NOTACIÓN MATRICIAL EN EL ENTRENAMIENTO DE REDES NEURONALES

RESUMEN
Este documento contiene la formulación para efectuar el entrenamiento de una red neuronal con el algoritmo *Backpropagation*, en el cual se ha usado un método poco usual, el cual es la formulación del mismo utilizando matrices. Esto se realiza aprovechando el hecho de que algunas operaciones permiten ser realizadas como las que usualmente se utilizan con matrices. Además las operaciones matriciales proporcionan una notación y una programación mucho más resumida, lo cual se ve reflejado en el tiempo computacional.

PALABRAS CLAVES: Red neuronal, entrenamiento, *Backpropagation*, notación matricial.

JASON EDWIN MOLINA V.
Ingeniero Electricista,
Profesor
Universidad Tecnológica de Pereira
jason@ohm.edu.edu.co

CARLOS ALBERTO RESTREPO
Ingeniero Electricista,
Universidad Tecnológica de Pereira
cr@utp.edu.co

ABSTRACT
This document contains the formulation to carry out the training of a neural network using the Backpropagation algorithm, in which a little usual method has been used, which is its formulation using matrices. This is made taking advantage of the fact that some operations allow to be done like which usually they are used with matrices, and in addition the matrix operations provide a notation and a programming much more summarized, which is reflected in the computational time.

KEYWORDS: *Neural network, training, Backpropagation, Matricial notation.*

1. INTRODUCCIÓN

Las redes neuronales en el desarrollo científico moderno se han consolidado como una de las herramientas más útiles, no obstante, esta herramienta depende estrictamente de un adecuado entrenamiento. El entrenamiento de una red neuronal depende de muchos factores tales como la forma de los datos, y los parámetros de la arquitectura de la red neuronal.

Muchos documentos actuales presentan algunas metodologías para el entrenamiento de las redes neuronales [1,2], pero no siempre éstos especifican la forma que debe tener el algoritmo de entrenamiento para que converja de forma más rápida. En este artículo se desarrolla una forma matricial para entrenar una red neuronal, tomando como base la deducción del entrenamiento *Backpropagation*, y se muestra que el algoritmo desarrollado es muy general permitiendo variaciones en los parámetros de la arquitectura de la red.

Se dan inicialmente los conceptos que definen la arquitectura grafica de una red neuronal, seguido de la deducción de las operaciones matemáticas necesarias y luego la deducción del algoritmo de optimización que logra que la red neuronal alcance el objetivo, desde una inicialización aleatoria.

2. ARQUITECTURA DE UNA RED NEURONAL

Se define la red neuronal como el paradigma:

$$\left\langle \mathbf{W}^L, \mathbf{A}^L; 1 \le L \le M \right\rangle$$

en donde $\mathbf{W}^L$ es una colección de matrices, llamadas matrices de pesos y $\mathbf{A}^L$ es una colección de funciones escalares de variable escalar no lineales, y M es el tamaño de la colección [1]. Para ilustrar el comportamiento del flujo de datos por el paradigma, se realiza un gráfico que ilustra la estructura de la red neuronal, como se ve en la figura 1.

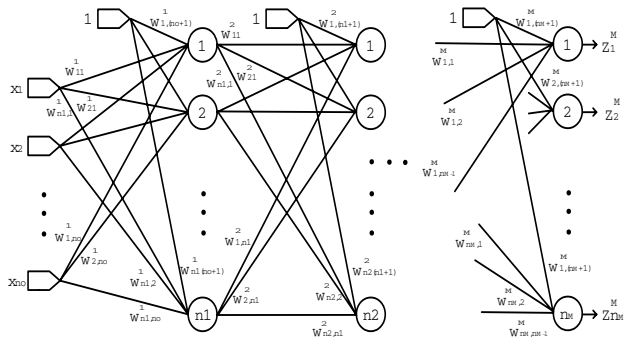

Figura 1. Arquitectura gráfica de una red neuronal.

En el gráfico se distingue una capa de entrada, *M-1* capas ocultas y una capa de salida. Las capas que no son de entrada ni de salida se denominan ocultas. La capa de entrada no tiene neuronas sino que sólo distribuye los datos del vector de entrada a la primera capa oculta. Cada neurona recibe cierto número de entradas y las procesa para dar paso a una salida según se indica en la figura 2.

Fecha de Recepción: 09 Febrero de 2007
Fecha de Aceptación: 03 Mayo de 2007

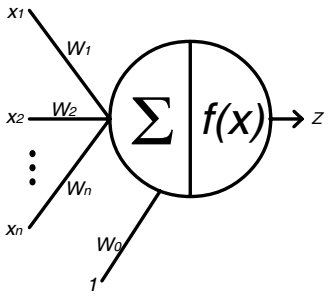

Figura 2. Neurona genérica de una red neuronal.

Si ésta es la neurona *i* correspondiente a la capa *L*, la salida z_i^L está dada por [2]:

$$z_i^L = f\left(w_{i,n_{L-1}}^L + \sum_{j=1}^{n_{L-1}} w_{ij}^L x_j \right) = f\left(\underline{\mathbf{w}}_i^L \begin{bmatrix} \underline{z}^{L-1} \\ 1 \end{bmatrix} \right)$$

3. ENTRENAMIENTO DE LA RED NEURONAL

Partiendo de un conjunto de datos experimentales de un modelo desconocido de cualquier índole (por ejemplo un sistema físico), se pueden obtener dos subconjuntos de datos, uno para entrenar y otro para validar. El objetivo de este procedimiento es que pueda obtenerse un criterio de generalización para el modelo, en el caso en que la entrada no coincida con las entradas del conjunto de entrenamiento, lo cual es muy probable. Se dan entonces, en este orden, los conjuntos:

$$\left\{ \underline{\mathbf{x}}(l), \underline{\mathbf{d}}(l); l \in \Omega_1 \right\} \text{ y } \left\{ \underline{\mathbf{x}}(l), \underline{\mathbf{d}}(l); l \in \Omega_2 \right\}$$

En los que Ω_1 y Ω_2 son disjuntos y definen los conjuntos de entrenamiento y de validación.

Para realizar el entrenamiento de la red, se ajusta la colección de matrices $\mathbf{W}^L$ para producir el resultado deseado [3]. Esto se realiza minimizando la función definida como el promedio de los errores medios cuadráticos de las salidas de la capa *M* entre todo el conjunto de entrenamiento $\left\{ \underline{\mathbf{x}}(k), \underline{\mathbf{d}}(k); 1 \le k \le K \right\}$ [4]:

$$E = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{2} \sum_{q=1}^{n_M} \left[d_q(k) - z_q^M(k) \right]^2$$

Aquí, *k* se refiere a la muestra de entrenamiento evaluada del conjunto de *K* muestras y *q* es una neurona genérica en la capa de salida. Cada superíndice *L* hace referencia a la capa *L* en un total de *M* capas, a su vez cada capa tiene n_M neuronas. Se usará el subíndice *i* para denotar neuronas de la capa actual y el subíndice *j* para denotar neuronas de la capa anterior, de modo que cada matriz de pesos $\mathbf{W}^L$ se asigna a la capa *L* y tiene dimensiones n_L x (n_{L-1} + 1), y *L* = 1 corresponde a la primera capa oculta.

El error *E* es función de las matrices $\mathbf{W}^L$ y el problema de entrenamiento consiste en hallar dichas matrices de tal manera que *E* sea mínimo; en cuyo caso, las matrices darán una aproximación considerable a las entradas del conjunto de entrenamiento [2]. El mínimo valor que toma *E* es cero, cuando las salidas del conjunto de entrenamiento son iguales a las salidas de la red para todo $1 \le k \le K$.

El paradigma usado para ajustar las matrices de pesos es:

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \Delta\mathbf{W}(t)$$

Donde $\Delta\mathbf{W}(t)$ tiene varios métodos de cálculo; en este artículo se va a usar el llamado *Backpropagation*, pues contiene cálculos relativamente simples. Este método consiste en hacer que el paso $\Delta\mathbf{W}(t)$ sea un pequeño incremento de las matrices de pesos en la dirección del gradiente negativo de *E* respecto a $\mathbf{W}$, que es la dirección del máximo descenso de *E* en el punto $\mathbf{W}(t)$ [5], esto es:

$$\Delta\mathbf{W}(t) = -\eta \frac{\partial E}{\partial \mathbf{W}}$$

El proceso se realiza iterativamente hasta obtener una tolerancia de error medio cuadrático, o un número de iteraciones dado, para un η dado. La dificultad de este proceso está en que todas las matrices de pesos están acopladas mediante operaciones matemáticas no lineales, y el cálculo del gradiente puede ser costoso. A continuación se presenta la deducción de una metodología de cálculo para este gradiente.

Para una capa dada, se tiene:

$$\underline{\mathbf{u}}^L = \begin{pmatrix} u_1^L \\ u_2^L \\ \vdots \\ u_{n_L}^L \end{pmatrix}, \quad u_i^L = w_{i(n_{L-1}+1)}^L + \sum_{j=1}^{n_{L-1}} w_{ij}^L z_j^{L-1}, \quad 1 \le i \le n_L \quad (1)$$

Por lo cual:

$$\underline{\mathbf{u}}^L = \mathbf{W}^L \begin{bmatrix} \underline{\mathbf{z}}^{L-1} \\ 1 \end{bmatrix} = \mathbf{W}^L \left(\underline{\mathbf{z}}^{L-1^T}, 1 \right)^T, \text{ donde}$$

$$\underline{\mathbf{z}}^L = f\left(\underline{\mathbf{u}}^L \right), \quad 1 \le L \le M$$

Se entiende que en la capa de entrada $\underline{\mathbf{z}}^0$ se toma como el vector de entrada.

$$\frac{\partial E}{\partial \mathbf{W}^L} = \frac{1}{2K} \sum_{k=1}^{K} \sum_{q=1}^{n_M} \frac{\partial}{\partial \mathbf{W}^L} \left\{ \left[d_q(k) - z_q^M(k) \right]^2 \right\} =$$

$$= \frac{1}{2K} \sum_{k=1}^{K} \sum_{q=1}^{n_M} -2 \left[d_q(k) - z_q^M(k) \right] \frac{\partial z_q^M(k)}{\partial \mathbf{W}^L}$$

Para determinar $\frac{\partial z_q^M(k)}{\partial \mathbf{W}^L}$ se tiene:

$$\frac{\partial z_q^M(k)}{\partial w_{ij}^L} = \frac{\partial z_q^M(k)}{\partial u_i^L(k)} \frac{\partial u_i^L(k)}{\partial w_{ij}^L}$$

entonces:

$$\frac{\partial E}{\partial w_{ij}^L} = -\frac{1}{K}\sum_{k=1}^{K}\sum_{q=1}^{n_M}\left[d_q(k) - z_q^M(k)\right]\frac{\partial z_q^M(k)}{\partial u_i^L(k)}\frac{\partial u_i^L(k)}{\partial w_{ij}^L}$$

$$\frac{\partial z_q^M(k)}{\partial u_i^L(k)} = \frac{\partial z_q^M(k)}{\partial z_i^L(k)}\frac{\partial z_i^L(k)}{\partial u_i^L(k)} = f'\left(u_i^L(k)\right)\frac{\partial z_q^M(k)}{\partial z_i^L(k)}$$

$z_q^M(k)$ es función de $u_r^{L+1}(k)$; con $1 \le r \le n_{L+1}$, y $u_r^{L+1}(k)$ es función de $z_i^L(k)$; con $1 \le i \le n_L$, por lo cual, aplicando la regla de la cadena [5] se tiene:

$$\frac{\partial z_q^M(k)}{\partial z_i^L(k)} = \nabla z_q^M(k)\Big|_{\underline{\mathbf{u}}^{L+1}(k)} \bullet \frac{\partial \underline{\mathbf{u}}^{L+1}(k)}{\partial z_i^L(k)}$$

Por lo tanto:

$$\frac{\partial z_q^M(k)}{\partial z_i^L(k)} = \sum_{r=1}^{n_{L+1}} \frac{\partial z_q^M(k)}{\partial u_r^{L+1}(k)}\frac{\partial u_r^{L+1}(k)}{\partial z_i^L(k)}$$

De la ecuación (1):

$$\frac{\partial u_r^{L+1}(k)}{\partial z_i^L(k)} = w_{ri}^{L+1}$$

con lo cual:

$$\frac{\partial z_q^M(k)}{\partial u_i^L(k)} = f'\left(u_i^L(k)\right)\sum_{r=1}^{n_{L+1}}\frac{\partial z_q^M(k)}{\partial u_r^{L+1}(k)} w_{ri}^{L+1}.$$

Se define:

$$\delta_i^L(k,q) \triangleq \frac{\partial z_q^M(k)}{\partial u_i^L(k)}\ , \text{ de manera que,}$$

$$\delta_i^L(k,q) \triangleq f'\left(u_i^L(k)\right)\sum_{r=1}^{n_{L+1}}\delta_r^{L+1}(k,q) w_{ri}^{L+1}$$

Este último hecho, hace que el cálculo de $\delta_i^L(k,q)$ se haga recursivo, calculando sus valores a partir de los mismos en capas posteriores, haciendo un flujo de datos hacia atrás respecto a las capas; de allí que reciba el nombre de error *Backpropagation* [4]. Dado que el cálculo es recursivo, requiere una inicialización en la última capa, que se obtiene derivando la salida en la última capa:

$$\delta_i^M(k,q) = \frac{\partial z_q^M(k)}{\partial u_i^M(k)} = f'\left(u_i^M(k)\right)\delta(q-i)$$

Donde δ(*r*) es la función delta de Kronecker [6]:

$$\delta(r) = \begin{cases} 1, & r = 0 \\ 0, & r \neq 0 \end{cases}$$

Con lo anterior se obtiene que:

$$\frac{\partial E}{\partial w_{ij}^L} = \begin{cases} -\frac{1}{k}\sum_{k=1}^{K}\sum_{q=1}^{n_M}\left[d_q(k) - z_q^M(k)\right]\delta_i^L(k,q)\, z_j^{L-1}(k) & ,1 \le j \le n_{L-1} \\ -\frac{1}{k}\sum_{k=1}^{K}\sum_{q=1}^{n_M}\left[d_q(k) - z_q^M(k)\right]\delta_i^L(k,q) & ,j = n_{L-1}+1 \end{cases}$$

Para recuperar la notación matricial mencionada al principio, se presenta a $\mathbf{W}^L$ como un arreglo por filas:

$$\mathbf{W}^L = \left[\mathbf{w}_i^L\right]_{i=1:n_L} = \begin{bmatrix} \underline{\mathbf{w}}_1^L \\ \underline{\mathbf{w}}_2^L \\ \vdots \\ \underline{\mathbf{w}}_{n_L}^L \end{bmatrix} \text{ donde,}$$

$\underline{\mathbf{w}}_i^L = \left[w_{i1}^L,\ \ldots,\ w_{i(n_{L-1})}^L,\ \ w_{i(n_{L-1}+1)}^L\right]$, es una fila de $\mathbf{W}^L$.

Entonces:

$$\frac{\partial u_i^L(k)}{\partial \underline{\mathbf{w}}_i^L} = \begin{bmatrix} \underline{\mathbf{z}}^{L-1}(k) \\ 1 \end{bmatrix} \qquad (2)$$

independiente del valor de i: $1 \le i \le n_L$.

Se puede presentar también $\underline{\mathbf{W}}^{L+1}$ como un arreglo por columnas:

$$\underline{\mathbf{W}}^{L+1} = \left[\underline{\mathbf{w}}_1^{L+1},\ \ldots,\ \underline{\mathbf{w}}_{n_L}^{L+1},\ \underline{\mathbf{w}}_{n_L+1}^{L+1}\right], \text{ donde}$$

$$\underline{\mathbf{w}}_i^{L+1} = \left[w_{1i}^{L+1}\ \ ,\ldots,\ \ w_{(n_{L-1})i}^{L+1}\right] \text{ es la columna } i.$$

Así:

$$\sum_{r=1}^{n_{L+1}}\delta_r^{L+1}(k,q) w_{ri}^{L+1} = \left[\underline{\mathbf{w}}_i^{L+1}\right]^T \underline{\delta}^{L+1}(k,q) \text{ donde}$$

$$\underline{\delta}^{L+1}(k,q) = \left[\delta_1^{L+1}(k,q),\ldots,\delta_{n_{L+1}}^{L+1}(k,q)\right]$$

Con lo cual:

$$\delta_i^L(k,q) = f'\left(u_i^L(k)\right)\left[\underline{\mathbf{w}}_i^{L+1}\right]^T \underline{\delta}^{L+1}(k,q)$$

De la ecuación (1), se tiene:

$$\underline{\delta}^L(k,q) = \mathbf{f}'\left(\underline{\mathbf{u}}^L(k)\right) * \begin{bmatrix} \left(\underline{\mathbf{w}}_1^{L+1}\right)^T \underline{\delta}^{L+1}(k,q) \\ \left(\underline{\mathbf{w}}_2^{L+1}\right)^T \underline{\delta}^{L+1}(k,q) \\ \vdots \\ \left(\underline{\mathbf{w}}_{n_L}^{L+1}\right)^T \underline{\delta}^{L+1}(k,q) \end{bmatrix}$$

El operador * entre dos vectores de igual dimensión se define aquí, como la multiplicación de ellos componente a componente, es decir:

$$\underline{\mathbf{x}} * \underline{\mathbf{y}} = diag\left(\underline{\mathbf{x}}\right)\underline{\mathbf{y}} = \left[\underline{\mathbf{x}} \cdot diag\left(\underline{\mathbf{y}}\right)\right]^T \text{ [1]}$$

Con todo:

$$\underline{\delta}^L\left(k,q\right) = \mathbf{f}'\left(\underline{\mathbf{u}}^L\left(k\right)\right) * \left[\left({}_{(n_L+1)}\mathbf{W}^{L+1}\right)^T \underline{\delta}^{L+1}\left(k,q\right)\right]$$

o también:

$$\underline{\delta}^L\left(k,q\right) = diag\left(\mathbf{f}'\left(\underline{\mathbf{u}}^L\left(k\right)\right)\right)\left({}_{(n_L+1)}\mathbf{W}^{L+1}\right)^T \underline{\delta}^{L+1}\left(k,q\right)$$

Donde la notación ${}_{(p)}\mathbf{A}$ indica que a la matriz **A** se le ha retirado la columna *p*, de manera que la notación matricial queda:

$$\frac{\partial E}{\partial \mathbf{W}^L} = -\frac{1}{K}\sum_{k=1}^{K}\sum_{q=1}^{n_M}\left[d_q\left(k\right) - z_q^M\left(k\right)\right]\underline{\delta}^L\left(k,q\right)\begin{bmatrix}\underline{\mathbf{z}}^{L-1}\left(k\right)\\ 1\end{bmatrix}^T \quad (3)$$

o también:

$$\frac{\partial E}{\partial \mathbf{W}^L} = -\frac{1}{K}\sum_{k=1}^{K}\sum_{q=1}^{n_M}\left[d_q\left(k\right) - z_q^M\left(k\right)\right]\underline{\delta}^L\left(k,q\right)\left[\left(\underline{\mathbf{z}}^{L-1}\left(k\right)\right)^T, 1\right]$$

Esto debido a que en (2), $\frac{\partial u_i^L(k)}{\partial w_{ij}^L}$ no depende de *i* sino solo de *j* y por tanto la expresión:

$$\frac{\partial z_q^M\left(k\right)}{\partial u_i^L\left(k\right)}\frac{\partial u_i^L\left(k\right)}{\partial w_{ij}^L} \text{ para todo } 1 \le i \le n_L, \quad 1 \le j \le n_{L-1}+1$$

da como resultado el producto externo:

$$\frac{\partial z_q^M\left(k\right)}{\partial \underline{\mathbf{u}}^L\left(k\right)}\left[\frac{\partial u_i^L\left(k\right)}{\partial \underline{\mathbf{w}}_i^L}\right]^T$$

tal como se define el producto externo en [7].

La expresión (3) se puede simplificar aun más, si se tiene en cuenta lo siguiente:

$$\frac{\partial E}{\partial \mathbf{W}^L} = -\frac{1}{K}\sum_{q=1}^{n_M}\sum_{k=1}^{K}\left[d_q\left(k\right) - z_q^M\left(k\right)\right]\underline{\delta}^L\left(k,q\right)\begin{bmatrix}\underline{\mathbf{z}}^{L-1}\left(k\right)\\ 1\end{bmatrix}^T$$

y sea:

$e_k^q = d_q\left(k\right) - z_q^M\left(k\right)$ y $\underline{\mathbf{e}}_q = \left[e_1^q, \ldots, e_k^q\right]$

el vector:

$$\left[d_q\left(k\right) - z_q^M\left(k\right)\right]\underline{\delta}^L\left(k,q\right) = e_k^q\,\underline{\delta}^L\left(k,q\right),$$

para todo $1 \le k \le K$, puede escribirse en forma matricial; así:

$$\mathbf{\Lambda}_q^L \triangleq \left[e_1^q\underline{\delta}^L\left(1,q\right), \quad e_2^q\underline{\delta}^L\left(2,q\right), \quad \ldots, \quad e_k^q\underline{\delta}^L\left(K,q\right)\right]$$

donde $\mathbf{\Lambda}_q^L$ puede obtenerse como:

$$\mathbf{\Lambda}_q^L \triangleq \mathbf{1}\left\{n_L,1\right\}\underline{\mathbf{e}}^q * \left[\underline{\delta}^L\left(1,q\right), \quad \underline{\delta}^L\left(2,q\right), \quad \ldots, \quad \underline{\delta}^L\left(K,q\right)\right]$$

$\mathbf{1}\left\{n_L,1\right\}$ es un vector columna de n_L unos, y la operación * se ha definido antes.

Con lo anterior:

$$\frac{\partial E}{\partial \mathbf{W}^L} = -\frac{1}{K}\sum_{q=1}^{n_M}\sum_{k=1}^{K}e_k^q\,\underline{\delta}^L\left(k,q\right)\begin{bmatrix}\underline{\mathbf{z}}^{L-1}\left(k\right)\\ 1\end{bmatrix}^T$$

$$\frac{\partial E}{\partial \mathbf{W}^L} = -\frac{1}{K}\sum_{q=1}^{n_M}\mathbf{\Lambda}_q^L\begin{bmatrix}\underline{\mathbf{z}}^{L-1}\left(1\right) & \underline{\mathbf{z}}^{L-1}\left(2\right) & \ldots & \underline{\mathbf{z}}^{L-1}\left(K\right)\\ 1 & 1 & \ldots & 1\end{bmatrix}^T$$

Sea:

$$\mathbf{1}\mathbf{Z}^{L-1} = \begin{bmatrix}\underline{\mathbf{z}}^{L-1}\left(1\right) & \underline{\mathbf{z}}^{L-1}\left(2\right) & \ldots & \underline{\mathbf{z}}^{L-1}\left(K\right)\\ 1 & 1 & \ldots & 1\end{bmatrix}^T = \begin{bmatrix}\mathbf{Z}^{L-1}\\ \mathbf{1}\left\{1,K\right\}\end{bmatrix}$$

Ahora el cálculo del gradiente se reduce a:

$$\frac{\partial E}{\partial \mathbf{W}^L} = -\frac{1}{K}\sum_{q=1}^{n_M}\mathbf{\Lambda}_q^L\mathbf{1}\mathbf{Z}^{L-1}$$

Con todo se pasa a especificar la función **f**(**u**), la cual puede tomar varias formas. Entre éstas están las formas diferenciables que son necesarias para evaluar cualquier algoritmo basado en el gradiente. Estas formas son las siguientes [2]:

Nombre	Fórmula	Derivada
Logística o *Sigmoid*	$f\left(x\right) = \frac{1}{1+e^{-x}}$	$f'\left(x\right) = f\left(x\right)\left[1 - f\left(x\right)\right]$
Tangente hiperbólica	$f\left(x\right) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'\left(x\right) = 1 - \left[f\left(x\right)\right]^2$
Lineal	$f\left(x\right) = x$	$f'\left(x\right) = 1$

Tabla 1. Funciones de activación diferenciables.

Nótese que para el entrenamiento se hace necesario evaluar las derivadas de las funciones de activación para las neuronas, y con las funciones mencionadas antes se pueden calcular de forma directa a partir del valor de la

[1] Dicha operación es conmutativa.

función y no de su parámetro, esto resulta útil en el entrenamiento.

4. RESULTADOS OBTENIDOS

Dada la formulación anterior, es realizado un algoritmo en Matlab ®, que calcula en cada iteración, el gradiente del error y actualiza las matrices de pesos con varias tasas de entrenamiento η. Además de esto se puede aplicar en algunos casos el método del momento, para varios valores de la constante de momento μ. El algoritmo con momento se da mediante la expresión [1, 2]:

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \Delta\mathbf{W}(t) + \mu\left[\mathbf{W}(t) - \mathbf{W}(t-1)\right]$$

Para hacer uso del algoritmo se ha realizado el entrenamiento de una red neuronal, como un sistema que modela el músculo humano del antebrazo en [8], en ésta se realizó el entrenamiento de once redes neuronales entre las cuales se obtienen resultados para el entrenamiento con una capa oculta y para dos capas ocultas como se muestra en las tablas 1 y 2.

Parámetros	Error $(Nm)^2$	Error porcentual	Tiempo de entrenamiento
η=0,006, μ=0,6	23,9910	0,09%	41 min
η=0,05, μ=0	18,6003	0,08%	25 min
η=0,07, μ=0	22,6619	0,09%	33 min

Tabla 1. Resultados del entrenamiento del modelo del músculo, con 50 neuronas en la capa oculta.

Parámetros	Error $(Nm)^2$	Error porcentual	Tiempo de entrenamiento
Capa 1, 25; Capa 2, 25	5,1495	0,04%	48 min
η=0,05, μ=0	3,9447	0,04%	26 min
η=0,07, μ=0	2,0372	0,03%	55 min

Tabla 2. Resultados del entrenamiento del modelo del músculo, con 2 capas ocultas y con η=0,01.,μ=0,2.

5. CONCLUSIONES Y RECOMENDACIONES

Se ha realizado la deducción de una forma matricial para el entrenamiento de una red neuronal *Backpropagation*, para lo cual se han usado propiedades básicas de las operaciones con matrices, que permiten calcular de forma directa el gradiente del error con respecto a todos los pesos de una capa. Lo anterior ha sido deducido para la forma general de una red neuronal multicapa, donde el gradiente es calculado por capas y de forma no recursiva. Se obtiene en cada capa, el valor del gradiente como una matriz que es entrada directamente al algoritmo de actualización de las matrices de pesos de la red neuronal.

La importancia de la expresión obtenida para el gradiente del error, es que solo debe realizar una suma por las neuronas de la ultima capa, pues el resto se calcula con una operación de multiplicación de matrices que se obtienen a partir de los datos de entrenamiento.

Fue realizado un algoritmo en Matlab ®, que realiza las operaciones necesarias y sirve para entrenar en forma general una red neuronal de M capas con cualquier cantidad de neuronas en cada capa. En este se ha notado que el tiempo que tarda el algoritmo en realizar una actualización de las matrices de pesos no depende en forma estricta del número de neuronas en cada capa, ni del número de datos de entrenamiento, sino más bien del número de capas.

Se ha mostrado como la notación matricial para el entrenamiento de redes neuronales además de simplificar el número de cálculos, también presenta un mejor rendimiento al ser implementada en Matlab ® ya que este programa esta basado en la computación matricial.

6. BIBLIOGRAFÍA

[1] MathWorks, Neural Network Toolbox: User's Guide Version 4. For Use with MATLAB, The MathWorks, Inc., Natick, MA, 2000.

[2] Y. H. Hu and J.-N. Hwang, Handbook of Neural Network Signal Processing. CRC Press LLC, 2002.

[3] B. Widrow and M. H. Jr., Adaptive Switching Circuits, IRE WESCON Convention Record, Pt. 4, pp. 96–104, 1960.

[4] M. T. Hagan, H. B. Demuth, and M. Beale, Neural Network Design. Boston, MA, USA: PWS Publishing Co., 1996.

[5] T. M. Apostol, Calculus, 2nd ed., 1967, vol. 2.

[6] L. Kronecker, Mémoire Sur les Facteurs Irreductibles de l'expression $x^n - 1$, vol. 19, pp. 177–192, 1854, reprinted in *Werke*, Vol. I, pp. 77–92.

[7] A. Gelb, J. Kasper, R. Nash, C. Prince, and A. Sutherland, Applied Optimal Estimation. USA: M.I.T. Press, 1974.

[8] J. E. Molina, "Desarrollo de un sistema de control de un exoesqueleto para asistencia del movimiento tipo flexión y extensión", Universidad Tecnológica de Pereira, 2006.