CAPITULO IV

4. ANÁLISIS Y APLICACIÓN DE UN MODELO DE REDES NEURONALES UTILIZANDO UN SOFTWARE PARA RESOLVER EL PROBLEMA DE CARTERA VENCIDA.

4.1 Introducción

En este último capitulo procederemos a plantear en específico y resolver el problema de neuroclasificación de los datos obtenidos de Bellsouth al someterlos a un proceso de aprendizaje de un modelo de redes neuronales previamente diseñado y por medio de un paquete computacional obtendremos un resultado esperando que este sea positivo. Así, al final de este capítulo podremos ya tener un proceso neuro-discriminatorio artificial para diferenciar posibles sujetos de crédito de los que probablemente no lo sean.

4.2 Sobre los datos

Si observamos la Hoja de Administración de Riesgo del capitulo 1, podremos saber que obviamente la información que vamos a utilizar proviene única y exclusivamente de ahí. Hemos tomado en cuenta sólo 6 variables cuyo único resultado puede ser: "El cliente aparece eventualmente en mora o no".

Las variables que utilizamos son: $\mathbf{i_1}$: Edad (entero), $\mathbf{i_2}$: Estado Civil (boolean), $\mathbf{i_3}$: Casa Propia / alquilada (boolean), $\mathbf{i_4}$: Trabajo Empleado-Operativo / Ejecutivo-Propietario (boolean), $\mathbf{i_5}$: Años en el último trabajo (entero), $\mathbf{i_6}$: Pago con Débito-Cuenta / Débito-Tarjeta (boolean). La variable de resultado es $\mathbf{d_1}$: Aparece eventualmente en mora por más de un mes (boolean).

Estos datos constituyen una muestra de aquellos clientes cuyos planes tarifarios son cuentas individuales de Bellsouth guayaquil del período julio 2000 – julio 2001. El total de la muestra es $n = 100$ sujetos. No se toma en cuenta ningún caso de planes de prepago por obvias razones.

La muestra es tan reducida debido a lo confidencial de los datos y sobretodo a la mala calidad de la información. En todos los casos no había mas que papeles (cero registros digitalizados) y en la mayoría había datos incompletos o ilegibles; también debemos tener en cuenta que es muy probable que exista algún tipo de sesgo.

Evidente es que no tomamos en cuenta la variable “ingresos” (que hubiera sido de suprema importancia) ya que nunca este campo fue llenado para caso alguno. El resto de variables fueron descartadas por su poca influencia dentro del modelo (i.e. nombre, apellido, teléfono, dirección, nombre del cónyuge, etc.).

Aquellos registros donde aparecen ceros en el campo “deuda” significan que de lo que lleva el cliente al cabo de ese año, este no aparece con deudas por mora (más allá del mes) dentro de los registros de Bellsouth.

Cabe mencionar que esta es la única información digitalizada que se posee. En esta parte hay que tomar en cuenta lo complicado que se torna el obtener esta información por lo delicado de su contenido.

Hemos dividido a la mitad el conjunto de observaciones de tal forma que el primer subconjunto nos sirva para entrenar la red y la otra mitad para efectos de prueba. En los anexos podemos encontrar esta información.

La información originalmente fue digitada en MSExcel pero los subconjuntos necesariamente fueron importados a dos archivos de texto distintos (*.txt) para poder utilizarlos en el programa que escogimos.

4.3 Acerca del Software utilizado

Después de una ardua búsqueda por algún “Trial version” software de redes neuronales (ya que casi la totalidad existente requiere de licencia y esta es muy costosa) optamos por emplear el programa

ThinksPro–Neural Networks for Windows Trial Version 1.05 de ***Logical Designs Consulting***.

Este programa de 1995, por antiguo que parezca presta todos los requerimientos necesarios para llevar a cabo nuestro trabajo. Ofrece las facilidades de crear nuevos proyectos, crear redes, editarlas, parametrizarlas, medir el error al iterar, ofrecer y exportar los resultados, etc.

Vamos nosotros a mencionar paso a paso el procedimiento que seguimos para ejecutar el entrenamiento de aquella red neuronal que creamos para hacer el reconocimiento de los patrones crediticios.

4.4 Proceso de Aplicación del modelo sobre los datos

Los conjuntos de datos tienen por denominación prueba11.txt y prueba22.txt y se pueden obtener de los anexos.

4.4.1 Creación de la red

Para crear una red neuronal nueva primero creamos un "proyecto nuevo" tal y como crearíamos un documento nuevo en un procesador de palabras. La primera tarea a cumplir será diseñar y parametrizar una estructura neuro-artificial que concuerde con la lógica de nuestro problema y que utilice la regla de aprendizaje que hemos estudiado.

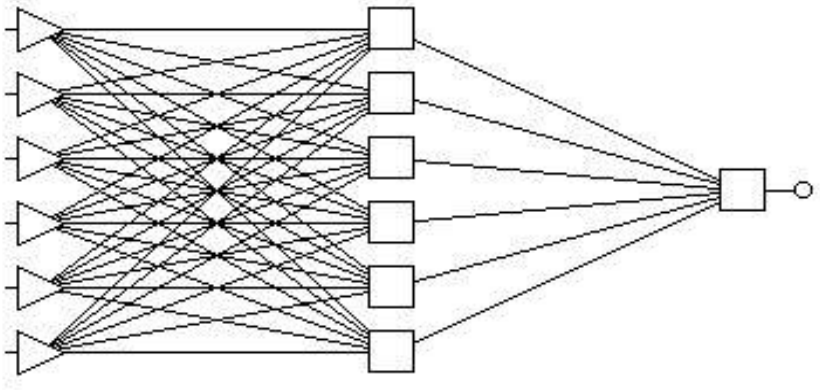


Figura 4.1 Estructura de la red neuronal que resuelve el problema

Como podemos observar en la figura 4.1, esa es la estructura o "esqueleto" de la red que utilizaremos: 3 capas, una de entrada con 6 nodos, una intermedia también con 6 nodos y una de salida con un único nodo. Todos los nodos de la capa

anterior se interconectan con todos los nodos de la capa posterior, como es usual en Back-Propagation.

La simple lógica nos explica por qué tenemos 6 nodos en la capa de entrada y 1 en la capa de salida: cada nodo de la capa de entrada representará la entrada de cada variable de la hoja de riesgo, esto es $i_1, i_2, \ldots, i_6$; y el nodo de la capa de salida representa el resultado del análisis crediticio d_1. El número de capas ocultas y nodos en la misma provienen de razones un tanto distintas.

Por experiencia de anteriores trabajos de distintos autores y distintos archivos de ayuda de muchos programas, podemos decir que en general, si el “tamaño del problema” es pequeño entonces no es necesario sobredimensionar la red con más nodos ni capas ocultas de lo proporcional al resto de la red.

En todo caso consideramos que el número de capas y nodos en la sección intermedia de la red que hemos utilizado en este

problema han sido el máximo posible basados en la experiencia de trabajos de distintos autores.

Después de determinar la estructura procedemos a escoger la regla de aprendizaje de la red y sus parámetros. De la hoja de “Network Setup” escogemos “back-propagation” para nuestra regla de aprendizaje.

Como la capa de entrada sirve únicamente como “input buffer” entonces la única parametrización en esa capa es el preprocesamiento de entrada como MAX/MIN, esto sirve para transformar los datos hacia un intervalo decimal fijo considerando los valores máximos y mínimos de cada variable y recodificándolos al hacerlos variar dentro de esos intervalos. Esto se hace debido a que presta facilidad a los cálculos y se acomoda mejor a la función de transferencia que es umbraloide en la capa oculta.

La figura 4.2 muestra el diseño de la capa de entrada. En la capa oculta procedemos a parametrizarla conforme lo antes establecido. Tenemos 6 nodos, utilizamos una suma ponderada (en la figura se ve: dot product) y una función de transferencia sigmoide (establece una especie de umbral 0-1).

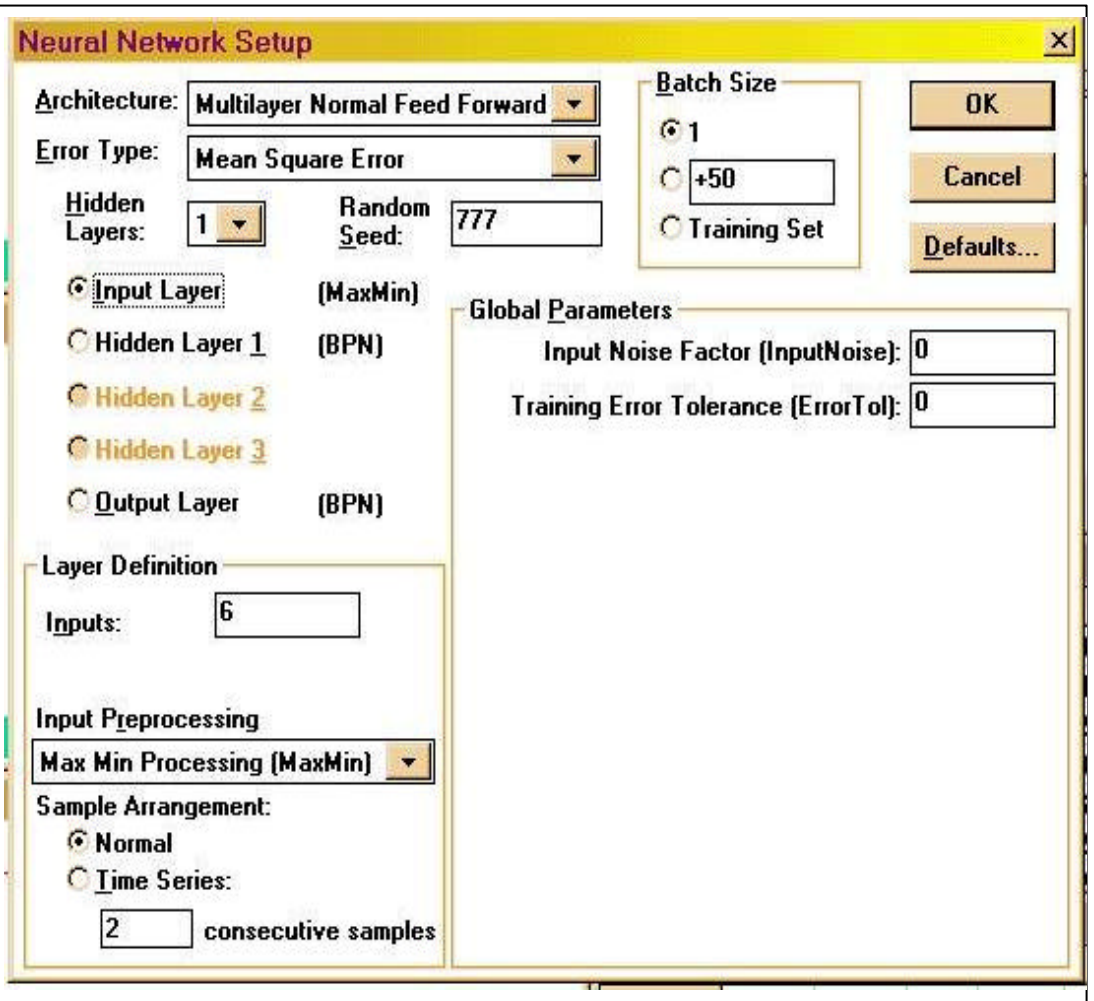


Figura 4.2 Parametrización en la capa de entrada

En la figura 4.3 vemos además que las tasas de velocidad de aprendizaje y de momento antes discutidas se han elegido con valores de 0.09 y 0.07 respectivamente.

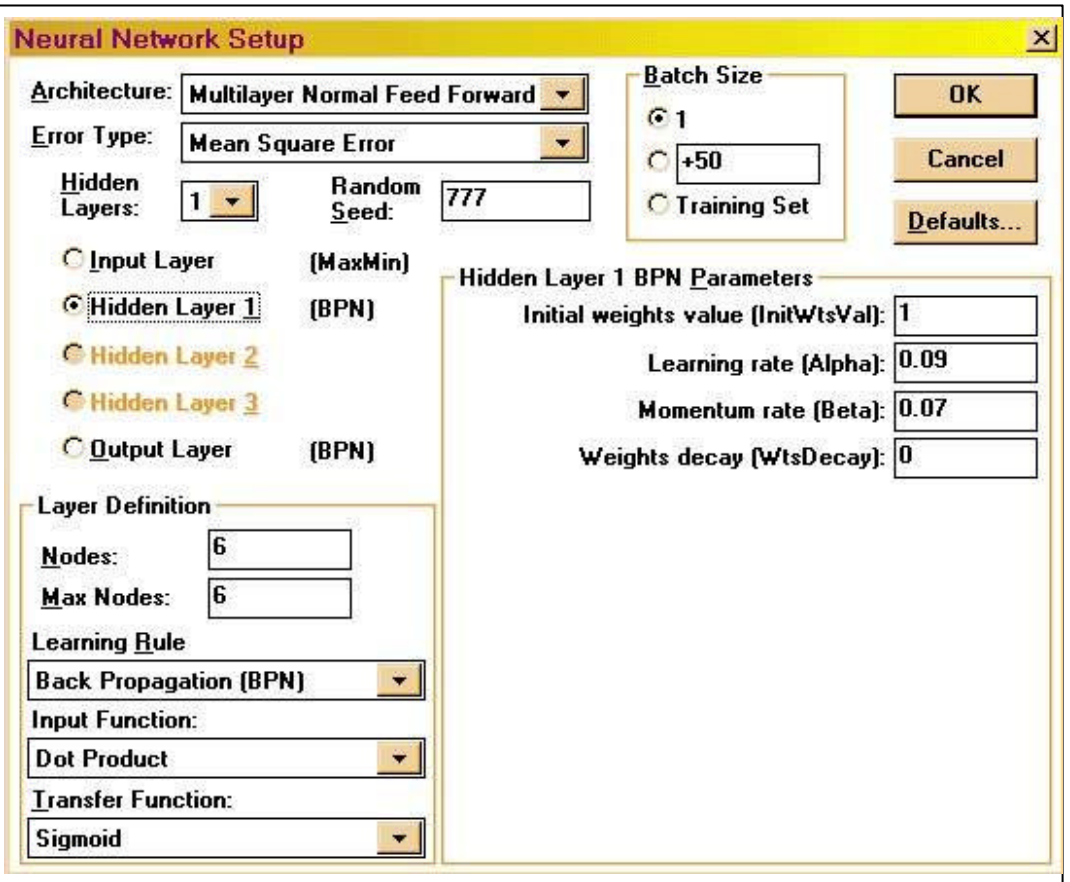


Figura 4.3 Parametrización de la red en la capa oculta

Estos valores fueron elegidos conforme hicimos algunos ensayos hasta encontrarlos como los más adecuados al equilibrar la velocidad del aprendizaje y el riesgo de caer en mínimos locales.

En el primer intento por elegir una tasa de aprendizaje optamos por 0.01 y probamos que era una tasa muy baja ya que el aprendizaje era muy lento.

La mayor parte del aprendizaje la hace la capa oculta en este caso debido al mayor número de nodos que esta tiene con respecto a la capa de salida.

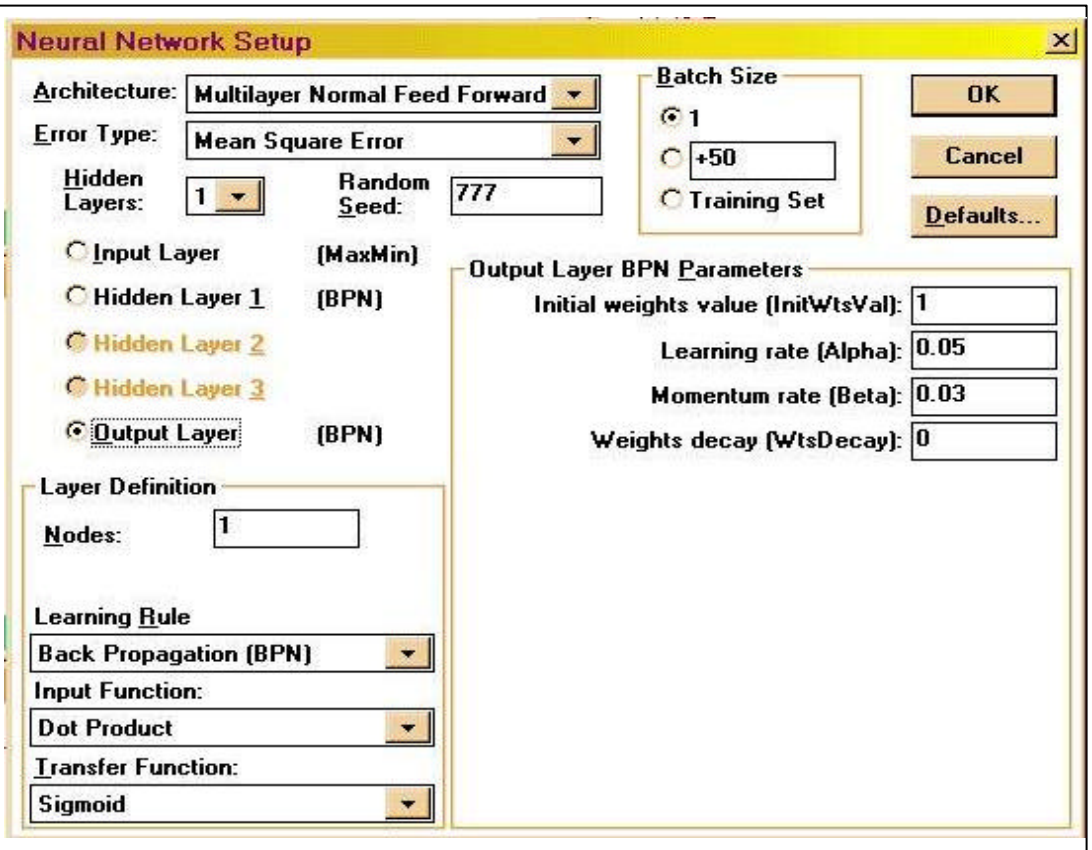


Figura 4.4 Parametrización de la red en la capa de salida

En la figura 4.4 observamos la parametrización de la capa de salida. Se ve un único nodo cuya función de resultado o transferencia también es una sigmoide. Nótese que en cada capa se puede elegir la regla y parámetros de aprendizaje independientemente de las otras capas, esto es de cierto modo una característica única del programa que hemos utilizado ya que la mayoría no permite tanta interactividad con el diseño de las redes.

Esto se debe a que los creadores de los programas los hacen con el fin de quien tiene muy pocas nociones sobre el diseño de las redes neuronales puedan acceder a esta tecnología por medio de “asistentes virtuales” como los tiene por ejemplo PowerPoint al crear una presentación.

Esto obviamente es desventajoso hasta cierto punto para quien desea total interactividad con el diseño debido a su familiaridad con el mismo. La ventaja es el ahorro de tiempo.

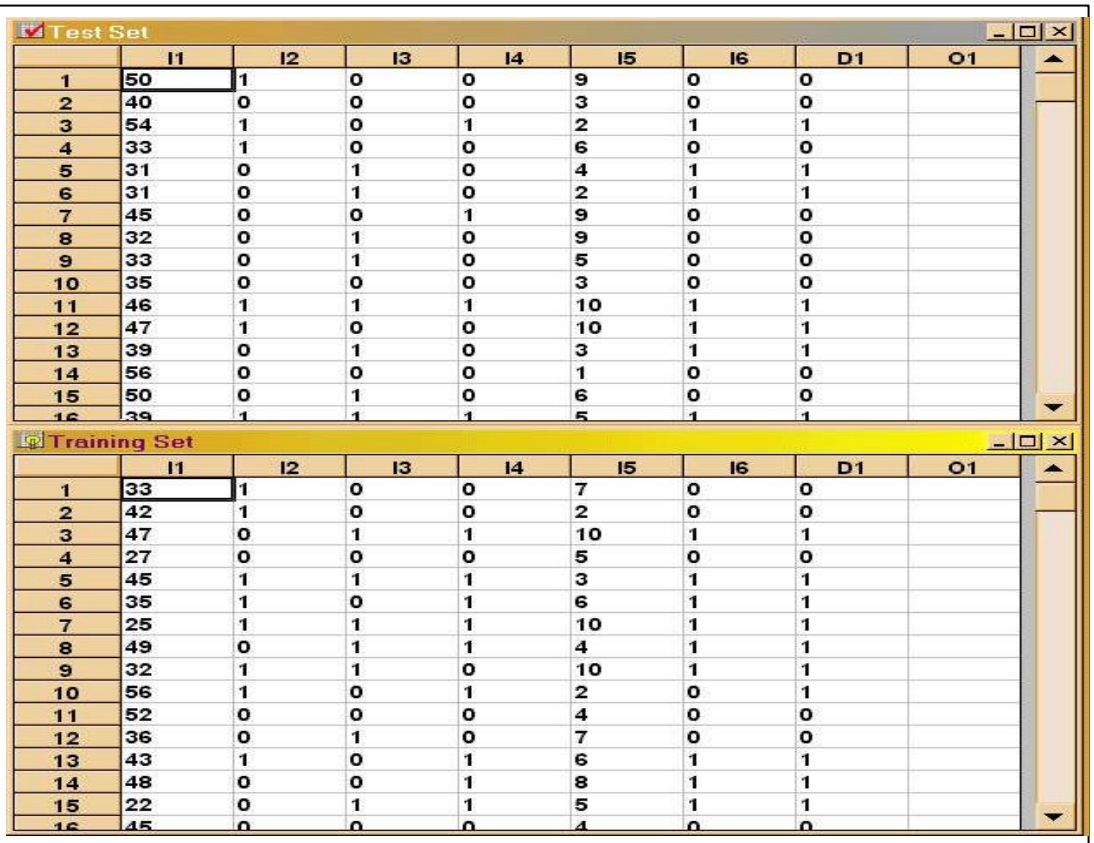
De la misma figura 4.4 nótese que los parámetros de velocidad y momento escogidos fueron 0.05 y 0.03 respectivamente. La razón de ser de estos valores es similar a la expuesta en la capa oculta con la diferencia que en esta capa la velocidad del aprendizaje no es muy influenciada debido a la existencia de un único nodo.

4.4.2 Importación de los datos

El procedimiento de importación de datos de entrenamiento y prueba es bastante sencillo. Lo que hay que cuidar es digitar la información previamente de tal manera que ThinksPro pueda reconocerla sin ningún problema. Por ejemplo no se debe grabar los datos con el encabezado como nombre de la variable ya que el programa lo hace por omisión.

Simplemente optamos por el comando ***File / Import File / Training set*** y ***File / Import File / Test set*** y para importar los archivos prueba11.txt y prueba22.txt que previamente guardamos desde MSExcel.

Después de cada importación la hoja de datos que al principio estuvo vacía se llena automáticamente con los datos digitados, también las variables reciben su nombre automáticamente y ThinksPro considera de manera automática que la última variable es la de salida, por eso es necesario que conservemos ese orden al ingresar los datos.



Test Set

	I1	I2	I3	I4	I5	I6	D1	O1
1	50	1	0	0	9	0	0	
2	40	0	0	0	3	0	0	
3	54	1	0	1	2	1	1	
4	33	1	0	0	6	0	0	
5	31	0	1	0	4	1	1	
6	31	0	1	0	2	1	1	
7	45	0	0	1	9	0	0	
8	32	0	1	0	9	0	0	
9	33	0	1	0	5	0	0	
10	35	0	0	0	3	0	0	
11	46	1	1	1	10	1	1	
12	47	1	0	0	10	1	1	
13	39	0	1	0	3	1	1	
14	56	0	0	0	1	0	0	
15	50	0	1	0	6	0	0	
16	39	1	1	1	5	1	1	

Training Set

	I1	I2	I3	I4	I5	I6	D1	O1
1	33	1	0	0	7	0	0	
2	42	1	0	0	2	0	0	
3	47	0	1	1	10	1	1	
4	27	0	0	0	5	0	0	
5	45	1	1	1	3	1	1	
6	35	1	0	1	6	1	1	
7	25	1	1	1	10	1	1	
8	49	0	1	1	4	1	1	
9	32	1	1	0	10	1	1	
10	56	1	0	1	2	0	1	
11	52	0	0	0	4	0	0	
12	36	0	1	0	7	0	0	
13	43	1	0	1	6	1	1	
14	48	0	0	1	8	1	1	
15	22	0	1	1	5	1	1	
16	45	0	0	0	4	0	0	

Figura 4.5 Muestra de las hojas de datos en ThinksPro

En la figura 4.5 observamos el caso para los conjuntos de entrenamiento y prueba.

La columna o_1 representa el valor que la red le dará al resultado producto del entrenamiento, así, el error se construye a partir de la diferencia entre el valor deseado d_1 y el valor actual o_1. El valor inicial de o_1 está dado por una semilla aleatoria puesta en los valores de los pesos de los nodos en las distintas capas.

4.4.3 Entrenamiento de la red

Inicialmente es posible probar la red o entrenarla (opción abierta que el programa permite) pero suena ilógico probar una red que no ha sido entrenada. Entonces procedemos a habilitar la opción de entrenamiento. Para esto hemos elegido un valor mínimo del error de 0.028 como criterio de convergencia. Elegimos también que el número de iteraciones sea infinito hasta que se encuentre bajo convergencia. Elegimos ***Run*** para el conjunto de entrenamiento y procedemos a entrenar la red.

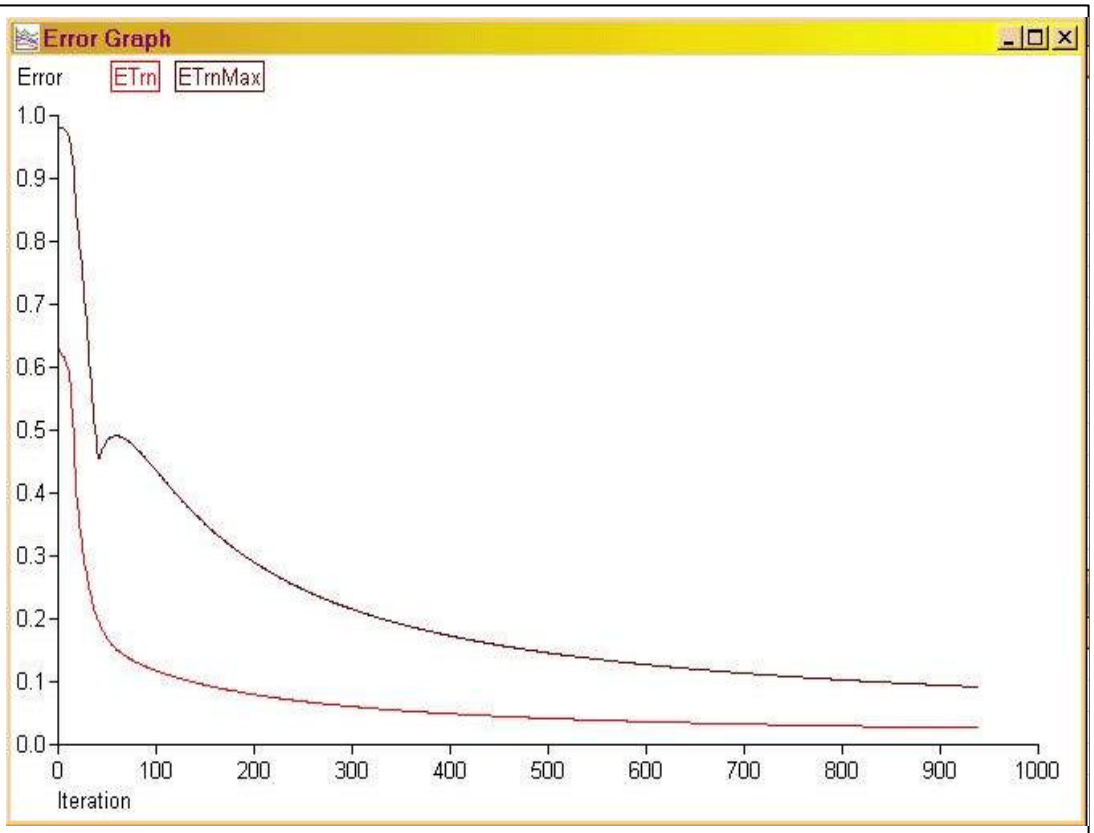


Figura 4.6 Evolución del error global conforme la red es entrenada

En la figura 4.6 se puede ver que el error baja rápidamente hasta que encuentra el criterio de convergencia. La línea de color roja representa el error al que nos referimos: ***ETrn*** o "error de entrenamiento".

La evolución del error también se captura en un archivo, la figura 4.7 muestra parte del visor de los valores de error de entrenamiento en ThinksPro.

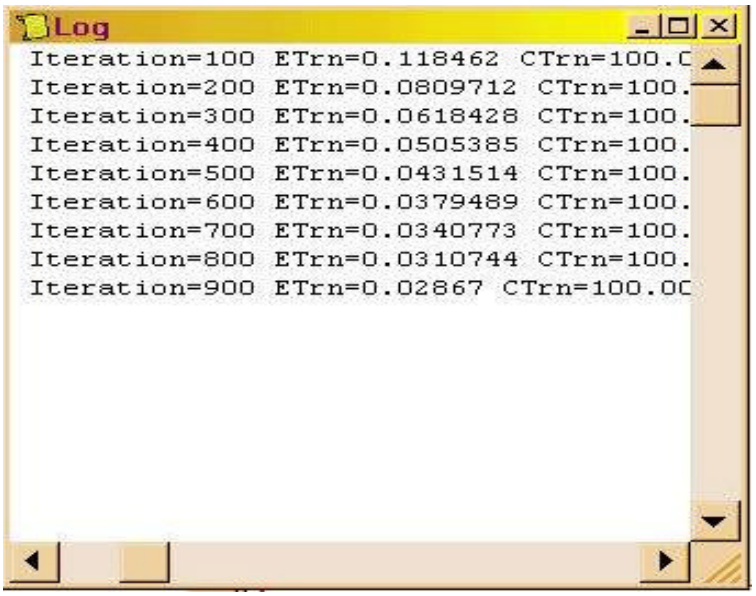


Figura 4.7 Visor de la evolución del error cada 100 iteraciones

El valor final del error no se muestra en este visor ya que cuando se llega convergencia el programa muestra un mensaje afirmándolo.

4.5 Obtención de los resultados

Parte de los resultados ya los hemos mostrado en lo que respecta del valor final del error. Observemos la figura 4.8.

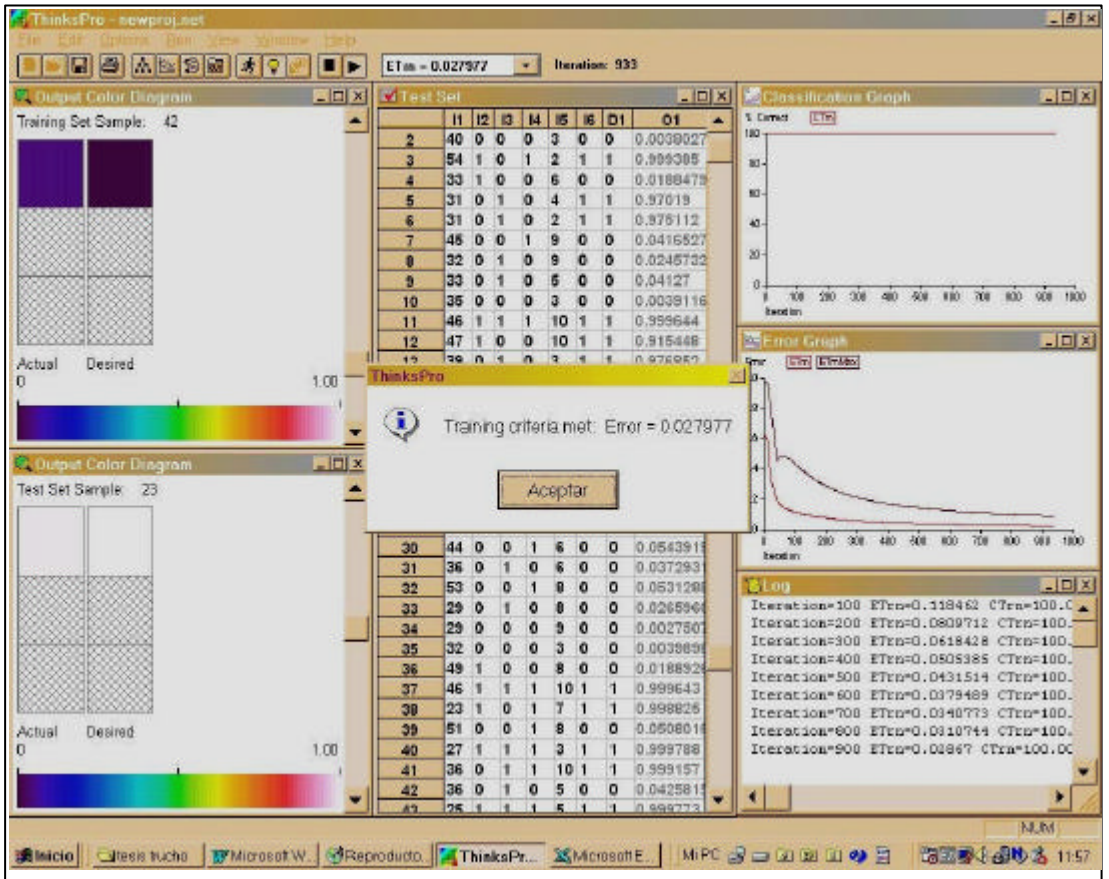


Figura 4.8 Resultados de ThinksPro al cabo de 933 iteraciones

Notemos el mensaje que ThinksPro muestra al final. El error, al cabo de 933 iteraciones sobre el conjunto de datos de entrenamiento es de 0.027977 que ciertamente es inferior al error propuesto de 0.028.

Hay dos cosas más de interés que podemos observar en la figura 4.8: la primera es aquellos los valores que adquieren las variables de salida o_1 para los conjuntos de entrenamiento y prueba.

Para los casos en que el resultado deseado es “deudor” (es decir, d_1 = 1) el valor actual de o_1 es muy cercano a 1, y para los casos en que el resultado deseado es “no deudor” (d_1 = 0) el valor actual de o_1 es muy cercano a cero; cosa que constituye otra prueba que el entrenamiento ha sido eficaz.

La segunda cuestión de interés es el visor de resultados de ThinksPro. De manera novedosa se crea una escala de colores que varia desde cero hasta uno. Esta escala representa todos los posibles valores que o_1 puede tomar. Caso por caso se puede ver como aquellos valores cercanos a cero tienen un color muy oscuro (negro = 0) y que los valores cercanos a uno tienen un color claro (blanco = 1).

Este visor existe para ambos conjuntos, de entrenamiento y de prueba. En la figura 4.8 observamos los colores que tienen los valores de d_1 y o_1 para los casos 42 y 23 de entrenamiento y prueba respectivamente.

Cuando la red no está entrenada los colores no coinciden y rara vez se parecen, en este caso tenemos todo lo contrario.

Al final tenemos a éste como último criterio para establecer la eficacia del entrenamiento de la red. En los anexos podemos ver el archivo de resultados que concuerdan con lo aquí expuesto.

4.6 Observaciones

4.6.1 Sobre el conjunto de datos

Una observación final en este capítulo es que hemos detectado cierto sesgo. Lo bueno de esto es que si tenemos un nivel de certeza del origen del mismo.

El sesgo consiste en que en la mayoría de los casos aquellos clientes que pagan a través de su tarjeta de crédito se reportan como deudores en la muestra.

Esto obviamente distorsiona el aprendizaje ya que resulta ilógico el generalizar esta premisa. Con esto queremos decir que tal y como está entrenada la red no fuera suficiente (ni confiable) para que esta evalúe a un nuevo cliente y lo clasifique dentro del grupo deudor o no deudor.

El origen de este sesgo proviene de las pocas fuentes confiables de datos que obtuvimos. Lo lógico es pensar que la mayoría de deudores que digitamos provenían de algún archivo específicamente de tarjeta habientes.

Es el riesgo que asumimos al llevar a cabo este trabajo. En las conclusiones trataremos este tema con mayor amplitud.

4.6.2 Sobre el uso práctico de la red

Otra observación será mas bien el explicar como sería la utilización práctica de la red. Supongamos que la red esta bien entrenada (como de hecho está), con un conjunto confiable de datos (cosa difícil en este caso), entonces supongamos también que somos el departamento de crédito de Bellsouth, encargado de aprobar y desaprobar los planes tarifarios otorgados a los nuevos clientes.

La red neuronal entrenada me servirá para que de ahora en adelante yo pueda tener un mejor criterio de a quién aprobarle los planes y a quién no, de la siguiente manera: En el conjunto de datos de Prueba inserto un nuevo caso.

Este nuevo registro representa al nuevo potencial cliente y sus variables crediticias a ser evaluadas. En el campo d_1 no debemos ingresar valor alguno ya que no sabemos a priori si es que este nuevo sujeto será deudor o no.

Al ejecutar el archivo de prueba desde su visor de datos obtenemos inmediatamente un valor para o_1. Al hacer la misma ejecución desde el visor de colores obtenemos lo mismo, que es un nuevo color (o valor) para o_1.

Entonces si este nuevo valor se aproxima a 1 sabremos que probablemente este cliente no sea un buen pagador y si se aproximara a cero pensáramos lo contrario.

Es así como finalmente hemos utilizado una tecnología complicada en una idea sencilla. Obviamente que a medida que la base de datos es alimentada habría que reentrenar la red periódicamente.