Entrenamiento de una red neuronal en LabVIEW para la identificación en línea de un sistema dinámico

Juan José Martínez Nolasco
Instituto Tecnológico de Celaya
juan.martinez@itcelaya.edu.mx

Carlos Fernando Hernández Figueroa
Instituto Tecnológico de Celaya
karlozfernando@hotmail.com

Francisco Gutiérrez Vera
Instituto Tecnológico de Celaya
francisco.gutierrez@itcelaya.edu.mx

Agustín Vidal Lesso
Instituto Tecnológico de Celaya
agustin.vidal@itcelaya.edu.mx

Abdón Javier Ruiz Guerrero
Instituto Tecnológico de Celaya
abdon.ruiz@itcelaya.edu.mx

RESUMEN

En el presente trabajo se describe el diseño de un instrumento virtual implementado en LabVIEW para el uso, creación y entrenamiento de redes neuronales artificiales mediante el algoritmo de retropropagación. Como prueba de su correcto funcionamiento se realiza la identificación de un sistema dinámico en línea y la adaptación del instrumento virtual para generar redes recurrentes; el sistema dinámico es un sistema resistivo – capacitivo. Por último, se comparan los resultados de varias redes entrenadas con diferentes estructuras.

Palabras Clave*:* Redes neuronales artificiales, LabVIEW y entrenamiento en línea.

1. INTRODUCCIÓN

La identificación de un sistema dinámico es imprescindible para la predicción y para un adecuado control del mismo. La mayoría de los sistemas están conformados con un gran número de parámetros que intervienen entre sí en menor o mayor medida, los cuales pueden o no relacionarse linealmente (o inclusive pueden ser variantes en el tiempo). Esto tiende a generar problemas al realizar el modelo matemático que, irrevocablemente, involucra el conocimiento extensivo del propio sistema.

En ausencia de resultados teóricos concretos necesarios para modelar un sistema, las redes neuronales artificiales presentan una excelente alternativa para la solución de este tipo de problemas.

Las redes neuronales están constituidas por un gran número de neuronas, conectadas en forma masiva. Las redes neuronales biológicas pueden establecerse como grupos de neuronas activas especializadas en tareas como: cálculos matemáticos, posicionamiento y memoria. Las redes neuronales procesan información, no requieren de modelos de referencia, aprenden a realizar nuevas tareas y se adaptan con facilidad a ambientes cambiantes. Así pues, las redes neuronales artificiales tratan de extraer las excelentes capacidades del cerebro para resolver ciertos problemas complejos, como visión, reconocimiento de patrones o control motosensorial.

LabVIEW (en conjunto con una DAQ) es una plataforma de programación ideal para la adquisición y procesamiento de señales. Por lo tanto un programa de entrenamiento de redes neuronales en LabVIEW es una excelente opción para el entrenamiento en línea.

LabVIEW (acrónimo de Laboratory Virtual Instrumentation Engineering Workbench) es un entorno de programación gráfico recomendado para la implementación de sistemas de pruebas, control y diseño.

Existe una muy escasa variedad de programas que entrenan redes neuronales en LabVIEW, sin embargo, no aprovechan las características de LabVIEW para la adquisición de señales **[3]**. Además de que no siguen técnicas de programación adecuadas y no implementan el entrenamiento en línea.

Otra alternativa para la integración de las redes neuronales es mediante una herramienta en LabVIEW llamada "*matlab script node"* que ejecuta al software matlab, el cual tiene un toolbox especializado en redes neuronales, para ejecutar códigos en un VI **[5]**.

2. MÉTODOS

2.1 Consideraciones teóricas

2.1.1 Modelo de una red neurona artificial

Las RN se modelan mediante unidades fundamentales llamadas neuronas artificiales. Cada neurona artificial es representada por un conjunto de enlaces sinápticos, un enlace de polarización y un enlace de activación. Los enlaces sinápticos ponderan la señal de entrada, multiplicándola por los pesos sinápticos. La suma de las señales *pesadas* define el potencial de activación de la neurona (v_n). Y por último, el enlace de activación transforma el potencial de activación en la señal de salida (y_n). En la figura 1 se muestra el modelo de una neurona artificial.

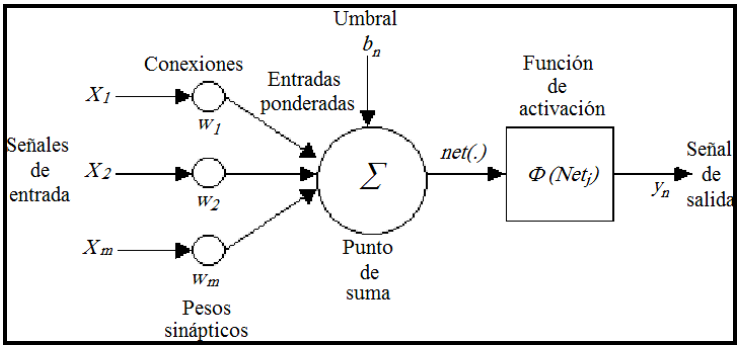

Figura 1. Modelo de una neurona artificial.

En el modelo de una neurona artificial presentado en la figura 1 se pueden identificar cinco elementos:

- **Enlaces de conexión**. Son parametrizados por los pesos sinápticos w.
- **Sumador** (Σ). Suma los componentes de las señales de entrada multiplicadas por w.
- **Función de activación**. Denotadas por φ, definen la salida de la neurona en función del potencial de activación; existen tres tipos básicos: la función escalón o umbral, función lineal y la función sigmoidal.
- **Umbral o polarización** b_n. Desplaza a la suma de las entradas ponderas.
- **La señal de salida** y_n. Puede ir a muchas entradas de otras neuronas.

La salida de una neurona puede ser, a su vez, la entrada de otras neuronas. Esto da lugar a tres diferentes arquitecturas de redes:

1. Redes unicapa: Es el caso más simple, la capa de entrada se conecta directamente a la capa de neuronas de salida por medio de las sinapsis (Figura 2).

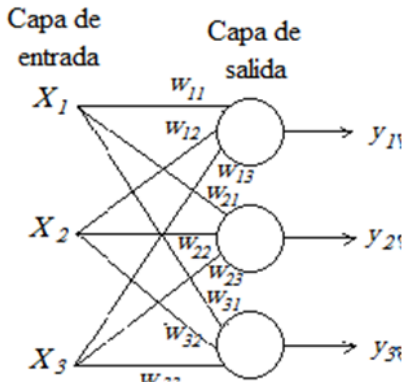

Figura 2. Estructura unicapa.

2. Redes multicapa: Se distinguen por tener una o más capas ocultas, cuyos nodos computacionales se denominan neuronas ocultas (Figura 3).

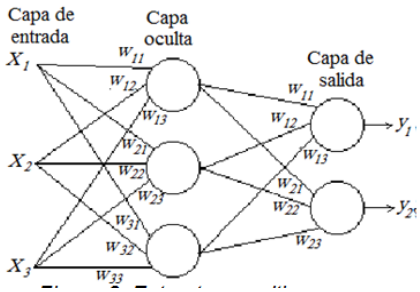

Figura 3. Estructura multicapa.

3. Redes recurrentes: Tienen al menos un lazo de retroalimentación donde se introduce un operador de retardo unitario q^{-1}. También pueden ser unicapa o multicapa (figura 4) **[1]**.

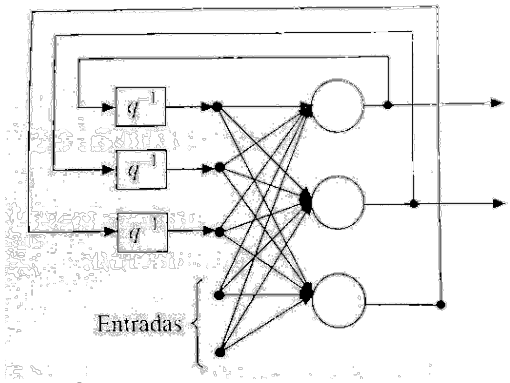

Figura 4. Estructura recurrente.

2.1.2 Entrenamiento

La red neuronal aprende variando sus pesos sinápticos y el umbral de polarización. El entrenamiento es un proceso por el cual la propia red es modificada ante una señal de entrada externa del medio. El tipo de aprendizaje está determinado por la forma en que se realiza dicha modificación.

Existen distintos algoritmos de entrenamiento, algunos son:

- Corrección del error (gradiente descendiente).
- Boltzman,
- Regla de Hebb.
- Aprendizaje Competitivo.
- Ley de Konohen.

La decisión del algoritmo de aprendizaje depende de características como la aplicación de la red (reconocimiento de patrones, predicción, etc.), estructura de la red, entre otros **[2]**.

2.1.3 Algoritmo de retropropagación

El algoritmo de retropropagación es una técnica específica para implementar el método del gradiente descendiente en un espacio de pesos, para una red multicapa.

Procedimiento:

1. *Inicialización:* Determinar la estructura de la red y los valores iniciales de los pesos.
2. Seleccionar una época de pesos y verificar el error promedio.
3. Presentar un patrón.

4. *Etapa hacia adelante:* Se fijan los parámetros de la red y se presenta una señal de entrada a la red, que se propaga para producir la salida.
5. *Etapa hacia atrás:* El error entre la salida deseada y la red se propaga hacia atrás. Los parámetros de la red se modifican para minimizar el cuadrado de dicho error.

2.1.4 Desarrollo del algoritmo de retropropagación

Considerando una red multicapa como la que se muestra en la Figura 5 y la notación descrita, el j-ésimo componente del error de aproximación se puede expresar como:

$$e_j(k) = d_j(k) - y_j(k)$$

Entonces la suma de los errores instantáneos al cuadrado se formula como:

$$\varepsilon(n) = \frac{1}{2}\sum_{j \in C}^{l} e_j^2(k)$$

Donde C es el conjunto de las neuronas de salida, *C= {1,2,…, l}*.

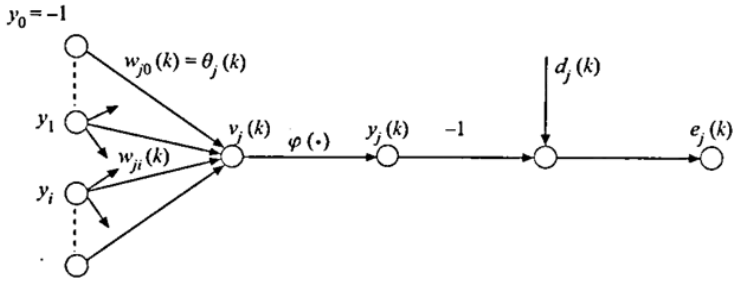

Figura 5. Algoritmo de retropropagación, neurona en capa de salida.

El objetivo es minimizar el error con respecto a los pesos, para esto se usa una solución aproximada a partir del gradiente, es decir, es la dirección en la que la función se aproxima a sus mínimos.

El gradiente del error con respecto a los pesos se define como:

$$\frac{\partial \varepsilon(k)}{\partial w_{ji}(k)}$$

Por la regla de la cadena se tiene:

$$\frac{\partial \varepsilon(k)}{\partial w_{ji}(k)} = \frac{\partial \varepsilon(k)}{\partial e_j(k)} \frac{\partial e_j(k)}{\partial y_j(k)} \frac{\partial y_j(k)}{\partial v_j(k)} \frac{\partial v_j(k)}{\partial w_{ji}(k)}$$

$$\frac{\partial \varepsilon(k)}{\partial e_j(k)} = e_j(k)$$

$$\frac{\partial e_j(k)}{\partial y_j(k)} = -1$$

$$\frac{\partial y_j(k)}{\partial v_j(k)} = \varphi'\big(v_j(k)\big)$$

$$\frac{\partial v_j(k)}{\partial w_{ji}(k)} = y_i(k)$$

Definimos el gradiente local como la siguiente expresión:

$$\delta_j(k) = \frac{\partial \varepsilon(k)}{\partial v_j(k)} = e_j(k)\varphi'\big(v_j(k)\big)$$

Así, tenemos que:

$$\frac{\partial \varepsilon(k)}{\partial w_{ji}(k)} = \delta_j(k) y_i(k)$$

A partir de la ecuación anterior se observa que el cálculo del gradiente local se involucra a la señal de error *e(k)* de la neurona de salida *j*. En este contexto se puede identificar un caso adicional cuando la neurona *j* está situada en alguna capa oculta y no en la capa de salida.

El valor deseado de dicha neurona no está disponible (Figura 6). Por lo tanto, la respectiva señal de error se debe determinar de manera recursiva en función de la señal de error de las neuronas de salida. Se puede redefinir el gradiente local para una neurona oculta como:

$$\delta_j(k) = \sum_n \left[w_{nj}(k)\delta_n(k) \right] \varphi'\left(v_j(k)\right)$$

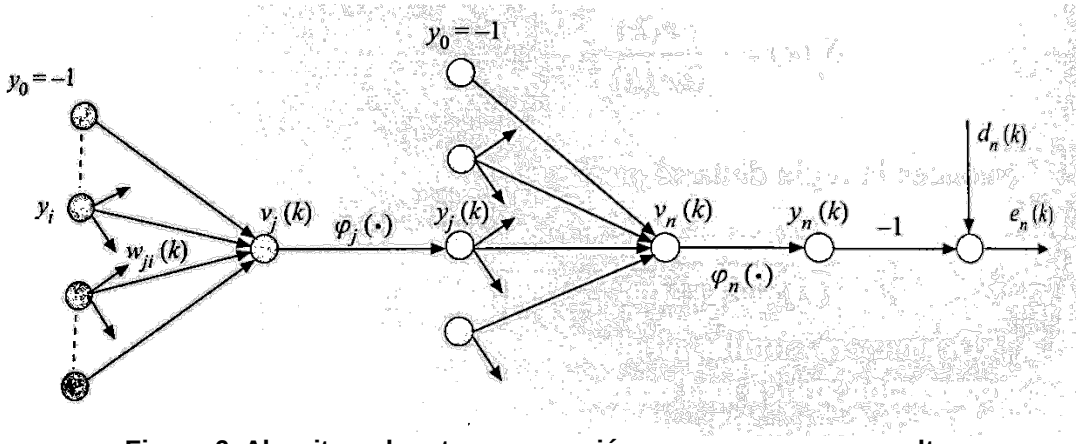

Figura 6. Algoritmo de retropropagación, neurona en capa oculta.

El algoritmo de retropropagación aplica una corrección $\Delta w_{ji}(k)$ al peso sináptico $w_{ji}(k)$, la cual es proporcional al valor del gradiente local. La siguiente ecuación se le denomina regla delta:

$$\Delta w_{ji} = \eta \delta_j(k) y_i(k)$$

donde: η es la tasa o razón de aprendizaje.

Una forma de incrementar el parámetro de la velocidad de aprendizaje de modo que se evite la posibilidad de inestabilidad consiste en modificar la regla delta, incluyendo un término de momento **[4]**:

$$\Delta w_{ji} = \eta \delta_j(k) y_i(k) + \alpha \Delta w_{ji}(k-1)$$

2.2 Diseño de instrumentos

El VI (Instrumento Virtual) de entrenamiento de redes neuronales artificiales en LabVIEW se diseñó con las siguientes características:

- Crear una red neuronal artificial con una estructura deseada (número de capas, neuronas por capa, funciones de activación).
- Entrenar la red con el algoritmo de retropropagación.
- Visualizar la respuesta de la red y la señal a aprender durante el entrenamiento.
- La posibilidad de cargar una red existente o guardar la red entrenada.
- Calcular el promedio del error al cuadrado (MSE).
- Visualizar los pesos de la red.
- Controlar manual o automáticamente (amplitud y tiempo de cambio aleatorios) el voltaje de salida de la DAQ para alimentar la entrada de un sistema dinámico.
- Adquirir mediante la DAQ la respuesta en voltaje del sistema a identificar.
- Modificar la tasa de aprendizaje y el momento durante el entrenamiento.

La interface con el usuario del instrumento virtual se presenta en la figura 7.

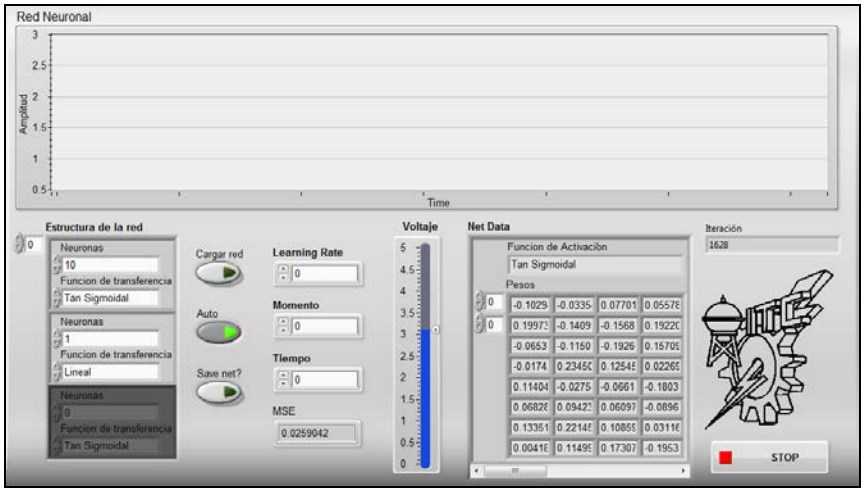

Figura 7. Panel frontal del VI.

En el diagrama de bloques del VI (figura 8) se observan 2 registros de corrimiento con 4 retardos unitarios cada uno, esto implica que la red a crear (independientemente de la estructura general) es de tipo **recurrente** de cuarto orden. También se puede observar que se crea una red con 8 entradas que corresponden a cada salida de los registros de corrimiento.

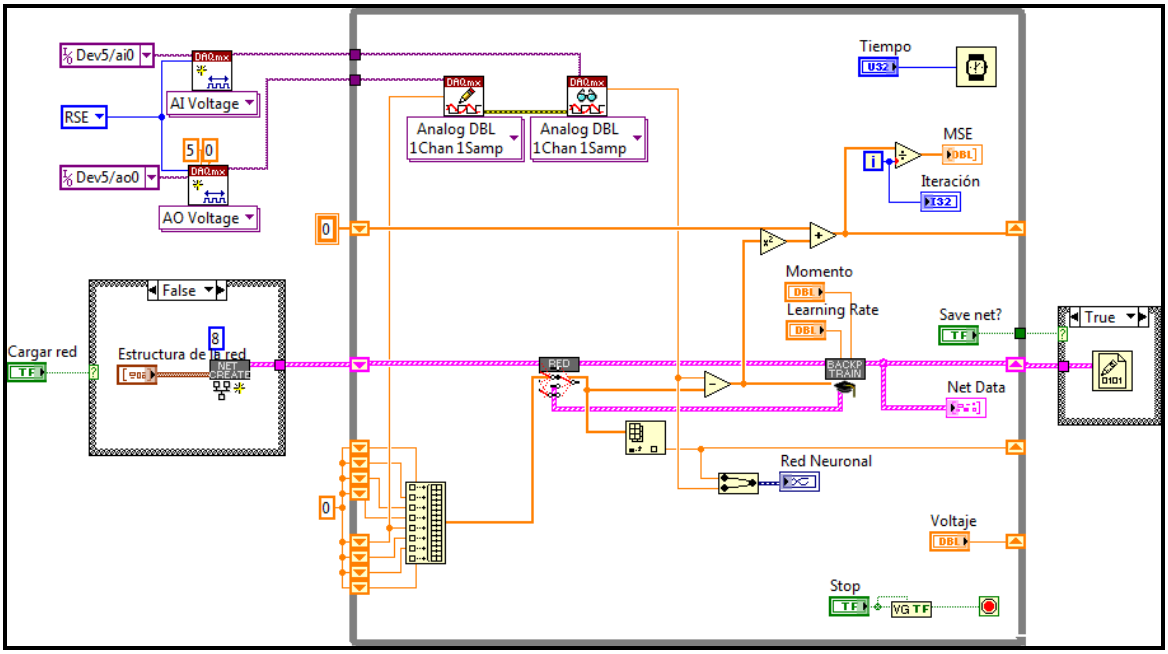

Figura 8. Diagrama de bloques del VI.

Se programaron una serie de distintos subinstrumentos virtuales (subVI's) con el motivo de descomponer el problema y emplear una buena técnica de programación, además de hacer bloques funcionales de código para futuros programas.

El subVI "NET CREATE" (figura 9) genera una matriz de clúster (*Net Data*) que incluye pesos aleatorios en un rango numérico y una función de activación para cada capa de la red (φ). El esquema que se emplea es el de una red totalmente conectada. Como entradas del subVI, se tiene el número de entradas de la red y la estructura (número de

neuronas y función de activación) de cada capa. Además se incluye un incremento para considerar el umbral de polarización en las neuronas.

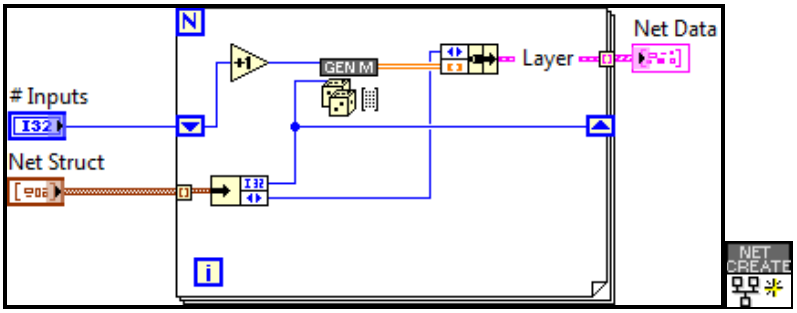

Figura 9. Diagrama de bloques del subVI Net Create.

El subVI “NET” (figura 10) devuelve la salida de la red a partir de un vector de entradas y de la variable *Net Data* (Pesos y Función de activación para cada capa), para esto las salidas de la capa anterior se vuelven las entradas de la siguiente capa ($y_i(k) = x_j(k)$), lo que se logra con un registro de corrimiento. También como salida esta una matriz de vectores de entrenamiento que genera el subVI “LAYER” en cada capa.

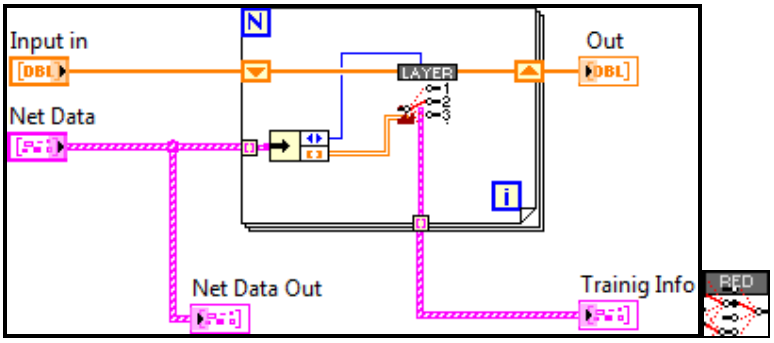

Figura 10. Diagrama de bloques del subVI NET.

El subVI “LAYER” (figura 11), a partir de los pesos (W), las entradas (x) y la función de transferencia (φ), calcula matricialmente la salida de una capa de la red (WX). También añade a las entradas un 1, que representa el umbral de polarización, es decir, al

multiplicarse por el peso correspondiente da el mismo resultado que si se sumará separadamente. Por último, almacena información necesaria para el algoritmo de retropropagación: la derivada de la función de activación evaluada en el potencial de activación ($\varphi'(v_j(k))$) y las entradas de la capa ($x_j(k)$).

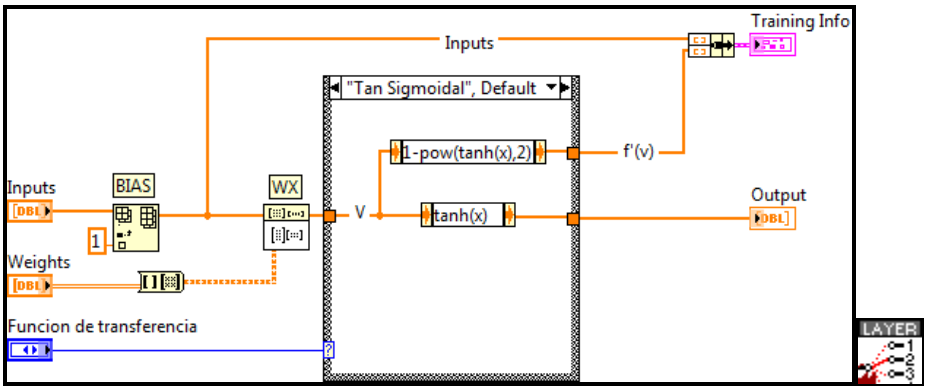

Figura 11. Diagrama de bloques del subVI Layer.

El subVI de entrenamiento llamado “BACKPROPAGATION TRAIN” (figura 12) consta de los siguientes pasos:

1. Se invierten las matrices *Training Info* y *Net Data* (pesos) para poder usar la información desde la última capa de salida hasta la primera capa de entrada. En el ciclo FOR se indexan estas matrices, y se obtiene información capa por capa.
2. En la primera iteración del ciclo FOR se calcula el gradiente local de la última capa a partir del error, éste es mandado al registro de corrimiento para la siguiente iteración para la capa anterior ($e_j(k)\varphi'(v_j(k))$).
3. En la siguiente iteración del ciclo se calculan los gradientes locales para cada una de las neuronas de la capa en cuestión a partir de los pesos y de los gradientes de la capa posterior. Nuevamente los gradientes locales pasan a la

capa anterior, y se repite este paso para el número de capas ocultas restantes ($\sum_n [w_{nj}(k)\delta_n(k)]\varphi'(v_j(k))$).

4. Para cada iteración se calcula Δw_{ji} y se suma en la matriz *Net Data.* Todos los Δw_{ji} calculados se disponen en una matriz y se almacenan en el registro del ciclo WHILE (que solo se ejecuta una sola vez por entrenamiento) para la siguiente ocasión y así poder emplear el término de momento ($\alpha \Delta w_{ji}(k-1)$).

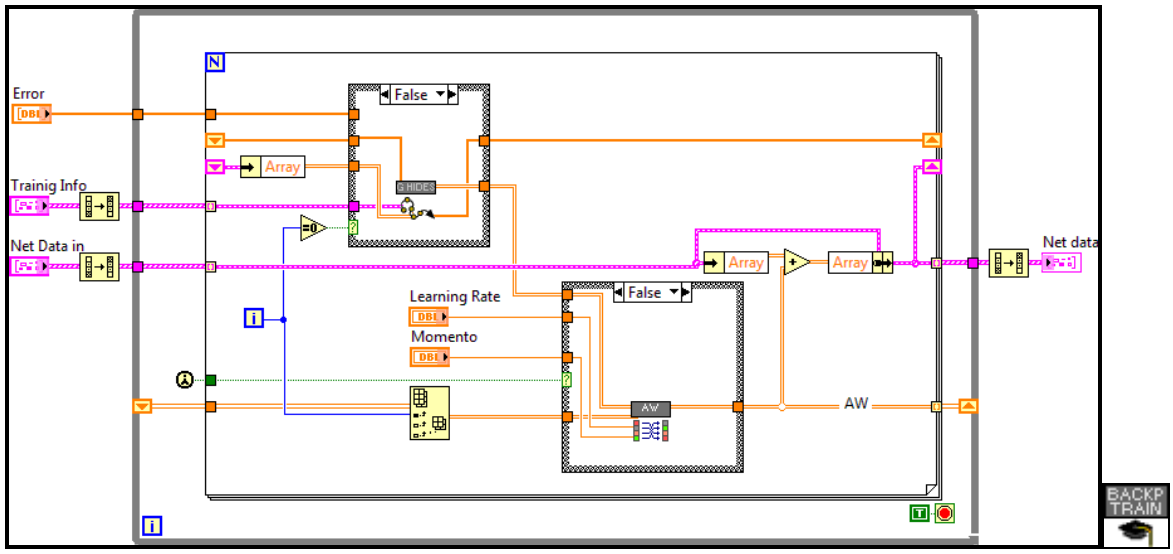

Figura 12. Diagrama de bloques "Backpropagation Train".

2.3 Diseño de la muestra

Para la comprobación del algoritmo implementado, se eligió un sistema dinámico simple que constó de un circuito eléctrico compuesto por un resistor y un capacitor conectados en serie como se muestra en la figura 13.

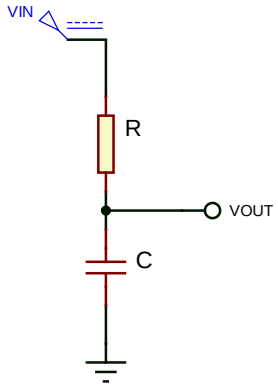

Figura 13. Sistema a identificar (Circuito RC).

Los valores que se usaron en el circuito son:

- Vin (Entrada del sistema): 0 a 5v
- R: 330K
- C: 2.2uF

Dicho circuito se conectó a una entrada y salida analógicas de una tarjeta de adquisición de datos (DAQ 6009).

3. RESULTADOS

Los resultados se obtuvieron después de cada entrenamiento con aproximadamente 2000 iteraciones para cada prueba. Se probaron distintas arquitecturas de redes con dos tipos de función de activación: lineal y tan-sigmoidal (tangente hiperbólica). El tiempo entre cada iteración fue de 1 milisegundo. También la entrada del circuito se genera aleatoriamente de 0-5v cambiando de 1-1000 ms cada vez.

Para cada una de las pruebas más significativas se despliega una gráfica que muestra la señal del sistema (en negro) y la señal de salida de la red neuronal (en naranja)

durante 30 segundos (Figuras 14-17). También se muestran el error (MSE) y la estructura (número de neuronas y función de activación por capa).

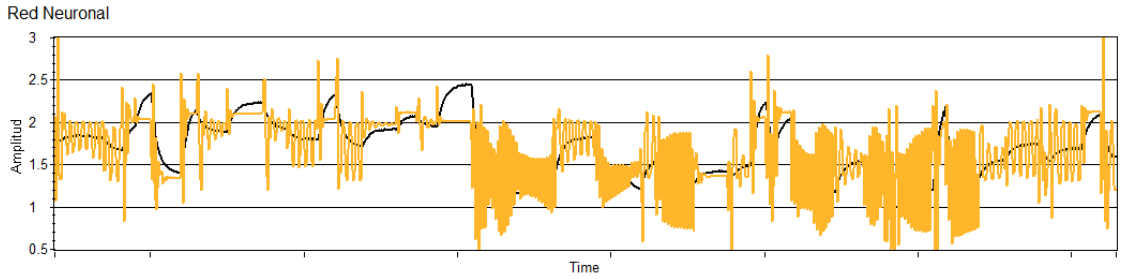

Figura 24. Respuesta de la red: MSE 0.762, Estructura 10 Tanh-1Lineal (Inicialización de los pesos >1).

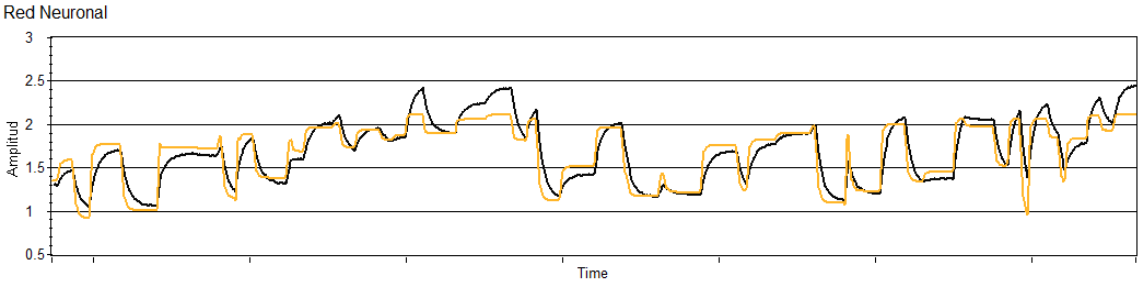

Figura 35. Respuesta de la red: MSE 0.074, Estructura 10 Tanh-1Lineal.

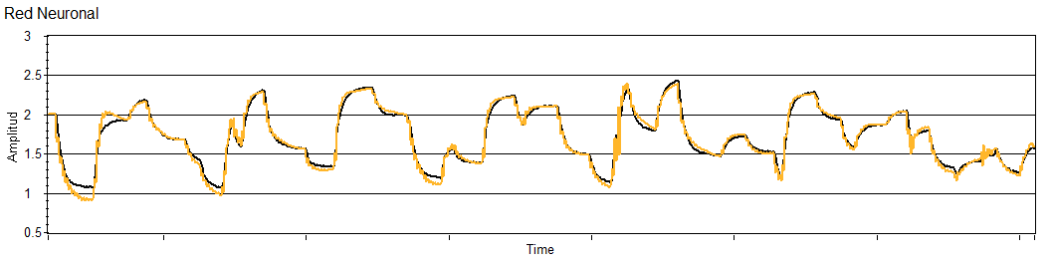

Figura 46. Respuesta de la red: MSE 0.028, Estructura 10 Tanh-10Tanh-1Lineal.

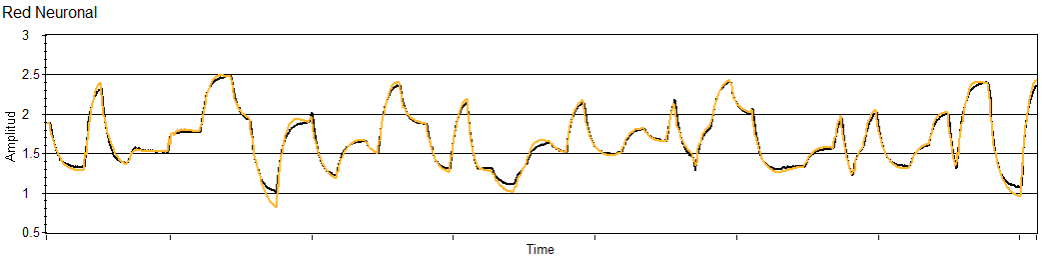

Figura 57. Respuesta de la red: MSE 0.0131, Estructura 4 Lineal-100 Tanh-1 Lineal.

DISCUSIÓN

De los resultados se descubrió en primera instancia que, si los pesos sinápticos *w* se inicializan con cantidades mayores a 1, éstos hacen que las neuronas con función de activación *Tangente Hiperbólica* se saturen, es decir, que no operen en la zona lineal de la función, y así provoquen que la red neuronal no converja en la solución esperada, como se muestra en la Figura 14.

Se puede distinguir que una cantidad pequeña de neuronas es capaz de aprender el comportamiento del sistema en estudio (Figura 15), sin embargo, se observó que una cantidad mayor de neuronas ayuda a una mejor aproximación del comportamiento real del sistema.

A su vez, se observó que añadir una o dos capas ocultas a la estructura de la red también disminuyó el error entre las señales pero provocó un mayor tiempo en la convergencia de la solución, y si no se seleccionaba una tasa de aprendizaje η menor a la de los entrenamientos anteriores de una capa oculta (0.001), la red diverge de la solución.

Se propuso una arquitectura con funciones de activación lineal para la capa de entrada, que supone un acondicionamiento de las señales de entrada para que las siguientes capas con función de activación tangente hiperbólica no se saturen. De esta red, se obtuvo un error MSE de 0.0131 y de la figura 17 se observa que la red efectivamente sigue a la señal de salida del sistema.

CONCLUSIONES

Es posible aprovechar las características de LabVIEW para implementar, eficientemente en un VI, el algoritmo de retropropagación para el entrenamiento de redes neuronales artificiales.

Los resultados de cada red entrenada varían dependiendo de parámetros como los son la estructura de la red, la inicialización de pesos, la tasa de aprendizaje η, el momento, y el número de iteraciones del entrenamiento. Por lo que es necesaria una supervisión del entrenamiento y de la red neuronal para el algoritmo implementado.

El algoritmo de retropropagación usa el gradiente descendiente para encontrar el mínimo del error con respecto a los pesos de la red. Sin embargo, el algoritmo puede "estancarse" en lo que se denominan mínimos locales, donde el gradiente es igual a cero, pero el error aun es grande. Esto es un problema usual con el algoritmo de retropropagación, por lo que para futuros diseños se planea la opción de entrenar con un algoritmo diferente como el de Levenberg-Marquard.

Además, en sistemas lentos, con tiempo entre iteración e iteración bajo y con una tasa de aprendizaje alta, las redes tienden a sobre-entrenarse, y perder generalización, es decir, las redes "perciben" una entrada constante que se ve reflejado en la salida (también constante).

Si busca evitar la divergencia en el entrenamiento de la red con una tasa de aprendizaje pequeño, se tendrá que requerir una gran cantidad de iteraciones para que la red tenga

un error mínimo deseado. Por lo que se propone implementar una heurística para cambiar la tasa de aprendizaje con respecto al cambio en el error promedio.

REFERENCIAS BIBLIOGRAFICAS

[1] Anil K. Jain, Artificial neural networks, Michigan state university, 1996.

[2] Heaton, Jeff. Introduction to Neural Networks for C#, Ed. Words Ru, 2008.

[3] Martínez, Juan José. Automatización inteligente en procesos industriales utilizando redes Neuronales, Instituto Tecnológico de Celaya, 2007.

[4] Sánchez Camperos, Edgar N. Redes neuronales, conceptos fundamentales y aplicaciones a control automático, pp. 165-175, Ed. Prentice-Hall, 2006.

[5] Manurung, Auralius. *Implementation of Artificial Neural Network on LABVIEW 8.5*, Gyeongsang National University, 2009.