Simulador de redes neuronales artificiales UV-SRNA

❒ **Jesús Alfonso López Sotelo, M.Sc.**
Profesor Universidad Autónoma de Occidente. Cali-Colombia

❒ **Eduardo Francisco Caicedo Bravo, Ph.D.**
Profesor Titular
Universidad del Valle.
Escuela de Ingeniería Eléctrica y Electrónica

Grupo de Inteligencia Computacional.

* La Revista recibió este artículo en enero de 2.000

RESUMEN

En este trabajo se presenta un simulador de redes neuronales artificiales (RNA) desarrollado en el grupo de inteligencia computacional de la E.I.E.E de la Universidad del Valle. Dadas las características de este simulador es adecuado para usarlo como herramienta en un curso introductorio a las redes neuronales así como en el diseño y simulación de aplicaciones basadas en RNA. El proceso de concepción y diseño de la herramienta es presentado además de unos ejemplos donde se muestra la aplicabilidad del simulador.

ABSTRACT

This paper presents an environment to design and simulation basic applications based on Artificial Neural Networks. This tool can be used an introductory course at graduate and postgraduate levels.

KEYWORDS:

Neural Networks, Computacional Intelligence, Educational Software, Object Oriented Programming.

1. INTRODUCCIÓN

UV-SRNA (Universidad del Valle-Simulador de Redes Neuronales Artificiales) es una herramienta desarrollada en la Escuela de Inge-

niería Eléctrica y Electrónica de la Universidad del Valle, concebida con los siguientes requerimientos:

- Proporcionar una base modular para posteriores desarrollos en el área de las R.N.A.
- Poseer un ambiente que sirva como soporte a las prácticas de laboratorio que un curso introductorio a las RNA necesita.
- Brindar al usuario un ambiente de trabajo adecuado para el diseño y simulación de las arquitecturas más representativas de RNA.

Con base en estos requerimientos se diseño e implementó una herramienta de simulación de RNA con las siguientes características:

- Lectura o carga de los patrones de entrenamiento por medio de un archivo,.
- Inicialización de la red neuronal con una arquitectura definida por el usuario.
- Entrenamiento de la red: dependiendo de la arquitectura de RNA el entrenamiento involucrará minimizar un error (backpropagation), asignación de pesos (aprendizaje hebbiano) o una competencia de las neuronas para parecerse a un subconjunto de los patrones de entrada.
- Definición de parámetros de entrenamiento: de esta forma se garantiza el control del usuario sobre el proceso de aprendizaje.
- Posibilidad de validación de la red luego de haber sido entrenada.
- Almacenamiento de una red ya entrenada para posterior uso de la misma.
- Modularidad, ya que la herramienta de simulación de RNA puede crecer en la medida que las necesidades lo vayan planteando.
- Fácil mantenimiento y actualización.
- Amigabilidad: característica que garantiza una interfaz con el usuario agradable y fácil de usar, esto se logra por medio de menús, botones que controlan las acciones del simulador, un archivo de ayuda*.hlp en que el usuario podrá consultar dudas que tenga sobre el manejo de programa. La amigabilidad de la herramienta hace posible que un usuario neófito en el uso de simuladores de RNA pueda llevar a cabo los diferentes pasos necesarias para utilizar una RNA, para resolver un problema específico.

2. DISEÑO Y ESTRUCTURA DE UV-SRNA

Una vez se han establecido los requisitos del software, el diseño del software es la primera de tres actividades técnicas (diseño, codificación y prueba). Cada actividad transforma la información de forma que finalmente se obtiene un software para computadora validado.

El flujo de información durante las fases de desarrollo se muestra en la figura 1, como se puede observar, se parte de unos requisitos provenientes del exterior, para cumplir con los mismos se genera una propuesta de solución por medio del diseño del software, con este diseño se pasa a la codificación y posteriormente a la prueba del mismo.

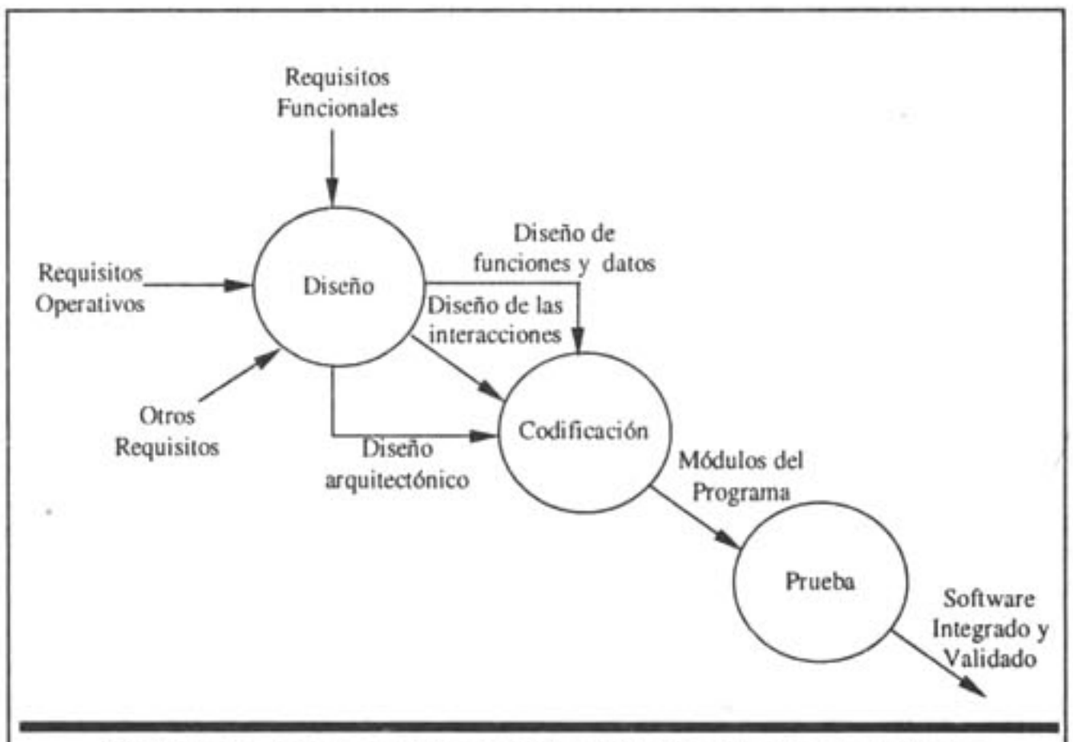

Figura 1. *Pasos para el desarrollo de un software*

El diseño de software es la única forma mediante la cual se puede traducir con precisión los requisitos del cliente o de la necesidad que se desee solucionar, mediante el software en un producto o sistema acabado. El diseño de software sirve como base de todas las etapas posteriores del desarrollo y de la fase de mantenimiento. Sin diseño se corre el riesgo de construir un sistema inestable, sistema que falle cuando se realicen pequeños cambios, un sistema que puede ser difícil de probar.

El diseño involucra dos pasos, el diseño preliminar que se centra en la transformación de los requisitos en los datos y la arquitectura del software. El diseño detallado se ocupa de refinar las estructura de datos planteadas y a la representación algorítmica del software.

Además del diseño de datos y arquitectónico, en muchas aplicaciones modernas se requiere una actividad distinta denominada diseño de la interfaz, la cual establece la disposición y los mecanismos para la interacción hombre-máquina.

Una manera de plantear algunos requisitos del software a diseñar es plantear un diagrama de flujo de información del sistema con el exterior, figura 2, pues aquí ya tendríamos una base para los pasos que el diseño involucra.

Una vez definido el enfoque para el diseño del sistema, se debía elegir el lenguaje en el que se haría la implementación. Inicialmente se pensó en C++ pues ya se poseía cierta cantidad de código que implementan RNA en este lenguaje, pero lo dispendioso de construir GUI en C++ hizo buscar otro compilador que permitiera construir a las mismas sin tanto esfuerzo y que permitiera tener todas las ventajas de un lenguaje orientado a objetos.

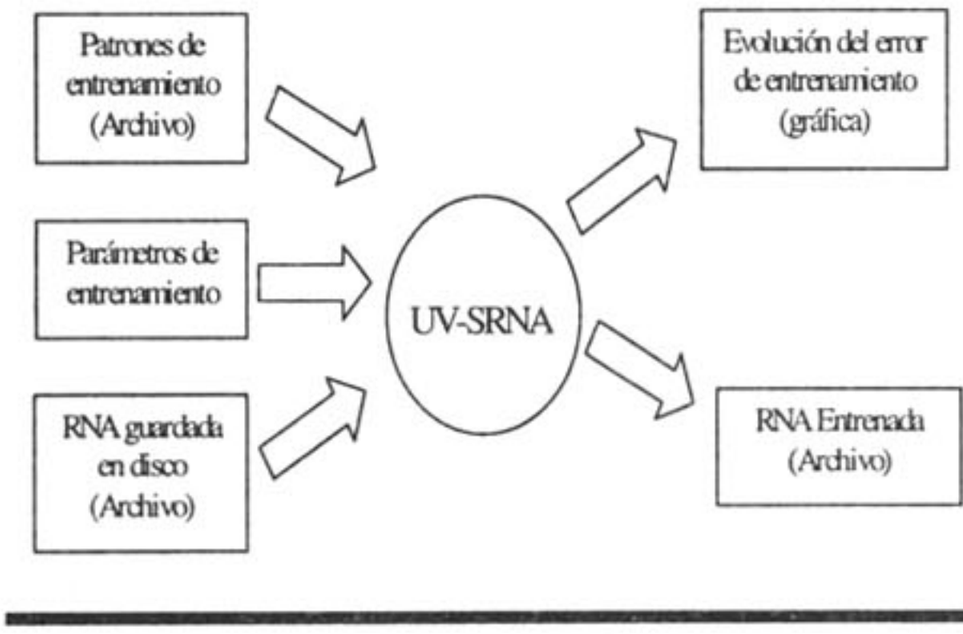

Figura 2. *Relación del sistema UV-SRNA con el exterior*

Como resultado de una investigación para seleccionar el lenguaje en que se haría la aplicación se decidió trabajar en Delphi, pues a pesar de existir en el mercado otros lenguajes de características similares, por ejemplo Visual Basic, Delphi ofrece un código que es más eficiente, es compatible en cierta medida con C++, pues desde Delphi se puede invocar DLL hechas en este otro lenguaje; además de ser una lenguaje que permite desarrollar de manera rápida interfaces con el usuario, se tienen todas las ventajas de trabajar con objetos pues en Delphi se programa con Object Pascal (Pascal orientado a objetos)

Producto del diseño orientado a objetos resultan unas clases que al interaccionar van a permitir que se puedan codificar sin mucha complejidad los algoritmos de RNA más representativos.

Las clases implementadas son:

- Clase neurona
- Clase capa
- Clase red

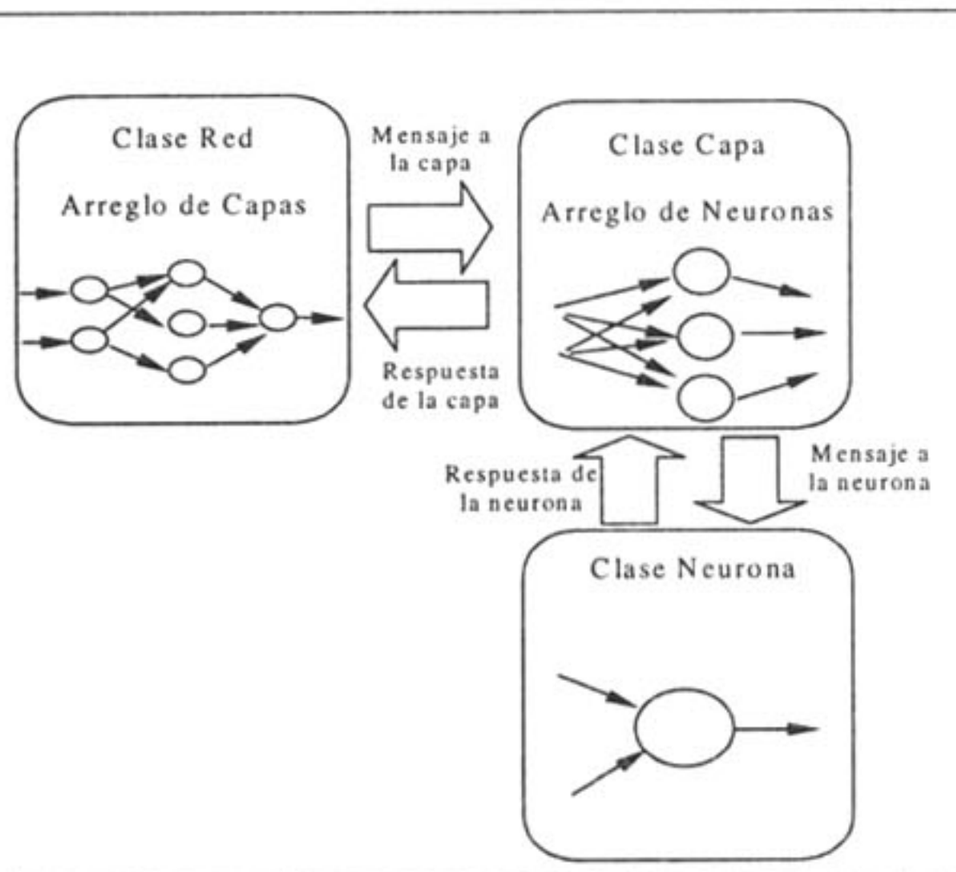

Figura 3. *Relación entre las clases que se utilizan para la codificación de los algoritmos de UV-srna.*

La figura 3. Muestra como se relacionan las clases mencionadas

La codificación del sistema llevó a que el mismo tuviera una estructura como la mostrada en la figura.4, en ella se puede apreciar las siguientes partes o módulos:

- Biblioteca de enlace dinámico (DLL)
  En una DLL (dynamic link lybrary), se tienen empaquetadas las clases realizadas en C++ que, como ya se ha mencionado, son la clase neurona, la clase capa y la clase red.
  Dichas clases son por decirlo así, los módulos con los que se pueden construir los diferentes algoritmos de entrenamiento.
- Algoritmos de entrenamiento
  En el módulo de algoritmos de entrenamiento a través de las clases empaquetadas en la DLL, se implementan los diferentes algoritmos de R.N.A. que posee la herramienta. Los algoritmos que posee la aplicación son los siguientes:
  - Algoritmo de aprendizaje tipo perceptron
  - Algoritmo de aprendizaje tipo backpropagation
  - Algoritmo de aprendizaje tipo hopfield
  - Algoritmo de aprendizaje tipo kohonen

Para aclarar como se codificaron los diferentes algoritmos de entrenamiento, en el siguiente seudocódigo se muestra el proceso que se realiza en UV-SRNA para una red MLP con algoritmo de entrenamiento Backpropagation.

- Definición de la variable red
  obj_red: red; { Declare TMyObject Object }
- Creación del objeto red
  obj_red:= InitObject_Red;
  Cuando se crea el objeto se llama a la DLL (neural.dll)
  function InitObject_Red: red ; stdcall; far; external 'neural.dll' name 'InitObject_Red';
- Carga del archivo de patrones
  Esto se hace leyendo un archivo tipo texto donde están los patrones que serán usados en el proceso de entrenamiento
- Inicialización de la red
  Habiendo cargado los patrones se procede a inicializar la red
  obj_red.Init_red(Backbp.Niveles,fb,Backbp.num_neuronas);
  En la inicialización de la red, también se le asignan valores aleatorios a todos los pesos sinapticos
  obj_red.Getcapa(i).Getneurona(j).SetW(k,-0.9+0.018*(random(100)mod 100+));
- Habiendo inicializado la red y con los patrones ya cargados, se procede a entrenarla
  TrainBack(entradas,salidas,error_min,alfa,beta);
  En el proceso de entrenamiento se va mostrando la evolución del error de manera gráfica.
- Estando la red entrenada se procede a probarla
  Para esto se propaga el patrón de prueba y se mira

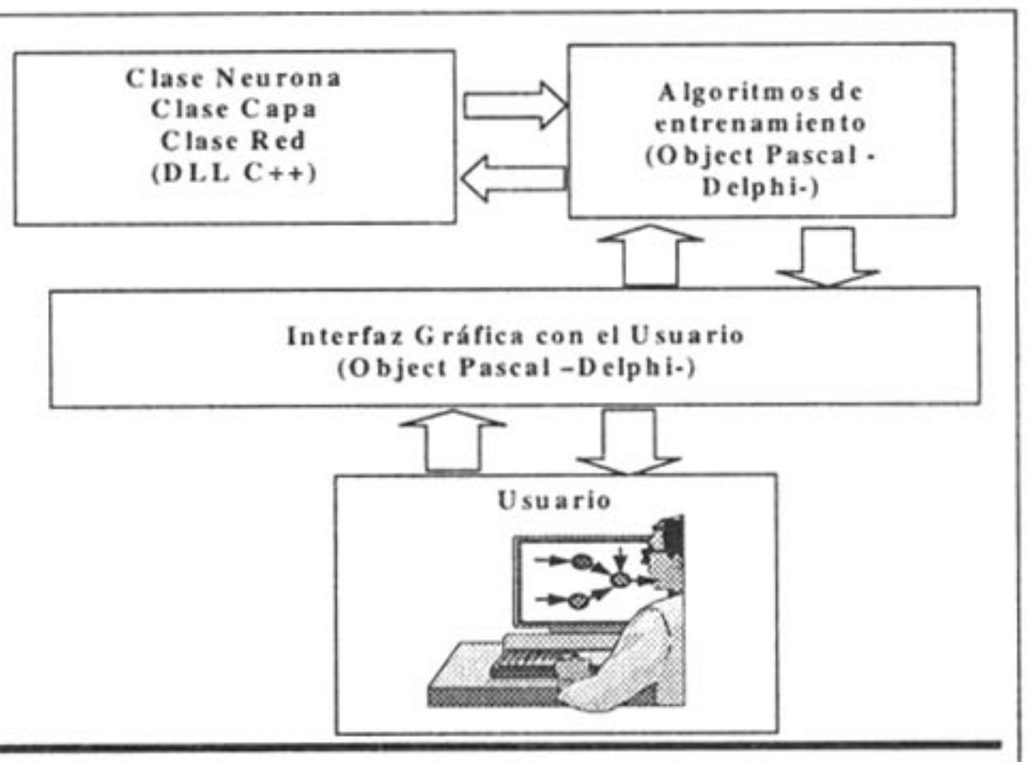

Figura 4. *Estructura de la herramienta para simular R.N.A.*

cual es la salida de la red.
FeedBack(Pat_prue,Sender);
Salida=obj_red.Getcapa(C_Out).Getneurona(j).Getf_neta;

- Si el entrenamiento no es satisfactorio, el usuario puede repetir todo el proceso desde la inicialización de la red para observar si obtiene un mejor entrenamiento.
- Si el entrenamiento es satisfactorio, el usuario puede guardar la red entrenada en un archivo.
- Al terminar la sesión con el simulador se destruye el objeto creado invocando nuevamente a la DLL
UnInitObject_Red(obj_red);
procedure UnInitObject_Red(obj_red: red); stdcall; far; external 'neural.dll' name 'UnInitObject_Red';

Debido a la modularidad de la programación orientada a objetos, en la medida que se realicen nuevos algoritmos de aprendizaje se podrán ir anexando a la herramienta, de hecho ya se están implementando lo siguientes algoritmos:

- Algoritmo de aprendizaje para redes R.B.F.
- Algoritmo de aprendizaje para redes A.R.T.
- Algoritmo de aprendizaje gradiente conjugado
- Algoritmo de aprendizaje levenberg-marquardt

Interfaz gráfica con el usuario (GUI)

La GUI (Graphical User Interface) es un tipo de exhibición que permite al usuario escoger comandos, iniciar programas, observar listas de archivos y otras opciones mediante el señalamiento de representaciones gráficas (botones, íconos) y lista de menús sobre la pantalla. Por lo general las opciones pueden activarse desde el teclado o el ratón.

La interfaz gráfica con el usuario es la parte de la aplicación que interactúa con él, pues por medio de ella es que la implementación recibe los valores, las órdenes del usuario, para procesarlos y posteriormente, al tener un resultado, entregárselo al éste.

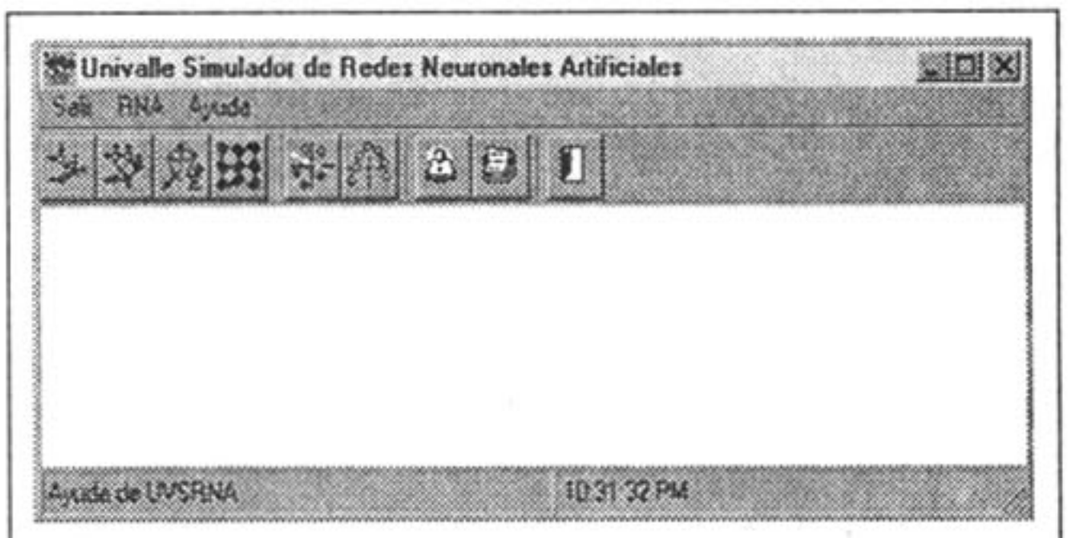

Figura 5. *Ventana inicial de UV-SRNA*

Figura 6. *Barra de herramientas de UV-SRNA*

En el desarrollo de esta herramienta se prestó especial atención al desarrollo de la GUI pues un buen diseño de la misma garantiza en gran medida la que el usuario pueda realizar lo que desea con el programa.

3. GENERALIDADES DE UV-SRNA

UV-SRNA implementa los siguientes algoritmos de RNA:

- RNA tipo perceptron.
- RNA tipo backpropagation (BP).
- RNA tipo hopfield.
- RNA tipo kohonen (Mapas autoorganizados).

A cualquiera de las RNA mencionadas se puede acceder por medio de la ventana inicial de UV-SRNA (figura 5.).

Para entrar a alguno de los módulos que conforman a UV-SRNA se puede hacer por medio de la barra de herramientas o por medio del menú RNA.

La barra de herramientas consta de los siguientes elementos.

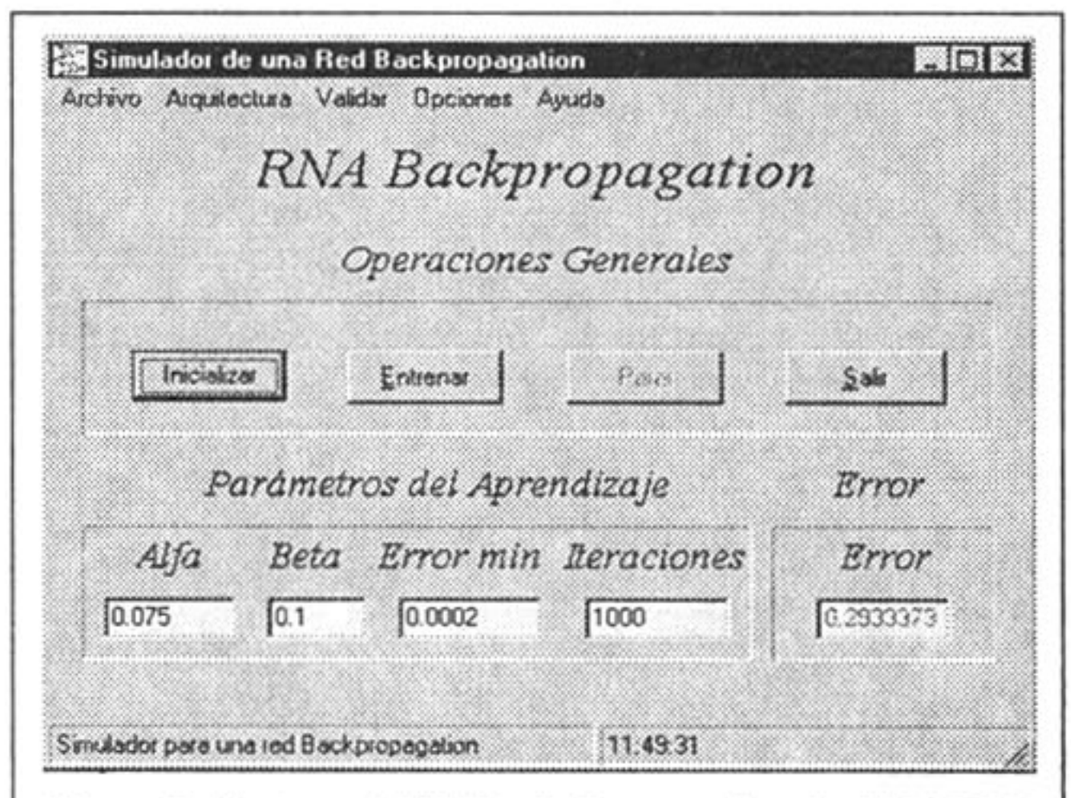

Figura 7. *Ventana RNA Back Propagation de UV-SRNA*

Por ejemplo cuando seleccionamos la RNA back propagation se despliega la siguiente ventana.
UV-SRNA ofrece, además, una ayuda en ambiente Windows que le permitirá al usuario consultar los aspectos necesarios para poder utilizar adecuadamente el simulador.
En todas la RNA implementadas se pueden usar archivos de entrenamiento, hasta 2500 patrones de una dimensión máxima de 100 componentes.
Cuando se entrena una RNA es muy importante tener información de cómo va el entrenamiento de la misma, UV-SRNA para la redes tipo MLP ofrece la posibilidad de visualizar la evolución del error cuadrático.
Si la red a entrenar es un mapa autoasociativo se puede visualizar la evolución del mapa.

Figura 8. *Ventana donde se muestra la evolución del error cuadrático.*

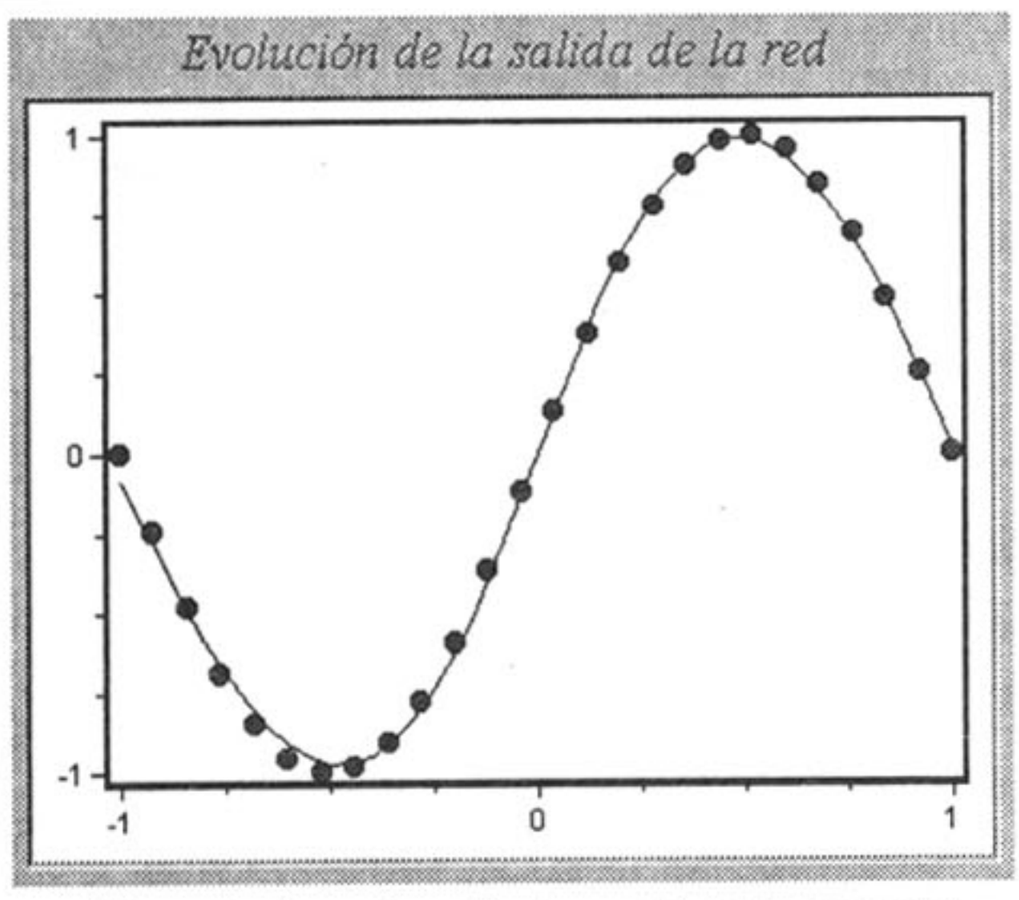

Figura 9. *Aproximación de una función senoidal*

4. EJEMPLOS

Los ejemplos siguientes muestran la versatilidad de UV-SRNA para resolver problemas de mediana complejidad en el campo de las RNA.

- Ajuste a curva en el plano
  En este caso lo que se pretende es entrenar una RNA para que se ajuste a una curva definida por unos puntos en el plano. Para esto se usa el módulo aprendizaje de curvas del simulador, en el mismo se pueden cargar los patrones que definen una curva y posteriormente se entrena la RNA. El simulador muestra un gráfica (figura 9) donde se pueden apreciar los puntos del entrenamiento y la aproximación que va realizando la RNA.

- SOM que aprende una distribución de puntos en el plano
  En este caso lo que se pretende es tener una red neuronal que se adapte a una distribución de puntos en el plano sin importar como sea la misma. Lo interesante es que la RNA aprende la distribución de sin variar la arquitectura de la misma. (figuras 10 y 11)

5. CONCLUSIONES

Este trabajo presentó la manera como se concebió y diseñó UV-SRNA Universidad del Valle Simulador de Redes Neuronales Artificiales (RNA).
Como resultado del proceso de diseño, el simulador posee unas características de modularidad, de facilidad para el diseño y simulación de RNA por medio de una GUI y de facilidad de mantenimiento.
El simulador ha mostrado cumplir con los requerimientos para usarlo como herramienta de trabajo en un curso introductorio de RNA, debido a sus características.
El diseñar, concebir e implementar este simulador ha permitido a los autores enfrentar trabajos de mayor complejidad en el campo de las redes neuronales tales como neurocontrol, identificación de sistemas, procesamiento de imágenes entre otros.

6. BIBLIOGRAFÍA

[1.] AGUILAR Luis J. Turbo Pascal 7.0, Manual de bolsillo. Mc Graw Hill. España. 1995. 456p
[2.] CORNELL Gary – Strain Troy. Programación en Delphi. Mc Graw Hill. España. 1996. 277p
[3.] FAUSETT Laurene. Fundamentals Of Neural Networks. Prentice Hall International, Inc. London. 1994. 461p.
[4.] FREEMAN James A. Skapura David M. Redes Neuronales , Algoritmos, aplicaciones y técnicas de programación. Addison-Wesley Iberoamericana

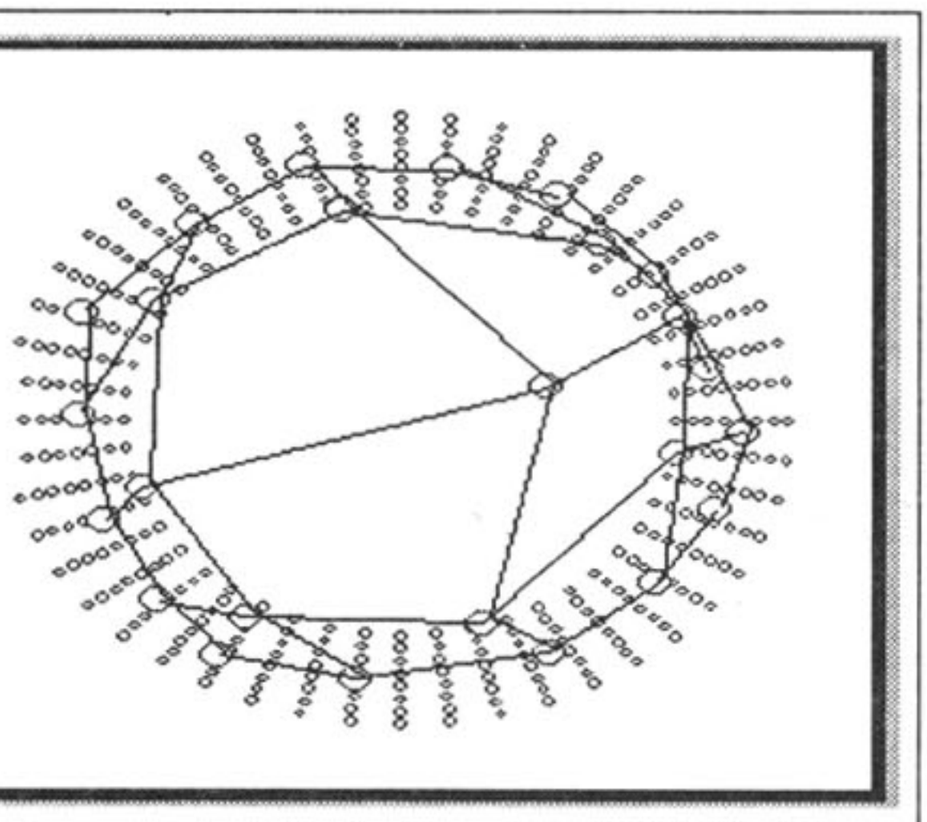

Figura 10. *Aprendizaje de una distribución de puntos toroidal*

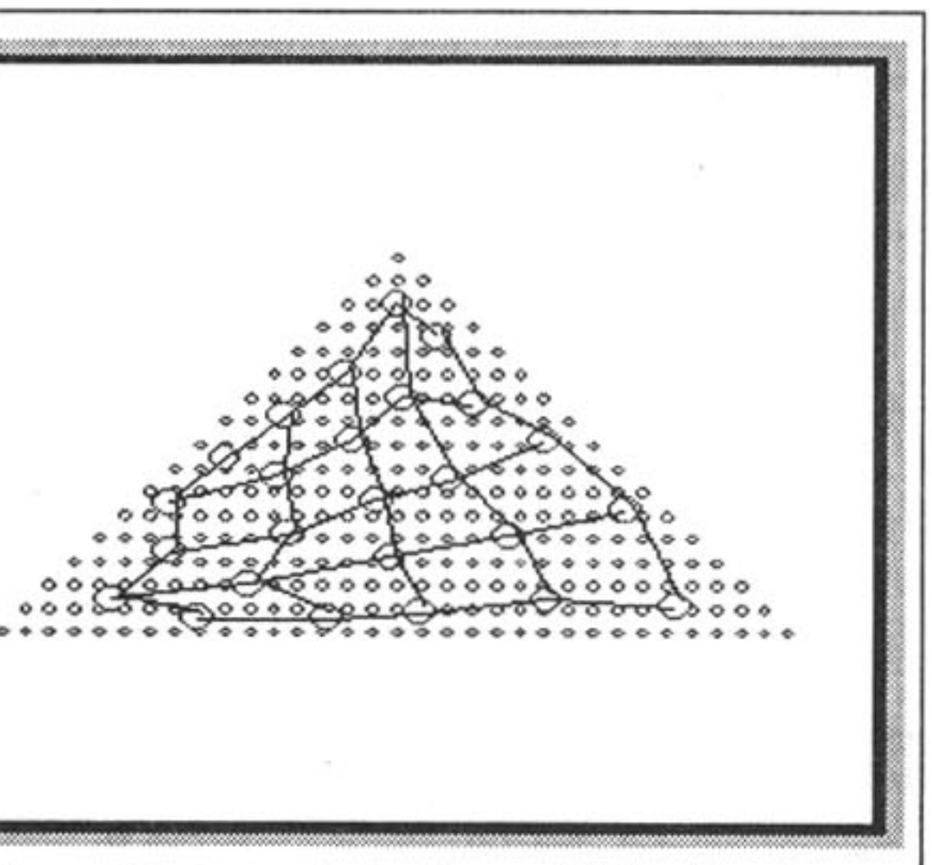

Figura 11. *Aprendizaje de una distribución de puntos triangular*

S.A. Wilmington, Delaware, E.U.A. 1993. 431p.

[5.] FRERKING Gary, Wallace Nathan y Niddery Wayne. Borland Delphi How-To. Waite Group Préss. Unites States. 1995. 880p.

[6.] HAYKIN Simon. Neural Networks, A Comprehensive Foundation. Prentice Hall, Inc. E.U.A. 1994. 696p.

[7] HILERA José Ramón - Martínez Víctor José. Redes Neuronales Artificiales. Addison-Wesley Iberoamericana S.A. Wilmington, Delaware, E.U.A. 1995. 390p.

[8.] HOPFIELD J.J. Neural Network and Physical Systems with Emergent Collective Computational Abilities. Proceedings of the National Academy Of Sciences of the U.S.A. 79, 2554-2558. 1982.

[9.] KOHONEN T. Self-Organizing Maps, Second Edition. Springer Verlag. Germany. 1997. 450p.

[10.]LÓPEZ Jesús A. Concepción e Implementación de un Entorno de Diseño de Sistemas Basados en Redes Neuronales Artificiales. Tesis de Maestría. Universidad del Valle. 1998.

[11.]LIPPMANN Richard. An Introduction To Computing with Neural Nets. IEEE ASSP Magazine. April 1987 pp 4-22.

[12.]LOONEY Carl G. Pattern Recognition Using Neural Nerworks. Oxford University Press. United States. 1997. 480p.

[13.]MCCULLOCH W.S. and Pitts W. A Logical Calculus of the Ideas Immanent in Nervous Activity. Bulletin of Mathematical Biophysics. 5, 115-133. 1943

[14.]MASTERS Timothy. Practical Neural Networks Recipes en C++. Academic Press. United States. 1993. 510p.

[15.]MISKY M.L and Papert S.A. Perceptrons. MIT Press. Cambridge MA. United States. 1969.

[16.]OSIER Dan, Grobman Steve y Batson Steve. Aprendiendo Delphi 2 en 21 Días. Prentice Hall Hispanoamericana. México. 1996. 736p

[17.]RAO Valluru y Rao Hayagriva. C++ Neural Networks and Fuzzy Logic. MIS: Press. United States. 1995. 575p.

[18.]WELSTEAD Stephen. Neural Networks and Fuzzy Logic Applications in C/C++. John Wiley and Dons, Inc. United States. 1994. 510p.

[19.]WIDROW B. and Hoff M.E. Adaptive Switching Circuits. IRE WESCON Convention Record, pag. 96-104.1960

AUTORES

Jesús Alfonso López S. Ingeniero Electricista de la Universidad del Valle (1996). M.Sc. en Automática (1998) de la Universidad del Valle. Miembro del grupo de redes neuronales de la Universidad del Valle. Docente hora cátedra en la Corporación Universitaria Autónoma de Occidente y en la Universidad Antonio Nariño. Sus áreas de interés son las Redes Neuronales Artificiales y sus aplicaciones y los sistemas de control automático.
jes_alf_lopez@yahoo.com.

Eduardo F. Caicedo B. Ingeniero Electricista de la Universidad del Valle (1984), Máster en Informática Industrial (1993) y Doctor Ingeniero en Informática Industrial de la Universidad Politecnica de Madrid (1995). Profesor Titular de la Escuela de Ingeniería Eléctrica y Electrónica, vinculado a la cátedra de Instrumen-tación y Percepción Inteligente. Su trabajo se centra en el área de los microprocesadores y de los sistemas inteligentes.
ecaicedo@eiee.univalle.edu.co