OPTIMIZACION DE UN RECONOCEDOR DE TEXTO MANUSCRITO POR MEDIO DE MAPAS AUTOORGANIZADOS DE KOHONEN (RNA's)

RESUMEN
Este texto describe una aplicación más de las redes neuronales artificiales y su gran capacidad para la memorización de patrones. En esta ocasión se implementa una estructura neuronal de Kohonen, que es entrenada para que pueda identificar cada una de las letras del abecedario (mayúsculo), las cuales son suministradas de pulso y letra del propio usuario por medio de un periférico de entrada como lápiz óptico o mouse. Las aplicaciones fueron programadas en C++ utilizando la librería Allegro.

PALABRAS CLAVES: Redes Neuronales, Kohonen, C++, reconocimiento de patrones.

JUAN DIEGO GOMEZ V.
Estudiante Ingeniería de Sistemas
Universidad Tecnológica de Pereira.
juanogo@utp.edu.co
juanogo@gmail.com

ABSTRACT
This text describes an application more than the nets artificial neuronals and its great capacity for the memorization of patterns. In this occasion a structure neuronal of Kohonen is implemented that it is trained so that each one of the letters of the alphabet can identify (uppercase), which are given of pulse and the own user's letter by means of an entrance peripheral as optic pencil or mouse. The applications were programmed in C++ using the bookstore Allegro.

KEY WORDS: *Neuronals Networks, Kohonen, C++, recognition of patterns*

1 INTRODUCCION

En la extensa bibliografía sobre redes neuronales se ataca el problema del reconocimiento de letras como ejemplo en varias secciones, y por lo regular se implementa una red neuronal tipo Hopfield. A diferencia de dichas aplicaciones, la que concierne a este texto, busca un grado mucho mayor de generalización en el ámbito de la caligrafía humana.

En verdad el problema de reconocer patrones de texto por medio de un sistema inteligente definido de manera un poco mas técnica, consiste en poseer un patrón cualquiera T, del cual esperamos que el sistema no le identifique como el patrón X debido a que T es mas parecido o tiene más características de X que de cualquier otro patrón conocido.

He aquí el motivo por el cual una red Hopfield no realiza una impecable labor de reconocimiento de patrones para nuestro propósito específico. Pues este sistema neuronal solucionará el problema planteado anteriormente de reconocer al patrón X, cuando se le muestra el patrón T si y sólo si el patrón T es igual al patrón X, pero con un tanto de ruido o distorsión; con las palabras ruido o distorsión se quiere decir específicamente que el patrón T no es más que el mismo X alterado sea por que le falte una parte, porque se le agregue otra que no tenia o ambas a la vez , sin que sobre mencionar que dicha alteración debe ser relativamente pequeña (máximo 25%). Debido a esto la red no tendrá un buen desempeño si el patrón T dejase de ser X con ruido para convertirse en uno no necesariamente igual sino simplemente algo "semejante" a X o con algunas (muchas o pocas) características de X.

Si lo que se desea en realidad es que un agente inteligente aprenda a reconocer un patrón, tenderemos a adaptar dicho agente al mundo real, por medio de la práctica. Y ya que en realidad no todas las personas escriben la misma letra *a* con diferentes ruidos, si no que basados en un prototipo usan infinitos modelos distintos de esta, lo que se espera es que nuestro sistema busque vínculos, características o relaciones, no entre un prototipo y él mismo ruidoso, sino entre un arquetipo y una amplia gama de modelos representativos de este. Con esto lograremos que el sistema adquiera la capacidad de emitir un juicio o conclusión luego de una estimulación u observación, imitando así la reacción del cerebro humano cuando se ve enfrentado al problema de leer un texto manuscrito que contiene una letra *y* que se parece a una *v* o que quizás pudiese ser una *r*; en realidad lo que hará el cerebro para salir de este dilema es buscar vínculos o características que le ayuden a definir ese carácter como alguno de los posibles candidatos; en realidad al que mejor se le ajuste, sin olvidar el margen de error en que se puede incurrir, tanto en el modelo biológico (cerebro), como en el modelo artificial (kohonen).

Fecha de recepción: 27 Enero de 2005
Fecha de aceptación: 01 Abril de 2005

Si nos detenemos un momento a analizar la técnica de aprendizaje de un mapa de Kohonen observaremos claramente que éste lo que hace es un acomodamiento o agrupamiento de un grupo de patrones ya vectorialmente adaptados según lo que pueda distar uno del otro, así pues el patrón T, pertenecerá al conjunto de patrones *S*, si la distancia entre T y un patrón X es relativamente pequeña, pero además este requisito lo deberán haber cumplido todos los patrones que pertenezcan a *S,* es decir *S* es el grupo de todos los patrones que tienen una distancia corta entre cada uno de ellos y X, en donde X es el patrón arquetipo de *S* y por lo tanto es el patrón que queremos que el sistema nos identifique cada vez que le mostremos un patrón cualquiera que pertenezca a *S;* y adaptando lo anteriormente dicho a nuestro problema es apenas lógico pensar que podemos entrenar un mapa de Kohonen para que construya un conjunto de distintos estilos de letras que tengan una distancia corta o que se parezcan mucho por decir algo a la letra *A* así cada vez que le mostremos al mapa, alguna letra de las pertenecientes al conjunto, este nos la identificará como la *A,* y así sucesivamente le entrenaremos para que forme un grupo para la letra *B, C, D*... Pero la verdadera ventaja en realidad es que en algún momento le podremos mostrar a la red un patrón que nunca ha visto en entrenamiento y esta sin ninguna dificultad haciendo algunos cálculos de medidas de distancias entre el patrón y todos los arquetipos, le adjuntara al grupo que menos diste, así en la practica si le mostramos una letra este la identificara como una letra *E,* si y sólo si dicha letra en verdad se parece a una *E,* lo que quiere decir también, sólo si la distancia vectorial entre ella y el vector que representa la letra *E*, es muy corta. Esta conclusión además nos indica que la red nunca nos llevará a un estado espurio o inconcluso [2] como si lo hace la red Hopfield, ya que esta red siempre relacionará a cualquier patrón que vea con el que más se le parezca. Y con todo esto dicho ya es claro que la red de Kohonen será muy eficiente a la hora de reconocer un carácter; sólo bastaría con vectorizar cada una de las letras del abecedario y entrenar dicha red.

2 PROCESO DE VECTORIZACION Y DEFINICION DE PATRONES

Nuestro problema consiste, como ya lo hemos mencionado, en reconocer caracteres escritos por medio de una red neuronal y para poder solucionarlo debemos hallar su modelo en equivalente matemático pero aun más que matemático debemos modelarlo en "lenguaje" de red, pues ya debe ser claro para nosotros que una RNA solo puede extraer del mundo exterior cosas representadas numéricamente en forma vectorial, y esto nos obliga a realizar la equivalencia entre una letra del abecedario y un vector n-dimensional, para ello debemos recurrir a un proceso de pixelado en el cual representaremos una letra en una matriz de S filas por K columnas, la que después de creada procesaremos no como matriz sino como el vector que resulta después de unir todas y cada una de las S filas de la matriz representativa en una sola, y así obtendremos un vector compuesto de S veces K elementos, lo que nos indica que el tamaño de dicho vector seria de S**x**K=n, para hablar en términos de un vector n-dimensional y definir entonces a n como el numero que nos determina la cantidad o la calidad del pixelado que aplicamos a dicha letra, pues si n es muy grande la letra estará en un nivel de pixelamiento en el que se le podrán definir de buena manera las curvas y los picos que posea la letra, pero si n es pequeño la letra tomará un estilo no tan real sino un poco mas digital. En nuestro caso particular se definió una matriz de 11 filas por 7 columnas con un n=77 que no es muy grande pero que nos representa una letra escrita como vemos en la figura No 1, medianamente bien. La matriz representativa será binaria y el valor 0 nos indicara un espacio blanco, pero el valor de 1 representará que en ese espacio va parte del trazo de la letra y después de transformarla en vector lógicamente binario también dicha letra estará ya traducida neuronalmente hablando, a un patrón que estará listo para ser presentado, memorizado y clasificado por nuestro mapa de Kohonen.

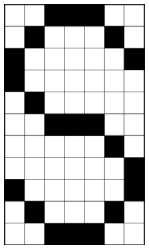

Fig. 1. letra S pixelada en una matriz de 11x7

0	0	**1**	**1**	**1**	0	0
0	**1**	0	0	0	**1**	0
1	0	0	0	0	0	**1**
1	0	0	0	0	0	0
0	**1**	0	0	0	0	0
0	0	**1**	**1**	**1**	0	0
0	0	0	0	0	**1**	0
0	0	0	0	0	0	**1**
1	0	0	0	0	0	**1**
0	**1**	0	0	0	**1**	0
0	0	**1**	**1**	**1**	0	0

Fig.2. Matriz binaria representativa de la S

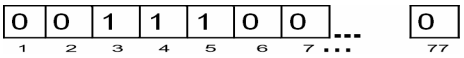

Fig. 3. Patrón neuronal de la letra S

3 ARQUITECTURA DE LA RED

El ya renombrado modelo de Kohonen que estamos implementando fue extractado conceptualmente de [1] y adaptado a nuestros requerimientos de la manera que describiremos en la secuencia del texto.

Apegándonos al modelo común de un mapa autoorganizado o un sofm [1], se construyó esta red compuesta de dos capas o pisos de neuronas la primera de ellas que es la capa receptora es una capa unidimensional que se representa por medio de un vector de neuronas de tamaño igual al de los patrones representativos de las letras (77), puesto que cada neurona de esta capa está encargada de recibir e introducir a la red neuronal cada una de las componentes del patrón que estemos entregando ya sea un 0 o un 1; por ejemplo la neurona 1 recibe la componente 1 del patrón abreviadamente representado en la figura No 3 y

así sucesivamente. La segunda capa de neuronas es una capa bidimensional de forma matricial y esta capa se compone de tantas neuronas como letras se desee que aprenda a reconocer la red, estas neuronas se distribuirán en tantas filas y tantas columnas como se quiera; la cantidad de neuronas que se aplicaron en la segunda capa para nuestros fines fue de 27, que son la cantidad de letras del abecedario y por lo tanto la cantidad de patrones que la red debe aprender, la distribución de esta capa en filas y columnas particularmente fue de 1x27 y esto tiene una razón de ser que será explicada más adelante, por el momento es necesario observar que finalmente nuestra segunda capa de red regularmente bidimensional, se convirtió al igual que la primera capa en un vector unidimensional, debido a la hasta ahora inexplicada elección de 1 fila y 27 columnas para ella.

Es en nuestra segunda capa donde finalmente se arrojará una respuesta, proceso que describiremos más adelante, cabe resaltar aquí por el momento que para que se de una salida en la red todas y cada una de las neuronas de la segunda capa deben tener una conexión con todas y cada una de las neuronas de la primera capa, por lo tanto todas y cada una de las neuronas de la segunda capa deben poseer un total de 77 conexiones las cuales entran todas a cada una de ellas y salen de cada una de las 77 neuronas de la capa de entrada, cada conexión es representada por un numero w y el vector que resulta de todas las conexiones que tiene una neurona i de la segunda capa, es un vector de 77 componentes que se denomina como Wi, el vector de pesos asociado con la i-esima neurona de la capa de salida, finalmente cada neurona de las 27 de la segunda capa debe tener asociado un vector Wi (i=1..27) que es el que representa componente a componente los valores actuales de la conexión entre ella y las cada una de las 77 neuronas de entrada.

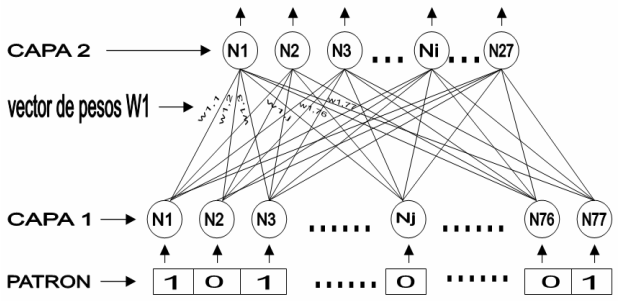

Fig. 4. Arquitectura Sofm para reconocer texto

4 FASE DE ENTRENAMIENTO

El primer componente del entrenamiento de cualquier red neuronal es un buen set de entrenamiento que consta de todos los patrones que servirán como ejemplo a la red para que esta aprenda bien su tarea, nuestro set esta compuesto por 135 patrones de los cuales cada 5 representan cada una de las 27 letras por ejemplo los patrones del 1 al 5 son todos letras A pero de diferentes estilos cada una, del 6 al 10 son la letra B y así sucesivamente.

Seguido de esto, cada uno de estos patrones será introducido a la red por medio de la capa 1, la red tomará la distancia vectorial entre ese patrón y cada uno de los 27 vectores de pesos, aquella neurona que obtenga la distancia o la diferencia más pequeña entre su vector de pesos y el patrón en mención, será la neurona ganadora y por ese hecho sólo a ella se le modificará su vector de pesos asociado de la siguiente manera:

$$\mathbf{Wg(t+1) = Wg(t) + \alpha . V(t) . (P - Wg(t))} \quad (1)$$

En donde Wg es el vector de pesos de la neurona ganadora, P es el patrón en cuestión, α es una constante de aprendizaje o rata de cambio y V es la función de vecindad que en nuestro trabajo fue considerada nula ya que este concepto de vecindad [2], no fue puesto en practica acentuando así que es sólo la neurona ganadora a quien le mejoramos los pesos, motivo por el cual queda aclarado el porque de la unídimensionalidad en la segunda capa, pues ya que si la red no tiene un rango de vecindad no hay necesidad de distribuir la segunda capa de manera matricial. Y así pues (1) quedaría transformada de la siguiente manera, haciendo V(t)=1 en cualquier t:

$$\mathbf{Wg(t+1) = Wg(t) + \alpha . (P - Wg(t))} \quad (2)$$

Se aplicará pues (2) para todos y cada uno de los patrones del set (135 en total) y después de pasados todos los patrones, este proceso se repetirá nuevamente 500 veces.

Finalmente lo que se está logrando es que cuando un patrón entra, la neurona que se activa o neurona ganadora, es la que se parece más a él; y si seguimos el orden mencionado de 5 en 5 de A hasta Z, cuando un patrón entre a la red y se active la neurona 1 de la segunda capa esto querrá decir que ese patrón es una A y si la que se activa es la 2, el patrón es una B y así sucesivamente. Luego en el proceso de ejecución cuando la red termine el entrenamiento de 500 iteraciones, podremos mostrarle a ella otro patrón diferente de los 135 del set, o sea uno que ella nunca ha visto, esperaremos pues a ver cual de las neuronas se activa y esa me representará la letra a la que corresponde ese patrón, por ejemplo si se activa la neurona 4, esto nos indicara que ese patrón que nunca había visto la red es una D. No sobra mencionar que a esta altura la red ya no esta siendo entrenada si no probada y por lo tanto ya los pesos no se modifican mas para ninguna neurona de la red.

Finalmente hay que resaltar la cantidad de ratas de cambio [2], que se usaron incluso algunas en función del

tiempo como lo plantea [1], pero fue finalmente una rata constante (α=0.2) la que nos arrojó el mejor desempeño en la red.

5 DISEÑO DE SOFTWARE

Como mencionamos antes todo el software fue diseñado en C++ utilizando Allegro. Este se compone básicamente de tres aplicaciones: un Capturador, un Entrenador y un Simulador.

El Capturador, como su nombre lo dice, es el que atrapa mediante mouse o lápiz óptico los patrones tanto del set de entrenamiento como los de prueba, en esta aplicación se muestra en pantalla un bitmap de 55x35 que es sobre el que escribe su letra el usuario mediante el mouse, este además representa una matriz tipo bolean de 11x7 cuyas componentes serán true si el usuario hizo clic sobre el espacio del bitmap que representa ese espacio de memoria en pantalla, y ese mismo cambiara de color indicando que se acabó de dibujar allí un pedazo de trazo de la letra que finalmente se esta dibujando; este proceso se repite tantas veces como patrones se quieren capturar, después cada matriz que representa a un patrón capturado la transformamos en un vector binario de 77 componentes mediante una sentencia for, y después este será archivado en un fichero llamado PATRONES.FIL.

El entrenador lo que hace es abrir y descargar el fichero PATRONES.FIL que en verdad contiene los patrones con que vamos a entrenar nuestra red. Luego toma cada vector descargado y le hace un sondeo por las funciones void mide_distancia(), que cuantifica la distancia entre el patrón y todos los vectores de pesos W, la función void ganadora(), verifica cual neurona se activó, y la funcion void actualiza(), que modifica los pesos de la ganadora mediante (2), hay que aclarar que en t=0; todos los vectores de pesos son inicializados al azar desde un archivo nombrado como PESOS.FIL y para t =500 fin del entrenamiento la aplicación entrenador, guarda todos los pesos finales en un fichero llamado PESOSFIN.FIL.

Finalmente el simulador descarga del fichero PESOSFIN.FIL los vectores W que resultaron del entrenamiento, luego llama al capturador para recibir un patrón del usuario y compararlo con los pesos descargados mediante la función void mide_distancia(), y finalmente emite en la parte inferior de la pantalla el resultado del la neurona ganadora pero su equivalente en letra.

A continuación se presentará el código fuente de las principales funciones de las aplicaciones capaturador, entrenador y simulador como una ayuda básica para cualquier prueba o mejoramiento que quiera realizar el lector:

```
Void pinta() //Esta función pinta y captura  en una
  //matriz la letra que el usuario dibuja mediante el Mouse
{
bool ban=true;int x,y;
show_mouse(screen);

while (ban)
{
 x=mouse_x;   //recoge coordenada x del Mouse
 y=mouse_y;  //recoge coordenada y del Mouse

if((x<135)&&(y<155)&&(mouse_b&1)&&(x>100)&&(y>100)
  { //si dan click izq pinta un trozo de letra y carga  matriz en 1
   scare_mouse();
rectfill(screen,x/5*5,y/5*5,x/5*5+5,y/5*5+5,makecol(25,5,4));
   mat[(y-100)/5][(x-100)/5]=1;
   show_mouse(screen);
  }

if((x<135)&&(y<155)&&(mouse_b&2)&&(x>100)&&(y>100)
  { //si dan click der borra un trozo de letra y carga  matriz en0
   scare_mouse();
rectfill(screen,x/5*5,y/5*5,x/5*5+5,y/5*5+5,makecol(25,2,0));
   mat[(y-100)/5][(x-100)/5]=0;
   show_mouse(screen);
  }

struct NEURONA  // la estructura de una  neurona
{
float w[77];   //su vector de pesos
bool activada;  //se activa cuando es la ganadora
float distancia;  // guarda la distancia entre ella y un patrón
} mapa[27];//se declara un vector de 27 componentes con esta
//estructura que representa la capa 2 de la red

//R=77

void distancia()  //mide la distancia entre un patrón propagado
  //y las 27 neuronas de la capa 2
{ int i;
  for( i=0;i<27;i++)
  {
   mapa[i].distancia=0;
   for(int r=0;r<R;r++)
     mapa[i].distancia=mapa[i].distancia+(mapa[i].w[r]-
propagado[r]) *(mapa[i].w[r]-propagado[r]);
  }
}

void activa()// activa la neurona ganadora para un patrón
//propagado
{
float menor=1000000;
 for(int i=0;i<27;i++)
  if(mapa[i].distancia<menor)
    menor=mapa[i].distancia;
  else
    menor=menor;
  for(int j=0;j<27;j++)
   if(mapa[j].distancia==menor)
    {
     mapa[j].activada=true;
     j=1000;
    }
}
```

```
void actualiza()  //actualiza pesos a la ganadora
{int i;int gan;float vecina;
 for( i=0;i<27;i++)

 if(mapa[i].activada)
  {
   for(int r=0;r<R;r++) //rata=0.2
   mapa[i].w[r]=mapa[i].w[r]+(rata*(propagado[r]-
mapa[i].w[r]));
   mapa[i].activada=false;i=90000;
  }
}


void cargar() //carga los patrones ya recogidos para el
//entrenamiento, en la matriz p[][[
{int i,u;
f=fopen("patrones.FIL", "rb");
if(f==NULL)
{textprintf(screen,font,100,100,makecol(0,0,0),"error");exit(1);
}
for(i=0;i<135;i++)
   {
   fread(carga,sizeof(carga),1,f);
   for (u=0;u<R;u++)
      p[u][i]=carga[u];
   }
fclose(f);
}


//esta se repetirá t=500 veces para que se logre el
//entrenamiento
void entrenamiento() //carga en el vector propagado uno de
//los vectores que se cargaron antes desde archivo a la matriz p
//y luego comienza el ciclo de entrenamiento
{
for (int i=0;i<135;i++)//son 135 patrones cargados pues
//representan la 27 letras de 5 en 5
{
 for(int k=0;k<R;k++)
  propagado[k]=p[k][i];
  distancia();
  activa();
  actualiza();
}
}

void guarda()//guarda los pesos finales del entrenamiento
{
f=fopen("pesosfin.FIL", "ab+");
if(f==NULL)

{textprintf(screen,font,100,100,makecol(0,0,0),"error");exit(1);
}
for(int i=0;i<27;i++)
 fwrite(mapa[i].w, sizeof(mapa[i].w), 1, f);
fclose(f);
}
```

6 CONCLUSIONES

Al culminar el proyecto los resultados que arrojó éste fueron prácticamente inmejorables, y esto es fácil de deducir después de enfrentar la red neuronal a un conjunto test de patrones compuesto por 100 letras nunca antes vistas a las cuales la red reaccionó con una certeza del 94% y haciendo énfasis en el hecho de que los errores que se dieron en la prueba fueron errores muy lógicos como confundir una U con una V o una letra I con una letra J.

Después de hacerle un análisis exhaustivo a la red se pudieron detectar ciertas insuficiencias en las letras L, I, J ya que para cada una de estas la red concluía bien con cualquier estilo pero no con cualquier posición. La ultima falencia de notoriedad se dio en la letra G, la cual aprendió la red pero con una forma mas semejante a un numero 6, que en si a la letra G.

Es destacable también la experiencia de un entrenamiento de la red usando criterio de vecindad [1] en una capa secundaria de 3x9, con resultados menos óptimos que los anteriores; finalmente es válido concluir que la capacidad que la red adquirió de equivocarse o confundirse de cierto modo no la hace más ineficiente sino más humana que en ultimas es el objetivo que busca asiduamente la Inteligencia Artificial.

7 AGRADECIMIENTOS

El autor brinda sus agradecimientos a la Universidad Tecnológica de Pereira, a Diego Gómez López y Consuelo Valencia, baluartes en la investigación realizada y en su formación académica.

8 BIBLIOGRAFIA

[1] B.M del Brio, “Redes Neuronales y sistemas difusos” 2da ed. Vol 1. Ed. Zaragoza España: Alfaomega Ra-Ma 2002.

[2] J.R Hilera “Redes Neuronales Artificiales” Primera edición. Vol 1. Ed Madrid España: Alfaomega Ra-Ma 2000.