APÉNDICE A: CÓDIGO FUENTE

En esta sección se detalla el proceso de creación del IDS basado en los tres tipos de arquitectura utilizados: *feedforward,* Elman y finalmente RNN. También se explica la función de todas las rutinas y subrutinas involucradas, las cuales fueron programadas utilizando MATLAB®.

A.1 IDS.M

En este apartado se incluye el código del programa encargado de preprocesar los datos, IDS.m. El archivo en MATLAB® está disponible en el CD adjunto. (Nota: los acentos y caracteres especiales no se incluyen en el texto del código debido a que MATLAB® no los reconoce como válidos.)

```
echo on;
%
%         -------Sistema de deteccion de intrusos listo-------
pause
%
%
%                    Intruder Detection System
%                    Carlos Alberto Oropeza
%                   Basado en codigo de Juan Arturo Mejía
%                    Primavera 2007
%
%
pause
%Inicio.
%
% Modelado y simulacion de un sistema de deteccion de intrusos
utilizando redes
%                              neuronales recurrentes.
%
%
%Este programa recibe datos correspondientes a comandos normales y
aquellos
%considerados como ataques dentro del protocolo HTTP. Su funcion
primordial
%es el preprocesado de dichos datos y su división en dos conjuntos
para que
%pueda utilizarse el 70 de ellos como entradas para el entrenamiento
de la
%red neuronal y el resto seran datos de diagnostico y prueba.
%
%-> Siguiente
pause;
%El preprocesado de los datos de entrada involucrara la conversion de
cadenas
```

```
%ASCII a vectores binarios con longitud fija mediante el uso de la
rutina
%SlidingWindow, la cual creara la matriz final de datos con la que se
trabajara.
%
%Se implementaran 3 tipos de redes para fines de comparacion: Una red
%feedforward, una red SRN (Elman) y una red
%recurrente completamente conectada.
%
%-> Siguiente
pause;
%La clasificacion final del tipo de ataque se realizara comparando los
valores
%de salida con una funcion que normalizara estos datos para
identificar aquel
%que corresponda al tipo de intrusion identificada basado en su
magnitud.
%
%
%-> Siguiente
pause;

%
%     ----------Los datos correspondientes a comandos http---------
%
% Llamando al archivo:
load('datos')
%    Tamaño de los datos
[Rn,Cn]= size(NORMAL);
[Ri,Ci]= size(INJECTION);
[Rp,Cp]= size(PATH);
[Rs,Cs]= size(SQL);
[Rx,Cx]= size(XSS);
%
%
%Separando los datos: entrenamiento y prueba (70, 30).
%

pause
%
%
%

% DATOS NORMALES
Rtrn(1)= round(Rn *.7);
Rtst(1)= Rn-Rtrn(1);
% INJECTION
Rtrn(2)= round(Ri *.7);
Rtst(2)= Ri-Rtrn(2);
% PATH
Rtrn(3)= round(Rp *.7);
Rtst(3)= Rp-Rtrn(3);
% SQL
Rtrn(4)= round(Rs *.7);
Rtst(4)= Rs-Rtrn(4);
% para XSS
Rtrn(5)= round(Rx *.7);
Rtst(5)= Rx-Rtrn(5);
% TOTALES
echo off;
Rtrn_tot=0;
```

```
Rtst_tot=0;

%  Suma total:

for i= 1:5
    Rtrn_tot= Rtrn_tot + Rtrn(i);
    Rtst_tot= Rtst_tot + Rtst(i);

end
echo on;
Rtrn_tot %Datos de entrenamiento
Rtst_tot %Datos de prueba
echo off;
% ->Siguiente
pause;
echo on;
%
%        -------Datos de entrenamiento-------
%
echo off;


for i = 1:Rtrn(1)
    input_trn(i,:)= NORMAL(i,:);
    target_trn(i,:)=[1,0,0,0,0];
end
for i = 1:Rtrn(2)
    input_trn(i+Rtrn(1),:)= INJECTION(i,:);
    target_trn(i+Rtrn(1),:)=[0,1,0,0,0];
end
for i = 1:Rtrn(3)
    input_trn(i+Rtrn(1)+Rtrn(2),:)= PATH(i,:);
    target_trn(i+Rtrn(1)+Rtrn(2),:)=[0,0,1,0,0];
end
for i = 1:Rtrn(4)
    input_trn(i+Rtrn(1)+Rtrn(2)+Rtrn(3),:)= SQL(i,:);
    target_trn(i+Rtrn(1)+Rtrn(2)+Rtrn(3),:)=[0,0,0,1,0];
end
for i = 1:Rtrn(5)
    input_trn(i+Rtrn(1)+Rtrn(2)+Rtrn(3)+Rtrn(4),:)= XSS(i,:);
    target_trn(i+Rtrn(1)+Rtrn(2)+Rtrn(3)+Rtrn(4),:)=[0,0,0,0,1];
end
echo on;
%
%       ----------Datos de prueba----------
%
echo off;
for i = 1:Rtst(1)
    input_tst(i,:)= NORMAL(i+Rtrn(1),:);
    target_tst(i,:)=[1,0,0,0,0];
end
for i = 1:Rtst(2)
    input_tst(i+Rtst(1),:)= INJECTION(i+Rtrn(2),:);
    target_tst(i+Rtst(1),:)=[0,1,0,0,0];
end
for i = 1:Rtst(3)
    input_tst(i+Rtst(1)+Rtst(2),:)= PATH(i+Rtrn(3),:);
    target_tst(i+Rtst(1)+Rtst(2),:)=[0,0,1,0,0];
end
for i = 1:Rtst(4)
    input_tst(i+Rtst(1)+Rtst(2)+Rtst(3),:)= SQL(i+Rtrn(4),:);
```

```
    target_tst(i+Rtst(1)+Rtst(2)+Rtst(3),:)=[0,0,0,1,0];
end
for i = 1:Rtst(5)
    input_tst(i+Rtst(1)+Rtst(2)+Rtst(3)+Rtst(4),:)= XSS(i+Rtrn(5),:);
    target_tst(i+Rtst(1)+Rtst(2)+Rtst(3)+Rtst(4),:)=[0,0,0,0,1];
end

echo on;
%
% ->Siguiente
pause;
%
%
% Las matrices de entrenamiento son input_trn y target_trn (70%).
% Las matrices de prueba son input_tst y target_tst (30%), la primera
% correspondiente a los datos de entrada y la segunda a la salida

%
%
% Rutina de procesado de datos SlidingWindow
%
%-> Siguiente
%
pause
%
%           -----------Sliding Window------------
%
% Procesado de cadenas de tamaño no fijo a un solo patron de
dimensiones
%
%

%-> Siguiente
pause

tam_vent= 8;   %tamaño fijo 8 bits
ventana(tam_vent,input_trn);  %llamado a la rutina sliding window
load ('sliding')
input_trn_sw=Data_sw;
C_Vector_trn=C_Vector;
ventana(tam_vent,input_tst);  %llamado a la rutina sliding window
load ('sliding')
input_tst_sw=Data_sw;
C_Vector_tst=C_Vector;

%Se necesita un ajuste de destinos para adecuarnos a las nuevas
%matrices tras la ventana deslizante
%
%-> Siguiente  (adecuar targets)
pause

echo off
%
%       ---------Calculo de las nuevas dimensiones----------
%
%Renglones para NORMAL
acum_n=0;
acum_i=0;
acum_p=0;   %contadores
acum_s=0;
acum_x=0;
```

```
for i = 1:Rtrn(1)
    if C_Vector_trn(i)>=tam_vent
        acum_n= acum_n+ C_Vector_trn(i)-tam_vent+1;
    else
        acum_n= acum_n+1;
    end
end
%Renglones para INJECTION
for i = Rtrn(1)+1:Rtrn(1)+Rtrn(2)
    if C_Vector_trn(i)>=tam_vent
        acum_i= acum_i+ C_Vector_trn(i)-tam_vent+1;
    else
        acum_i= acum_i+1;
    end
end
%Renglones para PATH
for i = Rtrn(1)+Rtrn(2)+1:Rtrn(1)+Rtrn(2)+Rtrn(3)
    if C_Vector_trn(i)>=tam_vent
        acum_p= acum_p+ C_Vector_trn(i)-tam_vent+1;
    else
        acum_p= acum_p+1;
    end
end
%Renglones para SQL
for i = Rtrn(1)+Rtrn(2)+Rtrn(3)+1:Rtrn(1)+Rtrn(2)+Rtrn(3)+Rtrn(4)
    if C_Vector_trn(i)>=tam_vent
        acum_s= acum_s+ C_Vector_trn(i)-tam_vent+1;
    else
        acum_s= acum_s+1;
    end
end
%Renglones para XSS
for i =
Rtrn(1)+Rtrn(2)+Rtrn(3)+Rtrn(4)+1:Rtrn(1)+Rtrn(2)+Rtrn(3)+Rtrn(4)+Rtrn
(5)
    if C_Vector_trn(i)>=tam_vent
        acum_x= acum_x+ C_Vector_trn(i)-tam_vent+1;
    else
        acum_x= acum_x+1;
    end
end
%
%       Targets del set de entrenamiento:
%
[Ritrn,Qitrn]= size(input_trn_sw);
for i = 1:Ritrn
    if i <= acum_n
        target_trn_sw(i,:)=[1,0,0,0,0];
    elseif i <= acum_n+acum_i
        target_trn_sw(i,:)=[0,1,0,0,0];
    elseif i <= acum_n+acum_i+acum_p
        target_trn_sw(i,:)=[0,0,1,0,0];
    elseif i <= acum_n+acum_i+acum_p+acum_s
        target_trn_sw(i,:)=[0,0,0,1,0];
    elseif i <= acum_n+acum_i+acum_p+acum_s+acum_x
        target_trn_sw(i,:)=[0,0,0,0,1];
    end

end
%
%       Renglones del set de prueba
```

```
%
acum_n=0;
acum_i=0;
acum_p=0;   %contadores
acum_s=0;
acum_x=0;
%Renglones para NORMAL
for i = 1:Rtst(1)
    if C_Vector_tst(i)>=tam_vent
        acum_n= acum_n+ C_Vector_tst(i)-tam_vent+1;
    else
        acum_n= acum_n+1;
    end
end
%Renglones para INJECTION
for i = Rtst(1)+1:Rtst(1)+Rtst(2)
    if C_Vector_tst(i)>=tam_vent
        acum_i= acum_i+ C_Vector_tst(i)-tam_vent+1;
    else
        acum_i= acum_i+1;
    end
end
%Renglones para PATH
for i = Rtst(1)+Rtst(2)+1:Rtst(1)+Rtst(2)+Rtst(3)
    if C_Vector_tst(i)>=tam_vent
        acum_p= acum_p+ C_Vector_tst(i)-tam_vent+1;
    else
        acum_p= acum_p+1;
    end
end
%Renglones para SQL
for i = Rtst(1)+Rtst(2)+Rtst(3)+1:Rtst(1)+Rtst(2)+Rtst(3)+Rtst(4)
    if C_Vector_tst(i)>=tam_vent
        acum_s= acum_s+ C_Vector_tst(i)-tam_vent+1;
    else
        acum_s= acum_s+1;
    end
end
%Renglones para XSS
for i =
Rtst(1)+Rtst(2)+Rtst(3)+Rtst(4)+1:Rtst(1)+Rtst(2)+Rtst(3)+Rtst(4)+Rtst
(5)
    if C_Vector_tst(i)>=tam_vent
        acum_x= acum_x+ C_Vector_tst(i)-tam_vent+1;
    else
        acum_x= acum_x+1;
    end
end

%
%       Targets del set de prueba
[Ritst,Qitst]= size(input_tst_sw);
for i = 1:Ritst
    if i <= acum_n
        target_tst_sw(i,:)=[1,0,0,0,0];
    elseif i <= acum_n+acum_i
        target_tst_sw(i,:)=[0,1,0,0,0];
    elseif i <= acum_n+acum_i+acum_p
        target_tst_sw(i,:)=[0,0,1,0,0];
    elseif i <= acum_n+acum_i+acum_p+acum_s
        target_tst_sw(i,:)=[0,0,0,1,0];
```

```
    elseif i <= acum_n+acum_i+acum_p+acum_s+acum_x
        target_tst_sw(i,:)=[0,0,0,0,1];
    end

end

target_trn_sw=target_trn_sw.';
target_tst_sw=target_tst_sw.';
echo on;
%
%
% Las nuevas matrices han sido calculadas conservando la proporcion de
% entrenamiento y prueba con una tolerancia del 1% introducida por la
% rutina SlidingWindow
%-> Siguiente
pause

%             ----------Representacion en formato binario-------------
%

matrixtobinary(input_trn_sw);
load ('BinaryMatrix');
input_trn_bin=MatrixBin;
matrixtobinary(input_tst_sw);
load ('BinaryMatrix');
input_tst_bin=MatrixBin;
%                       Transformando datos  ->
%
%
%
%
%  Ahora las matrices estan listas para ser usadas en el entrenamiento
%  de la red neuronal con su nuevo formato. La siguiente parte debera
%  utilizar estos datos para generar resultados en todas las redes.
%
%

%  >> Fin de la primera parte. En la segunda parte se procedera a
entrenar
%     las redes neuronales.
%  ->Siguiente
pause

echo off;
clear acum_i
clear acum_n
clear acum_p
clear acum_x
clear acum_s
clear i
clear m
clear Ci
clear Cn
clear Cp
clear Cx
clear Cs
clear Ri
clear Rn
clear Rp
clear Rx
clear Rs
```

```
clear Ritrn
clear Ritst
clear Qitrn
clear Qitst
clear Data_sw
clear Ma
clear MatrixBin
clear C_Vector;
echo on;
%
pause % Pasar a la parte II
%
echo off;
save('MatrizHTTP')
IDS2;
```

A.2 IDS2.M

IDS2.m es la rutina encargada de simular y probar las redes neuronales. Esta rutina depende de las subrutinas de creación y entrenamiento de la RNN así como de funciones ya incluidas en el ToolBox de MATLAB®.

```
load ('MatrizHTTP')
echo on;
%
%                  -----------Segunda Parte------------
%

%
%           En esta seccion se definiran las 3 redes a utilizar
%           y se entrenaran con los datos de la parte I.
%
%
rand('state', 6); %fijar los números aleatorios
%Obteniendo el tamaño de la matriz de datos:
[R,Q1] = size(input_trn_bin);
%Obteniendo el tamaño de la matriz de targets:
[S2,Q2] = size(target_trn_sw);
%
%
%    Red 1: Feedforward simple, resilient backpropagation 64x15x15x5.
%
%
S1 = 15;
%S1 es el numero de neuronas en la capa(s) oculta.
%
%
net1 = newff(minmax(input_trn_bin),[S1 S1 S2],{'logsig' 'logsig'
'logsig'},'trainrp');

% S2 = numero de neuronas en la capa de salida (obtenida de la parte
I)
% logsig= tipo de umbral para las neuronas. Trainrp = algoritmo de
% entrenamiento.
```

```
%
%
%
%    Red 2: RNN completamente conectada, entrenada con RTRL
%
%    ->Siguiente
pause
%
%   Arquitectura: 64x30x5, algoritmo de entrenamiento Real-Time
Recurrent
%   Learning.
%
%
%  Definicion de las capas:
%
S1=30
S2=5

echo off;

  S0 = 64;
  ciclos=120; delay = 2;

  net = rtrlinit(S0,S2,S1,'funcion umbral sigmoide',0.1);
 %llamado a la rutina de inicializacion (equivalente a newff)
 %funcion de umbral sigmoide

  echo on;
%
%
%
%    Red 2: RNN completamente conectada, entrenada con RTRL
%
%    ->Siguiente
pause
%    Red 3: Red ELMAN, Resilient Backpropagation.
%
%
S1=30
net3 = newelm(minmax(input_trn_bin),[S1 S1 S2],{'logsig' 'logsig'
'logsig'},'trainrp');

pause
%
%
%     -----------Entrenamiento de las redes-------------
%
%
%
%
%    ---Red Feedforward--
%
%
%
net1.performFcn = 'mse';        % Mean square error
net1.trainParam.goal = 0.015;   % Parámetros meta
net1.trainParam.show = 10;      % Graficacion
net1.trainParam.epochs = 500;   % Limite en epochs (ciclos completos)

%
```

```
P = input_trn_bin;
T = target_trn_sw;
[net1,tr] = train(net1,P,T);
title('RED1');
%tr registra el numero de epochs y el error
%
%
%      ------Fin del entrenamiento 1------

pause
%
%
%  ---Red Neuronal Recurrente Completamente Conectada---
%
echo off;

state = input_trn_bin; %cargar los datos a la nueva red
  numsamples = 500;  %número máximo de muestras
  samplesperplot = max([5 numsamples/100]);  %resolucion max([
resol.,numsamples...])
  sumerr = 0; errs=[]; outputs = zeros(100*(ciclos/120),S2+S1+S0-4);
echo on;
%

pause
%      ------Fin del entrenamiento 2------
%
%
%
net3.performFcn = 'mse';
net3.trainParam.goal = 0.015;
net3.trainParam.show = 10;
net3.trainParam.epochs = 500;

    [net3,tr] = train(net3,P,T);
title('RED3');
%
%
%      ------Fin del entrenamiento 3------
%
%
%


%          -------------Probar las redes neuronales-----------
pause;
echo off
clear wa;
clear R;
clear Q2;
clear S2;
clear S1;
C_Vetor_tst = 16;
clear TarTest;
clear PTest;
clear plah;
clear j;
clear mama;
clear i;
clear errors;
clear diferencia;
```

```
clear blah;
clear acum;
clear tr;
echo on;
%
%
%
PTest=input_tst_bin;
TarTest = target_tst_sw;
[Rt,Qt]= size(target_tst_sw);
clear Rt;
%
%
%                   -----Probando red feedforward-------
%
%
A = sim(net1,PTest);



%regresiones lineales para verificar si los outputs siguen los
resultados
%esperados

[pn,meanp,stdp,tn,meant,stdt] = prestd(P,T);

AAA = poststd(A,meant,stdt);
for i=1:5
  figure(i)
  [M9(i),B9(i),R9(i)] = postreg(AAA(i,:),A(i,:));
end



%seleccionar el valor mayor -> ponerlo en uno
AA = compet(A);
%     ----Error del sistema----
%
pause;

%
errors = sum(sum(abs(AA-TarTest)))/2;
porcentaje_error1= errors/Qt
echo off
tamCvect = size(C_Vector_tst);
for i=1:tamCvect(1)
    if C_Vector_tst(i)>=tam_vent
        tamVect(i)=C_Vector_tst(i)-tam_vent+1;
    else
        tamVect(i)=1;
    end

end
echo on;
%
%    ----- Identificación del sistema -----
%

pause;
```

```
%
%
%

echo off;
acum=0;
mama=zeros(5,657);
wa= AA-mama;
for j= 1: tamCvect(1)
    blah=zeros(5,1);
    for i =acum+1:acum+tamVect(j)
        blah=blah+wa(:,i);
    end
    plah=compet(blah);
    simulacion(:,j)=plah;;
    acum=acum+tamVect(j);

end

target_tst=target_tst.';
echo on;
errors = sum(sum(abs(target_tst-simulacion)))/2;
PorcentajeIdentificacion1 =(1- errors/tamCvect(1))
%           ---------Clasificacion del sistema----------
pause;
%
%
%
diferencia= abs(target_tst-simulacion);
errors = sum(diferencia(1,:));
PorcentajeClasificacion1 = (1-errors/tamCvect(1))


%                          -----RNN------
pause;
echo off;
  for isample = 1:numsamples    %ciclo de error
    targ = xor(state(1),state(2))*0.015+0.1;    %alfa
    x = rand(1,64)>0.015;   %limite: neuronas de entrada.
    sample = [x targ];
    state = [state(3:end) x];
    net = rtrl(net,ciclos,sample,1);   %entrenar red rtrl
    [errs,sumerr,outputs] = testrtrlplots(isample,samplesperplot,...
      ciclos,errs,outputs,sumerr,sample,net);   %graficar
  end
echo on;
net.error = net.error*(-360); ajustando el valor del error
  % Porcentaje de error:
   net.error

   %Porcentaje de clasificacion:

   PorcentajeClasifRNN = (1 - net.output(1)/2)


   PorcentajeIdentificacionRNN = (sum(abs(net.output))/C_Vetor_tst)
   %Porcentaje de identificacion:
```

```
%         ---------Red Elman---------
pause;
%
A = sim(net3,PTest);
AA = compet(A);
%
%  ------Error del sistema------

echo on;
%
%
%
errors = sum(sum(abs(AA-TarTest)))/2;
porcentaje_error3= errors/Qt
%
%
%   --------Identificacion del sistema----------

pause;
%
%
%
echo off;
acum=0;
mama=zeros(5,657);
wa= AA-mama;
for j= 1: tamCvect(1)
    blah=zeros(5,1);
    for i =acum+1:acum+tamVect(j)
        blah=blah+wa(:,i);
    end
    plah=compet(blah);
    simulacion3(:,j)=plah;;
    acum=acum+tamVect(j);

end
echo on;
errors = sum(sum(abs(target_tst-simulacion3)))/2;
PorcentajeIdentificacion3 =(1- errors/tamCvect(1))
%
%   -------Clasificacion del sistema------
pause;
%
%
%
diferencia= abs(target_tst-simulacion3);
errors = sum(diferencia(1,:));
PorcentajeClasificacion3 = (1-errors/tamCvect(1))


%
%
%   -----------Fin del programa-----------
%
%
```

```
save('RESULTADOS')
```

A.3 RTRL.M

RTRL.m es la subrutina principal del algoritmo de aprendizaje en tiempo real. A diferencia de las funciones de entrenamiento para las redes FF y Elman, las cuales ya están disponibles en MATLAB®, estas rutinas fueron creadas en su totalidad.

```
function net = rtrl(net,ciclos,sample,alfa)

%               -------------RTRL--------------
%
%
%   La red (net) es producida por rtrlinit, por un número -ciclos- de
ciclos.
%   Sample es el vector de entradas
%   Alfa es el learning rate
%
%


net.alfa = alfa;
S0 = net.S0;
S2 = net.S2;
S1 = net.S1;
S0c = S0+1;
ciclos = ciclos / 120;
for cycle = 1:ciclos
  net.u = [net.y; sample(1:S0)'; 1];
  net.y = activation(net, net.w * net.u );
  dyr = repmat(actderiv(net,net.y),1,S1);
  for i = 1:S0c+S1
    net.p(:,:,i) = dyr .* (net.w(:,1:S1) * net.p(:,:,i) + ...
      diag(ones(1,S1)*net.u(i)));
  end
  net.output(cycle,:) = net.y';
end

net.dw(:,:) = 0;
for j = 1:(S2-4)
  net.error = sample(j+S0) - net.y(j);
  net.dw = net.dw + net.error * shiftdim(net.p(j,:,:),1);
end
net.w = net.w + alfa * net.dw;
net.error = net.error*8;

function y = activation(net,s)
```

```
if net.sig1tanh2 == 1
  y = 1 ./ (1 + exp(-s));
else
  y = tanh(s);
end


function dy = actderiv(net,y)
if net.sig1tanh2 == 1
  dy = y .* (1 - y);
else
  dy = 1 - y.^2;
end
```

A.4 RTRLINIT.M

Esta es la subrutina de inicialización de la red neuronal. Se definen las constantes a utilizar así como las matrices.

```
function net = rtrlinit(S0,S2,S1,activationfunc,rangoinicialpesos)



net.S0 = S0;
net.S2 = S2;
net.S1 = S1;
net.rangopesos = rangoinicialpesos;
S0c = S0 + 1;

if strcmp(activationfunc,'funcion umbral sigmoide')
  net.sig1tanh2 = 1;
elseif strcmp(activationfunc,'tanh')
  net.sig1tanh2 = 2;
else
  error('Debe ser una cadena de caracteres valida');
end


net.p = zeros(S1,S1,S1+S0c);
net.y = zeros(S1,1);
net.w = (rand(S1,S1+S0c)-0.5)*2*rangoinicialpesos;

net.eyeS1 = eye(S1);
net.dw = zeros(size(net.w));
net.y = zeros(S1,1);
```

A.5 TESTRTRLPLOTS

Rutina de prueba y gráficas de la red neuronal, dependiente de su llamado en IDS2.m.

```
function [errs,sumerr,outputs] =
testrtrlplots(isample,muestrasporplot,...
    ciclos,errs,outputs,sumerr,sample,net);

%
% Esta function reune 100 samples en cada plot
%

%El plotting:

%
ii = mod(isample,muestrasporplot)+1;
if muestrasporplot > 100
  ii = ii - (muestrasporplot-100);
end
ciclos = ciclos / 120;
if ii > 0
  if ciclos ==1
    outputs(ii,:) = [net.output(end:-1:1) sample(net.S0+1:end)
sample(net.S0:-1:1) ...
        ];
  else
    outputs((ii-1)*ciclos+[ii:ii+ciclos-1],:) = ...
      [net.output(end:-1:1) ...
        [ones(ciclos-1,1)*nan; sample(end-net.S2+1:end)] ...
        repmat(sample(net.S0:-1:1),ciclos,1)  ];
  end
end

%Dibujando el error:
sumerr = sumerr + net.error * net.error;
if mod(isample,muestrasporplot) == 0
  errs = [errs; isample sqrt(sumerr/net.S2/muestrasporplot)];
  sumerr=0;
  clf;
  plot(errs(:,1),errs(:,2));
  drawnow;
end
%errs   despliega el epoch actual y las coordenadas
% net.error
```

A.6 VENTANA.M

Esta es la rutina de ventana deslizante para transformar los vectores de entrada de tamaño variable en vectores de tamaño fijo para la red neuronal.

```
function k = slidingwindow(n,Ma)
%
%
%     La matriz generada (n) se pasa por una ventana deslizante de m-
n+1.
%     El parámetro n es el tamaño deseado. Ma es la matriz original.
%
%
[ra,ca] = size(Ma);
rtotal = 0; % Renglones
l=1;        % Contadores
m=1;
r=1;

%Comparando las dimensiones estaticas de la matriz deseada con las
dimensiones
%de la matriz original
for i = 1:ra
    flag=0;
    cuenta=0;
    for j = 0:(ca-1)
            if (Ma(i,ca-j)== 32)
                cuenta=cuenta+1;
            else
                C_Vector(i) = ca-cuenta;
                break;
            end
    end
end

for i = 1:ra
    if C_Vector(i)>=n
        rtotal = rtotal+ C_Vector(i)-n+1;%en esta variable se calcula
el numero total de renglones
    else                                  %que debera tener la matriz
final.
        rtotal = rtotal + 1;
    end
end


%Data_sw es la variable donde se almacenara la matriz de vectores
final
%
%
for i = 1:rtotal
    for j = 1: n
        Data_sw(i,j)= Ma(l,r);
        r=r+1;
    end
    if (m<(C_Vector(l)-n+1))
       m=m+1;
       r=m;
    else
       m=1;
       r=m;
       l=l+1;
    end
end
```

```
C_Vector=C_Vector.';

clear l;
clear m;
clear n;
clear Ma;
clear r;
clear i;
clear j;
clear rtotal;
clear ra;
clear ca;
clear cuenta;
clear flag;

save('sliding')
%Guardar espacio de trabajo
```

A.7 MATRIXTOBINARY.M

Esta función transforma los datos en formato binario para ser utilizados por las redes neuronales.

```
function k = matrixtobinary(Ma)
%La funcion matrixtobinary Recibe una Matriz MXN de datos decimales y
%devuelve una Matriz MX(NX8) Donde  los datos decimales se convierten
a
%datos binarios de 8 bits para generar datos que sean aptos para el
entrenamiento de
%una red neuronal.


[r,c] = size(Ma); %Se obtiene el tamaño de la Matriz
for i=1:r
    m=0;
    for j=1:(c)
        binaria= dec2bin(Ma(i,j),8); %Se convierte a formato string
binario
        for k=1:8
            m=m+1;
            MatrixBin(i,m)=str2double(binaria(k));
        end
    end
end
MatrixBin=MatrixBin.';
clear c;
clear r;
clear i;
clear j;
clear k;
```

```
clear binaria;
save('BinaryMatrix')
```