GENERACIÓN AUTOMÁTICA DE REDES NEURONALES CON AJUSTE DE PARÁMETROS BASADO EN ALGORITMOS GENÉTICOS

TESIS DE GRADO EN INGENIERIA INFORMATICA

FACULTAD DE INGENIERIA
UNIVERSIDAD DE BUENOS AIRES

TESISTA: **Sr. Abel FISZELEW**

DIRECTOR: **Prof. Dr. Ramón GARCIA-MARTINEZ**

Laboratorio de Sistemas Inteligentes

FEBRERO 2002

GENERACIÓN AUTOMÁTICA DE REDES NEURONALES CON AJUSTE DE PARÁMETROS BASADO EN ALGORITMOS GENÉTICOS

TESIS DE GRADO EN INGENIERIA INFORMATICA

Laboratorio de Sistemas Inteligentes

FACULTAD DE INGENIERIA
UNIVERSIDAD DE BUENOS AIRES

Sr. Abel Fiszelew — **Tesista**

Dr. Ramón García Martínez — **Director**

FEBRERO 2002

Resumen

Este trabajo trata con métodos para encontrar arquitecturas óptimas de red neuronal para aprender problemas particulares. Se utiliza un algoritmo genético para encontrar las arquitecturas adecuadas específicas de un dominio, este algoritmo evolutivo emplea codificación directa y usa el error de la red entrenada como medida de desempeño para guiar la evolución. El entrenamiento de la red se lleva a cabo mediante el algoritmo de retropropagación o back-propagation; se aplican técnicas como la repetición del entrenamiento, la detención temprana y la regularización de la complejidad para mejorar los resultados del proceso evolutivo. El criterio de evaluación se basa en las habilidades de aprendizaje y la precisión para la generalización de las arquitecturas generadas. Los resultados también se comparan con arquitecturas halladas por otros métodos.

Palabras clave: computación evolutiva, redes neuronales, algoritmos genéticos, métodos de codificación.

Abstract

This work deals with methods for finding optimal neural network architectures to learn particular problems. A genetic algorithm is used to discover suitable domain specific architectures; this evolutionary algorithm applies direct codification and uses the error from the trained network as a performance measure to guide the evolution. The network training is accomplished by the back-propagation algorithm; techniques such as training repetition, early stopping and complex regulation are employed to improve the evolutionary process results. The evaluation criteria are based on learning skills and classification accuracy of generated architectures. The results are also compared to those found by other methods.

Keywords: evolutionary computation, neural networks, genetic algorithms, codification methods.

Índice de contenidos

1. Introducción

La inteligencia artificial ha sido definida como la forma de diseñar procesos que exhiben características que comúnmente se asocian con el comportamiento humano inteligente [García Martínez, 1997, García-Martínez & Borrajo, 2000]. Sus enfoques abordan el modelado, con base en distintas arquitecturas, de distintos procesos propios del pensamiento humano tales como: la toma de decisiones, el razonamiento o el aprendizaje. Una de las arquitecturas que ha surgido para emular el comportamiento del aprendizaje es la red neuronal, que toma como modelo el cerebro humano [Rich y Knight, 1991].

Las redes neuronales artificiales ofrecen un paradigma atractivo para el diseño y el análisis de sistemas adaptativos inteligentes para un amplio rango de aplicaciones en inteligencia artificial por muchas razones incluyendo: flexibilidad para la adaptación y el aprendizaje (mediante la modificación de las estructuras computacionales empleadas), robustez ante la presencia de ruido (datos erróneos o incompletos), habilidad para generalizar, capacidad de recuperación ante fallas, potencial para su computación masiva paralela, y semejanza (si bien superficial) con las redes neuronales biológicas [Hinton, 1989; Hertz, Krogh y Palmer, 1991].

A pesar de la gran actividad e investigación en esta área durante los últimos años, que ha llevado al descubrimiento de varios resultados teóricos y empíricos significativos, el diseño de las redes neuronales artificiales para aplicaciones específicas bajo un conjunto dado de restricciones de diseño (por ejemplo, aquellas dictadas por la tecnología) es, por mucho, un proceso de prueba y error, dependiendo principalmente de la experiencia previa con aplicaciones similares [Dow y Siestma, 1991]. La performance (y el costo) de una red neuronal sobre problemas particulares es críticamente dependiente, entre otras cosas, de la elección de los elementos de procesamiento (neuronas), la arquitectura de la red y el algoritmo de aprendizaje utilizado. Por ejemplo, muchos de los algoritmos de aprendizaje utilizados en las redes neuronales esencialmente buscan un ajuste adecuado de los parámetros modificables (también llamados pesos) dentro de una topología de red especificada a priori, bajo la guía de muestras de entrada (ejemplos de entrenamiento) provenientes del ambiente de la tarea. Claramente, para que este enfoque sea exitoso, el ajuste deseado de parámetros debe de hecho existir en el espacio donde se busca (el cual a su vez está restringido por la elección de la topología de red) y el algoritmo de búsqueda usado debe ser capaz de encontrarlo [Honavar y Uhr, 1993]. Aún cuando se pueda encontrar un ajuste adecuado de parámetros usando tal enfoque, la habilidad de la red resultante para generalizar sobre datos no vistos durante el aprendizaje, o el costo de usar la red (medido por su tamaño, consumo de energía, implementación en hardware, etc.) pueden estar lejos del óptimo. Estos factores tornan difícil al proceso de diseño de las redes neuronales. Adicionalmente, la falta de principios de diseño constituye un obstáculo de importancia en el desarrollo de sistemas de redes neuronales a gran escala para una amplia variedad de problemas prácticos. Por consiguiente, son de gran interés las técnicas para automatizar el diseño de arquitecturas neuronales para clases particulares de problemas bajo distintas restricciones de diseño y performance.

Este trabajo se centra en el desarrollo de métodos para el diseño *evolutivo* de arquitecturas de redes neuronales artificiales. Las redes neuronales son vistas comúnmente como un método de implementar mapeos no-lineales complejos (funciones) usando unidades elementales simples que se conectan entre sí mediante conexiones con pesos adaptables [Yao, 1999, Yao y Liu 1998]. Nos concentraremos en la optimización de las estructuras de conexión de las redes.

Los algoritmos evolutivos, inspirados en los principios de la evolución biológica, son otro paradigma en inteligencia artificial que ha recibido mucha atención últimamente. Estos algoritmos, que emplean modelos idealizados del código genético, la recombinación de información genética, las mutaciones y la selección, han producido formas muy genéricas y robustas de encontrar soluciones para problemas de búsqueda computacionalmente difíciles. Uno de esos problemas es la adaptación (o entrenamiento) de las arquitecturas y parámetros de las redes neuronales.

Los algoritmos genéticos, que constituyen una de las formas más importantes de los algoritmos evolutivos, son herramientas de optimización muy poderosas [Goldberg, 1985a, 1985b]. Al igual que las redes neuronales, los algoritmos genéticos están inspirados en la biología: se basan en la teoría de la evolución genética y en el concepto de la supervivencia del más apto [Holland, 1975; 1980; 1986; Holland *et al*., 1986]. Entre las características más importantes se destacan su naturaleza estocástica, su capacidad de considerar simultáneamente una población de soluciones, y su adaptabilidad ante un rango amplio de problemas [Wolpert y Macready, 1995].

El entrenamiento de redes neuronales usando algoritmos evolutivos ha sido un tema de interés durante los últimos años, desde distintos puntos de vista. Uno de éstos es el de la inteligencia artificial, donde queremos dejar que las computadoras resuelvan problemas y aprendan cosas por sí mismas sin involucrar a un programador humano. Otro campo cercano, el reconocimiento de patrones estadístico, le ha dado al aprendizaje neuronal muchas herramientas matemáticas fuertemente conceptuales para el análisis de diferentes enfoques. Aunque se han desarrollado métodos bastante eficientes para el entrenamiento de los pesos de las conexiones, no existe un método definitivo para determinar las arquitecturas neuronales más apropiadas para problemas particulares.

Los métodos en que estamos interesados son aquellos que se utilizan específicamente en el diseño arquitectónico de redes. Queremos hacer el entrenamiento de los pesos de la red mediante métodos de aprendizaje neuronal tradicional (no evolutivo), ya que han demostrado ser muy eficientes para ese propósito. El algoritmo evolutivo utiliza el error de la red entrenada como medida de desempeño para guiar la evolución.

El objetivo de este trabajo es combinar dos técnicas de inteligencia artificial, los algoritmos genéticos y las redes neuronales, para crear un sistema híbrido, donde usamos la primera para mejorar la segunda. El algoritmo genético se aplica a la estructura de una red neuronal con el propósito de encontrar las topologías óptimas específicas del dominio que faciliten tanto el aprendizaje como la generalización.

A continuación se presenta la ubicación de este trabajo dentro de la inteligencia artificial en la sección 1.1, y el plan general del trabajo en la sección 1.2.

1.1. Ubicación de este trabajo dentro de la inteligencia artificial

La inteligencia artificial, en sus intentos de diseñar procesos que exhiben características que se asocian con el comportamiento humano inteligente, aborda diferentes enfoques, tales como:

- ***La toma de decisiones***
- ***El razonamiento***
- ***El aprendizaje***

Desde el punto de vista del propósito primario, los métodos de aprendizaje se pueden clasificar en *sintéticos* y *analíticos*. En los primeros, se intenta crear conocimiento nuevo, mientras que en los segundos se quiere transformar u organizar el conocimiento a priori en uno mejor de acuerdo a algún objetivo.

Si la entrada a un método sintético consiste en ejemplos clasificados por una fuente independiente de conocimiento – por ejemplo, un maestro, un experto, un modelo de simulación, etc. – entonces tenemos *aprendizaje a partir de ejemplos.* Cuando las entradas son hechos que necesitan ser clasificados u organizados por el aprendizaje mismo dentro de una estructura de conocimiento, entonces tenemos *aprendizaje por observación*. Un ejemplo de esto último es el clustering conceptual.

Los métodos conexionistas, los algoritmos genéticos y el descubrimiento cualitativo han sido clasificados como *métodos inductivos empíricos*. Esto es porque recaen en cantidades relativamente pequeñas de conocimiento de fondo (background), y su tipo de inferencia principal es inductivo.

Este trabajo se centra en el desarrollo de métodos para el diseño *evolutivo* de arquitecturas de redes neuronales artificiales con aprendizaje supervisado. Por lo tanto podemos ubicar a este trabajo, dentro del contexto de la inteligencia artificial, como se muestra en la figura 1.1 planteada por [Michalski, Carbonell y Mitchell, 1986].

Las categorías presentadas en esta figura no tienen bordes precisamente delineados, sino deben verse como tendencias centrales que pueden transformarse unas en otras continuamente.

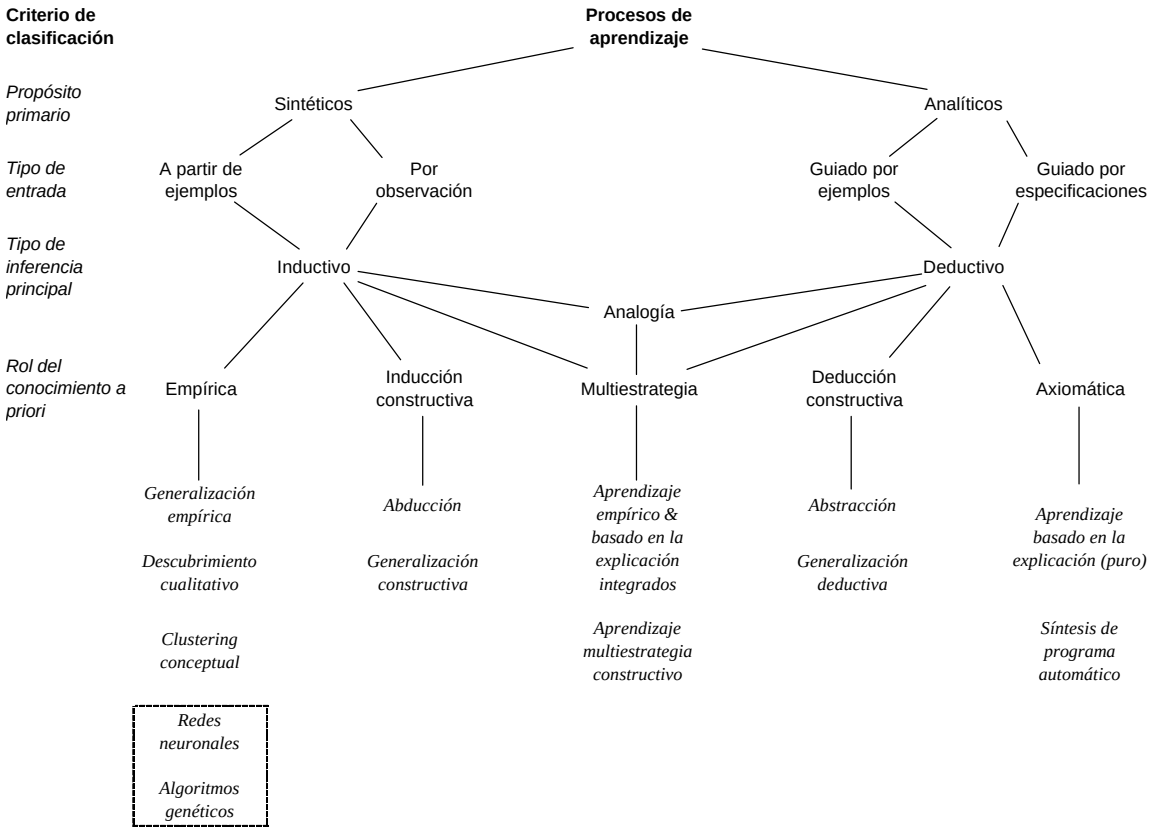


Figura 1.1 Ubicación de este trabajo dentro de la inteligencia artificial.

1.2. Plan general de este trabajo

El resto de este trabajo se dispone de la siguiente manera:

- Los capítulos 2 al 5 establecen el estado de la cuestión sobre la aplicación de los algoritmos genéticos al diseño de redes neuronales.
    - El capítulo 2 hace una introducción a las redes neuronales, sus arquitecturas y sus métodos de aprendizaje.
    - El capítulo 3 desarrolla el algoritmo de aprendizaje usado para entrenar a las redes neuronales, y distintos métodos para optimizarlo.
    - El capítulo 4 resume el funcionamiento del algoritmo genético y las distintas variantes para su implementación.
    - El capítulo 5 describe los aspectos fundamentales del diseño evolutivo de las redes neuronales, y presenta algunos enfoques representativos realizados sobre el tema.

- El capítulo 6 identifica las dificultades encontradas en las implementaciones actuales e introduce los problemas que serán tratados en este trabajo.

- El capítulo 7 presenta la solución propuesta, explicando la manera en que se intenta abordar los problemas introducidos en el capítulo 6. Se determina entonces cómo serán evaluadas las redes neuronales resultantes del proceso evolutivo en cuanto a sus capacidades de aprendizaje y generalización.

- El capítulo 8 brinda una visión del diseño experimental, y muestra gráficamente los resultados obtenidos de los experimentos llevados a cabo, realizando un análisis detallado de los mismos.

- El capítulo 9 toma las cuestiones planteadas en el capítulo 6 y sumariza las respuestas a las mismas que aparecen a lo largo de este trabajo. Contiene las conclusiones, la identificación de las limitaciones del trabajo realizado y el planteo de futuras líneas de investigación.

- El apéndice A realiza una descripción de la programación implementada para llevar a cabo la experimentación.

- El apéndice B despliega los datos numéricos de la experimentación.

2. Redes neuronales artificiales

En los últimos años de exploración en inteligencia artificial, los investigadores se han intrigado por las redes neuronales. Como lo implica su nombre, una red neuronal artificial consiste en una red de neuronas artificiales interconectadas. El concepto se basa vagamente en cómo pensamos que funciona el cerebro de un animal. Un cerebro consiste en un sistema de células interconectadas, las cuales son, aparentemente, responsables de los pensamientos, la memoria y la conciencia. Las neuronas se conectan a muchas otras neuronas formando uniones llamadas sinapsis. Las señales electroquímicas se propagan de una neurona a otra a través de estas sinapsis. Las neuronas demuestran plasticidad: una habilidad de cambiar su respuesta a los estímulos en el tiempo, o aprender. En una red neuronal artificial, se imitan estas habilidades por software.

A continuación se presenta el modelo de una neurona en la sección 2.1, las arquitecturas de red neuronal en la sección 2.2 y los distintos métodos de aprendizaje en la sección 3.3.

2.1. Modelo de una neurona

Una ***neurona*** es una unidad de procesamiento de información que es fundamental para la red neuronal [Hilera y Martínez, 1995]. El diagrama en bloques de la figura 2.1 muestra el ***modelo*** de una neurona, el cual forma las bases para diseñar redes neuronales artificiales. Aquí identificamos tres elementos básicos del modelo neuronal:

a) Un conjunto de ***sinapsis*** o ***conexiones***, cada una caracterizada por un ***peso*** propio. Específicamente, una señal x_j a la entrada de la sinapsis j conectada a la neurona k se multiplica por el peso sináptico w_{kj}. Es importante notar la manera en que se escriben los subíndices del peso sináptico w_{kj}. El primer subíndice se refiere a la neurona en cuestión y el segundo subíndice se refiere a la neurona de donde proviene la sinapsis correspondiente. A diferencia de una sinapsis en el cerebro, el peso sináptico de una red neuronal puede caer en un rango que incluye valores tanto positivos como negativos.

b) Un ***sumador*** para sumar las señales de entrada, ponderadas con los pesos respectivos de las sinapsis de la neurona. Las operaciones descriptas aquí constituyen un ***combinador lineal***.

c) Una ***función de activación*** para limitar la amplitud de la salida de la neurona. Generalmente, el rango de amplitud de salida de una neurona se normaliza a un intervalo cerrado unitario [0, 1] o alternativamente [-1, 1]. Usualmente la función de activación es no lineal, tal como una función escalón, sigmoidal o gaussiana.

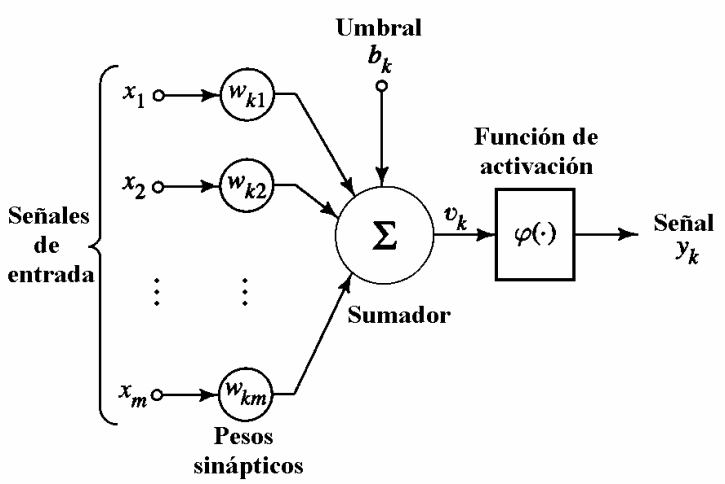


Figura 2.1 Modelo no lineal de una neurona.

El modelo neuronal de la figura 2.1 también incluye un ***umbral*** aplicado externamente, denotado por b_k. El umbral b_k tiene el efecto de subir o bajar la entrada neta de la función de activación, dependiendo de si es positivo o negativo, respectivamente.

En términos matemáticos, podemos describir a una neurona k al escribir las siguientes ecuaciones:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j$$

$$y_k = j\left(u_k + b_k\right)$$

donde $x_1, x_2, ..., x_m$ son las señales de entrada; $w_{k1}, w_{k2}, ..., w_{km}$ son los pesos sinápticos de la neurona k; u_k es la salida del combinador lineal debido a las señales de entrada; b_k es el umbral; $j(\cdot)$ es la función de activación; y por último y_k es la señal de salida de la neurona. El uso del umbral b_k tiene el efecto de aplicar una transformación a la salida u_k del combinador lineal en el modelo de la figura 2.1, como se muestra en la siguiente ecuación:

$$v_k = u_k + b_k$$

En particular, dependiendo de si el umbral b_k es positivo o negativo, la relación entre el ***campo local inducido*** v_k de la neurona k y la salida del combinador lineal u_k se modifica de la manera ilustrada en la figura 2.2, de aquí el nombre de "campo local inducido". Notar que como resultado de esta transformación, el gráfico de v_k versus u_k ya no pasa por el origen.

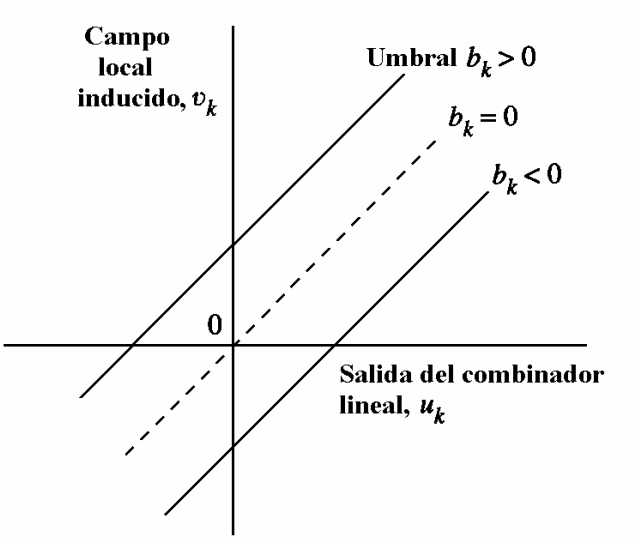


Figura 2.2 Transformación producida la presencia del umbral

El umbral b_k es un parámetro externo de la neurona artificial k. Se puede expresar su presencia al formular las ecuaciones anteriores en forma equivalente:

$$v_k = \sum_{j=0}^{m} w_{kj} x_j$$

$$y_k = j\left(v_k\right)$$

En la sumatoria hemos agregado una nueva sinapsis. Su entrada es

$$x_0 = +1$$

y su peso es

$$w_{k0} = b_k$$

Podemos entonces reformular el modelo de la neurona k como en la figura 2.3. En esta figura, el efecto del umbral se lleva a cabo al: (1) añadir una nueva señal de entrada fija en +1, y (2) añadir un nuevo peso sináptico igual al umbral b_k. Aunque los modelos de las figuras 2.1 y 2.3 parecen diferentes, son matemáticamente equivalentes.

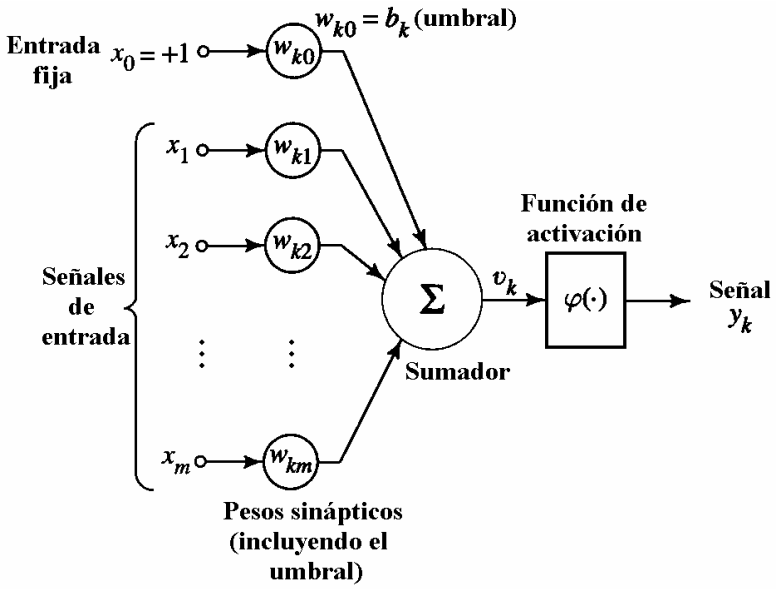


Figura 2.3 Otro modelo no lineal de una neurona.

El diagrama en bloques de la figura 2.1 o de la figura 2.3 provee una descripción funcional de los elementos varios que constituyen el modelo de una neurona artificial. Podemos simplificar la apariencia del modelo usando grafos de flujo de señal sin sacrificar ninguno de los detalles funcionales del modelo. Un ***grafo de flujo de señal*** es una red de vínculos dirigidos (ramas) que se conectan en ciertos puntos llamados nodos. Un nodo *j* típico tiene un nodo de señal asociado x_j. Un vínculo dirigido típico se origina en el nodo *j* y termina en el nodo *k*; tiene una función de transferencia asociada que especifica la manera en la cual la señal y_k en el nodo *k* depende de la señal x_j en el nodo *j*. Por ejemplo, podemos construir el grafo de flujo de señal de la figura 2.4 correspondiente al diagrama en bloques de la figura 2.3.

Basándonos en el grafo de flujo de señal de la figura 2.4 como modelo de neurona, podemos ahora escribir la siguiente definición matemática de una red neuronal:

1. Cada neurona se representa por un conjunto de vínculos sinápticos lineales, un umbral aplicado externamente, y un posible vínculo de activación no lineal. El umbral se representa por un vínculo sináptico conectado a una entrada fija en +1.

2. Los vínculos sinápticos de una neurona ponderan sus señales de entrada respectivas.

3. La suma ponderada de los vínculos sinápticos define al campo local inducido de la neurona en cuestión.

4. El vínculo de activación limita al campo local inducido de la neurona para producir una salida.

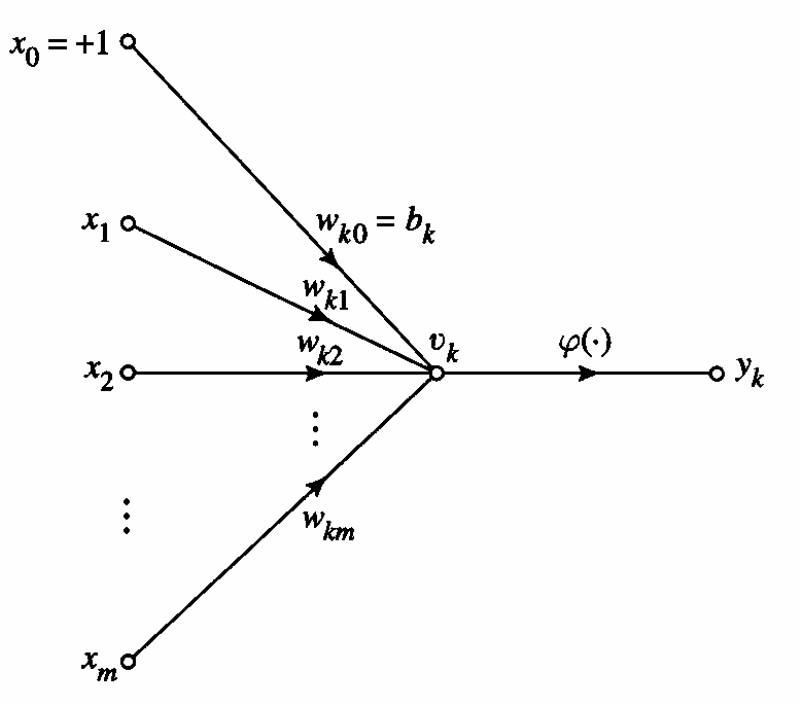


Figura 2.4 Grafo de flujo de señal de una neurona.

2.2. Arquitecturas

Luego de haber analizado a las unidades elementales, la cuestión que suerge es cómo deberían ser conectadas entre sí. El modo en que se conecta una red neuronal se denomina ***topología*** o ***arquitectura***. Las topologías de red usadas en este trabajo son las redes con conexión hacia delante o ***redes feedforward***. Estas redes son una generalización de los perceptrones multicapa (MLPs), que constituyen la categoría de topologías más utilizada en computación neuronal. El Perceptrón de una capa (SLP), una de las topologías de red más antiguas, consiste en una ***capa*** de unidades computacionales; las unidades de entrada no desarrollan ningún cálculo (ver figura 2.5a). La capa de entrada se conecta completamente con la capa de salida, es decir, cada unidad de entrada está conectada a cada unidad de salida.

El ***Perceptrón multicapa*** (MLP, ver figura 2.5b) tiene capas adicionales, llamadas capas ocultas. Aunque se dice que la mayoría de los algoritmos de entrenamiento se aplican a los MLPs, en realidad pueden entrenar a cualquier red que tenga una topología feedforward (ver figura 2.5c). Esta generalización a partir de los MLPs es muy útil para las redes neuronales evolutivas, como se podrá ver más adelante. Una red es feedforward si existe un método que numere todos los nodos en la red de forma tal que no exista ninguna conexión desde un nodo con número grande hacia un nodo con un número más pequeño. Todas las conexiones son desde nodos con números pequeños hacia nodos con números más grandes.

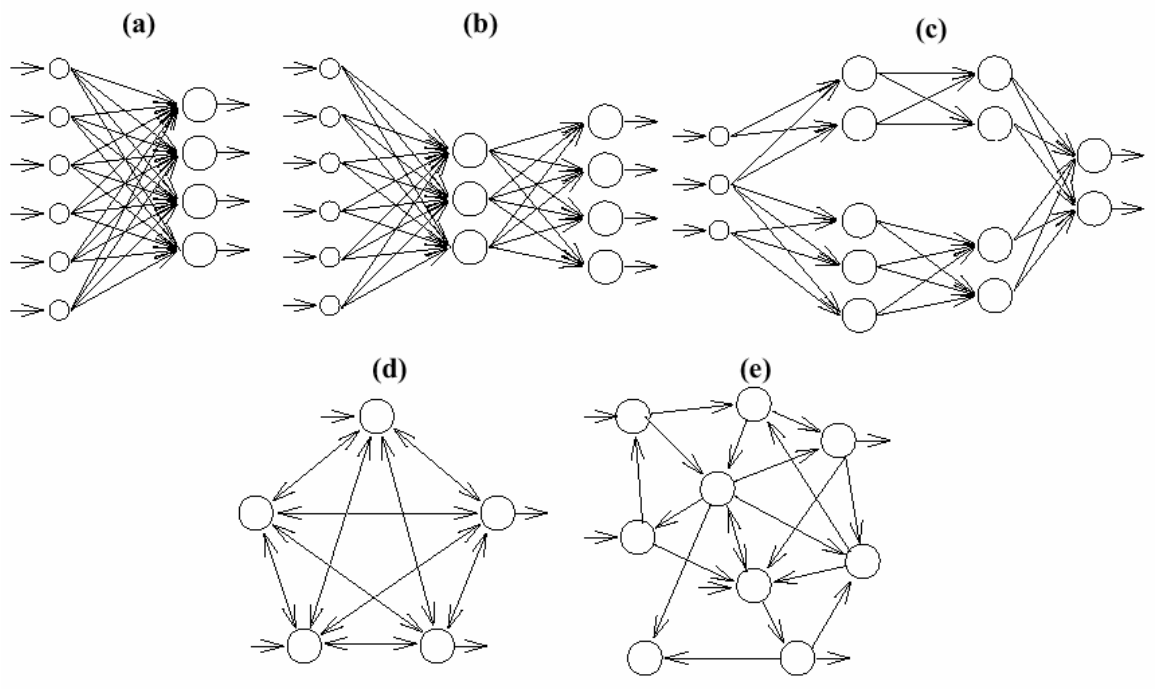


Figura 2.5 Algunas topologías de red. (**a**) Un Perceptrón de una capa (SLP) conectado completamente. (**b**) Un Perceptrón multicapa (MLP) conectado completamente. (**c**) Un MLP modular. (**d**) Una red recurrente conectada completamente. (**e**) Una red recurrente conectada parcialmente.

Si la red no cumple el criterio recién mencionado, tiene algunas conexiones hacia atrás o ***feedback***. Esta clase de topologías se denomina ***recurrente*** (ver figura 2.5d-e), y los algoritmos de entrenamiento más comunes que funcionan con redes feedforward no lo hacen con redes recurrentes sin previas modificaciones.

2.3. Aprendizaje

El aprendizaje en una red neuronal se lleva a cabo típicamente utilizando ejemplos. El entrenamiento se logra al ajustar los pesos de las conexiones iterativamente de manera tal que las redes entrenadas puedan desarrollar ciertas tareas. El aprendizaje en redes neuronales se puede dividir a grandes rasgos en *supervisado*, *no supervisado* y *por refuerzo* [Hertz, 1991].

a) El aprendizaje ***supervisado*** se basa en la comparación directa entre la salida actual de una red neuronal y la salida deseada correcta, también conocida como la salida o dato objetivo. Se formula a menudo como la minimización de una función de error tal como el error cuadrático medio total entre la salida actual y la salida deseada sumada sobre todos los datos disponibles.

b) El aprendizaje ***por refuerzo*** es un caso especial del aprendizaje supervisado donde la salida deseada exacta es desconocida. Se basa sólo en la información de si la salida actual es o no correcta.

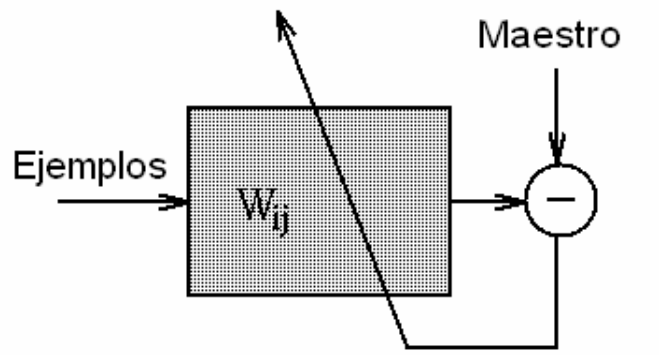


Figura 2.6 Diagrama que representa el aprendizaje supervisado.

c) El aprendizaje **no supervisado** está basado solamente en las correlaciones entre los datos de entrada. No hay información disponible sobre la"salida correcta" para el aprendizaje.

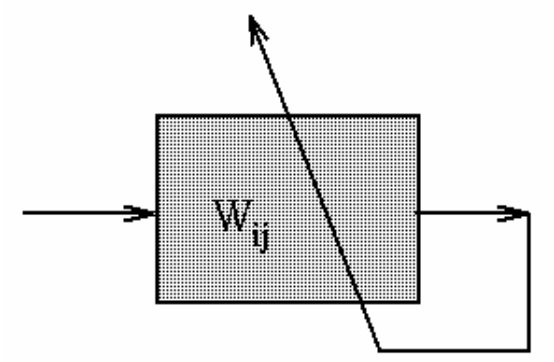


Figura 2.7 Diagrama que representa el aprendizaje no supervisado.

Existen dos fases en toda aplicación de las redes neuronales: la *fase de entrenamiento (o aprendizaje)* y la *fase de prueba*.

a) ***Fase de entrenamiento***: Se usa un conjunto de datos o patrones de entrenamiento para determinar los pesos (parámetros de diseño) que definen el modelo neuronal.

b) ***Fase de prueba o funcionamiento directo***: Una vez entrenado este modelo, se procesan los patrones de prueba que constituyen la entrada habitual de la red, analizándose de esta manera las prestaciones definitivas de la red.

Las aplicaciones del mundo real deben acometer dos tipos diferentes de requisitos en el procesamiento. En un caso, se requiere la prueba en tiempo real pero el entrenamiento se puede realizar "fuera de línea". En otras ocasiones, se requieren ambos procesos, el de prueba y el de entrenamiento, en tiempo real. Estos dos requisitos implican velocidades de proceso muy diferentes, que afectan a los algoritmos y hardware usados.

En este trabajo implementamos un aprendizaje supervisado con su fase de entrenamiento separada de la fase de prueba. La fase de entrenamiento constituye una parte fundamental del proceso evolutivo: el algoritmo genético utilizará los resultados del entrenamiento de las distintas redes candidatas para poder determinar cuáles son las más aptas.

3. Entrenamiento de la red neuronal

Para entrenar una red neuronal para aproximar una función deseada, se utiliza un ***algoritmo de aprendizaje***. El algoritmo de aprendizaje ajusta automáticamente los pesos de una red neuronal de manera de mejorar su habilidad para entregar una salida deseada a partir de una entrada dada. En el aprendizaje supervisado se requiere tener disponibles para el algoritmo al conjunto predefinido de entradas y las salidas deseadas correspondientes. Este conjunto de ejemplos se denomina ***set de entrenamiento***. Un algoritmo de aprendizaje entrena una red al examinar repetidamente cómo responde la red a sus datos de entrenamiento y determinando cómo debería ajustar sus pesos para mejorar la salida para cada ejemplo.

A través del uso de un algoritmo de aprendizaje y un set de entrenamiento adecuado, una red neuronal con la complejidad suficiente puede ser entrenada para aproximar cualquier función. Por ejemplo, la entrada de un set de entrenamiento podría consistir de 100.000 letras escritas a mano, mientras que la salida deseada para cada ejemplo de entrenamiento podría ser la letra que realmente se quiso escribir.

Uno de tales los algoritmos de entrenamiento se denomina "***back-propagation***" (retropropagación), utilizado para entrenar a las redes ***feedforward multicapa***, una clase importante de redes neuronales. Ya que es una parte importante de este trabajo, el algoritmo back-propagation se verá con más detalle a continuación.

En primer lugar, se brinda una introducción en la sección 3.1. Luego se presentan algunos preliminares en 3.2, el algoritmo back-propagation en 3.3, un resumen del mismo en 3.4, un conjunto de heurísticas para que este algoritmo se desempeñe mejor en 3.5, y algunas técnicas que afectan la generalización de las redes entrenadas en 3.6.

3.1. Introducción

Las redes feedforward multicapa consisten en un conjunto de unidades sensoriales (nodos fuente) que constituyen la ***capa de entrada***, una o más ***capas ocultas*** de nodos computacionales, y una ***capa de salida***. Las señales de entrada se propagan a través de la red en dirección hacia adelante, capa por capa. Estas redes neuronales son llamadas comúnmente "***perceptrones multicapa***" (MLPs), y representan una generalización del Perceptrón de una capa propuesto por [Rosemblatt, 1958].

Los perceptrones multicapa se han aplicados exitosamente para resolver problemas difíciles diversos al entrenarlos de una manera supervisada con el algoritmo "error back-propagation". Este algoritmo se basa en la regla de aprendizaje de "corrección del error". Como tal, debe verse como una generalización de un algoritmo de filtro adaptativo muy utilizado, el LMS (Least Mean Error).

Básicamente, el aprendizaje error back-propagation consiste en dos pasadas a través de las diferentes capas de la red: una pasada hacia adelante y una hacia atrás. En la ***pasada hacia adelante***, se aplica un patrón de actividad (vector de entrada) a los nodos sensoriales de la red, y su efecto se propaga a través de la red capa por capa. Finalmente, se produce un conjunto de salidas como respuesta real de la red. Durante la pasada hacia adelante los pesos de las conexiones de la red están todas fijas. Durante la ***pasada hacia atrás***, por el contrario, se ajustan los pesos de acuerdo con la regla de corrección del error. Específicamente, la respuesta real de la red se resta de una respuesta deseada (objetivo) para producir una señal de error. La señal de error entonces se propaga hacia atrás a través de la red, en contra de la dirección de las conexiones - de aquí el nombre "error back-propagation". Los pesos de las conexiones se ajustan para hacer que la respuesta real de la red se "mueva" más cerca de la respuesta deseada en un sentido estadístico.

3.2. Algunos preliminares

La figura 3.1 muestra el grafo arquitectónico de un Perceptrón multicapa con dos capas ocultas y una capa de salida. Para poder describir al Perceptrón multicapa en su forma general, la red aquí mostrada está ***completamente conectada***. Esto significa que una neurona de cualquier capa está conectada a todas las neuronas de la capa previa. Las redes neuronales producidas por la evolución no deben cumplir necesariamente con este requisito (en la mayoría de los casos no lo harán). El flujo de la señal a través de la red progresa en dirección hacia delante, de izquierda a derecha y capa por capa.

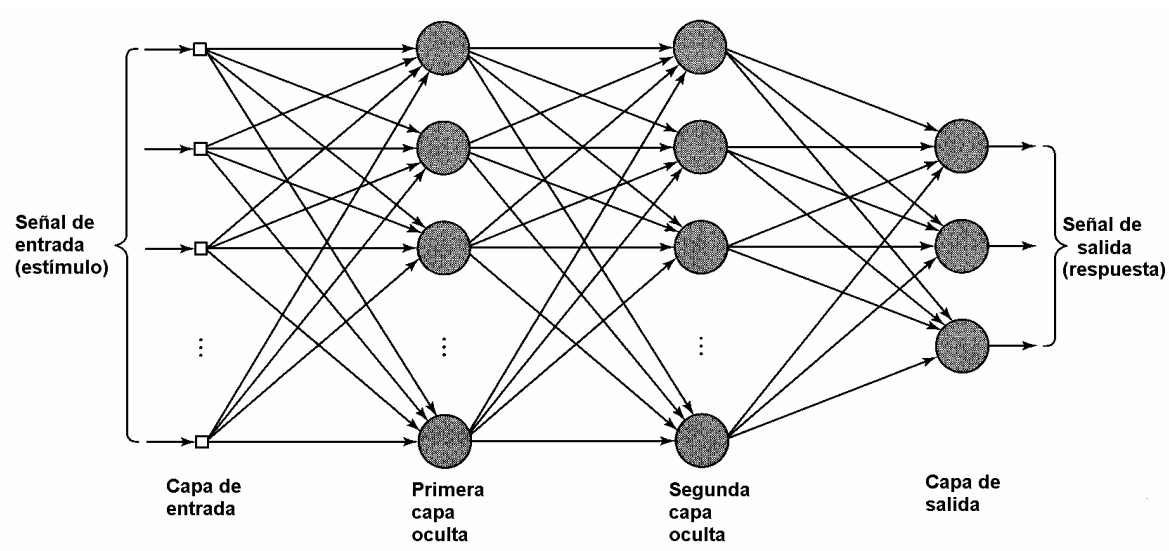


Figura 3.1 Grafo arquitectónico de un Perceptrón multicapa con dos capas ocultas

La figura 3.2 muestra una porción del Perceptrón multicapa. Dos clases de señales se identifican en esta red [Parker, 1987]:

a) ***Señales funcionales***: Una señal funcional es una señal de entrada (estímulo) que entra en un extremo de la red, se propaga hacia delante (neurona por neurona) a través de la red, y emerge en el otro extremo de la red como señal de salida. Se la llama "señal funcional" por dos razones. Primero, se presume que cumple una función útil a la salida de la red. Segundo, en cada neurona de la red a través de la cual pasa una señal funcional, se calcula la señal como una función de las entradas y los pesos asociados aplicados a esa neurona. A la señal funcional también se la denomina señal de entrada.

b) ***Señales de error***: Una señal de error se origina en una neurona de salida de la red, y se propaga hacia atrás, capa por capa, a través de la red. Se la llama "señal de error" porque su computación en cada neurona de la red involucra una función dependiente del error en una forma u otra.

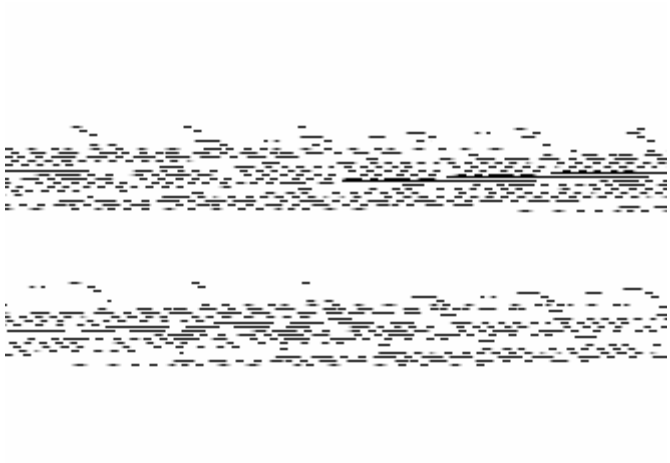


Figura 3.2 Ilustración de la dirección de dos flujos de señal básicos en un Perceptrón multicapa: propagación hacia delante de señales funcionales y propagación hacia atrás de señales de error.

Las neuronas de salida constituyen la capa de salida de la red. Las neuronas restantes constituyen las capas ocultas de la red. Por lo tanto las capas ocultas no forman parte de la salida o de la entrada de la red – de aquí la denominación de ocultas. La primer capa oculta se alimenta de la capa de entrada hecha de unidades sensoriales (nodos fuente); las salidas resultantes de la primer capa oculta son a su vez aplicadas a la próxima capa oculta, y así sucesivamente por el resto de la red.

Cada neurona oculta o de salida de un Perceptrón multicapa se diseña para desarrollar dos cómputos:

a) El cómputo de la señal funcional que aparece a la salida de una neurona, la cual se expresa como una función continua no-lineal de la señal de entrada y los pesos de las conexiones asociadas a esa neurona.

b) El cómputo de una estimación del vector gradiente (es decir, los gradientes de la superficie de error con respecto a los pesos conectados a las entradas de una neurona), la cual se requiere para la pasada hacia atrás a través de la red.

Para alivianar el peso matemático involucrado en la derivación del algoritmo de back-propagation, primero se presenta un resumen de la notación utilizada en su deducción.

3.2.1. Notación

- Los índices *i, j, k* se refieren a distintas neuronas en la red; con las señales propagándose a través de la red de derecha a izquierda, la neurona *j* se ubica a la derecha de la neurona *i*, y la neurona *k* se ubica a la derecha de la neurona *j* cuando la neurona *j* es una unidad oculta.
- En la iteración (paso de tiempo) *n*, se le presenta a la red el *n*-ésimo patrón de entrenamiento (ejemplo).
- El símbolo $e(n)$ se refiere a la suma de los cuadrados de los errores instantáneos o energía de error en la iteración *n*. El promedio de $e(n)$ sobre todos los valores de *n* (es decir, el set de entrenamiento completo) se representa como la energía de error promedio e_{av}.
- El símbolo $e_j(n)$ se refiere a la señal de error a la salida de la neurona *j* para la iteración *n*.
- El símbolo $d_j(n)$ se refiere a la respuesta deseada para la neurona *j* y se utiliza para computar $e_j(n)$.
- El símbolo $y_j(n)$ se refiere a la señal funcional que aparece a la salida de la neurona *j* en la iteración *n*.
- El símbolo $w_{ji}(n)$ denota el peso que conecta la salida de la neurona *i* a la entrada de la neurona *j* en la iteración *n*. La corrección aplicada a este peso en la iteración *n* se denota por $Dw_{ji}(n)$.
- El campo total inducido (es decir, la suma ponderada de todas las entradas más el umbral) de la neurona *j* en la iteración *n* se denota por $v_j(n)$, constituye la señal aplicada a la función de activación asociada con la neurona *j*.
- La función de activación que describe la relación funcional entrada-salida de la no-linealidad asociada con la neurona *j* se denota como $j_j(\cdot)$.
- El umbral aplicado a la neurona *j* se denota por b_j; su efecto se representa por un peso $w_{j0} = b_j$ conectada a una entrada fija igual a +1.
- El *i*-ésimo elemento del vector de entrada (patrón) se denota por $x_i(n)$.
- El *k*-ésimo elemento del vector de salida (patrón) se denota por $o_k(n)$.
- El parámetro de la tasa de aprendizaje se denota por η.
- El símbolo m_l denota el tamaño (número de neuronas) en la capa *l* del Perceptrón multicapa; $l = 0, 1, ..., L$, donde *L* es la "profundidad" de la red. Así, m_0 denota el tamaño de la capa de entrada, m_1 denota el tamaño de la primer capa oculta, y m_L denota el tamaño de la capa de salida. También se utiliza la notación $m_L = M$.

3.3. El algoritmo back-propagation

La ***señal de error*** a la salida de la neurona *j* en la iteración *n* (la presentación del *n*-ésimo ejemplo de entrenamiento) se define como:

$$e_j(n) = d_j(n) - y_j(n), \qquad \text{neurona } j \text{ es un nodo de salida} \qquad (1)$$

Definimos al valor instantáneo de la energía de error para la neurona *j* como ½ $e_j^2(n)$. Por lo tanto, el valor instantáneo *e(n)* de la ***energía de error total*** se obtiene al sumar ½ $e_j^2(n)$ sobre todas las neuronas en la capa de salida; éstas son las únicas neuronas visibles para las cuales se pueden calcular directamente las señales de error. Escribimos entonces:

$$e(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n) \qquad (2)$$

Donde el conjunto *C* incluye todas las neuronas en la capa de salida de la red. Sea *N* el número total de patrones (ejemplos) contenidos en el set de entrenamiento. La ***energía de error cuadrático promedio*** se obtiene sumando *e(n)* sobre cada *n* y luego normalizando con respecto al tamaño *N* del set, como se muestra en:

$$e_{av}(n) = \frac{1}{N}\sum_{n=1}^{N} e(n) \qquad (3)$$

La energía de error instantánea *e(n)*, y por lo tanto la energía de error promedio e_{av}, es función de todos los parámetros libres (pesos de conexión y umbrales) de la red. Para un set de entrenamiento dado, e_{av} representa la ***función de costo*** como medida de la performance del aprendizaje. El objetivo del proceso de aprendizaje es ajustar los parámetros libres de la red para minimizar e_{av}. Para lograr esta minimización, se utiliza una aproximación similar a la empleada para la derivación del algoritmo LMS. Específicamente, se considera un método simple de entrenamiento en el cual los pesos son actualizados *patrón por patrón* hasta que un ***epoch***, es decir, una presentación completa del set de entrenamiento, ha sido tratado. Los ajustes a los pesos se hacen de acuerdo con los errores respectivos computados para cada patrón presentado a la red. El promedio aritmético de estos cambios de peso individuales producidos por el set de entrenamiento es entonces una *estimación* del cambio verdadero que resultaría de modificar los pesos basándose en la minimización de la función de costo e_{av} sobre el set completo de entrenamiento.

La figura 3.3 muestra la neurona *j* siendo alimentada por un conjunto de señales funcionales producidas por una capa de neuronas a su izquierda. El ***campo local inducido*** $v_j(n)$ producido a la entrada de la función de activación asociada con la neurona *j* es entonces:

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) \cdot y_i(n) \qquad (4)$$

donde *m* es el número total de entradas (excluyendo el umbral) aplicado a la neurona *j*. El peso $w_{j0}(n)$ (correspondiente a la entrada fija $y_0 = +1$) es igual al umbral b_j aplicado a la neurona *j*. De aquí, la ***señal funcional*** $y_j(n)$ que aparece a la salida de la neurona *j* en la iteración *n* es:

$$y_j(n) = j_j(v_j(n)) \qquad (5)$$

En una manera similar al algoritmo LMS, el algoritmo back-propagation aplica una corrección $Dw_{ji}(n)$ *al peso* $w_{ji}(n)$, la cual es proporcional a la derivada parcial $\partial e(n)/\partial w_{ji}(n)$. De acuerdo a la *regla de la cadena*, se puede expresar este ***gradiente*** como:

$$\frac{\partial e(n)}{\partial w_{ji}(n)} = \frac{\partial e(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \qquad (6)$$

La derivada parcial $\partial e(n)/\partial w_{ji}(n)$ representa un ***factor de sensibilidad*** determinando la dirección de la búsqueda, en el espacio de pesos, del peso w_{ji}.

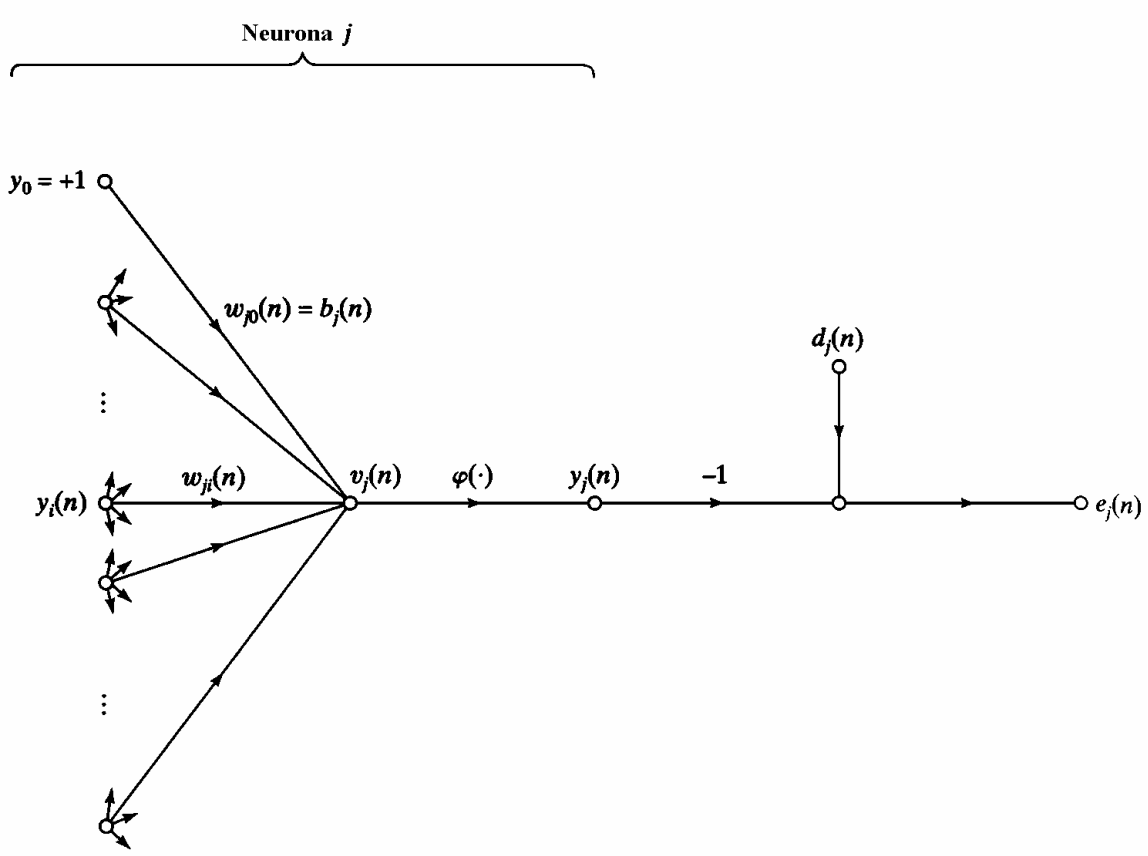


Figura 3.3 Grafo del flujo de señal resaltando los detalles de la neurona de salida *j*.

Al diferenciar ambos lados de la ecuación (1) con respecto a $e_j(n)$ obtenemos:

$$\frac{\partial e(n)}{\partial e_j(n)} = e_j(n) \qquad (7)$$

Al diferenciar ambos lados de la ecuación (1) con respecto a $y_j(n)$ obtenemos:

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \quad (8)$$

Luego, al diferenciar la ecuación (5) con respecto a $v_j(n)$ obtenemos:

$$\frac{\partial y_j(n)}{\partial v_j(n)} = j_j'(v_j(n)) \quad (9)$$

donde el uso de la prima (en el lado derecho) significa diferenciación con respecto al argumento. Finalmente, diferenciando la ecuación (4) con respecto a $w_{ji}(n)$ produce:

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_j(n) \qquad (10)$$

El uso de las ecuaciones (7) a (10) en (6) queda:

$$\frac{\partial e(n)}{\partial w_{ji}(n)} = -e_j(n) j_j'(v_j(n)) y_j(n) \qquad (11)$$

La corrección $Dw_{ji}(n)$ aplicada a $w_{ji}(n)$ está definido por la ***regla delta***:

$$\Delta w_{ji}(n) = -h \frac{\partial e(n)}{\partial w_{ji}(n)} \qquad (12)$$

donde h es el parámetro de la ***tasa de aprendizaje*** del algoritmo back-propagation. El uso del signo menos en la ecuación (12) se coloca por ser un ***gradiente descendente*** en el espacio de pesos (buscando una dirección para el cambio de pesos que reduzca el valor de $e(n)$). En consecuencia, el uso de la ecuación (11) en (12) produce:

$$\Delta w_{ji}(n) = h d_j(n) y_j(n) \qquad (13)$$

donde el ***gradiente local*** $d_j(n)$ se define por:

$$d_j(n) = -\frac{\partial e(n)}{\partial v_j(n)} = \frac{\partial e_j(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial v_j(n)}{\partial v_j(n)} = e_j(n) j_j'(v_j(n)) \qquad (14)$$

El gradiente local indica los cambios requeridos en los pesos de las conexiones. Según la ecuación (14) el gradiente local $d_j(n)$ para la neurona de salida j es igual al producto de la señal de error $e_j(n)$ correspondiente para esa neurona y la derivada $j'_j(v_j(n))$ de la función de activación asociada.

De las ecuaciones (13) y (14) notamos que un factor clave involucrado en el cálculo del ajuste de peso $Dw_{ji}(n)$ es la señal de error $e_j(n)$ a la salida de la neurona j. En este contexto se pueden identificar dos casos distintos, dependiendo de dónde está localizada la neurona j en la red. En el caso 1, la neurona j es un nodo de salida. Este caso es simple de manejar porque a cada nodo de salida de la red se le provee con una respuesta deseada propia, haciendo que el cálculo de la señal de error asociada sea un asunto directo. En el caso 2, la neurona j es un nodo oculto. Aún aunque las neuronas ocultas no son accesibles directamente, comparten la responsabilidad por cualquier error cometido a la salida de la red. La pregunta, sin embargo, es cómo penalizar o recompensar a las neuronas ocultas por su parte de la responsabilidad. El problema de ***asignación de crédito*** es resuelto en forma elegante por el algoritmo back-propagation. En estas situaciones, se puede descomponer al problema de asignación de crédito en dos subproblemas [Sutton, 1984]:

a) *La asignación de crédito para los resultados de las acciones*. A ésto se lo llama *problema de la asignación de crédito temporal*, ya que involucra el instante de tiempo *cuando* realmente se toman las acciones que merecen créditos.

b) *La asignación de crédito para las acciones en decisiones internas*. A ésto se lo llama *problema de la asignación de crédito estructural*, ya que involucra la asignación de crédito a las estructuras internas de acciones generadas por el sistema.

Caso 1: La neurona *j* es un nodo de salida

Cuando la neurona j está localizada en la capa de salida de la red, se le suministra una respuesta deseada propia. Se puede usar la ecuación (1) para computar la señal de error $e_j(n)$ asociada con esa neurona, ver la figura 3.3. Habiendo determinado $e_j(n)$ se hace directo el cómputo del gradiente local $\delta_j(n)$ usando la ecuación (14).

Caso 2: La neurona *j* es un nodo oculto

Cuando la neurona j se localiza en una capa oculta de la red, no está especificada ninguna respuesta para esa neurona. En consecuencia, la señal de error para una neurona oculta debería ser determinada recursivamente en términos de las señales de error de todas las neuronas a las cuales esa neurona oculta está conectada directamente; aquí es donde el desarrollo del algoritmo back-propagation se hace más complicado. Consideremos la situación ilustrada en la figura 3.4, la cual muestra la neurona j como una neurona oculta de la red. De acuerdo a la ecuación (14) podemos redefinir al gradiente local $\delta_j(n)$ para la neurona oculta j como:

$$d_j(n) = -\frac{\partial e(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial e(n)}{\partial y_j(n)} j_j'\left(v_j(n)\right), \qquad \text{la neurona } j \text{ es oculta} \quad (15)$$

donde en el último término utilizamos la ecuación (9). Para calcular la derivada parcial $\partial e(n)/\partial y_j(n)$, se procederá como sigue. De la ecuación (4) vemos que

$$e(n) = \frac{1}{2}\sum_{k \in C} e_k^2(n), \qquad \text{la neurona } k \text{ es un nodo de salida} \qquad (16)$$

la cual es la ecuación (2) con el índice *k* usado en lugar del índice *j*. Hicimos esto para evitar la confusión con el uso del índice *j* que se refiere a una neurona oculta bajo el caso 2. Al diferenciar la ecuación (16) con respecto a la señal funcional $y_j(n)$ obtenemos:

$$\frac{\partial e(n)}{\partial y_j(n)} = \sum_k \frac{\partial e_k(n)}{\partial y_j(n)} \qquad (17)$$

Luego usamos la regla de la cadena para la derivada parcial $\partial e(n)/\partial y_j(n)$, y rescribimos la ecuación (17) en su forma equivalente:

$$\frac{\partial e(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} \qquad (18)$$

Sin embargo, de la ecuación (4), notamos que:

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - j_k(v_k(n)), \qquad \text{la neurona } k \text{ es un nodo de salida} \qquad (19)$$

Entonces:

$$\frac{\partial e(n)}{\partial v_k(n)} = -j_k'(v_k(n)) \qquad (20)$$

También notamos de la figura 3.4 que para la neurona *k* el campo inducido total es:

$$v_k(n) = \sum_{j=0}^{m} w_{kj}(n) y_j(n) \qquad (21)$$

donde *m* es el número total de entradas (excluyendo al umbral) aplicadas a la neurona *k*. Nuevamente, el peso $w_{k0}(n)$ es igual al umbral aplicado a la neurona *k*, y la entrada correspondiente está fija con el valor +1. Diferenciando la ecuación (21) con respecto a $y_j(n)$ produce:

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n) \qquad (22)$$

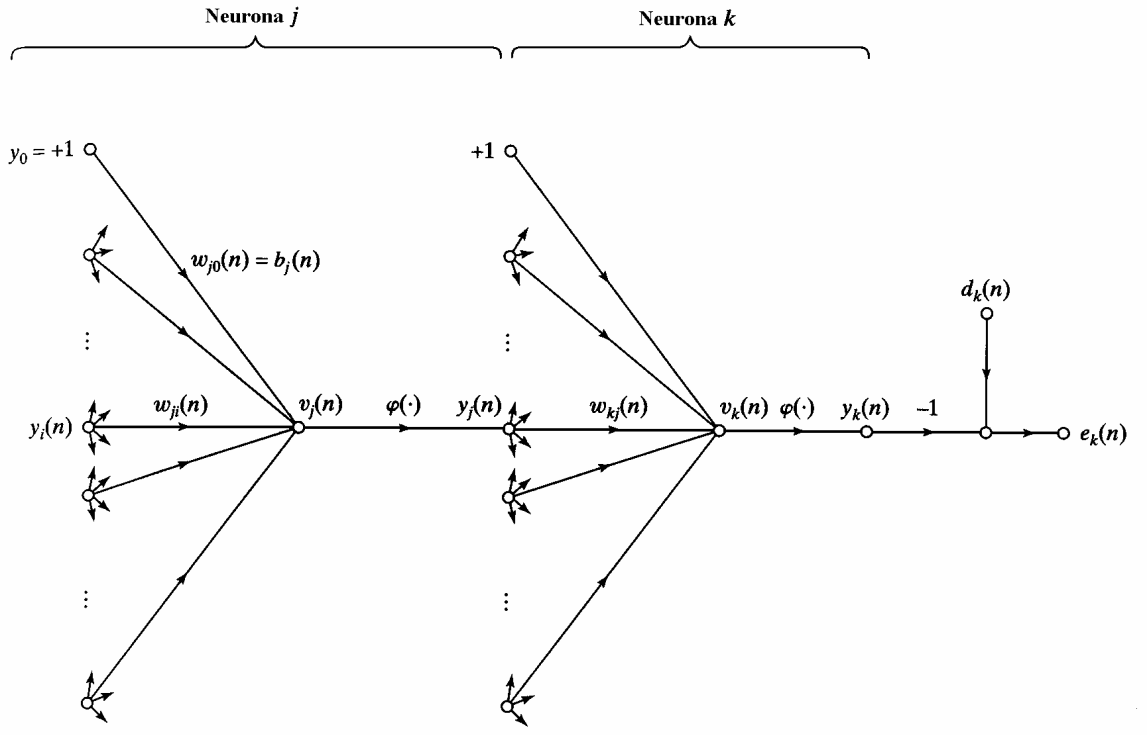


Figura 3.4 Grafo del flujo de señal resaltando los detalles de la neurona de salida *k* conectada a la neurona oculta *j*.

Usando las ecuaciones (20) y (22) en (18) se obtiene la derivada parcial deseada:

$$\frac{\partial e(n)}{\partial y_j(n)} = -\sum_k e_k(n) j_k'(v_k(n)) w_{kj}(n) = -\sum_k d_k(n) w_{kj}(n) \qquad (23)$$

donde en el último término usamos la definición del gradiente local $\delta_k(n)$ dada en la ecuación (14) con el índice *j* sustituido por *k*.

Finalmente, usando las ecuaciones (23) en (15), obtenemos la ***fórmula back-propagation*** para el gradiente local $d_j(n)$ como:

$$dj(n) = j_j'(v_j(n)) \sum_k d_k(n) w_{kj}(n), \qquad \text{la neurona } j \text{ es oculta} \qquad (24)$$

La figura 3.5 muestra la representación del grafo de flujo de señal de la ecuación (24), asumiendo que la capa de salida consiste en m_L neuronas.

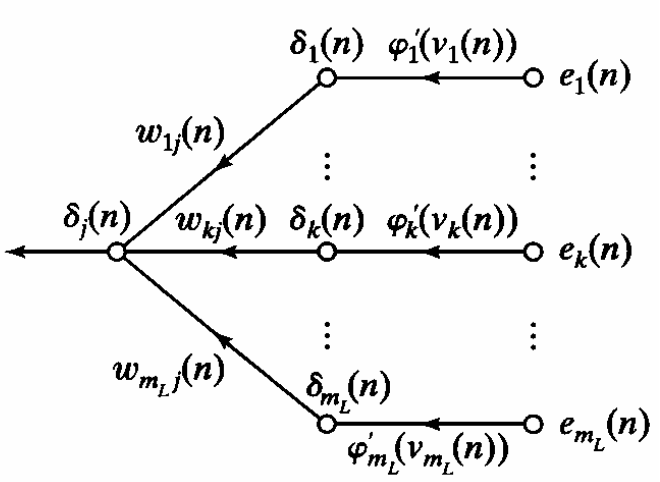


Figura 3.5 Grafo del flujo de señal que detalla la propagación hacia atrás de las señales de error.

El factor $j'_j(v_j(n))$ involucrado en la computación del gradiente local $\delta_j(n)$ en la ecuación (24) depende solamente de la función de activación asociada con la neurona oculta *j*. El factor restante involucrado en este cómputo, la sumatoria sobre *k*, depende de dos conjuntos de términos. El primer conjunto, los $\delta_k(n)$, requiere el conocimiento de las señales de error $e_k(n)$, para todas las neuronas que se ubican en la capa inmediatamente a la derecha de la neurona oculta *j*, y que se conectan directamente con la neurona *j*; ver figura 3.4. El segundo conjunto de términos, los $Dw_{ji}(n)$, consiste en los pesos asociados con esas conexiones.

Ahora resumimos las relaciones que se han derivado para el algoritmo back-propagation. Primero, la corrección $Dw_{ji}(n)$ aplicada al peso que conecta la neurona *i* con la neurona *j* está definida por la regla delta:

$$\begin{pmatrix} \text{corrección} \\ \text{de peso} \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} \text{tasa de} \\ \text{aprendizaje} \\ h \end{pmatrix} \cdot \begin{pmatrix} \text{gradiente} \\ \text{local} \\ d_j(n) \end{pmatrix} \cdot \begin{pmatrix} \text{señal de entrada} \\ \text{de la neurona } j \\ y_i(n) \end{pmatrix} \qquad (25)$$

Segundo, el gradiente local $\delta_j(n)$ depende de si la neurona *j* es un nodo de salida o un nodo oculto:

a) Si la neurona j es un nodo de salida, $d_j(n)$ es igual al producto de la derivada $j'_j(v_j(n))$ y la señal de error $e_j(n)$, ambos asociadas con la neurona *j*, ver la ecuación (14).

b) Si la neurona j es un nodo oculto, $d_j(n)$ es igual al producto de la derivada asociada $j'_j(v_j(n))$ y la sumatoria de los *d*s calculados para las neuronas en la próxima capa oculta o de salida que se conectan a la neurona *j*, ver la ecuación (24).

3.3.1. El Cómputo de Doble Pasada

En la aplicación del algoritmo back-propagation, se distinguen dos pasadas distintas de cómputo. La primera pasada se refiere como la pasada hacia delante, y la segunda como la pasada hacia atrás.

En la ***pasada hacia delante*** los pesos permanecen inalterados por toda la red, y las señales funcionales de la red se computan neurona por neurona. La señal funcional que aparece a la salida de la neurona *j* se calcula como:

$$y_j(n) = j_j(v_j(n)) \qquad (26)$$

donde $v_j(n)$ es el campo local inducido de la neurona *j*, definido por:

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) \cdot y_i(n) \qquad (27)$$

donde *m* es el número total de entradas (excluyendo el umbral) aplicado a la neurona *j*, y $y_i(n)$ es la señal de entrada de la neurona *j* o, equivalentemente, la señal funcional que aparece a la salida de la neurona *i*. Si la neurona *j* está en la primer capa oculta de la red, $m = m_0$ y el índice *i* se refiere a la *i*-ésima entrada de la red, para la cual escribimos:

$$y_i(n) = x_i(n) \qquad (28)$$

donde $x_i(n)$ es el *i*-ésimo elemento del vector de entrada (patrón). Si, por el contrario, la neurona *j* está en la capa de salida de la red, $m = m_L$ y el índice *j* se refiere a la *j*-ésima salida de la red, para la cual escribimos:

$$y_j(n) = o_j(n) \qquad (29)$$

donde $o_j(n)$ es el *j*-ésimo elemento del vector de salida (patrón). Esta salida se compara con la respuesta deseada $d_j(n)$, obteniéndose la señal de error $e_j(n)$ para la *j*-ésima neurona de salida. Así, la fase hacia delante comienza en la primer neurona oculta al presentarle el vector de entrada, y termina en la capa de salida al computar la señal de error para cada neurona de esta capa.

La ***pasada hacia atrás***, por el otro lado, comienza en la capa de salida al pasar las señales de error hacia la izquierda a través de la red, capa por capa, y computando recursivamente el *d* (gradiente local) para cada neurona. Este proceso recursivo permite que los pesos experimenten cambios de acuerdo a la regla delta de la ecuación (25). Para una neurona localizada en la capa de salida, el *d* es simplemente igual a la señal de error multiplicada por la primer derivada de su no-linealidad. Por eso se usa la ecuación (25) para computar los cambios de los pesos de todas las conexiones alimentando a la capa de salida. Dados los *d*s de las neuronas de la capa de salida, luego se usa la ecuación (24) para computar

los *d*s para todas las neuronas en la penúltima capa y por lo tanto los cambios de todas las conexiones que las alimentan. Esta computación recursiva se continúa, capa por capa, al propagar los cambios de todos los pesos de la red.

Notar que para la presentación de cada ejemplo de entrenamiento, el patrón de entrada permanece fijo a lo largo de todo el proceso de ida y vuelta, que comprende a la pasada hacia delante seguida por la pasada hacia atrás.

3.3.2. Función de Activación

El cómputo de *d* para cada neurona del Perceptrón multicapa requiere el conocimiento de la derivada de la función de activación $j_j(\cdot)$ asociada con esa neurona. Para que esta derivada exista, se requiere que la función $j_j(\cdot)$ sea continua. En realidad, la *diferenciabilidad* es el único requerimiento que tiene que satisfacer la función de activación. Un ejemplo de función continua diferenciable y no lineal comúnmente usada en los perceptrones multicapa es la no-linealidad sigmoidal; se describen dos formas:

a) ***Función Logística***: esta forma de no-linealidad sigmoidal en su forma general se define como:

$$j_j(v_j(n)) = \frac{1}{1+\exp(-av_j(n))} \qquad a > 0 \text{ y } -\infty < v_j < \infty \qquad (30)$$

donde $v_j(n)$ es el campo local inducido de la neurona *j*. De acuerdo con esta no-linealidad, la amplitud de la salida cae dentro del rango $0 \le y_j \le 1$. Diferenciando la ecuación (30) con respecto a $v_j(n)$ conseguimos:

$$j_j'(v_j(n)) = \frac{a\exp(-av_j(n))}{[1+\exp(-av_j(n))]^2} \qquad (31)$$

Con $y_j(n) = j_j(v_j(n))$, podemos eliminar el término exponencial $\exp(-av_j(n))$ de la ecuación (31), y entonces expresar la derivada $j'_j(v_j(n))$ como:

$$j_j'(v_j(n)) = ay_j(n)[1-y_j(n)] \qquad (32)$$

Para una neurona *j* ubicada en la capa de salida, $y_j(n) = o_j(n)$. En consecuencia, podemos expresar el gradiente local para la neurona *j* como:

$$\begin{aligned} d_j(n) &= e_j(n)j_j'(v_j(n)) \\ &= a[d_j(n)-o_j(n)]\, o_j(n)[1-o_j(n)] \end{aligned} \qquad \text{la neurona } j \text{ es un nodo de salida} \qquad (33)$$

donde $o_j(n)$ es la señal funcional a la salida de la neurona j, y $d_j(n)$ es la respuesta deseada para ella. Por el otro lado, para una neurona oculta j arbitraria, podemos expresar el gradiente local como:

$$\begin{aligned} d_j(n) &= j_j'(v_j(n))\sum_k d_k(n)w_{kj}(n) \\ &= ay_j(n)[1-y_j(n)]\sum_k d_k(n)w_{kj}(n) \end{aligned} \qquad \text{la neurona } j \text{ es oculta} \qquad (34)$$

Notar de la ecuación (32) que la derivada $j'_j(v_j(n))$ alcanza su valor máximo cuando $y_j(n)$=0.5, y su mínimo valor en $y_j(n)$=0, o $y_j(n)$=1.0. Ya que la medida del cambio en los pesos de la red es proporcional a la derivada $j'_j(v_j(n))$, inferimos que, para una función de activación sigmoidal, los pesos sufren más cambios para aquellas neuronas de la red donde las señales funcionales están en sus rangos medios. De acuerdo a [Rumelhart, 1986], es esta característica del aprendizaje back-propagation que contribuye a su estabilidad como un algoritmo de aprendizaje.

b) ***Función Tangente Hiperbólica***: Otra forma comúnmente usada para la no-linealidad sigmoidal es la función tangente hiperbólica, la cual en su forma más general está definido por:

$$j_j'(v_j(n)) = a\tanh(bv_j(n)) \quad (a, b) > 0 \qquad (35)$$

donde a y b son constantes. En realidad, la función tangente hiperbólica es solo la función logística re-escalada y desplazada de origen. Su derivada con respecto a $v_j(n)$ está dada por:

$$j_j'(v_j(n)) = ab\,\mathrm{sech}^2(bv_j(n)) = ab(1-\tanh^2(bv_j(n))) = \frac{b}{a}[a-y_j(n)]\,[a+y_j(n)] \qquad (36)$$

Para una neurona j localizada en la capa de salida, el gradiente local es:

$$\begin{aligned} d_j(n) &= e_j(n)j_j'(v_j(n)) \\ &= \frac{b}{a}[d_j(n)-o_j(n)]\,[a-o_j(n)]\,[a+o_j(n)] \end{aligned} \qquad \text{la neurona } j \text{ es un nodo de salida} \qquad (37)$$

Para una neurona j localizada en una capa oculta, se tiene:

$$\begin{aligned} d_j(n) &= j_j'(v_j(n))\sum_k d_k(n)w_{kj}(n) \\ &= \frac{b}{a}[a-y_j(n)]\,[a+y_j(n)]\sum_k d_k(n)w_{kj}(n) \end{aligned} \qquad \text{la neurona } j \text{ es oculta} \qquad (38)$$

Usando las ecuaciones (33) y (34) para la función logística y las ecuaciones (37) y (38) para la función tangente hiperbólica, podemos calcular el gradiente local δ_j sin requerir un conocimiento explícito de la función de activación.

3.3.3. Tasa de Aprendizaje

El algoritmo back-propagation provee una "aproximación" a la trayectoria en el espacio de pesos calculada por el método del gradiente descendente. Cuanto más chico hacemos el parámetro η (tasa de aprendizaje), más pequeños serán los cambios a los pesos en la red de una iteración a la siguiente, y más suave será la trayectoria en el espacio de pesos. Esta mejora, sin embargo, se alcanza a costa de un aprendizaje más lento. Si, por el contrario, hacemos el parámetro η demasiado grande de manera de acelerar el aprendizaje, los grandes cambios resultantes en los pesos asumirán una forma en la cual la red puede llegar a ser inestable (es decir, oscilatoria). Un método sencillo de incrementar la tasa de aprendizaje y al mismo tiempo evitar el riesgo de inestabilidad es modificar a la regla delta de la ecuación (13) incluyendo el término *momentum*, como se muestra en [Rumelhart, 1986]:

$$\Delta w_{ji}(n) = a\Delta w_{ji}(n-1) + hd_j(n)y_i(n) \qquad (39)$$

donde a es usualmente un número positivo llamado ***constate de momentum***. Este controla el lazo de realimentación actuando alrededor de $Dw_{ji}(n)$, como se ilustra en la figura 3.6, donde z^{-1} es el operador de retardo unitario. La ecuación (39) se llama ***regla delta generalizada***; incluye la regla delta de la ecuación (13) como un caso especial ($a = 0$).

Para ver los efectos sobre los pesos de la presentación de la secuencia de patrones debido a la constante de momentum α, rescribimos a la ecuación (39) como una serie de tiempo con índice t. El índice t va desde el tiempo inicial 0 hasta el tiempo actual n.

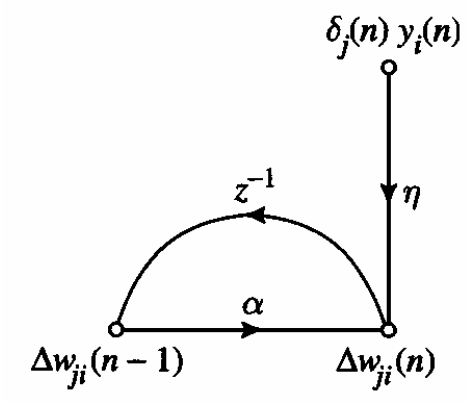


Figura 3.6 Grafo del flujo de señal ilustrando el efecto de la constante de momento α

La ecuación (39) puede verse como una ecuación diferencial de primer orden de la corrección de peso $Dw_{ji}(n)$. Resolviendo esta ecuación para $Dw_{ji}(n)$ tenemos:

$$\Delta w_{ji}(n) = h\sum_{t=0}^{n} a^{n-t} d_j(t) y_i(t) \qquad (40)$$

la cual representa una serie de tiempo de longitud $n + 1$. De la ecuación (11) y la (14) notamos que el producto $\delta_j(n)\, y_i(n)$ es igual a $-\partial e(n)/\partial w_{ji}(n)$,. Como consecuencia, podemos rescribir a la ecuación (40) en su forma equivalente:

$$\Delta w_{ji}(n) = -h\sum_{t=0}^{n} a^{n-t} \frac{\partial e(t)}{\partial w_{ji}(t)} \qquad (41)$$

Basado en esta relación, podemos hacer las siguientes observaciones [Watrous, 1987; Jacobs, 1988]:

a) El ajuste actual $Dw_{ji}(n)$ representa a la suma de una serie de tiempo ponderada exponencialmente. Para que la serie de tiempo sea convergente, la constante de momentum debe estar restringida al rango $0 \leq |a| < 1$. Cuando a es cero, el algoritmo back-propagation opera sin momentum. La constante de momentum puede ser positiva o negativa, aunque es improbable que se utilice en la práctica un a negativo.

b) Cuando la derivada parcial $\partial e(t)/\partial w_{ji}(t)$ tiene el mismo signo algebraico en iteraciones consecutivas, la suma ponderada exponencialmente $Dw_{ji}(n)$ crece en magnitud, por lo tanto se ajusta al peso $w_{ji}(n)$ con un valor alto. La inclusión del momentum en el algoritmo back-propagation tiende a *acelerar el descenso* en las direcciones cuesta debajo (de la superficie de error).

c) Cuando la derivada parcial $\partial e(t)/\partial w_{ji}(t)$ tiene el signo opuesto en iteraciones consecutivas, la suma ponderada exponencialmente $Dw_{ji}(n)$ se achica en magnitud, por lo tanto se ajusta al peso $w_{ji}(n)$ con un valor pequeño. La inclusión del momentum en el algoritmo back-propagation tiende un *efecto estabilizador* en las direcciones que oscilan en signo.

La incorporación del momentum en el algoritmo back-propagation representa una modificación menor a la actualización de pesos, pero puede tener efectos benéficos en el comportamiento del aprendizaje del algoritmo. El término momentum también puede ser beneficioso para prevenir que el proceso de aprendizaje termine en un mínimo local sobre la superficie de error.

En la derivación del algoritmo de back-propagation, se asumió que el parámetro de aprendizaje es una constante denotada por η. En realidad debería definirse como η_{ji}; ésto es, el parámetro de la tasa de aprendizaje debería ser dependiente de la conexión. Ciertamente, se pueden hacer muchas cosas interesantes al variar la tasa de aprendizaje para diferentes partes de la red. Se verán los detalles más adelante.

También es digno de destacar el hecho de que en la aplicación del algoritmo back-propagation se puede elegir que todos los pesos en la red sean ajustables, o se puede restringir cualquier número de pesos para que permanezcan fijos durante el proceso de adaptación. En el último caso, las señales de error se propagan hacia atrás a través de la red de la manera usual; sin embargo, los pesos fijos se dejan inalterados. Esto se puede lograr simplemente haciendo que la tasa de aprendizaje η_{ji} para el peso w_{ji} igual a cero.

3.3.4. Modos de Entrenamiento Secuencial y Batch

En una aplicación práctica del algoritmo back-propagation, el aprendizaje resulta de las muchas presentaciones al Perceptrón multicapa de un set de entrenamiento determinado. Como se mencionó anteriormente, una presentación completa del set de entrenamiento completo durante el proceso de entrenamiento se denomina un *epoch*. El proceso de aprendizaje se mantiene epoch por epoch hasta que los pesos de las conexiones y los umbrales de la red se estabilizan y el error cuadrático promedio sobre el set de entrenamiento completo converge a algún valor mínimo. Es una buena práctica randomizar el orden de presentación de los ejemplos de entrenamiento de un epoch al siguiente. Esta randomización tiende a hacer estocástica a la búsqueda en el espacio de pesos, evitando así la posibilidad de limitar al aprendizaje en la evolución de los pesos de la red.

Para un set de entrenamiento determinado, el aprendizaje back-propagation puede proceder en uno de los dos modos básicos:

a) ***Modo Secuencial***: A este modo de aprendizaje back-propagation también se lo denomina *modo on-line, por patrón o estocástico*. En este modo de operación, la actualización de pesos se lleva a cabo después de la presentación de cada ejemplo de entrenamiento; éste es modo que aplicamos en la derivación del algoritmo back-propagation. Para ser específicos, consideremos un epoch consistente en N ejemplos de entrenamiento (patrones) arreglados en el orden ($\mathbf{x}(1)$, $\mathbf{d}(1)$), ..., ($\mathbf{x}(N)$, $\mathbf{d}(N)$). El primer par de ejemplo ($\mathbf{x}(1)$, $\mathbf{d}(1)$) en el epoch se presenta a la red, y se desarrolla la secuencia de cómputos hacia delante y hacia atrás descripta previamente, resultando en ciertos ajustes a los pesos y a los umbrales de la red. Entonces se presenta el segundo par de ejemplo en el epoch ($\mathbf{x}(2)$, $\mathbf{d}(2)$), y se repite la secuencia de cómputos hacia delante y hacia atrás, resultando en más ajustes a los pesos y a los umbrales de la red. Se continúa este proceso hasta que se utiliza el último par de ejemplo ($\mathbf{x}(N)$, $\mathbf{d}(N)$).

b) ***Modo Batch***: En el modo batch del aprendizaje back-propagation, la actualización de los pesos se lleva a cabo *después* de la presentación de *todos* los ejemplos de entrenamiento que constituye un epoch. Para un epoch en particular, se define a la función de costo como el error cuadrático promedio de las ecuaciones (2) y (3), reproducido aquí en la forma compuesta:

$$e_{av}(n) = \frac{1}{2N}\sum_{n=1}^{N}\sum_{j\in C} e_j^2(n) \qquad (43)$$

donde la señal de error $e_j(n)$ pertenece a la neurona de salida j para el ejemplo de entrenamiento n y fue definida por la ecuación (1). El error $e_j(n)$ es igual a la diferencia entre $d_j(n)$ y $y_j(n)$, los cuales representan al j-ésimo elemento del vector $\mathbf{d}(n)$ de respuesta deseada y al valor correspondiente de salida de la red, respectivamente. En la ecuación (42) la sumatoria interna con respecto a j se desarrolla sobre todas las neuronas en la capa de salida de la red, mientras que la sumatoria externa con respecto a n se desarrolla sobre el set completo de entrenamiento del epoch en curso. Para una tasa de aprendizaje η, el ajuste aplicado al peso w_{ji}, conectando la neurona i a la neurona j, está definida por la regla delta:

$$\Delta w_{ji}(n) = -\eta \frac{\partial e_{av}(n)}{\partial w_{ji}(n)} = -\frac{\eta}{N}\sum_{n=1}^{N} e_j(n)\frac{\partial e_j(n)}{\partial w_{ji}(n)} \qquad (44)$$

Para calcular la derivada parcial $\partial e(n)/\partial w_{ji}$ procedemos de la misma forma que antes. De acuerdo a la ecuación (43), en el modo batch el ajuste de peso Δw_{ji} se realiza solo después de que el set de entrenamiento completo se ha presentado a la red.

Desde el punto de vista de la operación "on-line", se prefiere al modo secuencial por sobre el modo batch porque requiere menos almacenamiento local para cada conexión. Adicionalmente, dado que los patrones se presentan a la red de manera aleatoria, el uso de la actualización de pesos patrón-por-patrón convierte a la búsqueda en el espacio de pesos en estocástica por naturaleza. Esto a su vez disminuye la probabilidad que el algoritmo back-propagation quede atrapado en un mínimo local.

De la misma manera, la naturaleza estocástica del modo secuencial dificulta el establecimiento de condiciones teóricas para la convergencia del algoritmo. En contraste, el uso del modo batch de entrenamiento provee una estimación precisa del vector gradiente; la convergencia hacia un mínimo local está por lo tanto garantizada bajo condiciones sencillas. También, la composición del modo batch hace más fácil su paralelización que el modo secuencial.

Cuando los datos de entrenamiento son *redundantes* (el set de datos contiene varias copias exactas del mismo patrón), se observa que, a diferencia del modo batch, el modo secuencial es capaz de tomar ventaja de esta redundancia porque los ejemplos se presentan de a uno por vez. Esto es particularmente cierto cuando el set de datos es grande y altamente redundante.

En resumen, a pesar del hecho de que el modo secuencial del aprendizaje back-propagation tiene algunas desventajas, es ampliamente utilizado (particularmente para resolver problemas de clasificación de patrones) por dos importantes razones prácticas:

- El algoritmo es simple de implementar.
- Provee soluciones efectivas a problemas grandes y difíciles.

3.3.5. Criterios de Finalización

En general, no puede demostrarse que el algoritmo back-propagation converja, y no hay un criterio bien definido para detener su operación. Sin embargo hay algunos criterios razonables, cada uno con sus propios méritos prácticos, que pueden ser utilizados para finalizar el ajuste de pesos. Para formular tales criterios, es lógico pensar en términos de las propiedades únicas de un *mínimo local o global* de la superficie de error. Consideremos al vector de pesos **w*** que denota un mínimo, ya sea local o global. Una condición necesaria para que **w*** sea un mínimo es que el vector gradiente **g(w)** (es decir, la derivada parcial de primer orden) de la superficie de error con respecto al vector de pesos **w** sea cero cuando **w** = **w***. En consecuencia, podemos formular un criterio sensible a la convergencia para el aprendizaje back-propagation como [Kramer, 1989]:

Se considera que el algoritmo back-propagation ha convergido cuando la norma euclidiana del vector gradiente alcanza un valor lo suficientemente pequeño.

La desventaja de este criterio de convergencia es que, para experimentos exitosos, los tiempos de aprendizaje pueden ser grandes. También requiere el cómputo del vector gradiente **g(w)**.

Otra propiedad única de un mínimo que se puede usar es el hecho de que la función de costo o medida de error $e_{av}(\mathbf{w})$ es estacionaria en el punto **w** = **w***. Se puede entonces sugerir un criterio de convergencia distinto:

Se considera que el algoritmo back-propagation ha convergido cuando la tasa absoluta de cambio en el error cuadrático promedio por epoch es suficientemente pequeña.

A la tasa de cambio en el error cuadrático promedio se la considera lo suficiente pequeña normalmente si cae en el rango de 0,1 a 1 porciento por epoch. A veces se usa un valor incluso menor, como 0,01. Desafortunadamente, este criterio puede resultar en una finalización prematura del proceso de aprendizaje.

Existe otro criterio útil y con apoyo teórico para la convergencia. Después de cada iteración de aprendizaje, se evalúa la red en su capacidad de generalización. El proceso de aprendizaje se detiene cuando la performance de generalización es adecuada, o cuando aparentemente ha alcanzado un pico. Esto se verá con más detalle posteriormente.

3.4. Resumen del algoritmo back-propagation

Previamente se mencionó que la actualización secuencial es el método elegido para la implementación on-line del algoritmo back-propagation. Para este modo de operación, el algoritmo itera a través de los ejemplos de entrenamiento $\{(x(n), d(n))\}_{n=1}^{N}$ como sigue:

a) ***Inicialización***: Asumiendo que no hay disponible información a priori, escoger los pesos y los umbrales a partir de una distribución uniforme cuya media sea cero y cuya varianza se elige para hacer que la desviación estándar del campo local inducido de las neuronas caiga en la transición entre la parte lineal y saturada de la función de activación sigmoidal.

b) ***Presentación de los ejemplos de entrenamiento***: Presentar a la red un epoch de ejemplos de entrenamiento. Para cada ejemplo en el set, ordenado de alguna manera, llevar a cabo la secuencia de cómputos hacia adelante y hacia atrás descriptos en los puntos c y d, respectivamente.

c) ***Cómputo hacia adelante***: Sea un ejemplo de entrenamiento en el epoch denotado por ((**x**(*n*), **d**(*n*)), con el vector de entradas **x**(*n*) aplicado a la capa de entrada de nodos sensoriales y el vector de respuestas deseadas **d**(*n*) presentada a la capa de salida. Calcular los campos locales inducidos y las señales funcionales de la red al proceder hacia delante a través de la red, capa por capa. El campo local inducido $v_j^{(l)}(n)$ por la neurona *j* en la capa *l* es:

$$v_j^{(l)} = \sum_{i=0}^{m_{l-1}} w_{ji}^{(l)}(n) y_i^{(l-1)}(n)$$

donde $y_i^{(l-1)}(n)$ es la señal de salida de la neurona *i* en la capa previa $l - 1$ en la iteración *n*, y $w_{ji}^{(l)}(n)$ es el peso de la neurona *j* en la capa *l* que es alimentada desde la neurona *i* en la capa $l - 1$. Para $i = 0$, se tiene $y_0^{(l-1)}(n) = +1$ y $w_{j0}^{(l)}(n) = b_j^{(l)}(n)$ es el umbral aplicado a la neurona *j* en la capa *l*. Asumiendo el uso de una función sigmoidal, la señal de salida de la neurona *j* es:

$$y_j^{(l)}(n) = j_{\ j}\left(v_j(n)\right)$$

Si la neurona *j* está en la primer capa oculta ($l = 1$), setear:

$$y_j^{(0)}(n) = x_j^0(n)$$

donde $x_j(n)$ es el *j*-ésimo elemento del vector de entradas **x**(*n*). Si la neurona *j* está en la neurona de salida ($l = L$, donde L se refiere como la ***profundidad*** de la red), setear:

$$y_j^{(L)}(n) = o_j(n)$$

Calcular la señal de error

$$e_j(n) = d_j(n) - o_j(n)$$

donde $d_j(n)$ es el *j*-ésimo elemento del vector de respuestas deseadas **d**(*n*).

d) ***Cómputo hacia atrás***: Calcular los *d*s (gradientes locales) de la red, definidos por:

$$d_j^{(l)}(n) = \begin{cases} e_j^{(L}(n) j_{\ j}'\left(v_j^{(L)}(n)\right) & \text{para la neurona j en la capa de salida } L \\ j_{\ j}'\left(v_j^{(l)}(n)\right)\sum_k d_k^{(l+1)}(n) w_{kj}^{(l+1)}(n) & \text{para la neurona j en la capa ocula } l \end{cases}$$

donde la prima en $j\,'_j(\cdot)$ denota diferenciación con respecto al argumento. Ajustar los pesos de la red en la capa *l* de acuerdo a la regla delta generalizada:

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + a\left[w_{ji}^{(l)}(n-1)\right] + h d_j^{(l)}(n) y_{kj}^{(l-1)}(n)$$

donde η es el parámetro de la tasa de aprendizaje y α es la constante de momentum.

e) ***Iteración***: Iterar los cómputos hacia adelante y hacia atrás de los puntos c y d presentando nuevos epochs de ejemplos de entrenamiento a la red hasta que se alcance el criterio de finalización.

Nota: El orden de presentación de los ejemplos de entrenamiento debería ser randomizado de epoch a epoch. El momentum y la tasa de aprendizaje normalmente se ajustan (usualmente se disminuyen) al incrementarse el número de iteraciones. Las justificaciones para estos puntos se verán más adelante.

3.5. Heurísticas para hacer que el back-propagation se desempeñe mejor

A menudo se dice que el diseño de una red neuronal usando el algoritmo back-propagation es más un arte que una ciencia en el sentido en que el ajuste de muchos de los factores involucrados en el diseño es el resultado de la experiencia personal. Si bien hay algo de verdad en esto, existen métodos que mejoran significativamente la performance del algoritmo back-propagation como se describe aquí.

a) ***Actualización secuencial versus batch***: Como se mencionó anteriormente, el modo secuencial del algoritmo back-propagation (que involucra actualización patrón por patrón) es más rápido computacionalmente que el modo batch. Esto es especialmente cierto cuando el set de entrenamiento es grande y altamente redundante (los datos altamente redundantes imponen problemas computacionales para la estimación del Jacobiano, requerido en la actualización batch).

b) ***Maximizar el contenido de información***: Como regla general, cada ejemplo de entrenamiento presentado al algoritmo back-propagation debería elegirse basándose en que su contenido de información sea lo más grande posible para la tarea dada [LeCun, 1993].

Dos modos de lograr este objetivo son:

- El uso de un ejemplo que resulte en el mayor error de entrenamiento.
- El uso de un ejemplo que es radicalmente diferente de aquellos previamente usados.

Estas dos heurísticas están motivadas por el deseo de buscar en mayor profundidad el espacio de pesos.

En las tareas de clasificación de patrones usando el aprendizaje back-propagation secuencial, una técnica sencilla que se utiliza comúnmente es randomizar (barajar) el orden en el cual se presentan los ejemplos al Perceptrón multicapa de un epoch al siguiente. Idealmente, la randomización asegura que los ejemplos sucesivos en un epoch presentado a la red raramente pertenecen a la misma clase.

c) ***La función de activación***: Un Perceptrón multicapa entrenado con el algoritmo back-propagation puede, en general, aprender más rápido (en términos del número de iteraciones de entrenamiento requerido) cuando la función de activación sigmoidal construída dentro del modelo de neurona es antisimétrica, comparado a cuando ésta es no-simétrica [Haikin, 1999]. Decimos que una función de activación $j(v)$ es *antisimétrica* (función impar de su argumento) si:

$$j(-v) = -j(v)$$

como se muestra en la figura 3.7a. Esta condición no se satisface en la función logística estándar mostrada en la figura 3.7b.

Un ejemplo de función de activación antisimétrica es la no-linealidad sigmoidal en la forma de una *tangente hiperbólica*, definida por:

$$j(v) = a \tanh(bv)$$

donde a y b son constantes. Valores adecuados para las constantes a y b son [Le Cun, 1989, 1993]:

$$a = 1{,}7159 \qquad\qquad b = 2/3$$

La tangente hiperbólica así definida tiene las siguientes propiedades:

- $j(1) = 1$ y $j(-1) = -1$
- En el origen la pendiente (ganancia efectiva) de la función de activación está próxima a la unidad, como se muestra en:

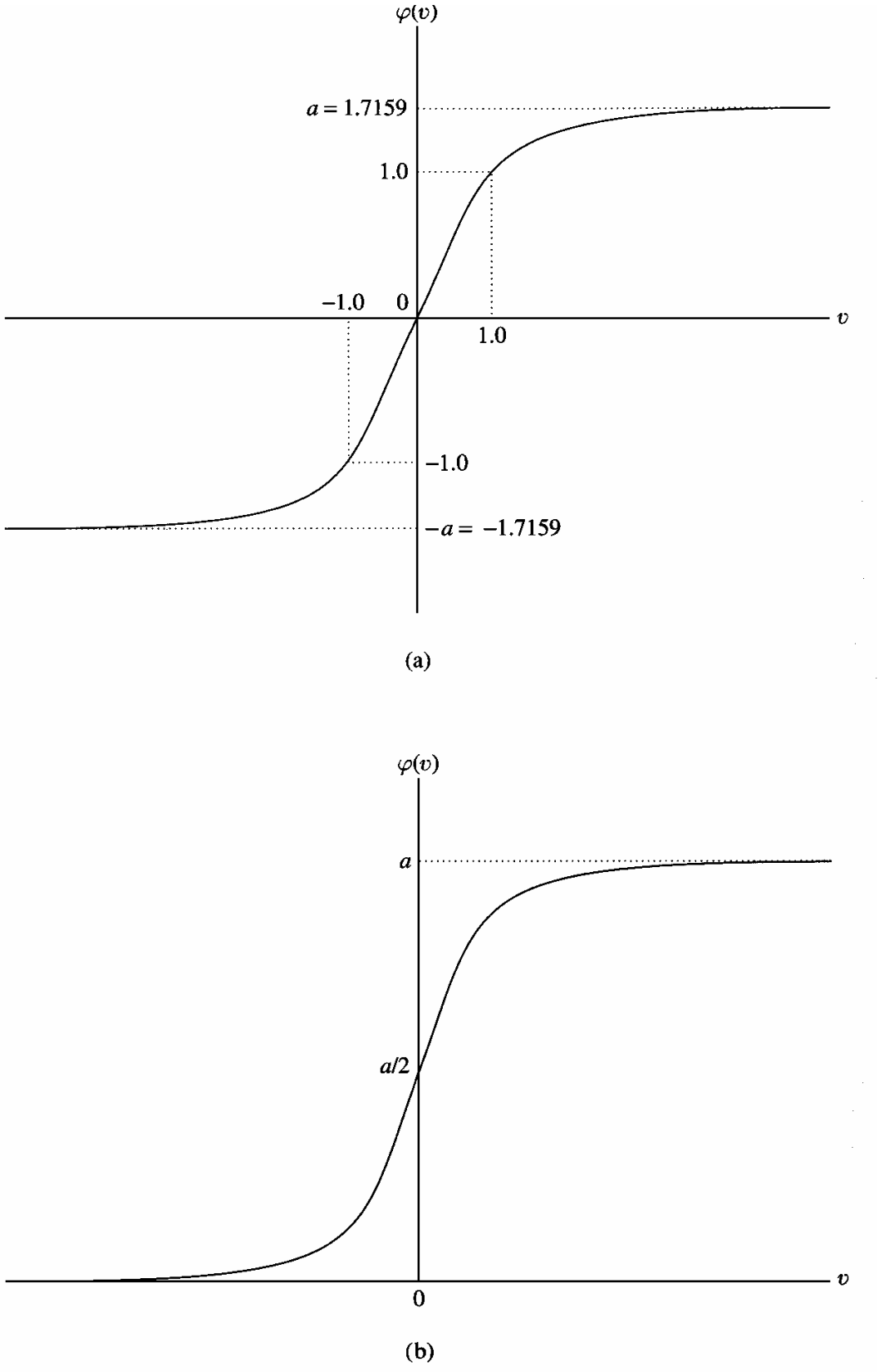


Figura 3.7 (a) Función de activación antisimétrica. (b) Función de activación no simétrica.

$$j'(0) = ab$$

$$= 1{,}7159 \text{ x } 2/3$$

$$= 1{,}1424$$

La segunda derivada de $j(v)$ alcanza su máximo valor en $v = 1$.

d) ***Valores objetivo***: Es importante que los valores objetivo (las respuestas deseadas) sean elegidos dentro del rango de la función de activación sigmoidal. Más específicamente, la respuesta deseada d_j para la neurona j en la capa de salida del Perceptrón multicapa debería ser desplazada mediante un offset de valor ε, alejándola del valor limitante de la función de activación sigmoidal, dependiendo de si este valor limitante es positivo o negativo. De lo contrario el algoritmo back-propagation tiende a llevar a infinito a los parámetros libres de la red, y por lo tanto el proceso de aprendizaje se hace más lento al ser dirigido por neuronas ocultas en saturación. Para ser específicos, consideremos la función de activación antisimétrica de la figura 3.7a. Para el valor limitante $+a$, se setea:

$$d_j = a - \varepsilon$$

y para el valor limitante $-a$, se setea:

$$d_j = -a + \varepsilon$$

donde ε es una constante positiva apropiada. Para la elección de $a = 1{,}7159$ referida anteriormente, se puede setear $\varepsilon = 0{,}7159$ en cuyo caso el valor objetivo (respuesta deseada) d_j puede elegirse convenientemente como ±1 como se indica en la figura 3.7a.

e) ***Normalizar las entradas***: Cada variable de entrada debería ser *preprocesada* de manera tal que su valor medio, promediado sobre el set de entrenamiento completo, esté cercano a cero [LeCun, 1993]. Para apreciar el significado práctico de esta regla, consideremos el caso extremo donde las variables de entrada son consistentemente positivas. En esta situación, los pesos de una neurona en la primer capa oculta solo pueden incrementarse juntos o decrementarse juntos. En consecuencia, si el vector de pesos de esas neuronas tiene que cambiar de dirección, solo puede hacerlo al zigzaguear su camino a través de la superficie de error, proceso que es normalmente lento y debería ser evitado.

f) ***Inicialización***: Una buena elección para los valores iniciales de los pesos y los umbrales puede ser de gran ayuda en el diseño exitoso de la red. La pregunta clave es: ¿Cuál es una buena elección?

Cuando a los pesos se les asignan valores iniciales grandes, es altamente probable que las neuronas de la red entren en saturación. Si esto ocurre, los gradientes locales en el algoritmo back-propagation asumen valores pequeños, lo cual a su vez causará que el proceso de aprendizaje sea más lento. Sin embargo, si a los pesos se les asignan valores iniciales chicos, el algoritmo back-propagation puede operar en un área muy aplanada alrededor del origen de la superficie de error; esto es particularmente cierto en el caso de las fun-

ciones de activación antisimétricas como la función tangente hiperbólica. Desafortunadamente el origen es un punto de ensilladura, el cual se refiere a un punto estacionario donde la curvatura de la superficie de error a lo ancho de la ensilladura es negativa y la curvatura a lo largo de la ensilladura es positiva. Por estos motivos, debe evitarse el uso de valores tanto grandes como pequeños para la inicialización de los pesos. La elección apropiada para la inicialización cae en algún lugar en medio de estos dos casos extremos.

Describiremos entonces una buena estrategia para inicializar los pesos de forma tal que la desviación estándar del campo local inducido de una neurona caiga en el área de transición entre la parte lineal y la parte saturada de su función de activación sigmoidal. Por ejemplo, para el caso de la función tangente hiperbólica con sus parámetros *a* y *b* especificados previamente, conviene utilizar una distribución uniforme, de donde se elegirán los pesos, con una media de cero y una varianza σ_w igual a:

$$\sigma_w = m^{-1/2}$$

donde *m* es el número de pesos de la neurona [LeCun, 1993].

g) ***Aprender desde pistas***: Al aprender a partir de un set de entrenamiento se trabaja con una función $f(\cdot)$ de mapeo entrada-salida desconocida. En efecto, el proceso de aprendizaje explota la información contenida en los ejemplos acerca de la función $f(\cdot)$ para inferir una implementación aproximada de ella. El proceso de aprendizaje desde ejemplos puede ser generalizado para incluir *aprendizaje desde pistas*, que puede lograrse al permitir la inclusión de información a priori que podemos tener de la función $f(\cdot)$ en el proceso de aprendizaje [Abu-Mostafa, 1995]. Tal información puede incluir propiedades de la varianza, simetrías, o cualquier otro conocimiento acerca de la función $f(\cdot)$ que puede emplearse para acelerar la búsqueda de su implementación aproximada, y aún más importante, mejorar la calidad de la estimación final. El uso de la ecuación de varianza en el punto anterior es un ejemplo de ello.

h) ***Tasa de aprendizaje***: Todas las neuronas en un Perceptrón multicapa idealmente aprenderían a la misma velocidad. Las últimas capas usualmente tienen gradientes locales mayores que las capas en la parte frontal de la red. Por lo tanto, se debería asignarle a la tasa de aprendizaje η un valor más pequeño en las últimas capas que en las primeras. Las neuronas con muchas entradas deberían tener una tasa de aprendizaje menor que las neuronas con pocas entradas de manera de mantener un tiempo de aprendizaje similar para todas las neuronas de la red. En [LeCun, 1993] se sugiere que para una neurona dada, la tasa de aprendizaje debería ser inversamente proporcional a la raíz cuadrada de los pesos conectados a esa neurona.

3.6. Generalización

En el aprendizaje back-propagation, normalmente comenzamos con una muestra de entrenamiento y usamos el algoritmo back-propagation para computar los pesos de un Perceptrón multicapa al cargar tantos ejemplos de entrenamiento como sea posible dentro de

la red. Entonces esperamos que la red así diseñada generalice. Se dice que una red neuronal *generaliza* bien cuando el mapeo entrada-salida computado por la red es correcto (o casi) para datos de testeo nunca utilizados en la creación o entrenamiento de la red; el término "generalización" se deriva de la psicología. Aquí se asume que los datos de testeo son obtenidos de la misma población usada para generar el set de entrenamiento.

El proceso de aprendizaje (entrenamiento de la red neuronal) puede ser visto como un problema de "*ajuste de curva*". La red en sí misma puede considerarse simplemente como un mapeo entrada–salida no lineal. Tal punto de vista permite ver a la generalización no como una propiedad mística de las redes neuronales sino simplemente como el efecto de una buena interpolación no-lineal de los datos de entrada [Wieland, 1987]. La red desarrolla una interpolación útil principalmente porque los perceptrones multicapa con funciones de activación continuas conducen a funciones de salida que también son continuas.

La figura 3.8a ilustra cómo puede ocurrir la generalización en una red hipotética. El mapeo entrada-salida no lineal representado por la curva mostrada en esta figura es computada por la red como resultado de aprender los puntos etiquetados como "datos de entrenamiento". El punto sobre la curva marcado como "generalización" es visto entonces como el resultado de la interpolación desarrollada por la red.

Una red neuronal que se diseña para generalizar bien producirá un mapeo entrada-salida correcto aún cuando la entrada es ligeramente distinta de los ejemplos usados para entrenar a la red, como se ilustra en la figura. Sin embargo, cuando una red neuronal aprende demasiados ejemplos de entrada-salida, la red terminará memorizando el set de entrenamiento. Puede hacerlo al encontrar una característica (debido al ruido, por ejemplo) que está presente en los datos de entrenamiento pero no es cierta para la función subyacente que debe modelizarse. A este fenómeno se lo conoce como *sobreajuste* o *sobreentrenamiento*. Cuando la red está sobreentrenada, pierde su habilidad para generalizar entre patrones de entrada-salida similares.

Comúnmente, la carga de datos en un Perceptrón multicapa de esta forma requiere el uso de más neuronas ocultas de lo que realmente es necesario, con el resultado de que las contribuciones indeseadas en el espacio de entrada debido al ruido sean almacenadas en los pesos de la red. Un ejemplo de cuán pobre puede llegar a ser la generalización debido a la memorización en una red neuronal se ilustra en la figura 3.8b para los mismos datos mostrados en la figura 3.8a. La "memorización" es esencialmente una "búsqueda en una tabla", lo cual implica que el mapeo entrada-salida no es suave. Como se señala en [Poggio, 1990], la suavidad del mapeo se relaciona íntimamente con un criterio de selección cuya esencia es seleccionar la función más simple en la ausencia de algún conocimiento a priori que indique lo contrario. En nuestro contexto, la función más simple significa la función más suave que aproxime el mapeo para un criterio de error determinado, porque tal elección generalmente demandará menos recursos computacionales. La suavidad también es natural en muchas aplicaciones, dependiendo de la escala del fenómeno siendo estudiado. Es importante por lo tanto buscar un mapeo no-lineal suave para las relaciones débilmente formuladas entre entradas y salidas, de manera que la red sea capaz de clasificar nuevos patrones correctamente con respecto a los patrones de entrenamiento [Wieland, 1987].

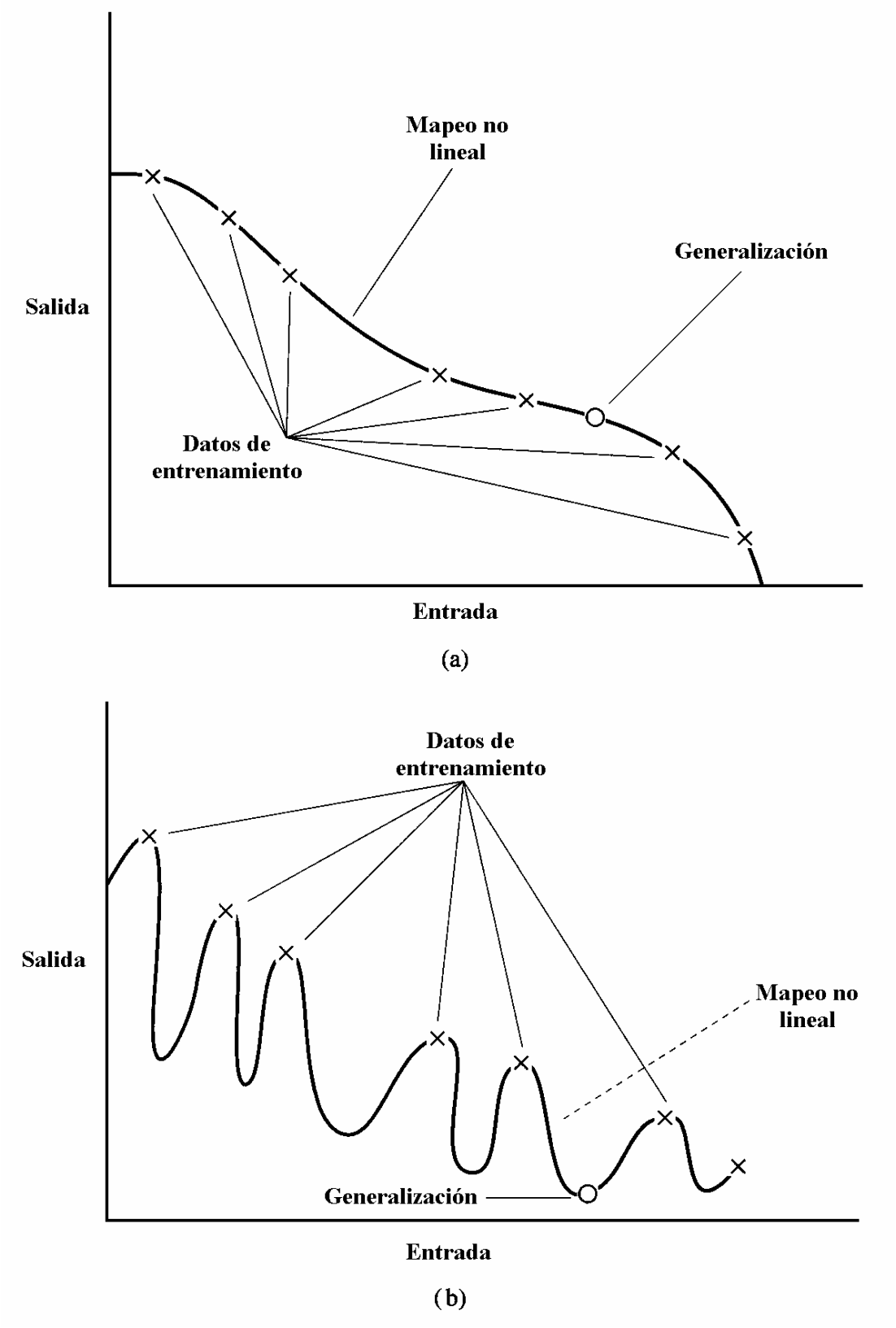


Figura 3.8 (**a**) Datos ajustados apropiadamente (generalización buena). (**b**) Datos sobreajustados (generalización pobre).

3.6.1. Validación Cruzada

La esencia del aprendizaje back-propagation es codificar un mapeo entrada-salida (representado por un conjunto de ejemplos etiquetados) en los pesos y umbrales de un Perceptrón multicapa. Lo que queremos es que la red se entrene bien de forma tal que aprenda lo suficiente acerca del pasado para generalizar en el futuro. Desde esta perspectiva el proceso de aprendizaje debe elegir la parametrización de la red más acorde a este conjunto de ejemplos. Más específicamente, podemos ver al problema de selección de la red como una elección, dentro del conjunto de modelos de estructuras candidatas (parametrizaciones), de la mejor estructura de acuerdo a un cierto criterio.

En este contexto, una herramienta estándar en estadística conocida como *validación cruzada* provee una guía interesante [Stone, 1974]. El set de entrenamiento se particiona aleatoriamente en dos subsets disjuntos:

- El ***subset de estimación***, usado para seleccionar el modelo.
- El ***subset de validación***, usado para evaluar o validar el modelo.

La motivación aquí es validar el modelo sobre un set de datos diferente de aquel utilizado para la estimación de los parámetros. De este modo podemos usar el set de entrenamiento para evaluar la performance de varios modelos candidatos, y así elegir el mejor.

El uso de la validación cruzada resulta particularmente interesante cuando debemos diseñar una red neuronal grande que tenga como objetivo a la generalización.

3.6.2. Entrenamiento con Detención Temprana (Early Stopping)

Comúnmente, un Perceptrón multicapa entrenado con el algoritmo de back-propagation aprende en etapas, moviéndose desde la realización de funciones de mapeo bastante simples a más complejas a medida que progresa la sesión de entrenamiento. Esto se ejemplifica por el hecho de que en una situación típica el error cuadrático medio disminuye con el incremento del número de epochs durante el entrenamiento: comienza con un valor grande, decrece rápidamente, y luego continúa disminuyendo lentamente mientras la red hace su camino hacia un mínimo local sobre la superficie de error. Teniendo como meta a una buena generalización, es muy difícil darse cuenta cuándo es el mejor momento de detener el entrenamiento si solamente estamos mirando a la curva de aprendizaje para el entrenamiento. En particular, como mencionamos anteriormente, es posible que la red termine sobreajustándose a los datos de entrenamiento si la sesión de entrenamiento no se detiene en el momento correcto.

Podemos identificar el comienzo del sobreajuste a través del uso de la validación cruzada, para lo cual los ejemplos de entrenamiento se separan en un subset de estimación y un subset de validación. El subset de estimación se utiliza para entrenar a la red en el modo usual, excepto por una modificación menor: la sesión de entrenamiento se detiene perió-

dicamente (cada tantos epochs), y se evalúa la red con el set de validación después de cada período de entrenamiento. Más específicamente, el proceso periódico de estimación-seguida-de-validación procede de la siguiente manera:

- Después del período de estimación (entrenamiento), se fijan todos los pesos y los umbrales del Perceptrón multicapa, y la red opera en su modo hacia delante. El error de validación se mide así para cada ejemplo en el set de validación.

- Cuando la fase de validación se completa, la estimación (entrenamiento) se reanuda para otro período y el proceso se repite.

Este procedimiento se denomina método de entrenamiento con detención temprana.

La figura 3.9 muestra las formas conceptualizadas de dos curvas de aprendizaje, una perteneciente a las medidas sobre el subset de estimación y la otra sobre el subset de validación. Normalmente, el modelo no trabaja tan bien sobre el subset de validación como lo hace sobre el set de estimación, en el cual se basó su diseño. La *curva de aprendizaje de estimación* decrece monótonamente para un número creciente de epochs en la forma acostumbrada. En contraste, *curva de aprendizaje de validación* decrece monótonamente hasta un mínimo, entonces empieza a incrementarse mientras continúe el entrenamiento. Cuando miramos a la curva de aprendizaje de estimación puede parecer que podríamos mejorar si vamos más allá del punto mínimo sobre la curva de aprendizaje de validación.

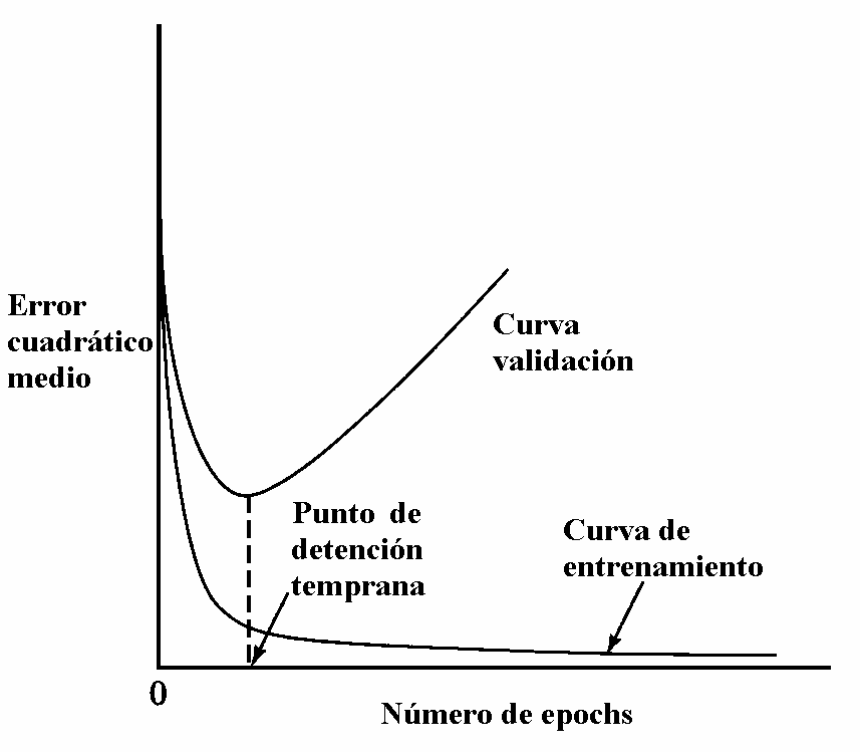


Figura 3.9 Ilustración de la regla de detención temprana basada en la validación cruzada.

En realidad, lo que la red está aprendiendo más allá de ese punto es esencialmente ruido contenido en el set de entrenamiento. Esta heurística sugiere que el punto mínimo sobre la curva de aprendizaje de validación sea utilizado como un criterio para detener la sesión de entrenamiento.

¿Qué ocurre si el set de entrenamiento está libre de ruido? ¿Cómo podemos entonces justificar la detención temprana para un escenario determinístico? Parte de la respuesta en este caso es que si ambos errores, el de estimación y el de validación, no se pueden llevar simultáneamente a cero, implica que la red no tiene la capacidad de modelizar exactamente a la función. Lo mejor que podemos hacer en esta situación es tratar de minimizar, por ejemplo, el error cuadrático integrado que es (aproximadamente) equivalente a minimizar el error cuadrático medio global con una densidad de entrada uniforme.

Una teoría estadística del fenómeno del sobreajuste se presenta en [Amari, 1996], donde se identifican dos modos de comportamiento dependiendo del tamaño del set de entrenamiento:

a) ***Modo no-asintótico***, para el cual $N < W$, donde N es el tamaño del set de entrenamiento y W es el número de parámetros libres de la red. Para este modo de comportamiento, el método de entrenamiento con detención temprana mejora la performance de generalización de la red por sobre el entrenamiento exhaustivo (es decir, cuando se utiliza el set de ejemplos completo para el entrenamiento y la sesión no se detiene). Este resultado sugiere que el sobreajuste puede ocurrir cuando $N < 30\ W$, y hay méritos prácticos en el uso de la validación cruzada para la detención del entrenamiento. El valor óptimo del parámetro r que determina la separación de los datos de entrenamiento entre los subsets de estimación y validación se define como:

$$r_{opt} = 1 - \frac{\sqrt{2W - 1} - 1}{2(W - 1)}$$

Para W grandes, esta fórmula se aproxima a:

$$r_{opt} = 1 - \frac{1}{\sqrt{2W}}$$

Por ejemplo, para $W = 100$, $r_{opt} = 0{,}07$ lo cual significa que el 93 porciento de los datos de entrenamiento será asignado al subset de estimación y el restante 7 porciento será asignado al subset de validación.

b) ***Modo asintótico***, para el cual $N > 30\ W$. Para este modo de comportamiento, la mejora en la performance de generalización producida por el uso del método de entrenamiento con detención temprana por sobre el entrenamiento exhaustivo es pequeña. En otras palabras, el entrenamiento exhaustivo es satisfactorio cuando el tamaño del set de entrenamiento es grande comparado con el número de parámetros de la red.

3.6.3. Variantes a la Validación Cruzada

El enfoque a la validación cruzada descripto se denomina método ***hold-out***. Existen otras variantes de validación cruzada que encuentran sus propios usos prácticos, particularmente cuando hay escasez de ejemplos etiquetados (entrada y salida deseada). En tal situación podemos usar validación cruzada ***multiplegada*** (multifold) dividiendo el set de N ejemplos disponibles en K subsets, $K > 1$; se asume que N es divisible por K. El modelo se entrena sobre todos los subsets excepto por uno, y el error de validación se mide al evaluarlo sobre el subset dejado de lado. Este proceso se repite para un total de K pruebas, cada vez usando un subset diferente para la validación, como se ilustra en la figura 3.10 para $K = 4$. La performance del modelo se calcula al promediar el error cuadrático bajo la validación sobre todas las pruebas del experimento. Hay una desventaja de la validación cruzada multiplegada: requiere una cantidad excesiva de computación ya que el modelo tiene que ser entrenado K veces, donde $1 < K \leq N$.

Cuando el número disponible de ejemplos etiquetados, N, está severamente limitado, podemos usar la forma extrema de la validación cruzada multiplegada conocida como el método ***dejar-uno-afuera***. En este caso, N - 1 ejemplos se utilizan para entrenar el modelo, y el modelo se valida al evaluarlo con el ejemplo dejado afuera. El experimento se repite un total de N veces, cada vez dejando de lado un ejemplo diferente para validación. El error cuadrático bajo validación se promedia entonces sobre las N pruebas del experimento.

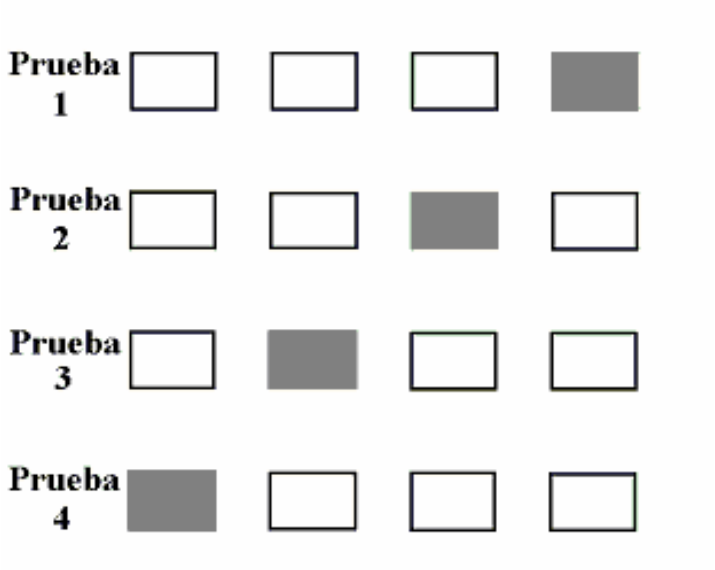


Figura 3.10 Ilustración del método de hold-out de validación cruzada. Para una prueba dada, el subset de datos sombreado se usa para validar el modelo entrenado con el resto de los datos.

3.6.4. Poda de la Red (Pruning)

Para resolver problemas del mundo real con redes neuronales usualmente se requiere el uso de redes altamente estructuradas de bastante tamaño. Un asunto práctico que asoma en este contexto es aquel sobre la minimización del tamaño de la red manteniendo la buena performance. Una red neuronal con tamaño mínimo es menos probable que aprenda el ruido en los datos de entrenamiento, y puede así generalizar mejor sobre datos nuevos. Se puede lograr este objetivo de diseño con una de las siguientes formas:

a) ***Crecimiento de la red***: en cuyo caso comenzamos con un Perceptrón multicapa pequeño (pequeño para cumplir la tarea en cuestión) y luego agregamos una nueva neurona o una nueva capa de neuronas ocultas solo cuando no seamos capaces de cubrir la especificación de diseño.

b) ***Poda de la red (pruning)***: en cuyo caso se comienza con un Perceptrón multicapa grande con una performance adecuada para el problema en cuestión, y entonces lo podamos al debilitar o eliminar ciertos pesos en una manera selectiva y ordenada.

Nos concentraremos en la poda de la red, en particular describiremos un enfoque basado en una forma de ***regularización de la complejidad***. Esto se debe a que la producción de la topología (es decir, el número de capas y neuronas ocultas de la red) la dejaremos en manos del proceso evolutivo.

Al diseñar un Perceptrón multicapa por el método que sea, estamos en efecto construyendo un *modelo* no-lineal del fenómeno físico responsable de la generación de ejemplos de entrada-salida usados para entrenar la red. En la medida que el diseño de la red es estadístico por naturaleza, necesitamos un balance adecuado entre la confiabilidad de los datos de entrenamiento y la calidad del modelo. En el contexto del aprendizaje back-propagation o cualquier otro procedimiento de aprendizaje supervisado, podemos producir el balance minimizando el riesgo total expresado como [Haykin, 1999]:

$$R(\mathbf{w}) = \varepsilon_S(\mathbf{W}) + \lambda\, \varepsilon_C(\mathbf{w})$$

El primer término, $\varepsilon_S(\mathbf{W})$, es la ***medida de performance*** estándar, la cual depende tanto de la red (modelo) y de los datos de entrada. En el aprendizaje back-propagation se define típicamente como un error cuadrático medio cuya evaluación se extiende sobre las neuronas de salida de la red, y el cual se lleva a cabo para todos los ejemplos de entrenamiento epoch por epoch. El segundo término, $\varepsilon_C(\mathbf{w})$, es la ***penalidad de complejidad***, la cual depende solamente de la red (modelo); su inclusión impone sobre la solución un conocimiento a priori que tengamos sobre los modelos siendo considerados. Podemos pensar a λ como un ***parámetro de regularización***, representando la importancia relativa del término de la penalidad de complejidad con respecto al término de la medida de performance. Cuando λ es cero, el proceso de aprendizaje no está restringido, y la red se determina completamente con los ejemplos de entrenamiento. Cuando λ se hace infinitamente grande, por el contrario, la implicación es que la restricción impuesta por la penalidad de complejidad es por sí misma suficiente para especificar la red, lo cual es otra manera de decir que los ejemplos de entrenamiento no son confiables. En los casos prácticos donde

se utiliza la regularización de la complejidad para mejorar la generalización, al parámetro λ se le asigna un valor entre medio de estos dos casos extremos.

La forma de regularización de complejidad que utilizamos en este trabajo es la ***degradación de pesos (weight decay)*** [Hinton, 1989]. En el procedimiento weight-decay, el término penalidad de complejidad se define como la norma al cuadrado del vector de pesos **w** (es decir, todos los parámetros libres) en la red, como se muestra en:

$$e_C(w) = \|w\|^2 = \sum_{i \in C_{total}} w_i^2$$

donde el conjunto C_{total} se refiere a todos los pesos de la red. Este procedimiento opera al forzar a algunos de los pesos en la red a tomar valores cercanos a cero, mientras permite que otros pesos retengan sus valores relativamente grandes. En consecuencia, los pesos de la red se agrupan a grandes rasgos en dos categorías: aquellos que tienen una gran influencia sobre la red (modelo), y aquellos que tienen poca o ninguna influencia sobre ella. Los pesos en la última categoría se llaman *pesos excedentes*. En la ausencia de la regularización de complejidad, estos pesos resultan en una generalización pobre en virtud de sus altas probabilidades de tomar valores completamente arbitrarios o causar que la red sobreajuste los datos al tratar de producir una leve reducción en el error de entrenamiento [Hush, 1993]. El uso de la regularización de complejidad fomenta que los pesos excedentes asuman valores cercanos a cero, y por lo tanto mejoran la generalización.

4. Cómo trabajan los Algoritmos Genéticos (AG)

A continuación se brinda una introducción a los conceptos básicos sobre algoritmos genéticos en la sección 4.1. Luego se presenta las cuestiones referentes a la representación genética en 4.2, la función de aptitud en 4.3, la población en 4.4, la selección en 4.5, la recombinación en 4.6, la mutación en 4.7, el reemplazo en 4.8, la convergencia en 4.9 y el análisis de la población en 4.10.

4.1. Introducción

El dominio de la computación evolutiva comprende el estudio de los principios y las aplicaciones de las técnicas computacionales basadas en el concepto de la evolución natural. La evolución en la naturaleza es responsable del "diseño" de todos los seres vivos sobre la tierra, y de las estrategias que éstos utilizan para interactuar entre sí. Los algoritmos evolutivos emplean esta poderosa filosofía de diseño para encontrar soluciones a problemas difíciles.

En general, las técnicas evolutivas pueden ser vistas como métodos de búsqueda, o como técnicas de optimización. Según [Michalewicz, 1996]: "Cualquier tarea abstracta a realizarse puede pensarse como la resolución de un problema, que, a su vez, pude percibirse como una búsqueda a través de un espacio de soluciones posibles. Ya que comúnmente estamos tras la "mejor" solución, podemos ver a esta tarea como un proceso de optimización.

Tres mecanismos básicos dirigen la evolución natural: ***reproducción***, ***mutación*** y ***selección***. Los dos primeros actúan sobre los ***cromosomas*** conteniendo la información genética del individuo (el ***genotipo***), y el tercero lo hace sobre el individuo en sí (el ***fenotipo***). La reproducción es el proceso por el cual se introducen nuevos individuos dentro de una ***población***. Durante la reproducción sexual, ocurre la ***recombinación*** (o ***cruza***), transmitiendo información genética de ambos padres a los cromosomas de la descendencia. La mutación introduce cambios pequeños a estos cromosomas; a menudo es el resultado de errores en la copia durante la reproducción. La selección es un proceso guiado por el principio Darwiniano de la supervivencia del más apto. Los individuos más aptos son aquellos que están más adaptados a su medio ambiente, los cuales sobreviven y se reproducen.

La computación evolutiva hace uso de la metáfora de la evolución natural. De acuerdo a esta metáfora, un problema juega el rol de un medio ambiente donde vive una población de individuos, cada uno representando una solución posible al problema. El grado de adaptación de cada individuo (i.e., solución candidata) a su medio ambiente se expresa mediante una medida adecuada conocida como ***función de aptitud (fitness)***. El fenotipo de cada individuo generalmente se codifica de alguna manera dentro de su ***genoma*** (genotipo). Como la evolución en la naturaleza, los algoritmos evolutivos potencialmente producen progresivamente mejores soluciones al problema. Esto es posible gracias a la

introducción constante de material genético "nuevo" dentro de la población, al aplicar los llamados ***operadores genéticos*** que son los equivalentes computacionales a los mecanismos evolutivos naturales antes descriptos.

Los algoritmos evolutivos tienen cuatro diferencias principales con los métodos más utilizados o conocidos de optimización y búsqueda:

- Trabajan con una codificación del conjunto de parámetros, no con estos directamente.
- Buscan simultáneamente la solución en una población de puntos, no en uno sólo.
- Utilizan la función de aptitud, no derivadas u otro conocimiento auxiliar
- Utilizan reglas de transición probabilísticas, y no determinísticas.

Existen varios tipos de algoritmos evolutivos, entre los cuales el algoritmo genético, o AG, es el más conocido. Básicamente, el AG realiza un ciclo consistente en un proceso iterativo para hacer que la población evolucione, a cada paso de este proceso se lo conoce como ***generación***, y a la condición para su finalización se la llama ***convergencia***. El funcionamiento del AG se puede esquematizar en el siguiente pseudocódigo [Hue Xavier, 1997]:

```
Comienzo /* algoritmo genético */

     Generar la población inicial;
     Computar la aptitud de cada individuo;
     Mientras no terminado hacer ciclo
          Comenzar
               Seleccionar individuos de las generaciones anteriores para reproducción;
               Crear descendencia aplicando recombinación y/o mutación a los individuos seleccionados;
               Computar la aptitud de los nuevos individuos;
               Eliminar a los individuos antiguos para hacer lugar para los nuevos cromosomas e insertar la descendencia en la nueva generación;
               Si la población ha convergido
                    Entonces terminado:= VERDADERO;
          Fin
Fin
```

Este capítulo intenta hacer un resumen de los aspectos técnicos del AG. Los distintos pasos del algoritmo se estudian en forma separada, analizando particularmente las dificultades y los puntos esenciales que utilizamos en este trabajo.

4.2. Representación genética

Para aplicar el AG a un problema específico, debemos definir primero una representación genética apropiada para la solución. Debemos encontrar un modo de codificar cualquier solución del problema dentro de un ***cromosoma*** de una cierta estructura. Esta estructura compartida por todos los cromosomas se llama ***representación genética***.

Generalmente las soluciones son codificadas en una cadena de bits de una longitud dada. Los cromosomas de cadenas de bits son bastante prácticos porque pueden representar de todo. Históricamente, los AG han intentado ser universales, capaces de tratar con un rango amplio de problemas. Por lo tanto la codificación binaria fue vista como una representación estándar que podía ajustarse a casi toda clase de espacio de búsqueda. De hecho, una cadena de bits puede codificar enteros, reales, conjuntos o lo que sea apropiado para el problema. Además, los cromosomas binarios son muy sencillos de manipular. La mutación y la recombinación de cadenas de bits se pueden hacer con operadores muy simples. Sin embargo, las cadenas de bits a veces no son apropiadas para problemas particulares. Se podría personalizar una representación específica al problema, enteros para representar enteros, reales para representar reales, etc., de una manera que le dé más sentido y coherencia al algoritmo. Pero, como se verá más adelante, una representación diferente también requiere un conjunto de operadores específicamente adaptado.

Para un individuo particular, usamos la palabra ***genotipo*** para describir el conjunto de sus genes. El ***fenotipo*** describe el aspecto físico de un individuo, su composición. La ***morfogénesis*** es el proceso de decodificar el genotipo para producir el fenotipo. Un punto interesante de la evolución es que la selección siempre se hace sobre el fenotipo, mientras que la reproducción recombina genotipos. En consecuencia se debería considerar la importancia de la morfogénesis como un vínculo entre la selección y la reproducción.

La función de morfogénesis asocia a cada genotipo con su fenotipo. Esto quiere decir simplemente que cada cromosoma debe definir una única solución, pero no significa que cada solución es codificada exactamente por un cromosoma. De hecho, la función de morfogénesis no tiene que ser necesariamente biyectiva, y hasta a veces es imposible que así sea (especialmente con la representación binaria). Sin embargo esta función al menos debería ser sobreyectiva (todas las soluciones candidatas del problema deben corresponder al menos a un cromosoma posible) para que pueda ser explorada la totalidad del espacio de búsqueda. Cuando la función de morfogénesis que asocia cada cromosoma a una solución no es inyectiva (diferentes cromosomas codifican la misma solución) se dice que la representación es degenerada. Una ligera degeneración no es preocupante, aún si inevitablemente aumenta el espacio de búsqueda donde el algoritmo busca a la solución óptima. Pero una degeneración importante podría ser un problema más serio, afectando el

comportamiento del AG. El significado de cada gen obviamente no corresponderá a una característica específica de la solución; puede incluso añadir cierta clase de confusión en la búsqueda.

Hay algunos puntos importantes que una representación genética debería tratar de respetar. Cada ***gen,*** o elemento básico del cromosoma, debería corresponder a una característica en particular. En la medida de lo posible, cada una de estas características tendría que ser independiente para evitar la interacción entre los genes. Idealmente, no debería haber correlación entre los valores de los genes. En este caso hipotético, se puede encontrar la solución óptima al seleccionar independientemente el mejor ***alelo*** (valor) para cada gen, La correlación entre genes se denomina ***epístasis***.

4.3. Función de aptitud

Una vez definida la representación genética, el próximo paso es asociar a cada solución (cromosoma) un valor correspondiente a la función de aptitud. Generalmente no hay inconvenientes en determinar la función de aptitud. De hecho, la mayoría de las veces se encuentra definida implícitamente por el problema, ya que se sabe qué es lo que se quiere optimizar. Pero, tan simple como parezca, hay que prestar particular atención debido al hecho de que la selección se lleva a cabo de acuerdo a la aptitud de los individuos. La función de aptitud debería no solo indicar cuán buena es la solución, sino también cuán cerca está el cromosoma de la solución óptima.

Uno de los problemas más molestos de los AG es su incapacidad de abordar restricciones. Idealmente, la representación debería ser tal que solo se pudieran generar soluciones factibles. Si todos los cromosomas posibles corresponden a soluciones factibles, entonces no hay necesidad de preocuparse por las restricciones. Pero para que esto ocurra se supone que la función que codifica los cromosomas en soluciones candidatas es capaz de tomar en cuenta las violaciones a las restricciones. Desafortunadamente no siempre es fácil "reparar" malos cromosomas, y a veces no se puede encontrar una codificación que evite soluciones no factibles. El enfoque clásico a este caso es el uso de la función de penalidad que evalúa la magnitud de la violación. Si la penalidad es lo suficientemente grande, individuos altamente inviables raramente serán seleccionados para la selección, y el AG se concentrará en soluciones factibles o cuasi factibles.

4.4. Población

Típicamente hay dos preguntas que suelen hacerse acerca de la población utilizada en el algoritmo genético: ¿Cómo se genera la ***población inicial***? ¿Cuál es el ***tamaño*** adecuado de la población?

De hecho, se las podría formular de un modo un poco diferente: ¿Cómo afectará la diversidad genética en la población al comportamiento del algoritmo? Idealmente, la primera población debería tener un contenido genético tan amplio como sea posible para que pueda explorar la totalidad del espacio de búsqueda. En esta población deberían estar presentes todos los distintos alelos posibles de cada gen. Para lograrlo, en la mayoría de los casos se elige a la primera población de forma aleatoria. Sin embargo, a veces se puede usar alguna clase de heurística para establecer la población inicial. Así, la aptitud promedio de la población ya es alta y puede ayudar al algoritmo genético a encontrar soluciones más rápidamente. Pero al hacer ésto hay que asegurarse de que el contenido genético todavía siga siendo lo suficientemente amplio. De lo contrario, si la población carece de diversidad, el algoritmo solo explorará una pequeña parte del espacio de búsqueda y nunca encontrará soluciones óptimas globales.

El tamaño de la población tampoco trae muchos problemas. Cuanto mayor es la población, más fácil es explorar el espacio de búsqueda. Pero el tiempo requerido por un AG para converger está dado por funciones de evaluación del orden $(n \log n)$ donde n es el tamaño de la población [Goldberg, 1991]. Se dice que la población ha convergido cuando todos los individuos se parecen y nuevas mejoras son posibles solo por mutación. Por otro lado, se demuestra que la eficiencia de un AG para alcanzar un óptimo global (evitando los locales) está determinado en gran medida por el tamaño de la población. En síntesis, una población grande es muy útil pero requiere un costo computacional mayor tanto en memoria como en tiempo. En la práctica, una población de alrededor de 20 individuos es bastante frecuente, pero de todas formas este tamaño puede cambiarse de acuerdo al tiempo y la memoria que se dispone comparado con la calidad de los resultados que se quiere alcanzar.

4.5. Selección

Existen muchas maneras de seleccionar individuos. Lo que se desea para la selección es un método que aleatoriamente elija cromosomas de la población de acuerdo a la función de evaluación. Cuanto más alta sea la función de aptitud, mayor será la oportunidad que un individuo tiene de ser seleccionado. La ***presión selectiva*** se define como el grado en el cual los mejores individuos son favorecidos. Esta presión selectiva conduce a que el AG mejore la aptitud de la población sobre las sucesivas generaciones.

La ***tasa de convergencia*** de un AG está determinada en gran parte por la magnitud de la presión selectiva, a más alta presión resultan tasas de convergencia más elevadas. Los AG deberían ser capaces de identificar soluciones óptimas o cuasi óptimas bajo un amplio rango de presiones selectivas [Miller, 1995]. Sin embargo, si la presión selectiva es demasiada baja, la tasa de convergencia será baja, y el AG se tomará un tiempo innecesariamente largo para hallar la solución óptima. Si la tasa de convergencia es demasiado alta, hay una creciente probabilidad de que el AG converja prematuramente a una solución incorrecta (sub-óptima). Además de proveer presión selectiva, los esquemas de se-

lección deberían también preservar la diversidad de la población, ya que esto ayuda a evitar la convergencia prematura.

Comúnmente se pueden distinguir dos tipos de esquemas de selección, la ***selección proporcional*** y la ***selección basada en el orden***. La selección proporcional escoge a los individuos basada en sus valores de aptitud relativos a la aptitud de los otros individuos en la población. La selección basada en el orden elige a los individuos no por sus valores de aptitud en bruto, sino por la posición relativa de los individuos en un ranking de aptitud de la población, de manera tal que la presión selectiva es independiente de la distribución de aptitud de dicha población.

También es posible usar una función de escala para redistribuir el rango de aptitud de la población de modo de adaptar la presión selectiva. Por ejemplo, si todas las soluciones tienen sus valores de aptitud en el rango [999,1000], la probabilidad de seleccionar un mejor individuo por sobre los otros usando un método proporcional no será grande. Si se traslada la aptitud de cada individuo al rango [0,1] equitativamente, la probabilidad de seleccionar buenos individuos en lugar de malos individuos se incrementará.

La ***selección por ruleta*** es la más clásica y utilizada de las selecciones proporcionales. Simplemente asigna a cada solución un sector de la ruleta cuyo tamaño (determinado por el ángulo que sustenta) es proporcional a la medida de aptitud apropiada (usualmente provista por una función de escala de algún tipo). Entonces se elige una posición al azar sobre la ruleta, y por consiguiente a la solución a la cual esa posición fue asignada (como si se hubiera girado la ruleta).

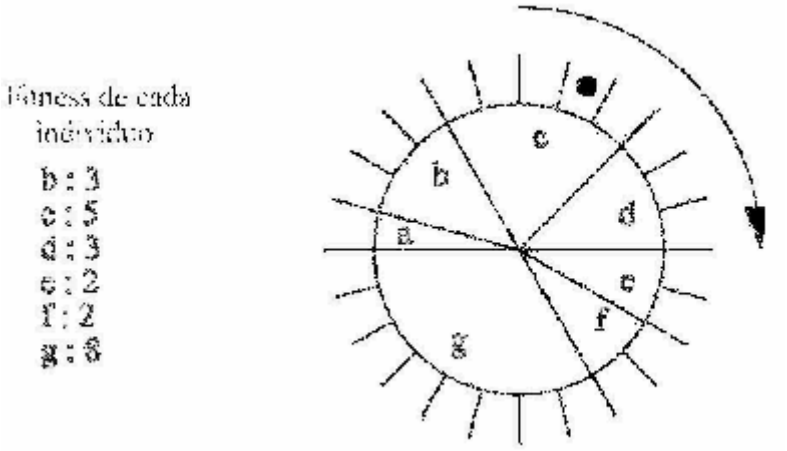


Figura 4.1 Ilustración del método de selección por ruleta, donde se aprecia que el área correspondiente a cada individuo es proporcional a su medida de aptitud. En este ejemplo, se selecciona al individuo *c*.

La ***selección por torneo***, basada en el orden, constituye un esquema alternativo, probablemente uno de los más razonables para utilizar por ser robusto y relativamente sencillo. Muchas variantes existen, pero el mecanismo básico es hacer competir a los cromosomas en grupos de k miembros elegidos al azar (normalmente compiten en parejas) en un tor-

neo del que resultarán ganadores aquellos que tengan valores de aptitud más altos. Eligiendo a los mejores miembros de un torneo se produce una presión selectiva relativamente fuerte, por lo tanto se elige generalmente al mejor con una probabilidad p. Típicamente el tamaño (k) de cada grupo del torneo es 2, y la probabilidad (p) de escoger al mejor en cada competición del torneo es alrededor de 0,7. La presión selectiva puede ajustarse al cambiar k y p. Claramente la selección por torneo tal como fue descripta no se ve afectada por los valores absolutos de aptitud, y en efecto depende solo del rango (posición) de una solución cualquiera en la población.

4.6. Recombinación (Reproducción)

La recombinación es el proceso de tomar dos soluciones padres y producir un hijo a partir de ellos. El operador de reproducción más común es la ***cruza de N-puntos***, que toma dos cromosomas, los alinea y los corta en $N + 1$ segmentos usando N puntos de corte. Se produce un hijo al elegir alternativamente un segmento de uno u otro padre. Se pueden producir dos hijos diferentes dependiendo de si el primer segmento se toma de la madre o del padre.

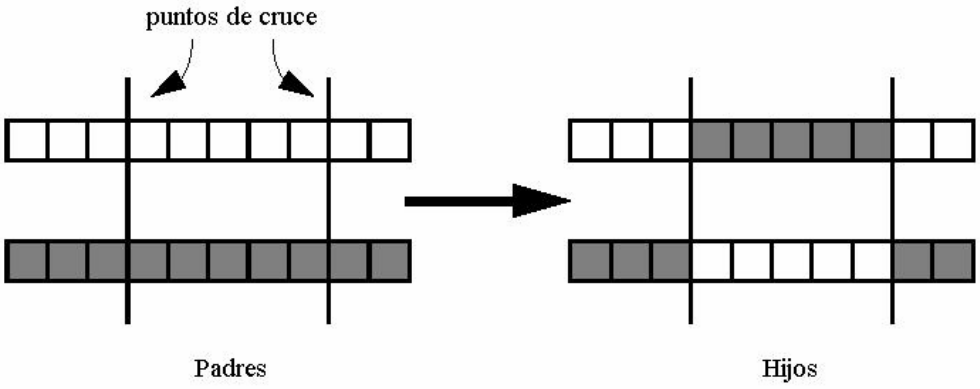


Figura 4.2 Ejemplo de cruza de 2 puntos

Originalmente, los AG utilizaban cruza de 1 punto, el cual cortaba dos cromosomas en un punto y empalma las mitades para crear los nuevos individuos. Pero con esta técnica, la cabeza y la cola de un cromosoma no pueden pasarse juntas a la descendencia. Si tanto la cabeza como la cola contienen buena información genética, ninguno de los descendientes heredará estas dos características. Al usar la cruza de 2 o más puntos se evita esta desventaja, y entonces se la considera generalmente una técnica mejor. En general, los genes que se encuentran próximos en un cromosoma tienen más chances de pasar juntos a la descendencia obtenida por cruza de *N*-puntos. Esto conduce a una correlación no deseada entre los genes cercanos entre sí. En consecuencia, la eficiencia de una cruza de *N*-puntos depende de la posición de los genes en el cromosoma. En una representación genética, los

genes que codifican características de la solución dependientes deberían estar próximos. Para evitar todos estos problemas sobre la posición de los genes (***locus***), una alternativa es usar una cruza uniforme como operador de recombinación.

En la ***cruza uniforme***, cada gen en la descendencia se crea copiando el gen correspondiente de uno de los padres, elegido de acuerdo a una máscara de cruza generada al azar, de la misma longitud de los cromosomas. Donde hay un 1 en la máscara, el gen se copia del primer padre, y donde hay un 0 el gen se copia del segundo padre. Se genera una nueva máscara de cruza aleatoriamente para cada par de padres. La descendencia entonces contiene una mezcla de genes de cada padre. El número de puntos de cruza efectivos no es fijo, pero el promedio será L/2 (donde L es la longitud del cromosoma).

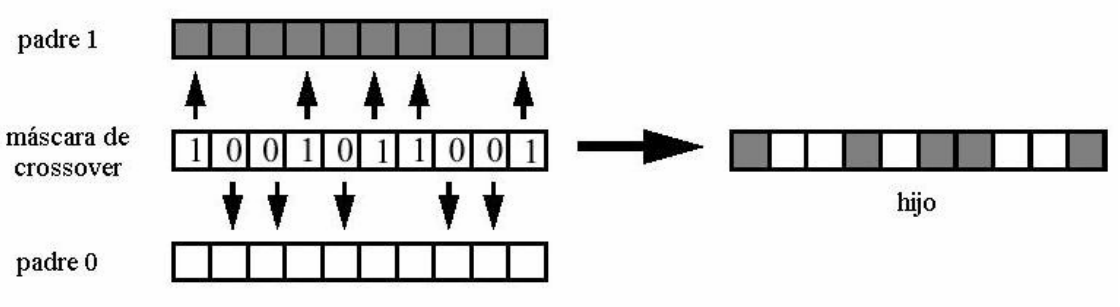


Figura 4.3 Ejemplo de cruza uniforme

4.7. Mutación

La mutación es un operador de búsqueda simple consistente en cambios aleatorios en los valores de los genes en un cromosoma. Si asumimos que la recombinación explora las soluciones actuales para encontrar mejores, la mutación ayudaría entonces a explorar la totalidad del espacio de búsqueda. Usualmente se la considera como un operador de fondo para mantener la diversidad genética en la población, permitiendo asegurar la ***ergodicidad*** (se dice que un espacio de búsqueda es ergódico si hay una probabilidad distinta de cero de generar cualquier solución partiendo de cualquier estado de la población).

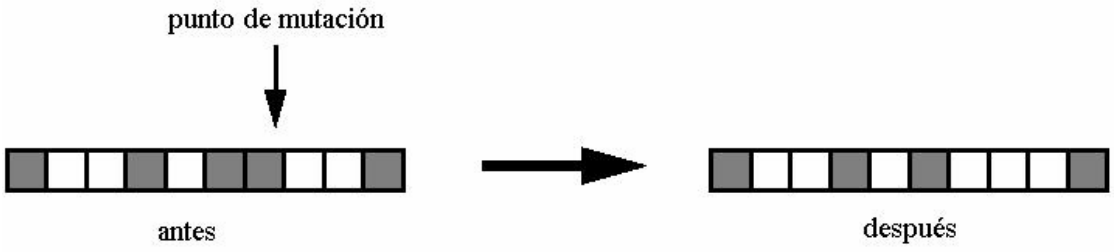


Figura 4.4 Ejemplo de mutación

Hay muchas formas distintas de mutación para las distintas clases de representación. Para la representación binaria, una mutación sencilla puede consistir en invertir el valor de cada gen con una probabilidad pequeña. La probabilidad tomada usualmente es alrededor de 1/L (donde L es la longitud del cromosoma).

4.8. Reemplazo

Una vez que se produce la descendencia, un método debe determinar cuáles de los miembros actuales de la población, si alguno, deberían ser reemplazados por las nuevas soluciones. Básicamente hay dos clases de métodos para el mantenimiento de la población:

a) ***Actualización generacional***: El esquema básico de la actualización generacional consiste en producir *N* hijos de una población de tamaño *N* para formar la población del siguiente paso (generación), y esta nueva población reemplaza completamente a la generación de padres. Claramente esta clase de actualización implica que un individuo puede reproducirse solamente con individuos de su misma generación. Otras formas derivadas de la actualización generacional también se utilizan, como la actualización (λ,μ) o la actualización $(\lambda+\mu)$. Esta vez, a partir de una población de tamaño μ, se produce una camada de hijos de tamaño $\lambda \geq \mu$. Entonces los mejores μ individuos de la población de hijos (para (λ,μ)), o de la población combinada de padres e hijos (para $(\lambda+\mu)$), forman la siguiente generación.

b) ***Actualización gradual***: En la actualización gradual, los nuevos individuos se insertan en la población tan pronto como se crean, en oposición a la actualización generacional donde una nueva generación completa se produce en cada paso. La inserción de un nuevo individuo generalmente necesita el reemplazo de otro miembro de la población. El individuo a borrarse puede elegirse como el peor miembro de la población (lo que conduce a una presión selectiva muy alta), o como el miembro más antiguo de la población, pero estos métodos son bastante radicales. Generalmente la actualización gradual utiliza un método basado en el orden tanto para la selección como para el reemplazo, usualmente un método por torneo. El reemplazo por torneo es exactamente igual a la selección por torneo, excepto que las peores soluciones se eligen más seguido que las mejores.

Una política de ***elitismo*** puede ser una buena elección. El elitismo consiste en nunca reemplazar a los mejores individuos en la población con soluciones inferiores, por lo tanto la mejor solución siempre está disponible para reproducción. Pero el elitismo puede fomentar la convergencia prematura, haciéndole difícil a la solución escapar de un óptimo local.

Evitar la ***duplicación*** es algo que también se podría buscar, ya que mantener más de una copia de un individuo en la solución actual parecería un derroche. Los duplicados necesariamente reducen la diversidad de la población, pero además pueden distorsionar el modo en que los individuos son seleccionados. Aunque el elitismo y la no-duplicación son polí-

ticas comunes en la actualización gradual, también se pueden perfectamente aplicar en los esquemas de actualización generacional.

4.9. Convergencia

Muchas veces la evolución se asemeja a un proceso sin fin. No se puede esperar que los AG se detengan espontáneamente en la solución óptima global por arte de magia. Los AG ni siquiera garantizan encontrar dicha solución. Por lo tanto la evolución tiene que ser detenida en algún punto de acuerdo a algún criterio preestablecido. De hecho hay muchas formas de decidir cuándo parar el algoritmo. La manera más simple es detener la evolución luego de un número fijo de iteraciones. Una solución mejor consiste en continuar las iteraciones en tanto las mejoras sigan siendo apreciables. Finalmente se puede también esperar hasta que la mayoría o todos los miembros de la población son similares o idénticos, es decir cuando la población ha convergido.

La convergencia es simplemente una pérdida de diversidad, debido a la presión selectiva. En ausencia de operadores genéticos, la selección tenderá a reproducir los mejores miembros de la población. La recombinación usada por sí sola, reorganizando el material genético, tiende a oponerse a la convergencia. Sin embargo la recombinación usualmente no crea nuevo material genético en la población, solo los operadores de mutación son realmente capaces de mantener una buena distribución genética.

La ***convergencia prematura*** es una falla común de los AG, normalmente significa que la solución converge a una solución inaceptablemente pobre (cualquiera menos la solución global). Para superarla, se pueden desarrollar varias técnicas, como incrementar la tasa de mutación, o mejor aún, usando una tasa de mutación adaptativa. Típicamente, la tasa de mutación debería ser más alta sobre el final de la corrida para evitar la pérdida de diversidad al (re)introducir material genético (perdido) [Davis, 1991]. Un tamaño de población mayor o una presión selectiva menor pueden ayudar a evitar la convergencia prematura. Al intentar solucionar la convergencia prematura se puede provocar que el algoritmo se comporte como una búsqueda aleatoria, si es que se usa una presión selectiva muy baja o una tasa de mutación demasiado elevada.

Por otro lado, cuando el algoritmo toma mucho tiempo para converger hacia el óptimo, se habla de ***finalización lenta***. La finalización lenta se puede resolver por medio de un incremento en la presión selectiva. En conclusión, solo un buen compromiso entre presión selectiva, tamaño de población, tasa de mutación y demás parámetros, puede conducir a un algoritmo adecuado que encuentre buenas soluciones en un lapso de tiempo relativamente corto.

4.10. Análisis de la población

Una característica única de los AG es que proveen como resultado no sólo la solución óptima, sino también una lista de soluciones. La población final en su totalidad son las mejores soluciones del problema. Cuando se traza el progreso de un AG a través de sus generaciones, el costo promedio y el costo mínimo de los organismos se utiliza como medida de la aptitud total de la población. La historia de un AG se presenta usualmente como un gráfico mostrando generación versus costo promedio y costo mínimo. También suele ser útil obtener el seguimiento de costo máximo y varianza del costo

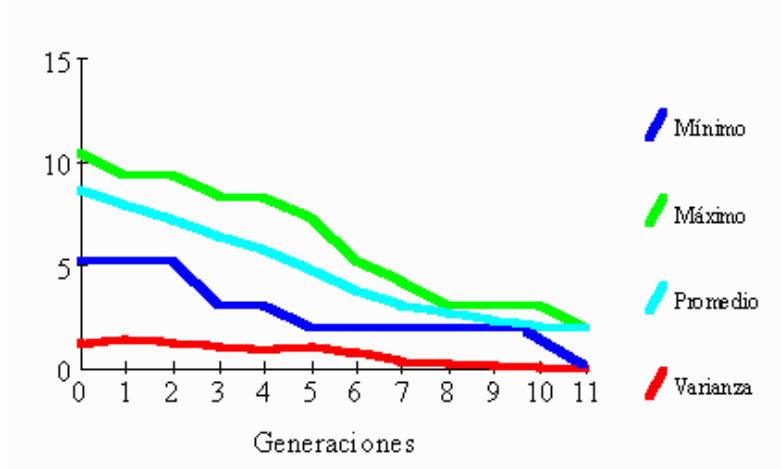


Figura 4.5 Ejemplo del progreso del AG a través de las sucesivas generaciones, expresado como el promedio, el mínimo, el máximo y la varianza de la función de costo.

5. Diseño evolutivo de arquitecturas neuronales

El diseño de una arquitectura óptima (o cuasi-óptima) para una red neuronal puede verse como un problema de búsqueda en el espacio de arquitecturas donde cada punto representa una arquitectura distinta. Dado un criterio de performance acerca de las arquitecturas, como por ejemplo el error de entrenamiento más bajo, la complejidad de red más baja, etc., el nivel de performance de todas las arquitecturas formará una superficie discreta en el espacio. Obtener el diseño de arquitectura óptimo es equivalente a encontrar el punto más alto sobre esta superficie. Hay muchas características sobre dicha superficie que hacen que los AG sean buenos candidatos para la búsqueda, como por ejemplo [Yao, 1999]:

- La superficie es infinitamente grande ya que el número de nodos y conexiones posibles es ilimitado.
- La superficie es no diferenciable porque los cambios en el número de nodos o conexiones son discretos y pueden tener un efecto discontinuo sobre la performance de las redes neuronales.
- La superficie es compleja y ruidosa puesto que el mapeo desde una arquitectura hacia su performance es indirecto y dependiente del método de evaluación utilizado.
- La superficie es engañosa ya que arquitecturas similares pueden tener una performance bastante diferente.
- La superficie es multimodal porque diferentes arquitecturas pueden tener performances similares.

A continuación se brinda una descripción general del proceso evolutivo en la sección 5.1. Luego se presenta las cuestiones referidas a la representación del genotipo en 5.2, la estructura del fenotipo en 5.3, la evaluación del fenotipo en 5.4, las variables sometidas a la evolución en 5.5, el dominio de aplicación en 5.6, y sobre el algoritmo híbrido en 5.7.

5.1. Descripción del proceso evolutivo

El proceso clave en el enfoque evolutivo del diseño de topologías se ve en la figura 5.1.

En el caso más general, un genotipo puede pensarse como un arreglo (una cadena) de genes, donde cada gen toma valores de un dominio definido adecuadamente (también conocidos como alelos). Cada genotipo codifica un fenotipo o solución candidata en el dominio de interés – en nuestro caso una clase de arquitectura neuronal. Tales codificaciones podrían emplear genes que toman valores numéricos para representar unos pocos parámetros o estructuras complejas de símbolos que se transforman en fenotipos (en este caso, redes neuronales) por medio del proceso de *decodificación* apropiado. Este proceso puede ser extremadamente simple o bastante complejo. Las redes neuronales resultantes (los fenotipos) pueden también equiparse con algoritmos de aprendizaje que las entrenen usando el estímulo del entorno o simplemente ser evaluadas en la tarea dada (asumiendo que los pesos de la red se determinan también por el mecanismo de codificación / decodi-

ficación). Esta evaluación de un fenotipo determina la aptitud del genotipo correspondiente.

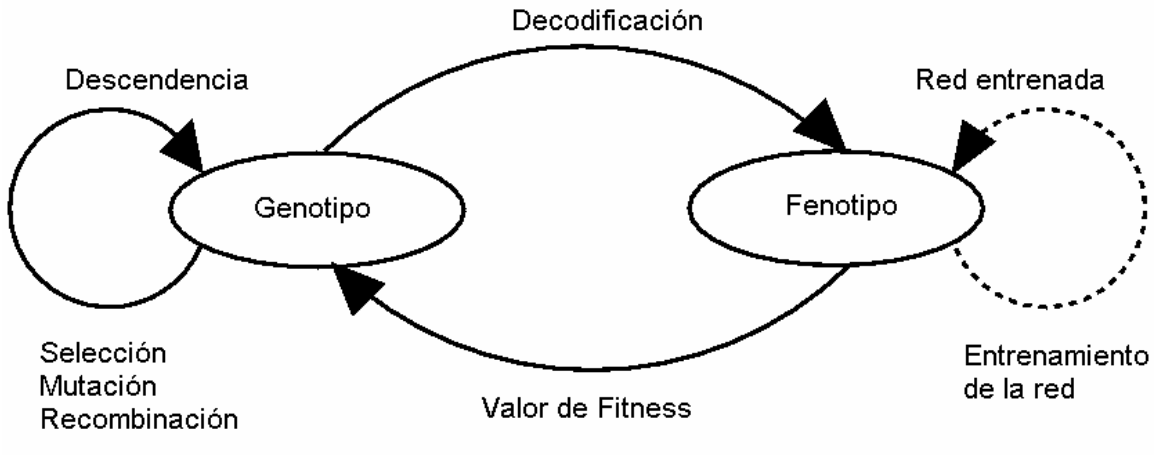


Figura 5.1 Proceso de diseño evolutivo de arquitecturas neuronales

El procedimiento evolutivo trabaja en una población de tales genotipos, preferentemente seleccionando genotipos que codifican a fenotipos de aptitud alta, y reproduciéndolos. Los operadores genéticos tales como la mutación, la cruza, etc., se utilizan para introducir variedad dentro de la población y probar variantes de soluciones candidatas representadas en la población actual. Así, sobre varias generaciones, la población gradualmente evolucionará hacia genotipos que corresponden a fenotipos de aptitud alta.

5.2. Representación del genotipo

La pregunta de cómo se representa una arquitectura neuronal en un genotipo es crítica para el funcionamiento de un sistema de diseño evolutivo de redes neuronales. La representación o codificación usada no solo determina qué clases de arquitecturas pueden posiblemente evolucionar, sino que también restringe el proceso de decodificación. Por ejemplo, si el problema a mano requiere el descubrimiento de redes neuronales con estructura recurrente, para asegurar una probabilidad distinta de cero de tener éxito, el esquema de codificación debe ser lo suficientemente expresivo como para describir arquitecturas neuronales recurrentes, y el mecanismo de decodificación debe ser capaz de transformar tal descripción en la red recurrente apropiada (fenotipo). Puede existir un amplio rango de opciones (con varios grados de eficiencia, robustez, etc.) tanto para los esquemas de codificación como para los esquemas de decodificación.

En forma general, estos esquemas pueden ser clasificados en dos categorías basándose en el esfuerzo de decodificación que se necesita [Karthik, 1998]:

a) ***Codificación directa o fuerte***: requiere poco esfuerzo para decodificar, es decir, la transformación del genotipo en un fenotipo es bastante trivial. Un ejemplo de tal codificación es una matriz de conexión que precisa y directamente especifica la arquitectura de la red neuronal correspondiente. El esquema de codificación directo es bastante sencillo de implementar. Es muy indicado para la búsqueda precisa y afinada de arquitecturas de redes neuronales compactas, ya que una conexión simple puede agregarse o quitarse fácilmente de la red. Un problema potencial de esta codificación es la escalabilidad. Una red neuronal grande requeriría una matriz muy grande y así incrementaría el tiempo de computación de la evolución.

b) ***Codificación indirecta o débil***: requiere un esfuerzo de decodificación considerable para la construcción de un fenotipo. De manera de reducir la longitud de la representación genética, estos esquemas sólo codifican algunas características de la arquitectura en el cromosoma. Los detalles de cada conexión en una red neuronal están, o predefinidos de acuerdo a un conocimiento previo, o especificados por un conjunto de reglas de desarrollo determinísticas. Las codificaciones indirectas caen dentro de dos categorías:

   i. ***Representación paramétrica***: se especifican parámetros tales como el número de capas ocultas, el número de nodos ocultos por capa, el número de conexiones entre dos capas, etc.

   ii. ***Representación de reglas de desarrollo***: descriptas por una ecuación recursiva o una regla de generación similar a una regla de producción en un sistema de producción donde existe una parte izquierda o precedente y una parte derecha o consecuente.

   Los esquemas de codificación indirectos pueden producir una representación genética más compacta, pero pueden no ser muy buenos para encontrar una red neuronal compacta con buena habilidad para generalizar.

Actualmente existen varias implementaciones para la representación del genotipo, pero todas ellas se basan en dos enfoques principales: el método de Miller, Todd y Hedge, y el método de Kitano. Seleccionamos esto métodos porque son bien conocidos y representativos de estado de la cuestión en la evolución de topologías de red neuronal. Los métodos restantes pueden derivarse fácilmente a partir de ellos.

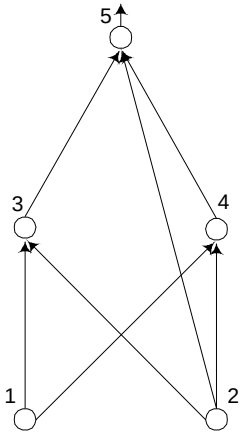
5.2.1. Miller, Todd y Hedge

En este primer enfoque [Miller, Todd y Hedge, 1989], todos los detalles, es decir, cada conexión y nodo de una arquitectura se especifican dentro del cromosoma. Este esquema pertenece a la clase de representación directa.

Cada conexión de una arquitectura se especifica directamente por su representación binaria. Por ejemplo, una matriz $N \times N$ $\boldsymbol{C = (c_{ij})_{NxN}}$ puede representar una arquitectura de red

neuronal con N nodos, donde c_{ij} indica la presencia o ausencia de la conexión desde el nodo i hacia el nodo j. Se puede usar $c_{ij} = 1$ para indicar una conexión y $c_{ij} = 0$ para indicar que no hay conexión. De hecho, c_{ij} puede ser un valor real que represente el peso de la conexión desde el nodo i hacia el nodo j, de manera tal que la arquitectura y los pesos de las conexiones puedan evolucionar simultáneamente.

Cada matriz C tiene un mapeo uno-a-uno directo con la arquitectura neuronal correspondiente. La cadena binaria representando una arquitectura es la concatenación de filas (o columnas) de la matriz. Se pueden incorporar fácilmente restricciones sobre las arquitecturas siendo exploradas dentro de tal esquema de representación, al imponer restricciones en la matriz, por ejemplo una red neuronal feedforward tendrá entradas distintas de cero sólo en el triángulo superior derecho de la matriz. Las figuras 5.2 y 5.3 dan dos ejemplos del esquema de codificación directo de arquitecturas de redes neuronales. Es obvio que tal esquema de codificación puede manejar tanto redes feedforward como recurrentes.



(a)

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(b)

0010 101 01 1

(c)

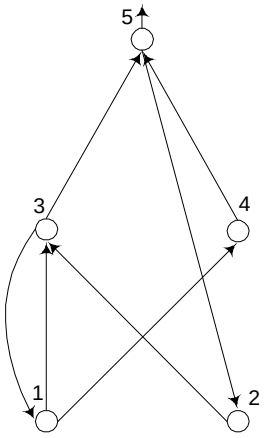
Figura 5.2 Un ejemplo de la codificación directa de una red neuronal feedforward. (**a**), (**b**) y (**c**) muestran la arquitectura, su matriz de conectividad y su cadena binaria de representación, respectivamente. Ya que solo se consideran arquitecturas feedforward, la cadena binaria de representación sólo necesita considerar el triángulo superior derecho de la matriz.

La figura 5.2(a) muestra una red neuronal feedforward con dos entradas y una salida. Su matriz de conectividad esta dada por la figura 5.2(b), donde c_{ij} indica la presencia o ausencia de una conexión desde el nodo i hacia el nodo j. Por ejemplo, la primer fila indica conexiones desde el nodo 1 hacia todos los nodos restantes. Las dos primeras columnas son 0's porque no hay conexión desde el nodo 1 hacia sí mismo ni hay conexión hacia el nodo 2. Sin embargo, el nodo 1 se conecta con los nodos 3 y 4. Por ésto, las columnas 3 y 4 tienen 1's. La conversión de la matriz de conectividad en un cromosoma es un proceso directo. Se pueden concatenar todas las filas (o columnas) y obtener:

00110 00101 00001 0001 0000

Ya que la red neuronal es feedforward, solo se necesita representar el triángulo superior de la matriz de conectividad de manera de reducir la longitud del cromosoma. El cromosoma reducido se muestra en la figura 5.2(c). Un algoritmo híbrido se puede emplear entonces para una población de tales cromosomas. Para evaluar la aptitud de cada cromosoma, se necesita convertir el cromosoma nuevamente en una red neuronal, inicializar sus pesos iniciales aleatoriamente, y entrenarla. El error de entrenamiento se utilizará para medir la aptitud. Un algoritmo híbrido es capaz de explorar todas las conectividades posibles.

La figura 5.3 muestra una red neuronal recurrente. Su representación es básicamente la misma que para la red feedforward. La única diferencia es que no es posible la reducción en la longitud del cromosoma si se quiere explorar el espacio de conectividad en su totalidad. El algoritmo genético utilizado para evolucionar las redes neuronales recurrentes puede ser el mismo que el usado para las redes feedforward.



$$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

00110 00100 10001 00001 01000

(a) (b) (c)

Figura 5.3 Un ejemplo de la codificación directa de una red neuronal recurrente. (**a**), (**b**) y (**c**) muestran la arquitectura, su matriz de conectividad y su cadena binaria de representación, respectivamente.

5.2.2. Kitano

En este enfoque [Kitano, 1990], solo se codifican los parámetros más importantes de una arquitectura, como el número de capas ocultas y nodos ocultos en cada capa. Los otros detalles acerca de la arquitectura se dejan para que el proceso de entrenamiento decida. Este esquema pertenece a la clase de representación indirecta. El enfoque utiliza un método de codificación de gramática basada en reglas de desarrollo.

Una regla de desarrollo usualmente se describe mediante una ecuación recursiva, o una regla de generación similar a una regla de producción en un sistema de producción, con

un lado izquierdo (precedente) y un lado derecho (consecuente). El patrón de conectividad de una arquitectura en la forma de una matriz se construye a partir de una base (una matriz de un elemento), aplicando repetitivamente reglas de desarrollo a los elementos no terminales en la matriz actual hasta que la matriz contenga solo elementos terminales, los cuales indican la presencia o ausencia de una conexión, esto es, hasta que se especifica completamente un patrón de conectividad.

$$S \to \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

$$A \to \begin{pmatrix} a & a \\ a & a \end{pmatrix} \quad B \to \begin{pmatrix} i & i \\ i & a \end{pmatrix} \quad C \to \begin{pmatrix} i & a \\ a & c \end{pmatrix} \quad D \to \begin{pmatrix} a & e \\ a & e \end{pmatrix} \quad \cdots$$

$$a \to \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \quad c \to \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \quad e \to \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \quad i \to \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \cdots$$

Figura 5.4 Ejemplos de algunas reglas de desarrollo usadas para construir una matriz de conectividad. S es el elemento (estado) inicial.

Se dan algunos ejemplos de reglas de desarrollo en la figura 5.4. Cada regla de desarrollo consiste en una parte izquierda, que es un elemento no terminal, y en una parte derecha, que es una matriz de 2x2 con elementos terminales o no terminales. Un paso típico en la construcción de una matriz de conexión es encontrar las reglas cuyas partes izquierdas aparezcan en la matriz actual y reemplazar todas las apariciones con la parte derecha respectiva. Por ejemplo, dado un conjunto de reglas como las descriptas en la figura 5.4, donde S es el símbolo inicial (estado), un paso de aplicación de estas reglas producirá la matriz:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

al reemplazar S con $\begin{pmatrix} A & B \\ C & D \end{pmatrix}$. Si aplicamos estas reglas otra vez, podremos generar la siguiente matriz:

$$\begin{pmatrix} a & a & i & i \\ a & a & i & a \\ i & a & a & e \\ a & c & a & e \end{pmatrix}$$

reemplazando A con $\begin{pmatrix} a & a \\ a & a \end{pmatrix}$, B con $\begin{pmatrix} i & i \\ i & a \end{pmatrix}$, C con $\begin{pmatrix} i & a \\ i & c \end{pmatrix}$ y D con $\begin{pmatrix} a & e \\ a & e \end{pmatrix}$

Otro paso al aplicar estas reglas nos conduciría a la siguiente matriz:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Cuando reemplazamos *a* con $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$, *i* con $\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$, *c* con $\begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$ y *e* con $\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$

Ya que la última matriz consiste en 1's y 0's solamente (es decir, solo elementos terminales) no habrá más aplicaciones de las reglas de desarrollo. Esta matriz será entonces la matriz de conexión de una red neuronal.

La figura 5.5 resume los tres pasos previos de reescritura y la red neuronal final generada de acuerdo a la figura 5.5(d). Notar que los nodos 2, 4 y 6 no aparecen en la red neuronal porque no se conectan con ningún otro nodo.

El ejemplo descripto por las figuras 5.4 y 5.5 ilustran como una red neuronal puede ser definida dado un conjunto de reglas. La pregunta ahora es: ¿Cómo conseguir tales reglas para construir una red neuronal? Una solución es evolucionarlas. Se puede codificar el conjunto entero de reglas como un grupo individuos, o codificar cada regla en un solo individuo. Cada regla puede ser representada por 4 posiciones de alelos correspondientes a los 4 elementos en la parte derecha de la regla. La parte izquierda se puede representar implícitamente por la posición de la regla en el cromosoma. Cada posición en un cromosoma puede tomar uno de los muchos valores distintos, dependiendo en cuántos elementos no terminales (símbolos) se usan en el conjunto de reglas. Por ejemplo, los símbolos no terminales pueden ir desde la "A" hasta la "Z" y de la "a" hasta la "p". Las 16 reglas con "a" hasta "p" en la parte izquierda y las matrices de 2x2 con solo 1's y 0's en la parte derecha están predefinidas y no participan en la evolución de manera de garantizar que se puedan alcanzar distintos patrones de conectividad. Por haber 26 reglas diferentes, cuya parte izquierda es "A", "B", ..., "Z" respectivamente, un cromosoma codificándolos a todos ellos necesitaría 26 x 4 = 104 alelos, cuatro por regla. La parte izquierda de una regla se determina implícitamente por su posición en el cromosoma. Por ejemplo el conjunto de reglas en la figura 5.4 se puede representar por el siguiente cromosoma:

ABCDaaaaiiiaiaacaeae ....

donde los primeros cuatro elementos indican la parte derecha de la regla "*S*", los segundos cuatro indican la parte derecha de la regla "*A*", etc.

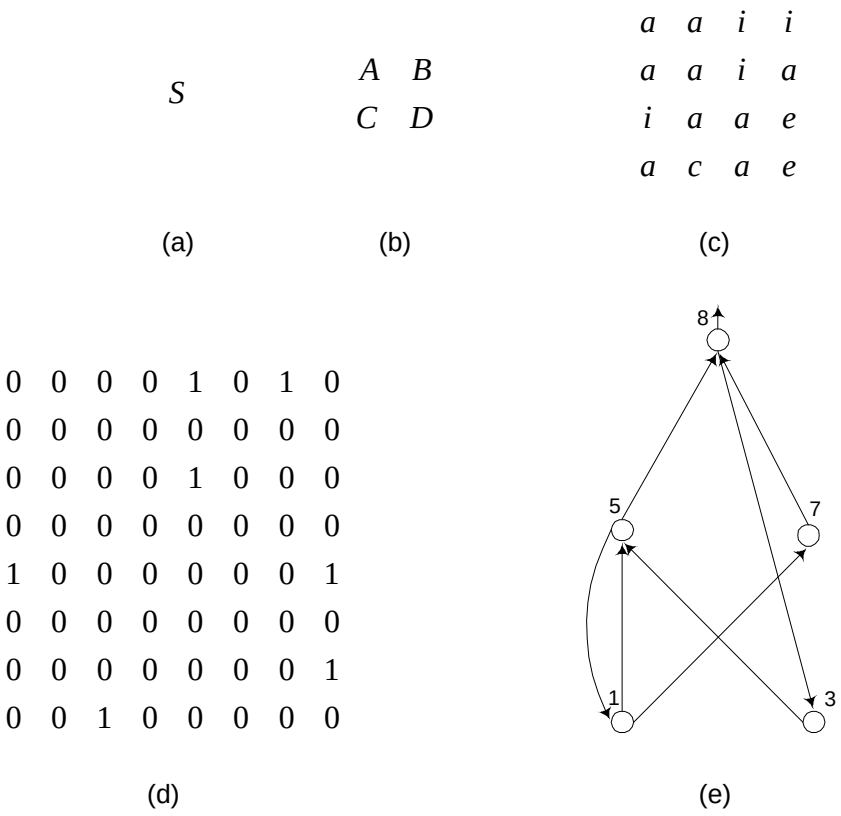


Figura 5.5 Desarrollo de una arquitectura de red neuronal usando las reglas dadas en la figura 5.4. (**a**) El estado inicial; (**b**) Paso 1; (**c**) Paso 2; (**d**) Paso 3 cuando todas las entradas en la matriz son elementos terminales, es decir, 1's o 0's; (**e**) La arquitectura. Los nodos en la arquitectura se numeran del 1 al 8. No se muestran los nodos aislados.

5.3. Estructura del fenotipo

El éxito de una arquitectura neuronal en resolver un problema en particular (o clase de problemas) depende críticamente de la topología de la red (o la estructura del fenotipo en nuestro sistema de diseño evolutivo). Por ejemplo, una red neuronal puramente feedforward es incapaz de descubrir o responder a dependencias temporales en su ambiente; se necesita una red neuronal recurrente para esta tarea. Similarmente, decisiones con límites no lineales no pueden ser descubiertas por redes de 1 capa (por ejemplo perceptrones); se requieren redes multicapa. Es por ello que resulta útil clasificar el trabajo en diseño evolutivo de redes neuronales basándose en las clases de topologías neuronales que se evolu-

cionan. Por ejemplo, como se mencionó anteriormente, las topologías de las redes neuronales pueden ser divididas en dos tipos generales: redes *feedforward* y redes *recurrentes*. Cada uno de estos dos tipos básicos puede ser a su vez clasificado en redes que son – *multicapa, conectadas al azar, conectadas localmente, conectadas completamente, irregulares, modulares, jerárquicas*, etc.

La elección de la clase objetivo de la estructura de red determina la elección de la representación genética. De esta forma, se puede ganar mucho en términos de eficiencia de búsqueda en base al *conocimiento a priori* acerca de las propiedades del conjunto solución. En el diseño de las arquitecturas neuronales evolutivas, una forma de hacerlo es eligiendo una clase de topología de red lo suficientemente restringida basándose en el conocimiento del dominio del problema. Sin esta limitación, mucho del esfuerzo de búsqueda puede ser desperdiciado.

5.4. Evaluación del fenotipo

Las redes neuronales son conocidas por su habilidad para manejar entradas ruidosas. Esto significa que la entrada de una red neuronal no tiene por qué coincidir con lo que la red fue entrenada inicialmente para reconocer. Del mismo modo, datos de entrenamiento irrelevantes no deberían afectar adversamente a la red. Esto hace que las redes neuronales resulten útiles para aplicaciones como el reconocimiento de entradas humanas (la escritura manual o el habla) y datos que ocurren naturalmente (predicciones financieras o patrones de comportamiento).

Para poder aplicar una red neuronal en un problema dado, el programador primero debe determinar cómo se deben representar los datos en la red. Por ejemplo, si se usa una red neuronal para el reconocimiento de la escritura manual sobre una plantilla, tanto los datos que muestran dónde el usuario tocó la superficie como la información sobre la presión empleada y la medida de los tiempos pueden servir como entradas de la red.

La eficiencia de una red puede determinarse por muchos factores. La velocidad de entrenamiento de la red (la habilidad de la red para aprender el set de entrenamiento) puede ser una medida útil de la aptitud de una red. Por otra parte, la habilidad de una red para aproximar la salida deseada según el set de entrenamiento utilizado es un factor importante. Usualmente una red neuronal no se entrenará para alcanzar el 100% de perfección para clasificar el set de entrenamiento. Además, la entrada de una red no siempre consistirá en valores presentes en el set de entrenamiento, por lo tanto la capacidad de generalización a veces es el criterio más importante con el que se juzga la performance de una red.

El promedio de las diferencias entre la salida deseada y la actual de una red neuronal se llama aptitud estándar. Esta es la medida que la mayoría de las implementaciones tratan de minimizar [Hubert, 1995]. Existen numerosas variantes de la función de aptitud, como por ejemplo el agregado de una penalidad por clasificaciones incorrectas que se incrementa cuadráticamente por cada salida incorrecta.

Cuando se calcula la performance global de una red, es importante tomar en cuenta cómo se va a utilizar la red. Por ejemplo:

- A menudo, la habilidad que tiene una red para generalizar es más importante que su habilidad para aprender el set de entrenamiento. En este caso la red debería evaluarse con datos que no estuvieron presentes en el set de entrenamiento, y recién entonces se debería calcular la aptitud estándar.

- Si una red está en una situación donde el aprendizaje tomará lugar frecuentemente, o incluso en tiempo real, entonces la velocidad del aprendizaje sería una parte importante la función de aptitud. En la evaluación de performance, por lo tanto, puede ser útil incluir factores como la habilidad para aprender el set de entrenamiento.

En este trabajo nos enfrentamos al primer escenario, donde buscamos encontrar aquellas topologías óptimas que generalicen bien.

5.5. Variables sometidas a la evolución

Las arquitecturas neuronales se especifican típicamente en términos de la topología (o patrón de conectividad), funciones computadas por las neuronas (sigmoidales, umbrales, etc.) y los pesos de las conexiones (o un algoritmo de aprendizaje que ajusta los pesos). Una descripción más completa de una arquitectura neuronal requiere la especificación de estructuras de coordinación, de control y de aprendizaje (entre otras) [Honovar, 1993]. Virtualmente cualquier subconjunto de estas variables es candidato a ser operado por los procesos evolutivos. Por ejemplo, un sistema *A* podría evolucionar tanto la conectividad de la red como los pesos (manteniendo todo lo demás constante), mientras que un sistema *B* podría evolucionar solo la conectividad, dependiendo de una búsqueda local de pesos para la red tal vez más eficiente. El balance tiempo / performance para ambos sistemas, en un problema dado, será diferente, convirtiendo a la elección de variables sujetas a la evolución en un factor extremadamente crítico.

Adicionalmente a la conectividad y a los pesos de la red, uno podría evolucionar un algoritmo de aprendizaje, funciones de control o regulación, las funciones computadas por varias neuronas, la distribución de los distintos tipos de neuronas, la densidad relativa de las conexiones, los parámetros que gobiernan la decodificación de un genotipo en un fenotipo, etc.

5.6. Dominio de aplicación

Gran parte del trabajo en diseño evolutivo de arquitecturas neuronales es manejado por aplicaciones particulares de la ingeniería (diseño de redes para la clasificación de patrones, control, navegación de robots, fusión de sensores, etc.) y/o hipótesis científicas espe-

cíficas (explicaciones para la evolución de una cierta clase de organización modular, o el predominio de estructuras de redes y unidades funcionales particulares en los cerebros de mamíferos a lo largo de las diferentes especies, etc.). En este contexto, intentos de entender y caracterizar la relación entre las propiedades de las clases de problemas de aplicación y aquellas de las redes neuronales utilizadas en sus soluciones son de gran interés teórico y práctico. Por estos motivos, este trabajo se concentra en encontrar topologías óptimas específicas del dominio.

5.7. El algoritmo híbrido

La evolución de las arquitecturas puede progresar de acuerdo al ciclo mostrado a continuación [Yao, 1999]. Este ciclo representa un ***algoritmo híbrido*** pues combina las técnicas de algoritmos genéticos y de redes neuronales, en lo sucesivo se referirá a este proceso como AG/RN. El ciclo se detendrá cuando se encuentre una red neuronal satisfactoria.

1. Decodificar a cada individuo (cromosoma) en la generación actual para generar una arquitectura. Si se utiliza codificación indirecta, más detalles sobre la arquitectura deben ser especificados por alguna regla de desarrollo o proceso de entrenamiento.

2. Entrenar a cada red neuronal con la arquitectura decodificada, por medio de alguna regla / algoritmo predefinido de aprendizaje (algunos de los parámetros de la regla de aprendizaje podrían aprenderse durante el entrenamiento) comenzando con diferentes conjuntos de valores aleatorios para los pesos iniciales.

3. Calcular la aptitud de cada individuo (arquitectura decodificada) de acuerdo al resultado del entrenamiento del paso anterior y otros criterios de performance como la complejidad de la arquitectura.

4. Seleccionar padres de la población basándose en su aptitud.

5. Aplicar los operadores genéticos a los padres y generar la descendencia que formará la próxima generación.

6. Descripción del problema

El entrenamiento de redes neuronales usando algoritmos evolutivos ha sido un tema de interés en inteligencia artificial durante los últimos años. Se han desarrollado métodos bastante eficientes para el entrenamiento de los pesos de las conexiones, como es el caso del algoritmo back-propagation utilizado en el aprendizaje de las redes feedforward. Sin embargo no existe un método definitivo para determinar las arquitecturas neuronales más apropiadas para problemas particulares.

Los métodos en que estamos interesados son aquellos que se utilizan específicamente en el diseño arquitectónico de redes. Queremos hacer el entrenamiento de los pesos de la red mediante métodos de aprendizaje neuronal tradicional (no evolutivo) ya que han demostrado ser muy eficientes para ese propósito.

Los métodos que determinan el diseño de las arquitecturas neuronales, sin embargo, afrontan diversas dificultades, que detallamos a continuación. Se presentan los problemas referidos a la generalización en 6.1, la permutación en 6.2, la evaluación ruidosa de la aptitud en 6.3, y se resumen las cuestiones a resolver en 6.4.

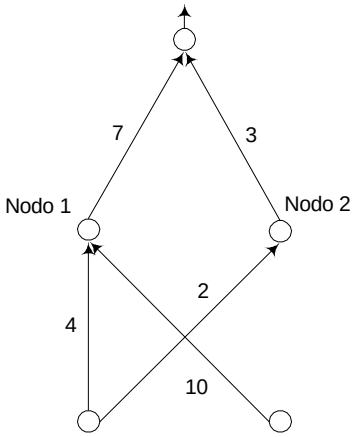
6.1. El problema de la generalización

La topología de una red neuronal, es decir, el número de nodos y la ubicación y el número de conexiones entre ellos, tiene un impacto significativo en la performance de la red y su habilidad para generalizar. La densidad de conexiones en una red neuronal determina su habilidad para almacenar información. Si una red no tiene suficientes conexiones entre nodos, el algoritmo de entrenamiento puede no converger nunca; la red neuronal no es capaz de aproximar la función. Por el otro lado, en una red densamente conectada, puede ocurrir el sobreajuste (overfitting). El sobreajuste es un problema de los modelos estadísticos donde se presentan demasiados parámetros. Esto es una mala situación porque en lugar de aprender a aproximar la función presente en los datos, la red simplemente puede memorizar cada ejemplo de entrenamiento. El ruido en los datos de entrenamiento se aprende entonces como parte de la función, a menudo destruyendo la habilidad de la red para generalizar.

Los métodos actuales para el diseño arquitectónico de redes, en este sentido, carecen de técnicas que permitan controlar la habilidad de aprendizaje y generalización de las redes generadas. No aplican técnicas que, por ejemplo, detengan el aprendizaje una vez que la red comienza a sobreajustar a los datos de entrenamiento. Tampoco contemplan técnicas como la regularización de la complejidad. Estas técnicas de regularización de la complejidad se concentran en la minimización del tamaño de la red manteniendo la buena performance, ya que una red neuronal con tamaño mínimo es menos probable que aprenda el ruido en los datos de entrenamiento y puede así generalizar mejor sobre datos nuevos.

6.2. El problema de la permutación

Un problema que enfrentan las redes neuronales evolutivas es el problema de la permutación. La causa es el mapeo muchos-a-uno desde la representación codificada de una red neuronal hacia la red neuronal real decodificada, porque dos redes que ordenan sus nodos ocultos en forma diferentes tienen distinta representación pero pueden sen funcionalmente equivalentes. Por ejemplo, las redes neuronales mostradas por las figuras 6.1(a) y 6.2(a) son equivalentes, pero tienen distintas representaciones del genotipo, como se ve en las figuras 6.1(b) y 6.2(b) usando el esquema de codificación directo. En general, cualquier permutación de los nodos ocultos producirá redes neuronales funcionalmente equivalentes pero con distintos representaciones del genotipo. Esto también es verdad para los esquemas de codificación indirecta.

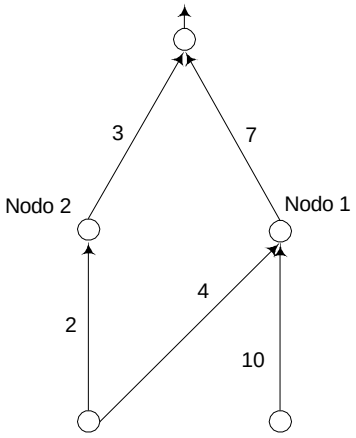


(a)

0100 1010 0010 0000 0111 0011

(b)

Figura 6.1 (**a**) Una red neuronal con sus pesos de conexión; (**b**) Una representación binaria de los pesos, asumiendo que cada peso se representa con 4 bits. Cero significa sin conexión.



(a)

0010 0000 0100 1010 0011 0111

(b)

Figura 6.2 (**a**) Una red neuronal que es equivalente a la dada en la figura 6.1(a); (**b**) Su representación del genotipo bajo el mismo esquema de representación.

Este problema no solo hace a la evolución menos eficiente, sino que también dificulta a los operadores de recombinación la producción de hijos con alta aptitud. El uso de operadores de recombinación como la cruza parece contradecir las ideas básicas detrás de las redes neuronales, porque la cruza trabaja mejor cuando existen "bloques de construcción", pero no queda claro qué es lo que debería ser un bloque de construcción en una red neuronal, ya que las redes neuronales enfatizan la representación distribuida. El conocimiento en una red neuronal se distribuye entre todos los pesos de las conexiones. Recombinar una parte de una red neuronal con otra parte de otra red neuronal probablemente destruya ambas redes.

Los métodos actuales para el diseño arquitectónico de redes, en este sentido, tampoco dejan en claro cómo atenuar el problema de la permutación en el diseño de sus operadores de cruza.

6.3. El problema de la evaluación ruidosa de la aptitud

La evaluación de la aptitud de las arquitecturas de redes neuronales siempre será ruidosa si la evolución de las arquitecturas se separa del entrenamiento de los pesos. La evaluación es ruidosa porque lo que se utiliza para evaluar la aptitud del genotipo es la arquitectura real con pesos (es decir, el fenotipo creado a partir del genotipo a través de reglas de desarrollo), y el mapeo entre fenotipo y genotipo no es uno-a-uno. En la práctica, un modo de evaluar genotipos es transformando un genotipo (una arquitectura codificada) en un fenotipo (una arquitectura completamente especificada) a través de reglas de desarrollo, y entonces entrenar al fenotipo a partir de distintos pesos iniciales de conexión generados al azar. El resultado del entrenamiento se utiliza como parte de la función de aptitud para evaluar el genotipo. En otras palabras, se utiliza la aptitud del fenotipo para estimar la aptitud del genotipo. Existen dos fuentes principales de ruido:

1. La primer fuente es la inicialización de los pesos al azar. Diferentes pesos iniciales aleatorios pueden producir diferentes resultados de entrenamiento. Así, el mismo genotipo puede tener una aptitud bastante distinta debido a los distintos pesos iniciales aleatorios utilizados en el entrenamiento.
2. La segunda fuente es el algoritmo de entrenamiento. Diferentes algoritmos de entrenamiento pueden producir diferentes resultados de entrenamiento aún partiendo del mismo conjunto de pesos iniciales. De aquí la importancia de la función de error utilizada y la capacidad de búsqueda global del algoritmo.

Tal ruido puede engañar a la evolución, porque el hecho de que la aptitud de un fenotipo generado a partir del genotipo G1 sea más alto que aquel generado a partir del genotipo G2 no implica que G1 tenga verdaderamente mejor calidad que G2.

Los métodos actuales para el diseño arquitectónico de redes, en este sentido, también carecen de una técnica para atenuar el problema de la evaluación ruidosa de la aptitud.

6.4. Cuestiones a resolver

El propósito de este trabajo es encontrar las topologías óptimas específicas del dominio que faciliten tanto el aprendizaje como la generalización. Con esta finalidad, se tomará el enfoque evolutivo de diseño de arquitecturas de red neuronal. Dentro de este contexto, las cuestiones que plantean las implementaciones actuales y que queremos resolver son:

1. ¿En qué medida afectan los parámetros del algoritmo back-propagation al aprendizaje de la red?

2. ¿Qué operador de cruza es el más adecuado en lo que respecta al problema de la permutación?

3. ¿Cómo influye la variación de la presión selectiva en la evolución del algoritmo genético?

4. ¿Cómo se puede atenuar el problema de la evaluación ruidosa de la aptitud?

5. ¿De qué forma se puede mejorar la capacidad de generalización de las redes neuronales generadas?

6. ¿Cómo deberían evaluarse a las redes neuronales en lo que respecta a su habilidad de aprendizaje y generalización?

Estas cuestiones son abordadas en el capítulo 7 "solución propuesta", en el que se presenta la manera en que se intenta resolver los problemas recién introducidos. Posteriormente, las cuestiones son tratadas en el capítulo 8 "experimentos", donde se muestran y analizan los resultados experimentales al aplicar dichas soluciones.

7. Solución propuesta

A continuación se presenta la solución propuesta, explicando la manera en que se intenta abordar los problemas introducidos en el capítulo 6. En 7.1 se trata sobre la disminución del ruido en la evaluación de la aptitud, en 7.2 sobre la atenuación de los efectos del problema de la permutación, en 7.3 sobre las mejoras a la capacidad de generalización y finalmente en 7.4 sobre la evaluación del proceso evolutivo. La técnica presentada para la atenuación de los efectos del problema de la permutación es una propuesta novedosa con respecto a aquellas encontradas en el estado del arte, mientras que las otras técnicas no fueron aplicadas particularmente a la generación de redes neuronales por medio de los algoritmos genéticos.

7.1. Disminución del ruido en la evaluación de la aptitud

Como mencionamos anteriormente, el problema la evaluación ruidosa de la aptitud proviene del hecho de que se utiliza la aptitud del fenotipo para estimar la aptitud del genotipo. De manera de reducir este ruido, se entrenará a cada arquitectura muchas veces partiendo de diferentes pesos iniciales aleatorios. Tomamos entonces el mejor resultado para estimar la aptitud del genotipo, y guardamos la semilla generadora de números aleatorios utilizada para poder reproducir este resultado. Este método incrementa el tiempo de cómputo para la evaluación de la aptitud, así que se analizará en la experimentación en que medida se atenúa el ruido en función de número de repeticiones del entrenamiento.

En general, el esquema de codificación directa es mejor para el ajuste fino y preciso de una arquitectura para una aplicación específica, debido a que a partir de la decodificación del cromosoma obtenemos directamente la arquitectura de red neuronal. La codificación dentro de cromosomas de reglas de desarrollo para generar redes neuronales (lo cual constituye uno de los mejores esquemas de codificación indirecta) puede producir representaciones muy compactas, pero sufre los problemas del ruido causados por la transformación de las reglas de desarrollo en arquitecturas de red neuronal. Para evitar este ruido adicional introducido por las reglas de desarrollo, se utilizará el esquema de codificación directa en el algoritmo.

7.2. Atenuación de los efectos del problema de la permutación

El problema de la permutación surge del hecho de que dos representaciones distintas de red neuronal pueden resultar funcionalmente equivalentes, haciendo menos eficiente a los operadores de reproducción. Para atenuar los efectos del problema de la permutación, implementaremos una cruza de fenotipos, es decir, una cruza que trabaja sobre redes neuronales en lugar de hacerlo sobre las cadenas de genes que forman la población (como describimos en 4.6).

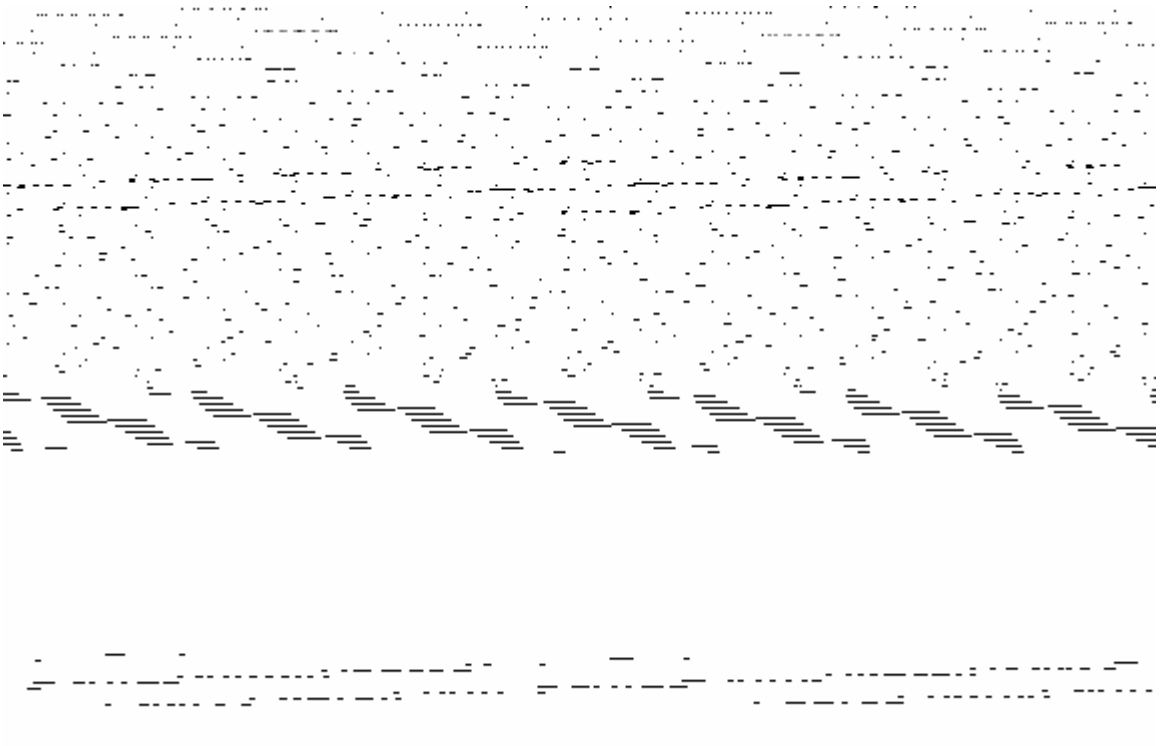


Figura 7.1 Cruza entre la entrada y la primer capa oculta.

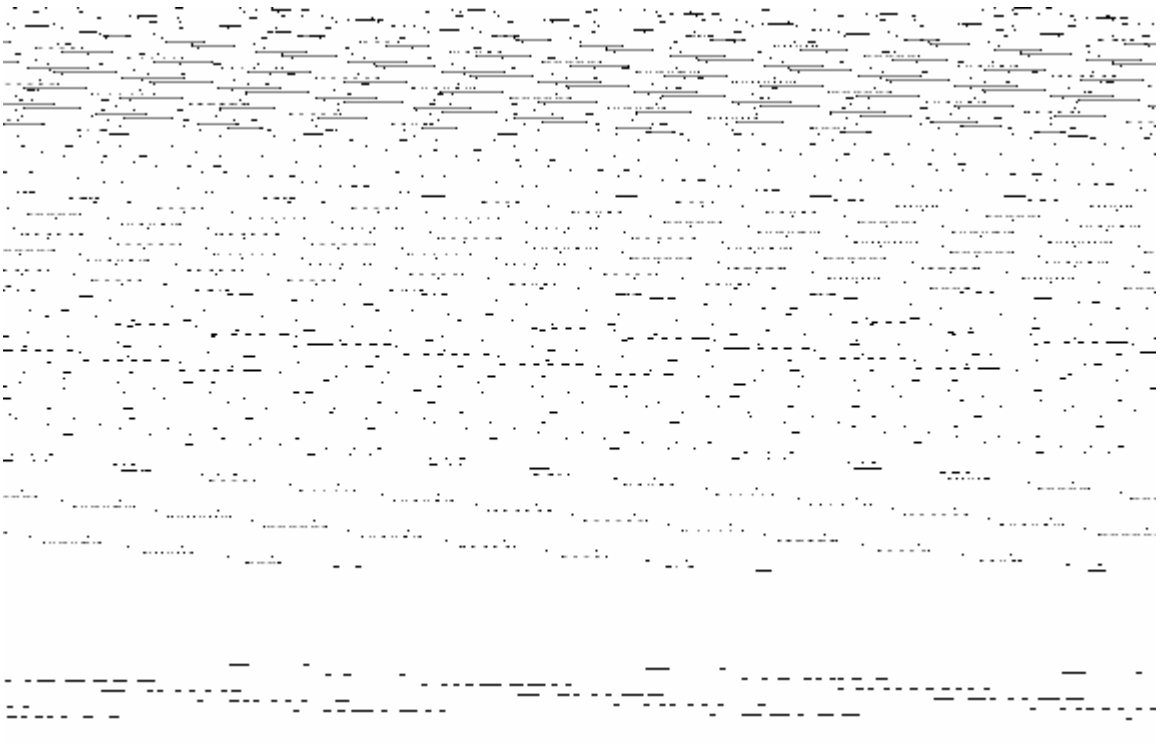


Figura 7.2 Cruza entre capas ocultas.

En cada capa oculta de cada padre hacemos un corte simple separando los nodos de esta capa en dos partes: una a la izquierda y la otra a la derecha del corte. Usamos cortes aleatorios, pero restringimos el corte de manera tal que al menos un cuarto de todos los nodos en la capa que se corta quede en la parte más pequeña.

Primero producimos al primer hijo con la parte izquierda de la madre y la parte derecha del padre. Después se combinan las partes opuestas para hacer al otro hijo.La parte de la cruza que se lleva a cabo entre la entrada y la primer capa oculta se muestra en la figura 7.1. La parte izquierda de la primer capa oculta del primer padre con todas sus conexiones a la capa de entrada se copian al hijo. La parte derecha de la primer capa oculta del segundo padre con todas sus conexiones a la capa de entrada se concatenan a la primer capa oculta del hijo.

En la figura 7.2 se muestra cómo se produce la cruza entre capas ocultas. Para cada capa oculta, la parte izquierda del primer padre se copia al hijo, incluyendo todas las conexiones en ambas direcciones, cuando sea posible. Es decir, se copian todas las conexiones que apuntan hacia adentro del hijo, pero si la capa por encima o por debajo, en el hijo, tiene menos nodos que uno de los padres entonces ese padre tiene conexiones que no entrarán en el hijo. Por ejemplo, si el primer padre tiene una conexión del nodo 1 en la capa 2 hacia el nodo 10 en la capa 3 y el hijo solo tiene 7 nodos en la capa 3 entonces esta conexión no se puede copiar, simplemente se ignora.

La cruza entre la última capa oculta y la capa de salida es muy parecida a la que se realiza entre la capa de entrada y la primer capa oculta. Se copian en el hijo todas las conexiones de la parte izquierda del primer padre desde la última capa oculta hacia la capa de salida. De la misma manera, se copian en el hijo todas las conexiones de la parte derecha del segundo padre desde la última capa oculta hacia la capa de salida.

El modo en que trabaja esta cruza hace que el número de nodos en las capas ocultas sea muy dinámico. Esta propiedad debería permitir que el AG cubra la totalidad del espacio de búsqueda, al menos en lo que se refiere al número de nodos ocultos.

En la cruza, tal como fue definida, la parte izquierda de una red neuronal se combina siempre con la parte derecha de otra red neuronal. Pero de esta forma si existieran buenas características en las dos partes izquierdas, éstas no se podrían combinar, lo cual no es bueno desde el punto de vista del problema de la permutación. De manera de hacer que este problema sea menos significativo, introducimos un ***operador de inversión***, previo a la aplicación de la recombinación. La inversión consiste en intercambiar dos nodos ocultos elegidos al azar, con todas sus conexiones, en cada capa oculta en cada padre.

Otro operador que ayuda ante el problema de la permutación es la mutación. Este operador induce a explorar la totalidad del espacio de búsqueda y permite mantener la diversidad genética en la población, para que el algoritmo genético sea capaz de encontrar soluciones entre todas las permutaciones posibles de la red.

La mutación se aplica antes de evaluar al hijo en cuanto a su aptitud. Durante la misma se intercambian un número al azar de conexiones entre los nodos de la red.

7.3. Mejoras a la capacidad de generalización

Para evitar el sobreajuste de los datos de entrada de la red, se implementará en el algoritmo híbrido las técnicas de detención temprana y regularización de la complejidad presentadas anteriormente.

La primer técnica se concentra en detener al algoritmo de aprendizaje haciendo uso de la validación cruzada, ya que es posible que la red termine sobreajustándose a los datos de entrenamiento si la sesión de entrenamiento no se detiene en el momento correcto. La cuestión que se deberá resolver durante la experimentación es cuántas veces se debe permitir que el entrenamiento no mejore sobre el set de validación, antes de detener el entrenamiento.

La segunda técnica se concentra en la minimización del tamaño de la red manteniendo la buena performance, ya que una red neuronal con tamaño mínimo es menos probable que aprenda el ruido en los datos de entrenamiento y puede así generalizar mejor sobre datos nuevos. Se buscará en la experimentación el parámetro de regularización más adecuado para el entrenamiento back-propagation durante el algoritmo híbrido.

7.4. Evaluación del proceso evolutivo

7.4.1. El set de datos

Para probar la forma de encontrar una topología óptima de red neuronal se requerirá un set de entrenamiento para llevar a cabo la experimentación. Este set de entrenamiento deberá contar con un buen número de ejemplos, ya que la idea es entrenar a una red neuronal sobre estos ejemplos, y entonces ver si esa red neuronal es capaz de clasificar cualquier otro ejemplo presentado.

7.4.2. Evaluación de la performance del algoritmo híbrido AG/RN

Los resultados de la experimentación se pueden dividir en tres partes: (1) Evolucionar las topologías usando el algoritmo híbrido. Para determinar si estas nuevas topologías son realmente buenas en el dominio específico empleado se requerirá: (2) Ver cómo estas redes neuronales se entrenan sobre datos adicionales y (3) Ver cómo pueden clasificar un tercer set de datos una vez que fueron entrenadas sobre los datos anteriores (del punto 2).

7.4.3. Evaluación de la performance del AG

En la mayoría de los problemas de optimización que usan AG's, el objetivo es disminuir el costo de las soluciones, creando cada vez mejores soluciones tras cada generación sucesiva. Como cada solución en este AG es una red neuronal, la función de costo representa la habilidad de entrenamiento de cada red neuronal sobre un cierto set de datos. La función de costo es el error, o diferencia, entre las salidas deseadas y la salida de activación real de la red neuronal, también conocida como aptitud estándar, promediado sobre todos los ejemplos de entrenamiento en el set de datos. La performance del AG se puede determinar mirando el costo de la red neuronal que mejor se desempeña en cada generación y ver si el costo disminuye a medida que el AG progresa. Esto se visualiza mejor al graficar, para cada generación, el costo de la red neuronal con la mayor performance de su generación. Este método mostrará si el AG está o no produciendo topologías de red neuronal que pueden aprender mejor el set de entrenamiento. Sin embargo, el propósito de este trabajo es crear topologías de redes neuronales que sean específicas del dominio, es decir, topologías que trabajen bien para entrenar y clasificar un cierto tipo de datos. Para evaluar si una topología trabaja bien o no sobre un cierto tipo de datos, las redes neuronales que resulten del AG pueden ser testeadas individualmente en sus habilidades de aprender y clasificar datos que no fueron usados en el proceso AG/RN. Para lograr ésto, el set de datos completo se particiona en tres partes iguales, cada una con igual número de ejemplos. Una partición se puede usar en el proceso recién descripto, y los otros dos se dejan como datos nuevos con el propósito de demostrar la habilidad de una topología de red neuronal para manipular datos del mismo dominio. La partición del set de entrenamiento (SE) utilizada en el proceso AG/RN será referenciada como $SE_{AG/RN}$.

7.4.4. Evaluación de la performance de una red neuronal resultante

Las preguntas importantes que se deben responder para evaluar la performance de una topología de red neuronal resultante son:

- ¿Qué capacidad tiene una topología de red neuronal para aprender nuevos datos de entrenamiento?

- ¿Qué tan bien puede generalizar una red neuronal una vez que fue entrenada con los nuevos datos de entrenamiento?

Para verificar que una red neuronal, de hecho, tiene una buena topología específica al dominio, se testea su habilidad de aprendizaje de nuevos datos. Para hacerlo, la red neuronal se entrena con uno de los sets de datos distintos a $SE_{AG/RN}$, que será llamado $SE_{Entrenamiento}$. El entrenamiento de la red neuronal con $SE_{Entrenamiento}$ brinda un valor de cuán bien se entrena la red. Adicionalmente, una tercer partición de datos, SE_{Test}, puede determinar cuán bien puede generalizar la red entrenada una vez que aprendió el $SE_{Entrenamiento}$.

Es una práctica común separar al set de datos de validación de una red neuronal en dos partes con el propósito de evaluar la habilidad de generalización [Russel y Norvig, 1995]. Para lograrlo, una red neuronal primero se entrena con un "set de entrenamiento", y luego

trata de clasificar el "set de testeo". En este caso, el set de entrenamiento es el $SE_{Entrenamiento}$, y el set de testeo será el tercer set de datos denominado SE_{Test}. La figura 7.5 lo muestra gráficamente.

Para obtener mejores resultados, se utilizará en la experimentación una **validación cruzada**. La validación cruzada significa que $SE_{Entrenamiento}$ y SE_{Test} se intercambian y cada uno es utilizado para el propósito opuesto. Esto asegura que cualquier tendencia en los resultados son, de hecho, tendencias, y no sólo ciertas "casualidades" en los datos. Afortunadamente, si una red neuronal puede aprender un set de entrenamiento, puede también aprender otro set de la misma clase de datos. Los resultados de la habilidad de entrenamiento y la habilidad de generalización se darán entonces promediados para ambas orientaciones de $SE_{Entrenamiento}$ y SE_{Test}.

Nota: La validación cruzada que se acaba de presentar no debe ser confundida con la validación cruzada empleada por el método de entrenamiento con detención temprana del algoritmo back-propagation. En aquel caso la técnica se aplica a cada individuo de la población para detener su entrenamiento durante el proceso AG/RN, mientras que aquí se aplica a las topologías resultantes de dicho proceso durante su evaluación.

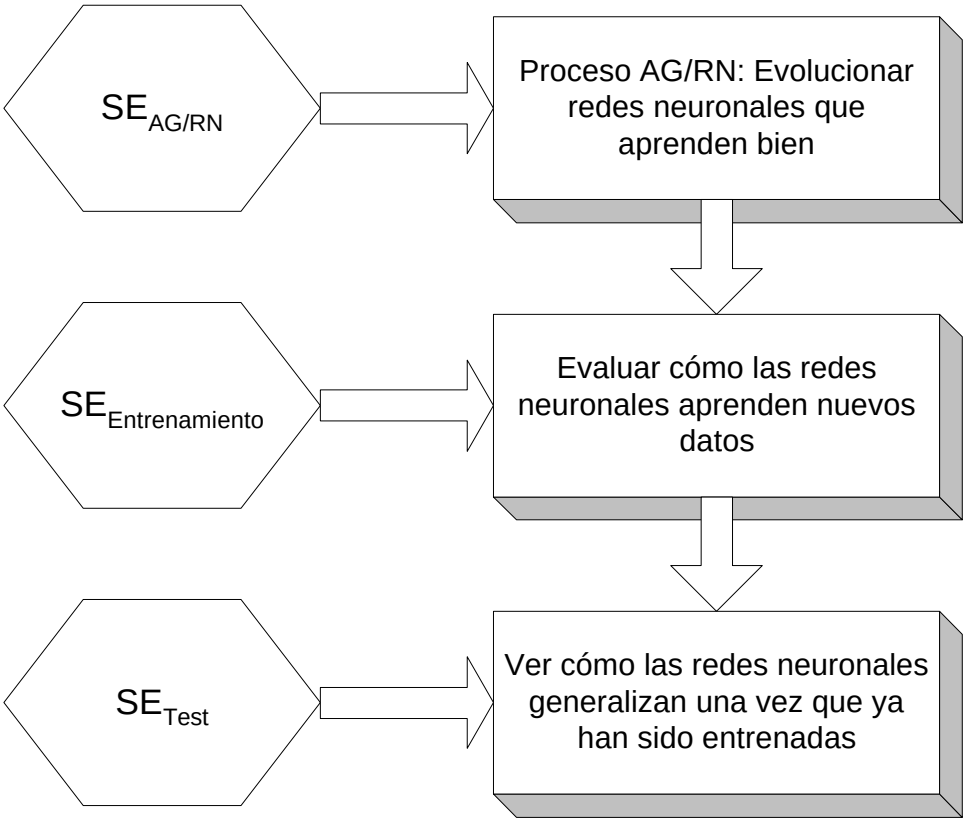


Figura 7.5 El set de datos se particiona en tres partes de igual tamaño de manera que las redes neuronales resultantes del AG puedan analizarse en sus habilidades para tratar con nuevos datos. Primero, se usa $SE_{AG/RN}$ en el paso del AG. Entonces, las redes neuronales resultantes del AG se entrenan usando $SE_{Entrenamiento}$. Finalmente, estas redes entrenadas clasifican a SE_{Test} para obtener una evaluación de sus capacidades de generalización.

7.4.5. Extensión de la validación cruzada

Para una evaluación más exhaustiva del algoritmo AG/RN, cada una de las tres particiones del set de datos completo tomará su turno como la partición inicial $SE_{AG/RN}$, dejando las otras dos para el proceso descripto previamente. Ya que ahora ninguna partición del set de datos tiene un propósito específico, será útil referirse a cada una como partición *A*, *B* y *C*.

8. Experimentación

A continuación se presenta la descripción del algoritmo híbrido utilizado en la sección 8.1, y los experimentos llevados a cabo en la sección 8.2.

8.1. Diseño experimental

El algoritmo híbrido que empleamos para la generación automática de redes neuronales es el siguiente:

1. Crear una población inicial de individuos (redes neuronales) con topologías aleatorias. Entrenar a cada individuo usando el algoritmo de back-propagation.
2. Seleccionar al padre y a la madre mediante un torneo: para cada uno, elegir tres individuos al azar de la población, quedarse con aquel que tenga el menor error.
3. Recombinar ambos padres para obtener dos hijos.
4. Mutar aleatoriamente a cada hijo.
5. Entrenar a cada hijo usando el algoritmo de back-propagation.
6. Ubicar a los hijos en la población mediante un torneo: para cada uno, elegir tres individuos al azar de la población, reemplazar por el hijo a aquel que tenga el mayor error.
7. Repetir desde el paso 2 durante un número dado de generaciones.

Este algoritmo emplea un esquema de codificación directo, utiliza la selección por torneo (basada en el orden) y el reemplazo consiste en una actualización gradual también aplicando la técnica de torneo. Cada red neuronal se entrena durante 500 repeticiones con back-propagation. Este valor es más alto del que se utiliza generalmente para entrenar redes neuronales, dándole al entrenamiento tiempo suficiente para converger y aprovechando entonces todo el potencial de cada red.

Cada experimento se lleva a cabo tres veces, 100 generaciones por vez. En cada ocasión se utiliza una partición distinta para $SE_{AG/RN}$.

Todos los experimentos utilizan un tamaño de población de 20 individuos. Este es un valor estándar usado en algoritmos genéticos. Aquí debemos hacer un compromiso entre

presión selectiva y tiempo de computación. El empleo de 20 individuos sirve para acelerar el desarrollo de los experimentos sin afectar a los resultados.

El algoritmo back-propagation se basa en el modo de entrenamiento secuencial, la función de activación elegida para cada neurona es la tangente hiperbólica.

8.1.1. Inicio del algoritmo híbrido

En el paso inicial de un AG, se produce una población aleatoria de organismos. Claramente no tendría sentido generar redes neuronales con topologías completamente random. Existen altas probabilidades de que un conjunto de nodos que se conectan aleatoriamente entre sí no constituyan una red neuronal válida. Entonces debemos limitar de algún modo al espacio de estructuras para que la generación aleatoria de redes neuronales sea más exitosa.

El set de entrenamiento que se utiliza dictamina el tamaño de las capas de entrada y de salida. Es decir, si los datos de entrenamiento tienen *a* entradas y *b* salidas, se sabe con anticipación que la capa de entrada tendrá *a* nodos, y la capa de salida tendrá *b* nodos. Antes de correr el algoritmo híbrido, el usuario también tiene que seleccionar el número de capas ocultas con el que trabajará el sistema, tiene que haber al menos una capa oculta, pero ésta es la única restricción sobre el número de capas ocultas.

El número de nodos en las capas ocultas es simplemente un valor al azar permitido, es decir mayor a cero y menor a un máximo definido. El rango de inicialización de pesos (los valores entre los cuales pueden estar los pesos de la red) también se ajusta en forma aleatoria.

La manera más obvia de inicializar las conexiones de la red en la población inicial sería para cada conexión posible decidir si dicha conexión va a estar presente o no. Pero se induciría entonces a que todos los individuos tengan aproximadamente un 50% de conectividad, y no existiría una alta diversidad entre los individuos. Por lo tanto, para cada individuo primero seleccionamos un porcentaje de conectividad entre el 10% y 90%, y ese será el porcentaje de las posibles conexiones del individuo. Así obtenemos la diversidad que estábamos buscando.

8.1.2. Cruza de la tasa de mutación y del rango de inicialización de pesos

Tanto la cruza de la tasa de mutación como del rango de inicialización de pesos se llevan a cabo de la misma forma. La estrategia general es tomar un promedio entre los padres y modificarlo con un número aleatorio dentro de ciertos límites. Tenemos un par de valores para la modificación de cada promedio, un par para la mutación y otro para la inicialización de pesos. Dentro del par, el primer valor es más grande que el segundo. En la prime-

ra generación, la modificación máxima que se puede aplicar al promedio es la suma de ambos valores, luego esta modificación máxima decrece linealmente hasta el segundo valor, el cual se alcanza cuando haya transcurrido la mitad de las generaciones. Desde ese punto, la modificación máxima es igual al segundo valor.

La idea de ir disminuyendo la máxima modificación aplicable es que al comienzo queremos explorar la totalidad del espacio de búsqueda, pero al final queremos la convergencia del algoritmo, por lo tanto a medida que avanza la evolución las modificaciones pequeñas deberían ser mejores.

8.1.3. El set de datos utilizado

El set de entrenamiento elegido para usarse en estos experimentos se obtuvo desde un archivo de data sets en Internet [Blake y Merz, 1998]. Consiste en datos concernientes a 600 solicitudes de tarjeta de crédito. Cada solicitud de tarjeta de crédito constituye un ejemplo de entrenamiento. Para el aprendizaje, la información de la solicitud forma la entrada a la red neuronal, y la salida es un valor verdadero o falso que especifica si la solicitud fue o no aprobada. La idea, por supuesto, es entrenar a la red neuronal sobre estos ejemplos, y luego tener una red neuronal capaz de aprobar solicitudes de tarjeta de crédito.

Toda la información de las solicitudes está cambiada a símbolos sin significado para mantener la confidencialidad. Los atributos de un ejemplo de entrenamiento se ven como:

b,30.83,0,u,g,w,v,1.25,t,t,01,f,g,00202,0,+

Algunos de los parámetros toman la forma de números. Para presentar estos números a la red, se determina el mínimo y el máximo valor para ese atributo dentro del set completo de entrenamiento y luego se escalan los atributos entre –1 y +1. Por ejemplo, si un atributo del set de entrenamiento va desde 0 a 100.000, entonces se debe comprimir entre –1 y +1 a todos los valores para este atributo, de manera tal que puedan ser usados por la red neuronal. Las entradas no numéricas (las letras en el ejemplo de arriba) son respuestas a preguntas múltiple choice. Por ejemplo, el primer atributo en cada ejemplo de entrenamiento podría ser *a* o *b*. Ya que hay dos opciones, este atributo tomará la forma de dos entradas a la red donde una es verdadera y la otra es falsa, dependiendo de si los ejemplos usan *a* o *b*. Verdadero se representa en la red como +1, y falso se representa como –1. Utilizando estos dos métodos de transformar los datos de la solicitud de aprobación de tarjetas de crédito en una forma que pueda utilizar la red, llegamos a una red que tiene 47 entradas. El signo + del final del ejemplo de arriba constituye la *clase* del ejemplo. En este caso es un + o un – dependiendo de si la solicitud se aprobó o no, por lo tanto la red neuronal tendrá dos salidas, una que se activa para cada caso.

El set de datos, como se describió anteriormente, se particiona en 3 partes de igual tamaño, cada una con la misma cantidad de solicitudes aprobadas y desaprobadas.

8.1.4. Significado del error

En el análisis de los experimentos, es necesario ser capaz de asignarle una medida a qué tan bien una red neuronal clasifica un set de datos. En este trabajo, al valor utilizado para reportar este resultado se lo refiere como error. El error es inversamente proporcional a la performance sobre el set de validación, caracterizada por el número de patrones clasificados correctamente. Existe una penalidad sobre el número de nodos ocultos que utiliza la red, dividida por el número de capas, de lo contrario las redes con muchas capas tendrían una penalidad muy elevada. Además hay un término de penalidad para el porcentaje de conexiones que tiene la red, dividida (nuevamente por los mismos motivos recién mencionados) por el número de capas.

8.1.5. La representación directa utilizada

El formato del genotipo contiene los siguientes campos:

geno<id>	en uso	error	N_0	N_1 ...	N_n	semilla	P	M_0	M_1 ...	M_{n-1}

geno<Id>: Identifica unívocamente al genotipo.

en uso (booleano): Indica si el genotipo se encuentra participando en una cruza.

error: El error resultante del entrenamiento.

N_i: Número de nodos en la capa *i*, *i* va de 0 a *n*.

semilla: La semilla del generador random.

P: Rango para los pesos iniciales, el rango es [-P,+P].

M_i: La matriz de conexión entre la capa *i* y la capa *i*+1.

El campo "*en uso*" se usa para bloquear a un individuo antes del reemplazo, esto evita que el individuo sea elegido nuevamente para formar parte de un torneo. Luego que el individuo se inserta en la población, vuelve a ser liberado.

Debido al ruido en la evaluación de la aptitud, se almacena en el genotipo a la semilla para generar números aleatorios, por lo tanto es posible la reconstrucción de la red con exactamente el mismo error reportado. Este parámetro no forma parte de las operaciones genéticas (recombinación y mutación).

8.1.6. Tipos de gráfico

Los gráficos mostrados en la siguiente sección presentan los resultados de los experimentos llevados a cabo. Estos gráficos son de dos tipos diferentes que no deberían confundirse:

- Un tipo de gráfico muestra a las generaciones del AG a lo largo del eje "x", y muestra el error de la red neuronal de mejor performance en cada generación sobre el eje "y".

- El otro tipo de gráfico despliega el error de entrenamiento de una red neuronal individual, aquella producida por el algoritmo híbrido. Las repeticiones del back-propagation están sobre el eje "x", mientras que el error de la red está sobre el eje "y".

8.1.7. Variables independientes

Tamaño de la población: esta variable indica la cantidad de individuos que van a usarse en el algoritmo genético. A mayor cantidad, mayor diversidad genética y mejores posibilidades de no caer en un mínimo local, pero los tiempos de computación aumentan.

Tamaño del torneo: esta variable indica, en el algoritmo genético, la cantidad de individuos que se toman de la población para elegir el individuo a ser reproducido, y la cantidad de individuos que se toman de la población para elegir el individuo a ser reemplazado.

Número de generaciones: esta variable indica la cantidad de veces que itera el ciclo completo del algoritmo genético, el mismo involucra la selección, la reproducción, la mutación y el reemplazo.

Tasa de aprendizaje: esta variable indica cómo serán los cambios a los pesos en la red de una iteración a la siguiente en el algoritmo de back-propagation. A menor tasa de aprendizaje, más suave será la trayectoria en el espacio de pesos. Esta mejora, sin embargo, se alcanza a costa de un aprendizaje más lento

Tasa de momentum: esta variable indica cómo se pondera el término de momentum en el aprendizaje del back-propagation. Este término permite estabilizar el algoritmo y utilizar tasas de aprendizaje mayores.

Número de capas ocultas: esta variable indica el número de capas ocultas con el que trabajará el sistema para generar las redes neuronales, debiendo haber al menos una capa oculta.

Número de epochs: esta variable indica el número de presentaciones completas del set de entrenamiento que se utilizan en el entrenamiento de una red neuronal con el algoritmo de back-propagation.

Parámetro de regularización: esta variable indica la importancia relativa del término de la penalidad de complejidad con respecto al término de la medida de performance en el algoritmo de back-propagation.

Parámetro de detención temprana: esta variable indica cuántas veces se permite que el entrenamiento no mejore sobre el set de validación, antes de detener el entrenamiento de back-propagation.

8.1.8. Variables dependientes

Error de la red neuronal: esta variable indica una medida de la performance de la red neuronal a medida que es entrenada por el algoritmo de back-propagation. Permite observar el progreso del entrenamiento y puede utilizarse para detenerlo.

Error de la mejor red neuronal: esta variable indica una medida de la performance de la mejor red neuronal dentro la población de una generación. Permite observar la convergencia del algoritmo genético.

Porcentaje de aciertos: esta variable indica el porcentaje de ejemplos no vistos previamente que son clasificados correctamente por una red neuronal. Permite observar la capacidad de generalización de las redes neuronales generadas.

8.2. Experimentos

8.2.1. Constantes de Aprendizaje y Momentum óptimos

En la implementación del algoritmo back-propagation necesitamos seleccionar la tasa de aprendizaje η y el momentum α. Estas constantes determinan cómo se aplican los cambios a los pesos en la red de una iteración a la siguiente.

Utilizamos las siguientes combinaciones de taza de aprendizaje η y de constante de momentum α para observar sus efectos sobre la convergencia de la red neuronal:

$\eta \in \{0.01; 0.1; 0.5; 0.9\}$

$\alpha \in \{0.0; 0.1; 0.5; 0.9\}$

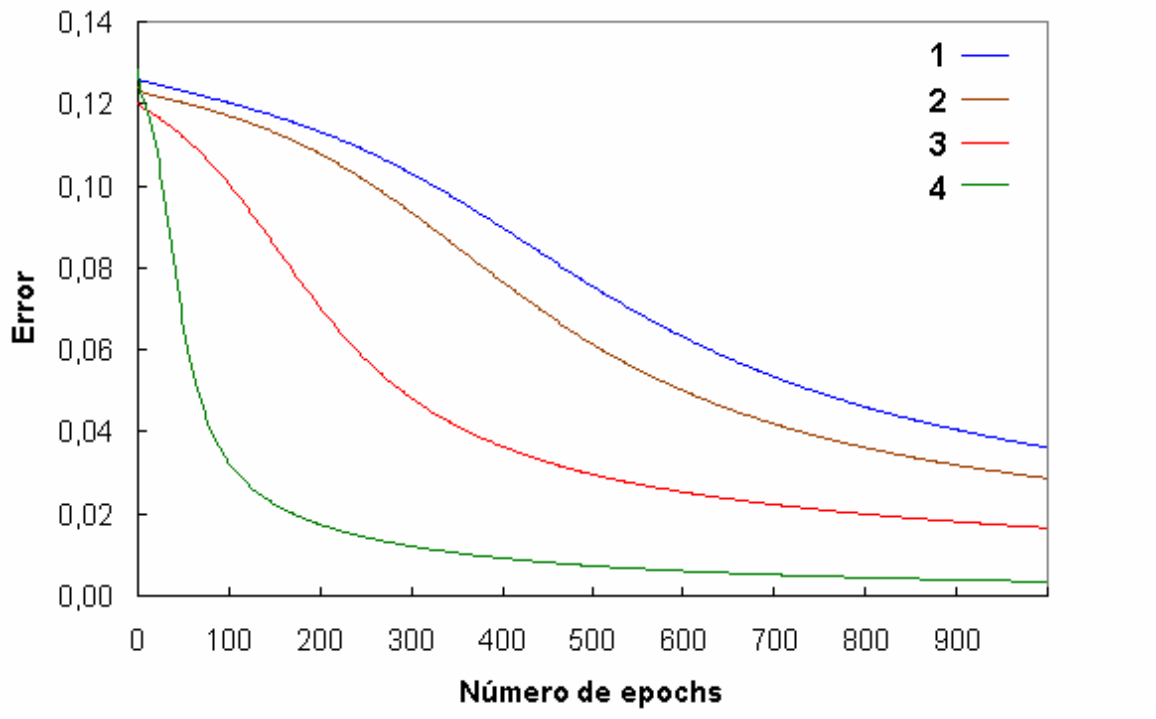


Figura 8.1 Tasa de aprendizaje h**=0.01** y momentum: (**1**) α=0; (**2**) α=0,1; (**3**) α=0.5; (**4**) α=0.9

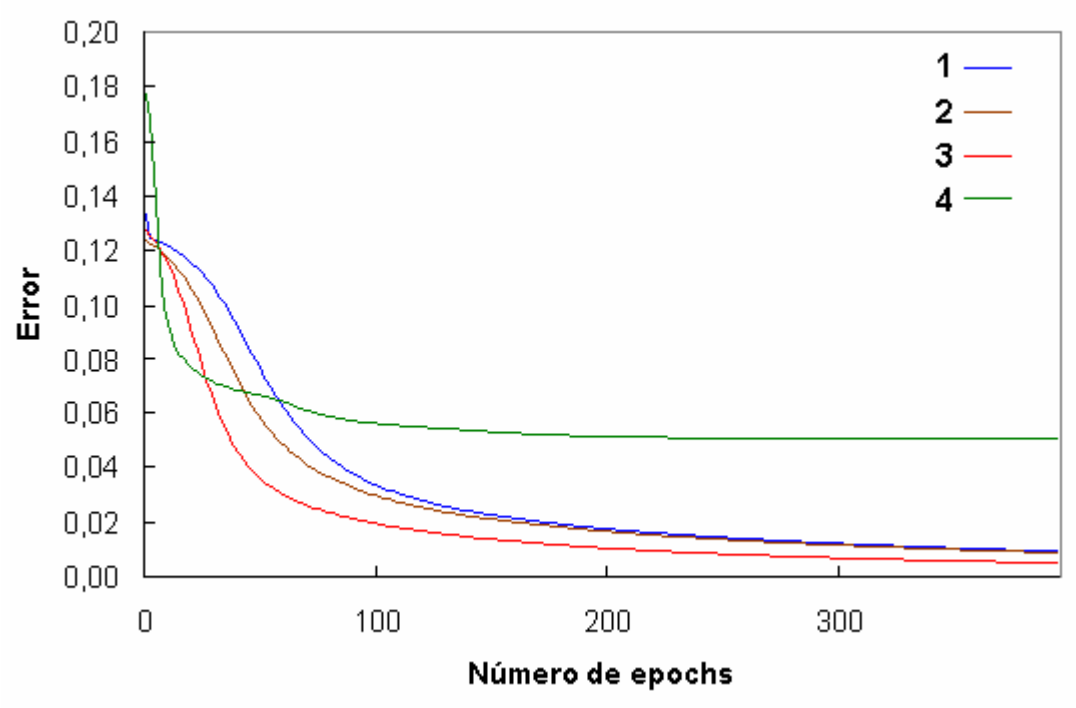


Figura 8.2 Tasa de aprendizaje h**=0.1** y momentum: (**1**) α=0; (**2**) α=0,1; (**3**) α=0.5; (**4**) α=0.9

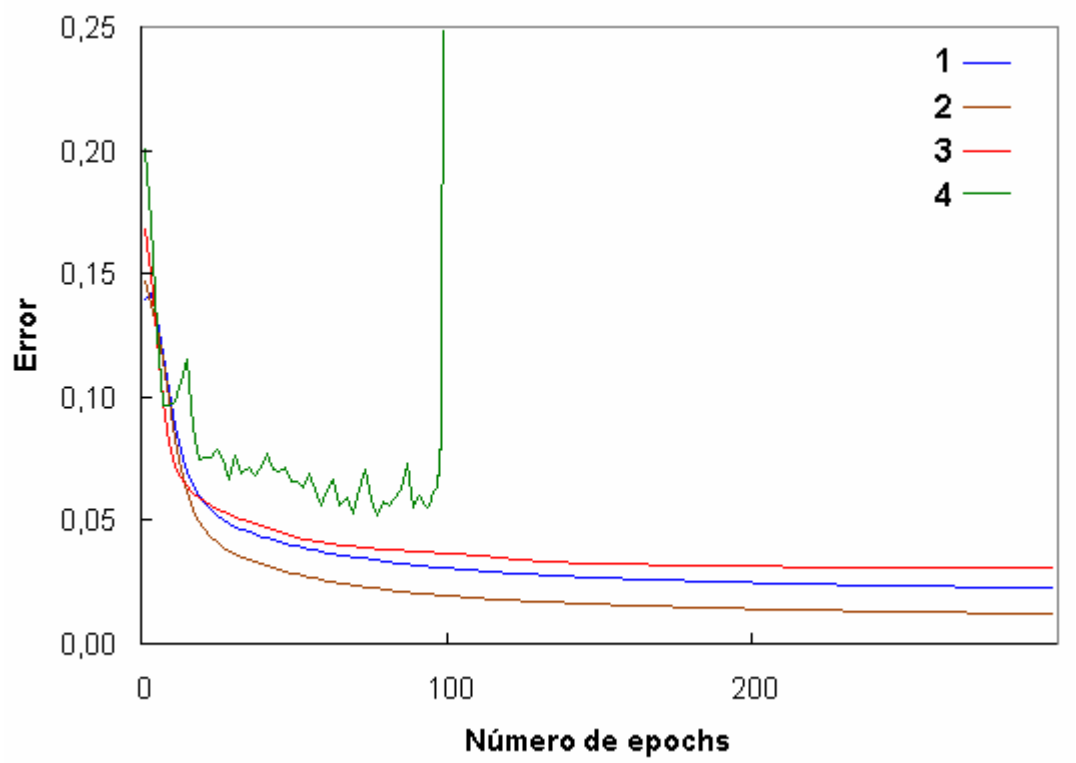


Figura 8.3 Tasa de aprendizaje h**=0.5** y momentum: (**1**) α=0; (**2**) α=0,1; (**3**) α=0.5; (**4**) α=0.9

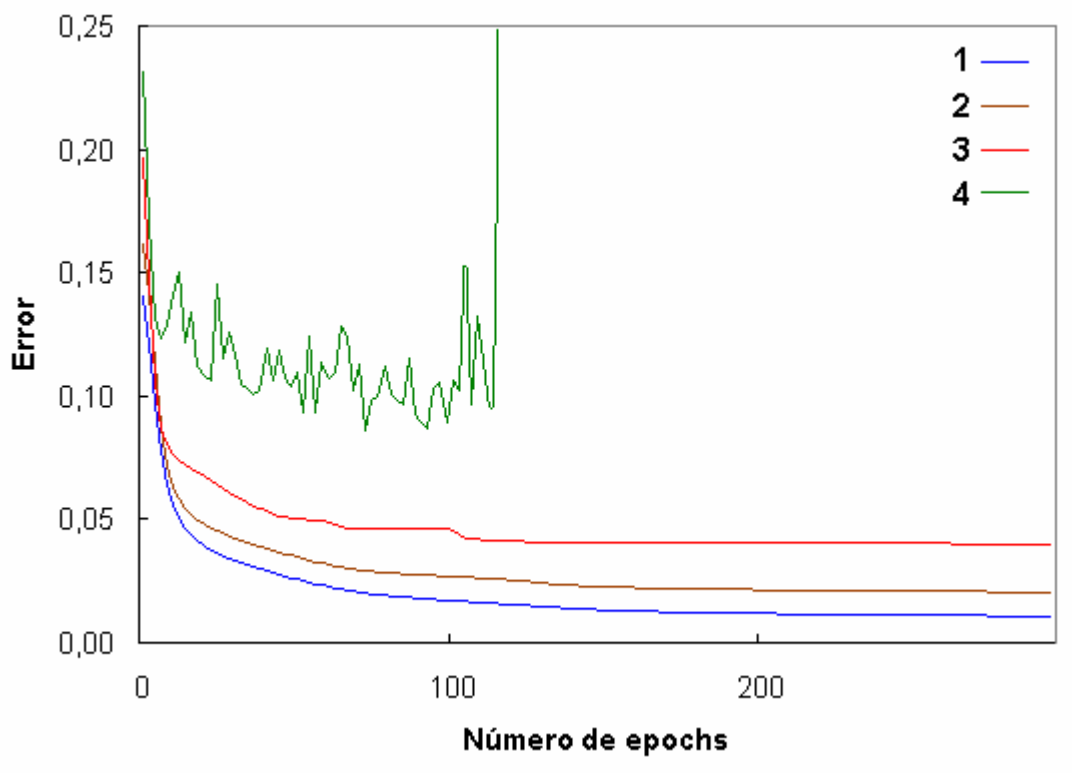


Figura 8.4 Tasa de aprendizaje h**=0.9** y momentum: (**1**) α=0; (**2**) α=0,1; (**3**) α=0.5; (**4**) α=0.9

Las curvas de aprendizaje sugieren las siguientes tendencias:

- Mientras que una tasa de aprendizaje η pequeña, en general, resulta en una convergencia lenta, puede localizar mejores mínimos locales en la superficie de error que los η grandes. Este resultado es intuitivo, ya que un η pequeño implica que en la búsqueda del mínimo se cubre más de la superficie de error que en el caso del η grande.

- Para $\eta \rightarrow 0$, el uso de $\alpha \rightarrow 1$ produce un incremento de la velocidad de convergencia. Por el otro lado, para $\eta \rightarrow 1$, se requiere el uso de $\alpha \rightarrow 0$ para asegurar la estabilidad del aprendizaje.

- El uso de las constantes $\eta = [0.5; 0.9]$ y $\alpha = 0.9$ causa oscilaciones en el error cuadrático medio durante el aprendizaje y valores más altos del error final en la convergencia, los cuales son efectos no deseados.

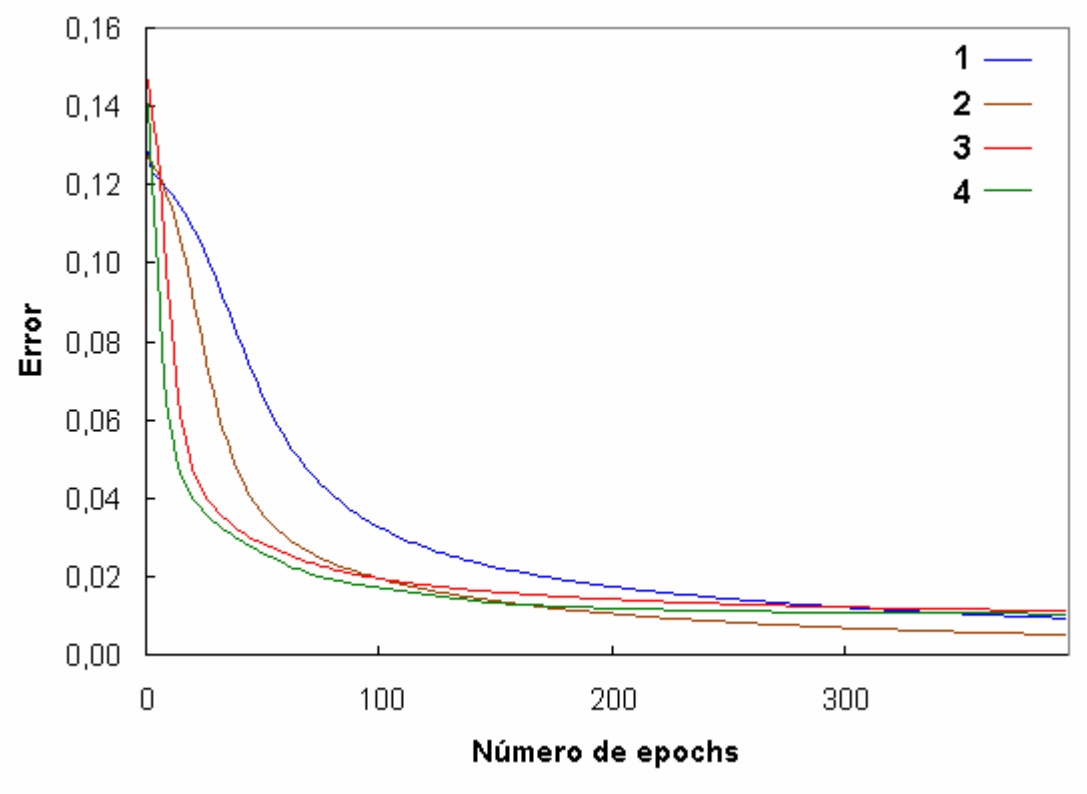


Figura 8.5 Mejores curvas de aprendizaje, siendo la tercera la curva con parámetros óptimos: (**1**) η=0.01 y α=0.9; (**2**) η=0.1 y α=0,5; (**3**) η=0.5 y α=0.1; (**4**) η=0.9 y α=0

En la figura 8.5 se muestran las mejores curvas de aprendizaje seleccionadas de los gráficos anteriores. De esta figura, se observa que la constante óptima de aprendizaje η es 0.1 y la constante de momentum óptima es 0.5. Estos valores serán utilizados en los experimentos sucesivos.

8.2.2. Número de capas ocultas óptimo

Como mencionamos previamente, antes de correr el algoritmo híbrido se tiene que seleccionar el número de capas ocultas con el que trabajará el sistema, debiendo haber al menos una capa oculta. Esta restricción surge de la manera en que definimos el operador de recombinación, que trabaja con dos redes neuronales con el mismo número de capas ocultas.

Utilizamos diferentes configuraciones de *n* capas ocultas, con $n \in \{1; 2; 3; 4\}$ para evaluar cómo va disminuyendo el error de la red neuronal más exitosa a través de las sucesivas generaciones, como se aprecia en la figura 8.6.

De la figura 8.6 se puede concluir que la red con 2 capas ocultas es preferible, ya que converge a un error menor. De aquí en adelante usaremos esta configuración en los experimentos. En cuanto a las redes con 1 y 3 capas, sus performances resultaron bastante similares, mientras que la red con 4 capas quedó por debajo en los resultados. En este último caso comienza a pesar la penalidad por el número de nodos y conexiones utilizadas, ya que un tamaño de red grande no implica una mejora en el desempeño.

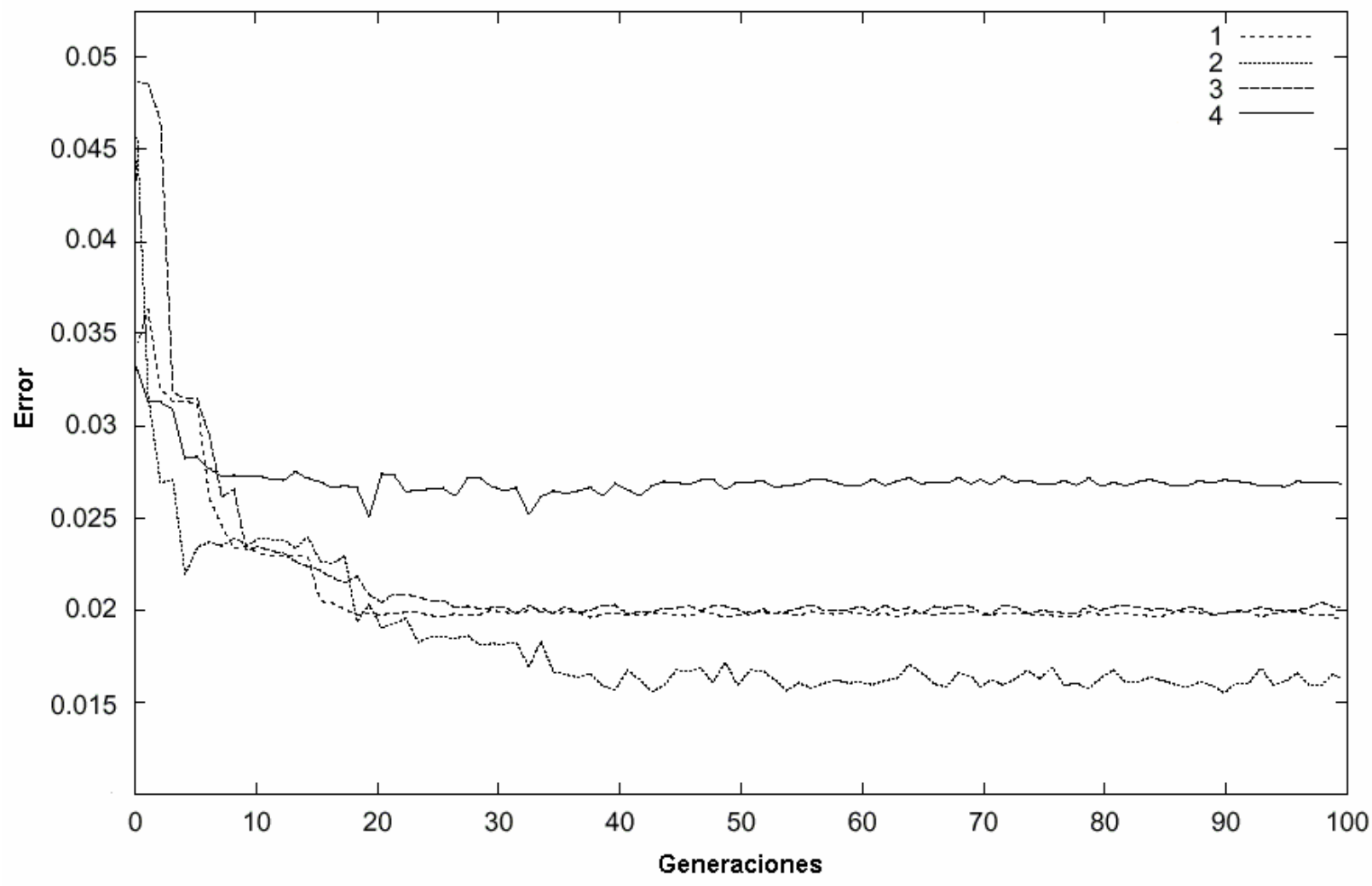


Figura 8.6 Evolución de las generaciones ante distintos números de capas ocultas: (**1**) *n* = 1; (**2**) *n* = 2; (**3**) *n* = 3; (**4**) *n* = 4.

8.2.3. Tamaño de torneo óptimo

La técnica de torneo se utiliza tanto para la selección como para el reemplazo. Durante la selección, cada vez que necesitamos un padre elegimos al azar a un número de individuos de la población, el mejor de los cuales se usa como padre. Durante el reemplazo, por el contrario, cada vez que tenemos que insertar un nuevo individuo en la población elegimos al azar a un número de individuos de la población, y reemplazamos a aquel con el peor desempeño.

El tamaño del torneo afecta a la presión selectiva que se aplica sobre la población. A mayor tamaño de la población, mayor presión selectiva. Consideremos el caso extremo donde el tamaño del torneo es igual al tamaño de la población. Aquí la presión selectiva se hace muy alta, siempre se van a seleccionar los mejores individuos para la reproducción y a los peores individuos para el reemplazo. Se corre el riesgo de perder la diversidad y converger prematuramente a un mínimo local. Por el contrario, un tamaño de población de tamaño 1 implica una selección y reemplazo al azar, donde la población no convergiría nunca.

Utilizamos un tamaño k para el torneo, con $k \in \{2; 3; 4; 5\}$ para evaluar cómo va disminuyendo el error de la red neuronal más exitosa a través de las sucesivas generaciones, como se aprecia en la figura 8.7.

En la figura 8.7 observamos que el tamaño de torneo k óptimo es igual a 3. El error es mayor cuando k =2. En [Schmidt 1996] se demuestra que un tamaño de torneo igual a 2 hace a la presión selectiva lineal con respecto a la función de aptitud, mientras que un tamaño de 3 la hace polinómica. Sin embargo utilizar valores para k de 4 y de 5 el error vuelve a aumentar. Esto posiblemente se debe a que el aumento en la presión selectiva no permite explorar el espacio de búsqueda en su totalidad. De aquí en adelante usaremos el valor de $k = 3$ en los experimentos.

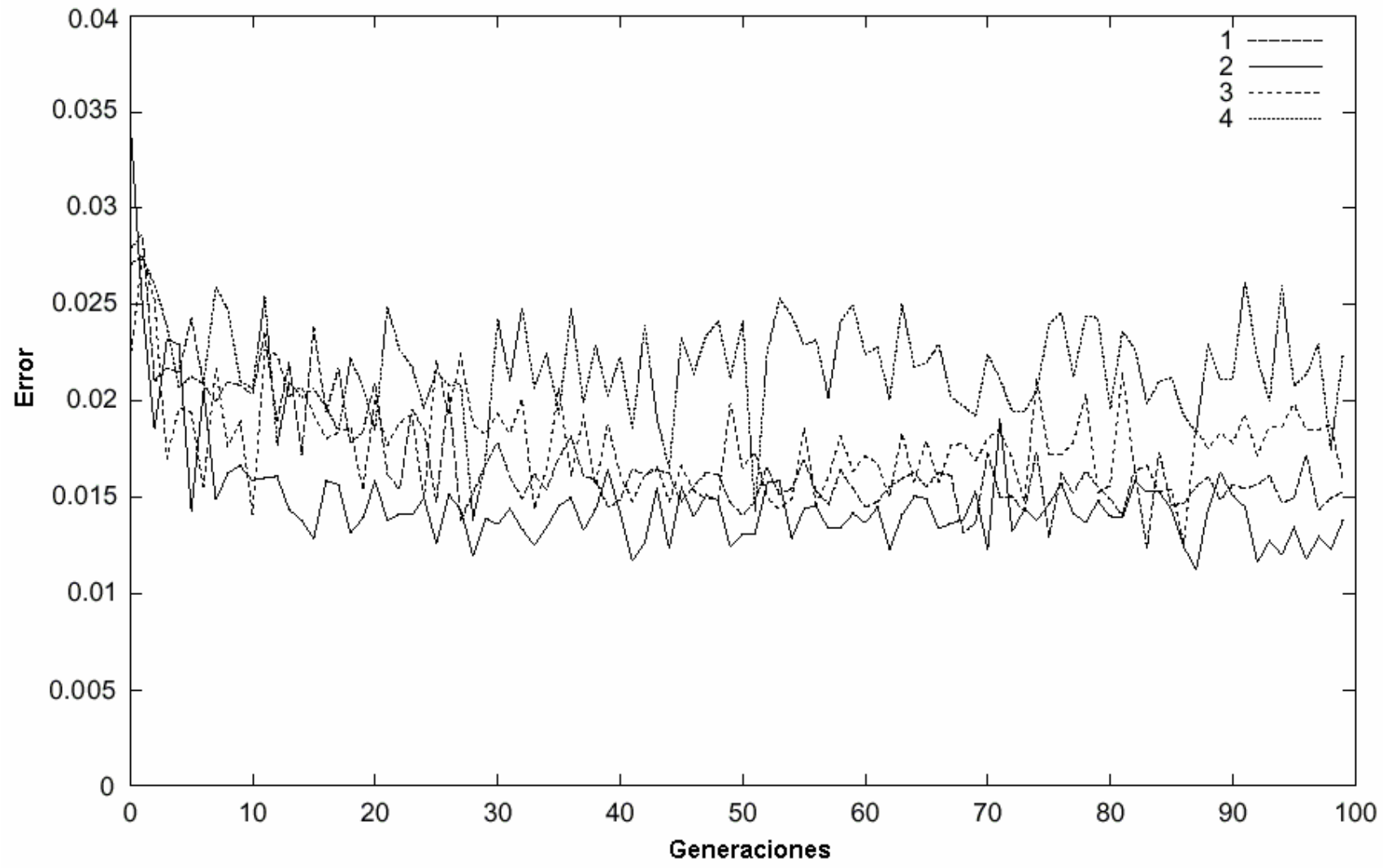


Figura 8.7 Evolución de las generaciones ante distintos tamaños de torneo (**1**) *k* = 2; (**2**) *k* = 3; (**3**) *k* = 4; (**4**) *k* = 5.

8.2.4. Número de repeticiones de entrenamiento óptimo

El problema de la evaluación ruidosa de la aptitud proviene de utilizar la aptitud del fenotipo para estimar la aptitud del genotipo. Una forma de atenuar sus efectos es entrenar muchas veces a cada red neuronal partiendo de diferentes pesos iniciales aleatorios. Se usa entonces el mejor resultado para estimar la aptitud de la red. Utilizamos un número de repeticiones de back-propagation $n \in \{1; 3; 5; 10\}$ para evaluar cómo va disminuyendo el error de la red neuronal más exitosa a través de las sucesivas generaciones, como se aprecia en la figura 8.8.

En esta figura 8.8 observamos que, como esperábamos, el error es mayor cuando no se repite el entrenamiento de la red neuronal. Esto se debe a que diferentes pesos iniciales aleatorios pueden producir diferentes resultados de entrenamiento. Al reiterar el entrenamiento y luego tomar la mejor aptitud de todas las iteraciones, estamos obteniendo una evaluación más acertada de la red neuronal, y la evolución puede verdaderamente seleccionar a los mejores individuos. El error, sin embargo, no parece disminuir apreciablemente al subir el número de repeticiones; el tiempo de computación, por el contrario, crece espectacularmente. Concluimos entonces que el número óptimo de repeticiones de entrenamiento de back-propagation n es igual a 3, y utilizaremos este valor en los experimentos siguientes.

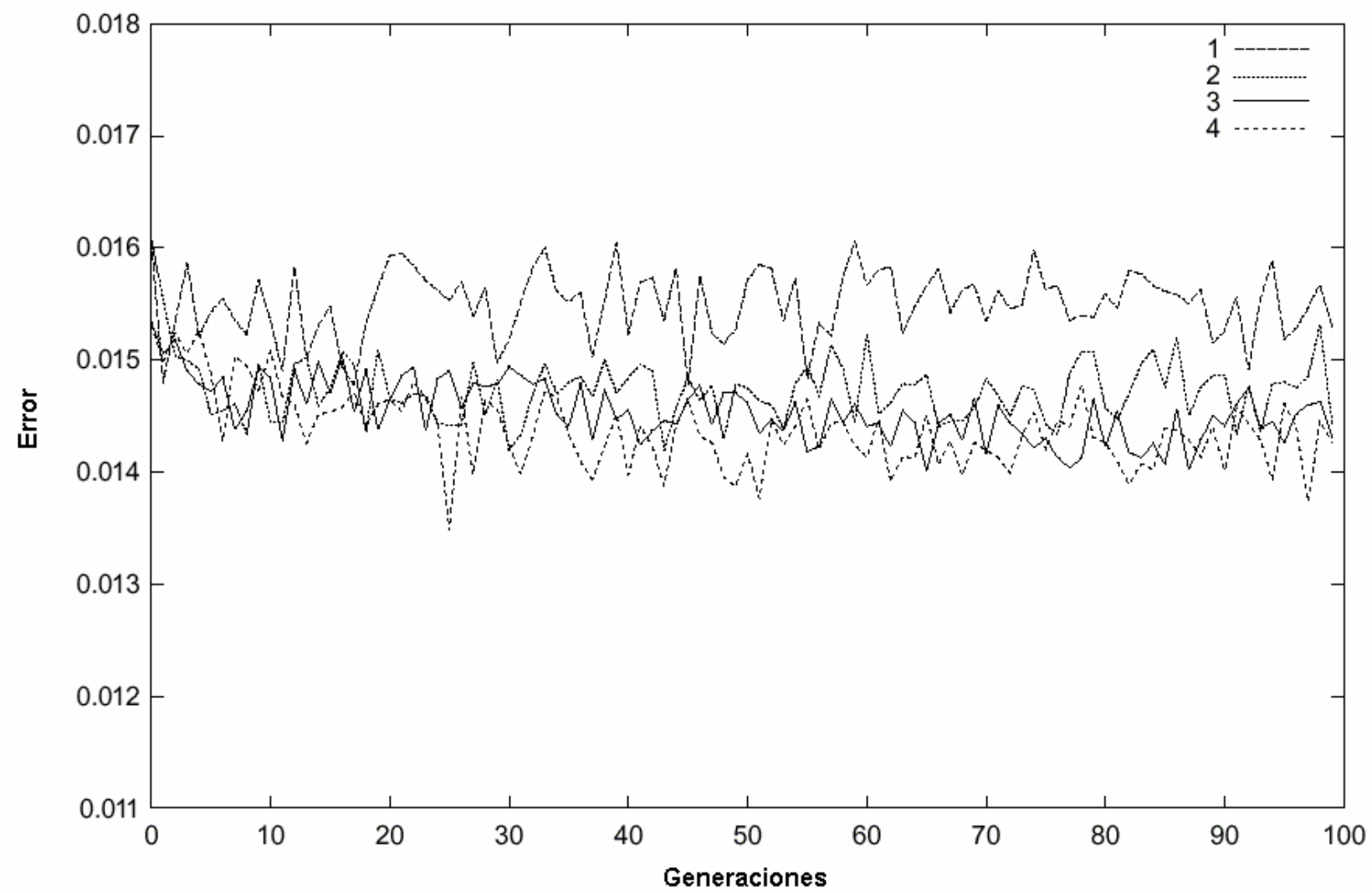


Figura 8.8 Evolución de las generaciones ante distintos valores de repetición del entrenamiento (**1**) *n* = 1; (**2**) *n* = 3; (**3**) *n* = 5; (**4**) *n* = 10.

8.2.5. Regularización de la complejidad óptima

Una vez encontrados los parámetros óptimos para el algoritmo híbrido, es momento de analizar las técnicas para mejorar la capacidad de generalización de las redes neuronales generadas.

Recordemos que la regularización de la complejidad se concentra en la minimización del tamaño de la red manteniendo la buena performance, ya que una red neuronal con tamaño mínimo es menos probable que aprenda el ruido en los datos de entrenamiento y puede así generalizar mejor sobre datos nuevos. La función de riesgo que intentamos minimizar es:

$$R(\mathbf{w}) = \varepsilon_S(\mathbf{W}) + \lambda\, \varepsilon_C(\mathbf{w})$$

El primer término $\varepsilon_S(\mathbf{W})$ (representando la medida de performance) es la misma aptitud estándar usada hasta ahora. El segundo término $\varepsilon_C(\mathbf{w})$ (representando la penalidad de complejidad) es el empleado en el procedimiento de degradación de pesos (weight decay).

Utilizamos el parámetro de regularización $\lambda \in \{0.05; 0.1; 0.15; 0.2\}$ para entrenar a la red con la partición A (para que los valores sean útiles deben ser pequeños). Luego observamos sus efectos sobre la convergencia de la red neuronal cuando se entrena con la partición B, como se aprecia en la figura 8.9. Por último evaluamos su habilidad de clasificación sobre la partición C, mostrada en la figura 8.10.

De la figura 8.9 se ve que la red generada con el parámetro λ=0.1 es capaz de aprender mejor un nuevo set de datos que las otras redes, incluyendo aquella que directamente no implementaba la regularización (λ=0). Se comprueba entonces que el uso de la regularización de la complejidad fomenta que los pesos excedentes (aquellos que tienen poca o ninguna influencia sobre la red) asuman valores cercanos a cero, y por lo tanto mejoran la generalización. Esto se refleja en la figura 8.10, donde la red generada con el parámetro λ=0.1 posee el mejor porcentaje de clasificación de ejemplos no vistos con anterioridad (los de la partición C).

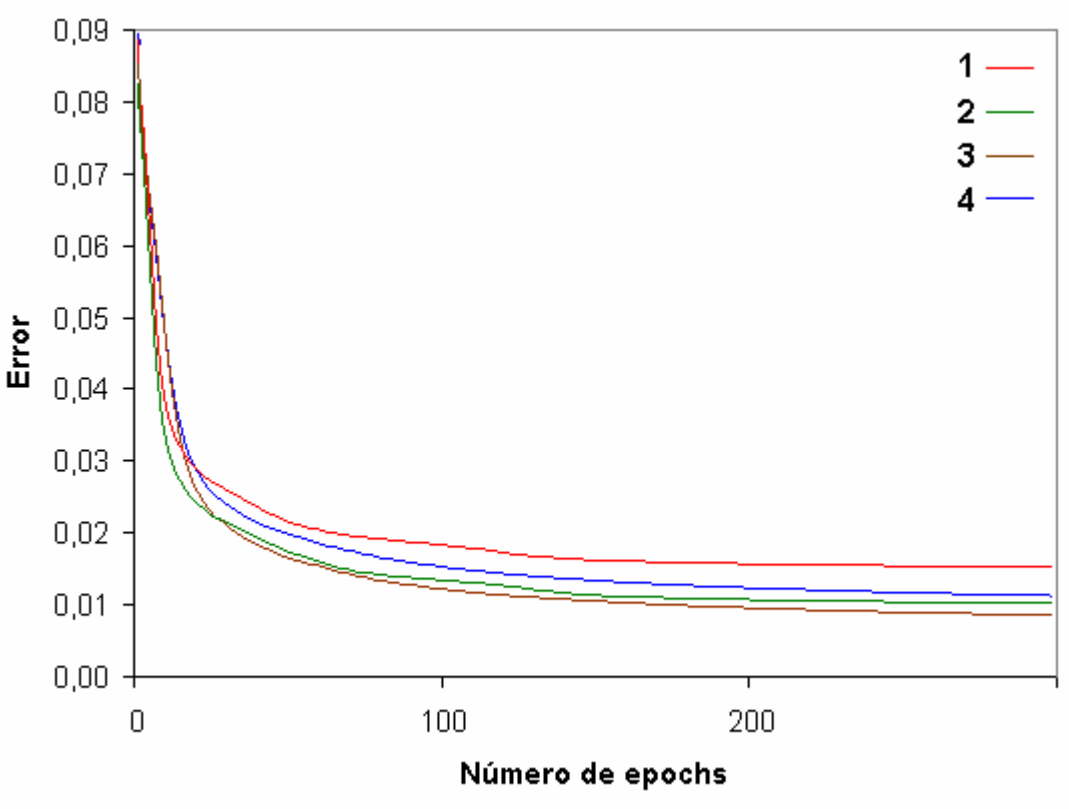


Figura 8.9 Regularización de la complejidad con parámetro de regularización ajustable: (**1**) λ=0 (sin regularización); (**2**) λ=0.05; (**3**) λ=0.01; (**4**) λ=0.15

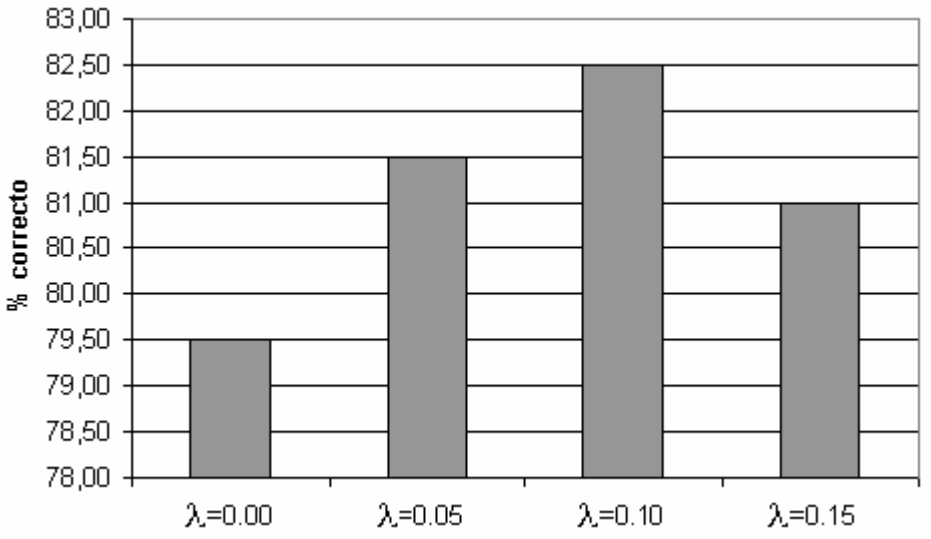


Figura 8.10 Comparación de los porcentajes de aciertos de las redes neuronales generadas con distintos parámetros de regularización λ.

8.2.6. Detención temprana óptima

Otra de las técnicas para mejorar la capacidad de generalización de las redes neuronales generadas es la detención temprana (early stopping).

Recordemos que esta heurística se concentra en detener al algoritmo de aprendizaje haciendo uso de la validación cruzada, ya que es posible que la red termine sobreajustándose a los datos de entrenamiento si la sesión de entrenamiento no se detiene en el momento correcto.

La pregunta que surge aquí es cuántas veces deberíamos permitir que el entrenamiento no mejore sobre el set de validación, antes de detener el entrenamiento. Definimos entonces al parámetro de detención β para representar a este número de entrenamientos.

Utilizamos el parámetro de detención $\beta \in \{2; 5; 10\}$ para entrenar a la red con la partición A. Después observamos sus efectos sobre la convergencia de la red neuronal cuando se entrena con la partición B, como se aprecia en la figura 8.11. Posteriormente evaluamos su habilidad de clasificación sobre la partición C, mostrada en la figura 8.12.

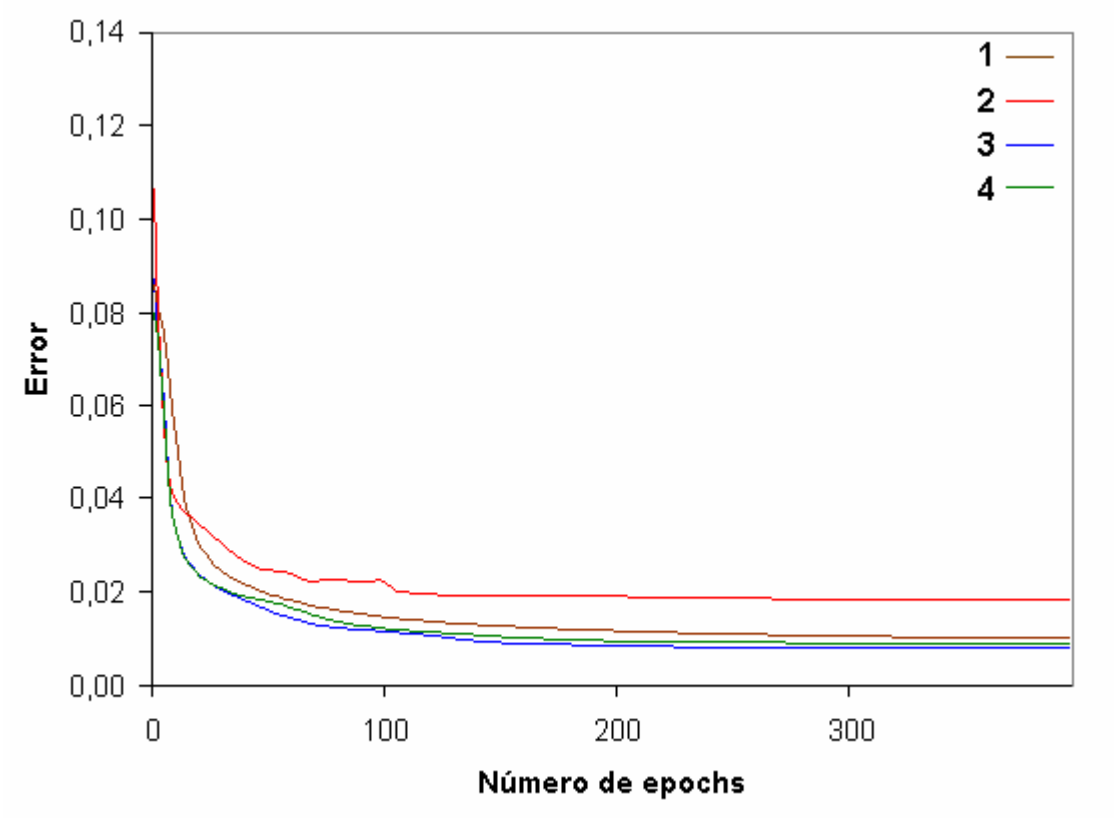


Figura 8.11 Detención temprana con parámetro de detención ajustable: (**1**) Sin detención temprana; (**2**) β=2; (**3**) β=5; (**4**) β=10.

De la figura 8.11 vemos que, si bien la diferencia es pequeña, la red neuronal generada con el parámetro de detención $\beta=5$ es capaz de aprender mejor un nuevo set de datos que las otras redes, incluyendo aquella que directamente no implementaba la detención (lo que equivaldría a $\beta\rightarrow\infty$). Esto verifica que lo que la red aprende más allá del punto de detención óptimo es esencialmente ruido contenido en el set de entrenamiento. En el caso de $\beta=2$, el crecimiento del error posiblemente se deba a que la detención se realiza demasiado pronto, sin dejarle a la red el tiempo suficiente para aprender bien.

Los resultados anteriores se corroboran en la figura 8.12, donde la red generada con el parámetro $\beta=5$ posee el mejor porcentaje de clasificación de ejemplos no vistos con anterioridad (los de la partición C).

Una variante interesante es aplicar nuevamente el método de detención temprana cuando se entrena con la partición B a la mejor red neuronal generada (en nuestro caso cuando usamos $\beta=5$). En este caso, la habilidad de clasificación de la red sobre la partición C se muestra en la figura 8.13. Esta red entrenada con $\beta=5$ sigue teniendo el mejor porcentaje.

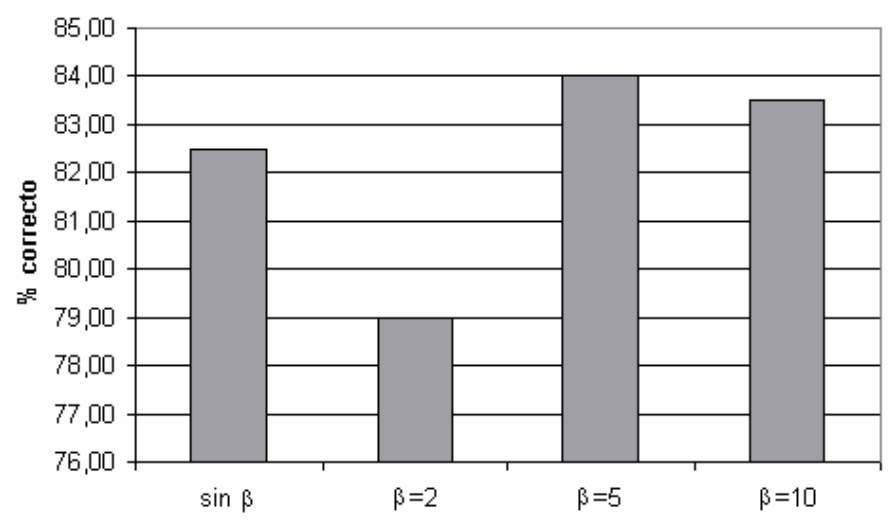


Figura 8.12 Comparación de los porcentajes de aciertos de las redes neuronales generadas con distintos parámetros de detención β.

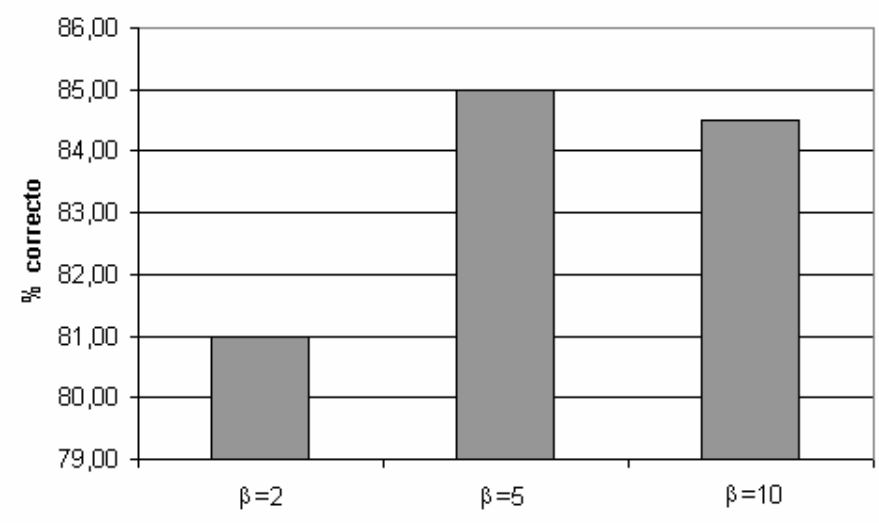


Figura 8.13 Comparación de los porcentajes de aciertos de la red neuronal generada usando β = 5, entrenada con la partición B utilizando distintos β

8.2.7. Comparación de la red neuronal óptima con otras redes

Para determinar si el proceso evolutivo está realmente mejorando o no a las redes neuronales en cuanto a sus topologías específicas del dominio, vamos a comparar a la red generada por el algoritmo híbrido (usando los parámetros óptimos determinados previamente) con la mejor topología random (generada en la primera generación del AG). Estas redes también son comparadas con una topología similar a la obtenida por el algoritmo híbrido pero 100% conectada (o conectada completamente).

Observamos los efectos de las distintas topologías sobre la convergencia de la red neuronal cuando se entrena con la partición B, como se aprecia en la figura 8.14. Posteriormente evaluamos su habilidad de clasificación sobre la partición C, mostrada en la figura 8.15.

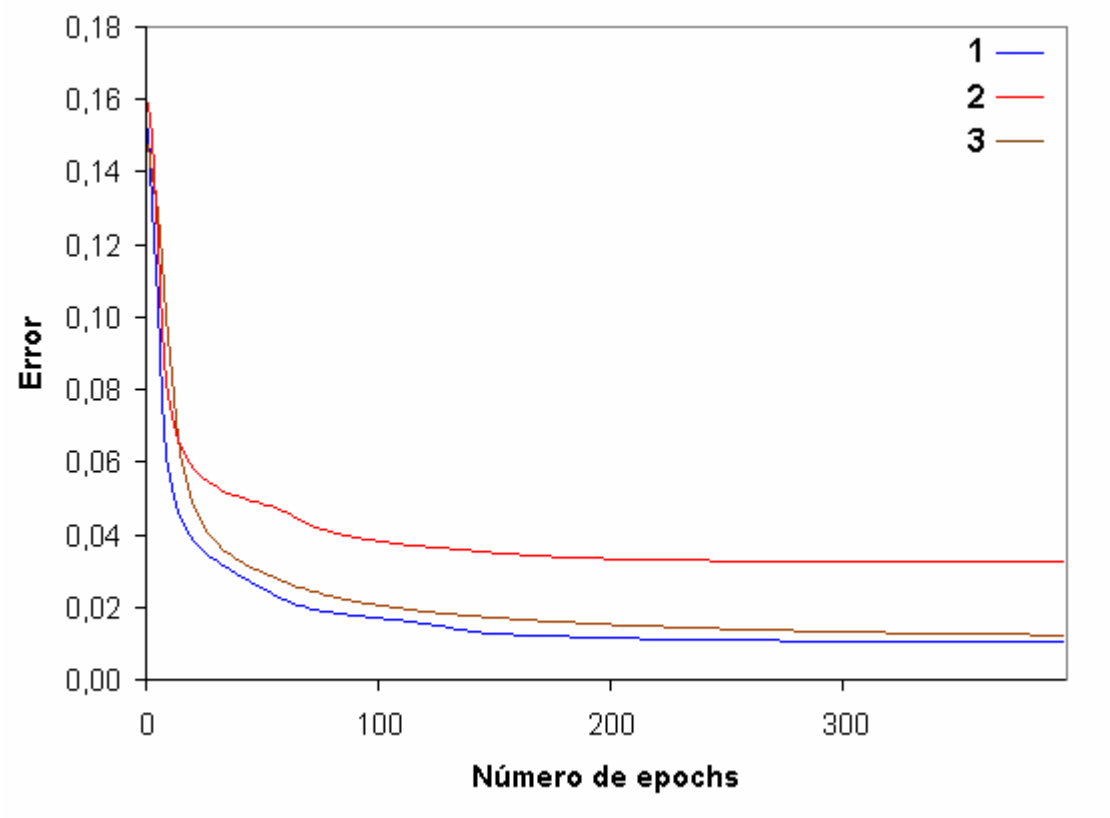


Figura 8.14 Capacidad de aprender nuevos datos usando: (**1**) La topología resultante del algoritmo híbrido; (**2**) La mejor topología random; (**3**) La topología del algoritmo híbrido pero 100% conectada.

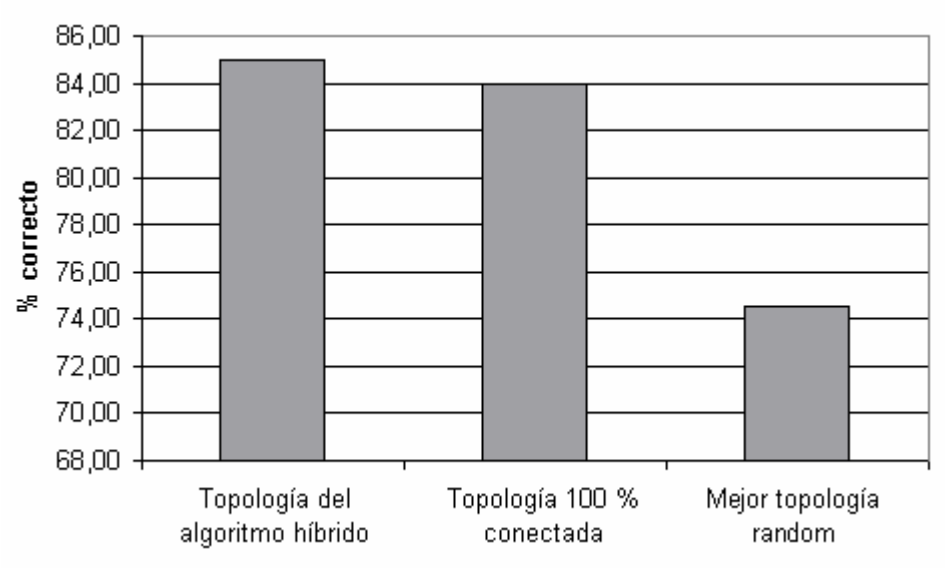


Figura 8.15 Comparación de los porcentajes de aciertos de las redes neuronales con distintas topologías y conexiones.

De la figura 8.14 vemos que la red neuronal generada con el algoritmo híbrido es capaz de aprender mejor un nuevo set de datos que las otras redes, incluyendo aquella que implementaba su misma topología pero conectada completamente.

La red generada con el algoritmo híbrido también posee el mejor porcentaje de clasificación de ejemplos no vistos con anterioridad (los de la partición C), como se aprecia en la figura 8.15.

9. Conclusiones, limitaciones y planes futuros

El mundo real a menudo posee problemas que no pueden ser resueltos exitosamente por una sola técnica básica; cada técnica tiene sus pros y sus contras. El concepto de sistema híbrido en inteligencia artificial consiste en combinar dos enfoques, de manera de que se compensen sus debilidades y se potencien sus fortalezas.

El objetivo de este trabajo es crear una forma de generar topologías de red neuronal que puedan fácilmente aprender y clasificar una cierta clase de datos. Para lograrlo, el algoritmo genético se usa para encontrar la mejor topología que cumpla con esta tarea. Cuando el proceso termina, el resultado es una población de redes neuronales específicas del dominio, listas para tomar nuevos datos no vistos anteriormente.

Este trabajo presenta la implementación y los experimentos resultantes del sistema híbrido que utiliza el algoritmo genético para buscar en el espacio de estructuras de una red neuronal de manera de encontrar una topología óptima.

Este trabajo contribuye al campo de los sistemas híbridos en inteligencia artificial al aplicar un método interesante de codificación directa para la generación y evaluación de topologías de redes neuronales multicapa dentro de un AG.

El análisis de los resultados de los experimentos presentados en el capítulo 8 demuestran que esta implementación AG/RN es capaz de crear topologías de red neuronal que en general, se desempeñan mejor que topologías random o conectadas completamente cuando se aprende y clasifica nuevos datos de un dominio específico.

Ahora se toman las cuestiones abordadas en el capítulo 6 y se sumarizan las respuestas a las mismas que aparecen a lo largo de este trabajo.

1. **¿En qué medida afectan los parámetros del algoritmo back-propagation al aprendizaje de la red?**

   El experimento 8.2.1 demuestra la importancia de las constantes que representan al parámetro de aprendizaje y al momentum en el desarrollo del algoritmo back-propagation. Si estos parámetros no se encuentran bien ajustados, pueden surgir varias dificultades y efectos no deseados, como la convergencia lenta o la inestabilidad del algoritmo que produce oscilaciones en el aprendizaje. Mediante la utilización de diversas combinaciones de valores para estos parámetros, se pudo determinar cuales eran las constantes de aprendizaje y momentum óptimos.

2. **¿Qué operador de cruza es el más adecuado en lo que respecta al problema de la permutación?**

   Para disminuir los efectos de la permutación, o equivalencia funcional de genotipos, implementamos un operador de cruza directamente sobre los fenotipos, como se explicó en 7.2. De manera que características buenas ubicadas sobre la misma parte de

los padres se puedan combinar, introducimos un operador de inversión, previo a la aplicación de la recombinación. La mutación también ayuda en este sentido, induciendo a explorar la totalidad del espacio de búsqueda y manteniendo la diversidad genética en la población. Se puede observar en la figura 8.14 que la evolución a través de sucesivas generaciones siempre baja el error de la mejor red neuronal dentro de la población, no existiendo variaciones que podrían ser producidas por la pérdida de eficiencia ocasionada por el problema de la permutación.

3. ¿Cómo influye la variación de la presión selectiva en la evolución del algoritmo genético?

Al implementar la técnica de torneo durante la selección y el reemplazo, podemos ajustar la presión selectiva sobre el AG al variar el tamaño del torneo. De acuerdo al experimento 8.2.3, se obtuvieron los mejores resultados al aplicar un tamaño de torneo igual a 3. El algoritmo genético alcanzó una convergencia más veloz y hacia un error menor con respecto a los otros tamaños de torneo. Cuando el tamaño del torneo es menor al óptimo, se produce una convergencia lenta del AG debido a una presión selectiva relativamente baja. Por el contrario, cuanto mayor se hace el tamaño del torneo con respecto al óptimo, la calidad de los resultados también disminuye, ésto se debe a que una presión selectiva demasiado alta provoca la convergencia prematura del AG hacia un mínimo local.

4. ¿Cómo se puede atenuar el problema de la evaluación ruidosa de la aptitud?

Considerando que diferentes pesos iniciales aleatorios en las redes neuronales pueden producir diferentes resultados de entrenamiento, la solución propuesta consistió en entrenar varias veces a cada arquitectura partiendo de diferentes pesos iniciales aleatorios, y usar entonces el mejor resultado para estimar la aptitud del genotipo. El experimento 8.2.4 muestra que un incremento en el número de repeticiones del algoritmo mejora el progreso del algoritmo genético, debido a que la evaluación del genotipo es más precisa. Sin embargo, el aumento del número de repeticiones también conlleva a un crecimiento notorio en los tiempos de computación insumidos por el algoritmo. Teniendo en cuenta que la mejora en el desempeño del algoritmo genético con respecto al número de repeticiones del entrenamiento se hace cada vez menos apreciable, se determinó el mejor valor de esta última variable para lograr un compromiso tiempo-performance.

5. ¿De qué forma se puede mejorar la capacidad de generalización de las redes neuronales generadas?

En primer lugar, se planteó una forma de poda de la red (pruning), la regularización de la complejidad, como un modo de fomentar que los pesos excedentes, que tienen poca o ninguna influencia sobre la red, asuman valores cercanos a cero, y por lo tanto mejoren la generalización. En el experimento 8.2.5 se empleó el método de degradación de pesos (weight decay), y se observó sus efectos al variar el parámetro de regu-

larización. Se pudo apreciar que el uso de este método produjo una mejora en las redes resultantes del proceso evolutivo en cuanto al aprendizaje y a la generalización.

En segundo lugar, se planteó un método de validación cruzada, la detección temprana (early stopping) del entrenamiento back-propagation, como un modo de evitar que la red sobreajuste a los datos de entrenamiento. En el experimento 8.2.6 se utilizaron distintos valores para el parámetro de detención, que determina el momento en que se para el entrenamiento, determinándose el valor óptimo. En este experimento también se pudo visualizar una mejora de las redes neuronales generadas, que fueron capaces de aprender mejor y clasificar mejor.

6. ¿Cómo deberían evaluarse a las redes neuronales en lo que respecta a su habilidad de aprendizaje y generalización?

Para verificar que una red neuronal, de hecho, tiene una buena topología específica al dominio, el set de datos se particionó en tres partes de igual tamaño de manera que las redes neuronales resultantes del AG puedan analizarse en sus habilidades para tratar con nuevos datos. Primero, se usó $SE_{AG/RN}$ en el paso del AG. Entonces, las redes neuronales resultantes del AG se entrenaron usando $SE_{Entrenamiento}$. Finalmente, estas redes entrenadas clasificaron a SE_{Test} para obtener una evaluación de sus capacidades de generalización. Para asegurarse que cualquier tendencia en los resultados son, de hecho, tendencias, y no sólo ciertas "casualidades" en los datos, se utilizó una validación cruzada, consistente en alternar en sus funciones a cada partición del set de datos.

El algoritmo híbrido AG/RN tuvo éxito en disminuir el costo (error) de las topologías, pero la búsqueda de topologías no es del todo eficiente, ya que los resultados dependen de las muchas condiciones iniciales, como el número y tamaño de las capas ocultas.

La clave para mejorar la habilidad del sistema AG/RN es cambiar la forma en que se codifican las redes neuronales. La implementación actual requiere se especifiquen de antemano el número máximo de nodos y capas ocultas. Las implementaciones futuras de este sistema deberían explorar la codificación indirecta de las topologías.

Otro aspecto que debería examinarse con más profundidad es cómo se debería determinar el costo de una topología. En la implementación actual, el costo es simplemente el error de entrenamiento de la red neuronal sobre una partición del set de datos. La pregunta es si ésta es la mejor forma de determinar la aptitud de una topología. La virtud más importante del criterio que utilizamos es su manejo matemático sencillo. Sin embrago, en muchas situaciones que se dan en la práctica, la minimización de la función de costo corresponde a optimizar una cantidad intermedia que no es el objetivo final del sistema, y por lo tanto puede conducir a una performance subóptima. Por ejemplo, en sistemas para mercados financieros, la meta final de un inversionista es maximizar la expectativa de retorno a mínimo riesgo [Choey y Wiegend, 1996]. La relación *recompensa-volatilidad* como medida de performance del *retorno con riesgo ajustado* es intuitivamente más atractiva que la aptitud estándar.

Otro paso muy importante sería repetir los experimentos presentados aquí para set de datos diferentes. Como mencionamos previamente, la escalabilidad es un problema importante en las implementaciones de redes neuronales, por lo tanto sería interesante ver cómo la implementación actual escala a redes grandes que contienen miles de entradas.

Por último, otro punto que debería ser explorado es la paralelización del algoritmo genético, especialmente considerando los tiempos elevados de procesamiento que se manejaron durante los experimentos. Excepto en la fase de selección, donde hay competencia entre los individuos, las únicas interacciones entre los miembros de la población ocurren durante la fase de reproducción, y solo son necesarios dos padres para engendrar un hijo. Cualquier otra operación de la evolución, en particular la evaluación de cada miembro de la población, puede hacerse separadamente. Por lo tanto, casi todas las operaciones en un AG son implícitamente paralelas. El uso de computadoras que trabajan en paralelo no solo provee más espacio de almacenamiento sino también permite el uso de múltiples procesadores para producir y evaluar más soluciones en menos tiempo. Al paralelizar el algoritmo, es posible incrementar el tamaño de la población, reducir el costo computacional y mejorar entonces la performance del AG. Los algoritmos genéticos paralelos o PGA (Parallel Genetic Algorithms) constituyen un área reciente de investigación, y existen enfoques muy interesantes como el modelo de islas o el modelo de grano fino [Hue, 1997].

Bibliografía

Amari S., Murata N., Muller R., Finke M. Y Yang H. (1996) *Statistical theory of overtraining - Is cross-validation asymptotically effective?* Advances in Neural Information Processing Systems, vol. 8, pp. 176-182: MIT Press.

Blake C. L. y Merz C. J. (1998) *UCI Repository of machine learning databases.* http://www.ics.uci.edu/~mlearn/MLRepository.html Irvine, CA: University of California, Department of Information and Computer Science.

Choey M. y Weigend A. (1996) *Nonlinear trading models through sharp ratio maximization*. Decision Technologies for Financial Engineering, pp. 3-22, Singapore.

Davis L. (1991) *Handbook of genetic algorithms.* Van Nostrand Reinhold, New York, Academic Press.

Dow R. J. y Sietsma J. (1991) *Creating Artificial Neural Networks that generalize*. Neural Networks, vol. 4, no. 1, pp. 198-209.

García Martínez Ramón (1997) *Sistemas Autónomos. Aprendizaje Automático*. Nueva Librería.

García Martínez, R. y Borrajo, D. (2000) *An Integrated Approach of Learning, Planning and Executing.* Journal of Intelligent and Robotic Systems. Volumen 29, Número 1, Páginas 47-78. Kluwer Academic Press. 2000

Goldberg D. E. (1991) *A comparative analysis of selection schemes used in genetic algorithms*. In Gregory Rawlins, editor. Foundations of Genetic Algorithms, pages 69-93, San Mateo, CA: Morgan Kaufmann Publishers.

Hancock P. J (1992) *Genetic algorithms and permutation problems: A comparison of permutation operators for neural net structure specification*. Proc. of the Int'l Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN-95), pages 108-122.

Haykin Simon (1999) *Neural Networks. A Comprehensive Foundation*. Second Edition. Pretince Hall.

Hertz J., A. Krogh y R. Palmer (1991) *Introduction to the Theory of Neural Computation*. Reading, MA: Addison-Wesley.

Hilera J. y Martínez V. (1995) *Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones*. RA-MA, Madrid.

Hinton G. E. (1989) *Connectionist Learning Procedures*. Artificial Intelligence, vol. 40, pp. 185-234

Holland J. H. (1975) *Adaptation in Natural and Artificial Systems*. University of Michigan Press (Ann Arbor).

Holland, J. H. (1980) *Adaptive algorithms for discovering and using general patterns in growing knowledge-based.* International Journal of Policy Analysis and Information Systems, 4(3), 245-268.

Holland, J. H. (1986) *Escaping brittleness: The possibilities of general purpose learning algorithms applied in parallel rule-based systems*. Morgan Kaufmann.

Holland, J. H., Holyoak, K. J., Nisbett, R. E., & Thagard, P. R. (1987). *Classifier systems, Q-morphisms, and induction*. In L. Davis (Ed.), Genetic algorithms and simulated annealing (pp. 116-128).

Honavar V. and L. Uhr. (1993) *Generative Learning Structures and Processes for Generalized Connectionist Networks*. Information Sciences, 70:75--108.

Hubert R. Mayer H. A. (1995) *A parallel system generating problem adapted topologies of artificial neural networks by means of genetic algorithms*. In Proceedings of Lernen der GI-Fachgruppe, Dormunt, Germany.

Hue Xavier (1997) *Genetic Algorithms for Optimization*. Edinburgh Parallel Computing Centre. The University of Edinburgh.

Hush D. R. y Horne B. G. (1993) *Progress in supervised neural networks: What's new since Lippmann?* IEEE Signal Processing Magazine, vol. 10, pp. 8-39.

Jacobs R. A. (1988) *Increased rates of convergence through learning rate adaptation.* Neural Networks, vol. 1, pp. 295-307.

Karthik Balakrishnan (1998) *Evolutionary design of neural architectures: A preliminary taxonomy*. Artificial Intelligence Research Group. Department of Computer Science. Iowa State University.

Kitano H. (1990) *Designing neural networks using genetics algorithms with graph generation system*. Complex Systems, vol. 4, no. 4, pp. 461-476.

Kramer A. H. y Sangiovanni-Vincentelli A. (1989) *Efficient parallel learning algorithms for neural networks.* Advances in neural Information Processing Systems, vol 1, pp. 40-48, San Mateo, CA: Morgan Kaufmann.

LeCun Y. (1989) *Generalization and network design strategies*. Technical Report CRG-TR-89-4, Department of Computer Science, University of Toronto, Canada.

LeCun Y. (1993) *Efficient Learning and Second-order Methods*. A Tutorial at NIPS 93, Denver, CO.

Michalski R. S., Carbonell J. G. y Mitchell T. M. (1986) *Machine Learning: An Artificial Intelligence Approach, Volume II*. Morgan Kaufmann.

Michalewicz Z. (1996) *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag. Heindelberg, third edition.

Miller G., Todd P. M. y Hedge S. U. (1989) *Designing neural networks using genetic algorithms.* In Proc. Of the Third Int'l Conf. On Genetic Algorithms and Their Applications (J. D. Schaffer, ed.), pp. 379-384.

Miller B. L. y Goldberg D. E. (1995) *Genetic Algorithms, selection schemes and the varying effect of noise*. IlliGAL report No. 95009.

Parker D. B. (1985) *Learning-logic: Casting the cortex of the human brain in silicon.* Technical Report TR-47, Center for computational Research in Economics and Management Science, Cambridge, MA. MIT Press.

Poggio T. y Girosi F.(1990) *Networks for approximation and learning*. Proceedings of the IEEE, vol. 78, pp. 1481-1497.

Prechelt L. (1998) *Automatic early stopping using cross validation*. Neural Networks 11, IEEE Computer Society Press

Rich E. y Knight K. (1991) *Introduction to Artificial Networks*. MacGraw-Hill Publications.

Rosemblatt M. (1958) *The Perceptron: A probabilistic model for information storage and organization in the brain*. Psychological Review, vol. 65, pp. 386-408.

Rumelhart D. E. y McClelland J. L. (1986) *Parallel Distributed Processing: Explorations in the Microstructure of cognition.* Vol. 1, Cambridge, MA: MIT Press.

Russel S. and Norvig P. (1995) *Artificial Intelligence – A Modern Approach*. Pretince-Hall, Englewood Cliffs, NJ.

Sauders G. M. y Angeline P. J. (1994) *An Evolutionary Algorithm that constructs Recurrent Neural Networks*. IEEE Trans. on Neural Networks, vol. 5, no. 1, pp. 54-65.

Schaffer J. D., Caruana R. A. and Eshelman L. J. (1990) *Using genetic search to exploit the emergent behavior of neural networks.* Physics D, vol. 42, pp. 244-248

Schmidt M. (1996) *GA to train Fuzzy Logic Rulebases and Neural Networks.* FLAMOC'96.

Stone M. (1974) *Cross-validatory choice and assessment of statistical predictions*. Journal of the Royal Statistical Society, vol. B36, pp. 111-133.

Sutton R. S. (1984) *Temporal credit assignment in reinforcement learning.* Ph. D. Dissertation, University of Massachusetts, MA.

Watrous R. L. (1987) *Learning algorithms for connectionist networks: Applied gradient methods of nonlinear optimization.* First IEEE International Conference on Neural Networks, vol. 2, pp. 619-627, San Diego, CA.

Weigend A. S., Rumelhart D. E. y Huberman D. (1991) *Generalization by weight-elimination with application to forecasting*. Advances in Neural Information Processing Systems, vol. 3, pp. 875-882, San Mateo, CA.

Wieland A. y Leighton R. (1987) *Geometric analysis of neural networks capabilities.* First IEEE International Conference on Neural Networks, vol. III, pp. 385-392, San Diego, CA.

Wolpert D. H. y Macready W. G. (1995) *No Free Lunch Theorems for Search*. Technical Report SFI-TR-95-02-010, Santa Fe Institute.

Yao Xin (1999) *Evolving Artificial Neural Networks. School of Computer Science*. The University of Birmingham. B15 2TT.

Yao X. y Liu Y. (1998) *Toward Designing Artificial Neural Networks by Evolution*. Applied Mathematics and Computation, 91(1): 83-90.

Apéndice A: Sobre la programación

Descripción de los archivos

Los algoritmos utilizados en la experimentación de este trabajo fueron implementados utilizando el lenguaje de programación C++. Para lograr una modularización del software para la realización del sistema híbrido, se crearon los siguientes archivos:

- **Estructdatos.h**: Contiene la definición de ciertas constantes del algoritmo híbrido, como son los valores límite para el número de neuronas en cada capa, las mutaciones, los parámetros de aprendizaje, las penalidades de la función de aptitud, la selección y la reproducción. Si se modifican estos valores se puede cambiar la forma en que se comporta la evolución.

- **Prototipos.h**: Contiene la declaración de las funciones implementadas en el archivo Híbrido.cpp

- **Híbrido.cpp**: Este archivo constituye el corazón del sistema. Contiene el algoritmo híbrido que realiza el proceso evolutivo para generar topologías de red neuronal, definiendo los operadores genéticos para la selección, la reproducción, la mutación y el reemplazo.

- **RN.h**: Contiene la declaración de la clase RN que encapsula una red neuronal feedforward con aprendizaje back-propagation.

- **RN.cpp**: Contiene la definición de la clase RN. Para la construcción de la topología, recibe un conjunto de matrices que determinan los nodos de cada capa de la red y las conexiones entre las mismas. Permite evaluar la aptitud de la red una cantidad determinada de veces, y continuar el entrenamiento aunque la performance del set validación no mejore por un número de iteraciones también preestablecido.

- **Problema.h**: Contiene la declaración de las variables y las funciones para el archivo Problema.cpp

- **Problema.cpp**: Contiene las funciones específicas del problema en cuestión. Permite generar listas que contienen los patrones de entrenamiento y de validación a partir de los ejemplos de solicitudes de tarjetas de crédito.

- **Matriz.h**: Declaración de la clase Matriz que encapsula un template de matriz.

- **Matriz.cpp**: Definición de la clase Matriz. Esta clase permite implementar una matriz conteniendo cualquier tipo de elementos.

El anexo se encuentra dividido por estos archivos fuente. Por último se puede encontrar un índice analítico de funciones que facilita la ubicación rápida de cualquier función del sistema dentro de este anexo.

Archivo Estructdatos.h

Archivo: Estructdatos.h

Descripción: Definiciones de constantes, variables externas y estructuras de datos.

```
/*  Estructdatos.h  */

/* Header del AG para topologias de red neuronal */

// definiciones

#define FALSE 0
#define TRUE  1

#define PENALIDAD_OCULTAS 1
#define PENALIDAD_CONEX 10

#define MIN_NO_DE_OCULTAS 1
#define MAX_NO_DE_OCULTAS 50

#define MAX_NO_DE_ENTRADAS  128
#define MAX_NO_DE_SALIDAS 128

#define ETA 0.1
#define ALFA 0.5

#define INIC_MAX      1.00
#define INIC_MIN      0.00
#define INIC_MUTACION_MAX 0.100
#define INIC_MUTACION_MIN 0.003

#define MUTACIONES_IH_MAX 0.100
#define MUTACIONES_IH_MIN 0.005
#define MUTACIONES_HH_MAX 0.100
#define MUTACIONES_HH_MIN 0.005
#define MUTACIONES_HO_MAX 0.100
#define MUTACIONES_HO_MIN 0.005

#define CORTE_MIN 0.25

#define TAMANIOTORNEO 3

#define LOG

// variables globales

extern int entradas;
extern int salidas;
extern char *nombreproblema;

/* Estructura del genotipo */

typedef struct genotipo {
  int id;
  int en_uso;
  float error;
```

```
  int *no_de_ocultas;
  unsigned char ih_conex[MAX_NO_DE_ENTRADAS][MAX_NO_DE_OCULTAS];
  unsigned char ***hh_conex;
  unsigned char ho_conex[MAX_NO_DE_OCULTAS][MAX_NO_DE_SALIDAS];
  long semilla;
  float pesosInic;
  int no_de_conex;
} GENOTIPO;

struct geno_estadisticas {
    int mejorGenoId;
    float menorError;
};

/* Estructura del fenotipo */

typedef struct nodo {
  double h;
  double estado;
  double *peso;
  double *delta_peso;
  double *peso_guardado;
  double delta;
} NODO;

/* Estructura para la representacion de los patrones */

typedef struct patron {
  float *pat;
  int cl;
  struct patron *proximo;
} PATRON;

/* Fin de Estructdatos.h */
```

Archivo Prototipos.h

Archivo: Prototipos.h

Descripción: Declaración de funciones para el archivo Híbrido.cpp

```
// Prototipos.h

void cerrarLogFile(void);
void cerrarPobFile(void);
void evalError(void);
int  encontrar_N_mejores(int n);
void error(GENOTIPO *la_red);
void inic_AG(void);
void inversion(GENOTIPO *geno);
void marcarGeno(GENOTIPO *geno, int marcando);
void mutacion(GENOTIPO *hijo);
void abrirLogFile(void);
void abrirPobFile(void);
int  randInt(int desde, int hasta);
int  leer_geno(GENOTIPO *geno, int lock, int genoId);
int imprimir_datos_prob (void);
void recombinacion(GENOTIPO *madre, GENOTIPO *padre, GENOTIPO *hijo);
void reemplazo(void);
void reemp_torneo(GENOTIPO *hijo);
void seleccionarPadres(void);
void selec_torneo(GENOTIPO **padre_o_madre);
void escribir_geno(GENOTIPO *geno);
void generar_poblacion(void);

// Fin de Prototipos.h
```

Archivo Hibrido.cpp

Archivo: Hibrido.cpp

Descripción: Declaración de variables e inclusión de archivos.

```
//  Hibrido.cpp

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <limits.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include "Estructdatos.h"
#include "Prototipos.h"
#include "Problema.h"
#include "RN.h"
#include "Matriz.cpp"

// Variables globales

int tamanio_pob = 20;
int entradas = 47;
int salidas = 2;
int capas = 1;
int *no_de_ocultas;
int gen=0;
int maxGen=100;
int eval_error = 5;
int fallas_valid = 5;
char *nombreproblema=”credito”;
GENOTIPO *padre,*madre,*hijo1,*hijo2;
GENOTIPO *selec_Candidatos[TAMANIOTORNEO];
GENOTIPO *reemp_Candidatos[TAMANIOTORNEO];
FILE *pobfile;
FILE *logfile;
```

Función: main()

Archivo: Hibrido.cpp

Llama a: abrirLogFile(); generar_población(); inic_AG(); leer_geno(); error(); escribir_geno(); seleccionarPadres(); recombinacion(); mutacion(); evalError(); reemplazo(); cerrarLogFile()

Descripción: Contiene al algoritmo híbrido. En primer lugar, genera la población inicial y entrena a cada uno de los individuos. Luego se desarrolla el ciclo evolutivo, donde se suceden la selección, la recombinación, la mutación, la evaluación y el reemplazo de los individuos, durante un número especificado de generaciones.

```
//  main

int main (int argc, char *argv[])
{
    abrirLogFile();
    generar_poblacion();
    inic_AG();

//entrenar primera generacion
     int p = leer_geno(madre,TRUE,0);
     while (p<tamanio_pob) {
                error(madre);
                escribir_geno(madre);
                p = leer_geno(madre,TRUE,++p);
           }

   //ciclo principal
   for (gen=0; gen<maxGen; gen++) {
                seleccionarPadres();
                recombinacion(madre,padre,hijo1);
                recombinacion(padre,madre,hijo2);
                mutacion(hijo1);
                mutacion(hijo2);
                evalError();
                reemplazo();
           }

    cerrarLogFile();
    return 0;
} // main
```

Función: cerrarLogFile()

Archivo: Hibrido.cpp

Descripción: Cierra el archivo de log utilizado para registrar los eventos principales durante la ejecución del algoritmo híbrido.

```
void cerrarLogFile(void)
{
   fprintf(logfile, "Termino normalmente\n");
   fclose(logfile);
} // cerrarLogFile
```

Función: cerrarPobFile()

Archivo: Hibrido.cpp

Descripción: Cierra el archivo de población donde se almacenan los individuos utilizados por el algoritmo híbrido.

```
void cerrarPobFile(void)
{
    fclose(pobfile);

} // cerrarPobFile
```

Función: evalError()

Archivo: Hibrido.cpp

Llama a: error()

Descripción: Realiza la evaluación del error de ambos hijos generados por el algoritmo híbrido tras la recombinación de sus padres.

```
void evalError(void)
{
   error(hijo1);
   error(hijo2);
} // evalError
```

Función: encontrar_N_mejores()

Archivo: Hibrido.cpp

Llama a: abrirPobFile(); cerrarPobFile()

Entrada: Un entero que representa el número de individuos con el menor error que se quiere encontrar dentro de la población.

Salida: Un entero que representa el Id del individuo con menor error de la población.

Descripción: Localiza los *n* individuos con menor error dentro de la población. Es útil para llevar estadísticas acerca de la evolución de la población a través de las sucesivas generaciones.

```
int encontrar_N_mejores(int n)
{
 char auxiliar[MAX_NO_DE_OCULTAS*entradas + MAX_NO_DE_OCULTAS*salidas + ca-
  pas*MAX_NO_DE_OCULTAS + 20];
 char prueba_str[20];
 struct geno_estadisticas *geno_estadisticas_array;
 int auxiliarint;
 float error;

 geno_estadisticas_array = new (struct geno_estadisticas)[n];

 for (int i=0; i<n; i++){
  geno_estadisticas_array[i].mejorGenoId = 0;
  geno_estadisticas_array[i].menorError = -FLT_MAX;
 }

 // Encuentra al mejor individuo para hacer estadisticas
    abrirPobFile();
    for (int i=0; i<tamanio_pob; i++){
          sprintf(prueba_str,"geno%d",i);
          fscanf(pobfile, "%s", auxiliar);
          while (strcmp(auxiliar,prueba_str)!=0)
             fscanf(pobfile, "%s", auxiliar);
          fscanf(pobfile, "%d", &auxiliarint);
          fscanf(pobfile, "%f", &error);

          if (error < geno_estadisticas_array[n-1].menorError)
             for (int a=n-1; a>=0; a--)
                   if ((error > geno_estadisticas_array[a-1].menorError)||(a==0)) {
                      geno_estadisticas_array[a].menorError = error;
                      geno_estadisticas_array[a].mejorGenoId = i;
                      break;
                   }
                   else {
                        geno_estadisticas_array[a].menorError = geno_estadisticas_array[a-
                        1].menorError;
                        geno_estadisticas_array[a].mejorGenoId = geno_estadisticas_array[a-
                        1].mejorGenoId;
                   }
    }
```

```
    cerrarPobFile();

   printf("-------------------------------------------------------\n");
   for (int m=0; m<n; m++)
     printf("%2d mejor red, id %2d, error %8.5f\n"
         ,m+1,geno_estadisticas_array[m].mejorGenoId,geno_estadisticas_array[m].menorError);
   printf("-------------------------------------------------------\n");
  }

  return geno_estadisticas_array[n-1].mejorGenoId;
} // encontrar_N_mejores
```

Función: imprimir_datos_prob()

Archivo: Hibrido.cpp

Descripción: Imprime en el log los datos principales del problema que se está resolviendo, incluyendo los valores del tamaño de la población, el número de entradas, salidas y capas ocultas de las redes que se generan a partir del algoritmo híbrido.

```
int imprimir_datos_prob()
{
   fprintf(logfile, "Tamanio de poblacion = %d\n",tamanio_pob);
   fprintf(logfile, "Entradas = %d\n",entradas);
   fprintf(logfile, "Salidas = %d\n",salidas);
   fprintf(logfile, "Capas = %d\n",capas);
   fprintf(logfile, "Problema = %s\n\n",nombreproblema);
   fflush(logfile);
  return 0;
} // imprimir_datos_prob
```

Función: error()

Archivo: Hibrido.cpp

Llama a: Matriz::Matriz(); Matriz::set(); RN::RN(); RN::pesosInic(); RN::entrenarValidBP()

Entrada: El genotipo (individuo) que será entrenado para obtener su error.

Descripción: Realiza la evaluación del error del genotipo. En primer lugar, se crea una matriz que representa al genotipo, luego se genera una red neuronal (fenotipo) a partir de la matriz, por último se entrena a dicha red neuronal una cantidad dada de veces. Cada entrenamiento provee un valor de error, se guarda el menor error y la semilla que dio origen a los pesos iniciales del entrenamiento correspondiente.

```
void error(GENOTIPO *geno)
{
   Matriz<int> **mLista;
   mLista = new (Matriz<int> *)[capas+1];

   mLista[0] = new Matriz<int>(entradas, geno->no_de_ocultas[0]);
   for(int i=0; i<capas-1; i++)
         mLista[i+1] = new Matriz<int>(geno->no_de_ocultas[i], geno->no_de_ocultas[i+1]);
   mLista[capas] = new Matriz<int> (geno->no_de_ocultas[capas-1], salidas);

   for (int i=0; i<entradas; i++)
         for (int j=0; j<geno->no_de_ocultas[0]; j++)
            mLista[0]->set(i,j, geno->ih_conex[i][j]);

   for (int k=0; k<capas-1; k++)
         for (int i=0; i<geno->no_de_ocultas[k]; i++)
            for (int j=0; j<geno->no_de_ocultas[k+1]; j++)
                  mLista[k+1]->set(i,j, geno->hh_conex[k][i][j]);


   for (int i=0; i<geno->no_de_ocultas[capas-1]; i++)
         for (int j=0; j<salidas; j++)
            mLista[capas]->set(i,j, geno->ho_conex[i][j]);

   float mejorValid = 0;
   float menorError = 0;

   for (int i=0; i<eval_error; i++){

         RN laRN(capas+2, mLista);

         laRN.eta = ETA;
         laRN.alpha = ALFA;

         long semilla;
         time(&semilla);
         laRN.pesosInic(geno->pesosInic,semilla);
```

```
        float aptitud = ((float) laRN.entrenarValidBP(no_de_entrenam, patrones_entrenam,
        sal_deseadas_entrenam, no_de_valid, patrones_valid, sal_deseadas_valid, fallas_valid,
        1)*100) / no_de_valid;

        int sum_ocultas = 0;
        for (int j=0; j<capas; j++)
           sum_ocultas += geno->no_de_ocultas[j];

        geno->error = 1 / (aptitud
                - (PENALIDAD_OCULTAS * ((double) sum_ocultas/ (dou-
                ble)MAX_NO_DE_OCULTAS))/capas
                - (PENALIDAD_CONEX * ((double) geno->no_de_conex / ((double) (entradas +
                salidas)* MAX_NO_DE_OCULTAS)))/capas);



        fprintf(logfile, "     Validacion:  %4.2f\n", aptitud);
        fprintf(logfile, "     Error:     %4.2f\n", geno->error);
        fflush(logfile);

        if (geno->error < menorError){
           menorError = geno->error;
           mejorValid = aptitud;
           geno->semilla = semilla;
        }
   }

   fprintf(logfile, "\n     Mejor Validacion:  %4.2f\n", mejorValid);
   fprintf(logfile, "     Menor Error:      %4.2f\n", menorError);
   fflush(logfile);

   for(int i=0; i<capas+1; i++)
        delete mLista[i];
   delete[] mLista;

} // error
```

Función: inic_AG()

Archivo: Hibrido.cpp

Llama a: imprimir_datos_prob(); inic_patrones()

Descripción: Realiza la inicialización del algoritmo genético necesaria para comenzar el ciclo evolutivo. Primero se genera la semilla de números aleatorios, luego se reserva memoria para cada uno de los padres y de los hijos, y por último se cargan los patrones a utilizarse en el entrenamiento de las redes neuronales generadas.

```
void inic_AG()
{
   long ahora;

   // genera la semilla
   srand(time(&ahora));
   srandom(time(&ahora));

   imprimir_datos_prob();

   // malloc de individuos
   padre   = new (struct genotipo);
   madre   = new (struct genotipo);
   hijo1   = new (struct genotipo);
   hijo2   = new (struct genotipo);
   madre->hh_conex = new (unsigned char **)[capas-1];
   padre->hh_conex = new (unsigned char **)[capas-1];
   hijo1->hh_conex = new (unsigned char **)[capas-1];
   hijo2->hh_conex = new (unsigned char **)[capas-1];
   for (int i=0; i<capas-1; i++){
         madre->hh_conex[i] = new (unsigned char *)[MAX_NO_DE_OCULTAS];
         padre->hh_conex[i] = new (unsigned char *)[MAX_NO_DE_OCULTAS];
         hijo1->hh_conex[i] = new (unsigned char *)[MAX_NO_DE_OCULTAS];
         hijo2->hh_conex[i] = new (unsigned char *)[MAX_NO_DE_OCULTAS];
         for (int j=0; j<MAX_NO_DE_OCULTAS; j++){
            madre->hh_conex[i][j] = new (unsigned char)[MAX_NO_DE_OCULTAS];
            padre->hh_conex[i][j] = new (unsigned char)[MAX_NO_DE_OCULTAS];
            hijo1->hh_conex[i][j] = new (unsigned char)[MAX_NO_DE_OCULTAS];
            hijo2->hh_conex[i][j] = new (unsigned char)[MAX_NO_DE_OCULTAS];
         }
   }

   madre->no_de_ocultas = new int[capas-1];
   padre->no_de_ocultas = new int[capas-1];
   hijo1->no_de_ocultas = new int[capas-1];
   hijo2->no_de_ocultas = new int[capas-1];

   // malloc de candidatos para torneo
   for (int m=0; m<TAMANIOTORNEO; m++){
         selec_Candidatos[m] = new (struct genotipo);
         reemp_Candidatos[m] = new (struct genotipo);
         selec_Candidatos[m]->hh_conex = new (unsigned char **)[capas-1];
         reemp_Candidatos[m]->hh_conex = new (unsigned char **)[capas-1];
```

```
        for (int i=0; i<capas-1; i++){
            selec_Candidatos[m]->hh_conex[i] = new (unsigned char *)[MAX_NO_DE_OCULTAS];
            reemp_Candidatos[m]->hh_conex[i] = new (unsigned char
*)[MAX_NO_DE_OCULTAS];
            for (int j=0; j<MAX_NO_DE_OCULTAS; j++){
                selec_Candidatos[m]->hh_conex[i][j] = new (unsigned
char)[MAX_NO_DE_OCULTAS];
                reemp_Candidatos[m]->hh_conex[i][j] = new (unsigned
char)[MAX_NO_DE_OCULTAS];
            }
        }
        selec_Candidatos[m]->no_de_ocultas = new int[capas-1];
        reemp_Candidatos[m]->no_de_ocultas = new int[capas-1];
   }

   inic_patrones();

} // inic_AG
```

Función: inversion()

Archivo: Hibrido.cpp

Entrada: El genotipo (individuo) al que se le aplicará el operador de inversión.

Descripción: Alterna a dos nodos al azar, con todas sus conexiones, en cada capa oculta del genotipo.

```
void inversion(GENOTIPO *geno)
{
   //// Entrada a Ocultas
   int izquierda1  = random()%geno->no_de_ocultas[0];
   int derecha1 = random()%geno->no_de_ocultas[0];

   for(int i=0; i<entradas; i++){
         int temp = geno->ih_conex[i][izquierda1];
         geno->ih_conex[i][izquierda1] = geno->ih_conex[i][derecha1];
         geno->ih_conex[i][derecha1] = temp;
   }

   //// Ocultas a Ocultas
   for (int l=1; l<capas; l++){
         // Intercambia la segunda mitad de los nodos en el nivel l-1.
         for(int i=0; i<geno->no_de_ocultas[l]; i++){
            int temp = geno->hh_conex[l-1][izquierda1][i];
            geno->hh_conex[l-1][izquierda1][i] = geno->hh_conex[l-1][derecha1][i];
            geno->hh_conex[l-1][derecha1][i] = temp;
         }

         // Selecciona dos nodos para invertir
         int izquierda2  = random()%geno->no_de_ocultas[l];
         int derecha2 = random()%geno->no_de_ocultas[l];

         // Intercambia la segunda mitad de los nodos en el nivel l.
         for(int j=0; j<geno->no_de_ocultas[l-1]; j++){
            int temp = geno->hh_conex[l-1][j][izquierda2];
            geno->hh_conex[l-1][j][izquierda2] = geno->hh_conex[l-1][j][derecha2];
            geno->hh_conex[l-1][j][derecha2] = temp;
         }

         // Para la proxima iteracion
         izquierda1  = izquierda2;
         derecha1 = derecha2;
   }

   //// Ocultas a Salida
   for(int j=0; j<salidas; j++){
         int temp = geno->ho_conex[izquierda1][j];
         geno->ho_conex[izquierda1][j] = geno->ho_conex[derecha1][j];
         geno->ho_conex[derecha1][j] = temp;
   }
}
```

Función: marcarGeno()

Archivo: Hibrido.cpp

Llama a: abrirPobFile(); cerrarPobFile()

Entrada: El genotipo (individuo) que será marcado o desmarcado

Descripción: La marcación o desmarcación del genotipo indica si el mismo está en uso por el AG, o se encuentra libre, respectivamente.

```
void marcarGeno(GENOTIPO *geno, int marcando)
{
   char auxiliar[MAX_NO_DE_OCULTAS*entradas + MAX_NO_DE_OCULTAS*salidas +20];
   char prueba_str[20];

   abrirPobFile();

   fscanf(pobfile, "%s", auxiliar);

   sprintf(prueba_str,"geno%d",geno->id);

   rewind(pobfile);
   fscanf(pobfile, "%s", auxiliar);
   while (strcmp(auxiliar,prueba_str)!=0) {
         fscanf(pobfile, "%s", auxiliar);
   }

   if (marcando)
         fprintf(pobfile, " 1 ");
   else
         fprintf(pobfile, " 0 ");

   fflush(pobfile);
   cerrarPobFile();
} // marcarGeno
```

Función: mutacion()

Archivo: Hibrido.cpp

Entrada: El genotipo (individuo) al que se le aplicará el operador de mutación.

Descripción: La mutación alterna un número aleatorio de conexiones. La tasa de mutación es diferente para las conexiones entre las capas de entrada y las ocultas, entre ocultas, y entre las ocultas y la de salida. La tasa de mutación también varía a lo largo de las sucesivas generaciones entre un valor máximo y un valor mínimo preestablecidos.

```
void mutacion(GENOTIPO *hijo)
{
   int mutacionesIH, mutacionesHH, mutacionesHO;

   // Entrada a Ocultas y Ocultas a Salida
   if (gen<maxGen/2){
         mutacionesIH = (int)(entradas*hijo-
         >no_de_ocultas[0]*MUTACIONES_IH_MAX*((maxGen/2-gen)/(maxGen/2))  + entra-
         das*hijo->no_de_ocultas[0]*MUTACIONES_IH_MIN);
         mutacionesHO = (int)(hijo->no_de_ocultas[capas-
         1]*salidas*MUTACIONES_HO_MAX*((maxGen/2-gen)/(maxGen/2)) + hijo-
         >no_de_ocultas[capas-1]*salidas*MUTACIONES_OH_MIN);
   }
   else {
          mutacionesIH = (int)(entradas*hijo->no_de_ocultas[0]*MUTACIONES_IH_MIN) + 1;
          mutacionesHO = (int)(hijo->no_de_ocultas[capas-1]*salidas*MUTACIONES_HO_MIN) +
          1;
   }

   while(mutacionesIH--){
         int i = random()%entradas;
         int h = random()%hijo->no_de_ocultas[0];
         hijo->ih_conex[i][h] = 1 - hijo->ih_conex[i][h];
   }

   while(mutacionesHO--){
         int h = random()%hijo->no_de_ocultas[capas-1];
         int o = random()%salidas;
         hijo->ho_conex[h][o] = 1 - hijo->ho_conex[h][o];
   }

   // Ocultas a Ocultas
   for (int i=0; i<capas-1; i++) {
         int hhConex = hijo->no_de_ocultas[i]*hijo->no_de_ocultas[i+1];
         if (gen<maxGen/2)
            mutacionesHH = (int)(hhConex*MUTACIONES_HH_MAX);
         else
            mutacionesHH = (int)(hhConex*MUTACIONES_HH_MIN);

         while(mutacionesHH--){
            int a = random()%hijo->no_de_ocultas[i];
            int b = random()%hijo->no_de_ocultas[i+1];
            hijo->hh_conex[i][a][b] = 1 - hijo->hh_conex[i][a][b];
```

```
            }
    }
} // mutacion
```

Función: abrirLogFile()

Archivo: Hibrido.cpp

Descripción: Abre el archivo de log utilizado para registrar los eventos principales durante la ejecución del algoritmo híbrido.

```
void abrirLogFile()
{
   if ((logfile = fopen(“registro.log”, "w")) == NULL) {
         fprintf(stderr, "No se puede abrir archivo: registro.log\n");
         exit(-1);
   }
} // abrirLogFile
```

Función: abrirPobFile()

Archivo: Hibrido.cpp

Descripción: Abre el archivo de población donde se almacenan los individuos utilizados por el algoritmo híbrido.

```
void abrirPobFile()
{
   if ((pobfile = fopen(“pobfile”, "r+")) == NULL) {
         fprintf(logfile, "No se puede abrir archivo: %s\n", “pobfile.pob”);
         exit(-1);
   }
} // abrirPobFile
```

Función: randInt()

Archivo: Hibrido.cpp

Entrada: El rango de enteros en el cual se desea obtener un número aleatorio.

Salida: Un número entero aleatorio dentro del rango correspondiente.

Descripción: Se utiliza para generar números al azar dentro de un rango determinado.

```
int randInt(int desde, int hasta)
{
   return desde + (int)((hasta-desde)*(rand()/(RAND_MAX+1.0)));
} // randInt
```

Función: leer_geno()

Archivo: Hibrido.cpp

Entrada: El genotipo (individuo) que se quiere leer de la población. Si genoId vale –1, el genotipo se elige al azar, sino se lee el genotipo correspondiente a este Id. Si lock está activado, se marca al individuo para indicar que se encuentra en uso por el AG.

Salida: El Id del genotipo que fue leído.

Descripción: Realiza la lectura del genotipo desde el archivo de población. Se leen todos sus valores almacenados, incluyendo el Id, los límites para los pesos iniciales que determinan el menor error, la semilla para generar dichos pesos, y las conexiones entre las capas de entrada, ocultas y de salida. Por último se imprimen en el log las estadísticas para el genotipo recién leído.

```
int leer_geno(GENOTIPO *geno, int lock, int genoId)
{
   char auxiliar[MAX_NO_DE_OCULTAS*entradas + MAX_NO_DE_OCULTAS*salidas + ca-
     pas*MAX_NO_DE_OCULTAS + 20];
   char prueba_str[20];
   long pos=0;
   int j,b;
   int tempId;

   // Encuentra un individuo random que no este en uso
   geno->en_uso = TRUE;
   abrirPobFile();

   while(geno->en_uso) {
         if ((genoId == -1)
            geno->id = random()%tamanio_pob;
         else if (genoId == tamanio_pob)
            return genoId;
         else
            geno->id = genoId++;

         sprintf(prueba_str,"geno%d",geno->id);


         fscanf(pobfile, "%s", auxiliar);
         while (strcmp(auxiliar,prueba_str)!=0) {
            fscanf(pobfile, "%s", auxiliar);
         }

         pos = ftell(pobfile);
         fscanf(pobfile, "%d", &geno->en_uso);
         fscanf(pobfile, " %f", &geno->error);
   }

#ifdef LOG
   // Imprime el id del individuo encontrado
   fprintf(logfile, "  Leyendo geno%d\n", geno->id);
```

```
    fflush(logfile);
#endif

    // Se marca o desmarca al individuo en uso
    fseek(pobfile, pos, SEEK_SET);
    fflush(pobfile);
    if (lock)
        fprintf(pobfile, " 1 ");
    else
        fprintf(pobfile, " 0 ");
    fflush(pobfile);

    // Lee el error
    fscanf(pobfile, "%f ", &geno->error);

    // Lee el numero de nodos ocultos
    for (int p=0; p<capas; p++){
        fscanf(pobfile, "%d", &geno->no_de_ocultas[p]);
    }

    // Lee la semilla
    fscanf(pobfile, "%ld", &geno->semilla);

    // Lee los limites para los pesos iniciales
    fscanf(pobfile, "%f", &geno->pesosInic);

    // Lee las conexiones desde la entrada hasta la primer capa oculta
   int ihConex = 0;
      unsigned char temp;
      for (int i=0; i<entradas; i++){
          for (j=0; j<geno->no_de_ocultas[0]/8; j++){
              temp = fgetc(pobfile);
              for (int k=0; k<8; k++){
                    geno->ih_conex[i][8*j+k] = temp & 1;
                    temp = temp>>1;
                    if (geno->ih_conex[i][8*j+k])
                        ihConex++;
              }
          }
          if (geno->no_de_ocultas[0]%8){
              temp = fgetc(pobfile);
              for (int k=0; k<geno->no_de_ocultas[0]%8; k++){
                    geno->ih_conex[i][8*j+k] = temp & 1;
                    temp = temp>>1;
                    if (geno->ih_conex[i][8*j+k])
                        ihConex++;
              }
          }
      }

    // Lee las conexiones entre las capas ocultas
    int hhConex = 0;
    for(int l=0; l<capas-1; l++){
          for (int a=0; a<geno->no_de_ocultas[l]; a++){
              for (b=0; b<geno->no_de_ocultas[l+1]/8; b++){
                    temp = fgetc(pobfile);
```

```
                    for (int k=0; k<8; k++){
                        geno->hh_conex[l][a][8*b+k] = temp & 1;
                        temp = temp>>1;
                        if (geno->hh_conex[l][a][8*b+k])
                            hhConex++;
                    }
                }
                if (geno->no_de_ocultas[l+1]%8){
                    temp = fgetc(pobfile);
                    for (int k=0; k<geno->no_de_ocultas[l+1]%8; k++){
                        geno->hh_conex[l][a][8*b+k] = temp & 1;
                        temp = temp>>1;
                        if (geno->hh_conex[l][a][8*b+k])
                            hhConex++;
                    }
                }
            }
        }

    // Lee las conexiones desde la ultima capa oculta hasta la salida
    int hoConex = 0;
    for (int i=0; i<geno->no_de_ocultas[capas-1]; i++){
            for (j=0; j<salidas/8; j++){
                temp = fgetc(pobfile);
                for (int k=0; k<8; k++){
                    geno->ho_conex[i][8*j+k] = temp & 1;
                    temp = temp>>1;
                    if (geno->ho_conex[i][8*j+k])
                        hoConex++;
                }
            }
            if (salidas%8){
                temp = fgetc(pobfile);
                for (int k=0; k<salidas%8; k++){
                    geno->ho_conex[i][8*j+k] = temp & 1;
                    temp = temp>>1;
                    if (geno->ho_conex[i][8*j+k])
                        hoConex++;
                }
            }
    }

    cerrarPobFile();
    geno->no_de_conex = hoConex + ihConex + hhConex;

    int maxHHConex = 0;
    for(int m=0; m<capas-1; m++)
            maxHHConex += geno->no_de_ocultas[m]*geno->no_de_ocultas[m+1];

#ifdef LOG
    // Imprime estadisticas sobre el individuo recien leido
    for (int i=0; i<capas; i++)
            fprintf(logfile, "    ocultas%d:        %d\n", i, geno->no_de_ocultas[i]);
    fprintf(logfile, "    Error:        %4.2f\n", geno->error);
    fprintf(logfile, "    Inic:         %4.2f\n", geno->pesosInic);
    fprintf(logfile, "    entrada - ocultas:   %d \n", ihConex);
```

```
    fprintf(logfile, "    ocultas - ocultas:  %d \n", hhConex);
    fprintf(logfile, "    ocultas - salida:  %d \n", hoConex);
    fflush(logfile);
#endif

return genoId;
} // leer_geno
```

Función: recombinacion()

Archivo: Hibrido.cpp

Llama a: inversion(); randInt()

Entrada: El genotipo de los padres, a los cuales se les va a aplicar el operador de recombinación, y el genotipo del hijo, donde se colocará el material genético obtenido.

Descripción: Realiza el proceso de reproducción entre los padres para generar un nuevo hijo. Primero se aplica el operador de inversión a ambos padres para intercambiar material genético bueno entre las distintas partes de cada individuo. Luego se realiza un corte aleatorio en cada padre separando en dos partes a los nodos de las capas ocultas. La restricción aquí es que la parte más pequeña tenga al menos un cuarto del total de los nodos del individuo. Luego se utilizan las partes obtenidas de cada padre para crear un nuevo hijo. Por último se calcula el límite para los pesos iniciales del hijo como un promedio de los valores que tienen los padres, que es modificado con un número aleatorio dentro un cierto rango.

```
void recombinacion(GENOTIPO *madre, GENOTIPO *padre, GENOTIPO *hijo)
{
   // Aplica las inversiones

   inversion(madre);
   inversion(padre);

   // Realiza los cortes
   int padreCorte[capas];
   int madreCorte[capas];
   for (int i=0; i<capas; i++){
         padreCorte[i]=MAX_NO_DE_OCULTAS+1;
         madreCorte[i]=0;
         while (padreCorte[i] + madre->no_de_ocultas[i] - madreCorte[i] >
         MAX_NO_DE_OCULTAS) {
            padreCorte[i] = randInt((int)(CORTE_MIN*padre->no_de_ocultas[i]), (int)((1- COR-
            TE_MIN)*padre->no_de_ocultas[i]));
            madreCorte[i] = randInt((int)(CORTE_MIN*madre->no_de_ocultas[i]), (int)((1-
            CORTE_MIN)*madre->no_de_ocultas[i]));
         }
   }

   ///// Entrada a Ocultas
   for(int i=0; i<entradas; i++) {
         for(int h=0; h<padreCorte[0]; h++)
            hijo->ih_conex[i][h] = padre->ih_conex[i][h];

         for(int h=madreCorte[0]; h<madre->no_de_ocultas[0]; h++)
            hijo->ih_conex[i][h-madreCorte[0]+padreCorte[0]] = madre->ih_conex[i][h];
   }
   hijo->no_de_ocultas[0] = padreCorte[0] + madre->no_de_ocultas[0] - madreCorte[0];
```

```
   ///// Ocultas a Ocultas
   for (int i=0; i<capas-1; i++)
         hijo->no_de_ocultas[i] = padreCorte[i] + madre->no_de_ocultas[i] - madreCorte[i];

   // Hace las conexiones hacia adelante
   for (int l=0; l<capas-1; l++){
         for(int i=0; i<padreCorte[l]; i++)
            for(int j=0; j<padre->no_de_ocultas[l+1] && j<hijo->no_de_ocultas[l+1]; j++)
                  hijo->hh_conex[l][i][j] = padre->hh_conex[l][i][j];

         for(int i=madreCorte[l]; i<madre->no_de_ocultas[l]; i++)
            for(int j=0; j<madre->no_de_ocultas[l+1] && j<hijo->no_de_ocultas[l+1]; j++)
                  hijo->hh_conex[l][padreCorte[l]+i-madreCorte[l]][j] = madre->hh_conex[l][i][j];
   }

   // Hace las conexiones hacia atras
   for (int l=1; l<capas; l++){
         for(int i=0; i<padreCorte[l]; i++)
            for(int j=0; j<padre->no_de_ocultas[l-1] && j<hijo->no_de_ocultas[l-1]; j++)
                  hijo->hh_conex[l-1][j][i] = padre->hh_conex[l-1][j][i];

         for(int i=madreCorte[l]; i<madre->no_de_ocultas[l]; i++)
            for(int j=0; j<madre->no_de_ocultas[l-1] && j<hijo->no_de_ocultas[l-1]; j++)
                  hijo->hh_conex[l-1][j][padreCorte[l]+i-madreCorte[l]] = madre->hh_conex[l-1][j][i];
   }

   ///// Ocultas a Salida
   for(int h=0; h<padreCorte[capas-1]; h++)
         for(int o=0; o<salidas; o++)
                  hijo->ho_conex[h][o] = padre->ho_conex[h][o];

   for(int h=madreCorte[capas-1]; h<madre->no_de_ocultas[capas-1]; h++)
         for(int o=0; o<salidas; o++)
                  hijo->ho_conex[h-madreCorte[capas-1]+padreCorte[capas-1]][o] = madre-
>ho_conex[h][o];
   hijo->no_de_ocultas[capas-1] = padreCorte[capas-1] + madre->no_de_ocultas[capas-1] - ma-
dreCorte[capas-1];

   // Recombinacion de los limites para los pesos iniciales
   float inicMut;

   if (gen<maxGen/2)
         inicMut = INIC_MUTACION_MAX*((maxGen/2-gen)/(maxGen/2)) +
         INIC_MUTACION_MIN;
   else
         inicMut = INIC_MUTACION_MIN;

   float inicTemp;
   if (random()&1)
         inicTemp = madre->pesosInic;
   else
         inicTemp = padre->pesosInic;
   // float inicTemp = (madre->pesosInic + padre->pesosInic)/2;
   float inicTemp2 = inicTemp + ((float)((random()%((int)(2*inicMut*10000)))-
     inicMut*10000))/10000.0;
   while(inicTemp2<INIC_MIN || inicTemp2>INIC_MAX){
```

```
            inicTemp2 = inicTemp + ((float)((random()%((int)(2*inicMut*10000)))-
            inicMut*10000))/10000.0;
        }
        hijo->pesosInic = inicTemp2;

} // recombinacion
```

Función: reemplazo()

Archivo: Hibrido.cpp

Llama a: marcarGeno(); reemp_torneo()

Descripción: Inserta a los hijos generados en el ciclo evolutivo dentro de la población, reemplazando a otros individuos con mayores errores.

```
void reemplazo(void)
{
    marcarGeno(madre,FALSE);
    marcarGeno(padre,FALSE);

   fprintf(logfile, "Insertando hijo1:\n");
   fflush(logfile);
   reemp_torneo(hijo1);
   fprintf(logfile, "Insertando hijo2:\n");
   fflush(logfile);
   reemp_torneo(hijo2);

} // reemplazo
```

Función: reemp_torneo()

Archivo: Hibrido.cpp

Llama a: escribir_geno()

Entrada: El genotipo (individuo) que reemplazará a otro genotipo elegido mediante la técnica de torneo.

Descripción: Primero elije tantos individuos al azar de la población como lo indica el tamaño del torneo. Luego escoge al individuo con mayor error del torneo y lo reemplaza con el individuo recién generado por el proceso evolutivo. El nuevo individuo se desmarca para indicar que está liberado y puede ser elegido por el proceso de selección en las futuras generaciones.

```
void reemp_torneo(GENOTIPO *hijo)
{
   float    error=FLT_MAX;
   int      errorIndice=0;

   for (int i=0; i<TAMANIOTORNEO; i++){
          leer_geno(reemp_Candidatos[i],FALSE,-1);
          if (reemp_Candidatos[i]->error > error){
             error = reemp_Candidatos[i]->error;
             errorIndice = i;
          }
   }
   hijo->id = reemp_Candidatos[errorIndice]->id;
   escribir_geno(hijo);
   fprintf(logfile, "geno%d reemplazado\n\n",hijo->id);
   fflush(logfile);
} // reemp_torneo
```

Función: seleccionarPadres()

Archivo: Hibrido.cpp

Llama a: selec_torneo()

Descripción: Selecciona de la población al padre y a la madre que se utilizarán en el proceso evolutivo para generar dos hijos.

```
void seleccionarPadres(void)
{
	fprintf(logfile, "Seleccionando madre:\n");
	selec_torneo(&madre);
	fprintf(logfile, "Seleccionando padre:\n");
	selec_torneo(&padre);

} // seleccionarPadres
```

Función: selec_torneo()

Archivo: Hibrido.cpp

Llama a: leer_geno(); marcarGeno()

Entrada: Un puntero al genotipo (individuo) que resulte seleccionado mediante la técnica de torneo.

Descripción: Primero elige tantos individuos al azar de la población como lo indica el tamaño del torneo. Luego escoge al individuo con menor error del torneo y lo marca como en uso para que no pueda ser elegido nuevamente; este individuo será utilizado como uno de los padres durante el proceso evolutivo.

```
void selec_torneo(GENOTIPO **padre_o_madre)
{
   float    error=-1.0;
   int      errorIndice=0;

   for (int i=0; i<TAMANIOTORNEO; i++){
          leer_geno(selec_Candidatos[i],FALSE,-1);
          if (selec_Candidatos[i]->error < error){
             error = selec_Candidatos[i]->error;
             errorIndice = i;
          }
   }

   fprintf(logfile, "Genotipo seleccionado geno%d\n\n",(*padre_o_madre)->id);
   fflush(logfile);

   marcarGeno(*padre_o_madre,TRUE);
} // selec_torneo
```

Función: generar_poblacion()

Archivo: Hibrido.cpp

Llama a: abrirPobFile(); cerrarPobFile()

Descripción: Genera la población inicial correspondiente a la primera generación. La cantidad de individuos se fija a un valor preestablecido. Los individuos generados tendrán una densidad de conexiones entre capas que varía en forma aleatoria dentro de un rango determinado.

```
void generar_poblacion(void)
{
   long ahora;


   // Inicializar funcion random
   srand(time(&ahora));
   srandom(time(&ahora));

  abrirPobFile();

  no_de_ocultas = new int[capas];


   for (int i=0; (i<<tamanio_pob); i++) {
          fprintf(pobfile, "geno%d ", i);
          fprintf(pobfile, "0 ");
          fprintf(pobfile, "%20f ", 0.0);

          for (int j=0; j<capas; j++){
             no_de_ocultas[j] = rand()%(MAX_NO_DE_OCULTAS-
             MIN_NO_DE_OCULTAS)+MIN_NO_DE_OCULTAS;
             fprintf(pobfile,"%3d ", no_de_ocultas[j]);
          }

          // Semilla
          struct timeval *tv;
          struct timezone *tz = NULL;
          tv = (struct timeval *)malloc(sizeof(struct timeval));
          gettimeofday(tv,tz);
          fprintf(pobfile, "%20ld ", tv->tv_usec+tv->tv_sec);

          // pesosInic
          fprintf(pobfile, "%1.4f ",0.5*(((float)rand())/RAND_MAX));

          // densidad inicial
          long conex_dens = (int)(rand()*0.8) + (int)(0.1*RAND_MAX);

          // conexiones Entrada a Ocultas
          for(int j=0; j<entradas; j++){
            for(int k=0; k<no_de_ocultas[0]/8; k++){
              temp = (rand()>conex_dens)?1:0;
              for (int q=0; q<7; q++){
```

```
            temp = 2*temp;
            if (rand()>conex_dens)
              temp++;
      }
      fprintf(pobfile,"%c", (unsigned char)temp);

    }
    if(no_de_ocultas[0]%8){
      temp = (rand()>conex_dens)?1:0;
      for (int q=0; q<no_de_ocultas[0]%8-1; q++){
            temp = 2*temp;
            if (rand()>conex_dens)
              temp++;
      }
      fprintf(pobfile,"%c", (unsigned char)temp);
    }
  }

  // conexiones Ocultas a Ocultas
  for(int l=0; l<capas-1; l++){
     for(int j=0; j<no_de_ocultas[l]; j++){
          for(int k=0; k<no_de_ocultas[l+1]/8; k++){
            temp = (rand()>conex_dens)?1:0;
            for (int q=0; q<7; q++){
                  temp = 2*temp;
                  if (rand()>conex_dens)
                    temp++;
            }
            fprintf(pobfile,"%c", (unsigned char)temp);
          }
          if(no_de_ocultas[l+1]%8){
            temp = (rand()>conex_dens)?1:0;
            for (int q=0; q<no_de_ocultas[l+1]%8-1; q++){
                  temp = 2*temp;
                  if (rand()>conex_dens)
                    temp++;
            }
            fprintf(pobfile,"%c", (unsigned char)temp);
          }
     }

  }

  // conexiones Ocultas a Salida
  for(int j=0; j<no_de_ocultas[capas-1]; j++){
    for(int k=0; k<salidas/8; k++)
      temp = (rand()>conex_dens)?1:0;
      for (int q=0; q<7; q++){
            temp = 2*temp;
            if (rand()>conex_dens)
              temp++;
      }
      fprintf(pobfile,"%c", (unsigned char)temp);
    if(salidas%8){
      temp = (rand()>conex_dens)?1:0;
      for (int q=0; q<salidas%8-1; q++){
```

```
                    temp = 2*temp;
                    if (rand()>conex_dens)
                      temp++;
                }
                fprintf(pobfile,"%c", (unsigned char)temp);
              }
            }
      }
      cerrarPobFile();
  }

} /* generar_poblacion */
```

Función: escribir_geno()

Archivo: Hibrido.cpp

Llama a: abrirPobFile(); cerrarPobFile()

Entrada: El genotipo (individuo) que se quiere escribir en la población.

Descripción: Realiza la escritura del genotipo en el archivo de población, reemplazando al viejo genotipo que tenía su mismo Id. Se escriben todos los valores de relevancia, que incluyen el Id, los límites para los pesos iniciales que determinan el menor error, la semilla para generar dichos pesos, y las conexiones entre las capas de entrada, ocultas y de salida.

```
void escribir_geno(GENOTIPO *geno)
{
   char prueba_str[256];
   char comando[256];
   char nombre_tempFile[256];
   int tempChar;
   int matchPuntero = 0;
   int match = FALSE;
   FILE *tempFile;
   int o;

#ifdef LOG
   // Imprime el id del individuos
   fprintf(logfile, "  Escribiendo geno%d\n", geno->id);
   fflush(logfile);
#endif

   sprintf(prueba_str,"geno%d",geno->id);

   abrirPobFile();

   sprintf(nombre_tempFile, "pobarch.tmp");

   if ((tempFile = fopen(nombre_tempFile, "w")) == NULL) {
         fprintf(logfile, "No se puede abrir archivo %s para escritura\n", nombre_tempFile);
         exit(-1);
   }

   // Copia la primer parte del archivo de poblacion
   while (!match){
         tempChar = fgetc(pobfile);
         fputc(tempChar, tempFile);
         if (prueba_str[matchPuntero] == tempChar){
            if (++matchPuntero == (int)strlen(prueba_str))
                  match = TRUE;
         }
         else
            matchPuntero = 0;
   }
```

```
// Escribe el Nuevo genotipo
fprintf(tempFile, " 0 %20f ", geno->error);
for (int i=0; i<capas; i++)
        fprintf(tempFile, "%3d ", geno->no_de_ocultas[i]);
fprintf(tempFile, "%20ld ", geno->semilla);
fprintf(tempFile, "%f ", geno->pesosInic);
fprintf(tempFile, "%.4f %.4f ", geno->eta, geno->alpha);

//// conexionesEntrada a Ocultas
unsigned char temp;
for (int i=0; i<entradas; i++) {
        int h;
        for (h=0; h<geno->no_de_ocultas[0]/8; h++) {
           temp = geno->ih_conex[i][h*8+7];
           for (int k=6; k>=0; k--){
                   temp = temp<<1;
                   temp += geno->ih_conex[i][h*8+k];
           }
           fputc(temp,tempFile);
        }
        if (geno->no_de_ocultas[0]%8){
           temp = geno->ih_conex[i][h*8+(geno->no_de_ocultas[0]%8)-1];
           for (int k=(geno->no_de_ocultas[0]%8)-2; k>=0; k--){
                   temp = temp<<1;
                   temp += geno->ih_conex[i][h*8+k];
           }
           fputc(temp,tempFile);
        }
}

//// Conexiones Ocultas a Ocultas
for (int l=0; l<capas-1; l++)
        for (int i=0; i<geno->no_de_ocultas[l]; i++) {
           int h;
           for (h=0; h<geno->no_de_ocultas[l+1]/8; h++) {
                   temp = geno->hh_conex[l][i][h*8+7];
                   for (int k=6; k>=0; k--){
                      temp = temp<<1;
                      temp += geno->hh_conex[l][i][h*8+k];
                   }
                   fputc(temp,tempFile);
           }
           if (geno->no_de_ocultas[l+1]%8){
                   temp = geno->hh_conex[l][i][h*8+(geno->no_de_ocultas[l+1]%8)-1];
                   for (int k=(geno->no_de_ocultas[l+1]%8)-2; k>=0; k--){
                      temp = temp<<1;
                      temp += geno->hh_conex[l][i][h*8+k];
                   }
                   fputc(temp,tempFile);
           }
        }

//// Conexiones Ocultas a Salida
for (int h=0; h<geno->no_de_ocultas[capas-1]; h++) {
        for (o=0; o<salidas/8; o++) {
           temp = geno->ho_conex[h][o*8+7];
```

```
            for (int k=6; k>=0; k--){
                temp = temp<<1;
                temp += geno->ho_conex[h][o*8+k];
            }
            fputc(temp,tempFile);
        }
        if (salidas%8){
            temp = geno->ho_conex[h][o*8+(salidas%8)-1];
            for (int k=(salidas%8)-2; k>=0; k--){
                temp = temp<<1;
                temp += geno->ho_conex[h][o*8+k];
            }
            fputc(temp,tempFile);
        }
    }

    // Se saltea al genotipo
    sprintf(prueba_str,"geno%d",geno->id+1);

    match = FALSE;
    matchPuntero = 0;
    while (!match && (tempChar = fgetc(pobfile))!=EOF){
        if (prueba_str[matchPuntero] == tempChar){
            if (++matchPuntero == (int)strlen(prueba_str))
                match = TRUE;
        }
        else
            matchPuntero = 0;
    }

    // Copia la ultima parte del archivo de poblacion
    if (match){
        fprintf(tempFile,"\n\n");
        fprintf(tempFile,prueba_str);
        while ((tempChar = fgetc(pobfile))!=EOF)
            fputc(tempChar, tempFile);
    }

    // Actualiza los archivos
    fflush(tempFile);
    cerrarPobFile();
    fclose(tempFile);

    sprintf(comando,"mv %s pobarch",nombre_tempFile);
    system(comando);
} // escribir_geno

// Fin de Hibrido.cpp
```

Archivo RN.h

Archivo: RN.h

Descripción: Declaración de la clase RN que encapsula una red neuronal feedforward con aprendizaje back-propagation, implementada en RN.cpp

```
//  RN.h

#ifndef RN_H
#define RN_H

#include "Matriz.h"

#define DEFAULT_ALPHA 0.5;
#define DEFAULT_ETA 0.1;

#define REGULARIZ_DEFAULT 0.1

class RN {

public:

   RN(int capas, Matriz<int> **conexiones);
   ~RN();
   void pesosInic(float pesoInic, int randomSemilla);
   void eval(float *entrada);
   void bp(float *entrada, int salida_deseada);
   void entrenarBP(int patConteo, float **patrones, int *sal_deseadas, int iteraciones=1, int alter-
     nar=1);
   int RN::entrenarValidBP(int patConteo, float **patrones, int *sal_deseadas,
                           int validPatConteo, float **patrones_valid, int *sal_deseadas_valid,
                           int maxReducciones=5, int alternar=1);
   int  test(int patConteo, float **patrones, int *sal_deseadas);
   void imprimir(FILE *archivo);
   void imprimir_salida(FILE *archivo);


private:

   // Para el Backpropagation
   float alpha;       //  Momentum
   float eta;         //  Tasa de aprendizaje
   float regulariz;   // Regularizacion con weight decay

   // Constantes de escala para la función de activacion  f(x) = A * tanh(S * x).
   float A;
   float S;

   int debugConteo;
   float tan_deriv(float x);
   int no_Capas;      // Incluye capa de entrada, ocultas y salida.
   float **h;         // Flujo de entrada flow (para cada nodo en cada capa que no sea la de entra-
     da)
   float **act;       // Activaciones  (para cada nodo en cada capa)
   float **d;         // Delta  (para cada nodo en cada capa que no sea la de entrada)
```

```
    float **dda;       // d^2E/da^2    (para cada nodo en cada capa que no sea la de entrada)
    float **bias;       // Umbral (para cada nodo en cada capa que no sea la de entrada)
    float **dbias;      // Delta umbral (para cada nodo en cada capa que no sea la de entrada)
    int **inWC;         // Numero de pesos de entrada (para cada nodo en cada capa que no sea la
      de entrada)
    int ***enW;         // Lista de indices dentro del array de pesos (para cada nodo en cada capa
      que no sea la de entrada)
    int ***enPos;       // Lista de indices dentro de la capa mas baja (igualando los pesos de entra-
        da correspondientes)
    Matriz<int> **conex;  // conexiones (entre dos capas) como indices para el vector de pesos
    float **w;          // Vector de Pesos
    int *wc;            // Conteo de pesos (wc[i] = longitud de w[i])
    float **dw;         // Vector de Delta pesos
    int **dwc;          // Numero de conexiones que comparten el peso correspondiente.

};

#endif

// Fin de RN.h
```

Archivo RN.cpp

Archivo: RN.cpp

Descripción: Inclusión de archivos.

```
// RN.cpp


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "RN.h"
#include "Matriz.cpp"
#include "Estructdatos.h"
#include "Problema.h"
```

Función: RN()

Archivo: RN.cpp

Entrada: Recibe el número total de capas (incluyendo las capas de entrada y de salida) y un array de matrices ('número de capas'-1 en total) que especifican las conexiones.

Descripción: El constructor de la clase. Reserva memoria para implementar la estructura de una red neuronal, incluyendo las capas y las conexiones entre las mismas.

```
RN::RN (int capas, Matriz<int> **conexiones) {

   no_Capas = capas;
   conex = conexiones;

   // valores default para el Backpropagation
   alpha  = DEFAULT_ALPHA;
   eta    = DEFAULT_ETA;

   // Constantes de escala para la función de activacion
   A = 1.7159;
   S = 2.0/3.0;

   // Reservando memoria
   h      = new (float *)[no_Capas];
   act    = new (float *)[no_Capas];
   d      = new (float *)[no_Capas];
   bias   = new (float *)[no_Capas];
   dbias  = new (float *)[no_Capas];
   bias_hkk = new (float *)[no_Capas];

   dda     = new (float *)[no_Capas];

   w      = new (float *)[no_Capas - 1];
   wc     = new int      [no_Capas - 1];
   dw     = new (float *)[no_Capas - 1];
   viejo_dw = new (float *)[no_Capas - 1];
   dwc    = new (int *)  [no_Capas - 1];

   hkk     = new (float *)[no_Capas - 1];

   inWC   = new (int *)  [no_Capas - 1];
   enW    = new (int **) [no_Capas - 1];
   enPos  = new (int **) [no_Capas - 1];

   h[0]=0;
   act[0]=0;
   d[0]=0;
   bias[0]=0;
   dbias[0]=0;
   bias_hkk[0]=0;
   dda[0] = 0;
```

```
// Encuentra los indices de pesos mas altos entre las conexiones
// y crea los vectores de pesos.
for (int l=1; l<no_Capas; l++) {
        // Cheque los tamanios de la matriz:
        //   Para cada capa oculta, en las dos matrices
        //   tiene que concordar el numero de nodos.
        if (l<no_Capas-1 && (conex[l-1]->obtenerAncho() != conex[l]->obtenerAlto())) {
           fprintf(stderr, "RN: Error en la dimension de la matriz de conexion (entre capa %d y
           %d)\n", l-1, l);
           fprintf(stderr, "RN: Ancho(%d) = %d.  Alto(%d) = %d.\n", l-1, conex[l-1]-
           >obtenerAncho(), l, conex[l]->obtenerAlto());
           exit(1);
        }

        // Reserva espacio para los nodos (entrada, activacion, delta, delta_pesos)
        int ancho = conex[l-1]->obtenerAncho();
        int alto = conex[l-1]->obtenerAlto();

        h[l]     = new float[ancho];
        act[l]   = new float[ancho];
        d[l]     = new float[ancho];
        bias[l]  = new float[ancho];
        dbias[l] = new float[ancho];
        bias_hkk[l] = new float[ancho];

        dda[l] = new float[ancho];

        inWC[l-1]  = new int[ancho];
        enW[l-1]   = new (int *)[ancho];
        enPos[l-1] = new (int *)[ancho];

        // Encuentra el maximo indice.
        int max=0;
        for (int j=0; j<ancho; j++) {
           inWC[l-1][j]=0;
           for (int i=0; i<alto; i++) {
              int idx = conex[l-1]->obtener(i,j);
                 if (idx > 0) {
                    inWC[l-1][j]++; // Otra conexion hacia el nodo j

                    if (idx>max) {
                           max = idx;
                    }
                 }
           }
        }

        // Genera indices unicos (>max) en lugar de 1's.
        for (int j=0; j<ancho; j++) {
           // Reserva W[l][j] y enPos[l][j] para todo j.
           enW[l-1][j] = new int[inWC[l-1][j]];
           enPos[l-1][j] = new int[inWC[l-1][j]];
           int k=0;
           for (int i=0; i<alto; i++) {
                  int c = conex[l-1]->obtener(i,j);
                  if (c>0) {
```

```
                    if (c==1) {
                        conex[l-1]->set(i,j,++max);
                        c = max;
                    }
                    enW[l-1][j][k] = c;
                    enPos[l-1][j][k] = i;
                    k++;
                }
            }
        }
        // Conteo de pesos (el ultimo indice de pesos + 1 en esta capa)
        wc[l-1] = max + 1;


        // Reserva el espacio para los pesos actuales.
        w[l-1] = new float[wc[l-1]];
        w[l-1][0] = 0.0;
        w[l-1][1] = 0.0;
        dw[l-1] = new float[wc[l-1]];
        viejo_dw[l-1] = new float[wc[l-1]];
        dwc[l-1] = new int[wc[l-1]];

        hkk[l-1]    = new float[wc[l-1]];

        for (int i=0; i<wc[l-1]; i++) {
            dw[l-1][i] = 0.0;
            viejo_dw[l-1][i] = 0.0;
            dwc[l-1][i] = 0;

            hkk[l-1][i] = 0.0;
        }

        for (int i=0; i<alto; i++) {
            for (int j=0; j<ancho; j++) {
                dwc[l-1][conex[l-1]->obtener(i,j)]++;
            }
        }

    } // for (l=1; l<no_Capas; l++)

} // RN
```

Función: ~RN()

Archivo: RN.cpp

Descripción: Destructor de la clase. Libera la memoria reservada en el constructor.

```
RN::~RN () {
   delete[] h[0];
   delete[] act[0];

   for (int l=1; l<no_Capas; l++) {
         delete[] h[l];
         delete[] act[l];
         delete[] d[l];
         delete[] bias[l];
         delete[] dbias[l];

         delete[] w[l-1];
         delete[] dw[l-1];
         delete[] viejo_dw[l-1];
         delete[] dwc[l-1];

         int ancho = conex[l-1]->obtenerAncho();
         for (int j=0; j<ancho; j++) {
            delete[] enW[l-1][j];
            delete[] enPos[l-1][j];
         }

         delete[] enW[l-1];
         delete[] enPos[l-1];
         delete[] inWC[l-1];

         delete[] hkk[l-1];
   }

   delete[] h;
   delete[] act;
   delete[] d;
   delete[] bias;
   delete[] dbias;

   delete[] w;
   delete[] wc;
   delete[] dw;
   delete[] viejo_dw;
   delete[] dwc;

   delete[] inWC;
   delete[] enW;
   delete[] enPos;

   delete[] hkk;

} // ~RN
```

Función: pesosInic()

Archivo: RN.cpp

Entrada: El límite para los valores de los pesos iniciales, y el valor de la semilla de números aleatorios.

Descripción: Inicializa los pesos al azar en el rango [-pesoInic ; pesoInic] utilizando la semilla correspondiente.

```
void RN::pesosInic(float pesoInic, int randomSemilla) {

        srand(randomSemilla);
        for (int l=0; l<no_Capas-1; l++) {
           for (int i=2; i<wc[l]; i++) {
                w[l][i] = (float) (2*pesoInic * rand()/(RAND_MAX+1.0)) - pesoInic;
           }
           int ancho = conex[l]->obtenerAncho();
           for (int i=0; i<ancho; i++) {
                bias[l+1][i] = (float) (2*pesoInic * rand()/(RAND_MAX+1.0)) - pesoInic;
                dbias[l+1][i] = 0;
           }
        }

} // pesosInic
```

Función: tan_deriv()

Archivo: RN.cpp

Entrada: Un valor real al que se le aplicará la derivada de la tangente hiperbólica.

Descripción: Calcula la derivada de la tangente hiperbólica de x: f(x)=A*tanh(S*x) como parte del algoritmo back-propagation.

```
float RN::tan_deriv(float x) {
    float tmp=tanh(S*x);
    return A*(1-tmp*tmp)*S;
} // tan_deriv()
```

Función: eval()

Archivo: RN.cpp

Entrada: Un array de entradas a la red

Descripción: Dado un conjunto de entradas, la función de activación se propaga a través de la red y se calcula la salida.

```
void RN::eval(float *entrada) {

   act[0] = entrada;

   // propaga a traves de la red
   for (int l=0; l<no_Capas-1; l++) {

         int ancho = conex[l]->obtenerAncho();

         for (int j=0; j<ancho; j++) {

            // Calcula la entrada al j-esimo nodo
            h[l+1][j] = 0.0;
            for (int i=0; i<inWC[l][j]; i++) {
               int p = enPos[l][j][i];
               int wIdx = enW[l][j][i];
               h[l+1][j] += act[l][p] * w[l][wIdx];
            }

            h[l+1][j] += bias[l+1][j];

            // Aplica la funcion de activacion
            act[l+1][j] = (float) A*tanh((double)S*h[l+1][j]);
         }
   }
} // eval
```

Función: bp()

Archivo: RN.cpp

Llama a: RN::tan_deriv()

Entrada: Una patrón de entrada y su salida deseada.

Descripción: Dado un patrón y su salida correspondiente, se ajustan los pesos usando el algoritmo back-propagation por gradiente descendente.

```
void RN::bp(float *entrada, int salida_deseada) {

   // Propaga la señal hacia adelante
   eval(entrada);

   // 1. Calcula el Error (deltas)
   //    ...sobre la capa de salida
   int ancho = conex[no_Capas-2]->obtenerAncho();
   for (int i=0; i<ancho; i++) {
         d[no_Capas-1][i] = tan_deriv(h[no_Capas-1][i]) * (((salida_deseada==i)?1:-1) -
         act[no_Capas-1][i]);
   }


   //    ...sobre la capa oculta
   for (int l=no_Capas-2; l>=1; l--) {

         int alto = conex[l]->obtenerAlto();
         int ancho = conex[l]->obtenerAncho();

         // Inicializa las deltas...
         for (int i=0; i<alto; i++) {
            d[l][i] = 0;
         }

         // Actualiza las deltas (solo considerando los pesos que estan presentes)
         for (int j=0; j<ancho; j++) {              // el nodo superior
            int pesoConteo = inWC[l][j];
            for (int i=0; i<pesoConteo; i++) {    // hasta el numero de pesos entrantes
                  int wIdx = enW[l][j][i];         // Indice de los pesos en el array de pesos
                  int p = enPos[l][j][i];          // Indice del nodo correspondiente al peso actual
                  d[l][p] += w[l][wIdx] * d[l+1][j]; // Contribucion al delta
            }
         }

         // Termina las deltas...
         for (int i=0; i<alto; i++) {
            d[l][i] *= tan_deriv(h[l][i]);
         }

   }

   // 2. Ajuste de pesos y umbrales
```

```
    // Intercambio entre  dw y viejo_dw.
    //  dw se inicializa en 0's en el constructor y se resetea despues de cada epoch
    float **tmp = viejo_dw;
    viejo_dw = dw;
    dw = tmp;

    for (int l=no_Capas-2; l>=0; l--) {

            int ancho = conex[l]->obtenerAncho();
            for (int j=0; j<ancho; j++) {
               for (int i=0; i<inWC[l][j]; i++) {
                       int p = enPos[l][j][i];
                       int c = enW[l][j][i];
                       dw[l][c] += d[l+1][j] * act[l][p];
               }
               dbias[l+1][j] = eta * d[l+1][j] * (-1) + alpha * dbias[l+1][j];
               bias[l+1][j] += dbias[l+1][j];
            }


            for (int i=2; i<wc[l]; i++) {

               dw[l][i] *= eta;                 // multiplica por la tasa de aprendizaje
               dw[l][i] /= dwc[l][i];        // promedia sobre todos las actualizaciones de pesos
               dw[l][i] += alpha * viejo_dw[l][i]; // agrega el momentum
               w[l][i] += dw[l][i];             // la actualizacion del peso
               viejo_dw[l][i] = 0.0;               // se resetea y se preparar para el proximo epoch
            }
    }
}  // bp()
```

Función: entrenarBP()

Archivo: RN.cpp

Llama a: RN::bp()

Entrada: El número de patrones a utilizar, dichos patrones, sus salidas respectivas, el número de entrenamientos a realizar, y la indicación de alternar o no los patrones.

Descripción: Dados ciertos patrones ('patConteo' en total), la función entrena a la red usando la función bp(). El entrenamiento se repite un número de veces según 'iteraciones'. Si 'alternar' esta seteado, se alterna el orden de los patrones entre cada iteración.

```
void RN::entrenarBP(int patConteo, float **patrones, int *sal_deseadas, int iteraciones=1, int al-
ternar=1) {

   for (int it=0; it<iteraciones; it++) {
         if (alternar) {

            // Crea e inicializa el array de orden
            int orden[patConteo];
            for (int i=0; i<patConteo; i++) {
                  orden[i]=i;
            }

            // Alterna el contenido del array de ‘orden'
            // Para cada i...
            for (int i=0; i<patConteo; i++) {
                  // ...elije al azar un j intercambiando el contenido de orden[i] y orden[j].
                  int j = rand() % patConteo;
                  int tmp = orden[i];
                  orden[i] = orden[j];
                  orden[j] = tmp;
            }

            // Entrena con cada patron en el orden encontrado
            for (int mu=0; mu<patConteo; mu++) {
                  bp(patrones[orden[mu]], sal_deseadas[orden[mu]]);
            }

         } else {
            // Entrena con cada patron en el orden dado
            for (int mu=0; mu<patConteo; mu++) {
                  bp(patrones[mu], sal_deseadas[mu]);
            }
         }
   }
} // entrenarBP
```

Función: entrenarValidBP()

Archivo: RN.cpp

Llama a: RN::entrenarBP(); RN::test()

Entrada: El número de patrones de entrenamiento a utilizar, los patrones de entrenamiento con sus salidas deseadas respectivas, el número de patrones de validación a utilizar, los patrones de validación con sus salidas deseadas respectivas, la cantidad de veces a realizarse el entrenamiento de la red, y la indicación de alternar o no los patrones.

Salida: Devuelve el mejor valor de aptitud sobre el set de validación.

Descripción: Dado un conjunto de patrones de entrenamiento y de validación (para detención temprana), se entrena a la red usando la función entrenarBP(). El entrenamiento continúa mientras la performance de validación mejora. 'maxReducciones' es el número de veces que continúa el entrenamiento aunque no ocurra ninguna mejora en la performance de validación. Se le pasa a la función entrenarBP() la indicación de si alternar o no los patrones de entrenamiento.

```
int RN::entrenarValidBP(int patConteo, float **patrones, int *sal_deseadas,
                        int validPatConteo, float **patrones_valid, int *sal_deseadas_valid,
                        int maxReducciones=5, int alternar=1) {

   float **pesos;
   float **biasguardado;
   int maxCorr=0;
   int corr=0;
   int reducciones=0;


   pesos = new (float *)[no_Capas-1];
   biasguardado = new (float *)[no_Capas];
   biasguardado[0] = 0;
   for (int l=0; l<no_Capas-1; l++) {
         pesos[l] = new float[wc[l]];
         biasguardado[l+1] = new float[conex[l]->obtenerAncho()];
   }

   while (reducciones < maxReducciones) {
         debugConteo = 0;

         entrenarBP(patConteo, patrones, sal_deseadas, 1, alternar);
         corr = test(validPatConteo, patrones_valid, sal_deseadas_valid, validFP);
         if (corr>maxCorr) {
            maxCorr = corr;
            reducciones = 0;
            // Guarda los pesos
            for (int l=0; l<no_Capas-1; l++) {
                  for (int i=0; i<wc[l]; i++) {
                     pesos[l][i] = w[l][i];
                  }
```

```
                    int ancho = conex[l]->obtenerAncho();
                    for (int i=0; i<ancho; i++) {
                        biasguardado[l+1][i] = bias[l+1][i];
                    }
                }
            } else {
                reducciones++;
            }
            printf("Validacion %2d:  %d/%d = %5.2f %%  ", it, corr, validPatConteo,
            ((float)100*corr)/validPatConteo);
            if (reducciones) {
                printf("(reduccion %d)", reducciones);
            }

            printf("\n  DEBUG Conteo = %d", debugConteo);

            printf("\n");
            fflush(stdout);
    }

    // Restaura los pesos buenos
    printf("Restaurando pesos\n");
    fflush(stdout);
    for (int l=0; l<no_Capas-1; l++) {
            delete[] w[l];
            delete[] bias[l+1];
    }
    delete[] w;
    delete[] bias;
    w = pesos;
    bias = biasguardado;

    return maxCorr;
} // entrenarValidBP
```

Función: test()

Archivo: RN.cpp

Entrada: El número de patrones a utilizar, dichos patrones y sus salidas respectivas.

Salida: El número de patrones clasificados correctamente.

Descripción: Testea la performance sobre los patrones dados y permite obtener una medida de la capacidad de clasificación de la red neuronal.

```
int  RN::test(int patConteo, float **patrones, int *sal_deseadas) {
   float max, maxCand;
   int cl, corr=0;

   int sal_ancho = conex[no_Capas-2]->obtenerAncho();

   // Para cada patron...
   for (int mu=0; mu<patConteo; mu++) {

         // ...lo evalua y encuentra al nodo de salida con la activacion mas grande.
         eval(patrones[mu]);
         cl = 0;
         max = act[no_Capas-1][cl];
         for (int i=1; i<sal_ancho; i++) {
            maxCand = act[no_Capas-1][i];
            if (maxCand > max) {
                  cl = i;
                  max = maxCand;
            }
         }

         // Si el nodo era el correcto, el patron se clasifico correctamente
         if (cl==sal_deseadas[mu])
            corr++;

   }

   return corr;
} // test
```

Función: imprimir()

Archivo: RN.cpp

Entrada: El archivo donde se almacenarán los datos de la red a imprimirse.

Descripción: Imprime a la red (incluyendo todos los pesos). Útil para debugging

```
void RN::imprimir(FILE *archivo) {

   // Imprime todas las capas, menos la ultima
   for (int l=0; l<no_Capas-1; l++) {
         fprintf(archivo, "Capa %d:\n", l);

         fprintf(archivo, "  Activacion: ");
         int alto = conex[l]->obtenerAlto();
         for (int i=0; i<alto; i++) {
            if (act[l])
                  fprintf(archivo, "%9.3f  ", act[l][i]);
         }

         fprintf(archivo, "\n  Flujo ent.:   ");
         for (int i=0; i<alto; i++) {
            if (h[l])
                  fprintf(archivo, "%9.3f  ", h[l][i]);
         }
         fprintf(archivo, "\n  Indices de pesos:\n");
         conex[l]->print();
         printf("\n  Pesos:\n   ");
         for (int i=0; i<wc[l]; i ++) {
            printf("%9.3f  ", w[l][i]);
         }
         printf("\n");
   }

   // Imprime la ultima capa
   fprintf(archivo, "Capa %d:\n", no_Capas-1);
   fprintf(archivo, "  Activacion:   ");
   int ancho = conex[no_Capas-2]->obtenerAncho();
   for (int i=0; i<ancho; i++) {
         if (act[no_Capas-1])
            fprintf(archivo, "%9.3f ", act[no_Capas-1][i]);
   }
   fprintf(archivo, "\n  Flujo ent.:     ");
   for (int i=0; i<ancho; i++) {
         if (h[no_Capas-1])
            fprintf(archivo, "%9.3f ", h[no_Capas-1][i]);
   }
   fprintf(archivo, "\n");

} // imprimir
```

Función: imprimir _salida()

Archivo: RN.cpp

Entrada: El archivo donde se almacenarán los datos de la red a imprimirse.

Descripción: Imprime la activación de la capa de salida (sin pesos). Útil para debugging.

```
void RN::imprimir _salida(FILE *archivo) {

    // Imprime la ultima capa
    fprintf(archivo, "Capa %d:\n", no_Capas-1);
    fprintf(archivo, "  Activacion:   ");
    int ancho = conex[no_Capas-2]->obtenerAncho();
    for (int i=0; i<ancho; i++) {
            if (act[no_Capas-1])
                fprintf(archivo, "%9.3f ", act[no_Capas-1][i]);
    }
    fprintf(archivo, "\n");

} // imprimir _salida

// Fin de RN.cpp
```

Archivo Problema.h

Archivo: Problema.h

Descripción: Declaración de variables y funciones para el archivo Problema.cpp

```
//Problema.h

extern int no_de_entrenam;
extern int no_de_valid;

extern float **patrones_entrenam;
extern float **patrones_valid;

extern int *sal_deseadas_entrenam;
extern int *sal_deseadas_valid;


void inic_patrones();

PATRON leer_archivo(char *nombre_archivo, int *conteo);

// Fin de Problema.h
```

Archivo Problema.cpp

Archivo: Problema.cpp

Descripción: Declaración de variables e inclusión de archivos.

```
// Problema.cpp

#include <stdlib.h>
#include <stdio.h>
#include <strings.h>
#include "Estructdatos.h"
#include "Prototipos.h"
#include "Problema.h"



PATRON entrenam_set;
PATRON valid_set;

int no_de_entrenam=0;
int no_de_valid=0;

float **patrones_entrenam;
float **patrones_valid;

int *sal_deseadas_entrenam;
int *sal_deseadas_valid;
```

Función: inic_patrones()

Llama a: leer_archivo()

Archivo: Problema.cpp

Descripción: Genera una lista de los patrones que serán utilizados por el algoritmo back-propagation para el entrenamiento y la validación de las redes neuronales.

```
void inic_patrones(void)
{
    char entrenam_file[256];
    char valid_file[256];

    sprintf(entrenam_file, "%s.ent", nombreproblema);
    sprintf(valid_file, "%s.val", nombreproblema);

   entrenam_set = leer_archivo (entrenam_file, &no_de_entrenam);
   valid_set = leer_archivo (valid_file, &no_de_valid);

   patrones_entrenam = new (float *)[no_de_entrenam];
   patrones_valid = new (float *)[no_de_valid];

   sal_deseadas_entrenam = new int[no_de_entrenam];
   sal_deseadas_valid = new int[no_de_valid];

   PATRON *p = &entrenam_set;
   // PATRON *q = p;
   for (int i=0; i<no_de_entrenam; i++) {
      patrones_entrenam[i] = p->pat;
      sal_deseadas_entrenam[i] = p->cl;
      p = p->proximo;
      // free(q);
      // q = p;
   }

   p = &valid_set;
   // q = p;
   for (int i=0; i<no_de_valid; i++) {
      patrones_valid[i] = p->pat;
      sal_deseadas_valid[i] = p->cl;
      p = p->proximo;
      // free(q);
      // q = p;
   }

}
```

Función: leer_archivo()

Archivo: Problema.cpp

Entrada: Nombre del archivo a leer, un entero donde se almacenará el número de patrones leídos.

Salida: La lista de patrones leídos.

Descripción: Genera una lista con los patrones que serán utilizados por el algoritmo back-propagation para el entrenamiento o la validación de las redes neuronales, según sea el nombre del archivo leído.

```
PATRON leer_archivo(char *nombre_archivo, int *conteo)
{
  int i;
  float temp;
  char auxiliar[7];
  FILE *patron_file;
  PATRON pat_lista, *este_pat;

  (*conteo) = 0;

  if ((patron_file = fopen(nombre_archivo, "r")) == NULL) {
    fprintf(logfile, "No se puede abrir archivo: %s\n", nombre_archivo);
    exit(-1);
  }

  este_pat = &pat_lista;

  fscanf(patron_file, "%s", auxiliar);

  int stop = strcmp(auxiliar,"class:");
  while (!stop) {
    (*conteo)++;

    este_pat->pat = (float *) malloc (sizeof(float)*entradas);
    fscanf(patron_file, "%d", &este_pat->cl);

    for (i=0; i<entradas; i++){
      fscanf(patron_file, "%f", &temp);
      este_pat->pat[i] = (float)temp;
    }

    auxiliar[0] = 0;
    fscanf(patron_file, "%s", auxiliar);

    stop = strcmp(auxiliar,"clase:");
    if (!stop) {
      este_pat->proximo = (struct patron *) malloc(sizeof(struct patron));
      este_pat = (PATRON *)este_pat->proximo;
    }
    else
      este_pat->proximo = NULL;
```

```
  }

  fclose(patron_file);
  return pat_lista;
} /* leer_archivo() */

// Fin de Problema.cpp
```

Archivo Matriz.h

Archivo: Matriz.h

Descripción: Declaración de la clase Matriz que encapsula un template de matriz, implementada en Matriz.cpp

```
//  Matriz.h

#ifndef MATRIZ_H
#define MATRIZ_H

template<class T>
class Matriz {
public:
   Matriz(int al, int an
   ~Matriz
   void set(int i, int j, T x
   T obtener(int i, int j);
   int obtenerAlto();
   int obtenerAncho();
   void imprimir_matriz();

private:
   int alto;                  // La altura de la matriz
   int ancho;                 // El  ancho de la matriz
   T *entradas;               // Las entradas de la matriz
};

#endif

// Fin de Matriz.h
```

Archivo Matriz.cpp

Archivo: RN.cpp

Descripción: Inclusión de archivos.

```
//  Matriz.cpp

#include <stdio.h>
#include "Matriz.h"
```

Función: Matriz()

Archivo: Matriz.cpp

Entrada: La altura y el ancho de la matriz.

Descripción: Constructor de la clase. Reserva espacio para los elementos de la matriz.

```
template<class T> Matriz<T>::Matriz (int al, int an) {
   alto = al;
   ancho = an;
   entradas = new T[h*w];
}
```

Función:~Matriz()

Archivo: Matriz.cpp

Descripción: Destructor de la clase. Libera el espacio de los elementos de la matriz.

```
template<class T> Matriz<T>::~Matriz () {
   // delete[] entradas;
}
```

Función: set()

Archivo: Matriz.cpp

Entrada: Las coordenadas del elemento y el valor a almacenar.

Descripción: Almacena el valor indicado en el elemento de la matriz: M[i,j] = x

```
template<class T> void Matriz<T>::set(int i, int j, T x) {
   int *punteroAux=(int *) 87;
   if (i<0 || i>=alto || j>ancho || j<0) {
         fprintf(stderr, "Matriz::set: Error en el indice\n");
         fflush(stderr);
         fprintf(stderr, "%d\n", *punteroAux);
         exit(1);
   }
   entradas[i*ancho+j] = x;
}
```

Función: obtener()

Archivo: Matriz.cpp

Entrada: Las coordenadas del elemento.

Salida: El valor del elemento.

Descripción: Obtiene el valor del elemento indicado de la matriz: M[i,j].

```
template<class T> T Matriz<T>::obtener (int i, int j) {
   int *punteroAux=(int *) 87;
   if (i<0 || i>=alto || j>=ancho || j<0) {
         fprintf(stderr, "Matriz::obtener: Error en el indice!\n");
         fflush(stderr);
         fprintf(stderr, "%d\n", *punteroAux);
         exit(1);
   }
   return entradas[i*ancho+j];
}
```

Función: obtenerAlto()

Archivo: Matriz.cpp

Salida: El alto de la matriz.

Descripción: Devuelve el alto de la matriz.

```
template<class T> int Matriz<T>::obtenerAlto() {
   return alto;
}
```

Función: obtenerAncho()

Archivo: Matriz.cpp

Salida: el ancho de la matriz.

Descripción: Devuelve el ancho de la matriz.

```
template<class T> int Matriz<T>::obtenerAncho() {
    return ancho;
}
```

Función: imprimir_matriz()

Archivo: Matriz.cpp

Descripción: Imprime la matriz. Útil para debugging.

```
template<class T> void Matriz<T>::imprimir_matriz() {
   for (int i=0; i<alto; i++) {
           printf("   ");
           for (int j=0; j<ancho; j++) {
               printf("%9.3f  ", (float) entradas[i*ancho+j]);
           }
           printf("\n");
   }
}

// Fin de Matriz.cpp
```

Índice de funciones

Apéndice B: Datos numéricos experimentales

Presentación completa del set de entrenamiento	**Tasa de aprendizaje y momentum**			
	Aprendizaje=0,01 Momentum=0,9	***Aprendizaje=0,1 Momentum=0,5***	***Aprendizaje=0,5 Momentum=0,1***	***Aprendizaje=0,9 Momentum=0,0***
1	0,1281930	0,1274290	0,1465700	0,1407160
2	0,1231820	0,1254250	0,1378520	0,1209030
3	0,1219060	0,1231900	0,1283190	0,0974183
4	0,1205920	0,1206630	0,1149520	0,0775726
5	0,1191960	0,1177310	0,0987156	0,0643354
6	0,1176980	0,1142980	0,0830958	0,0559834
7	0,1160820	0,1103010	0,0705916	0,0504966
8	0,1143320	0,1057220	0,0613894	0,0465803
9	0,1124330	0,1006050	0,0547400	0,0435850
10	0,1103770	0,0950569	0,0498767	0,0411951
11	0,1081550	0,0892394	0,0462266	0,0392406
12	0,1057660	0,0833407	0,0433977	0,0376135
13	0,1032120	0,0775446	0,0411309	0,0362344
14	0,1005030	0,0720046	0,0392574	0,0350414
15	0,0976548	0,0668294	0,0376681	0,0339862
16	0,0946886	0,0620823	0,0362914	0,0330317
17	0,0916317	0,0577880	0,0350796	0,0321495
18	0,0885152	0,0539428	0,0339994	0,0313194
19	0,0853724	0,0505231	0,0330266	0,0305270
20	0,0822365	0,0474942	0,0321423	0,0297636
21	0,0791392	0,0448155	0,0313315	0,0290242
22	0,0761086	0,0424455	0,0305822	0,0283061
23	0,0731681	0,0403442	0,0298843	0,0276083
24	0,0703360	0,0384747	0,0292299	0,0269300
25	0,0676254	0,0368047	0,0286124	0,0262707
26	0,0650445	0,0353056	0,0280268	0,0256298
27	0,0625972	0,0339533	0,0274692	0,0250074
28	0,0602840	0,0327272	0,0269366	0,0244038
29	0,0581028	0,0316099	0,0264266	0,0238204
30	0,0560496	0,0305869	0,0259375	0,0232592
31	0,0541189	0,0296459	0,0254680	0,0227227
32	0,0523046	0,0287765	0,0250171	0,0222135
33	0,0506000	0,0279700	0,0245841	0,0217337
34	0,0489982	0,0272191	0,0241684	0,0212847
35	0,0474927	0,0265176	0,0237695	0,0208670
36	0,0460767	0,0258601	0,0233869	0,0204803
37	0,0447440	0,0252419	0,0230202	0,0201235

38	0,0434887	0,0246593	0,0226690	0,0197947
39	0,0423052	0,0241087	0,0223330	0,0194919
40	0,0411883	0,0235872	0,0220115	0,0192127
41	0,0401332	0,0230921	0,0217041	0,0189547
42	0,0391353	0,0226213	0,0214104	0,0187154
43	0,0381906	0,0221726	0,0211297	0,0184924
44	0,0372952	0,0217444	0,0208614	0,0182837
45	0,0364457	0,0213349	0,0206051	0,0180870
46	0,0356387	0,0209429	0,0203601	0,0179006
47	0,0348714	0,0205669	0,0201258	0,0177225
48	0,0341410	0,0202058	0,0199017	0,0175511
49	0,0334450	0,0198586	0,0196871	0,0173849
50	0,0327810	0,0195244	0,0194815	0,0172224
51	0,0321471	0,0192023	0,0192844	0,0170623
52	0,0315411	0,0188915	0,0190953	0,0169036
53	0,0309614	0,0185913	0,0189135	0,0167451
54	0,0304061	0,0183010	0,0187388	0,0165859
55	0,0298739	0,0180201	0,0185705	0,0164255
56	0,0293633	0,0177479	0,0184084	0,0162631
57	0,0288730	0,0174841	0,0182519	0,0160984
58	0,0284017	0,0172281	0,0181007	0,0159313
59	0,0279483	0,0169795	0,0179545	0,0157617
60	0,0275119	0,0167379	0,0178129	0,0155897
61	0,0270914	0,0165030	0,0176757	0,0154159
62	0,0266859	0,0162744	0,0175425	0,0152408
63	0,0262946	0,0160518	0,0174131	0,0150653
64	0,0259168	0,0158350	0,0172872	0,0148902
65	0,0255516	0,0156236	0,0171648	0,0147168
66	0,0251985	0,0154174	0,0170454	0,0145463
67	0,0248568	0,0152163	0,0169290	0,0143797
68	0,0245258	0,0150200	0,0168154	0,0142185
69	0,0242052	0,0148282	0,0167045	0,0140634
70	0,0238942	0,0146409	0,0165961	0,0139155
71	0,0235925	0,0144578	0,0164901	0,0137753
72	0,0232996	0,0142788	0,0163864	0,0136430
73	0,0230151	0,0141037	0,0162848	0,0135189
74	0,0227386	0,0139324	0,0161854	0,0134027
75	0,0224697	0,0137648	0,0160880	0,0132942
76	0,0222081	0,0136007	0,0159925	0,0131930
77	0,0219534	0,0134400	0,0158990	0,0130985
78	0,0217053	0,0132826	0,0158073	0,0130103
79	0,0214635	0,0131284	0,0157173	0,0129278
80	0,0212278	0,0129773	0,0156291	0,0128505
81	0,0209979	0,0128293	0,0155426	0,0127779
82	0,0207736	0,0126841	0,0154577	0,0127096

83	0,0205546	0,0125418	0,0153745	0,0126451
84	0,0203407	0,0124022	0,0152928	0,0125841
85	0,0201317	0,0122653	0,0152127	0,0125262
86	0,0199274	0,0121309	0,0151341	0,0124711
87	0,0197277	0,0119991	0,0150570	0,0124186
88	0,0195323	0,0118698	0,0149814	0,0123682
89	0,0193410	0,0117429	0,0149072	0,0123199
90	0,0191539	0,0116183	0,0148345	0,0122734
91	0,0189706	0,0114959	0,0147631	0,0122285
92	0,0187910	0,0113758	0,0146931	0,0121850
93	0,0186151	0,0112578	0,0146244	0,0121428
94	0,0184426	0,0111419	0,0145571	0,0121018
95	0,0182735	0,0110281	0,0144910	0,0120619
96	0,0181077	0,0109163	0,0144262	0,0120229
97	0,0179450	0,0108064	0,0143626	0,0119848
98	0,0177853	0,0106984	0,0143003	0,0119474
99	0,0176286	0,0105923	0,0142391	0,0119108
100	0,0174748	0,0104880	0,0141790	0,0118750
101	0,0173236	0,0103855	0,0141201	0,0118397
102	0,0171752	0,0102847	0,0140623	0,0118051
103	0,0170293	0,0101856	0,0140056	0,0117712
104	0,0168860	0,0100881	0,0139499	0,0117378
105	0,0167451	0,0099923	0,0138952	0,0117050
106	0,0166065	0,0098980	0,0138415	0,0116727
107	0,0164703	0,0098053	0,0137887	0,0116411
108	0,0163363	0,0097141	0,0137369	0,0116101
109	0,0162044	0,0096244	0,0136860	0,0115796
110	0,0160747	0,0095361	0,0136360	0,0115497
111	0,0159470	0,0094493	0,0135868	0,0115205
112	0,0158213	0,0093638	0,0135385	0,0114918
113	0,0156975	0,0092796	0,0134910	0,0114638
114	0,0155756	0,0091968	0,0134442	0,0114363
115	0,0154556	0,0091153	0,0133982	0,0114095
116	0,0153374	0,0090350	0,0133530	0,0113833
117	0,0152209	0,0089560	0,0133085	0,0113577
118	0,0151061	0,0088782	0,0132647	0,0113327
119	0,0149930	0,0088016	0,0132215	0,0113083
120	0,0148815	0,0087261	0,0131791	0,0112845
121	0,0147716	0,0086518	0,0131373	0,0112613
122	0,0146632	0,0085786	0,0130961	0,0112387
123	0,0145564	0,0085065	0,0130555	0,0112166
124	0,0144510	0,0084354	0,0130156	0,0111950
125	0,0143471	0,0083654	0,0129762	0,0111740
126	0,0142445	0,0082965	0,0129374	0,0111535
127	0,0141434	0,0082285	0,0128992	0,0111335

128	0,0140436	0,0081615	0,0128615	0,0111140
129	0,0139451	0,0080954	0,0128244	0,0110949
130	0,0138479	0,0080304	0,0127879	0,0110763
131	0,0137520	0,0079662	0,0127518	0,0110581
132	0,0136573	0,0079029	0,0127163	0,0110404
133	0,0135638	0,0078406	0,0126813	0,0110231
134	0,0134715	0,0077790	0,0126468	0,0110062
135	0,0133804	0,0077184	0,0126129	0,0109897
136	0,0132904	0,0076586	0,0125794	0,0109736
137	0,0132015	0,0075996	0,0125464	0,0109578
138	0,0131137	0,0075414	0,0125139	0,0109425
139	0,0130270	0,0074840	0,0124818	0,0109274
140	0,0129413	0,0074273	0,0124503	0,0109128
141	0,0128567	0,0073714	0,0124192	0,0108984
142	0,0127731	0,0073163	0,0123886	0,0108844
143	0,0126905	0,0072619	0,0123584	0,0108707
144	0,0126088	0,0072082	0,0123287	0,0108574
145	0,0125282	0,0071552	0,0122995	0,0108443
146	0,0124484	0,0071028	0,0122707	0,0108315
147	0,0123696	0,0070512	0,0122424	0,0108190
148	0,0122917	0,0070002	0,0122144	0,0108068
149	0,0122147	0,0069498	0,0121870	0,0107949
150	0,0121385	0,0069001	0,0121599	0,0107832
151	0,0120632	0,0068510	0,0121333	0,0107718
152	0,0119888	0,0068026	0,0121070	0,0107607
153	0,0119152	0,0067547	0,0120812	0,0107498
154	0,0118424	0,0067074	0,0120558	0,0107391
155	0,0117704	0,0066607	0,0120308	0,0107287
156	0,0116992	0,0066145	0,0120062	0,0107185
157	0,0116288	0,0065689	0,0119820	0,0107086
158	0,0115592	0,0065239	0,0119582	0,0106988
159	0,0114903	0,0064794	0,0119347	0,0106893
160	0,0114221	0,0064354	0,0119116	0,0106799
161	0,0113547	0,0063919	0,0118889	0,0106708
162	0,0112880	0,0063489	0,0118665	0,0106619
163	0,0112220	0,0063065	0,0118445	0,0106532
164	0,0111567	0,0062645	0,0118229	0,0106446
165	0,0110920	0,0062230	0,0118015	0,0106362
166	0,0110281	0,0061820	0,0117805	0,0106280
167	0,0109648	0,0061414	0,0117599	0,0106200
168	0,0109022	0,0061013	0,0117395	0,0106122
169	0,0108402	0,0060616	0,0117195	0,0106045
170	0,0107788	0,0060224	0,0116998	0,0105970
171	0,0107181	0,0059836	0,0116804	0,0105896
172	0,0106580	0,0059452	0,0116612	0,0105824

173	0,0105985	0,0059073	0,0116424	0,0105753
174	0,0105396	0,0058697	0,0116239	0,0105684
175	0,0104813	0,0058326	0,0116056	0,0105616
176	0,0104236	0,0057958	0,0115877	0,0105550
177	0,0103664	0,0057595	0,0115700	0,0105484
178	0,0103098	0,0057235	0,0115525	0,0105421
179	0,0102538	0,0056879	0,0115353	0,0105358
180	0,0101983	0,0056527	0,0115184	0,0105297
181	0,0101433	0,0056178	0,0115018	0,0105237
182	0,0100889	0,0055833	0,0114853	0,0105178
183	0,0100351	0,0055491	0,0114692	0,0105120
184	0,0099817	0,0055153	0,0114532	0,0105063
185	0,0099288	0,0054818	0,0114375	0,0105008
186	0,0098765	0,0054487	0,0114221	0,0104953
187	0,0098246	0,0054158	0,0114068	0,0104900
188	0,0097733	0,0053833	0,0113918	0,0104848
189	0,0097224	0,0053512	0,0113770	0,0104796
190	0,0096720	0,0053193	0,0113624	0,0104746
191	0,0096221	0,0052877	0,0113481	0,0104696
192	0,0095726	0,0052565	0,0113339	0,0104648
193	0,0095236	0,0052255	0,0113199	0,0104600
194	0,0094751	0,0051949	0,0113062	0,0104553
195	0,0094270	0,0051645	0,0112926	0,0104507
196	0,0093794	0,0051344	0,0112793	0,0104462
197	0,0093322	0,0051046	0,0112661	0,0104418
198	0,0092854	0,0050750	0,0112531	0,0104374
199	0,0092390	0,0050458	0,0112403	0,0104331
200	0,0091931	0,0050168	0,0112277	0,0104289
201	0,0091476	0,0049880	0,0112153	0,0104248
202	0,0091024	0,0049596	0,0112030	0,0104207
203	0,0090577	0,0049313	0,0111909	0,0104168
204	0,0090134	0,0049034	0,0111790	0,0104128
205	0,0089695	0,0048757	0,0111673	0,0104090
206	0,0089260	0,0048482	0,0111557	0,0104052
207	0,0088828	0,0048209	0,0111443	0,0104015
208	0,0088401	0,0047939	0,0111331	0,0103978
209	0,0087977	0,0047672	0,0111220	0,0103942
210	0,0087557	0,0047406	0,0111111	0,0103907
211	0,0087140	0,0047143	0,0111003	0,0103872
212	0,0086727	0,0046882	0,0110897	0,0103838
213	0,0086318	0,0046624	0,0110792	0,0103804
214	0,0085912	0,0046367	0,0110689	0,0103771
215	0,0085510	0,0046113	0,0110587	0,0103738
216	0,0085111	0,0045861	0,0110487	0,0103706
217	0,0084716	0,0045611	0,0110388	0,0103675

218	0,0084324	0,0045362	0,0110290	0,0103644
219	0,0083935	0,0045116	0,0110194	0,0103613
220	0,0083549	0,0044872	0,0110099	0,0103583
221	0,0083167	0,0044630	0,0110006	0,0103553
222	0,0082788	0,0044390	0,0109913	0,0103524
223	0,0082412	0,0044152	0,0109822	0,0103495
224	0,0082039	0,0043916	0,0109733	0,0103467
225	0,0081670	0,0043681	0,0109644	0,0103439
226	0,0081303	0,0043449	0,0109557	0,0103411
227	0,0080940	0,0043218	0,0109471	0,0103384
228	0,0080579	0,0042989	0,0109386	0,0103358
229	0,0080221	0,0042762	0,0109303	0,0103331
230	0,0079867	0,0042536	0,0109220	0,0103305
231	0,0079515	0,0042312	0,0109139	0,0103280
232	0,0079166	0,0042090	0,0109059	0,0103255
233	0,0078819	0,0041870	0,0108979	0,0103230
234	0,0078476	0,0041651	0,0108901	0,0103205
235	0,0078135	0,0041434	0,0108824	0,0103181
236	0,0077797	0,0041219	0,0108748	0,0103158
237	0,0077462	0,0041005	0,0108674	0,0103134
238	0,0077130	0,0040793	0,0108600	0,0103111
239	0,0076800	0,0040582	0,0108527	0,0103088
240	0,0076472	0,0040373	0,0108455	0,0103066
241	0,0076148	0,0040166	0,0108384	0,0103043
242	0,0075825	0,0039960	0,0108314	0,0103022
243	0,0075506	0,0039755	0,0108245	0,0103000
244	0,0075188	0,0039552	0,0108177	0,0102979
245	0,0074874	0,0039350	0,0108109	0,0102958
246	0,0074561	0,0039150	0,0108043	0,0102937
247	0,0074251	0,0038952	0,0107977	0,0102916
248	0,0073944	0,0038754	0,0107913	0,0102896
249	0,0073639	0,0038558	0,0107849	0,0102876
250	0,0073336	0,0038364	0,0107786	0,0102857
251	0,0073035	0,0038171	0,0107724	0,0102837
252	0,0072737	0,0037979	0,0107663	0,0102818
253	0,0072441	0,0037788	0,0107602	0,0102799
254	0,0072147	0,0037599	0,0107542	0,0102780
255	0,0071855	0,0037411	0,0107483	0,0102762
256	0,0071566	0,0037225	0,0107425	0,0102743
257	0,0071279	0,0037039	0,0107368	0,0102725
258	0,0070994	0,0036855	0,0107311	0,0102708
259	0,0070710	0,0036673	0,0107255	0,0102690
260	0,0070430	0,0036491	0,0107200	0,0102673
261	0,0070151	0,0036311	0,0107145	0,0102655
262	0,0069874	0,0036132	0,0107091	0,0102638

263	0,0069599	0,0035954	0,0107038	0,0102622
264	0,0069326	0,0035777	0,0106985	0,0102605
265	0,0069055	0,0035602	0,0106933	0,0102589
266	0,0068786	0,0035427	0,0106882	0,0102573
267	0,0068519	0,0035254	0,0106832	0,0102557
268	0,0068254	0,0035082	0,0106782	0,0102541
269	0,0067991	0,0034911	0,0106732	0,0102525
270	0,0067730	0,0034741	0,0106683	0,0102510
271	0,0067471	0,0034573	0,0106635	0,0102494
272	0,0067213	0,0034405	0,0106587	0,0102479
273	0,0066957	0,0034239	0,0106540	0,0102464
274	0,0066703	0,0034073	0,0106494	0,0102450
275	0,0066451	0,0033909	0,0106448	0,0102435
276	0,0066201	0,0033746	0,0106403	0,0102421
277	0,0065952	0,0033583	0,0106358	0,0102406
278	0,0065705	0,0033422	0,0106314	0,0102392
279	0,0065460	0,0033262	0,0106270	0,0102378
280	0,0065216	0,0033103	0,0106227	0,0102365
281	0,0064975	0,0032945	0,0106184	0,0102351
282	0,0064734	0,0032788	0,0106142	0,0102337
283	0,0064496	0,0032631	0,0106100	0,0102324
284	0,0064259	0,0032476	0,0106059	0,0102311
285	0,0064024	0,0032322	0,0106018	0,0102298
286	0,0063790	0,0032169	0,0105977	0,0102285
287	0,0063558	0,0032017	0,0105938	0,0102272
288	0,0063327	0,0031865	0,0105898	0,0102259
289	0,0063098	0,0031715	0,0105859	0,0102247
290	0,0062871	0,0031566	0,0105821	0,0102234
291	0,0062645	0,0031417	0,0105783	0,0102222
292	0,0062420	0,0031269	0,0105745	0,0102210
293	0,0062197	0,0031123	0,0105708	0,0102198
294	0,0061976	0,0030977	0,0105671	0,0102186
295	0,0061756	0,0030832	0,0105635	0,0102174
296	0,0061537	0,0030688	0,0105599	0,0102162
297	0,0061320	0,0030545	0,0105563	0,0102151
298	0,0061104	0,0030403	0,0105528	0,0102139
299	0,0060889	0,0030261	0,0105493	0,0102128
300	0,0060676	0,0030121	0,0105459	0,0102117
301	0,0060465	0,0029981	0,0105424	0,0102106
302	0,0060255	0,0029842	0,0105391	0,0102095
303	0,0060046	0,0029705	0,0105357	0,0102084
304	0,0059838	0,0029567	0,0105324	0,0102073
305	0,0059632	0,0029431	0,0105292	0,0102062
306	0,0059427	0,0029296	0,0105260	0,0102052
307	0,0059223	0,0029161	0,0105228	0,0102041

308	0,0059021	0,0029027	0,0105196	0,0102031
309	0,0058820	0,0028894	0,0105165	0,0102021
310	0,0058620	0,0028762	0,0105134	0,0102011
311	0,0058421	0,0028630	0,0105103	0,0102000
312	0,0058224	0,0028500	0,0105073	0,0101990
313	0,0058028	0,0028370	0,0105043	0,0101981
314	0,0057833	0,0028240	0,0105013	0,0101971
315	0,0057639	0,0028112	0,0104984	0,0101961
316	0,0057446	0,0027984	0,0104955	0,0101951
317	0,0057255	0,0027858	0,0104926	0,0101942
318	0,0057065	0,0027731	0,0104898	0,0101932
319	0,0056876	0,0027606	0,0104869	0,0101923
320	0,0056688	0,0027481	0,0104842	0,0101914
321	0,0056501	0,0027358	0,0104814	0,0101905
322	0,0056315	0,0027234	0,0104787	0,0101895
323	0,0056131	0,0027112	0,0104760	0,0101886
324	0,0055947	0,0026990	0,0104733	0,0101877
325	0,0055765	0,0026869	0,0104706	0,0101869
326	0,0055584	0,0026749	0,0104680	0,0101860
327	0,0055404	0,0026629	0,0104654	0,0101851
328	0,0055224	0,0026510	0,0104628	0,0101842
329	0,0055046	0,0026392	0,0104603	0,0101834
330	0,0054869	0,0026274	0,0104577	0,0101825
331	0,0054693	0,0026157	0,0104552	0,0101817
332	0,0054518	0,0026041	0,0104528	0,0101809
333	0,0054345	0,0025926	0,0104503	0,0101800
334	0,0054172	0,0025811	0,0104479	0,0101792
335	0,0054000	0,0025697	0,0104455	0,0101784
336	0,0053829	0,0025583	0,0104431	0,0101776
337	0,0053659	0,0025470	0,0104407	0,0101768
338	0,0053490	0,0025358	0,0104384	0,0101760
339	0,0053322	0,0025246	0,0104361	0,0101752
340	0,0053155	0,0025135	0,0104338	0,0101744
341	0,0052989	0,0025025	0,0104315	0,0101737
342	0,0052823	0,0024915	0,0104292	0,0101729
343	0,0052659	0,0024806	0,0104270	0,0101721
344	0,0052496	0,0024698	0,0104248	0,0101714
345	0,0052333	0,0024590	0,0104226	0,0101706
346	0,0052172	0,0024483	0,0104204	0,0101699
347	0,0052011	0,0024376	0,0104183	0,0101691
348	0,0051852	0,0024270	0,0104161	0,0101684
349	0,0051693	0,0024165	0,0104140	0,0101677
350	0,0051535	0,0024060	0,0104119	0,0101670
351	0,0051378	0,0023956	0,0104099	0,0101662
352	0,0051222	0,0023852	0,0104078	0,0101655

353	0,0051066	0,0023749	0,0104058	0,0101648
354	0,0050912	0,0023647	0,0104038	0,0101641
355	0,0050758	0,0023545	0,0104017	0,0101635
356	0,0050605	0,0023444	0,0103998	0,0101628
357	0,0050453	0,0023343	0,0103978	0,0101621
358	0,0050302	0,0023243	0,0103958	0,0101614
359	0,0050152	0,0023143	0,0103939	0,0101607
360	0,0050002	0,0023044	0,0103920	0,0101601
361	0,0049853	0,0022946	0,0103901	0,0101594
362	0,0049705	0,0022848	0,0103882	0,0101588
363	0,0049558	0,0022751	0,0103863	0,0101581
364	0,0049412	0,0022654	0,0103845	0,0101575
365	0,0049266	0,0022558	0,0103827	0,0101568
366	0,0049121	0,0022462	0,0103808	0,0101562
367	0,0048977	0,0022367	0,0103790	0,0101556
368	0,0048833	0,0022272	0,0103772	0,0101549
369	0,0048691	0,0022178	0,0103755	0,0101543
370	0,0048549	0,0022084	0,0103737	0,0101537
371	0,0048408	0,0021991	0,0103720	0,0101531
372	0,0048267	0,0021899	0,0103702	0,0101525
373	0,0048128	0,0021806	0,0103685	0,0101519
374	0,0047989	0,0021715	0,0103668	0,0101513
375	0,0047850	0,0021624	0,0103651	0,0101507
376	0,0047713	0,0021533	0,0103635	0,0101501
377	0,0047576	0,0021443	0,0103618	0,0101495
378	0,0047440	0,0021354	0,0103601	0,0101489
379	0,0047304	0,0021265	0,0103585	0,0101483
380	0,0047169	0,0021176	0,0103569	0,0101478
381	0,0047035	0,0021088	0,0103553	0,0101472
382	0,0046902	0,0021001	0,0103537	0,0101466
383	0,0046769	0,0020914	0,0103521	0,0101461
384	0,0046637	0,0020827	0,0103505	0,0101455
385	0,0046505	0,0020741	0,0103490	0,0101450
386	0,0046374	0,0020655	0,0103474	0,0101444
387	0,0046244	0,0020570	0,0103459	0,0101439
388	0,0046115	0,0020485	0,0103444	0,0101433
389	0,0045986	0,0020401	0,0103429	0,0101428
390	0,0045857	0,0020317	0,0103414	0,0101423
391	0,0045730	0,0020234	0,0103399	0,0101417
392	0,0045603	0,0020151	0,0103384	0,0101412
393	0,0045476	0,0020069	0,0103370	0,0101407
394	0,0045350	0,0019987	0,0103355	0,0101402
395	0,0045225	0,0019906	0,0103341	0,0101397
396	0,0045100	0,0019825	0,0103327	0,0101391
397	0,0044976	0,0019744	0,0103312	0,0101386

398	0,0044853	0,0019664	0,0103298	0,0101381
399	0,0044730	0,0019584	0,0103284	0,0101376
400	0,0044608	0,0019505	0,0103270	0,0101371
401	0,0044486	0,0019426	0,0103257	0,0101366
402	0,0044365	0,0019348	0,0103243	0,0101361
403	0,0044245	0,0019270	0,0103230	0,0101356
404	0,0044125	0,0019192	0,0103216	0,0101352
405	0,0044005	0,0019115	0,0103203	0,0101347
406	0,0043886	0,0019038	0,0103190	0,0101342
407	0,0043768	0,0018962	0,0103176	0,0101337
408	0,0043650	0,0018886	0,0103163	0,0101333
409	0,0043533	0,0018810	0,0103150	0,0101328
410	0,0043417	0,0018735	0,0103138	0,0101323
411	0,0043300	0,0018661	0,0103125	0,0101319
412	0,0043185	0,0018586	0,0103112	0,0101314
413	0,0043070	0,0018513	0,0103100	0,0101309
414	0,0042955	0,0018439	0,0103087	0,0101305
415	0,0042841	0,0018366	0,0103075	0,0101300
416	0,0042728	0,0018293	0,0103062	0,0101296
417	0,0042615	0,0018221	0,0103050	0,0101291
418	0,0042502	0,0018149	0,0103038	0,0101287
419	0,0042390	0,0018078	0,0103026	0,0101283
420	0,0042279	0,0018007	0,0103014	0,0101278
421	0,0042168	0,0017936	0,0103002	0,0101274
422	0,0042057	0,0017866	0,0102990	0,0101269
423	0,0041947	0,0017796	0,0102979	0,0101265
424	0,0041838	0,0017726	0,0102967	0,0101261
425	0,0041729	0,0017657	0,0102955	0,0101257
426	0,0041620	0,0017588	0,0102944	0,0101252
427	0,0041512	0,0017520	0,0102933	0,0101248
428	0,0041405	0,0017452	0,0102921	0,0101244
429	0,0041298	0,0017384	0,0102910	0,0101240
430	0,0041191	0,0017316	0,0102899	0,0101236
431	0,0041085	0,0017249	0,0102888	0,0101232
432	0,0040979	0,0017183	0,0102877	0,0101228
433	0,0040874	0,0017116	0,0102866	0,0101224
434	0,0040769	0,0017051	0,0102855	0,0101220
435	0,0040665	0,0016985	0,0102844	0,0101216
436	0,0040561	0,0016920	0,0102834	0,0101212
437	0,0040458	0,0016855	0,0102823	0,0101208
438	0,0040355	0,0016790	0,0102812	0,0101204
439	0,0040252	0,0016726	0,0102802	0,0101200
440	0,0040150	0,0016662	0,0102792	0,0101196
441	0,0040048	0,0016599	0,0102781	0,0101192
442	0,0039947	0,0016536	0,0102771	0,0101188

443	0,0039846	0,0016473	0,0102761	0,0101185
444	0,0039746	0,0016410	0,0102751	0,0101181
445	0,0039646	0,0016348	0,0102741	0,0101177
446	0,0039546	0,0016286	0,0102731	0,0101173
447	0,0039447	0,0016225	0,0102721	0,0101170
448	0,0039348	0,0016163	0,0102711	0,0101166
449	0,0039250	0,0016103	0,0102701	0,0101162
450	0,0039152	0,0016042	0,0102691	0,0101159
451	0,0039055	0,0015982	0,0102681	0,0101155
452	0,0038958	0,0015922	0,0102672	0,0101151
453	0,0038861	0,0015862	0,0102662	0,0101148
454	0,0038764	0,0015803	0,0102653	0,0101144
455	0,0038669	0,0015744	0,0102643	0,0101141
456	0,0038573	0,0015685	0,0102634	0,0101137
457	0,0038478	0,0015627	0,0102625	0,0101134
458	0,0038383	0,0015569	0,0102615	0,0101130
459	0,0038289	0,0015511	0,0102606	0,0101127
460	0,0038195	0,0015454	0,0102597	0,0101123
461	0,0038101	0,0015397	0,0102588	0,0101120
462	0,0038008	0,0015340	0,0102579	0,0101117
463	0,0037915	0,0015283	0,0102570	0,0101113
464	0,0037823	0,0015227	0,0102561	0,0101110
465	0,0037731	0,0015171	0,0102552	0,0101106
466	0,0037639	0,0015115	0,0102543	0,0101103
467	0,0037547	0,0015060	0,0102535	0,0101100
468	0,0037456	0,0015005	0,0102526	0,0101097
469	0,0037366	0,0014950	0,0102517	0,0101093
470	0,0037275	0,0014895	0,0102509	0,0101090
471	0,0037185	0,0014841	0,0102500	0,0101087
472	0,0037096	0,0014787	0,0102492	0,0101084
473	0,0037007	0,0014734	0,0102483	0,0101080
474	0,0036918	0,0014680	0,0102475	0,0101077
475	0,0036829	0,0014627	0,0102467	0,0101074
476	0,0036741	0,0014574	0,0102458	0,0101071
477	0,0036653	0,0014522	0,0102450	0,0101068
478	0,0036566	0,0014469	0,0102442	0,0101065
479	0,0036478	0,0014417	0,0102434	0,0101062
480	0,0036392	0,0014366	0,0102426	0,0101058
481	0,0036305	0,0014314	0,0102418	0,0101055
482	0,0036219	0,0014263	0,0102410	0,0101052
483	0,0036133	0,0014212	0,0102402	0,0101049
484	0,0036047	0,0014161	0,0102394	0,0101046
485	0,0035962	0,0014111	0,0102386	0,0101043
486	0,0035877	0,0014061	0,0102378	0,0101040
487	0,0035793	0,0014011	0,0102370	0,0101037

488	0,0035709	0,0013961	0,0102363	0,0101034
489	0,0035625	0,0013912	0,0102355	0,0101031
490	0,0035541	0,0013862	0,0102347	0,0101029
491	0,0035458	0,0013813	0,0102340	0,0101026
492	0,0035375	0,0013765	0,0102332	0,0101023
493	0,0035292	0,0013716	0,0102325	0,0101020
494	0,0035210	0,0013668	0,0102317	0,0101017
495	0,0035128	0,0013620	0,0102310	0,0101014
496	0,0035046	0,0013573	0,0102303	0,0101011
497	0,0034964	0,0013525	0,0102295	0,0101008
498	0,0034883	0,0013478	0,0102288	0,0101006
499	0,0034802	0,0013431	0,0102281	0,0101003
500	0,0034722	0,0013384	0,0102274	0,0101000

Presentación completa del set de entrenamiento	**Número de repeticiones del entrenamiento**			
	n=1	***n=3***	***n=5***	***n=10***
1	0,1316120	0,1216970	0,1273290	0,1257230
2	0,1226550	0,1206510	0,1253250	0,1207120
3	0,1219780	0,1194630	0,1230900	0,1194360
4	0,1211410	0,1181300	0,1205630	0,1181220
5	0,1202490	0,1166500	0,1176310	0,1167260
6	0,1193000	0,1150020	0,1141980	0,1152280
7	0,1182770	0,1131640	0,1102010	0,1136120
8	0,1171660	0,1111160	0,1056220	0,1118620
9	0,1159510	0,1088460	0,1005050	0,1099630
10	0,1146150	0,1063430	0,0949569	0,1079070
11	0,1131450	0,1036070	0,0891394	0,1056850
12	0,1115240	0,1006450	0,0832407	0,1032960
13	0,1097400	0,0974745	0,0774446	0,1007420
14	0,1077820	0,0941251	0,0719046	0,0980330
15	0,1056430	0,0906346	0,0667294	0,0951848
16	0,1033170	0,0870495	0,0619823	0,0922186
17	0,1008070	0,0834206	0,0576880	0,0891617
18	0,0981180	0,0798000	0,0538428	0,0860452
19	0,0952655	0,0762367	0,0504231	0,0829024
20	0,0922684	0,0727733	0,0473942	0,0797665
21	0,0891538	0,0694443	0,0447155	0,0766692
22	0,0859536	0,0662745	0,0423455	0,0736386
23	0,0827037	0,0632799	0,0402442	0,0706981
24	0,0794415	0,0604682	0,0383747	0,0678660

25	0,0762039	0,0578404	0,0367047	0,0651554
26	0,0730247	0,0553928	0,0352056	0,0625745
27	0,0699335	0,0531181	0,0338533	0,0601272
28	0,0669540	0,0510070	0,0326272	0,0578140
29	0,0641039	0,0490488	0,0315099	0,0556328
30	0,0613950	0,0472324	0,0304869	0,0535796
31	0,0588340	0,0455467	0,0295459	0,0516489
32	0,0564227	0,0439810	0,0286765	0,0498346
33	0,0541596	0,0425252	0,0278700	0,0481300
34	0,0520407	0,0411699	0,0271191	0,0465282
35	0,0500598	0,0399062	0,0264176	0,0450227
36	0,0482098	0,0387261	0,0257601	0,0436067
37	0,0464828	0,0376224	0,0251419	0,0422740
38	0,0448706	0,0365883	0,0245593	0,0410187
39	0,0433651	0,0356177	0,0240087	0,0398352
40	0,0419584	0,0347053	0,0234872	0,0387183
41	0,0406428	0,0338461	0,0229921	0,0376632
42	0,0394113	0,0330356	0,0225213	0,0366653
43	0,0382571	0,0322697	0,0220726	0,0357206
44	0,0371740	0,0315449	0,0216444	0,0348252
45	0,0361563	0,0308578	0,0212349	0,0339757
46	0,0351988	0,0302056	0,0208429	0,0331687
47	0,0342966	0,0295853	0,0204669	0,0324014
48	0,0334453	0,0289948	0,0201058	0,0316710
49	0,0326409	0,0284316	0,0197586	0,0309750
50	0,0318798	0,0278939	0,0194244	0,0303110
51	0,0311586	0,0273798	0,0191023	0,0296771
52	0,0304742	0,0268877	0,0187915	0,0290711
53	0,0298240	0,0264159	0,0184913	0,0284914
54	0,0292053	0,0259633	0,0182010	0,0279361
55	0,0286159	0,0255284	0,0179201	0,0274039
56	0,0280536	0,0251102	0,0176479	0,0268933
57	0,0275167	0,0247076	0,0173841	0,0264030
58	0,0270033	0,0243196	0,0171281	0,0259317
59	0,0265117	0,0239453	0,0168795	0,0254783
60	0,0260407	0,0235840	0,0166379	0,0250419
61	0,0255888	0,0232348	0,0164030	0,0246214
62	0,0251548	0,0228971	0,0161744	0,0242159
63	0,0247375	0,0225702	0,0159518	0,0238246
64	0,0243360	0,0222536	0,0157350	0,0234468
65	0,0239492	0,0219466	0,0155236	0,0230816
66	0,0235763	0,0216487	0,0153174	0,0227285
67	0,0232165	0,0213596	0,0151163	0,0223868
68	0,0228690	0,0210787	0,0149200	0,0220558
69	0,0225331	0,0208056	0,0147282	0,0217352

70	0,0222082	0,0205400	0,0145409	0,0214242
71	0,0218937	0,0202815	0,0143578	0,0211225
72	0,0215890	0,0200297	0,0141788	0,0208296
73	0,0212936	0,0197844	0,0140037	0,0205451
74	0,0210070	0,0195452	0,0138324	0,0202686
75	0,0207287	0,0193119	0,0136648	0,0199997
76	0,0204585	0,0190842	0,0135007	0,0197381
77	0,0201958	0,0188619	0,0133400	0,0194834
78	0,0199403	0,0186447	0,0131826	0,0192353
79	0,0196916	0,0184325	0,0130284	0,0189935
80	0,0194495	0,0182250	0,0128773	0,0187578
81	0,0192137	0,0180220	0,0127293	0,0185279
82	0,0189838	0,0178234	0,0125841	0,0183036
83	0,0187596	0,0176291	0,0124418	0,0180846
84	0,0185408	0,0174387	0,0123022	0,0178707
85	0,0183273	0,0172523	0,0121653	0,0176617
86	0,0181187	0,0170695	0,0120309	0,0174574
87	0,0179149	0,0168904	0,0118991	0,0172577
88	0,0177158	0,0167148	0,0117698	0,0170623
89	0,0175210	0,0165425	0,0116429	0,0168710
90	0,0173305	0,0163735	0,0115183	0,0166839
91	0,0171441	0,0162076	0,0113959	0,0165006
92	0,0169615	0,0160448	0,0112758	0,0163210
93	0,0167828	0,0158849	0,0111578	0,0161451
94	0,0166076	0,0157278	0,0110419	0,0159726
95	0,0164360	0,0155735	0,0109281	0,0158035
96	0,0162677	0,0154218	0,0108163	0,0156377
97	0,0161027	0,0152727	0,0107064	0,0154750
98	0,0159408	0,0151261	0,0105984	0,0153153
99	0,0157820	0,0149820	0,0104923	0,0151586
100	0,0156261	0,0148402	0,0103880	0,0150048
101	0,0154730	0,0147007	0,0102855	0,0148536
102	0,0153227	0,0145634	0,0101847	0,0147052
103	0,0151750	0,0144283	0,0100856	0,0145593
104	0,0150299	0,0142952	0,0099881	0,0144160
105	0,0148873	0,0141643	0,0098923	0,0142751
106	0,0147471	0,0140353	0,0097980	0,0141365
107	0,0146093	0,0139083	0,0097053	0,0140003
108	0,0144737	0,0137831	0,0096141	0,0138663
109	0,0143403	0,0136598	0,0095244	0,0137344
110	0,0142091	0,0135384	0,0094361	0,0136047
111	0,0140799	0,0134186	0,0093493	0,0134770
112	0,0139528	0,0133006	0,0092638	0,0133513
113	0,0138276	0,0131842	0,0091796	0,0132275
114	0,0137044	0,0130695	0,0090968	0,0131056

115	0,0135830	0,0129563	0,0090153	0,0129856
116	0,0134634	0,0128447	0,0089350	0,0128674
117	0,0133456	0,0127347	0,0088560	0,0127509
118	0,0132295	0,0126261	0,0087782	0,0126361
119	0,0131151	0,0125189	0,0087016	0,0125230
120	0,0130023	0,0124132	0,0086261	0,0124115
121	0,0128912	0,0123089	0,0085518	0,0123016
122	0,0127815	0,0122059	0,0084786	0,0121932
123	0,0126734	0,0121043	0,0084065	0,0120864
124	0,0125668	0,0120040	0,0083354	0,0119810
125	0,0124616	0,0119049	0,0082654	0,0118771
126	0,0123579	0,0118072	0,0081965	0,0117745
127	0,0122555	0,0117106	0,0081285	0,0116734
128	0,0121545	0,0116152	0,0080615	0,0115736
129	0,0120548	0,0115211	0,0079954	0,0114751
130	0,0119564	0,0114280	0,0079304	0,0113779
131	0,0118592	0,0113362	0,0078662	0,0112820
132	0,0117634	0,0112454	0,0078029	0,0111873
133	0,0116687	0,0111557	0,0077406	0,0110938
134	0,0115752	0,0110671	0,0076790	0,0110015
135	0,0114829	0,0109796	0,0076184	0,0109104
136	0,0113917	0,0108931	0,0075586	0,0108204
137	0,0113016	0,0108076	0,0074996	0,0107315
138	0,0112126	0,0107231	0,0074414	0,0106437
139	0,0111247	0,0106396	0,0073840	0,0105570
140	0,0110379	0,0105570	0,0073273	0,0104713
141	0,0109521	0,0104754	0,0072714	0,0103867
142	0,0108673	0,0103947	0,0072163	0,0103031
143	0,0107835	0,0103150	0,0071619	0,0102205
144	0,0107007	0,0102362	0,0071082	0,0101388
145	0,0106188	0,0101582	0,0070552	0,0100582
146	0,0105379	0,0100811	0,0070028	0,0099784
147	0,0104579	0,0100049	0,0069512	0,0098996
148	0,0103789	0,0099295	0,0069002	0,0098217
149	0,0103007	0,0098550	0,0068498	0,0097447
150	0,0102234	0,0097812	0,0068001	0,0096685
151	0,0101469	0,0097083	0,0067510	0,0095932
152	0,0100713	0,0096362	0,0067026	0,0095188
153	0,0099966	0,0095648	0,0066547	0,0094452
154	0,0099227	0,0094942	0,0066074	0,0093724
155	0,0098495	0,0094244	0,0065607	0,0093004
156	0,0097772	0,0093553	0,0065145	0,0092292
157	0,0097057	0,0092870	0,0064689	0,0091588
158	0,0096349	0,0092194	0,0064239	0,0090892
159	0,0095648	0,0091525	0,0063794	0,0090203

160	0,0094956	0,0090863	0,0063354	0,0089521
161	0,0094270	0,0090207	0,0062919	0,0088847
162	0,0093592	0,0089559	0,0062489	0,0088180
163	0,0092921	0,0088918	0,0062065	0,0087520
164	0,0092257	0,0088283	0,0061645	0,0086867
165	0,0091600	0,0087654	0,0061230	0,0086220
166	0,0090949	0,0087032	0,0060820	0,0085581
167	0,0090306	0,0086417	0,0060414	0,0084948
168	0,0089668	0,0085808	0,0060013	0,0084322
169	0,0089038	0,0085204	0,0059616	0,0083702
170	0,0088414	0,0084607	0,0059224	0,0083088
171	0,0087796	0,0084016	0,0058836	0,0082481
172	0,0087184	0,0083431	0,0058452	0,0081880
173	0,0086579	0,0082852	0,0058073	0,0081285
174	0,0085979	0,0082278	0,0057697	0,0080696
175	0,0085385	0,0081711	0,0057326	0,0080113
176	0,0084798	0,0081149	0,0056958	0,0079536
177	0,0084216	0,0080592	0,0056595	0,0078964
178	0,0083640	0,0080041	0,0056235	0,0078398
179	0,0083069	0,0079495	0,0055879	0,0077838
180	0,0082504	0,0078954	0,0055527	0,0077283
181	0,0081945	0,0078419	0,0055178	0,0076733
182	0,0081391	0,0077889	0,0054833	0,0076189
183	0,0080842	0,0077364	0,0054491	0,0075651
184	0,0080298	0,0076844	0,0054153	0,0075117
185	0,0079760	0,0076329	0,0053818	0,0074588
186	0,0079227	0,0075819	0,0053487	0,0074065
187	0,0078699	0,0075314	0,0053158	0,0073546
188	0,0078176	0,0074814	0,0052833	0,0073033
189	0,0077658	0,0074318	0,0052512	0,0072524
190	0,0077144	0,0073827	0,0052193	0,0072020
191	0,0076636	0,0073341	0,0051877	0,0071521
192	0,0076132	0,0072859	0,0051565	0,0071026
193	0,0075633	0,0072382	0,0051255	0,0070536
194	0,0075138	0,0071909	0,0050949	0,0070051
195	0,0074649	0,0071440	0,0050645	0,0069570
196	0,0074163	0,0070976	0,0050344	0,0069094
197	0,0073682	0,0070516	0,0050046	0,0068622
198	0,0073205	0,0070060	0,0049750	0,0068154
199	0,0072733	0,0069609	0,0049458	0,0067690
200	0,0072265	0,0069161	0,0049168	0,0067231
201	0,0071801	0,0068718	0,0048880	0,0066776
202	0,0071342	0,0068279	0,0048596	0,0066324
203	0,0070886	0,0067843	0,0048313	0,0065877
204	0,0070435	0,0067412	0,0048034	0,0065434

205	0,0069988	0,0066984	0,0047757	0,0064995
206	0,0069544	0,0066560	0,0047482	0,0064560
207	0,0069105	0,0066140	0,0047209	0,0064128
208	0,0068669	0,0065724	0,0046939	0,0063701
209	0,0068237	0,0065311	0,0046672	0,0063277
210	0,0067809	0,0064902	0,0046406	0,0062857
211	0,0067385	0,0064497	0,0046143	0,0062440
212	0,0066964	0,0064095	0,0045882	0,0062027
213	0,0066547	0,0063697	0,0045624	0,0061618
214	0,0066134	0,0063302	0,0045367	0,0061212
215	0,0065724	0,0062910	0,0045113	0,0060810
216	0,0065318	0,0062522	0,0044861	0,0060411
217	0,0064915	0,0062137	0,0044611	0,0060016
218	0,0064516	0,0061756	0,0044362	0,0059624
219	0,0064120	0,0061378	0,0044116	0,0059235
220	0,0063727	0,0061003	0,0043872	0,0058849
221	0,0063338	0,0060631	0,0043630	0,0058467
222	0,0062952	0,0060262	0,0043390	0,0058088
223	0,0062569	0,0059897	0,0043152	0,0057712
224	0,0062190	0,0059534	0,0042916	0,0057339
225	0,0061814	0,0059175	0,0042681	0,0056970
226	0,0061440	0,0058818	0,0042449	0,0056603
227	0,0061070	0,0058465	0,0042218	0,0056240
228	0,0060703	0,0058114	0,0041989	0,0055879
229	0,0060339	0,0057767	0,0041762	0,0055521
230	0,0059978	0,0057422	0,0041536	0,0055167
231	0,0059620	0,0057080	0,0041312	0,0054815
232	0,0059265	0,0056741	0,0041090	0,0054466
233	0,0058912	0,0056404	0,0040870	0,0054119
234	0,0058563	0,0056070	0,0040651	0,0053776
235	0,0058216	0,0055739	0,0040434	0,0053435
236	0,0057872	0,0055411	0,0040219	0,0053097
237	0,0057531	0,0055086	0,0040005	0,0052762
238	0,0057193	0,0054762	0,0039793	0,0052430
239	0,0056857	0,0054442	0,0039582	0,0052100
240	0,0056524	0,0054124	0,0039373	0,0051772
241	0,0056194	0,0053809	0,0039166	0,0051448
242	0,0055866	0,0053496	0,0038960	0,0051125
243	0,0055541	0,0053185	0,0038755	0,0050806
244	0,0055218	0,0052877	0,0038552	0,0050488
245	0,0054898	0,0052572	0,0038350	0,0050174
246	0,0054581	0,0052268	0,0038150	0,0049861
247	0,0054265	0,0051968	0,0037952	0,0049551
248	0,0053953	0,0051669	0,0037754	0,0049244
249	0,0053642	0,0051373	0,0037558	0,0048939

250	0,0053335	0,0051079	0,0037364	0,0048636
251	0,0053029	0,0050787	0,0037171	0,0048335
252	0,0052726	0,0050498	0,0036979	0,0048037
253	0,0052425	0,0050210	0,0036788	0,0047741
254	0,0052126	0,0049925	0,0036599	0,0047447
255	0,0051830	0,0049642	0,0036411	0,0047155
256	0,0051536	0,0049361	0,0036225	0,0046866
257	0,0051244	0,0049083	0,0036039	0,0046579
258	0,0050955	0,0048806	0,0035855	0,0046294
259	0,0050667	0,0048531	0,0035673	0,0046010
260	0,0050382	0,0048259	0,0035491	0,0045730
261	0,0050098	0,0047988	0,0035311	0,0045451
262	0,0049817	0,0047720	0,0035132	0,0045174
263	0,0049538	0,0047453	0,0034954	0,0044899
264	0,0049261	0,0047189	0,0034777	0,0044626
265	0,0048986	0,0046926	0,0034602	0,0044355
266	0,0048713	0,0046665	0,0034427	0,0044086
267	0,0048442	0,0046406	0,0034254	0,0043819
268	0,0048173	0,0046149	0,0034082	0,0043554
269	0,0047906	0,0045894	0,0033911	0,0043291
270	0,0047641	0,0045641	0,0033741	0,0043030
271	0,0047378	0,0045390	0,0033573	0,0042771
272	0,0047117	0,0045140	0,0033405	0,0042513
273	0,0046857	0,0044892	0,0033239	0,0042257
274	0,0046599	0,0044646	0,0033073	0,0042003
275	0,0046344	0,0044401	0,0032909	0,0041751
276	0,0046090	0,0044158	0,0032746	0,0041501
277	0,0045838	0,0043917	0,0032583	0,0041252
278	0,0045587	0,0043678	0,0032422	0,0041005
279	0,0045339	0,0043440	0,0032262	0,0040760
280	0,0045092	0,0043204	0,0032103	0,0040516
281	0,0044846	0,0042970	0,0031945	0,0040275
282	0,0044603	0,0042737	0,0031788	0,0040034
283	0,0044361	0,0042506	0,0031631	0,0039796
284	0,0044121	0,0042276	0,0031476	0,0039559
285	0,0043883	0,0042048	0,0031322	0,0039324
286	0,0043646	0,0041822	0,0031169	0,0039090
287	0,0043411	0,0041597	0,0031017	0,0038858
288	0,0043177	0,0041373	0,0030865	0,0038627
289	0,0042945	0,0041151	0,0030715	0,0038398
290	0,0042714	0,0040931	0,0030566	0,0038171
291	0,0042486	0,0040712	0,0030417	0,0037945
292	0,0042258	0,0040494	0,0030269	0,0037720
293	0,0042032	0,0040278	0,0030123	0,0037497
294	0,0041808	0,0040063	0,0029977	0,0037276

295	0,0041585	0,0039850	0,0029832	0,0037056
296	0,0041364	0,0039638	0,0029688	0,0036837
297	0,0041144	0,0039427	0,0029545	0,0036620
298	0,0040926	0,0039218	0,0029403	0,0036404
299	0,0040709	0,0039010	0,0029261	0,0036189
300	0,0040493	0,0038804	0,0029121	0,0035976
301	0,0040279	0,0038599	0,0028981	0,0035765
302	0,0040066	0,0038395	0,0028842	0,0035555
303	0,0039855	0,0038193	0,0028705	0,0035346
304	0,0039645	0,0037991	0,0028567	0,0035138
305	0,0039436	0,0037791	0,0028431	0,0034932
306	0,0039229	0,0037593	0,0028296	0,0034727
307	0,0039023	0,0037395	0,0028161	0,0034523
308	0,0038818	0,0037199	0,0028027	0,0034321
309	0,0038615	0,0037004	0,0027894	0,0034120
310	0,0038413	0,0036810	0,0027762	0,0033920
311	0,0038212	0,0036618	0,0027630	0,0033721
312	0,0038012	0,0036426	0,0027500	0,0033524
313	0,0037814	0,0036236	0,0027370	0,0033328
314	0,0037617	0,0036047	0,0027240	0,0033133
315	0,0037421	0,0035859	0,0027112	0,0032939
316	0,0037227	0,0035673	0,0026984	0,0032746
317	0,0037033	0,0035487	0,0026858	0,0032555
318	0,0036841	0,0035303	0,0026731	0,0032365
319	0,0036650	0,0035119	0,0026606	0,0032176
320	0,0036460	0,0034937	0,0026481	0,0031988
321	0,0036272	0,0034756	0,0026358	0,0031801
322	0,0036084	0,0034576	0,0026234	0,0031615
323	0,0035898	0,0034397	0,0026112	0,0031431
324	0,0035713	0,0034219	0,0025990	0,0031247
325	0,0035528	0,0034042	0,0025869	0,0031065
326	0,0035345	0,0033867	0,0025749	0,0030884
327	0,0035164	0,0033692	0,0025629	0,0030704
328	0,0034983	0,0033518	0,0025510	0,0030524
329	0,0034803	0,0033345	0,0025392	0,0030346
330	0,0034624	0,0033174	0,0025274	0,0030169
331	0,0034447	0,0033003	0,0025157	0,0029993
332	0,0034270	0,0032833	0,0025041	0,0029818
333	0,0034095	0,0032665	0,0024926	0,0029645
334	0,0033920	0,0032497	0,0024811	0,0029472
335	0,0033747	0,0032330	0,0024697	0,0029300
336	0,0033575	0,0032164	0,0024583	0,0029129
337	0,0033403	0,0031999	0,0024470	0,0028959
338	0,0033233	0,0031835	0,0024358	0,0028790
339	0,0033063	0,0031672	0,0024246	0,0028622

340	0,0032895	0,0031510	0,0024135	0,0028455
341	0,0032727	0,0031349	0,0024025	0,0028289
342	0,0032561	0,0031188	0,0023915	0,0028123
343	0,0032395	0,0031029	0,0023806	0,0027959
344	0,0032231	0,0030870	0,0023698	0,0027796
345	0,0032067	0,0030713	0,0023590	0,0027633
346	0,0031904	0,0030556	0,0023483	0,0027472
347	0,0031742	0,0030400	0,0023376	0,0027311
348	0,0031581	0,0030245	0,0023270	0,0027152
349	0,0031421	0,0030091	0,0023165	0,0026993
350	0,0031262	0,0029937	0,0023060	0,0026835
351	0,0031104	0,0029785	0,0022956	0,0026678
352	0,0030947	0,0029633	0,0022852	0,0026522
353	0,0030790	0,0029482	0,0022749	0,0026366
354	0,0030635	0,0029332	0,0022647	0,0026212
355	0,0030480	0,0029183	0,0022545	0,0026058
356	0,0030326	0,0029034	0,0022444	0,0025905
357	0,0030173	0,0028887	0,0022343	0,0025753
358	0,0030021	0,0028740	0,0022243	0,0025602
359	0,0029869	0,0028594	0,0022143	0,0025452
360	0,0029719	0,0028448	0,0022044	0,0025302
361	0,0029569	0,0028304	0,0021946	0,0025153
362	0,0029420	0,0028160	0,0021848	0,0025005
363	0,0029272	0,0028017	0,0021751	0,0024858
364	0,0029125	0,0027874	0,0021654	0,0024712
365	0,0028978	0,0027733	0,0021558	0,0024566
366	0,0028832	0,0027592	0,0021462	0,0024421
367	0,0028687	0,0027452	0,0021367	0,0024277
368	0,0028543	0,0027312	0,0021272	0,0024133
369	0,0028400	0,0027174	0,0021178	0,0023991
370	0,0028257	0,0027036	0,0021084	0,0023849
371	0,0028115	0,0026899	0,0020991	0,0023708
372	0,0027974	0,0026762	0,0020899	0,0023567
373	0,0027833	0,0026626	0,0020806	0,0023428
374	0,0027693	0,0026491	0,0020715	0,0023289
375	0,0027554	0,0026356	0,0020624	0,0023150
376	0,0027416	0,0026222	0,0020533	0,0023013
377	0,0027278	0,0026089	0,0020443	0,0022876
378	0,0027142	0,0025957	0,0020354	0,0022740
379	0,0027005	0,0025825	0,0020265	0,0022604
380	0,0026870	0,0025694	0,0020176	0,0022469
381	0,0026735	0,0025563	0,0020088	0,0022335
382	0,0026601	0,0025433	0,0020001	0,0022202
383	0,0026468	0,0025304	0,0019914	0,0022069
384	0,0026335	0,0025175	0,0019827	0,0021937

385	0,0026203	0,0025048	0,0019741	0,0021805
386	0,0026071	0,0024920	0,0019655	0,0021674
387	0,0025941	0,0024793	0,0019570	0,0021544
388	0,0025810	0,0024667	0,0019485	0,0021415
389	0,0025681	0,0024542	0,0019401	0,0021286
390	0,0025552	0,0024417	0,0019317	0,0021157
391	0,0025424	0,0024293	0,0019234	0,0021030
392	0,0025296	0,0024169	0,0019151	0,0020903
393	0,0025169	0,0024046	0,0019069	0,0020776
394	0,0025043	0,0023923	0,0018987	0,0020650
395	0,0024918	0,0023801	0,0018906	0,0020525
396	0,0024792	0,0023680	0,0018825	0,0020400
397	0,0024668	0,0023559	0,0018744	0,0020276
398	0,0024544	0,0023439	0,0018664	0,0020153
399	0,0024421	0,0023319	0,0018584	0,0020030
400	0,0024298	0,0023200	0,0018505	0,0019908
401	0,0024176	0,0023081	0,0018426	0,0019786
402	0,0024055	0,0022963	0,0018348	0,0019665
403	0,0023934	0,0022846	0,0018270	0,0019545
404	0,0023814	0,0022729	0,0018192	0,0019425
405	0,0023694	0,0022613	0,0018115	0,0019305
406	0,0023575	0,0022497	0,0018038	0,0019186
407	0,0023456	0,0022382	0,0017962	0,0019068
408	0,0023338	0,0022267	0,0017886	0,0018950
409	0,0023220	0,0022153	0,0017810	0,0018833
410	0,0023104	0,0022039	0,0017735	0,0018717
411	0,0022987	0,0021926	0,0017661	0,0018600
412	0,0022871	0,0021813	0,0017586	0,0018485
413	0,0022756	0,0021701	0,0017513	0,0018370
414	0,0022641	0,0021589	0,0017439	0,0018255
415	0,0022527	0,0021478	0,0017366	0,0018141
416	0,0022413	0,0021367	0,0017293	0,0018028
417	0,0022300	0,0021257	0,0017221	0,0017915
418	0,0022187	0,0021148	0,0017149	0,0017802
419	0,0022075	0,0021038	0,0017078	0,0017690
420	0,0021964	0,0020930	0,0017007	0,0017579
421	0,0021852	0,0020821	0,0016936	0,0017468
422	0,0021742	0,0020714	0,0016866	0,0017357
423	0,0021632	0,0020606	0,0016796	0,0017247
424	0,0021522	0,0020500	0,0016726	0,0017138
425	0,0021413	0,0020393	0,0016657	0,0017029
426	0,0021304	0,0020287	0,0016588	0,0016920
427	0,0021196	0,0020182	0,0016520	0,0016812
428	0,0021088	0,0020077	0,0016452	0,0016705
429	0,0020981	0,0019973	0,0016384	0,0016598

430	0,0020875	0,0019869	0,0016316	0,0016491
431	0,0020768	0,0019765	0,0016249	0,0016385
432	0,0020663	0,0019662	0,0016183	0,0016279
433	0,0020557	0,0019559	0,0016116	0,0016174
434	0,0020452	0,0019457	0,0016051	0,0016069
435	0,0020348	0,0019355	0,0015985	0,0015965
436	0,0020244	0,0019254	0,0015920	0,0015861
437	0,0020141	0,0019153	0,0015855	0,0015758
438	0,0020038	0,0019052	0,0015790	0,0015655
439	0,0019935	0,0018952	0,0015726	0,0015552
440	0,0019833	0,0018852	0,0015662	0,0015450
441	0,0019731	0,0018753	0,0015599	0,0015348
442	0,0019630	0,0018654	0,0015536	0,0015247
443	0,0019529	0,0018556	0,0015473	0,0015146
444	0,0019429	0,0018458	0,0015410	0,0015046
445	0,0019329	0,0018360	0,0015348	0,0014946
446	0,0019229	0,0018263	0,0015286	0,0014846
447	0,0019130	0,0018166	0,0015225	0,0014747
448	0,0019032	0,0018070	0,0015163	0,0014648
449	0,0018933	0,0017974	0,0015103	0,0014550
450	0,0018836	0,0017878	0,0015042	0,0014452
451	0,0018738	0,0017783	0,0014982	0,0014355
452	0,0018641	0,0017688	0,0014922	0,0014258
453	0,0018545	0,0017594	0,0014862	0,0014161
454	0,0018448	0,0017500	0,0014803	0,0014064
455	0,0018353	0,0017406	0,0014744	0,0013969
456	0,0018257	0,0017313	0,0014685	0,0013873
457	0,0018162	0,0017220	0,0014627	0,0013778
458	0,0018068	0,0017127	0,0014569	0,0013683
459	0,0017973	0,0017035	0,0014511	0,0013589
460	0,0017880	0,0016943	0,0014454	0,0013495
461	0,0017786	0,0016852	0,0014397	0,0013401
462	0,0017693	0,0016761	0,0014340	0,0013308
463	0,0017600	0,0016670	0,0014283	0,0013215
464	0,0017508	0,0016580	0,0014227	0,0013123
465	0,0017416	0,0016490	0,0014171	0,0013031
466	0,0017325	0,0016400	0,0014115	0,0012939
467	0,0017233	0,0016311	0,0014060	0,0012847
468	0,0017143	0,0016222	0,0014005	0,0012756
469	0,0017052	0,0016134	0,0013950	0,0012666
470	0,0016962	0,0016046	0,0013895	0,0012575
471	0,0016872	0,0015958	0,0013841	0,0012485
472	0,0016783	0,0015870	0,0013787	0,0012396
473	0,0016694	0,0015783	0,0013734	0,0012307
474	0,0016605	0,0015696	0,0013680	0,0012218

475	0,0016517	0,0015610	0,0013627	0,0012129
476	0,0016429	0,0015524	0,0013574	0,0012041
477	0,0016342	0,0015438	0,0013522	0,0011953
478	0,0016254	0,0015352	0,0013469	0,0011866
479	0,0016167	0,0015267	0,0013417	0,0011778
480	0,0016081	0,0015183	0,0013366	0,0011692
481	0,0015995	0,0015098	0,0013314	0,0011605
482	0,0015909	0,0015014	0,0013263	0,0011519
483	0,0015823	0,0014930	0,0013212	0,0011433
484	0,0015738	0,0014847	0,0013161	0,0011347
485	0,0015653	0,0014763	0,0013111	0,0011262
486	0,0015569	0,0014680	0,0013061	0,0011177
487	0,0015484	0,0014598	0,0013011	0,0011093
488	0,0015401	0,0014516	0,0012961	0,0011009
489	0,0015317	0,0014434	0,0012912	0,0010925
490	0,0015234	0,0014352	0,0012862	0,0010841
491	0,0015151	0,0014271	0,0012813	0,0010758
492	0,0015068	0,0014190	0,0012765	0,0010675
493	0,0014986	0,0014109	0,0012716	0,0010592
494	0,0014904	0,0014029	0,0012668	0,0010510
495	0,0014822	0,0013949	0,0012620	0,0010428
496	0,0014741	0,0013869	0,0012573	0,0010346
497	0,0014660	0,0013789	0,0012525	0,0010264
498	0,0014579	0,0013710	0,0012478	0,0010183
499	0,0014499	0,0013631	0,0012431	0,0010102
500	0,0014419	0,0013553	0,0012384	0,0010022

Presentación completa del set de entrenamiento	**Parámetro de regularización de la complejidad**			
	landa=0	***landa=0,05***	***landa=0,15***	***landa=0,1***
1	0,0913720	0,1260930	0,0965570	0,1044060
2	0,0912560	0,1210820	0,0959290	0,1040190
3	0,0911400	0,1198060	0,0955290	0,1036990
4	0,0910250	0,1184920	0,0952590	0,1034020
5	0,0909100	0,1170960	0,0950620	0,1031120
6	0,0907960	0,1155980	0,0949050	0,1028230
7	0,0906810	0,1139820	0,0947710	0,1025330
8	0,0905670	0,1122320	0,0946500	0,1022390
9	0,0904540	0,1103330	0,0945360	0,1019430
10	0,0903400	0,1082770	0,0944250	0,1016430
11	0,0902270	0,1060550	0,0943170	0,1013390

12	0,0901140	0,1036660	0,0942090	0,1010310
13	0,0900010	0,1011120	0,0941020	0,1007180
14	0,0898880	0,0984030	0,0939950	0,1004010
15	0,0897760	0,0955548	0,0938880	0,1000790
16	0,0896640	0,0925886	0,0937810	0,0997530
17	0,0895520	0,0895317	0,0936730	0,0994210
18	0,0894400	0,0864152	0,0935640	0,0990850
19	0,0893280	0,0832724	0,0934550	0,0987430
20	0,0892160	0,0801365	0,0933460	0,0983970
21	0,0891040	0,0770392	0,0932350	0,0980440
22	0,0889920	0,0740086	0,0931240	0,0976870
23	0,0888800	0,0710681	0,0930130	0,0973230
24	0,0887680	0,0682360	0,0929000	0,0969540
25	0,0886560	0,0655254	0,0927870	0,0965790
26	0,0885440	0,0629445	0,0926730	0,0961980
27	0,0884320	0,0604972	0,0925580	0,0958110
28	0,0883200	0,0581840	0,0924420	0,0954190
29	0,0882080	0,0560028	0,0923250	0,0950200
30	0,0880950	0,0539496	0,0922080	0,0946140
31	0,0879820	0,0520189	0,0920890	0,0942030
32	0,0878700	0,0502046	0,0919700	0,0937850
33	0,0877560	0,0485000	0,0918490	0,0933600
34	0,0876430	0,0468982	0,0917280	0,0929300
35	0,0875300	0,0453927	0,0916050	0,0924930
36	0,0874160	0,0439767	0,0914820	0,0920490
37	0,0873020	0,0426440	0,0913570	0,0915990
38	0,0871870	0,0413887	0,0912310	0,0911420
39	0,0870730	0,0402052	0,0911040	0,0906780
40	0,0869570	0,0390883	0,0909760	0,0902090
41	0,0868420	0,0380332	0,0908470	0,0897320
42	0,0867260	0,0370353	0,0907160	0,0892490
43	0,0866100	0,0360906	0,0905850	0,0887600
44	0,0864930	0,0351952	0,0904520	0,0882640
45	0,0863760	0,0343457	0,0903170	0,0877620
46	0,0862590	0,0335387	0,0901820	0,0872530
47	0,0861400	0,0327714	0,0900450	0,0867390
48	0,0860220	0,0320410	0,0899070	0,0862180
49	0,0859030	0,0313450	0,0897670	0,0856910
50	0,0857830	0,0306810	0,0896260	0,0851580
51	0,0856630	0,0300471	0,0894830	0,0846190
52	0,0855420	0,0294411	0,0893400	0,0840739
53	0,0854210	0,0288614	0,0891940	0,0835235
54	0,0852990	0,0283061	0,0890470	0,0829676
55	0,0851770	0,0277739	0,0888990	0,0824063
56	0,0850530	0,0272633	0,0887490	0,0818399

57	0,0849300	0,0267730	0,0885980	0,0812683
58	0,0848050	0,0263017	0,0884440	0,0806919
59	0,0846800	0,0258483	0,0882900	0,0801108
60	0,0845540	0,0254119	0,0881340	0,0795251
61	0,0844270	0,0249914	0,0879760	0,0789351
62	0,0843000	0,0245859	0,0878160	0,0783410
63	0,0841720	0,0241946	0,0876550	0,0777431
64	0,0840430	0,0238168	0,0874920	0,0771414
65	0,0839130	0,0234516	0,0873270	0,0765362
66	0,0837830	0,0230985	0,0871610	0,0759279
67	0,0836510	0,0227568	0,0869920	0,0753165
68	0,0835190	0,0224258	0,0868220	0,0747024
69	0,0833860	0,0221052	0,0866510	0,0740858
70	0,0832520	0,0217942	0,0864770	0,0734669
71	0,0831170	0,0214925	0,0863020	0,0728460
72	0,0829810	0,0211996	0,0861240	0,0722233
73	0,0828450	0,0209151	0,0859450	0,0715992
74	0,0827070	0,0206386	0,0857640	0,0709738
75	0,0825690	0,0203697	0,0855810	0,0703474
76	0,0824290	0,0201081	0,0853960	0,0697204
77	0,0822890	0,0198534	0,0852090	0,0690928
78	0,0821470	0,0196053	0,0850200	0,0684651
79	0,0820050	0,0193635	0,0848300	0,0678374
80	0,0818610	0,0191278	0,0846370	0,0672100
81	0,0817170	0,0188979	0,0844420	0,0665833
82	0,0815710	0,0186736	0,0842450	0,0659573
83	0,0814240	0,0184546	0,0840460	0,0653323
84	0,0812760	0,0182407	0,0838460	0,0647087
85	0,0811280	0,0180317	0,0836430	0,0640867
86	0,0809780	0,0178274	0,0834380	0,0634663
87	0,0808260	0,0176277	0,0832310	0,0628480
88	0,0806740	0,0174323	0,0830220	0,0622319
89	0,0805210	0,0172410	0,0828110	0,0616183
90	0,0803660	0,0170539	0,0825970	0,0610073
91	0,0802100	0,0168706	0,0823820	0,0603991
92	0,0800540	0,0166910	0,0821640	0,0597940
93	0,0798950	0,0165151	0,0819450	0,0591921
94	0,0797360	0,0163426	0,0817230	0,0585935
95	0,0795750	0,0161735	0,0814990	0,0579986
96	0,0794140	0,0160077	0,0812730	0,0574074
97	0,0792510	0,0158450	0,0810450	0,0568201
98	0,0790860	0,0156853	0,0808150	0,0562368
99	0,0789210	0,0155286	0,0805820	0,0556577
100	0,0787540	0,0153748	0,0803480	0,0550830
101	0,0785860	0,0152236	0,0801110	0,0545126

102	0,0784160	0,0150752	0,0798720	0,0539468
103	0,0782450	0,0149293	0,0796310	0,0533857
104	0,0780730	0,0147860	0,0793880	0,0528294
105	0,0779000	0,0146451	0,0791430	0,0522778
106	0,0777250	0,0145065	0,0788960	0,0517313
107	0,0775490	0,0143703	0,0786460	0,0511897
108	0,0773710	0,0142363	0,0783940	0,0506533
109	0,0771930	0,0141044	0,0781410	0,0501220
110	0,0770120	0,0139747	0,0778850	0,0495959
111	0,0768310	0,0138470	0,0776270	0,0490750
112	0,0766480	0,0137213	0,0773670	0,0485595
113	0,0764630	0,0135975	0,0771050	0,0480493
114	0,0762780	0,0134756	0,0768400	0,0475445
115	0,0760900	0,0133556	0,0765740	0,0470450
116	0,0759020	0,0132374	0,0763060	0,0465510
117	0,0757120	0,0131209	0,0760350	0,0460625
118	0,0755200	0,0130061	0,0757630	0,0455793
119	0,0753280	0,0128930	0,0754880	0,0451017
120	0,0751330	0,0127815	0,0752120	0,0446294
121	0,0749380	0,0126716	0,0749340	0,0441626
122	0,0747400	0,0125632	0,0746530	0,0437013
123	0,0745420	0,0124564	0,0743710	0,0432454
124	0,0743420	0,0123510	0,0740870	0,0427949
125	0,0741400	0,0122471	0,0738010	0,0423497
126	0,0739370	0,0121445	0,0735120	0,0419100
127	0,0737330	0,0120434	0,0732230	0,0414756
128	0,0735270	0,0119436	0,0729310	0,0410465
129	0,0733200	0,0118451	0,0726370	0,0406226
130	0,0731110	0,0117479	0,0723419	0,0402041
131	0,0729010	0,0116520	0,0720448	0,0397907
132	0,0726890	0,0115573	0,0717459	0,0393825
133	0,0724760	0,0114638	0,0714453	0,0389794
134	0,0722610	0,0113715	0,0711430	0,0385814
135	0,0720450	0,0112804	0,0708390	0,0381885
136	0,0718280	0,0111904	0,0705334	0,0378006
137	0,0716090	0,0111015	0,0702262	0,0374176
138	0,0713890	0,0110137	0,0699174	0,0370395
139	0,0711670	0,0109270	0,0696071	0,0366662
140	0,0709440	0,0108413	0,0692952	0,0362978
141	0,0707190	0,0107567	0,0689819	0,0359341
142	0,0704930	0,0106731	0,0686671	0,0355751
143	0,0702650	0,0105905	0,0683508	0,0352208
144	0,0700360	0,0105088	0,0680332	0,0348710
145	0,0698060	0,0104282	0,0677143	0,0345258
146	0,0695740	0,0103484	0,0673940	0,0341850

147	0,0693410	0,0102696	0,0670724	0,0338487
148	0,0691070	0,0101917	0,0667496	0,0335168
149	0,0688710	0,0101147	0,0664256	0,0331891
150	0,0686340	0,0100385	0,0661004	0,0328657
151	0,0683950	0,0099632	0,0657741	0,0325465
152	0,0681550	0,0098888	0,0654467	0,0322315
153	0,0679140	0,0098152	0,0651182	0,0319205
154	0,0676710	0,0097424	0,0647887	0,0316136
155	0,0674270	0,0096704	0,0644582	0,0313106
156	0,0671820	0,0095992	0,0641268	0,0310116
157	0,0669360	0,0095288	0,0637945	0,0307164
158	0,0666880	0,0094592	0,0634613	0,0304250
159	0,0664390	0,0093903	0,0631273	0,0301374
160	0,0661890	0,0093221	0,0627925	0,0298535
161	0,0659370	0,0092547	0,0624570	0,0295732
162	0,0656840	0,0091880	0,0621208	0,0292965
163	0,0654305	0,0091220	0,0617839	0,0290234
164	0,0651754	0,0090567	0,0614464	0,0287537
165	0,0649191	0,0089920	0,0611083	0,0284875
166	0,0646617	0,0089281	0,0607697	0,0282246
167	0,0644031	0,0088648	0,0604306	0,0279651
168	0,0641433	0,0088022	0,0600910	0,0277089
169	0,0638825	0,0087402	0,0597511	0,0274558
170	0,0636206	0,0086788	0,0594107	0,0272060
171	0,0633576	0,0086181	0,0590701	0,0269593
172	0,0630936	0,0085580	0,0587291	0,0267157
173	0,0628286	0,0084985	0,0583880	0,0264751
174	0,0625625	0,0084396	0,0580466	0,0262374
175	0,0622955	0,0083813	0,0577051	0,0260028
176	0,0620274	0,0083236	0,0573634	0,0257710
177	0,0617585	0,0082664	0,0570217	0,0255420
178	0,0614886	0,0082098	0,0566799	0,0253159
179	0,0612178	0,0081538	0,0563382	0,0250925
180	0,0609461	0,0080983	0,0559965	0,0248719
181	0,0606736	0,0080433	0,0556549	0,0246539
182	0,0604002	0,0079889	0,0553134	0,0244385
183	0,0601260	0,0079351	0,0549720	0,0242257
184	0,0598510	0,0078817	0,0546309	0,0240155
185	0,0595753	0,0078288	0,0542900	0,0238078
186	0,0592988	0,0077765	0,0539494	0,0236025
187	0,0590215	0,0077246	0,0536092	0,0233997
188	0,0587436	0,0076733	0,0532692	0,0231992
189	0,0584650	0,0076224	0,0529297	0,0230011
190	0,0581858	0,0075720	0,0525906	0,0228053
191	0,0579059	0,0075221	0,0522520	0,0226118

192	0,0576255	0,0074726	0,0519139	0,0224205
193	0,0573445	0,0074236	0,0515763	0,0222314
194	0,0570629	0,0073751	0,0512393	0,0220445
195	0,0567808	0,0073270	0,0509029	0,0218598
196	0,0564982	0,0072794	0,0505672	0,0216771
197	0,0562151	0,0072322	0,0502321	0,0214965
198	0,0559316	0,0071854	0,0498977	0,0213179
199	0,0556477	0,0071390	0,0495641	0,0211414
200	0,0553634	0,0070931	0,0492313	0,0209668
201	0,0550787	0,0070476	0,0488992	0,0207941
202	0,0547937	0,0070024	0,0485680	0,0206234
203	0,0545084	0,0069577	0,0482377	0,0204545
204	0,0542227	0,0069134	0,0479082	0,0202875
205	0,0539369	0,0068695	0,0475797	0,0201223
206	0,0536508	0,0068260	0,0472521	0,0199589
207	0,0533644	0,0067828	0,0469255	0,0197973
208	0,0530779	0,0067401	0,0465999	0,0196374
209	0,0527913	0,0066977	0,0462753	0,0194792
210	0,0525045	0,0066557	0,0459518	0,0193227
211	0,0522176	0,0066140	0,0456294	0,0191679
212	0,0519306	0,0065727	0,0453080	0,0190147
213	0,0516436	0,0065318	0,0449878	0,0188631
214	0,0513566	0,0064912	0,0446688	0,0187130
215	0,0510695	0,0064510	0,0443509	0,0185646
216	0,0507825	0,0064111	0,0440342	0,0184176
217	0,0504956	0,0063716	0,0437187	0,0182722
218	0,0502087	0,0063324	0,0434045	0,0181283
219	0,0499219	0,0062935	0,0430915	0,0179859
220	0,0496352	0,0062549	0,0427798	0,0178448
221	0,0493487	0,0062167	0,0424693	0,0177052
222	0,0490624	0,0061788	0,0421602	0,0175671
223	0,0487763	0,0061412	0,0418524	0,0174303
224	0,0484903	0,0061039	0,0415459	0,0172948
225	0,0482047	0,0060670	0,0412408	0,0171607
226	0,0479193	0,0060303	0,0409371	0,0170279
227	0,0476342	0,0059940	0,0406347	0,0168964
228	0,0473494	0,0059579	0,0403338	0,0167662
229	0,0470650	0,0059221	0,0400342	0,0166373
230	0,0467809	0,0058867	0,0397361	0,0165096
231	0,0464972	0,0058515	0,0394394	0,0163831
232	0,0462140	0,0058166	0,0391442	0,0162578
233	0,0459311	0,0057819	0,0388504	0,0161337
234	0,0456487	0,0057476	0,0385581	0,0160108
235	0,0453668	0,0057135	0,0382673	0,0158891
236	0,0450854	0,0056797	0,0379779	0,0157685

237	0,0448044	0,0056462	0,0376901	0,0156490
238	0,0445241	0,0056130	0,0374037	0,0155306
239	0,0442442	0,0055800	0,0371189	0,0154134
240	0,0439650	0,0055472	0,0368356	0,0152972
241	0,0436863	0,0055148	0,0365538	0,0151820
242	0,0434082	0,0054825	0,0362735	0,0150679
243	0,0431308	0,0054506	0,0359948	0,0149548
244	0,0428540	0,0054188	0,0357176	0,0148428
245	0,0425779	0,0053874	0,0354420	0,0147318
246	0,0423025	0,0053561	0,0351679	0,0146217
247	0,0420277	0,0053251	0,0348953	0,0145126
248	0,0417537	0,0052944	0,0346243	0,0144045
249	0,0414804	0,0052639	0,0343549	0,0142973
250	0,0412078	0,0052336	0,0340870	0,0141911
251	0,0409361	0,0052035	0,0338207	0,0140858
252	0,0406650	0,0051737	0,0335560	0,0139814
253	0,0403948	0,0051441	0,0332928	0,0138779
254	0,0401254	0,0051147	0,0330312	0,0137753
255	0,0398568	0,0050855	0,0327712	0,0136736
256	0,0395891	0,0050566	0,0325127	0,0135727
257	0,0393221	0,0050279	0,0322558	0,0134727
258	0,0390561	0,0049994	0,0320005	0,0133735
259	0,0387909	0,0049710	0,0317467	0,0132752
260	0,0385266	0,0049430	0,0314945	0,0131776
261	0,0382632	0,0049151	0,0312438	0,0130809
262	0,0380007	0,0048874	0,0309947	0,0129850
263	0,0377392	0,0048599	0,0307472	0,0128898
264	0,0374785	0,0048326	0,0305012	0,0127955
265	0,0372188	0,0048055	0,0302568	0,0127019
266	0,0369601	0,0047786	0,0300139	0,0126091
267	0,0367023	0,0047519	0,0297726	0,0125170
268	0,0364454	0,0047254	0,0295328	0,0124256
269	0,0361896	0,0046991	0,0292945	0,0123350
270	0,0359347	0,0046730	0,0290578	0,0122451
271	0,0356809	0,0046471	0,0288226	0,0121559
272	0,0354280	0,0046213	0,0285889	0,0120674
273	0,0351761	0,0045957	0,0283568	0,0119796
274	0,0349253	0,0045703	0,0281262	0,0118924
275	0,0346755	0,0045451	0,0278970	0,0118060
276	0,0344267	0,0045201	0,0276694	0,0117202
277	0,0341790	0,0044952	0,0274433	0,0116351
278	0,0339322	0,0044705	0,0272187	0,0115506
279	0,0336866	0,0044460	0,0269956	0,0114668
280	0,0334420	0,0044216	0,0267739	0,0113836
281	0,0331984	0,0043975	0,0265538	0,0113010

282	0,0329560	0,0043734	0,0263351	0,0112190
283	0,0327146	0,0043496	0,0261178	0,0111377
284	0,0324742	0,0043259	0,0259021	0,0110569
285	0,0322350	0,0043024	0,0256877	0,0109768
286	0,0319968	0,0042790	0,0254748	0,0108972
287	0,0317597	0,0042558	0,0252634	0,0108183
288	0,0315237	0,0042327	0,0250534	0,0107399
289	0,0312888	0,0042098	0,0248448	0,0106620
290	0,0310549	0,0041871	0,0246376	0,0105848
291	0,0308222	0,0041645	0,0244318	0,0105081
292	0,0305906	0,0041420	0,0242275	0,0104319
293	0,0303600	0,0041197	0,0240245	0,0103563
294	0,0301306	0,0040976	0,0238229	0,0102812
295	0,0299023	0,0040756	0,0236227	0,0102066
296	0,0296750	0,0040537	0,0234239	0,0101326
297	0,0294489	0,0040320	0,0232264	0,0100591
298	0,0292239	0,0040104	0,0230303	0,0099860
299	0,0290000	0,0039889	0,0228355	0,0099135
300	0,0287772	0,0039676	0,0226421	0,0098415
301	0,0285555	0,0039465	0,0224499	0,0097700
302	0,0283349	0,0039255	0,0222592	0,0096990
303	0,0281154	0,0039046	0,0220697	0,0096285
304	0,0278970	0,0038838	0,0218815	0,0095584
305	0,0276798	0,0038632	0,0216947	0,0094888
306	0,0274636	0,0038427	0,0215091	0,0094197
307	0,0272486	0,0038223	0,0213248	0,0093510
308	0,0270346	0,0038021	0,0211418	0,0092828
309	0,0268217	0,0037820	0,0209601	0,0092151
310	0,0266100	0,0037620	0,0207796	0,0091477
311	0,0263993	0,0037421	0,0206003	0,0090809
312	0,0261898	0,0037224	0,0204223	0,0090144
313	0,0259813	0,0037028	0,0202455	0,0089484
314	0,0257740	0,0036833	0,0200700	0,0088829
315	0,0255677	0,0036639	0,0198957	0,0088177
316	0,0253625	0,0036446	0,0197225	0,0087530
317	0,0251584	0,0036255	0,0195506	0,0086887
318	0,0249554	0,0036065	0,0193799	0,0086247
319	0,0247535	0,0035876	0,0192103	0,0085612
320	0,0245527	0,0035688	0,0190420	0,0084981
321	0,0243529	0,0035501	0,0188748	0,0084354
322	0,0241542	0,0035315	0,0187087	0,0083731
323	0,0239566	0,0035131	0,0185438	0,0083111
324	0,0237601	0,0034947	0,0183800	0,0082496
325	0,0235646	0,0034765	0,0182174	0,0081884
326	0,0233702	0,0034584	0,0180559	0,0081276

327	0,0231768	0,0034404	0,0178955	0,0080672
328	0,0229845	0,0034224	0,0177362	0,0080071
329	0,0227933	0,0034046	0,0175780	0,0079474
330	0,0226031	0,0033869	0,0174209	0,0078881
331	0,0224139	0,0033693	0,0172649	0,0078291
332	0,0222258	0,0033518	0,0171100	0,0077705
333	0,0220387	0,0033345	0,0169561	0,0077122
334	0,0218527	0,0033172	0,0168033	0,0076543
335	0,0216677	0,0033000	0,0166516	0,0075967
336	0,0214837	0,0032829	0,0165008	0,0075394
337	0,0213007	0,0032659	0,0163512	0,0074825
338	0,0211188	0,0032490	0,0162025	0,0074259
339	0,0209379	0,0032322	0,0160549	0,0073697
340	0,0207579	0,0032155	0,0159082	0,0073137
341	0,0205790	0,0031989	0,0157626	0,0072581
342	0,0204011	0,0031823	0,0156180	0,0072028
343	0,0202242	0,0031659	0,0154744	0,0071478
344	0,0200483	0,0031496	0,0153317	0,0070932
345	0,0198733	0,0031333	0,0151900	0,0070388
346	0,0196994	0,0031172	0,0150493	0,0069848
347	0,0195264	0,0031011	0,0149095	0,0069310
348	0,0193544	0,0030852	0,0147707	0,0068776
349	0,0191833	0,0030693	0,0146328	0,0068244
350	0,0190133	0,0030535	0,0144959	0,0067716
351	0,0188441	0,0030378	0,0143599	0,0067190
352	0,0186760	0,0030222	0,0142248	0,0066667
353	0,0185088	0,0030066	0,0140906	0,0066147
354	0,0183425	0,0029912	0,0139573	0,0065630
355	0,0181772	0,0029758	0,0138249	0,0065116
356	0,0180128	0,0029605	0,0136935	0,0064604
357	0,0178494	0,0029453	0,0135628	0,0064095
358	0,0176868	0,0029302	0,0134331	0,0063589
359	0,0175252	0,0029152	0,0133042	0,0063086
360	0,0173645	0,0029002	0,0131762	0,0062585
361	0,0172048	0,0028853	0,0130491	0,0062087
362	0,0170459	0,0028705	0,0129228	0,0061592
363	0,0168879	0,0028558	0,0127973	0,0061099
364	0,0167308	0,0028412	0,0126727	0,0060609
365	0,0165747	0,0028266	0,0125489	0,0060121
366	0,0164194	0,0028121	0,0124259	0,0059636
367	0,0162650	0,0027977	0,0123038	0,0059154
368	0,0161114	0,0027833	0,0121824	0,0058674
369	0,0159588	0,0027691	0,0120618	0,0058196
370	0,0158070	0,0027549	0,0119421	0,0057721
371	0,0156560	0,0027408	0,0118231	0,0057248

372	0,0155060	0,0027267	0,0117049	0,0056777
373	0,0153567	0,0027128	0,0115875	0,0056309
374	0,0152084	0,0026989	0,0114708	0,0055844
375	0,0150608	0,0026850	0,0113549	0,0055380
376	0,0149141	0,0026713	0,0112398	0,0054919
377	0,0147683	0,0026576	0,0111254	0,0054461
378	0,0146232	0,0026440	0,0110117	0,0054004
379	0,0144790	0,0026304	0,0108988	0,0053550
380	0,0143356	0,0026169	0,0107866	0,0053098
381	0,0141930	0,0026035	0,0106752	0,0052648
382	0,0140513	0,0025902	0,0105644	0,0052200
383	0,0139103	0,0025769	0,0104544	0,0051755
384	0,0137701	0,0025637	0,0103451	0,0051312
385	0,0136307	0,0025505	0,0102364	0,0050871
386	0,0134921	0,0025374	0,0101285	0,0050432
387	0,0133543	0,0025244	0,0100213	0,0049995
388	0,0132173	0,0025115	0,0099147	0,0049560
389	0,0130810	0,0024986	0,0098088	0,0049127
390	0,0129455	0,0024857	0,0097036	0,0048696
391	0,0128108	0,0024730	0,0095991	0,0048268
392	0,0126769	0,0024603	0,0094952	0,0047841
393	0,0125436	0,0024476	0,0093920	0,0047416
394	0,0124112	0,0024350	0,0092894	0,0046993
395	0,0122795	0,0024225	0,0091874	0,0046573
396	0,0121485	0,0024100	0,0090861	0,0046154
397	0,0120182	0,0023976	0,0089855	0,0045737
398	0,0118887	0,0023853	0,0088854	0,0045322
399	0,0117600	0,0023730	0,0087860	0,0044909
400	0,0116319	0,0023608	0,0086872	0,0044498
401	0,0115045	0,0023486	0,0085891	0,0044089
402	0,0113779	0,0023365	0,0084915	0,0043681
403	0,0112520	0,0023245	0,0083945	0,0043275
404	0,0111268	0,0023125	0,0082981	0,0042872
405	0,0110022	0,0023005	0,0082023	0,0042470
406	0,0108784	0,0022886	0,0081071	0,0042070
407	0,0107553	0,0022768	0,0080125	0,0041671
408	0,0106328	0,0022650	0,0079185	0,0041274
409	0,0105110	0,0022533	0,0078250	0,0040880
410	0,0103899	0,0022417	0,0077321	0,0040486
411	0,0102695	0,0022300	0,0076398	0,0040095
412	0,0101497	0,0022185	0,0075480	0,0039705
413	0,0100306	0,0022070	0,0074568	0,0039317
414	0,0099122	0,0021955	0,0073661	0,0038931
415	0,0097944	0,0021841	0,0072760	0,0038546
416	0,0096772	0,0021728	0,0071864	0,0038163

417	0,0095607	0,0021615	0,0070973	0,0037782
418	0,0094449	0,0021502	0,0070088	0,0037402
419	0,0093297	0,0021390	0,0069208	0,0037024
420	0,0092151	0,0021279	0,0068333	0,0036647
421	0,0091011	0,0021168	0,0067464	0,0036273
422	0,0089878	0,0021057	0,0066599	0,0035899
423	0,0088750	0,0020947	0,0065740	0,0035527
424	0,0087629	0,0020838	0,0064886	0,0035157
425	0,0086514	0,0020729	0,0064036	0,0034789
426	0,0085406	0,0020620	0,0063192	0,0034421
427	0,0084303	0,0020512	0,0062353	0,0034056
428	0,0083206	0,0020405	0,0061518	0,0033692
429	0,0082115	0,0020298	0,0060689	0,0033329
430	0,0081030	0,0020191	0,0059864	0,0032968
431	0,0079951	0,0020085	0,0059044	0,0032608
432	0,0078878	0,0019979	0,0058228	0,0032250
433	0,0077810	0,0019874	0,0057418	0,0031894
434	0,0076748	0,0019769	0,0056612	0,0031538
435	0,0075692	0,0019665	0,0055810	0,0031185
436	0,0074642	0,0019561	0,0055013	0,0030832
437	0,0073597	0,0019458	0,0054221	0,0030481
438	0,0072558	0,0019355	0,0053433	0,0030132
439	0,0071524	0,0019252	0,0052650	0,0029784
440	0,0070496	0,0019150	0,0051871	0,0029437
441	0,0069473	0,0019048	0,0051097	0,0029091
442	0,0068456	0,0018947	0,0050326	0,0028747
443	0,0067444	0,0018846	0,0049561	0,0028405
444	0,0066438	0,0018746	0,0048799	0,0028063
445	0,0065437	0,0018646	0,0048042	0,0027723
446	0,0064441	0,0018546	0,0047289	0,0027385
447	0,0063450	0,0018447	0,0046540	0,0027047
448	0,0062465	0,0018348	0,0045795	0,0026711
449	0,0061485	0,0018250	0,0045055	0,0026377
450	0,0060510	0,0018152	0,0044318	0,0026043
451	0,0059540	0,0018055	0,0043585	0,0025711
452	0,0058575	0,0017958	0,0042857	0,0025380
453	0,0057615	0,0017861	0,0042132	0,0025051
454	0,0056660	0,0017764	0,0041412	0,0024722
455	0,0055711	0,0017669	0,0040695	0,0024395
456	0,0054766	0,0017573	0,0039983	0,0024069
457	0,0053826	0,0017478	0,0039274	0,0023745
458	0,0052891	0,0017383	0,0038569	0,0023421
459	0,0051960	0,0017289	0,0037867	0,0023099
460	0,0051035	0,0017195	0,0037170	0,0022778
461	0,0050114	0,0017101	0,0036476	0,0022458

462	0,0049198	0,0017008	0,0035786	0,0022140
463	0,0048287	0,0016915	0,0035100	0,0021822
464	0,0047381	0,0016823	0,0034417	0,0021506
465	0,0046479	0,0016731	0,0033738	0,0021191
466	0,0045581	0,0016639	0,0033062	0,0020877
467	0,0044689	0,0016547	0,0032390	0,0020564
468	0,0043800	0,0016456	0,0031722	0,0020253
469	0,0042917	0,0016366	0,0031057	0,0019942
470	0,0042037	0,0016275	0,0030395	0,0019633
471	0,0041163	0,0016185	0,0029737	0,0019325
472	0,0040292	0,0016096	0,0029083	0,0019017
473	0,0039426	0,0016007	0,0028432	0,0018711
474	0,0038565	0,0015918	0,0027784	0,0018407
475	0,0037708	0,0015829	0,0027139	0,0018103
476	0,0036855	0,0015741	0,0026498	0,0017800
477	0,0036006	0,0015653	0,0025860	0,0017498
478	0,0035162	0,0015566	0,0025226	0,0017198
479	0,0034321	0,0015478	0,0024594	0,0016898
480	0,0033485	0,0015392	0,0023966	0,0016600
481	0,0032653	0,0015305	0,0023341	0,0016302
482	0,0031826	0,0015219	0,0022720	0,0016006
483	0,0031002	0,0015133	0,0022101	0,0015711
484	0,0030182	0,0015047	0,0021485	0,0015417
485	0,0029367	0,0014962	0,0020873	0,0015123
486	0,0028555	0,0014877	0,0020264	0,0014831
487	0,0027748	0,0014793	0,0019657	0,0014540
488	0,0026944	0,0014709	0,0019054	0,0014250
489	0,0026145	0,0014625	0,0018454	0,0013960
490	0,0025349	0,0014541	0,0017856	0,0013672
491	0,0024557	0,0014458	0,0017262	0,0013385
492	0,0023769	0,0014375	0,0016670	0,0013099
493	0,0022985	0,0014292	0,0016082	0,0012814
494	0,0022204	0,0014210	0,0015496	0,0012529
495	0,0021428	0,0014128	0,0014914	0,0012246
496	0,0020655	0,0014046	0,0014334	0,0011964
497	0,0019886	0,0013964	0,0013756	0,0011682
498	0,0019120	0,0013883	0,0013182	0,0011402
499	0,0018359	0,0013802	0,0012611	0,0011122
500	0,0017601	0,0013722	0,0012042	0,0010844

Presentación completa del set de entrenamiento	Parámetro de detención temprana			
	beta=2	***beta=infinito***	***beta=10***	***beta=5***
1	0,1293020	0,1364700	0,1317160	0,1312120
2	0,1217330	0,1277520	0,1119030	0,1222550
3	0,1111540	0,1182190	0,0884183	0,1215780
4	0,0974177	0,1048520	0,0685726	0,1207410
5	0,0823357	0,0886156	0,0553354	0,1198490
6	0,0686955	0,0729958	0,0469834	0,1189000
7	0,0579326	0,0604916	0,0414966	0,1178770
8	0,0499490	0,0512894	0,0375803	0,1167660
9	0,0441023	0,0446400	0,0345850	0,1155510
10	0,0397658	0,0397767	0,0321951	0,1142150
11	0,0364669	0,0361266	0,0302406	0,1127450
12	0,0338791	0,0332977	0,0286135	0,1111240
13	0,0317848	0,0310309	0,0272344	0,1093400
14	0,0300405	0,0291574	0,0260414	0,1073820
15	0,0285513	0,0275681	0,0249862	0,1052430
16	0,0272537	0,0261914	0,0240317	0,1029170
17	0,0261043	0,0249796	0,0231495	0,1004070
18	0,0250729	0,0238994	0,0223194	0,0977180
19	0,0241375	0,0229266	0,0215270	0,0948655
20	0,0232822	0,0220423	0,0207636	0,0918684
21	0,0224944	0,0212315	0,0200242	0,0887538
22	0,0217642	0,0204822	0,0193061	0,0855536
23	0,0210835	0,0197843	0,0186083	0,0823037
24	0,0204454	0,0191299	0,0179300	0,0790415
25	0,0198444	0,0185124	0,0172707	0,0758039
26	0,0192755	0,0179268	0,0166298	0,0726247
27	0,0187347	0,0173692	0,0160074	0,0695335
28	0,0182188	0,0168366	0,0154038	0,0665540
29	0,0177250	0,0163266	0,0148204	0,0637039
30	0,0172510	0,0158375	0,0142592	0,0609950
31	0,0167951	0,0153680	0,0137227	0,0584340
32	0,0163557	0,0149171	0,0132135	0,0560227
33	0,0159318	0,0144841	0,0127337	0,0537596
34	0,0155223	0,0140684	0,0122847	0,0516407
35	0,0151266	0,0136695	0,0118670	0,0496598
36	0,0147440	0,0132869	0,0114803	0,0478098
37	0,0143741	0,0129202	0,0111235	0,0460828
38	0,0140165	0,0125690	0,0107947	0,0444706

39	0,0136709	0,0122330	0,0104919	0,0429651
40	0,0133370	0,0119115	0,0102127	0,0415584
41	0,0130144	0,0116041	0,0099547	0,0402428
42	0,0127030	0,0113104	0,0097154	0,0390113
43	0,0124025	0,0110297	0,0094924	0,0378571
44	0,0121126	0,0107614	0,0092837	0,0367740
45	0,0118330	0,0105051	0,0090870	0,0357563
46	0,0115636	0,0102601	0,0089006	0,0347988
47	0,0113039	0,0100258	0,0087225	0,0338966
48	0,0110537	0,0098017	0,0085511	0,0330453
49	0,0108126	0,0095871	0,0083849	0,0322409
50	0,0105804	0,0093815	0,0082224	0,0314798
51	0,0103566	0,0091844	0,0080623	0,0307586
52	0,0101409	0,0089953	0,0079036	0,0300742
53	0,0099330	0,0088135	0,0077451	0,0294240
54	0,0097326	0,0086388	0,0075859	0,0288053
55	0,0095392	0,0084705	0,0074255	0,0282159
56	0,0093526	0,0083084	0,0072631	0,0276536
57	0,0091724	0,0081519	0,0070984	0,0271167
58	0,0089983	0,0080007	0,0069313	0,0266033
59	0,0088300	0,0078545	0,0067617	0,0261117
60	0,0086671	0,0077129	0,0065897	0,0256407
61	0,0085095	0,0075757	0,0064159	0,0251888
62	0,0083569	0,0074425	0,0062408	0,0247548
63	0,0082089	0,0073131	0,0060653	0,0243375
64	0,0080653	0,0071872	0,0058902	0,0239360
65	0,0079260	0,0070648	0,0057168	0,0235492
66	0,0077906	0,0069454	0,0055463	0,0231763
67	0,0076591	0,0068290	0,0053797	0,0228165
68	0,0075311	0,0067154	0,0052185	0,0224690
69	0,0074065	0,0066045	0,0050634	0,0221331
70	0,0072852	0,0064961	0,0049155	0,0218082
71	0,0071670	0,0063901	0,0047753	0,0214937
72	0,0070518	0,0062864	0,0046430	0,0211890
73	0,0069394	0,0061848	0,0045189	0,0208936
74	0,0068296	0,0060854	0,0044027	0,0206070
75	0,0067224	0,0059880	0,0042942	0,0203287
76	0,0066177	0,0058925	0,0041930	0,0200585
77	0,0065154	0,0057990	0,0040985	0,0197958
78	0,0064153	0,0057073	0,0040103	0,0195403
79	0,0063174	0,0056173	0,0039278	0,0192916
80	0,0062216	0,0055291	0,0038505	0,0190495
81	0,0061279	0,0054426	0,0037779	0,0188137

82	0,0060360	0,0053577	0,0037096	0,0185838
83	0,0059461	0,0052745	0,0036451	0,0183596
84	0,0058580	0,0051928	0,0035841	0,0181408
85	0,0057717	0,0051127	0,0035262	0,0179273
86	0,0056871	0,0050341	0,0034711	0,0177187
87	0,0056041	0,0049570	0,0034186	0,0175149
88	0,0055228	0,0048814	0,0033682	0,0173158
89	0,0054430	0,0048072	0,0033199	0,0171210
90	0,0053647	0,0047345	0,0032734	0,0169305
91	0,0052880	0,0046631	0,0032285	0,0167441
92	0,0052126	0,0045931	0,0031850	0,0165615
93	0,0051387	0,0045244	0,0031428	0,0163828
94	0,0050661	0,0044571	0,0031018	0,0162076
95	0,0049949	0,0043910	0,0030619	0,0160360
96	0,0049249	0,0043262	0,0030229	0,0158677
97	0,0048563	0,0042626	0,0029848	0,0157027
98	0,0047888	0,0042003	0,0029474	0,0155408
99	0,0047226	0,0041391	0,0029108	0,0153820
100	0,0046575	0,0040790	0,0028750	0,0152261
101	0,0045935	0,0040201	0,0028397	0,0150730
102	0,0045307	0,0039623	0,0028051	0,0149227
103	0,0044690	0,0039056	0,0027712	0,0147750
104	0,0044083	0,0038499	0,0027378	0,0146299
105	0,0043486	0,0037952	0,0027050	0,0144873
106	0,0042900	0,0037415	0,0026727	0,0143471
107	0,0042323	0,0036887	0,0026411	0,0142093
108	0,0041756	0,0036369	0,0026101	0,0140737
109	0,0041198	0,0035860	0,0025796	0,0139403
110	0,0040649	0,0035360	0,0025497	0,0138091
111	0,0040109	0,0034868	0,0025205	0,0136799
112	0,0039577	0,0034385	0,0024918	0,0135528
113	0,0039054	0,0033910	0,0024638	0,0134276
114	0,0038539	0,0033442	0,0024363	0,0133044
115	0,0038032	0,0032982	0,0024095	0,0131830
116	0,0037533	0,0032530	0,0023833	0,0130634
117	0,0037042	0,0032085	0,0023577	0,0129456
118	0,0036557	0,0031647	0,0023327	0,0128295
119	0,0036080	0,0031215	0,0023083	0,0127151
120	0,0035611	0,0030791	0,0022845	0,0126023
121	0,0035148	0,0030373	0,0022613	0,0124912
122	0,0034692	0,0029961	0,0022387	0,0123815
123	0,0034242	0,0029555	0,0022166	0,0122734
124	0,0033799	0,0029156	0,0021950	0,0121668

125	0,0033362	0,0028762	0,0021740	0,0120616
126	0,0032931	0,0028374	0,0021535	0,0119579
127	0,0032507	0,0027992	0,0021335	0,0118555
128	0,0032088	0,0027615	0,0021140	0,0117545
129	0,0031676	0,0027244	0,0020949	0,0116548
130	0,0031269	0,0026879	0,0020763	0,0115564
131	0,0030867	0,0026518	0,0020581	0,0114592
132	0,0030471	0,0026163	0,0020404	0,0113634
133	0,0030081	0,0025813	0,0020231	0,0112687
134	0,0029696	0,0025468	0,0020062	0,0111752
135	0,0029316	0,0025129	0,0019897	0,0110829
136	0,0028942	0,0024794	0,0019736	0,0109917
137	0,0028573	0,0024464	0,0019578	0,0109016
138	0,0028209	0,0024139	0,0019425	0,0108126
139	0,0027849	0,0023818	0,0019274	0,0107247
140	0,0027495	0,0023503	0,0019128	0,0106379
141	0,0027146	0,0023192	0,0018984	0,0105521
142	0,0026801	0,0022886	0,0018844	0,0104673
143	0,0026462	0,0022584	0,0018707	0,0103835
144	0,0026127	0,0022287	0,0018574	0,0103007
145	0,0025796	0,0021995	0,0018443	0,0102188
146	0,0025471	0,0021707	0,0018315	0,0101379
147	0,0025150	0,0021424	0,0018190	0,0100579
148	0,0024833	0,0021144	0,0018068	0,0099789
149	0,0024521	0,0020870	0,0017949	0,0099007
150	0,0024214	0,0020599	0,0017832	0,0098234
151	0,0023910	0,0020333	0,0017718	0,0097469
152	0,0023611	0,0020070	0,0017607	0,0096713
153	0,0023317	0,0019812	0,0017498	0,0095966
154	0,0023027	0,0019558	0,0017391	0,0095227
155	0,0022740	0,0019308	0,0017287	0,0094495
156	0,0022458	0,0019062	0,0017185	0,0093772
157	0,0022180	0,0018820	0,0017086	0,0093057
158	0,0021907	0,0018582	0,0016988	0,0092349
159	0,0021637	0,0018347	0,0016893	0,0091648
160	0,0021371	0,0018116	0,0016799	0,0090956
161	0,0021109	0,0017889	0,0016708	0,0090270
162	0,0020851	0,0017665	0,0016619	0,0089592
163	0,0020596	0,0017445	0,0016532	0,0088921
164	0,0020346	0,0017229	0,0016446	0,0088257
165	0,0020099	0,0017015	0,0016362	0,0087600
166	0,0019856	0,0016805	0,0016280	0,0086949
167	0,0019616	0,0016599	0,0016200	0,0086306

168	0,0019380	0,0016395	0,0016122	0,0085668
169	0,0019147	0,0016195	0,0016045	0,0085038
170	0,0018918	0,0015998	0,0015970	0,0084414
171	0,0018692	0,0015804	0,0015896	0,0083796
172	0,0018470	0,0015612	0,0015824	0,0083184
173	0,0018251	0,0015424	0,0015753	0,0082579
174	0,0018035	0,0015239	0,0015684	0,0081979
175	0,0017822	0,0015056	0,0015616	0,0081385
176	0,0017613	0,0014877	0,0015550	0,0080798
177	0,0017406	0,0014700	0,0015484	0,0080216
178	0,0017203	0,0014525	0,0015421	0,0079640
179	0,0017002	0,0014353	0,0015358	0,0079069
180	0,0016805	0,0014184	0,0015297	0,0078504
181	0,0016610	0,0014018	0,0015237	0,0077945
182	0,0016418	0,0013853	0,0015178	0,0077391
183	0,0016229	0,0013692	0,0015120	0,0076842
184	0,0016043	0,0013532	0,0015063	0,0076298
185	0,0015859	0,0013375	0,0015008	0,0075760
186	0,0015678	0,0013221	0,0014953	0,0075227
187	0,0015500	0,0013068	0,0014900	0,0074699
188	0,0015324	0,0012918	0,0014848	0,0074176
189	0,0015151	0,0012770	0,0014796	0,0073658
190	0,0014980	0,0012624	0,0014746	0,0073144
191	0,0014812	0,0012481	0,0014696	0,0072636
192	0,0014646	0,0012339	0,0014648	0,0072132
193	0,0014483	0,0012199	0,0014600	0,0071633
194	0,0014322	0,0012062	0,0014553	0,0071138
195	0,0014163	0,0011926	0,0014507	0,0070649
196	0,0014006	0,0011793	0,0014462	0,0070163
197	0,0013852	0,0011661	0,0014418	0,0069682
198	0,0013700	0,0011531	0,0014374	0,0069205
199	0,0013550	0,0011403	0,0014331	0,0068733
200	0,0013402	0,0011277	0,0014289	0,0068265
201	0,0013256	0,0011153	0,0014248	0,0067801
202	0,0013112	0,0011030	0,0014207	0,0067342
203	0,0012971	0,0010909	0,0014168	0,0066886
204	0,0012831	0,0010790	0,0014128	0,0066435
205	0,0012693	0,0010673	0,0014090	0,0065988
206	0,0012557	0,0010557	0,0014052	0,0065544
207	0,0012423	0,0010443	0,0014015	0,0065105
208	0,0012291	0,0010331	0,0013978	0,0064669
209	0,0012161	0,0010220	0,0013942	0,0064237
210	0,0012032	0,0010111	0,0013907	0,0063809

211	0,0011905	0,0010003	0,0013872	0,0063385
212	0,0011780	0,0009897	0,0013838	0,0062964
213	0,0011657	0,0009792	0,0013804	0,0062547
214	0,0011535	0,0009689	0,0013771	0,0062134
215	0,0011415	0,0009587	0,0013738	0,0061724
216	0,0011297	0,0009487	0,0013706	0,0061318
217	0,0011180	0,0009388	0,0013675	0,0060915
218	0,0011065	0,0009290	0,0013644	0,0060516
219	0,0010951	0,0009194	0,0013613	0,0060120
220	0,0010839	0,0009099	0,0013583	0,0059727
221	0,0010729	0,0009006	0,0013553	0,0059338
222	0,0010620	0,0008913	0,0013524	0,0058952
223	0,0010512	0,0008822	0,0013495	0,0058569
224	0,0010406	0,0008733	0,0013467	0,0058190
225	0,0010301	0,0008644	0,0013439	0,0057814
226	0,0010198	0,0008557	0,0013411	0,0057440
227	0,0010096	0,0008471	0,0013384	0,0057070
228	0,0009995	0,0008386	0,0013358	0,0056703
229	0,0009896	0,0008303	0,0013331	0,0056339
230	0,0009798	0,0008220	0,0013305	0,0055978
231	0,0009701	0,0008139	0,0013280	0,0055620
232	0,0009606	0,0008059	0,0013255	0,0055265
233	0,0009512	0,0007979	0,0013230	0,0054912
234	0,0009419	0,0007901	0,0013205	0,0054563
235	0,0009327	0,0007824	0,0013181	0,0054216
236	0,0009237	0,0007748	0,0013158	0,0053872
237	0,0009147	0,0007674	0,0013134	0,0053531
238	0,0009059	0,0007600	0,0013111	0,0053193
239	0,0008972	0,0007527	0,0013088	0,0052857
240	0,0008886	0,0007455	0,0013066	0,0052524
241	0,0008801	0,0007384	0,0013043	0,0052194
242	0,0008718	0,0007314	0,0013022	0,0051866
243	0,0008635	0,0007245	0,0013000	0,0051541
244	0,0008553	0,0007177	0,0012979	0,0051218
245	0,0008473	0,0007109	0,0012958	0,0050898
246	0,0008393	0,0007043	0,0012937	0,0050581
247	0,0008315	0,0006977	0,0012916	0,0050265
248	0,0008237	0,0006913	0,0012896	0,0049953
249	0,0008161	0,0006849	0,0012876	0,0049642
250	0,0008085	0,0006786	0,0012857	0,0049335
251	0,0008011	0,0006724	0,0012837	0,0049029
252	0,0007937	0,0006663	0,0012818	0,0048726
253	0,0007864	0,0006602	0,0012799	0,0048425

254	0,0007793	0,0006542	0,0012780	0,0048126
255	0,0007722	0,0006483	0,0012762	0,0047830
256	0,0007652	0,0006425	0,0012743	0,0047536
257	0,0007582	0,0006368	0,0012725	0,0047244
258	0,0007514	0,0006311	0,0012708	0,0046955
259	0,0007446	0,0006255	0,0012690	0,0046667
260	0,0007380	0,0006200	0,0012673	0,0046382
261	0,0007314	0,0006145	0,0012655	0,0046098
262	0,0007249	0,0006091	0,0012638	0,0045817
263	0,0007185	0,0006038	0,0012622	0,0045538
264	0,0007121	0,0005985	0,0012605	0,0045261
265	0,0007058	0,0005933	0,0012589	0,0044986
266	0,0006996	0,0005882	0,0012573	0,0044713
267	0,0006935	0,0005832	0,0012557	0,0044442
268	0,0006875	0,0005782	0,0012541	0,0044173
269	0,0006815	0,0005732	0,0012525	0,0043906
270	0,0006756	0,0005683	0,0012510	0,0043641
271	0,0006698	0,0005635	0,0012494	0,0043378
272	0,0006640	0,0005587	0,0012479	0,0043117
273	0,0006583	0,0005540	0,0012464	0,0042857
274	0,0006527	0,0005494	0,0012450	0,0042599
275	0,0006471	0,0005448	0,0012435	0,0042344
276	0,0006416	0,0005403	0,0012421	0,0042090
277	0,0006362	0,0005358	0,0012406	0,0041838
278	0,0006308	0,0005314	0,0012392	0,0041587
279	0,0006255	0,0005270	0,0012378	0,0041339
280	0,0006202	0,0005227	0,0012365	0,0041092
281	0,0006151	0,0005184	0,0012351	0,0040846
282	0,0006099	0,0005142	0,0012337	0,0040603
283	0,0006049	0,0005100	0,0012324	0,0040361
284	0,0005998	0,0005059	0,0012311	0,0040121
285	0,0005949	0,0005018	0,0012298	0,0039883
286	0,0005900	0,0004977	0,0012285	0,0039646
287	0,0005851	0,0004938	0,0012272	0,0039411
288	0,0005804	0,0004898	0,0012259	0,0039177
289	0,0005756	0,0004859	0,0012247	0,0038945
290	0,0005709	0,0004821	0,0012234	0,0038714
291	0,0005663	0,0004783	0,0012222	0,0038486
292	0,0005617	0,0004745	0,0012210	0,0038258
293	0,0005572	0,0004708	0,0012198	0,0038032
294	0,0005527	0,0004671	0,0012186	0,0037808
295	0,0005483	0,0004635	0,0012174	0,0037585
296	0,0005439	0,0004599	0,0012162	0,0037364

297	0,0005396	0,0004563	0,0012151	0,0037144
298	0,0005353	0,0004528	0,0012139	0,0036926
299	0,0005311	0,0004493	0,0012128	0,0036709
300	0,0005269	0,0004459	0,0012117	0,0036493
301	0,0005227	0,0004424	0,0012106	0,0036279
302	0,0005186	0,0004391	0,0012095	0,0036066
303	0,0005145	0,0004357	0,0012084	0,0035855
304	0,0005105	0,0004324	0,0012073	0,0035645
305	0,0005066	0,0004292	0,0012062	0,0035436
306	0,0005026	0,0004260	0,0012052	0,0035229
307	0,0004987	0,0004228	0,0012041	0,0035023
308	0,0004949	0,0004196	0,0012031	0,0034818
309	0,0004911	0,0004165	0,0012021	0,0034615
310	0,0004873	0,0004134	0,0012011	0,0034413
311	0,0004836	0,0004103	0,0012000	0,0034212
312	0,0004799	0,0004073	0,0011990	0,0034012
313	0,0004762	0,0004043	0,0011981	0,0033814
314	0,0004726	0,0004013	0,0011971	0,0033617
315	0,0004690	0,0003984	0,0011961	0,0033421
316	0,0004655	0,0003955	0,0011951	0,0033227
317	0,0004620	0,0003926	0,0011942	0,0033033
318	0,0004585	0,0003898	0,0011932	0,0032841
319	0,0004551	0,0003869	0,0011923	0,0032650
320	0,0004517	0,0003842	0,0011914	0,0032460
321	0,0004483	0,0003814	0,0011905	0,0032272
322	0,0004450	0,0003787	0,0011895	0,0032084
323	0,0004417	0,0003760	0,0011886	0,0031898
324	0,0004384	0,0003733	0,0011877	0,0031713
325	0,0004352	0,0003706	0,0011869	0,0031528
326	0,0004319	0,0003680	0,0011860	0,0031345
327	0,0004288	0,0003654	0,0011851	0,0031164
328	0,0004256	0,0003628	0,0011842	0,0030983
329	0,0004225	0,0003603	0,0011834	0,0030803
330	0,0004194	0,0003577	0,0011825	0,0030624
331	0,0004164	0,0003552	0,0011817	0,0030447
332	0,0004134	0,0003528	0,0011809	0,0030270
333	0,0004104	0,0003503	0,0011800	0,0030095
334	0,0004074	0,0003479	0,0011792	0,0029920
335	0,0004045	0,0003455	0,0011784	0,0029747
336	0,0004016	0,0003431	0,0011776	0,0029575
337	0,0003987	0,0003407	0,0011768	0,0029403
338	0,0003958	0,0003384	0,0011760	0,0029233
339	0,0003930	0,0003361	0,0011752	0,0029063

340	0,0003902	0,0003338	0,0011744	0,0028895
341	0,0003874	0,0003315	0,0011737	0,0028727
342	0,0003847	0,0003292	0,0011729	0,0028561
343	0,0003819	0,0003270	0,0011721	0,0028395
344	0,0003793	0,0003248	0,0011714	0,0028231
345	0,0003766	0,0003226	0,0011706	0,0028067
346	0,0003739	0,0003204	0,0011699	0,0027904
347	0,0003713	0,0003183	0,0011691	0,0027742
348	0,0003687	0,0003161	0,0011684	0,0027581
349	0,0003661	0,0003140	0,0011677	0,0027421
350	0,0003636	0,0003119	0,0011670	0,0027262
351	0,0003610	0,0003099	0,0011662	0,0027104
352	0,0003585	0,0003078	0,0011655	0,0026947
353	0,0003560	0,0003058	0,0011648	0,0026790
354	0,0003536	0,0003038	0,0011641	0,0026635
355	0,0003511	0,0003017	0,0011635	0,0026480
356	0,0003487	0,0002998	0,0011628	0,0026326
357	0,0003463	0,0002978	0,0011621	0,0026173
358	0,0003439	0,0002958	0,0011614	0,0026021
359	0,0003416	0,0002939	0,0011607	0,0025869
360	0,0003393	0,0002920	0,0011601	0,0025719
361	0,0003369	0,0002901	0,0011594	0,0025569
362	0,0003346	0,0002882	0,0011588	0,0025420
363	0,0003324	0,0002863	0,0011581	0,0025272
364	0,0003301	0,0002845	0,0011575	0,0025125
365	0,0003279	0,0002827	0,0011568	0,0024978
366	0,0003257	0,0002808	0,0011562	0,0024832
367	0,0003235	0,0002790	0,0011556	0,0024687
368	0,0003213	0,0002772	0,0011549	0,0024543
369	0,0003191	0,0002755	0,0011543	0,0024400
370	0,0003170	0,0002737	0,0011537	0,0024257
371	0,0003148	0,0002720	0,0011531	0,0024115
372	0,0003127	0,0002702	0,0011525	0,0023974
373	0,0003106	0,0002685	0,0011519	0,0023833
374	0,0003086	0,0002668	0,0011513	0,0023693
375	0,0003065	0,0002651	0,0011507	0,0023554
376	0,0003045	0,0002635	0,0011501	0,0023416
377	0,0003024	0,0002618	0,0011495	0,0023278
378	0,0003004	0,0002601	0,0011489	0,0023142
379	0,0002985	0,0002585	0,0011483	0,0023005
380	0,0002965	0,0002569	0,0011478	0,0022870
381	0,0002945	0,0002553	0,0011472	0,0022735
382	0,0002926	0,0002537	0,0011466	0,0022601

383	0,0002907	0,0002521	0,0011461	0,0022468
384	0,0002887	0,0002505	0,0011455	0,0022335
385	0,0002869	0,0002490	0,0011450	0,0022203
386	0,0002850	0,0002474	0,0011444	0,0022071
387	0,0002831	0,0002459	0,0011439	0,0021941
388	0,0002813	0,0002444	0,0011433	0,0021810
389	0,0002794	0,0002429	0,0011428	0,0021681
390	0,0002776	0,0002414	0,0011423	0,0021552
391	0,0002758	0,0002399	0,0011417	0,0021424
392	0,0002740	0,0002384	0,0011412	0,0021296
393	0,0002722	0,0002370	0,0011407	0,0021169
394	0,0002705	0,0002355	0,0011402	0,0021043
395	0,0002687	0,0002341	0,0011397	0,0020918
396	0,0002670	0,0002327	0,0011391	0,0020792
397	0,0002653	0,0002312	0,0011386	0,0020668
398	0,0002635	0,0002298	0,0011381	0,0020544
399	0,0002618	0,0002284	0,0011376	0,0020421
400	0,0002602	0,0002270	0,0011371	0,0020298
401	0,0002585	0,0002257	0,0011366	0,0020176
402	0,0002568	0,0002243	0,0011361	0,0020055
403	0,0002552	0,0002230	0,0011356	0,0019934
404	0,0002536	0,0002216	0,0011352	0,0019814
405	0,0002519	0,0002203	0,0011347	0,0019694
406	0,0002503	0,0002190	0,0011342	0,0019575
407	0,0002487	0,0002176	0,0011337	0,0019456
408	0,0002471	0,0002163	0,0011333	0,0019338
409	0,0002456	0,0002150	0,0011328	0,0019220
410	0,0002440	0,0002138	0,0011323	0,0019104
411	0,0002425	0,0002125	0,0011319	0,0018987
412	0,0002409	0,0002112	0,0011314	0,0018871
413	0,0002394	0,0002100	0,0011309	0,0018756
414	0,0002379	0,0002087	0,0011305	0,0018641
415	0,0002364	0,0002075	0,0011300	0,0018527
416	0,0002349	0,0002062	0,0011296	0,0018413
417	0,0002334	0,0002050	0,0011291	0,0018300
418	0,0002319	0,0002038	0,0011287	0,0018187
419	0,0002304	0,0002026	0,0011283	0,0018075
420	0,0002290	0,0002014	0,0011278	0,0017964
421	0,0002276	0,0002002	0,0011274	0,0017852
422	0,0002261	0,0001990	0,0011269	0,0017742
423	0,0002247	0,0001979	0,0011265	0,0017632
424	0,0002233	0,0001967	0,0011261	0,0017522
425	0,0002219	0,0001955	0,0011257	0,0017413

426	0,0002205	0,0001944	0,0011252	0,0017304
427	0,0002191	0,0001933	0,0011248	0,0017196
428	0,0002177	0,0001921	0,0011244	0,0017088
429	0,0002164	0,0001910	0,0011240	0,0016981
430	0,0002150	0,0001899	0,0011236	0,0016875
431	0,0002137	0,0001888	0,0011232	0,0016768
432	0,0002124	0,0001877	0,0011228	0,0016663
433	0,0002110	0,0001866	0,0011224	0,0016557
434	0,0002097	0,0001855	0,0011220	0,0016452
435	0,0002084	0,0001844	0,0011216	0,0016348
436	0,0002071	0,0001834	0,0011212	0,0016244
437	0,0002058	0,0001823	0,0011208	0,0016141
438	0,0002045	0,0001812	0,0011204	0,0016038
439	0,0002033	0,0001802	0,0011200	0,0015935
440	0,0002020	0,0001792	0,0011196	0,0015833
441	0,0002007	0,0001781	0,0011192	0,0015731
442	0,0001995	0,0001771	0,0011188	0,0015630
443	0,0001983	0,0001761	0,0011185	0,0015529
444	0,0001970	0,0001751	0,0011181	0,0015429
445	0,0001958	0,0001741	0,0011177	0,0015329
446	0,0001946	0,0001731	0,0011173	0,0015229
447	0,0001934	0,0001721	0,0011170	0,0015130
448	0,0001922	0,0001711	0,0011166	0,0015032
449	0,0001910	0,0001701	0,0011162	0,0014933
450	0,0001898	0,0001691	0,0011159	0,0014836
451	0,0001887	0,0001681	0,0011155	0,0014738
452	0,0001875	0,0001672	0,0011151	0,0014641
453	0,0001863	0,0001662	0,0011148	0,0014545
454	0,0001852	0,0001653	0,0011144	0,0014448
455	0,0001840	0,0001643	0,0011141	0,0014353
456	0,0001829	0,0001634	0,0011137	0,0014257
457	0,0001818	0,0001625	0,0011134	0,0014162
458	0,0001807	0,0001615	0,0011130	0,0014068
459	0,0001795	0,0001606	0,0011127	0,0013973
460	0,0001784	0,0001597	0,0011123	0,0013880
461	0,0001773	0,0001588	0,0011120	0,0013786
462	0,0001762	0,0001579	0,0011117	0,0013693
463	0,0001752	0,0001570	0,0011113	0,0013600
464	0,0001741	0,0001561	0,0011110	0,0013508
465	0,0001730	0,0001552	0,0011106	0,0013416
466	0,0001719	0,0001543	0,0011103	0,0013325
467	0,0001709	0,0001535	0,0011100	0,0013233
468	0,0001698	0,0001526	0,0011097	0,0013143

469	0,0001688	0,0001517	0,0011093	0,0013052
470	0,0001677	0,0001509	0,0011090	0,0012962
471	0,0001667	0,0001500	0,0011087	0,0012872
472	0,0001657	0,0001492	0,0011084	0,0012783
473	0,0001647	0,0001483	0,0011080	0,0012694
474	0,0001636	0,0001475	0,0011077	0,0012605
475	0,0001626	0,0001467	0,0011074	0,0012517
476	0,0001616	0,0001458	0,0011071	0,0012429
477	0,0001606	0,0001450	0,0011068	0,0012342
478	0,0001597	0,0001442	0,0011065	0,0012254
479	0,0001587	0,0001434	0,0011062	0,0012167
480	0,0001577	0,0001426	0,0011058	0,0012081
481	0,0001567	0,0001418	0,0011055	0,0011995
482	0,0001558	0,0001410	0,0011052	0,0011909
483	0,0001548	0,0001402	0,0011049	0,0011823
484	0,0001538	0,0001394	0,0011046	0,0011738
485	0,0001529	0,0001386	0,0011043	0,0011653
486	0,0001520	0,0001378	0,0011040	0,0011569
487	0,0001510	0,0001370	0,0011037	0,0011484
488	0,0001501	0,0001363	0,0011034	0,0011401
489	0,0001492	0,0001355	0,0011031	0,0011317
490	0,0001482	0,0001347	0,0011029	0,0011234
491	0,0001473	0,0001340	0,0011026	0,0011151
492	0,0001464	0,0001332	0,0011023	0,0011068
493	0,0001455	0,0001325	0,0011020	0,0010986
494	0,0001446	0,0001317	0,0011017	0,0010904
495	0,0001437	0,0001310	0,0011014	0,0010822
496	0,0001428	0,0001303	0,0011011	0,0010741
497	0,0001419	0,0001295	0,0011008	0,0010660
498	0,0001411	0,0001288	0,0011006	0,0010579
499	0,0001402	0,0001281	0,0011003	0,0010499
500	0,0001393	0,0001274	0,0011000	0,0010419