Entrenamiento de Redes Neuronales MLP Basado en el Algoritmo ARγ

Fausto Miguel Castro Caicedo
Ingeniero Electrónico, estudiante Ciclo I del Doctorado
en Ciencias de la Electrónica
Universidad del Cauca
faustocastro@unicauca.edu.co

Pablo Emilio Jojoa Gómez
Dr. En Ingeniería Eléctrica
Docente Universidad del Cauca
pjojoa@unicauca.edu.co

Resumen: Este documento presenta un nuevo algoritmo para el entrenamiento de redes neuronales MLP basado en las propiedades de otro algoritmo conocido como algoritmo ARγ, el cual se ha desarrollado en el contexto del filtrado adaptativo. El algoritmo obtenido se valida mediante una aplicación de reconocimiento de patrones, demostrando buen desempeño en cuanto a rapidez, y generalización.

Palabras clave: Redes Neuronales, MLP, Algoritmo ARγ, Gradiente Local de Error.

1. Introducción

Las redes neuronales artificiales son modelos de procesamiento (inspirados en los sistemas nerviosos biológicos) con gran impacto en las aplicaciones modernas. El número de aplicaciones para el cual está indicado el uso de las redes neuronales así como la complejidad de dichas aplicaciones, se vienen incrementado constantemente desde hace varios años. Este desarrollo permanente se sostiene gracias a los esfuerzos investigativos que la comunidad internacional realiza para mejorar las prestaciones de los algoritmos de entrenamiento utilizados [1]. Así, esta investigación es una propuesta que busca aportar en el campo de la computación neuronal y representa un salto desde la teoría del filtrado adaptativo lineal hacia las redes neuronales, que permite hacer uso de las buenas prestaciones del algoritmo ARγ [7], [8] (mostradas en filtrado adaptativo) en el entrenamiento de redes neuronales MLP.

El Objetivo entonces, es diseñar un algoritmo para el entrenamiento de redes neuronales MLP basado en las propiedades del algoritmo ARγ. Para esto se utiliza un funcional conocido como gradiente local de error, el cual se minimiza haciendo uso de los principios que rigen la actualización de parámetros en el algoritmo ARγ. Este documento se organiza en cinco secciones: La sección 2 introduce los conceptos teóricos fundamentales y establece la nomenclatura y simbología a utilizar; la sección 3 presenta el concepto de gradiente local de error haciendo uso de las expresiones propias del algoritmo ARγ; la sección 4 describe la implementación propuesta; la sección 5 presenta un caso de aplicación relacionado con reconocimiento de patrones, con el cual se valida experimentalmente el algoritmo propuesto; la sección 6 presenta los resultados del estudio experimental y Finalmente, en la sección 7 se enuncian las conclusiones.

2. Conceptos Teóricos

2.1 Neurona artificial

La unidad funcional de un sistema neuronal artificial es la neurona artificial, la cual al agruparse con otras, forma capas y a su vez varias capas constituyen una red neuronal o ANS (*Artificial Neural Networks*). Las neuronas artificiales son procesadores elementales que pueden tener varias entradas pero únicamente una salida. El funcionamiento general de una neurona i estándar obedece a las siguientes reglas [2]:

> Cada conexión de entrada trae asociado un parámetro denominado peso sináptico ($w_{i,1}, w_{i,2}, w_{i,2}, \ldots w_{i,R}$,) que se

multiplica respectivamente por las señales de entrada ($x_1, x_2, x_3, \ldots x_R$,), los productos obtenidos se suman para calcular el potencial post sináptico a_i. Ver figura 1.

> Una neurona *i* obtiene su salida y_i aplicando una función de activación o transferencia *f*(.) al potencial post sináptico. Las principales funciones de activación son mostradas en la figura 1.

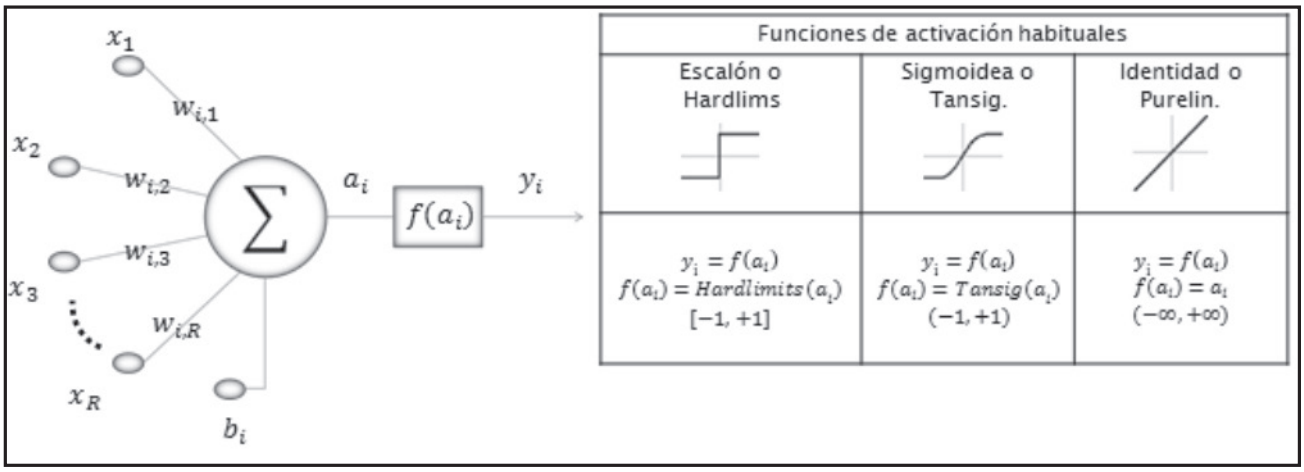


Figura 1. Modelo de neurona estándar con las funciones de activación más habituales.

Con frecuencia se añade al conjunto de pesos de la neurona un parámetro adicional b_i denominado umbral o bias, que puede considerarse como un peso sináptico adicional, al cual se le asocia permanentemente una entrada unitaria de valor negativo. La función de este parámetro es sencillamente brindarle a la neurona un grado de libertad adicional.

2.2. Redes neuronales MLP

Un Perceptrón multicapa o MLP (*Multi-Layer Perceptrón)* es un modelo de ANS no lineal compuesto de varias capas de neuronas entre la entrada y la salida, dichas capas se unen sin ningún tipo de realimentación, de tal manera que las entradas se unen con la primera capa y las salidas de ésta con la siguiente capa y así sucesivamente hasta la última capa. Esta arquitectura puede usarse para resolver problemas de clasificación que involucren patrones de entrada linealmente no separables y también como generador universal de funciones [3].

La figura 2 muestra la arquitectura abreviada de un MLP con una capa oculta. Por lo general, las neuronas utilizan funciones de activación sigmoideas para las capas ocultas y funciones de activación lineales para la capa de salida, esta es la arquitectura más común del MLP. Sin embargo, existen numerosas variantes (por lo general en problemas de clasificación) que utilizan funciones de activación sigmoideas en la capa de salida.

Si se denominan x_i a las entradas de la red, y_j a las salidas de la capa oculta y z_k a las salidas de la capa final o de salida, y por otro lado, $w_{i,j}$ a los pesos de la capa oculta y b_j a sus biases, $w'_{k,j}$ a los pesos de la capa de salida y b'_k a sus biases, entonces la operación de un MLP con una capa oculta se expresa matemáticamente como [4]:

$$z_k = f_2\left(\sum_{j=1}^{U_1} w'_{k,j} f_1\left(\sum_{i=1}^{R} w_{i,j} x_i - b_j\right) - b'_j\right), \qquad (1)$$

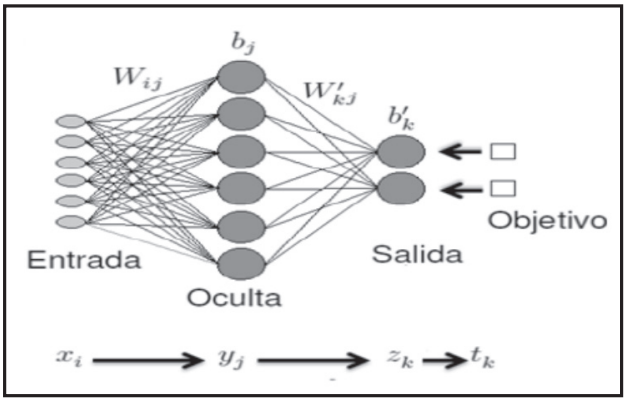


Figura 2. Arquitectura de un MLP

donde f_1 (.) y f_2 (.) son respectivamente las funciones de activación para las neuronas de la capa oculta y la capa de salida, *R* Representa el número de entradas y U_1 el número de neuronas en la capa oculta.

Una red neuronal MLP puede operar en dos modos, *modo de entrenamiento y modo de recuerdo*. El primero se

realiza haciendo uso de un algoritmo de entrenamiento que modifica los pesos sinápticos de la red con el propósito de que aprenda a realizar una tarea. En el segundo modo, una vez la red cuenta con una estructura de pesos fija, se aplican entradas jamás utilizadas en el entrenamiento y se espera que la red esté preparada para brindar respuestas adecuadas.

2.3 Algoritmo ARγ

Es un algoritmo utilizado en filtrado adaptativo que ajusta la segunda derivada de parámetros. Sus tres parámetros de ajuste, denominados α, γ y m_1, permiten lograr una buena velocidad de convergencia y paralelamente una considerable reducción del error de medida final. Las ecuaciones que describen el algoritmo ARγ son:

$$e[n] = \boldsymbol{x}^T[n]\boldsymbol{w}[n-1] - d[n], \quad (2)$$

$$g[n] = \frac{e[n]+\gamma \boldsymbol{x}^T[n]\boldsymbol{w}[n-1]}{1+\alpha\gamma m_1 \boldsymbol{x}^T[n]\boldsymbol{x}[n]}, \quad (3)$$

$$\boldsymbol{q}[n] = \frac{\gamma}{\alpha+\gamma}\left[\boldsymbol{q}[n-1] - \alpha m_1 g[n]\boldsymbol{x}[n]\right], \quad (4)$$

$$\boldsymbol{w}[n] = w[n-1] + \alpha \boldsymbol{q}[n], \quad (5)$$

donde $\boldsymbol{x}[n]$es el vector de la señal de entrada, $d[n]$ un escalar correspondiente a la señal deseada, $e[n]$ un escalar correspondiente a la señal de error, $g[n]$ un escalar auxiliar en el instante n, $\boldsymbol{q}[n]$ un vector auxiliar del filtro adaptativo, $\boldsymbol{w}[n]$ es el vector de coeficientes del filtro adaptativo y las constantes α, γ y m_1 son parámetros de ajuste fijo.

El algoritmo converge para $\gamma > 0$, $\alpha > 0$ y $m_1 > 0$ y se determinó que éste presenta un mínimo error de desajuste cuando $\alpha\gamma$ $m_1 \approx 2$ [7], [8].

3. Gradientes Locales de Error

Como resultado de aplicar el método de descenso por el gradiente a las redes MLP se obtiene los gradientes locales de error ($\delta'_k[n]$ y $\delta_j[n]$) los cuales se definen matemáticamente como el gradiente de la energía de error total $E[n]$ de la red en función del potencial post sináptico ($a'_k[n]$ o $a_j[n]$) de una neurona en particular [6].

Así entonces, teniendo en cuenta la red MLP de la figura 2; considerando $b'_k = w'_{k,0}$, $b_j = w_{j,0}$ y en consistencia con la dinámica interna del algoritmo ARγ para un instante n, se tiene a partir de la ecuación (1) que

$$z_k[n] = f_2(a'_k), \quad (6)$$

$$a'_k[n] = \sum_{j=0}^{U_1} y_j[n] w'_{k,j}[n-1], \quad (7)$$

$$y_j[n] = f_1(a_j[n]), \quad (8)$$

$$a_j[n] = \sum_{i=0}^{R} x_i[n] w_{j,i}[n-1], \quad (9)$$

siendo $a'_k[n]$ y $a_j[n]$ los potenciales post sinápticos para neuronas en la capa de salida y oculta respectivamente.

Y definiendo la energía total del error $E[n]$ para la red como.

$$E[n] = \frac{1}{2}\sum_{k=1}^{U_2} e_k^2[n], \quad (10)$$

$$e_k[n] = z_k[n] - t_k[n], \quad (11)$$

donde $e_k[n]$ y $t_k[n]$ son el error y la señal deseada para una neurona k de la capa de salida respectivamente y U_2 el número de neuronas en la capa de salida. Pueden ocurrir dos casos de gradiente local de error:

> Caso 1: Neurona en capa oculta.

$$\delta_j[n] = \frac{\partial E[n]}{\partial a_j} = \frac{\partial f_1(a_j[n])}{\partial a_j[n]} \sum_k \delta'_k[n] w'_{k,j}[n-1]. \quad (12)$$

> Caso 2: Neurona en capa de salida.

$$\delta'_k[n] = \frac{\partial E[n]}{\partial a'_k} = e_k[n] \frac{\partial f_2(a'_k[n])}{\partial a'_k[n]}. \quad (13)$$

4. Implementación Propuesta para el Entrenamiento de la Red MLP.

El procedimiento seguido para realizar la implementación se resume a continuación.

1) Iniciar una configuración aleatoria de pesos sinápticos (si se parte de pesos y biases nulos el aprendizaje no progresa).

2) Para cada patrón de aprendizaje:

2.2. Calcular $\delta'_k[n]$ y $\delta_j[n]$ para las correspondientes neuronas de la red.

2.3. Calcular el incremento parcial de los pesos con.

$$q'_{k,j}[n] = \frac{\gamma}{\alpha+\gamma}\left[q'_{k,j}[n-1] - \alpha m_1 g'[n] y_j[n]\right], \quad (14)$$

$$g'[n] = \frac{\delta'_k[n] + \gamma \sum y_j[n] w'_{k,j}[n-1]}{1+\alpha\gamma m_1 \sum y_j^2[n]}, \quad (15)$$

$$q_{j,i}[n] = \frac{\gamma}{\alpha+\gamma}\left[q_{j,i}[n-1] - \alpha m_1 g[n] x_i[n]\right], \quad (16)$$

$$g[n] = \frac{\delta_j[n] + \gamma \sum x_i[n] w_{j,i}[n-1]}{1+\alpha\gamma m_1 \sum x_j^2[n]}, \quad (17)$$

donde $g[n]$ y $g'[n]$ son escalares auxiliares en el instante n; $q'_{k,j}$ [n] y $q_{j,i}$ [n] incrementos parciales de pesos y biases; γ, α y m_1 son parámetros de ajuste fijo.

3) Calcular los incrementos totales ($Q'_{k,j}[n]$ y $Q_{j,i}[n]$) como la suma de los incrementos parciales $q'_{k,j}[n]$ y $q'_{j,i}[n]$ para todos los patrones de entrenamiento.

4) Actualizar los pesos sinápticos con.

$$w'_{k,j}[n] = w'_{k,j}[n-1] + \alpha Q'_{k,j}[n] \quad (18)$$

$$w_{j,i}[n] = w_{j,i}[n-1] + \alpha Q_{j,i}[n] \quad (19)$$

5) Regresar al paso 2 si el resultado no es satisfactorio de acuerdo a algún criterio (Ejemplo error, número de épocas).

El algoritmo converge para valores positivos de γ, α y m_1 (γ>0, α>0 y m_1>0) y puede extenderse a redes de múltiples capas ocultas siguiendo la misma heurística de gradientes locales de error para esas capas. El procedimiento se implementó en la herramienta computacional Matlab® y se validó mediante un problema de clasificación de patrones. El planteamiento y diseño del experimento se presentan a continuación.

5. Planteamiento y Diseño del Experimento

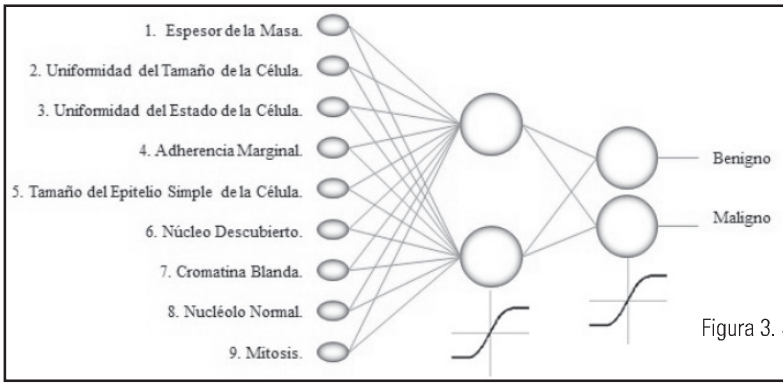
La finalidad es entrenar una red neuronal tipo MLP para que determine la condición benigna o maligna de un tumor mamario basándose en nueve características de muestras de biopsia. Como no existe un criterio general para determinar cuántas neuronas son suficientes en la capa oculta (este es un problema específico que no está teóricamente esclarecido [9]), se hicieron pruebas con diferentes variantes del algoritmo BP (*Back-Propagation*) [4], disponibles en Matlab®, y se determinó que con dos neuronas en la capa oculta se obtienen buenos resultados. La figura 3 muestra las salidas y arquitectura de red utilizada en el experimento. Las funciones de activación fueron sigmoideas para ambas capas.

Los datos para el entrenamiento de la red fueron obtenidos de la base de datos de Matlab® (cáncer dataset) y son muestras de biopsias tomadas a diferentes pacientes. Se cuenta con 699 muestras (patrones de entrenamiento) [5].

6. Resultados

El entrenamiento se realizó para diferentes valores de los parámetros γ, α y m_1 dejando dos de ellos constantes y variando el sobrante. En todos los casos los pesos sinápticos y biases iniciales fueron los mismos. Las gráficas de entrenamiento resultantes se muestran en la figura 4.

El conjunto de aprendizaje se dividió en tres subconjuntos de 499, 100 y 100 patrones, los cuales permiten respectivamente; entrenar la red, determinar la generalización y realizar el test de entrenamiento final. Los resultados obtenidos se muestran en la figura 5 para dos configuraciones de parámetros γ, α y m_1.



Salidas		Tipo de tumor
Neurona 1	Neurona 2	
1	0	Benigno
0	1	Maligno
0	0	Error de clasificación
1	1	Error de clasificación

Figura 3. Salidas y arquitectura de red utilizada en el experimento.

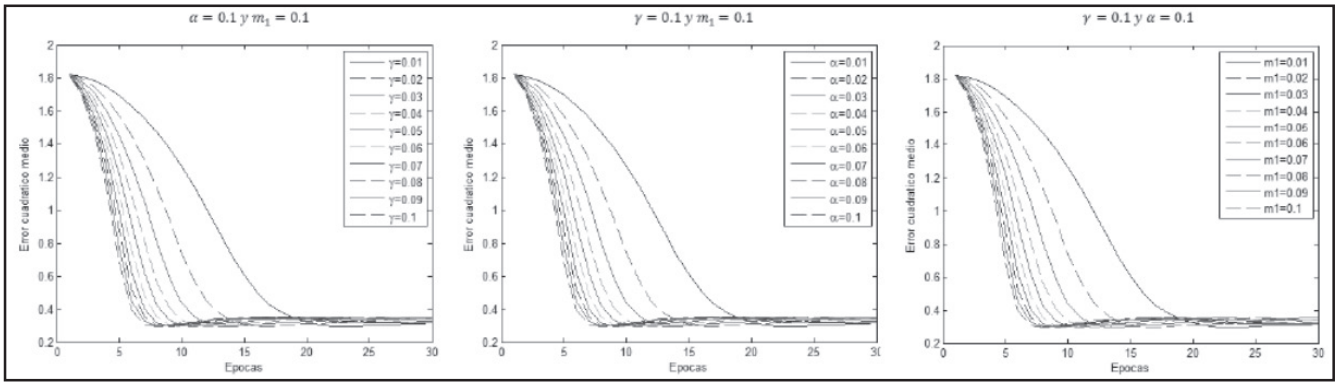


Figura 4. Entrenamiento para diferentes valores de γ, α y m_1.

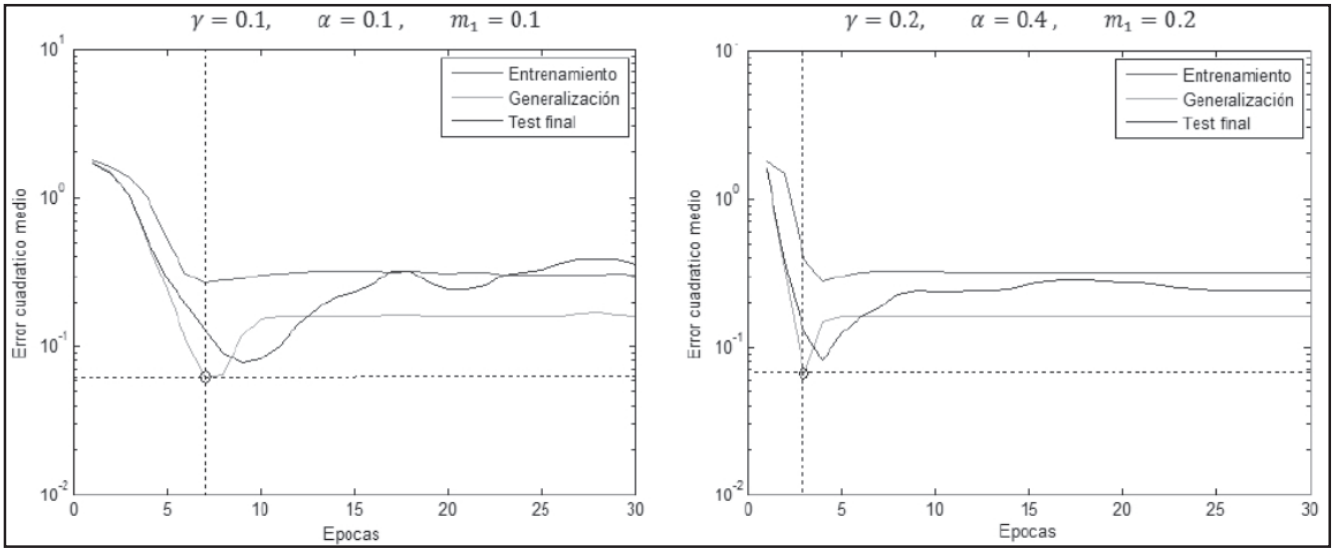


Figura 5. Curva de Entrenamiento, generalización y test final (α=0.1, γ=0.1 y m_1 =0.1).

Los resultados permiten afirmar que la implementación realizada es consistente experimentalmente, alcanza errores mínimos de generalización y muy buenos tiempos de convergencia. En esencia la red aprende rápido (en este caso a realizar una tarea de clasificación). Lo cual es bastante destacable en relación con otros experimentos realizados con variantes del algoritmo BP en Matlab®.

7. Conclusiones

Un algoritmo para el entrenamiento de redes neuronales MLP basado en las propiedades del algoritmo ARγ debe implementar una heurística de propagación de errores para el entrenamiento de las capas ocultas de la red. Esto se logra incorporando un funcional de error conocido como gradiente local de error.

El gradiente local de error es un concepto que se deriva de aplicar el método de descenso por el gradiente a las redes multicapa y representa un sistema eficaz para la administración de errores en una red neuronal MLP.

Los resultados que genera la implementación propuesta son consistentes desde el punto de vista experimental, pues demuestran buen desempeño en cuanto a rapidez, y generalización. Sin embargo es necesario esclarecer el efecto que tienen los parámetros α, γ y m_1 en lo relacionado al entrenamiento y generalización de la red.

Bibliografía

[1] W. WU, J. WANG, M. CHENG, Z. LI, Convergence analysis of online gradient method for BP neural networks, Contenido disponible en SciVerse Science Direct, Neural Networks 24, 2011.

[2] W. G. LÓPEZ, Pronóstico de una serie temporal usando redes neuronales, Universidad Tecnológica de la Mixteca, Oaxaca, 2010.

[3] V. H. DE ALBUQUERQUE, A. R. ALEXANDRIA, P. C. CORTEZ, J. M.TAVARES, Evaluation of multilayer perceptron and

self-organizing map neural network topologies applied on microstructure segmentation from metallographic images, Contenido disponible en Science Direct, NDT&E International 42. 2009.

[4] B. MARTIN DEL BRIO, A. SANZ MOLINA. Redes neuronales artificiales y sistemas borrosos, 3° Edición, Alfaomega Ra-Ma. 2007.

[5] M. H. BEALE, M. T. HAGAN, H. DEMUTH, Neural network toolbox 7 user's guide, Natick, MA: The MathWorks; 2010.

[6] S. HAYKIN, Neural networks: a comprehensive foundation, 2° Edición, McMaster University, Hamilton, Ontario, Canada. 1998.

[7] P.E. JOJOA. Um algoritmo acelerador de parámetros, Tesis de Doctorado. Escola Polit'ecnica da Universidad de S'ao Paulo, Brasil, 2003.

[8] J. C. REALPE, Estudio del efecto de la variación temporal de parámetros del algoritmo acelerador regresivo versión "gamma", Universidad del Cauca, Facultad de Ingeniería Electrónica y Telecomunicaciones. Popayan 2009.

[9] Q. DAI, N. LIU, Alleviating the problem of local minima in backpropagation through competitive learning, Contenido disponible en SciVerse Science Direct, Neurocomputing 94, 2012.