RECONOCIMIENTO DE CARACTERES CON REDES NEURONALES

Character Recognition with Neural Networks

MSc. Franz Cuevas Q.
franzcq@gmail.com, fcuevas@umsa.bo

RESUMEN

El reconocimiento automático de caracteres, ya sean de letras o números de documentos impresos es muy fundamental, donde la transcripción llevaría mucho tiempo, si bien existen muchos adelantos como el reconocimiento óptico de caracteres OCR, aun están en la etapa de investigación para eliminar los errores de reconocimiento. El trabajo de investigación fue el estudio minucioso y la aplicación de las redes neuronales artificiales Backpropagation (multicapa) al reconocimiento de las 27 letras y 10 números, donde se valora el tamaño de las letras y se trabaja con el tipo de letra arial 8, el tamaño de las neuronas de entrada y salida es el tamaño del carácter (8 x Nro. Columnas), dos capas ocultas, con 8 neuronas cada una, Los experimentos muestran el entrenamiento o aprendizaje de las letras y números en un promedio de 104 y 85 iteraciones respectivamente, con un error máximo permitido de 0.10.

Palabras clavè:

Aprendizaje supervisado; Redes neuronales Artificiales; Reconocimiento de caracteres, Retropropagación

ABSTRACT

Automatic recognition of characters, either letters or numbers printed documents is very important, where transcription would be time consuming, while there are many advances such as optical character recognition OCR, are still in the research stage to eliminate errors recognition. The research work was the detailed study and application of backpropagation artificial neural networks (multilayer) the recognition of the 27 letters and 10 numbers, which assesses the size of the letters and works with the font arial 8, the size of the input and output neurons is the character size (8 x No. Columns), two hidden layers with 8 neurons each, experiments show the training or learning letters and numbers on an average of 104 and 85 iterations respectively, with a maximum permissible error of 0.10.

keywords:

Supervised learning; artificial neural networks; Character Recognition; Backprogation

INTRODUCCIÓN

En 1888 Ramón y Cajal demostró que el sistema nervioso estaba compuesto por una red de células individuales, las neuronas, ampliamente conectadas entre sí. Pero no solo observó al microscopio los pequeños vacios que separaban unas neuronas de otras, sino que también estableció que la información fluye en la neurona desde las dendritas hacia el axón, atravesando el soma. (Bonifacio Martin, Alfredo Sanz, 2002). Ver figura 1.

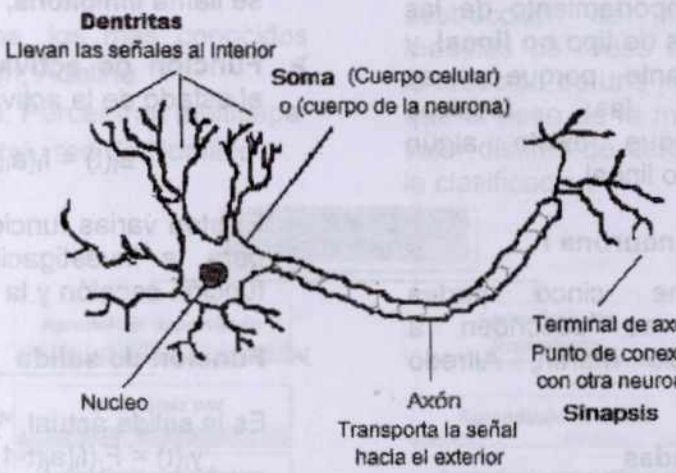


Fig. 1. Estructura general de una neurona biológica

Para una red neuronal artificial RNA existen varias definiciones: Elemento computacional de procesamiento, como unidades similares a las del cerebro humano, que tiene la capacidad de procesar información y aprender de ella. (Hilera 1995). Una forma de computación inspirada en modelos biológicos (Hilera 1995, Freeman 1993, Príncipe 1994). También se puede considerar, como una emulación a través de algoritmos formales de la conducta del cerebro humano, de captar, procesar, analizar, aprender y tomar una decisión (salida) FCQ.

Actualmente aun existen y posiblemente seguirá existiendo instituciones públicas, privadas y personas particulares que tengan documentación impresa y en algunos casos manuscrito que deseen analizar procesar esta información utilizando un ordenador. Dependiendo de la cantidad se puede decidir posiblemente por dos soluciones **a)** La transcripción (funciona cuando es muy poco) **b)** Reconocimiento de manera automática, que involucra el reconocimiento de caracteres con un conjunto de métodos formales y algoritmos. Para este efecto se realizará el análisis de los caracteres (27 letras) y dígitos (10 números) en tamaño y en forma para la aplicación de las redes neuronales artificiales backprogation.

METODOS

ESTRUCTURA DE LOS MODELOS DE RNA

McCulloch y Pitts en 1943 construyeron un modelo abstracto y simple de una neurona artificial, que es una unidad de cálculo que modela el comportamiento de una neurona natural. Este modelo es elemento básico y esencial de procesamiento y construcción de una red neuronal artificial. Ver figura 2. Generalmente el comportamiento de las neuronas biológicas es de tipo **no lineal**, y esto lo hace interesante, porque no se resuelven con las técnicas convencionales, aunque existe algún comportamiento de tipo lineal.

Características de la neurona i

Una neurona tiene cinco partes fundamentales que se describen a continuación (Bonifacio Martin, Alfredo Sanz):

- **Conjunto de entradas**

$$x_j(t)\ (x_1, x_2, x_3, \ldots x_n)$$

- **Conjunto de Pesos,** denominado ***pesos sinápticos*** de la neurona i, denotado por w_{ij} ($w_{i1}, w_{i2}, \ldots w_{in}$) , es la intensidad de interacción entre cada neurona presináptica j y la neurona postsináptica i.

- **Función de propagación** (también se denomina función de excitación). Representa el valor del potencial postsináptico:

$$h_i(t) = \sigma(w_{ij}, x_j(t))$$
$$h_i(t) = \Sigma\, w_{ij}\, x_j = w_i^T \cdot \mathbf{x}$$

Por otro lado, si el peso es positivo, la conexión se llama **excitatoria**, quiere decir, que tendera a excitar a la neurona postsináptica. Si es negativo se llama **inhibitoria,** tendera a inhibirla.

- **Función de activación** Proporciona el estado de la activación actual

$$a_i(t) = f_i(a_i(t\text{-}1), h_i(t))$$

Existen varias funciones de activación, para la investigación se utilizara la función escalón y la función sigmoidal.

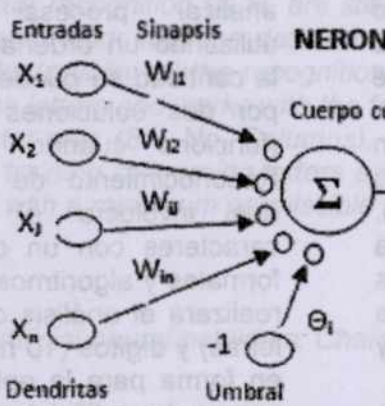
- **Función de salida**

Es la salida actual, $y_i(t) = F_i(a_i(t))$

$$y_i(t) = F_i(f_i[a_i(t\text{-}1), \sigma(w_{ij}, x_j(t))])$$

La función de salida a veces se considera como la identidad F(x) = x, de modo que el estado de activación de la neurona se considera como la propia salida.

$$y_i(t) = F_i(a_i(t)) = a_i(t)$$



Fuente: Bonifacio Martin y Alfredo Sanz

Fig. 2. Modelo de una neurona estándar

CLASIFICACION DE REDES NEURONALES

La clasificación es por el tipo de conexión y el tipo de aprendizaje

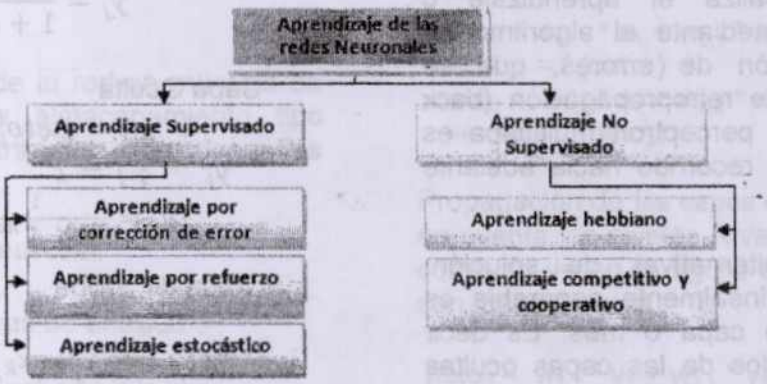
Tipo de conexión

Representa la arquitectura, estructura o patrón de conexiones de una red neuronal, que son tres:

1) **Redes Monocapa**, los más conocidos son el perceptron, Adaline
2) **Redes Multicapa**, Perceptron multicapa
3) **Redes Recurrentes**, red de Hopfield

Tipo de aprendizaje

Según J. Hilera, el **aprendizaje** es el proceso por el cual una red neuronal modifica sus pesos en respuesta a una información de entrada. Los cambios que se producen durante el proceso de aprendizaje se reducen a la destrucción, modificación y creación de conexiones entre las neuronas. En los sistemas biológicos existe una continua creación y destrucción de conexiones. En los modelos de redes neuronales artificiales, la creación de una nueva conexión implica que el peso de la misma pasa a tener un valor distinto de cero. La figura 3 muestra la clasificación



Fuente: Grafico realizado en base a la clasificación de Hilera

Fig. 3. Clasificación por el tipo de aprendizaje

ANALISIS DEL PERCEPTRON EN LA LOGICA BOOLEANA

Linealmente separable

Se utilizan los operadores OR, AND, XOR.

$Función\ de\ Propagación:\quad Neto = \sum_{i=1}^{N} w_i\ x_i$

$$Función\ de\ Activación:\quad y = \begin{cases} 1, & Neto > 0 \\ 0, & Neto \leq 0 \end{cases}$$

Desarrollando para dos valores de entrada:

$$Neto = w_1 * x_1 + w_2 * x_2 = 0$$

Despejando x_2, se tiene una ecuación de una ***recta*** que pasa por el origen.

$$x_2 = -\frac{w_1}{w_2}\ x_1$$

La figura 4, muestra la representación de los valores lógicos OR, AND y XOR y la agrupación de valores de unos (1) y ceros (0): El OR y el AND (con el Umbral) son **linealmente separables**, pero el XOR es imposible.

Y la función de activación cambia a:

$$y = \begin{cases} 1, & Neto > \theta\ \ (\mathbf{umbral}) \\ 0, & Neto \leq \theta\ \ (\mathbf{umbral}) \end{cases}$$

El perceptrón es la única red neuronal que tiene un **teorema de convergencia** que

indica, si el problema es linealmente separable, el perceptron siempre encuentra la solución, pero no se sabe el tiempo que tarda, tampoco se sabe si es optimo la solución.

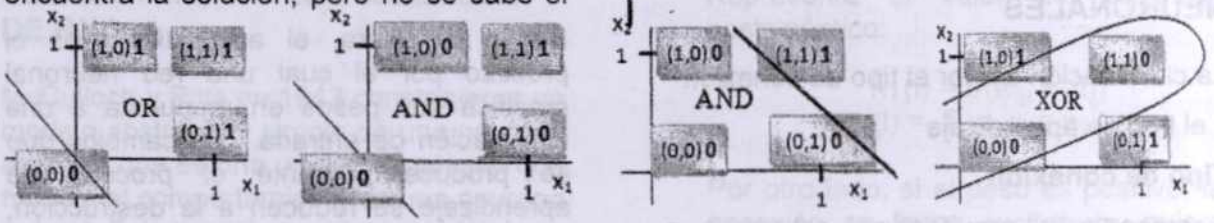


Fig. 4. Separación lineal de OR, AND y XOR

PERCEPTRON MULTICAPA

El perceptron multicapa o multinivel, es un perceptron simple, pero, añadidos con capas ocultas o capas intermedias. Esta arquitectura, realiza el aprendizaje o entrenamiento mediante el algoritmo de retro propagación de errores, que se denomina red de retropropagación (back propagation). El perceptron multicapa es una red de tipo recorrido hacia adelante (feed forward).

Una de las alternativas de solución, cuando no es linealmente separable es incrementar una capa o más. Es decir entrenar los nodos de las capas ocultas pertenecientes a arquitectura multicapa.

Pasos para la creación de una Red Neuronal

En este caso aplicaremos el entrenamiento para la red perceptron multicapa, dividiremos su aplicación en las capa de entrada, capas ocultas y capa de salida.

a) Crear la red neuronal

Determinar número de entradas o patrones de entrada
Determinar las Capas Ocultas
Determinar la salida o patrones de salida.

b) Propagar

Capa de entradas

$$neto_j = \sum_{i=1}^{n} w_{ji} * x_i \; ; j = 1$$

Función sigmoidal

$$y_j = \frac{1}{1 + exp^{-neto_j}}$$

Capa Oculta

$$neto_j^h = \sum_{i=1}^{l} w_{ji} * y_i^{h-1} \; ; j = 2$$

$$y_j = \frac{1}{1+exp^{-neto_j}}$$

Capa de Salida

$$neto_k = \sum_{j=1}^{o} x_{kj} * y_j$$

$$y_j = \frac{1}{1+exp^{-neto_j}}$$

c) Calculo de errores

I. Error Cuadrado o error global

$$e = \frac{1}{2} \sum_{k=1}^{m} (d_k - y_k)^2$$

II. Error Salida

$$e_k = (d_k - y_k) * y_k * (1 - y_k)$$

III. Error Oculto

$$e_{hj} = (\sum_{k=1}^{m} (e_k * w_{kj}) * y_j * (1 - y_j))$$

d) Modificación de pesos W

Modificar pesos, capa de entrada $w_{e(j,i)}$
Modificar pesos de capa oculta $w_{o(j,i)}$
Modificar pesos de capa de salida $w_{s(j,i)}$

Construcción y diseño del Perceptron de Multicapa o Multinivel

Para el aprendizaje es fundamental el cálculo de la variación de los pesos, donde estos valores tienen que ser los óptimos, esta variación se mide por la derivada parcial del error global respecto a los pesos.

$$\Delta w_{ji} = \alpha \frac{\partial e}{\partial w_{ji}}$$

Después de realizar la derivada y los cálculos se tiene:

$$w_{ji}(t+1) = w_{ji}(t) + \alpha (d_j - y_j) y_j^i y_i$$

A esta ecuación se denomina regla de aprendizaje

Para la creación de la red se muestra de manera grafica y almacenamiento tipo matricial de las entradas, salidas deseadas y los pesos.

a) Crear la red neuronal

Número de neuronas de entrada
i=1,2,**3**,... **ne**
Es el de vector de entrada **x [ne]**
Número de neuronas ocultas por cada capa oculta j = 1,**2**,...**no**
Número de capas ocultas k = 1, 2, **3**,...**co**
Número de Neuronas salida m=1,2,**3**,...**ns**
Es el vector de salida deseada **d[ns]**

Asignación de Pesos

La asignación de pesos es por partes (para la capa de entrada, oculta y salida) a través de una generación de números aleatorios en el rango de -1 a 1, $-1 <= w_{ij} <= 1$.

La asignación de pesos es por partes (para la capa de entrada, oculta y salida) a través de una generación de números aleatorios en el rango de -1 a 1, $-1 \leq w_{ij} \leq 1$, como se muestra en la figura 5. Estos valores se cargan a la matriz de entrada, oculta y salida **pesosE[ne][no]**, **pesosO[(co-1) x no][no]** y **pesosS[no][ns]**.

b) Propagar

Propagación de las entradas: Se toma en cuenta las neuronas de entrada del vector **x** y la matriz de **pesosE**

Para **propagar la capa de entrada**, se realiza los siguientes cálculos y se almacena el valor en los primeros nodos de la capa oculta como se muestra en la figura 6.

$$\mathbf{Neto_e = pesosE * x = W * X}$$

y se aplica la función sigmoidal

$neto_1 = w_{1,1} * x_1 + w_{2,1} * x_2 + w_{3,1} * x_3$
$\rightarrow \quad y_1 = f(neto_1)$
$neto_2 = w_{1,2} * x_1 + w_{2,2} * x_2 + w_{3,2} * x_3$
$\rightarrow \quad y_2 = f(neto_2)$

Propagación de **las capas ocultas**, se toma en cuenta los niveles Nivel 2, Nivel 3, etc. como se ve en la figura 5, y los cálculos son los siguientes

$neto_3 = w_{1,1} * y_1 + w_{2,1} * y_2 \rightarrow$
$y_3 = f(neto_3) = Neto_{2,1} = y_{2,1}$ (2 es Nivel 2)
$neto_4 = w_{1,2} * y_1 + w_{2,2} * y_2 \rightarrow$
$y_4 = f(neto_4) = Neto_{2,2} = y_{2,2}$ (2 es Nivel 2)
$neto_5 = w_{3,1} * y_3 + w_{4,1} * y_4 \rightarrow$
$y_5 = f(neto_5) = Neto_{3,1} = y_{3,1}$ (3 es Nivel 3)
$neto_6 = w_{3,2} * y_3 + w_{4,2} * y_4 \rightarrow$
$y_6 = f(neto_6) = Neto_{3,2} = y_{3,3}$ (3 es Nivel 3)

Para propagar la capa de salida, se realiza con la última capa de la capa oculta (ver fig. 6), y los cálculos son los siguientes:

$neto_7 = w_{1,1} * y_5 + w_{2,1} * y_6 \rightarrow \quad y_7 = f(neto_7)$
$neto_8 = w_{1,2} * y_5 + w_{2,2} * y_6 \rightarrow \quad y_8 = f(neto_8)$
$neto_9 = w_{1,3} * y_5 + w_{2,3} * y_6 \rightarrow \quad y_9 = f(neto_9)$

c) Calculo de errores

Error Cuadrado o error global

Para efectos de programación cambiamos los subíndices y_7 a y_1, y_8 a y_2, y_9 a y_3, según muestra la figura 7.

error_global = (½) * ($(d_1 - y_1)^2$ + $(d_2 - y_2)^2$ + $(d_3 - y_3)^2$) ...

Error de salida

$e_1 = (d_1 - y_1) * y_1 * (1 - y_1)$;
$e_2 = (d_2 - y_2) * y_2 * (1 - y_2)$;
$e_3 = (d_3 - y_3) * y_3 * (1 - y_3)$; ...

Error en capa Oculta

Suma = $(e_1 * w_{1,1}) + (e_2 * w_{1,2}) + (e_3 * w_{1,3})$ =
Error $e_{3,1}$ = (Suma * y_5) * $(1 - y_5)$ =
Suma = $(e_1 * w_{2,1}) + (e_2 * w_{2,2}) + (e_3 * w_{2,3})$ =
Error $e_{3,2}$ = (Suma * y_6) * $(1 - y_6)$ =

Continuamos el retroceso en las capas ocultas

$e_{2,1} = (e_{3,1} * w_{3,1}) + (e_{3,2} * w_{3,2})$ = Suma =>
Error $e_{2,1}$ = (Suma * y_3) * $(1 - y_3)$
$e_{2,2} = (e_{3,1} * w_{4,1}) + (e_{3,2} * w_{4,2})$ = Suma =>
Error $e_{2,2}$ = (Suma * y_4) * $(1 - y_4)$

$e_{1,1} = (e_{2,1} * w_{1,1}) + (e_{2,2} * w_{1,2})$ = Suma =>
Error $e_{1,1}$ = (Suma * y_1) * $(1 - y_1)$
$e_{1,2} = (e_{2,1} * w_{2,1}) + (e_{2,2} * w_{2,2})$ = Suma =>
Error $e_{1,2}$ = (Suma * y_2) * $(1 - y_2)$

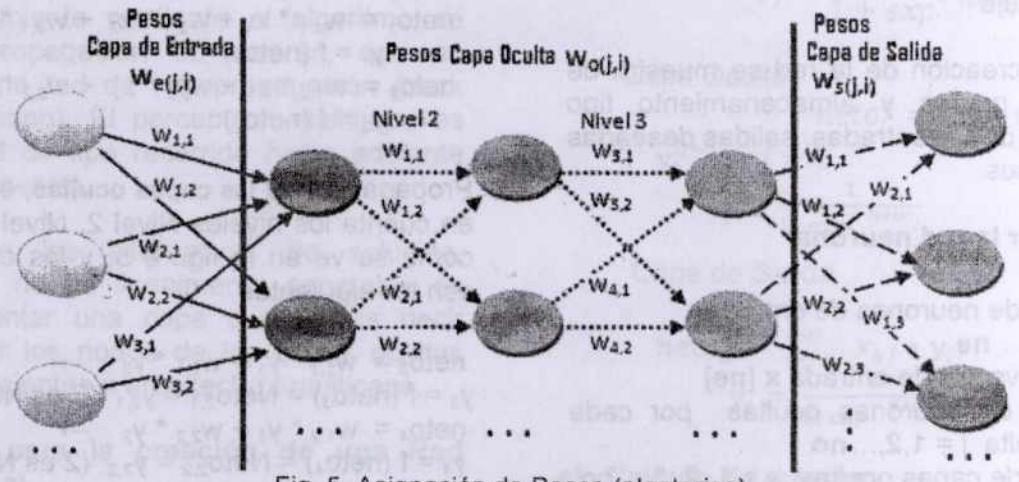


Fig. 5. Asignación de Pesos (aleatorios)

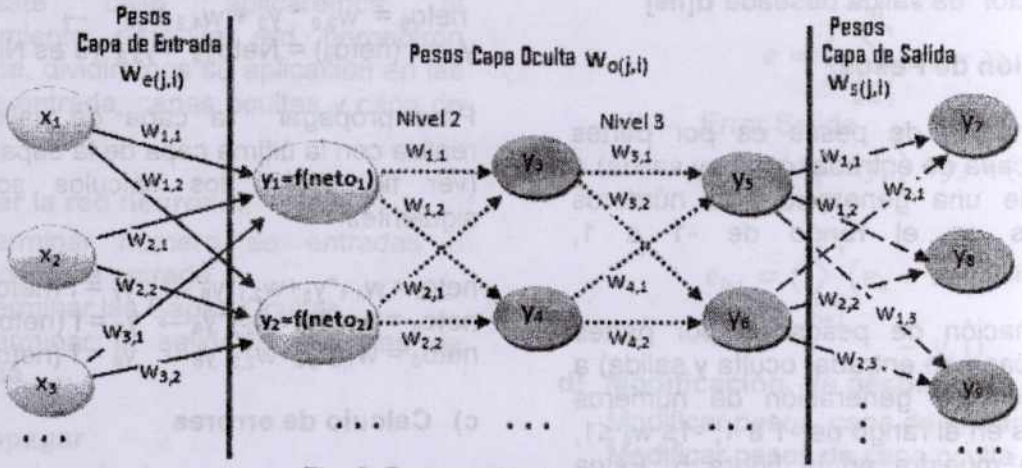


Fig. 6. Propagación hacia adelante

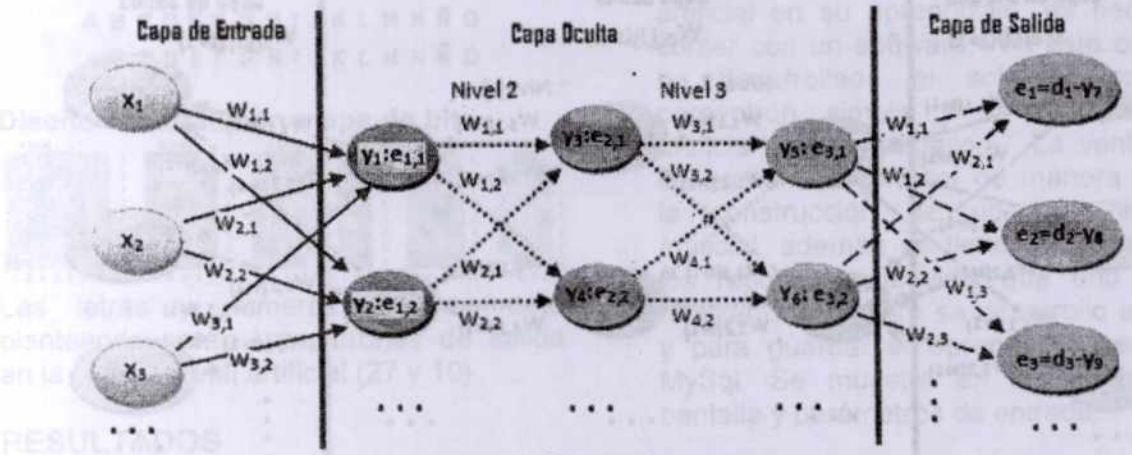


Fig. 7. Cálculo de errores

d) Modificación de pesos W

Modifica Pesos de Entrada

$w_{1,1(t+1)} = w_{1,1(t)}+(alfa*e_{1,1} * x_1)$;
$w_{2,1(t+1)}=w_{2,1(t)}+(alfa*e_{1,1}*x_2)$;
$w_{3,1(t+1)} = w_{3,1(t)} + (alfa\ e_{1,1}*x_3)$
$w_{1,2(t+1)} = w_{1,2(t)}+(alfa*e_{1,2}*x_1)$;
$w_{2,2(t+1)}=w_{2,2(t)}+(alfa * e_{1,2}*x_2)$;
$w_{3,2(t+1)} = w_{3,2(t)}+(alfa*e_{1,2}* x_3)$

Modificar pesos de capa oculta

$w_{1,1(t+1)} = w_{1,1(t)} + (alfa * e_{2,1} * y_1)$;
$w_{2,1(t+1)} = w_{2,1(t)} + (alfa * e_{2,1} * y_2)$
$w_{1,2(t+1)} = w_{1,2(t)} + (alfa * e_{2,2} * y_1)$;
$w_{2,2(t+1)} = w_{2,2(t)} + (alfa * e_{2,2} * y_2)$

$w_{3,1(t+1)} = w_{3,1(t)} + (alfa * e_{3,1} * y_3)$;
$w_{4,1(t+1)} = w_{4,1(t)} + (alfa * e_{3,1} * y_4)$
$w_{3,2(t+1)} = w_{3,2(t)} + (alfa * e_{3,2} * y_3)$;
$w_{4,2(t+1)} = w_{4,2(t)} + (alfa * e_{3,2} * y_4)$

Modificar pesos de capa de Salida

$w_{1,1(t+1)} = w_{1,1(t)} + (alfa * e_1 * y_5)$;
$w_{2,1(t+1)} = w_{2,1(t)} + (alfa * e_1 * y_6)$
$w_{1,2(t+1)} = w_{1,2(t)} + (alfa * e_2 * y_5)$;
$w_{2,2(t+1)} = w_{2,2(t)} + (alfa * e_2 * y_6)$
$w_{1,3(t+1)} = w_{1,3(t)} + (alfa * e_3 * y_5)$;
$w_{2,3(t+1)} = w_{2,3(t)} + (alfa * e_3 * y_6)$

Ver figura 8. Se repite el algoritmo hasta encontrar los mejores pesos para el entrenamiento o aprendizaje de la red neuronal artificial.

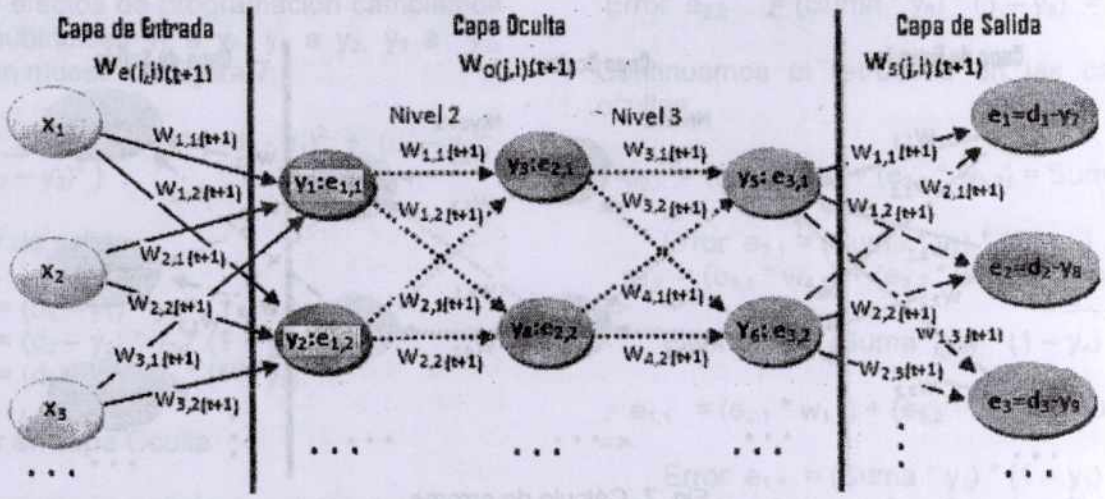


Fig. 8. Modificación de Pesos en t+1

MODELOS DE LETRAS y DIGITOS

Una letra es un signo grafico de un sistema de escritura, la cual sirve para la comunicación, en este caso para el estudio nos basamos en las 27 letras del alfabeto español y los dígitos del 0 al 9. Las formas de representación son infinitas a parte de las minúsculas y mayúsculas.

En los procesadores de textos actualmente ya tienen clasificados por tipo de letra y tamaño, y tienen sus respectivas funcionalidades como búsquedas, sorteos, etc.

Pero uno de los problemas que existe aun es en la identificación si la letra es de tipo grafico, aunque fuera de tipo imprenta, y es más complejo si es manuscrito. Actualmente existen impresoras que reconocen las letras con cierto grado de error, pero el "como se hace" es desconocido o no es de código abierto.

Análisis de las letras tipo grafico
Arial 8

A B C D E F G H I J K L M N Ñ O P Q R S T U V W)
Arial 8 con zoom 800

Si analizamos de manera un poco detallada, se ven manchas y colores diferentes en degrade del color negro, esto complica, si se lleva a cabo el mapeo con bits, se tiene dos alternativas

a) Tomar en cuenta el degrade mas, para colocar en una matriz booleana

b) Solamente tomar en cuenta la parte más negrita. Para colocar en una matriz booleana

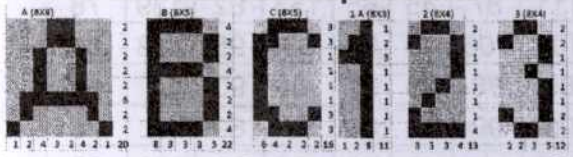
Ejemplo de a)

Ejemplo de b)

Con una vista normal sin zoom se tiene el original y las dos modificaciones

A B C D E F G H I J K L M N Ñ O
A B C D E F G H I J K L M N Ñ O
A B C D E F G H I J K L M N Ñ O

Diseño de letras con mapa de bits



Las letras y números que se está planteando serán los patrones de salida en la red neuronal artificial (27 y 10).

RESULTADOS

Software desarrollado

Para ver los resultados de la red neuronal artificial en su aprendizaje, es necesario contar con un software, en este caso se ha desarrollado el software para el perceptron simple y la red neuronal artificial Backpropagation. La ventaja de desarrollar es conocer de manera detalla la construcción de un red neuronal artificial, además se tiene el control para los reportes y análisis que uno pueda realizar. El software se desarrollo en PHP y para guardar el aprendizaje se utilizo MySql. Se muestra en las figura 9 la pantalla y parámetros de entrada.

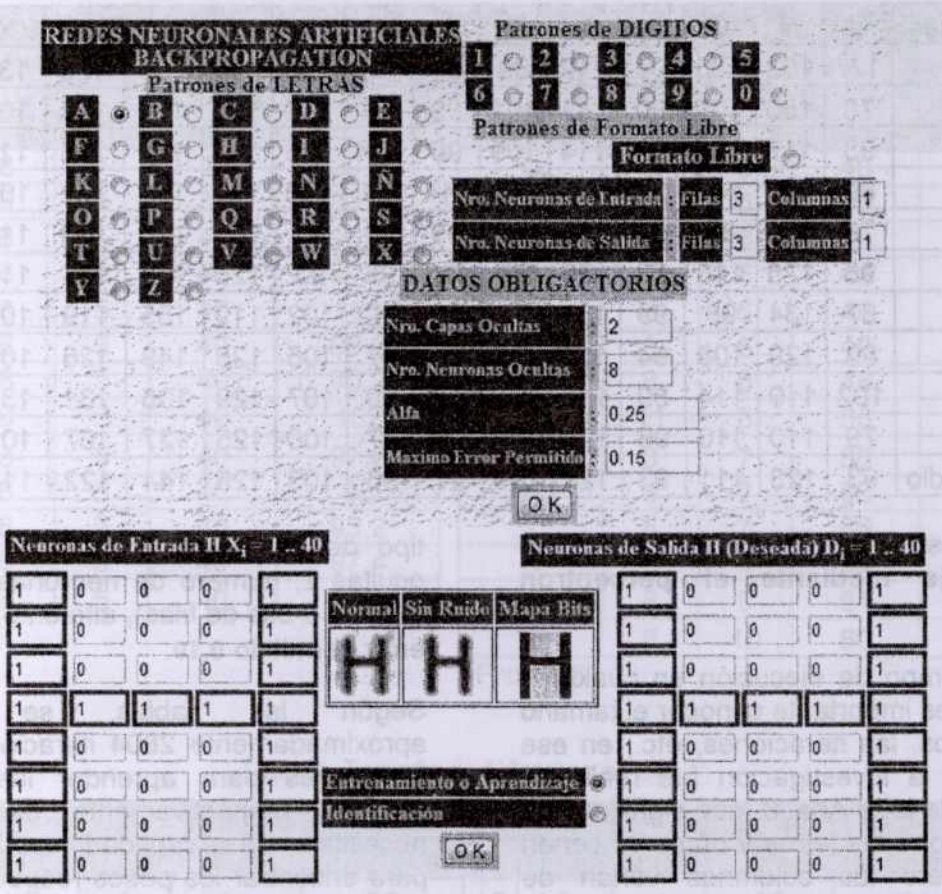


Fig. 9 Parámetros de entrada de la RNA Backpropagation

Tabla 1. Número de iteraciones para el aprendizaje o entrenamiento de letras

Corridas	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	126	87	95	93	130	107	96	131	34	91	115	132	99	127
2	132	108	104	106	107	88	111	87	45	65	153	82	139	78
3	126	88	100	109	89	75	105	106	45	83	99	95	113	93
4	131	97	79	85	87	88	89	86	33	91	100	90	87	89
5	140	102	107	111	86	93	96	99	39	86	97	89	96	78
6	127	95	90	110	110	99	115	115	40	76	128	88	120	98
7	130	100	102	82	98	100	94	97	41	72	110	90	127	87
8	141	90	88	107	92	79	98	109	43	90	112	108	92	99
9	131	104	93	102	109	76	94	97	37	88	105	115	100	102
10	126	94	105	100	90	97	96	91	35	90	109	87	95	79
Promedio	131	97	96.3	101	100	90	99.4	102	39	83	113	98	107	93

Corridas	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	127	102	130	97	112	89	83	86	108	119	120	128	139	91
2	78	150	100	90	141	101	99	99	100	128	158	118	104	116
3	93	117	100	87	114	109	96	124	106	119	157	136	111	100
4	89	100	143	97	138	96	114	80	108	117	159	122	103	102
5	78	148	85	95	115	107	95	92	91	141	128	106	115	102
6	98	125	118	88	128	93	112	95	93	126	156	127	117	92
7	87	134	98	99	131	97	89	112	102	119	155	119	109	115
8	99	129	108	94	119	108	86	97	105	125	149	126	106	113
9	102	119	114	89	115	109	101	90	107	129	135	131	137	98
10	79	110	110	96	127	95	99	87	106	125	127	107	103	93
Promedio	93	123	111	93	124	100	97	96	103	125	144	122	114	102

Resultados de corridas en el aprendizaje mediante el perceptron multicapa

Para el tiempo de ejecución en cualquier programa es importante conocer el tamaño de los datos, las iteraciones, etc., en ese entendido, la investigación fue realizada en el tipo de letra Arial 8, que significa que en su mayoría las letras y números tienen 8 filas, pero las columnas varían de acuerdo a la forma de la letra y número (ver tabla 1 y 2), los resultados son bajo las siguientes características, neuronas de entrada y salida son determinados por el tipo de letra arial 8, numero de capas ocultas 2, numero de neuronas ocultas 8 (por los 8 bits de filas), alfa 0.25, y máximo error permitido **0.10**.

Según las tablas, se requiere aproximadamente 2804 iteraciones y 853 iteraciones para aprender las letras y números respectivamente, es decir, se necesitan esta cantidad de iteraciones para encontrar los pesos respectivos para cada capa, talque estos sean los mejores valores.

Tabla 2. Número de iteraciones para el aprendizaje o entrenamiento para Números

Corridas	1	2	3	4	5	6	7	8	9	0
1	79	80	79	95	89	74	94	75	97	77
2	75	83	73	89	91	78	70	91	112	88
3	88	85	94	96	102	87	77	78	99	78
4	70	69	83	88	106	94	77	68	73	79
5	62	118	74	106	75	92	87	119	69	80
6	74	117	77	104	99	89	89	90	75	80
7	80	79	85	93	97	83	91	104	100	79
8	79	77	83	100	88	91	72	90	101	74
9	71	75	73	88	90	77	80	77	75	87
10	70	87	85	89	89	85	73	70	99	82
Promedio	74.8	87	80.6	94.8	93	85	81	86.2	90	80

Tabla 3. Tamaño de letras en Bits

	N.Fila	N.Col	Tamaño Bits	Promedio Iteraciones
I	8	1	8	39.2
J	8	4	32	83.2
B	8	5	40	96.5
C	8	5	40	96.3
D	8	5	40	101
E	8	5	40	99.8
F	8	5	40	90.2
H	8	5	40	102
L	8	5	40	97.6
N	8	5	40	93
P	8	5	40	93.2
S	8	5	40	97.4
T	8	5	40	96.2
U	8	5	40	103
Z	8	5	40	102
G	8	6	48	99.4
K	8	6	48	113
M	8	6	48	107
O	8	6	48	111
R	8	6	48	100
Ñ	11	5	55	123
Q	8	7	56	124
X	8	7	56	122
Y	8	7	56	114
A	8	8	64	131
V	8	8	64	125
W	8	10	80	144
Promedio				104

Tabla 4. Tamaño de Números en Bits

	N.Fila	N.Col	Tamaño Bits	Promedio Iteraciones
1	8	3	24	75
2	8	4	32	87
3	8	4	32	81
5	8	4	32	93
6	8	4	32	85
7	8	4	32	81
8	8	4	32	86
9	8	4	32	90
0	8	4	32	80
4	8	5	40	95
				85

Tabla 5. Reconocimiento de letras a través de Backpropagation

Letra Original	Letra a Comparar		Primera Iteración	Ultima Iteración
O	Q	Numero de Iteraciones	%	%
		0	5.0000	
		1	5.0286	
		2	5.0426	5.0066
Q	O	3	5.1039	5.0292
		0	5.0000	
		1	5.0029	
E	F	6	5.2727	5.0424
		13	5.6734	5.0316
		5	5.2338	5.0407
F	E	0	5.0000	
		3	5.0892	5.0264
		4	5.1288	5.0159
P	R	0	5.0000	
		10	5.3732	5.0193
		11	5.5756	5.0150
		3	5.0903	5.0027
B	C	16	5.9042	5.0022
		19	6.0900	5.0316
		24	7.0974	5.0463

Las tablas Tabla 3 y Tabla 4 (ordenado por tamaño) también muestran que a mayor tamaño de letras y números incrementa las iteraciones.

Experimento de reconocimiento de letras

La tabla 5 muestra el experimento de comparación de aprendizaje o entrenamiento con la red neuronal backpropagation, bajo las características similares a la tabla 1, pero, con la variación que el máximo error permitido es **0.05**.

En esta tabla se muestra particularmente, la comparación de caracteres similares, la letra O es parecida a la letra Q, y la letra Q es parecida a la letra O (según el mapa de bits de estas letras la variación es solo en 1 bit). La letra B no es similar en aproximadamente en un 6% o 7%, pero, puede, reacomodarse en un promedio de 20 iteraciones. Para el análisis se debe tomar en cuenta solamente la primera iteración.

DISCUSION

En la red neuronal de Perceptron básico, y llevado al experimento mediante la ejecución del software construido para tal efecto, se muestra una convergencia muy rápida para encontrar los pesos máximos para el aprendizaje de los Patrones del OR, para los patrones del AND, no tiene solución, pero cuando se aumenta el umbral, que permite la translación, tiene la

solución respectiva para el aprendizaje, mientras para el XOR, definitivamente no tiene solución. Por lo tanto, se puede indicar lo que mostraba de manera teórica, también se cumple de manera experimental, donde el Perceptron básico resuelve problemas netamente que **sean linealmente separables**. En la red neuronal multicapa o Backpropagation permite la solución a los anteriores problemas, ya que permite introducir las capas ocultas, pero de acuerdo a la cantidad de las neuronas de entrada, neuronas ocultas y numero de neuronas de salida puede tardar la ejecución, en el experimento, con el máximo error permitido, de 0.10 se requieren 2804 iteraciones para las letras y 853 iteraciones para los números, pero si se quiere el máximo error permitido de 0.05, se necesita mayor tiempo de ejecución y el numero de iteraciones aproximadamente se incrementa en un 75 %, es decir con este error se requiere 4907 iteraciones para aprender todas las letras y 1492 para entrenar a los números.

CONCLUSIONES

La explicación teórica de una red neuronal artificial, es muy importante, pero, llegar a comprender, el entrenamiento o el aprendizaje de esta red, es fundamental, para tal efecto se desarrollo el programa perceptron simple en PHP., donde permitió el entrenamiento de los valores de la lógica booleana OR, AND y XOR, coincidiendo al 100% con la parte teórica de que la resolución en el perceptron simple es en aquellos valores que **pueden ser linealmente separables**.

Para el diseño de la red neuronal artificial que identifique las letras y los números, se considero el tipo de letra arial 8, el número de las neuronas de entrada y las neuronas de salida se determina por el número de filas y columnas que pueda tener cada letra, en este caso el numero de filas es 8 (por el tamaño arial 8). Varía desde la letra I que tiene 8 bits (8 x 1) hasta W que es la letra más grande en términos de bits 8 x 10 = 80 neuronas, y el 48% es de tamaño 8 x 5 = 40 y el resto está de 48 a 64 bits.

En los números el más pequeño es el numero 1 de 8 x 3 = 24 bits, el más grande el numero 4 de 8 x 5 = 40 bits, y el resto de los números es de 8 x 4 = 32 bits o neuronas.

En el diseño de capas ocultas, se debía resolver con una sola capa, o aplicar la regla de la pirámide geométrica (numero de neuronas de entrada * numero de neuronas de salida y a este resultado sacar la raíz cuadrada), porque si es mayor a 2, el entrenamiento se hace más lento y la capa adicional a través de la cual se propaga el error hace el gradiente más inestable, y el numero de mínimos locales se incrementa.
Para el entrenamiento se utilizo dos capas ocultas, cada capa oculta tiene 8 neuronas por el tamaño de la letra (numero de filas), donde la red mostro una estabilidad alrededor del promedio de iteraciones o entrenamientos.

PROYECCCIONES

Las redes neuronales artificiales siguen en vigencia su estudio, y como base es la red neuronal backpropagation, si bien tiene avances el reconocimiento óptico de caracteres OCR, que es la conversión escaneada de texto mecanografiado, manuscrito en un texto codificado. Aun plantea problemas al escanear y reconocer estos caracteres, más aun si es manuscrito, por la misma complejidad y la velocidad de ejecución que pueda tener la red neuronal artificial. Es importante seguir profundizando tanto en la parte teórica que aporta la convergencia para tener pesos aceptables, y en la parte practica o programación experimentar para reducir las iteraciones y tener un aprendizaje o entrenamiento más rápido.

BIBLIOGRAFIA

Constela G., Aguilera S. (2010). *Aplicación De Redes Neuronales En El Reconocimiento De Caracteres Ocr De Software Libre, Tecnologia Informática-Universidad De Belgrano.* Buenos Aires: Workshop De Investigadores En Ciencias De La Computación.

Hilera J., Martinez V. (2002). *Redes Neuronales Artificiales, Fundamentos, Modelos Y Aplicaciones* (Vol. 2da Ed). España: Sddison Wesley Iberoamiercana S.A.

Howard, A. (1986). *Calculo Y Geometria Analitica* (Vol. 2). Mexico: Limusa S.A.

Martin B., Sanz A. . (2002). Mexico D.F: Alfaomega Ra-Ma.

N. J., N. (2001). *Inteligencia Artificial: Una Nueva Sintesis.* España: Mcgraw Hill Interamericana.

Prawda, J. (1981). *Metodos Y Modelos De Investigacion De Operaciones* (Vol. 1). Mexico D.F: Limusa S.A.

Russell S., Norving P. (1996). *Inteligencia Artificial: Un Enfoque Moderno.* Mexico: Prentice Hall Hispanoamericana S.A.