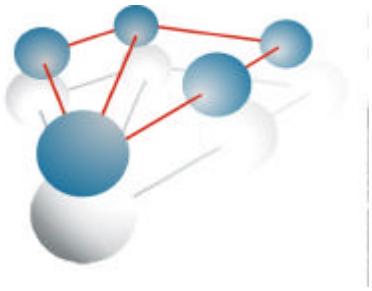
Universidad Nacional del Nordeste
Facultad de Ciencias Exactas, Naturales y Agrimensura

Trabajo Final de Aplicación

SIMULACIÓN DE REDES NEURONALES ARTIFICIALES

Una Aplicación Didáctica



Alumna: Anita del C. Alegre López - L.U.: 25.014

Prof. Coordinador: Agr. Castor Herrmann
Prof. Orientadora: Mgter. Sonia I. Mariño

Licenciatura en Sistemas
Corrientes - Argentina

2003

A mi niñita Victoria.

Prefacio

Unas de las múltiples técnicas que emplea la Inteligencia Arti…cial para simular el comportamiento inteligente de los seres humanos, son las denominadas Redes Neuronales Arti…ciales. Las mismas seducen a profesionales de distintas disciplinas por el gran potencial que ofrecen para resolver problemas complejos.

El principal objetivo de este Trabajo Final de Aplicación es estudiar e investigar el modo de aprendizaje de las redes neuronales y sus aplicaciones didácticas. Por esta razón se presentan los algoritmos más conocidos que ellas utilizan. En las redes neuronales el conocimiento se incorpora mediante el aprendizaje a partir de ejemplos.

Se ha desarrollado un software de simulación en el cual se implementa como método de aprendizaje supervisado el Algoritmo de Retropropagación. Para ello se ha recurrido a la técnica del Ciclo de Vida del Software, dedicando la etapa inicial a la investigación y estudio del problema, para luego pasar a la etapa de diseño e implementación. En ésta se optó por el lenguaje orientado a objetos Java, por su robustez, y también por su capacidad de manejar hilos de ejecución y su arquitectura neutra (multiplataforma).

A continuación se describirá el contenido de este Trabajo Final de Aplicación. El capítulo 1 comienza introduciendo ideas básicas sobre las Redes Neuronales, fundamentales para comprender la simulación desarrollada. En el capítulo 2 se presenta el lenguaje de programación Java. En el capítulo 3 se hace una introducción a la preparación de documentos con LaTeX. En el capítulo 4 se presenta el Sistema de Sumulación de Redes Neuronales Arti…ciales, y …nalizando, en el capítulo 5 se exponen las conclusiones a las que se ha llegado.

Anita del C. Alegre López
Licenciatura en Sistemas
Universidad Nacional del Nordeste

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Redes Neuronales

1.1 Introducción a las Redes Neuronales

La computación paralela y las redes neuronales despiertan en la actualidad un ferviente interés investigador en todo el mundo, de manera tal que se han convertido en dos nuevos paradigmas de la inteligencia arti…cial.

Profesionales de campos tan diversos como la ingeniería, la …losofía y la psicología, intrigados por el potencial ofrecido por esta tecnología, buscan aplicaciones dentro de sus respectivas disciplinas.

El elemento clave de estos paradigmas es una nueva estructura computacional compuesta de un gran número de pequeños elementos procesadores interconectados trabajando en forma paralela. Este procesamiento paralelo permite realizar muchas operaciones simultáneamente, en contraposición al proceso en serie tradicional en el cual los cálculos se realizan en orden secuencial. En la década del 50 se construyeron redes neuronales sencillas, pero no se llegó a obtener grandes progresos debido a la falta de tecnología apropiada y a la ruptura con otras áreas de la inteligencia arti…cial.

El aumento en los recursos computacionales, junto con nuevas técnicas e…cientes de computación, en los años 70, ha renovado el interés hacia este campo.

Los ordenadores que se utilizan hoy en día pueden realizar una gran variedad de tareas (siempre que éstas estén bien de…nidas) a una velocidad y con

una habilidad muy superior a las alcanzables por los seres humanos. Ninguno de nosotros será, por ejemplo, capaz de resolver complejas ecuaciones matemáticas a la velocidad que lo hace una computadora personal. Sin embargo, la capacidad mental del ser humano es todavía muy superior a la de las máquinas en gran cantidad de tareas. Ningún sistema arti…cial de reconocimiento de imágenes es capaz de competir con la capacidad de un ser humano para discernir entre objetos de diversas formas y orientaciones, es más, ni siquiera será capaz de competir con la capacidad de un insecto. Del mismo modo, mientras que una computadora precisa de una enorme cantidad de computación y de condiciones restrictivas para reconocer, por ejemplo, fonemas, un humano adulto reconoce sin ningún esfuerzo palabras pronunciadas por distintas personas, a diferentes velocidades, acentos y entonaciones, incluso en presencia de ruido ambiental.

Se observa pues que, mediante reglas aprendidas de la experiencia, el ser humano se mani…esta mucho más e…caz que las computadoras en la resolución de problemas de…nidos de manera imprecisa, ambiguos, o que requieren procesar gran cantidad de información. Nuestro cerebro alcanza estos objetivos, mediante miles de millones de células simples e interconectadas entre sí, llamadas neuronas, localizadas en un recinto determinado del cerebro.

Ahora bien, se estima que los ampli…cadores operacionales y las puertas lógicas pueden realizar operaciones varios órdenes de magnitud más rápido que las neuronas. Si con estos elementos se siguiera la misma técnica de procesado que los cerebros biológicos, se podrán construir máquinas relativamente baratas y capaces de procesar tanta información, al menos, como la que procesa un cerebro biológico. Lógicamente, saber si se construirán estas máquinas es una incógnita muy lejos de ser contestada en los tiempos actuales.

Así pues, hay razones fundadas que hacen pensar en la viabilidad de abordar ciertos problemas mediante sistemas paralelo que procesen información y aprendan mediante principios tomados de los sistemas cerebrales de los seres vivos. A estos sistemas se los denomina Redes Neuronales (también reciben los nombres de modelos conexionistas, modelos de proceso paralelo distribuido y sistemas neuromór…cos). Las redes neuronales arti…ciales nacieron pues de la intención del hombre de simular de manera arti…cial los sistemas cerebrales biológicos.

1.2 Inspiración en la Neurociencia

Antes de la aparición de las redes neuronales y de la computación paralela, los métodos y las herramientas de computación utilizadas para el procesamiento de la información tenían las siguientes características:

- el conocimiento se representaba explícitamente usando reglas, redes semánticas, modelos probabilísticos, etc.,
- se imitaba el proceso humano de razonamiento lógico para resolver los problemas centrando la atención en las causas que intervienen en el problema y en sus relaciones, y
- se procesaba la información secuencialmente.

Con el vertiginoso desarrollo de algunos campos de la inteligencia arti…cial, como el reconocimiento de patrones, aparecieron un gran número de problemas complejos en los cuales no era conveniente una representación explícita del conocimiento y no se contaba con un procedimiento de razonamiento lógico para resolverlo. Por este motivo, las aproximaciones algorítmicas y las estructuras computacionales estándar no eran apropiadas para resolver estos problemas. Fue así que surgieron las redes neuronales arti…ciales como estructuras de computación alternativas, creadas con el …n de imitar las funciones del cerebro humano.

El cerebro está compuesto por cerca de 10^{11} neuronas que reciben señales electroquímicas de otras neuronas a través de la conexiones sinápticas que unen el axón de las neuronas emisoras y las dentritas de las receptoras (el axón de una neurona típica tiene unas pocas miles de sinapsis con otras neuronas). De acuerdo a las informaciones o impulsos recibidos, la neurona computa y envía su propia señal. Este proceso es controlado por el potencial interno asociado a cada neurona. Si este potencial supera un cierto umbral, se envía un impulso eléctrico al axón; en caso contrario no se envía la señal. Las redes neuronales son entonces modelos computacionales, inspiradas en las características neuro…siológicas anteriores, que están formadas por un gran número de procesadores, o neuronas, dispuestos en varias capas e interconectadas entre sí mediante pesos. Los procesadores realizan cálculos simples basados en la información que reciben de los procesadores vecinos. Cabe aclarar que las redes neuronales no usan reglas de…nidas rígidamente como lo hacen las computadoras digitales más convencionales, sino que usan un proceso de aprendizaje

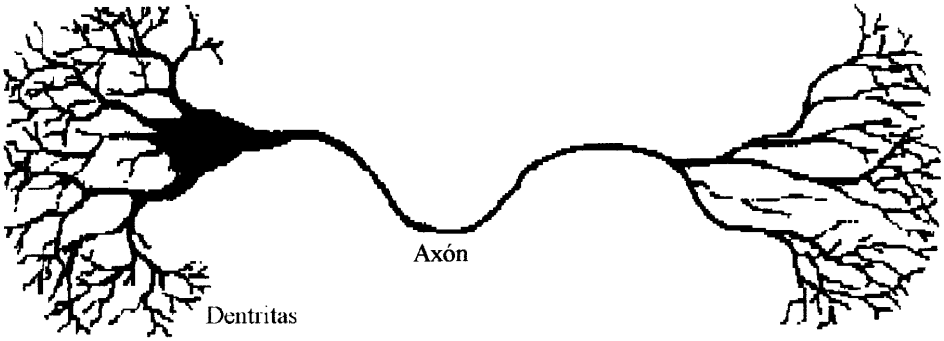


Figura 1.1: Sistema nervioso real simpli…cado.

por analogía donde los pesos de las conexiones son ajustados automáticamente para reproducir un conjunto de patrones representativo del problema a aprender. Este aprendizaje también está inspirado en la forma de aprender que tiene lugar en las neuronas, cambiando la efectividad de las sinapsis, de tal manera que la in‡uencia de una neurona a otra varía.

En la Figura 1.1 de la pág. 4 se expone el grá…co de un sistema nervioso real que ha sido simpli…cado drásticamente a efectos de visualizar la conexión sináptica de las neuronas a través del axón.

1.3 Componentes de las Redes Neuronales

Las siguientes de…niciones describen las principales componentes de una red neuronal arti…cial (RNA).

De¯nici¶on 1:1 : Neurona o Unidad Procesadora

Una Neurona, o Unidad Procesadora, sobre un conjunto de nodos N, es una tripleta $(X; f; Y)$, donde X es un subconjunto de N; Y es un único nodo de N y $f : R \to R$ es una función neuronal (también llamada de activación) que calcula un valor de salida para Y basado en una combinación lineal de los valores de las componentes de X, es decir:

$$Y = f\left(\sum_{x_i \in X} W_i X_i\right):$$

Los elementos $X; Y$ y f de denominan conjunto de nodos de entrada, nodo de salida, y función neuronal de la unidad neuronal, respectivamente.

De¯nici¶on 1:2 : Red Neuronal Arti…cial

Una Red Neuronal Arti…cial (RNA) es un par $(N;U)$, donde N es un subconjunto de nodos y U es un conjunto de unidades procesadoras sobre N que satisface la siguiente condición: cada nodo $X_i \in N$ tiene que ser un nodo de entrada o de salida de al menos una unidad procesadora de U.

La Figura 1.2 de la pág. 6 muestra un ejemplo de una red neuronal con ocho nodos $\{x_1; :::; x_8\}$ y cinco unidades procesadoras:

$$\begin{aligned} U_1 &= (\{x_1, x_2; x_3\}; f_1; \{x_4\}); \\ U_2 &= (\{x_1, x_2; x_3\}; f_1; \{x_5\}); \\ U_3 &= (\{x_1, x_2; x_3\}; f_1; \{x_6\}); \\ U_4 &= (\{x_1, x_2; x_3\}; f_1; \{x_7\}); \text{ y} \\ U_5 &= (\{x_1, x_2; x_3\}; f_1; \{x_8\}); \end{aligned} \tag{1.1}$$

Para favorecer a la simplicidad, las funciones neuronales no están representadas explícitamente en la Figura 1.2 de la pág. 6. La Figura 1.3 de la pág. 6, muestra una descripción detallada de una unidad neuronal típica.

1.3.1 Neuronas: Unidades Procesadoras

Las neuronas son los elementos procesadores de la red neuronal. Como ya se ha visto, una neurona típica, es decir $(\{x_1; :::; x_n\}; f; y_i)$, realiza un sencillo cálculo con las entradas para obtener un valor de salida:

$$y_i = \sum_{j=1} f(w_{ij} x_i); \tag{1.2}$$

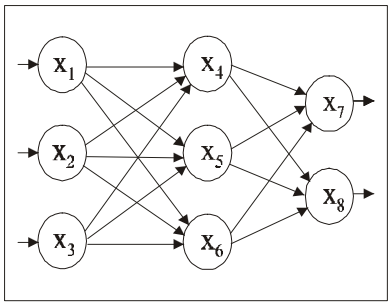


Figura 1.2: Una red neuronal arti…cial.

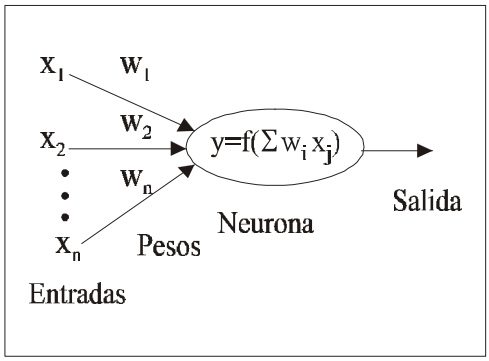


Figura 1.3: La función procesadora de una única neurona.

Donde $f(x)$ es la función neuronal y los pesos w_{ij} pueden ser positivos o negativos, reproduciendo el llamado carácter excitador o inhibitorio de las sinapsis de las neuronas, respectivamente. A veces se utiliza únicamente el nodo de salida y_i para hacer referencia a toda la unidad neuronal.

Un concepto importante es el de la actividad lineal de una neurona y_i, que es simplemente la suma de los pesos de las entradas de otras neuronas:

$$Y_i = \sum_{j=1} w_{ij} x_j : \tag{1.3}$$

Con el …n de adoptar una notación más simple, se usarán letras mayúsculas, Y_i, para hacer referencia a la actividad lineal de la neurona.

Por tanto, la salida de una neurona y_i se obtiene simplemente transformando la actividad lineal usando la función de activación (ver Figura 1.4 de la pág. 8).

En algunos casos, con el objeto de tener en cuenta un valor umbral $Á_i$ para la neurona y_i, se puede conectar una nueva neurona auxiliar $x_0 = ¡1$ a y_i con un peso $w_{i0} = Á_i$.

$$Y_i = \sum_{j=1} w_{ij} x_j ¡ Á_i = \sum_{j=0} w_{ij} x_j : \tag{1.4}$$

La Figura 1.4 de la pág. 8 ilustra los cálculos involucrados en la neurona procesadora, consistentes en la suma de los pesos de entradas procedentes de otras neuronas, incluyendo un valor umbral $Á_i$, y una función procesadora de actividad $f(x)$. Esta de…nición de la actividad neuronal fue sugerida primeramente por Mc-Culloch y Pitts [13] como un modelo matemático simple de actividad neuronal. En particular, ellos consideraban una función de activación de peso binario.

1.3.2 Funciones de activación más populares

A continuación se describen las funciones de activación más populares:

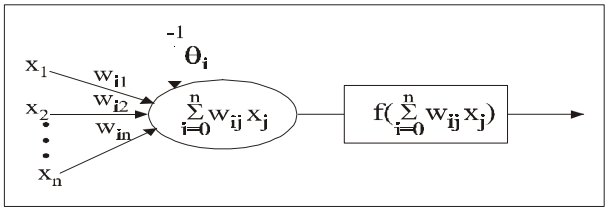


Figura 1.4: Computación esquemática de una neurona.

1. Funciones lineales: Son funciones que dan una salida lineal:

$$f(x) = x; \; x \; 2 \; R$$

2. Funciones escalón: Son funciones que dan una salida binaria dependiente de si el valor de entrada está por encima o por debajo del valor umbral. Por ejemplo la función signo, sgn(x), y la escalón estándar, Á(x), de...-nidas como sigue:

$$\text{sgn}(x) = \begin{cases} -1; & \text{si } x < 0 \\ 1; & \text{en otro caso} \end{cases} \quad ; Á(x) = \begin{cases} 0, & \text{si } x < 0 \\ 1, & \text{en otro caso} \end{cases}$$

3. Funciones sigmoidales: Son funciones monótonas acotadas que dan una salida gradual no lineal para entradas. Las funciones sigmoidales más populares son:

- La función logística:

$$f_c(x) = \frac{1}{1 + e^{-cx}} :$$

- La función tangente hiperbólica de -1 a 1:

$$f_c(x) = \tanh(cx) :$$

1.3.3 Arquitecturas de Redes

Las neuronas se pueden organizar en capas conectadas por varios tipos de uniones incluyendo conexiones hacia delante, laterales y hacia atrás:

- Conexiones hacia delante: Conectan neuronas de una capa con la capa siguiente (ver Figura 1.5 de la pág. 10). Cada una de estas conexiones implica una composición funcional entre las funciones de activación de las correspondientes neuronas y da a la red neuronal la capacidad de reproducir una amplia clase de funciones no lineales.
- Conexiones laterales: Conectan neuronas de la misma capa. Junto con el caso simple de redes de una sola capa (ver Figura 1.6 de la pág. 11), este tipo de conexión se usa generalmente en capas competitivas, donde cada nodo se conecta a sí mismo mediante un peso positivo (excitante) y a los demás nodos de la capa con pesos negativos (inhibitorios, ver Figura 1.7 de la pág. 11).
- Conexiones hacia atrás: Incorporadas a las redes para tratar modelos dinámicos y temporales, es decir, modelos con memoria.

La arquitectura de red se puede representar por una matriz de pesos $W = (w_1; w_2; ....; w_n)$ donde w_i es el vector que contiene los pesos de las conexiones con las demás neuronas como, por ejemplo de la neurona x_j a la neurona $x_i : w_i = (w_{i1}; ...; w_{ij}; ...; w_{in})$.

En algunos casos la topología de la red permite clasi…car las unidades neuronales de la siguiente forma:

De¯nición 1:3 : Capa de Entrada de una Red Neuronal: Una unidad se dice que está en la capa de entrada de una red neuronal $(X; U)$, si es la entrada de al menos una unidad procesadora de U y no es la salida de ninguna unidad procesadora de U.

De¯nición 1:4 : Capa de Salida de una Red Neuronal: Una unidad se dice que está en la capa de salida de una red neuronal $(X; U)$, si es la salida de al menos una unidad procesadora de U y no es la entrada de ninguna unidad procesadora de U.

De¯nición 1:5 : Capas Intermedias u Ocultas de una Red Neuronal: Una unidad se dice que está en la capa intermedia de una red neuronal $(X; U)$, si

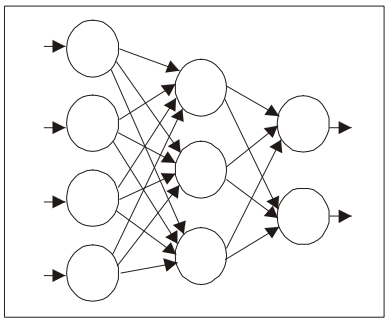


Figura 1.5: Red neuronal de retro-propagación con una entrada consistente en cuatro neuronas, tres unidades ocultas en una única capa, y una salida con dos neuronas.

es la entrada de al menos una unidad procesadora de U y, al mismo tiempo, es la salida de al menos una unidad procesadora de U.

Entre las arquitecturas de red más poderosas y populares se destacan las llamadas redes de retro-propagación, o perceptrones multicapa, que están formadas por una capa de entrada, un número arbitrario de capas ocultas, y una capa de salida. Cada una de las neuronas ocultas o de salida, recibe una entrada de las neuronas de la capa previa (conexiones hacia atrás).

La Figura 1.5 de la pág. 10 muestra una red neuronal de retro-propagación con una entrada consistente en cuatro neuronas, tres unidades ocultas en una única capa, y una salida con dos neuronas.

Otra arquitectura popular y simple es la red de Hop…eld que contiene una única capa y todas las conexiones laterales posibles entre las diferentes neuronas (ver Figura 1.6 de la pág. 11). En este caso, las capas de entrada, intermedia y de salida es la misma capa.

Una RNA también puede incluir capas competitivas, donde las neuronas compiten por tener la mayor actividad para un patrón dado (ver Figura 1.7 de la pág. 11).

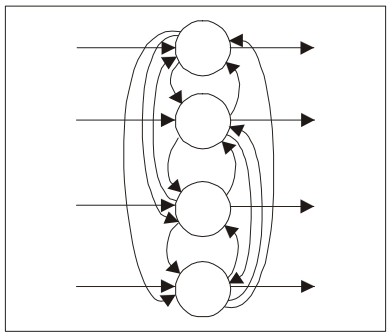


Figura 1.6: Red neuronal de Hop…eld con cuatro nodos.

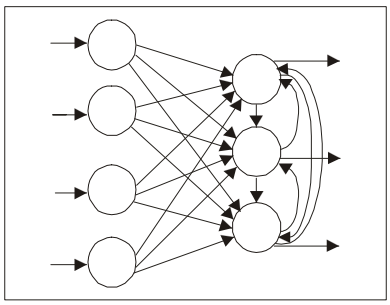


Figura 1.7: Red bicapa de retro-propagación con capa de salida competitiva.

1.4 Aprendizaje

Una de las características de las RNA es su capacidad de aprender a partir de ciertos datos. Una vez que ha sido elegida la arquitectura de red para un determinado problema, los pesos de las conexiones se ajustan para codi…car la información contenida en un conjunto de datos de entrenamiento. Los métodos de aprendizaje se clasi…can en dos categorías [14], dependiendo del tipo de información disponible:

1. Aprendizaje supervisado: En este caso, los patrones para el aprendizaje están formados por parejas, $f(a_p; b_p)$ $p = 1; :::; rg$, que constan de un vector de variables de entrada, a, junto con las salidas correspondientes b. Por tanto cada valor de salida es la respuesta deseada a las señales de entrada. En este caso, los pesos se obtienen minimizando alguna función de error que mide la diferencia entre los valores de salida deseados y los calculados por la red neuronal. En este tipo de aprendizaje se debe tener en cuenta el problema de la convergencia del error. En general, la función de error resultante puede contener múltiples mínimos locales y, por lo tanto, el proceso de aprendizaje puede no converger al mínimo global óptimo.
2. Aprendizaje no supervisado: En este caso, los datos se presentan a la red sin información externa, y la red tiene que descubrir por sí misma patrones o categorías. Este tipo de aprendizaje se encuadra dentro de las técnicas autoorganizativas, o técnicas automáticas para descubrir la estructura de datos. Algunos métodos de aprendizaje no supervisado son:
   - Aprendizaje Hebbiano: consiste en modi…car los pesos de acuerdo con algún criterio de correlación entre las actividades neuronales,
   - Aprendizaje competitivo: neuronas diferentes se conectan con pesos negativos (inhibitorios) que fuerzan una competición para ganar la actividad neuronal, y
   - Representación de características: que concierne a la ordenación geométrica de los vectores peso de las unidades competitivas.

Los datos del aprendizaje no supervisado pueden contener valores de entrada y valores de salida pero, en contraposición al aprendizaje supervisado, no hay información acerca de que salidas corresponden a cada una de las entradas de los datos.

1.5 Validación

Una vez que ha terminado el proceso de aprendizaje y los pesos de la red neuronal han sido calculados, es importante comprobar la calidad del modelo resultante. Por ejemplo, en el caso del aprendizaje supervisado, una medida de la calidad puede darse en términos de los errores entre los valores de salida deseados y los obtenidos por la red neuronal. Algunas medidas estándar del error son:

1. La suma de los cuadrados de los errores, de…nida como:

$$\sum_{p=1} \left(b_p - \hat{b}_p \right)^2 : \tag{1.5}$$

2. La raíz cuadrada del error cuadrático medio, de…nida como:

$$\sqrt{\sum_{p=1} \left(b_p - \hat{b}_p \right)^2 = r} : \tag{1.6}$$

3. El error máximo,

$$\max_{p=1;:::;r} \left(b_p - \hat{b}_p \right); \tag{1.7}$$

donde b_p es la salida de la red para el vector de entrada a_p. Cabe aclarar que en el caso de una única salida, la función jj:jj se reduce a la función valor absoluto j:j usual.

También es necesario realizar una validación cruzada para obtener una medida de la calidad de predicción del modelo. Por este propósito, los datos obtenidos se pueden dividir en dos partes, una parte destinada al entrenamiento de la red y otra parte a la comprobación (testeo o validación). Cuando el error de comprobación es mucho mayor que el error de entrenamiento, se produce un problema de sobreajuste durante el proceso de entrenamiento. En Estadística se sabe que cuando se utiliza un modelo con muchos parámetros para ajustar un conjunto de datos procedente de un proceso con pocos grados

de libertad, el modelo obtenido puede no mostrar las tendencias reales del proceso original, aunque presente un error pequeño de ajuste a los datos. En este caso, el entrenamiento se reduce a la interpolación de los datos, incluyendo el ruido, utilizando una función sigmoidal complicada.

1.6 Redes de Retro-propagación

Las neuronas de este tipo de redes están organizadas en diferentes capas de forma que cada una de las neuronas de una capa puede recibir una entrada de las neuronas de la capa previa (ver Figura 1.5 de la pág. 10). Los perceptrones son las arquitecturas más simples de este tipo de red y consisten en una capa de entrada $\{x_1; :::; x_n\}$, y una salida $\{y_1; :::; y_m\}$ [15]. El número de entradas y de salidas suele denotarse de forma abreviada mediante $n : m$.

Los perceptrones también se denominan redes de retro-propagación de una única capa. La capa de entrada no se contabiliza en el número total de capas, ya que no realiza ningún cálculo. La Figura 1.8 de la pág. 15 muestra un perceptrón 4:3.

En un perceptrón, una unidad de salida típica, y_i, realiza el cálculo:

$$y_i = f(Y_i) = f(\sum_{j=1}^{n} w_{ij} x_j);\ i = 1; :::; m:$$

donde $f(x)$ es la función de activación y w_i el correspondiente vector de peso. Se debe recordar que las letras mayúsculas indican la activación lineal de una neurona, es decir, la combinación de pesos de las entradas.

El problema de aprender este tipo de redes neuronales se reduce a obtener los pesos apropiados para aproximar un conjunto dado de patrones entrada-salida $(a_{p1}; :::; a_{pn}; b_{p1}; :::; b_{pm})$, de tal forma que las salidas $\hat{b}_{p1}; :::; \hat{b}_{pm}$ obtenidas de las neuronas $(y_1; :::; y_m)$ cuando se dan los valores de entrada $(a_{p1}; :::; a_{pn})$ estén lo más cerca posible de los valores de salida deseados $(b_{p1}; :::; b_{pm})$.

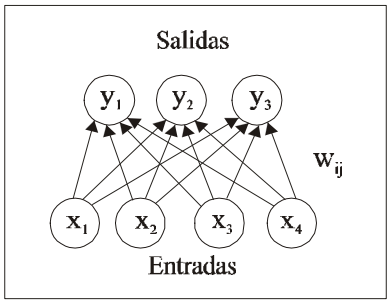


Figura 1.8: Un perceptrón de una única capa con 4 entradas y 3 salidas.

1.6.1 Algoritmos de Aprendizaje

Se procederá a analizar dos populares algoritmos de aprendizaje para perceptrones. El primero de ellos está inspirado en la analogía con el refuerzo de la sinapsis neuronal propuesto por Hebb. El segundo usa un método de optimización matemática para obtener la con…guración de pesos que minimiza una cierta función de error. Como se verá, en el caso de los perceptrones lineales, ambos métodos, el heurístico y el matemático, coinciden.

- Aprendizaje Hebbiano: Como primer paso se eligen los pesos aleatoriamente. Luego se utilizan los patrones, uno a uno, y se modi…can los pesos según la correlación existente entre los valores de entrada y el error resultante:

$$\Delta w_{ij} = \eta \, (b_{pi} - \hat{b}_{pi}) \, a_{pj}; \tag{1.8}$$

  Donde el parámetro η es el índice de aprendizaje, que indica el índice de cambio de los pesos. Nótese que cuando las predicciones coinciden con las salidas deseadas, el peso no cambia.

- Método de descenso de gradiente (la regla delta, “deltarule”): Al igual que en el caso anterior se comienza eligiendo los pesos aleatoriamente. La idea de éste método consiste en variar los pesos de manera que se minimice la suma de los cuadrados de los errores:

$$E(w) = \frac{1}{2} \sum_{i;p} \left(b_{pi} - \hat{b}_{pi} \right)^2 : \tag{1.9}$$

El algoritmo más simple de optimización consiste en "moverse", dentro del espacio de las con…guraciones de los pesos, en el sentido de disminución del gradiente y e la dirección del mismo (método del descenso de gradiente). Esto puede hacerse modi…cando iterativamente cada uno de los pesos w_{ij} mediante un incremento $¢w_{ij}$ proporcional al gradiente del error. Entonces:

$$\begin{aligned} ¢w_{ij} &= - \eta \frac{@E(w)}{@w_{ij}} = - \eta \sum_{p} \left(b_{pi} - \hat{b}_{pi} \right) \frac{@\hat{b}_{pi}}{@w_{ij}} a_{pj;} \\ &= - \eta \sum_{p} \left(b_{pi} - \hat{b}_{pi} \right) f'(B_{pi}) \, a_{pj;} \end{aligned} \tag{1.10}$$

donde B_{pi} es la activación lineal de la neurona b_{pi}, es decir $b_{pi} = f(B_{pi})$ y el parámetro η da el índice de aprendizaje.

En el caso lineal ($f(x) = x$) la fórmula de aprendizaje anterior se reduce a:

$$¢w_{ij} = - \eta \frac{@E}{@w_{ij}} = \eta \sum_{p} \left(b_{pi} - \hat{b}_{pi} \right) a_{pj;} \tag{1.11}$$

que es la misma que la obtenida del método Hebbiano en 1.8 de la pág. 15. Este método de aprendizaje se ilustra esquemáticamente en la Figura 1.9 de la pág. 17.

Cuando la función de activación no es lineal, sino sigmoidal contínua, la Fórmula 1.11 de la pág. 16 se simpli…ca, porque no involucra derivadas formales:

$$f(x) = \frac{1}{1 + e^{-cx}} \Rightarrow f'(x) = c f(x) (1 - f(x)) ;$$

y

$$f(x) = \tanh(cx) \Rightarrow f'(x) = c \left(1 - f(x)^2 \right) :$$

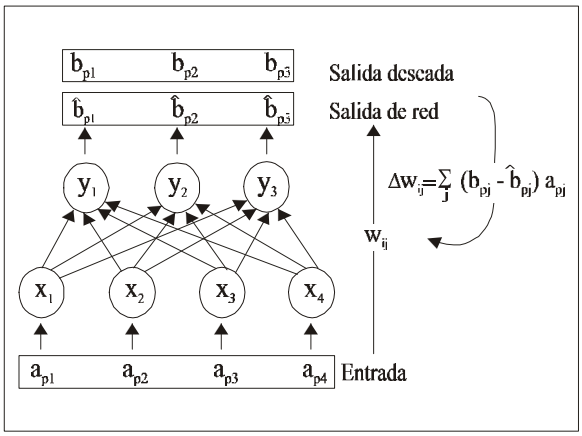


Figura 1.9: Ilustración esquemática del algoritmo de aprendizaje para perceptrones lineales.

1.7 Perceptrones Multi-Capa

Además de una capa de entrada y una de salida, un perceptrón multi-capa tiene capas intermedias ocultas. Los nodos de la capa de entrada alimentan la red distribuyendo hacia delante las señales de entrada. Cada nodo en las capas ocultas y de salida recibe una entrada de los nodos de las capas previas y calcula un valor de salida para la siguiente capa. La Figura 1.10 de la pág. 18 muestra un perceptrón de dos capas. La adición de capas ocultas proporciona a este tipo de arquitectura su…ciente ‡exibilidad para resolver muchos problemas en los que los perceptrones simples fallan.

Para cada una de las salidas y_i, un perceptrón multicapa computa una función $y_i = F_i(x_1; :::; x_n)$ de las entradas. Por ejemplo, la red neuronal mostrada en la Figura 1.10 de la pág. 18 de…ne la función:

$$y_i = f\Big(\sum_k W_{ik} f\Big(\sum_j w_{kj} x_j - \acute{A}_k\Big) - \acute{A}_i\Big): \qquad (1.12)$$

Se ha demostrada que un perceptrón multicapa de dos capas ocultas puede aproximar con un grado de exactitud dado cualquier conjunto de funciones $F_i(x_1; :::; x_n)$. Cuando éstas funciones son contínuas entonces una ú nica capa

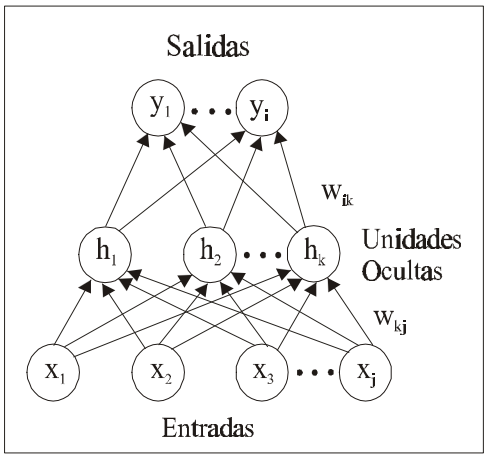


Figura 1.10: Perceptrón de dos capas j: k: i.

oculta es su…ciente. Esto resuelve parcialmente uno de los principales defectos de los perceptrones multi-capa: el diseño de una estructura de red apropiada para un problema dado. Ahora, el problema se reduce a elegir un número apropiado de unidades ocultas para ajustar el modelo, evitando el problema de sobreajuste. La solución a este problema se logra mediante un procedimiento de prueba y error en la mayoría de los casos.

El método de aprendizaje más popular para perceptrones multi-capa es conocido como retro-propagación (backpropagation) y está basado en minimizar la función que da el error cuadrático total usando el método del descenso del gradiente.

1.7.1 El Algoritmo de Retro-propagación

Suponiendo que se tienen unos conjuntos de entradas $fa_{p1}; :::; a_{pn}g$ y sus correspondientes salidas $fb_{p1}; :::; b_{pm}g$; $p = 1; :::; r$ como patrones de entrenamiento. Como en el caso del perceptrón simple se considera la función que da el error cuadrático total:

$$E(w) = \frac{1}{2} \sum_{p;i} \left(b_{pi} - \hat{b}_{pi} \right)^2$$

$$
\begin{aligned}
&= \frac{1}{2}\sum_{i,p}\left[b_{pi} - f\left(\hat{B}_{pi}\right)\right]^2 \\
&= \frac{1}{2}\sum_{i,p}\left[b_{pi} - f\left(\sum_k W_{ik}\hat{h}_k\right)\right]^2 \\
&= \frac{1}{2}\sum_{i,p}\left[b_{pi} - f\left(\sum_k W_{ik}\hat{h}_k\right)\right]^2 \\
&= \frac{1}{2}\sum_{i,p}\left[b_{pi} - f\left(\sum_k W_{ik} f\left(\hat{H}_k\right)\right)\right]^2 \\
&= \frac{1}{2}\sum_{i,p}\left[b_{pi} - f\left(\sum_k W_{ik} f\left(\sum_j w_{kj} a_{pj}\right)\right)\right]^2
\end{aligned}
\tag{1.13}
$$

Este algoritmo está basado en la misma idea del algoritmo del descenso de gradiente usando el método de la regla delta. Entonces se deben combinar los pesos de forma que se descienda por la pendiente de la función de error:

$$
\Delta W_{ik} = -\eta \frac{\partial E}{\partial W_{ik}}; \quad \Delta W_{ik} = -\eta \frac{\partial E}{\partial W_{kj}}; \tag{1.14}
$$

donde el término η es el parámetro de aprendizaje, relacionado con el índice del peso cambiante.

Como la Fórmula 1.13 de la pág. 19 involucra los pesos de las neuronas ocultas y las de salida, el problema de actualizar iterativamente los pesos no es tan simple como en el caso de los perceptrones de una capa. Una solución a este problema fue dada por el algoritmo de retro-propagación de dos pasos. Como primera medida, la entrada del patrón ap se propaga hacia delante obteniendo el valor de las unidades ocultas , y la salida , y, por tanto, el error asociado. Los valores obtenidos se usan luego para actualizar los pesos W_{ik} de la capa de salida. Luego, los pesos obtenidos se utilizarán para actualizar los pesos de la capa oculta, w_{kj}, usando la Fórmula 1.13 de la pág. 19 para propagar hacia atrás los errores anteriores.

Considerando en primer lugar la capa de salida, se usa la regla de la cadena como sigue:

$$\Delta W_{ik} = -\eta \frac{\partial E}{\partial W_{ik}} = -\eta \frac{\partial E}{\partial \hat{o}_{pi}} \frac{\partial \hat{o}_{pi}}{\partial \hat{\beta}_{pi}} \frac{\partial \hat{\beta}_{pi}}{\partial W_{ik}}; \tag{1.15}$$

donde $\hat{o}_{pi} = f'(\hat{\beta}_{pi})$ es la i-ésima salida de la red obtenida por propagación hacia adelante de la entrada $(a_{p1}, \ldots, a_{pn})$. Luego

$$\begin{aligned} \frac{\partial E}{\partial \hat{o}_{pi}} &= -(b_{pi} - \hat{o}_{pi}); \\ \frac{\partial \hat{o}_{pi}}{\partial \hat{\beta}_{pi}} &= f'(\hat{\beta}_{pi}); \\ \frac{\partial \hat{\beta}_{pi}}{\partial W_{ik}} &= \hat{h}_k. \end{aligned}$$

Entonces se obtiene la siguiente regla de actualización:

$$\Delta W_{ik} = \eta \hat{h}_k \delta_{pi} \tag{1.16}$$

donde

$$\delta_{pi} = (b_{pi} - \hat{o}_{pi}) f'(\hat{\beta}_{pi})$$

Una vez que los pesos de entradas han sido actualizados, el valor resultante junto con los valores de las entradas, los de las neuronas ocultas y los de la salida se usan para modificar los pesos de la capa oculta:

$$\Delta w_{ik} = -\eta \frac{\partial E}{\partial w_{ik}} = -\eta \frac{\partial E}{\partial \hat{o}_{pi}} \frac{\partial \hat{o}_{pi}}{\partial \hat{\beta}_{pi}} \frac{\partial \hat{\beta}_{pi}}{\partial \hat{h}_k} \frac{\partial \hat{h}_k}{\partial \hat{H}_k} \frac{\partial \hat{H}_k}{\partial w_{kj}}; \tag{1.17}$$

Los dos primeros términos han sido obtenidos anteriormente. Para el resto se tiene:

$$\frac{\partial \hat{\beta}_{pi}}{\partial \hat{h}_k} = W_{ik};$$

$$\frac{@\hat{n}_k}{@\hat{H}_k} = f^{0}\left(\hat{H}_k\right);$$
$$\frac{@\hat{n}_k}{@w_{kj}} = a_{pj};$$

$$¢w_{ik} = ¡\acute{}a_{pj}\pm_{pi}\tilde{A}_{pi} \tag{1.18}$$

donde

$$\tilde{A}_{pi} = \sum_k \pm_{pi}W_{ki}f^{0}(H_k):$$

1.7.2 Algoritmo de retro-propagación (propagación hacia atrás)

En el siguiente algoritmo se consideran funciones de activación sigmoidales, con el propósito de evitar derivadas formales:

1. Iniciar los pesos con valores aleatorios.
2. Elegir un patrón de entrenamiento y propagarlo hacia delante obteniendo los valores $\hat{n}_p$ y $\hat{o}_{pi}$ para las neuronas de las capas ocultas y de salida.
3. Calcular el error asociado a las unidades de salida:
$$\pm_{pi} = b_{pi} ¡ \hat{o}_{pi}\acute{}f\left(\hat{B}_{pi}\right) = b_{pi} ¡ \hat{o}_{pi}\acute{}\hat{o}_{pi}\left(1 ¡ \hat{o}_{pi}\right):$$
4. Calcular el error asociado a las unidades ocultas:
$$\tilde{A}_{pi} = \sum_k \pm_{pi}W_{ki}f^{0}(H_k) = \sum_k \pm_{pi}W_{ki}\hat{n}_k\left(1 ¡ \hat{n}_k\right):$$
5. Calcular:
$$¢W_{ik} = \acute{}\hat{n}_k\pm_{pi};$$
y
$$¢w_{ik} = ¡\acute{}a_{pj}\pm_{pi}\tilde{A}_{pi};$$
y actualizar los pesos con los valores obtenidos.
6. Repetir los pasos anteriores para cada uno de los patrones de entrenamiento.

1.8 Otras arquitecturas de redes

1.8.1 La Red Neuronal de Hop…eld

Esta arquitectura (ver Figura 1.6 de la pág. 11) está formada por una sola capa que se utiliza como memoria autoasociativa, para almacenar y recuperar información. Utiliza un método de aprendizaje no supervisado que obtine la matriz de pesos que hace que dado cada uno de los patrones de entrenamiento (almacenamiento) la red devuelva el mismo patrón (recuperación). Luego, cuando se tiene una con…guración arbitraria de las neuronas como entradas, la red devolverá aquel patrón almacenado que esté más cerca de la con…guración de partida, en términos de distancia de Hamming. Así, dada una versión incompleta de la información almacenada, la red es capaz de recuperar la información original.

1.8.2 Redes Neuronales Competitivas

Las redes competitivas son un tipo de arquitectura de red populares muy utilizadas para detectar automáticamente grupos o categorías dentro de datos disponibles. Una red neuronal competitiva simple está formada por una capa de entrada y por una de salida, conectadas mediante conexiones de alimentación hacia delante (ver Figura 1.11 de la pág. 22).

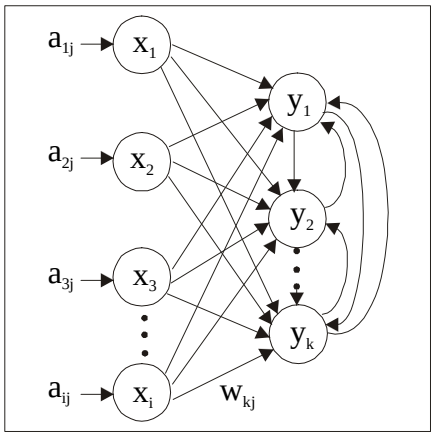


Figura 1.11: Una red neuronal competitiva simple.

Cada patrón de entrada representa un punto en el espacio de con…guración (el espacio de entradas) donde se quieren obtener clases. Para ello, las capas de salida contienen tantas neuronas como clases, o categorías, se quieran obtener. Las dinámicas competitivas entre las clases se crean incluyendo conexiones laterales entre las neuronas de las capas de salida, es decir, cada neurona de salida se conecta consigo misma mediante un pesos positivo (conexión exitatoria) y con las otras por conexiones negativas (conexión inhibitoria).

Este tipo de arquitectura se entrena normalmente con un algoritmo consistente en seleccionar la ganadora, por lo que sólo son actualizados los pesos asociados a la neurona de mayor salida (la ganadora) para un patrón dado.

Estas arquitecturas no son implementadas en el software de simulación desarrollado, por lo cual no se abunda en detalles sobre las mismas.

Capítulo 2

Java

2.1 Introducción a Java

Java surgió en 1991 cuando un grupo de ingenieros de Sun Microsystems trataron de diseñar un nuevo lenguaje de programación destinado a electrodomésticos. La reducida potencia de cálculo y memoria de los electrodomésticos llevó a desarrollar un lenguaje sencillo capaz de generar código de tamaño muy reducido.

Debido a la existencia de distintos tipos de CPUs y a los continuos cambios, era importante conseguir una herramienta independiente del tipo de CPU utilizada. Desarrollan un código "neutro" que no depende del tipo de electrodoméstico, el cual se ejecuta sobre una "máquina hipotética o virtual" denominada Java Virtual Machine (JVM). Es la JVM quien interpreta el código neutro convirtiéndolo a código particular de la CPU utilizada. Esto permitía lo que luego se ha convertido en el principal lema del lenguaje: "Write Once, Run Everywhere" ("Escribir una vez, ejecutar siempre").

A pesar de los esfuerzos realizados por sus creadores, ninguna empresa de electrodomésticos se interesó por el nuevo lenguaje.

Java, como lenguaje de programación para computadoras, se introdujo a …nales de 1995. La clave fue la incorporación de un intérprete Java en el programa Netscape Navigator, versión 2.0, produciendo una verdadera revolución en Internet. Java 1.1 apareció a principios de 1997, mejorando sustancialmente la primera versión del lenguaje.

Al programar en Java no se parte de cero. Cualquier aplicación que se desarrolle se apoya en un gran número de clases preexistentes. Algunas de ellas las ha podido hacer el propio usuario, otras pueden ser comerciales, pero siempre hay un número muy importante de clases que forman parte del propio lenguaje (el API o Application Programming Interface de Java). Java incorpora muchos aspectos que en cualquier otro lenguaje son extensiones propiedad de empresas de software o fabricantes de ordenadores (threads, ejecución remota, componentes, seguridad, acceso a bases de datos, etc.). Por eso es un lenguaje ideal para aprender la informática moderna, porque incorpora todos estos conceptos de un modo estándar, mucho más sencillo y claro que con las citadas extensiones de otros lenguajes. Esto es consecuencia de haber sido diseñado más recientemente y por un único equipo.

Java es un lenguaje muy completo. En cierta forma casi todo depende de casi todo. Por ello, hay que aprenderlo de modo iterativo: primero una visión muy general, que se va re...nando en sucesivas iteraciones. Una forma de hacerlo es empezar con un ejemplo completo en el que ya aparecen algunas de las características más importantes.

La compañía Sun [12] describe el lenguaje Java como “simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico”. Además de una serie de halagos por parte de Sun hacia su propia criatura, el hecho es que todo ello describe bastante bien el lenguaje Java.

2.1.1 Qué es Java 2

Java 2 (antes llamado Java 1.2 o JDK 1.2) es la tercera versión importante del lenguaje de programación Java.

No hay cambios conceptuales importantes respecto a Java 1.1 (en Java 1.1 sí los hubo respecto a Java 1.0), sino extensiones y ampliaciones, lo cual hace que a muchos efectos sea casi lo mismo trabajar con Java 1.1 o con Java 1.2.

Los programas desarrollados en Java presentan diversas ventajas frente a los desarrollados en otros lenguajes como C/C++. La ejecución de programas en Java tiene muchas posibilidades: ejecución como aplicación independiente (Stand-alone Application), ejecución como applet, ejecución como servlet, etc..

Un applet es una aplicación especial que se ejecuta dentro de un navegador

o browser (por ejemplo Netscape Navigator o Internet Explorer) al cargar una página HTML desde un servidor Web. El applet se descarga desde el servidor y no requiere instalación en el ordenador donde se encuentra el browser.

Un servlet es una aplicación sin interface grá…ca que se ejecuta en un servidor de Internet. La ejecución como aplicación independiente es análoga a los programas desarrollados con otros lenguajes.

Además de incorporar la ejecución como Applet, Java permite fácilmente el desarrollo tanto de arquitecturas cliente-servidor como de aplicaciones distribuidas, consistentes en crear aplicaciones capaces de conectarse a otros ordenadores y ejecutar tareas en varios ordenadores simultáneamente, repartiendo por lo tanto el trabajo. Aunque también otros lenguajes de programación permiten crear aplicaciones de este tipo, Java incorpora en su propio API estas funcionalidades.

2.2 El Entorno de Desarrollo Java

Existen distintos programas comerciales que permiten desarrollar código Java. La compañía Sun, creadora de Java, distribuye gratuitamente el Java(tm) Development Kit (JDK). Se trata de un conjunto de programas y librerías que permiten desarrollar, compilar y ejecutar programas en Java.

Incorpora además la posibilidad de ejecutar parcialmente el programa, deteniendo la ejecución en el punto deseado y estudiando en cada momento el valor de cada una de las variables (es el denominado Debugger). Cualquier programador con un mínimo de experiencia sabe que una parte muy importante (muchas veces la mayor parte) del tiempo destinado a la elaboración de un programa se destina a la detección y corrección de errores. Existe también una versión reducida del JDK, denominada JRE (Java Runtime Environment) destinada únicamente a ejecutar código Java (no permite compilar).

Los IDEs (Integrated Development Environment), tal y como su nombre indica, son entornos de desarrollo integrados. En un mismo programa es posible escribir el código Java, compilarlo y ejecutarlo sin tener que cambiar de aplicación. Algunos incluyen una herramienta para realizar Debug grá…camente, frente a la versión que incorpora el JDK basada en la utilización de una consola (denominada habitualmente ventana de comandos de MS-DOS, en Windows NT/95/98) bastante difícil y pesada de utilizar. Estos entornos

integrados permiten desarrollar las aplicaciones de forma mucho más rápida, incorporando en muchos casos librerías con componentes ya desarrollados, los cuales se incorporan al proyecto o programa. Como inconvenientes se pueden señalar algunos fallos de compatibilidad entre plataformas y …cheros resultantes de mayor tamaño que los basados en clases estándar.

2.2.1 El compilador de Java

Se trata de una de las herramientas de desarrollo incluidas en el JDK. Realiza un análisis de sintaxis del código escrito en los …cheros fuente de Java (con extensión *.java). Si no encuentra errores en el código genera los …cheros compilados (con extensión *.class). En otro caso muestra la línea o líneas erróneas. En el JDK de Sun dicho compilador se llama javac.exe. Tiene numerosas opciones, algunas de las cuales varían de una versión a otra. Se aconseja consultar la documentación de la versión del JDK utilizada para obtener una información detallada de las distintas posibilidades.

2.2.2 La Java Virtual Machine

La existencia de distintos tipos de procesadores y ordenadores llevó a los ingenieros de Sun a la conclusión de que era muy importante conseguir un software que no dependiera del tipo de procesador utilizado. Se plantea la necesidad de conseguir un código capaz de ejecutarse en cualquier tipo de máquina. Una vez compilado no debería ser necesaria ninguna modi…cación por el hecho de cambiar de procesador o de ejecutarlo en otra máquina. La clave consistió en desarrollar un código “neutro” el cual estuviera preparado para ser ejecutado sobre una “máquina hipotética o virtual”, denominada Java Virtual Machine (JVM). Es esta JVM quien interpreta este código neutro convirtiéndolo a código particular de la CPU o chip utilizada. Se evita tener que realizar un programa diferente para cada CPU o plataforma.

La JVM es el intérprete de Java. Ejecuta los “bytecodes” (…cheros compilados con extensión .class) creados por el compilador de Java (javac.exe). Tiene numerosas opciones entre las que destaca la posibilidad de utilizar el denominado JIT (Just-In-Time Compiler), que puede mejorar entre 10 y 20 veces la velocidad de ejecución de un programa.

2.2.3 Las variables PATH y CLASSPATH

El desarrollo y ejecución de aplicaciones en Java exige que las herramientas para compilar (javac.exe) y ejecutar (java.exe) se encuentren accesibles. El ordenador, desde una ventana de comandos de MS-DOS, sólo es capaz de ejecutar los programas que se encuentran en los directorios indicados en la variable PATH del ordenador. Si se desea compilar o ejecutar código en Java en estos casos el directorio donde se encuentran estos programas (java.exe y javac.exe) deberán encontrarse en el PATH. Tecleando set PATH en una ventana de comandos de MS-DOS se muestran los nombres de directorios incluidos en dicha variable de entorno. Java utiliza además una nueva variable de entorno denominada CLASSPATH, la cual determina dónde buscar tanto las clases o librerías de Java (el API de Java) como otras clases de usuario. A partir de la versión 1.1.4 del JDK no es necesario indicar esta variable, salvo que se desee añadir conjuntos de clases de usuario que no vengan con dicho JDK. La variable CLASSPATH puede incluir la ruta de directorios o …cheros *.zip o *.jar en los que se encuentren los …cheros *.class. En el caso de los …cheros *.zip hay que indicar que los …cheros en él incluidos no deben estar comprimidos. En el caso de archivos *.jar existe una herramienta (jar.exe), incorporada en el JDK, que permite generar estos …cheros a partir de los archivos compilados *.class. Los …cheros *.jar son archivos comprimidos y por lo tanto ocupan menos espacio que los archivos *.class por separado o que el …chero *.zip equivalente.

2.3 Nomenclatura Habitual en la Programación en Java

Los nombres de Java son sensibles a las letras mayúsculas y minúsculas. Así, las variables masa, Masa y MASA son consideradas variables completamente diferentes. Las reglas del lenguaje respecto a los nombres de variables son muy amplias y permiten mucha libertad al programador, pero es habitual seguir ciertas normas que facilitan la lectura y el mantenimiento de los programas de ordenador. Se recomienda seguir las siguientes instrucciones:

1. En Java es habitual utilizar nombres con minúsculas, con las excepciones que se indican en los puntos siguientes.

2. Cuando un nombre consta de varias palabras es habitual poner una a continuación de otra, poniendo con mayúscula la primera letra de la palabra que sigue a otra.

   Ejemplos:
   CerrarVentana
   ObjetoEntrenable
   GradienteDescendente
   addWindowListener( )

3. Los nombres de clases e interfaces comienzan siempre por mayúscula (ver Tabla 2.1 de la pág. 30).

Clases	Interfaces
Simu	ItemListener
Red	ActionListener
MLP	MouseListener

Tabla 2.1: Ejemplo de clases e interfaces.

4. Los nombres de objetos, los nombres de métodos y variables miembro, y los nombres de las variables locales de los métodos, comienzan siempre por minúscula (ver Tabla 2.2 de la pág. 30).

Objetos	Métodos	Variables
barra	main()	errorDeseado
campoNeuronasCapas	itemStateChanged()	numeroCapas
campoCapas	...jarPesosDesdeRed()	neuronasPorCapa

Tabla 2.2: Ejemplo de objetos, métodos y variables.

5. Los nombres de las variables ...nales, es decir de las constantes, se de...nen siempre con mayúsculas.

   Ejemplo:
   PI

2.4 Estructura General de un Programa Java

En el siguiente ejemplo se presenta la estructura habitual de un programa realizado en cualquier lenguaje orientado a objetos u OOP (Object Oriented Programming), y en particular en el lenguaje Java.

```
import java.awt.*;
import java.lang.String;
import java.lang.Integer;
import java.awt.event.WindowEvent;
import java.util.*;
import java.awt.TextField;
public class Simu extends Frame implements ActionListener,ItemListener{
      MenuBar barra;
      m1 =new Menu("Archivo");
      barra.add(m1);
      m2 =new Menu("Ver");
      barra.add(m2);
....
public static void main(String argv [ ]){
            Simu menus = new Simu();
            menus.setTitle("Simulación de Redes Neuronales");
            menus.setVisible(true);
 }
}
```

Aparece una clase que contiene el programa principal Simu (aquel que contiene la función main()) y algunas clases de usuario (las especí…cas de

la aplicación que se está desarrollando) que son utilizadas por el programa principal. La aplicación se ejecuta por medio del nombre de la lase que contiene la función main(). Las clases de Java se agrupan en packages, que son librerías de clases. Si las clases no se de…nen como pertenecientes a un package, se utiliza un package por defecto (default) que es el directorio activo.

2.5 Conceptos Básicos

2.5.1 Clase

Una clase es una agrupación de datos (variables o campos) y de funciones (métodos) que operan sobre esos datos. A estos datos y funciones pertenecientes a una clase se les denomina variables y métodos o funciones miembro. La programación orientada a objetos se basa en la programación de clases [2]. Un programa se construye a partir de un conjunto de clases.

Una vez de…nida e implementada una clase, es posible declarar elementos de esta clase de modo similar a como se declaran las variables del lenguaje (int, double, String). Los elementos declarados de una clase se denominan objetos de la clase. De una única clase se pueden declarar o crear numerosos objetos. La clase es lo genérico: es el patrón o modelo para crear objetos. Cada objeto tiene sus propias copias de las variables miembro, con sus propios valores, en general distintos de los demás objetos de la clase. Las clases pueden tener variables static, que son propias de la clase y no de cada objeto.

Ejemplo:

```
public abstract class FuncionActivacion implements Cloneable,Serializable{
    /*constructor sin argumentos que permite la herencia */
    public FuncionActivacion () {
    }
}
```

2.5.2 Herencia

La herencia permite que se puedan de…nir nuevas clases basadas en clases existentes, lo cual facilita reutilizar código previamente desarrollado. Si una clase deriva de otra (extends) hereda todas sus variables y métodos. La clase derivada puede añadir nuevas variables y métodos y/o rede…nir las variables y métodos heredados.

En Java, a diferencia de otros lenguajes orientados a objetos, una clase sólo puede derivar de una única clase, con lo cual no es posible realizar herencia múltiple en base a clases. Sin embargo es posible "simular" la herencia múltiple en base a las interfaces.

2.5.3 Interface

Una interface es un conjunto de declaraciones de funciones. Si una clase implementa (implements) una interface, debe de…nir todas las funciones especi…cadas por la interface. Una clase puede implementar más de una interface, representando una forma alternativa de la herencia múltiple.

Una interface puede derivar de otra o incluso de varias interfaces, en cuyo caso incorpora todos los métodos de las interfaces de las que deriva.

Ejemplo: La clase TangenteHiperbolica se extiende de la clase FuncionActivacion que implementa la interface Serializable.

```
/*función de activación tangente hiperbólica */

public class TangenteHiperbolica extends FuncionActivacion implements Serializable{

        /*constructor sin argumentos */

        public TangenteHiperbolica () {

        }

}
```

2.5.4 Package

Un package es una agrupación de clases. Existen una serie de packages incluidos en el lenguaje.

Además el programador puede crear sus propios packages. Todas las clases que formen parte de un package deben estar en el mismo directorio.

Los packages se utilizan con las siguientes …nalidades:

1. Para agrupar clases relacionadas.
2. Para evitar con‡ictos de nombres. En caso de con‡icto de nombres entre clases importadas, el compilador obliga a cuali…car en el código los nombres de dichas clases con el nombre del package.
3. Para ayudar en el control de la accesibilidad de clases y miembros.

Por las razones citadas, durante la etapa de Diseño del Software desarrollado, se ha decido crear dos paquetes, calculos e interfase, utilizando la sentencia package.

```
package myprojects.simu;

import myprojects.calculos.*;

import myprojects.interfase.*;
```

2.5.5 La jerarquía de clases de Java (API)

Durante la generación de código en Java, es recomendable y casi necesario tener siempre a la vista la documentación on-line del API de Java 1.1 ó Java 1.2. En dicha documentación es posible ver tanto la jerarquía de clases, es decir la relación de herencia entre clases, como la información de los distintos packages que componen las librerías base de Java.

Es importante distinguir entre lo que signi…ca herencia y package. Un package es una agrupación arbitraria de clases, una forma de organizar las clases. La sin embargo consiste en crear nuevas clases en base a otras ya existentes. Las clases incluidas en un package no derivan en general de la misma clase.

En la documentación on-line se presentan ambas visiones: “Package Index” y “Class Hierarchy”. La primera presenta la estructura del API de Java agrupada por packages, mientras que en la segunda aparece la jerarquía de clases. Hay que resaltar el hecho de que todas las clases en Java son derivadas

de la clase java.lang.Object, por lo que heredan todos los métodos y variables de ésta.

Si se selecciona una clase en particular, la documentación muestra una descripción detallada de todos los métodos y variables de la clase. A su vez muestra su herencia completa (partiendo de la clase java.lang.Object).

2.6 Programación en Java

En esta sección se presentan las características generales de Java como lenguaje de programación algorítmico. En este apartado Java es muy similar a C/C++, lenguajes en los que está inspirado.

2.6.1 Variables

Una variable es un nombre que contiene un valor que puede cambiar a lo largo del programa. De acuerdo con el tipo de información que contienen, en Java hay dos tipos principales de variables:

1. Variables de tipos primitivos. Están de…nidas mediante un valor único.
2. Variables referencia. Las variables referencia son referencias o nombres de una información más compleja: arrays u objetos de una determinada clase.

Desde el punto de vista de su papel en el programa, las variables pueden ser:

1. Variables miembro de una clase: Se de…nen en una clase, fuera de cualquier método; pueden ser tipos primitivos o referencias.
2. Variables locales: Se de…nen dentro de un método o más en general dentro de cualquier bloque entre llaves {}. Se crean en el interior del bloque y se destruyen al …nalizar dicho bloque. Pueden ser también tipos primitivos o referencias.

2.6.2 Nombres de Variables

Los nombres de variables en Java se pueden crear con mucha libertad. Pueden ser cualquier conjunto de caracteres numéricos y alfanuméricos, sin algunos caracteres especiales utilizados por Java como operadores o separadores (, . + - * / etc.).

Existe una serie de palabras reservadas (ver Tabla 2.3 de la pág. 36), las cuales tienen un signi…cado especial para Java y por lo tanto no se pueden utilizar como nombres de variables. Las palabras (*) son palabras reservadas, pero no se utilizan en la actual implementación del lenguaje Java.

abstract	boolean	break	byte	case	catch
char	class	const*	continue	default	do
double	else	extends	…nal	…nally	‡oat
for	goto*	if	implements	import	instanceof
int	interface	long	native	new	null
package	private	protected	public	return	short
static	super	switch	synchronized	this	throw
throws	transient	try	void	volatile	while

Tabla 2.3: Palabras reservadas para Java

2.6.3 Tipos Primitivos de Variables

Se llaman tipos primitivos de variables de Java a aquellas variables sencillas que contienen los tipos de información más habituales: valores boolean, caracteres y valores numéricos enteros o de punto ‡otante.

Java dispone de ocho tipos primitivos de variables: un tipo para almacenar valores true y false (boolean); un tipo para almacenar caracteres (char), y 6 tipos para guardar valores numéricos, cuatro tipos para enteros (byte, short, int y long) y dos para valores reales de punto ‡otante (‡oat y double).

Los tipos primitivos de Java tienen algunas características importantes que se resumen a continuación:

1. El tipo boolean no es un valor numérico: sólo admite los valores true

o false. El tipo bolean no se identi…ca con el igual o distinto de cero, como en C/C++. El resultado de la expresión lógica que aparece como condición en un bucle o en una bifurcación debe ser boolean.

2. El tipo char contiene caracteres en código UNICODE (que incluye el código ASCII), y ocupan 16 bits por carácter. Comprende los caracteres de prácticamente todos los idiomas.

3. Los tipos byte, short, int y long son números enteros que pueden ser positivos o negativos, con distintos valores máximos y mínimos. A diferencia de C/C++, en Java no hay enteros unsigned.

4. Los tipos ‡oat y double son valores de punto ‡otante (números reales) con 6-7 y 15 cifras decimales equivalentes, respectivamente.

5. Se utiliza la palabra void para indicar la ausencia de un tipo de variable determinado.

6. A diferencia de C/C++, los tipos de variables en Java están perfectamente de…nidos en todas y cada una de las posibles plataformas. Por ejemplo, un int ocupa siempre la misma memoria y tiene el mismo rango de valores, en cualquier tipo de ordenador.

7. Extensiones de Java 1.2 para aprovechar la arquitectura de los procesadores Intel, que permiten realizar operaciones con una precisión extendida de 80 bits.

2.6.4 Cómo se de…nen e inicializan las variables

Una variable se de…ne especi…cando el tipo y el nombre de la variable. Estas variables pueden ser tanto de tipos primitivos como referencias a objetos de alguna clase perteneciente al API de Java o generada por el usuario. Las variable primitivas se inicializan a cero (salvo boolean y char, que se inicializan a false y ‘n0’) si no se especi…ca un valor en su declaración. Análogamente las variables de tipo referencia son inicializadas por defecto a un valor especial: null.

Es importante distinguir entre la referencia a un objeto y el objeto mismo. Una referencia es una variable que indica dónde está en la memoria del ordenador un objeto. Al declarar una referencia todavía no se encuentra “apuntando” a ningún objeto en particular (salvo que se cree explícitamente

un nuevo objeto en la declaración) luego se le asigna el valor nul. Si se desea que esta referencia apunte a un nuevo objeto es necesario utilizar el operador new. Este operador reserva en la memoria del ordenador espacio para ese objeto (variables y funciones). También es posible igualar la referencia declarada a un objeto existente previamente.

Ejemplos:

```
//número de neuronas de entrada de la red
int numeroNeuronasEntrada;
//número de neuronas de salidas de la red
int numeroNeuronasSalida;
//patrones del aprendizaje que seran leídos desde ...chero
public double[ ][ ] patrones;
//fecha de inicio del aprendizaje
public Date horaInicial;
//fecha ...nal del aprendizaje
public Date horaFinal;
// nombre actual del ...chero desde donde se leen los elementos de la red
public String nombreFicheroActual;
// string con la función de activación seleccionada
public String stringFuncion;
```

2.6.5 Visibilidad y vida de las variables

Se entiende por visibilidad, ámbito o scope de una variable, la parte de la aplicación donde dicha variable es accesible y por lo tanto puede ser utilizada en cualquier expresión. En Java todos las variables deben estar incluidas en una clase. En general las variables declaradas dentro de unas llaves {}, es decir dentro de un bloque, son visibles y existen dentro de estas llaves. Por ejemplo las variables declaradas al principio de una función existen mientras

se ejecute la función; las variables declaradas dentro de un bloque if no serán válidas al …nalizar las sentencias correspondientes a dicho if y las variables miembro de una clase (es decir declaradas entre las llaves {} de la clase pero fuera de cualquier método) son válidas mientras existe el objeto de la clase.

Las variables miembro de una clase declaradas como public son accesibles a través de una referencia a un objeto de dicha clase utilizando el operador punto (.). Las variables miembro declaradas como private no son accesibles directamente desde otras clases. Las funciones miembro de una clase tienen acceso directo a todas las variables miembro de la clase sin necesidad de anteponer el nombre de un objeto de la clase. Sin embargo las funciones miembro de una clase B derivada de otra A, tienen acceso a todas las variables miembro de A declaradas como public o protected, pero no a las declaradas como private. Una clase derivada sólo puede acceder directamente a las variables y funciones miembro de su clase base declaradas como public o protected. Otra característica del lenguaje es que es posible declarar una variable dentro de un bloque con el mismo nombre que una variable miembro, pero no con el nombre de otra variable local. La variable declarada dentro del bloque oculta a la variable miembro en ese bloque. Para acceder a la variable miembro oculta será preciso utilizar el operador this.

Uno de los aspectos más importantes en la programación orientada a objetos (OOP) es la forma en la cual son creados y eliminados los objetos. La forma de crear nuevos objetos es utilizar el operador new. Cuando se utiliza el operador new, la variable de tipo referencia guarda la posición de memoria donde está almacenado este nuevo objeto. Para cada objeto se lleva cuenta de por cuántas variables de tipo referencia es apuntado. La eliminación de los objetos la realiza el denominado garbage collector, quien automáticamente libera o borra la memoria ocupada por un objeto cuando no existe ninguna referencia apuntando a ese objeto. Lo anterior signi…ca que aunque una variable de tipo referencia deje de existir, el objeto al cual apunta no es eliminado si hay otras referencias apuntando a ese mismo objeto.

2.7 Operadores de Java

Java es un lenguaje rico en operadores, que son casi idénticos a los de C/C++. Estos operadores se describen brevemente a continuación.

2.7.1 Operadores aritméticos

Son operadores binarios (requieren siempre dos operandos) que realizan las operaciones aritméticas habituales: suma (+), resta (-), multiplicación (*), división (/) y resto de la división (%).

2.7.2 Operadores de asignación

Los operadores de asignación permiten asignar un valor a una variable. El operador de asignación por excelencia es el operador igual (=). La forma general de las sentencias de asignación con este operador es:

variable = expression;

Java dispone de otros operadores de asignación. Se trata de versiones abreviadas del operador (=) que realizan operaciones "acumulativas" sobre una variable.

La siguiente Tabla 2.4 de la pág. 40, muestra estos operadores y su equivalencia con el uso del operador igual (=).

Operador	Utilización	ExpresiónEquivalente
+ =	op1 + = op2	op1 = op1 + op2
- =	op1 - = op2	op1 = op1 - op2
= *	op1 * = op2	op1 = op1 * op2
= /	op1 / = op2	op1 = op1 / op2
% =	op1% = op2	op1 = op1 % op2

Tabla 2.4: Operadores de asignación.

2.7.3 Operadores unarios

Los operadores más (+) y menos (-) unarios sirven para mantener o cambiar el signo de una variable, constante o expresión numérica. Su uso en Java es el estándar de estos operadores.

2.7.4 Operadores incrementales

Java dispone del operador incremento (++) y decremento (–). El operador (++) incrementa en una unidad la variable a la que se aplica, mientras que (–) la reduce en una unidad. Estos operadores se pueden utilizar de dos formas:

1. Precediendo a la variable (por ejemplo: ++i). En este caso primero se incrementa la

   variable y luego se utiliza (ya incrementada) en la expresión en la que aparece.

2. Siguiendo a la variable (por ejemplo: i++). En este caso primero se utiliza la variable en la expresión (con el valor anterior) y luego se incrementa.

En muchas ocasiones estos operadores se utilizan para incrementar una variable fuera de una expresión. En este caso ambos operadores son equivalente. Si se utilizan en una expresión más complicada, el resultado de utilizar estos operadores en una u otra de sus formas será diferente. La actualización de contadores en bucles for es una de las aplicaciones más frecuentes de estos operadores.

2.7.5 Operadores relacionales

Los operadores relacionales sirven para realizar comparaciones de igualdad, desigualdad y relación de menor o mayor. El resultado de estos operadores es siempre un valor boolean (true o false) según se cumpla o no la relación considerada. La siguiente Tabla 2.5 de la pág. 42 muestra los operadores relacionales de Java.

Estos operadores se utilizan con mucha frecuencia en las bifurcaciones y en los bucles, que se verán luego.

Ejemplo de Operadores Incrementales y Operadores Relacionales en un método.

```
public void cambiarParesEntrenamiento(double[ ] paresEntrenamiento){

    /* inicialización de sus valores a partir de los valores pasados como
```

Operador	Utilización	El resultado es true
>	op1 > op2	si op1 es mayor que op2
> =	op1 >= op2	si op1 es mayor o igual que op2
<	op1 < op2	si op1 es menor que op 2
< =	op1 <= op2	si op1 es menor o igual que op2
= =	op1 == op2	si op1 y op2 son iguales
! =	op1 != op2	sio p1 y op2 son diferentes

Tabla 2.5: Operadores relacionales.

```
        argumentos */
        for(int i = 0; i< paresEntrenamiento.length; i++)
        {for(int j = 0; j< numeroNeuronasEntrada; j++)
            {entradaEntrenamiento[i][j] = paresEntrenamiento[i][j];
            }
            for(int j = 0; j< numeroSalidas; j++)
            {salidaEntrenamiento[i][j] = paresEntrenamiento[i]
            [j+numeroNeuronasEntrada];
            }
        }
    }
```

2.7.6 Operador de concatenación de cadenas de caracteres (+)

El operador más (+) se utiliza también para concatenar cadenas de caracteres. Por ejemplo, para escribir una cantidad con un rótulo puede utilizarse la sentencia:

```
editor.append("Error Obtenido:" + String.valueOf(imprimoError) + "nn");
editor.append("Iteraciones:"+ String.valueOf(imprimoIteraciones) + "nn");
```

editor.append(“Inicio: ” + horaInicial.toString() + “nn”);

editor.append(“Final: ” + horaFinal.toString() + “nn”);

donde el operador de concatenación se utiliza dos veces para construir la cadena de caracteres que se desea imprimir. Las variables imprimoErrror, imprimoIteraciones, horaInicial, horaFinal son convertidas en cadena de caracteres para poder concatenarlas.

2.7.7 Precedencia de operadores

El orden en que se realizan las operaciones es fundamental para determinar el resultado de una expresión. Por ejemplo, el resultado de x/y*z depende de qué operación (la división o el producto) se realice primero. La Tabla 2.6 de la pág. 43 muestra el orden en que se ejecutan los distintos operadores en una sentencia, de mayor a menor precedencia:

Nombre	Sintáxis
Post…jos	[ ] .(params) expr++ expr-
Unarios	++expr –expr +expr -expr !
De creación	(type) expr
Multiplicativo	* / %
Adición	+ -
Shift	<< >> >>>
Relacional	<> <= >= instanceof
Igualdad	= = ! =
AND	&
Or Excluyente	^
Or Incluyente	j
Logico AND	&&
Logico OR	jj
Condicional	?:
Asignación	= += -= *= /= %= &= ^= j= <<= >>= >>>=

Tabla 2.6: Precedencia de Operadores.

En Java, todos los operadores binarios, excepto los operadores de asignación, se evalúan de izquierda a derecha. Los operadores de asignación se

evalúan de derecha a izquierda, lo que signi…ca que el valor de la izquierda se copia sobre la variable de la derecha.

2.8 Estructuras de Programación

Las estructuras de programación o estructuras de control permiten tomar decisiones y realizar un proceso repetidas veces. Son los denominados bifurcaciones y bucles. En la mayoría de los lenguajes de programación, este tipo de estructuras son comunes en cuanto a concepto, aunque su sintaxis varía de un lenguaje a otro. La sintaxis de Java coincide prácticamente con la utilizada en C/C++, lo que hace que para un programador de C/C++ no suponga ninguna di…cultad adicional.

2.8.1 Sentencias o expresiones

Una expresión es un conjunto variables unidos por operadores. Son órdenes que se le dan al computador para que realice una tarea determinada.

Una sentencia es una expresión que acaba en punto y coma (;). Se permite incluir varias sentencias en una línea, aunque lo habitual es utilizar una línea para cada sentencia. A continuación se muestra un ejemplo de una línea compuesta de tres sentencias:

```
i = 0; j = 5; x = i + j;
```

2.8.2 Comentarios

Existen dos formas diferentes de introducir comentarios entre el código de Java (en realidad son tres, como pronto se verá). Son similares a la forma de realizar comentarios en el lenguaje . Los comentarios son tremendamente útiles para poder entender el código utilizado, facilitando de ese modo futuras revisiones y correcciones. Además permite que cualquier persona distinta al programador original pueda comprender el código escrito de una forma más rápida. Se recomienda acostumbrarse a comentar el código desarrollado. De esta forma se simpli…ca también la tarea de estudio y revisión posteriores.

Java interpreta que todo lo que aparece a la derecha de dos barras “//”

en una línea cualquiera del código es un comentario del programador y no lo tiene en cuenta. El comentario puede empezar al comienzo de la línea o a continuación de una instrucción que debe ser ejecutada. La segunda forma de incluir comentarios consiste en escribir el texto entre los símbolos " /* */ ". Este segundo método es válido para comentar más de una línea de código. Por ejemplo:

```
// Esta línea es un comentario

int a=1; // Comentario a la derecha de una sentencia

// Esta es la forma de comentar más de una línea utilizando

// las dos barras. Requiere incluir dos barras al comienzo de cada línea

/* Esta segunda forma es mucho más cómoda para comentar un número

elevado de líneas ya que sólo requiere modi…car el comienzo y el …nal. */
```

En Java existe además una forma especial de introducir los comentarios (utilizando /***/ más algunos caracteres especiales) que permite generar automáticamente la documentación sobre las clases y packages desarrollados por el programador. Una vez introducidos los comentarios, el programa javadoc.exe (incluido en el JDK) genera de forma automática la información de forma similar a la presentada en la propia documentación del JDK. La sintaxis de estos comentarios y la forma de utilizar el programa javadoc.exe se puede encontrar en la información que viene con el JDK.

2.8.3 Bifurcaciones

Las bifurcaciones permiten ejecutar una de entre varias acciones en función del valor de una expresión lógica o relacional. Se tratan de estructuras muy importantes ya que son las encargadas de controlar el ‡ujo de ejecución de un programa. Se exponen dos variantes del de tipo if.

Bifurcación if

Esta estructura permite ejecutar un conjunto de sentencias en función del valor que tenga la expresión de comparación. Ejemplo: se ejecuta si la expresión de comparación (error < errorMinimo) tiene valor true:

```
protected void comprobarNuevoMinimo() {
  if (error < errorMinimo)
    {errorMinimo = error;
    vectorDisMinimo = (double[ ])(vectorDis.clone());
    } /* …n del if */
}
```

Las llaves {} sirven para agrupar en un bloque las sentencias que se han de ejecutar, y no son necesarias si sólo hay una sentencia dentro del if.

Bifurcación if else

Análoga a la anterior, de la cual es una ampliación. Las sentencias incluidas en el else se ejecutan en el caso de no cumplirse la expresión de comparación (false),

Ejemplo:

```
public double decirSalidaActual(int indiceEtapa) {
    if(pila != null)
        {return pila[indiceEtapa];}
    else
        {System.out.println("Fallo: Pila no creada");
        return 0;
        }
}
```

2.8.4 Bucles

Un bucle se utiliza para realizar un proceso repetidas veces. Se denomina también lazo o loop. El código incluido entre las llaves {} (opcionales si el

proceso repetitivo consta de una sola línea), se ejecutará mientras se cumpla unas determinadas condiciones. Hay que prestar especial atención a los bucles in…nitos, hecho que ocurre cuando la condición de …nalizar el bucle (booleanExpression) no se llega a cumplir nunca. Se trata de un fallo muy típico, habitual sobre todo entre programadores poco experimentados.

Bucle while

En el siguiente ejemplo se muestra que se ejecutará la sentencia …n++ mientras la expresión (capas.charAt(…n)!=',' && capas.charAt(…n)!=-1) sea verdadera.

```
for (int j=0; j < numeroCapas; j++)
  {int …n = principio;
    try {
        while (capas.charAt(…n) != ',' && capas.charAt(…n) != -1)
        {…n++;
        }
    }
  }
```

Bucle for

A continuación se podrá apreciar la utilización del bucle for:

```
/* calcular el nuevo vector de diseño */
    for (int i = 0; i < vectorDis.length; i++)
    {vectorDis[i] = vectorDis[i] + learningRate * S[i];
    }
```

La sentencia int i = 0 (inicialización) se ejecuta al comienzo del for, e i++ (incremento) después de vectorDis[i] = vectorDis[i] + learningRate * S[i] (sentencia). La expresión booleana (vectorDis.length) se evalúa al comienzo de cada iteración; el bucle termina cuando la expresión de comparación toma

el valor false.

Bucle do while

Es similar al bucle while pero con la particularidad de que el control está al …nal del bucle (lo que hace que el bucle se ejecute al menos una vez, independientemente de que la condición se cumpla o no). Una vez ejecutados las sentencias, se evalúa la condición: si resulta true se vuelven a ejecutar las sentencias incluidas en el bucle, mientras que si la condición se evalúa a false …naliza el bucle.

```
do{
   /* calcular el gradiente del vector …jar el vector de diseño */
   problema.…joVector(vectorDis);
   /* incrementar el contador de iteraciones*/
   step++;
   } while (error > errorDeseado && step < iteracionesMaximas);
      /* ... hasta que el error sea menor o igual que el deseado o */
      /* se alcance el número de iteraciones pasado como argumento */
      problema.…joVector(vectorDis);
```

Sentencia return

Una forma de salir de un bucle es utilizar la sentencia return. Esta sentencia sale también de un método o de una función. En el caso de que la función devuelva alguna variable, este valor se deberá poner a continuación del return.

```
public double devuelveErrorMinimo()
{return errorMinimo;
}
```

Bloque try{...} catch{...} ...nally{...}

Java incorpora en el propio lenguaje la gestión de errores. El mejor momento para detectar los errores es durante la compilación. Sin embargo prácticamente sólo los errores de sintaxis son detectados en esta operación. El resto de problemas surgen durante la ejecución de los programas.

En el lenguaje Java, una Exception es un cierto tipo de error o una condición anormal que se ha producido durante la ejecución de un programa. Algunas excepciones son fatales y provocan que se deba ...nalizar la ejecución del programa. En este caso conviene terminar ordenadamente y dar un mensaje explicando el tipo de error que se ha producido. Otras excepciones, como por ejemplo no encontrar un ...chero en el que hay que leer o escribir algo, pueden ser recuperables. En este caso el programa debe dar al usuario la oportunidad de corregir el error (dando por ejemplo un nuevo path del ...chero no encontrado).

Los errores se representan mediante clases derivadas de la clase Throwable, pero los que tiene que chequear un programador derivan de Exception (java.lang.Exception que a su vez deriva de Throwable). Existen algunos tipos de excepciones que Java obliga a tener en cuenta. Esto se hace mediante el uso de bloques try, catch y ...nally.

El código dentro del bloque try está "vigilado": Si se produce una situación anormal y se lanza como consecuencia una excepción, el control pasa al bloque catch que se hace cargo de la situación y decide lo que hay que hacer. Se pueden incluir tantos bloques catch como se desee, cada uno de los cuales tratará un tipo de excepción. Finalmente, si está presente, se ejecuta el bloque ...nally, que es opcional, pero que en caso de existir se ejecuta siempre, sea cual sea el tipo de error.

En el caso en que el código de un método pueda generar una Exception y no se desee incluir en dicho método la gestión del error (es decir los bucles try/catch correspondientes), es necesario que el método pase la Exception al método desde el que ha sido llamado. Esto se consigue mediante la adición de la palabra throws seguida del nombre de la Exception concreta, después de la lista de argumentos del método. A su vez el método superior deberá incluir los bloques try/catch o volver a pasar la Exception. De esta forma se puede ir pasando la Exception de un método a otro hasta llegar al último método del programa, el método main().

2.9 Clases en Java

Las clases son el centro de la Programación Orientada a Objetos (OOP - Object Oriented Programming). Algunos conceptos importantes de la POO son los siguientes:

1. Encapsulación: Las clases pueden ser declaradas como públicas (public) y como package (accesibles sólo para otras clases del package). Las variables miembro y los métodos pueden ser public, private, protected y package. De esta forma se puede controlar el acceso y evitar un uso inadecuado.
2. Herencia:Una clase puede derivar de otra (extends), y en ese caso hereda todas sus variables y métodos. Una clase derivada puede añadir nuevas variables y métodos y/o rede…nir las variables y métodos heredados.
3. Polimor…smo:Los objetos de distintas clases pertenecientes a una misma jerarquía o que implementan una misma interface pueden tratarse de una forma general e individualizada, al mismo tiempo. Esto facilita la programación y el mantenimiento del código.

2.9.1 Características importantes de las clases

A continuación se enumeran algunas características importantes de las clases:

1. Todas las variables y funciones de Java deben pertenecer a una clase. No hay variables y funciones globales.
2. Si una clase deriva de otra (extends), hereda todas sus variables y métodos.
3. Java tiene una jerarquía de clases estándar de la que pueden derivar las clases que crean los usuarios.
4. Una clase sólo puede heredar de una única clase (en Java no hay herencia múltiple). Si al de…nir una clase no se especi…ca de qué clase deriva, por defecto la clase deriva de Object. La clase Object es la base de toda la jerarquía de clases de Java.

5. En un …chero se pueden de…nir varias clases, pero en un …chero no puede haber más que una clase public. Este …chero se debe llamar como la clase public que contiene con extensión *.java. Con algunas excepciones, lo habitual es escribir una sola clase por …chero.

6. Si una clase contenida en un …chero no es public, no es necesario que el …chero se llame como la clase.

7. Los métodos de una clase pueden referirse de modo global al objeto de esa clase al que se aplican por medio de la referencia this.

8. Las clases se pueden agrupar en packages, introduciendo una línea al comienzo del …chero (package packageName;). Esta agrupación en packages está relacionada con la jerarquía de directorios y …cheros en la que se guardan las clases.

2.9.2 Métodos o Funciones Miembros

Métodos de objeto

Los métodos son funciones de…nidas dentro de una clase. Salvo los métodos static o de clase, se aplican siempre a un objeto de la clase por medio del operador punto (.). Dicho objeto es su argumento implícito. Los métodos pueden además tener otros argumentos explícitos que van entre paréntesis, a continuación del nombre del método.

La primera línea de la de…nición de un método se llama declaración o header; el código comprendido entre las llaves {} es el cuerpo o body del método. Considérese el siguiente ejemplo:

```
imprimoError=algor.devuelveErrorMinimo();

public double devuelveErrorMinimo()

{return errorMinimo;

}
```

La clase Object

Como ya se ha dicho, la clase Object es la raíz de toda la jerarquía de clases de Java. Todas las clases de Java derivan de Object.

La clase Object tiene métodos interesantes para cualquier objeto que son heredados por cualquier clase. Entre ellos se pueden citar los siguientes:

1. Métodos que pueden ser rede…nidos por el programador:

   - clone(): Crea un objeto a partir de otro objeto de la misma clase. El método original heredado de Object lanza una CloneNotSupportedException. Si se desea poder clonar una clase hay que implementar la interface Cloneable y rede…nir el método clone(). Este método debe hacer una copia miembro a miembro del objeto original. No debería llamar al operador new ni a los constructores.
   - equals(): Indica si dos objetos son o no iguales. Devuelve true si son iguales, tanto si son referencias al mismo objeto como si son objetos distintos con iguales valores de las variables miembro.
   - toString(): Devuelve un String que contiene una representación del objeto como cadena de caracteres, por ejemplo para imprimirlo o exportarlo.
   - …nalize(): Este método ya se ha visto al hablar de los …nalizadores.

2. Métodos que no pueden ser rede…nidos (son métodos …nal):

   - getClass(): Devuelve un objeto de la clase Class, al cual se le pueden aplicar métodos para determinar el nombre de la clase, su superclase, las interfaces implementadas, etc. Se puede crear un objeto de la misma clase que otro sin saber de qué clase es.
   - notify(), notifyAll() y wait(): Son métodos relacionados con los threads (hilos).

2.10 Algunas Clases de Utilidad

2.10.1 Clase Arrays

Los arrays de Java (vectores, matrices, hiper-matrices de más de dos dimensiones) se tratan como objetos de una clase prede…nida. Los arrays son objetos, pero con algunas características propias.

Los arrays pueden ser asignados a objetos de la clase Object y los métodos de Object pueden ser utilizados con arrays.

Algunas de sus características más importantes de los arrays son las siguientes:

1. Los arrays se crean con el operador new seguido del tipo y número de elementos.
2. Se puede acceder al número de elementos de un array con la variable miembro implícita length (por ejemplo, vect.length).
3. Se accede a los elementos de un array con los corchetes [ ] y un índice que varía de 0 a length-1.
4. Se pueden crear arrays de objetos de cualquier tipo. En principio un array de objetos es un array de referencias que hay que completar llamando al operador new.
5. Los elementos de un array se inicializan al valor por defecto del tipo correspondiente (cero para valores numéricos, la cadena vacía para Strings, false para boolean, null para referencias).
6. Como todos los objetos, los arrays se pasan como argumentos a los métodos por referencia.
7. Se pueden crear arrays anónimos (por ejemplo, crear un nuevo array como argumento actual en la llamada a un método).

Inicialización de arrays

Los arrays se pueden inicializar con valores entre llaves {...} separados por comas. También los arrays de objetos se pueden inicializar con varias llamadas a new dentro de unas llaves {...}.

Si se igualan dos referencias a un array no se copia el array, sino que se tiene un array con dos nombres, apuntando al mismo y único objeto.

Ejemplo de creación de una referencia a un array:

```
/*vector de pesos */

public double[ ] pesos;
```

También existen arrays bidimensionales, que se crean de un modo muy similar al de C++ (con reserva dinámica de memoria). En Java una matriz es un vector de vectores …la, o más en concreto un vector de referencias a los vectores …la. Con este esquema, cada …la podría tener un número de elementos diferente.

Una matriz se puede crear directamente en la siguiente forma:

```
/*vector de las entradas de los pares de entrenamiento*/

protected double[ ][ ] entradaEntrenamiento;
```

2.10.2 Clases String y StringBu¤er

Las clases String y StringBu¤er están orientadas a manejar cadenas de caracteres. La clase String está orientada a manejar cadenas de caracteres constantes, es decir, que no pueden cambiar. La clase StringBu¤er permite que el programador cambie la cadena insertando, borrando, etc. La primera es más e…ciente, mientras que la segunda permite más posibilidades.

Ambas clases pertenecen al package java.lang, y por lo tanto no hay que importarlas. Hay que indicar que el operador de concatenación (+) entre objetos de tipo String utiliza internamente objetos de la clase StringBu¤er y el método append().

Los métodos de String se pueden utilizar directamente sobre literals (cadenas entre comillas), como por ejemplo: “Hola”.length().

Métodos de la clase String

Algunos métodos de String y la función que realizan:

- String(...): Constructores para crear Strings a partir de arrays de bytes o de caracteres.
- String(String str) yString(StringBu¤er sb): Costructores a partir de un objeto String o StringBu¤er.
- charAt(int): Devuelve el carácter en la posición especi…cada.
- getChars(int, int, char[ ], int): Copia los caracteres indicados en la posición indicada de un array de caracteres.
- length(): Devuelve el número de caracteres de la cadena.
- toLowerCase(): Convierte en minúsculas (puede tener en cuenta el locale).
- toUpperCase(): Convierte en mayúsculas (puede tener en cuenta el locale).
- valueOf(): Devuelve la representación como String de sus argumento. Admite Object, arrays de caracteres y los tipos primitivos.

2.10.3 Clase Double

La clase java.lang.Double deriva de Number, que a su vez deriva de Object. Esta clase contiene un valor primitivo de tipo double.

Algunos métodos de la clase Double

- Double(double) y Double(String): Los constructores de esta clase.
- doubleValue(), ‡oatValue(), longValue(), intValue(), shortValue(), byteValue(): Métodos para obtener el valor del tipo primitivo.
- String toString(), Double valueOf(String): Conversores con la clase String.
- isIn…nite(), isNaN(): Métodos de chequear condiciones.
- equals(Object): Compara con otro objeto.

2.10.4 Clase Integer

La clase java.lang.Integer tiene como variable miembro un valor de tipo int.

Algunos métodos de la clase Integer

- Integer(int) y Integer(String): Constructores de la clase.
- doubleValue(), ‡oatValue(), longValue(),intValue(), shortValue(), byteValue(): Conversores con otros tipos primitivos.
- Integer decode(String), Integer parseInt(String), String toString(), Integer ValueOf(String): Conversores con String del sistema a partir del nombre de dicha propiedad.

2.11 El AWT (Abstract Windows Toolkit)

2.11.1 Qué es el AWT

El AWT (Abstract Windows Toolkit) es la parte de Java que se ocupa de construir interfaces grá…cas de usuario. Aunque el AWT ha estado presente en Java desde la versión 1.0, la versión 1.1 representó un cambio notable, sobre todo en lo que respecta al modelo de eventos. La versión 1.2 ha incorporado un modelo distinto de componentes llamado Swing, que también está disponible en la versión 1.1 como package adicional.

2.11.2 Creación de una Interface Grá…ca de Usuario

Para construir una interface grá…ca de usuario hace falta:

1. Un “contenedor” o container, que es la ventana o parte de la ventana donde se situarán los componentes (botones, barras de desplazamiento, etc.) y donde se realizarán los dibujos. Se correspondería con un formulario o una picture box de Visual Basic.

2. Los componentes: menús, botones de comando, barras de desplazamiento, cajas y áreas de texto, botones de opción y selección, etc. Se corresponderían con los controles de Visual Basic.

3. El modelo de eventos. El usuario controla la aplicación actuando sobre los componentes, de ordinario con el ratón o con el teclado. Cada vez que el usuario realiza una determinada acción, se produce el evento correspondiente, que el sistema operativo transmite al AWT.

El AWT crea un objeto de una determinada clase de evento, derivada de AWTEvent. Este evento es transmitido a un determinado método para que lo gestione. En Visual Basic el entorno de desarrollo crea automáticamente el procedimiento que va a gestionar el evento (uniendo el nombre del control con el tipo del evento mediante el carácter _) y el usuario no tiene más que introducir el código. En Java esto es un poco más complicado: el componente u objeto que recibe el evento debe "registrar" o indicar previamente qué objeto se va a hacer cargo de gestionar ese evento.

2.11.3 Objetos "event source" y objetos "event listener"

El modelo de eventos de Java está basado en que los objetos sobre los que se producen los eventos (event sources) "registran" los objetos que habrán de gestionarlos (event listeners), para lo cual los event listeners habrán de disponer de los métodos adecuados. Estos métodos se llamarán automáticamente cuando se produzca el evento. La forma de garantizar que los event listeners disponen de los métodos apropiados para gestionar los eventos es obligarles a implementar una determinada interface Listener. Las interfaces Listener se corresponden con los tipos de eventos que se pueden producir.

Las capacidades grá...cas del AWT resultan pobres y complicadas en comparación con lo que se puede conseguir con otros lenguajes de programación, como Visual Basic, pero tienen la ventaja de poder ser ejecutadas casi en cualquier ordenador y con cualquier sistema operativo.

2.11.4 Proceso a seguir para crear una aplicación interactiva (orientada a eventos)

Pasos que se pueden seguir para construir una aplicación orientada a eventos sencilla, con interface grá…ca de usuario:

1. Determinar los componentes que van a constituir la interface de usuario (botones, cajas de texto, menús, etc.).
2. Crear una clase para la aplicación que contenga la función main().
3. Crear una clase Ventana, sub-clase de Frame, que responda al evento WindowClosing().
4. La función main() deberá crear un objeto de la clase Ventana (en el que se van a introducir las componentes seleccionadas) y mostrarla por pantalla con el tamaño y posición adecuados.
5. Añadir al objeto Ventana todos los componentes y menús que deba contener.
6. De…nir los objetos Listener (objetos que se ocuparán de responder a los eventos, cuyas clases implementan las distintas interfaces Listener) para cada uno de los eventos que deban estar soportados. En aplicaciones pequeñas, el propio objeto Ventana se puede ocupar de responder a los eventos de sus componentes. En programas más grandes se puede crear uno o más objetos de clases especiales para ocuparse de los eventos.
7. Finalmente, se deben implementar los métodos de las interfaces Listener que se vayan a hacer cargo de la gestión de los eventos.

En la aplicación se podrá observar la gran variedad de componentes de AWT que se utilizaron. También se acudió a los paquetes de Swing , por ejemplo para la implementación de un hilo en la presentación de la aplicación.

El objetivo de este capítulo es brindar nociones sobre este lenguaje de programación, mostrando algunos ejemplos que se pueden encontrar en el código fuentede la aplicación.

Capítulo 3

Preparación de documentos con LaTeX

3.1 Introducción

El LaTeX es un sistema de composición profesional ideado especialmente para trabajo con fórmulas matemáticas que da una altísima calidad de edición [6].

LaTeX consiste en un conjunto de macrocomandos construidos a partir de LaTeX, un lenguaje de composición de texto creado por Donald Knuth de la Universidad de Stanford en los años 1970s. El LaTeX original fue creado por Leslie Lamport en Digital Equipment Corporation a mitad de los años 1980s. Una versión más reciente, llamada LaTeX2e, es una parte del proyecto LaTeX3 liderado por Frank Mittelbach. AMS-LaTeX, un conjunto de mejoras de LaTeX realizado por la American Mathematical Society, suministra una mejora de las posibilidades de composición matemática, incluyendo tres nuevas clases de documentos LaTeX2e y cuatro nuevos alfabetos matemáticos.

Los artículos escritos mediante AMS-LaTeX son de la misma calidad que los que se encuentran en las revistas de la AMS.

3.1.1 Forma de trabajo con LATEX

La primera característica del LATEX con la que el usuario ha de enfrentarse es el hecho, a veces ciertamente mal recibido, de que no sea un procesador de texto visual, sino de diseño lógico. Es decir, lo que el usuario escribe en el teclado no es lo que después ve en el documento …nal.

La preparación de un documento con LATEX consta de tres pasos:

- Edición del documento.
- Composición o compilación.
- Impresión.

Por tanto, va a existir un …chero de entrada, que es el que se edita, y un …chero de salida, que es el que se imprime. El proceso de ejecución que convierte el …chero de entrada (sin formatear) en el …chero de salida es la composición o compilación.

Proceso	Programa	Fichero entrada	Fichero resultado
Edición	Editor	.tex	
Composición	TEX	.tex .aux	.dvi .aux .log
Visualización Impresión	DVI Driver	.dvi	

Tabla 3.1: Programas implicados en la preparación de un documento con LaTeX.

La Tabla 3.1 de la pág. 60 esquematiza los programas implicados en la preparación de un documento con LATEX así como las extensiones de los …cheros que intervienen.

La Figura 3.1 de la pág. 61 muestra un esquema de la mayor parte de los diferentes …cheros que intervendrán en el proceso de edición y composición de un documento LATEX.

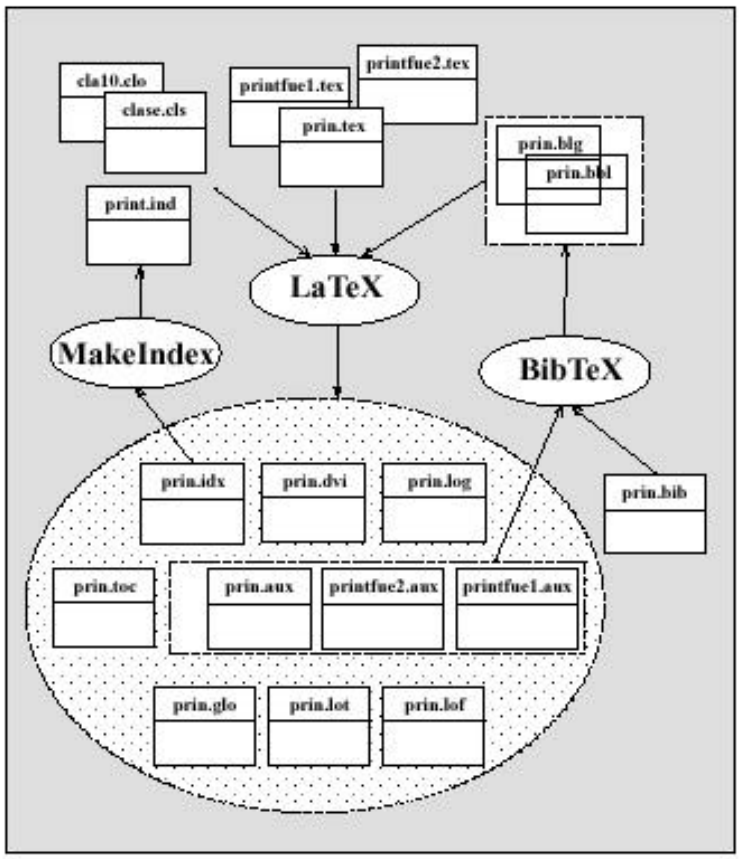


Figura 3.1: Algunos de los …cheros resultantes del proceso de edición y composición de un documento con LaTeX.

A continuación, se describe el papel que juegan en ese proceso los diferentes …cheros de…nidos por su extensión:

- tex: …chero de entrada para LATEX o …chero que se incluye en éste con ninclude.
- bib: …chero preparado por el usuario, conteniendo una base de datos bibliográ…ca.
- cls: contiene las de…niciones estándar propias de una clase de documento; por ejemplo, book.cls.
- clo: asociado a algún .cls, contiene más opciones para la clase; por ejemplo, bk10.clo.
- sty: contiene de…niciones sobre la estructura y el aspecto; generalmente se guarda en un directorio estándar[1]; por ejemplo, babel.sty.
- ist: contiene información sobre el estilo del índice alfabético.
- bst: contiene información sobre el estilo de la bibliografía.
- dvi: contiene la representación del texto formateado, resultado de la composición; guarda únicamente los nombres de las fuentes y no las imágenes reales de los caracteres, por lo que para poder visualizarlo es necesario servirse de un dvi driver.
- log: …chero generado en la composición; guarda información de ella, como puede ser los nombres de los …cheros que han intervenido, las páginas que han sido procesadas, los mensajes de aviso y error, etc.
- aux: …chero auxiliar generado en la composición; guarda información de contadores, etiquetas, etc. que permiten las referencias cruzadas y las citas bibliográ…cas. Hay un …chero .aux para cada .tex que intervenga en la composición.
- toc: …chero generado en la composición si se ha pedido tabla de contenidos; contiene toda la información necesaria para elaborarla, tal como los nombres de las unidades de seccionamiento, la página donde comienzan, etc.

[1] También tienen esta extención los …cheros de clase de la versión LATEX 2.09.

- lof: …chero generado en la composición si se ha pedido lista de …guras; contiene los pies de cada …gura, su numeración, su página, etc.
- lot: igual que el anterior, pero referido a tablas.
- idx: …chero generado en la composición si se ha pedido índice alfabético; contiene una lista de las entradas para el índice.
- ind: generado por el programa MakeIndex a partir del .idx, contiene una lista ordenada y uni…cada de entradas y páginas, para la elaboración del índice alfabético.
- ilg: contiene los mensajes de aviso y error ocurridos en la ejecución de MakeIndex.
- bbl: generado por BibTeX a partir de .bib y .aux, contiene la lista ordenada de las entradas bibliográ…cas requeridas en el texto.
- blg: contiene los mensajes de aviso y error ocurridos en la ejecución de BibTEX.
- tfm: contiene la información (dimensiones, ...) de cada carácter de una fuente.
- fd: …chero de de…nición de fuentes, que guarda la correspondencia entre los nombres de las fuentes internas y de las fuentes externas.
- fmt: especi…ca los patrones de guionado.

Para enviar electrónicamente un trabajo (por correo electrónico o en un disquete), debemos mandar no sólo el …chero principal, sino también acompañar los …cheros que se incluyen en el principal con ninclude, los …cheros .bbl de bibliografía, los …cheros conteniendo paquetes no estándar o especi…caciones propias del usuario.

Una forma de saber cuáles son los …cheros que deben ser enviados, es poner en el preámbulo el comando

```
nlistfiles
```

con el que LATEX lista todos los …cheros utilizados.

3.1.2 Modo de trabajo en LATEX

El contenido de un documento LATEX puede estar en uno de estos tres modos:

- Modo párrafo: modo normal, por defecto. Texto ordinario: considera la entrada como una sucesión de palabras divididas en líneas, párrafos y páginas.
- Modo matemático: fórmulas; para entrar en este modo:

```
$, $$, n[,n(,nbegin{equation},nbegin{displaymath}
```

y para salir, respectivamente

```
$, $$, n], n), nend{equation}, nend{displaymath}
```

- Modo izquierda-derecha (LR): como el modo párrafo pero sin cortar entre líneas. Para entrar en este modo se usa el comando:

```
nmbox{ }
```

El modo LR puede utilizarse para evitar cortes de palabras entre líneas.

El cambio de modo se realiza muy fácilmente y es usual en fórmulas que incluyan texto, para pasar de modo matemático a modo LR El compilador LATEX actúa por defecto interrumpiendo la compilación cuando encuentra un error y preguntando al usuario qué hacer. Con la opción nbatchmode avisa cuando se usa un comando incorrectamente y, si puede, evita la parada.

3.2 Documentos en LATEX

En LATEX hay muchos tipos de documentos y se pueden de…nir muchos más.

Entre los más corrientes destacan los de carta, artículo, libro e informe, que se describen en este capítulo.

La información que recibe el programa LATEX para su tratamiento es un …chero de texto. Este …chero debe tener la extensión .tex y estará compuesto de dos partes: un preámbulo y el documento propiamente dicho.

El documento propiamente dicho comienza con el comando

```
nbegin{document}
```

y termina con el comando

```
nend{document}
```

En medio se encuentra todo el texto y …guras del documento. Todo lo que siga es ignorado por LATEX.

Lo que precede a nbegin{document} es el preámbulo, que sólo contiene declaraciones.

3.2.1 El Preámbulo

El preámbulo contiene aquellas órdenes que afectan a todo el documento y comienza generalmente con ndocumentclass con la especi…cación del estilo del texto.

3.2.2 Especi…cación y Estilo de Documento

El tipo y el estilo del documento se especi…can mediante el comando:

```
n documentclass[opciones] {clase}
```

que especi…ca la clase del documento y sus opciones. La clase …jada, así como todas las especi…caciones que se hagan en el preámbulo, serán comunes a todo el texto. En el argumento clase puede ponerse una de las siguientes clases estándar:

article report book letter slides

También puede usarse como argumento una clase diseñada por el usuario.

El comando ndocumentclass lee el correspondiente …chero clase.cls.

En el argumento opciones, puede ponerse una lista de opciones de estilo, separadas por comas.

El estilo del documento viene determinado mediante el uso de las clases y los paquetes. La mayoría de los comandos de LATEX son válidos con todas las clases, no existiendo gran número de ellos que sean especí…cos de alguna clase en particular.

Con los paquetes se añadirán especi…caciones de estilo a las contenidas en la clase, permitiendo también cargar macros y de…niciones del usuario.

Existen además las opciones sobre clases y paquetes, que modi…can el formato de algunos elementos.

Las de…niciones propias de cada clase se guardan en el …chero de extensión .cls (por ejemplo article.cls); las propias de cada paquete en uno de extensión .sty (por ejemplo amstex.sty). Además, a las opciones de clase les corresponden normalmente los …cheros .clo (por ejemplo art11.clo).

3.2.3 Estilo de Página

Una página del texto de salida tiene tres partes:

- Encabezamiento
- Cuerpo
- Pié

Diseñar una página consiste en determinar la altura y anchura de esas tres partes y los márgenes, el contenido del encabezamiento y del pié, así como el formato del cuerpo del texto.

Para determinar cuál será el contenido del encabezamiento y del pié, se dispone de la declaración

```
\pagestyle{estilo}
```

donde el estilo puede ser:

plain: número de página en el pié y el encabezamiento vacío

empty: pié y encabezamiento vacíos

headings: encabezamiento diseñado en el documento de estilo y pié vacío

myheadings: como headings, pero encabezamiento diseñado por el usuario.

Para especi…car el contenido del encabezamiento cuando se trabaja con headings y myheadings se utilizan los comandos:

```
\markboth{enc. pag. izq}{enc. pag. dcha.}
\markright{enc. pag. dcha.}
```

de manera que para una página

- de la mano izquierda: se toma enc. pag. izq del último \markboth ocurrido hasta el …nal de la página;
- de la mano derecha: se toma enc. pag. dcha del primer \markright o \markboth de la página y si no hay ninguno, del último ocurrido antes del principio de la página.

Con el estilo headings, estos comandos van a especi…car marcas para las unidades de seccionamiento según se recoge en la Figura 3.2 de la pág. 68.

Se dispone también de la declaración

```
\thispagestyle{estilo}
```

en todo análoga a la anterior salvo en que únicamente determina el estilo de una página.

La declaración

Estilo	Comando	Clase de Documento book, report, article
twoside	nmarkboth nmarkright	nchapter nsection nsection nsubsection
oneside	nmarkright	nchapter nsection

Tabla 3.2: Programas implicados en la preparación de un documento con LaTeX.

npagenumbering{estilo numeración}

especi…ca el estilo de numeración de las páginas. Además rede…ne el comando nthepage para que sea

nestilo numeración {page}

donde page es el contador de página. Por defecto la numeración se hace en arabic, pero puede cambiarse fácilmente.

3.2.4 La página del Título

Todo documento suele tener una página en la que …gura su título, el nombre del autor y posiblemente la fecha de publicación.

La información necesaria para construir el título se le da a través de los comandos siguientes del preámbulo:

nauthor{nombres}: los posibles distintos autores se separan por nand; admite nn para incluir más informaciones.

ndate{fecha}: este comando es opcional; el defecto es la fecha actual.

ntitle{título}: admite nn para dividir un título largo.

Además, dentro de los argumentos de estos tres comandos puede incluirse

nthanks{texto}: produce un pié de página, conteniendo posibles agradecimientos, direcciones, etc.

Para generar el título de un documento se cuenta con el comando

nmaketitle

Con la opción de clase notitlepage colocará el título en la parte superior de la nueva página. Con la opción titlepage genera el título en una hoja separada.

El comando nmaketitle va tras el comando nbegin{document}, y los ntitle y nauthor deben ir en el preámbulo, es decir, antes que aquél. Si se omite el comando ndate, LATEX pone la fecha del día en que se imprime, lo cual también puede hacerse de forma explícita utilizando el comando ntoday.

Estos comandos sirven sólo para de…nir los autores, la fecha y el título, pero no, para escribirlos. Como se verá, el título se escribe dentro del documento mediante el comando nmaketitle.

Otra posible herramienta para generar una página de título es el entorno titlepage, que además restablece el número de página a 1.

3.2.5 El Documento Propiamente Dicho

El documento propiamente dicho es la parte del …chero entre los comandos nbegin{document} y nend{document}. Consta de texto y opcionalmente de …guras.

Resumen

Para generar un "abstract" se dispone del entorno abstract, no de…nido en la clase book. Con la opción de clase notitlepage, por defecto en la clase article, el título y el "abstract" se disponen al comienzo de la primera página; la opción titlepage genera dos páginas iniciales distintas con esos contenidos.

Tabla de Contenidos. Lista de Figuras y Tablas

La tabla de contenidos es el listado de las unidades de seccionamiento del documento, con diferentes aspectos (estilo, sangría, . . .) para cada tipo de

unidad y, por supuesto, acompañados del número de página en que comienzan. Se incluye aquí información sobre la generación de listas de …guras y tablas por ser análoga a la de la tabla de contenidos.

Para generar la tabla de contenidos, la lista de …guras y la lista de tablas se utilizarán los tres comandos

```
ntableofcontents nlistoffigures nlistoftables
```

que además generan tres …cheros de extensiones .toc, .lof y .lot, respectivamente, que contienen la información necesaria para confeccionar esas listas.

Las entradas se producen de dos formas

- en .toc mediante los comandos de seccionamiento de tipo nsection para la tabla de contenidos y
- en .lof y .lot con el comando ncaption presente en los entornos …gure y table para las listas de …guras y tablas, respectivamente.

Como puede verse, una entrada del …chero .toc tiene la forma

```
ncontentsline{tipo}{texto}{página}
```

es decir, contiene el tipo de entrada (tipo), qué debe aparecer en el resultado anunciando esa entrada (texto) y el número de página que debe aparecer(página). Para generar una de esas entradas directamente, podemos utilizar el comando

```
naddcontentsline{fichero}{unidad}{entrada}
```

donde

…chero: es la extensión del …chero al que se quiere añadir: .toc, .lot o .lof,

Comandos de seccionamiento	nivel		en tabla de contenidos?		numerados?	
	book rep.	art.	book rep.	art.	book rep.	art.
\part	-1	0	Si	Si	Si	Si
\chapter	0	—	Si	—	Si	—
\section	1	1	Si	Si	Si	Si
\subsection	2	2	Si	Si	Si	Si
\subsubsection	3	3	No	Si	No	Si
\paragraph	4	4	No	No	No	No
\subparagraph	5	5	No	No	No	No

Figura 3.2: Niveles y comportamiento por defecto de las distintas unidades de seccionamiento, y según las clases de documento (se ha tomado art. y rep. como abreviatura de article y report).

unidad: es el nombre de una unidad de seccionamiento (si ...chero es .toc), table (si ...chero es .lot) y ...gure (si ...chero es .lof),

entrada: es el texto de la entrada que quiere incluirse y para incluir cualquier texto (texto) directamente, el comando

naddtocontents{fichero}{texto}

donde ...chero es como antes.

Para controlar hasta qué nivel de profundidad debe aparecer reseñado en la tabla de contenidos, existe el contador

tocdepth

Su valor por defecto está re‡ejado en la Figura 3.2 de la pág. 71, y puede cambiarse a nuestra conveniencia utilizando el comando nsetcounter.

3.2.6 División de un documento en partes

Cuando el documento es largo, es más conveniente tener el texto de entrada repartido en distintos ...cheros; podríamos tener, por ejemplo, cada capítulo

de un libro de apuntes en su propio …chero (chap1.tex, chap2.tex, . . .). Siempre, por supuesto, debemos mantener un …chero principal que constituya la entrada para LaTeX y que contenga:

```
nbegin{document}
ninclude{chap1}
ninclude{chap2}
nend{document}
```

Con el comando

```
ninclude{fichero}
```

se incluye todo el texto de …chero tal y como si estuviera escrito en el …chero principal. Sin embargo, sí existe una diferencia, que puede ser importante tener en cuenta: cada ninclude comienza una nueva página.

Es importante no olvidar que los …cheros de entrada tienen que ser siempre no formateados (formateados sólo texto). La composición o inclusión de …cheros formateados producirá un error grave que interrumpirá el proceso.

Para ejecutar LaTeX únicamente sobre una parte del documento, el preámbulo debe contener la orden:

```
nincludeonly{…cheros}
```

que hace que sólo se componga el texto de esos …cheros pero se tenga en cuenta los incluídos con ninclude para las cuestiones de paginación, tablas de contenidos, etc.

La manera más simple de descomponer nuestro documento en …cheros más pequeños, es haciendo uso del comando:

```
ninput{…chero}
```

que no permite componer sólo una parte del texto como ninclude{…chero}.

3.2.7 Documento Tipo Libro

Este estilo está diseñado para escribir libros. Normalmente consta de:

1. Un título, con los nombres de los autores.
2. Un prefacio, con una introducción del libro, una sucinta descripción de los capítulos, una descripción de a quién va dirigido, y aspectos varios generales sobre el libro.
3. Una tabla de contenidos, con el índice de secciones y subsecciones indicando páginas en las que se encuentran.
4. Capítulos con una introducción, secciones y subsecciones y, opcionalmente, algún apéndice al capítulo.
5. Unos apéndices generales al libro, conteniendo material auxiliar que puede cortar el ritmo del libro de incluirlo en los capítulos.
6. Un apéndice de notación, con la notación utilizada en el libro, incluyendo los acrónimos. Ésta puede ser global o por capítulos.
7. Una bibliografía, incluyendo todas las referencias citadas en el libro en uno de los formatos estándar.
8. Un índice de palabras, para poder localizar fácilmente material tratado en el libro.

La Figura 3.3 de la pág. 76 muestra un ejemplo de la organización de un libro.

Para escribir una obra con las características de un libro, se utiliza la clase de documento book, que se declara con:

```
ndocumentclass{book}
```

A continuación se recogen las particularidades por defecto de un documento de esta clase.

Los comandos:

```
nfrontmatter nmainmatter nbackmatter
```

indicarán, respectivamente, el comienzo de la parte inicial del libro (página del título, tabla de contenidos, prefacio, . . .),de la parte principal (capítulos) y de la parte …nal (bibliografía, índices, . . .).

Imprime a dos caras: por estar activa la opción twoside la situación y contenidos de la cabecera, el pié y el cuerpo de todas las páginas serán distintos según que la hoja sea de numeración impar o de numeración par.

Paginación: está activa la opción n‡ushbottom, por lo que el espacio entre los párrafos será elástico, intentando que no haya variaciones en la longitud de las páginas.

Título y abstract: una hoja separada para el título; no admite abstract.

Estilo de página: headings, esto es, en la página de numeración par la cabecera contiene el nombre del capítulo y el número de página; en la de numeración impar contendrá la sección y la página.

Comandos de seccionamiento: permitidos con esta clase son:

```
npart nchapter nsection
nsubsection nsubsection nparagraph
nsubparagraph
```

Los contadores secnumdepth y tocdepth valen 2, por lo que resultarán numeradas y reseñadas en la tabla de contenidos hasta las subsecciones.

Numeración: consecutiva a lo largo de cada capítulo tanto para ecuaciones como para …guras y tablas; numera con dos dígitos (el primero para el número de capítulo). Si no ha encontrado un nchapter, y encuentra algo que numerar, pondrá 0.1.

```
ndocumentclass[]{book}
% Dar lista de opciones entre los corchetes
nusepackage{}
```

```
% listar los paquetes entre llaves
ninput{preamble}
% incluir los comandos definidos por el usuario

nbegin{document}

nfrontmatter

ninclude{titulo} % incluir pn'agina con tn'{ni}tulo
ninclude{prefacio} % incluir prefacio
ntableofcontents % generar la tabla de contenidos
nlistoffigures % generar la lista de figuras
nlistoftables % generar la lista de tablas

nmainmatter

ninclude{chapter1} % incluir primer capitulo
ninclude{chapter2} % incluir capitulo segundo
ninclude{chapter3} % incluir capitulo tercero

nappendix

ninclude{appendice1} % incluir primer apendice
ninclude{appendice2} % incluir segundo apendice

nbackmatter

ninclude{notacion}
ninclude{bibliography} % incluir bibliografia
```

```
\documentclass[opciones]{book}
\usepackage[opciones]{paquetes}
\usepackage{makeidx}
\makeindex
%\includeonly{fichero1}
\begin{document}
\bibliographystyle{estilo}
\begin{titlepage}
Título con formato
\end{titlepage}
\frontmatter
\tableofcontents
\listoffigures
\listoftables
\include{fichero0}
\mainmatter
\include{fichero1}
\include{fichero2}
\include{fichero3}
\backmatter
\bibliography{fichero}
\printindex
\end{document}
```

Figura 3.3: Esquema de un …chero para preparar libros.

ninclude{index} % incluir indice

nend{document}

Bibliografía: mediante el entorno thebibliography, que genera la lista de referencias en un capítulo sin numerar.

3.3 Partes Flotantes

Los documentos serían mucho más fáciles de leer si no se cortaran sus diferentes partes y estuvieran colocadas en el lugar adecuado. Sin embargo, esto no es posible y hay que desplazar algunos elementos del mismo, tales como …guras y tablas, a otras páginas, por no caber en el lugar más conveniente. Puesto

que la solución óptima no es posible, no cabe otro remedio que contentarse con menos. Sin embargo, tienen que satisfacerse unas condiciones mínimas de calidad en cuanto a la ordenación de estos elementos en el texto. Esto requiere un gran esfuerzo, para el que LaTeX está muy bien preparado. Éste se encarga de gestionar el posicionamiento de este material, que recibe el nombre de ‡otante. Por ejemplo, en una página se pueden citar cinco …guras y tres tablas, que como es natural, no cabrán en la misma página. LaTeX se encarga de controlar qué …guras o tablas quedan por colocar y las coloca según ciertos criterios de calidad (por ejemplo, no coloca …guras o tablas en páginas anteriores a su citación) procede a colocarlas en determinados lugares (en la misma página en que se citan, si esto es posible, o en la parte superior o inferior de una página, etc.).

De esta manera se consigue rellenar las páginas con el texto, notas de pié de página, etc. necesarios, hasta conseguir un buen equilibrio entre páginas y un grado de llenado uniforme. Ésta es una de las características de calidad que diferencia el LaTeX de otros procesadores de texto menos profesionales. Los entornos ‡otantes más conocidos y utilizados son los table y …gure que, como sus nombres indican, se utilizan para tablas y …guras, respectivamente.

Cuerpos Flotantes

Se entiende por “cuerpo ‡otante” aquel que, por no poder ser dividido entre páginas, no está sujeto al lugar del texto en el que se incluyó. Así, para evitar huecos en blanco, pueden desplazarse dentro de la página e incluso a distintas páginas del documento. Existen dos entornos que producen cuerpos ‡otantes:

```
nbegin {figure}[loc] cuerpo n end {figure}
nbegin {table}[loc] cuerpo nend {table}
```

y sus versiones …gure* y table*, que generan …guras y tablas a dos columnas trabajando con la opción twoside, pero que trabajan igual a las anteriores con la opción oneside. Estos dos entornos sirven para colocar cuerpo y para ponerle un pié y un número. No sirven para dibujar ni para confeccionar tablas: el argumento cuerpo se les da totalmente de…nido.

El signi…cado de sus argumentos es como sigue:

cuerpo: Es el contenido de la tabla o …gura. El cuerpo de estos entornos se escribe en modo párrafo, en una caja "parbox" de anchura igual a la del texto (ntextwidth).

loc: El argumento opcional loc indica dónde se quiere que se sitúe preferentemente la …gura o tabla. Es una sucesión de 1 a 4 letras, que la situarán en:

t la parte superior de una página

b la parte inferior

h el punto donde se pone el entorno

p una hoja separada que contiene solo …guras y tablas

! para que se sitúa lo mas cerca posible.

Se cuenta además con el comando nsuppress‡oats[sit] que impide la colocación de más tablas o …guras en la página actual:

- en la parte superior si sit es t,
- en la parte inferior si sit es b,
- en ambas si sit no aparece.

Un ejemplo de situación que puede ser evitada así, es que una sección comience después de que haya aparecido, en la parte superior de la página, una …gura o tabla perteneciente a ella.

La colocación de …guras y tablas se rige por unas ciertas normas:

1. Se imprimirá en el primer lugar que no viole las reglas que siguen, excepto si en loc h precede a t.
2. No se imprimirá en una página anterior a aquella que contiene el entorno que la introduce en el texto.
3. No se imprimirá antes de otra …gura o tabla cuyo entorno sea anterior.
4. Aparecerá sólo en una posición permitida por el argumento loc; en su defecto, seguirá la norma tbp.

5. Una …gura o tabla nunca producirá una página demasiado larga.

6. Existen unos límites marcados por los parámetros de estilo que se comentarán más adelante; estos límites no se respetan con la opción !,

7. que sólo conserva los referentes a páginas de cuerpos ‡otantes (que son n‡oatpagefraction y ndblpagefraction).

En cuerpo, puede aparecer el comando

ncaption[entradas]{título},

que produce un pié a la …gura o tabla con la correspondiente leyenda, donde:

título: es el título de la leyenda a poner en la tabla o …gura. Si el comando se pone antes que la tabla o …gura la leyenda se situada en cabeza de las mismas. Si fuera detrás se situaría en el pié de las mismas.

entradas: El argumento opcional entradas será lo que aparezca en la lista de …guras o lista de tablas referente a esta …gura o tabla; si no, la entrada en esas listas será título. Si se quiere etiquetar esta …gura o tabla, se pone nlabel{etiqueta} dentro de título.

3.3.1 Referencia Cruzada e Índice de Palabras

Referencia Cruzada

LaTeX permite referenciar cualquier elemento (entorno, item, sección, página, etc.) que esté numerado, es decir, que tenga asociado un contador.

Por ejemplo, para citar una ecuación hay que indicar la señal que debió asignársele en su momento. El programa guardó, en un …chero de extensión .aux, qué valor del contador de ecuaciones se correspondió con esa señal y lo colocará en el lugar deseado.

Por tanto, podemos poner una señal en:

[2] Las unidades de seccionamiento: chapter, section, etc.

- En los entornos que pueden numerarse: equation, eqnarray, ...gure, table, así como en los de...nidos mediante nnewtheorem.
- En los items del entorno enumerate.
- En las páginas.

Para poner etiquetas, se utiliza el comando

nlabel{etiqueta}

que debe incluirse dentro del elemento a etiquetar y se encarga de escribir una entrada en el ...chero auxiliar, que consta de la señal, el valor del contador que se trate (entorno, item, sección, . . .) y el valor del contador de página.

Para hacer referencia a un bloque etiquetado, se pone:

nref{etiqueta}

que debe sustituir a la referencia concreta.

También es posible referirse a la página en que se encuentra el elemento. Para ello se utiliza:

pageref{etiqueta}

La posibilidad de utilizar referencias cruzadas es una gran ventaja que ofrece LATEX. Ello permite no tener que preocuparse de cambiar la numeración de partes, capítulos, secciones, subsecciones, tablas, theoremas, etc. Si se suprime un capítulo, tres secciones y cuatro fórmulas, las referencias cruzadas permiten que LATEX2e se encargue de la renumeración correspondiente. Para ello hay que poner etiquetas en los elementos referidos y utilizar las mismas al hacer referencias.

Índice de Palabras

Para incluir en el documento un índice alfabético de palabras a dos columnas, se debe tener todas las palabras o expresiones que se quieran reseñar agrupadas en un entorno theindex:

```
nbegin{theindex} texto nend {theindex}
```

donde texto es una lista de entradas, cada una de ellas encabezada por el comando nitem; a su vez, cada entrada puede tener "subentradas" (encabezadas por nsubitem) y "subsubentradas" (encabezadas por subsubitem); para un espacio extra separando bloques de palabras, se pondrá nindexspace.

Para generarlo hay que hacer lo siguiente:

- colocar

```
nindex{expresión}
```

al lado de cada expresión que se quiera para el índice; este comando no "imprime" nada ni deja un blanco

- poner en el preámbulo

```
nusepackage{makeidx}
```

para cargar el paquete y nmakeindex que se encargará de ir leyendo cada nindex que se haya colocado y escribir para él una entrada

```
nindexentry{expresión}{página}
```

- a partir de ese …chero .idx, generar el entorno theindex y llamarlo a componer; este proceso puede hacerce de dos formas:

(a) haciendo uso del programa MakeIndex:

- que genera el entorno theindex automáticamente
- lo guarda en un …chero de extensión .ind
- tal …chero se compone poniendo

  \printindex

  en el punto del documento donde se quiera que aparezca el índice (normalmente al …nal).

(b) modi…cando el …chero de extensión .idx se puede generar directamente el entorno theindex y colocarlo luego donde se desee.

3.3.2 Bibliografía

Referencias Bibliográ…cas

Para referencias bibliográ…cas el funcionamiento es bastante parecido. Las referencias o entradas bibliográ…cas estarán todas agrupadas, formando el cuerpo del entorno thebibliography. Irá colocado o bien en uno de los …cheros que se incluyan en el principal o bien en un …chero aparte. En este último caso, el …chero deberá tener extensión .bbl y será incluido en la composición mediante el comando

\bibliography{…chero}

Este entorno (thebibliography) está de…nido como una lista y como tal se escribirá, en la salida, precedida cada entrada de una etiqueta. La estructura es

\begin {thebibliography}{ancho-max} entradas \end
{thebibliography}

donde el argumento ancho-max indica cuál es la mayor anchura de las etiquetas de las entradas. Cada una de esas entradas, viene dada por el comando

\bibitem[etiqueta] {señal}

Si el argumento opcional etiqueta está presente, este comando asocia la entrada con etiqueta, activándose en caso contrario el contador enumi y etiquetándose las entradas por orden. El argumento obligatorio señal será el que se utilice para referenciar esa entrada en el texto, poniendo

\cite{señal}

El …chero con extensión .bbl que contiene la lista de referencias bibliográ…cas puede ser generado por nosotros mismos, o de forma mucho más automatizada, aplicando el programa BibTEX. Este programa lee un …chero de extensión .bib, donde se encuentra recogida toda la información sobre las referencias bibliográ…cas que se utilizan en el trabajo, escogiendo de entre ellas sólo aquellas que aparezcan citadas (con ncite) en el documento actual.

De esta forma, el proceso de incluir citas bibliográ…cas y una lista de las referencias citadas, si se dispone del programa BibTEX, será:

1. Disponer de la base de referencias bibliográ…cas guardada en un …chero .bib.
2. Para cada referencia bibliográ…ca, colocar ncite{señal} donde se quiera citar la referencia señalada con señal.
3. Indicar en el …chero a componer (por ejemplo prin.tex) el estilo en que se escribirá la lista, con:

   \bibliographystyle{estilo}

   así como el nombre del …chero .bib, con

   \bibliography{…chero}.

4. Componer con LATEX el …chero prin.tex; por cada ncite se genera una entrada en el …chero auxiliar prin.aux.

5. Correr BibTEX sobre prin.tex; a partir del .aux se genera un nuevo ...chero .bbl que contiene las referencias formateadas según el estilo elegido (este estilo se guarda en un ...chero .bst). Los errores de este proceso se guardan en un ...chero .blg.

6. Componer prin.tex con LATEX, para que lea las referencias del .bbl.

7. Componer otra vez para que escriba la etiqueta adecuada en cada ncite.

Este capítulo tiene por objetivo presentar brevemente las características y ventajas de este procesador de texto, las cuales motivaron la elección del mismo para la realización de todos los documentos del presente Trabajo Final de Aplicación.

Capítulo 4

La Aplicación: Simulación de Redes Neuronales Arti...ciales

4.1 Descripción del Entrenamiento

Se describe a continuación un ejemplo del entrenamiento de una red MLP (perceptrón multicapa) mediante un algoritmo de Gradiente Descendente, un algoritmo de optimización del paso de entrenamiento Paso Fijo, y una función de costo Cuadrática.

En primer lugar, se crean los 5 objetos que se van a relacionar para el entrenamiento: ObjetoEntrenable, Algoritmo, AlgoritmoOptimizacionPaso, Problema y FuncionCosto. Para ello se invoca al constructor de la clase correspondiente con los argumentos adecuados. En el caso que nos ocupa el objeto a entrenar es un MLP (perceptrón multicapa), cuyo código fuente es:

public MLP (int numeroCapas, int[ ] neuronasPorCapa, FuncionActivacion funcionActivacion) {

por lo tanto se necesita crear un vector indicando el número de neuronas por cada capa y el objeto funcionActivacion. Si por ejemplo, la función de activación escogida es la Sigmoidal, se crea previamente este objeto, utilizando el constructor del objeto Sigmoide.

public Sigmoide ()

Este constructor no admite argumentos, por lo que se puede crear directamente. En el caso que se investiga, se desea entrenar una red MLP con tres capas, con 2 neuronas de entrada en cada capa. La creación del objeto correspondiente presenta un código similar a este:

```
/* inicialización del número de capas */

int numeroCapas = 3

/* vector del número de neuronas por capa, de tres elementos */

int[ ] neuronasPorCapa = new int[numeroCapas];

neuronasPorCapa={ 2,2,2 };

/* función de activación */

Sigmoide funcion= new Sigmoide();

/* MLP */

MLP redEntrenable = new MLP (numeroCapas, neuronasPorCapa, funcion);
```

Con estas sencillas líneas se obtiene un MLP de las dimensiones señaladas con todos sus elementos (neuronas, pesos, etc). En el caso más general se deberían crear los elementos particulares, conectarlos y hacerlos depender del objeto Red: basta con comunicar al objeto las entradas y las salidas. Cada objeto Peso se construiría con el índice que le permitiese acceder al vector de pesos global. En nuestro caso, todas estas operaciones vienen incluidas en el constructor del MLP.

Una vez creada la red, se puede crear el objeto Problema. El constructor tiene la forma:

```
public Problema(ObjetoEntrenable objetoProblema,

double[ ][ ] paresEntrenamiento, FuncionCosto funcionCosto) {
```

Suponiendo un caso sencillo con dos pares de entrenamiento con la siguiente forma:

1,1,1——0,1,0

1,0,0——1,1,0

se ubican estos datos en un array bidimensional de 2 x 6 (2 pares de

entrenamiento, 6 datos por par de entrenamiento) nombrándolo, por ejemplo, vectorPatrones. La función de costo escogida es la cuadrática (constructor sin argumentos, con lo que de paso se construye el objeto FuncionCosto). Un posible código es:

```
/* inicializar el vector de patrones */

double[ ] vectorPatrones = new double[2][6];

/* asignar las componentes del vector */

vectorPatrones={ {1,1,1,0,1,0} , {1,0,0,1,1,0} };

/* creación de la función de costo */

Cuadratica funcionCosto= new Cuadratica();

Problema problema= new Problema(redEntrenable,

vectorPatrones, funcionCosto);
```

El siguiente paso es construir los objetos Algoritmo y AlgoritmoOptimizacionPaso, que en este caso corresponden a un GradienteDescendente y a una optimización a través de PasoFijo. El constructor del algoritmo es:

```
public GradienteDescendente (Problema problema, double[ ] vectorDis ){
```

Por lo tanto, deben haber sido creados el problema (lo cual ya está hecho) y un vector de diseño inicial. La mayoría de las ocasiones se utiliza un vector con valores aleatorios, teniendo en cuenta que la dimensión del vector de pesos debe coincidir con el vector de pesos inicial de Red, con lo que se pasa este objeto a través del método ...joVector de Problema:

```
/* inicialización del vector de pesos de la red */

problema.inicializarPesos();

/* construir el objeto Algoritmo */

GradienteDescendente algoritmo = new GradienteDescendente

        (problema, problema.decirVectorPesos());
```

Además, al ser el Gradiente Descendente un algoritmo que necesita un vector de derivadas, es necesario inicializar este vector.

```
/* inicialización del vector de derivadas de la red */

red.iniciarDerivadas();
```

Una vez creado el Algoritmo, queda por construir el AlgoritmoOptimizacionPaso, y en concreto uno que implemente el algoritmo Paso Fijo. Una vez más se analiza el constructor correspondiente:

```
public PasoFijo(Algoritmo algoritmo) {
```

Por lo que la construcción es simple. El objeto Algoritmo conoce la existencia de este objeto para poder requerirle el valor del optimizado del paso de entrenamiento. Esto se consigue mediante la invocación del método ...jarAlgoritmoPaso. En caso de no invocarlo, el algoritmo de optimización por defecto es PasoFijo (no se optimiza el paso de entrenamiento). El código es:

```
/*construcción del objeto PasoFijo*/

PasoFijo algoritmoPaso = new PasoFijo(algoritmo);

/*indicar al algoritmo qué algoritmo de optimización de paso

se está utilizando*/

algoritmo....jarAlgoritmoPaso(algoritmoPaso);
```

Con estas líneas se han construido los objetos que intervienen en el entrenamiento y se encuentran preparados para los cálculos. Recapitulando, los pasos seguidos hasta ahora han sido:

1. Crear el objeto Red. Para ello, en el caso más general se deben crear objetos previos (elementos de red, funciones de activación...) y relacionarlos con este objeto (por ejemplo a través del constructor).
2. Crear el objeto FuncionCosto.
3. Crear el objeto Problema. Para ello se necesita haber creado el objeto a entrenar (paso 1), la función de costo a usar (paso 2) y un array con los pares de entrenamiento de dimensiones.
4. Crear el objeto Algoritmo. Su constructor necesita un vector de diseño que coincida con el vector inicial de pesos del objeto a entrenar. Este vector puede ser generado de múltiples formas. En el caso de un vector con pesos aleatorios se invoca el método inicializarPesos del objeto Problema. Asimismo requiere un objeto Problema (paso 3).

5. Crear el objeto AlgoritmoOptimizacionPaso. Su constructor acepta como argumento el objeto Algoritmo (paso4).
6. Relacionar los objetos Algoritmo y AlgoritmoOptimizacionPaso. Para ello se invoca al método ...jarAlgoritmoPaso de este último objeto.

Ya inicializados todos los objetos que forman parte del entrenamiento, éste puede comenzar. Se invoca al método comienzaOptimiza del objeto Algoritmo. En el caso analizado se trata de un GradienteDescendente, cuyo método es de la forma:

```
public void comienzaOptimiza (double errorDeseado,
int iteracionesMaximas, double learningRateInicial) {
```

Este método ...ja las condiciones del entrenamiento y del ...n de su ejecución. El entrenamiento se detiene cuando se obtenga un error (costo) menor que el deseado o se alcance el número máximo de iteraciones. Suponiendo un error máximo deseado de 0.001, un número máximo de iteraciones de 1000, y un valor inicial del parámetro de entrenamiento de 0.25 (aunque este último será optimizado por el objeto PasoFijo), se tendría la siguiente línea de código:

```
/* comienza el entrenamiento */
algoritmo.comienzaOptimiza(0.001,1000,0.25);
```

Una vez ...nalizado el aprendizaje, se podrá acceder a los resultados del entrenamiento, por ejemplo el vector de pesos que hace mínimo el error, evolución de la función de error, etc.).Para obtener el vector de pesos y la evolución del error (el Algoritmo almacena los valores del costo en un vector), el código es:

```
/* obtención del vector de pesos de mínimo costo*/
double[ ] pesosEntrenamiento = algoritmo.decirVectorDisMinimo();
/* obtención del costo mínimo durante el entrenamiento */
double errorMinimo = algoritmo.devuelveErrorMinimo();
/* obtención de la evolución del error */
double[ ] vectorError = algoritmo.evolucionError();
```

Con esto, se realiza el entrenamiento deseado. El código completo de este ejemplo es:

```
/* inicialización del número de capas */
int numeroCapas = 3
/* vector del número de neuronas por capa, de tres elementos */
int[ ] neuronasPorCapa = new int[numeroCapas];
neuronasPorCapa= {2,2,2};
/* función de activación */
Sigmoide funcion= new Sigmoide();
/* MLP */
MLP redEntrenable = new MLP (numeroCapas,neuronasPorCapa,funcion);
/* inicializar el vector de patrones */
double[ ] vectorPatrones = new double[2][6];
/* asignar las componentes del vector */
vectorpatrones={ {1,1,1,0,1,0} , {1,0,0,1,1,0} };
/* creación de la función de costo */
Cuadratica funcionCosto= new Cuadratica();
Problema problema= new Problema(redEntrenable,vectorPatrones,
funcionCosto);
/* inicialización del vector de pesos de la red */
problema.inicializarPesos();
/* construir el objeto Algoritmo */
GradienteDescendente algoritmo = new
    GradienteDescendente(problema,problema.decirVectorPesos());
/* inicialización del vector de derivadas de la red */
redEntrenable.iniciarDerivadas();
```

```
/* construcción del objeto PasoFijo */
PasoFijo algoritmoPaso = new PasoFijo(algoritmo);
/* indicar qué algoritmo de optimización de paso se está utilizando */
algoritmo.…jarAlgoritmoPaso(algoritmoPaso);
/* comienza el entrenamiento */
algoritmo.comienzaOptimiza(0.001,1000,0.25);
/* obtención del vector de pesos de mínimo costo*/
double[ ] pesosEntrenamiento = algoritmo.decirVectorDisMinimo();
/* obtención del costo mínimo durante el entrenamiento */
double errorMinimo = algoritmo.devuelveErrorMinimo();
/* obtención de la evolución del error */
double[ ] vectorError = algoritmo.evolucionError();
```

De manera similar a la expuesta, para el caso de un Perceptrón Multicapa (MLP), es posible crear un nuevo objeto Red, el cual puede representar a otra arquitectura de red, por ejemplo una red neuronal de Hop…eld o una Competitiva.

4.2 El Código Fuente

A continuación se expone el Código Fuente de la Aplicación desarrollada. En primer término se podrá apreciar la clase principal Simu y luego las clases que ella utiliza.

4.2.1 Clase Simu

```
/* @(#)Simu.java 1.0 03/01/17 */
package myprojects.simu;
import myprojects.calculos.*;
```

```
import myprojects.interfase.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import java.util.*;
import java.lang.*;
import java.io.*;
import java.awt.print.*;
import java.awt.Image;
import java.awt.Toolkit;
import java.awt.Container;
import javax.swing.ImageIcon;
class CerrarVentana extends WindowAdapter{
public void windowClosing(WindowEvent e){
System.exit(0);
}
}
//Choice de función de activación
class Chmama extends Choice{
public Chmama(){
super();
addItem("Signo");
addItem("Lineal");
```

```
addItem("Sigmoidal");
addItem("Tang. Hiperb.");}
}
public class Simu extends Frame
implements ActionListener,ItemListener{
//Barra de Menú
MenuBar barra;
//Opciones de la Barra de Menú
Menu marchivo,mver,mopciones,mayuda;
//MenuItem de cada opción
MenuItem miAcerca,miGrafError,miGrafVal,miGrafEV,miAbrirPatron,
miAbrirPatVal,miFijarPeso,miSalvarPeso,mi1,mi2,miVectPes,miResult,
miResVal,miResGen,miAnalisis,miAnalisisVal,miDetallado,miGrafPesos,
miNuevaRed,miAbrirRed,miGuardarRed,mi8,miSalir,miImpri,mi10,mi11,
miEjecAleat,miAyuda,miCita,miEjecActual,miValid;
/*Botones: "Patrones", "Pesos", "Ejec. Aleat.", "Ejec. Actual y "Salir"
respectivamente*/
Button botentren,botpesos,botcanc,botejec,botejval,botentact;
//Etiquetas
Label labotraeti,labfcosto,labpaso,labcapas,labneurocap,labelfa,
labelch1,lablr,labma,labmarerr,label8,label9,label10;
//Campos de Texto
public TextField campoLimiteSuperior,campoLimiteInferior,campoCapas,
campoNeuronasCapas,campoError,campoIteraciones,campoLR,
```

```
t¤a,tf1,tf2,tf6,tf8,tf9,tf10;
//Choices
Choice chaprendizaje,chfactiv,chadaptativo,chfcosto;
/* Variables y Objetos */
//numero de neuronas de entrada de la red
int numeroNeuronasEntrada;
//numero de neuronas de salidas de la red
int numeroNeuronasSalida;
//patrones del aprendizaje que seran leidos desde ...chero
public double[][] patrones;
//fecha de inicio del aprendizaje y validación
public Date horaInicial, horaInicialVal;
//fecha ...nal del aprendizaje
public Date horaFinal,horaFinalVal ;
// nombre actual del ...chero desde donde se leen los elementos de la red
public String nombreFicheroActual;
//nombre del ...chero de los datos actuales
public String nombreFicheroDatos,nombreFicheroEnt,nombreFicheroVal;
// string con el tipo de paso
public String stringPaso;
// string con la funcion de activacion seleccionada
public String stringFuncion;
//string con la funcion de costo seleccionada
public String stringFuncionCosto;
```

```
// objeto problema actual
Problema problemaActual;
// objeto funcionCosto;
FuncionCosto funcionCosto;
// algoritmo de aprendizaje
Algoritmo algor = new GradienteDescendente();
//string del aprendizaje seleccionado
public String aprendizaje;
// algoritmo de optimización de paso
AlgoritmoOptimizacionPaso
parapaso = new AlgoritmoOptimizacionPaso();
//parametro de error maximo deseado en el algoritmo de gradiente
descendente
public double errorDeseado= 1e-5;
//parametro de iteraciones maximas en el algoritmo de gradiente
descendente
public int iteracionesMaximas = 1000;
//parametro de learning rate inicial en el algoritmo de gradiente
descendente
public double learningRateInicial= 0.25;
// variable para informar el error obtenido en el entrenamiento y la
validación
public double imprimoError,imprimoErrorVal;
// variable para informar las iteraciones resultantes
```

```
public int imprimoIteraciones,imprimoIteracionesVal;
/*string que indican los diferentes tipos de
aprendizajes disponibles*/
public String opcion1 = "Gradiente Descendentenn";
public String opcion2 = " ";
/*conjunto de strings que indican los diferentes algoritmos de
* optimización de paso disponibles*/
public String paso0 = "Paso Fijonn";
public String paso1 = "Ajuste Parabóliconn";
/*conjunto de strings que indican las diferentes
* funciones de costo disponibles*/
public String funcionCosto0 = "Cuadráticann";
public String funcionCosto1 = "Cuadrática Ponderadann";
//string que indica la funcion de costo seleccionada
public String stringCostoActual = funcionCosto0;
//red global
public Red redGlobal;
//lectura del tiempo …nal
long milisegFin,milisegFinVal;
//lectura del tiempo inicial
long milisegInic,milisegInicVal;
//limite inferior de inicio de los pesos
double limiteInferior = -1;
//limite superior inicial de los pesos
```

```
double limiteSuperior = 1;
//vector de ponderación de los pesos
double[] vectorPonderacion;
//vector de los errores
public double[] datos,datos2;
//número de capas;
public int numeroCapas;
// neuronas por cada capa
public int[] neuronasPorCapa;
// red actual
MLP redActual;
//dibujo de la red
DibujoMLP dibujoRed;
/*string relativo a la función de activación sigmoide*/
String funcion0 = "Sigmoidenn";
//string relativo a la función de activación tanh(x)
String funcion1 = "Tanh(x)nn";
/*string relativo a la función de activación lineal*/
String funcion2 = "Linealnn";
/*string relativo a la función de activación seleccionada*/
String funcionActivacionActual = funcion0;
//public double[] pesosActuales;
/*Constructor Simu*/
public Simu() {
```

```
barra =new MenuBar();
setMenuBar(barra);
marchivo =new Menu("Archivo");
barra.add(marchivo);
mver =new Menu("Ver");
barra.add(mver);
mopciones =new Menu("Opciones");
barra.add(mopciones);
mayuda =new Menu("Ayuda");
barra.add(mayuda);
miAbrirRed =new MenuItem("Abrir Red");
miAbrirRed.addActionListener(new AbrirRed());
marchivo.add(miAbrirRed);
marchivo.addSeparator();
miAbrirPatron =new MenuItem("Abrir Patrones");
miAbrirPatron.addActionListener(new LeerDatosFichero());
marchivo.add(miAbrirPatron);
miFijarPeso =new MenuItem("Abrir Pesos");
miFijarPeso.addActionListener(new FijarPesos());
marchivo.add(miFijarPeso);
marchivo.addSeparator();
miSalvarPeso =new MenuItem("Guardar Pesos");
miSalvarPeso.addActionListener(new GrabarPesos());
marchivo.add(miSalvarPeso);
```

```
miSalvarPeso.setShortcut(new MenuShortcut(KeyEvent.VK_P));
miGuardarRed =new MenuItem("Guardar Red");
miGuardarRed.addActionListener(new SalvarFichero());
marchivo.add(miGuardarRed);
marchivo.addSeparator();
miGuardarRed.setShortcut(new MenuShortcut(KeyEvent.VK_R));
miSalir =new MenuItem("Salir");
miSalir.addActionListener(this);
marchivo.add(miSalir);
miSalir.setShortcut(new MenuShortcut(KeyEvent.VK_S));
//Opcion Ver
miResult =new MenuItem("Resultados del Entrenamiento");
miResult.addActionListener(new MostrarHora());
mver.add(miResult);
miResVal =new MenuItem("Resultados de Validación");
miResVal.addActionListener(new PublicarValidacion());
mver.add(miResVal);
miResGen =new MenuItem("Resultados Generales");
miResGen.addActionListener(new PublicarGenerales());
mver.add(miResGen);
mver.addSeparator();
miAnalisis =new MenuItem("Vector de Errores del Entrenamiento");
miAnalisis.addActionListener(new MostrarAnalisis());
mver.add(miAnalisis);
```

```
miAnalisisVal =new MenuItem("Vector de Errores de la Validación");
miAnalisisVal.addActionListener(new MostrarAnalisisVal());
mver.add(miAnalisisVal);
miVectPes =new MenuItem("Vector de Pesos del Entrenamiento");
miVectPes.addActionListener(new MostrarPesosMLP());
mver.add(miVectPes);
mver.addSeparator();
miGrafError=new MenuItem("Grá...co de Error de Entrenamiento");
miGrafError.addActionListener(new MostrarGra...ca());
mver.add(miGrafError);
miGrafError.setShortcut(new MenuShortcut(KeyEvent.VK_G));
miGrafVal=new MenuItem("Grá...co de Error de Validación");
miGrafVal.addActionListener(new MostrarGra...caValidacion());
mver.add(miGrafVal);
miGrafVal.setShortcut(new MenuShortcut(KeyEvent.VK_A));
miGrafEV=new MenuItem("Grá...co de Error de Entrenamiento y
Validación");
miGrafEV.addActionListener(new MostrarGra...caErrVal());
mver.add(miGrafEV);
miGrafEV.setShortcut(new MenuShortcut(KeyEvent.VK_X));
//Opcion Opciones
miEjecAleat =new MenuItem("Ejecutar (Pesos Aleatorios)");
miEjecAleat.addActionListener(new Ejecutar());
mopciones.add(miEjecAleat);
```

```
miEjecActual =new MenuItem(”Ejecutar (Pesos Actuales)”);
miEjecActual.addActionListener(new Seguir());
mopciones.add(miEjecActual);
miEjecActual.setShortcut(new MenuShortcut(KeyEvent.VK_E));
mopciones.addSeparator();
miValid =new MenuItem(”Validar”);
miValid.addActionListener(new Validar());
mopciones.add(miValid);
miValid.setShortcut(new MenuShortcut(KeyEvent.VK_V));
//Opcion Ayuda
miAyuda =new MenuItem(”Ayuda”);
miAyuda.addActionListener(new VerAyuda());
mayuda.add(miAyuda);
miCita =new MenuItem(”Cita”);
miCita.addActionListener(new MostrarCita());
mayuda.add(miCita);
miAcerca =new MenuItem(”Acerca de...”);
miAcerca.addActionListener(new Acerca());
mayuda.add(miAcerca);
JPanel panel = new JPanel();
Border border = BorderFactory.createEtchedBorder();
panel.setBorder(BorderFactory.createTitledBorder(border,
”Topología de Red”));
panel.setLayout(new GridLayout(3,2,5,5));
```

```
panel.setPreferredSize(new Dimension(70,70));
JPanel panel8 = new JPanel();
panel8.setLayout(new GridLayout(1,1,5,5));
panel8.setPreferredSize(new Dimension(30,300));
JPanel panel2 = new JPanel(new GridLayout(8,2,5,5));
panel2.setBorder(BorderFactory.createTitledBorder(border,
”Parámetros de Entrenamiento”));
panel2.setPreferredSize(new Dimension(250,300));
add( panel2, BorderLayout.WEST);
//Componentes de panel2
lablr = new Label(”Parám. de Aprend.”);
panel2.add(lablr);
campoLR = new TextField(”0.25”);
panel2.add(campoLR);
label8 = new Label(”Iteraciones”);
panel2.add(label8);
campoIteraciones = new TextField(”1000”);
panel2.add(campoIteraciones);
labmarerr = new Label(”Error Máximo”);
panel2.add(labmarerr);
campoError = new TextField(”1.0E-5”);
panel2.add(campoError);
label9 = new Label(”Peso Mínimo”);
panel2.add(label9);
```

```
campoLimiteInferior = new TextField("-1.0");
panel2.add(campoLimiteInferior);
label10 = new Label("Peso Máximo");
panel2.add(label10);
campoLimiteSuperior = new TextField("1.0");
panel2.add(campoLimiteSuperior);
JPanel panel3 = new JPanel(new GridLayout());
panel3.setBorder(BorderFactory.createTitledBorder(border,
"Grá…ca de Red"));
panel3.setPreferredSize(new Dimension(350,300));
add( panel3, BorderLayout.CENTER);
panel3.add(panel);
//Choice Funcion de Activacion
chfactiv = new Chmama();
labcapas = new Label("Número de Capas");
panel.add(labcapas);
campoCapas = new TextField();
panel.add(campoCapas);
labneurocap = new Label("Neuronas por Capa");
panel.add(labneurocap);
campoNeuronasCapas = new TextField();
panel.add(campoNeuronasCapas);
labelfa = new Label("F( x ) de Activ.");
panel.add(labelfa);
```

```
panel.add(chfactiv);//Agrego el choice de func. de act.
//choice algoritmo de aprendizaje
chaprendizaje = new Choice();
chaprendizaje.addItem("Aprend. de Hebb");
chaprendizaje.addItem("Gradiente Descend.");
labelch1 = new Label("Algorit. de Entren.");
panel2.add(labelch1);
panel2.add(chaprendizaje);
//choice Paso Adaptativo
chadaptativo = new Choice();
chadaptativo.addItem("Paso Fijo");
chadaptativo.addItem("Ajuste Parabólico");
labpaso = new Label("Paso Adaptativo");
chadaptativo.addItemListener(this);
panel2.add(labpaso);
panel2.add(chadaptativo);
//choice Funcion Costo
chfcosto = new Choice();
chfcosto.addItem("Cuadrática");
chfcosto.addItem("Cuadrát. Ponderada");
labfcosto = new Label("Función Costo");
chfcosto.addItemListener(this);
panel2.add(labfcosto);
panel2.add(chfcosto);
```

```
//Panel6 es un auxiliar para disponer panel 4 y 5 dentro
JPanel panel6 = new JPanel(new GridLayout(1,2,5,5));
add( panel6, BorderLayout.SOUTH);
JPanel panel5 = new JPanel(new GridLayout(2,3,5,5));
panel5.setBorder(BorderFactory.createTitledBorder(border,
"Archivos de Datos / Opciones"));
panel5.setPreferredSize(new Dimension(130,130));
add( panel5, BorderLayout.EAST);
/*Agrego "Topología de Red" (panel y panel5)*/
panel6.add(panel);
panel6.add(panel5);
/*Botones*/
botentren = new Button("Patrones");
panel5.add(botentren);
botentren.addActionListener(new LeerDatosFichero());
botejec = new Button("Ejec. Aleat.");
panel5.add(botejec);
botejec.addActionListener(new Ejecutar());
botentact = new Button("Ejec. Actual");
panel5.add(botentact);
botentact.addActionListener(new Seguir());
botpesos = new Button("Pesos");
panel5.add(botpesos);
botpesos.addActionListener(new FijarPesos());
```

```
botejval = new Button("Validación");
panel5.add(botejval);
botejval.addActionListener(new Validar());
botcanc = new Button("Salir");
panel5.add(botcanc);
botcanc.addActionListener(new Cancelar());
/*En forma predeterminada la red
tiene 3 capas con dos neuronas en cada capa*/
campoError.setText("0");
campoIteraciones.setText("0");
numeroCapas = 3;
neuronasPorCapa = new int[30];
for (int i=0;i<numeroCapas;i++)
{neuronasPorCapa[i]=2;
}
/* Se dibuja la red predeterminada*/
dibujoRed = new DibujoMLP(numeroCapas,neuronasPorCapa);
panel8.add(dibujoRed);
panel3.add(panel8);
patrones=null;
campoCapas.setText(String.valueOf(numeroCapas));
/* Se imprimen los valores actuales de las neuronas por */
/* capa, para ello declaro un String y le añado */
/* las componentes del vector neuronasPorCapa */
```

```
String u = "";
for (int j=0;j<numeroCapas-1;j++)
{u=u+(String.valueOf(neuronasPorCapa[j])+",");
}
u=u+(String.valueOf(neuronasPorCapa[numeroCapas-1]));
campoNeuronasCapas.setText(u);
/*Método para la selección del Algoritmo de Aprendizaje
que …ja como variable la opción escogida*/
chaprendizaje.addItemListener( new ItemListener(){
public void itemStateChanged(ItemEvent evt){
String alg=(String)evt.getItem();
if (alg.equals("Gradiente Descend."))
aprendizaje = opcion1;
}
});
tomarPaso();
tomarCosto();
/*Método para la selección de la F(x) de activ.
que …ja como variable la opción escogida*/
chfactiv.addItemListener( new ItemListener(){
public void itemStateChanged(ItemEvent evt){
String str=(String)evt.getItem();
if (str.equals("Sigmoidal"))
funcionActivacionActual = funcion0;
```

```
else if (str.equals("Tang. Hiperb."))
funcionActivacionActual = funcion1;
else if (str.equals("Lineal"))
funcionActivacionActual = funcion2;
}
});
/*Se crea un objeto de la clase MLP, pasándole los argumentos necesarios*/
FuncionActivacion funcionAct = new Sigmoide();
redActual = new MLP(numeroCapas,neuronasPorCapa,funcionAct);
redGlobal = redActual;
CerrarVentana v = new CerrarVentana();
addWindowListener(v);
}/*Fin del constructor Simu*/
public void actionPerformed(ActionEvent evt){
MenuItem c=(MenuItem)evt.getSource();
String arg=c.getLabel();
if(arg.equals("Salir")){
Salir sale = new Salir();
sale.setLocation(300,300);
sale.setSize(300,100);
sale.show();
}
}
public void itemStateChanged(ItemEvent evt){
```

```
}
public static void main(String args[]) {
//presentacion();
Simu mainFrame = new Simu();
mainFrame.pack();
mainFrame.setLocation(100,40);
mainFrame.setTitle("Simulación de Redes Neuronales Arti…ciales");
mainFrame.setVisible(true);
}
/*Método que muestra la presentación del sistema*/
private static void presentacion() {
Thread mensaje = new Thread() {
public void run() {
ImageIcon anilogo = new ImageIcon
( "C:/Tesis/Simu/logo.gif" );
Border borde = BorderFactory.createEtchedBorder();
JLabel etiLogo = new JLabel( anilogo );
etiLogo.setHorizontalAlignment( JLabel.CENTER );
JPanel logoPanel = new JPanel();
logoPanel.setBorder( borde );
logoPanel.add( etiLogo );
int seg = 10;
JLabel etiSegundos = new JLabel( Integer.toString( seg ) );
etiSegundos.setHorizontalAlignment( JLabel.CENTER );
```

```
JPanel secondsPanel = new JPanel();
secondsPanel.setBorder( borde );
secondsPanel.add( etiSegundos );
JWindow Inicio = new JWindow();
Container cont = Inicio.getContentPane();
cont.add( "Center", logoPanel );
//Inicio.setLocation( 210,160 );//210*160
Inicio.setLocation( 110,60 );//210*160
Inicio.setSize(650,450);
Inicio.show();
Inicio.toFront();
long cont2;
long cont1 = System.currentTimeMillis();
while( seg > 0 ) {
if( seg == 8 ) Inicio.toFront();
if( ( cont2 = System.currentTimeMillis() ) >= cont1 + 1000 ) {
cont1 = cont2;
etiSegundos.setText( Integer.toString( --seg ) );
try { Thread.sleep( 250 ); } catch( Exception e ) { continue; }
}
}
Inicio.dispose();
Inicio = null;
}
```

```
};
mensaje.setPriority( Thread.MIN_PRIORITY );
mensaje.start();
}
/*Método para la selección del Paso de optimización
que …ja como variable la opción escogida*/
public void tomarPaso()
{ chadaptativo.addItemListener( new ItemListener(){
public void itemStateChanged(ItemEvent evt){
String pas=(String)evt.getItem();
if (pas.equals("Paso Fijo"))
stringPaso = paso0;
else if (pas.equals("Ajuste Parabólico"))
stringPaso = paso1;
}
});
}
/*Método para la selección de la Función de Costo
que …ja como variable la opción escogida*/
public void tomarCosto()
{ chfcosto.addItemListener( new ItemListener(){
public void itemStateChanged(ItemEvent evt){
String cto=(String)evt.getItem();
if (cto.equals("Cuadrática"))
```

```
stringCostoActual = funcionCosto0;
else if (cto.equals("Cuadrát. Ponderada"))
{stringCostoActual = funcionCosto1;
leerVectorPonderacion();
}
}
});
}
class Ejecutar implements ActionListener{
public void actionPerformed(ActionEvent e) {
tomarCapas();
tomarLimites();
tomarOpcionesGradiente();
ejecutar();
}
}
public void tomarCapas()
{/* recoger los valores de los campos de texto numeroCapas y */
/* campoNeuronasCapas, convertirlos de string a número y aplicar */
/* los cambios a las variables error, learning */
/* rate y el número de capas */
numeroCapas=Integer.parseInt(campoCapas.getText());
dibujoRed.numeroCapas=numeroCapas;
/* cambios en el vector de neuronas por capa */
```

```
/* se recoge el campo de texto correspondiente */
/* en un nuevo string y se detecta la posición del */
/* carácter ',' , se separan los caracteres, se */
/* transforma a integer y se asigna el valor al */
/* vector neuronasPorCapa */
String capas = campoNeuronasCapas.getText();
int principio = 0;
for (int j=0;j<numeroCapas;j++)
{int …n =principio;
try {
while (capas.charAt(…n)!=',' && capas.charAt(…n)!=-1)
{…n++;
}
} catch (StringIndexOutOfBoundsException exc) {}
neuronasPorCapa[j]=Integer.valueOf(capas.substring(principio,…n)
.trim()).intValue();
principio=…n+1;
dibujoRed.neuronasPorCapa[j]=neuronasPorCapa[j];
}
FuncionActivacion funcionAct = new Sigmoide();
redActual = new MLP(numeroCapas,neuronasPorCapa,funcionAct);
redGlobal = redActual;
/* se actualiza la grá…ca */
dibujoRed.repaint();
```

```
}
public void tomarLimites()
{/* recoger los valores de los campos de texto
convertirlos de string a número y aplicar
los cambios a las variables */
limiteInferior=Double.parseDouble(campoLimiteInferior.getText());
limiteSuperior=Double.parseDouble(campoLimiteSuperior.getText());
}
public void tomarOpcionesGradiente()
{/*Recoger los valores de los campos de texto
convertirlos de string a número y aplicar
los cambios a las variables */
errorDeseado=Double.parseDouble(campoError.getText());
int iteracionesLeidas = Integer.parseInt(campoIteraciones.getText());
if(iteracionesLeidas==0)
{iteracionesMaximas= Integer.MAX_VALUE;
}
else
{iteracionesMaximas= iteracionesLeidas;
}
learningRateInicial=Double.parseDouble(campoLR.getText());
}
/*En caso de tener …chero de patrones grabados (en
* caso contrario lanza un mensaje de error), toma los
```

```
* valores temporales del inicio del aprendizaje,
* …ja las variables del mismo (tipo de aprendizaje...),
* y da comienzo al mismo; una vez terminado lanza los Frames
* de resultados*/
public void ejecutar()
{ if(patrones!=null)
{horaInicial = new Date();
milisegInic = System.currentTimeMillis();
FuncionActivacion funcionAct = new Sigmoide();
if(funcionActivacionActual == funcion1)
{funcionAct = new TangenteHiperbolica();
}
if(funcionActivacionActual == funcion2)
{funcionAct = new Lineal();
}
redActual = new MLP(numeroCapas,neuronasPorCapa,funcionAct);
redGlobal = redActual;
FuncionCosto gtr = new Cuadratica();
if(stringCostoActual==funcionCosto1)
gtr = new CuadraticaPonderada(vectorPonderacion);
problemaActual = new ProblemaBP(redActual,patrones,gtr);
problemaActual.inicializarPesos(limiteInferior,limiteSuperior);
…jarAlgoritmos();
sacarResultados();
```

```
ImprimirPesosMLP mostrarPesos =
new ImprimirPesosMLP(redActual.decirPesos(), neuronasPorCapa,
numeroCapas);
mostrarPesos.setLocation(407,250);
mostrarPesos.setSize(390,190);
mostrarPesos.show();
miEjecAleat.setEnabled(true);
botejec.setEnabled(true);
}
else
{Mensaje mens = new Mensaje();
mens.setLocation(270,380);
mens.setSize(250,100);
mens.show();
}
}
/*Fijar los algoritmos de aprendizaje y de
* optimizar el paso además de iniciar el
* aprendizaje de la red */
public void …jarAlgoritmos()
{if(aprendizaje==opcion1)
{algor = new GradienteDescendente
(problemaActual,problemaActual.decirVectorPesos());
}
```

```
if(stringPaso==paso0)
{parapaso = new PasoFijo(algor);
}
if(stringPaso==paso1)
{parapaso = new Parabolico(algor);
}
algor....jarAlgoritmoPaso(parapaso);
if(aprendizaje==opcion1)
{((GradienteDescendente)(algor)).
comienzaOptimiza(errorDeseado,iteracionesMaximas,learningRateInicial);
}
}
/*Leer los datos desde un ...chero, asignando el numero de
* neuronas de entrada a neuronasPorCapa[0] y el número
* de salida a neuronasPorCapa[numeroCapas-1], repintando
* la representación del MLP*/
class LeerDatosFichero implements ActionListener {
public void actionPerformed(ActionEvent e) {
FuncionActivacion funcionAct = new Sigmoide();
if(funcionActivacionActual == funcion1)
{funcionAct = new TangenteHiperbolica();
}
if(funcionActivacionActual == funcion2)
{funcionAct = new Lineal();
```

```
    }
    redActual = new MLP(numeroCapas,neuronasPorCapa,funcionAct);
    redGlobal = redActual;
    leerDatosFichero();
    neuronasPorCapa[0]=numeroNeuronasEntrada;
    neuronasPorCapa[numeroCapas-1]=numeroNeuronasSalida;
    }
    }
    /* Lee los patrones desde ...chero */
    public void leerDatosFichero()
    {String openFile;
    String linea;
    FileDialog dw = new FileDialog(this,"Fichero a abrir");
    dw.setFile("*.txt");
    dw.setDirectory(".");
    dw.show();
    openFile = (dw.getDirectory() + "//" + dw.getFile());
    if((dw.getFile())!= null)
```

4.2.2 Clase Problema

```
package myprojects.calculos;
    import java.io.*;
    /*clase Problema*/
    public abstract class Problema implements Cloneable,Serializable
```

```
{
/*red problema*/
public ObjetoEntrenable objetoProblema;
/*funcion de costo elegida*/
protected FuncionCosto funcionCosto;
/*vector de las salidas de los pares de entrenamiento*/
protected double[][] salidaEntrenamiento;
/*vector de las entradas de los pares de entrenamiento*/
protected double[][] entradaEntrenamiento;
/*salida global para todos los pares del entrenamiento*/
protected double[][] salidaGlobal ;
/*constructor sin argumentos*/
public Problema()
{
}
/*constructor que inicializa las variables propias del problema
a los valores de las variables pasadas como argumentos, y asignando
los valores de los pares de entrenamiento a las variables
entradaEntrenamiento y salidaEntrenamiento*/
public Problema(ObjetoEntrenable objetoProblema,
double[][] paresEntrenamiento,
FuncionCosto funcionCosto)
{
/* inicialización de los valores propios de las variables */
```

```
/* a partir de los pasados como argumentos */
this.objetoProblema = objetoProblema;
this.funcionCosto = funcionCosto;
/* cambiar los pares de entrenamiento por aquellos pasados */
/* como argumento del constructor */
cambiarParesEntrenamiento(paresEntrenamiento);
}
/*constructor que inicializa las variables propias del
* problema a los valores de las variables pasadas como argumentos */
public Problema(ObjetoEntrenable objetoProblema,
FuncionCosto funcionCosto)
{
/* inicialización de los valores propios de las variables */
/* a partir de los pasados como argumentos */
this.objetoProblema = objetoProblema;
this.funcionCosto = funcionCosto;
}
/*devuelve el valor del costo con el vector de pesos
* y salida global actuales*/
public abstract double decirCosto ();
/*devuelve el valor del costo con el vector de pesos
* y salida global actuales*/
public abstract double decirCosto (double[] pesos);
/*...ja un nuevo vector de pesos en el objeto entrenable*/
```

```
public void …joVector (double[] pesos)
{objetoProblema.…jarPesos(pesos);
}
/*calcula el vector de error a partir de los valores deseados
* para la salida y el calculado en el caso de la función de costo
* escogida*/
public double[] calculoVectorError (double[] deseada, double[] calculada)
{
/* genera el vector de error */
double[] vectorError = funcionCosto.generarVectorError (deseada,calculada);
/* devuelve el vector calculado */
return vectorError;
}
/*devuelve el valor de la dimension del vector de pesos*/
public int decirDimension()
{int dimension = objetoProblema.dimensionProblema();
return dimension;
}
/*devuelve el vector de Pesos actual, realizando una copia
* del mismo */
public double[] decirVectorPesos()
{double[] copiaPesos = (double[])((objetoProblema.decirPesos()).clone());
return copiaPesos;
}
```

```
/*devuelve el vector Gradiente actual,
* realizando una copia del mismo*/
public double[] decirGradienteActual()
{double[] copiaDerivadas = (double[])((objetoProblema.decirDerivada()).clone());
return copiaDerivadas;
}
/* devuelve el valor de la salida, con la entrada
* actual y el vector de pesos pasado como argumento,
* sin cambiar el vector actual de pesos*/
public double[] decirSalida (double[] vectorDis)
{
/* realiza la copia del vector de pesos */
double[] copiaPesos = (double[])((objetoProblema.decirPesos()).clone());
/* ...ja el vector de diseño pasado como argumento */
...joVector(vectorDis);
/* calcula la salida con el nuevo vector */
double[] salidaActual = objetoProblema.decirSalida();
/* ...ja el vector antiguo de pesos */
...joVector(copiaPesos);
/* devuelve el valor calculado */
return salidaActual;
}
/*devuelve el valor de la salida, con la entrada
* y el vector de pesos pasados como argumentos,
```

```
* sin cambiar el vector actual de pesos*/
public double[] decirSalida (double[] vectorDis, double[] entradas)
{
/* realiza una copia del vector actual de pesos */
double[] copiaPesos = (double[])((objetoProblema.decirPesos()).clone());
/* ...ja el nuevo vector de pesos */
...joVector(vectorDis);
/* calcula la nueva salida */
double[] salidaActual = new double[salidaEntrenamiento[0].length];
salidaActual = objetoProblema.decirSalida(entradas);
/* restaura el valor del vector de pesos del objeto entrenable */
...joVector(copiaPesos);
return salidaActual;
}
/*calcula la salida para todas las entradas de
* los pares de entrenamiento, asignando el resultado
* a la variable salidaGlobal*/
public void calcularSalidaGlobal()
{
/* coloca el contador de las pilas a 0 */
objetoProblema.inicializarObjeto();
/* dimensiones de la variable salida global */
salidaGlobal = new double[salidaEntrenamiento.length]
[salidaEntrenamiento[0].length];
```

```
/* para cada uno de los pares de entrenamiento */
for (int i=0; i< entradaEntrenamiento.length; i++)
{
/* vector de apoyo para calcular la salida */
/* no es conveniente salida[i] = objetoProblema .... */
/* porque entonces todos los vectores salida[i] apuntan al */
/* mismo resultado */
double[] ayuda = objetoProblema.decirSalida(entradaEntrenamiento[i]);
salidaGlobal[i] = (double[])(ayuda.clone());
/* se renueva el elemento entrenable (delays piden la salida del
elemento anterior) */
objetoProblema.renovarObjeto();
/* se incrementa el contador de las pilas */
objetoProblema.incrementarIndiceEtapas();
}
/* inicializar las pilas para siguientes cálculos */
objetoProblema.inicializarObjeto();
}
/*calcula el gradiente con los valores actuales,
* y con los pares de entrenamiento escogidos*/
public abstract double[] calculaGradiente ();
/* calcula el gradiente con los valores actuales,
* y con los pares de entrenamiento escogidos,
* con el vector de pesos pasado como argumento*/
```

```
public double[] calculaGradiente (double[] vector)
{
/* se crea un vector temporal de pesos donde se guarda */
/* el valor actual del vector de pesos */
double[] pesosTemporal = (double[])((objetoProblema.decirPesos()).clone());
objetoProblema....jarPesos(vector);
double[] gradiente = calculaGradiente();
/* devolver el vector de pesos */
objetoProblema....jarPesos(pesosTemporal);
return gradiente;
}
/*crea las pilas del objeto problema con la dimensión especi...cada como
* argumento*/
public void inicializarObjeto(int dimension)
{objetoProblema.inicializarObjeto(dimension);
}
/*cambia los pares de entrenamiento del objeto entrenable por los
pasados como argumento, creando asímismo las pilas con la
dimensión adecuada*/
public void cambiarParesEntrenamiento(double[][] paresEntrenamiento)
{
/* inicializar las variables del número de entradas y salidas */
int numeroNeuronasEntrada=objetoProblema.decirNumeroEntradas();
int numeroSalidas= objetoProblema.decirNumeroSalidas();
```

```
/* crear las pilas de los elementos */
inicializarObjeto(paresEntrenamiento.length);
/* creación de los vectores de dos índices salida de entrenamiento */
/* y entrada de entrenamiento. En ambos casos el primer índice */
/* corresponde al orden dentro del patrón, mientras que el segundo */
/* corresponde a la coordenada dentro del vector de Neuronas de */
/* entrada /salida */
salidaEntrenamiento = new double [paresEntrenamiento.length]
[numeroSalidas];
entradaEntrenamiento = new double [paresEntrenamiento.length]
[numeroNeuronasEntrada];
/* inicialización de sus valores a partir de los valores pasados */
/* como argumentos */
for(int i = 0; i< paresEntrenamiento.length; i++)
{for(int j = 0; j< numeroNeuronasEntrada; j++)
{entradaEntrenamiento[i][j] = paresEntrenamiento[i][j];
}
for(int j = 0; j< numeroSalidas; j++)
{salidaEntrenamiento[i][j] =
paresEntrenamiento[i][j+numeroNeuronasEntrada];
}
}
} /* ...n del método */
/*crea e inicializa los pesos de la red, con una dimensión indicada
```

```
* por el valor actual de la variable contadorPesos
* con valores entre los límites pasados como argumentos
(inferior y superior)*/
public void inicializarPesos (double inferior, double superior)
{
/* crear el vector de pesos con la dimensión */
int contadorPesos = objetoProblema.dimensionProblema();
double[] pesos = new double[contadorPesos];
double[] pesosIniciales = new double[contadorPesos];
for(int i=0 ; i< pesos.length ; i++)
{pesos[i] = inferior + (superior - inferior) *
Math.random();
pesosIniciales[i] = pesos[i];
}
objetoProblema....jarPesos(pesos);
objetoProblema....jarPesosIniciales(pesosIniciales);
}
/* crea e inicializa los pesos de la red, con la dimensión indicada y
* con valores entre los límites pasados
* como argumentos (inferior y superior)*/
public void inicializarPesos (double inferior, double superior,
int dimension)
{
/* crear el vector de pesos con la dimensión */
```

```
double[] pesos = new double[dimension];
double[] pesosIniciales = new double[dimension];
for(int i=0 ; i< pesos.length ; i++)
{pesos[i] = inferior + (superior - inferior) *
Math.random();
pesosIniciales[i]=pesos[i];
}
objetoProblema....jarPesos(pesos);
objetoProblema....jarPesosIniciales(pesosIniciales);
}
/*inicializa los pesos entre valores comprendidos entre 1 y -1*/
public void inicializarPesos ()
{inicializarPesos(-1,1);
}
public synchronized Object clone()
{Problema o = null;
try
{ o = (Problema)super.clone();
} catch (CloneNotSupportedException e)
{System.out.println("Esto de la clonacion a veces no funciona...");}
o.objetoProblema=(ObjetoEntrenable)objetoProblema.clone();
o.funcionCosto=(FuncionCosto)funcionCosto.clone();
o.salidaGlobal = (double[][])salidaGlobal.clone();
for(int i=0; i<salidaGlobal.length;i++)
```

```
{o.salidaGlobal[i] = (double[])salidaGlobal[i].clone();
}
return o;
}
public double[][] decirSalidaGlobal()
{return salidaGlobal;
}
} /* ...n de la clase */
```

4.2.3 Clase FuncionCosto

```
package myprojects.calculos;
    import java.io.*;
    /*clase de la que derivan las diferentes
    funciones de costo implementadas */
    public abstract class FuncionCosto implements Cloneable, Serializable
    {
    /*constructor sin argumentos*/
    public FuncionCosto() {
    }
    /*calcula el costo según la salida deseada y
    * la salida calculada, en este caso un 0 */
    public abstract double calcularCosto(double[][] salidaDeseada,
    double[][] salidas);
    /*genera el vector Error según la salida deseada y
```

```
    * la salida calculada*/
    public abstract double[] generarVectorError(double[] deseada,
    double[] calculada);
    public Object clone() {
    FuncionCosto o = null;
    try
    {o = (FuncionCosto)super.clone();
    } catch (CloneNotSupportedException e)
    {System.out.println("la clonación a veces no funciona...");}
    return o;
    }
    } /* ...n de la clase */
```

4.2.4 Clase Algoritmo

```
package myprojects.calculos;
    import java.util.*;
    import java.io.*;
    /* clase de la que derivan los diferentes
    algoritmos implementados*/
    public abstract class Algoritmo implements Serializable{
    // problema padre que lo invoca
    protected Problema problema;
    //vector de diseño actual
    protected double[] vectorDis;
```

```
// learning rate del aprendizaje, inicializado en principio a 0.25
protected double learningRate = 0.25;
// vector de diseño con mínimo error, hasta el instante
protected double[] vectorDisMinimo;
// error actual
protected double error;
/* error minimo hasta el momento, inicializado en principio al
máximo valor posible, a …nes de que en la primera iteración
esta variable se iguale al error obtenido */
protected double errorMinimo= Double.MAX_VALUE;
// iteración actual del algoritmo
protected int step;
/* vector en el que se almacena el error producido
en cada iteración; su tamaño ocupado va aumentando
según se almacenan los errores*/
protected Vector vectorError = new Vector();
/* algoritmo de optimización de paso, inicializado por defecto
a PasoFijo (learning rate constante durante el aprendizaje)*/
protected AlgoritmoOptimizacionPaso algoritmoPaso = new PasoFijo(this);
/* constructor sin argumentos para poder extender la clase*/
public Algoritmo (){
}
/*constructor que acepta como argumento el problema a resolver
* y un vector de diseño, igualando las variables propias
```

```
* a las pasadas como argumento*/
public Algoritmo(Problema problema, double[] vectorDis ){
/* se igualan las variables propias del objeto a las variables pasadas
como argumentos */
this.problema = problema;
this.vectorDis = (double[])(vectorDis.clone());
vectorDisMinimo = new double[vectorDis.length];
}
/*constructor que acepta como argumento el problema a resolver
* y la longitud del vector de diseño, igualando la variable propia problema,
* y creando un vector de diseño de la dimensión indicada
como argumento */
public Algoritmo(Problema problema, int longitudVectorDis ){
/* se igualan las variables propias del objeto a las variables pasadas
como argumentos */
this.problema = problema;
vectorDis = new double[longitudVectorDis];
}
/* devuelve el error actual de la minimización*/
public double devuelveError()
{return error;
}
/*devuelve el error minimo encontrado hasta el momento
en la minimización*/
```

```
public double devuelveErrorMinimo()
{return errorMinimo;
}
/*devuelve la iteración actual dentro de la minimización*/
public int devuelveStep()
{return step;
}
/*almacena el error en el vector correspondiente
que se produce en la minimización*/
protected void almacenarError()
{
/* se almacena el error primero se debe crear un objeto
de tipo Double, de valor igual al elemento a añadir */
Double errorDouble = new Double(error);
/* se añade el elemento */
vectorError.addElement(errorDouble);
}
/*devuelve el vector de error almacenado, creando un
nuevo vector unidimensional*/
public double[] evolucionError ()
{
/* se crea el vector de datos de igual tamaño que el vector de Error */
double[] datosError = new double[vectorError.size()];
/* se traspasan los datos componente a componente, realizando */
```

```
/* la transformación correspondiente a través del método
doubleValue() de la clase Double */
for (int i=0;i<datosError.length ;i++)
{datosError[i] =((Double)vectorError.elementAt(i)).doubleValue() ;
}
/* se devuelve el vector de error datosError */
return datosError;
}
/*devuelve vector de diseño actual*/
public double[] decirVectorDisActual() {
return vectorDis;
}
/*devuelve el vector de diseño encontrado que hace mínimo el error*/
public double[] decirVectorDisMinimo() {
return vectorDisMinimo;
}
/* comprueba si el error actual es menor que el minimo, en caso
a…rmativo coloca el vector de diseño actual
* como el nuevo vector de diseño encontrado con mínimo error*/
protected void comprobarNuevoMinimo() {
if(error<errorMinimo)
{errorMinimo = error;
/* se traspasan los datos componente a componente, puesto que una */
/* instrucción del tipo vectorDisMinimo = vectorDis ocasionaría */
```

```
/* un cambio de puntero, por lo que ambas etiquetas apuntarían al */
/* mismo objeto */
vectorDisMinimo = (double[])(vectorDis.clone());
} /* …n del if */
} /* …n del método */
/* cambia el valor actual del learning rate,
por el valor pasado como argumento*/
public void cambiaLearningRate (double nuevoLR)
{learningRate= nuevoLR;
}
/* devuelve el costo del vector de diseño pasado como argumento
* mediante el método decirCosto de la clase Problema*/
protected double decirCosto(double[] vectorDise)
{double evaluarFuncion = problema.decirCosto(vectorDise);
return evaluarFuncion;
}
/*…ja la variable propia de esta clase algoritmoPaso
al valor pasado como argumento del método*/
public void …jarAlgoritmoPaso(AlgoritmoOptimizacionPaso
algoritmoPaso)
{this.algoritmoPaso=algoritmoPaso;
}
/* crea un vector unitario aleatorio de la dimension del vectorDis*/
protected double[] crearVectorUnitario () {
```

```
    /* se crea un vector con la dimensión adecuada */
    double[] vectorUnitario = new double [vectorDis.length];
    /* se inicializa cada componente entre -1 y 1 */
    /* la función Math.random crea un valor aleatorio entre 0 y 1 */
    for (int i=0; i< vectorUnitario.length; i++)
    { vectorUnitario[i]= -1+2*Math.random(); }
    /* se normaliza el vector */
    algoritmoPaso.normalizarVector(vectorUnitario);
    /* y se devuelve el vector unitario */
    return vectorUnitario;
    }
    } /* ...n de la clase */
```

4.2.5 Clase AlgoritmoOptimizacionPaso

```
package myprojects.calculos;
    import java.io.*;
    /*clase de la que derivan los diferentes algoritmos
    * implementados de optimizacion de paso*/
    public class AlgoritmoOptimizacionPaso implements Serializable{
    /*algoritmo de aprendizaje del cual depende*/
    protected Algoritmo algoritmo;
    /*constructor sin argumentos*/
    public AlgoritmoOptimizacionPaso (){
    }
```

```
/* constructor que acepta como argumento el algoritmo de aprendizaje del
cual depende, igualando la variable propia algoritmo a la pasada
como argumento */
public AlgoritmoOptimizacionPaso(Algoritmo algoritmo){
/* se iguala la variable propia del objeto a la */
/* variable pasada como argumento */
this.algoritmo = algoritmo;
}
/*metodo que optimiza el paso, aceptando como argumento el vector de
cambio en el vector de diseño S, dicho vector de diseño, y el valor del
learning rate actual; valor que devuelve el metodo por defecto.*/
public double optimizaPaso(double[] S, double[] vectorDis,
double lambdaActual)
{return lambdaActual;
}
/*método que normaliza el vector pasado como argumento, manteniendo
la misma referencia en memoria, y escogiendo como norma la
raíz cuadrada de la suma de las componentes al cuadrado*/
protected static void normalizarVector (double[] vectorArgumento) {
/* variable que almacena el valor de la norma del vector, inicializada a 0;
aunque por defecto se inicializa a 0 al declararla, se inicializa explícitamente
para evitar errores de compilación */
double norma = 0;
/*para cada una de las componentes del vector pasado como argumento*/
```

```
for(int i = 0; i<vectorArgumento.length; i++)
{
/* sumar su cuadrado al valor de la norma */
norma = norma + (vectorArgumento[i])*vectorArgumento[i];
}
/* calcular la raiz cuadrada de la norma */
norma = Math.sqrt(norma);
/*y para cada una de las componentes del vector pasado como argumento*/
for (int i = 0; i < vectorArgumento.length; i++)
{
/* dividirla por el valor calculado de su norma */
vectorArgumento[i] = vectorArgumento[i]/norma;
}
} /* ...n del método */
/*método para escoger los puntos A,B,C tales que f(B) sea menor que el
valor medio de f(A) y f(C), devolviendo en un vector de parámetros
(dimensión 4) el valor de t, fA, fB y fC por este orden, y normalizando el
vector S pasado como argumento, manteniendo la misma referencia*/
protected double[] encontrarTresPuntos(double[] S, double[] vectorDis)
{
/* declaración de las variables que necesito*/
double fA,fB,fC,f1,f2,t;
/* asignación a t del valor arbitrario 0.125 */
t=0.125;
```

```
/* creación de un vector de apoyo para el cálculo del costo de los
diversos puntos */
double[] vectorPrueba= new double[vectorDis.length];
/* a …nes de ahorrar tiempo computacional , pedimos el paso en que se
encuentra el algoritmo de aprendizaje */
int step = algoritmo.devuelveStep();
/* si nos encontramos en el primer paso (todavía no ha calculado ningún
error) */
if(step == 0)
{
/* asigna a fA el valor del costo del vector de diseño pasado como
argumento */
fA = algoritmo.decirCosto(vectorDis);
}
/* en caso contrario */
else
{
/* fA es el error del paso anterior, con lo que se ahorra el cálculo de
un costo */
fA = algoritmo.devuelveError();
}
/* normalizar el vector S, sin cambiar la referencia en memoria */
normalizarVector(S);
/* se calcula el valor de x+t*S */
```

```
for (int i= 0; i<vectorDis.length; i++)
{ vectorPrueba[i] = vectorDis[i] + t * S[i];
}
/* y se asigna a f1 */
f1 = algoritmo.decirCosto(vectorPrueba);
/* algoritmo para hallar los tres puntos, asegurando que el mínimo se
encuentre en el intervalo [0,2t] */
if(f1>fA)
{do
{fC=f1;
t=t*0.5;
for (int i= 0; i<vectorDis.length; i++)
{vectorPrueba[i] = vectorDis[i] + t * S[i];
}
f1= algoritmo.decirCosto(vectorPrueba);
} while (f1>fA);
fB=f1;
} /* ...n del if */
else
{fB=f1;
for (int i= 0; i<vectorDis.length; i++)
{vectorPrueba[i] = vectorDis[i] +2*t * S[i];
}
f2= algoritmo.decirCosto(vectorPrueba);
```

```
do
{t = 2*t;
fB=f2;
for (int i= 0; i<vectorDis.length; i++)
{vectorPrueba[i] = vectorDis[i] + t * S[i];
}
f2= algoritmo.decirCosto(vectorPrueba);
} while (f2<fB);
fC=f2;
}
/* creación del vector de parámetros, con los valores calculados y en el
orden …jado */
double[] parametros = new double[4];
parametros[0] = t;
parametros[1] = fA;
parametros[2] = fB;
parametros[3] = fC;
/* devolver el vector de parámetros */
return parametros;
} /* …n del método */
} /* …n de la clase */
```

4.2.6 Clase Red

```
package myprojects.calculos;
```

```
import java.io.*;
/**clase red, de la que se derivan los diferentes tipos*/
public class Red extends ObjetoEntrenable implements Cloneable,Serializable
{
/*número de veces que se ha clonado la red*/
public int numeroGlobalClonaciones;
/*contador de iteración global*/
public int contadorRed;
/*indice etapa*/
public int indiceEtapa;
/*lista de los elementos Delay de la red*/
protected Delay[] listaDelay;
/*número de salidas de la red*/
public int numeroSalidas;
/*vector de pesos */
public double[] pesos;
/*vector de pesos iniciales*/
public double[] pesosIniciales;
/*vector de derivadas*/
public double[] vectorDerivadas;
/*vector de salida actual*/
public double[] salida;
/*vector actual de entradas*/
public double[] entrada;
```

```
/*vector de neuronas de entrada*/
protected NeuronaEntrada[] vectorNeuronasEntrada;
/*vector salidas*/
protected Salida[] vectorSalidas;
/*vector Error actual*/
public double[] vectorError;
/*elemento unidad que servirá para la implementación del término bias
para todas las neuronas*/
protected ElementoUnidad unidad = new ElementoUnidad();
/*contador del número total de pesos a ajustar*/
public int contadorPesos=0;
/*constructor sin argumentos para facilitar la extensión de esta clase*/
public Red() {
}
/*indica el valor actual de la salida correspondiente
* a la componente del vector de Neuronas de Salida
* indicada por el índice que se pasa como argumento */
public double decirSalidaActual(int indice)
{return vectorSalidas[indice].decirSalida(contadorRed,indiceEtapa);
}
/*devuelve el vector de salida de la red para el vector de
entradas pasado como argumento*/
public double[] decirSalida (double[] entradas){
/* se inicializa la variable propia entrada al valor del vector
```

```
pasado como argumento */
entrada=(double[])(entradas.clone());
/* para cada neurona de entrada */
for(int i =0; i< vectorNeuronasEntrada.length ; i++)
{
/* poner su salida al valor correspondiente de la entrada */
vectorNeuronasEntrada[i].cambiaSalida(entradas[i],indiceEtapa);}
/* se pide la salida de la red */
salida = decirSalida();
/* devolver el valor de la salida */
return salida;
}
/*devuelve el vector de entrada actual*/
public double[] decirEntrada(){
return entrada;
}
/*devuelve el vector de salida de la red para el vector de
entradas pasado como argumento*/
public void …jarEntrada (double[] entradas){
/* se inicializa la variable propia entrada al valor del
vector pasado como argumento */
entrada=(double[])(entradas.clone());
/* para cada neurona de entrada */
for(int i =0; i< vectorNeuronasEntrada.length ; i++)
```

```
{
/* poner su salida al valor correspondiente de la entrada */
vectorNeuronasEntrada[i].cambiaSalida(entradas[i],indiceEtapa);}
}
/*calcula y devuelve el valor de salida para el valor de entrada actual,
aumentando el contador global de la red, situándose en los elementos de
salida de la red, los cuales recorren la red pidiendo a los elementos
anteriores su salida*/
public double[] decirSalida (){
/* incrementar el contador global de los elementos (el mismo para
todos los elementos) */
incrementarContadorRed();
/* para cada uno de los elementos del vector de salidas */
salida = new double[vectorSalidas.length];
for(int i=0; i<vectorSalidas.length; i++)
{
/* pedir su salida */
salida[i]=vectorSalidas[i].decirSalida(contadorRed,indiceEtapa);
}
/* devolver el valor de la salida */
return salida;
}
/* renovar la red, haciendo que todos los elementos del tipo Delay
pidan la salida a sus elementos anteriores*/
```

```
public void renovarObjeto()
{
/* en caso de que haya al menos un elemento Delay */
if(listaDelay!=null)
{
/* para cada elemento del tipo Delay */
for (int i =0; i< listaDelay.length; i++)
{
/* renovar el elemento pidiendo la salida a su elemento
anterior en la red*/
listaDelay[i].renovarElemento(contadorRed,indiceEtapa);
}
}
} /* ...n del método */
/*renovar las derivadas de la red, haciendo que todos los elementos
* del tipo Delay manden su derivada
* a sus elementos anteriores en la red*/
public void renovarDerivadasObjeto()
{
/* en caso de que haya al menos un elemento Delay */
if(listaDelay!=null)
{
/* para cada elemento del tipo Delay */
for (int i =0; i< listaDelay.length; i++)
```

```
{
/* renovar la derivada mandando su derivada a su elemento
anterior en la red*/
listaDelay[i].mandaDerivadaAnteriores(indiceEtapa);
}
}
} /* …n del método */
/*devuelve la dimensión del vector de pesos*/
public int dimensionProblema() {
return contadorPesos;
}
/*…ja como pesos de la red el vector pasado como argumento, manteniendo
la dualidad de posiciones de memoria*/
public void …jarPesos (double[] nuevosPesos)
{pesos = (double[])(nuevosPesos.clone());
/*System.out.println("RED FIJADOS "+ q + ":" +
String.valueOf(pesos[0]) + "," +
String.valueOf(pesos[1]) + "," +
String.valueOf(pesos[2]) );
*/
}
/*inicializa la red para los cálculos; en este caso solamente inicializa el
vector de derivadas a 0*/
public void limpiarObjeto() {
```

```
limpiarDerivadas();
}
/*…ja los pesos iniciales*/
public void …jarPesosIniciales(double[] pesosIniciales)
{this.pesosIniciales=pesosIniciales;
pesos = pesosIniciales;
}
/*pone a 0 el vector de derivadas*/
public void limpiarDerivadas()
{if(vectorDerivadas==null)
vectorDerivadas = new double[pesos.length];
for(int i = 0; i< vectorDerivadas.length ; i++)
{vectorDerivadas[i]=0;
}
}
/*devuelve el peso correspondiente al indice pasado como argumento*/
public double decirPeso (int indice)
{return pesos[indice];
}
/*devuelve la componente del vector de derivadas
correspondiente al índice pasado como argumento*/
public double decirDerivada (int indice)
{return vectorDerivadas[indice];
}
```

```
/*devuelve el vector de pesos actual*/
public double[] decirPesos ()
{return pesos;
}
/*devuelve el vector de derivadas actual*/
public double[] decirDerivada ()
{return vectorDerivadas;
}
/*devuelve el vector de pesos iniciales*/
public double[] decirPesosIniciales ()
{return pesosIniciales;
}
/* iguala el peso correspondiente al índice del argumento
* a el valor pasado asímismo como argumento*/
public void …jarPesoParticular(int indice, double nuevoPeso)
{pesos[indice] = nuevoPeso;
}
/*iguala la componente del vector de derivadas relativa al índice
* pasado como argumento del método al valor (también pasado como
argumento del método) */
public void …jarDerivadaParticular(int indice, double nuevaDerivada)
{vectorDerivadas[indice] = nuevaDerivada;
}
/*calcula el vector de derivadas de la red a partir del vector error pasado
```

```
como argumento copiando el vector error en la variable propia de la clase*/
public void calculaDerivada (double[] vectorError)
{/*renovarDerivadasObjeto();*/
/* crea el vector propio con la dimensión adecuada */
this.vectorError = (double[])(vectorError.clone());
/* para cada neurona de salida */
for(int i = 0; i< vectorSalidas.length ; i++)
{
/* su derivada es la componente correspondiente del vector error */
vectorSalidas[i].recibeDerivada(vectorError[i],indiceEtapa);
}
}
/*devuelve el número de entradas de la red*/
public int decirNumeroEntradas()
{return vectorNeuronasEntrada.length;
}
/*devuelve el número de salidas de la red*/
public int decirNumeroSalidas()
{return vectorSalidas.length;
}
/*crea e inicializa a 0 el vector de derivadas, con una dimensión igual
* a la indicada por el valor actual de la variable contador de pesos */
public void iniciarDerivadas ()
{vectorDerivadas = new double[contadorPesos];
```

```
}
/*crea e inicializa a 0 el vector de derivadas, con una dimensión igual
* a la indicada como argumento */
public void iniciarDerivadas (int dimension)
{vectorDerivadas = new double[dimension];
}
/*incrementa el índice de etapas de los elementos de la red,
* (el mismo para todos los elementos) lo que in‡uye
* en las pilas de dichos elementos */
public void incrementarIndiceEtapas()
{indiceEtapa++;
}
/* decrementa el índice de etapas de los elementos de la red,
* (el mismo para todos los elementos) lo que in‡uye en las
pilas de dichos elementos */
public void decrementarIndiceEtapas()
{indiceEtapa–;
}
/*crea las pilas de todos los elementos de la red; para ello se sitúa
* en los elemntos de salida de la red, para posteriormente propagar
el método de creación de pilas a sus elementos anteriores*/
public void inicializarObjeto (int dimension)
{
/* para cada uno de los elemntos de salida de la red */
```

```
for(int i=0; i<vectorSalidas.length; i++)
{
/* crear la pila y propagar a sus elementos anteriores */
vectorSalidas[i].crearPila(dimension);
}
} /* ...n del método */
/* coloca el indice de etapa de las pilas de los elementos de la red a 0*/
public void inicializarObjeto()
{indiceEtapa=0;
}
/*incrementa el contador de red*/
public void incrementarContadorRed()
{contadorRed++;
}
/*decrementa el contador de red*/
public void decrementarContadorRed()
{contadorRed–;
}
/*inicializa el contador de red*/
public void inicializarContadorRed()
{contadorRed=0;
}
/* clonaciones*/
public Object clone()
```

```
{Red o = null;
o = (Red)super.clone();
o.salida = (double[])salida.clone();
o.vectorDerivadas = (double[])vectorDerivadas.clone();
o.pesosIniciales = (double[])pesosIniciales.clone();
o.pesos = (double[])pesos.clone();
o.entrada = (double[])entrada.clone();
numeroGlobalClonaciones++;
o.vectorSalidas = (Salida[])vectorSalidas.clone();
for(int i=0; i<vectorSalidas.length;i++)
{o.vectorSalidas[i] = (Salida)vectorSalidas[i].
clonar(numeroGlobalClonaciones);
}
o.vectorNeuronasEntrada = (NeuronaEntrada[])
vectorNeuronasEntrada.clone();
for(int i=0; i<vectorNeuronasEntrada.length;i++)
{o.vectorNeuronasEntrada[i] =
(NeuronaEntrada)vectorNeuronasEntrada[i].
clonar(numeroGlobalClonaciones);
}
if(listaDelay!=null)
{o.listaDelay = (Delay[])listaDelay.clone();
for(int i=0; i<listaDelay.length;i++)
{o.listaDelay[i] = (Delay)(listaDelay[i].
```

```
clonar(numeroGlobalClonaciones));
o.listaDelay[i].recogerAnterioresClonados
(numeroGlobalClonaciones,listaDelay[i]);
}
}
for(int i=0; i<vectorSalidas.length;i++)
{o.vectorSalidas[i].decirPadreAtras(o);
}
o.pesos = (double[])pesos.clone();
return o;
}
} /* ...n de la clase */
```

4.3 Manual del Usuario del Sistema de Simulación de Redes Neuronales Arti...ciales

4.3.1 Presentación del Sistema

El Sistema de Simulación de Redes Neuronales Arti...ciales tiene la presentación que muestra la Figura 4.1 de la pág. 155.

4.3.2 Ventana Principal

Al iniciar el Sistema de Simulación de Redes Neuronales Arti...ciales, la pantalla que encuentra el usuario es la que aparece en la Figura 4.2 de la pág. 156.

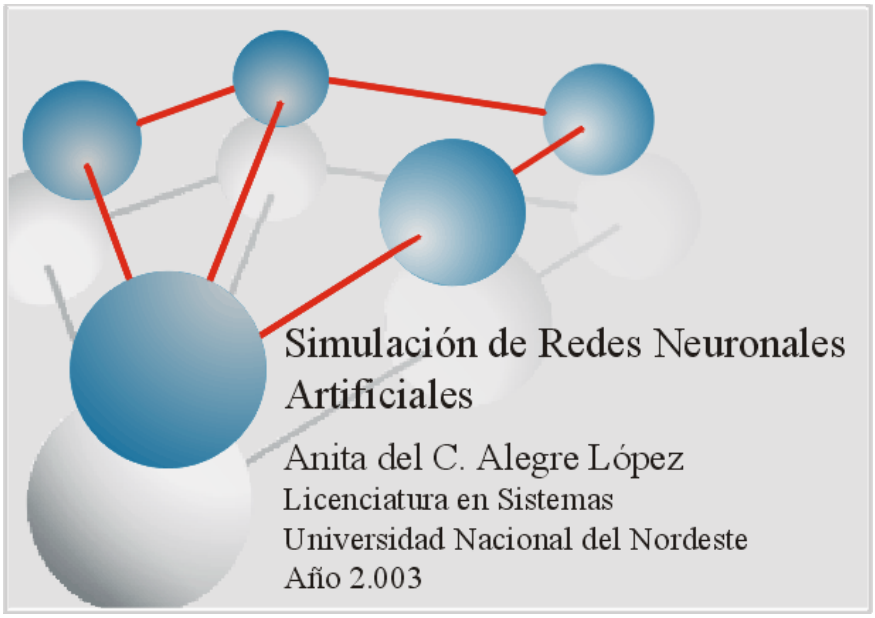


Figura 4.1: Presentación de la aplicación.

4.3.3 Barra de Menús de la Ventana Principal

Menú Archivo

- Abrir Red: muestra una ventana en la que se puede seleccionar la red que se desea abrir (ver Figura 4.3 de la pág. 157). Los datos relevantes de la red se presentan mediante el editor de texto Bloc de Notas.
- Abrir Patrones: muestra una ventana (ver Figura 4.4 de la pág. 157) con la que se puede seleccionar el ...chero o archivo de patrones desde donde leer los patrones de entrenamiento y los de validación. Por defecto, estos ...cheros tienen la extensión .txt, aunque esta circunstancia no es imprescindible. Estos ...cheros deben amoldarse al siguiente formato:

  Número de neuronas de entrada

  Número de salidas

  Número de pares de entrenamiento a utilizar

  Pares de entrenamiento, uno en cada línea, con cada componente separada por una " , ".

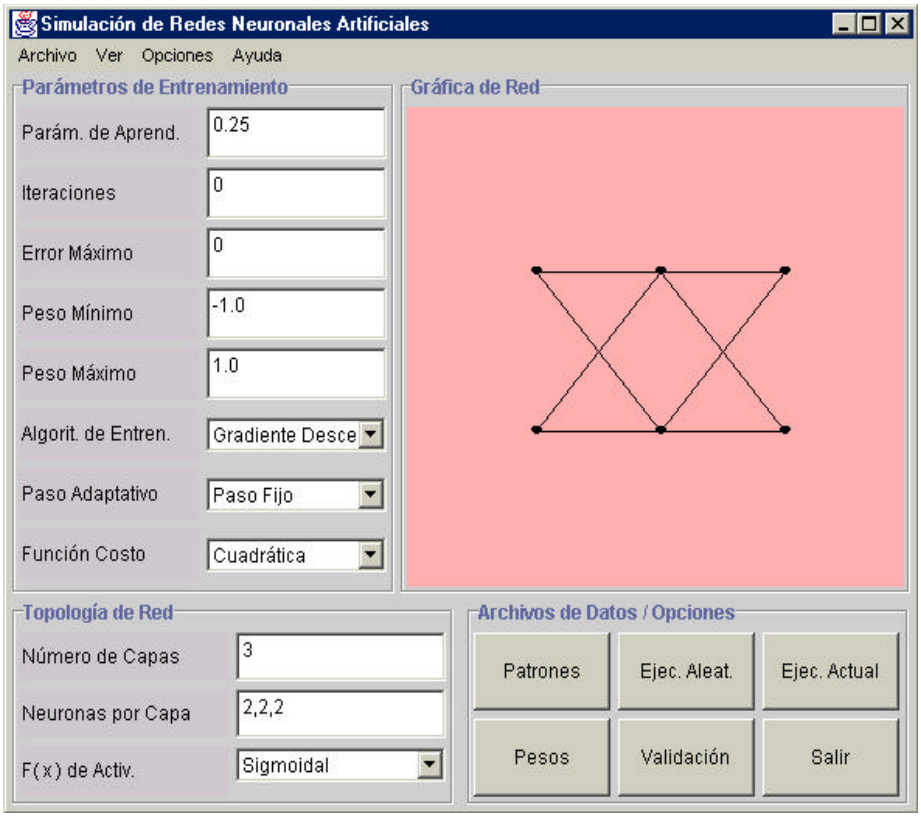


Figura 4.2: Pantalla de la aplicación.

Un ejemplo de esta estructura es la siguiente, donde se desea entrenar un Perceptrón Multicapa de 5 entradas y 3 salidas con 5 pares de entrenamiento:

5

3

5

1,0,1,1,1,1,1,1

1,1,1,0,0,1,1,0

1,0,0,1,0,1,1,0

0,0,1,0,1,1,0,0

0,1,0,1,0,1,1,0

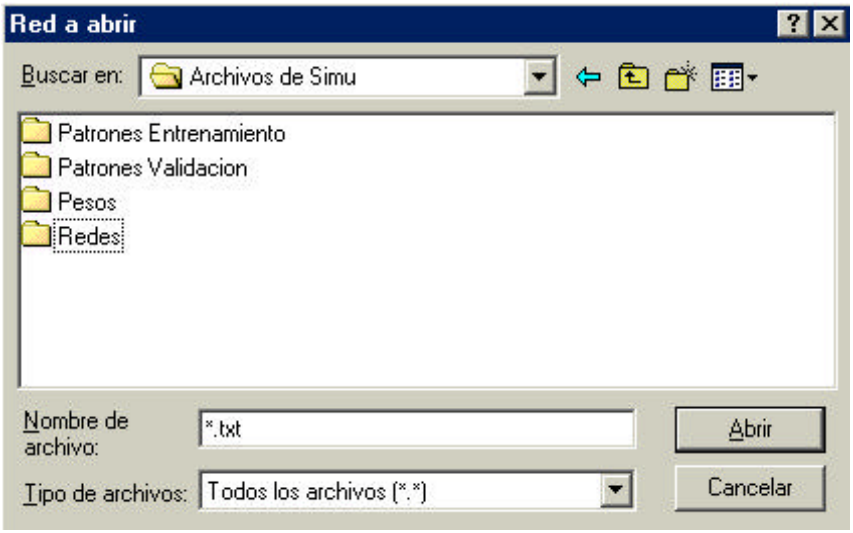


Figura 4.3: Red a abrir.

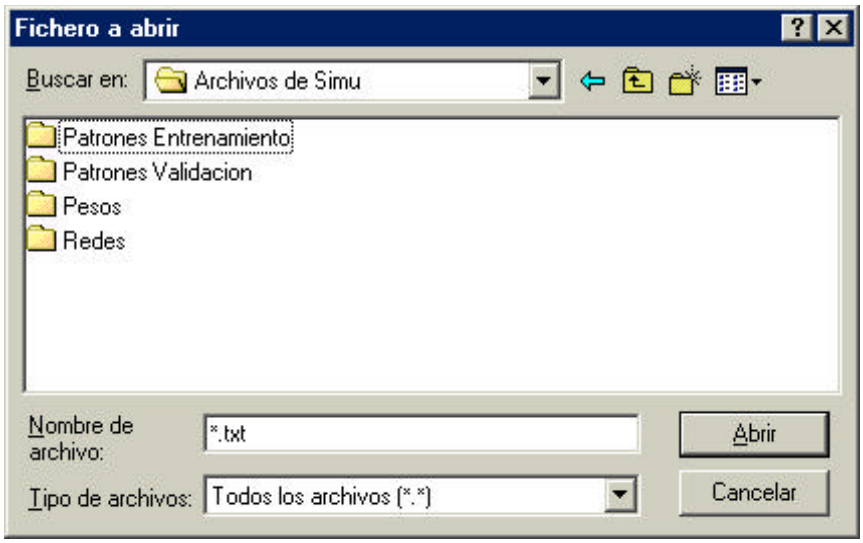


Figura 4.4: Fichero a abrir.

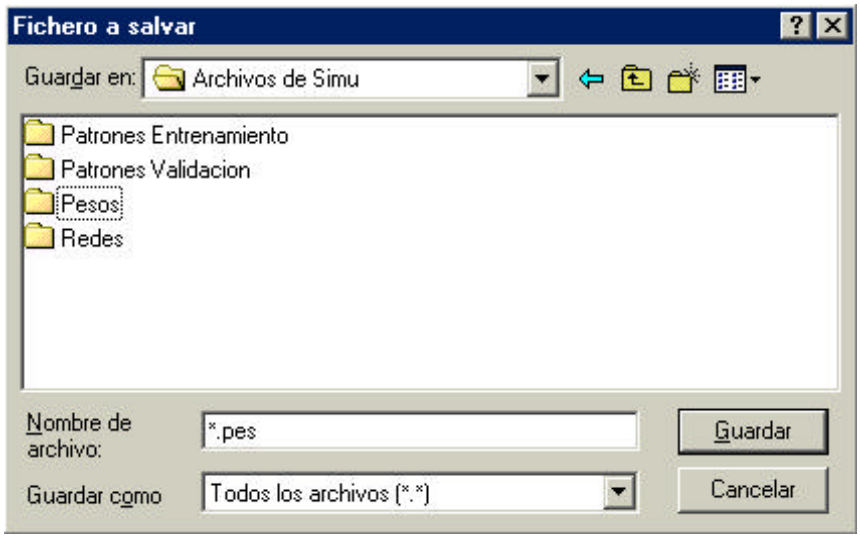


Figura 4.5: Fichero a salvar.

- Abrir Pesos: abre un cuadro de diálogo similar al de la Figura 4.4 de la pág. 157, que permite cargar los pesos, almacenados en un …chero de texto, como valor inicial del aprendizaje. El archivo tiene el siguiente orden: capa, neurona de origen de la capa anterior, neurona destino.

- Guardar Pesos: abre un cuadro de diálogo para salvar los pesos resultantes de la ejecución. Estos pesos se guardan con el mismo orden indicado en Abrir Pesos (ver Figura 4.5 de la pág. 158).

  Ctrol + P: teclas de método abreviado, para realizar rápidamente la tarea.

- Guardar Red: abre un cuadro de diálogo para salvar los datos relevantes del aprendizaje y la validación (ver Figura 4.5 de la pág. 158). El archivo que se genera es de tipo .txt, el cual podrá ser abierto mediante la opción Abrir Red, del menú Archivo.

  Ctrol + R: teclas de método abreviado, para realizar rápidamente la tarea.

- Salir: abre un cuadro de diálogo que pregunta si realmente se desea salir del sistema (ver Figura 4.6 de la pág. 159).

Figura 4.6: Salir.

Ctrol + S: teclas de método abreviado, para realizar rápidamente la tarea.

Menú Ver

- Resultados del Entrenamiento: abre un marco (ver Figura 4.10 de la pág. 165) que expone los resultados que se lograron luego del aprendizaje.
- Resultados de Validación: abre un marco que expone los resultados que se lograron en la validación (ver Figura 4.15 de la pág. 169).
- Resultados Generales: de manera similar a la opción anterior, muestra los resultados obtenidos en el entrenamiento y la validación (ver Figura 4.17 de la pág. 170).
- Vector de Errores del Entrenamiento: muestra un marco con el Vector de Errores obtenido, luego del aprendizaje (ver Figura 4.12 de la pág. 166).
- Vector de Errores de la Validación: muestra un marco con el Vector de Errores obtenido en la validación (ver Figura 4.16 de la pág. 170).
- Vector de Pesos del Entrenamiento: muestra un marco con el Vector de Pesos obtenido, luego del aprendizaje (ver Figura 4.11 de la pág. 165).
- Grá...co de Error del Entrenamiento: muestra un marco con el grá...co del error de entrenamiento obtenido en cada iteración. Dicho marco posee un menú con la opción Zoom.

  Zoom: abre un marco para seleccionar los límites del intervalo de iteraciones, permitiendo tener una vista ampliada del grá...co (ver Figura 4.14 de la pág. 167).

Ctrol + G: teclas de método abreviado, para realizar rápidamente la tarea.

- Grá...co de Error de Validación: muestra un marco con el grá...co del error de validación en cada iteración (ver Figura 4.18 de la pág. 171). Este marco también posee un menú con la opción Zoom (ver Figura 4.14 de la pág. 167).

  Ctrol + A: teclas de método abreviado, para realizar rápidamente la tarea.

- Grá...co de Error de Entrenamiento y Validación: muestra un marco con el grá...co del error de entrenamiento y el de validación (ver Figura 4.19 de la pág. 171). Este marco también posee un menú con la opción Zoom (ver Figura 4.14 de la pág. 167).

  Ctrol + X: teclas de método abreviado, para realizar rápidamente la tarea.

Menú Opciones

- Ejecutar (Pesos Aleatorios): realiza la ejecución con pesos aleatorios, sin necesidad de elegir un archivo de pesos.

- Ejecutar (Pesos Actuales): realiza la ejecución con los pesos que se eligieron en la opción Abrir Pesos del menú Archivo o mediante el botón Pesos. Esta opción se encuentra habilitada únicamente cuando tiene sentido.

  Ctrol + E: teclas de método abreviado, para realizar rápidamente la ejecución.

- Validar: realiza la Validación del Entrenamiento.

  Ctrol + V: teclas de método abreviado, para realizar rápidamente la acción.

Menú Ayuda

- Ayuda: proporciona la ayuda del sistema , en forma de archivo de texto .html (ver Figura 4.7 de la pág. 161).

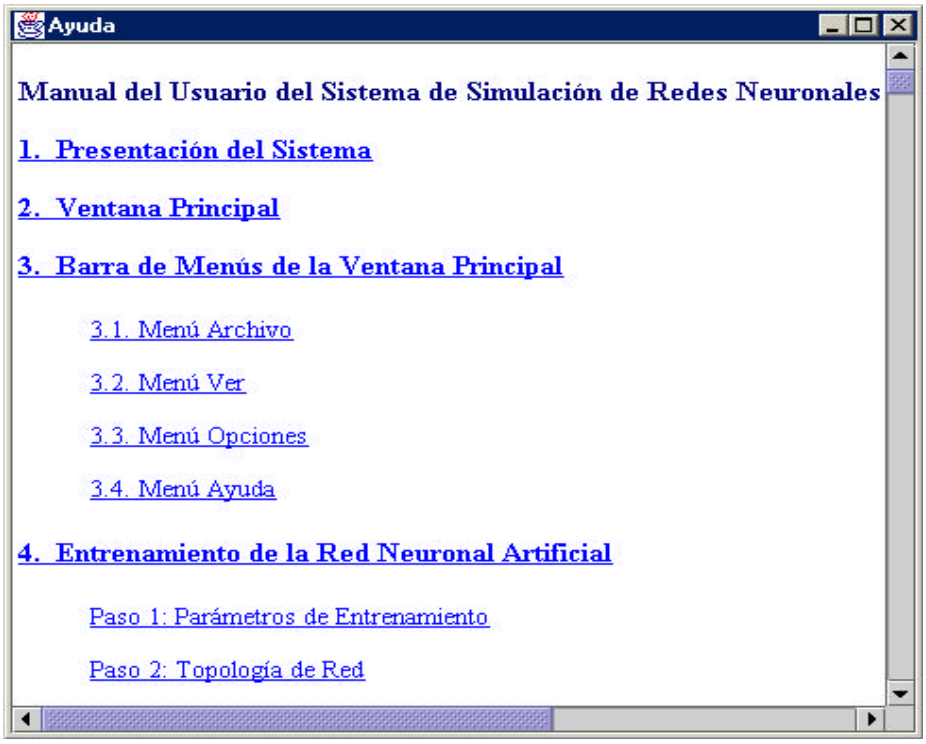


Figura 4.7: Ayuda.

- Cita: muestra un pasaje tomado de las Sagradas Escrituras (ver Figura 4.8 de la pág. 162).

- Acerca de...: proporciona los datos relativos al sistema desarrollado (ver Figura 4.1 de la pág. 155).

4.3.4 Entrenamiento de la Red Neuronal Arti...cial

A continuación se describirán brevemente los pasos necesarios para ejecutar el sistema y visualizar los resultados.

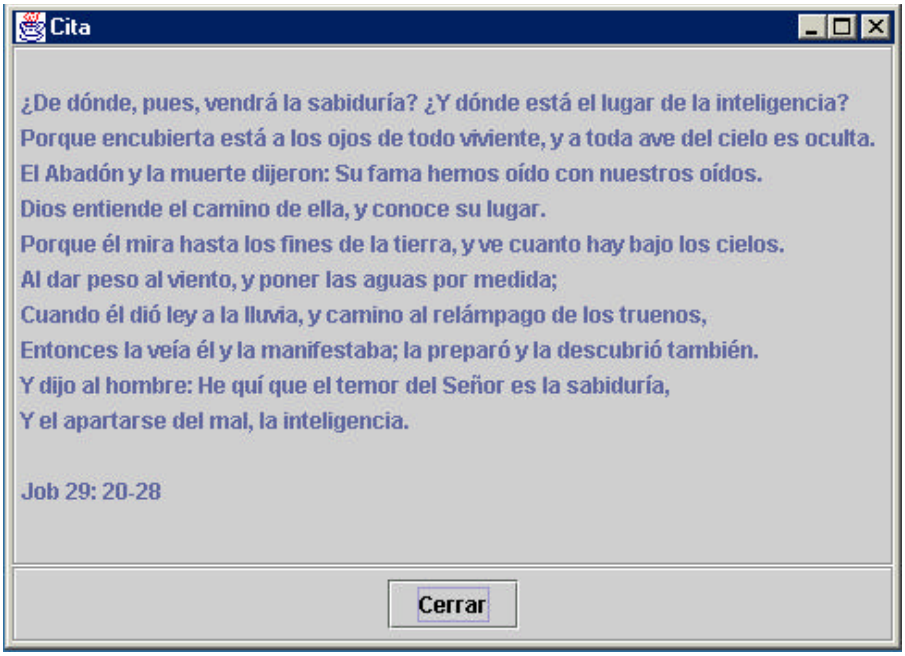


Figura 4.8: Cita.

Paso 1: Parámetros de Entrenamiento

- Parámetro de Aprendizaje: indica el parámetro de aprendizaje con el que se dará inicio al entrenamiento.
- Iteraciones: representa el número máximo de iteraciones que realizará el sistema al ejecutarse.
- Error Máximo: representa el error máximo deseado al ejecutar.
- Peso Máximo y Peso Mínimo: representan el intervalo en el cual variarán los pesos aleatorios.
- Paso Adaptativo: permite seleccionar el algoritmo que se desea aplicar para optimizar el paso durante el entrenamiento de la red. Las opciones disponibles son:

  Paso Fijo: valor …jo del paso de entrenamiento durante todo el aprendizaje.

Ajuste Cuadrático.

- Función Costo: es la función que se utilizará durante la ejecución para calcular el valor de salida. Se podrá optar por:

  Función Cuadrática.

  Función Cuadrática Ponderada: al seleccionar esta opción el programa solicita el …chero de pesos que ponderan el valor de las diferentes salidas.

Paso 2: Topología de Red

- Número de Capas: indica las cantidad de capas que tendrá la red.
- Neuronas por Capa: indica las cantidad de neuronas en cada capa, pudiendo variar en las distintas capas.
- F(x) de Activación: es la función neuronal que se utilizará durante la ejecución para calcular el valor de salida.

Grá…ca de Red

- En forma predeterminada se muestra el dibujo de una red de 3 capas con 2 neuronas por capa (ver Figura 4.9 de la pág. 164), pero al realizar la ejecución del entrenamiento se dibujará la red actual.

Paso 3: Archivos de Datos / Opciones

- Patrones: abre un cuadro de diálogo que permite seleccionar un archivo de patrones (ver Figura 4.4 de la pág. 157).
- Pesos: abre un cuadro de diálogo que permite seleccionar un archivo de pesos (ver Figura 4.4 de la pág. 157).
- Ejecución Aleat.: permite al usuario realizar la ejecución con pesos aleatorios, sin necesidad de cargar un archivo de patrones.
- Ejecución Actual (con pesos actuales): realiza la ejecución con los pesos cargados mediante la opción Abrir Pesos del menú Archivo o el botón Pesos.

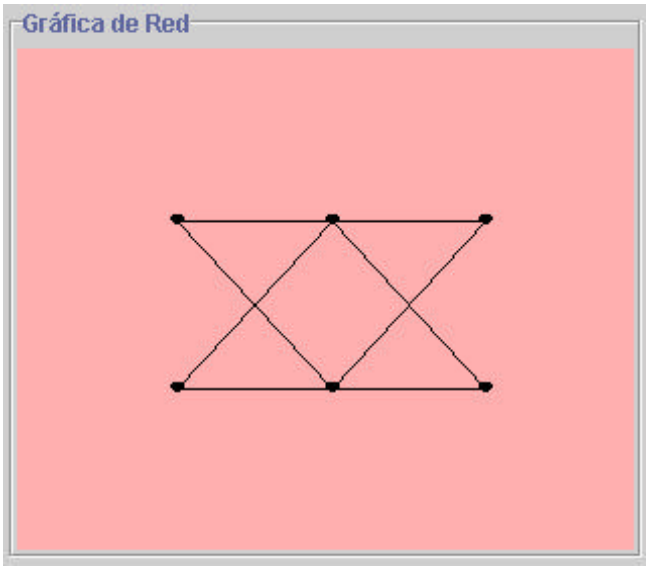


Figura 4.9: Grá…ca de la red.

Una vez establecidos los parámetros solicitados por el sistema, el usuario puede elegir entrenar la red con pesos aleatorios o a partir del conjunto de pesos actuales (provenientes de un entrenamiento anterior o cargado desde un …chero).

Al realizar el aprendizaje, con cualquiera de las dos opciones señaladas, el Sistema dibuja la red actual y muestra dos ventanas. En la primer ventana aparecen los valores relativos a los Resultados del Entrenamiento (ver Figura 4.10 de la pág. 165) y en la segunda, los valores de los Pesos del Entrenamiento (ver Figura 4.11 de la pág. 165).

En la ventana de Resultados del Entrenamiento (ver Figura 4.10 de la pág. 165), se informa el Error Obtenido, las Iteraciones alcanzadas, también sobre los instantes Inicial y Final del entrenamiento, así como el Tiempo empleado para el aprendizaje.

Por otra parte, los pesos aparecen identi…cados a través de tres índices: así $w_{[l\,i\,j]}$ corresponde al peso de la unión entre la neurona j de la capa l ¡ 1 y la neurona i de la capa l.

Figura 4.10: Resultados del Entrenamiento.

Figura 4.11: Pesos obtenidos en el Entrenamiento.

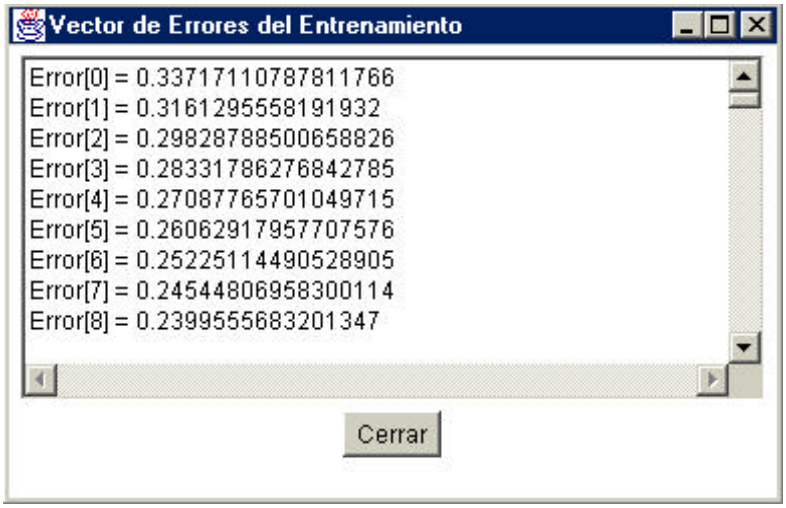


Figura 4.12: Vector de Errores del Entrenamiento.

Es necesario guardar los pesos obtenidos luego del aprendizaje, si se desea realizar la Validación, ya que son estos pesos los que deberán aplicarse en la misma.

Con la opción Vector de Errores del Entrenamiento del menú Ver, se aprecian los valores que toma el vector de error del entrenamiento, en cada iteración (ver Figura 4.12 de la pág. 166).

El Sistema ofrece la posibilidad de visualizar la grá...ca del error que se obtiene durante el entrenamiento. Para ello, basta ir al menú Ver y seleccionar la opción Grá...ca del Error de Entrenamiento o presionar las teclas del método abreviado (Ctrol + G). De esta manera aparecerá la ventana (ver Figura 4.13 de la pág. 167), en la cual el eje X representa el número de iteraciones y el eje Y representa el error en cada iteración.

Una vez representada la grá...ca, el usuario puede estar interesado en estudiar una parte concreta de la grá...ca. Para ello debe seleccionar la opción de Zoom en la barra de menús de esta ventana. Inmediatamente el Sistema presenta una ventana que permite establecer los límites (número de iteraciones) entre los cuales se desea obtener un detalle mayor (ver Figura 4.14 de la pág. 167).

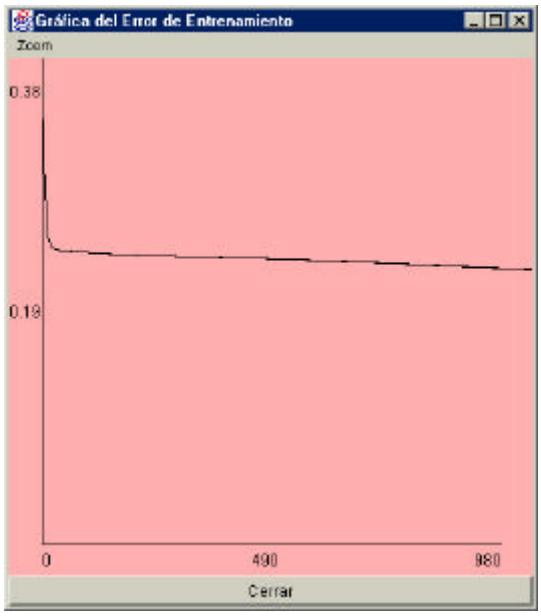


Figura 4.13: Grá...co del Error del Entrenamiento.

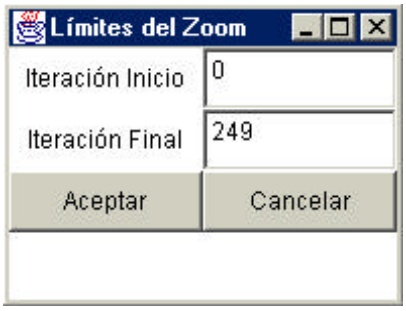


Figura 4.14: Cuadro de Diálogo: Límetes del Zoom.

Cuando se ha seleccionado estos límites, y tras pulsar sobre el botón Aceptar, se muestra la evolución del error en la zona seleccionada. La opción de Zoom se encuentra también disponible para esta grá…ca ampliada, con lo que se pueden realizar sucesivas ampliaciones.

4.3.5 Validación de la Red Neuronal Arti…cial

Paso 1: Patrones de Validación

La opción Abrir Patrones del menú Archivo o el botón Patrones, abre un cuadro de diálogo que permite seleccionar un archivo de patrones para realizar la validación. Dicho archivo de patrones debe tener un formato similar al archivo de patrones de entrenamiento. También se debe cargar el archivo de pesos, obtenido en el entrenamiento, a través del botón Pesos o la opción Abrir Pesos del menú Archivo.

Paso 2: Validación

Este paso se realiza a través de la opción Validar del menú Opciones o el botón Validación, utilizando el archivo seleccionado en el Paso 1 (los patrones de validación y los pesos obtenidos en el entrenamiento, los cuales deben ser guardados una vez …nalizado el mismo).

Luego de establecer los parámetros solicitados por el sistema, el usuario obtiene los resultados de la validación a través del marco que se muestra en la Figura 4.15 de la pág. 169.

La opción Vector de Errores de la Validación del menú Ver, permite apreciar los valores que toma el vector de error de la validación, en cada iteración (ver Figura 4.16 de la pág. 170).

La opción Resultados Generales del menú Ver permite visualizar los resultados del Entrenamiento y de la Validación (ver Figura 4.17 de la pág. 170).

El usuario puede también visualizar el grá…co del error de validación, simplemente accediendo al menú Ver y seleccionando la opción Grá…ca del Error de Validación o presionando las teclas del método abreviado (Ctrol + A). De esta manera aparece el grá…co, en la cual el eje X representa el número de

iteraciones y el eje Y el error obtenido en cada iteración (ver Figura 4.18 de la pág. 171).

Está disponible la opción Grá…ca del Error de Entrenamiento y de Validación, también en el menú Ver, en la cual se puede observar ambas grá…cas superpuestas (ver Figura 4.19 de la pág. 171).

Figura 4.15: Resultados de la Validación.

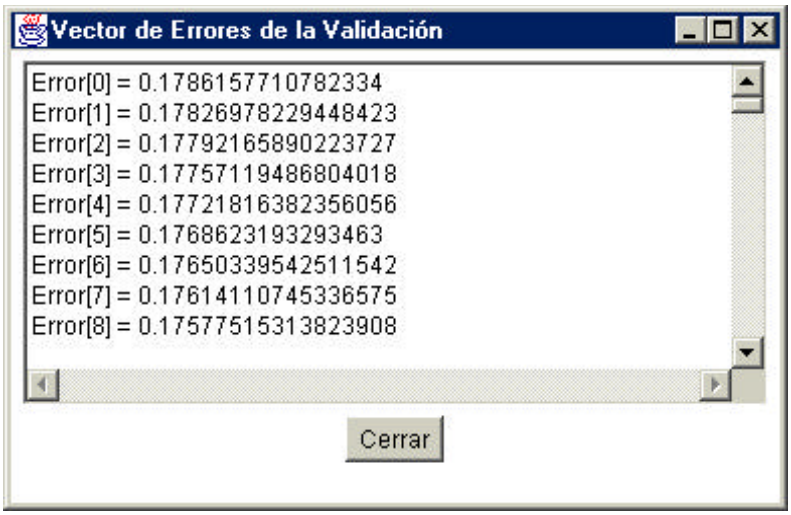


Figura 4.16: Vector de Errores de la Validación.

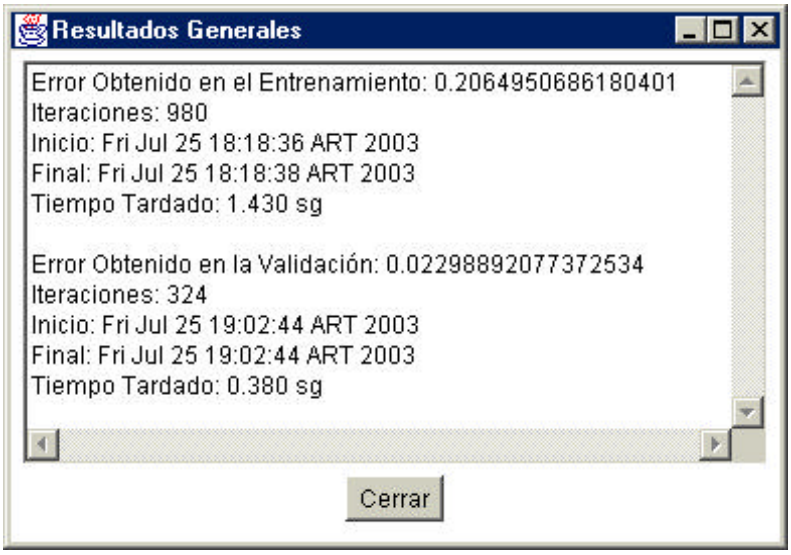


Figura 4.17: Resultados Generales.

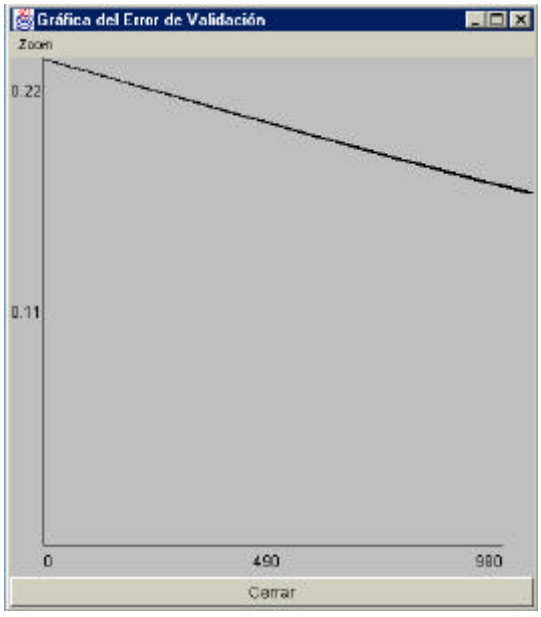


Figura 4.18: Grá…co del Error de Validación.

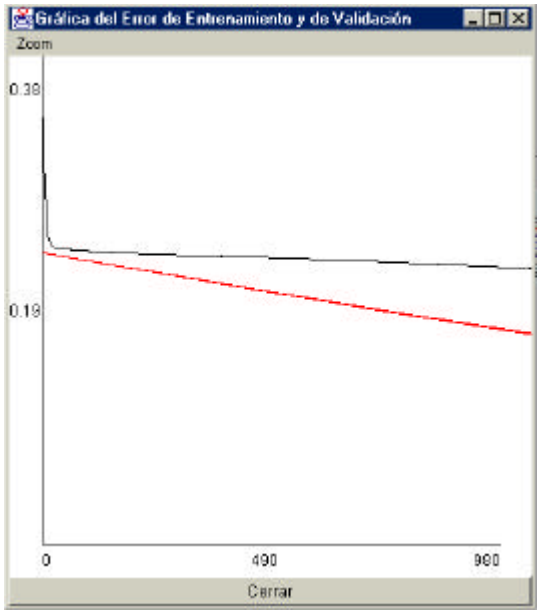


Figura 4.19: Grá…co del Error de Entrenamiento y de Validación.

4.4 Análisis de Resultados

El único objetivo de este análisis es comprobar, a través de la Validación, si existe un problema de sobreajuste en el modelo del proceso de Entrenamiento de la red neuronal arti…cial.

Se utiliza una red de 3 capas con 2 neuronas por capa. Los patrones de entrenamiento están contenidos en el archivo Carmen.txt, mientras que los de validación están en CarmenVal2.txt. El tipo de aprendizaje es el Gradiente Descendente y la función costo es la Cuadrática. El parámetro de aprendizaje a aplicar es igual a 0.25 y los pesos (en todos los casos aleatorios) varían entre 1 y -1. Resulta obvio mencionar que la Validación se realiza con los pesos obtenidos en el Entrenamiento.

Parámetros de Entrenamiento: archivo Carmen.txt:

```
2
2
5
1  0:5  0:7  0:3
1  0    0:2  0:8
1  0:2  0:8  0:2
0  1    0:4  0:6
0  0:5  0:5  0:5
```

Parámetros de Validación: archivo CarmenVal2.txt:

```
2
2
5
1  0:9  0:6  0:5
1  0    0:2  0:8
1  0:2  0:8  0:2
1  0    0:3  1
0  0:5  0:5  0:5
```

Los archivos utilizados (patrones de entrenamiento y de validación), y los producidos por el sistema (pesos y redes) se encuentran disponibles en el cd adjunto.

La Tabla 4.1 de la pág. 173 expone los datos obtenidos en el Entrenamiento aplicando el algoritmo de optimización Paso Fijo, con la función de activación

Sigmoidal, y a continuación la Tabla 4.2 de la pág. 174 muestra la Validación de los datos.

Red	Error	Iteración
red15075	0.22639332587115513	450
red1707	0.20598687596912865	780
red21072	0.22564399702849844	125
red21073	0.1942932642043669	965
red21074	0.223027315011076	385
red21075	0.20762687411327424	621
red21076	0.043781614915215754	520
red21077	0.22162343342370294	520
red21078	0.21704279746216232	854
red21071	0.18804868693480029	640

Tabla 4.1: Entrenamiento: Algoritmo de Optimización Paso Fijo y Función de Activación Sigmoidal.

Para continuar con la exposición de los datos, la Tabla 4.3 de la pág. 174 muestra los obtenidos en el Entrenamiento utilizando el algoritmo de optimización Paso Fijo, con la función de activación Tangente Hiperbólica. De la misma forma, en la Tabla 4.4 de la pág. 175 se observa la Validación.

Ya presentados los datos obtenidos con el algoritmo de optimización Paso Fijo, se hace de manera similar con el algoritmo Ajuste Parabólico. En primer término se aprecia su aplicación en el Entrenamiento, con la función de activación Sigmoidal (ver Tabla 4.5 de la pág. 175), y la Validación (ver Tabla 4.6 de la pág. 176).

Para …nalizar, se exponen los datos arrojados con el algoritmo Ajuste Parabólico aplicando la función Tangente Hiperbólica (ver Tabla 4.7 de la pág. 176) y seguidamente la Validación respectiva en la Tabla 4.8 de la pág. 177.

En la Tabla 4.9 de la pág. 177 y la Tabla 4.10 de la pág. 177, se observa el comportamiento de las Iteraciones del Entrenamiento (IR) y de las de Validación (IV), en relación al número de ejecuciones (Ejec.), tanto en el Algoritmo de Optimización Paso Fijo como en el Ajuste Parabólico respectivamente.

Red	Error	Iteración
red15075	0.22264686686751478	450
red1707	0.19249646975312204	780
red21072	0.22290700107971195	125
red21073	0.1636498839056296	965
red21074	0.21038714556714372	385
red21075	0.1669453436092714	621
red21076	0.037957742582392345	520
red21077	0.1874930651941517	520
red21078	0.19621608280101202	854
red21071	0.07657331577799532	640

Tabla 4.2: Validación: Algoritmo de Optimización Paso Fijo y Función de Activación Sigmoidal.

Red	Error	Iteración
red1507	0.1436435818025859	450
red15072	0.13621128212078015	450
red15073	0.14876157211695057	400
red15074	0.029605667854396656	780
red15076	0.09795667261958775	389
red15077	0.13066576077731012	450
red15078	0.086413723095369	502
red15079	0.18027679665619722	456
red1807	0.05386127712350889	287
red18074	0.055381057087061016	250

Tabla 4.3: Entrenamiento: Algoritmo de Optimización Paso Fijo y Función de Activación Tangente Hiperbólica.

Red	Error	Iteración
red1507	0.0902449628120662	450
red15072	0.0291199886238059	450
red15073	0.0924977955252413	400
red15074	0.009771234577499972	436
red15076	0.09708218783243741	1
red15077	0.0869352239370872	96
red15078	0.08567911967813366	2
red15079	0.1692132557150713	456
red1807	0.053842991209309715	275
red18074	0.05368944993102041	17

Tabla 4.4: Validación: Algoritmo de Optimización Paso Fijo y Función de Activación Tangente Hiperbólica.

Red	Error	Iteración
red16074	0.13378968975696262	690
red16079	0.05396243511411047	316
red22072	0.04699203432743898	282
red22073	0.04105175035490719	999
red22077	0.0879208721123809	236
red22078	0.0867372923242541	57
red22079	0.08482777221783726	461
red220710	0.13197595349122754	321
red220718	0.0967839814105256	175
red220719	0.09707642960976834	163

Tabla 4.5: Entrenamiento: Algoritmo de Optimización Ajuste Parabólico y Función de Activación Sigmoidal.

Red	Error	Iteración
red16074	0.08120315798855943	1
red16079	0.05396122299118757	81
red22072	0.04690595760968152	64
red22073	0.021897701060722214	999
red22077	0.08730059994026809	65
red22078	0.08690218960665153	1
red22079	0.0848099166485239	181
red220710	0.08384256631208872	158
red220718	0.0951123734495764	2
red220719	0.09634986444922408	54

Tabla 4.6: Validación: Algoritmo de Optimización Ajuste Parabólico y Función de Activación Sigmoidal.

Red	Error	Iteración
red16073	0.2111390827551419	630
red2207	0.10641190672834319	265
red22071	0.08339922197213752	689
red22074	0.11357945406184064	350
red22075	0.08507630070856342	580
red22076	0.10183285167849358	1000
red220714	0.10335104512081918	354
red220716	0.13707957165529058	1000
red220717	0.09796453071829127	720
red220720	0.09232843980390357	900

Tabla 4.7: Entrenamiento: Algoritmo de Optimización Ajuste Parabólico y Función de Activación Tangente Hiperbólica.

Red	Error	Iteración
red16073	0.13769134225601978	630
red2207	0.0960744444060038	265
red22071	0.07808612169530886	689
red22074	0.08648782868455775	350
red22075	0.08656114026325604	580
red22076	0.08656114026325604	580
red220714	0.09795677965765814	185
red220716	0.09789502234591263	614
red220717	0.09791614194452765	2
red220720	0.0887387675431682	900

Tabla 4.8: Validación: Algoritmo de Optimización Ajuste Parabólico y Función de Activación Tangente Hiperbólica.

F(x) de Activ.	Ejec. donde IV < IE	Ejec. donde IV = IE	Tot.Ejec.
Sigmoidal	10	0	10
Tang. Hiperb.	6	4	10

Tabla 4.9: Algoritmo de Optimización Paso Fijo.

F(x) de Activ.	Ejec. donde IV < IE	Ejec. donde IV = IE	Tot.Ejec.
Sigmoidal	9	1	10
Tang. Hiperb.	4	6	10

Tabla 4.10: Algoritmo de Optimización Ajuste Parabólico.

4.4.1 Interpretación de los datos

1. Algoritmo de Optimización Paso Fijo y Función de Activación Sigmoidal (ver Tabla 4.1 de la pág. 173 y Tabla 4.2 de la pág. 174).

   En el 100% de las ejecuciones se observa que el Error de Validación es menor que el Error de Entrenamiento. Se puede decir también que en el 100% de las ejecuciones, el número de iteraciones de la Validación resulta ser igual al número de iteraciones del Entrenamiento.

2. Algoritmo de Optimización Paso Fijo y Función de Activación Tangente Hiperbólica (ver Tabla 4.3 de la pág. 174 y Tabla 4.4 de la pág. 175).

   En este caso se observa que en el 100% de las ejecuciones, el Error de Validación es menor que el Error de Entrenamiento. En el 40% de las ejecuciones, el número de iteraciones de la Validación es igual al número de iteraciones del Entrenamiento, mientras que el 60% restante es menor.

3. Algoritmo de Optimización Ajuste Parabólico y Función de Activación Sigmoidal (ver Tabla 4.5 de la pág. 175 y Tabla 4.6 de la pág. 176).

   Los datos arrojados para este caso revelan que en el 100% de las ejecuciones, el Error de Validación es menor que el Error de Entrenamiento. En el 90% de las ejecuciones, el número de iteraciones de la Validación resulta ser menor al número de iteraciones del Entrenamiento, y en el 10% igual.

4. Algoritmo de Optimización Ajuste Parabólico y Función de Activación Tangente Hiperbólica (ver Tabla 4.7 de la pág. 176 y Tabla 4.8 de la pág. 177).

   Para …nalizar, los datos de este modelo arrojan que en el 100% de las ejecuciones, el Error de Validación también es menor que el Error de Entrenamiento. En este caso, en el 40% de las ejecuciones el número de iteraciones de la Validación es menor al número de iteraciones del Entrenamiento, siendo el 60% igual.

Se realizaron 10 ejecuciones en cada modelo, dando un total de 40 ejecuciones. Esto reveló que para los archivos considerados, los tiempos de Entrenamiento y los de Validación se presentan en márgenes aceptables, variando entre menos de 1 y 10 segundos, siendo el tiempo de Validación menor que el de Entrenamiento en el 100% de las ejecuciones. Además se observa que en el 100% de las ejecuciones el Error de Validación es menor que el Error de

Entrenamiento, lo que demuestra que estos modelos no presentan un problema de sobreajuste en el proceso de Entrenamiento.

Capítulo 5

Conclusiones

Las RNAs se aplican a un creciente número de problemas reales de considerable complejidad, como el reconocimiento de patrones, la clasi…cación de datos, etc. Su ventaja más importante está en su potencial de cálculo para solucionar problemas que son demasiado complejos para las técnicas convencionales: problemas que no tienen un algoritmo especí…co para su solución, o cuyo algoritmo es demasiado complejo para ser encontrado. En general, las Redes Neuronales Arti…ciales han sido claramente aceptadas como nuevos sistemas muy e…caces para el tratamiento de la información en muchas disciplinas. Ello ha dado como resultado su aplicación en el Tratamiento de textos y proceso de formas, Ciencia e Ingeniería, Medicina y Salud, Transporte y Comunicaciones, Finanzas, Negocios, etc.

En otro orden de cosas, el lenguaje orientado a objetos Java, demuestra ser un lenguaje interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones y multihilo. Java es un lenguaje total, justa y verdaderamente apto para realizar este Trabajo Final de Aplicación.

Para la preparación de los documentos del presente Trabajo, se ha utilizado LaTeX por ser un sistema de composición profesional que logra una alta calidad de edición.

El Sistema de Simulación de Redes Neuronales Arti…ciales, puede ser aplicado sin inconvenientes en cualquier disciplina, debido a su principal característica: la generalidad.

El análisis de los resultados arrojados por la aplicación revela que los mo-

delos implementados no presentan un problema de sobreajuste en el proceso de Entrenamiento y que los tiempos de ejecución son aceptables para los archivos considerados.

En el futuro se tiene previsto la implementación de otras arquitecturas de redes, como las de Hop…eld o las Competitivas.

Para …nalizar, es necesario mencionar que los objetivos propuestos al inicio del Trabajo Final de Aplicación han sido alcanzados y esto produjo una gran satisfacción personal, por el carácter de desafío que tomaron.

Estamos viviendo tiempos de cambios y adelantos cientícos como nunca antes. Este tiempo es el profetizado por la palabra de Dios, la cual dice que "Muchos correrán de aquí para allá y la ciencia se aumentará" (Dn. 12: 4) y "Profesando ser sabios, se hicieron necios" (Ro. 1: 22). Considerando esto, se puede pensar que la idea de imitar el comportamiento humano en las Redes Neuronales Arti…ciales resulta soberbia. Pues entonces, que nuestro Padre celestial proporcione a la humanidad sabiduría para aplicar todos los adelantos cientí…cos y tecnológicos logrados hasta hoy, y los que han de lograrse.

Bibliografía

[1] La Santa Biblia - Revisión de 1960. Sociedades Bíblicas Unidas, D.F.-México, 1991.

[2] L. Joyanes Aguilar. Programación Orientada a Objetos - Segunda Edición. Mc Graw Hill/Interamericana de España, S.A.U.-España, 1998.

[3] M. Azmoodeh. Abstract Data Types and Algorithms. 2nd Ed. MacMillan, 1990.

[4] G. Brassard; P. Bratley. Fundamentos de Algoritmia. Prentice Hall, Madrid-España, 1997.

[5] A. C. Esteban; N. M. Carrascal. Reconocimiento de voz mediante MLP. http://intersaint.org/acid/rupm, 2002.

[6] E. Castillo. Curso de LaTex. Universidad de Cantabria, Cantabria-España, 1998.

[7] Universidad de La Laguna. Técnica del Grafo de Flujo de Señal aplicada a las Redes Neuronales. http://www.evenet.cyc.ull.es/introduccion.htm, España, 2001.

[8] Universidad de las Américas. Ejemplo de un Perceptrón desarrollado en Java. http://udlap.mx/is108851/REDESNEURO/Perceptron, Puebla-México, 2003.

[9] B. Eckel. Thinking in Java. Prentice-Hall, 1998.

[10] J. García; J. Rodriguez; I. Mingo; A. Imaz; A. Brazalez; A. Larzabal; J. Calleja; J. García. Aprenda Java como si estuviera en primero. Universidad de Navarra, San Sebastián-España, 1999.

[11] C. Gershenson. Algoritmos de Redes Neuronales desarrollados en Java. http://student.vub.ac.be/cgershen, Free University of Brussels, 2001.

[12] Sun Microsystems. The Java Language version 1.1.4. http://java.sun.com, 1998.

[13] W. McCulloch; W. Pitts. A Logical Calculus of Ideas Inminent in Nervous Activity. 1943.

[14] E. Castillo; A. Cobo; J. M. Gutiérrez; R. E. Pruneda. Introducción a las Redes Funcionales con Aplicaciones - Un Nuevo Paradigma Neuronal. Paraninfo, España, 1999.

[15] F. Rosenblatt. Principles of Neurodynamics. Science Editions, New York, 1962.

[16] F. J. Ceballos Sierra. Java 2 Curso de Programación. Editorial Ra-ma, Madrid-España, 2000.

Índice de Materias