AJUSTE E INTEGRACIÓN NUMÉRICA DE FUNCIONES MUY OSCILANTES USANDO REDES NEURONALES ARTIFICIALES SUPERVISADAS

Iván Chaman-García[1] y Lourdes Sandoval-Solís[2]
Facultad de Ciencias de la Computación
Benemérita Universidad Autónoma de Puebla, México

ABSTRACT
This paper presents a methodology to adjust and integrate numerically very oscillating functions, even with a discontinuity, using a vectorial supervised artificial neural network (RNASV), which consists of three layers: one input, one output and one hidden learning layer, in which the activation function is an array. It is proposed to adjust very oscillating functions $f(x)$ as a linear combination of cosine functions $f(x) = \sum_{i=0}^{n} w_i cos(ix),\ x \in [0, \pi]$.
We present the important properties of the Hessian matrix derived from the resulting matrix activation function in the case that the nodes are equidistant $x_k = x_0 + kh$, where h is the spacing between nodes.
To adjust very oscillating functions, there are several training techniques of the RNASV, two constant factor learning and two variable factor learning. Using known highly oscillating functions, shows the comparison between different training techniques respect the number of training and time required to obtain a good fit.
Finally, we present the numerical approximation of definite integrals highly oscillating functions, obtained from the integral of the linear combination that approximates cosine functions $\int_a^b f(x)dx \approx (b-a)w_0 + \sum_{i=1}^{n} \frac{b-a}{i} w_i[sin(i\pi)]$
This shows the comparison with exact results and Adaptive Simpson's method, for each of the different techniques of training the neural network.

KEYWORDS: numerical integration, supervised neural network, highly oscillating functions

MSC: 65,68

RESUMEN
En este trabajo se presenta una metodología para ajustar e integrar numéricamente funciones muy oscilantes, incluso que tengan una discontinuidad, usando una red neuronal artificial supervisada vectorial (RNASV), que consta de tres capas, una de entrada, otra de salida y una capa oculta de aprendizaje, en la cual la función de activación es una matriz. Se propone ajustar las funciones muy oscilantes $f(x)$ como una combinación lineal de funciones coseno, es decir $f(x) = \sum_{i=0}^{n} w_i cos(ix),\ x \in [0, \pi]$.
Se presentan las propiedades importantes de la matriz Hessiana derivada de la función de activación matricial resultante, en el caso de que los nodos sean equidistantes $x_k = x_0 + kh$, donde h es el espaciamiento entre los nodos.
Para ajustar las funciones muy oscilantes, se presentan varias técnicas de entrenamiento de la RNASV, dos con factor de aprendizaje constante y dos con factor de aprendizaje variable. Usando funciones muy oscilantes conocidas, se muestra la comparación entre las diferentes técnicas de entrenamiento respecto al número de entrenamientos y tiempo requerido para obtener un buen ajuste.
Por último, se presenta la aproximación numérica de las integrales definidas de las funciones muy oscilantes, obtenida de la integral definida de la combinación lineal de funciones coseno que las aproxima, es decir $\int_a^b f(x)dx \approx (b-a)w_0 + i=1nb-aiwisiniπ$

Se muestra la comparación con los resultados exactos y el método de Simpson Adaptivo, para cada una de las diferentes técnicas de entrenamiento de la red neuronal.

1. Introducción

Las funciones muy oscilantes son importantes para describir fenómenos de las áreas como son sismología, campos electromagnéticos, espectroscopia, etc. (Palma, Sandoval, Churyumov, Chavusshyan, & Bereshnoy, 2007).

[1] chaman030687@gmail.com
[2] sandoval@siu.buap.mx

Actualmente existe una gran tendencia a establecer nuevas maneras de resolver problemas que no pueden ser descritos de una manera fácil por algoritmos tradicionales, estas nuevas técnicas de resolverlo tiene su inspiración en la emulación de sistemas biológicos, como son el uso de Redes Neuronales Artificiales (RNA) (Montaño Moreno, 2002).

Haciendo una revisión sobre el ajuste de curvas con RNA en aplicaciones prácticas, los trabajos presentan el uso de funciones con base radial de solo tres capas en la red neuronal y el uso de redes multicapa (Leonard, MA, & LH, Sep 1992), muestran una buena aproximación para datos donde se utilizan nodos equidistantes y no equidistantes (Iglesias, Echevarria, & Galvez, Nov 2004). Pero existe una diferencia importante en todos los trabajos revisados en las arquitecturas de las RNA, los datos son entrenados uno a uno (Blanco M. & J, Dec 1995) y los pesos sinápticos son tomados de manera matricial (Bezazi A., Sg, & al., Apr 2007), mientras en la arquitectura presentada en este trabajo, las capas y los pesos sinápticos son tomadas de forma sincronizada en forma de un vector y no entre cada componente del vector (uno por uno), haciendo que nuestra red (n+1)-dimensional haga uso de una matriz de activación para aproximar la función (Zeng & Yao-Nan, Jul/Aug 2005), en lugar de n funciones de activación unidimensionales.

En este trabajo se ajusta una función muy oscilante a una combinación lineal de funciones coseno, y para determinar los coeficientes de la combinación lineal se utiliza una red neuronal vectorial supervisada con cuatro técnicas de entrenamiento: Back-Propagation y Momentum, con factor de aprendizaje constante y, Gradientes Conjugados y el método de Newton con factor de aprendizaje variable, éstas últimas utilizan las propiedades de la matriz Hessiana derivada de la función de activación.

En la primera sección se presenta la arquitectura de la RNASV propuesta. En la segunda sección se presentan las diferentes técnicas de entrenamiento de la red, así como las propiedades de la matriz Hessiana. En la tercera sección se muestran las tablas obtenidas al ajustar funciones muy oscilantes conocidas, algunas con una discontinuidad, usando las diferentes técnicas de entrenamiento, respecto al número de entrenamientos y tiempo requerido para obtener un buen ajuste. También se muestran gráficas donde se observa cómo se va aproximando la combinación lineal a la función muy oscilante conforme la red neuronal aprende. En la cuarta sección, se presenta la aproximación numérica de las integrales definidas de las funciones ajustadas en la tercera sección presentándose los cuadros comparativos con los resultados exactos y las aproximaciones numéricas usando el método de Simpson Adaptivo. Finalmente, se dan las conclusiones y las referencias requeridas para realizar este trabajo.

2. DESCRIPCIÓN DE LA RED NEURONAL

Para cualquier función $f(x)$ continua en $[a, b]$ se puede escribir como una combinación lineal de funciones cosenos (Burden & Douglas Faires, 2002), en los puntos conocidos $(x_0, f(x_0)), (x_1, f(x_1)), \dots, (x_n, f(x_n))$,

$$f(x) = w_0 + w_1\, cos(x) + w_2\, cos(x) + \cdots + w_n\, cos(x), \tag{1}$$

Sustituyendo

$$\begin{aligned} f(x_0) &= w_0 + w_1\, cos(x_0) + w_2\, cos(x_0) + \cdots + w_n\, cos(x_0) \\ f(x_1) &= w_0 + w_1\, cos(x_1) + w_2\, cos(x_1) + \cdots + w_n\, cos(x_1) \\ &\vdots \\ f(x_n) &= w_0 + w_1\, cos(x_n) + w_2\, cos(x_n) + \cdots + w_n\, cos(x_n), \end{aligned} \tag{2}$$

entonces en forma matricial se puede escribir

$$Y^k = CW \tag{3}$$

Definiendo el error E, donde F es la salida deseable y Y^k es el valor de la combinación lineal. Observe que la matriz C es simétrica y constante para nodos equidistantes $x_k = x_0 + kh$,

$$E = F - Y^k \tag{4}$$

Así el error cuadrático es:

$$E^t E = (F - Y)^t (F - Y) \quad \textbf{(5)}$$

Para obtener los mejores pesos se desea reducir el error, por lo que se puede plantear en forma equivalente a un problema de mínimos cuadrados.

$$min\left\{\frac{1}{2}E^t E\right\} = min\left\{\frac{1}{2} = (F - Y)^t (F - Y)\right\} = min\{\phi\} \quad \textbf{(6)}$$

Donde el gradiente (g^k) y el Hessiano (Q) están dados por (ver Apéndice)

$$\nabla\phi = g^k = -CE^k \quad \textbf{(7)}$$
$$\nabla^2\phi = CC^t = Q \quad \textbf{(8)}$$

Este problema de mínimos cuadrados se puede interpretar como una red neuronal artificial supervisada, descrita más adelante, donde los W^k son los pesos sinápticos de la red neuronal artificial y la matriz C es la función de activación.

La arquitectura de la RNASV es de tres capas, en la primera capa se ingresan vectorialmente el vector de pesos sinápticos inicial W^0 y la salida deseable F, en la tercera capa que es la de salida se obtiene el vector de pesos sinápticos W^k que minimiza el error cuadrático

$$\phi = \frac{1}{2}\|F - Y^k\|^2, \quad \textbf{(9)}$$

en la capa intermedia la red neuronal aprende de las reglas de entrenamiento. En este trabajo se presentan cuatro técnicas de entrenamiento, las dos primeras con un factor de aprendizaje constante Back-Propagation y Factor Momentum, y las otras dos con una factor de aprendizaje variable conocidos como Gradientes Conjugados y Método de Newton

En la figura 1 se describe de manera general la arquitectura de la Red Neuronal Supervisada utilizada para el ajuste de una función.

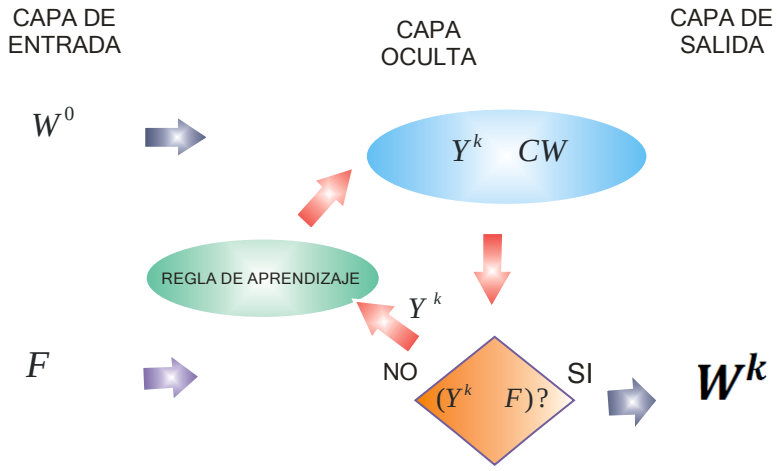

Figura 1. Arquitectura de la Red Neuronal

3. ALGORITMO GENERAL DE LA RED NEURONAL

Dados los siguientes parámetros iniciales:
Un $n \geq 5\ \epsilon\ N$ como número de nodos de la función.
Un vector $X \epsilon R^{n+1}$ donde $X = \{x_0, x_1, \dots, x_n\}$ como patrón de entrada.
Un vector de pesos sinápticos aleatorios $W^0 \epsilon R^{n+1}$ donde $W^0 = \{w_0, w_1, \dots, w_n\}$ en el intervalo [0,1].
Se calcula una matriz de activación $C \epsilon M_{(n+1)x(n+1)}$, donde C (ver Ecuación 2).

En el intervalo de integración $[a, b]$ se hace un cambio de variable al intervalo $[0, \pi]$.
Una salida deseada F donde $F = \{f(x_0), f(x_1), \dots, f(x_n)\}$.
Una tolerancia tol cualquiera y sea $k = 0$.
Un factor de aprendizaje $\eta = \frac{1.35}{\|C\|^2}$.

Paso 1: Calcular el vector Y^k (Ver Ecuación 3)

Paso 2: Calcular el vector error con respecto a los valores reales deseados $\phi^k = \frac{1}{2}\|F - Y^k\|^2$.

Paso 3:

Si $\phi^k < tol$ entonces
Ir al paso 4
Si no
Hacer $k = k + 1$
Utilizar la regla de aprendizaje para obtener la siguiente W^k
Ir al Paso 1.

Paso 4: Detener y el vector solución es W^k

Observe que en el paso 3 se aplica cada una de las diferentes técnicas de aprendizaje de la RNASV, las cuales se describen a continuación.

RNASV con Factor de Aprendizaje Constante

Back-Propagation

Este es el **algoritmo clásico de Redes Neuronales**, también recibe el nombre de regla delta generalizada que utiliza la técnica de gradiente descendente basada en el criterio de mínimos cuadrados, ya que en cada paso, el vector de pesos W es ajustado en la dirección en la cual, la función error decrece más rápidamente. Esta dirección está determinada por el gradiente de la superficie error del punto actual, en el espacio del vector de pesos. Usando el gradiente del error, podemos ajustar el vector de pesos para la conexión en una dirección negativa al gradiente. Por lo tanto, el vector de pesos es actualizado como:

$$W^{k+1} = W^k + \eta CE^k. \quad \textbf{(10)}$$

Donde CE^kes el gradiente de ϕ (ver Apéndice).

Factor Momentum

Se conoce que el algoritmo anterior oscila mucho por lo que para **reducir la oscilación** del método Back-Propagation, se usa el algoritmo de entrenamiento Momentum, donde se modifica la función error de modo que una porción del vector de pesos anterior es modificado completamente por el vector de pesos actual. Por lo tanto, el vector de pesos es actualizado como:

$$W^{k+1} = W^k + \eta(\lambda + 1)CE^k. \quad \textbf{(11)}$$

RNASV con Factor de Aprendizaje Variable

Para los algoritmos de entrenamiento siguientes se aprovecha que la matriz Hessiana derivada de la función de activación es simétrica y definida positiva, por lo que se puede hacer el uso de métodos de optimización, haciendo que el factor de aprendizaje sea variable, esto permite mejorar considerablemente el tiempo y número de entrenamientos de la Red Neuronal.

En general la matriz Hessiana para n nodos equidistantes, cumple que:

$CC^t(1,1) = n$
$CC^t(k,k) = \frac{(n+1)}{2}$, para $2 \leq k \leq n-1$
$CC^t(n,n) = n$

$$CC^t(i,j) = \begin{cases} 1, & i+j \; es \; par \\ 0, & i+j \; es \; impar \end{cases}$$

Por otra parte, si una matriz es estrictamente dominante diagonalmente y sus elementos de la diagonal son positivos entonces la matriz es definida positiva (Abad, Gassó, & Torregrosa, 2009).

Gradientes Conjugados

Gradientes Conjugados resuelve adecuadamente problemas de optimización con Hessiano definido positivo, por lo que este método se usa para entrenar la Red Neuronal. Para minimizar ϕ (ver ecuación 9), actualizamos el vector de peso W según una regla de Gradientes Conjugados (Luenberger, August 1984).

$$W^{k+1} = W^k + \alpha^k p^k, \tag{12}$$

donde p^k es una dirección de descenso conjugada y α^k es el factor de aprendizaje adaptable con respecto a la k-ésima iteración. Podemos describir la regla de aprendizaje con Gradientes Conjugados como sigue (Luenberger, August 1984):

Definir $g^0 = \nabla\phi(0) = -CE^0$, $p^0 = CE^0$, donde E^0 es el primer vector error.

Paso 1: Calcular el factor de aprendizaje $\alpha^k = \frac{\phi^k}{\|g^k\|^2}$.

Paso 2: Calcular el nuevo vector de pesos $W^{k+1} = W^k + \alpha^k p^k$.

Paso 3: Calcular el vector de salida dada la ecuación Y^k (ver Ecuación 3)

Paso 4: Calcular el vector error con respecto a los valores reales deseados $E^k = \frac{1}{2}\|F - Y^k\|^2$.

Paso 5: Calcular el índice de búsqueda conjugado: $\beta^k = \frac{\|g^k\|^2}{\|g^{k-1}\|^2}$.

Paso 6: Actualizar el vector dirección conjugado $p^{k+1} = -g^{k+1} + \beta^k p^k$.

Los métodos de Gradientes Conjugados se puede expresar de **manera residual**, esto nos permite obtener una nueva regla de aprendizaje para problemas de mayor escala (Chaman Garcia, Mayo 2010).

Método de Newton

El **método clásico** para obtener W^*(vector de pesos solución) es el método de Newton (Escudero, 1982) que, dada una estimación inicial W^0, obtiene una secuencia de direcciones de búsqueda p^k tal que la iteración k resuelve el sistema:

$$Qp^k = -g^k, \tag{13}$$

donde $Q = CC^t$ es la matriz Hessiana de ϕ (ver Apéndice). Se obtiene la estimación $W^{k+1} = W^k + \alpha^k p^k$, donde α^k es el factor de aprendizaje k, tal que minimiza $\phi(W)$ en la dirección p^k.

El método de Newton es local y cuadráticamente convergente, cuando la matriz Hessiana es definida positiva, por lo que para resolver el sistema de ecuaciones se usa la factorización de Cholesky.

Ahora podemos ajustar la red neuronal con el vector de pesos W, el algoritmo que modifica el vector de pesos sinápticos es el siguiente:

Definir $Q = CC^t$, $g^0 = -CE^0$, donde E^0 es el primer vector de error.

Paso 1:

Resolver el sistema $Qp^k = -Qg^k$. usando Factorización de Cholesky para $Q = LL^t$.

Resolver $L^t s = -g^k$ para s por sustitución hacia atrás.

Resolver $Lp^k = s$ para p^k por sustitución hacia adelante.

Paso 2: Calcular el nuevo vector de pesos $W^{k+1} = W^k + p^k$

Paso 3: Calcular el vector de salida dada la ecuación Y^k (ver Ecuación 3).

Paso 4: Calcular el vector error con respecto a los valores reales deseados $\phi^k = \frac{1}{2}\|F - Y^k\|^2$
Paso 5: Actualizar el gradiente $g^{k+1} = -CE^k$

Otro posible método de entrenamiento es el método de **Newton Truncado,** produce buenos resultados en problemas de gran escala; su convergencia es súper-lineal, elimina inconvenientes teóricos del método de newton y no requiere obtener y almacenar la matriz Hessiana, cumpliendo las mismas condiciones de las técnicas anteriores para que converja. (Chaman Garcia, Mayo 2010).

4. PRUEBA DE AJUSTE DE FUNCIONES MUY OSCILANTES

Las Redes Neuronales fueron **implementadas en Java** como software libre en una laptop HP-Pavilion dv4-1140go Core2Duo, 2GHz, 4GB RAM. A continuación en la figura 2 se presentan las gráficas del ajuste de la función $F = \frac{cos(201x)}{cos\,(x)}$ con 201 nodos, una tolerancia de $1e-10$, aumentando el número de entrenamientos usando RNASV Back-Propagation.

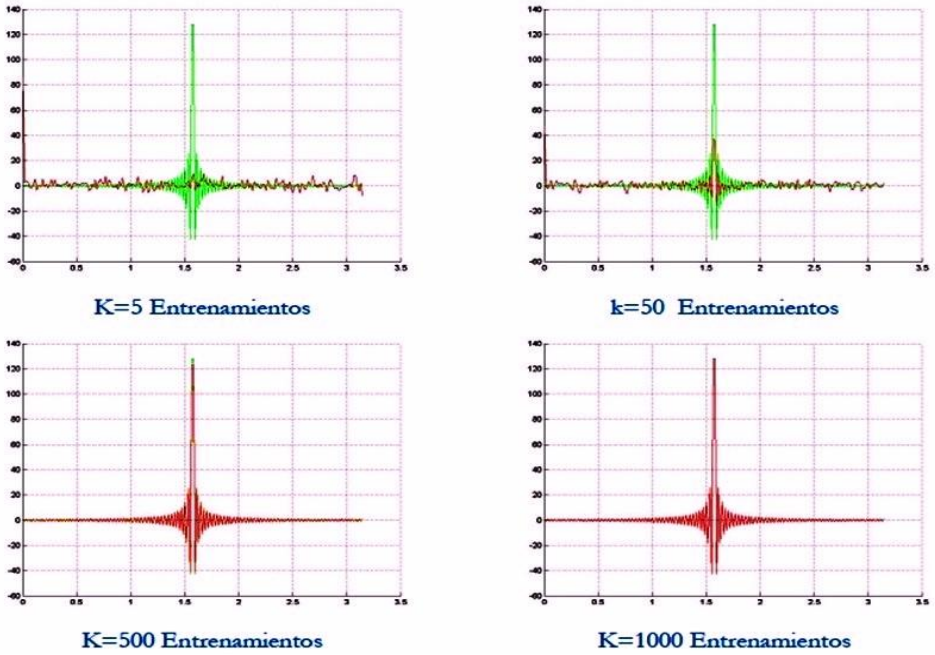

Figura 2. $F = \frac{cos(201x)}{cos\,(x)}$ con 201 nodos, una tolerancia de $1e-10$, aumentando el número de entrenamientos.

A continuación se presentan los cuadros comparativos para Funciones Oscilantes Continuas (Gradshteyn & Ryzhik, March 2007) con los diferentes métodos de entrenamiento:

- $sin(mx)$
- $x^3 sin(mx)$
- $e^{ax} sin\,(bx)$
- $sin(ax)cos(bx)$
- $\big(1-cos(x)\big)^m sin(mx)$

$sin(5x)$	K	Error	Tiempo
Back-Propagation	**12268**	**9.98e-11**	**588.35**
Factor Momentum	**4512**	**9.97e-11**	**456.43**
Gradientes Conjugados	**18**	**7.36e-11**	**1.250**
Newton	**1**	**6.91e-22**	**5.353**

$sin(100x)$	k	Error	Tiempo
Back-Propagation	**12262**	**9.98e-11**	**619.35**
Factor Momentum	**4521**	**9.93e-11**	**454.57**
Gradientes Conjugados	**18**	**7.63e-11**	**1.142**
Newton	**1**	**7.55e-22**	**5.212**

Funciones Oscilantes con una Discontinuidad (Gradshteyn & Ryzhik, March 2007):

- $x \cdot tan(x)$
- $sin\big(m \cdot cot(x)\big) sin(2x)$
- $\frac{cos(x)}{cos\,(x)}$
- $\frac{sin(3x)}{cos(x)}$

$\frac{cos(3x)}{cos(x)}$	K	Error	Tiempo
Back-Propagation	**12296**	**9.99e-11**	**457.74**
Momentum	**6151**	**9.98e-11**	**351.08**
Gradientes Conjugados	**18**	**7.32e-11**	**1.12**
Newton	**1**	**7.63e-22**	**5.45**

$\frac{cos(201x)}{cos(x)}$	k	Error	Tiempo
Back-Propagation	**12936**	**9.97e-11**	**510.19**
Momentum	**6464**	**9.99e-11**	**366.67**
Gradientes Conjugados	**18**	**7.35e-11**	**1.15**
Newton	**1**	**7.14e-22**	**5.53**

$x\,tan(x)$	K	Error	Tiempo
Back-Propagation	**13804**	**9.97e-11**	**506.35**
Momentum	**6895**	**9.97e-11**	**435.23**
Gradientes Conjugados	**17**	**7.63e-11**	**1.12**
Newton	**1**	**1.00e-20**	**5.84**

$sin(5\,cot(x))\,sin(x)$	k	Error	Tiempo
Back-Propagation	**12277**	**9.99e-11**	**604.05**
Momentum	**4529**	**9.96e-11**	**526.87**
Gradientes Conjugados	**18**	**7.57e-11**	**1.17**
Newton	**1**	**5.06e-20**	**5.73**

$sin(100\,cot(x))\,sin(x)$	K	Error	Tiempo
Back-Propagation	**12252**	**9.98e-11**	**616.10**
Momentum	**4522**	**9.96e-11**	**842.49**
Gradientes Conjugados	**18**	**7.80e-11**	**1.11**
Newton	**1**	**5.34e-20**	**5.33**

Observe que en todos los cuadros comparativos las RNASV con un menor número de entrenamientos es la que utiliza el método de newton y la RNASV con menor tiempo de cálculo es la que utiliza el método de gradientes conjugados, como algoritmos de entrenamiento.

5. INTEGRACIÓN NUMÉRICA DE FUNCIONES MUY OSCILANTES

Funciones		$\int_0^{10} x^3 \sin(5x)\,dx$	$\int_0^{10} x^3 \sin(10x)\,dx$	$\int_0^{2\pi} (1-\cos(x))^5 \sin(5x)\,dx$	$\int_0^{2\pi} (1-\cos(x))^{10} \sin(10x)\,dx$
Integral Real		**-195.65918**	**-87.69068**	**0**	**0**
Error Simpson Adaptivo		**1.681e-2**	**8.260e-3**	**7.771e-15**	**6.411e-13**
Back-Propagation	N	1000	1000	1001	91
	K	15044	15045	12718	1341
	Tiempo (s)	746.608677	759.895155	645.964104	0.396049
	Error Integral	**2.273e-2**	**6.234e-2**	**8.859e-14**	**1.151e-13**
Momentum	N	1000	1000	1001	91
	K	7518	7519	12718	1341
	Tiempo (s)	547.404727	552.913711	645.964104	0.396049
	Error Integral	**2.273e-2**	**6.234e-2**	**8.859e-14**	**1.151e-13**
	λ	1	1	1000	100
Gradientes Conjugados	N	3000	3000	3000	3000
	K	18	18	19	18
	Tiempo (s)	6.827979	6.873922	6.977661	6.990940
	Error Integral	**1.242e-2**	**4.168e-4**	**5.761e-07**	**4.595e-09**
Newton	N	3000	3000	300	30
	K	1	1	1	1
	Tiempo (s)	417.562057	465.044473	2.047893	1.186237
	Error Integral	**1.242e-002**	**4.169e-4**	**6.580e-016**	**5.838e-014**

El uso de nodos no equidistantes no garantiza que los matriz Hessiana sea simétrica y definida positiva, por lo que tampoco garantiza que los algoritmos de entrenamiento converjan.

Para obtener la integral numérica de funciones muy oscilante se aprovecha el ajuste como una combinación lineal de funciones cosenos, usando las RNASV.

Sea $f(x)$ definida en $[a,b]$, y $W=[w_0,w_1,\dots,w_N]^t$ el vector pesos de la red neuronal, considerando que el ajuste se obtuvo haciendo un cambio de variable del intervalo $[a,b]$ al intervalo $[0,\pi]$, se tiene:

$$I=\int_a^b f(x)dx \approx (b-a)w_0+\sum_{i=1}^{N}\frac{b-a}{i}w_i[sin(i\pi)]. \qquad \textbf{(14)}$$

Funciones		$\int_0^{\frac{\pi}{2}} sin(5\,cot(x))sin(2x)$	$\int_0^{\frac{\pi}{2}} sin(10\,cot(x))sin(2x)$	$\int_0^{\frac{\pi}{2}} sin(50\,cot(x))sin(2x)$	$\int_0^{\frac{\pi}{2}} sin(100\,cot(x))sin(2x)$
Integral Real		**5.29197e-2**	**7.13140e-4**	**1.514836e-20**	**5.843481e-42**
Error Simpson Adaptivo		**7.177e-5**	**2.480e-4**	**5.600e-4**	**1.998e-3**
Back-Propagation	N	3	200	750	750
	K	12277	2248	9042	9060
	Tiempo (s)	604.052392	0.901774	217.652134	216.172727
	Error Integral	**2.948e-4**	**8.942e-6**	**2.277e-3**	**8.754e-4**
Momentum	N	60	1000	90	1000
	K	325	6129	492	6127
	Tiempo (s)	0.160450	878.928895	3.519536	842.499731
	Error Integral	**1.267e-4**	**2.839e-5**	**1.423e-3**	**1.021e-4**
	λ	1	1	1	1
Gradientes Conjugados	N	3000	3000	3000	3000
	K	19	19	19	19
	Tiempo (s)	6.979154	7.053597	7.149253	6.822643
	Error Integral	**1.069e-4**	**9.798e-5**	**1.619e-4**	**4.645e-4**
Newton	N	3000	3000	300	
	K	1	1	1	1
	Tiempo (s)	418.732004	422.344307	448.496313	461.996848
	Error Integral	**4.651e-5**	**5.503e-5**	**7.997e-6**	**5.585e-4**

Las funciones oscilantes de esta prueba son las siguientes (Gradshteyn & M., 2007):

Funciones Oscilantes Continuas:

- $\int sin\,(mx)\,dx=-\frac{1}{m}cos\,(mx)$
- $\int x^3 sin(mx)\,dx=\left(\frac{3x^2}{m^2}-\frac{6}{m^4}\right)sin(mx)+\left(\frac{6x}{m^3}-\frac{x^3}{m}\right)cos\,(mx)$
- $\int e^{ax} sin\,(bx)\,dx=\frac{e^{ax}}{a^2+b^2}\left(a\,sin(bx)-b\,cos(bx)\right)$
- $\int sin(ax)cos(bx)\,dx=-\frac{cos(a-b)}{2(a-b)}-\frac{cos(a+b)}{2(a+b)}$
- $\int_0^{2\pi}\left(1-cos(x)\right)^m sin(mx)\,dx=0$

Funciones Oscilantes con Discontinuidad:

- $\int_0^{\frac{\pi}{2}} xtan(x)\,dx = -\pi\ ln(2)$
- $\int_0^{\frac{\pi}{2}} sin\left(m\ cot(x)\right) sin(2x)\,dx = \frac{m\pi}{2} e^{-m}$
- $\int_0^{\pi} \frac{cos\left((2m+1)x\right)}{cos\,(x)} dx = (-1)^m \pi$

$$\int \frac{sin(3x)}{cos(x)} dx = 2sin^2(x) - 4ln\left(cos(x)\right)$$

Funciones		$\int_0^{\pi} x\,tan(x)dx$	$\int_0^{\pi} \frac{sin(3x)}{cos(x)} dx$
Integral Real		**-2.177586090303602**	**0**
Error Simpson Adaptivo		**9.663191186362639**	**6.151779252201354**
Back-Propagation	N	1001	1001
	K	13804	3819
	Tiempo (s)	506.358620	5.052069
	Error Integral	**2.578e-06**	**1.837e-14**
Momentum	N	1000	301
	K	6898	1906
	Tiempo (s)	329.886692	3.121622
	Error Integral	**2.578e-06**	**1.291e-14**
	λ	1	1
Gradientes Conjugados	N	3001	3001
	K	18	19
	Tiempo (s)	6.874513	6.923010
	Error Integral	**3.340e-07**	**5.981e-08**
Newton	N	31	5
	K	1	1
	Tiempo (s)	4.293082	4.292185
	Error Integral	**4.44e-016**	**3.487e-016**

Funciones		$\int_0^{\pi} \frac{cos(3x)}{cos\,(x)} dx$	$\int_0^{\pi} \frac{cos(11x)}{cos\,(x)} dx$	$\int_0^{\pi} \frac{cos(21x)}{cos\,(x)} dx$	$\int_0^{\pi} \frac{cos(201x)}{cos\,(x)} dx$
Integral Real		**-3.1415926535897**	**-3.141592653589793**	**3.14159265358979**	**3.1415926535897**
Error Simpson Adaptivo		**0**	**1.7674e-013**	**1.4805e-012**	**1.5107e-008**
Back-Propagation	N	1001	1001	1001	1001
	K	12296	12351	12422	12936
	Tiempo (s)	457.744996	419.042243	500.849788	510.196471
	Error Integral	**1.327e-13**	**1.301e-13**	**1.030e-13**	**1.052e-13**
Momentum	N	1001	1001	1001	301
	K	6140	6180	6222	1888
	Tiempo (s)	214.875383	219.364123	295.756086	2.600673
	Error Integral	**9.681e-14**	**8.704e-14**	**8.482e-14**	**5.195e-14**
	λ	1	1	1	1
Gradientes Conjugados	N	3001	3001	3001	3001
	K	19	19	19	19
	Tiempo (s)	6.954365	7.040782	7.051032	7.111690
	Error Integral	**1.496e-07**	**1.973e-07**	**2.34106e-07**	**5.6713e-08**
Newton	N	51	81	201	3001
	K	1	1	1	1
	Tiempo (s)	12.872087	11.914467	11.650442	448.048384
	Error Integral	**0**	**0**	**1.554e-014**	**2.575e-014**

6. CONCLUSIONES

Se muestra que el problema de mínimos cuadrados para ajustar funciones muy oscilantes a una combinación lineal de funciones coseno, es equivalente a entrenar a una red neuronal artificial supervisada vectorial. En el caso de nodos equidistantes la matriz Hessiana es simétrica y definida positiva.

La RNASV consta de tres capas donde los vectores son entrenados de manera sincronizada y la función de activación está en función de una matriz simétrica.

Se realizaron cuatro técnicas para ajustar funciones muy oscilantes usando un enfoque de Redes Neuronales Artificiales Supervisadas; para entrenar esta RNASV se utilizó algoritmos clásicos con un factor de aprendizaje constante y con un factor variable de aprendizaje, donde estos aprovechan la potencialidad de los métodos de optimización, como son los Gradientes Conjugados y el método de Newton.

El método de Gradientes Conjugados presenta el menor tiempo de ejecución para entrenar la RNASV, mientras que el método de Newton requiere de un solo entrenamiento, en general, los métodos con las RNASV con factor de aprendizaje variable reducen el número de entrenamientos y el tiempo de cálculo.

Respecto a la aproximación numérica de integrales definidas de funciones muy oscilantes, se observa que al comparar los resultados para varias funciones muy oscilantes continuas la aproximación de la integral es equivalente al método de Simpson Adaptivo. Mientras que para las funciones con una discontinuidad se mejora el método de Simpson Adaptivo para aproximar numéricamente la integral definida.

Se creó una aplicación en Java donde se implementaron todas las RNASV que permite la aproximación de la integral definida para funciones de manera general

Agradecimientos: Uno de los autores (Chaman-García, Iván) agradece el apoyo económico de PROMEP-SEP y VIEP-BUAP para la realización del trabajo. También agradecemos a los revisores de la primera versión de este trabajo ya que gracias a sus observaciones se mejoró el presente.

RECEIVED APRIL, 2010
REVISED JULY, 2011

REFERENCIAS

[1] ABAD, M., GASSÓ, M., y TORREGROSA, J. (2009): Algunos Resultados acerca de B-matrices. **XXI Congreso de Ecuaciones Diferenciales y Aplicaciones**, 1-8. Ciudad Real, España.

[2] BEZAZI A., P., SG, W. K., and AL., E. (2007): Fatugue Life Prediction of Sanwich Composite Materials under Flexual tests using a Bayesian trained Artificial Neural Network. **International Journal of Fatigue**, 29, 738-747.

[3] BLANCO M., C., and J, I. H. (1995): Articial Neural Networks for multicomponent Kinetic Determinations. **Analytical Chemistry**, 67, 4477-4483.

[4] BURDEN, R. L., and DOUGLAS FAIRES, J. (2002): **Numerical Analysis** (2 ed.). International Thomson Editores, México

[5] CHAMAN GARCIA, I. (2010): **Integración Numérica con Redes Neuronales**. Puebla: Benemérita Universidad Autónoma de Puebla.

[6] ESCUDERO, L. F. (1982): On Diagonally-Preconditioning the Truncated-Newton Method for Super-Scale Linearly Constrained Nonlinear Programming. **Questiió: Quaderns d'Estadística, Sistemes, Informatica i Investigació Operativa**, 6, 261-281.

[7] GRADSHTEYN, I. S. and RYZHIK, I. M. (2007): **Table of integrals, Series, and Products** (7 ed.). (A. Jeffrey, D. Zwillinger, Edits., & S. Technica, Trad.) Academic Press, N. York.

[8] IGLESIAS, A., ECHEVARRIA, G. and & GALVEZ, A. (2004): Functional Networks for B-spline Surface Reconstruction. **Future Generation Computer Systems**, 20, 1337-1363.

[9] LEONARD, J., MA, K., and LH, U. (1992): A Neural Network Architecture that Computes its owm Reliability. **Computers & Chemical Engineering**, 16, 819-835.

[10] LUENBERGER, D. G. (1982): **Linear and Nonlinear Programming** (2 ed.). Addison-Wesley Inc. Wilmington, Delaware .

[11] MONTAÑO MORENO, J. J. (2002): **Redes Neuronales Artificiales aplicadas al Análisis de Datos** (Tesis Doctoral). Universitat De Les Illes Balers, Facultad de Psicología.

[12] PALMA, A., SANDOVAL, L., CHURYUMOV, K., CHAVUSSHYAN, V., & BERESHNOY, A. (2007): Franck-Condon Factors for Molecules Observed in Comets. **International Journal Quantum Chemistry**, 107, 2650-2653.

[13] ZENG, Z. Z. W., and YAO-NAN, W. H. (2005): Numerical Integration Based on a Neural Network Algorithm. **IEEE**, 8, 42-48 7.

Apéndice

Haciendo su desarrollo de Taylor de la Ecuación (9), hasta segundo orden, tenemos que:

$$\phi(W) = \phi(W^k) + \nabla\phi(W^k)(W - W^k) + \tfrac{1}{2}(W - W^k)^t\nabla^2\phi(W^k)(W - W^k).$$

Por definición de Gradiente y la condición de primer orden $(\nabla\phi) = 0$ tenemos que:

$$\nabla\phi(W) = \nabla\phi(W^k) + \nabla^2\phi(W^k)(W - W^k) = 0, \quad \textbf{(A1)}$$

es decir:

$$\nabla^2\phi(W^k)(W - W^k) = -\nabla\phi(W^k).$$
$$W = W^k - [\nabla^2\phi(W^k)]^{-1}\nabla\phi(W^k). \quad \textbf{(A2)}$$

donde $\eta = [\nabla^2\phi(W^k)]^{-1}$ y $\nabla\phi(W^k) = \frac{\partial\phi}{\partial W^k}$ sustituyendo en la ecuación (A2) se tiene:

$$W^{k+1} = W^k - \eta\frac{\partial\phi}{\partial W^k}, \quad \textbf{(A3)}$$

donde $\eta > 0$ es el factor de aprendizaje. La ecuación (A3) es nuestra regla de aprendizaje que se sigue para modificar los pesos sinápticos de la RNASV Back-Propagation. Por otro lado, diferenciando el error cuadrático usando la regla de la cadena tenemos que:

$$\frac{\partial\phi}{\partial W^k} = \frac{\partial\phi^k}{\partial E^k}\frac{\partial E^k}{\partial Y^k}\frac{\partial Y^k}{\partial W^k}, \quad \textbf{(A4)}$$

donde

$$Y^k = C^tW \Rightarrow \frac{\partial Y^k}{\partial W^k} = C^t.$$
$$E^k = F - Y \Rightarrow \frac{\partial E^k}{\partial Y^k} = -1.$$
$$\phi = \frac{1}{2}\|E^k\|^2 \Rightarrow \frac{\partial\phi^k}{\partial E^k} = (E^k)^t.$$

Finalmente sustituyendo en la ecuación (A4):

$$\frac{\partial\phi}{\partial W^k} = (E^k)^t(-1)C^t = -(E^kC)^t.$$

Puesto que $(AB)^t = B^tA^t$ y $(A^t)^t = A$ tenemos que

$$\nabla\phi = g^k = \frac{\partial\phi}{\partial W^k} = -CE^k. \quad \textbf{(A5)}$$

Calculando la segunda derivada de ϕ tenemos:

$$\nabla^2\phi = \frac{\partial}{\partial W^k}\left(\frac{\partial\phi}{\partial W^k}\right) = -\frac{\partial CE^k}{\partial W^k} = -\frac{\partial CE^k}{\partial E^k}\frac{\partial E^k}{\partial Y^k}\frac{\partial Y^k}{\partial W^k} = -C(-1)C^t = CC^t = Q, \quad \textbf{(A6)}$$

donde la matriz Hessiana Q es $Q = CC^t$.