ecuación 1.12. El efecto de la Temperatura es el de realizar un procedimiento de recocido simulado [**?**] que ayuda a que la tasa de aprendizaje vaya disminuyendo con el paso del tiempo, produciendo sólo grandes cambios en las primeras etapas del entrenamiento, y cambios menores al final de éste.

En la ecuaciones anteriores, it es la iteración actual del perceptrón y se incrementa en uno cada vez que éste modifica sus pesos sinápticos. El perceptrón termal, por lo tanto, tiene sólo dos parámetros que se han de definir antes del entrenamiento: el número de iteraciones máximas ($ItMax$), y la temperatura inicial (T_0). El primero define la velocidad con que la temperatura decrece en cada iteración, y el segundo define la importancia del error del potencial sináptico a la hora de decrementar la tasa de aprendizaje, ya que la Temperatura del perceptrón (T) comienza con este valor, y divide al potencial dentro de la exponencial de $T_{fac}(\mathbf{x})$. C-Mantec definirá siempre $T_0 = N$, siendo N el número de entradas de cada uno de los patrones, por lo que C-Mantec tendrá que definir un valor $ItMax$ de cada uno de los perceptrones termales antes de comenzar el entrenamiento.

1.3. La función lógica de mayoría

La función de mayoría es conocida también como el operador de la mediana [**?**], y es una función lógica que tiene como salida la mayoría del vector entrada, es decir, la función se activará si la mayoría de las entradas están activas, permaneciendo desactivada en caso contrario. La ecuación 1.14 refleja claramente este comportamiento:

$$M(\mathbf{x}) = \begin{cases} ON = 1 & si\, h_{may}(\mathbf{x}) \geq 0 \\ OFF = -1 & si\, h_{may}(\mathbf{x}) < 0 \end{cases} \tag{1.14}$$

donde $h_{may}(\mathbf{x})$ es el operador suma del vector de entrada $\mathbf{x}$.

$$h_{may}(\mathbf{x}) = \sum_{i=1}^{N} x_i \, donde \, x_i \in \{1, -1\} \tag{1.15}$$

La función mayoría es una función linealmente separable, y se puede implementar con una neurona poniendo todos sus pesos sinápticos a uno y su valor umbral a cero:

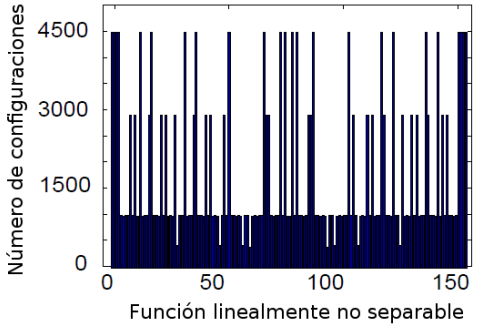
$$h_{may}(\mathbf{x}) \iff h(\mathbf{x}) = \sum_{i=1}^{N} w_i x_i + b \quad donde \quad w_i = 1, b = 0 \tag{1.16}$$

Se ha analizado la capacidad computacional de redes neuronales *feedf*orward con entrada binaria [**?**], en concreto, haciendo uso del algoritmo desarrollado por el autor de este trabajo de tesis en [**?**], se hizo un estudio exhaustivo en el que se analizó la función de salida más optima para poder resolver cada una de las 152 funciones booleanas no linealmente separables de tres entradas. Para ello se usó una red neuronal con una arquitectura de tres neuronas umbrales en la capa oculta y una única neurona umbral en la capa de salida, ya que con una arquitectura de tres neuronas en la capa oculta se pueden generar todas las funciones no linealmente separables de tres entradas (una neurona umbral de tres entradas puede implementar una función linealmente separable de tres entradas). Existen 154 funciones linealmente separables de tres entradas, por lo que para esta arquitectura de red existen $K = 154^3 = 3{,}652{,}264$ configuraciones de red posibles de la capa oculta para cada una de las 154 configuraciones de neuronas umbrales de salida, y cada una de esas configuraciones produce una función booleana distinta.

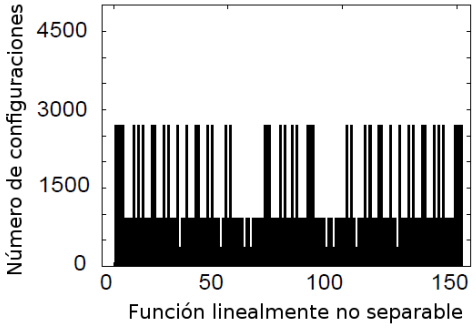
Se realizó un análisis de las distribuciones de configuraciones de red generadas por una función de salida para una función objetivo no linealmente separable. La Fig. 1.4 muestra las distribuciones de la mejor función de salida encontrada (que puede ser distinta en cada caso), de la función de mayoría y de la función de paridad.

Uno de los resultados obtenidos en el análisis fue que existe una fuerte correlación entre el número de configuraciones de funciones LS en la capa oculta y el grado de similitud del porcentaje de ejemplos activos de la función objetivo y de la función implementada por la neurona de salida de la red, es decir, las funciones de salida con un porcentaje de ejemplos con salida activa similar a la de la función objetivo, generan una gran cantidad de configuraciones de funciones LS en la capa oculta capaces de encontrar una solución al problema. El problema es que esas mismas funciones (implementadas en la neurona de salida) tienen muy pocas o ninguna posibilidad de encontrar una función objetivo con un porcentaje de ejemplos con salida activa opuesto a su porcentaje de ejemplos con salida activa.

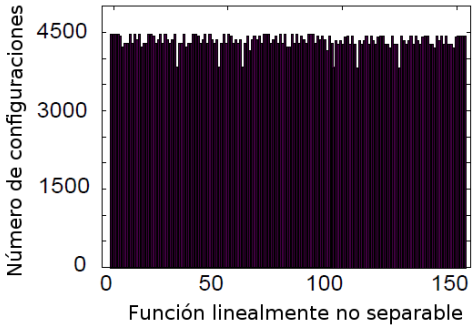
Por otro lado, se ha encontrado que la función mayoría, función que es balanceada y considerada como una de la funciones más complejas que pertenecen al conjunto de



(a) Mejor función encontrada



(b) Función mayoría



(c) Función paridad

Figura 1.4: Histogramas de configuraciones de red para cada una de las funciones linealmente no separables de tres entradas para la mejor función de salida encontrada, la función de salida Mayoría y la función de paridad.

funciones linealmente separables, genera buenos resultados para todas las funciones no linealmente separables de tres entradas, resultado que se puede observar en la figura 1.4b. Estos resultados condujeron a la posible implementación de un nuevo algoritmo, que basándose en una neurona fija en la capa de salida implementando la función mayoría (1.16), generase una capa oculta adecuada al problema usando un enfoque constructivo. Este análisis constituyó el punto de partida para el posterior diseño del algoritmo C-Mantec.

1.4. Arquitectura de la red para el algoritmo C-Mantec

En esta sección se introduce el algoritmo C-Mantec (*Competitive MAjority Network Training Error Correcting*). C-Mantec genera redes neuronales de perceptrones multicapas, conocidos en la literatura como MLP (*Multi Layer Perceptron*) [**?**]. Un MLP es una modificación del perceptrón lineal estándar, modelo muy conocido de red neuronal artificial (RNA) cuya arquitectura de múltiples capas le permite resolver problemas que no son linealmente separables (principal limitación del perceptrón lineal o perceptrón simple). La arquitectura se divide en 3 niveles:

Capa de entrada: Constituída por aquellas neuronas que introducen los patrones de entrada en la red. En estas neuronas no se produce procesamiento alguno.

Capas ocultas: Formada por aquellas neuronas cuyas entradas provienen de capas anteriores y cuyas salidas pasan a neuronas de capas posteriores. Cada capa transforma el conjunto de datos de entrada de ésta en un conjunto de salida apropiado para la siguiente capa.

Capa de salida: Neuronas cuyas entradas provienen de la última capa oculta y sus salidas se corresponden con las salidas de la red.

Aunque un MLP clásico puede tener varias capas ocultas, C-Mantec sólo genera redes con una única capa oculta, en la que cada uno de sus nodos esta formado por una neurona que implementa la función de activación signo (1.4), aunque en versiones posteriores de este algoritmo se han implementado redes C-Mantec con salida continua usando la función tangente hiperbolica (1.7) y una única neurona en la capa de salida que implementa la función mayoría (sec. 1.3). Así, la salida de una red neuronal C-Mantec con N neuronas en la capa oculta queda definida mediante la siguiente ecuación:

$$CMantec(\mathbf{x}) = sign\left(\sum_{n=1}^{N} sign\left(h\left(\mathbf{x}\right)\right)\right) \tag{1.17}$$

1.5. Algoritmo de aprendizaje

El algoritmo de aprendizaje genera la arquitectura de la red al mismo tiempo que se ajustan los pesos sinápticos de las neuronas en el proceso de aprendizaje. Este proceso constructivo es incremental, en el cual la red aprenderá un conjunto de ejemplos de dimensión N empezando con una arquitectura de red de N neuronas en la capa de entrada, una neurona en la capa oculta, y una neurona en la capa de salida tal y como se muestra en la Fig. 1.6. La neurona de la capa de salida

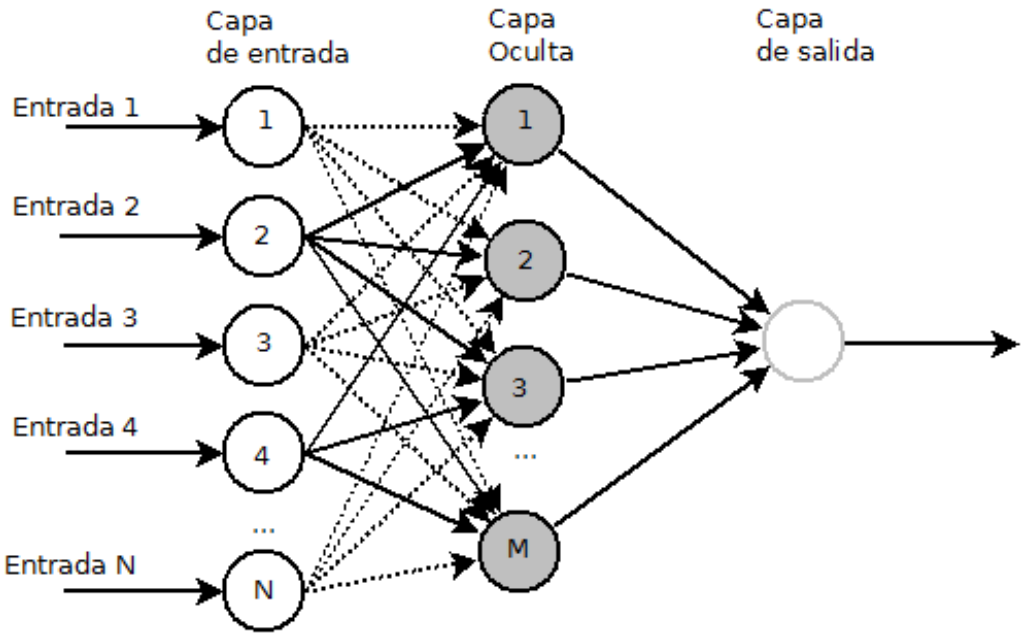


Figura 1.5: Arquitectura de una red C-Mantec con una única capa oculta y una única neurona de salida.

implementará siempre la función de mayoría (sec. 1.3), mientras que la arquitectura de la red irá creciendo añadiendo neuronas en la capa oculta conforme sea necesario durante el proceso de aprendizaje. Al final del aprendizaje la red neuronal tendrá una capa oculta con M neuronas (Fig. 1.7). Nótese que con esta configuración de arquitectura inicial, si el algoritmo termina el entrenamiento sin necesidad de añadir nuevas neuronas, la capa de salida puede ser eliminada quedando como capa de salida la capa oculta, formada en este caso únicamente por una neurona. Esta seria la configuración de una red lineal de un perceptrón simple.

C-Mantec necesita muy pocos parámetros de configuración, en un principio cuatro. Dos de ellos son propios del perceptrón termal (sec. 1.2): la temperatura inicial T_0, y el número de iteraciones máximas $itMax$, de cada uno de lo perceptrones. Considerando que definiremos siempre $T_0 = N$, sólo será necesario definir el parámetro $ItMax$ para cada perceptrón. El tercer parámetro es el factor de crecimiento G_{fact}, que definirá el nivel de precisión exigible a las neuronas a la hora de aprender un ejemplo. En sec. 1.8 se definirá el ultimo parámetro, Φ, que se usará sólo cuando queramos trabajar con conjuntos de datos no separables en los cuales es necesario eliminar ejemplos del conjunto de aprendizaje con el fin de evitar el sobre-ajuste.

Diremos que el algoritmo ha ejecutado un ciclo cada vez que ha recorrido todos los ejemplos del conjunto de aprendizaje. En cada ciclo el algoritmo irá seleccionando de forma aleatoria cada uno de los patrones del conjunto de aprendizaje hasta haberlos seleccionado todos, de manera que cuando un ejemplo no es clasificado correctamente la red intentará aprenderlo modificando los pesos sinápticos de alguno de sus

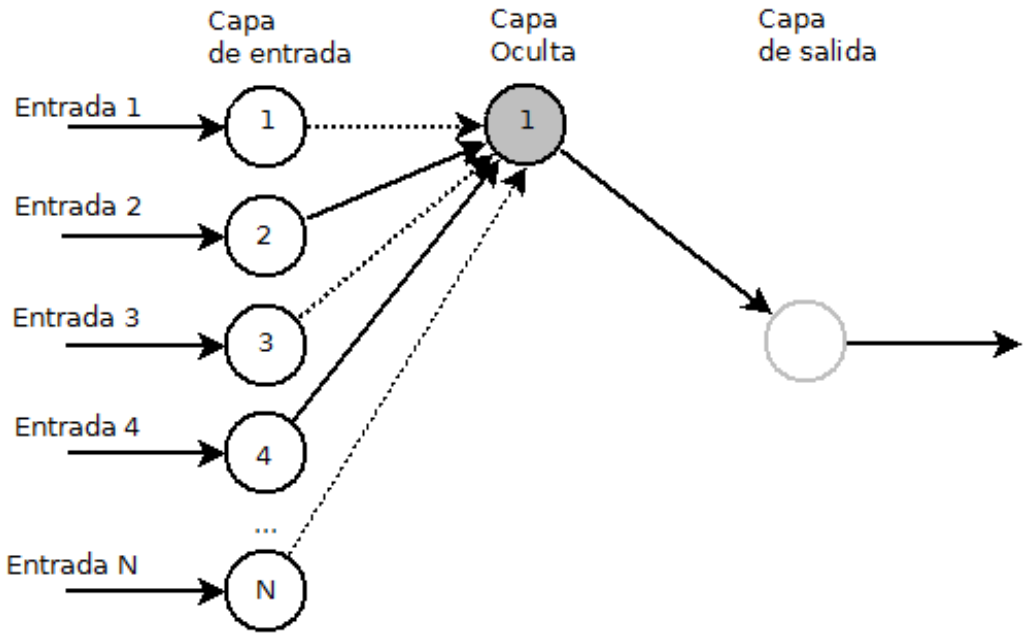


Figura 1.6: Red C-Mantec al comienzo del entrenamiento.

perceptrones de la capa oculta. Cuando se modifican los pesos de un perceptrón de la capa oculta no podemos asegurar que sigan bien clasificados los ejemplos que había clasificado correctamente la red con anterioridad. Definiremos un ciclo donde se ha producido alguna modificación de los pesos de alguna neurona como un ciclo con modificación de estado. El algoritmo de aprendizaje terminará cuando se ejecute un ciclo completo sin modificar el estado de la red.

Cuando un ejemplo del conjunto de aprendizaje no ha sido clasificado correctamente, la red necesita modificar su salida dado que en la capa oculta de la red existe un mayor número de neuronas que han clasificado mal el ejemplo respecto a las que sí lo clasificaron bien (Fig. 1.7). Se puede afirmar esto porque la única neurona de la capa de salida siempre computa la función de mayoría (sec. 1.3) y nunca será modificada en la etapa de aprendizaje. Para modificar la salida de la red se elegirá una neurona de las que ha clasificado mal el ejemplo con objeto de modificar su salida.

La elección de la neurona que debe modificar su salida no es una tarea trivial, ya que se ha de elegir aquella neurona más capacitada para aprender el ejemplo presentado minimizando el desaprendizaje de la red. Para ello, se elegirá la neurona que debe realizar una menor modificación de sus pesos sinápticos para cambiar su estado. En el caso de que estemos con la arquitectura inicial el problema es trivial, ya que sólo existe una neurona en la capa oculta, y por lo tanto es esta neurona la que se ha equivocado y la única que podría aprender el ejemplo mal clasificado. En cambio, cuando se tienen M neuronas en la capa oculta, si el ejemplo ha sido mal clasificado es porque existen $m > M/2$ neuronas que han clasificado el ejemplo erróneamente y hemos de elegir la neurona más capacitada. Usaremos la tasa de

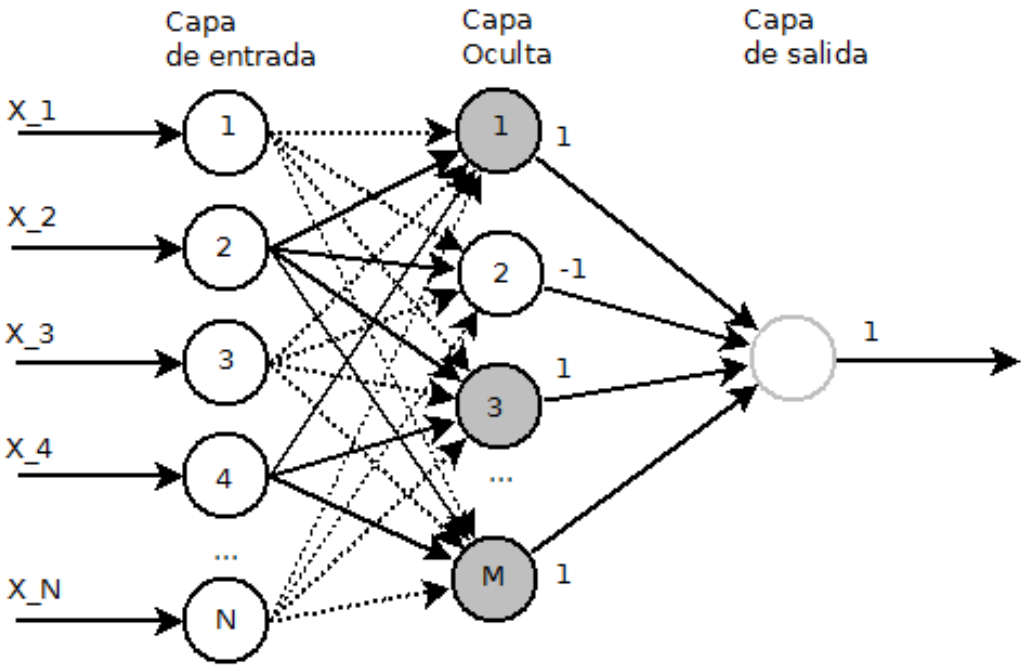


Figura 1.7: Red C-Mantec de M neuronas en la capa oculta siendo estimulada por un patrón $\mathbf{X}$. En la figura se puede observar como se activan m neuronas, mientras que $M - m$, se quedan desactivadas. Como $m > (M - m)$ la salida de la red estará activa.

aprendizaje dinámica proporcionada por el perceptrón termal, $T_{fac}(\mathbf{x})$ (ecuación 1.11), seleccionando como neurona candidata aquella neurona que haya clasificado mal el ejemplo con un valor de $T_{fact}(\mathbf{x})$ más alto, donde $\mathbf{x}$ es el vector de entrada de la neurona. Una tasa de aprendizaje alta (cercana a uno) nos indica que la neurona está cerca de aprender el ejemplo, por el contrario, un tasa baja (cercana a cero) indica que la neurona esta muy lejos de aprender el ejemplo.

Una neurona nueva se añadirá a la capa oculta cuando ninguna neurona esté dispuesta a aprender un ejemplo mal clasificado. Esto ocurre cuando la neurona candidata seleccionada para cambiar su estado se encuentra muy lejos de aprenderlo. Para poder identificar esta situación, definimos un nuevo parámetro al que llamaremos factor de crecimiento G_{fact}. Este parámetro define un valor umbral que si no es superado por la neurona candidata ($T_{fact} > G_{fact}$) permite añadir una nueva neurona a la capa oculta.

Desde un punto de vista conceptual, el efecto del parámetro G_{fact} en la fase de entrenamiento no es muy diferente del efecto producido por la tasa de aprendizaje dinámica T_{fact}, ya que ambos valores previenen el aprendizaje de ejemplos erróneos que se encuentran muy lejos del hiperplano generado por el perceptrón, pudiendo, en principio, evitarse el uso del parámetro G_{fact}, ya sea eliminándolo o igualándolo a cero. Ahora bien, desde un punto de vista computacional el efecto del parámetro G_{fact} es bastante importante, ya que controla en qué punto se añade una nueva

neurona en la capa oculta, afectando al tamaño final de la arquitectura generada. Este aspecto es muy importante, ya que como se describe en la sec. 1.6 se ha observado que en algunos casos la arquitectura con menos neuronas no es siempre la mejor arquitectura a la hora de obtener la mejor generalización sobre un conjunto de datos (en el sec. 1.6.1 se puede observar este efecto).

Algoritmo 1.1 Pseudocódigo del algoritmo de aprendizaje C-Mantec.

1. Creación de una arquitectura inicial con N neuronas en la capa de entrada, 1 neurona en la capa oculta y 1 neurona en la capa de salida que compute la función de mayoría.
2. Inicialización de los parámetros.

   $G_{fact} \in [0..,0{,}5]$
   $Itmax \in [1000..,100000]$
   $T_0 = N$ //N = Número de entradas de los ejemplos.

3. Introducir un patrón aleatorio y comprobar la salida de la red.
4. Si Salida de la Red <> Salida deseada.

   4.1. Obtener la neurona con el valor de T_{fact} máximo del subconjunto de neuronas de la capa oculta que han clasificado el patrón incorrectamente.
   4.2 Si el valor del T_{fact} de la neurona candidata no supera el valor umbral G_{fact} ($T_{fact} < G_{fact}$)

      4.2.1. Añadimos una neurona a la capa oculta, y la seleccionamos como neurona candidata.
      4.2.2 Reseteamos las temperaturas ($It_i = 0, \forall i = 1...M$)

   4.3 Modificamos los pesos sinapticos de la neurona candidata disminuyendo la temperatura de la neurona ($It_{candidata}$++).

5. Volver al paso 3 hasta que todos los patrones han sido clasificados correctamente.

Cada una de las neuronas de la capa oculta tiene definido un contador (it) que controla su temperatura actual de acuerdo con la ecuación 1.12. Cada vez que un perceptrón termal aprende un ejemplo decrece su temperatura al incrementar su it en uno. Consideraremos una iteración del algoritmo de aprendizaje como todas las modificaciones que realiza la red en los pesos sinápticos de sus neuronas en distintos ciclos hasta que decide incluir una nueva neurona, o termina la fase de entrenamiento al recorrer un ciclo sin modificar su estado. Cuando se inicia una nueva iteración

debido a que la red ha añadido una nueva neurona, el algoritmo inicializará las temperaturas de cada una de las neuronas de la capa oculta al valor T_0, igualando su contador *it* a cero.

El Algoritmo 1.1 muestra un pseudocódigo en el que se resumen los pasos más importantes de la etapa de aprendizaje para C-mantec, y la Fig. 1.8 muestra un diagrama de flujo propio del algoritmo, si bien, en la figura Fig. 1.8 se añade una fase de eliminación de ejemplos del conjunto de aprendizaje con el fin de evitar el sobre-entrenamiento que será descrita en detalle en la sección sec. 1.8.

1.6. El efecto de la competición

Con objeto de comprobar los efectos de haber introducido la competición entre las neuronas de la capa oculta en el algoritmo C-Mantec se ejecutaron un gran número de simulaciones midiendo el tamaño de las arquitecturas generadas y la generalización obtenida. Para poder analizar su influencia se implementó un algoritmo casi idéntico al algoritmo C-Mantec, al que se le quitó el mecanismo de la competición entre neuronas en la fase de aprendizaje, denominado "Mantec", el cual congela los pesos de todas las neuronas de la capa oculta cada vez que se añade una nueva neurona, permitiendo solamente aprender a la ultima neurona añadida. C-Mantec también fue comparado con otro algoritmo constructivo sin competición conocido como Upstart [**?**], eligiéndose este algoritmo ya que está implementado con perceptrones termales y su aplicación está ampliamente constatada en la literatura.

En un primer experimento se aplicaron los tres algoritmos (C-Mantec, Mantec y Upstart) a la síntesis de funciones booleanas aleatorias de diferentes tamaños de entrada (N) comprendidos en el intervalo $[5, 12]$, utilizando toda la función como conjunto de patrones de entrada, es decir, las redes fueron entrenadas con 2^N patrones. En la Fig. 1.9 se muestran los resultados obtenidos, donde se puede apreciar que el algoritmo C-Mantec genera redes mucho más compactas que los otros dos métodos cuando el número de entradas crece. Para $N = 12$, C-Mantec mejora claramente respecto a los otros dos algoritmos, necesitando aproximadamente unas 160 neuronas, lo que supone sólo una cuarta parte de las 600 que necesitaron Upstart y Mantec (nótese la escala logarítmica usada en el eje y de la Fig. 1.9).

En un segundo experimento se analizó la capacidad de generalización y el tamaño de las arquitecturas generadas utilizando como función objetivo la función booleana *two-or-moreclumbs*. Esta función ha sido usada en varios estudios [**?**, **?**, **?**, **?**, **?**], siendo la función tal que devuelve "1" para cualquier patrón que tenga al menos dos

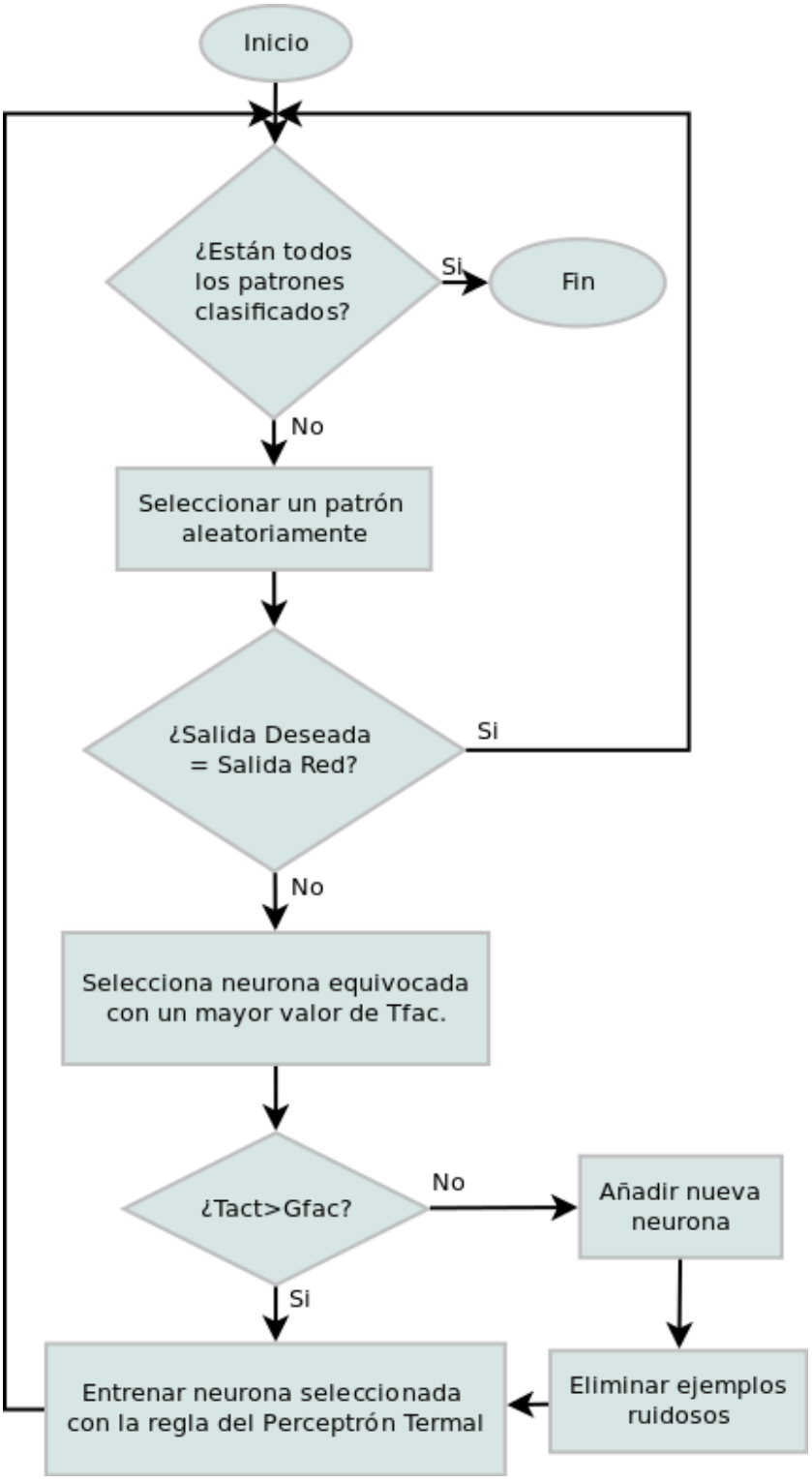


Figura 1.8: Diagrama de flujo del algoritmo de aprendizaje constructivo C-Mantec donde se introduce la competición entre las neuronas como principal característica.

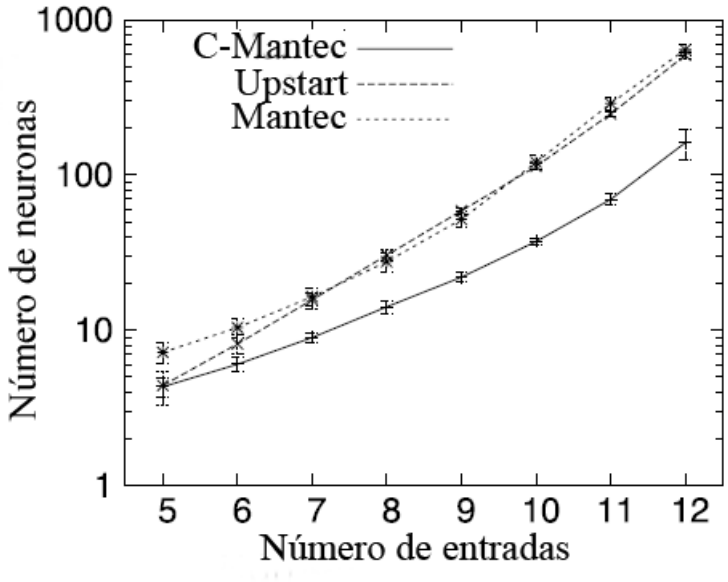


Figura 1.9: Número de neuronas generadas por cada algoritmo al aprender funciones booleanas aleatorias con un número de entradas comprendidas en el intervalo $[5, 12]$.

grupos de bits de entrada adyacentes activos. Así, por ejemplo, para una función *two-or-moreclumbs* de ocho entradas, la entrada 11011101 daría como resultado 1 ya que tiene dos grupos de 1's activos, la entrada 11010101 daría 0 ya que sólo existe un grupo en este caso, y la entrada 10011101 daría 1 ya que la función es cíclica y el primer bit junto con el último forman un grupo.

Esta función es una función linealmente no separable que puede implementarla fácilmente con N neuronas en la capa oculta cuando la función tiene un tamaño de N entradas. El proceso de entrenamiento se repitió 25 veces con patrones que fueron generados de una manera similar tal como aparece en Denker [**?**] y Frean [**?**]. La media y desviación estándar del número de neuronas generadas se muestra en la Fig. 1.10, mientras que el acierto obtenido en la generalización se muestra en la Fig. 1.11. La función usada en estas gráficas es la función *two-or-moreclumbs* de 25 entradas.

Los resultados muestran claramente que el algoritmo C-Mantec supera a los otros dos en la generación de una arquitectura más compacta (Fig. 1.10), mientras que en generalización se puede observar una mejora de aproximadamente un 0.5 % a los resultados obtenidos por Upstart y Mantec (Fig. 1.11). Cada punto de la gráfica fue

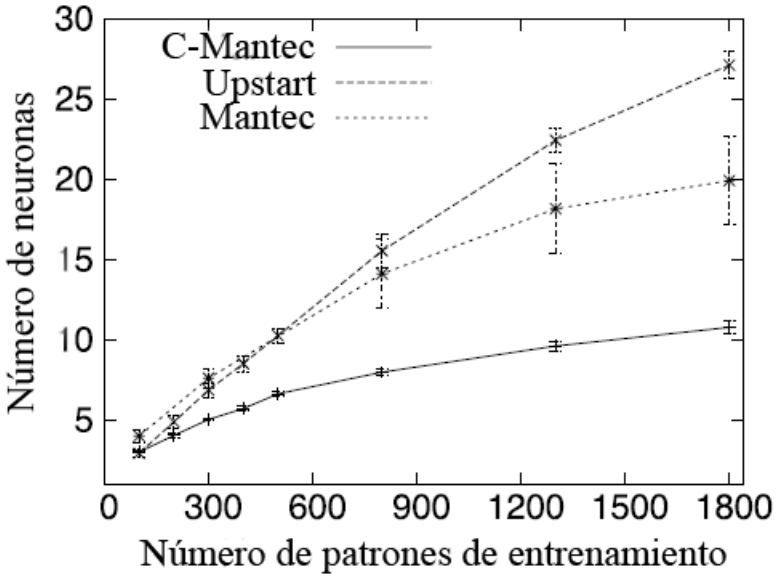


Figura 1.10: Número de neuronas generadas por cada algoritmo al aprender la función booleana *two-or-moreclumbs* de 25 entradas en función del número de patrones de entrenamiento que fueron usados para su aprendizaje.

obtenido promediando el resultado de 25 simulaciones en las se escogieron aleatoriamente k patrones como conjunto de entrenamiento, donde k viene reflejado en la gráfica en el eje de coordenadas con valores desde 100 hasta 1800.

1.6.1. El rol de los parámetros G_{fact} e $ItMax$ en el tamaño de las arquitecturas y la capacidad de generalización

Si no consideramos la eliminación de patrones del conjunto de entrenamiento, C-Mantec sólo tiene dos parámetros de entrada, G_{fact} e $ItMax$: el primero define un valor umbral, tal que si el valor del T_{fac} de la neurona candidata a aprender no lo supera, una nueva neurona será añadida a la capa oculta. Por lo tanto, valores muy bajos de G_{fact} generan arquitecturas muy pequeñas dado que las neuronas de la capa oculta no renuncian a aprender los ejemplos, mientras que valores muy altos (en torno a 0.5) generan arquitecturas con mayor número de neuronas, dado que éstas renuncian a aprender con mucha más frecuencia. El segundo parámetro ($ItMax$) indica como de rápido decrece la temperatura de cada uno de los perceptrones cada vez que aprende un ejemplo. Así, para valores muy altos de $ItMax$ las neuronas tardarán más tiempo en ver reducida su temperatura impidiendo que el algoritmo quede atrapado en mínimos locales. A continuación se analiza como varían los re-

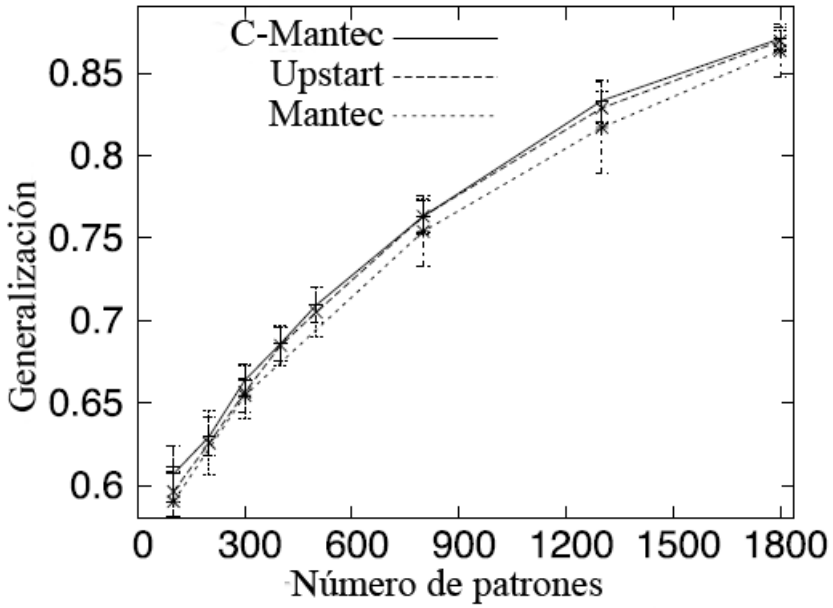


Figura 1.11: Acierto de generalización obtenido por cada algoritmo al aprender la función booleana *two-or-moreclumbs* de 25 entradas en función del número de patrones de entrenamiento que fueron usados para su aprendizaje.

sultados del algoritmo en capacidad de generalización y generación de arquitecturas compactas, modificando estos dos parámetros en distintas simulaciones con objeto de analizar también su robustez.

Con este propósito se realizan diferentes simulaciones usando como conjunto de entrenamiento un subconjunto de los ejemplos de las funciones booleanas *two-or-moreclumbs* ya descritas en la sección anterior, con diferentes tamaños de vectores de entrada 8, 10, y 12. Cada conjunto de entrenamiento es aprendido con diferentes configuraciones de parámetros G_{fact} y $ItMax$. Las distintas configuraciones de parámetros fueron: $G_{fact} = \{0{,}05,\ 0{,}1,\ 0{,}2,\ 0{,}3,\ 0{,}4,\ 0{,}5,\ 0{,}55,\ 0{,}65,\ 0{,}7,\ 0{,}75\}$ e $ItMax = 1000\times\{1,\ 3,\ 5,\ 10,\ 15,\ 20,\ 30,\ 50,\ 70,\ 100,\ 200\}$. Para cada configuración de parámetros diferentes se ejecutó el algoritmo C-Mantec 25 veces. Los conjuntos de entrenamiento se obtuvieron con un 75 % de los ejemplos de la función, mientras que el 25 % restante fue usado para medir la capacidad de generalización.

La Fig. 1.12 muestra el número de neuronas generadas por la red C-Mantec para la función *two-or-moreclumbs* de 12 entradas respecto al parámetro G_{fac}. Las tres curvas que se generaron se calcularon con la media, mínimo y mayor valor obtenido de todas las simulaciones con distintos valores de $ItMax$. Nótese que las tres curvas son muy parecidas, mostrándonos la robustez del algoritmo con respecto a cambios en los valores de $ItMax$. En cambio, se puede observar que el comportamiento

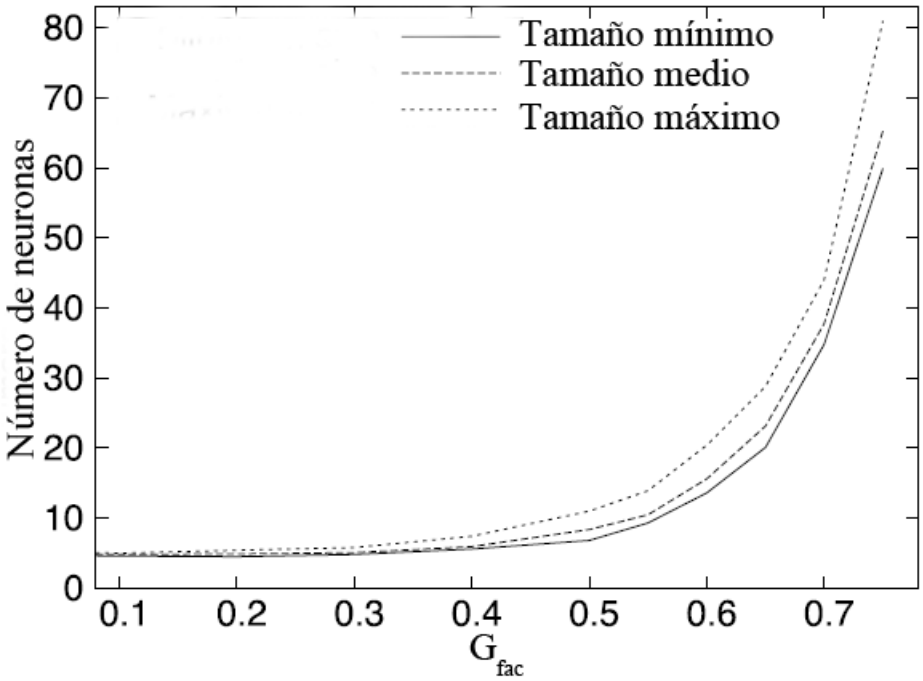


Figura 1.12: Número de neuronas generadas en función del valor G_{fac} para la función *two-or-moreclumbs* de 12 entradas.

del parámetro G_{fact} es el esperado, aumentando el número de neuronas en la capa oculta cuando se aumenta su valor. Se puede observar que para valores de G_{fact} inferiores a 0.65, C-Mantec sigue generando arquitecturas muy compactas con muy pocas neuronas en la capa oculta, creciendo muy rápidamente a partir de ese valor.

La Fig. 1.13 muestra el comportamiento de la capacidad de generalización del algoritmo en función del parámetro G_{fac}. Para este análisis se usó igualmente la función *two-or-moreclumbs* de 12 entradas, comprobándose que un aumento del valor G_{fac} genera un aumento en la capacidad de generalización con los mejores valores en el intervalo $[0{,}55 - 0{,}7]$. Para valores mayores de 0.7 se produjo una gran disminución en la capacidad de generalización, ya que como se mostró en la Fig. 1.12, la generación descontrolada de neuronas provoco problemas de sobre-entrenamiento. Nótese nuevamente las tres curvas son muy parecidas, confirmándonos la robustez respecto del párametro $ItMax$.

Con el fin de tener una idea más precisa de qué selección de parámetros hemos de realizar para maximizar el acierto de generalización o minimizar la arquitectura de la red, en la Fig. 1.14 se muestra la capacidad de generalización y la arquitectura mínima obtenida para una determinada configuración de valores G_{fac} e $ItMax$. Parece claro que el valor de G_{fac} es muy significativo en un sentido u otro. Si se quieren obtener arquitecturas con muy pocas neuronas en la capa oculta, el valor de G_{fac} ha de ser muy bajo, y mayor si lo que queremos obtener es una alta capacidad

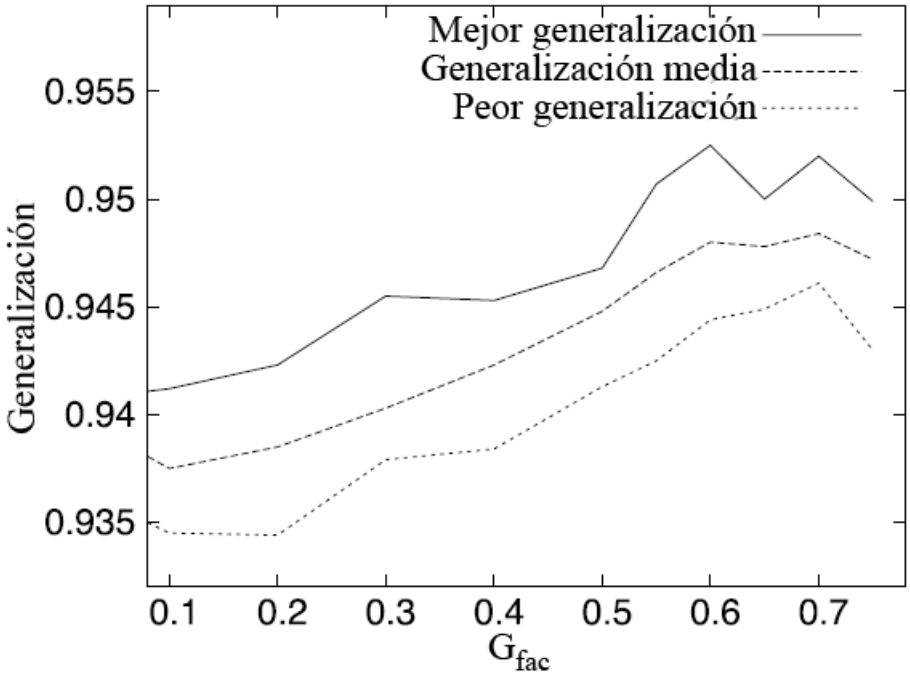


Figura 1.13: Capacidad de generalización obtenida en función del valor de G_{fac} para la función *two-or-moreclumbs* de 12 entradas.

de generalización. Es un resultado interesante que las pequeñas arquitecturas no coincidan con los mejores resultados en términos de capacidad de generalización, como sugiere el principio de la navaja de Occam [**?**], aspecto que será discutido en detalle en las conclusiones (sec. 1.11).

Los resultados para las dimensiones 8 y 10 fueron similares a las mostradas para el caso de N = 12. Es importante observar que el valor umbral de 0,7 para el parámetro G_{fac} obtenido con la función *two-or-moreclumbs* no se corresponde con el valor umbral obtenido por otros conjuntos de datos analizados con posterioridad, por lo que, como los resultados obtenidos con valores bajos G_{fac} son muy competitivos, se usaron valores de G_{fac} comprendidos en el intervalo $[0{,}05 - 0{,}1]$ en el resto de experimentos (los valores exactos se indican en los experimentos correspondientes).

1.7. Convergencia

No se ha podido demostrar matemáticamente la convergencia del proceso de aprendizaje. El uso la función de la mayoría como neurona de salida, la competencia entre las neuronas, la posibilidad de modificar los pesos sinápticos de cualquier neurona en todo el proceso de aprendizaje, junto con una regla de aprendizaje estocástica a nivel neuronal complica la aplicación de las técnicas estándar para probar la

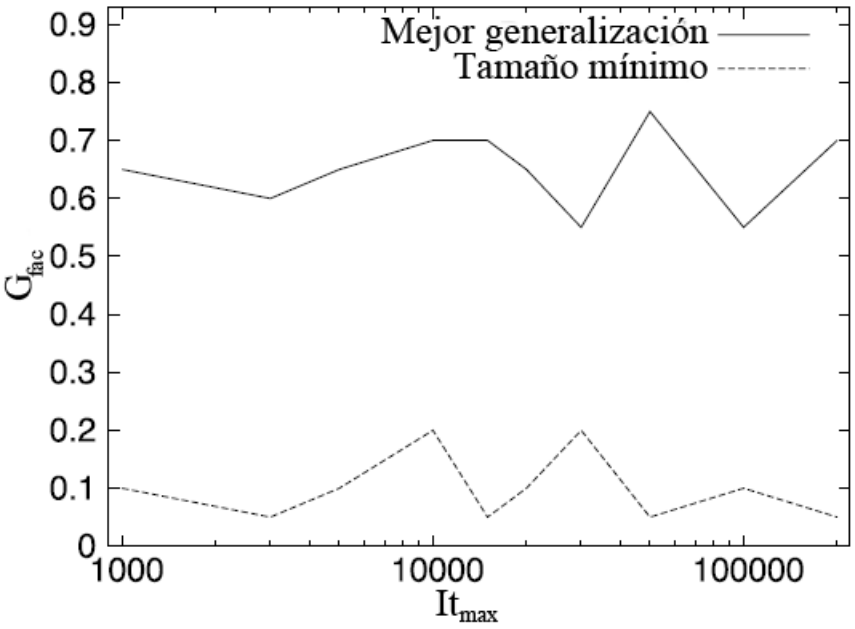


Figura 1.14: Mejor generalización y menor numero de neuronas en función de los valores G_{fac} e $Itmax$ para la función *two-or-moreclumbs* de 12 entradas.

convergencia usadas en los algoritmos constructivos en general. Sin embargo, una fuerte convergencia del proceso de aprendizaje se ha observado en todos los casos analizados y esto puede ser explicado por el uso de la regla del perceptrón termal, la cual es muy estable evitando desaprender lo aprendido hasta ese momento, junto con el hecho de que todo momento se pueden añadir nuevas neuronas para aprender aquellos ejemplos que no pueden ser aprendidos con las neuronas actuales.

En el anexo *Estudio preliminar de la convergencia del algoritmo C-Mantec* se realiza un ánalisis más profundo de las propiedades de convergencia del algoritmo.

1.8. Conjuntos de datos no separables o con ruido: evitar el sobre-ajuste

Durante las pruebas realizadas con el algoritmo C-Mantec en el sec. 1.10.2, se observó que cuando el algoritmo se aplica a un conjunto de funciones lógicas libres de ruido la capacidad de generalización obtenida es muy buena. Esto nos hace pensar que las arquitecturas compactas generadas por C-Mantec no sufren del problema de sobre-entrenamiento [**?**, **?**, **?**]. Sin embargo, como el algoritmo de aprendizaje termina sólo cuando todos los patrones del conjunto de aprendizaje son aprendidos (error cero), cuando el conjunto de aprendizaje no es separable o contiene ejemplos ruidosos, como

es el caso de conjuntos de datos reales, se produce el efecto de sobre-entrenamiento (sec. 1.10.3).

Con objeto de evitar este efecto de sobre-entrenamiento, se introdujo un procedimiento en el algoritmo de aprendizaje que trata de identificar y eliminar los ejemplos ruidosos o que impiden la separabilidad de las clases. En general, se conoce como aprendizaje activo a un esquema de aprendizaje en el que hay un cierto control sobre las entradas, que se ha demostrado puede conducir a una mejora de la capacidad de generalización [**?**, **?**, **?**, **?**].

El procedimiento para identificar y eliminar los ejemplos ruidosos o que impiden la separabilidad de las clases consiste en analizar el número de veces que un ejemplo ha sido presentado a la red y fue clasificado erróneamente, necesitando una modificación de los pesos sinaptico de alguna neurona de la capa oculta. Al final de un ciclo de aprendizaje los ejemplos que han necesitado un mayor número de modificaciones, en comparación con la media, serán eliminados al considerarse que son los ejemplos que están provocando el sobre-entrenamiento.

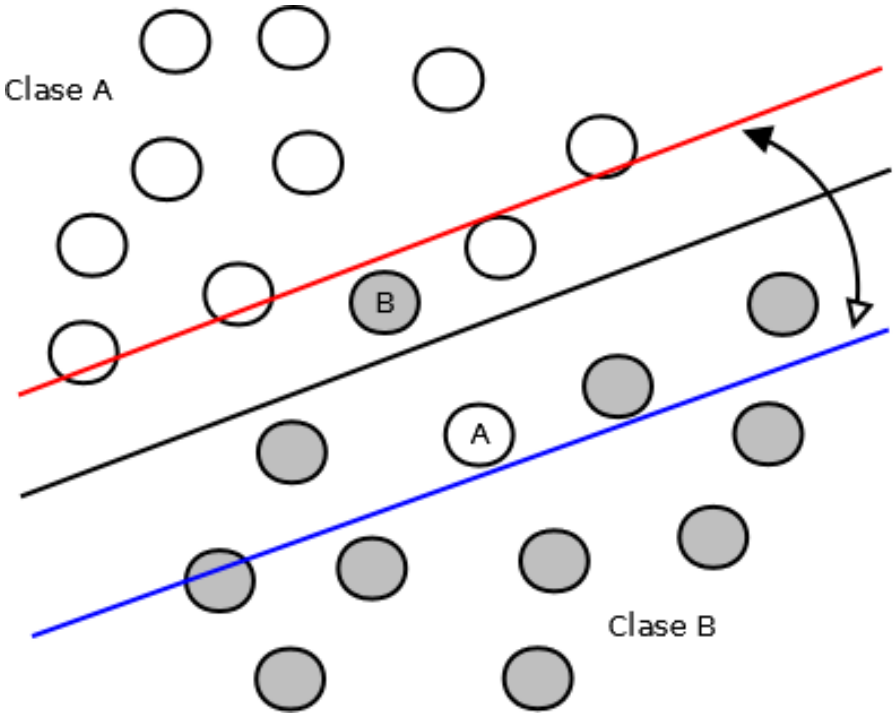


Figura 1.15: Dibujo esquemático del "efecto de resonancia". El perceptrón termal oscila intentando aprender los ejemplos contradictorios A y B (a la derecha de la figura). El número de veces que la red intenta aprender unejemplo se puede utilizar para detectar entradas contradictorias.

En la Fig. 1.15 se observa un caso típico en el que se necesita filtrar los ejemplos en conflicto. En el ejemplo se muestran dos patrones (A y B) con entrada similar pero con una clase de salida distinta, y totalmente en discordancia con el resto de patro-

nes. Si estos patrones no existieran el conjunto de datos sería fácilmente separable con una única neurona (representada en negro). En este caso una neurona entrenada con la regla de aprendizaje del perceptrón termal oscilará alrededor de los patrones A y B (líneas roja y azul). Denominamos a este comportamiento "efecto resonante". Consideraremos una iteración del algoritmo de aprendizaje como todas las modificaciones que realiza la red en los pesos sinápticos de sus neuronas hasta que decide incluir una nueva neurona o termina la fase de entrenamiento. Este comportamiento oscilatorio termina al final de una iteración del algoritmo de aprendizaje.

Sea P_a la probabilidad a priori de la clase A, P_b la probabilidad a priori de la clase B y un patrón $P = (\mathbf{X}, C)$ del conjunto de aprendizaje con un vector de entrada $\mathbf{X}$ y una clase de salida $C \in \{A, B\}$, definimos el ruido de un patrón P como el número de veces que ese patrón ha sido aprendido en una iteración del algoritmo de aprendizaje ponderado por su probabilidad a priori.

$$ruido(P) = NúmVecesAprendido_p \, P_c \tag{1.18}$$

Con lo cual, el procedimiento incrementa el contador $NúmVecesAprendido_p$ cada vez que un ejemplo P es aprendido.

Sea R la media del ruido de todos los patrones del conjunto de entrenamiento y $DesvEst(R)$ su desviación estándar, el proceso de eliminación de ruido consistirá en eliminar todos aquellos patrones que tengan un ruido superior a $R + \Phi \cdot DesvEst(R)$ tal y como indica la siguiente ecuación:

$$Condición\, Eliminación\,(P) = ruido(P) > R + \Phi\, DesvEst(R) \tag{1.19}$$

donde Φ es un parámetro adicional del algoritmo. El valor óptimo de Φ dependerá del nivel de ruido que presente el conjunto de datos de entrenamiento, siendo más alto cuando menos ruido tenga el conjunto de aprendizaje, aunque en todas las simulaciones realizadas, se obtuvieron buenos resultados en capacidad de generalización con valores de Φ en el intervalo [2,3].

1.9. Extensión a problemas multiclase

El algoritmo C-Mantec es un algoritmo de clasificación binario, por lo que para poder utilizarlo en la clasificación de conjuntos de datos con salida multiclase (1.20) se aplicaran tres esquemas muy conocidos [**?**, **?**]: Uno-contra-todos, Uno-contra-Uno y P-contra-Q . Los tres métodos obtienen un clasificador de K clases usando estrategias que combinan M clasificadores binarios y un modulo de decisión simple que tiene como entrada la salida de los clasificadores (Fig. 1.16). Los M clasificadores

son entrenados independientemente con diferentes conjuntos de entrada Ce_i que variarán en función de la estrategia elegida y del conjunto de entrada original Ce.

$$Ce = \{(\mathbf{X}, C_i) / \mathbf{X} = (x_1, ..., x_N),\ C_i \in \{C_1, ..., C_k\}\} \tag{1.20}$$

donde $(\mathbf{X}, C_i)$ es una tupla del conjunto de entrenamiento compuesta por un vector de entrada $\mathbf{X}$ de tamaño N y una clase de salida C_i perteneciente al conjunto de clases posibles $\{C_1, ..., C_k\}$. Usaremos C-Mantec como clasificador binario en este caso, detallándose a continuación cada uno de los esquemas usados.

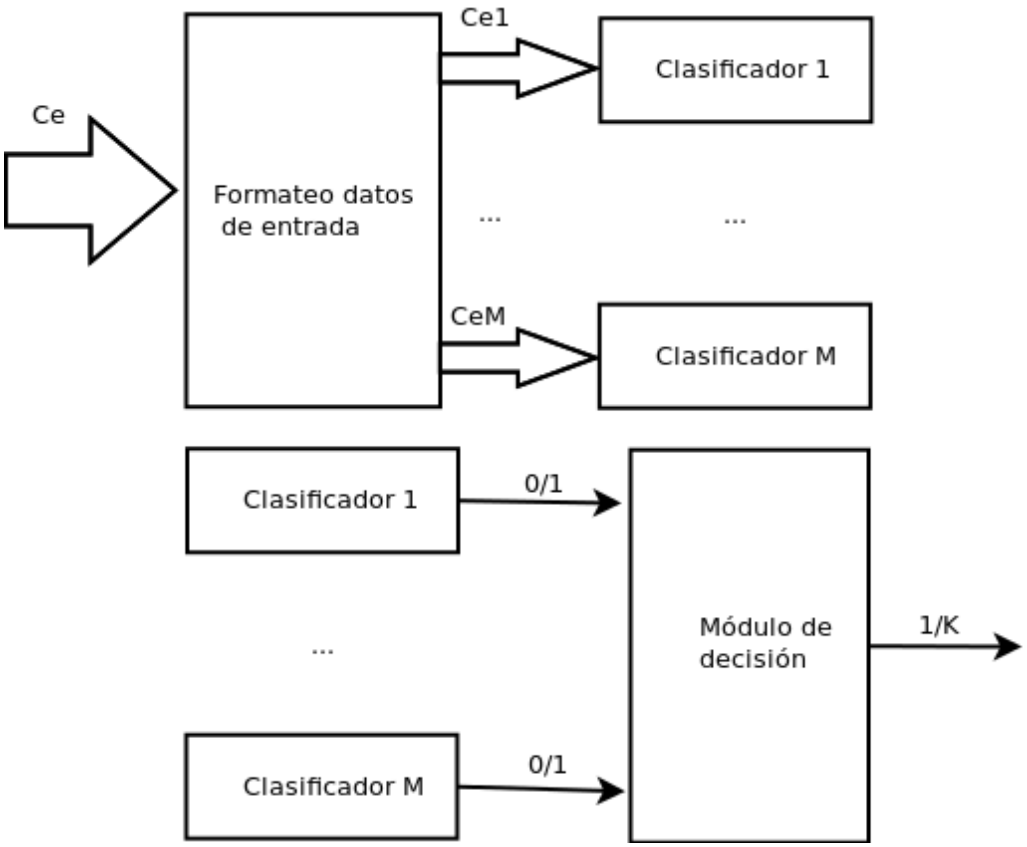


Figura 1.16: Esquema clasificador de K clases. En la figura se observa como un conjunto de entrenamiento multiclase Ce es transformado en diferentes conjuntos de entrenamiento binarios, Ce_i, dependiendo de la estrategia elegida (OAA, OAO, y PAQ). Estos conjuntos de entrenamientos binarios son clasificados por un clasificador binario independiente. Finalmente, estos clasificadores binarios, junto con un modulo de decisión simple, generará la arquitectura final necesaria para la clasificación de K clases.

1.9.1. Uno-Contra-Todos (OAA)

En este esquema, cada red C-Mantec tratará de clasificar cada clase con respecto a las demás. Para ello se utilizarán $M = K$ redes C-Mantec binarias, donde K es el número de clases del problema que se desea resolver. Cada red es entrenada con el mismo conjunto de datos de entrada pero con diferentes salidas tal y como indica la ecuación 1.21, de manera que, para cada entrada $\mathbf{X}$, la salida binaria será 1 para una red $CMantec_i$, $i \in \{1, 2, \cdots, M\}$, si en el conjunto de entrenamiento original la entrada $\mathbf{X}$ tiene como salida la clase C_i, y -1 en caso contrario.

$$Ce_i = \{(\mathbf{X}, S_i(\mathbf{X}, C_e)) / \mathbf{X} = (x_1, ..., x_N)\} \tag{1.21}$$

$$S_i(\mathbf{X}, C_e) = \begin{cases} 1 & si\ (\mathbf{X}, C_i) \in Ce \\ -1 & en\ otro\ caso \end{cases} \tag{1.22}$$

Con este esquema el módulo de decisión tendrá que decidir cuál será la clase de salida de acuerdo a la activación de cada una de las M redes (las cuales pueden generar 2^M entradas diferentes). Estas posibilidades pueden ser agrupadas en tres grupos diferentes de entradas: 1) Sólo una de las M redes se activa (caso deseado), ya que se está activando la red que indica a qué clase pertenece el patrón de entrada. 2) Más de una red C-Mantec se activa, caso en que el modulo de decisión ha de decidir cual será la clase de salida entre las redes que se han activado. 3) Ninguna se activa, caso en el que el modulo de decisión tiene que decidir cuál es la clase de salida de entre todas las clases posibles. Para resolver los casos de empates, utilizaremos las siguientes estrategias. En el segundo caso se elegirá la red activa con mejor porcentaje de clasificación, y en el tercer caso el modulo de decisión resuelve el empate escogiendo la red con peor porcentaje de clasificación. Los porcentajes de generalización se obtienen testando las redes con sus respectivos conjuntos de entrenamiento Ce_i una vez entrenadas.

1.9.2. Uno-Contra-Uno (OAO)

Este esquema transforma un problema de clasificación de K clases en $M = K(K-1)/2$ problemas binarios que serán resueltos por redes C-Mantec binarias. Cada red clasifica los patrones de una determinada clase contra los patrones de otra clase, y no tendrá en cuenta el resto de los patrones. Para ello se le asignará una salida binaria al conjunto de entrenamiento de cada red formado por el subconjunto de patrones de entrenamiento que contienen a estas dos clases (1 a los elementos de una clase, y un -1 a los de la otra).

$$Ce_{i,j} = \{(\mathbf{X}, S_{i,j}(\mathbf{X}, C_e)) / \mathbf{X} = (x_1, ..., x_N), (\mathbf{X}, C_i) \in C_e \vee (\mathbf{X}, C_j) \in C_e\} \tag{1.23}$$

$$S_{i,j}\left(\mathbf{X}, C_e\right) = \begin{cases} 1 & si\ \left(\mathbf{X}, C_i\right) \in Ce \\ -1 & si\ \left(\mathbf{X}, C_j\right) \in Ce \end{cases} \tag{1.24}$$

Dado un vector de entrada $\mathbf{X}$, el modulo de decisión tomará como entrada el resultado binario de las M redes C-Mantec generando un vector $\mathbf{V}$ de tamaño K en el cual cada V_i almacenara la suma de todos aquellos clasificadores $CMantec_{i,j}$ o $CMantec_{j,i}$ en los que la clase i esta involucrada y ha resultado ganadora. Con ayuda de este vector, el modulo de decisión decidirá la clase de salida en función de la clase mas votada

$$V_i = \sum_{j=1,\, j\neq i}^{K} C - Mantec_{i,j}(\mathbf{X}) \tag{1.25}$$

El problema en este caso es cómo el clasificador resuelve los empates, esto es, dos o más clases con igual número de votos. Un empate puede ser resuelto escogiendo la clase con mayor probabilidad a priori o, en caso de que no pudiera ser resuelto, escogiendo una aleatoriamente. La principal ventaja de este enfoque es que proporciona algo de redundancia que puede dar lugar a un sistema más robusto, mientras que la desventaja es que genera un gran número de clasificadores, especialmente cuando K es grande.

1.9.3. P-Contra-Q (PAQ)

El esquema P-Contra-Q puede ser visto como el punto medio entre los otros dos esquemas, ya que en este caso cada red $CMantec_i$, separará un conjunto clases P_i del resto $Q_i = K - P_i$, siendo K el número de clases del problema. Esta red separa dos grupos de clases entre ellas, pero no separa las clases que pertenecen al mismo grupo, por lo que serán necesarias más redes C-Mantec que nos permita discriminarlas. Por lo tanto, en este esquema serán necesarias tantas redes como hagan falta hasta conseguir discriminar todas las clases. El esquema OAA podría ser visto como un tipo de esquema PAQ donde cada P_i sólo contiene una clase y Q_i el resto de clases. El número de redes C-Mantec mínimo para poder clasificar K patrones es $M = \log\left(K\right)$ siguiendo una codificación binaria. Sin embargo, esta codificación mínima, aunque es muy eficiente en términos de redes a generar, no lo es tanto en términos de generalización ya que no genera ninguna redundancia, por lo que es beneficioso utilizar un número mayor de redes $CMantec_i$.

$$Ce_i = \left\{\left(\mathbf{X}, S_i\left(\mathbf{X}, C_e, Cod\right)\right) / \mathbf{X} = \left(x_1, ..., x_N\right)\right\} \tag{1.26}$$

$$S_i\left(\mathbf{X}, C_e\right) = \begin{cases} 1 & si\, C_i \in P_i \\ -1 & si\, C_i \in Q_i \end{cases} \tag{1.27}$$

Al igual que en el caso OAA, el modulo de decisión tendrá que decidir cual será la clase de salida a partir del resultado generado por cada una de las M redes C-Mantec. Cada una de las K clases espera un vector de salida de tamaño M asociado a esa clase a la que nos referiremos como vector prototipo de la clase k ($\boldsymbol{V}_k$). Si el vector resultante de la salida de las M redes, $\mathbf{V}$, no coincide con ninguno de los K vectores prototipo, se escogerá como salida aquella clase k que tenga un vector $\mathbf{V}_k$ más cercano a $\mathbf{V}$ en distancia Hamming, y en caso de empate, puede ser resuelto escogiendo la clase con mayor probabilidad a priori o, en caso de que no pudiera ser resuelto, escogiendo una aleatoriamente. En la sec. 1.10 hemos añadido redundancia a este método usando $K + log\left(K\right)$ redes, siguiendo las K primeras una codificación similar a la utilizada en el método OAA y una codificación binaria para las $log\left(K\right)$ siguientes.

Redes C-Mantec						Clase
Red_0	Red_1	Red_2	Red_3	Red_4	Red_5	
P_0	Q_1	Q_2	Q_3	Q_4	Q_5	C_0
Q_0	P_1	Q_2	Q_3	Q_4	P_5	C_1
Q_0	Q_1	P_2	Q_3	P_4	Q_5	C_2
Q_0	Q_1	Q_2	P_3	P_4	P_5	C_3

Tabla 1.1: Ejemplo del esquema P-contra-Q (PAQ) en el cual un problema 4 clases ha sido codificado por 6 redes binarias.

La Tab. 1.1 muestra un ejemplo de cómo es posible codificar un problema multiclase de $K = 4$ usando el esquema PAQ con $M = K + \log\left(K\right)$ redes C-Mantec, en las cuales las 4 primeras ($Red_0 ... Red_3$) utilizan una codificación de tipo OAA, mientras que las dos últimas (Red_4 y Red_5) utilizan una codificación de tipo binaria. Con esta codificación el conjunto de entrenamiento Ce_i de una Red_i debería corresponderse con la columna i de la tabla y la fila j de la clase que se desea codificar siguiendo la ecuación 1.26.

1.10. Aplicación del algoritmo C-Mantec a bases de datos de problemas de uso público

El rendimiento del algoritmo C-Mantec se ha analizado utilizando dos conjuntos diferentes de funciones: un conjunto de funciones lógicas libres de ruido utilizadas

con frecuencia en el diseño de circuitos y un conjunto de problemas que pertenecen al repositorio UCI[1], siendo estos datos del "mundo real" con ruido y clases no separables. El análisis se centró principalmente en estudiar el tamaño de las redes construidas y la capacidad de generalización obtenida.

1.10.1. Rendimiento del algoritmo en la obtención de arquitecturas compactas

En la sección sec. 1.6 se comprobó que el algoritmo C-Mantec generaba redes compactas con muy pocas neuronas en la capa oculta para la función lógica *two-or-moreclumbs*. Este estudio se realizó con el objetivo de testar la eficacia de la competición entre las neuronas en los algoritmos de aprendizaje. Seguidamente, analizaremos la capacidad de C-Mantec de producir redes compactas en el conjunto de prueba MCNC[2]. Este conjunto de prueba está formado por funciones lógicas de circuitos y nos será de gran utilidad dado que es el conjunto de validación de referencia utilizado en la literatura en la literatura [**?**, **?**]. El problema conocido como lógica umbral consiste en reducir el número de puertas umbrales necesarias para implementar un circuito, reduciendo así el área de integración necesaria para implementarlo. Este es un problema muy estudiado desde los años 60 [**?**, **?**, **?**, **?**, **?**] y de difícil solución. El conjunto de prueba contiene 14 funciones lógicas con varias salidas para una misma entrada, por lo que para usar nuestra red C-Mantec de una única salida, transformamos cada función lógica de N entradas y M salidas en M funciones lógicas de N entradas y una salida, considerándose finalmente 91 funciones lógicas diferentes con un número de entradas comprendido en el intervalo $[9 - 19]$.

Se comparó el rendimiento de C-Mantec con los resultados obtenidos por DASG* (**??**), ya que este algoritmo obtuvo un gran mejora respecto a otros algoritmos previamente publicados [**?**]. La Tab. 1.2 muestra los tamaños de las arquitecturas generadas por el algoritmo C-Mantec y DASG*, donde la última columna representa el porcentaje de mejora del algoritmo C-Mantec respecto al algoritmo DASG*. Asimismo, la última fila muestra los resultados promedios de cada columna, en la que puede observarse que el algoritmo C-Mantec supera los resultados del algoritmo DASG* obteniendo una reducción media del 39.08 % en canto al número de puertas umbrales.

1.10.2. Capacidad de generalización de funciones booleanas libres de ruido

Para analizar la capacidad de generalización del algoritmo C-Mantec en un entorno libre de ruido se usaron 17 funciones booleanas del conjunto de datos MCNC. El

[1] http://archive.ics.uci.edu/ml/
[2] http://cadlab.cs.ucla.edu/ %7Ekirill/blif-benchmarks.zip

Función	Número de entradas/salidas	Neuronas DASG	Neuronas C-Mantec	Reducción (%)
9symml	9/1	21	3	85.71
alu2	10/6	96	51	46.88
x2	10/7	17	12	29.41
cm152a	11/1	9	8	11.11
cm85a	11/3	19	5	73.68
cm151a	12/2	18	6	66.67
alu4	14/8	279	150	46.24
cm162a	14/5	15	15	0.00
cu	14/11	22	18	18.18
cm163a	16/5	21	13	38.10
cmb	16/4	71	4	94.37
pm1	16/13	17	15	11.76
tcon	17/16	24	24	0.00
pcle	19/9	32	24	25.00
Media	-	47.21	24.86	39.08

Tabla 1.2: Número de neuronas generadas en la capa oculta obtenidas por dos algoritmos constructivos DASG y C-Mantec (columnas 2 y 3, respectivamente) para un conjunto de 14 funciones booleanas. La primera columna indica el nombre de la función analizada y el número de entradas y salidas de la función se indica en la segunda columna. La última columna muestra en porcentajes la reducción en el número de neuronas obtenidas por el algoritmo C-Mantec en comparación con DASG.

análisis de la capacidad de generalización se realiza en comparación con otros tres algoritmos de clasificación muy conocidos: el algoritmo de árboles de decisión C4.5 [**?**], Redes Neuronales (FFNNs) y el algoritmo de K-vecinos más cercanos (K-NN-gen). Estos tres métodos fueron obtenidos usando la plataforma de código abierto WEKA [**?**], usando valores de configuración por defecto, mientras que se configuraron los siguientes valores para los parámetros del algoritmo C-Mantec: $G_{fact} = 0.05$ y $ItMax = 100000$. La estrategia de validación usada fue la validación cruzada usando diez cajas ("*10 fold cross validation*").

En la Tab. 1.3 se muestran los resultados obtenidos utilizando los 4 métodos mencionados, en donde se observa que C-Mantec supera en capacidad de generalización promedio (94.18 %) a FFNN (92.18 %), C4.5 (81.34 %), y K-NN-gen (80.5 %). La diferencia entre la capacidad de generalización de C-Mantec y los métodos de C4.5 y K-NN-gen fue estadísticamente significativo con unos valores de t-test de $p = 0.002$ y $p = 0.001$ respectivamente. La diferencia en generalización no resultó estadísticamente significativa ($p = 0.21$) entre el algoritmo C-Mantec y FFNN.

Función	Número de Entradas	Neuronas Generadas	Capacidad de generalización			
			C-Mantec	FFNN	C4.5	KNN-Gen
cm82af	5	3.00±0.00	**93.33±11.11**	90.00±31.62	60.83±38.10	65.00±35.53
cm82ag	5	3.00±0.00	60.00±37.27	**75.00±28.60**	35.50±11.95	34.17±23.06
cm82ah	5	1.00±0.00	**100.00±0.00**	**100.00±0.00**	57.50±22.38	65.83±23.06
z4ml24	7	3.00±0.00	98.33±3.67	**100.00±0.00**	78.21±6.84	81.28±6.37
z4ml25	7	3.13±0.74	90.83±12.34	**91.35±12.00**	56.47±17.42	46.15±13.45
z4ml26	7	3.00±0.00	**96.67±5.89**	92.76±13.68	78.78±18.12	80.19±17.11
z4ml27	7	3.00±0.00	99.17±2.78	**100.00±0.00**	83.72±21.89	78.27±26.75
9symml	9	3.00±0.00	**99.41±0.86**	96.67±5.47	76.76±4.56	75.02±6.24
alu2k	10	11.21±0.92	**97.36±1.9**	84.39±5.28	94.43±4.21	90.25±7.73
alu2l	10	18.90±1.45	79.22±5.54	75.79±5.18	**82.33±3.55**	78.03±4.88
alu2o	10	11.16±0.91	**90.24±2.27**	90.04±2.54	89.46±3.58	88.67±4.78
cm85al	11	1.00±0.00	99.95±0.16	**100.00±0.00**	98.19±0.66	98.73±0.62
cm85am	11	3.00±0.00	**99.76±0.42**	99.41±1.24	96.88±0.25	96.78±1.36
cm85an	11	1.00±0.00	**100.00±0.00**	**100.00±0.00**	98.34±0.70	99.07±0.81
alu4q	14	45.80±3.62	**99.72±0.14**	86.72±2.06	98.80±0.64	99.07±0.21
alu4r	14	75.70±0.98	**97.73±0.47**	88.93±1.39	96.45±0.73	94.90±1.18
alu4u	14	27.40±0.99	**99.39±0.27**	95.94±0.37	98.16±0.24	97.110.35
Media	9.2	12.66±20.20	**94.18±10.33**	92.18±8.17	81.34±18.44	80.5±18.97

Tabla 1.3: Capacidad de generalización obtenida con el algoritmo C-Mantec y otros tres algoritmos estándares, C4.5, redes neuronales (FFNNs) y K-vecinos más cercanos para la generalización en un conjunto de 17 funciones booleanas. (Ver texto para más detalles.)

1.10.3. Aprendiendo problemas del mundo real

Se han obtenido 17 conjuntos de datos del repositorio de aprendizaje computacional UCI[3] [**?**] para poder probar el funcionamiento del algoritmo con conjuntos de datos reales. Este repositorio contiene tanto conjuntos de datos de clasificación binaria, como conjuntos con varias clases de salida, con un número de entradas que oscila entre 4 y 125, y un número de salidas entre 2 y 19.

El ajuste de parámetros del algoritmo C-Mantec para todos las simulaciones fue el siguiente: $G_{fact} = 0{,}1$, $Itmax = 100000$. Para el procedimiento de eliminación de ejemplos ruidosos se aplicó un valor $Phi = 2$, a fin de evitar así el sobre-entrenamiento. Para los problemas de varias salidas el esquema de extensión a multiclases elegido ha sido el enfoque P-contra-Q (PAQ), donde hemos introducido redundancia usando $K + log\,(K)$ redes, en las cuales las K primeras redes siguen una codificación similar a el esquema OAA y las $log\,(K)$ siguientes una codificación binaria.

[3]http://archive.ics.uci.edu/ml/

Función	Número de entradas	Número de clases	Neuronas C-Mantec	Generalización		
				C-Mantec	FFNN	SVM
Diab1	8	2	3.34±1.11	76.62±2.69	74.17±0.56	**76.80±4.54**
Cancer1	9	2	1.00±0.00	96.86±1.19	**97.07±0.18**	96.68±2.68
Ionosphere	34	2	2.00±0.00	87.44±0.06	**91.06±4.86**	88.07±5.32
Heart1	35	2	2.66±0.74	82.63±2.52	79.35±0.31	**83.86±6.21**
Heartc1	35	2	1.28±0.57	82.48±3.33	80.27±0.56	**84.8±5.75**
Kr-vs-Kp	40	2	3.00±0.00	98.96±0.62	**99.34±0.54**	95.79±1.34
Card1	51	2	1.78±0.87	85.16±2.48	**86.63±0.67**	84.80±5.75
Sonar	60	2	1.00±0.00	75.00±14.10	**81.68±8.66**	73.61±9.34
Mushroom	125	2	1.00±0.00	99.98±0.04	**100.00±0.0**	**100.00±0.0**
Iris	4	3	4.60±0.8	96.00±3.33	**96.93±4.07**	96.27±4.58
Bal-scale	4	3	12.40±0.04	90.53±3.74	**90.69±3.04**	87.57±2.49
Thyroid	21	3	5.00±0.00	94.16±0.51	**96.85±0.83**	93.79±0.31
Waveform	40	3	9.00±0.00	86.50±1.23	83.56±1.66	**86.68±1.97**
Horse1	58	3	9.40±0.93	67.79±5.71	66.79±5.49	**68.66±5.16**
Gene1	120	3	8.40±2.70	86.24±1.10	90.93±1.80	**90.96±1.28**
Glass	9	6	35.10±3.50	**67.00±6.38**	53.96±2.21	57.36±8.77
Soybean	82	19	24.00±0.00	92.96±2.68	90.53±0.52	**93.10±2.76**
Media				**86.25±10.14**	85.87±12.41	85.81±11.24

Tabla 1.4: Capacidad de generalización obtenida con los algoritmos C-Mantec, FFNN y SVM sobre un conjunto de datos reales obtenidos del repositorio de aprendizaje computacional UCI.

Los resultados se muestran en la Tab. 1.4 donde las primeras tres columnas muestran información del conjunto de datos a tratar: el nombre, el número de variables de entrada y el número de salidas, mientras que el número de neuronas generadas en la capa oculta se muestra en la cuarta columna, donde se sigue verificando que las arquitecturas generadas continúan resultando muy compactas. En las tres últimas columnas se muestran la capacidad de generalización del algoritmo C-Mantec junto con otros dos algoritmos de clasificación muy usados para resolver este tipo de problemas en la literatura: una red Neuronal *feedforward* (FFNN) y una máquina de soporte vectorial (SVM). Ambos algoritmos se ejecutan en el entorno WEKA usando la mejor selección de parámetros de configuración obtenida de diferentes combinaciones de éstos incluyendo sus valores por defecto. La estrategia de validación usada fue la validación cruzada con diez cajas, presentándose la media y la desviación estándar de los diez valores observados. En la última fila de la Tab. 1.4 se muestran los valores medios de capacidad de generalización obtenidos por todo el conjunto de datos, donde los valores de la desviación estándar se calculan a partir de los valores de la capacidad de generalización obtenidos para cada uno de los problemas.

C- Mantec obtiene los mejores resultados en promedio con una capacidad de generalización media de 86.24 %, seguido por FFNN (85.87 %) y SVM (85.81 %), aunque la diferencia media entre los tres métodos analizados no es estadísticamente significativa. El algoritmo C-Mantec superó en promedio a los otros dos métodos, aunque en la mayoría de los casos nunca resulto ser el mejor método, siendo normalmente la segunda mejor opción.

1.10.4. Enfoques multiclase del algoritmo C-Mantec

En el sec. 1.10.3 se aplica el esquema de extensión a multiclases P-contra-Q (PAQ) para clasificar los conjuntos de datos con más de dos clases de salida.

Función	Número de entradas	Número de clases	Num. de Neuronas		Generalización	
			OAA	OAO	OAA	OAO
Iris	4	3	6.60±1.51	12.00±0.00	95.33±4.50	96.00±7.16
Bal-scale	4	3	12.00±0.00	12.40±1.26	89.89±4.67	91.72±3.43
Thyroid	21	3	2.00±0.00	3.00±0.00	94.20±0.60	94.20±0.60
Waveform	40	3	14.90±0.32	13.90±1.28	85.22±1.74	83.24±1.35
Horse1	58	3	4.70±0.70	3.00±0.00	64.80±6.40	66.20±5.20
Gene1	120	3	5.20±3.00	3.50±1.10	84.00±1.10	88.50±1.00
Glass	9	6	11.40±2.00	17.80±1.70	58.40±7.80	64.80±6.30
Soybean	82	19	19.00±0.00	171.00±0.00	90.60±3.20	92.60±2.00
Media			9.48±5.80	29.58±57.42	82.81±13.76	84.66±12.45

Tabla 1.5: Número de neuronas generadas y capacidad de generalización obtenida por el algoritmo C-Mantec con los esquemas OAA y OAO para los conjuntos de datos con más de dos clases de salida

En la Tab. 1.5 se muestran los resultados obtenidos para estos mismos conjuntos de datos con los otros dos esquemas propuestos en sec. 1.9. Los conjuntos de datos que se utilizan en este estudio son los conjuntos de datos con más de dos clases usados en la Tab. 1.4, con objeto de analizar los resultados obtenidos en comparación con el esquema PAQ. En total son 8 conjuntos de datos con un número de clases de salida entre 3 y 19. La capacidad de generalización media y el número de neuronas en la capa oculta para estos conjuntos de datos con el enfoque PAQ es de 85.11 $\pm$ 11.48 y 13.49 $\pm$ 10.64 respectivamente mejorando la capacidad de generalización obtenida con los enfoques alternativos. Con respecto al tamaño de las arquitecturas generadas, las redes más pequeñas se obtienen utilizando el enfoque OAA tal y como se esperaba.

1.11. Conclusiones

En este capitulo se ha introducido el algoritmo de red neuronal constructivo C-Mantec para su aplicación en problemas de clasificación supervisada. C-Mantec incorpora el concepto biológico de competición neuronal en el proceso de aprendizaje. La principal característica del algoritmo es por lo tanto su aprendizaje competitivo, y su principal objetivo es el de no olvidar el conocimiento aprendido, modificando lo menos posible los pesos sinápticos durante la fase de aprendizaje. El algoritmo usa una regla local de aprendizaje estable del perceptrón termal para implementar la competencia de las neuronas en la capa oculta. Esta competencia global de todas las neuronas evita la congelación de los pesos sinápticos de la red, procedimiento estándar en la mayoría de los algoritmos constructivos existentes y marca una gran diferencia esta con respecto a otros algoritmos constructivos publicados en la literatura.

La regla de aprendizaje del perceptrón termal modifica la regla de aprendizaje del perceptrón simple, proporcionándole estabilidad ante un conjunto de datos no separable. El perceptrón simple es un modelo matemático muy simple de una neurona biológica que asegura su convergencia sólo cuando el conjunto de datos con el que se le entrena es linealmente separable siendo totalmente inestable en caso contrario. El perceptrón termal por el contrario, es capaz de aprender sólo aquellos ejemplos cercanos al hiperplano separador generado por él que no sean difíciles de aprender, y rechazar aquellos ejemplos que implicaran modificar sus pesos sinápticos en exceso.

Se ha realizado un análisis de los parámetros del algoritmo, demostrando que el mismo es muy robusto respecto a cambios en sus valores con lo que su ajuste resulta ser muy sencillo. Por otro lado, se ha desarrollado una técnica de filtrado de ejemplos ruidosos totalmente integrada en el algoritmo de aprendizaje, conocida como aprendizaje activo. Esta técnica elimina los ejemplos considerados ruidosos al mismo tiempo que aprende los que considera que no lo son, lo que permite evitar los problemas de sobre-ajuste conduciendo a la obtencion de valores de generalización muy buenos cuando se utilizan conjuntos de datos reales.

La capacidad del algoritmo C-Mantec para crear arquitecturas compactas de muy pocas neuronas fue, en primer lugar, analizada a fondo con un subconjunto de funciones binarias del conjunto de datos de referencia MCNC, conjunto de datos ampliamente utilizado para testar algoritmos de síntesis lógica de circuitos. Los resultados presentados en la Tab. 1.2 indican claramente una gran capacidad de síntesis superando en un 39.08 % los resultados obtenidos previamente por el algoritmo DASG* [**?**]. Vale la pena señalar que los resultados obtenidos por el algoritmo DASG eran ya una mejora en los resultados obtenidos por Zhang et al. [**?**], y por lo tanto los presentes resultados constituyen los mejores resultados conocidos con estas funciones de referencia hasta la fecha actual.

En cuanto a la capacidad de generalización, se ha testado en dos conjuntos diferentes de problemas de referencia: en primer lugar, se analizó capacidad de generalización

con funciones booleanas libres de ruido para comprobar la capacidad del algoritmo para generalizar funciones sin necesidad de tener que eliminar ejemplos. Se usaron 17 problemas de funciones lógicas que pertenecen al conjunto de referencia MCNC, logrando un gran rendimiento en comparación con otros métodos de clasificación estándar, FFNN, árboles de decisión C4.5 y K vecinos más cercanos. La capacidad de generalización media obtenida con C-Mantec fue superior a la obtenida por los otros métodos en este caso. Se realizó una segunda prueba para analizar la capacidad de generalización del algoritmo utilizando un repositorio de 17 conjuntos de datos con entrada continua y dos o más clases de salida. Con el fin de evitar los efectos de sobre-entrenamiento se uso la técnica de aprendizaje activo para la selección y eliminación de ejemplos considerados como ruidosos. Los resultados mostraron que C-Mantec obtiene resultados muy competitivos en generalización en comparación a los obtenidos con FFNN y SVM. A pesar de que los resultados en generalización fueron bastantes similares entre los tres enfoques, la ventaja de los algoritmos constructivos estándares contra FFNN es que la elección de la arquitectura y el ajuste de los pesos sinápticos se realiza automáticamente durante el proceso de aprendizaje.

En cuanto a los tres enfoques diferentes analizados para la extensión del algoritmo C-Mantec en el tratamiento de problemas con más de 2 clases de salida, los mejores resultados, en términos de la capacidad de generalización, se obtuvieron utilizando el enfoque PAQ, que también permitió obtener arquitecturas de un tamaño razonable.

Los resultados obtenidos muestran que el efecto de la aplicación de la competencia en el algoritmo es beneficiosa y relevante para la construcción de arquitecturas más compactas y la obtención de buenos valores de generalización.

En relación a la capacidad de generalización del algoritmo, el hecho de que todas las neuronas pueden aprender en todo momento hace que la carga de las neuronas en el aprendizaje esté muy distribuida, una clara diferencia con los algoritmos constructivos existentes. De hecho, Smieja [**?**] mostró que la distribución no balanceada de carga entre las neuronas es un factor que degrada la capacidad de generalización de los algoritmos constructivos en comparación con el enfoque del perceptrón multicapa estándar, que equilibra el aprendizaje entre todas las neuronas presentes en las capas ocultas.

Como conclusión final de este capítulo, los resultados obtenidos permiten mostrar que el algoritmo C-Mantec es una herramienta útil y robusta para la construcción de arquitecturas de redes neuronales compactas con muy buena capacidad de generalización en problemas de clasificación.