Neural Networks and Wireless Communications Modeling

M. Rubiolo, G. Stegmayer, *IEEE Member*, y O. Chiotti

***Abstract* -- This paper presents a free software tool that supports the next-generation Mobile Communications, through the automatic generation of models of components and electronic devices based on neural networks. This tool enables the creation, training, validation and simulation of the model directly from measurements made on devices of interest, using an interface totally oriented to non-experts in neural models. The resulting model can be exported automatically to a traditional circuit simulator to test different scenarios.**

***Keywords* -- wireless communications; modeling; simulation; neural networks; software tools.**

I. INTRODUCCION

EN los sistemas de comunicaciones móviles de tercera generación (3G), por ejemplo WCDMA (Wideband Code Division Multiple Access) y UMTS (Universal Mobile Telecomunications System), hacia los cuales migrarán la mayoría de las redes de comunicación celulares, el modelado de los componentes del sistema se ha convertido en un punto crítico del ciclo de diseño del sistema, debido a las técnicas de modulación digital modernas [1]. Los nuevos estándares pueden introducir una distorsión en el comportamiento de los dispositivos que son parte del sistema (p.e. teléfonos móviles y sus componentes internos tales como amplificadores de potencia) debido generalmente a los cambios que aplican a la señal modulada, generando efectos de no-linealidad y de memoria (cuando una señal de salida depende de valores retrasados en el tiempo de una señal de entrada).

Los amplificadores de potencia (PA por su sigla en inglés power amplifier) son partes esenciales de un transmisor digital inalámbrico moderno (p.e. los teléfonos celulares). La Figura 1 muestra un diagrama de bloques simplificado de cómo podría ser una comunicación digital por teléfono celular. La voz que proviene del interlocutor (señal analógica) debe ser digitalizada para ser transmitida a través de la red celular inalámbrica, y esta tarea es realizada por un convertidor Analógico/Digital. Luego, la voz digitalizada es comprimida para reducir la taza de bits y el ancho de banda utilizado. También es codificada, para dar formato a los datos de manera tal que el receptor pueda detectar y minimizar los errores mediante la operación de decodificación. Luego de esta etapa, un modulador de señal ajusta la señal portadora para garantizar la comunicación. Pero la señal sufre atenuación y necesita una previa amplificación. Por lo tanto, el elemento final perteneciente a esta cadena es un PA, que se encarga de amplificar la señal antes de su viaje hacia la antena más próxima y el extremo receptor de la cadena de comunicación.

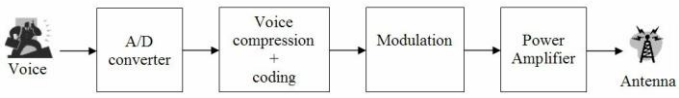

Figura 1. Diagrama de bloques simplificado de un transmisor digital inalámbrico.

El modelado del comportamiento no lineal y con memoria en dispositivos electrónicos ha sido objeto de investigación y de un creciente interés a lo largo de los últimos años [2]-[4] lo cual ha llevado recientemente a proponer nuevas alternativas al clásico análisis con circuito equivalente, tales como las Redes Neuronales Artificiales (RNAs) [3]. Actualmente, el modelado y simulación de elementos no lineales componentes de un sistema de comunicación inalámbrica con RNAs es un campo en crecimiento que genera una creciente atención por la gran variedad de posibles aplicaciones industriales [5][6].

En el dinámico mercado de las comunicaciones móviles, los modelos basados en RNAs son especialmente adecuados ya que podrían ser utilizados para acelerar el desarrollo de nuevos productos. Por ejemplo, los modelos neuronales podrían contribuir a una significativa reducción del ciclo de diseño de un producto cuando una nueva tecnología de dispositivo aparece en el mercado, lo que los convierte en objeto de interés particular en la industria. Un modelo neuronal puede ser utilizado durante la etapa de diseño de sistema para lograr una evaluación rápida de su performance y de sus características principales.

Los modelos neuronales pueden ser un enlace eficiente entre las mediciones y las simulaciones, permitiendo anticipar las consecuencias tecnológicas para la performance de un circuito. El modelo puede ser entrenado directamente con los datos extraídos del sistema real en cuestión, logrando una disminución en el ciclo del diseño y desarrollo de nuevos productos (más comúnmente llamado *time-to-market*).

Sin embargo, el proceso de desarrollo de un modelo neuronal no es trivial e involucra considerables puntos críticos tales como la generación de datos, la normalización de los mismos, la definición de una topología de RNA, la cantidad de neuronas en capa oculta, las reglas de aprendizaje, entre otras. Como las técnicas de RNAs son relativamente nuevas para la comunidad de los ingenieros electrónicos que deben desarrollar un modelo neuronal en vez de un circuito equivalente, es frecuente que se presenten dificultades para ellos en el

Los autores son miembros del Centro de I+D CIDISI de la Universidad Tecnológica Nacional, Facultad Regional Santa Fe, Lavaise 610 (3000) Santa Fe, Argentina (e-mail: georgina.stegmayer@ieee.org).

sinc(i) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
M. Rubiolo, G. Stegmayer & O. Chiotti; "Neural Networks and Wireless Communications Modeling"
IEEE Latin America Transactions, Vol. 8, pp. 486-492, 2010.

momento de la toma de decisiones para generar un modelo.

Por lo tanto, una herramienta que dé soporte al desarrollo de modelos neuronales puede ser de especial interés para los ingenieros involucrados en el mundo de las comunicaciones móviles, cuyos conocimientos sobre la teoría de RNAs puede ser limitada. Para este propósito, este trabajo presenta una herramienta de software que brinda soporte a la generación de modelos neuronales directamente desde las mediciones. Dichos modelos pueden ser exportados también a un simulador de circuitos. La generación automática de los modelos puede ayudar a asegurar implementaciones de modelos mutuamente consistentes para los diversos simuladores soportados.

La organización del trabajo es la siguiente: en la Sección II se explican las características principales de la herramienta. La Sección III muestra detalles de la implementación La Sección IV muestra un caso de estudio modelado con la herramienta de software desarrollada. Finalmente, se pueden encontrar las conclusiones en la Sección V.

II. Características de la Herramienta de Software

En la actualidad, existen productos disponibles en el mercado, tanto comerciales como no-comerciales, que permiten la creación y uso de modelos neuronales. Aún así, en la mayoría de éstos, se asume que el usuario es un experto en teoría de RNAs, porque es necesario un buen entendimiento sobre cómo trabaja el paradigma de RNA para definir un modelo. Es muy probable que un ingeniero electrónico que debe diseñar un modelo neuronal para, p.e. un amplificador PA, no tenga un nivel de conocimiento muy profundo acerca de un modelo neuronal.

El software matemático MatLab® ha incorporado recientemente un Neural Network Toolboox®[7] que permite la creación, entrenamiento y uso de varios tipos de RNAs, que pueden ser aplicadas para resolver una gran variedad de problemas. Para poder instalarlo es necesario comprar una licencia específica del software con un alto costo asociado. Otras alternativas comerciales (NeuralPlanner [8], NeuroSolutions [9]) y no comerciales (Lens Neural Network Simulator [10], PDP++ [11]) existen, pero todas estas herramientas son demasiado generales para ser usadas en el campo de las comunicaciones móviles ya que fueron diseñadas para la creación genérica de cualquier tipo de red neuronal y requieren de un usuario experto.

Considerando solamente herramientas pensadas específicamente para la creación de modelos "caja negra" de dispositivos electrónicos, podemos mencionar MπLog [12] cuyo uso es específicamente limitado al modelado de "drivers" analógico-digitales. Otro punto desfavorable de este software, aparte de su especificidad, es que fue desarrollado con el compilador de MatLab®, es decir que requiere que las librerías de este programa estén instaladas para poder funcionar. Una herramienta que trata de solucionar estas desventajas es NeuroModeler®[13], diseñada específicamente para la creación de modelos de dispositivos electrónicos, de cualquier tipo. El problema de esta herramienta es que posee un pobre diseño de interfaz y requiere también mucho conocimiento específico de los modelos neuronales que se quieren crear. Aparte de su diseño poco intuitivo, su mayor punto débil es que solamente genera modelos que pueden implementarse en un simulador de circuitos específico, el cual requiere costosas licencias de instalación y uso.

En resumen, para poder aprovechar al máximo todas las características de las herramientas neuronales existentes, es necesario conocer muy bien el lenguaje y comandos específicos de la herramienta. Se requiere un conocimiento profundo sobre parámetros de modelos neuronales, reglas de aprendizaje, etc., lo cual limita su uso a usuarios expertos. Finalmente, otro punto importante es que en estas herramientas, si se puede crear un modelo neuronal, es muy probable que éste no pueda ser exportado automáticamente para ser utilizado en cualquier simulador de circuitos comerciales disponibles. Por lo tanto no pueden ser utilizados prácticamente para simulaciones reales, p.e. de un sistema de comunicación completo.

En este contexto, surge la necesidad de una herramienta de software fácil de usar para un ingeniero que debe crear y simular modelos basados en RNAs; pero que no conoce en profundidad la teoría neuronal, o posee el conocimiento mínimo que es necesario para la construcción de un modelo "caja negra" y de la configuración de algunos parámetro con facilidad. Este trabajo presenta un prototipo de herramienta de software que intenta satisfacer estos requerimientos. Ésta ha sido diseñada pensando en los usuarios que tienen la intención de utilizar una herramienta poderosa para la creación de modelos neuronales pero que no poseen un conocimiento profundo sobre la teoría de RNAs.

Esta herramienta ha sido programada para ser "open-source" e independiente de la plataforma, lo cual facilitaría especialmente su uso por parte de Universidades, donde el costo de las licencias para programas de este tipo es muchas veces prohibitivo. Algunas de sus características son el soporte a la creación de modelos, edición, entrenamiento, selección de diferentes funciones de activación, testeo y ploteo de los resultados. Particularmente, esta herramienta posee varias ventajas sobre otras herramientas existentes: interfaces "user-friendly" especialmente diseñada para usuarios no-expertos en la teoría de RNAs; soporte para la creación de un modelo neuronal, permitiendo extraer automáticamente las variables de entrada/salida desde los datos de simulaciones o mediciones; un modelo neuronal ya entrenado puede ser exportado directamente como modelo "caja negra" a un simulador de circuitos electrónicos. De esta manera, el modelo neuronal embebido puede utilizarse para simular p.e. de una cadena completa de comunicación inalámbrica,. Esto es representado esquemáticamente en la Figura 2.

La Figura 3 muestra un diagrama de flujo con el proceso principal de desarrollo de un modelo neuronal dentro de la herramienta propuesta. Los archivos de datos de las mediciones de laboratorio son cargados directamente a la herramienta.

sinc(i) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
M. Rubiolo, G. Stegmayer & O. Chiotti; "Neural Networks and Wireless Communications Modeling"
IEEE Latin America Transactions, Vol. 8, pp. 486-492, 2010.

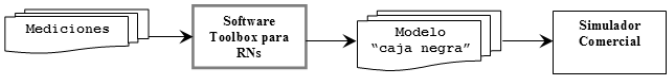

Figura 2. Representación esquemática del funcionamiento de la herramienta de software propuesta en relación a un simulador comercial.

Un punto fuerte a destacar al respecto es que estos archivos no deberán tener un formato específico. Por ejemplo, en el caso de querer crear una red time-delay (TDNN, explicada en el caso de estudio) los datos para entrenamiento de la red se crearán automáticamente a partir de los datos originales, escalonando los mismos según la cantidad de "delays" elegidos para cada variable. Se provee la opción de crear un modelo neuronal en forma manual o sino el modelo neuronal es automáticamente creado, lo que simplifica la tarea de diseño y provee un punto de inicio para el diseñador, quien puede cambiar el diseño de la red originalmente propuesto por la herramienta y configurar algunos parámetros del modelo, tales como el número de neuronas ocultas y las funciones de activación, si así lo desea. El modelo neuronal que se propone automáticamente es un modelo perceptrón multicapa (MLP), debido a sus conocidas propiedades de aproximador universal de cualquier tipo de problemas no-lineales [19].

Una vez que el modelo ha sido definido, puede ser entrenado y/o validado y/o simulado con los datos disponibles, a fin de lograr exactitud en la definición del usuario. Si luego del entrenamiento, la exactitud deseada ha sido alcanzada, el modelo puede ser guardado (exportado) como un archivo de texto el que incluirá el modelo neuronal y los valores de sus parámetros. Esto permitirá luego su implementación como un modelo caja negra dentro de un simulador de circuitos.

Debido al problema de "overfitting" de un modelo neuronal, por el cual ciertos modelos que tienen muy bajo error de aproximación, no se comportan bien cuando se les presentan nuevos datos no vistos en entrenamiento, es que han aparecido ciertas técnicas que tratan de evitar este problema, tal como "cross-validation" [14]. Este procedimiento sugiere dividir los datos totales disponibles en dos subconjuntos: uno para entrenamiento y otro para validación; e ir controlando el error del modelo neuronal en ambos subconjuntos. Cuando la diferencia entre ellos se hace cada vez más grande, el entrenamiento se termina. Este procedimiento ha sido incorporado en la herramienta propuesta con un algoritmo propietario, el cual sugiere al usuario presentar datos para validación de la red además de datos para entrenamiento, y al momento del entrenamiento controla ambos errores..

III. Implementación.

La herramienta de software ha sido implementada en Java usando la herramienta de desarrollo de IBM Eclipse [15] y el Framework JOONE (Java Object Oriented Neural Engine) para creación de RNAs [16]. Este Framework fue seleccionado debido a que provee estructura y algoritmos para el paradigma neuronal, en Java, lo cual nos permite cumplir con el objetivo de diseñar un a herramienta open-source y multi-plataforma al mismo tiempo, lo cual no sería posible de usar otros lenguajes de programación. Además este framework provee, de un modo ya integrado, las clases y algoritmos básicos para entrenamiento y uso de RNAs, que otras implementaciones en C o C++, por ejemplo, no proveen. Además, permite fácilmente su extensión, agregando o definiendo nuevos tipos de modelos neuronales e incluso nuevos algoritmos de entrenamiento. Se puede entrenar un conjunto de Redes Neuronales en paralelo, inicializadas con diferentes pesos, parámetros o diferentes arquitecturas, lo que posibilita al usuario identificar la mejor configuración de red simplemente mediante la evaluación de los resultados luego del proceso de entrenamiento.

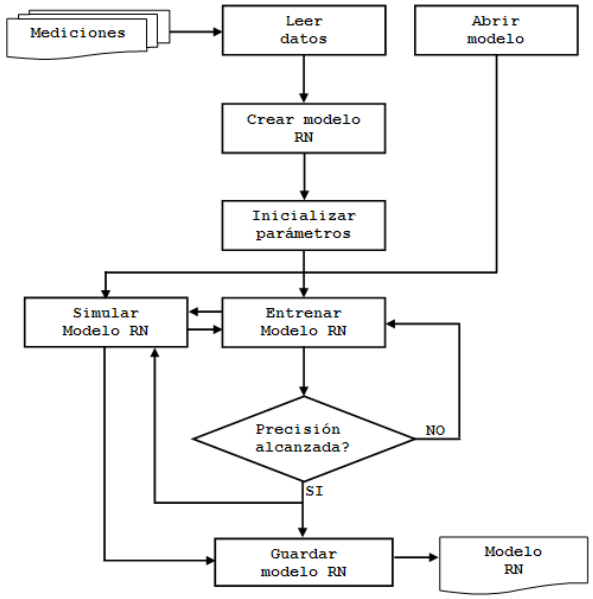

Figura 3. Diagrama de Flujo de la herramienta propuesta para el desarrollo automático de modelos basados en RNAs.

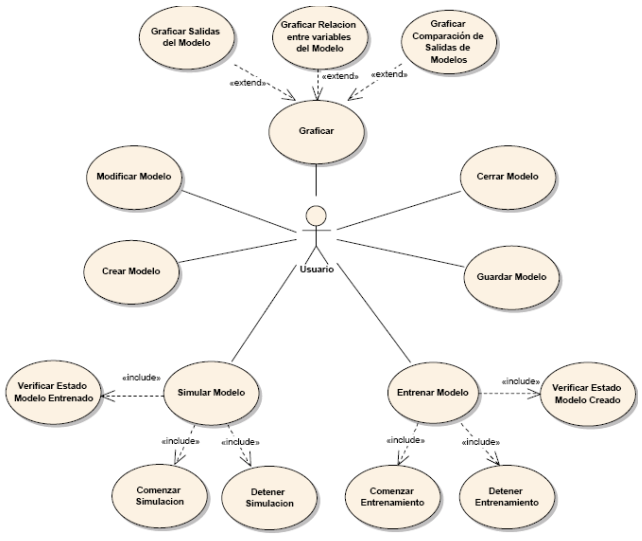

Figura 4. Diagrama de casos de uso UML de la herramienta propuesta.

Otro motivo por el cual fue elegido es que su arquitectura es modular lo cual permite que pueda ser fácilmente extendido,

sinc(i) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
M. Rubiolo, G. Stegmayer & O. Chiotti; "Neural Networks and Wireless Communications Modeling"
IEEE Latin America Transactions, Vol. 8, pp. 486-492, 2010.

p.e. para agregar un algoritmo de entrenamiento. Cada uno de sus componentes puede ser re-utilizado y modificado por los desarrolladores [17]. La aplicación que presentamos en este trabajo, posee embebidos módulos del "Core Engine" de este Framework con el fin de ser utilizados por la herramienta para la creación de arquitecturas neuronales. Es importante aclarar que Joone ha sido desarrollado a fin de ser utilizado por cualquiera que así lo desee, por lo que la licencia de uso es LGPL (Lesser General Public License). De esta manera, cualquier programador puede embeber el "Engine" dentro de nuevas o existentes aplicaciones. En este Framework, cada RNA es compuesta por un número de componentes (capas) conectadas entre sí mediante conexiones (sinapsis). Dependiendo de cómo son conectadas éstas componentes, varias arquitecturas neuronales pueden ser creadas (feedforward, recurrente, etc.). Cada capa es implementada como un objeto, que puede ser ejecutado independientemente de los otros (obteniendo los datos de entrada a la capa, aplicando la función de transferencia a estos datos y colocando los resultados en la sinapsis que comunica con la capa que los recibe). Este mecanismo también es utilizado para obtener el error de entrenamiento, permitiendo a los pesos y bias ir cambiando de acuerdo al algoritmo de entrenamiento. Para la descripción de las características principales de la herramienta se usará UML y diagramas de clases, debido que el software propuesto está basado en el modelo de objetos. La figura 4 muestra un diagrama de casos de uso en UML de la herramienta.

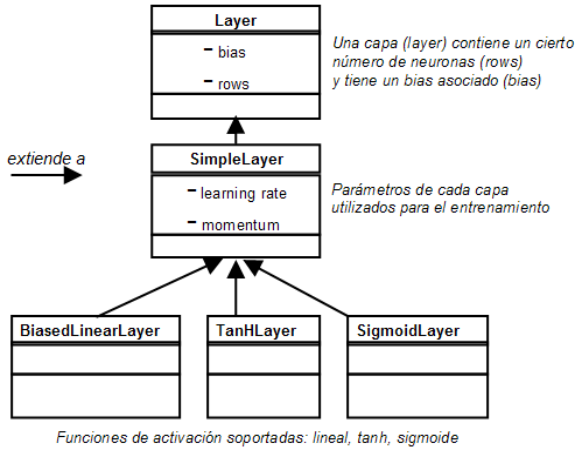

Figura 5. Diagrama simplificado de la clase *Layer*. Modelo que representa la jerarquía de objetos utilizados para representar cada capa de una RNA.

La Figura 5 presenta un diagrama de clase UML simplificado del elemento básico de una red neuronal, una capa (clase *Layer*). Un objeto de este tipo está compuesto de neuronas (se representan con filas o *rows*), posee un *bias*, y es extendido en el objeto *SimpleLayer* a fin de incorporar los parámetros utilizados por este objeto a la hora de responder al proceso de entrenamiento. Éstos son *learning rate* y *momentum*. A su vez, según la función de activación que presentan las neuronas de una capa (todas comparten la misma función de activación), se extienden los objetos *LinearLayer*, *TanhLayer* y *SigmoidLayer*¸ reflejando la utilización de las funciones lineal, tangente hiperbólica y sigmoide, respectivamente.

Otro de los componentes básicos es una sinapsis (clase *Synapse*), que representa la conexión entre dos capas. Permite pasar información (los datos utilizados para el entrenamiento o el error a fin de actualizar los pesos) de capa a capa. Durante el proceso de entrenamiento, los pesos de cada conexión son modificados de acuerdo al algoritmo de aprendizaje. La sinapsis utilizada por nuestra herramienta para realizar la conexión entre capas es la llamada *FullSynapse*, presente en la Figura 6. Esta clase implementa un mecanismo de relación de neuronas entre capas "todos con todos" sea cual sea el número de neuronas presentes en las capas que conecta, representando así el tipo de sinapsis más comúnmente utilizado en una RNA.

Por otra parte, el Framework, brinda soporte para la importación y exportación de datos desde archivos de texto, concepto fuertemente utilizado por nuestra herramienta a fin de alcanzar el objetivo de desarrollar un soporte ampliamente independiente del perfil del usuario que la utilice, sea éste conocedor de la teoría neuronal, o no. Es utilizado el objeto *FileInputSynapse* para realizar la incorporación de los datos presentes en un archivo de texto a la arquitectura neuronal representada por el conjunto de objetos que hayamos creado. Por su parte, es utilizado *FileOutputSynapse* para realizar lo propio, siendo ahora el objetivo la exportación del modelo neuronal resultante. Ambos objetos extienden de *StreamInputSynapse* y *StreamOutputSynapse* respectivamente, incluidas a fin de soportar los procesos de importación y exportación en otros formatos.

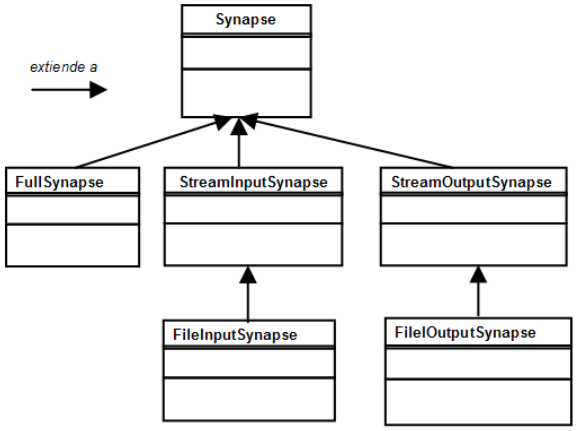

Figura 6. Diagrama simplificado de la clase *Synapse*. Modelo que representa la jerarquía de objetos utilizados para representar conexiones entre capas de una RNA.

La relación entre los objetos instanciados de las clases anteriormente explicadas en el momento de la creación de una RNAs, sea cual sea su configuración particular, surgirá como consecuencia de la utilización de las clases extendidas de las principales, según la característica particular a representar (p.e. una *SigmoidLayer* que extiende de una *Layer* a fin de representar un conjunto de neuronas con función de activación de tipo sigmoide). Al crearse una nueva red neuronal se instancia un objeto de la clase *NeuralNet*, donde cada capa de esta red será representada con una instancia de alguna clase

sinc(i) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
M. Rubiolo, G. Stegmayer & O. Chiotti; "Neural Networks and Wireless Communications Modeling"
IEEE Latin America Transactions, Vol. 8, pp. 486-492, 2010.

que extienda a la clase *Layer* (el tipo de clase dependerá de la función de activación deseada). Cada sinapsis se representa con una clase que extienda a *Synapse* (en nuestro caso será la clase *FullSynapse*). El objeto *NeuralNet* permite manejar, serializar (guardar) y recuperar la red como un todo. Cuando se genera una nueva Red se crea además una instancia de la clase *Monitor*, este objeto permite tener un control centralizado de la red, manejando parámetros internos como learning rate, número de épocas de entrenamiento, etc. Otra función del monitor es proveer información (mediante la generación de eventos) a aplicaciones que utilicen el C*ore Engine* de Joone (por ej: comienzo o fin del entrenamiento, cambio en los valores de error, etc.). En cuanto a los datos de entrada (inputs, targets, etc) estos son representados a través de objetos de la clase *Pattern*. La clase *Pattern* contiene un arreglo con los valores y un contador con la cantidad de valores (utilizado para determinar cuándo se debe detener el proceso). El uso dinámico de la herramienta propuesta se ejemplifica a través de un caso de estudio en la siguiente sección.

IV. Caso de estudio.

En esta Sección se muestra un caso de estudio, donde la herramienta desarrollada dá soporte para la creación de un tipo de modelo neuronal que es de especial interés para el modelado de comportamiento no lineal y dinámico en dispositivos electrónicos que son parte de una cadena de comunicación inalámbrica o móvil [18] como ser los modelos basados en redes neuronales TDNN.

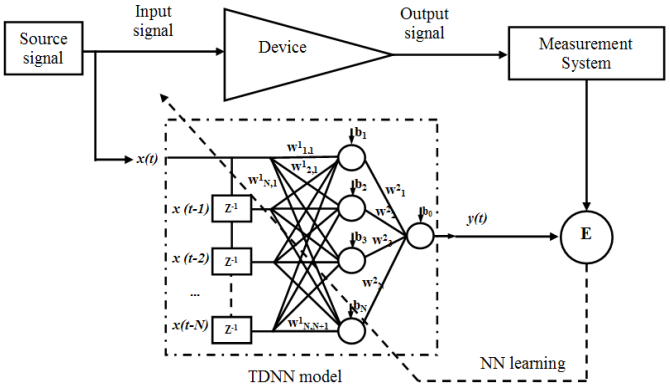

Figura 7. Modelo TDNN de un dispositivo electrónico entrenado directamente con las mediciones del dispositivo.

Una red TDNN está basada en el modelo perceptron multicapa (MLP) con el agregado de líneas de delay (Z^{-1}) que introducen valores históricos de las señales de entrada al modelo, necesarios para modelar los efectos de memoria en componentes tales como amplificadores de potencia (PAs)[19]. La Figura 7 muestra cómo la señal de salida de un dispositivo, en respuesta a un estímulo de entrada, p.e. una señal a ser amplificada por un PA, es muestreada por un sistema que produce datos que son utilizados para entrenar el modelo TDNN en modo supervisado. Una vez que el modelo TDNN ha aprendido el comportamiento no-lineal y dinámico del dispositivo en estudio, puede ser utilizado como un modelo de caja negra dentro de un simulador de circuitos y probado bajo diferentes condiciones de trabajo. El proceso de construcción de un modelo TDNN, entrenamiento e importación dentro de un simulador de circuitos es completamente soportado por la herramienta de software que estamos presentando en este trabajo.

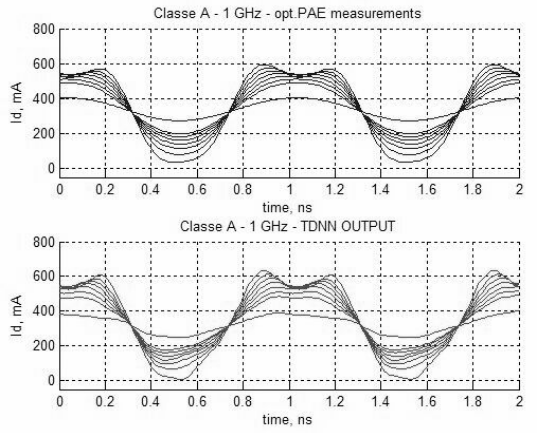

Figura 8. Entrenamiento: mediciones de la corriente I_{DS} en el dominio del tiempo, frecuencia 1 GHz, para un PA trabajando en clase A. Mediciones (arriba), modelo TDNN (abajo).

Las mediciones usadas para entrenar el modelo se muestran en la Figura 8, en la parte superior; la parte inferior muestra la aproximación lograda por el modelo TDNN generado. Estas mediciones son cargadas en la herramienta y automáticamente se crea una red TDNN a partir de las variables de entrada y salida contenidas en los datos, como muestra la Figura 9. Un detalle de la clase *TDNN* se muestra en la Figura 10.

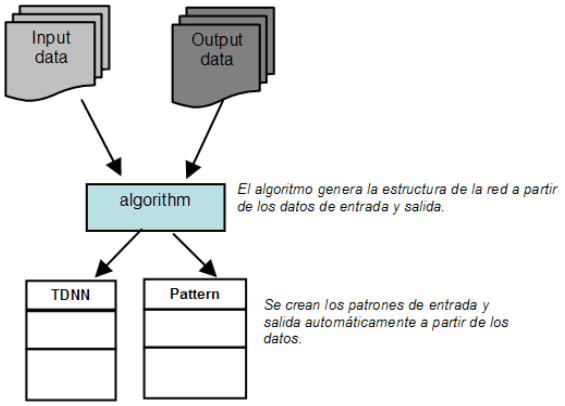

Figura 9. Creación automática de un modelo neuronal a partir de los datos.

La clase TDNN representa la red neuronal de tipo time-delay (estructurada como una red MLP de 3 capas). Esta permite manejar los distintos elementos de la red. La clase BiasedLinearLayer representa las capas de entrada y salida con función de activación lineal (utiliza bias), la clase TanHLayer representa la capa oculta con función de activación tangente hiperbólica. La clase Pattern representa los datos de entrada (entrenamiento, validación, etc.) y la clase Monitor realiza el control de los distintos procesos que ejecuta la red.

sinc(i) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
M. Rubiolo, G. Stegmayer & O. Chiotti; "Neural Networks and Wireless Communications Modeling"
IEEE Latin America Transactions, Vol. 8, pp. 486-492, 2010.

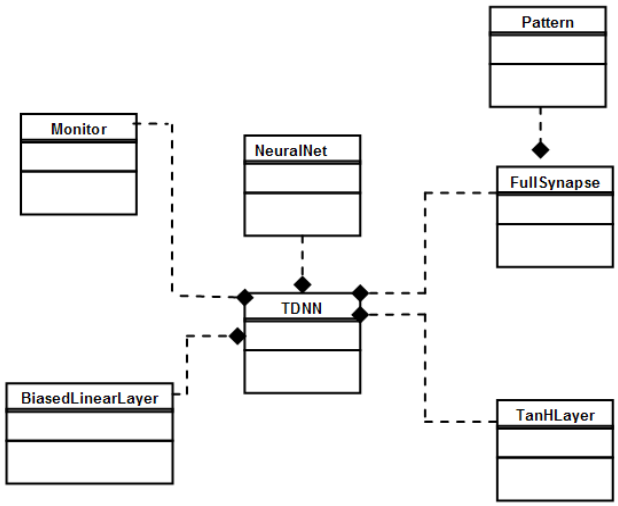

Figura 10. Diagrama simplificado de la clase *TDNN*.

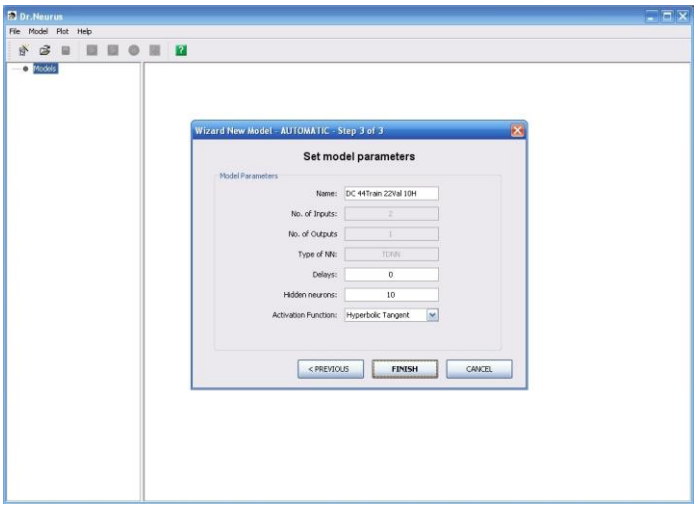

Figura 11. Creación automática en la herramienta propuesta de un modelo *TDNN* a partir de archivos de datos.

La figura 11 muestra una captura del wizard automático que ayuda a un usuario no experto en RNA a obtener el modelo del sistema, en el cual solamente deben especificarse los archivos que contienen los datos.

La figura 12 muestra capturas de la pantalla principal de la herramienta, donde se muestra la barra de navegación a la izquierda que asocia a cada modelo creado, las cosas que pueden hacerse sobre el modelo (entrenarlo, simularlo, graficarlo, compararlo con otros modelos, generar un reporte, etc.) y muestra al usuario en todo momento sobre qué modelo está trabajando y qué opción del menú está activa.

La Figura 13 muestra el template que se puede usar dentro un simulador de circuitos para usar el modelo neuronal creado como un modelo caja negra. En el template, el elemento NETLIST INCLUDE apunta a un archivo .txt que contiene el modelo neuronal con todos sus parámetros y sus respectivos valores. Este archivo ha sido generado por la herramienta de software propuesta en este trabajo.

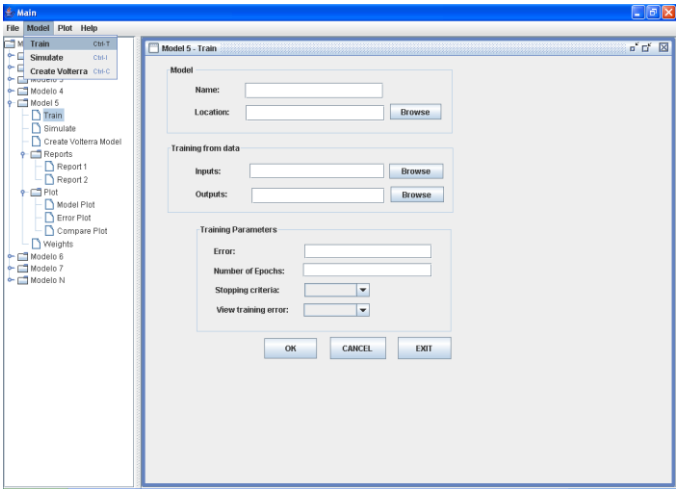

Figura 12. Captura de la pantalla principal de la herramienta de software.

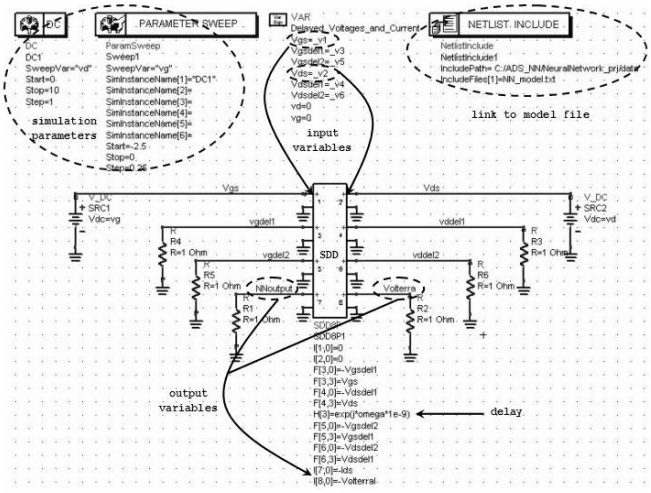

Figura 13. Template para la implementación del modelo neuronal como modelo caja negra en un simulador de circuitos.

V. Conclusiones.

En este trabajo hemos presentado una herramienta de software libre para dar soporte a la creación de modelos basados en Redes Neuronales Artificiales.

Esta herramienta de software permite la creación automática de modelos neuronales usando, p.e. las mediciones hechas en laboratorio de un dispositivo electrónico, simplificando la tarea de diseño a un ingeniero electrónico que luego quiere usar el modelo como caja negra dentro de un simulador de circuitos. Una de las ventajas principales de esta herramienta es que puede ayudar a reducir el tiempo de diseño de un modelo, automatizando las tareas de creación, definición y simulación de un modelo neuronal, especialmente importante para usuarios que no tienen conocimiento profundo acerca de la teoría neuronal.

sinc(i) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
M. Rubiolo, G. Stegmayer & O. Chiotti; "Neural Networks and Wireless Communications Modeling"
IEEE Latin America Transactions, Vol. 8, pp. 486-492, 2010.

VI. REFERENCIAS

[1] C. Evci, U. Barth, P. Sehier and R. Sigle, "The path to beyond 3G systems: strategic and technological challenges", en Proc. 4th Int. Conf. on 3G Mobile Communication Technologies, London, England, pp. 299-303, 2003.

[2] A. Ahmed, M. Abdalla, E. Mengistu and G. Kompa, "Power Amplifier Modeling Using Memory Polynomial with Non-Uniform Delay Taps", in Proc. IEEE 34th European Microwave Week, pp. 1457-1460, 2004.

[3] H. Ku and J. Kenney, "Behavioral Modeling of Nonlinear RF Power Amplifiers Considering Memory Effects", *IEEE Trans. Microwave Theory Tech.*, vol. 51, no. 12, pp. 2495-2504, 2004.

[4] H. Qian and G. Zhou, "A Neural Network Predistorter for Nonlinear Power Amplifiers with Memory", in Proc. 10th IEEE DSP Workshop, pp. 312-316, 2002.

[5] Q. Zhang, K. Gupta and V. Devabhaktuni, "Artificial Neural Networks – From Theory to practice", *IEEE Trans. Microwave Theory Tech.*, vol. 51, no. 12, pp. 1339-1350, 2003.

[6] M. Meireles, P. Almeida and M. Simoes, "A comprehensive review for industrial applicability of Artificial NN", *IEEE Trans. Industrial Electronics*, vol. 50, no. 3, pp. 585-601, 2003.

[7] http://www.mathworks.com/products/neuralnet/

[8] http://www.tropheus.demon.co.uk/nplan.htm

[9] http://www.nd.com/download.htm

[10] http://tedlab.mit.edu/~dr/Lens/

[11] http://www.cnbc.cmu.edu/Resources/PDP++//PDP++.htm

[12] I. Stievano, I. Maio and F. Canavero, "M[pi]log, Macromodeling via Parametric Identification of Logic Gates", *IEEE Trans. on Advanced Packaging*, vol. 27, no. 2, pp. 15-23, 2004.

[13] http://web.doe.carleton.ca/~qjz/qjz.html

[14] J. Sjoberg and L. Ljung, "Overtraining, regularization and searching for a minimum, with application to Neural Networks", *Int. Journal of Control*, no. 62, pp. 1391-1407, 1995.

[15] http://www.research.ibm.com/eclipse/

[16] http://www.jooneworld.com/

[17] P. Marrone. *JOONE The complete guide All you need to know about Joone*. 2005.

[18] D. Root and J. Wood. *Fundamentals of nonlinear behavioral modeling for RF and microwave design*. Ed Artech House, Boston, 2005.

[19] G. Stegmayer, O. Chiotti, "Volterra NN-based behavioral model for new wireless communications devices", *Neural Computing and Applications*, vol. 18, pp. 283-291, 2009.

sinc(i) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
M. Rubiolo, G. Stegmayer & O. Chiotti; "Neural Networks and Wireless Communications Modeling"
IEEE Latin America Transactions, Vol. 8, pp. 486-492, 2010.