Mapas auto-organizativos dinámicos

Alumno: Waldo Hasperué Número de alumno: 1851/5
Director: Lic. Laura Lanzarini

El método AVGSOM propuesto en este trabajo de grado fue publicado con el título “Dynamics Self-organizing Maps. A new strategy to enhance topology preservation” de W. Hasperué y L. Lanzarini en la XXXI Conferencia Latinoamericano de Informática, CLEI 2005. Octubre de 2005, Cali, Colombia.

Agradecimientos

A mis padres, Jorge y Mercedes, quienes me brindaron todo lo necesario (y mucho más) para que pudiera finalizar mis estudios. A Laura Lanzarini, quien con su inagotable paciencia logró que este trabajo de grado llegara a buen puerto.

Tabla de contenido

1. INTRODUCCIÓN

La aparición de las computadoras digitales y el desarrollo de las modernas teorías acerca del aprendizaje y del procesamiento neuronal se produjeron aproximadamente al mismo tiempo, al final de los años cuarenta. A partir de ese momento, las computadoras han sido utilizadas como herramientas para modelar neuronas individuales, así como agrupaciones de neuronas, denominadas redes neuronales artificiales.

El estudio de las redes neuronales artificiales, inspirado por la comprensión actual del cerebro, ha logrado varias metas. Los primeros sistemas aparecieron al final de la década de los cincuenta, con el aporte de Frank Rosenblatt: el perceptrón, y años mas tarde Bernard Widrow introdujo se modelo: el ADALINE. Por más de 10 años estos estudios habían sido dejados de lado por muchos investigadores, luego de que Marvin Minsky en 1969, escribiera un libro mostrando las limitaciones del Perceptrón.

A pesar del libro de Minsky, hubo otros que siguieron adelante. A principios de la década de los ochenta, apareció el exitoso algoritmo de backpropagation, y con él un conjunto de nuevas arquitecturas y estrategias de entrenamiento basadas en diversas ideas.

Así, hoy en día podemos observar, por un lado, el rápido avance tecnológico que ha logrado que las computadoras actuales sean capaces de resolver complejos cálculos matemáticos a velocidades increíblemente rápidas; y por el otro, el estudio de las teorías del aprendizaje y de las redes neuronales artificiales que aún muestran varias dificultades para la realización de tareas sencillas, como identificar un rostro conocido entre muchos otros, tarea que resulta fácil de hacer para cualquier ser humano.

Evidentemente, las computadoras basadas en la filosofía de los sistemas secuenciales, desarrollados por Von Neuman, muestran gran incapacidad para interpretar el mundo de la manera en que lo hacen los humanos.

Estas dificultades y limitaciones aún llevan a que un gran número de investigadores centre su atención en el desarrollo de nuevos sistemas de tratamiento de la información, que permitan solucionar problemas cotidianos, tal como lo hace el cerebro humano.

El cerebro humano constituye una computadora muy notable, capaz de interpretar información imprecisa suministrada por los sentidos a un ritmo increíblemente veloz. Lo más impresionante de todo es que aprende, sin instrucciones explícitas de ninguna clase, a crear las representaciones internas que hacen posibles estas habilidades. Las principales características de este órgano biológico deseables para cualquier sistema de procesamiento digital son:

- Es robusto y tolerante a fallas. Esto puede observarse en la cantidad de neuronas que mueren diariamente sin afectar su desempeño.

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

- Puede manejar información difusa, con ruido o inconsistente.

- Es altamente paralelo.

Áreas de estudio tales como Minería de Datos, el Reconocimiento de Patrones, Procesamiento de Imágenes y de voz, Control y optimización, entre otras, requieren del uso de herramientas especializadas para encontrar una solución. Una de las herramientas mas utilizadas para ayudar al ser humano a resolver los problemas de estas áreas son las redes neuronales artificiales.

Hoy en día existen muchas arquitecturas de redes neuronales distintas, cada una con sus propias características, algoritmos de aprendizajes y capacidades específicas para solucionar distintas clases de problemas.

Algunas de las arquitecturas más conocidas son la red neuronal Backpropagation, y los Mapas Auto-Organizativos.

La primera ha sido ampliamente utilizada para obtener funciones de correspondencia entre los datos de entrada y de salida. Su entrenamiento es supervisado ya que para realizarlo, es preciso contar con la respuesta esperada para cada uno de los patrones y se basa en adaptaciones sucesivas utilizando una técnica de gradiente descendente.

Los Mapas Auto-Organizativos han demostrado ser una excelente herramienta de agrupamiento (clustering) ya que permiten descubrir, sin información alguna, las características y similitudes entre los datos de entrada, preservando la distribución topológica de los mismos. Esto es lo que los convierte en una estrategia para reducir la dimensionalidad del espacio de entrada facilitando de esta forma su visualización. Preservar la topología de los datos significa que ante patrones de entrada con características similares, las neuronas que mejor respondan a esa entrada estarán ubicadas en posiciones cercanas dentro de la arquitectura.

Las arquitecturas antes mencionadas, así como el resto de las arquitecturas convencionales poseen dos grandes limitaciones. En primer lugar, la dimensión y estructura de la red deben ser definidas a priori, antes de comenzar con el entrenamiento, condicionando de esta forma los resultados y la eficiencia de la respuesta obtenida. En segundo lugar, la capacidad de la red también depende de los valores iniciales utilizados así como los parámetros de aprendizaje.

Existen una variada gama de soluciones que facilita la toma de decisiones al momento de diseñar y entrenar una Red Neuronal. Por ejemplo, para determinar la cantidad de neuronas de una arquitectura se han definido estrategias constructivas (aquellas que parten de un estructura mínima que va creciendo durante el entrenamiento) y destructivas (las que van “recortando” las partes de la red que no cumplen ninguna función).

En lo referido a los problemas de agrupamiento o clustering y como forma de facilitar el diseño de una red neuronal competitiva, el primer enfoque es el más utilizado dando lugar a la aparición de los Mapas Auto-Organizativos Dinámicos. Básicamente poseen un comportamiento similar a los Mapas Auto-Organizativos clásicos en lo que se refiere a la organización de sus neuronas competitivas y la preservación de la

topología de los datos pero se diferencian en que el número de sus elementos varía durante el entrenamiento. Así durante el entrenamiento de un Mapa Auto-Organizativo dinámico, se pueden crear y eliminar tantas neuronas como sea necesario.

Entre los distintos métodos existentes propuestos para definir la arquitectura puede observarse que la incorporación de elementos es variada encontrando redes neuronales que los agregan de manera aislada hasta otras que adicionan capas completas. En general, estos procesos se realizan en posiciones donde el error es alto o para mantener la topología de los datos. Dado que se trata de una estructura de crecimiento dinámico, su problema radica en garantizar que las neuronas que se vayan incorporando respeten la topología de los datos de entrada de manera que elementos similares conserven posiciones próximas dentro de la arquitectura.

1.1. OBJETIVOS

En este trabajo de grado, es de interés resolver problemas de agrupamiento de información utilizando mapas auto-organizativos dinámicos. El objetivo es estudiar y analizar las estrategias considerando los mapas auto-organizativos tanto estáticos como dinámicos y proponer un nuevo método de entrenamiento para estas arquitecturas competitivas que permita mejorar la preservación de la topología de los datos de entrada. La eficacia del método propuesto será medida en base a su aplicación en la resolución de diversos problemas.

1.2. RESULTADOS OBTENIDOS

Se ha desarrollado e implementado un nuevo método denominado AVGSOM, basado en la filosofía de los mapas auto-organizativos dinámicos y se compararon los resultados de su aplicación a varios conjuntos de datos respecto de otras propuestas similares existentes. Para tal fin se ha implementado una herramienta que permite seleccionar el problema a resolver, realizar distintas pruebas con los algoritmos implementados modificando los parámetros de entrada. También es posible visualizar los resultados obtenidos.

1.3. ORGANIZACIÓN DEL TRABAJO DE GRADO

En el capítulo 2 se detalla el esquema y funcionamiento de las redes neuronales artificiales, comenzando por su principal componente, la neurona artificial, comentando las similitudes y diferencias con las neuronas biológicas. Luego son estudiados los distintos tipos de aprendizaje existentes, y hacia el final del capítulo se comentan las arquitecturas de redes neuronales artificiales clásicas.

Dado que el objetivo del presente trabajo de grado son los mapas auto-organizativos dinámicos, en el capítulo 3 se estudia con más detalle los mapas auto-organizativos de Kohonen, los cuales son la base para sus variantes dinámicas. En el mismo capítulo se estudia, dado que es una gran característica de estos mapas, la preservación de la topología de los datos de entrada junto con distintas métricas. En el capítulo 4 se presentan los mapas auto-organizativos dinámicos, sus diferencias con los SOM de Kohonen, y se mencionan algunos métodos existentes.

En el capítulo 5 es presentado el método AVGSOM, una variante de los mapas auto-organizativos dinámicos, propuesto como objetivo en este trabajo de grado como alternativa para mejorar la preservación de la topología de los datos. En el mismo capítulo se propone un agregado a este método, que actúa como capa de salida. En el capítulo 6 se muestran los resultados obtenidos, luego de entrenar el AVGSOM con distintos conjuntos de datos, comparando dichos resultados con los obtenidos por un método existente, el GSOM.

2. REDES NEURONALES ARTIFICIALES

Basados en la eficiencia de los procesos llevados a cabo por el cerebro, e inspirados en su funcionamiento, varios investigadores han desarrollado desde hace más de 40 años la teoría de las Redes Neuronales Artificiales (RNA), las cuales emulan las redes neuronales biológicas, y se han utilizado para aprender estrategias de solución basadas en ejemplos de comportamiento típico de patrones; estos sistemas no requieren que la tarea a ejecutar se programe, ellos generalizan y aprenden de la experiencia.

Los fundamentos teóricos, las estructuras y el funcionamiento de las redes neuronales artificiales están basados en las redes neuronales biológicas que forman parte del aparato de comunicación neuronal de los animales y del ser humano. Su estructura está formada básicamente por neuronas artificiales y conexiones entre ellas, donde una neurona artificial al ser "activada" envía señales a las neuronas artificiales a las cuales está conectada, llamadas neuronas vecinas.

Originalmente se utilizaron como una simulación abstracta de los sistemas nerviosos biológicos con el objetivo de simular todos los aspectos y características del aprendizaje y de la inteligencia dando origen a una importante área de estudio (Anderson *et al*. 1977; Grossberg 1987; Hoplfield 1982; Kohonen 1984; Minsky & Papert 1969; Rosemblatt 1962; Waltz & Feldman 1988; Widrow 1959 citado en Isasi Viñuela & Galván León 2004).

2.1. NEURONA BIOLÓGICA

El elemento estructural más esencial, en el sistema de comunicación neuronal que se encuentra en los animales y en el hombre, es la célula nerviosa o neurona. Las neuronas tienen tres componentes principales: las dendritas, el cuerpo de la célula o soma, y el axón. Las dendritas, son el árbol receptor de la neurona, son como fibras nerviosas que cargan de señales eléctricas el cuerpo de la célula. El cuerpo de la célula, realiza la suma de esas señales de entrada. El axón es una fibra larga que lleva la señal desde el cuerpo de la célula hacia otras neuronas. El punto de contacto entre un axón de una célula y una dendrita de otra célula es llamado sinapsis, la longitud de la sinapsis es determinada por la complejidad del proceso químico que estabiliza la función de la red neuronal. La figura 2.1 ilustra dos neuronas biológicas, sus componentes y las conexiones entre ellas, llamada sinapsis.

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

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

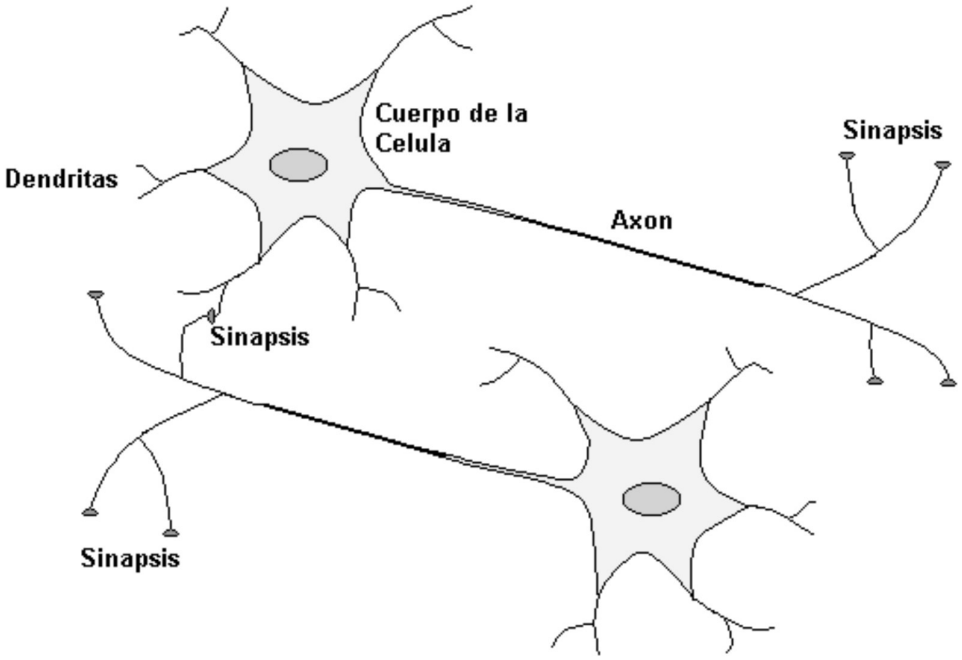


Figura 2.1. Neuronas biológicas.

La mayoría de las neuronas utilizan sus productos de secreción como señales químicas para la transmisión de la información. Dicha información se envía, entre las distintas neuronas, a través de prolongaciones, formando redes, en las cuales elabora y almacena conocimiento. Además, una parte de las neuronas está en relación con receptores, a través de los cuales llegan comunicaciones procedentes del exterior o el interior del organismo hasta las redes neuronales. Otra parte conduce las informaciones, elaboradas en forma de órdenes, hacia los efectores (músculos, glándulas, etc. que interpretan la información en forma de acciones motoras, hormonales, etc.).

Todas las neuronas conducen la información de forma similar, esta viaja a lo largo de axones en breves impulsos eléctricos, denominados potenciales de acción; los potenciales de acción que alcanzan una amplitud máxima de unos 100 mV y duran 1 ms, son resultado del desplazamiento a través de la membrana celular de iones de sodio dotados de carga positiva, que pasan desde el fluido extracelular hasta el citoplasma intracelular; la concentración extracelular de sodio supera enormemente la concentración intracelular.

La membrana en reposo mantiene un gradiente de potencial eléctrico de -70mv, el signo negativo se debe a que el citoplasma intracelular está cargado negativamente con respecto al exterior; los iones de sodio no atraviesan con facilidad la membrana en reposo, los estímulos físicos o químicos que reducen el gradiente de potencial, o que despolaricen la membrana, aumentan su permeabilidad al sodio y el flujo de este ión hacia el exterior acentúa la despolarización de la membrana, con lo que la permeabilidad al sodio se incrementa más aún.

Alcanzado un potencial crítico denominado "umbral", la realimentación positiva produce un efecto regenerativo que obliga al potencial de membrana a cambiar de signo. Es decir, el interior de la célula se torna positivo con respecto al exterior, al cabo de 1

ms, la permeabilidad del sodio decae y el potencial de membrana retorna a -70mv, su valor de reposo. Tras cada explosión de actividad iónica, el mecanismo de permeabilidad del sodio se mantiene refractario durante algunos milisegundos; la tasa de generación de potenciales de acción queda así limitada a unos 200 impulsos por segundo, o menos.

Aunque los axones puedan parecer hilos conductores aislados, no conducen los impulsos eléctricos de igual forma, como hilos eléctricos no serían muy valiosos, pues su resistencia a lo largo del eje es demasiado grande y a resistencia de la membrana demasiado baja; la carga positiva inyectada en el axón durante el potencial de acción queda disipada uno o dos milímetros más adelante, para que la señal recorra varios centímetros es preciso regenerar frecuentemente el potencial de acción a lo largo del camino la necesidad de reforzar repetidamente esta corriente eléctrica limita a unos 100 metros por segundo la velocidad máxima de viaje de los impulsos, tal velocidad es inferior a la millonésima de la velocidad de una señal eléctrica por un hilo de cobre.

Los potenciales de acción, son señales de baja frecuencia conducidas en forma muy lenta, estos no pueden saltar de una célula a otra, la comunicación entre neuronas viene siempre mediada por transmisores químicos que son liberados en las sinapsis. Un ejemplo de comunicación entre neuronas y del proceso químico de la liberación de neurotransmisores se ilustra en la figura 2.2.

Cuando un potencial de acción llega al terminal de un axón son liberados transmisores alojados en diminutas vesículas, que después son vertidos en una hendidura de unos 20 nanómetros de ancho que separa la membrana presináptica de la postsináptica; durante el apogeo del potencial de acción, penetran iones de calcio en el terminal nervioso, su movimiento constituye la señal determinante de la exocitosis sincronizada, esto es la liberación coordinada de moléculas neurotransmisoras. En cuanto son liberados, los neurotransmisores se enlazan con receptores postsinápticos, instando el cambio de la permeabilidad de la membrana.

Cuando el desplazamiento de carga hace que la membrana se aproxime al umbral de generación de potenciales de acción, se produce un efecto excitador y cuando la membrana resulta estabilizada en la vecindad el valor de reposo se produce un efecto inhibidor. Cada sinapsis produce sólo un pequeño efecto, para determinar la intensidad (frecuencia de los potenciales de acción) de la respuesta cada neurona ha de integrar continuamente hasta unas 1000 señales sinápticas, que se suman en el soma o cuerpo de la célula.

En algunas neuronas los impulsos se inician en la unión entre el axón y el soma, y luego se transmiten a lo largo del axón a otras células nerviosas. Cuando el axón está cerca de sus células destino, se divide en muchas ramificaciones que forman sinapsis con el soma o axones de otras células. Las sinapsis pueden ser excitatorias o inhibitorias según el neurotransmisor que se libere, cada neurona recibe de 10.000 a 100.000 sinapsis y su axón realiza una cantidad similar de sinapsis.

Las sinapsis se clasifican según su posición en la superficie de la neurona receptora en tres tipos: axo-somática, axo-dendrítica, axo-axónica. Los fenómenos que ocurren en la sinapsis son de naturaleza química, pero tienen efectos eléctricos laterales que se pueden medir.

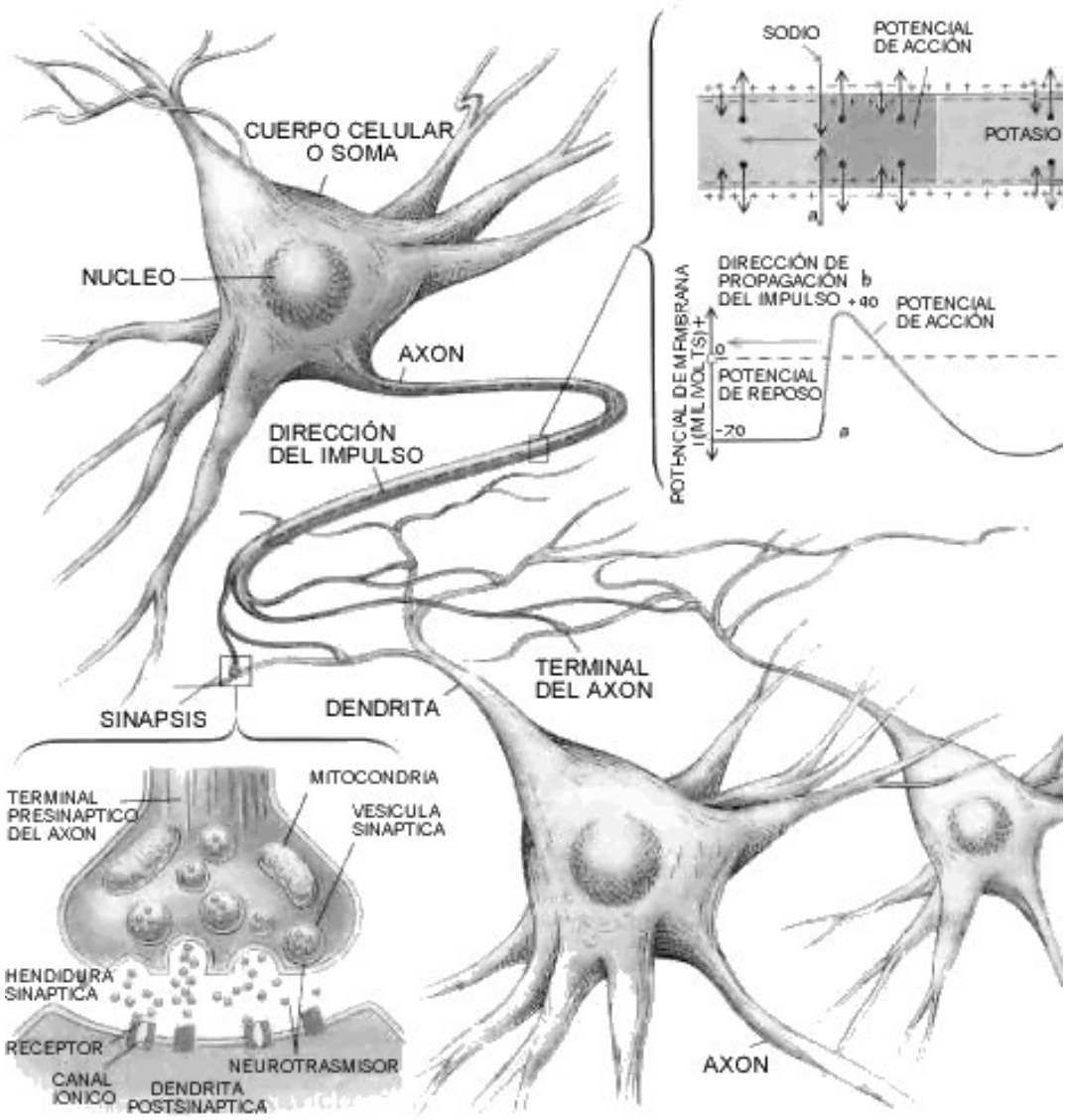


Figura 2.2. Un ejemplo de comunicación entre neuronas y del proceso químico de la liberación de neurotransmisores.

En la figura 2.3 se visualiza el proceso químico de una sinapsis y los diferentes elementos que hacen parte del proceso tanto en la neurona presináptica, como en la postsináptica.

El efecto de la sinapsis sobre la neurona receptora puede ser excitatorio o inhibitorio, y es variable, de manera que podemos hablar de la fuerza o efectividad de una sinapsis. Las señales excitatorias e inhibitorias recibidas por una neurona se combinan, y en función de la estimulación total recibida, la neurona toma un cierto nivel de activación, la cual genera breves impulsos nerviosos con una determinada frecuencia, y propaga, por su axón, una nueva señal hacia las neuronas con las cuales está conectada.

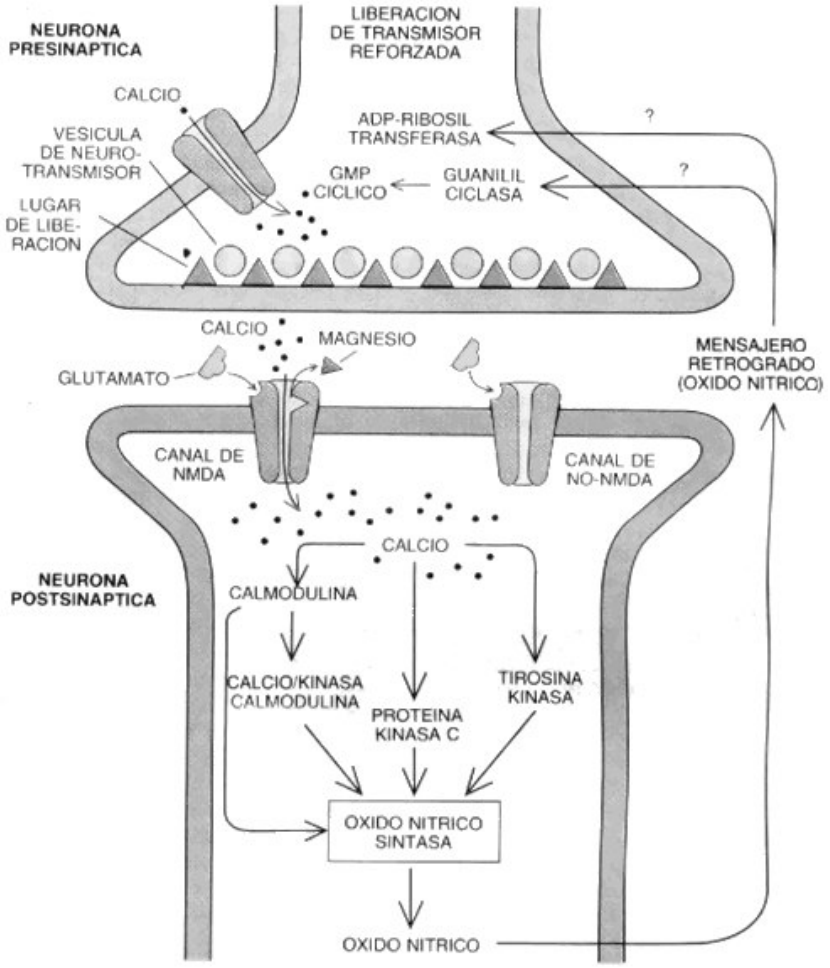


Figura 2.3. Proceso químico de una sinapsis.

El objetivo básico de las neuronas biológicas comprende cinco funciones:

- Recoger la información que llega a ellas en forma de impulsos procedentes de otras neuronas o de receptores.

- Integrar la información en un código de activación propio de la célula.

- Transmitir la información codificada en forma de frecuencia de impulsos a través de su axón.

- Efectuar, a través de las ramificaciones del axón, la distribución espacial de los mensajes.

- Transmitir los impulsos de los terminales de las ramificaciones del axón a las neuronas subsiguientes o a las células efectoras.

Algunas neuronas, denominadas receptores, reciben información directamente del exterior. A esta información se le da el nombre de estímulo. Los estímulos son el

mecanismo de contacto de un organismo que se encargan de recibir la información visual, auditiva, táctil, etc. Esta información, una vez elaborada, pasa a ser tratada como el resto de la información de sistema nervioso y convertida en impulsos electro-químicos. Son estos impulsos los que, básicamente, ponen en funcionamiento la red neuronal. Así se propaga todas las señales de la neurona a otras neuronas hasta que llega a los efectores, órganos, glándulas, músculos, que transforman la información recibida en acciones motoras, hormonales, etc.

Las neuronas artificiales, que serán descriptas en la próxima sección, si bien se asemejan bastante a sus homólogas biológicas, no alcanzan la complejidad de las mismas. Ambas comparten una característica fundamental: las conexiones entre ellas determinan la función de la red, tanto la biológica como la artificial.

2.2. NEURONA ARTIFICIAL

Las redes neuronales artificiales intentan incorporar el mecanismo y funcionamiento de las redes neuronales biológicas. Aún así, las redes neuronales artificiales están muy lejos de parecerse, en cuanto a capacidad de resolver problemas, a las redes neuronales biológicas.

Una de las diferencias fundamentales entre los sistemas biológicos y las redes neuronales artificiales es la complejidad de las sinapsis. En los sistemas biológicos estos puntos de interconexión tienen miles de componentes y de procesos activos de propagación de los impulsos electro-químicos. En cambio, las redes neuronales artificiales tienen conexiones relativamente simples, en la que por lo general se realiza una suma ponderada de las entradas, a la que se le aplica una función de umbral.

Por otro lado, en los sistemas biológicos la información se propaga por medio de impulsos electro-químicos que al llegar a las células, dependiendo de su intensidad y de una serie de factores fisiológicos de las neuronas, producirán una serie de reacciones en estas. Estos impulsos se reciben en cualquier momento, de forma que el funcionamiento de los sistemas biológicos se puede calificar como de totalmente asíncrono, y permite la reconfiguración de la red en cualquier instante de tiempo, por lo que producen patrones temporales de forma continua. En cambio, en las redes neuronales artificiales, los parámetros se actualizan de forma periódica, en intervalos de tiempo discretos, y por lo general, todos a la vez, con lo cual se le presentan patrones en intervalos de tiempo prefijados, y no los podrá recoger de forma esporádica de su entorno.

Además las redes neuronales biológicas tienen la propiedad de poder aprender, a partir de unas pocas presentaciones de patrones. Por el contrario, las redes neuronales artificiales normalmente convergen muy lentamente, y pueden necesitar de cientos o miles de presentaciones de patrones para llegar a realizar una generalización aceptable.

En cuanto a las interconexiones entre neuronas, las redes neuronales artificiales suelen tener unas arquitecturas simples (por capas, interconectadas todas con todas, autoasociativas, etc.) y fijas, mientras que las redes neuronales están estructuradas por niveles: córtex, circunvoluciones, cerebelo, hipocampo. Todos ellos con estructuras que no son simples filas de neuronas interconectadas, sino que forman una malla de conexiones muy densa, sin estructura aparente y variable con el tiempo.

Otra de las limitaciones importantes de los sistemas automáticos es que centran su aprendizaje y funcionamiento en una única tarea en concreto, al contrario de los sistemas biológicos, que pueden aprender simultáneamente un gran número de tareas de tipos muy diversos.

A pesar de estar muy lejos de conseguir representar todas las características de las redes neuronales biológicas, las redes neuronales artificiales tienen en común con ellas algunas de sus principales características, como son las de representación y procesamiento distribuido de la información, al igual que su estructura básica. Así, las redes neuronales artificiales, análogamente a las redes neuronales biológicas, están formadas por neuronas artificiales, las cuales a su vez tienen una analogía con las neuronas biológicas.

Las neuronas artificiales están formadas por conexiones de entrada por las cuales reciben señales de otras neuronas. En el caso de las neuronas que forman parte de la capa de entrada, reciben señales del "mundo exterior". Las neuronas artificiales también cuentan con una o más conexiones de salida, por las cuales envían señales a otras neuronas artificiales. Además, poseen una función de propagación y una función de activación.

En la figura 2.4 se ilustra un modelo típico de una neurona artificial. En esta figura un grupo de entrada x_1, x_2, ..., x_n son introducidas en la neurona artificial. La señal recibida en cada entrada de la neurona artificial es una variable continua, en lugar de los pulsos eléctricos de las neuronas biológicas. Al conjunto de señales de las conexiones de entrada de una neurona artificial se lo conoce como vector de entrada o patrón de entrada.

En una red neuronal artificial las neuronas artificiales poseen, generalmente, un vector de pesos. Este vector de pesos juega un rol fundamental en el proceso de calcular el "nivel" de señal que la neurona emitirá por la conexión de salida hacia otras neuronas artificiales (Figura 2.5). La longitud del vector será la misma que la longitud de los vectores de entrada, los usados para alimentar la red neuronal artificial, y dependerá de la representación del problema a resolver.

En la bibliografía existente se utilizan diferentes términos para hacer referencia a una neurona artificial, los más utilizados son neurona, unidad, célula, nodo, elemento de procesamiento y celda. En este trabajo se utilizará siempre el término neurona.

A partir de este punto, y a menos que se aclare lo contrario, cada vez que se mencionen tanto a las "redes neuronales" o simplemente "redes", como a las "neuronas", se estará haciendo referencia a las redes neuronales artificiales y a las neuronas artificiales respectivamente.

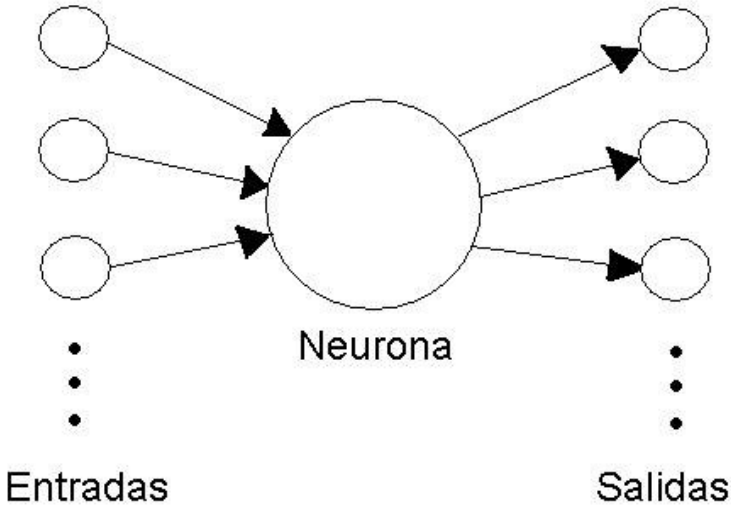


Figura 2.4. Una neurona artificial típica.

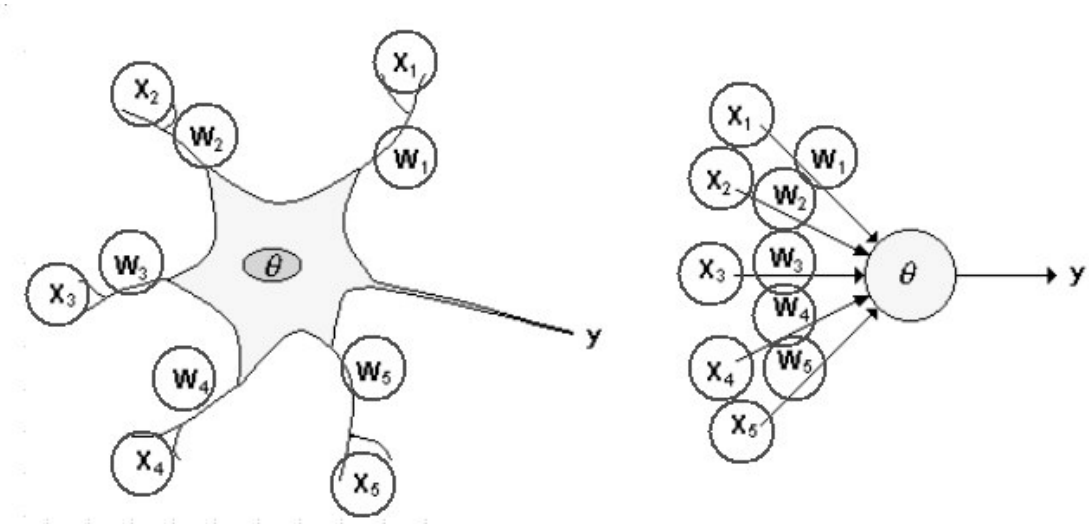


Figura 2.5. A la izquierda, una neurona biológica con cinco dendritas (entradas), las cuales son afectadas por su correspondiente peso, y el axón (salida) produciendo una señal y. A la derecha, una neurona artificial, con cinco señales de entrada, cada una afectada por su correspondiente peso y produciendo una señal de salida y. De esta ilustración se puede observar las similitudes del funcionamiento entre la neurona artificial y la neurona biológica.

2.3. FUNCIÓN DE PROPAGACIÓN Y FUNCIÓN DE ACTIVACIÓN

La función de propagación calcula el valor de entrada total de la neurona, generalmente como simple suma ponderada de todas las entradas recibidas (Figura 2.6). Equivale a la combinación de las señales excitatorias e inhibitorias de las neuronas biológicas. Cada señal de entrada pasa a través de una ganancia o peso, llamado peso sináptico o fortaleza de la conexión cuya función es análoga a la de la función sináptica de la neurona biológica.

La función de propagación más utilizada es la función de base lineal (LBF), que consiste en la suma ponderada de todas las entradas. Cada señal de entrada se multiplica

por su peso asociado, w_1, w_2, ..., w_n, y luego realiza la sumatoria de todas las señales ponderadas para obtener la señal de entrada neta ω:

$$\varpi = \sum_{i=1}^{n} x_i w_i \tag{2.1}$$

donde x es el vector de entrada a la neurona, y w es el vector de pesos de la neurona. n es la dimensión de los dos vectores, el vector de entrada y el vector de pesos.

Una función de propagación no lineal es la conocida como función de base radial (RBF), que es una función de tipo Gaussiana:

$$\varpi = e^{\frac{x-1}{\sigma^2}}$$

donde x es el vector de entrada a la neurona.

La función de activación o de transferencia es la característica principal y define el comportamiento de la neurona. Se encarga de calcular el nivel o estado de activación de la neurona en función de la entrada total. La función de activación determina que si las influencias excitadoras positivas predominan, entonces la neurona emite una señal positiva y manda dicha señal a otras neuronas por su conexión de salida. De la misma manera, si las influencias inhibitorias son las que predominan, entonces la neurona emite una señal negativa hacia otras neuronas.

Existen funciones en la cual la salida es un valor discreto (típicamente binario 0/1) que depende de si la estimulación total supera o no un determinado valor de umbral. Y funciones no lineales que no son proporcionales a la entrada, como las funciones sigmoidales y las funciones Gaussianas.

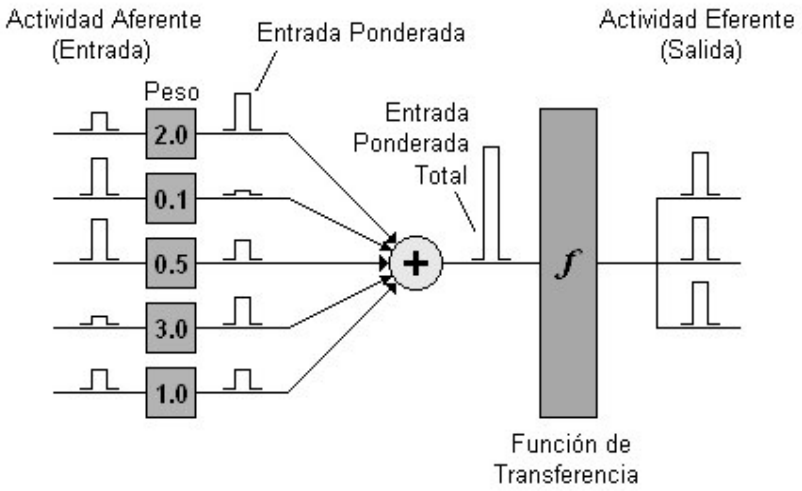


Figura 2.6. Funcionamiento de una neurona artificial. Cada entrada es multiplicada por el peso correspondiente. La neurona calcula su entrada por lo general utilizando la ecuación (2.1). El valor de salida (único) se obtiene como $y = f(\text{neta})$, donde f es la función de activación.

La salida de una neurona está conectada a una o más neuronas. Es importante aclarar que la señal de salida de una neurona es única, y la misma se emite a todas las neuronas a las que está conectada.

A continuación se detallan las funciones de activación más utilizadas.

2.3.1. Función umbral

La función umbral más simple se puede modelar con una función escalón. Por ejemplo, en el escalón binario, la función devuelve 0 si el valor de la función de activación está por debajo de un umbral, y devuelve 1 si supera el umbral. Una variante es el escalón bipolar que devuelve 1 o -1.

$$f(x) = \begin{cases} 1 & si \quad x \geq \mu \\ 0 & si \quad x < \mu \end{cases}$$

donde μ es un umbral establecido a priori, y que dependerá del problema que se quiera solucionar.

2.3.2. Función lineal

La salida tiene una relación lineal con la entrada:

$$f(x) = a + bx$$

En la figura 2.7 se muestra la forma de la función.

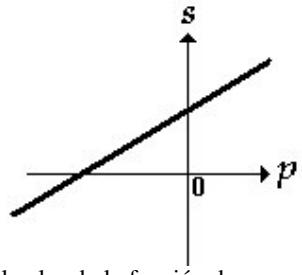


Figura 2.7. Relación lineal entre el valor de la función de propagación *p* y el valor de señal de salida *s* de la neurona.

2.3.3. Función sigmoidal

Se trata de una función continua no lineal. La función sigmoidal posee un rango comprendido entre 0 y 1. Por lo tanto, sin importar cual sea la entrada de la neurona, su salida estará comprendida entre 0 y 1. La salida de una neurona vale 0,5 cuando la entrada es nula, esto significa que la neurona tiene cierta actividad aún en ausencia de estimulación. Al aumentar la estimulación, la neurona aumenta su activación, y la disminuye si la estimulación es inhibitoria (Figura 2.8). Está definida como:

$$f(x) = \frac{1}{1 + e^{-x}}$$

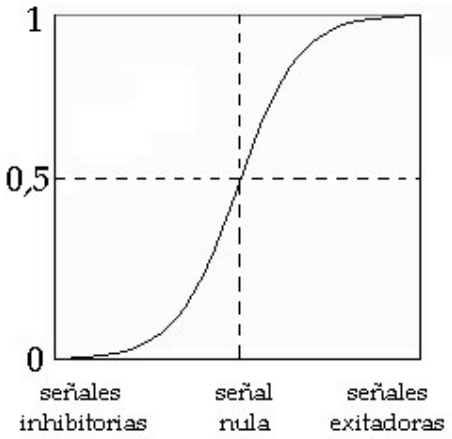


Figura 2.8. Forma de la ecuación sigmoidal. Cuando la señal de entrada es nula, el valor de la función es 0,5. Cuando las señales de entrada son excitadoras, el valor de la función tiende a 1, y cuando las señales de entrada son inhibitorias, el valor de la función tiende a 0.

2.3.4. Función Gaussiana

Esta función da como resultado el máximo valor que puede alcanzar cuando la señal de entrada es nula. Y a medida que las señales de entrada son más inhibitorias o más excitadoras, entonces la función devuelve valores cercanos a 0. Se define como:

$$f(x) = \alpha \quad e^{-x^2/\sigma^2}$$

donde α es un parámetro que indica el máximo valor que puede tomar la función, y σ indica la amplitud de la "campana". En la figura 2.9 se muestra la forma de la función.

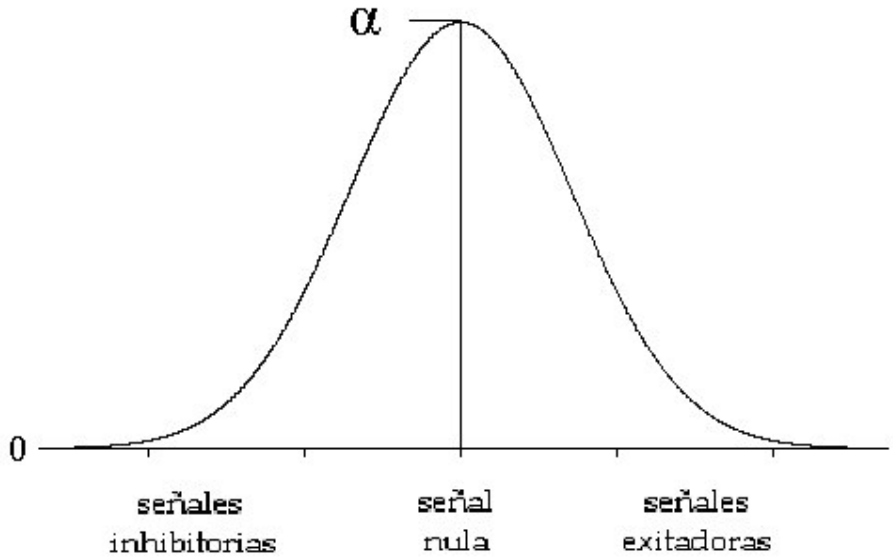


Figura 2.9. Forma de la ecuación Gaussiana.

2.4. ENTRENAMIENTO Y APRENDIZAJE

En un paradigma convencional de programación informática, el objetivo del programador es modelar matemáticamente el problema en cuestión y formular una solución (programa) mediante un algoritmo que tenga una serie de propiedades que permitan resolver dicho problema. De esta manera, el algoritmo es una entidad autómata que obedece sentencia por sentencia hasta finalizar su ejecución.

Las redes neuronales tienen un algoritmo, el cual es programado, que por si solo no da una solución al problema que se quiere resolver. El algoritmo de la red recibe como entrada un conjunto de datos y el procesamiento consiste en la modificación de los pesos de las neuronas con algún criterio pre-establecido. Estos dos pasos, el de recibir datos de entrada y el de modificar los pesos de las neuronas, se repiten una cantidad significativa de veces, hasta alcanzar algún estado que indica el fin del algoritmo. La ejecución de estos pasos se denomina "*entrenamiento*" de la red neuronal.

Básicamente el "*entrenamiento*" puede resumirse de la siguiente manera; cada vector de entrada es introducido en la red copiando cada valor de dicho vector en la neurona de entrada correspondiente. Cada neurona de la red, una vez recibida todas sus entradas, las procesa y genera una salida que es propagada a otras neuronas. Una vez que la entrada ha sido completamente propagada por toda la red, el vector de salida estará formado por cada uno de los valores de salida de las neuronas de salida.

El objetivo del "*entrenamiento*" es conseguir que la red "*aprenda*" a resolver el problema automáticamente. Esto significa que la red será capaz de distinguir entre distintos datos de entrada, y para cada uno de ellos, será capaz de decir a que grupo genérico pertenece. El "*aprendizaje*" consiste en la determinación de los valores precisos de los pesos para todas sus conexiones, de manera tal que la red quede capacitada para la resolución eficiente de un problema.

Si bien el algoritmo consta de pocos pasos, elegir la arquitectura adecuada para que el "*entrenamiento*" se lleve a cabo de manera exitosa, no es trivial. El "*aprendizaje*" debe poseer las siguientes características:

- Debe haber un número suficiente de patrones de entrada. Si el conjunto de patrones de entrada es reducido, la red no será capaz de adaptar sus pesos de forma eficaz.

- Los componentes del conjunto de patrones de entrada deberán ser diversos. Si el conjunto tiene muchas más instancias de un tipo que el resto, la red se especializará en dicho subconjunto. Por eso es importante que todas las regiones significativas del espacio de estados estén suficientemente representadas en el conjunto de patrones de entrada.

En esta misma sección se verán distintos tipos de "*aprendizajes*" que se pueden utilizar para "*entrenar*" una red neuronal. Y en la sección 2.5 se verán distintos tipos de arquitecturas, formas de modelar la red para que el "*entrenamiento*" tenga éxito.

2.4.1. Aprendizaje supervisado

Las redes neuronales con aprendizaje supervisado son aquellas en que todos los patrones de entrada, durante el entrenamiento, son comparados con algún patrón de salida. Algunas arquitecturas conocidas de las primeras redes con este tipo de adaptación son: red perceptrón y Adaline.

Por lo general, el entrenamiento de estas redes con aprendizaje supervisado está formado por dos etapas: la etapa de entrenamiento y la etapa de prueba y la función básica de la red es maximizar o minimizar algún criterio o función de performance definida a priori. Para poder determinar si la red produce salidas adecuadas, primero se divide el conjunto de patrones de entrada en dos, el subconjunto de patrones de entrada para el entrenamiento, y el subconjunto de patrones de entrada para la validación. Luego se entrena la red con el subconjunto de patrones de entrada reservados para tal fin y una vez finalizado el entrenamiento se presenta a la red el subconjunto de patrones de entrada destinados a la validación. Para cada patrón de entrada de este último subconjunto se mide el error producido por la red, y de acuerdo al error total acumulado luego de presentar todos los patrones de entrada se determina si la red está bien entrenada o no. Para que este proceso sea eficaz, al dividir el conjunto de patrones de entrada se deben tener presentes las siguientes observaciones:

- El subconjunto de validación debe ser independiente del de aprendizaje. No puede haber ningún tipo de sesgo en el proceso de selección de los datos de validación.

- El subconjunto de validación debe cumplir las propiedades del conjunto de entrenamiento descriptas en la sección 2.4.

En el aprendizaje supervisado, la red recibe como entrada un patrón que está formado por el dato de entrada, con el cual se pretende que la red aprenda y un patrón "maestro" o salida esperada con el cual se comparará la salida de la red neuronal. Cada vez que se ingresa un patrón de entrada a la red, se procesa para obtener una salida. Dicha salida se compara con el patrón "maestro" correspondiente. La diferencia entre ambas influirá en la modificación de los pesos. La figura 2.10 ilustra el proceso de entrenamiento del aprendizaje supervisado.

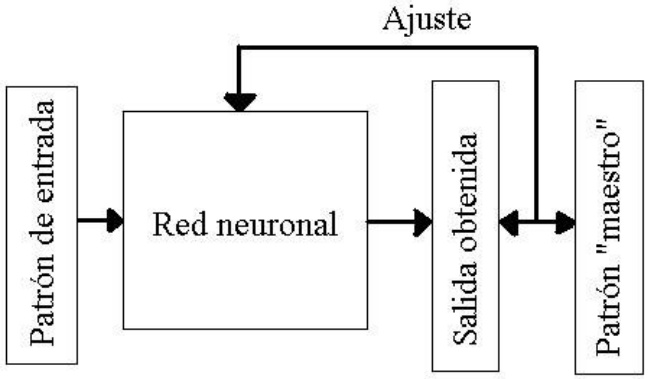


Figura 2.10. "Entrenamiento" de una red con aprendizaje supervisado

Existen dos aproximaciones al aprendizaje supervisado. Uno se basa en reglas de corrección de errores, y otro se basa en reglas de gradiente descendiente. Ambas se explican con detalle en Hassoun 1995.

2.4.2. Aprendizaje por refuerzo

El aprendizaje por refuerzo es un proceso de prueba y error diseñado a maximizar el valor esperado de una función criterio. La idea básica del aprendizaje por refuerzo tiene su origen en la psicología a raíz de distintos experimentos con el aprendizaje de animales (Thorndike 1911, citado en Hassoun 1995). Este aprendizaje está basado en la idea que si una acción es seguida de un “progreso”, entonces la tendencia a producir esa acción es incrementada.

Si bien para cada entrada se conoce la salida deseada, la idea no es que la red las asocie como en el aprendizaje supervisado sino que la salida sea una señal de refuerzo que informe a la red sobre su performance ante la entrada dada.

Esta "señal de refuerzo" está caracterizada por el hecho de que es menos informativa que en el caso de aprendizaje supervisado mediante ejemplos. Los pesos se ajustan en base a la señal de refuerzo según un mecanismo de probabilidades. Si una acción tomada por el sistema de aprendizaje es seguida por un estado satisfactorio, entonces la tendencia del sistema a producir esa particular acción es reforzada. En otro caso, la tendencia del sistema a producir dicha acción es disminuida.

Para el caso del aprendizaje por refuerzo, el aprendizaje tiene una serie de características que es importante destacar. En este caso el conjunto de patrones de entrada utilizado para el aprendizaje está compuesto por ejemplos que contienen datos y sus salidas deseadas. El proceso consiste en modificar los pesos de la red hasta que para todos los ejemplos del conjunto de entrenamiento, la salida producida sea lo más parecida a la deseada. Sin embargo, esto no siempre indica que la red será capaz de solucionar el problema, ya que lo importante no es que la red de buenas salida sobre el conjunto de aprendizaje, que son conocidas, sino sobre los datos que puedan presentarse en el futuro.

2.4.3. Aprendizaje no supervisado

En el aprendizaje no supervisado no existen patrones “maestros” con los cuales comparar los datos de entrada. El objetivo de la red es distinguir características en los datos de entrada por si sola. La clave para lograr esto es diseñar cuidadosamente la arquitectura de la red, como así también elegir correctamente el criterio de aprendizaje y los valores de los parámetros. La red no recibe ninguna información por parte del entorno que le indique si la salida generada en respuesta a una determinada entrada es o no correcta, por ello suele decirse que estas redes son capaces de autoorganizarse.

Los vectores n-dimensionales de entrada serán mapeados en una estructura de dos o tres dimensiones (en algunas arquitecturas hasta más de tres), y cualquier relación topológica entre los vectores de datos se verá reflejada en el espacio de salida. Algunas redes con aprendizaje no supervisado, como los mapas auto-organizativos, generan un mapeo de características (featuring mapping), de esta manera se obtiene en las neuronas

de salida una disposición de tal forma que si se presentan a la red informaciones similares, siempre estarán afectadas neuronas de salida próximas entre sí, en la misma zona de la red.

Algunos ejemplos típicos del aprendizaje no supervisado son el aprendizaje asociativo, y el aprendizaje competitivo. Tanto el aprendizaje asociativo como el competitivo normalmente se utilizan para medir la familiaridad o extraer características de los datos de entrada. El aprendizaje competitivo se caracteriza por dejar que las neuronas que conforman la red compitan por el derecho de representar un subconjunto de patrones de entrada de características similares y suele orientarse hacia la clasificación de los datos de entrada, técnica conocida como agrupamiento (clustering).

La figura 2.11 ilustra el proceso de entrenamiento del aprendizaje no supervisado.

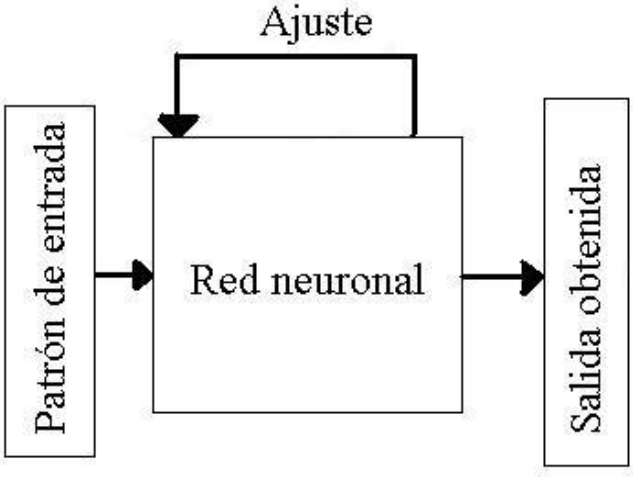


Figura 2.11. "Entrenamiento" de una red con aprendizaje no supervisado

2.4.3.1. Características básicas

Los procedimientos de clasificación no supervisados se basan en técnicas de clasificación, que tienen como objetivo formar grupos de patrones parecidos. Esta técnica de clasificación es muy útil para la clasificación de datos. Además, juega un papel muy importante en las redes de aprendizaje competitivo. En un procedimiento de clasificación, es necesario definir una distancia o medida de similitud, para evaluar el grado de semejanza de los patrones. La medida mas utilizada es la distancia euclídea. Pero pueden utilizarse otras medidas como el producto interno, la distancia Euclídea pesada, o la distancia de Hamming.

Sin la supervisión de ningún patrón "maestro" con el cual comparar, las redes de aprendizaje no supervisado adaptan los pesos y verifican los resultados únicamente a partir de los patrones de entrada. Un esquema que se usa mucho para la adaptación de los pesos es la regla de aprendizaje competitivo, que hace que las neuronas compitan por el derecho a responder por ellas mismas por un determinado tipo de entrada. El objetivo de esta competición es dividir un conjunto de patrones de entrada en un número de clases tal que los patrones de la misma clase exhiben un cierto grado de similitud. Las reglas de entrenamiento suelen ser la regla de Hebb para la red de propagación y la regla de "el ganador lo toma todo" para la red competitiva o de interacción lateral.

2.4.3.2. Regla de Hebb

Una asociación es cualquier vínculo entre la entrada de un sistema y su correspondiente salida. Cuando dos patrones son vinculados por una asociación, el patrón de entrada es a menudo referido como el estímulo, y la salida es referida como la respuesta.

El aprendizaje asociativo fue inicialmente estudiado por escuelas de psicología, las cuales se dedicaron a estudiar las relaciones entre el comportamiento humano y el comportamiento animal. Una de las primeras influencias en este campo fue el experimento clásico de Pavlov, en el cual se entrenó a un perro para salivar al escuchar el sonido de una campana si le era presentado un plato de comida, este es un ejemplo del llamado condicionamiento clásico. Otro de los principales exponentes de esta escuela fue B. F. Skinner, su experimento involucró el entrenamiento de ratas, las cuales debían presionar un botón para obtener comida, a este tipo de entrenamiento se le llamo condicionamiento instrumental.

Basado en este tipo de comportamiento, Donald Hebb postuló el siguiente principio conocido como la regla de Hebb:

"*Cuando un axón de una celda A está lo suficientemente cerca de otra celda B como para excitarla y repetidamente ocasiona su activación, un cambio metabólico se presenta en una o ambas celdas, tal que la eficiencia de A, como celda excitadora de B, se incrementa*". Con el término celda, Hebb se refería a un conjunto de neuronas fuertemente conexionadas a través de una estructura compleja, la eficiencia podría identificarse con la intensidad o magnitud de la conexión, es decir, el peso.

Si bien la regla de Hebb fue presentada en el área biológica, analógicamente tiene su correspondencia con las neuronas artificiales. Analíticamente, el vector de pesos de la neurona se modifica de acuerdo con la siguiente expresión:

$$w_{ij}^{new} = w_{ij}^{old} + \alpha\, x_{ij}\, y_{ij}$$

donde w_{ij} es el vector de pesos de la neurona, x_{ij} es la actividad de entrada e y_{ij} es la actividad de salida. α es un parámetro conocido como factor de aprendizaje, e indica la proporción en la cual se debe modificar el peso de la neurona.

2.4.3.3. Aprendizaje competitivo

Una red básica de aprendizaje competitivo tiene una capa de neuronas de entrada y una capa de neuronas de salida. Un patrón de entrada x es un punto en el espacio n-dimensional. Así, hay tantas neuronas de salida como número de clases y cada nodo de salida representa una categoría de patrones.

La red de aprendizaje competitivo está formada por una o mas redes excitadoras hacia adelante y redes inhibidoras laterales. La red hacia adelante normalmente implementa la regla de aprendizaje de Hebb. La red lateral es inhibidora por naturaleza. Esta red realiza la misión de seleccionar el ganador, normalmente por medio de un método de aprendizaje competitivo. El mas conocido es el método de “el ganador lo

toma todo". En este método, la neurona de salida que da el valor más alto se le asigna el valor total, mientras que todas las demás se le da un valor de 0.

El objetivo de este aprendizaje es categorizar (agrupar) los datos que se introducen en la red, de esta forma las informaciones similares son clasificadas formando parte de la misma categoría y por tanto deben activar la misma neurona de salida. Las clases o categorías deben ser creadas por la propia red, puesto que se trata de un aprendizaje no supervisado a través de las correlaciones entre los datos de entrada.

Otro ejemplo de red competitiva son los mapas auto-organizativos de Kohonen, en las cuales se deja que las neuronas ganadoras interactúen con sus vecinas, con el objetivo de que el entrenamiento logre cierto orden topológico. De esta forma, si la red recibe información con características similares, se generarían mapas parecidos, puesto que serían afectadas neuronas de salidas próximas entre sí.

2.5. ARQUITECTURA DE LAS REDES NEURONALES ARTIFICIALES

Dentro de una red neuronal, las neuronas se encuentran agrupadas en capas. Una capa es un conjunto de neuronas, y tienen distinta clasificación (Figura 2.12):

- Capa de entrada: recibe las señales de la entrada de la red y las envía a las neuronas de la segunda capa. La capa de entrada solo sirve para ejemplificar modelos en la teoría. Por lo general, esta capa solo se utiliza para recibir los datos de entrada sin llevar a cabo ningún procesamiento de los mismos.

- Capas ocultas: Pueden existir varias capas de este tipo en una red, según la arquitectura que se este usando o del problema que se quiera resolver. Estas capas son aquellas que no tienen contacto con el mundo exterior. Sus neuronas pueden tener diferentes conexiones y son estas las que determinan el funcionamiento de la red.

- Capa de salida: Recibe la información de la capa oculta y transmite la respuesta al mundo exterior.

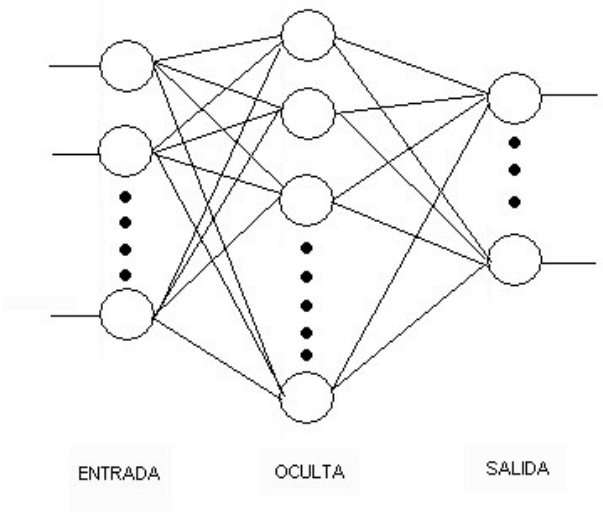


Figura 2.12. Un ejemplo sencillo de una red con tres capas y una conexión total entre neuronas.

Las conexiones entre una capa y otra pueden ser totales, es decir, todas las neuronas se conectan con todas las neuronas de la capa siguiente, o parciales, en las cuales una neurona se conecta con sólo algunas de las neuronas de la capa siguiente. La cantidad de capas, como la cantidad de neuronas por capa y las conexiones entre las neuronas definen lo que se conoce como arquitectura de la red neuronal.

Además del número de capas de una red, en función de como se interconectan unas capas con otras, podemos hablar de redes recursivas (feed-back) y redes no recursivas o redes en cascada (feed-forward). En las redes en cascada la información fluye en un único sentido de una capa a otra (desde la capa de entrada a las capas ocultas y de éstas a la capa de salida), y además, no se admiten conexiones entre neuronas de la misma capa (conexiones intracapa). En las redes recursivas la información puede volver a lugares por los que ya había pasado, formando bucles, y además, se admiten las conexiones intracapa, incluso de una neurona consigo misma.

La elección del número de capas, como del número de neuronas por capa y de cómo se conectan estas, es determinante para la solución de un problema. Una mala elección de la arquitectura de la red, puede causar que la misma no obtenga resultados aceptables.

En las próximas secciones se darán un detalle de las arquitecturas clásicas, de su estructura y de los tipos de problema que pueden resolver. En la bibliografía existente se pueden encontrar las estructuras que se comentan en las próximas secciones, y otras, mucho más detalladas (Freeman & Skapura 1991; Hassoun 1995; Isasi Viñuela & Galván León 2004; Skapura 1995).

2.5.1. Perceptrón

Introducción:

El perceptrón fue inventado por el psicólogo Frank Rosenblatt en el año 1957 (Rosemblatt 1962). Su intención era ilustrar algunas propiedades fundamentales de los sistemas inteligentes en general, sin entrar en mayores detalles con respecto a condiciones específicas y desconocidas para organismos biológicos concretos. Rosenblatt creía que la conectividad existente en las redes biológicas tiene un elevado porcentaje de aleatoriedad y que la herramienta de análisis más apropiada era la teoría de probabilidades. Esto lo llevó a una teoría de separabilidad estadística que utilizaba para caracterizar las propiedades más visibles de estas redes de interconexión ligeramente aleatorias.

Rosenblatt demostró que, dadas dos clases linealmente separables, el perceptrón era capaz de aprender en un *número finito de* pasos el conocimiento necesario para realizar dicha tarea correctamente. Además, este resultado era independiente de los valores de pesos de los arcos utilizados al comienzo de este proceso. En esencia, el entrenamiento implicaba un proceso de refuerzo mediante el cual la salida de las unidades *A* se incrementaba o se decrementaba dependiendo de si las mismas contribuían o no a las respuestas correctas del perceptrón para una entrada dada. Se aplicaba una entrada, y el estímulo se propagaba a través de las capas hasta que se activase una unidad de respuesta. Si se había activado la unidad de respuesta correcta, se incrementaba la salida

de las unidades *A* que hubieran contribuido. Si se activaba una unidad *R* incorrecta, se hacía disminuir la salida de las unidades *A* que hubiesen contribuido.

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

Arquitectura de la red:

El perceptrón está formado por una única neurona artificial como se muestra en la Figura 2.13.

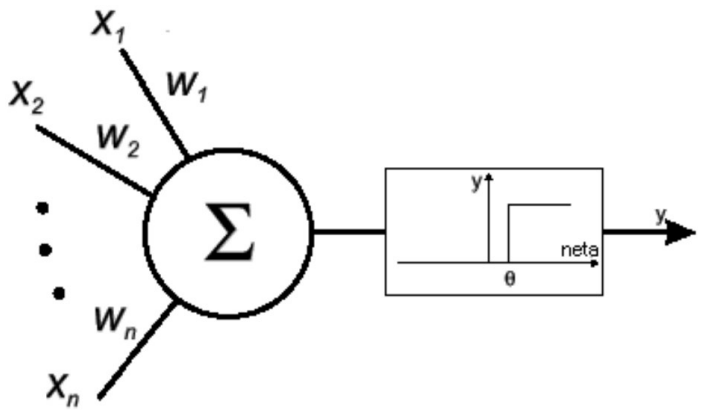


Figura 2.13. Arquitectura del Perceptrón. Cuando el estímulo recibido supera un cierto umbral, la salida de la neurona es, en caso contrario, es cero.

La única neurona de salida del perceptrón realiza la suma ponderada de todas sus entradas, y la salida definitiva será el resultado de aplicar una función de salida al nivel de activación. La regla de decisión es responder +1 si el patrón presentado pertenece a la clase A, o 0 si el patrón pertenece a la clase B, la salida depende de la entrada neta.

$$f(neta,\theta)=\begin{cases}1 & si \quad neta \geq \theta \\ 0 & si \quad neta < \theta\end{cases}$$

donde θ es el umbral y *neta* el nivel de activación calculado como la suma ponderada de las entradas:

$$neta=\sum_{i=1}^{n} x_i w_i \tag{2.2}$$

El perceptrón separa las regiones por un hiperplano cuya ecuación queda determinada por los pesos de las conexiones y el valor umbral de la función de activación de la neurona. Esto implica que sólo puede ser utilizado cuando se trate de separación lineal de dos clases.

Por ejemplo, el perceptrón de la Figura 2.14 se comporta como la función lógica AND.

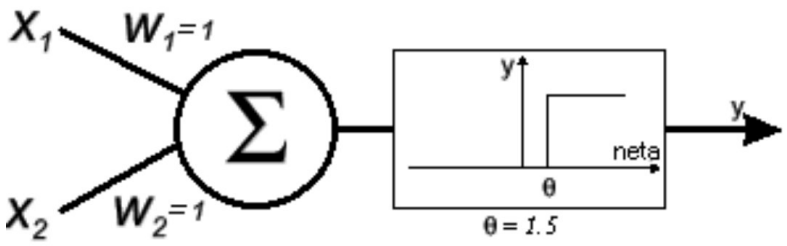


Figura 2.14. Perceptrón que resuelve la función lógica AND.

La función discriminante lineal utilizada en este ejemplo es:

$$x_1 + x_2 = 1{,}5$$

como puede verse en la Figura 2.15.

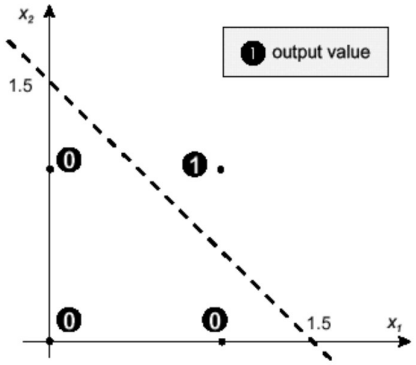


Figura 2.15. Función discriminante utilizada por el perceptrón que resuelve la función lógica AND.

El perceptrón utiliza aprendizaje supervisado, es decir necesita conocer durante el entrenamiento los valores esperados para cada una de las entradas presentadas. Si las muestras utilizadas para el aprendizaje son lo suficientemente representativas del problema, la neurona será capaz de aprender la función discriminante lineal correspondiente.

Para comprender el algoritmo de entrenamiento del perceptrón es necesario analizar la manera de obtener el estímulo total de la neurona para cada uno de los patrones de entrada. Como se dijo anteriormente, este valor se obtiene como resultado del producto interior entre el vector de entrada y el vector de pesos (Ecuación (2.2)).

Cuando la entrada neta supera un cierto umbral, la salida de la neurona será 1, en caso contrario será 0.

El entrenamiento del perceptrón se realiza ingresando los patrones iterativamente y realizando pequeñas modificaciones sobre el vector de pesos en aquellas situaciones en que la respuesta obtenida no sea la esperada. Si se trata de un problema de dos clases

linealmente separables y las modificaciones fueron realizadas en la dirección correcta, la neurona terminará aprendiendo la respuesta esperada.

Para saber como modificar el vector de pesos en cada iteración es necesario analizar el vector proyección del patrón de entrada X sobre el vector de pesos W (Figura 2.16).

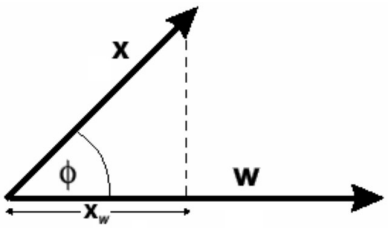


Figura 2.16. Proyección de un vector de entrada sobre el vector de pesos

Por definición el vector proyección se calcula de la siguiente forma:

$$x_w = \mid X \mid .\cos(\phi)$$

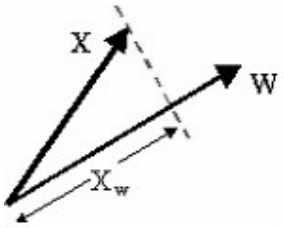
Dado que la entrada neta también puede expresarse como:

$$neta = X \,.\, W = \mid X \mid . \mid W \mid .\cos(\phi)$$

el vector proyección puede expresarse de la siguiente forma:

$$X_w = \frac{X.W}{\mid W \mid}$$

La figura 2.17 ilustra la situación de un vector de entrada *X* que cumple con la función discriminante indicada por el perceptrón.



$$X_w = \frac{\theta}{|W|}$$

$$W.X = \theta$$

Figura 2.17. El patrón de entrada produce una entrada neta igual al umbral esperado

Como puede observarse en la figura 2.18, aquellos patrones que formen un ángulo mayor con el vector de pesos, producirán una entrada neta inferior al umbral; mientras que los que formen un ángulo menor generarán una entrada neta superior al umbral.

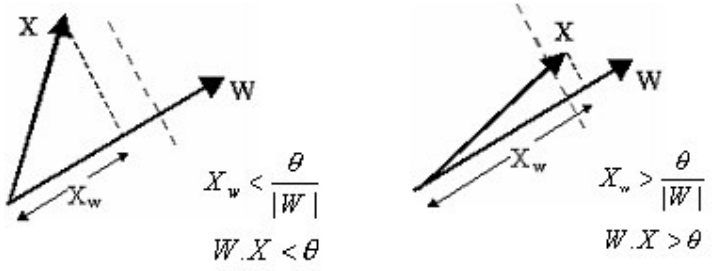


Figura 2.18. A la izquierda se observa un patrón que produce una entrada neta inferior al umbral mientras que a la derecha ocurre todo lo contrario.

Por lo tanto, durante el aprendizaje, los patrones que no posean la entrada neta adecuada deberán acercar o alejar el vector de pesos según corresponda.

A continuación se indica el pseudocódigo del algoritmo de entrenamiento del perceptrón.

```
Inicializar el vector de pesos W con valores random.
Mientras no se clasifiquen todos los patrones correctamente
  Para cada uno de los patrones de entrada
      Ingresar el patrón Xi a la red.
      Si fue clasificado incorrectamente
            Si esperaba obtener W. Xi > θ y no lo logró,
                  "acercar" el vector W al vector Xi.
            Fin si
            Si esperaba obtener W. Xi < θ y no lo logró
                  "alejar" el vector W al vector Xi.
            Fin si
      Fin si
  Fin Para
Fin mientras
```

El ajuste del vector de pesos debe realizarse de la siguiente forma (Figura 2.19):

a) Si $W.X < \theta$ no es el valor esperado entonces acercar W a X sumando al W actual una fracción del vector de entrada. Es decir que $w' = w + \alpha\ x$.

b) Si $W.X > \theta$ no es el valor esperado entonces aleje W a X restando al W actual una fracción del vector de entrada. Es decir que $w' = w - \alpha\ x$.

Figura 2.19: Modificación del vector de pesos. A la izquierda cuando $W.X < \theta$. A la derecha cuando $W.X > \theta$.

En ambos casos α es un número real perteneciente al intervalo (0,1] y representa la velocidad de aprendizaje ya que determina la fracción del vector de entrada que se utilizará para modificar el vector de pesos. Nótese que si su valor es muy pequeño las modificaciones también lo serán, incrementando el tiempo de convergencia del algoritmo. Por el contrario, si se trata de un valor cercano a 1, las modificaciones serán muy abruptas y podrán llevar al vector de pesos a realizar oscilaciones innecesarias evitando que el algoritmo termine.

Por lo tanto, si se denomina T_i al valor esperado para el patrón X_i e Y_i el valor obtenido del perceptrón para dicho patrón, el algoritmo de entrenamiento puede reescribirse de la siguiente forma:

```
Seleccionar los valores de α y θ.
Inicializar el vector de pesos W con valores random.
Mientras no se clasifiquen todos los patrones correctamente
  Para cada uno de los patrones de entrada
      Ingresar el patrón Xi a la red.
      Si Ti ≠ Yi
            W = W +  α (Ti  - Yi)  Xi
      Fin si
   Fin para
Fin mientras
```

Como ejemplo de funcionamiento de una red neuronal tipo perceptrón, se detallará la manera de entrenar una neurona para resolver el problema de la función lógica AND.

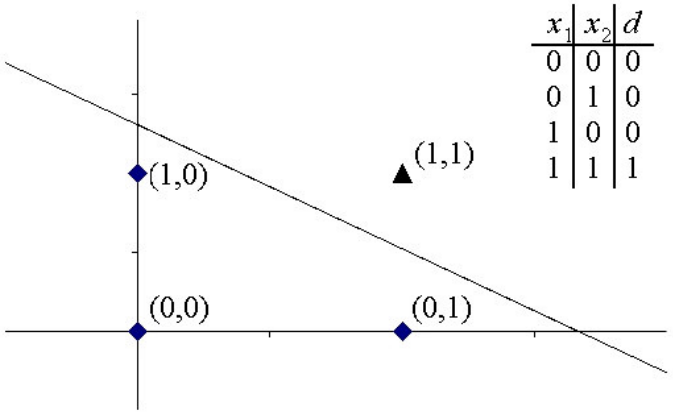
La arquitectura de la neurona a utilizar es la indicada en la figura 2.14. Ahora es el algoritmo de entrenamiento el que debe determinar los valores del vector W. La tabla 2.1 muestra las cuatro iteraciones que fueron necesarias realizar para alcanzar la función discriminante adecuada. En este caso se ha utilizado $\alpha = 0{,}3$, $\theta = 1{,}5$, $W_1 = 0$ y $W_2 = 0{,}25$.

Tabla 2.1. Valores de los pesos, antes y después de la actualización, salidas obtenidas y esperadas y entrada al perceptrón para cada paso de un entrenamiento.

Paso	Entrada		Salida esperada	Pesos iniciales		Salida obtenida	Actualización de pesos	
	x_1	x_2	d	w_1	w_2	y	w_1	w_2
1	0	0	0	0	0,25	0	0	0,25
2	1	0	0	0	0,25	0	0	0,25
3	0	1	0	0	0,25	0	0	0,25
4	1	1	1	0	0,25	0	0,3	0,55
5	0	0	0	0,3	0,55	0	0,3	0,55
6	1	0	0	0,3	0,55	0	0,3	0,55
7	0	1	0	0,3	0,55	0	0,3	0,55
8	1	1	1	0,3	0,55	0	0,6	0,85
9	0	0	0	0,6	0,85	0	0,6	0,85
10	1	0	0	0,6	0,85	0	0,6	0,85
11	0	1	0	0,6	0,85	0	0,6	0,85
12	1	1	1	0,6	0,85	0	0,9	1,15
13	0	0	0	0,9	1,15	0	0,9	1,15
14	1	0	0	0,9	1,15	0	0,9	1,15
15	0	1	0	0,9	1,15	0	0,9	1,15
16	1	1	1	0,9	1,15	1	0,9	1,15

Luego del entrenamiento el vector de pesos es el siguiente: $W_1 = 0{,}9$ y $W_2 = 1{,}15$. Por lo tanto la función discriminante representada por el perceptrón, y que se puede observar en la figura 2.20, será:

$0{,}9\ X_1 + 1{,}15\ X_2 = 1{,}5$



x_1	x_2	d
0	0	0
0	1	0
1	0	0
1	1	1

Figura 2.20. La recta que separa las dos clases de entrada para la función lógica AND.

De lo antes expuesto se deduce que resulta complejo determinar el umbral adecuado a cada tipo de problema. Esto se resuelve incorporando el cálculo de este valor al proceso de aprendizaje a través de una entrada adicional cuyo valor es siempre 1 como se observa en la figura 2.21.

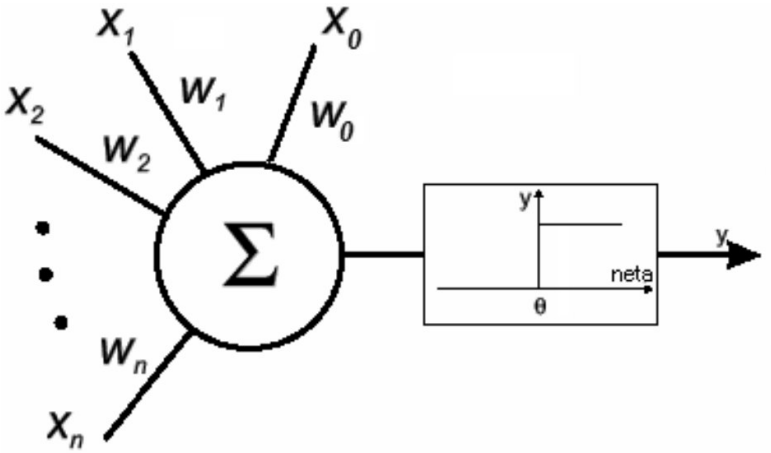


Figura 2.21. Arquitectura del un perceptrón con una entrada adicional $x_0 = 1$.

Por lo tanto la función discriminante a utilizar será:

$$\sum_{i=0}^{n} x_i w_i = 0$$

La tabla 2.2 muestra las modificaciones realizadas sobre el vector de pesos para entrenar un perceptrón que resuelva la función lógica AND fijando también el valor de W_0. Se ha utilizado $\alpha = 0.3$, $W_0 = 0$, $W_1 = 0$ y $W_2 = 0.25$.

Tabla 2.2. Valores de los pesos, antes y después de la actualización, salidas obtenidas y esperadas y entrada al perceptrón para cada paso del entrenamiento.

Paso	Entrada			Salida esperada	Pesos iniciales			Salida obtenida	Actualización de pesos		
	x_0	x_1	x_2	d	w_0	w_1	w_2	y	w_0	w_1	w_2
1	1	0	0	0	0	0	0,25	1	-0,3	0	0,25
2	1	1	0	0	0		0,25	0	-0,3	0	0,25
3	1	0	1	0	0		0,25	0	-0,3	0	0,25
4	1	1	1	1	0		0,25	0	0	0,3	0,55
5	1	0	0	0	0,3		0,55	1	-0,3	0,3	0,55
6	1	1	0	0	0,3		0,55	1	-0,6	0	0,55
7	1	0	1	0	0,3		0,55	0	-0,6	0	0,55
8	1	1	1	1	0,3		0,55	0	-0,3	0,3	0,85
9	1	0	0	0	0,6		0,85	0	-0,3	0,3	0,85
10	1	1	0	0	0,6		0,85	1	-0,6	0	0,85
11	1	0	1	0	0,6		0,85	1	-0,9	0	0,55
12	1	1	1	1	0,6		0,85	0	-0,6	0,3	0,85
13	1	0	0	0	0,9		1,15	0	-0,6	0,3	0,85
14	1	1	0	0	0,9		1,15	0	-0,6	0,3	0,85
15	1	0	1	0	0,9		1,15	1	-0,9	0,3	0,55
16	1	1	1	1	0,9		1,15	0	-0,6	0,6	0,85
17	1	0	0	0	0,6		0,85	0	-0,6	0,6	0,85
18	1	1	0	0	0,6		0,85	1	-0,9	0,3	0,85
19	1	0	1	0	0,6		0,85	0	-0,9	0,3	0,85
20	1	1	1	1	0,6		0,85	1	-0,9	0,3	0,85
21	1	0	0	0	0,9		1,15	0	-0,9	0,3	0,85
22	1	1	0	0	0,9		1,15	0	-0,9	0,3	0,85
23	1	0	1	0	0,9		1,15	0	-0,9	0,3	0,85
24	1	1	1	1	0,9		1,15	1	-0,9	0,3	0,85

Luego del entrenamiento el vector de pesos es el siguiente: W_0=-0,9, W_1 = 0,3 y W_2 = 0,85. Por lo tanto la función discriminante representada por el perceptrón será:

$-0{,}9\ X_0 + 0{,}3\ X_1 + 0{,}85\ X_2 = 0$

como se observa en la figura 2.22.

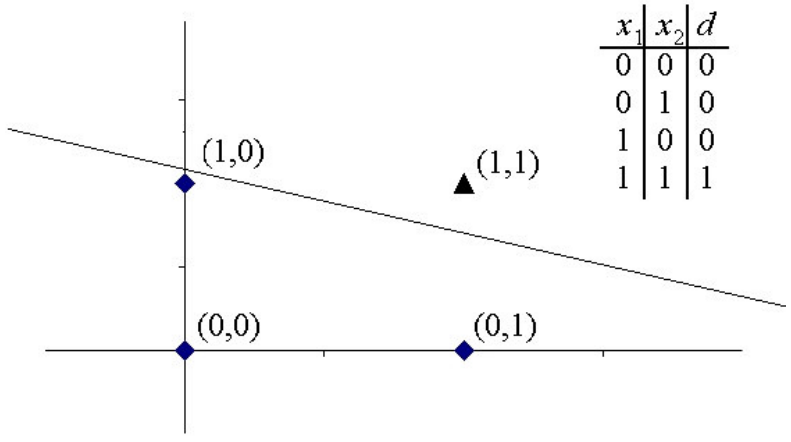


Figura 2.22. La recta que separa las dos clases de entrada para la función lógica AND, al utilizar la entrada adicional x_0.

Perceptrón multicapa:

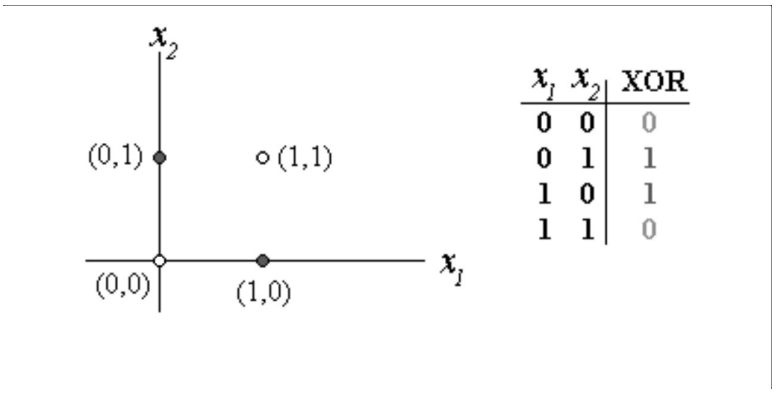
Con un único perceptrón sólo puede obtenerse una función discriminante lineal. Por este motivo, no posee la capacidad para resolver la función lógica XOR (Figura 2.23).

La solución a problemas que no son linealmente separables se puede encontrar descomponiendo el espacio de entrada en varias regiones y utilizando varias funciones discriminantes lineales para separarlas. En este caso, cada hiperplano se corresponde con un perceptrón y la delimitación de grupos de patrones estará dada por una segunda capa de perceptrones cuyas entradas serán las salidas de las neuronas anteriores.

De esta forma se obtiene una red neuronal feedforward conocida como perceptrón multicapa. Si bien esta arquitectura es capaz de resolver problemas que no son linealmente separables, como el problema del XOR, su entrenamiento es difícil de generalizar. Esto se debe a que la tanto la descomposición del problema original como la interpretación de cada una de las funciones discriminantes debe realizarse de manera individual (Figura 2.24).

Es importante aclarar que para obtener los valores de los pesos de los arcos indicados en la figura 2.14 es necesario realizar el entrenamiento de cada uno de los perceptrones por separado. Nótese que mientras la neurona marcada con (1) realiza la función OR entre los valores de entrada mediante la función discriminante p1+p2-0,5=0 y la neurona señalada con (2) realiza un AND de la siguiente forma p1+p2-1,5=0. Una vez logrado esto, la neurona de salida combina estas respuestas para cumplir con la clasificación del XOR.

Las capacidades del perceptrón con una, dos y tres capas y con una única neurona en la capa de salida se muestran en la figura 2.25.



x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Figura 2.23. El problema de la función lógica XOR. No es posible utilizar una única recta que separe los puntos (0,0) y (1,1) de los puntos (0,1) y (1,0).

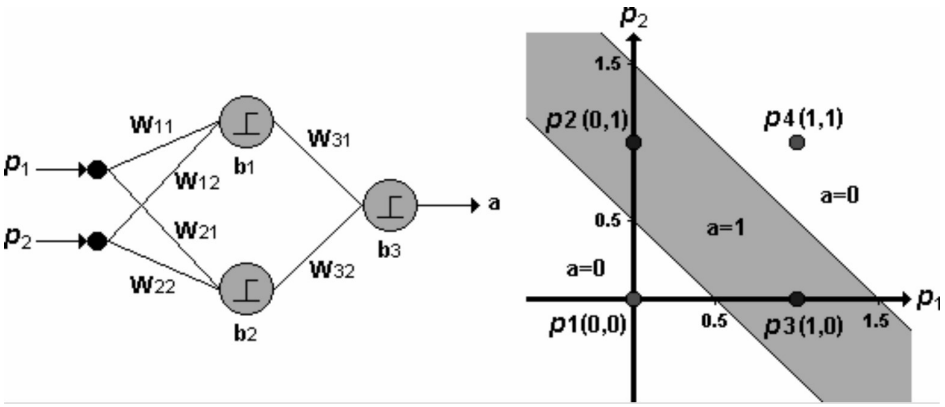


Figura 2.24. A la izquierda se muestra la estructura de la red perceptrón multicapa, que es capaz de resolver el problema de la función lógica XOR. Los valores utilizados son w_{11}=1; w_{12}=1; w_{21}=1; w_{22}=1; w_{31}=1; w_{32}=-1.5; b_1=-0.5; b_2=-1.5; b_3=-0.5. A la derecha se ilustra gráficamente su funcionamiento.

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

Figura 2.25. En la segunda columna se muestra el tipo de región de decisión que se puede formar con cada una de las configuraciones, en la siguiente se indica el tipo de región que se formaría para el problema de la función lógica XOR, en las dos últimas columnas se muestran las regiones formadas para resolver el problema de clases mezcladas y las formas más generales para cada uno de los casos.

2.5.2. ADALINE y el combinador Lineal

Introducción:

El Perceptrón si bien posee la capacidad de aprender en base a ejemplos, sólo puede ser utilizado para clasificar o separar los datos de entrada en dos clases diferentes. Sin embargo, existen un gran número de problemas abordables desde la perspectiva del aprendizaje basado en ejemplos que no se reducen a tareas de clasificación. La

característica de clasificador del Perceptrón viene dada por la naturaleza de sus salidas, las cuales utilizan una función umbral que transforma la suma ponderada de las entradas, que es un valor real, en una salida binaria limitando su uso. Si se suprimiera esta función umbral, se podría contar con una neurona capaz de aprender a aproximar una función cualquiera definida por un conjunto de datos de entrada.

Los datos de entrenamiento constituyen el conjunto de pares de valores formados por el vector de entrada y su salida asociada $P = \{(x_1, d_1), (x_2, d_2), \ldots, (x_n, d_n)\}$.

El comportamiento que se espera aprender es una función de aproximación de la forma:

$$F(x_i) = y_i \, , \forall p \in P$$

Es decir, habrá que buscar la función $F(x)$ tal que aplicada a cada una de las entradas x_i del conjunto de aprendizaje P produzca la salida y_i correspondiente en dicho conjunto de aprendizaje.

Otro aspecto a tener en cuenta es el uso de una estrategia de entrenamiento que modifique los pesos de manera proporcional a los errores observados. Es importante remarcar la diferencia que tiene este enfoque con el Perceptrón que utiliza sólo la dirección del error detectado.

En el año 1960, Widrow y Hoff (Widrow 1960) propusieron un sistema de aprendizaje que sí tuviera en cuenta el error producido y diseñaron lo que denominaron *ADAptive Linear Neuron* o en su acrónimo, ADALINE. Su estructura, representada por la figura 2.26, puede dividirse en dos partes:

- Un elemento combinador adaptativo también denominado Combinador Lineal que permite obtener un valor real para cada entrada.

- En caso de ser necesario puede incorporarse a la salida una función de activación bipolar que convierta el valor real anterior en sólo dos estados posibles: 1 si la salida es positiva y -1 si no.

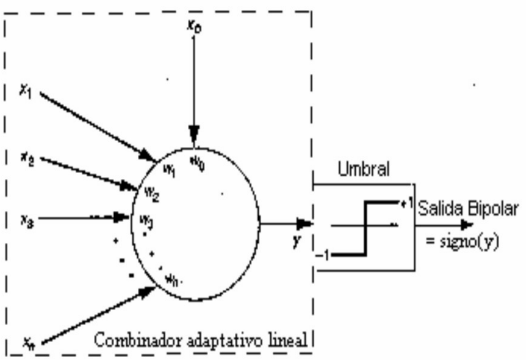


Figura 2.26. Estructura del ADALINE.

Trabajando sólo con el Combinador Lineal, la salida real obtenida será la suma ponderada de las entradas:

$$y = \sum_{k=0}^{L} x_k w_k$$

El aprendizaje en este caso incluye la diferencia entre el valor real obtenido para un patrón de entrada X_k y el valor esperado para dicho patrón, d_k. A esta regla de aprendizaje se la conoce como Regla Delta.

Es importante notar la gran diferencia que existe entre la estrategia de aprendizaje del Perceptrón y la Regla Delta. En el primer caso se utiliza la salida de una función umbral binaria mientras que en l Regla Delta se utiliza directamente la salida real, sin pasarla por ninguna función umbral. El objetivo es obtener un valor que se asemeje a la salida esperada.

Dado que no siempre será posible conseguir una salida exacta para todas las entradas, el objetivo es minimizar la desviación cometida por el Combinador Lineal; esto es, minimizar el error cometido sobre el conjunto de patrones de entrada, lo que lleva a elegir una medida del error global de la neurona.

Lo habitual es utilizar como medida de error global el error cuadrático medio, pero esto es dependiente del problema pudiéndose utilizar otras expresiones de error.

$$\xi = \frac{1}{L}\sum_{k=1}^{L}(d_k - y_k)^2$$

donde y_k es el valor de salida del adaline para el k-ésimo patrón de entrada y L es la cantidad total de patrones.

Al ser esta una medida de error global, la regla intentará minimizar este valor para todos los elementos del conjunto de patrones utilizados en el aprendizaje. Analíticamente puede hallarse el vector de pesos ideal igualando las derivadas parciales a cero para obtener el mínimo de la función; no obstante la realización de estos cálculos implica disponer de todos los patrones desde el inicio. Una manera alternativa es recurrir a un proceso iterativo en el que se van presentando lo patrones uno a uno y se van modificando los pesos de la neurona según la *regla del descenso del gradiente*.

Dado que el vector de pesos de este procedimiento es variable en el tiempo lo escribimos como una función explícita del paso temporal, t. En cada paso, el próximo vector de pesos se calcula según:

$$w(t+1) = w(t) + \Delta w(t)$$

en donde $\Delta w(t)$ es el cambio que sufre w en el instante de tiempo t. Como se está buscando la dirección del descenso más pronunciado en cada punto de la superficie, es necesario calcular el gradiente de la superficie –que proporciona la dirección de la pendiente más pronunciada hacia arriba–. La dirección opuesta es la que se utiliza para los cálculos. Para obtener una magnitud del cambio se multiplica por una constante positiva apropiada μ. Así se obtiene la siguiente expresión:

$$w(t+1) = w(t) - \mu \vec{\nabla} \xi(w(t)) \tag{2.3}$$

Dado que se trata de un aprendizaje gradual donde los patrones son presentados uno a uno, no es posible obtener el gradiente correspondiente a la superficie de error sino una aproximación del mismo para el patrón actual, es decir:

$$\nabla < \varepsilon_k^2 > \approx \nabla \varepsilon_k^2(t) = -2\varepsilon_k(t) f_k(neta_k) x_k$$

Para evaluar el gradiente en un patrón determinado será necesario derivar la expresión del error respecto de cada una de las componentes del vector de pesos de la siguiente forma:

$$\nabla \varepsilon_k^2(t) = \frac{\partial \varepsilon_k^2}{\partial w} = \left[\frac{\partial (d_k - y_k)^2}{\partial w_0}; \quad ...; \quad \frac{\partial (d_k - y_k)^2}{\partial w_n} \right]$$

aplicando la regla de la cadena:

$$\nabla \varepsilon_k^2(t) = \frac{\partial \varepsilon_k^2}{\partial w} = \left[-2(d_k - y_k) \frac{\partial y_k}{\partial w_0}; \quad ...; \quad -2(d_k - y_k) \frac{\partial y_k}{\partial w_n} \right]$$

$$\nabla \varepsilon_k^2(t) = \frac{\partial \varepsilon_k^2}{\partial w} = -2e_k \left[\frac{\partial y_k}{\partial w_0}; \quad ...; \quad \frac{\partial y_k}{\partial w_n} \right]$$

La derivada de la función de activación respecto de cada uno de los vectores de pesos es de la siguiente forma:

$$\frac{\partial y_k}{\partial w_j} = \frac{\partial f_k}{\partial (neta_k)} \frac{\partial (neta_k)}{\partial w_j}$$

donde

$$\frac{\partial f_k}{\partial (neta_k)} = f'(neta_k)$$

y

$$\frac{\partial (neta_k)}{\partial w_j} = \frac{\partial (\sum_{j=0}^{L} w_j x_j)}{\partial w_j} = x_j$$

por lo tanto

$$\nabla \varepsilon_k^2(t) = \frac{\partial \varepsilon_k^2}{\partial w} = -2e_k \; f'(neta_k) \left[x_{1k}, \quad x_{2k}, \quad ..., \quad x_{nk} \right] = -2 \; e_k \; f'(neta_k) \; x_k$$

La ecuación (2.3) representa la regla de aprendizaje del adaline. Esta fórmula ya ha sido generalizada para el caso de una función de activación diferenciable cualquiera:

$$w(t+1) = w(t) + 2\mu(d_k - y_k)\, f_k^{'}(neta_k)\, x_k$$

donde μ representa la velocidad de aprendizaje, x_k el patrón de entrada que se quiere ajustar, $neta_k$ es la entrada neta correspondiente a x_k, d_k es la salida esperada para x_k y y_k la salida obtenida para x_k.

Como resumen de todo lo antes expuesto, a continuación se indica el algoritmo utilizado para realizar el entrenamiento utilizando la Regla Delta.

Seleccionar el valor de μ
Inicializar el vector de pesos W con valores random.
Mientras el error no sea aceptable
  Para cada uno de los patrones de entrada
    Aplicar el vector de entrada x_k,
    Calcular el error cuadrático usando el W actual

$$\varepsilon_k^2(t) = (d_k - y_k)^2$$

    Calcular el gradiente utilizando

$$\nabla < \varepsilon_k^2 > \approx \nabla \varepsilon_k^2(t) = -2\varepsilon_k(t)\, f_k^{'}(neta_k)\, x_k$$

    Actualizar el vector de pesos

$$w(t+1) = w(t) + 2\mu\varepsilon_k\, f_k^{'}(neta_k)\, x_k$$

  Fin Para
Fin Mientras

El parámetro μ determina la estabilidad y la velocidad de convergencia del vector de pesos hacia el valor de error mínimo. Dado que se ha utilizado una aproximación del gradiente, el camino que sigue el vector de pesos al bajar por la superficie de pesos hacia el mínimo no será una curva suave. Además a medida que W se aproxime al óptimo, la reducción sucesiva del error producirá una disminución en el módulo del gradiente y por lo tanto el paso de actualización utilizado será cada vez más pequeño.

Como ya se mencionó anteriormente, a diferencia del perceptrón, la Regla Delta no utiliza la función umbral a la salida de la neurona. El único requisito establecido para esta función es que desea derivable. Esto quiere decir que podría utilizarse una función no acotada llevando a la expresión del error a valores muy grandes. Por tal motivo, resulta de fundamental interés seleccionar un valor adecuado para el parámetro μ, que representa la velocidad de aprendizaje.

La experiencia indica que incrementar el valor de μ cuando los cambios en el vector comienzan a hacerse demasiado pequeños, puede acelerar la convergencia al mínimo buscado. Sin embargo ha de tenerse en cuenta que un valor más grande del debido ocasionará que los pesos vayan "dando saltos en torno al fondo" de la superficie, con lo cual el error no convergerá nunca. La figura 2.27 muestra un ejemplo del aspecto que puede tener el camino de búsqueda del error mínimo.

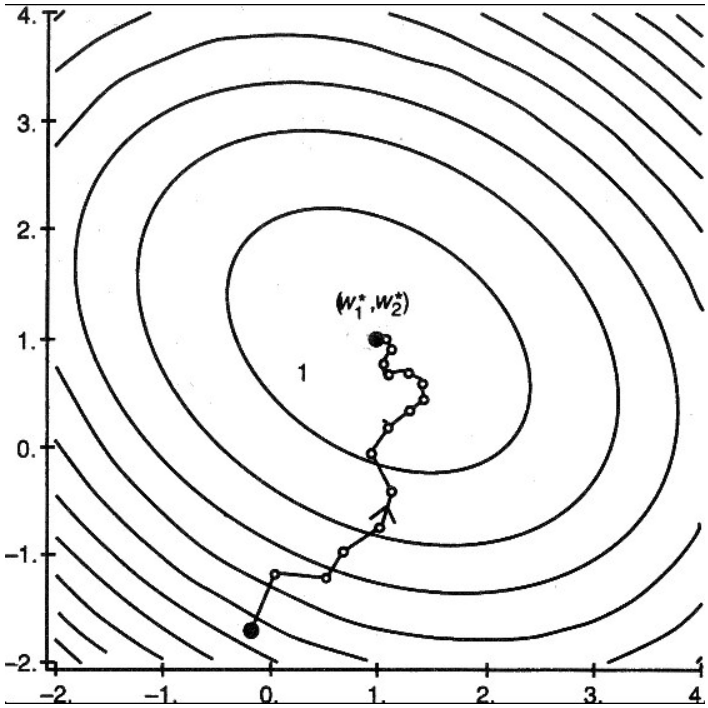


Figura 2.27. Una posible ruta hipotética que sigue el vector de pesos en la búsqueda del error mínimo utilizando la Regla Delta para una función de activación lineal. Como se puede observar, dicha ruta no es una curva suave, porque se está aproximando el gradiente en cada punto. Se puede apreciar también que el tamaño del paso se vuelve cada vez más pequeño a medida que se aproxima al error mínimo.

2.5.3. Un problema de clasificación

Esta sección tiene por objetivo resumir y ejemplificar lo antes desarrollado, aplicando las arquitecturas y reglas de aprendizajes hasta aquí descriptas, para resolver un mismo problema. Este ejercicio permitirá comparar y comprender las capacidades y limitaciones de una neurona artificial en base a la función de activación seleccionada y al mecanismo de aprendizaje elegido. Esto resulta de fundamental importancia para entender el comportamiento de varias neuronas artificiales dentro de una misma red.

Por simplicidad y para tener la posibilidad de graficar las respuestas obtenidas, se propondrá como ejemplo de clasificación la separación de los siguientes conjuntos de puntos del plano:

Clase A = {(1,4), (2,2), (2,1)}

Clase B = { (4,4), (5, 4), (6, 3), (2.5, 3.5)}

representados en la figura 2.28.

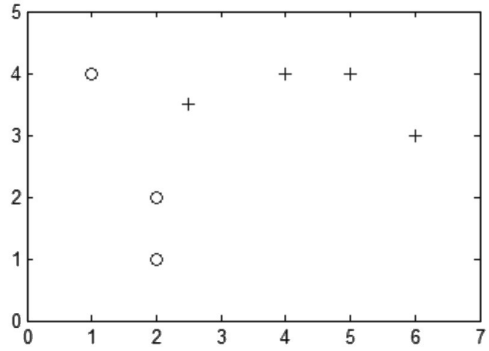


Figura 2.28. Los círculos representan a los tres puntos de la clase A y las cruces representan a los cuatro puntos de la clase B.

Como puede observarse su separación es simple y cualquiera de las neuronas descriptas hasta ahora posee capacidad para resolverlo rápidamente. Sin embargo, lo importante del ejemplo es analizar el objetivo del entrenamiento de cada una de las neuronas artificiales.

Las respuestas obtenidas se pueden observar en la figura 2.29.

Puede verse que cada una de ellas realiza, para el mismo problema, una interpretación diferente con respecto a la manera de ubicar la función discriminante:

a) Perceptrón: su algoritmo de entrenamiento ubica un hiperplano de manera tal que patrones de clases distintas poseen signos diferentes al ser proyectados sobre su superficie.

b) Combinador Lineal: Busca reducir la distancia entre el hiperplano y cada uno de los patrones de entrada. Esto se encuentra relacionado con el hecho de que el combinador lineal no fue pensado como un clasificador sino como un aproximador lineal de los patones de entrada con capacidad para retornar valores reales.

c) Neurona no lineal: En particular se trata de una neurona con salida sigmoide entrenada a través de la regla delta, tal como fue descripta para el combinador lineal pero utilizando la siguiente la función de activación:

$$f(neta) = \frac{2}{1+e^{-2\,neta}} - 1$$

Se trata de una función derivable y acotada en el intervalo (-1,1). En caso de que se requiera utilizar esta neurona para clasificación se agrega la función umbral como fue descripta en el ADALINE.

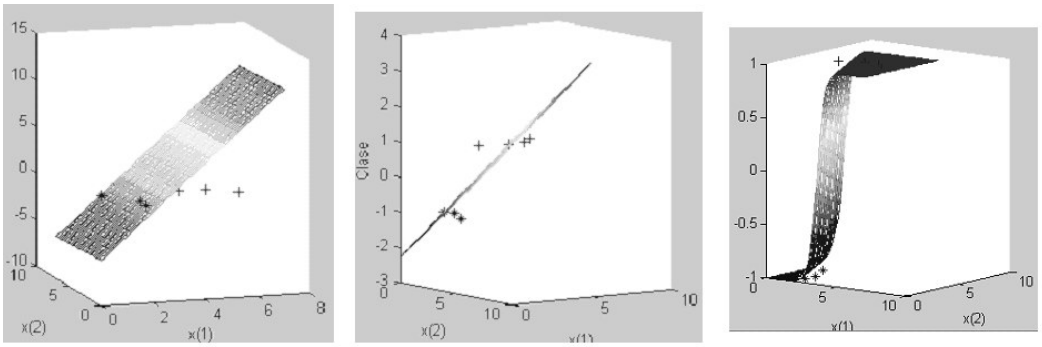


Figura 2.29. En a) se observa la respuesta del perceptrón, en b) la respuesta del combinador lineal y en c) la respuesta de una neurona no lineal.

Es importante notar que si bien las tres neuronas artificiales son capaces de separar correctamente las dos clases, la única que busca resolver un problema de clasificación es el perceptrón a través de su función umbral. Las soluciones b) y c), como se encuentran basadas en la regla delta, buscan que la función discriminante se ubique en una posición, dentro del espacio de los patrones de entrada, que minimice una expresión de error determinada.

Esta aproximación es lo que provoca que el algoritmo de entrenamiento del perceptrón logre obtener una respuesta en un tiempo mucho menor que la regla delta. Esto se debe a que sólo requiere que los patrones queden ubicados del lado adecuado de la función discriminante; en cambio la regla delta exige que dicha función se ubique en una posición específica con respecto a todos los patrones de entrada.

El objetivo de estas dos estrategias es totalmente diferente.

Existe otro aspecto a tener en cuenta que tiene que ver con la distancia entre las clases. Para esto, se incorporará a la clase B el patrón (1.6, 3) como lo muestra la figura 2.30. Puede observarse que ahora, la zona donde debe ubicarse la función discriminante es mucho más estrecha que antes. Esto dificulta la tarea independientemente de la estrategia de entrenamiento a utilizar.

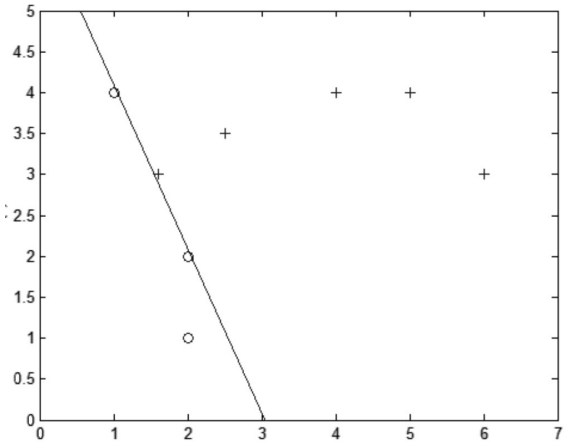


Figura 2.30. En este problema si bien las dos clases son linealmente separables, la posición de la función discriminante está determinada únicamente por tres de los ocho patrones de entrada.

La gráfica de la recta que aparece en la figura 2.30 corresponde a la respuesta de la neurona no lineal detallada anteriormente entrenada con los ocho patrones utilizando la regla delta. La figura 2.31 grafica el resultado de la función de activación. Aquí puede apreciarse la capacidad de una neurona con estas características para aproximar los patrones de las dos clases.

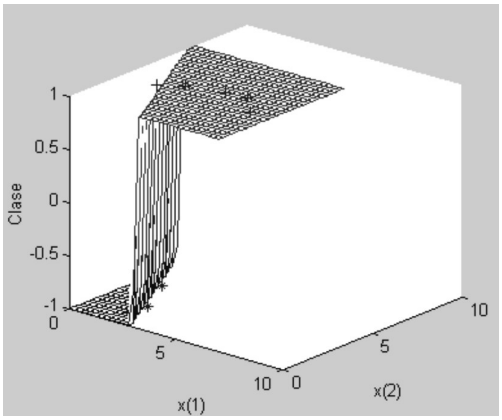


Figura 2.31. Gráfica de la función de activación.

Este comportamiento no puede ser realizado por el combinador lineal ya que si bien, buscará aproximar los patrones, no podrá posicionarse para separarlos adecuadamente.

La gráfica 2.32 muestra la salida del combinador lineal luego de actualizar el vector de pesos durante 30000 iteraciones con una velocidad de aprendizaje de 0,002.

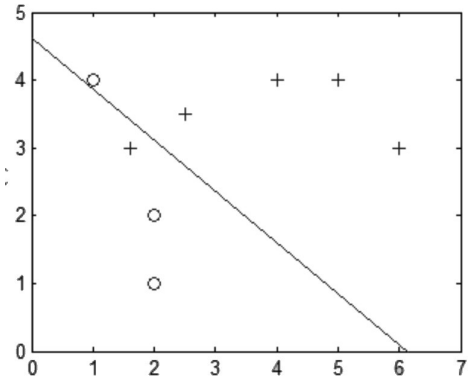


Figura 2.32. Respuesta del combinador lineal.

Finalmente y en relación a la velocidad de aprendizaje puede decirse que si bien la regla delta es muy sensible con respecto a este valor, el perceptrón no lo es menos.

En el caso del perceptrón, las actualizaciones sólo dependen de este parámetro ya que es el que indica la proporción del vector de entrada que modificará al vector de pesos. En cambio en la regla delta, esta proporción está también condicionada por el valor de la derivada de la función de activación. Esto último permite que cuando el vector de pesos se encuentre lejos del óptimo se actualice con un paso más grande que cuando está cerca.

De todas formas, en ambos casos, la elección de la velocidad de aprendizaje es totalmente heurística. Si bien existen otras estrategias basadas en evolución que permiten resolver este problema, su aplicación se encuentra fuera de los alcances de esta tesis.

2.5.4. Backpropagation

Introducción:

Ya se ha visto en este capítulo, cuando se analizó el Perceptrón de Rosenblatt y el ADALINE de Widrow, que utilizando una única neurona sólo es posible contar con una única función discriminante, aspecto que limita fuertemente el tipo de problemas que se pueden resolver. También se expuso cómo es posible combinar perceptrones simples en una red multicapa para resolver satisfactoriamente un problema que no es linealmente separable como la función XOR. Incluso se ha mencionado que una red Perceptrón con dos capas, puede formar cualquier región convexa de decisión, y con tres puede definir cualquier región arbitraria haciendo posible cualquier clasificación.

Sin embargo, aún resta definir un algoritmo de aprendizaje capaz de encontrar los pesos adecuados para este tipo de redes multicapa.

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

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

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

La importancia de este proceso consiste en que, a medida que se entrena la red, las neuronas de las capas intermedias se organizan a sí mismas de tal modo que las distintas neuronas aprenden a reconocer distintas características del espacio total de entrada.

Después del entrenamiento, cuando se les presente un patrón arbitrario de entrada que contenga ruido o que esté incompleto, las neuronas de la capa oculta de la red responderán con una salida activa si la nueva entrada contiene un patrón que se asemeje a aquella característica que las neuronas individuales hayan aprendido a reconocer durante su entrenamiento. Siguiendo el proceso inverso, las unidades de las capas ocultas tienen una tendencia a inhibir su salida si el patrón de entrada no contiene la característica para reconocer, para la cual han sido entrenadas.

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

Arquitectura de la red:

El entrenamiento de una red neuronal multicapa se realiza mediante un proceso de aprendizaje, para realizar este proceso se debe inicialmente tener definida la arquitectura de la red, esto es: número de neuronas en la capa de entrada el cual depende del número de componentes del vector de entrada, cantidad de capas ocultas y número de neuronas de cada una de ellas, número de neuronas en la capa de la salida el cual depende del número de componentes del vector de salida o patrones objetivo y funciones de transferencia requeridas en cada capa. Cada capa tiene su propio vector de pesos. La figura 2.33 ilustra una red backpropagation genérica.

La función de transferencia de cada neurona tanto en la capa de salida como en la capa oculta puede ser cualquier función derivable. Este requisito excluye la función escalón que se utilizó en el Perceptrón pues no es derivable en todo su dominio.

Las funciones más utilizadas son la función sigmoide en la capa oculta y la función Identidad o la función sigmoide en la capa de salida (figura 2.8).

En la capa de salida, la selección de esta función depende de la forma en que se pretende representar los datos de salida. Por ejemplo, si se desea que las unidades de salida sean binarias, se utiliza una función sigmoide, porque esta función limita los valores a obtener y es casi biestable y también es derivable. En otros casos es tan aplicable una función de salida lineal como una sigmoide.

Con respecto a la capa oculta, es importante utilizar una función que además de ser derivable sea acotada para evitar que los valores de actualización crezcan exponencialmente durante el entrenamiento. Por este motivo, no es recomendable el uso de la función identidad.

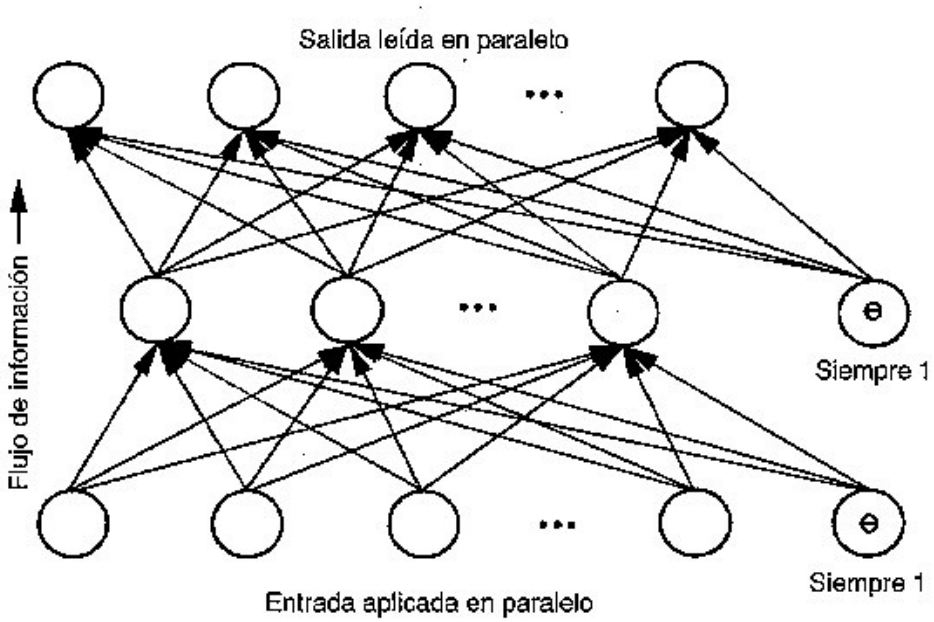


Figura 2.33. Arquitectura de una red neuronal backpropagation con una única capa oculta.

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

Regla Delta Generalizada:

La red backpropagation introduce en su entrenamiento una variante de la *regla delta* vista en la sección anterior, dando lugar a la *regla delta generalizada*. Esta generalización es necesaria debido a la presencia de neuronas ocultas en la red, para las cuales no se conoce el error cometido.

Para indicar los pesos de la arquitectura se utilizará la siguiente notación:

$$w^{capa}_{destino\ origen}$$

El superíndice permite distinguir si se trata de los pesos que van de la capa de entrada a la oculta o de la oculta a la de salida. En el primer caso el superíndice será *h* y en el segundo *o*.

El primer subíndice identifica la neurona a la cual llega el arco y el segundo subíndice indica la neurona de la cual sale el arco.

Por ejemplo: w_{kj}^{o} es el arco que sale de la neurona j y llega a la neurona k. El superíndice o indica que se trata de un arco que va de la capa oculta a la de salida.

Volviendo a la actualización de los pesos según la regla delta generalizada, los que conectan la capa oculta con la capa de salida se modifican según la siguiente ecuación:

$$w_{kj}^{o} = w_{kj}^{o} + \alpha(d_k - y_k) f_k^{o\prime}(neta) i_j \tag{2.4}$$

donde w^{o}_{kj} es el peso que une la *k*-ésima neurona de salida con la *j*-ésima neurona de la capa oculta, d_k es la salida esperada en la k-ésima neurona de salida, y_k es el valor devuelto por la red en la k-ésima neurona de salida, $f_k^{o\prime}(neta)$ la derivada de la función de activación de la capa de salida evaluada en la entrada neta correspondiente (note que, aunque no es habitual, cada neurona podría tener su propia función de activación), i_j es la salida de la *j*-ésima neurona de la capa oculta y α es el parámetro conocido como velocidad de aprendizaje.

Nótese que ahora los subíndices *j* y *k* hacen referencia a neuronas de la capa de salida y no a un patrón determinado.

La actualización de los pesos que conectan la capa de entrada con la capa oculta se realiza con la siguiente ecuación:

$$w_{ji}^{h} = w_{ji}^{h} + \alpha\ f_j^{h\prime}(neta) x_i \sum_{k=1}^{M} (d_k - y_k) w_{kj}^{o} \tag{2.5}$$

donde w^{h}_{ji} es el peso que va desde la i-ésima neurona de la capa de entrada a la j-ésima neurona de la capa oculta, d_k es la salida esperada en la k-ésima neurona de salida, y_k es el valor devuelto por la red en la k-ésima neurona de salida, w^{o}_{ki} es el peso entre la i-ésima neurona de la capa oculta y la k-ésima neurona de la capa de salida, *M* es la cantidad de neuronas de la capa de salida, $f_k^{h\prime}(neta)$ la derivada de la función de activación de la capa de salida evaluada en la entrada neta correspondiente (note que, aunque no es habitual, cada neurona podría tener su propia función de activación), x_i es la salida de la i-ésima neurona de la capa de entrada y α es el parámetro conocido como velocidad de aprendizaje.

Nótese que la sumatoria que aparece en la modificación de cada uno de los pesos ubicados entre las dos primeras capas hace referencia al error ocurrido en la respectiva neurona de la capa oculta.

Algoritmo de entrenamiento:

```
Inicializar al azar todos los vectores de pesos de las
conexiones.
Mientras el error no sea aceptable
Para cada uno de los patrones de entrada
   Presentar a la red el patrón de entrada y calcular la
   salida.
   Para cada neurona j-ésima de la capa de salida
```

```
      Usando las neuronas i-ésimas de la capa oculta
         inmediatamente anterior a las cuales está conectada,
         calcular la entrada total ponderada y la salida de la
         misma.
      Una vez calculadas las actividades de todas las neuronas
         de salida calcular la estimación del error.
      Actualizar los pesos de la capa de salida usando la
         ecuación (2.4).
      Actualizar los pesos de la capa oculta usando la
         ecuación (2.5).
   Fin para
Fin para
Fin Mientras
```

El algoritmo backpropagation presenta varios problemas.

- Los resultados dependen de los valores iniciales, aleatorios, de las conexiones. Esto hace que sea conveniente entrenar varias redes con distintas valores iniciales y elegir la que mejor funcione.

- A veces se requiere mucho tiempo para obtener soluciones sencillas. Este problema se reduce gracias al aumento de potencia de los procesadores y al uso de nuevas tecnologías. Sin embargo, el tiempo de cálculo se incrementa exponencialmente al aumentar el tamaño de la red. Si bien el volumen de cálculo es proporcional al número total de conexiones, en la práctica, al aumentar el tamaño de la red, hacen falta más ejemplos de aprendizaje lo que provoca un aumento mucho mayor de entrenamiento. Para incrementar la velocidad de convergencia se han desarrollado diferentes modificaciones del algoritmo.

- La "interferencia catastrófica" o empeoramiento en el rendimiento del sistema, como consecuencia de la incorporación de nuevos ejemplos de aprendizaje. Cuando se desea agregar conocimiento adicional a una red ya entrenada es preciso comenzar el aprendizaje desde el principio ya que, de no ser así, la red “olvidará” los primeros patrones.

- Inestabilidad temporal. Si se utiliza un coeficiente de aprendizaje elevado, se producirán incrementos grandes en los pesos generando oscilaciones continuas, de manera que es fácil pasarse de incremento y tener que tratar de compensarlo en el siguiente ciclo, de manera que se producirían oscilaciones continuas. Esto se soluciona usando un coeficiente pequeño, o, para no tener un aprendizaje muy lento, modificar dicho coeficiente adaptativamente (aumentarlo si el error global disminuye y disminuirlo en caso contrario).

- El problema del sobreajuste. Si la red tiene un exceso de neuronas ocultas se ajustará excesivamente a los patrones de entrada reduciendo la capacidad de generalización de la red. Esto provocará que los patrones que no hayan sido presentados durante el entrenamiento sean clasificados incorrectamente (Figura 2.34).

- El problema de los mínimos locales. El algoritmo de backpropagation usa la técnica de gradiente descendiente; esto significa que recorre la "superficie del

error" siempre hacia abajo, hasta alcanzar un mínimo local, pero no garantiza que se alcance una solución globalmente óptima (Figura 2.35). Sin embargo, se ha comprobado que el hecho de alcanzar mínimos locales no impide que se consigan resultados satisfactorios. Este tipo de problemas se resuelve mediante métodos empíricos. Por ejemplo, si la red deja de aprender antes de llegar a una solución aceptable, el cambio de número de neuronas en la capa oculta o de los parámetros de aprendizaje, o bien inicializando los pesos con valores distintos, puede lograr un mejor resultado en un próximo entrenamiento.

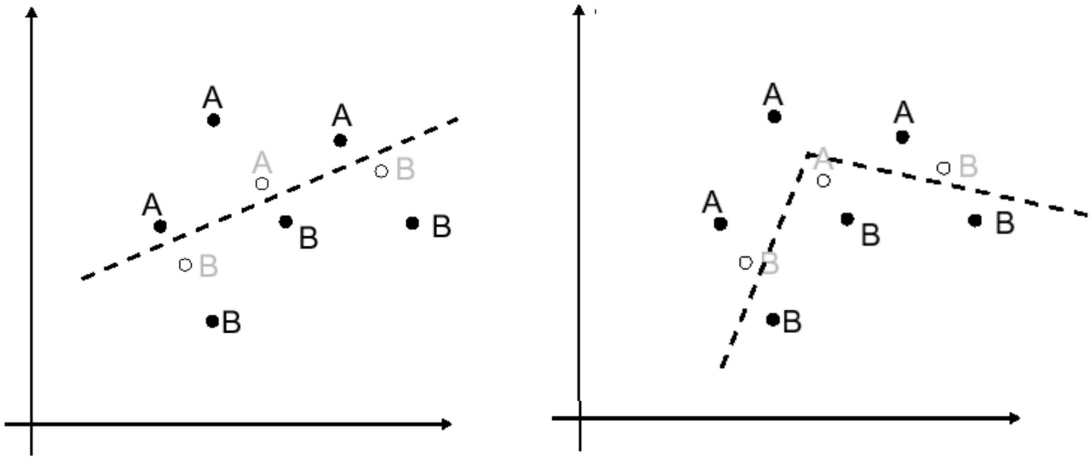


Figura 2.34. Cuando la cantidad de neuronas de la capa oculta es excesiva para el problema que se desea resolver, se produce un "sobreajuste" de los patrones de entrada. La gráfica muestra la resolución de un problema de separación de dos clases utilizando una única neurona oculta (izquierda) y dos neuronas ocultas (derecha). Los puntos sólidos corresponden a los patrones de entrenamiento y los demás a patrones de testeo.

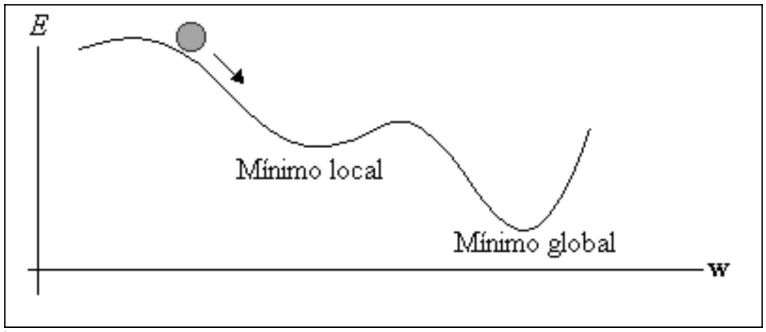


Figura 2.35. La figura ilustra como un vector de pesos (círculo gris) se dirige hacia un mínimo local sobre la superficie de error.

Consideraciones prácticas:

No existe una técnica para determinar el número de capas ocultas, ni el número de neuronas que debe contener cada una de ellas para un problema específico; esta elección es determinada por la experiencia del diseñador. En general, como ya se explicó anteriormente, tres capas son suficientes. Hay veces, sin embargo, en que parece que un problema es más fácil (la red aprende más rápido) de resolver con más de una capa oculta. El tamaño de la capa de entrada suele ser dado por la naturaleza de la aplicación.

Determinar el número de neuronas que hay que utilizar en la capa oculta no suele ser tan evidente como lo es para las capas de entrada y de salida. La idea consiste en utilizar el menor número posible de neuronas. Si la cantidad utilizada es inferior a la necesaria, la red no logra aprender los patrones adecuadamente. Para resolver esto será necesario aumentar la cantidad de neuronas ocultas y comenzar nuevamente con el entrenamiento. Por el contrario, si la red logra aprender todos los patrones, se puede probar con un número inferior de neuronas ocultas y determinar un tamaño final basado en el rendimiento global del sistema.

En general se pueden utilizar todos los datos que estén disponibles para entrenar la red, aunque quizá no sea necesario utilizarlos a todos. A menudo, lo único que se necesita para entrenar con éxito una red es un pequeño subconjunto de los datos de entrenamiento que sea representativo del espacio de entrada. Los datos restantes pueden utilizarse para probar la red. Si se pretende entrenar una red para que funcione en un entorno con ruido, entonces hay que incluir unos cuantos vectores de entrada con ruido.

Otra opción es crear tres conjuntos con los datos de entrenamiento, uno de aprendizaje, otro de validación y finalmente uno de test. La motivación es encontrar la red que tenga la mejor ejecución con casos nuevos, es decir, aquella mejor adquiera la capacidad de generalizar.

Para evitar el problema del sobreajuste, es aconsejable utilizar el segundo grupo de datos diferentes a los de entrenamiento, el grupo de validación, que permita controlar el proceso de aprendizaje. Durante el aprendizaje la red va modificando los pesos en función de los datos de entrenamiento y de forma alternada se la alimenta con los datos de validación.

Con el grupo de validación se puede averiguar cuál es el número de pesos óptimo –y así evitar el problema del sobreajuste–, en función de la arquitectura que ha tenido la mejor ejecución con los datos de validación.

Por último, si se desea medir de una forma completamente objetiva la eficacia final del sistema construido, no debería basarse en el error que se comete ante los datos de validación, ya que de alguna forma, estos datos han participado en el proceso de entrenamiento. Se debería contar con un tercer grupo de datos independientes, el grupo de test el cuál proporcionará una estimación acertada del error de generalización.

Si una red backpropagation se entrena de modo inadecuado o insuficiente empleando una clase concreta de vectores de entrada, la posterior identificación de miembros de esa clase puede ser imprecisa. Es muy importante que el conjunto de patrones de entrada utilizado para el entrenamiento cubra todo el espacio de entradas esperado. Además, los patrones deben ser presentados a la red en un orden aleatorio entre clases distintas. Si se presentan primero todos los patrones de una clase, y luego se prosigue con patrones de otra clase, la red “olvidará” lo aprendido para la primera clase.

Los pesos deben inicializarse con valores pequeños y aleatorios, al igual que los términos de tendencia. La elección de un valor para el parámetro de velocidad de aprendizaje tiene un efecto significativo en el rendimiento de la red. Normalmente, el

factor de aprendizaje debe ser un número pequeño del orden de 0,05 a 0,25 para asegurar que la red llegue a una solución.

Una forma de incrementar la velocidad de convergencia consiste en utilizar una técnica llamada "momento". Cuando se calcula el valor del cambio de peso, se añade una fracción del cambio "anterior". Este término adicional tiende a mantener los cambios de peso en la misma dirección. Las ecuaciones de actualización de pesos de la capa de salida se reescriben así:

$$w_{kj}^{o}(t+1) = w_{kj}^{o}(t) + \alpha(d_k - y_k) f_k^{o\prime}(neta) i_j + \lambda \Delta w_{kj}^{o}(t-1)$$

donde Δw^{0}_{kj} es el término de momento y λ es un parámetro que indica en que proporción participa el término de momento en la actualización de los pesos.

Debe tenerse en cuenta que para aprovechar la capacidad de las redes neuronales de aprender relaciones complejas o no lineales entre variables, es imprescindible la utilización de funciones no lineales al menos en las neuronas de la capa oculta. Las redes neuronales que no utilizan funciones no lineales, se limitan a solucionar tareas de aprendizaje que implican únicamente funciones lineales o problemas de clasificación que son linealmente separables. Por lo tanto, en general se utilizará la función sigmoide –logística o tangente hiperbólica– como función de activación en las neuronas de la capa oculta.

Por otra parte, la elección de la función de activación en las neuronas de la capa de salida dependerá del tipo de tarea impuesta. En tareas de clasificación, las neuronas normalmente toman la función de activación sigmoide. Así, cuando se presenta un patrón que pertenece a una categoría particular, los valores de salida tienden a dar como valor 1 para la neurona de salida que representa la categoría de pertenencia del patrón, y 0 ó -1 para las otras neuronas de salida. En cambio, en tareas de predicción o aproximación de una función, generalmente las neuronas toman la función de activación lineal.

2.5.5. Contrapropagación

Introducción:

Esta red neuronal, definida por Hecht-Nielsen en 1987, fue creada con la intención de diseñar una arquitectura para prototipado rápido. Su definición se basa en la combinación de una estructura competitiva con la estructura outstar de Grossberg (Hecht-Nielsen 1987a, 1987b citados en Freeman & Skapura 1991). A continuación se define su funcionamiento.

Dado un conjunto de pares de vectores (x_1, y_1),..., (x_L, y_L), la red de contrapropagación puede aprender a asociar un vector x en la capa de entrada con un vector y en la capa de salida, el entrenamiento es supervisado. Si la relación entre x e y puede describirse mediante una función continua, ϕ, tal que $y = \phi(x)$, entonces esta red aprenderá a aproximar esta correspondencia para todo valor de x en el intervalo especificado por el conjunto de vectores de entrenamiento. Además, si existe la inversa

de ϕ, de tal manera que x sea una función de y, entonces la CPN aprenderá la correspondencia inversa, $x = \phi^{-1}(y)$. En general, ϕ^{-1} no existe.

La red consta de tres capas. La capa 1 es sólo de entrada. Para cada unidad de la capa oculta se calcula su valor neto de entrada y se produce una competencia para ver qué unidad posee la entrada neta mayor. Esa unidad es la única que envía valores a la capa de salida.

Las principales características de esta arquitectura son:

- Combina estructuras existentes formando redes distintas.
- No utiliza un único algoritmo de aprendizaje a lo largo de toda la red, sino que es distinto en cada capa.
- El entrenamiento de la red es más rápido que otras arquitecturas.
- Es útil para prototipado rápido.

La desventaja de esta red es que no siempre converge con la precisión esperada.

Arquitectura de la red:

La figura 2.36 muestra la arquitectura completa de la red. En ella aparece en la capa de entrada el vector y que corresponde a la respuesta esperada para el vector de entrada x. Su presencia sólo es requerida durante el entrenamiento de la capa de salida

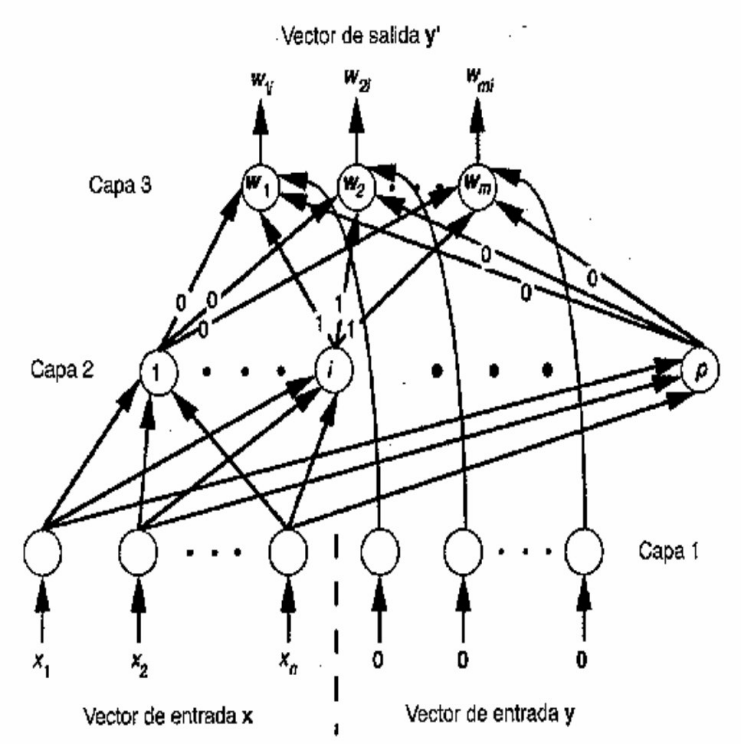


Figura 2.36. Arquitectura de la red de contrapropagación.

Como puede observarse, las neuronas están dispuestas de la manera habitual formando una arquitectura de tres capas pero el proceso empleado para obtener la respuesta de la red es un diferente de lo visto hasta ahora.

La red de contrapropagación es una red feedforward que posee una capa oculta competitiva. Esto último modifica la manera de propagar un patrón desde la entrada hasta la salida con el objeto de obtener la respuesta de la red. En base a la figura 2.36 se analizará este proceso.

Dejando de lado el entrenamiento de la red, proceso que será explicado en breve y suponiendo que la arquitectura ya ha sido entrenada adecuadamente, la siguiente secuencia indica el funcionamiento de esta arquitectura:

- Se presenta a la red el vector *x* que se desea clasificar. Se supone que para este vector el valor de salida esperado es desconocido, es decir, que no se utilizan las neuronas de la capa de entrada correspondientes al vector y como aparecen en la figura 2.36.

- Se calcula la entrada neta de cada una de las neuronas de la capa oculta. Aquella que posea el mayor valor será considerada la *neurona ganadora*, es decir, la que representa de la manera más adecuada al vector de entrada. En la figura 2.36, la neurona ganadora es la i-ésima neurona de la capa oculta.

- Como es habitual, cada neurona produce un único valor de salida que será replicado hacia todas las neuronas de salida. En este caso, por tratarse de una capa competitiva este valor será: 0 si no ha resultado ser la neurona ganadora y 1 en caso contrario. En la figura 2.36 se ve claramente que la única neurona que produce salida 1 es la i-ésima.

- Luego, cada neurona de la capa de salida evalúa su entrada neta de la forma habitual pero dada la competencia previa, este valor coincidirá con el peso del arco que la une a la neurona ganadora de la capa oculta.

Como puede apreciarse, el funcionamiento es muy sencillo. Los elementos de la capa oculta compiten entre si por la representación de los patrones de entrada. Es de esperar, que patrones similares, para alguna medida de similitud definida a priori, sean representados por la misma neurona de la capa oculta.

A continuación se detalla el proceso que debe seguirse para logar obtener esta arquitectura.

Preparación de los datos de entrada:

En la capa de entrada habitualmente, los datos que ingresan son escalados (en este caso normalizados) para adaptarse a los cálculos. Esta modificación en los valores de entrada no sólo pretende evitar desbordamientos durante los cálculos de sumas de productos que predominan en la mayoría de las simulaciones de redes, sino facilitar el funcionamiento de la arquitectura en general.

Con este objetivo, los valores de entrada son normalizados de la siguiente forma:

$$I_i = \frac{x_i}{\sqrt{\sum_j x_j^2}}$$

donde *x* es el vector de entrada.

Esta normalización se encuentra en una estrecha relación con la manera en que se realiza la competencia en la capa oculta. Luego de explicar como se realiza, se volverá a mencionar la adecuación del vector de entrada.

Capa oculta competitiva:

Cada neurona de la capa oculta calcula su entrada neta de la siguiente forma:

$Neta = I * w$
$Neta = \| I \| * \| w \| \cos(\theta)$
$Neta = \cos(\theta)$

donde *w* es el vector de pesos de la neurona y tanto I como w están normalizados. Esto reduce el valor de la entrada neta al coseno del ángulo que forman los vectores.

Como se trata de una capa competitiva, donde se considera ganador al elemento con mayor entrada neta, esta forma de calcular el estímulo total de entrada permite que cada vector sea representado por el *w* más próximo. Esto último se debe a que la función coseno retorna un valor mayor a medida que el ángulo entre los vectores se reduce.

Por este motivo, la regla de aprendizaje aplicada a los pesos que relacionan la capa de entrada (con sus valores normalizados) con la capa oculta, realiza una aproximación gradual del vector *w* asociado con la neurona ganadora en dirección a la entrada normalizada *I*. De esta forma, la próxima vez que esta entrada sea presentada a la red, el ángulo entre *I* y el vector de pesos de la ganadora será menor, dando lugar a una mayor entrada neta. Esto aumenta la representatividad de la neurona ganadora.

La actualización de pesos, que se aplica únicamente al vector *w* asociado con la neurona ganadora, es la siguiente:

$w(t+1) = w(t) + \alpha\,(I - w(t))$.

La idea es lograr que cada neurona de la capa oculta represente un conjunto de vectores de entrada relativamente próximos. Luego del entrenamiento se espera que el vector *w* aprendido sea un valor representativo del conjunto, por ejemplo, la media. Por lo tanto, es razonable ir reduciendo el valor de α a medida que avanza el entrenamiento. De esta forma, los vectores más alejados del centro no forzarían a *w* a salir de la zona. Se supone que *w* una vez que alcanza la media debe permanecer en un entorno.

Si hay varias neuronas en la capa oculta, cada una responde con un valor máximo a un cierto grupo de vectores de entrada perteneciente a una región distinta del espacio. Por lo tanto, esta capa clasifica una entrada dada. Es decir, la neurona con mayor respuesta es la que identifica a la región del espacio al que pertenece el vector.

La figura 2.37 representa esta situación. En ella puede observarse como quedan representados tres tipos de letras distintas mediante una capa competitiva que tendría como mínimo tres elementos. Luego del entrenamiento, cada vector de pesos representará aproximadamente el valor promedio de los patrones de cada clase.

Es importante destacar que si a priori se conociera la clase a la que pertenece cada patrón sería factible obtener los pesos que unen las capas de entrada y oculta sin necesidad de realizar ningún entrenamiento. De esta forma, al presentar un nuevo patrón que constituya una pequeña variante de los utilizados inicialmente, resultaría ganadora la neurona adecuada.

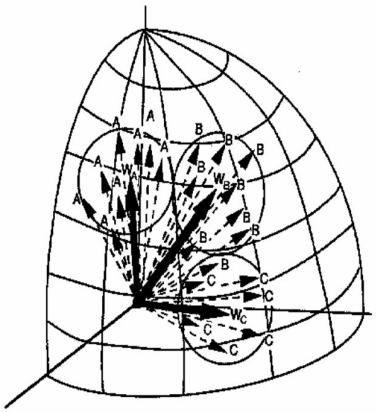


Figura 2.37. La gráfica representa los vectores de pesos de tres neuronas competitivas entrenadas para reconocer distintos tipos de letras.

Lamentablemente, es ocasiones no se dispone de este tipo de información y por lo tanto es necesario contar con alguna estrategia para el entrenamiento de los pesos que llegan a las neuronas ocultas. A continuación se detalla el algoritmo a utilizar:

Algoritmo de entrenamiento de los pesos que llegan a las neuronas competitivas:

```
Seleccionar el valor de alfa
Mientras no se estabilicen los vectores de pesos
   Para cada patrón de entrada
      Seleccionar un vector de entrada aleatorio.
      Normalizarlo e ingresarlo a la red.
      Calcular la neurona ganadora.
      Actualizar el vector de pesos de la neurona ganadora
              w(t+1) = w(t) +  α (x−w(t))
   Fin Para
Fin Mientras
```

Capa de Salida:

El entrenamiento de los pesos entre la capa oculta y la capa de salida pretende lograr algo similar a lo propuesto en la Regla de Hebb.

Cada neurona de la capa de salida recibirá básicamente dos valores: el correspondiente a la neurona ganadora de la capa oculta (estímulo condicionado) y el valor esperado, y (estímulo no condicionado), como está representado en la zona punteada de la figura 2.38.

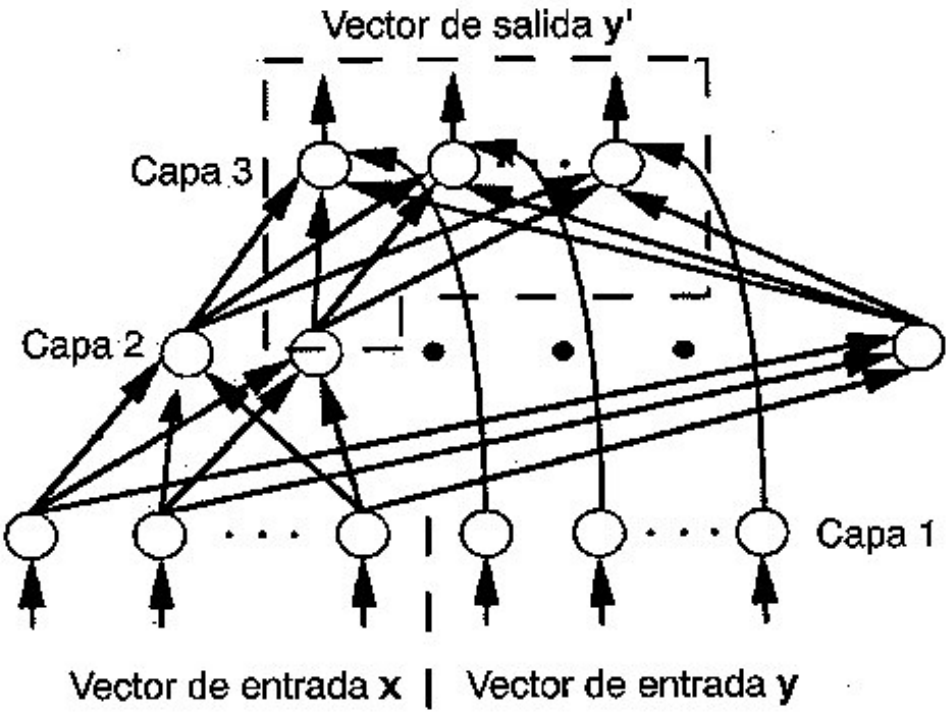


Figura 2.38. La zona punteada indica la parte de la red que se activa al determinarse la neurona ganadora.

El aprendizaje supervisado es el que produce una respuesta condicionada, que es el valor esperado. Esto llevará a que el valor del estímulo condicionado se incremente hasta un punto en que sea capaz de generar por si mismo la salida esperada.

La actualización del vector de pesos entre la capa oculta y la capa de salida se realiza de manera similar al entrenamiento descripto anteriormente:

$$w_i(t+1) = w(t) + \beta(y_i - w_i(t))$$

donde y es el vector esperado.

Algoritmo de entrenamiento en la capa de salida:

```
Seleccionar el valor de beta
Mientras no se clasifiquen correctamente todos los patrones
   Para cada patrón de entrada
      Seleccionar un vector de entrada aleatorio.
      Normalizarlo e ingresarlo a la red junto con su valor
         esperado y
      Calcular la neurona ganadora.
      Actualizar los pesos de las conexiones que van de la
         unidad competitiva ganadora a las unidades de salida
         según
               w(t+1) = w(t) + β(y − w(t))
   Fin Para
Fin Mientras
```

2.5.6. Red de Hopfield

Introducción:

La primera red neuronal recurrente de naturaleza dinámica fue propuesta por Hopfield en 1984 bajo el contexto de las memorias asociativas. La principal aplicación es el control e identificación de sistemas no lineales. Este desarrollo es posible debido a que las propiedades matemáticas de las redes recurrentes están enmarcadas en las mismas propiedades que fundamentan el control geométrico.

Las redes de Hopfield son redes de adaptación probabilística y recurrentes, funcionalmente entrarían en la categoría de las memorias autoasociativas, que aprenden a reconstruir los patrones de entrada que memorizaron durante el entrenamiento. Son arquitecturas de una capa con interconexión total, funciones de activación booleana de umbral (cada unidad puede tomar dos estados, 0 o 1, dependiendo de si la estimulación total recibida supera determinado umbral), adaptación probabilística de la activación de las neuronas, conexiones recurrentes y simétricas, y regla de aprendizaje no supervisado. Mientras que las redes en cascada (no recurrentes) dan soluciones estables, los modelos recurrentes dan soluciones inestables (dinámicas), lo que no siempre es aconsejable. El principal aporte de Hopfield consistió precisamente en conseguir que tales redes recurrentes fueran así mismo estables. Imaginó un sistema físico capaz de operar como una memoria autoasociativa, que almacenara información y fuera capaz de recuperarla aunque la misma se hubiera deteriorado.

Arquitectura de la red:

A cada estado de la red se le puede atribuir una cierta cantidad de energía, el sistema evoluciona tratando de disminuir la energía mediante un proceso de relajación, hasta alcanzar un mínimo donde se estabiliza. Los mínimos de energía se corresponden con los recuerdos almacenados durante el aprendizaje de la red.

Ante la presentación de un estímulo nuevo se obtendrá una configuración inicial más o menos parecida a alguno de los estímulos almacenados, el sistema evolucionará hasta caer en una configuración estable que representa el recuerdo asociado a ese estímulo. Si la configuración inicial discrepa mucho de los recuerdos almacenados

podemos alcanzar algún mínimo que no se corresponde con la respuesta esperada, recuperando en ese caso una información falsa, o podríamos no alcanzar ningún mínimo, quedando inestable: en ese caso diríamos que la red está "confundida", no es capaz de reconocer el estímulo, ni recuerda. Una tercera posibilidad es que al cabo de unos pasos de evolución empiece a repetir periódicamente una secuencia definida de estados; con esta dinámica se han modelado ciertas excitaciones nerviosas que regulan acciones rítmicas y repetitivas; y se ha tratado de reproducir la memoria de secuencias temporales.

Las neuronas se conectan todas entre sí, y consigo mismas (Figura 2.39). Al comienzo se le asigna a cada neurona el valor o estado correspondiente del patrón de entrada. En cada ciclo se elige una neurona al azar y se calcula su activación (que coincide con su salida) según una función de umbral.

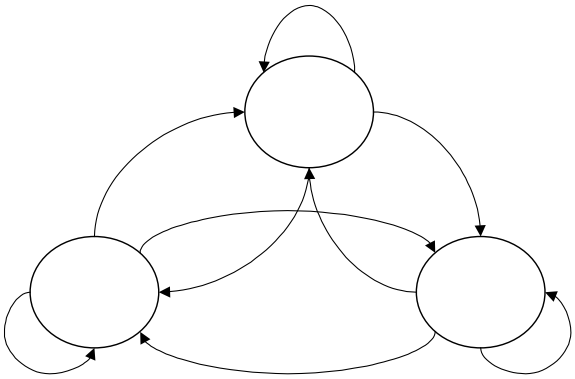


Figura 2.39. Arquitectura de una red de Hopfield de tres neuronas.

Se puede trabajar con cualquier valor de umbral para la función de activación, pero típicamente se usa el 0 como umbral, con la ventaja de simplificar las ecuaciones. El valor de la red se calcula como la sumatoria de todas las entradas ponderadas, incluida la procedente de la misma neurona.

La elección de la regla de aprendizaje no es trivial, depende de la interrelación de los patrones que se desea memorizar. Si estos patrones están poco correlacionados, podemos aplicar la regla de Cooper-Hebb, basada en la regla de Hebb o regla del producto:

$w_{ij} = w_{ji} = \sum_{m=1}^{M} (2p_i^m - 1)(2p_j^m - 1)$, si $j \neq i$. donde M es la cantidad de patrones a almacenar.

$w_{ij} = 0$, si $j = i$

Esta regla fortalece las conexiones cuando las neuronas i-ésima y j-ésima tienen la misma activación o valor de estado y las debilita en caso contrario.

Las redes de Hopfield pueden usarse desde un enfoque psicofisiológico, como un modelo sencillo para explicar como ocurren las asociaciones entre ideas. Las ideas parciales serían estados de activación en la zona de atracción de ideas más generales (puntos fijos o puntos de equilibrio), de forma que al introducir la idea parcial se puede llegar a alcanzar la idea general.

A su vez, debido a que las áreas de atracción indican sólo una probabilidad (generalmente diferente de 1), este modelo permite explicar también la incertidumbre que se produce en las asociaciones: una idea parcial, a pesar de tener alta probabilidad de desembocar en la idea general, puede llevar a otras ideas diferentes que también actúan como puntos de equilibrio.

Una posible aplicación informática de las redes de Hopfield es el desarrollo de memorias direccionadas por contenido: los elementos de la memoria no estarían ordenados según índices numéricos, sino según parte de su contenido. En las memorias actuales es necesario conocer la dirección de memoria de un dato para poder recuperarlo, mientras que en las memorias direccionadas por contenido se pueden recuperar datos completos (puntos de equilibrio) a partir de datos parciales (que formen parte de su área de atracción).

Las redes de Hopfield se han aplicado a campos como la percepción, el reconocimiento de imágenes y optimización de problemas, mostrando gran inmunidad al ruido y robustez. Incluso se han llegado a desarrollar chips específicos para este tipo redes. El estudio de las representaciones de secuencias temporales es un área de gran interés, con aplicaciones en reconocimiento automático de voces y movimientos.

Hopfield ha mostrado como aplicar los mismos principios con funciones de activación continuas como la función sigmoidal, con muy pocas modificaciones.

Pero pese a sus evidentes ventajas no están exentas de problemas:

- El número máximo de patrones no correlacionados que puede almacenar es igual al 15% del número de neuronas de la red.

- Requieren mucho tiempo de procesamiento hasta converger a una solución estable, lo que limita su aplicabilidad.

- Tienden a caer en mínimos locales.

Algoritmo de entrenamiento:

La salida de la red de Hopfield tenderá a converger a los patrones prototipos almacenados. En el caso que todos los patrones prototipo sean ortogonales, cada uno será un punto de equilibrio de la red; la red puede tener muchos otros puntos de equilibrio indeseables. Una regla práctica para evitarlos consiste en que cuando se utilice la regla de Hebb, el número de patrones almacenados no debe superar el 15% del número de neuronas de la red.

```
Asignar pesos iniciales a las neuronas
Mientras no se alcance un error mínimo hacer
  Presentar un patrón de entrada a la red
  Elegir un número de neuronas que cumpla el criterio del 15%
  Codificar los ítems que queremos memorizar de forma que los
     patrones para representarlos se parezcan lo menos posible
     entre sí, para aproximarnos a la condición de pseudo-
     ortogonalidad
```

```
  Calcular los pesos de las conexiones según la regla de
Cooper-Hebb
Fin mientras
```

2.5.7. Red de Hamming

Introducción:

La red de Hamming es uno de los ejemplos más simples de aprendizaje competitivo. A pesar de ello su estructura es un poco compleja ya que emplea el concepto de capas recurrentes en su segunda capa y aunque hoy en día en redes de aprendizaje competitivo se ha simplificado este concepto con el uso de funciones de activación más sencillas, la red de Hamming representa uno de los primeros avances en este tipo de aprendizaje, convirtiéndola en un modelo obligado de referencia dentro de las redes de aprendizaje competitivo.

Las neuronas en la capa de salida de esta red compiten unas con otras para determinar la ganadora, la cual indica el patrón prototipo más representativo en la entrada de la red. La competencia es implementada por inhibición lateral (un conjunto de conexiones negativas entre las neuronas en la capa de salida).

Arquitectura de la red:

Esta red consiste en dos capas: la primera realiza la correlación entre el vector de entrada y los vectores prototipo, la segunda capa realiza la competición para determinar cual de los vectores prototipo está más cercano al vector de entrada.

La primera capa es capaz de clasificar solo un patrón. Para que múltiples patrones sean reconocidos se necesitan múltiples neuronas y es precisamente de esa forma como está compuesta la primera capa de la red de Hamming.

La salida de la capa 1 es igual al producto de los vectores prototipo con la entrada más el vector de ganancias. Este producto indica cuan cercano está cada vector de entrada a los patrones prototipo.

$$a^1 = w^1 p + b^1$$

donde w es el vector de pesos, p es el vector prototipo que se quiere reconocer y b es el número de elementos en el vector de entrada.

En la capa 2 de la red de Hamming se utilizan múltiples neuronas, así se determinara por medio de una capa competitiva el patrón prototipo más cercano. Las neuronas en esta capa son inicializadas con la salida de la capa en realimentación, la cual índica la correlación entre los patrones prototipo y el vector de entrada. Las neuronas compiten unas con otras para determinar una ganadora; después de la competición solo una neurona tendrá una salida no cero. La neurona ganadora índica cual categoría de entrada fue presentada a la red (cada vector prototipo representa una categoría).

La salida de la primera capa a^1 es usada para inicializar la segunda capa. La salida de la segunda capa está determinada de acuerdo a la siguiente relación recurrente:

$$a^2(t+1) = f(w^2 a^2(t))$$

donde *f* es la función de transferencia.

En cada iteración, cada salida de la neurona se decrementará en proporción a la suma de las salidas de las otras neuronas. La salida de la neurona con la condición inicial más grande se decrementará más despacio que las salidas de otras neuronas. Eventualmente cada neurona tendrá una salida positiva y en ese punto la red habrá alcanzado el estado estable.

2.5.8. Red de Elman

Arquitectura de la red:

La red de Elman típicamente posee dos capas, cada una compuesta de una red tipo Backpropagation, con la adición de una conexión de realimentación desde la salida de la capa oculta hacia la entrada de la misma capa oculta, esta realimentación permite a la red de Elman aprender a reconocer y generar patrones temporales o variantes con el tiempo.

La red de Elman generalmente posee neuronas con función transferencia sigmoidal en su capa oculta, y neuronas con función de transferencia tipo lineal en la capa de salida. La ventaja de la configuración de esta red de dos capas con este tipo de funciones de transferencia, es que puede aproximar cualquier función con la precisión deseada mientras que esta posea un numero finito de discontinuidades, para lo cual la precisión de la aproximación depende de la selección del numero adecuado de neuronas en la capa oculta.

Para la red de Elman la capa oculta es la capa recurrente y el retardo en la conexión de realimentación almacena los valores de la iteración previa, los cuales serán usados en la siguiente iteración; dos redes de Elman con los mismos parámetros y entradas idénticas en las mismas iteraciones podrían producir salidas diferentes debido a que pueden presentar diferentes estados de realimentación.

Debido a la estructura similar de la red de Elman con una red tipo Backpropagation, esta red puede entrenarse con cualquier algoritmo de propagación inversa.

La red de Elman no es tan confiable como otros tipos de redes porque el gradiente se calcula con base en una aproximación del error, para solucionar un problema con este tipo de red se necesitan más neuronas en la capa oculta que si se solucionara el mismo problema con otro tipo de red.

Algoritmo de entrenamiento:

```
Presentar a la red, los patrones de entrenamiento y calcular
     la salida de la red con los pesos iniciales, comparar la
     salida de la red con los patrones objetivo y generar la
     secuencia de error.
```

```
Propagar inversamente el error para encontrar el gradiente
     del error para cada conjunto de pesos y ganancias.
Actualizar todos los pesos y ganancias con el gradiente
     encontrado con base en el algoritmo de propagación
     inversa.
```

2.5.9. Redes de Kohonen

Esta arquitectura será estudiada con mas detalle en el capítulo 3, por eso aquí se presenta brevemente.

Las redes de Kohonen poseen una arquitectura de dos capas (una de entrada y otra de salida), funciones de activación lineales y flujo de información unidireccional. Las neuronas de entrada reciben datos continuos. Tras el aprendizaje de la red, cada patrón de entrada activará una única neurona de salida. En algunos modelos, cada entrada puede provocar la activación de un conjunto de neuronas de salida.

El objetivo de este tipo de redes es clasificar los patrones de entrada en grupos de características similares, de manera que cada grupo active siempre la(s) misma(s) salida(s). Cada grupo de entradas queda representado en los pesos de las conexiones de la unidad de salida ganadora. La unidad de salida ganadora para cada grupo de entradas no se conoce a priori, es necesario averiguarlo después de entrenar a la red.

Una de las cualidades de este tipo de redes es la incorporación a la regla de aprendizaje de cierto grado de sensibilidad con respecto al vecindario. Esto hace que el número de neuronas que no aprenden desaparezca, aumentando así su capacidad de extraer o mapear características topológicas de los datos.

La red mapea el espacio de entrada hacia un espacio de salida con cierto orden topológico, Kohonen propone un método para que este orden se conserve al entrenar la red, la clave está en reducir el tamaño del vecindario de la unidad ganadora en cada iteración.

Una vez entrenada, podemos usar a la red para clasificar patrones de entrada similares en el espacio n-dimensional. Una clase o grupo de patrones similares tiende a controlar una neurona específica, que representará el centro de una esfera n-dimensional (de radio unitario, pues los datos sobre la neurona están normalizados). Esa neurona resultará la más activada frente a los patrones más parecidos a su vector de pesos.

Después del aprendizaje, la clasificación consiste en presentar una entrada y seleccionar la unidad más activada, la ganadora, mediante la función de distancia utilizada (gana la más cercana al patrón de entrada). Además, el vector de pesos nos servirá para reconstruir el patrón de entrada.

3. MAPAS AUTO-ORGANIZATIVOS DE KOHONEN

En ocasiones, cuando se desea obtener información a partir de grandes volúmenes de datos, resulta de utilidad medir las similitudes que existen entre ellos. Esto permite agrupar la información disponible para luego obtener conclusiones. Así, el objetivo de esta tarea es el de clasificar los datos, agrupándolos en subconjuntos de similares características.

La necesidad de descubrir los distintos clusters que conforman un conjunto de patrones resulta muy útil para distintas aplicaciones específicas, como la minería de datos, el tratamiento de imágenes, el tratamiento y reconocimiento de voz, y muchas más.

Los mapas auto-organizativos también conocidos como SOM (Self-Organizing Maps), definidos por Kohonen en 1990 (Kohonen 1990, 1997), son una herramienta sumamente útil para resolver este tipo de problemas, ya que a menudo se desconoce la relación que existe entre los datos de entrada. Estos mapas deben su nombre a su capacidad de organizarse automáticamente a medida que el entrenamiento es llevado a cabo. Esta organización, como se verá más adelante en detalle, implica que las neuronas que son vecinas en el espacio de salida, y a pesar que la estructura de la red y las conexiones entre las neuronas se mantiene fija durante el entrenamiento, serán representantes de patrones de entrada similares.

Esta red neuronal posee una capa competitiva cuyas neuronas, durante el entrenamiento, buscan representar a los patrones de entrada de la mejor manera posible. Una vez concluido el entrenamiento de la red, cada subconjunto de datos con características similares queda representado por una o más neuronas, con la particularidad de que cada subconjunto activará siempre la(s) misma(s) neurona(s) en la salida.

Cuando una neurona, finalizado el entrenamiento, representa un grupo de patrones de entrada, se dice que la neurona "mapea" los patrones de datos a los que representa. Siguiendo la misma idea, a los patrones que son representados por una o más neuronas, se dice que son "mapeados" por la neurona que los representa.

Cada una de las neuronas artificiales posee un vector de pesos, el cual será, una vez finalizado el entrenamiento, el vector prototipo que representará a los patrones de entrada que mapee. Las neuronas forman parte de una única estructura, dando lugar a la definición de una red neuronal. Las neuronas que la forman presentan un comportamiento competitivo y se adaptan a través de un entrenamiento no supervisado. Esto permite que la red descubra, por si misma, las cualidades y características de los datos de entrada.

La capacidad de proyectar el espacio de entrada en prototipos de una grilla regular de baja dimensionalidad permite utilizar los mapas auto-organizativos para visualizar y explorar las propiedades de la información. Esto es posible debido a que se crea una proyección que respeta la topología de los datos, de manera que elementos cercanos en el espacio de entrada lo seguirán siendo dentro de la estructura de baja dimensionalidad determinada por los vectores prototipos.

3.1. ARQUITECTURA DEL MODELO

El mapa auto-organizativo de Kohonen está organizado, por lo general, como una estructura bi-dimensional de neuronas. Cada uno de los datos o patrones de entrada está representado por un vector n-dimensional, donde en cada posición almacena alguna característica del dato o patrón de entrada. Cada una de las neuronas tiene asociado un vector de pesos n-dimensional, del mismo tamaño que los patrones de entrada. Las dimensiones de los vectores de pesos de las neuronas y la de los datos deben tener la misma longitud. Los patrones de entrada son utilizados para alimentar a la red durante el entrenamiento.

Una de las arquitecturas clásicas utilizadas en los mapas auto-organizativos es la arquitectura lineal, que está formada por un arreglo de neuronas uni-dimensional, donde cada neurona tiene solo dos vecinas directas, excepto las neuronas de las puntas que solo tienen una. Otra muy utilizada es la matriz rectangular, donde las neuronas pueden tener hasta cuatro vecinas directas, o su variante hexagonal, donde las neuronas pueden tener hasta seis neuronas vecinas directas (Figura 3.1).

Pueden utilizarse otras arquitecturas con formas irregulares y hasta incluso utilizar alguna arquitectura formada en tres o más dimensiones. Las arquitecturas bi-dimensionales son muy útiles ya que permiten una fácil visualización del resultado final.

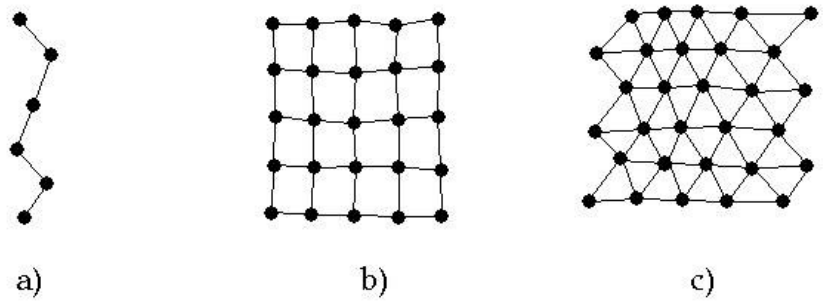


Figura 3.1. Los puntos negros representan a las neuronas de la red. Las líneas muestran la conectividad entre dos neuronas. En a) se muestra una arquitectura lineal de seis neuronas, en b) una arquitectura rectangular de 25 neuronas y en c) una arquitectura hexagonal de 30 neuronas.

3.2. INICIALIZACIÓN DE LOS VECTORES DE PESOS

La inicialización de los pesos de las neuronas al comienzo del entrenamiento es muy importante para lograr un resultado satisfactorio, especialmente si de antemano se conocen características de los patrones de entrada y la arquitectura de la red se construye teniendo en cuenta dichas características.

Existen distintas formas de inicializar los vectores de pesos de las neuronas en un mapa auto-organizativo. La más simple es asignar a los vectores de pesos valores aleatorios. Es recomendable que los valores aleatorios asignados a los vectores de pesos,

estén dentro del espacio de entrada. Se ha demostrado que los pesos de las neuronas, luego de varias pasadas de entrenamiento, logran un estado ordenado que respeta la topología de los datos de entrada (Kohonen 1997).

Otro método de inicialización es asignar como vectores de pesos a algunos de los patrones de entrada. Si se conoce de antemano que algunos patrones representan a un subconjunto bien definido de patrones de entrada, pueden ser utilizados como valores iniciales para los vectores de pesos de las neuronas.

También puede utilizarse una inicialización lineal, y comenzar con el entrenamiento teniendo en la red un orden topológico bien definido, ya que comenzar con las neuronas en un estado ordenado siempre es preferible, aunque la mayoría de las veces no se conoce la topología del espacio de entrada.

3.3. ENTRENAMIENTO DE LA RED

El entrenamiento de los mapas auto-organizativos consiste en presentar los patrones de entrada a la red. A cada uno de los patrones de entrada, se los compara con los vectores de pesos de todas las neuronas de la red, y con alguna medida de similitud, definida a priori, se elige la neurona que mejor respuesta obtenga para ese patrón de entrada. Tanto la neurona que obtuvo la mejor respuesta como sus neuronas vecinas llevan a cabo un ajuste de sus vectores de pesos para "parecerse" al patrón de entrada. Estos pocos pasos se repiten un número significativo de veces hasta lograr que las neuronas representen, cada una, a un subconjunto de patrones de entrada.

3.3.1. Elección de la neurona ganadora

Existen varias medidas de similitud para elegir a la neurona que mejor represente a un patrón de entrada. Las más conocidas y utilizadas son, el producto interno, distancia Euclidea y distancia de Hamming. Puede utilizarse cualquier otra medida de similitud, y la elección de esta medida dependerá del problema que se quiera resolver.

3.3.1.1. Producto interior

El producto interior de dos vectores responde a la manera en que las redes neuronales feedforward calculan la entrada neta de cada una de sus neuronas. Para poder utilizarlo es necesario que, tanto los patrones de entrada como los vectores de pesos de las neuronas estén normalizados:

$$I = \frac{x}{\|x\|} = \frac{x}{\sqrt{\sum_j x_j^{\,2}}}$$

Una vez normalizado el patrón de entrada, cada neurona calcula su entrada como:

$Neta = I * W = ||I||\ ||W|| \cos(\theta) = \cos(\theta)$,

donde I es el patrón de entrada normalizado, W el vector de pesos normalizado de la neurona y θ el ángulo entre ambos vectores.

La neurona con la entrada neta más alta (la que posea el menor ángulo respecto al patrón de entrada) será elegida como la neurona con mejor respuesta, también llamada "neurona ganadora", ya que su vector de pesos es el más similar, entre todos los vectores de pesos, al patrón de entrada.

La idea del aprendizaje es ir acercando el vector *W* al patrón de entrada *I*, como se muestra en la figura 3.2.

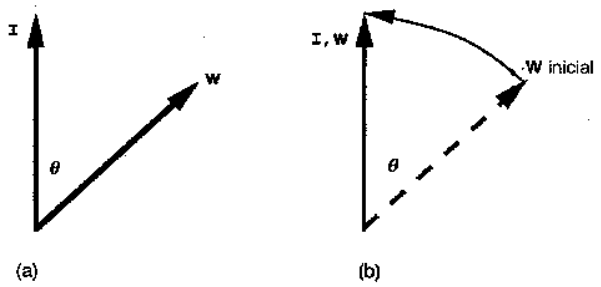


Figura 3.2. En a) se puede ver el patrón de entrada *I* y un vector de pesos *W*, separados en un ángulo *θ*. Si *W* resulta ser el vector de pesos más similar a *I* entonces el aprendizaje hará que *W* se acerque a *I*, como muestra b).

Durante el aprendizaje, la neurona ganadora que obtenga la mejor respuesta actualiza su vector de pesos de la siguiente manera:

$$W(t+1) = W(t) + \alpha\,(I - W(t)), \tag{3.1}$$

la idea es acercar el vector de pesos *W*, en una proporción $0 < \alpha < 1$, a *I* (Figura 3.3).

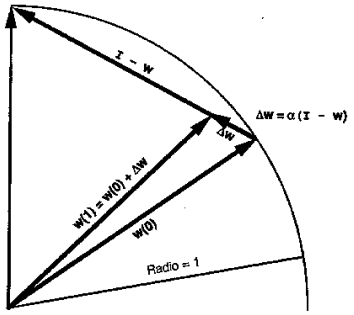


Figura 3.3. Se puede observar el efecto que produce la ecuación (3.1) sobre el vector de pesos *W*.

Este método tiene como desventaja la posibilidad de no elegir correctamente los representantes de los patrones de entrada. Un ejemplo que muestra este problema es cuando en el espacio de entrada hay dos subconjuntos de patrones bien definidos y distintos entre si, y con la característica de que ambos están en la misma dirección. Como todos los patrones de entrada son normalizados, ambos subconjuntos caerán dentro del mismo rango en la norma, de esta manera una misma neurona será

representante de ambos subconjuntos, cuando en realidad debieron ser dos, una neurona para cada subconjunto (Figura 3.4).

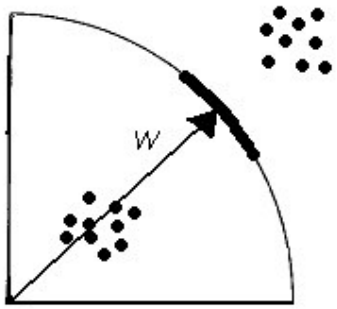


Figura 3.4. Se muestran dos subconjuntos bien definidos de patrones de entrada (puntos negros), distintos entre si, los cuales al ser normalizados caen dentro del mismo rango en la norma.

3.3.1.2. Distancia Euclidea

La distancia Euclidea como medida de similitud entre patrones no presenta el problema que tiene el producto interior. Cuando se utiliza, los vectores de pesos se actualizan de la misma forma que con el producto interior:

$$w_i(t+1) = w_i(t) + \alpha(t)\,(x - w_i(t))$$

donde w_i es el vector de pesos de la neurona i que resultó ganadora, x es el vector de entrada que fue comparado con todas las neuronas para buscar la neurona ganadora, y $\alpha(t)$ es un parámetro conocido como factor de aprendizaje, que es utilizado para calcular la proporción con el que $w(t)$ se acercará a I.

La neurona ganadora es aquella que esté más cerca, en el espacio Euclideo, del patrón de entrada. Cada neurona calcula su entrada neta como:

$$neta_i = -dist(x, w) = \sqrt{\sum_j (x_j - w_j)^2}$$

la neurona ganadora es la neurona cuya entrada neta sea la mas alta.

Nótese que la actualización del vector de pesos no ha cambiado pese a haber modificado el criterio para identificar la neurona ganadora. Esto tiene que ver con el hecho de que aproximar los vectores normalizados, utilizados en el producto interior, o aproximar el vector de entrada x al vector de pesos provoca, en ambos casos, un aumento de la medida de similitud.

3.3.2. Factor de aprendizaje “óptimo”

La literatura indica que en las arquitecturas neuronales clásicas, la manera de determinar el valor del factor de aprendizaje es totalmente heurística. Es importante considerar la importancia que este parámetro tiene en la velocidad de convergencia de la

red. Si se utiliza un valor muy pequeño, el algoritmo de aprendizaje puede llegar a requerir un excesivo número de iteraciones mientras que si su valor es alto, las modificaciones a realizar sobre los vectores de pesos serán muy abruptas provocando oscilaciones que impedirán a la red obtener el valor esperado (Figura 3.6).

Como forma de resolver este problema, existen distintas alternativas que van desde comenzar con un factor de aprendizaje alto e ir disminuyéndolo a medida que transcurre el entrenamiento (Figura 3.7), hasta el uso de estrategias evolutivas que permitan determinarlo automáticamente. Si bien esto último es una posibilidad atractiva, su implementación tiene un alto costo computacional y excede los alcances de esta tesis.

En Kohonen 1997 se describe un método para conseguir un factor de aprendizaje óptimo en los últimos pasos del entrenamiento, cuando la vecindad es cero. Se define un factor de aprendizaje $\alpha_i(t)$ para cada neurona m_i y se define:

$$m_i(t+1) = m_i(t) + \alpha_i(t) h_{ci}\left[x(t) - m_i(t)\right],$$

donde h_{ci} es invariante en el tiempo. Cuando m_i resulta ganadora, además de actualizarse se ajusta $\alpha_i(t)$ de la siguiente manera.

$$\alpha_i(t+1) = \frac{\alpha_i(t)}{1 + h_{ci}\alpha_i(t)} \tag{3.2}$$

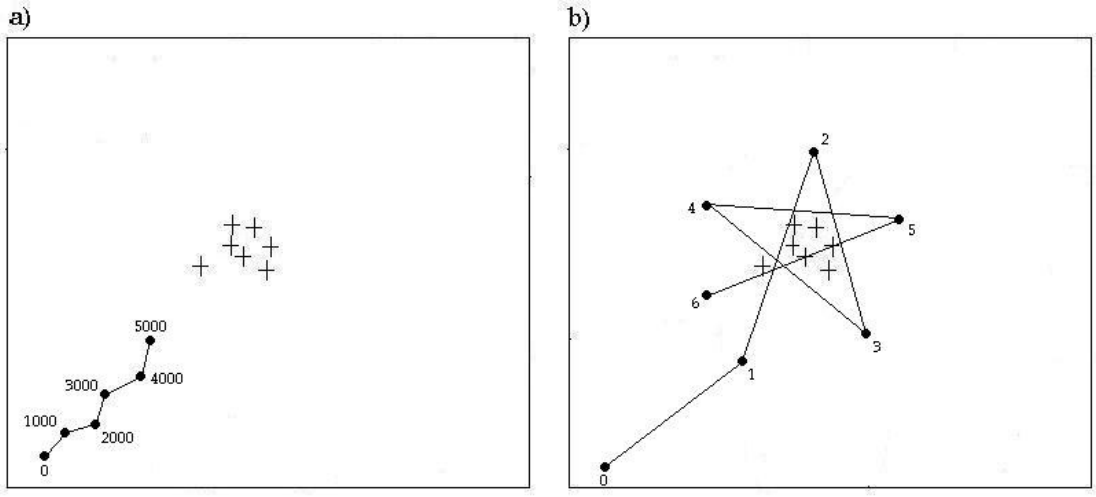


Figura 3.6. Los puntos negros representan los distintos valores que toma el vector de pesos de una misma neurona en el espacio de entrada durante un entrenamiento. Las cruces representan a los patrones de entrada. El factor de aprendizaje α (ver ecuación (3.1)) es un factor que afecta a la diferencia entre el vector de pesos de la neurona y el patrón de entrada. Si el factor α es muy pequeño causará que la neurona se acerque muy lentamente a los patrones de entrada. En a) se muestra como después de 5000 pasadas la neurona aún no llegó a los patrones de entrada. Por el contrario si el factor α es muy grande, causará que el aprendizaje de la neurona sea demasiado "abrupto". En b) se muestra a una neurona oscilar entre los patrones de entrada durante seis pasadas. El comportamiento de la neurona sería similar durante 1000 pasadas más.

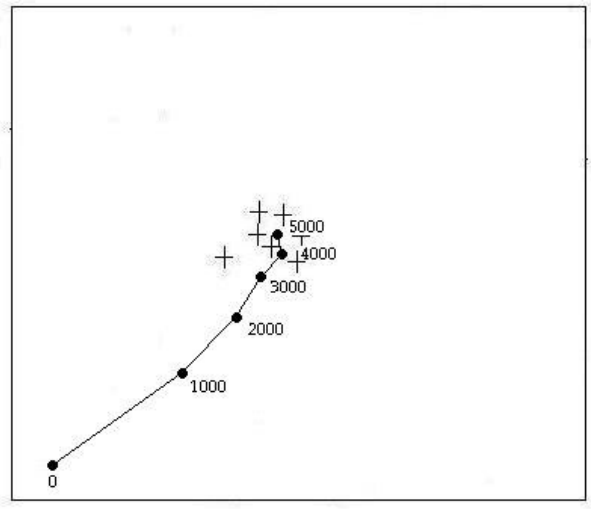


Figura 3.7. Los puntos negros representan los distintos valores que toma el vector de pesos de una misma neurona en el espacio de entrada durante un entrenamiento. Las cruces representan a los patrones de entrada. En este entrenamiento se comenzó con factor α grande, el cual fue disminuyendo a medida que transcurría el entrenamiento. Se puede observar que al comienzo los saltos para una misma cantidad de pasadas son grandes, y luego van disminuyendo hasta alcanzar, después de 5000 pasadas, los patrones de entrada.

Esta ecuación resulta de una especulación teórica y podría no funcionar correctamente en la práctica, ya que se pueden obtener valores muy diferentes de $\alpha_i(t)$ para grandes corridas y para diferentes *i*.

Por otro lado, ya que las regiones de Voronoi (ver sección 3.5) cambian cada vez que se modifican los pesos, los $x(t)$ para un m_i también cambian. Por eso, no es correcto decir que en la fase de ajuste se cuenta con un factor "óptimo", con la ecuación (3.2) solo se obtiene un factor que se aproxima al "óptimo".

3.3.3. Efecto de la forma de la función de vecindad utilizada

No solo la neurona ganadora participa en el proceso de aprendizaje, también lo hace el conjunto de neuronas vecinas de la neurona ganadora.

La forma más simple de determinar la vecindad de una neurona, es tomando aquellas neuronas que estén alrededor de ella (Figura 3.8). Una forma de expresarlo algebraicamente es:

$$v_{gi}(t) = \frac{1}{D_{gi}} \tag{3.3}$$

donde D_{gi} es la distancia en la estructura de la red que separa a la neurona ganadora *g* de una neurona *i* (ver figura 3.8). El valor de v_{gi} será cada vez mas chico a medida que *i* se aleje de *g*.

Por otra parte, una de las funciones continuas mas utilizadas para definir la vecindad es la ecuación Gaussiana:

$$v_{gi}(t) = \alpha(t).\exp\left(-\frac{\left\|r_g - r_i\right\|^2}{2\sigma^2(t)}\right) \tag{3.4}$$

Suponiendo que se utiliza la distancia Euclidea para seleccionar la neurona ganadora, las neuronas vecinas actualizan sus pesos de manera similar a ella:

$w_i(t+1) = w_i(t) + v_{gi}(t)\ \alpha(t)\ (I - w_i(t))$

Como puede verse, en esta actualización entra en juego la función $v_{gi}(t)$ que indica otra proporción en la que $w(t)$ se acercará a I. La función $v(t)$ puede ser tanto la función (3.3), como la (3.4), o bien alguna otra mas adecuada al problema que se quiera resolver. Con este factor se busca que los vectores de pesos de las neuronas vecinas se acerquen a I en una proporción menor que la neurona ganadora, y que las neuronas más alejadas de la ganadora se acerquen en menor medidas de las neuronas que están más cerca. Este factor, que es función de la cercanía entre una neurona y la neurona ganadora, tiende a cero a medida que la neurona "vecina" a actualizar, se aleja de ella.

Lo habitual es comenzar con el entrenamiento en una vecindad amplia, e ir disminuyéndola a medida que transcurre el entrenamiento. No existe una "receta" que diga cual debe ser el vecindario inicial y como debe ir decreciendo.

Al comenzar el entrenamiento con una vecindad relativamente amplia, y la misma decrece a medida que transcurren las pasadas del entrenamiento, es muy probable alcanzar un estado "estable". Elegir una vecindad amplia que esté presente mucho tiempo en el entrenamiento puede hacer que cuando una neurona gane, otras aprendan constantemente en distintas direcciones, ya que las neuronas pueden formar parte de múltiples vecindarios, con la consecuencia de una incorrecta o muy lenta organización de la red. Por otro lado, una vecindad pequeña desde el comienzo puede causar fallas en la organización y en la preservación de la topología.

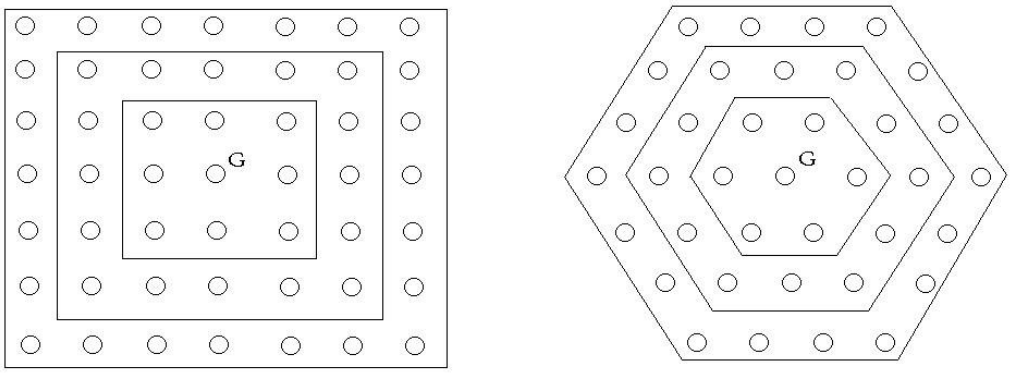


Figura 3.8. A la izquierda se muestra en una grilla cuadrangular la vecindad de tamaño 1, 2 y 3 para la neurona ganadora (G). A la derecha se muestran para una neurona ganadora (G), las vecindades de tamaño 1, 2 y 3 en una grilla hexagonal. (Figura sacadas de Kohonen 1997).

3.3.4. Entrenamiento

El entrenamiento comienza utilizando un vecindario amplio. Para cada patrón de entrada se busca la neurona ganadora y se actualizan los pesos de esa neurona y de sus vecinas. Se puede presentar todo el conjunto de patrones de entrada un número fijo de veces. Otra opción es la de presentar todos los patrones de entrada, al menos una vez, hasta conseguir un error mínimo preestablecido. Por ejemplo, se podría en cada pasada y por cada patrón de entrada, acumularse la diferencia entre *I* y *W*. Al finalizar la presentación de todos los patrones de entrada, si la diferencia acumulada es menor que un umbral establecido a priori, entonces se da por finalizado el algoritmo.

Cada cierto número de pasadas el tamaño de la vecindad va disminuyendo hasta llegar a cero. Con esto se busca que en las primeras pasadas, al tener un amplio vecindario, las neuronas se organicen de manera general, y a medida que el vecindario disminuye, las neuronas participan en menos aprendizajes, ya que solo aprenderá cuando sea ganadora o cuando ganen sus vecinas, que con el correr de las pasadas cada vez son menos. De esa manera se va logrando que represente un pequeño subconjunto de patrones de entrada.

El pseudocódigo del algoritmo para el entrenamiento de una red SOM que no utiliza término de tendencia es el siguiente:

```
Seleccionar el valor de alfa
Repetir
   Para cada patrón de entrada
      Presentar el patrón de entrada a la red.
      Seleccionar a la neurona ganadora según el criterio de
          medición establecido a priori.
      Actualizar el vector de pesos de la neurona ganadora.
      Actualizar los vectores de pesos de las neuronas vecinas
          de la neurona ganadora
   fin para
   Disminuir el tamaño del vecindario si corresponde.
hasta (alcanzar la condición de terminación del algoritmo).
```

3.3.5. Ejemplo

A continuación se ejemplifica el uso de un mapa auto-organizativo para el análisis de los patrones de las base de datos ZOO (ver apéndice 2) cuyos elementos de la capa competitiva se encuentran organizados como una grilla rectangular de ocho filas y seis columnas. La tabla 3.1 representa la información aprendida por la red luego del entrenamiento no supervisado.

Cada celda de la tabla se corresponde con una neurona competitiva de manera que celdas adyacentes indican neuronas competitivas vecinas. En su interior se encuentra la cantidad de patrones representados por la neurona competitiva y la clase a la que pertenece según la información conocida a priori. Por ejemplo, en la posición (4,2) aparece 'C1→3'. Esto debe interpretarse como: "la neurona competitiva ubicada en la posición (4,2) de la estructura de la RN representa a tres patrones que, según la información de entrada, pertenecen a la clase 1.

Tabla 3.1. Resultado del entrenamiento del SOM.

C7→1	0	C6→2	C6→2		C2→3
C7→5	C7→1		C6→2	C2→1	C2→4
	C7→1	C6→2;C7→2		C2→1	C2→1
C1→2	C1→3			C2→1	C2→1
C1→2	C1→2	C5→3		C2→3	C2→5
C1→4	C1→1		C3→3; C5→1	C3→1	
	C1→3	C1→1		C4→4	C4→4
C1→8	C1→10	C1→1	C1→4		C4→5

Analizando la distribución de clases dentro de la tabla puede verse que los mamíferos (clase 1), se encuentran representados por las neuronas de la región inferior izquierda de la capa competitiva y están claramente separados del resto. Lo mismo ocurre con la clase 2 que representa a las aves ubicada en el extremo superior derecho.

La grilla también permite analizar similitudes entre las clases. Por ejemplo, los cuatro patrones de la clase 1 que se encuentran representados por la neurona (8,4) son: el delfín, la marsopa, el león marino y la foca. Todos ellos son mamíferos acuáticos y son los más próximos a los peces (clase 4).

Es importante remarcar que los resultados obtenidos se encuentran estrechamente relacionados con la cantidad de neuronas que forman la arquitectura ya que si el número es inferior al necesario, se producirá una superposición de clases y si es excesivo, se generará una arquitectura innecesariamente grande.

En el apéndice 3 se detalla la implementación de este ejemplo en MatLab.

3.3.6. Bias

La inicialización de los vectores de pesos de las neuronas, antes de comenzar el entrenamiento puede ocasionar, en algunos casos, que una o mas neuronas no resulten nunca ganadoras. Esto sucede por ejemplo, cuando una neurona es inicializada muy lejos de los patrones de entrada. En estos casos, las neuronas que nunca resulten ganadoras estarán ocupando innecesariamente un lugar en la red. Este problema es independiente de la medida de similitud elegida.

Una solución a este problema es la técnica del "bias", o término de tendencia, que propone incrementar la entrada neta de las neuronas que nunca ganan. Cada neurona tiene su propio bias que es modificado durante el entrenamiento. Cada vez que la neurona gana este valor es disminuido y por el contrario, cuando la neurona no logra ganar la competencia, se incrementa. Como el bias se suma a la entrada neta, todas las neuronas tienen posibilidad de ganar y de esa manera organizarse correctamente.

El método de "ayudar" a las neuronas que no ganan, y "retrasar" a las que lo hacen siempre ganan es una buena solución para asegurarse de que todas participen en el entrenamiento. El valor inicial del bias y la manera que influye en la entrada neta de una

neurona pueden ser definidos de la forma más conveniente para la solución del problema. Es importante considerar que el bias debe ser utilizado en la etapa inicial del aprendizaje para que todos los vectores de las neuronas competitivas participen del entrenamiento, pero su intervención debe ir disminuyendo a lo largo de las iteraciones para no interferir con el normal funcionamiento del algoritmo.

A modo de ejemplo, a continuación se muestra una estrategia para implementar el uso del término de tendencia en una capa competitiva en base a la función exp(1-log(*x*)) cuya gráfica se muestra en la figura 3.5. Esta función tiene la particularidad de coincidir con su inversa, es decir, x = exp(1 – log(y)).

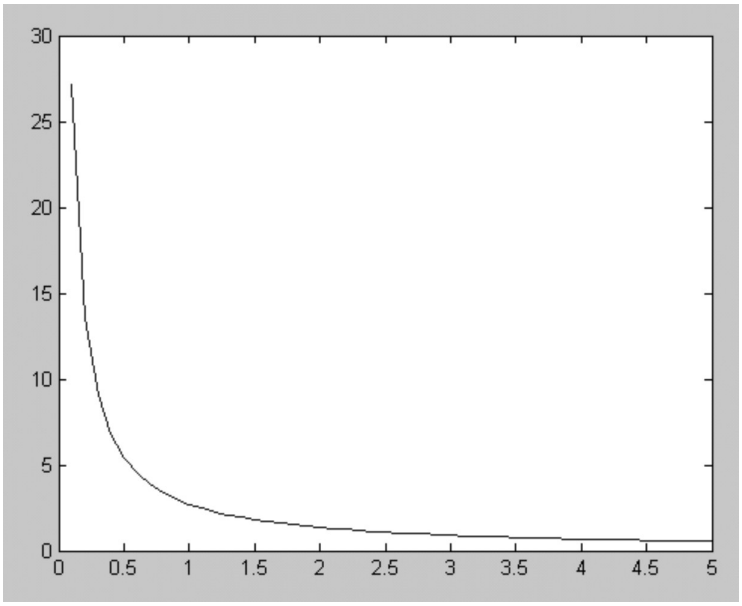


Figura 3.5. Gráfica de la función exp(1- log(*x*)) que se utilizará para calcular el bias. Aquellas neuronas que deseen obtener una ayuda grande deberán tomar valores pequeños en el eje x.

El algoritmo a utilizar es el siguiente

```
Sea N el número de neuronas competitivas de la red.
Seleccionar el valor de alfa.
Determinar el tamaño inicial del vecindario.
Para cada neurona competitiva (i=1..N)
      Bias(i) = exp(1-log( 1/N )) para i=1..N            (1)
Fin Para
Repetir
   Para cada patrón de entrada
   · Seleccionar un vector de entrada aleatorio
   · Ingresarlo a la red
   · Calcular la entrada neta de cada neurona oculta
     Neta(i) = Similitud(patrón, W) + Bias(i)  para i=1..N
   · Calcular la neurona ganadora
     Ganadora = índice del mayor valor de Neta
   · Actualizar el vector de pesos de la neurona ganadora
   · Actualizar los vectores de pesos de las vecinas de la
     ganadora
```

```
   · Para cada neurona competitiva recalcular su bias
     (i=1..N)
        % obtener el valor original
        C(i) = exp(1-log(Bias(i)))                          (2)
        % actualizar para equilibrar las posibilidades
        C(i) = (1-alfa) * c(i)
        Si (ganadora=i),
                c(i) = c(i) +alfa * a                       (3)
        Fin Si
        % nuevo bias
        Bias(i) = exp(1-log(c(i))) - alfa*Bias(i)           (4)
   Fin Para
   Disminuir el tamaño del vecindario si corresponde.
hasta (alcanzar la condición de terminación del algoritmo).
```

En (1) se calcula el valor inicial del bias que será el mismo para todas las neuronas competitivas ya que aun no se ha iniciado ninguna competencia. Este valor será fijado en el valor retornado por la función de la figura 3.5 evaluada en 1/(cantidad de neuronas ocultas). Por ejemplo si la red estuviera formada por tres neuronas competitivas, el bias sería un vector de tres elementos formado por los valores [8,15, 8,15, 8,15]. El valor 8,15 se obtiene de evaluar la función en 1/3, es decir, exp(1-log(*1/3*))=8,15.

Avanzando en el algoritmo, se ingresará el primer patrón y alguna de las neuronas resultará ganadora. A fin de ejemplificar el proceso de modificación del bias se supondrá ganadora a la primera neurona. Luego de actualizar su vector de pesos y el de sus vecinas debe recalcularse el valor del bias para todas las neuronas competitivas de la siguiente forma:

- En (2) se recupera el valor original donde fuera evaluada la función.
  C = [1/3, 1/3, 1/3]

- En (3) Se actualiza este valor de manera de equilibrar las posibilidades (alfa=0,5)
  C(1) = (1-alfa) * c + alfa = (1-0,25) * C(1) + 0,25 = 0,75 * 1/3 + 0,25 = 1/2
  C(2) = (1-alfa) * c = 0,75 * 1/ 3 = 1/4
  C(3) = (1-alfa) * c = 1/4

  Por lo tanto, C= [1/2;1/4;1/4]

- En (4) se calcula el nuevo bias
  Bias(1) = exp(1-log(C(1))) - alfa*Bias(1)
  Bias(1) = exp(1-log(½)) – 0,25 * 8,15

  Bias(2) = Bias(3) = exp(1-log(1/4)) –0,25 * 8,15

  Por lo tanto, Bias =[3.39; 8.83; 8.83]

Nótese que la próxima vez, la primer neurona recibirá menos ayuda que el resto.

Nuevo bias: c = [1/2; 1/4; 1/4]

b = exp(1-log(c)) - alfa*b c=[3,39; 8,83; 8,83]

3.4. MAPAS TOPOLÓGICOS

Las Redes Neuronales trabajan con dos espacios diferentes: el espacio de entrada y el espacio de salida. El primero se encuentra determinado por todos los vectores que ingresan a la red neuronal. El espacio de salida está formado por todas las respuestas obtenidas a partir de la red.

Dada la característica auto-organizativa de algunas redes, resulta conveniente utilizar salidas de 2 o 3 dimensiones, lo cual permite una fácil visualización de la organización de la red y de las relaciones existentes entre los datos de entrada que dieron lugar a la misma.

En el espacio de entrada, los vectores están distribuidos de tal forma que si dos de ellos se encuentran lo suficientemente cerca el uno del otro (de acuerdo a alguna medida de similitud) serán considerados "vecinos" en dicho espacio. De la misma forma, los elementos de una red auto-organizativa, se encuentran interconectados formando algún tipo de estructura. Dado que cada una de las neuronas de la red lleva asociado un vector de pesos, puede decirse que dos de ellas son vecinas si sus respectivos vectores están lo suficientemente cerca el uno del otro. Sin embargo, dos neuronas vecinas conectadas entre si pueden tener pesos sinápticos distantes en el espacio de entrada.

Un mapa topológico o mapa que preserva la topología es aquel en donde las neuronas vecinas tienen pesos sinápticos "vecinos" en el espacio de entrada. Así la relación de vecindad de los datos de entrada se refleja dentro de la red. En la figura 3.9 se muestra el caso de un mapa que no preserva completamente la topología.

La idea de mapa topológico o mapa que preserva la topología se originó a partir de distintos estudios biológicos que permitieron comprobar la existencia de sensores que están formados por este tipo de mapas. A modo de ejemplo puede citarse: el mapa retinotópico de la corteza visual (Hubel &Wisel 1974; Blasdel & Salama 1986), la corteza somatosensorial en la piel (Kaas *et al.* 1979), y el mapa tonotópico en la corteza auditiva (Suga & O'Neill 1979) entre otros.

En la práctica, los mapas que preservan la topología han sido aplicados con éxito en distintas áreas como en el reconocimiento de voz (Kohonen *et al.* 1984; Kohonen 1990; Naylor & Li 1988), procesamiento de imágenes (Nasrabadi & Feng 1988) y en la robótica (Ritter & Schulten 1996; Martinetz *et al.* 1990).

En la figura 3.10 se muestra un ejemplo de cómo se organizan las neuronas durante el entrenamiento del SOM cuando la estructura que forman es rectangular, y en la figura 3.11 se muestra otro ejemplo, pero con una estructura lineal. En ambos casos los vectores de pesos de las neuronas fueron inicializadas con valores dentro de un pequeño radio en el centro del espacio de entrada.

La capacidad de auto-organización de la red es la característica más importante de los SOM. Como se puede observar en las figuras 3.10 y 3.11, neuronas vecinas en la red, son representantes de patrones de entrada similares entre si en el espacio de entrada. A esta característica se la conoce como preservación de la topografía, ya que tanto el espacio de entrada como el de la salida (la red) son topológicamente iguales. En la

figura 3.12 se puede ver otro ejemplo de auto-organización en un espacio de entrada tri-dimensional, y cuan parecidos son, topológicamente, el espacio de entrada y el espacio de salida.

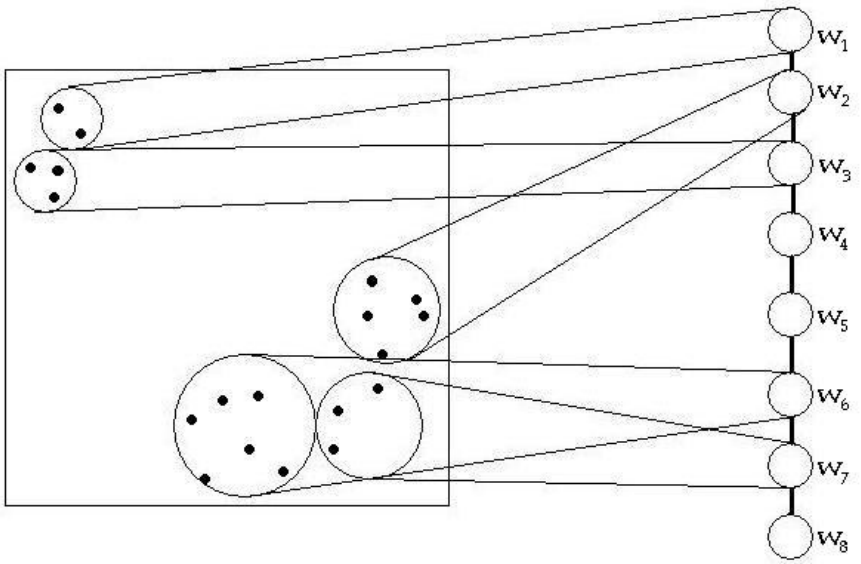


Figura 3.9. A la izquierda está graficado un espacio de entrada 2-dimensional. Los puntos negros representan los patrones de entrada. A la derecha una red de una dimensión. En este mapa se puede ver como w_6 y w_7, vecinas en el espacio de salida, mapean regiones vecinas en el espacio de entrada. Lo mismo ocurre con w_1 y w_3, quienes están dentro de una misma vecindad, tanto en el espacio de entrada como en el de salida. w_2, en el espacio de salida, pertenece a la vecindad de w_1 y w_3, pero mapea, en el espacio de entrada, una región vecina a w_6 y w_7. Por este último caso, se dice que el mapa no preserva completamente la topología.

Como se observa en la figura 3.12, algunas neuronas están ubicadas fuera del espacio de entrada, y otras aparecen como vecinas, cuando en realidad no lo son. La figura 3.13 muestra tres ejemplos bi-dimensionales donde el espacio de entrada es una forma irregular o disjunta, y como queda la estructura de la red luego de un entrenamiento. En la misma figura se puede observar claramente como algunas neuronas son ubicadas fuera del espacio de entrada, y por lo tanto no representan a ningún patrón de entrada, y como otras neuronas aparecen como vecinas, cuando los patrones de entrada a los que representan no lo son. Para solucionar estos problemas, se han propuesto varias soluciones. Por ejemplo, en Kohonen 1997 se describe una solución llamada “minimal spanning tree” (MST), que utiliza un concepto de vecindad más complejo que el definido por una simple medida, como lo puede ser la distancia Euclidea.

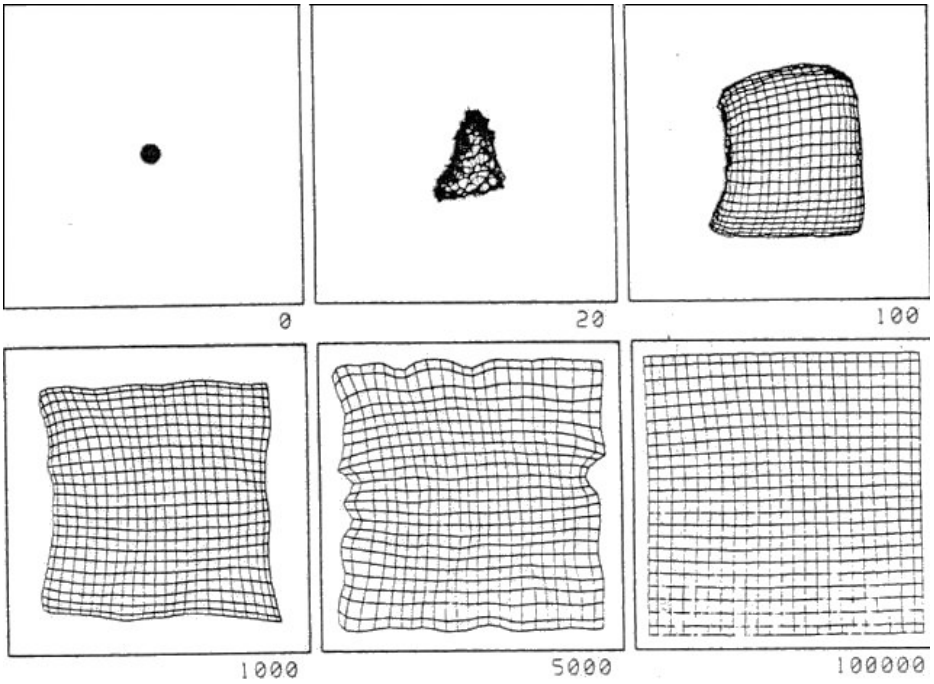


Figura 3.10. El cuadrado representa un espacio de entrada. Los vectores de pesos de las neuronas de una red, con una arquitectura rectangular, fueron inicializados con valores dentro de un pequeño círculo en el centro del espacio de entrada. Se puede observar como los vectores de pesos se van organizando, y la estructura va tomando la forma del espacio de entrada, a medida que el entrenamiento avanza. Se muestran el estado inicial, y los estados luego de 20, 100, 1000, 5000 y 100000 pasadas. (Figura sacada de Kohonen 1997).

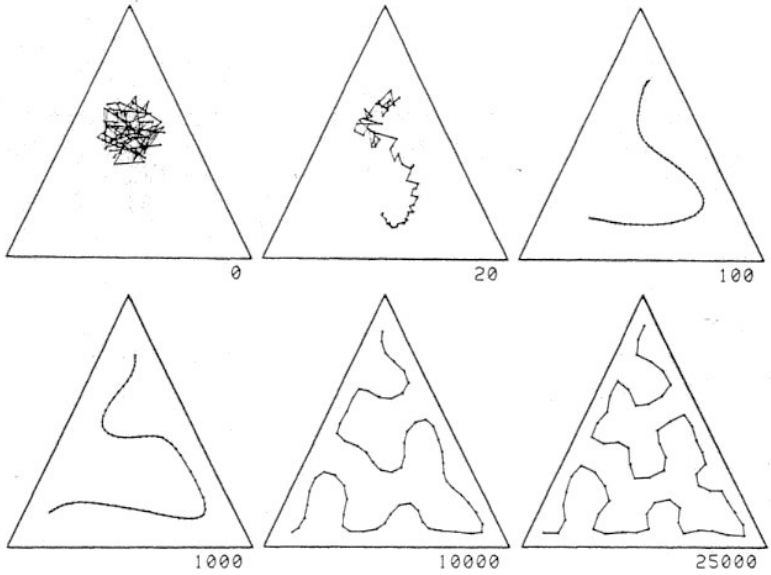


Figura 3.11. El triángulo representa un espacio de entrada. Los vectores de pesos de las neuronas de una red, con una arquitectura lineal, fueron inicializados con valores dentro de un pequeño círculo en el centro del espacio de entrada. Se puede observar como los vectores de pesos se van organizando, y la estructura va tomando la forma del espacio de entrada, a medida que el entrenamiento avanza. Se muestran el estado inicial, y los estados luego de 20, 100, 1000, 10000 y 25000 pasadas. (Figura sacada de Kohonen 1997).

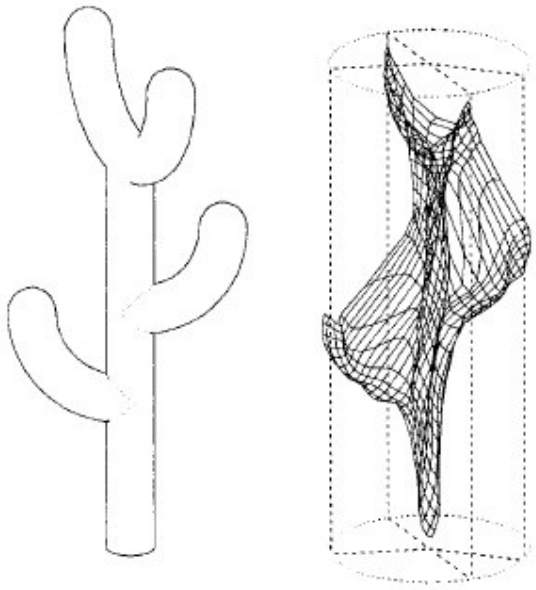


Figura 3.12. Se puede ver a la izquierda un espacio de entrada tri-dimensional, donde los patrones de entrada tienen la forma de un cactus. A la derecha se muestra la organización de las neuronas luego del entrenamiento. Se puede apreciar que, si bien existen algunas deformidades, la organización de las neuronas respeta casi en su totalidad la forma del cactus del espacio de entrada. (Figura sacada de Kohonen 1997).

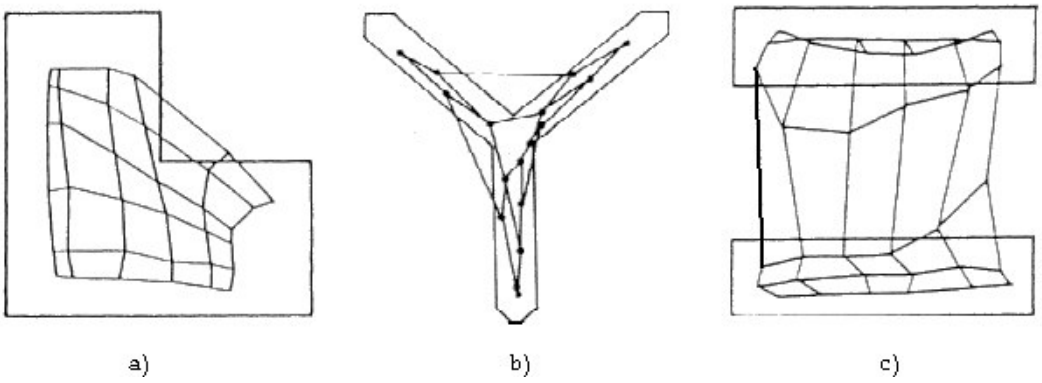


Figura 3.13. En a) se muestra una red rectangular de 25 neuronas dentro de un espacio de entrada con forma de "L". En b), una red rectangular de 16 neuronas dentro de un espacio de entrada con forma de "Y", y en c) se muestra una red rectangular de 36 neuronas dentro de un espacio de entrada disjunto, formado por dos rectángulos. En los tres ejemplos de pueden ver neuronas fuera del espacio de entrada, y vecindades entre neuronas que mapean patrones de entrada no-vecinos en el espacio de entrada. (Figuras sacadas de Kohonen 1997).

3.5. REGIONES DE VORONOI

Una vez finalizado el entrenamiento de un SOM, los vectores de pesos de las neuronas se corresponden con un subconjunto de patrones de entrada. Se dice que ese vector de pesos es el representante de aquellos patrones de entrada con los cuales se corresponde.

Una neurona i con un vector de pesos w_i será representante de un patrón de entrada I cuando w_i sea el vector de pesos (entre todos los vectores de pesos de todas las neuronas de la red) que mas cerca esté de I, medido con la misma magnitud utilizada para determinar la neurona ganadora que se uso durante el entrenamiento. Por ejemplo, cuando la magnitud utilizada es la distancia Euclidea, dicha neurona será representante de todos los patrones de entrada que estén dentro de la hiper-esfera con centro w_i, y con un radio r_i. Así, cada neurona será centro de una hiper-esfera, donde los radios de las mismas pueden ser distintos. Cada r_i deberá ser lo suficientemente grande, como para que todas las entradas estén por lo menos dentro de una hiper-esfera. Puede suceder que varias hiper-esferas se intersecten en algunas regiones, dada una hiper-esfera con centro w_1 y radio r_1, y otra con centro w_2 y radio r_2, y d es la distancia Euclidea entre w_1 y w_2, si $d < r_1 + r_2$ entonces ambas hiperesferas se intersectan en alguna región. En estos casos un patrón de entrada puede estar dentro de dos o más hiper-esferas, pero solo estará representado por una sola neurona, aquella en que su vector de pesos w_i esté más cerca del patrón de entrada.

Se define como región de Voronoi, también conocido como poliedro de Voronoi, a la porción v_i de la hiper-esfera donde todo punto de v_i este mas cerca de w_i que de cualquier otro w_j. Entonces, se dice que el vector de pesos w_i de una neurona i mapea la región de Voronoi v_i. En la figura 3.14 se muestra, en un espacio de entrada bi-dimensional, las circunferencias correspondientes a cada vector de pesos y como quedan delimitadas las regiones de Voronoi.

Algebraicamente, las regiones de Voronoi v_i están definidas por:

$$v_i = \left\{ v \in V \mid \left\| v - w_i \right\| \leq \left\| v - w_j \right\| \forall j \right\},$$

donde V es el conjunto de patrones de entrada.

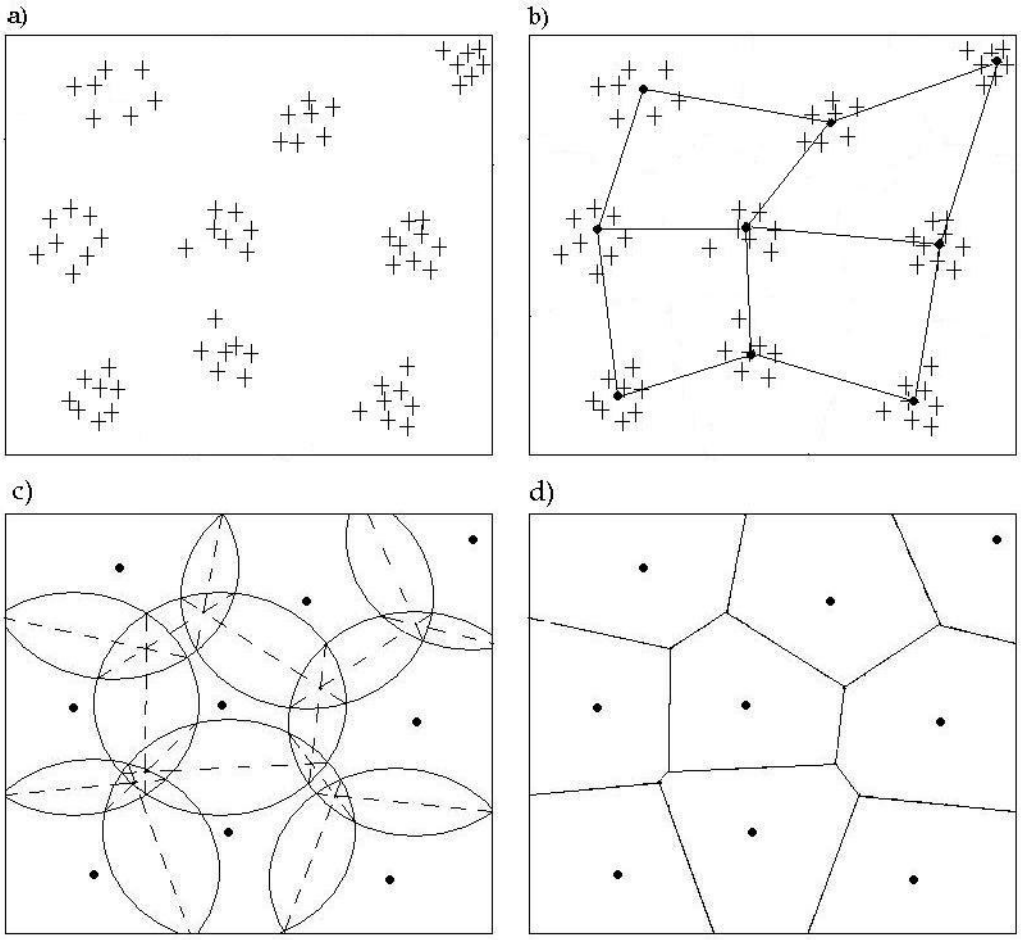


Figura 3.14. En a) se presenta un espacio bi-dimensional, donde las cruces representan a los patrones de entrada. En b) se muestra una red rectangular de nueve neuronas, una vez finalizado el entrenamiento. Los puntos negros representan a los vectores de pesos de las nueve neuronas. En c) aparecen solo los vectores de peso de las neuronas, los cuales son centro de una circunferencia (hiper-esfera de dos dimensiones), la cual está dibujada. Las líneas rayadas "cortan a la mitad" aquellas regiones donde se intersectan dos circunferencias, e indican los límites de las regiones de Voronoi. En d) se muestra como quedan delimitadas las regiones o poliedros de Voronoi para cada vector de pesos. Las líneas continuas se corresponden con los trazos rayados de c).

3.6. USOS Y APLICACIONES DEL SOM

Las redes de aprendizaje no supervisado presentan una convergencia más lenta que las que utilizan el valor de salida esperado para adquirir conocimiento. Sin embargo, es su capacidad de descubrir las similitudes entre los patrones de entrada lo que las hace una herramienta útil para resolver problemas para los cuales no se dispone de ninguna información relacionada con la solución a obtener.

Los patrones de características similares forman lo que se conoce como agrupamientos (clusters). Por ejemplo en la figura 3.12c se muestra un espacio de entrada con dos clusters bien definidos (los dos rectángulos). Si bien un cluster es un agrupamiento de patrones muy similares, dependiendo de cuan "detallista" sea construida y entrenada la red, varios cluster podrían estar formados por patrones con las

características muy parecidas. Si la red es muy "detallista", pequeñas diferencias entre los patrones harían que en la red se formen mas de un cluster con patrones similares.

El SOM es utilizado en varios tipos de aplicaciones distintos:

- Análisis de imágenes: visión por computadora, impresión, transmisión de imágenes, imágenes médicas, sensores remotos, y reconocimiento de objetivos. Para propósitos de reconocimiento de patrones, las imágenes deben ser segmentadas y etiquetadas (Koh *et al.* 1993). El análisis de texturas es otra tarea muy común (Ghosal & Mehrotra 1993). En el tratamiento de imágenes médicas, se usó en la clasificación de tumores cerebrales (Vercauteren *et al.* 1990 citado en Kohonen 1997).

- Lectura de caracteres ópticos: en esta área se ha utilizado al SOM como herramienta para resolver problemas como detección y orientación de los caracteres (Morris *et al.* 1989), y reconocimiento de caracteres escritos a mano (Idan & Chevallier 1991).

- Reconocimiento y análisis de voz: las tareas mas comunes en este campo son el reconocimiento de una palabra simple (Huang & Kuh 1992), segmentación de una frase (Anderson *et al.* 1990 citado en Kohonen 1997), y reconocimiento de fonemas para el tratamiento del habla natural continua (Danielson 1990).

- Estudios acústicos y musicales: el SOM se ha utilizado para la segmentación de características acústicas (Hernáez *et al.* 1993), y el análisis de patrones musicales (Gjerdingen 1989).

- Robótica: algunos investigadores sostienen que los robots autónomos son la próxima meta para las redes neuronales. El SOM se ha utilizado satisfactoriamente en el control del robot y coordinación visual (Graf & LaLonde 1989), navegación sin colisiones (Walker *et al.* 1993), y en la coordinación en movimientos en la búsqueda de un objetivo (Coiton *et al.* 1991 citado en Kohonen 1997).

- Química: se ha utilizado al SOM para la clasificación de estructuras atómicas complejas (Zell *et al.* 1994), y en la clasificación de proteínas (Ferrán & Ferrara 1991 citado en Kohonen 1997).

- Física: el SOM se ha utilizado en la clasificación de eventos sísmicos (Raiche 1991 citado en Kohonen 1997).

- Diseños de circuitos electrónicos: el diseño de complejos circuitos integrados necesita una optimización para la colocación adecuada de cada componente, el SOM ha sido utilizado para estos fines (Shen *et al.* 1992).

- Procesamiento de datos: el SOM ha resultado útil como herramienta en el análisis de datos económicos y financieros (Wilson 1994 citado en Kohonen 1997), en la clasificación de software reusable (Merkl 1993), en Kohonen *et al.* 2000 se describe el uso del SOM como herramienta para la organización de grandes cantidades de documentos, de acuerdo a similitudes textuales.

- Lingüística y problemas de inteligencia artificial: una tarea muy común en esta área es el parseo de expresiones y sentencias (Scholtes & Bloembergen 1992), otra tarea en donde se usó el SOM es en el reconocimiento de sentencias naturales (Pedrycz & Card 1992), también fue utilizado con éxito en la adquisición de conocimientos en métodos de inteligencia artificial (Ultsch *et al.* 1991).

- Problemas matemáticos: el SOM se ha utilizado para tareas como aproximaciones de funciones (Hannah *et al.* 1994 citado en Kohonen 1997), y predicción de series de tiempo (Barreto & Araújo 2004).

- Investigación neurofisiológica: el SOM y las redes neuronales artificiales en general fueron estudiadas como modelos para la interpretación de las funciones cerebrales. En este campo el SOM ha sido utilizado como un modelo fisiológico de la corteza (Kohonen 1994 citado en Kohonen 1997), y en teoría temporal de la sensación auditiva (Kita & Nishikawa 1993 citado en Kohonen 1997).

3.7. PRESERVACIÓN DE LA TOPOLOGÍA DE LOS DATOS

Una de las características mas importantes de los SOM es la de poder mapear en dos dimensiones patrones de entrada n-dimensionales. Esta característica permite una fácil visualización del agrupamiento o similitudes que existen en el conjunto de patrones de entrada.

El hecho de mapear espacios n-dimensionales a uno bi-dimensional lleva a una posible e inevitable distorsión de la topología. En la figura 3.15 se muestra un ejemplo de un espacio de entrada bi-dimensional con nueve clusters bien definidos, y un espacio de salida de una dimensión. La red tiene nueve neuronas, donde cada una representa a un cluster. En la misma figura se muestran cuatro posibles estados finales para la red, suponiendo que luego del entrenamiento cada una de las neuronas mapea a un cluster distinto. Se puede observar que en los cuatro posibles estados finales de la red, se encuentran algunas neuronas que están alejadas entre si en el espacio de salida, mientras que en el espacio de entrada son vecinas, y que no existe manera de ordenar las nueve neuronas de manera tal que se conserve perfectamente la topología. Si bien el caso presentado en la figura 3.14 es hipotético, en la práctica suceden casos similares. Esta distorsión es mas amplia cuando el espacio de entrada n-dimensional es muy grande. Por consiguiente cualquier medición encontrará estos casos como “malos” y afectará negativamente al valor final del grado de preservación.

Durante el entrenamiento es cuando se comienza a distorsionar la preservación de la topología, causando “aprendizajes incompletos”, ya que si dos neuronas vecinas en el espacio de entrada, no están cerca en el espacio de salida, cuando una de las dos resulte ganadora, la otra no aprenderá cuando debió haberlo hecho.

Es muy importante tener en cuenta lo explicado anteriormente al momento de trabajar con mapas auto-organizativos, y saber que es casi imposible lograr un mapa que preserve la topología en su totalidad. Una vez finalizado el entrenamiento, habrá que estudiar con cuidado el mapa final, ya que dos neuronas alejadas entre si podrían tener características similares.

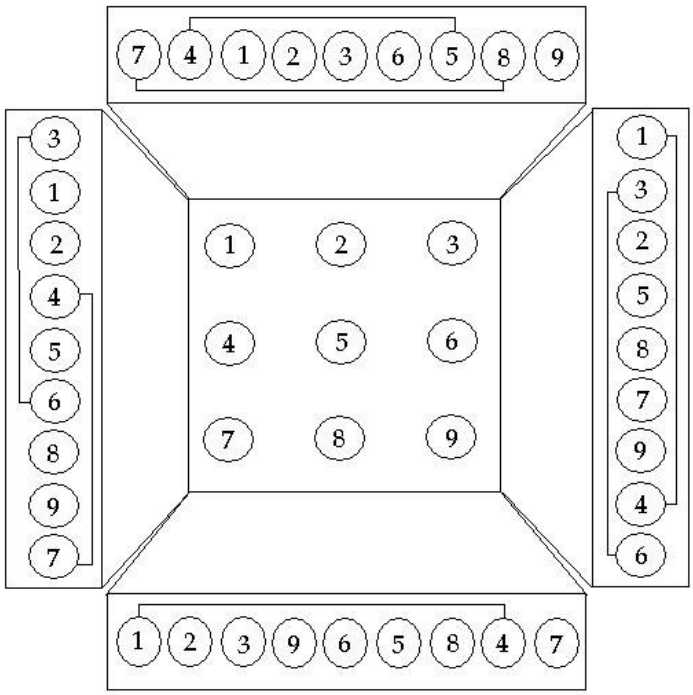


Figura 3.15. En el centro, un espacio de entrada bi-dimensional con nueve clusters bien definidos y numerados. En los bordes cuatro posibles resultados luego del entrenamiento. Cada red tiene nueve neuronas, y cada neurona tiene el número de cluster que mapea. En los cuatro casos están marcados los pares de neuronas que están muy alejadas, y cuyos respectivos clusters son vecinos en el espacio de entrada.

3.7.1. Medidas de performance de la red neuronal

Las mediciones de performance sobre las redes neuronales tiene sentido tanto en análisis teóricos, como en la práctica, a los defectos de poder determinar el grado de eficiencia del entrenamiento. Una medida de organización debería cuantificar el proceso de auto-organización durante el entrenamiento. También debería cuantificar la calidad del mapeo de la red en el espacio de entrada (Polani 1997).

3.7.2. Preservación de la topología

Cada neurona *i* de la red lleva asociado un vector prototipo, w_i , que define el campo receptivo o poliedro de Voronoi V_i formado por todos los datos de entrada para los cuales w_i es el más parecido según una medida de similitud definida a priori.

Se dice que un mapeo preserva la topología si dos unidades competitivas vecinas en la red neuronal poseen campos receptivos adyacentes en el espacio de los datos de entrada.

De esta forma, si la red preserva la topología adecuadamente, patrones de entrada similares serán mapeados dentro de neuronas cercanas en la red.

Villmann *et al.* 1997 propone una definición y medición exacta para la preservación de la topología en un SOM. Una red preserva la topología si el mapeo del espacio de

entrada en el espacio de salida, y viceversa, preserva la vecindad. Es decir, si dos neuronas en el espacio de salida son vecinas directas entre si, entonces sus vectores de pesos sinápticos deben ser vecinos en el espacio de entrada. Y por otro lado, dos vectores de pesos sinápticos vecinos en el espacio de entrada, deberán pertenecer a dos neuronas vecinas en el espacio de salida. La relación de vecindad en el espacio de salida está dada según la estructura de la red elegida. La relación de vecindad en el espacio de entrada está dada por los poliedros de Voronoi.

Los mapas auto-organizativos, justamente por su carácter de auto-organizativos, tienden a ordenar estímulos parecidos en lugares cercanos dentro de la red, de esta manera preservan la topología del espacio de entrada. En la práctica esto no se cumple completamente. Para medir el grado en que un mapa preserva la topología, existen varias métricas topográficas y serán descriptas a continuación.

3.7.3. Métricas topográficas

Como se puede ver en la figura 3.9, una única neurona puede causar que un mapa no preserve completamente la topología. Ante este caso podemos ver que un mapa podría preservar completamente la topología, o podría hacerlo parcialmente.. En el peor de los casos, el mapa puede no preservar la topología en absoluto, es decir no tendría ningún par de neuronas vecinas cuyos pesos sinápticos sean vecinos en el espacio de entrada. Para poder distinguir estos casos y ver en que grado un mapa preserva la topología, existen distintas métricas topográficas.

3.7.3.1. Una métrica simple

Una métrica muy simple para medir el grado en que un mapa preserva la topología es descripta y usada en Hsu & Halgamuge 2001. Esta medida calcula el porcentaje de distorsión que tiene el mapa. La idea básica consiste en: si, dado un patrón de entrada, la neurona ganadora para ese patrón y la segunda neurona ganadora no son vecinas directas entonces el mapa tiene una distorsión. Así el error topográfico queda definido como:

$$TE = \frac{\sum_{i=0}^{N} \begin{matrix} 1, si \left\| r_{bi} - r_{sbi} \right\| > 1 \\ 0, en\_otro\_caso \end{matrix}}{N}$$

donde N es la cantidad de patrones de entrada. r_{bi} es la neurona ganadora para la entrada i, y r_{sbi} es la segunda neurona ganadora para la entrada i. $\| \|$ mide la distancia que existe entre dos neuronas dentro del mapa, siendo 1 cuando las dos neuronas son vecinas directas.

Como puede verse en la ecuación el error topográfico será 0 cuando para todos los patrones de entrada sus respectivas neuronas ganadoras y segundas ganadoras sean vecinas directas y será igual a 1 cuando no exista una entrada cuya neurona ganadora y segunda neurona ganadora sean vecinas directas. En conclusión, el error topográfico será 0 cuando el mapa preserve completamente la topografía y 1 en un mapa con un completo desorden.

3.7.3.2. Producto topográfico

En Bauer & Pawelzik 1992 se describe una métrica más compleja que la descripta en la sección anterior. Esta métrica tiene en cuenta las vecindades de orden k, con $k \geq 1$. Un valor de 0 indica una preservación perfecta de la topología, un valor negativo indica una dimensionalidad muy pequeña en el espacio de salida, y un valor positivo indica una dimensionalidad demasiado grande en el espacio de salida.

En el producto topográfico se tiene en cuenta el orden de vecindad para las neuronas en el espacio de salida, y el orden de vecindad para los datos de entrada en el espacio de entrada. Definiendo tasas de orden de vecindad se determina si el orden de vecindad entre dos vectores de pesos sinápticos en el espacio de entrada es el mismo que poseen sus respectivas neuronas en el espacio de salida. Luego de normalizar las tasas se obtiene un valor resultado para una neurona j y un orden de vecindad k. Este valor indica si para la neurona j, y en un orden de vecindad k, el mapa preserva la topología o no.

Si se calculara un valor para cada neurona y para cada orden de vecindad entonces se tendría un histograma que muestra que ordenes de vecindad son los que menos preservan la topología. Para dar un único valor que indique el grado de preservación de la topología del mapa, se define el producto topográfico como:

$$P = \frac{1}{N(N-1)} \sum_{j=1}^{n} \sum_{k=1}^{n-1} \log(P_3(j,k))$$

donde N es la cantidad de neuronas de la red, y P_3 es el valor que indica para una neurona y un orden de vecindad, en cuanto se preserva la topología, y está definido como:

$$P_3(j,k) = \left(\prod_{l=1}^{k} Q_1(j,l) Q_2(j,l) \right)^{1/2k}$$

donde

$$Q_1(j,k) = \frac{d^V(w_j, w_{n_k^A(j)})}{d^V(w_j, w_{n_k^A(j)})}$$

$$Q_2(j,k) = \frac{d^A(j, n_k^A(j))}{d^A(j, n_k^V(j))}$$

donde $n_k^A(j)$ indica el k-ésimo vecino más cercano del nodo j en el espacio de salida, y $n_k^V(j)$ indica el k-ésimo vecino más cercano de la neurona j en el espacio de entrada.

3.7.3.3. Función topográfica

Las dos métricas descriptas en las secciones anteriores solo tienen en cuenta la vecindad de las neuronas y las de sus pesos. En Villmann *et al.* 1994 se describe una función topográfica que además de la vecindad de las neuronas y sus pesos sinápticos tiene en cuenta el conjunto de patrones de entrada.

El producto topográfico no utiliza directamente los vectores de pesos de las neuronas para determinar una vecindad, en su lugar utiliza el campo receptivo o la región de Voronoi correspondiente.

La función topográfica esta definida como:

$$\Phi = \sum_{j \in A} f_j(k)$$

donde $f_j(k)$ indica la cantidad de neuronas i que tienen un campo receptivo R_i adyacente a R_j, y al mismo tiempo tienen una distancia k hasta j en la red. Φ será igual a cero cuando la red preserve completamente la topología.

Para determinar si dos campos receptivos son adyacentes entre si, esta función utiliza el método propuesto en Martinetz 1993, utilizando una matriz de adyacencia para determinar las conexiones entre dos campos receptivos.

4. MAPAS AUTO-ORGANIZATIVOS DINÁMICOS

La gran desventaja del SOM radica en la definición estática de su arquitectura ya que la cantidad y modo de conexión de las neuronas que la forman deben ser definidos a priori, antes de comenzar con el entrenamiento, condicionando de esta manera la eficiencia y eficacia de la red.

Como forma de resolver esto se han propuesto soluciones alternativas denominadas Mapas Auto-organizativos Dinámicos los cuales permiten incorporar y/o eliminar neuronas durante el entrenamiento. En estos casos, el crecimiento de la estructura se realiza en función de la cantidad de veces que gana cada neurona. Aquellas que lo hacen un mayor número de veces son reforzadas con nuevas vecinas con el objetivo de distribuir de manera más adecuada los vectores prototipos. Un mapa auto-organizativo dinámico (o creciente) es una versión extendida del clásico SOM de Kohonen con la capacidad de expandir su estructura de manera controlable.

Para trabajar con este tipo de arquitecturas las neuronas de la red deben tener posibilidad de incorporar o eliminar vecinos dinámicamente. Este aspecto, sumado al hecho de que la sobrecarga de las neuronas es medida en forma aislada puede llevar a la pérdida de preservación de la topología. Esto se manifiesta en la ubicación de representantes de agrupaciones similares (neuronas competitivas) en posiciones muy distantes dentro de la red neuronal.

En general, la mayoría de las estrategias existentes para la definición de mapas auto-organizativos dinámicos poseen las siguientes características (Fritzke 1996):

- La estructura de la red es un grafo formado por neuronas competitivas conectadas entre si. La forma de conexión es regular y tiene fundamental importancia para la visualización de la reducción del espacio de entrada.

- Cada neurona de la red se corresponde con un vector prototipo, el cual busca representar un conjunto de datos de entrada similares. La medida de similitud a utilizar es dependiente del problema.

- El entrenamiento se realiza a través de un proceso competitivo donde las neuronas buscan representar a los datos de entrada. Para cada dato se evalúa su parecido con el vector prototipo de cada neurona, considerando ganadora a la más próxima. La adaptación se aplica principalmente a la neurona ganadora y en menor medida a su entorno cercano. Esto es lo que permite ir corrigiendo la estructura de manera que se preserve la topología.

- En cada paso de adaptación la información del error local es acumulada en la neurona ganadora. Esto tiene como objetivo evitar que un mismo elemento de la red acumule la representación de la mayoría de los patrones de entrada. El cálculo del error es dependiente de la aplicación.

- La información de error acumulada se utiliza para determinar donde deben insertarse nuevas unidades en la red. Cuando se realiza una inserción, la

información del error es redistribuida localmente evitando nuevas inserciones en el mismo lugar.

Puede observarse en la lista anterior que las tres primeras características corresponden al SOM definido por Kohonen (Kohonen 1990, 1997) mientras que las dos últimas se relacionan con la necesidad de identificar el lugar donde deben insertarse nuevos elementos en la arquitectura.

4.1. VENTAJAS RESPECTO AL SOM

Los mapas auto-organizativos dinámicos ofrecen la posibilidad de usar una medida de error dependiente del problema para determinar cuando deben insertarse nuevas neuronas. La inserción ocurre solo cuando es necesario.

Además, la necesidad de modificar durante el entrenamiento la cantidad de elementos de la red lleva a representar en forma separada, para cada neurona, el vector prototipo correspondiente y sus vecinos más cercanos.

Esto no ocurre en los SOM, ya que la cantidad de neuronas de la estructura y su forma de conexión se encuentran definidas a priori y por lo tanto sólo resta identificar los valores de los vectores prototipos correspondientes.

4.2. MÉTODOS EXISTENTES

Se han propuesto una cantidad importante de métodos basados en la filosofía de los mapas auto-organizativos dinámicos (Rodrigues & Almeida 1990; Xu 1990; Ritter 1991; Blackmore & Miikkulainen 1992; Bruske & Sommer 1995, citado en Fritzke 1996).

En las próximas secciones se describirán algunos de los métodos existentes: Growing cell structure (Fritzke 1993), Growing neural gas (Fritzke 1995), Grow when required (Marsland *et al.* 2002), Double self-organizing feature map (Su & Chang 2001), Growing self-organizing map (Bauer & Villmann 1997) y Growing self-organizing map (Alahakoon *et al.* 2000).

4.2.1. Growing Cell Structure (GCS)

En 1993, Brend Fritzke desarrolló una arquitectura de red neuronal auto-organizativa dinámica que, a diferencia de la mayoría de los métodos existentes, posee dos variantes de aprendizaje: no-supervisado y supervisado (Fritzke 1993). Esta última es una combinación de la variante de aprendizaje no-supervisado con una función base radial (Moody & Darken 1988).

Este modelo utiliza la figura de hipertetraedro de una dimensionalidad k definida a priori. Un hipertetraedro de dimensión k es el poliedro más simple que puede formarse con k+1 vértices. Ejemplo de hipertetraedros para $k \in \{1,2,3\}$ son líneas, triángulos y tetraedros.

En GCS la arquitectura comienza con un único hipertetraedro. Al igual que el SOM propuesto por Kohonen, para cada vector de entrada se elige la neurona más cercana. La neurona ganadora y sus vecinas forman parte del aprendizaje. La diferencia con SOM es que solo las vecinas directas forman parte del aprendizaje. Además, el peso de adaptación es constante en todo el entrenamiento.

Luego de un número λ de pasos de adaptación, se determina la unidad *q* que posea la máxima frecuencia de señal relativa y se inserta una nueva neurona dividiendo el arco que separa a *q* de su vecino directo más lejano. Luego se agregan las conexiones necesarias para preservar la forma de la arquitectura de manera que siempre esté integrada por hipertetraedros de dimensión *k*. Este proceso de reconexión es muy sencillo. Si la nueva neurona fue insertada entre *q* y *f*, sus vecinos directos serán *q*, *f* y todos los vecinos comunes de *q* y *f*.

Dado que GCS posee una dimensionalidad definida a priori puede emplearse como herramienta de visualización utilizando *k* con valor 2 o 3.

Pseudo código:

Mientras no se alcance el tamaño deseado **hacer**
  Repetir λ veces
    Elegir una entrada ξ.
    Determinar la neurona ganadora *s*.
    Incrementar un contador de éxito para s y sus vecinos directos
    $\Delta\omega_s = \varepsilon_b(\xi - \omega_s)$
    $\Delta\omega_c = \varepsilon_n(\xi - \omega_c)$ (para todo $c \in N_s$, donde N_s es el conjunto de vecinos directos de s)
    Incrementar en 1 el contador de señal para *s*.
    Decrementar todos los contadores de señal en una proporción de α.
      $\Delta\tau_c = -\alpha\,\tau_c$
  Fin repetir
  Determinar la neurona q con máximo frecuencia de señal relativa
    $h_q = \tau_q / \sum_{j \in A} \tau_j$
  Determinar la neurona f, vecina directa de q, tal que sea la más alejada de q.
  Se inserta una nueva neurona r entre q y f. Esta nueva neurona se conecta a las otras neuronas de tal forma que en la estructura solo quedan hipertetraedros de dimensión k.
  Se inicializa el vector de pesos sináptico de r
    $\omega_r = 0.5(\omega_q + \omega_f)$
  Se inicializa el contador de señal de r
    $\tau_r = -\sum_{c \in Nr} \Delta\tau_{c,}$
      donde $\Delta\tau_c = [\,(\,|F_c^{(new)}| - |F_c^{(old)}|\,) / F_c^{(old)}]\,\tau_c$ (para todo $c \in N_r$), donde $|F_c|$ es el volumen de la región de Voronoi de *c*.
Fin mientras

Este método también posee una variante para la eliminación de neuronas obsoletas. En algunos casos, pueden existir clusters bien separados entre si, y consecuentemente, dos neuronas, cada una en un cluster distinto, podrían estar conectadas, ver figura 4.1. Para determinar si una neurona está "fuera" de algún cluster se puede utilizar la frecuencia de señal relativa, calculando:

$$\rho_c = h_c / | F_c |$$

si ρ_c está por debajo de un umbral entonces la neurona c puede ser eliminada. Este chequeo puede hacerse periódicamente durante el entrenamiento.

Los autores del GCS desarrollaron una variante del método para el aprendizaje supervisado (Fritzke 1993). Otras variantes de este método pueden encontrarse en Fritzke 1997, Wang *et al.* 2000, Azuaje *et al.* 2000 y Hodge & Austin 2001.

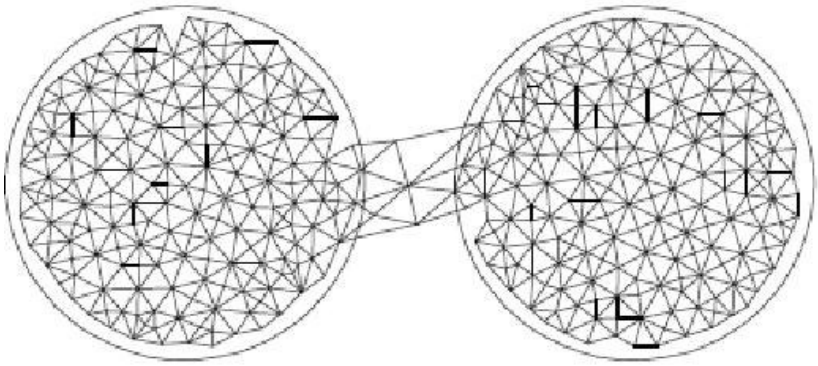


Figura 4.1. En el espacio de entrada bi-dimensional hay dos clusters bien definidos, representados por los círculos. También se representan las neuronas una vez finalizado el entrenamiento y las conexiones a sus vecinos directos. Entre los dos círculos se pueden observar que existen neuronas que están "fuera" de los clusters y que podrían haber sido eliminadas. Esta figura fue sacada del trabajo de Fritzke 1993.

4.2.2. Growing Neural Gas (GNG)

El método GNG (Fritzke 1995) no impone ninguna restricción sobre la forma de conexión entre las neuronas competitivas. Su funcionamiento combina el mecanismo de crecimiento de GCS y la generación de la topología a través del aprendizaje de Hebb competitivo, CHL (Competitive Hebbian Learning), propuesto por Martinetz (Martinetz & Schulten 1991; Martinetz 1993; Martinetz *et al.* 1993).

Este método de aprendizaje no cambia los valores de los vectores de referencia; sólo genera conexiones entre neuronas vecinas de manera de construir un grafo que preserva óptimamente la topología. En particular, las conexiones finales del grafo se corresponden con la triangulación de Delaunay del conjunto de vectores de referencia dados.

Para combinar ambas tareas, GNG propone aplicar la generación de conexiones según CHL incorporando el concepto de edad a cada conexión. En cada paso de adaptación, además de corregir el vector prototipo asociado a la neurona ganadora y de incrementar su valor de error como se describió previamente, se incrementan las edades

de los arcos que la conectan con sus vecinos directos. Cuando la edad de una conexión alcanza un determinado umbral la conexión se elimina. Las neuronas que pierden todas sus conexiones también son eliminadas.

El proceso de crecimiento de la estructura está regulado por un parámetro λ que representa el umbral de pasos de adaptación que debe esperarse para insertar una nueva neurona a la estructura. Alcanzado este umbral se determina la neurona *q* que posea el mayor error acumulado. Luego se determina entre los vecinos de *q* la neurona *f* con mayor valor de error acumulado. Se crea entonces una nueva neurona *r* entre *q* y *f*. Se conecta *r* con *q* y *f* y se elimina la conexión directa entre *q* y *f*. Finalmente se reducen los errores acumulados de *q* y *f* en una fracción indicada a priori y se calcula el error de *r* como el promedio de los errores de *q* y *f*.

El proceso completo se repite hasta que se haya alcanzado el número máximo de neuronas permitidas dentro de la red o hasta que cumpla algún criterio de finalización previamente establecido.

Holmström 2002 describe una variante de este algoritmo.

Pseudo código:

```
Inicializar la red con dos neuronas a y b cuyos vectores de
pesos estén inicializados al azar.
Mientras no se alcance el tamaño deseado o a una medida de
performance deseada hacer
    Elegir una entrada ξ.
    Encontrar la neurona más cercana s1 y la segunda más cercana
        s2.
    Incrementar la edad de todas las conexiones que salen de s1.
    Incrementar una variable de error de s1:
        Δe(s1) = ‖ ws1 - ξ ‖²
    Actualizar los pesos de s1 en una proporción εb y los pesos
        de las vecinas directas de s1 en una proporción εn.
        Δws1 = εb(ξ - ws1)
        Δwn = εn(ξ - wn),
           donde n son todos los vecinos directos de s1.
    Si s1 y s2 están conectadas
        entonces setear la edad de la conexión en cero.
        sino crear una conexión entre s1 y s2.
    Eliminar las conexiones cuya edad supere un umbral.
    Si en el paso anterior quedaron neuronas sin ninguna
        conexión a otra
        entonces eliminar esas neuronas.
    Si el número de entradas presentadas hasta este momento a la
        red es múltiplo de un parámetro λ
        entonces insertar nuevas neuronas, para esto:
              Determinar la neurona q con el máximo error
                  acumulado.
              Insertar una nueva neurona r entre q y su vecino
                  directo f, tal que f tenga el error acumulado
                  mas alto.
              Setear el vector de pesos de r:
                  wr = 0.5(wq + wf)
```

```
            Insertar conexiones entre r y q, y entre r y f.
            Eliminar la conexión entre q y f.
            Decrementar los errores de q y de f en una
                proporción α.
    Decrementar para todas las neuronas su error en una
        proporción d.
Fin mientras
```

4.2.3. Grow When Required (GWR)

El algoritmo GWR (Marsland *et al.* 2002) utiliza el CHL (Competitive Hebbian Learning), propuesto por Martinetz (Martinetz & Schulten 1991), creando conexiones entre las neuronas ganadoras y las neuronas "segunda" ganadoras. Las conexiones tienen una "edad" que se incrementa a medida que transcurre el entrenamiento.

La principal característica de este método es que las neuronas son agregadas en cualquier momento y no luego de *n* pasos de entrenamiento. Por ejemplo, puede agregarse una neurona al momento de presentar una entrada, agregar otra neurona con la entrada siguiente, y que en las próximas 100 pasadas no sea necesario agregar nuevas neuronas.

El método no tiene en cuenta variables de error para cada neurona y la inserción de la nueva neurona depende de la entrada y de la neurona ganadora. La inserción ocurre cuando la actividad de la neurona ganadora no es muy alta. En Marsland *et al.* 2002 los autores proponen varias maneras de medir la actividad de la neurona ganadora. Cuando la neurona que resulta ganadora tiene además una actividad superior a un umbral, se lleva a cabo una actualización de su vector de pesos, y los de sus vecinas.

Cada neurona posee un contador de éxito a fin de evitar que las recién creadas actualicen sus pesos en lugar de crear nuevas. Dicho contador recibe un valor alto cuando la neurona se crea y, a medida que resulte ganadora, o sea vecina de una neurona ganadora, va disminuyendo. Cuando una neurona ganadora tiene un "alto" contador de éxito se lleva a cabo una actualización del vector de pesos, aún cuando su actividad haya sido baja. Si la neurona ganadora tiene un "bajo" contador de éxito y una baja actividad entonces se genera una nueva neurona

A la edad de la conexión entre la neurona ganadora y la segunda ganadora se le asigna el valor cero. La edad se va incrementando durante el entrenamiento y cuando supera un umbral esa conexión se elimina.

Los autores ponen especial énfasis en que este algoritmo puede ser usado tanto en conjuntos de datos estáticos como en dinámicos, es decir, que los datos de entrada cambian con el tiempo.

Una variante a este método y que también está basado en el método CHL de Martinetz se describe en Aupetit 2003.

Pseudo código:

```
Inicializar la red con dos neuronas a y b cuyos vectores de
pesos estén inicializados al azar.
Mientras no se cumpla un criterio de finalización hacer
    Elegir una entrada ξ.
    Encontrar la neurona más cercana s y la segunda más cercana
        t.
    Si no existe conexión entre s y t
        entonces crear la conexión entre s y t
        sino setear la edad de la conexión entre s y t en 0.
    Calcular la actividad de la neurona s
        a=exp(-||ξ - w_s ||)
    Si a es menor que el umbral de actividad y el contador de
        éxito es menor que el umbral de éxito
        entonces      agregar una nueva neurona r
                setear el vector de pesos de r
                        w_r = (w_s + ξ) / 2
                crear conexiones entre r y s, y r y t
                eliminar la conexión entre s y t
    Si no se agregó una nueva neurona
        entonces actualizar los pesos de la neurona ganadora y
        sus vecinos
                Δw_s = ε_b h_s (ξ - w_s1)
                Δw_i = ε_i h_i (ξ - w_n), donde i son todos los vecinos
                                directos de s. h_s es el  contador de
                                éxito de la neurona s
    Incrementar en 1 las edades de todas las conexiones de s.
    Actualizar los contadores de éxito de s
        h_s(t) = h_0 – (S(t) / α_b)(1 – exp(-α_b t / τ_b))
    Actualizar los contadores de éxito de las vecinas de s
        h_i(t) = h_0 – (S(t) / α_n)(1 – exp(-α_n t / τ_n)), h_0 es la fuerza inicial, S(t)
        es la fuerza estímulo y α_b , α_n , τ_b y τ_n son constantes
        que controlan la forma de la curva.
    Eliminar las neuronas que no estén conectadas con ninguna
        otra neurona
    Eliminar las conexiones cuya edad supere un umbral.
Fin mientras
```

4.2.4. Growing Self-Organizing Map (GSOM)

En 1997, Hans-Ulrich Bauer y Thomas Villmann proponen un nuevo método que organiza los patrones de entrada en un espacio de salida formado por hipercubos. Este método no solo adapta los vectores de pesos de las neuronas sino que también hace crecer el espacio de salida (Bauer & Villmann 1997).

El método es inicializado con un hipercubo simple de una dimensión, y con dos neuronas las cuales son vértices del hipercubo inicial (en el espacio 1-dimensional las dos neuronas son los extremos de un segmento). A medida que el entrenamiento transcurre pueden ocurrir dos casos, o bien un nuevo nodo es agregado al espacio actual, o bien se hace crecer al espacio en alguna dimensión. Esta decisión se lleva a cabo analizando las regiones de Voronoi formadas por las neuronas.

El entrenamiento de este método es un SOM clásico. Se presenta un patrón de entrada a la red, se elige la neurona ganadora, y esta y sus vecinas forman parte del aprendizaje. El algoritmo consiste de continuos ciclos de aprendizaje y expansión. Se presentan todos los patrones de entrada a la red y luego se expande la red, ya sea agregando nuevos nodos o agregando dimensiones al espacio de salida, luego se vuelve a presentar todos los patrones de entrada a la red y así se continua hasta obtener un resultado satisfactorio.

Al momento de armar la región de Voronoi para ver en que dirección hace crecer la red, se calcula un error entre los patrones de entrada *v* pertenecientes a la región de la neurona *r* y el vector de pesos w_r de *r*. Se calcula el error para cada dirección del hipercubo, y la dirección con mayor error promedio es la utilizada para agregar nuevas neuronas.

En Villmann & Bauer 1998 se pueden encontrar casos reales sobre los cuales se probó este algoritmo.

Pseudo código:

```
Inicializar la red con dos neuronas a y b formando parte de un
hipercubo 1-dimensional.
Mientras no se cumpla un criterio de finalización hacer
    Llevar a cabo el entrenamiento según un SOM clásico
        presentando todos los patrones de entrada
    Para cada neurona r de la red.
        Calcular el error de la región de Voronoi representada
            por r
                \sum_{i=1}^{d} a_i(v) \frac{w_{r+e_i} - w_{r-e_i}}{\left\| w_{r+e_i} - w_{r-e_i} \right\|} + v',
                donde d es la dimensionalidad actual de la red, y
                e_i es la unidad del vector en la posición i.
        Determinar en que dimensión hacer crecer el espacio de
            salida, o en que dirección agregar nuevas neuronas.
            Inicializar los pesos de las nuevas neuronas.
    Fin para
Fin mientras
```

4.2.5. Growing Self-Organizing Map (GSOM)

Este método organiza las neuronas en una grilla bidimensional permitiendo que cada elemento de la red pueda tener a lo sumo cuatro vecina directas (Alahakoon *et al.* 2000).

La estructura inicial está formada por neuronas conectadas de manera que cada una de ellas sólo posea dos vecinas. Antes de comenzar con el entrenamiento se determina el umbral, GT, que debe alcanzar el error acumulado de una neurona para considerar la necesidad de agregarle un vecino inmediato.

El proceso de entrenamiento consta de dos partes: la primera genera todas las neuronas de la estructura y la segunda acomoda los vectores de pesos correspondientes a cada unidad de la red.

La etapa de generación respeta, en líneas generales, el proceso descrito inicialmente ya que en cada paso de adaptación se recalculan los vectores de pesos correspondientes y se incrementa el error de la neurona ganadora. Cuando dicho error supera el umbral indicado por GT, se agrega una nueva vecina de la neurona ganadora. Esto ocurre siempre que tenga menos de cuatro vecinos; sino es así, sólo se distribuyen los pesos. Esta etapa finaliza cuando se logra un crecimiento mínimo o cuando ya no se producen nuevas neuronas.

Finalmente, una vez obtenida la arquitectura de la red, se realiza una etapa de suavizado reduciendo la velocidad de aprendizaje y el tamaño de la vecindad.

Un concepto importante que Alahakoon presenta en su método es el uso de un parámetro llamado factor de dispersión (spread factor) que permite indicarle a la red en que medida debe dispersarse o crecer. De esa manera, la red puede crecer poco para tener una visión amplia de cuales son los principales clusters que se forman con los datos de entrada, o dispersarse mucho más obteniendo así un mayor detalle de cada agrupamiento.

La organización bidimensional de las neuronas permite una fácil visualización de los agrupamientos obtenidos luego del entrenamiento independientemente de la dimensionalidad de los patrones de entrada.

Los autores, en pruebas posteriores con conjuntos de patrones de muy alta dimensión, descubrieron algunas desventajas de este algoritmo, como la "fusión" de dos clusters distintos. Esto sucede cuando dos procesos de dispersión crecen a tal punto de juntarse. O también la gran cantidad de neuronas que, al finalizar el entrenamiento, no representan a ningún vector de entrada. Como solución a estos inconvenientes presentan una variante que denominaron HDGSOM (High dimensional growing self-organizing map)(Amarasiri *et al*. 2004).

Pseudo código:

```
Inicializar la red con cuatro neuronas, cada una con dos vecinas.
Calcular el umbral de crecimiento GT de acuerdo al factor de dispersión establecido.
Mientras no se cumpla un criterio de finalización de la etapa de crecimiento hacer
    Presentar un patrón de entrada a la red.
    Elegir la neurona ganadora j.
    Actualizar los vectores de pesos de la neurona ganadora j
        w_j(k+1) = w_j(k) + LR(k) (x_k – w_j(k)), donde LR(k) es el factor de
                   aprendizaje.
    Actualizar los vectores de pesos de las neuronas vecinas de
       j
        w_j(k+1) = w_j(k) + v(k) LR(k) (x_k – w_j(k)), donde LR(k) es el factor de
                                        aprendizaje, y v(k) es una
                                        función de vecindad
    Incrementar el error E_j de la neurona ganadora j
    Si E_j > GT
```

```
        entonces si j es una neurona límite (no tiene cuatro
                 vecinos directos)
                    entonces crear nuevas neuronas vecinas
                             directas de j
                             inicializar los vectores de pesos
                             de las nuevas neuronas

                    sino actualizar los pesos de las cuatro
                             neuronas vecinas
                Fin si
    Fin si
Fin mientras
Mientras no se cumpla un criterio de finalización de la etapa de
ajuste hacer
    Presentar un patrón de entrada a la red
    Elegir la neurona ganadora j
    Actualizar los vectores de pesos de la neurona ganadora j
    Actualizar los vectores de pesos de las neuronas vecinas
    directas de j
Fin mientras
```

5. AVGSOM

Como ya se explicó anteriormente, la preservación de la topología de los datos de entrada es una característica fundamental de los mapas auto-organizativos.

Cuando se utiliza una arquitectura estática, definida a priori, la preservación de la topología resulta un hecho casi natural debido a la forma en que se adaptan los pesos. Por el contrario, cuando la arquitectura crece dinámicamente, lograr que los conjuntos de datos de entradas similares se encuentren representados por neuronas vecinas dentro de la arquitectura de la red, no es una tarea sencilla.

El mayor problema se presenta en la creación de una nueva neurona. Dependiendo del criterio utilizado para determinar su vector prototipo de la nueva neurona, puede ocurrir que a dos neuronas alejadas en el espacio de salida, le sean asignados vectores similares. Nótese que esta situación no podrá ser resuelta durante el entrenamiento ya que las vecindades involucradas son totalmente disjuntas.

En esta dirección, el método presentado en este trabajo de grado, denominado AVGSOM, propone una nueva estrategia para resolver este problema.

AVGSOM utiliza una grilla rectangular donde cada neurona posee un máximo de cuatro vecinos directos. El entrenamiento comienza con una estructura mínima de cuatro neuronas, donde cada una solo posee dos vecinas directas, formando una matriz de 2x2 y los vectores prototipos correspondientes se inicializan en forma aleatoria.

Una de las principales ventajas del AVGSOM es la de presentar el resultado final en una estructura de dos dimensiones facilitando la visualización de las neuronas y sus conexiones. Además sabiendo para cada neurona que patrones de entrada está representando, se pueden visualizar los clusters que quedan formados en la red.

En cada paso de iteración se adaptan los vectores prototipos y se acumula el error en la neurona ganadora de la manera habitual. El hecho de que una neurona, durante el entrenamiento, supere el umbral de error acumulado establecido a priori, indica la necesidad de adaptar la estructura para evitar la excesiva acumulación de patrones alrededor de la ganadora. Cualquier modificación sobre la estructura debe respetar la topología rectangular propuesta inicialmente. Por tal motivo, si la ganadora ya posee sus cuatro vecinos directos, deberá reducirse la diferencia entre la ganadora y sus vecinas directas permitiendo de esta forma que en sucesivos pasos de adaptación sean otros los elementos de la red que ganen la competencia.

En caso que la ganadora posea menos de cuatro vecinas directas, se insertará en la estructura un nuevo elemento. Es muy importante que la nueva neurona sea inicializada con un vector de pesos tal que, no sea muy distinto del resto de sus vecinas, para que al momento de actualizar los pesos de la nueva neurona como la de sus vecinas, no distorsione la topología de la red durante el aprendizaje.

El método que se utiliza para determinar el vector de pesos de una nueva neurona en AVGSOM es el promedio de los vectores prototipos de las que serán vecinas de la nueva neurona. De esta manera, no sólo se relacionan los elementos de la red dentro del

grafo sino que el vector de pesos de la nueva neurona queda "en el medio" de los vectores de pesos de sus vecinas en el espacio de entrada preservando la topología de los datos casi en su totalidad.

Pseudo código:

```
Inicializar la red con cuatro neuronas, cada una con dos vecinas.
Inicializar el peso de las cuatro neuronas aleatoriamente.
Calcular el umbral de crecimiento GT de acuerdo al factor de dispersión establecido.
Mientras no se creen nuevas neuronas o la tasa de crecimiento sea mínima

    Para cada patrón de entrada
        Ingresar el patrón a la red neuronal
        Identificar como neurona ganadora a aquella que posea el vector prototipo más parecido al patrón recientemente ingresado a la red según alguna medida de similitud definida a priori. Si bien en este artículo se ha utilizado Distancia Euclídea, la selección de esta medida es dependiente del problema.
        Efectuar la adaptación en el vector prototipo correspondiente a la neurona ganadora y en su vecindad como habitualmente lo realiza SOM.
        Acumular en la neurona ganadora la magnitud del error correspondiente. Dicho valor se corresponde con la medida de similitud evaluada en el punto anterior.
    Fin Para
    Para cada neurona de la red
        Si el error de la neurona ha superado el umbral GT
            entonces Si la neurona tiene cuatro vecinas directas
                        entonces Repartir el error acumulado por la neurona ganadora entre sus vecinas directas. Con esto se busca que en las próximas pasadas, las vecinas tengan una mayor oportunidad de saturarse, y de esta manera "empujar" el error hacia los límites de la red logrando la expansión de la misma.
                        sino Seleccionar al azar uno de los lugares libres para generar la nueva neurona vecina.
                             El vector prototipo correspondiente a esta nueva neurona se calcula como el promedio de los vectores prototipos de las que serán sus vecinas.
```

```
                                Asignar cero como error acumulado
                                   para esta nueva neurona.
                                Asignar al error acumulado de esta
                                   neurona el valor cero.
                        Fin si
                Fin si
                Si esta neurona nunca resultó ganadora
                   entonces Incrementar en 1 su contador de fallas.
                Fin si
                Si el contador de fallas de esta neurona alcanzó un
                   umbral preestablecido
                   entonces Eliminar la neurona.
                Fin si
            Fin Para
        Fin mientras
```

El umbral *GT* se calcula de la misma forma que en GSOM (Alahakoon *et al.* 2000), esto es $GT = -D * ln(SF)$ siendo *SF* un valor entre 0 y 1 correspondiente al factor de dispersión, indicado como parámetro.

5.1. CAPA DE SALIDA

Como se mencionó anteriormente, una de las principales ventajas del AVGSOM es la facilidad que brinda su arquitectura bidimensional para visualizar las similitudes entre los datos de entrada.

Al tener todas las neuronas en un mismo plano y dado que cada una de ellas representa a un grupo de patrones de entrada, es posible determinar los clusters; es decir, agrupar los subconjuntos de neuronas que mapean patrones de similares características.

La tarea de descubrir los clusters es un proceso tedioso para resolver manualmente, en especial cuando la red final posee muchas neuronas. La clasificación manual consistiría en tomar una neurona, analizar los patrones que representa y repetir este proceso con sus vecinas a fin de determinar si las características de los patrones representados por cada una de ellas son similares o no. De esta manera y luego de un largo proceso podría identificarse todas las neuronas que forman parte de un mismo cluster. Esto debería repetirse con las neuronas restantes hasta lograr agruparlas a todas (Figura 5.1).

A fin de evitar la realización de un agrupamiento manual, se propone a continuación un método automático para realizar esta tarea. Este método puede verse como una capa de salida del método AVGSOM propuesto. El algoritmo de la capa de salida consiste en clasificar los pesos de las neuronas resultantes del primer entrenamiento, que llamaremos capa competitiva.

La capa de salida ejecuta el mismo algoritmo utilizado en la capa competitiva, con la diferencia de que la red de esta segunda etapa, comienza con una única neurona. Obviamente, como la capa de salida recibe como entrada los pesos de las neuronas de

la capa competitiva, esta última debe ser entrenada primero. Es decir que ambas capas se entrenan por separado.

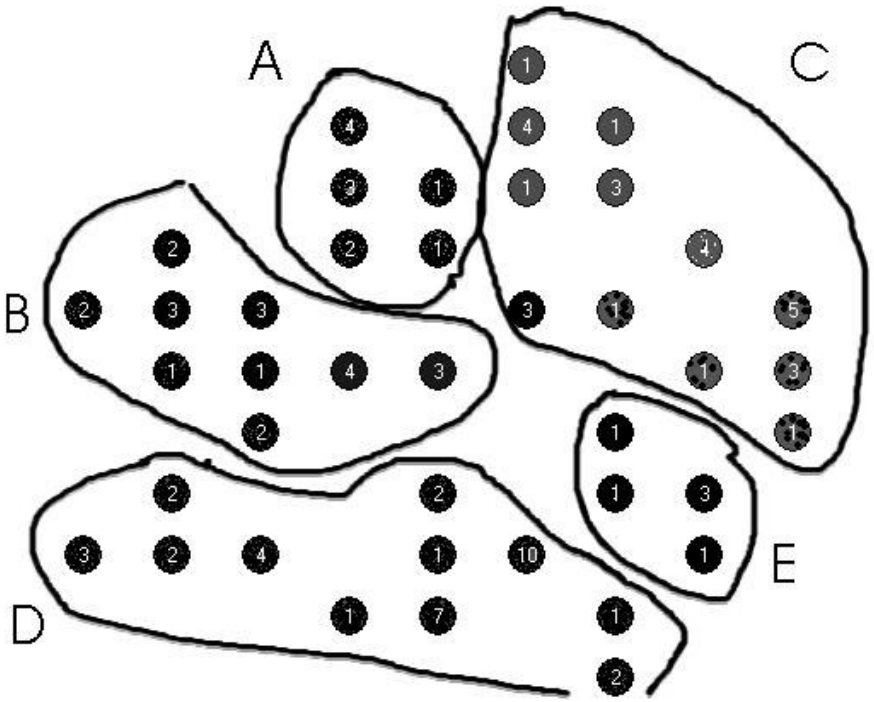


Figura 5.1. Ilustra el resultado de entrenar el AVGSOM con el juego de datos del zoológico. Los círculos con un número en su interior representan a las neuronas de la red, el número indica la cantidad de patrones de entrada a los que representa. Con un trazo a mano están marcados cinco subconjuntos de neuronas. Las neuronas de A representa a los insectos, las de B a las aves, las de C a los animales acuáticos, las de D a los mamíferos terrestres, y las de E a los mamíferos marinos. Se puede observar como los mamíferos marinos quedaron entre el conjunto de neuronas que representan a los mamíferos terrestres y el conjunto de neuronas que representa a los animales acuáticos. Esto se debe a que los mamíferos marinos tienen características en común a ambos subconjuntos. Los cinco subconjuntos se identificaron inspeccionando manualmente neurona por neurona, para observar que animal representaba.

La presentación de los vectores asociados a cada neurona competitiva, a la capa de salida, no se realiza en un orden aleatorio. Para incrementar su representatividad, se los ordena de mayor a menor según la cantidad de patrones a los cuales representa la neurona de la capa competitiva correspondiente. De esta manera, en primer lugar ingresan los pesos de las neuronas que mas patrones de entrada representen.

El agrupamiento de los vectores de pesos de las neuronas de la capa competitiva en función de una medida de similitud predefinida también se aplica a los patrones de entrada por ellos representados. El resultado final sería una gran agrupación de todos los patrones de entrada con características similares. La figura 5.2 ilustra el resultado de entrenar la capa de salida para el resultado de la capa competitiva mostrado en la figura 5.1.

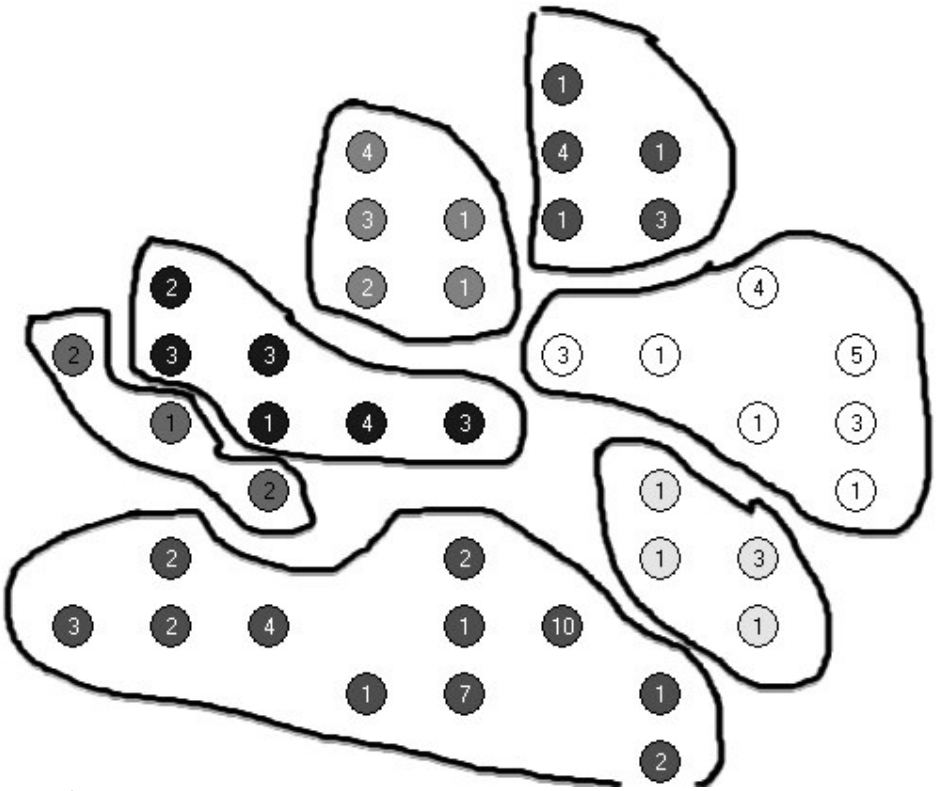


Figura 5.2. La gráfica muestra los grupos de neuronas competitivas determinados luego del entrenamiento de la capa de salida. Esta representación ha sido especialmente construida para esta tesis ya que la aplicación original utiliza colores para realizar esta tarea. Lamentablemente, los mismos no pueden ser apreciados en una figura en tonos de grises como esta.

6. RESULTADOS Y COMPARACIONES

Se utilizaron para medir el comportamiento de AVGSOM cinco juegos de bases de datos del repositorio de bases de datos UCI (Hettich *et al.* 1998). A continuación se describen brevemente y en el apéndice 2 se describen con mas detalles las características de cada uno.

I) *Iris Plants Database*

Es una base de datos que contiene información sobre distintos tipos de la planta iris. En ella aparecen tres clases distintas representadas por 50 patrones cada una. Para cada dato se presentan cuatro atributos conteniendo el ancho y el alto del pétalo y del sépalo medidos en centímetros. Tiene la particularidad de que sólo una de las clases es linealmente separable.

II) *Glass Identification Database*

Esta base de datos contiene información sobre siete tipos de vidrios distintos. Posee 214 patrones de entrada y cada patrón se encuentra representado por nueve atributos: el índice de refracción, y el porcentaje de presencia de ocho metales (sodio, magnesio, aluminio, silicio, potasio, calcio, bario y hierro).

III) *Zoo Database*

Esta base de datos contiene siete clases distintas de animales (mamíferos, aves, anfibios, peces, insectos, reptiles y otros invertebrados no insectos.). Contiene 101 patrones de entrada y cada patrón tiene 16 atributos de los cuales 15 son binarios (tiene pelo, pone huevos, da leche, es venenosos, vuela, tiene pluma, etc.), y uno no lo es (cantidad de patas).

IV) *Dermatology Database*

Esta base de datos contiene seis clases distintas de enfermedades en la piel. Contiene 366 patrones de entrada y cada patrón tiene 34 atributos, 33 correspondientes a características de la piel y el último a la edad del paciente.

V) *Faults in a urban waste water treatment plant Database*

Esta base de datos contiene 527 medidas diarias de sensores en una planta de tratamiento de agua pertenecientes a 13 clases. Cada clase representa el estado de la planta. La clase con más patrones es la que representa el funcionamiento correcto. El resto de las clases representan problemas que se encontraron en el producto final durante la toma de datos. Cada patrón tiene 38 atributos correspondientes a distintas mediciones en el agua registradas tanto en la entrada a la planta, como en la salida.

Como métrica para medir el grado en que la red preserva la topología de los datos de entrada se utilizó el método propuesto por Hsu & Halgamuge 2001. Esta métrica incrementa el error topográfico cuando las dos neuronas ganadoras para un patrón de entrada dado no son vecinas directas.

Se comparó AVGSOM con el método GSOM propuesto por Alahakoon *et al.* 2000, ya que en ambos métodos las neuronas están organizadas en una grilla de dos dimensiones. Los otros métodos existentes, estudiados en este trabajo de grado, utilizan

estructuras de más de dos dimensiones. Esto permite que una misma neurona posea un mayor número de vecinas directas que los métodos AVGSOM y GSOM, obteniendo de esta forma un menor error topológico. Sin embargo, un incremento en la dimensión de la estructura perjudica la visualización de las similitudes existentes entre los patrones de entrada.

Al momento de entrenar ambos métodos se tuvieron en cuenta las siguientes consideraciones:

- En todos los casos, los elementos de los patrones de entrada fueron normalizados entre 0 y 1 antes de comenzar con el entrenamiento.

- Se generó un orden aleatorio para ingresar los patrones de cada corrida.

- Para una misma corrida, los patrones fueron ingresados en el mismo orden en ambos métodos.

- Se llevaron a cabo varios entrenamientos utilizando los tres juegos de datos descriptos anteriormente junto con tres lotes de parámetros, los cuales están caracterizados por el factor de dispersión. Se realizaron corridas con poca dispersión (0,2), dispersión media (0,5), y dispersión alta (0,85) en el mapa.

La tabla 6.1 muestra los resultados obtenidos. Para cada uno de los tres juegos de datos, se muestra la cantidad de entrenamientos donde el método propuesto obtuvo un error topológico menor que GSOM junto con el error topológico promedio luego de las corridas. En la tabla se discriminan tres juegos distintos de parámetros, cada juego con un factor de dispersión diferente.

Por ejemplo, la tabla 1 muestra que para la base de datos Iris, luego de entrenar 40 veces cada uno de los métodos con un factor de dispersión de 0,2 se ha observado que en 25 oportunidades el error topológico de AVGSOM fue inferior al de GSOM. Esta situación también se observa en el error topológico promedio que para AVGSOM es 0,111 y para GSOM es 0,139.

Los datos de la Tabla 6.1 permiten observar que el algoritmo propuesto preserva satisfactoriamente la topología de los datos de entrada, siendo superado por el GSOM en la base de datos de plantas iris cuando se utiliza un factor de dispersión de 0,5 o 0,85, y por las bases de datos de tratamiento de agua y clasificación dermatológica con el factor de dispersión de 0,85.

Otra gran diferencia que existe entre ambos métodos es que el AVGSOM genera una grilla con muchas menos neuronas que las que genera el GSOM. La figura 6.1 muestra el resultado de ambos método para el mismo juego de parámetros y un mismo conjunto de patrones de entrada.

Una característica observada entre ambos métodos es que el AVGSOM necesita más pasadas de entrenamiento que el GSOM. En pocas ocasiones se observó que la cantidad de pasadas fue la misma. Esto generalmente ocurrió cuando el factor de dispersión era menor que 0,5. A su vez, el método AVGSOM, además de generar menos neuronas que

el GSOM, finaliza el entrenamiento con un error de las neuronas menor que el del GSOM (figura 6.2).

En el apéndice 1 se comenta el uso de la aplicación utilizada para medir estos resultados.

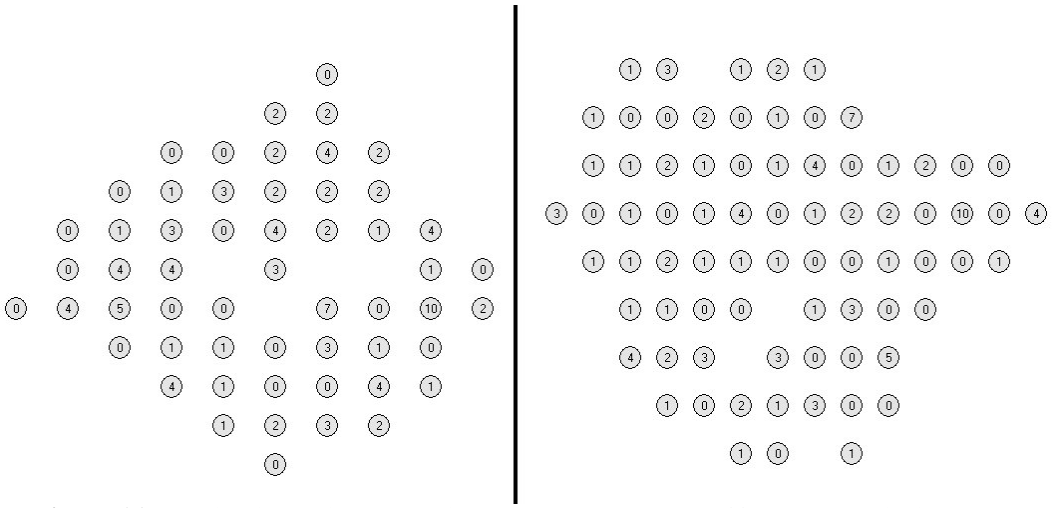
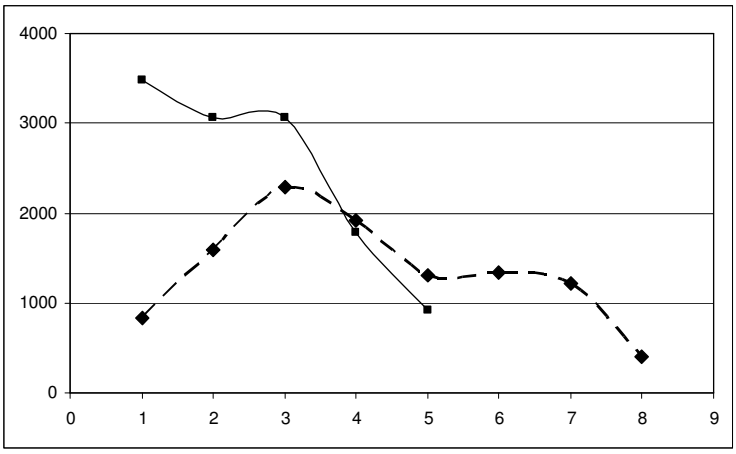


Figura 6.1. A la izquierda el resultado del AVGSOM, que generó 55 neuronas. A la derecha el resultado del GSOM, que generó 76 neuronas. Ambos métodos fueron entrenados con el mismo conjunto de datos y con los mismos valores de los parámetros.

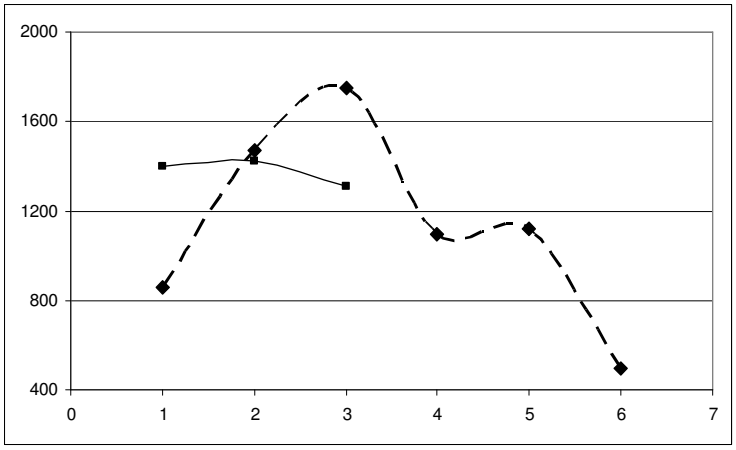
Tabla 6.1. En la primer columna figuran los cinco juegos de datos utilizados y entre paréntesis la cantidad de veces (*n*) que se entrenaron los métodos GSOM y AVGSOM con ese juego de datos. Para cada método se muestra la cantidad de corridas donde el error topológico fue menor que el otro método y el error topológico promedio para las *n* corridas.

	AVGSOM		GSOM	
	Cantidad de corridas con menor error	error topológico promedio	Cantidad de corridas con menor error	error topológico promedio
SF= 0.2				
Iris Plants (*n*=40)	25	0,111	15	0,139
Glass Identification (*n*=60)	36	0,140	24	0,163
Zoo (*n*=20)	14	0,071	6	0,107
Dermatology (*n*=90)	56	0,114	34	0,133
Water treatment (*n*=90)	61	0,203	29	0,264
SF= 0.5				
Iris Plants (*n*=40)	22	0,152	18	0,139
Glass Identification (*n*=40)	25	0,090	15	0,118
Zoo (*n*=70)	45	0,104	25	0,138
Dermatology (*n*=90)	60	0,173	30	0,198
Water treatment (*n*=100)	58	0,282	42	0,309
SF= 0.85				
Iris Plants (*n*=20)	5	0,151	15	0,100
Glass Identification (*n*=30)	16	0,108	14	0,108
Zoo (*n*=50)	32	0,153	18	0,165
Dermatology (*n*=46)	13	0,267	33	0,189
Water treatment (*n*=30)	10	0,301	20	0,264

a)



b)



c)

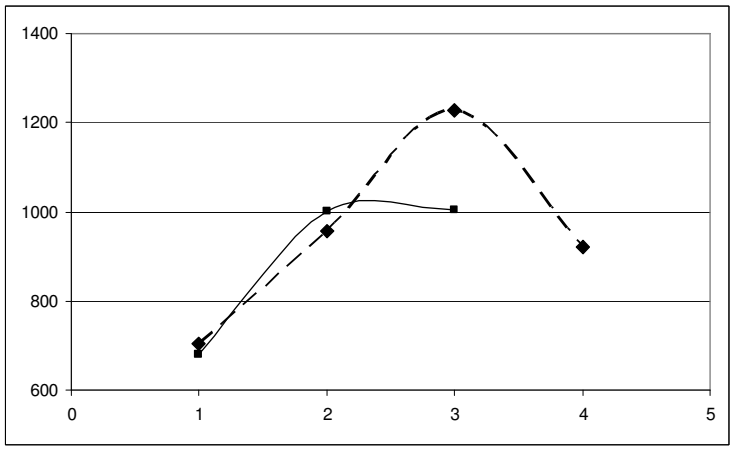


Figura 6.2. La línea rayada corresponde al error del AVGSOM, la línea continua al error del GSOM. Se puede observar que si bien el AVGSOM necesita más pasadas para finalizar el entrenamiento, cuando lo hace finaliza con un error menor. Con la base de datos del zoológico se muestran tres entrenamientos, en a) se utilizó un factor de dispersión igual a 0,9, en b) 0,5 y en c) 0,2.

7. CONCLUSIONES

A lo largo de esta tesis se han explicado y ejemplificado las arquitecturas de redes neuronales convencionales aplicables a tareas de clasificación, comenzando por las más simples hasta llegar a los mapas auto-organizativos dinámicos, los cuales constituyen el objetivo central. Se desea resaltar la importancia de contar con material de estas características dada la dificultad de interpretación que presenta la bibliografía existente. En esta dirección se ha intentado ilustrar y/o ejemplificar las arquitecturas descriptas a fin de facilitar su estudio.

Para los mapas auto-organizativos dinámicos se ha presentado una nueva estrategia para insertar neuronas en este tipo de arquitectura con resultados muy satisfactorios en lo que se refiere a la preservación de la topología de los datos de entrada. Su aplicación en la resolución de cinco casos concretos ha demostrado su superioridad con respecto a GSOM.

Sin embargo aún quedan varios aspectos por mejorar, algunos de los cuales se detallan a continuación:

a) Como ocurre habitualmente con las soluciones basadas en redes neuronales, la respuesta obtenida, si bien es satisfactoria, varía en función del orden de presentación de los patrones de entrada. Por este motivo sería interesante analizar la factibilidad de realizar un preprocesamiento de los datos de entrada y medir el impacto que esto tendría sobre el funcionamiento de la red.

b) Uno de los problemas observados durante el crecimiento de la arquitectura y que en ocasiones perjudica la preservación de la topología, es la inserción de nuevas neuronas únicamente en la periferia de la red. Esto se encuentra restringido por el conexionado rectangular de sus individuos. La inserción de un nuevo elemento en el interior de la red implica la generación de sus vecinos correspondientes. Esta acción no puede ser realizada dado que generaría una modificación de los vecinos inmediatos de otras neuronas que no se encuentran directamente involucradas con la nueva inserción. Esto lleva a que la aparición tardía de vectores pertenecientes a vecindades de regiones que ya poseen sus cuatro vecinos deban ser ubicados en posiciones desfavorables dentro de la red.

c) Como puede observase, la solución de este problema no es simple y se encuentra en relación con el tema del párrafo anterior, referido al orden de presentación de los patrones de entrada. Existe una relación de compromiso entre el esfuerzo realizado en el preprocesamiento de los patrones de entrada y la preservación de la topología de los datos de entrada representada por la red.

d) Finalmente cabe mencionar que el conexionado rectangular propuesto en esta arquitectura no es el único posible. Simplemente fue adoptado por su simplicidad de representación pero fácilmente podría ser reemplazado por otro donde cada neurona pudiese tener un número superior de vecinos. En tal caso sería deseable analizar el impacto que este nuevo conexionado tendría sobre el criterio actual de inserción de AVGSOM.

8. BIBLIOGRAFÍA

Alahakoon, D., S. K. Halgamuge & B. Srinivasan. 2000. *Dynamic Self-Organizing Maps with Controlled Growth for Knowledge Discovery*. IEEE Transactions on Neural Networks **11** (3), pp. 601-614.

Amarasiri, R., D. Alahakoon & K. A. Smith. 2004**.** *HDGSOM: A Modified Growing Self-Organizing Map for High Dimensional Data Clustering*. Proceedings of the Fourth International Conference on Hybrid Intelligent Systems.

Anderson, J. A, J. W. Silverstein, S. A. Ritz & R. S. Jomnes. 1977. *Distinctive features, categorical perception and probability learning: some aplications of a neural model*. Psychological Review **84**.

Anderson, O., P. Cosi & P. Dalsgaard. 1990. In 1st Workshop on Neural Networks and Speech Processing. Paoloni A., editor. Roma, Italy.

Azuaje, F., W. Dubitzky, N. Black & K. Adamson. 2000. *Discovering relevance knowledge in data: a growing cell structures approach*. IEEE Transactions on Systems, Man and Cybernetics, Part B, **30** (3), pp: 448-460.

Aupetit, M. 2003. *Robust topology representing networks.* ESANN´2003 proceedings. European Symposium on Artificial Neural Networks Bruges (Belgium), ISBN 2-930307-03-X, pp. 45-50.

Barreto, G.A. & A. F. R. Araújo. 2004. *Identification and Control of Dynamical Systems Using the Self-Organizing Map.*IEEE Transactions on Neural Networks **15** (5), pp. 1244-1259.

Bauer, H.-U. & K. Pawelzik. 1992. *Quantifying the neighborhood preservation of self-organizing feature maps*. IEEE Transactions on Neural Networks **3**(4):570-579.

Bauer, H.-U. & Th. Villmann. 1997. *Growing a hypercubical output space in a self-organzing map.* IEEE Transactions on Neural Networks **8** (2), pp. 218-226.

Blackmore, J. & R. Miikkulainen. 1993. *Incremental grid growing: encoding heigh dimensional structure into a two-dimensional feature map*. Proceedings of the IEEE International Conference on Neural Networks, Vol. **1**, pp. 450-455, San Francisco, California.

Bruske, J. & G. Sommer. 1995. *Dynamic cell structure learns perfectly topoly preserving map*. Neural Computation **7**(4): 845-865.

Coiton, Y., J. C. Gilhodes, J. L. Velay & J. P. Roll. 1991. Biol. Cyb. **66**, pp. 167.

Danielson, S. 1990. *Recognition of Danish phonemes using an artificial neural network*. In IEEE International Joint Conference on Neural Networks.

Ferrán, E. A. & P. Ferrara. 1991. Biol. Cyb. **65**, pp. 451.

Freeman, J. A. & D. M. Skapura. 1991. *Neural networks. Algorithms, applications and programming techniques*. Addison-Wesley Publishing Company, Inc. Reading, Massachusetts, U.S.A.

Fritzke, B. 1993. *Growing cell structures: A self organizing network for supervised and un-supervised learning*. Neural networks **7**, pp. 1441-1460.

Fritzke, B. 1995. *A growing neural gas network learns topologies.* G. Tesauro, D.S. Touretzky and T.K. Leen (eds.), Advances in neural information processing system **7**, pp. 625–632, MIT Press, Cambridge MA.

Fritzke, B. 1996. *Growing self-organizing networks – Why?*. En M. Verleysen (edit.), ESANN '96: European Symposium on Artificial Networks, D-Facto Publishers, Brussels. pp. 61-72.

Fritzke, B. 1997. *Unsupervised ontogenic network*. Handbook of neural computation. IOP Publishing Ltd and Oxford University Press. Capítulo 2.4, 16 págs.

Ghosal, S. & R. Mehrotra. 1993. *A two-stage neural net for segmentation of range images*. In IEEE International Conference on Neural Networks. San Francisco, California.

Gjerdingen, R. O. 1989. Computer Muscic Journal **13**, pp. 67.

Graf, D. & W. LaLonde. 1989. *Neuroplanners for hand/eye coordination.* In IEEE International Joint Conference on Neural Networks.

Grossberg, S. 1987. *Competitive learning: from interactive activation to adaptative resonance*. Cognitive Science **11**(1), pp. 23-63.

Hannah, P., R. Stonier & S. Smith. 1994. In Proc. 5th Australian Conference on Neural Networks. Tsoi, A. C. & T. Downs (edit.). University of Queensland, St. Lucia, Australia.

Hassoun, M.H. 1995. *Fundamentals of artificial neural networks*. Massachusetts Institute of Technology. ISBN 0-262-08239-X.

Hebb, D. 1949. *The organization of behavior*. Wiley, Ney York.

Hetch-Nielsen, R. 1987a. *Counterpropagation networks*. Applied Optics **26**(23):4979-4984.

Hetch-Nielsen, R. 1987b. *Counterpropagation networks*. Proceedings of the IEEE, First International Conference on Neural Networks. Caudill, M. & C. Butler (editores). Piscataway, NJ.

Heinke, D. & F. H. Hamker. 1998. *Comparing neural networks: a benchmark on growing neural gas, growing cell structures, and fuzzy ARTMAP*. IEEE Transactions on Neural Networks **9**(6), pp. 1279-1291.

Hernáez, I., J. Barandiarán, E. Monte & B. Extebarria. 1993. 3rd European Conference on Speech, Communication and Technology.

Hettich, S., C.L. Blake & C.J. Merz. 1998. *UCI Repository of machine learning databases*. University of California, Irvine, Dept. of Information and Computer Sciences. URL http://www.ics.uci.edu/~mlearn/MLRepository.html.

Hinton, G.E. 1992. *Redes neuronales que aprenden de la experiencia*. Investigación y ciencia, noviembre, 1992.

Hodge, V.J. & J. Austin. 2001. *Hierarchical growing cell structures: TreeGCS*. IEEE Transactions on Knowledge and Data Engineering **13** (2), pp. 207-218.

Hoplfield, J. J. 1982. *Neural networks and physical systems with emergent collective computational abilities*. Proceedings of the National Academy of Science USA, Vol. **79**, pp. 2554-2558.

Hsu, A.L. & S. K. Halgamuge. 2001. *Enhanced topology preservation of dynamic self-organizing maps for data visualization.* In proceedings of the Joint 9th IFSA World Congress and 20th NAFIPS International Conference 2001, Aug 2001, Vancouver, Canada.

Holmström, J. 2002. *Growing neural Gas: Experiments with GNG, GNG with utility and superviced GNG.* Uppsala Master's Thesis in Computer Science. Uppsala University. Departament of Information Technology. Computer Systems, Uppsala, Sweden.

Huang, Z. & A. Kuh. 1992. *A combined self-organizing feature map and multilayer perceptron for isolated word recognition*. IEEE Transactions on Signal Processing **40**(11), pp. 2651-2657.

Idan, Y. & R. C. Chevallier. 1991. *Handwritten digits recognition by a supervised Kohonen-like learning algorithm*. In IEEE International Joint Conference on Neural Networks. Singapore.

Isasi Viñuela, P. & I. M. Galván León. 2004. *Redes de neuronas artificiales*. 1ra edición. Pearson Educación, S. A. ISBN 84-205-4025-0.

Kita, H. & Y. Nishikawa. 1993. In Proc. World Congress on Neural Networks.

Koh, J., M. Suk & S. M. Bhandarkar. 1993. *A multi-layer Kohonen's self-organizing feature map for range image segmentation*. In IEEE International Conference on Neural Networks. San Francisco, California.

Kohonen, T. 1984. *Self-organization and associative memory*. Series in Information Sciences. Vol. **8**. Berlin: Springer-Verlag.

Kohonen, T., K. Mäkisara & T. Saramäki. 1984. *Phonotopic maps – insightful representation of phonological features for speech recognition*. Proceedings 7th International Conference on Pattern Recognition, Montreal. pp. 182-185.

Kohonen, T. 1990. *The self-organizing map*. Proceedings of the IEEE **78**(9), pp. 1464-1480.

Kohonen, T. 1994. In Proc. World Congress on Neural Networks.

Kohonen, T. 1997. *Self-Organizing Maps*. 2nd Edition. Springer. ISSN 0720-678X.

Kohonen, T., S. Kaski, K. Lagus, J. Salojarvi, J. Honkela, V. Paatero & A. Saarela. 2000. *Self organization of a massive document collection*. IEEE Transactions on Neural Networks **11**(3), pp. 574-585.

Marsland, S., J. Shapiro & U. Nehmzow. 2002. *A self-organising network that grows when required.* Neural networks **15**, pp. 1041-1058.

Martinetz, T.M., H. Ritter & K. Schulten. 1990. *Three-dimensional neural net for learning visuomotor-coordination of a robot arm*. IEEE Transactions on Neural Networks **1**(1): 131-136.

Martinetz, T.M. & K.J. Schulten. 1991. *A "neural-gas" network learns topologies*. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, Artificial Neural Networks, pages 397-402. North-Holland, Amsterdam.

Martinetz, T.M. 1993. *Competitive Hebbian learning rule forms perfectly topology preserving maps*. In ICANN'93: International Conference on Artificial Neural Networks, pages 427-434, Amsterdam. Springer.

Martinetz, T.M., S.G. Berkovich & K.J. Schulten. 1993. *"Neural-Gas" Networks for Vector Quantization and its application to Time-Series Prediction*. IEEE Transactions on Neural Networks **4**(4), pp. 558-569.

Martinetz, T.M. & K.J. Schulten. 1994. *Topology representing networks*. Neural Networks **7**(3):505-522.

Merkl, D. 1993. *Structuring software for reuse-the case of self-organizing maps.* In IEEE International Joint Conference on Neural Networks.

Minsky, M.L. & S. Papert. 1969. *Perceptrons*. Cambridge, MA: MIT Press.

Moody, J. & C. Darken. 1988. *Learning with localized receptive fields*. D. Touretzky, G. Ginton & T. Sejnowsky, eds. In Proceedings of the 1988 Connectionist Models Summer School, Morgan Kaufmann, San Mateo, pp. 133-143.

Morris, R. J. T., L. D. Rubin & H. Tirri. 1989. *A comparison of feedforward and self-organizing approaches to the font orientation problems*. In International Joint Conference on Neural Networks.

Nasrabadi, N.M. & Y. Feng. 1998. *Vector quantization of images based upon de Kohonen self-organizing feature maps*. In IEEE International Conference on Neural Networks. San Diego. pp. 1101-1108.

Naylor, J. & K. P. Li. 1988. *Analisys of a neural network algorithm for vector quantization of speech parameters*. In Proceedings of the First Annual INNS Meeting. New York. Pergamon press. pp. 310.

Parker, D. B. 1985. *Learning logic*. Technical report TR-47. Center for Computational Research in Economics and Management Science, MIT, Cambridge, MA.

Pedrycz, W. & H. C. Card. 1992. *Linguistic Interpretation of Self-organizing Maps.* In IEEE International Conference on Fuzzy Systems.

Polani, D. 1997. *Organization measures for self-organizing maps.* Proceedings of WSOM'97, Workshop on Self-Organizing Maps, Espoo, Finland, June 4-6.

Raiche, A. 1991. Geophysical Journal International. Vol. 105, pp. 629.

Ritter, H.J. & K. Schulten. 1986. *Topoly conserving mappings for learning motor task*. In J.S. Denker (ed.). Neural Network for Computing. AIP Conference proceedings. Snowbird, Utah. Vol. 151, pp. 376-380.

Ritter, H.J. 1991. *Learning with the som*. In Artificial Neural Networks, T. Kohonen, M. Mäkisara, O.Simula & J. Kangas eds. North-Holland, Amsterdam, pp. 379-384

Rodrigues J.S. & L.B. Almeida. 1990. *Improving the learning speed in topological maps of patterns*, Proceedings of INNC, Paris.

Rosemblatt, R. 1962. *Principles of neurodynamicas*. New York: Spartan Books.

Scholtes, J. C. & S. Bloembergen. 1992. *The design of a neural data-oriented parsing (DOP) system.* In IEEE International Joint Conference on Neural Networks.

Skapura, D. M. 1995. *Building neural networks*. Addison-Wesley Publishing Company. ISBN 0-201-53921-7.

Shen, T., J. Gan & L. Yao. 1992. *A generalized placement algorithm based on self-organization neural network.* In IEEE International Joint Conference on Neural Networks.

Su, M. & H. Chang. 2001. *A new model of self-organizing neural networks and its application in data projection*. IEEE Transactions on Neural Networks **12** (1), pp. 153-158.

Ultsch, A., G. Halmans & R. Mantyk. 1991. *CONKAT: a connectionist knowledge acquisition tool.* Proc. 24 Annual Hawaii International Conference on System Sciences. Milutinovic, V & B. D. Shriver (edit.).

Vercauteren, L., G. Sieben & M. Praet. 1990. In International Neural Network Conference. Kluwer, Dordrecht, Netherlands.

Villmann, Th., R. Der & T. Martinetz. 1994. *A new quantitative measure of topology preservation in Kohonen's feature maps*. Neural Networks. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on , Volume: 2 , 27 June-2 July 1994 Pages:645 - 648 vol.2.

Villmann, T., R. Der, M. Herrmann & M. Martinetz. 1997. *Topology preservation in Self-Organizing Feature Maps: Exact Definition and Measurement*. IEEE Transactions on Neural Networks **8**(2), pp. 256-266.

Villmann, Th. & H.-U Bauer. 1998. *Applications of the growing self-organizing map*. Neurocomputing **21**, pp. 91-100.

Walker, A., J. Hallam & D. Willshaw. 1993. *Bee-havior in a mobile robot: the construction of a self-organized cognitive map and its use in robot navigation within a complex, natural environment.* In IEEE International Conference on Neural Networks. San Francisco, California.

Waltz, D. & J.A. Feldman. 1988. *Connectionist Models and Their Implications*. In Waltz, D. & Feldman ,J.A. (De.), Connectionist Models and Their Implications, Norwood, NJ: Ablex Publishing.

Wang, J., J. Rau & C. Peng. 2000. *Toward optimizing a self-creating neural network* . IEEE Transactions on Systems, Man and Cybernetics, Part B **30**(4), pp. 586-593.

Werbos, P. 1974. *Beyond regression: new tools for prediction and analysis in the behavioral sciences*. Tesis doctoral, Harvard, Cambridge, MA.

Widrow, B. 1959. *Adaptative sampled-data systems, a statistical theory of adaptation*. 1959 IRE WESCON Convention Record, part 4. New York: Institute of Radio Engineers.

Widrow, B. 1960. *An adaptative "adaline" neuron using chemical "memistors"*. Technical Report 1552-2, Standford Electronics Laboratory.

Wilson, C. L. 1994. In IEEE International Conference on Neural Networks.

Xu L. 1990. *Adding learning expectation into the learning procedures of som.* Int. Journal of Neural Systems **1**, pp. 269-283.

Zell, A., H. Bayer & H. Bauknecht. 1994. In IEEE International Joint Conference on Neural Networks.

APÉNDICE 1. APLICACIÓN UTILIZADA PARA MEDIR LOS RESULTADOS

Para medir los resultados del método propuesto en este trabajo de grado, se ha desarrollado una simple aplicación, cuyo uso será comentado a continuación.

Instalación:

La aplicación no necesita ser instalada. Consta de un único archivo ejecutable y cinco archivos de texto, los cuales tienen los cinco conjuntos de datos comentados en la sección 6. Se recomienda, para simplicidad del usuario, copiar el ejecutable y los cinco archivos de texto en la misma carpeta del disco duro. Cuando se ejecuta la aplicación, esta lee el directorio de donde se ejecutó en busca de archivos de texto. De esta manera los conjuntos de datos estarán disponibles inmediatamente y así se evita la navegación por las carpetas buscando los archivos con los conjuntos de datos.

Uso:

En la figura a1.1 se puede observar la interface de la aplicación.

Haciendo doble click sobre un archivo de texto en la lista de archivos, la aplicación carga el conjunto de datos seleccionado, a la vez que ejecuta el método AVGSOM con los valores de los parámetros por defecto, y muestra a la derecha el resultado del entrenamiento. Para cargar otro conjunto de datos, simplemente se debe hacer doble click sobre el archivo que se desee abrir.

Debajo de la lista de archivos aparecen los parámetros del algoritmo que pueden ser modificados, donde se destaca el primero de ellos, “Factor de dispersión”. Este parámetro es un número mayor que cero y menor que uno, e indica el grado de dispersión que puede tener la red. Si el factor de dispersión es chico, entonces la red crecerá poco, y la estructura final tendrá pocas neuronas, cuanto mas grande sea el factor de dispersión, la red crecerá en proporción al factor, y por lo tanto la estructura final tendrá muchas neuronas. El parámetro “Vecindad inicial” indica el tamaño del vecindario que se usa al comienzo del entrenamiento. Con el correr de las pasadas este parámetro es automáticamente disminuido en una proporción que se indica con el parámetro “Reducción vecindario”. El parámetro “Factor de aprendizaje”, comentado en los primeros capítulos, indica la proporción del error que será utilizado para el ajuste de los pesos. Este parámetro también disminuye con el correr de las pasadas en una proporción que se indica con el parámetro “Reducción de aprendizaje”. Por último, el parámetro “Límite de iteraciones sin hits” indica el límite de pasadas que puede estar una neurona sin ganar, antes de ser eliminada.

Para poder observar el resultado del entrenamiento con otro, o con el mismo, juego de parámetros, se debe hacer click en el botón “Entrenar”. En cada entrenamiento el mismo conjunto de datos de entrada es utilizado para entrenar el método AVGSOM y el método GSOM. Para poder ver el resultado de uno u otro método están los botones “Ver AVGSOM” y “Ver GSOM”, los cuales muestran el resultado de sus correspondientes métodos.

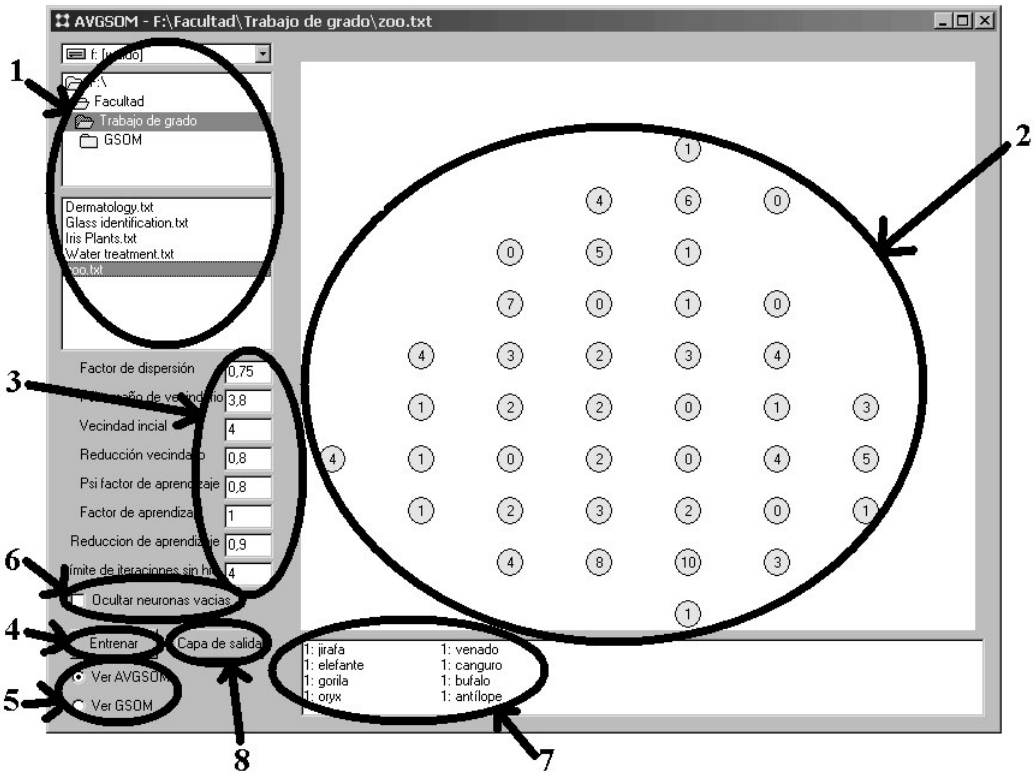


Figura a1.1. La interface de la aplicación. 1) Lista de archivos. 2) El resultado de los algoritmos, cada círculo representa una neurona. 3) Lista de parámetros. 4) Botón para realizar otro entrenamiento. 5) Opciones para visualizar el resultado del entrenamiento de ambos métodos. 6) Opción para ocultar aquellas neuronas que no representan patrones de entrada. 7) La lista de patrones que son representados por una neurona en particular. 8) Botón para entrenar la capa de salida en el método AVGSOM.

La opción "Ocultar neuronas vacías", al estar tildada, no muestra en la red de neuronas, aquellas que no representan a ningún patrón. Esto significa que dada una neurona con un vector de pesos w_i y el patrón de datos x_j mas cercano a w_i, se considera neurona vacía, y que no representa a ningún patrón, si existe otra neurona con vector de pesos w_k, tal que la distancia entre w_k y x_j es menor a la distancia entre w_i y x_j. Si la opción "Ocultar neuronas vacías" no está tildada entonces se muestran las neuronas vacías.

A la derecha, cada círculo representa a una neurona en la red. El número que figura dentro del círculo indica la cantidad de patrones de entrada que representa la neurona. Haciendo click sobre una neurona, en la lista de abajo aparecen los patrones que representa la neurona seleccionada. Cada patrón aparece como un par de valores separados por dos puntos, a la izquierda de los dos puntos aparece el número de clase dado por el creador del conjunto de datos (ver apéndice 2). A la derecha de los dos puntos aparece una descripción del patrón. Esta característica de la aplicación solo resulta útil cuando se está probando el conjunto de datos del zoológico, ya que la descripción es el nombre del animal. En los otros conjuntos de datos, la descripción corresponde a un número arbitrario.

Una vez entrenado el AVGSOM, se puede entrenar la capa de salida y así clasificar las neuronas de la red. Al presionar el botón “Capa de salida”, se comienza con el entrenamiento de la capa de salida del AVGSOM y como resultado se tendrán los distintos clusters detectados por el algoritmo. Las neuronas que pertenezcan a un mismo cluster serán pintadas con un mismo color.

APÉNDICE 2. CONJUNTOS DE DATOS UTILIZADOS PARA LAS PRUEBAS DEL MÉTODO AVGSOM

A continuación se describen las características de los cinco juegos de datos utilizados para medir los resultados del método AVGSOM propuesto en este trabajo de grado.

I) *Iris Plants Database*

El conjunto de datos contiene tres clases de 50 patrones cada una, donde cada una de las clases hace referencia a un tipo de la planta iris. Una de las clases es linealmente separable con respecto a las otras dos, estas a su vez no son linealmente separables entre si.

Cada patrón posee cuatro atributos numéricos:

- ancho del sépalo
- alto del sépalo
- ancho del pétalo
- alto del pétalo

Las cuatro mediciones están en centímetros. Las tres clases de la planta corresponden a *Iris setosa*, *Iris versicolour* e *Iris virginica*.

II) *Glass Identification Database*

El estudio de la clasificación de tipos de vidrios es muy utilizado en la investigación criminalística. En la escena del crimen, fragmentos de vidrios pueden ser utilizados como evidencia si son correctamente identificados.

Esta base de datos contiene información sobre siete tipos de vidrios distintos. Posee 214 patrones de entrada y cada patrón se encuentra representado por nueve atributos numéricos:

- Índice de refracción
- Porcentaje de sodio
- Porcentaje de magnesio
- Porcentaje de aluminio
- Porcentaje de silicio
- Porcentaje de potasio
- Porcentaje de calcio
- Porcentaje de bario
- Porcentaje de hierro

Los porcentajes de presencia corresponden al peso del óxido correspondiente a cada metal.

Las clases se corresponden a: vidrios por procesos de flotación para edificios, vidrios por procesos de no flotación para edificios, vidrios por procesos de flotación para vehículos, vidrios por procesos de no flotación para vehículos, (ninguno en esta base de datos), contenedores, vajilla y faro.

Los patrones están distribuidos de la siguiente manera:

- 163 vidrios de ventana (ventanas de edificios y de vehículos)
  - 87 procesos de flotación
    - 70 ventanas de edificios
    - 17 ventanas de vehículos
  - 76 procesos de no flotación
    - 76 ventanas de edificios y casas
    - 0 ventanas de vehículos
- 51 vidrios (no ventana)
  - 13 contenedores
  - 9 vajilla
  - 29 faro

III) *Zoo database*

Esta base de datos contiene siete clases distintas de animales (mamíferos, aves, anfibios, peces, reptiles, insectos y otros invertabrados no insectos). Contiene 101 patrones de entrada y cada patrón tiene 16 atributos de los cuales 15 son binarios, y uno numérico:

- Tiene pelo
- Tiene plumas
- Pone huevos
- Da leche
- Vuela
- Es acuático
- Es depredador
- Es dentado
- Es vertebrado
- Tiene branquias
- Es venenoso
- Tiene aletas
- Tiene cola
- Es doméstico
- Es de tamaño chico (relativo a un gato doméstico)
- Cantidad de patas

Los primeros 15 atributos son de tipo boolean cuya codificación es: 1 si posee la característica y 0 si no. El último atributo representa el número de patas y puede tomar uno de los siguientes valores: 0, 2, 4, 5, 6 y 8.

Los patrones están distribuidos de la siguiente manera:

- 41 mamíferos
- 20 aves
- 5 reptiles
- 13 peces
- 4 anfibios
- 8 insectos
- 10 otros invertabrados no insectos

IV) *Dermatology Database*

El diagnóstico diferencial de enfermedades eritematosas-escamosas es un problema real en dermatología. Todos ellos comparten las características clínicas del eritema las escamas, con muy pequeñas diferencias. Las enfermedades en este grupo son psoriasis, dermatitis seborreica, eccema liquenoide, pitiriasis, dermatitis crónica, y pitiriasis *rubra pilaris*. A menudo es necesaria una biopsia para el diagnóstico pero desafortunadamente estas enfermedades comparten muchas características también histopatológicas. Otra dificultad para el diagnóstico diferencial es que una enfermedad puede mostrar las características de otra enfermedad en una etapa inicial y puede tener las mismas características en las siguientes etapas.

Se evaluaron 12 características clínicas en pacientes. Luego, fueron tomadas muestras de la piel y se evaluaron 22 características histopatológicas. Los valores de las características histopatológicas son determinadas durante el análisis de las muestras bajo un microscopio.

Este conjunto de datos contiene 366 patrones de entrada pertenecientes a seis clases, donde cada una representa una enfermedad distinta. Cada patrón tiene 34 atributos, 32 correspondientes a características de la piel, uno corresponde a la historia familiar y el último a la edad del paciente:

- eritema
- escamosidad
- bordes definidos
- picazón
- fenómeno de Koebner
- pápulas poligonales
- pápulas foliculares
- involucramiento de las mucosas orales
- involucramiento de la rodilla y el codo
- involucramiento del cuero cabelludo
- incontinencia de melanina
- eosinófilos en el infiltrado
- infiltrado de PNL
- fibrosis de la dermis papilar
- exocitosis
- acantosis
- hiperqueratosis
- paraqueratosis

- clubbing of the rete ridges1
- elongation of the rete ridges1
- adelgazamiento de la epidermis suprapapilar
- pústulas espongiformes
- microabceso de Munro
- hipergranulosis focal
- disaparición de la capa granular
- vacuolización y daño de la capa basal
- espongiosis
- saw-tooth appearance of retes1
- follicular horn plug
- paraqueratosis perifolicular
- infiltrado monoluclear inflamatorio
- infiltrado en forma de banda
- Historia familiar
- Edad

En el conjunto de datos construido para este problema, el atributo historia familiar tienen el valor 1 si se observa que cualquiera de las enfermedades ha sido identificada en la familia y 0 en caso contrario. El atributo edad simplemente representa la edad del paciente. Cada uno de los otros atributos (clínicos e histopatológicos) están dentro del rango de 0 a 3. 0 indica que el atributo no está presente, 3 indica que está presente en su totalidad y 1 y 2 indican valores relativos intermedios

Los patrones están distribuidos de la siguiente manera:

- 112 psoriasis
- 61 dermatitis seborreica
- 72 eccema liquenoide
- 49 pitiriasis
- 52 dermatitis crónica
- 20 pitiriasis *rubra pilaris*

V) *Faults in a urban waste water treatment plant Database*

Esta base de datos contiene medidas diarias de sensores en una planta de tratamiento de agua de deshechos urbanos. El objetivo es clasificar el estado operacional de la planta con el objetivo de predecir fallos a través de los estados variables de la planta en cada etapa del proceso de tratamiento.

Los 527 patrones de entrada (cada patrón se corresponde con la medición de un día) se corresponden a 13 clases. Cada clase representa el estado de la planta, la clase con más patrones es la que representa el correcto funcionar de la planta. El resto de las clases están discriminadas según los problemas que se encontraron en el producto final durante la toma de datos, exceso de sólidos, etc. Cada patrón tiene 38 atributos correspondientes a distintas mediciones en el agua, tanto en la entrada como en la salida de la planta:

- flujo en la entrada de la planta
- zinc en la entrada de la planta
- pH en la entrada de la planta
- demanda biológica de oxígeno en la entrada de la planta
- demanda química de oxígeno en la entrada de la planta
- sólidos suspendidos en la entrada de la planta
- sólidos suspendidos volátiles en la entrada de la planta
- sedimentos en la entrada de la planta
- conductividad en la entrada de la planta
- pH en la entrada del precipitador primario
- demanda biológica de oxígeno en la entrada del precipitador primario
- sólidos suspendidos en la entrada del precipitador primario
- sólidos suspendidos volátiles en la entrada del precipitador primario
- sedimentos en la entrada del precipitador primario
- conductividad en la entrada del precipitador primario
- pH en la entrada del precipitador secundario
- demanda biológica de oxígeno en la entrada del precipitador secundario
- demanda química de oxígeno en la entrada del precipitador secundario
- sólidos suspendidos en la entrada del precipitador secundario
- sólidos suspendidos volátiles en la entrada del precipitador secundario
- sedimentos en la entrada del precipitador secundario
- conductividad en la entrada del precipitador secundario
- pH en la salida de la planta
- demanda biológica de oxígeno en la salida de la planta
- demanda química de oxígeno en la salida de la planta
- sólidos suspendidos en la salida de la planta
- sólidos suspendidos volátiles en la salida de la planta
- sedimentos en la salida de la planta
- conductividad en la salida de la planta
- performance en la demanda biológica de oxígeno en la entrada del precipitador primario
- performance en los sólidos suspendidos en la entrada del precipitador primario
- performance en los sedimentos en la entrada del precipitador primario
- performance en la demanda biológica de oxígeno en la entrada del precipitador secundario
- performance en la demanda química de oxígeno en la entrada del precipitador secundario
- performance global en la demanda biológica de oxígeno en la entrada
- performance global en la demanda química de oxígeno en la entrada
- performance global en los ólidos suspendidos en la entrada
- performance global en los sedimentos en la entrada

Los patrones están distribuidos de la siguiente manera:

- 275 días de situación normal-1
- 1 día con problemas del tipo 1 en el precipitador secundario
- 1 día con problemas del tipo 2 en el precipitador secundario
- 4 días con problemas del tipo 3 en el precipitador secundario

- 116 días de situación normal con una performance superior al promedio
- 3 días con sobrecarga de sólidos-2
- 1 día con problemas del tipo 4 en el precipitador secundario
- 1 día con tormenta-1
- 69 días de situación normal con baja entrada
- 1 día con tormenta-2
- 53 días de situación normal-2
- 1 día con tormenta-3
- 1 días con sobrecarga de sólidos-2

APÉNDICE 3. ALGORITMOS DE ENTRENAMIENTO DE ARQUITECTURAS DE REDES NEURONALES DESARROLLADAS EN MATLAB

Entrenamiento de un perceptrón

```
function [W, b, ite] = Perceptron(X, T, MAX_ITERA, alfa, W, b)

% X representa el conjunto de patrones de entrada. Cada columna corresponde a
% un patron distinto
% T contiene el valor esperado para cada patron. Sus valores son 0 o 1.
% MAX_ITERA  = maxima cantidad de iteraciones a realizar (por si los patrones
% no son linealmente separables)
% alfa = velocidad de aprendizaje
% W es el vector de pesos inicial correspondiente a c/caracteristica de
% entrada
% b es el peso del arco del bias que tiene entrada siempre igual a 1
% ------------------------------------------
% Esta funcion devuelve los valores W y b obtenidos por entrenamiento y la
% cant.de iteraciones realizadas


[Entradas, CantPatrones] = size(X);

Fun = 'purelin';
ite = 0;

CuantosMal = 1;
while (ite < MAX_ITERA) & (CuantosMal>0) ,
   % actualizar los valores de W con las diferencias de cada patrón
   for patr=1:CantPatrones,
      neta = W * X(:, patr)+b;
      salida = hardlim(feval(Fun, neta));

      Error_Actual = (T(patr) - salida);

      W = W + alfa * Error_Actual * X(:, patr)';  % X' es un vector fila
      b = b + alfa * Error_Actual ;
   end

   ite = ite + 1;

   CuantosMal = sum(T ~= hardlim(W*X+b*ones(1,CantPatrones)));
end
```

Entrenamiento utilizando la Regla Delta

```
function [W, b, ite]= Regla_Delta(Fun, X, T, MAX_ITERA, alfa, W, b, CotaError)

% Fun es la funcion de activacion a utilizar
% X representa el conjunto de patrones de entrada. Cada columna corresponde a
% un patron distinto
% T contiene el valor esperado para cada patron. Sus valores son 0 o 1.
% MAX_ITERA  = maxima cantidad de iteraciones a realizar (por si los patrones
% no son linealmente separables)
% alfa = velocidad de aprendizaje
% W es el vector de pesos inicial correspondiente a c/caracteristica de
% entrada
% b es el peso del arco del bias que tiene entrada siempre igual a 1
% ------------------------------------------
% Esta función devuelve los valores W y b obtenidos por entrenamiento y la
% cant.de iteraciones realizadas

[Entradas, CantPatrones] = size(X);

Fun_deriv = feval(Fun, 'deriv');

ite = 0;

ErrorAct = CotaError+1;
while (ite < MAX_ITERA) & (ErrorAct > CotaError) ,
   % actualizar los valores de W con las diferencias de cada patrón
   for patr=1:CantPatrones,
      neta = W * X(:, patr)+b;
      salida = feval(Fun, neta);
      Deriv_Salida = feval(Fun_deriv, neta, salida);
      Error_Actual = (T(patr) - salida)*Deriv_Salida;

      W = W + alfa * Error_Actual * X(:, patr)';  % X' es un vector fila
      b = b + alfa * Error_Actual ;
   end

   ite = ite + 1;
   salidas = feval(Fun, W*X+b*ones(1,CantPatrones));
   ErrorAct = sqrt(sumsqr(T - salidas))
End
```

Resolucion del problema del XOR utilizando una BackPropagation

```
P = [ -1 -1  1 1;   -1  1 -1 1];
T = [ 0 1 1 0 ];
T2 = [-1 1 1 -1];

plotpv(P,T);
xlabel('X1');      ylabel('X2');

[Entradas, CantPatrones] = size(P);
Ocultas = 2;
Salidas = 1;

% Inicializar la red
w1 = rand(Ocultas,Entradas) - 0.5 * ones(Ocultas,Entradas);
b1 = rand(Ocultas,1) - 0.5 * ones(Ocultas,1);

w2 = rand(Salidas,Ocultas) - 0.5 * ones(Salidas, Ocultas);
b2 = rand(Salidas,1) - 0.5 * ones(Salidas,1);

% Calcular la salida de la BPN y graficar
netah = w1*P+b1*ones(1,CantPatrones);
salida = [];        salida = w2 * tansig(netah) + b2;
linea = plotpc(w1, b1);
% Calcular el error
ErrorSalida = (T2 - salida);
AVGError = sum(ErrorSalida .^2);
% Corregir el vector de pesos hasta obtener un valor aceptable para W y B
fm = 10;    %frecuencia con la que se muestran los datos
alfa = 0.05;
CotaError = 0.001;
MAX_ITERA = 1000;
itera = 0;
while ( AVGError > CotaError ) & ( itera <= MAX_ITERA  );
   for patr=1:CantPatrones;
      % Calcular el error de la capa oculta
      netah=[];
      netah = w1*P(:,patr) + b1;
      salida = [];            salida = w2 * tansig(netah) + b2;
      ErrorSalida =[];        ErrorSalida = (T2(patr)-salida);
      ErrorOculta=[];
      ErrorOculta = (1-tansig(netah).^2) .* (w2' * ErrorSalida);
      % actualizar los pesos (W y B)
      w2 = w2 + alfa * ErrorSalida * tansig(netah)';
      b2 = b2 + alfa * ErrorSalida;
      w1 = w1 + alfa * ErrorOculta * P(:,patr)' ;
      b1 = b1 + alfa * ErrorOculta;
   end;
   if mod(itera,fm)==0
       title(sprintf('itera = %d\n',itera));
       linea = plotpc(w1, b1, linea);
       drawnow
   end;
   % Calcular el nuevo valor de salida
   netah = w1*P+b1*ones(1,CantPatrones);
   salida = [];
   salida = w2 * tansig(netah) + b2;
   % Calcular el error de la capa de salida = error de la BPN
   ErrorSalida = (T2-salida);
   AVGError = sum(ErrorSalida .^2);
   itera = itera + 1;
end

salida
ErrorSalida
AVGError
```

Clasificación de los patrones de la Base ZOO utilizando SOM

```
%inputVectorNames es un arreglo con los nombre de los animales
inputVectorNames = dlmread('names.dat');

%inputVectors es una matriz de 101 x 17 columnas con las caracteristicas de
cada animal
inputVectors = dlmread('zoo.dat');

%Descarto el atributo de tipo
NroClase = inputVectors(:,17);

inputVectors(:,17) = [];

inputVectorSize = size(inputVectors);

%Rango de los valores de entrada
inputVectorRange = [1,1,1,1,1,1,1,1,1,1,1,8,1,1,1];

[CantPatrones, entradas] = size(inputVectors);
%Escalar los datos de entradas
for i=1:CantPatrones,
    inputVectors(i,13) = inputVectors(i,13)/8;
end

[CantPatrones, entradas] = size(inputVectors);
%escalar las entradas entre 0 y 1
menor = min(inputVectors,[],1);
mayor = max(inputVectors,[],1);
for c = 1:entradas,
    inputVectors(:,c) = (inputVectors(:,c)-
menor(c)*ones(CantPatrones,1))/(mayor(c)-menor(c))
end

% Disposicion de las neuronas de la capa oculta
fil_ocultas = 8;
col_ocultas = 6;
pos=gridtop( col_ocultas, fil_ocultas);
pasos=linkdist(pos);

ocultas = fil_ocultas * col_ocultas;
W = 0.5 * ones(ocultas, entradas);

ITE_MAX = 150;

ite = 0;
alfa = 0.25;
cambio = 1;

cambio=1;
vecinas = 3;
intervalo = 20;
ite0 = 0

while ( ite <= ITE_MAX ) & ~(( ite0>intervalo ) & ( cambio<0.000000001 )),
    W_vie = W;
    ite = ite + 1;
    for i=1:CantPatrones,
       % Buscando la neurona ganadora
       DMax = -sqrt( sum((inputVectors(i,:) - W(1,:)).^2));
       mayor = 1;
       for j=2:ocultas,
           Dist = -sqrt( sum((inputVectors(i,:) - W(j,:)).^2) );
           if Dist > DMax
```

```
                DMax = Dist;
                mayor = j;
            end
        end
        %Actualizar la neurona mas proxima
        for entra = 1:entradas,
           W(mayor, entra) = W(mayor, entra) +
                             alfa * (inputVectors(i, entra) - W(mayor, entra));
        end

        %Actualizar la neurona ganadora y su vecindad
        for ventana = 1:ocultas,
           if pasos(mayor, ventana)<=vecinas,
              for entra = 1:entradas,
                 W(ventana, entra) = W(ventana, entra) +
                            alfa * (inputVectors(i,entra) - W(ventana, entra));
              end
           end
        end
    end
    if (vecinas>=1) & (mod(ite,intervalo)==0), vecinas = vecinas-1; end
    if (vecinas==0),
          ite0 = ite0 + 1;
    end;

    cambio = sum(sqrt(sum((W-W_vie).^2,2)));
end

% termino el entrenamiento
% calcular para cada clase la lista de patrones que contiene
clases = [];
clases = zeros(ocultas,max(NroClase));
nombre = cell(ocultas,ocultas,50);
for i = 1:CantPatrones,
    %Calcular la neurona ganadora
       DMax = -sqrt( sum((inputVectors(i,:) - W(1,:)).^2));
       mayor = 1;
       for j=2:ocultas,
           Dist = -sqrt( sum((inputVectors(i,:) - W(j,:)).^2) );
           if Dist > DMax
               DMax = Dist;
               mayor = j;
           end
       end

    ClaseOriginal = NroClase(i);
    clases(mayor, ClaseOriginal) = clases(mayor, ClaseOriginal) + 1;
    nombre(mayor, ClaseOriginal, clases(mayor, ClaseOriginal)) =
                                              {[char(inputVectorNames(i,:))]};
end

mapa = cell(fil_ocultas, col_ocultas);
for nro = 1:ocultas,
    fila = fil_ocultas - floor( (nro-1)/col_ocultas );
    col  = mod( (nro-1), col_ocultas ) + 1;
    texto = '';
    for cl =1:max(NroClase),
         if clases(nro, cl)>0,
             if texto ~= '',
                 texto = strcat(texto, ' ; ');
             end
             texto = strcat(texto , 'c', int2str(cl), '-->',
                     int2str(clases(nro,cl)));
         end;
     end;
     mapa(fila,col) = {[texto]};
 end;
```

APÉNDICE 4. DYNAMIC SELF-ORGANIZING MAPS.

El siguiente artículo, que presenta al método AVGSOM descripto en este trabajo de grado, fue publicado en la XXXI Conferencia Latinoamericana de Informática. 24-28 de octubre de 2005, Cali, Colombia.

Dynamic Self-Organizing Maps.
A new strategy to upgrade topology preservation

Waldo Hasperué
Universidad Nacional de La Plata, Facultad de Informática,
La Plata, Argentina, 1900
whaspe@infovia.com.ar

Laura Lanzarini
Instituto de Investigación en Informática LIDI
Universidad Nacional de La Plata, Facultad de Informática,
La Plata, Argentina, 1900
laural@lidi.info.unlp.edu.ar

Abstract

Self-organizing maps are a tool highly employed in the solution of clustering problems and count with the capacity of preserving data topology. This allows its direct application in visualization and the reduction of input space dimensions. However, the need of defining a priori the size of the architecture may lead to obtain a neural network with low efficiency.

In order to solve this problem, dynamic self-organizing maps propose strategies to adapt the neural network dimension to the problem's needs allowing, during the training phase, adding and/or deleting neurons within the structure. Unfortunately, this dynamic process may put the preservation of the topology at risk by inserting elements representing similar inputs in distant locations within the network. For this reason, it is of outmost importance to analyze the way in which this is carried out.

The method proposed in this paper, called AVGSOM, suggests a new strategy to insert elements within a rectangular architecture. The results obtained from its application into different sets of data have been satisfactory, and have been also compared to GSOM since it counts with a similar representation type.

Finally, conclusions are presented together with some future lines of work.

Keywords: Clustering Methods, Neural Networks, Self-Organizing Features Maps, Unsupervised Learning

1. Introduction

Self Organizing Maps (SOM) are a highly used tool in Data Mining. Their capacity to project the input space in a low-dimensional regular grid allows employing it to visualize and explore the data properties.

This is possible since a projection respecting the data topology is created, so that neighboring elements in the input space will remain as such within the low-dimensional structure determined by the prototype vectors.

Each of these vectors corresponds to an artificial neuron and is part of a unique structure which gives rise to the definition of a neural network. Neurons making up such network present a competitive behavior and adapt themselves through an unsupervised training. This allows the network to discover, on its own, the quality and characteristics of input data.

However, the great SOM disadvantage lies in the static definition of its architecture since the quantity and type of connection of the neurons making it up must be defined a priori, before the training, thus conditioning the network's efficiency and efficacy.

In order to solve this problem, alternative solutions have been proposed, referred to as Dynamic Self-organizing Maps, which allow incorporating and/or deleting elements during the training. In these cases, the growth of the structure is carried out in function of how many times each neuron wins. Those who win more times are reinforced with new neighbors with the aim of distributing the prototype vectors more appropriately.

So as to work with this type of architectures, the network neurons should count with the possibility of incorporating or deleting neighbors dynamically. This aspect, added to the fact that the neuron overload is measured separately, may lead to the loss of topology preservation. This is shown in the location of similar grouping representatives (competitive neurons) in really distant positions within the neural network.

The objective of the present paper is to propose a new method, called AVGSOM, which allows the generation of a dynamic self-organizing map with a proper preservation of the input data topology.

This paper is organized as follows:

- Section 2 mentions the general aspects verified in the definition of a dynamic self-organizing map and briefly describes, as examples, the Growing cell structures (GCS) and Growing neural gas (GNG) methods proposed by Fritzke [3][4] and the Growing self-organizing map (GSOM) method proposed by Alahakoon [1].
- Section 3 describes the AVGSOM method.
- Section 2 present the method used to measure the topology preservation.
- Since AVGSOM uses a rectangular architecture similar to GSOM, this method will be used in order to compare the results obtained. Section 5 shows the results of the application of both methods into the data clustering of three different testing sets.
- Finally, Section 6 presents the conclusions and some future lines of work.

2. Dynamic Self-Organizing Maps

In general, most of the existing strategies to define dynamic self-organizing maps have the following characteristics [5]:

- The network structure is a graph made up of networked competitive neurons. The connection type is regular and is of utmost importance for the visualization of the input space reduction.
- Each neuron corresponds to a prototype vector, which attempts to represent a set of similar input data. The similarity measure to be used depends on the problem.
- Training is carried out through a competitive process in which neurons attempt to represent input data. For each data, its resemblance to each neuron's prototype vector is assessed, considering the nearest as the winner. Adaptation is principally applied to the winner neuron and, to a lesser extent, to its neighboring environment. This is what allows correcting the structure so as to preserve topology.

- At each adaptation step, the local error information is accumulated in the winner neuron. This aims at avoiding that a same element of the network accumulates the representation of most of the input patterns. The error calculation depends on the application.
- The accumulated error information is used to determine the place where new units should be inserted in the network. When the insertion is carried out, the error information is redistributed locally, thus avoiding new insertions at the same location.

It can be observed in the previous list that the three first characteristics correspond to the SOM defined by Kohonen [9][10], whereas the last two are related to the need of identifying the location in which new elements should be inserted in the architecture.

In addition, the need of modifying, during the training, the quantity of elements of the network leads to represent separately, for each neuron, the corresponding prototype vectors and their closest neighbors.

This does not occur in SOM because the quantity of structure neurons and their connection type are defined a priori and, thus, it just remains to identify the corresponding prototype vector values.

Next, some of the proposals to obtain competitive neural networks, whose size is fixed during training, are presented as example.

2.1. Growing Cell Structures (GCS)

This model uses the figure of hypertetrahedron of k dimensionality defined a priori. A hypertetrahedron of k dimension is the simplest polyhedron which can be shaped with k+1 vertices. Examples of hypertetrahedrons for $k \in \{1,2,3\}$ are lines, triangles and tetrahedrons.

In GCS, the architecture begins with the only hypertetrahedron over which the adaptation steps previously described are applied. After a number λ of these adaptation steps, unit q is determined to have the maximum accumulated error and a new neuron is inserted dividing the arc that separates q from its farthest direct neighbor. Then, the necessary connections to preserve the architecture shape are added, so that it is always made up of k dimension hypertetrahedrons. This process of re-connection is really simple. If the new neuron was inserted between q and f, its direct neighbors will be q, f and all the neighbors common to q and f.

Since GCS has a dimensionality defined a priori, it can be employed as visualization tool using k with value 2 or 3.

A detailed description of this architecture and its training method can be found in [3].

2.2. Growing Neural Gas (GNG)

- The GNG method does not impose any restriction on how competitive neurons are connected. Its functioning combines the GCS growing mechanism and the topology generation through the CHL - Competitive Hebbian Learning - proposed by Martinetz [11][12].
- This learning method does not change the values of the reference vectors; it only generates connections among neighbor neurons so as to build a graph optimally preserving the topology. In particular, the graph's final connections correspond to the Delaunay triangulation of the given reference vector set.
- In order to combine both tasks, GNG proposes the application of connection generation according to CHL, incorporating the concept of *age* to each connection. In each step of the adaptation, apart from correcting the prototype vector associated to the winner neuron and increasing its error value as previously explained, the age of the arcs which connect it to its direct neighbor are increased. When the age of a connection reaches a determined threshold, the connection is eliminated. Neurons which lose all of their connections are also eliminated.
- The growth process of the structure is regulated by a parameter λ which represents the adaptation step threshold to be expected in order to insert a new neuron to the structure. When this threshold is reached at, the neuron q possessing the maximum accumulated error is determined. Then, a new neuron r is created, between q and f. r is connected to q and f and the direct connection between q and f is eliminated. Eventually, the accumulated errors of q and f are reduced in a fraction indicated a priori and the error of r is calculated as the average of the errors of q and f.

- The complete process is repeated until the maximum allowable number of neurons within the network is reached at, or until some criterion of previously established termination is fulfilled.
- We do recommend the reading of [4] in order to count with a detailed understanding of how this method works.

2.3. Growing Self-Organizing Map (GSOM)

- This method organizes neurons in a bi-dimensional grid, allowing each element of the network to have at least four direct neighbors.
- The initial structure is made up of connected neurons so that each of them only has two neighbors. Before the training is started, the threshold *GT* is determined; this threshold must reach the accumulated error of a neuron so as to consider the need of adding a new immediate neighbor.
- The training process consists of two parts: the first generates all the neurons of the structure, and the second arranges the weight vectors corresponding to each unit of the network.
- The generating stage respects, in general lines, the initially described process since in each step of the adaptation, the corresponding weight vectors are re-calculated and the winner neuron error is increased. When such error surpasses the threshold indicated by *GT*, a new neighbor is added to the winner neuron. This happens provided it has less than four neighbors; otherwise, only the weights are distributed. This step ends when a minimum growth is achieved or when no more neurons are produced.
- Finally, once the architecture of the network is obtained, a smoothing stage is carried out reducing the learning speed and the neighborhood size.
- An important concept presented by Alahakoon in [1] is the use of a parameter called *spread factor*, which allows indicating the network to what extent it should spread or grow. In this way, the network may grow little in order to have a wide vision of which the main clusters made up by the input data are, or may spread itself much more thus obtaining a deeper detail of each cluster.
- The neurons' bi-dimensional organization allows an easy visualization of the clusterings obtained after the training independently of the input pattern dimensionality.

3. AVGSOM

As already explained, the preservation of the input data topology is a fundamental characteristic of self-organizing maps. When the architecture is established a priori, it is almost a natural fact due to the way weights adapt themselves. However, when the architecture grows dynamically, it is very hard to represent the similar input data sets with neighboring neurons within the architecture of the network.

The biggest problem is the creation of a new neuron. Depending on the criterion employed to determine the prototype vector of a new neuron, similar vectors may be assigned to two distant neurons in the input space. Notice that this situation cannot be solved during the training since the neighbors involved are totally separated.

In this direction, the method presented in this paper proposes a new strategy to solve this problem.

AVGSOM uses a rectangular grid in which each neuron has a maximum of four neighbors. The training starts with a minimum structure of four neurons, where each of them has two neighbors, making up a 2x2 matrix and the corresponding prototype vectors are initialized at random. In each iteration step, prototype vectors are adapted, and the error is accumulated in the winner neuron as usual.

The fact that a neuron, during the training, surpasses the accumulated error threshold established a priori indicates the need of adapting the structure in order to avoid an excessive accumulation of patterns around the winner. Any modification over the structure should respect the rectangular topology initially proposed since its eases the visualization of input data. For this reason, if the winner already has four neighbors, the difference between the winner and its direct neighbors should be reduced, thus allowing the other elements of the network to win the competition in the successive steps of the adaptation.

If the winner has less than four direct neighbors, a new element will be inserted in the structure. It is very important to initialize the new neuron with a weight vector not very different from the rest of its

neighbors so that, when the weights of both the new neuron and its neighbors are updated, it does not distort the topology of the network during the learning.

The method used to determine the weight vector of a new neuron in AVGSOM is the average of the prototype vectors of those which will be neighbors of the new neuron. In this way, not only are the elements of the network related within the graph, but also the new neuron weight vector remains "in the middle" of the weight vectors of its neighbors in the input space, preserving the data topology almost in their totality.

Next, the training algorithm used by AVGSOM is presented.

3.1. Algorithm

Begin with a neuron network made up of four neurons where each of them only has two neighbors.

Initialize with random values of the prototype vectors corresponding to each neuron of the Network.

Repeat

For each input pattern

- Enter the pattern to the neural network
- Identify as winner neuron that having the prototype vector which does indeed resemble the pattern recently entered into the network according to some similarity measure defined a priori. Even though this paper has employed Euclidean Distance, the election of this measure depends on the problem.
- Carry out the adaptation in the prototype vector corresponding to the winner neuron and its neighborhood, such as SOM generally does.
- Accumulate in the winner neuron the magnitude of the corresponding error. Such value corresponds to the similarity measure assessed in the previous point.

End For

For each neuron of the network

- If the neuron error surpasses the threshold GT
  - If the neuron has four direct neighbor then
    - Distribute the error accumulated by the winner neuron among its direct neighbors. With this, we try to give the neighbors a greater opportunity to saturate themselves, in the next iterations, and, in this way, "push" the error towards the limits of the network thus achieving its expansion.

  If otherwise
    - Select at random one of the free places to generate the new neighbor neuron.
    - The prototype vector corresponding to this new neuron is calculated as the average of the prototype vectors of those that will be its neighbors.
    - Assign zero as the accumulated error for this new neuron.
    - Assign zero to the accumulated error of this neuron.
- If this neuron never wins then
  - Increase to 1 its failure counter.
- If the failure counter of this neuron reaches the pre-established threshold, it must then be eliminated.

End For

Until no new neurons are created or the growing rate is the minimum.

The *GT* threshold is calculated in the same way as in GSOM, i.e. $GT = -D * ln(SF)$ being *SF* a value between 0 and 1, corresponding to the spread factor, indicated as parameter [1].

4. Topology Preservation

Each neuron i of the network has a related prototype vector, w_i , which defines the receptor field or Voronoi's polyhedron V_i made up by all the input data for which w_i is the most alike according to a similarity measure defined a priori.

It is said that this mapping preserves the topology if two neighboring competitive units in the neural network have adjacent receptive fields in the input data space.

In this way, if the network properly preserves the topology, similar input patterns will be mapped within the neurons closer to the network.

One of the simplest topology preserving measures is that indicated in [7], and used in [8]. This measure calculates the network distortion percentage basing on a score measuring the relation among neighboring neurons as follows:

$$adjacents(i) = \begin{cases} 1 & si \parallel r_{mejor}(i) - r_{2doMejor}(i) \parallel > 1 \\ 0 & otherwise \end{cases} \quad (1)$$

$$ET = \frac{\sum_{i=1}^{N} adjacents(i)}{N} \quad (2)$$

where rMejor(i) and r2doMejor(i) are the positions of the neurons of the network whose prototype vectors have the first and second shortest distance to the i-th input data. Its difference module represents the length of the shortest path which links them within the neural network. Finally, N is the total number of inputs presented in the network.

As can be noticed, the function indicated in (2) will give back values between 0 and 1. In particular, it will yield 0 when, for each input pattern, the network neurons, whose prototype vectors are more alike, are direct neighbors. In this case, it can be said that the network completely preserves the topology. On the contrary, it will yield 1 when this cannot be verified for any input pattern. In this last case, the network does not preserve the input pattern topology at all.

Even though several methods have been proposed to measure the degree of data topology preservation in a self-organizing map [2][13][14][15], this simple measure will be enough to compare the results of the method proposed in this paper.

5. Results

Three sets of data bases of the repository of data bases UCI [6] have been used to measure the behavior of AVGSOM:

VI) *Iris Plants Database*

It is a data base which contains information about different types of the iris plant. In it, there are three different classes represented by 50 patterns in each. For each data, four features are presented containing the width and the height of the petal and the sepal measured in centimeters. It has the peculiarity that only one of the classes is linearly separable.

VII) *Glass Identification Database*

This database contains information about seven types of different glasses. It counts with 214 input patterns and each pattern is represented by nine features: the refraction index, and the presence percentage of eight metals (sodium, magnesium, aluminum, silicon, potassium, barium, calcium, and iron).

VIII) *Zoo database*

This database contains seven different classes of animals (mammals, birds, amphibious, fish, insects, etc.). It contains 101 input patterns and each pattern has 16 features, from which 15 are binary (it has fur, lays eggs, yields milk, is poisonous, flies, has feathers, etc.), and one is not (quantity of legs).

In all of these cases, the elements of the input patterns were normalized between 0 and 1 before starting with the training.

GSOM was used as comparison point in order to measure the efficiency of the results obtained with AVGSOM, with the following considerations:

- A random order was generated to enter the patterns of each iteration.
- For the same iteration, the patterns were entered in the same order in both methods.

- Several trainings were carried out using the three sets of data previously described together with three batches of parameters, which are characterized by the spread factor. Iterations were carried out with little spread (0.2), medium spread (0.5), and high spread (0.85) in the map.

Table 1 shows the results obtained. For each of the three sets of data, it shows the quantity of trainings in which the proposed method obtained a topological error smaller than GSOM, together with the average topological error after the iterations. The table discriminates three different sets of parameters, each set with a different spread factor.

For instance, table 1 shows that for the Iris Base, after training 40 times each of the methods with a spread factor of 0.2, in 25 times the topological error of AVGSOM was smaller than that of GSOM. This situation is also observed in the average topological error which, for AVGSO, is 0.111, and for GSOM is 0.139.

The data of Table 1 allows us to notice that the proposed algorithm successfully preserves the topology of the input data, being outperformed by GSOM in the iris plant database when a spread factor of 0.5 or 0.85 is used.

	AVGSOM		GSOM	
	Quant.iterat. with < error	Average topological error	Quant.iterat. with < error	Average topological error
SF= 0.2				
Iris Plants Database (*n*=40)	25	0.111	15	0.139
Glass Identification Database (*n*=60)	36	0.140	24	0.163
Zoo database (*n*=20)	14	0.071	6	0.107
SF= 0.5				
Iris Plants Database (*n*=40)	22	0.152	18	0.139
Glass Identification Database (*n*=40)	25	0.090	15	0.118
Zoo database (*n*=70)	45	0.104	25	0.138
SF= 0.85				
Iris Plants Database (*n*=20)	5	0.151	15	0.100
Glass Identification Database (*n*=30)	16	0.108	14	0.108
Zoo database (*n*=50)	32	0.153	18	0.165

Table 1: In the first column appeared the three sets of data used and, between parentheses, the quantity of times (*n*) the methods were trained with this set of data. For each method, the table shows the quantity of iterations where the topological error was smaller than the other method, and the average topological error for the *n* iterations.

6. Conclusions and future lines of work

A new strategy to insert neurons in a dynamic self-organizing map has been presented with really successful results in terms of input data topology preservation. Its application in the solution of three concrete cases has proved to be superior to GSOM.

We are currently working on a new insertion strategy which allows incorporating new neurons inside the neural network. The AVGSOM proposal makes it only in the borders of the structure. For this, we should take into account the need of preserving the rectangular connection without excessively incrementing the number of network neurons.

On the other hand, we are working on the combination of a hexagonal connection with the present insertion criterion of AVGSOM and its insertion modification inside the network, which is all under development.

References

[1] Alahakoon, D., Halgamuge, S.K. & Srinivasan, B. *Dynamic Self-Organizing Maps with Controlled*

Growth for Knowledge Discovery. IEEE Transactions On Neural Networks, Vol. 11 (3), pp. 601-614. 2000.

[2] Bauer, H.-U. & Pawelzik, K. *Quantifying the neighborhood preservation of self-organizing feature maps*. IEEE Transactions on Neural Networks, 3(4):570-579. 1992

[3] Fritzke, B. *Growing cell structures: A self organizing network for supervised and un-supervised learning*. Neural networks, Vol. 7, pp. 1441-1460. 1994.

[4] Fritzke, B. *A growing neural Gas Network Learns Topologies* In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, Advances in Neural Information Processing Systems 7, pages 625-632. MIT Press, Cambridge MA, 1995.

[5] Fritzke, B. *Growing Self-organizing Networks – Why?*. In: M. Verleysen (ed.), ESANN'96: European Symposium on Artificial Neural Networks, D-Facto Publishers, Brussels, pp. 61-72. 1996

[6] Hettich, S., Blake, C.L. & Merz, C.J. *UCI Repository of machine learning databases*. University of California, Irvine, Dept. of Information and Computer Sciences. URL http://www.ics.uci.edu/~mlearn/MLRepository.html. 1998.

[7] Hsu, A.L. & Halgamuge, S.K. *Enhanced topology preservation of dynamic self-organizing maps for data visualization*. IEEE, pp. 1786-1791. 2001.

[8] Kohonen T., Hynninen J., Kangas J., Laaksonen J and Torkkola K. Helsinki University of Technology. Laboratory of Computer and Information Science. http://www.cis.hut.fi/research/som_lvq_pack.shtml

[9] Kohonen, T. *The self-organizing map*. Proceedings of the IEEE, Vol. 78 (9), pp. 1464-1480. 1990.

[10] Kohonen, T. *Self-Organizing Maps*. 2nd Edition. Springer. ISSN 0720-678X. 1997.

[11] Martinetz, T. M. & Schulten, K. J. *A "neural-gas" network learns topologies*. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, Artificial Neural Networks, pages 397-402. North-Holland, Amsterdam. 1991

[12] Martinetz, T. M. *Competitive Hebbian learning rule forms perfectly topology preserving maps*. In *ICANN'93: International Conference on Artificial Neural Networks*, pages 427-434, Amsterdam, 1993. Springer.

[13] Polani, D. *Organization measures for self-organizing maps.* Proceedings of WSOM'97, Workshop on Self-Organizing Maps, Espoo, Finland, June 4-6. 1997.

[14] Villmann, Th., Der, R. & Martinetz, T. *A new quantitative measure of topology preservation in Kohonen's feature maps*. Neural Networks. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on , Volume: 2 , 27 June-2 July 1994 Pages:645 - 648 vol.2. 1994.

[15] Villmann, T., Der, R., Herrmann, M. & Martinetz, M. *Topology preservation in Self-Organizing Feature Maps: Exact Definition and Measurement*. IEEE Transactions on Neural Networks, Vol. 8 (2), pp. 256-266. 1997.