PRÁCTICA VII: COMPARACIÓN UNO CONTRA TODOS (REDES NEURONALES)

Karen González Arévalo, Julieth Carolina Montes y Nataly Tinoco.

INTRODUCCIÓN

Una red neuronal artificial es un procesador distribuido en paralelo de forma masiva que tiene una tendencia natural para almacenar conocimiento de forma experimental y lo hace disponible para su uso. Las semejanzas que presenta con respecto al cerebro humano son:

• El conocimiento es adquirido por la red a través de un proceso de aprendizaje.

• Los pesos sinápticos o fuerza con que están interconectadas las neuronas se utilizan para almacenar la información.

También se puede expresar como una nueva forma de computación, inspirada en modelos biológicos, que está compuesto por un gran número de elementos procesales organizados en niveles hecho por un gran número de elementos simples, elementos de proceso interconectados, los cuales procesan información por medio de su estado dinámico como respuesta a entradas externas. [1]

El perceptrón Multinivel Dentro de las redes neuronales, las que más utilizadas son las redes con múltiples capas que funcionan hacia delante. Esta red está compuesta por un conjunto de nodos de entrada que componen la capa de entrada, un conjunto de una o más capas ocultas de neuronas y una capa de neuronas de salida. La señal de entrada se propaga hacia adelante desde la capa de entrada por la oculta hasta la salida; este tipo de configuración se conoce como MLP o "MultiLayer Perceptrons" como se evidencia en la figura 1.[1]

El proceso backpropagation consiste en dos pasadas a través de las diferentes capas de la red, una pasada hacia adelante y una pasada hacia atrás. En la pasada hacia adelante, se aplica en la capa de entrada un patrón o vector de entrada, este propaga su efecto a través de las diferentes capas y como consecuencia produce un vector de salida. Durante este proceso, los pesos sinápticos de la red son fijos y no se modifican. Durante la pasada hacia atrás en cambio, los pesos si se modifican de acuerdo con la regla de corrección del error.

A comparación como se muestra en el modelo de cada neurona (figura 2) incluye una función no lineal. En este caso, a diferencia del perceptrón donde es la función escalón, y debido a la necesidad de que sea una función continua y derivable.

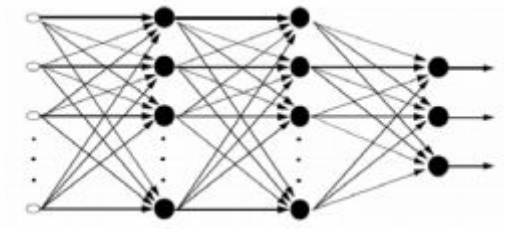

Figura 1: *Estructura de una perspectiva multinivel.*

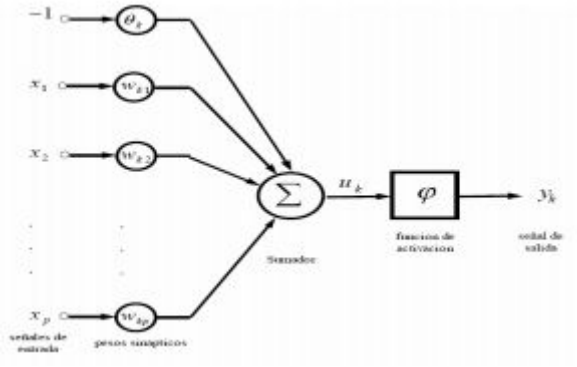

Figura 2: *Modelo de una neurona*

METODOLOGÍA

Se establece una capa de entrada de 400 nodos, es decir, 400 entradas las cuales poseen imágenes de 20x20 pixeles, capa intermedia de 25 nodos y la capa de salida de 10 nodos, estableciéndose 1 por cada clase. Para el desarrollo de esta metodología hay un número fijo de categorías en las cuales las muestras de entrada deben clasificarse. Para ello primero se requiere una fase de entrenamiento en la que se presenta a la red los patrones que debe aprender y la categoría en cual clasificarlo. Entonces se le presenta a la red un patrón nuevo y desconocido pero que pertenece a alguna de las categorías aprendidas y esta debe decidir a qué categoría se parece más.
Para llevar a cabo la ejecución de propagación hacia adelante se realiza un vector de unos para indexarlos con las variables de entrada definidas como equis (X), para posteriormente realizar la multiplicación con las variables de entrada, y con esta respuesta se calcula la función hipolog, y este procedimiento se vuelve a repetir para los segundos valores de theta. Posteriormente se lleva a cabo el rendimiento del modelo, en esta parte se debe obtener el porcentaje del número de aciertos de la clasificación en comparación con las observaciones reales, donde la gráfica arrojada permite comparar la salida del algoritmo (por clase) con los datos de entrenamiento proporcionados (y) de manera visual para tener una noción de lo obtenido con el algoritmo implementado.

Para la propagación hacia atrás a grandes rasgos su funcionamiento se basa en la inicialización, para cargar los datos junto con los parámetros iniciales teniendo como entradas, imagenes de 20X20 píxeles, 25 unidades en la capa oculta y 10 clases de salida (1-10), para posteriormente cargar los datos con sus respectivos parámetros donde se hace un vector de parámetros theta, posterior a estos resultados se computa el costo con los parámetros iniciales(FP), en esta parte del algoritmo se debe implementar la función de coste de la red neuronal y probar que su salida sea la correcta, donde inicialmente se puede asumir un costo sin regularización, estableciendo el factor de regularización para comprobar la implementación de la función de coste. Por consiguiente se implementa la regularización, una

vez haya comprobado que su función de coste está correctamente implementada, se comprueba la implementación de la regularización, utilizando el valor de lambda, por consiguiente a esto se inicializan los parámetros, en esta parte del algoritmo se implementa la función para inicializar aleatoriamente los parámetros de aprendizaje Theta de cada capa, donde se colocan todos los parámetros en un vector, para la implementación de Backpropagation y su respectiva validación, una vez comprobado que funciona la implementación de la función de coste, al implementar el algoritmo de Backpropagation para obtener la derivada de la función de coste y comprobar que funcione correctamente.En esta parte deberá implementar el algoritmo de comprobación del gradiente por métodos numéricos en la función del gradiente. Para la implementación de la regularización para el gradiente, si se establece que Backpropagation es correcta, se continúa con la implementación de la regularización, para así lograr comprobar el gradiente de BP con la comprobación del gradiente junto con la comprobación de la función de coste.
Con lo anterior se entrena la red neuronal, después de haber realizado las correspondientes validaciones, para entrenar la red neuronal, para ello se debe implementar diferentes valores de lambda para obtener las dos variables de theta. Para finalizar se implementa la metodología por predicción, donde una vez entrenada la red neuronal, los parámetros Theta sirven para predecir nuevos valores que le sean presentados. En esta parte se implementará la función para predecir a qué dígito pertenece cada una de las imágenes de los datos de entrenamiento. A partir de los valores predecidos, se logra calcular el rendimiento de la red neuronal donde se utiliza la función propagación hacia adelante.

RESULTADOS

Parte 1. Modelo de Propagación hacia adelante:

Para el desarrollo de la primera parte de la práctica de laboratorio se implementó la siguiente función:

clase_NN = PropAdelante (Theta1, Theta2, X)

Con la cual, conociendo los pesos asociados a cada capa y utilizando los parámetros proporcionados en ThetaFF1 y ThetaFF2 se realizó la transformación para cada nodo de la capa oculta y la de salida. Dentro de la función se agregó otras funciones como lo son indexación de vectores de uno e hipolog para cada una de las capas correspondientes. En donde, la matriz hipolog de la última capa es analizada a través de la función max, la cual da el valor máximo, indica a qué clase pertenece la observación y el porcentaje del número de aciertos clasificación en comparación con las observaciones reales.

Los resultados mencionados anteriormente, se pueden evidenciar a continuación:

- La clasificación de las observaciones se puede evidenciar en la tabla 1, en donde se observa que las observaciones fueron clasificadas de acuerdo al dato arrojado por la

función max ubicada en la programación. Al comparar esta tabla con los datos de validación se comprueba que la variación entre ellos es poca, es decir, son muy pocos los datos que son erróneos y fueron catalogados en una clase diferente.

- El porcentaje de aciertos del modelo de propagación hacia adelante es de 92.52 %, valor que indica un error porcentual de 7,48%. Este resultado indica que el método de propagación hacia adelante es eficiente y viable, permitiendo en gran parte clasificar acertadamente cada observación. Sin embargo, el error porcentual se puede considerar que posiblemente se debe a los valores aproximados que son resultado de las matrices operadas, así como también a la falta de normalización y regularización de algunos datos.

- Gráfica que permite comparar la salida del algoritmo (class) con los datos de entrenamiento proporcionados (y) de manera visual para tener una noción de lo obtenido con el algoritmo:

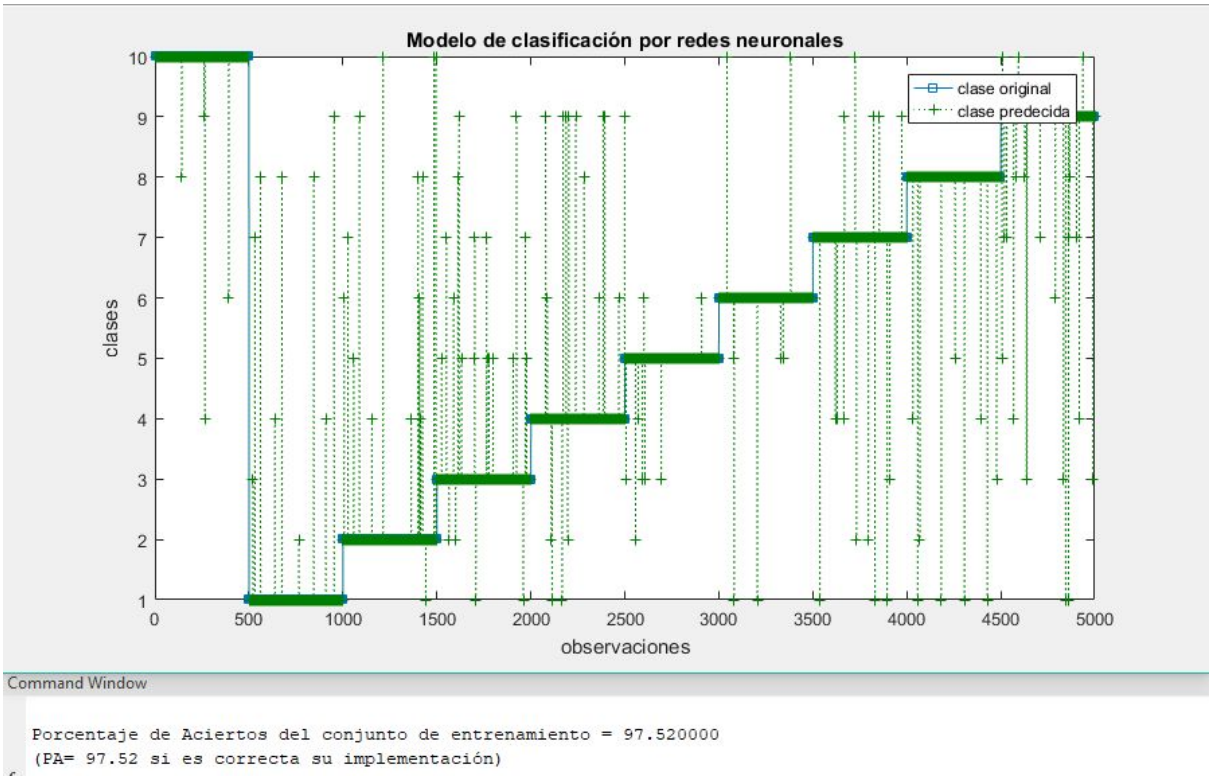

Figura 3: *Gráfica relación salida del algoritmo (class) vs datos de entrenamiento proporcionados (y).*

En la figura 3 se puede observar que la relación que se encuentra entre las variables o ejes de salida del algoritmo es propicia, ya que la incoherencia cuantitativa entre cada uno de estos valores como se aprecia en la figura es mínima, valor que junto con el valor porcentual el error entre estas variables es menor al 10%.

Parte 2. Modelo de propagación hacia atrás:

En el modelo de propagación hacia atrás para la segunda etapa se realizó la siguiente función con lambda 0:

J = CosteRedNeuronal(theta_in, n, sl, num_clases,...X, y, lambda)

Dicha función permite comprobar la implementación de la función de coste arrojando un valor de 0.28.

```
Con los valores iniciales Thetainl y Thetain2
el valor de la función de coste es: 0.287629
(si su implementación es correcta debe ser 0.287629)
Pausa. Presione enter para continuar.
```

Figura 4: *Resultados arrojados por el software*

Con la figura 4 se evidencia y corrobora que los valores de la función de coste y su implementación hacen referencia a 0.28.

Posteriormente, al implementar la regularización, la función implementada y el lambda escogido para su verificación es 1:

J = CosteRedNeuronal(theta_in, n, sl, num_clases,...X, y, lambda)

Dicha función permite comprobar la implementación de la función de coste arrojando un valor de 0.38.

```
Comprobando J con regularización ...
Para los parámetros theta_in previamente establecidos y con lamda= 1,
el valor de la función de coste es J= 0.384488
(debe ser igual a 0.383770)
```

Figura 5: Resultados arrojados por matlab

Estos valores permiten concluir que la efectividad del código del modelo de propagación hacia atrás es propicio y su proceso de desarrollo o metodología escogida posiblemente es la más certera para este tipo de datos y áreas de trabajo. A pesar de todo, cabe aclarar que los siguientes decimales que acompañan el dato de corroboración de la regularización no coinciden con los datos del valor teórico esperado, error que puede ser considerado debido a lo mencionado anteriormente o a la lógica implementada en la metodología.

```
Entrenando la Red Neuronal...
Entre valor de lambda= 3
Entre # de iteraciones= 50
Iteration     1 | Cost: 3.312835e+00
Iteration     2 | Cost: 3.234141e+00
Iteration     3 | Cost: 3.139472e+00
Iteration     4 | Cost: 2.973303e+00
Iteration     5 | Cost: 2.953311e+00
Iteration     6 | Cost: 2.432378e+00
Iteration     7 | Cost: 2.374640e+00
```

Figura 6: *Resultados del entrenamiento*

Los resultados arrojados para la figura 6 evidencian las comprobaciones para entrenar la red neuronal, optimizando las iteraciones, probando diferentes valores para lambda.

```
Para labmda= 3 y 50 iteraciones, se obtiene:
Porcentaje de aciertos del conjunto de entrenamiento: 81.000000
```

Figura 7: *Implementación de la predicción*

Según la figura 7 se evidencia la implementación para llevara cabo la predicción, donde una vez entrenada la red neuronal evaluando los parámetros theta, los cuales predecirán los valores presentados, donde a partir de los valores predecidos para así calcular el rendimiento de la red neuronal.

Cuando se probaba con diferentes números de interacciones se observó que en muchas ocasiones no llegaba a realizar ni la mitad de la interacciones que se querían (como se ve en la figura 8), sin embargo se tomaron aun así los datos de la siguiente tabla.

	Lambda			
# de interacciones	**0**	**0.5**	**1**	**3**
50	**82**	**84.02**	**86.96**	**81**
100	**87.12**	**87.62**	**84.92**	**65.46**
200	**48.52**	**42.5**	**81.44**	**82.3**

```
Entrenando la Red Neuronal...
Entre valor de lambda= 3
Entre # de iteraciones= 100
Iteration     1 | Cost: 3.508273e+00
Iteration     2 | Cost: 3.284152e+00
Iteration     3 | Cost: 3.216806e+00
Iteration     4 | Cost: 3.180904e+00
Iteration     5 | Cost: 3.052069e+00
Iteration     6 | Cost: 2.298446e+00
Iteration     7 | Cost: 2.202265e+00

Pausa. Presione enter para continuar.

Para labmda= 3 y 100 iteraciones, se obtiene:
Porcentaje de aciertos del conjunto de entrenamiento: 65.460000
```

Figura 8: *Ejemplo donde no se alcanzan a realizar las interacciones que se querían*

CONCLUSIONES

1. La ventaja de usar redes neuronales está en el hecho que se pueden separar regiones no lineales de decisión tan complicadas como se desee dependiendo del número de neuronas y capas. Por lo tanto, las redes neuronales artificiales sirven para resolver problemas de clasificación de alta complejidad.
2. La implementación de redes neuronales tiene un campo de acción significativo en la bioingeniería, como por ejemplo para la detección de enfermedades, gracias al entrenamiento que se lleva a cabo con base al genoma humano, a la implementación de señales biológicas e incluso para aplicaciones enfocadas en estudios biomecánicos con enfoque deportivo, animal y humano con enfoque en diseño de prótesis.
3. Con las metodologías establecidas de propagación hacia adelante se presenta una eficiencia y viabilidad de ejecución, permitiendo una clasificación acertada para cada una de las observaciones con un porcentaje de error porcentual mínimo evidenciado en la zona de resultados.
4. Al establecer la propagación hacia atrás la función de coste el valor arrojado es de 0.28, lo que indica que el error que se establece es mínimo para el procedimiento en general.

REFERENCIAS

[1] E. M. Aldana, J. L. Valverde and N. Fábregas, "Consciencia, cognición y redes neuronales: nuevas perspectivas," *Revista Española De Anestesiología y Reanimación,* vol. 63, *(8),* pp. 459-470, 2014;2016.

ANEXO

<u>códigos</u>

- **mainLab8**

```
%% Laboratorio Redes neuronales
%
% Instrucciones. Implementar las siguientes funciones para entrenar una red
% neuronal. No debe modificar esta rutina principal a menos que se pida lo
% contrario.
%     PropAdelante
%     LogDer.m
%     CosteRedNeuronal.m (puede basarse en su implementación de la regresión
%                         logística)
%     InicializarTheta.m
%     GradienteNumerico.m
% Nota: necesitará algunas de las funciones implementadas en el laboratorio
% de regresión logística

%% %%%%%%%%%%%%%%%%% PARTE A: PROPAGACIÓN HACIA ADELANTE
%%%%%%%%%%%%%%%%%%%
% Inicialización y definición de parámetros
clc; close all; clear;
n_entrada  = 400;        % Imagenes de 20x20 píxeles
n_oculta = 25;           % 25 nodos en la capa oculta
num_clases = 10;         % 10 nodos en la capa de salida

% cargar datos de entrenamiento y parámetros optimizados
load 'Lab8data.mat' ThetaFF1 ThetaFF2 X y;
m = size(X, 1);          % Número de observaciones

%% === Parte 1: Implementar Propgación hacia delante =========
%
claseNN = PropAdelante(ThetaFF1, ThetaFF2, X);

%% ============= Parte 2: Rendimiento del modelo =============
% En esta parte se debe obtener el porcentaje del número de aciertos de la
% clasificación en comparación con las observaciones reales

% La siguiente gráfica permite comparar la salida del algoritmo (class)
% con los datos de entrenamiento proporcionados (y) de manera visual para
% tener una noción de lo obtenido con el algoritmo
```

```
figure, plot(y,'marker','s');
hold on, plot(claseNN,'linestyle',':', 'marker','+','color',[0 0.5 0])
legend('clase original', 'clase predecida')
xlabel('observaciones'), ylabel('clases')
title('Modelo de clasificación por redes neuronales')
fprintf('\nPorcentaje de Aciertos del conjunto de entrenamiento = %f\n',
mean(double(claseNN == y)) * 100);
display('(PA= 97.52 si es correcta su implementación)');

%% %%%%%%%%%% PARTE B: PROPAGACIÓN HACIA ATRÁS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Inicialización
clear ; close all; clc
% Cargar datos y parámetros iniciales

load lab8data.mat Thetain1 Thetain2 X y
n  = 400;  % Entradas, imagenes de 20X20 pixeles
sl = 25;   % 25 unidades en la capa oculta
num_clases = 10;          % 10 clases de salida (1- 10) con 0== clase 10


%% ========== Parte 1: Cargar datos y parámetros ============
m = size(X, 1);

% hacer un vector de parámetros theta
theta_in = [Thetain1(:) ; Thetain2(:)];

%% ===== Parte 2: Computar el Costo con los parámetros iniciales(FP) ======
%  En esta parte del algoritmo debe implementar la función de coste de la red
%  neuronal y probar que su salida sea correcta. Inicialmente puede asumir
%  un costo si regularización

% Factor de regularización
lambda = 0;

J = CosteRedNeuronal(theta_in, n, sl, num_clases,...
                X, y, lambda);


% Comprobar la implementación de la función de coste
fprintf(['Con los valores iniciales Thetain1 y Thetain2'... %f '...
```

```
        '\nel valor de la función de coste es: %f'...
        '\n(si su implementación es correcta debe ser 0.287629)\n'], J);
fprintf('Pausa. Presione enter para continuar.\n');
pause;
%% =============== Parte 3: Implementar Regularización ===============
%  Una vez haya comprobado que su función de coste está correctamente
%  implementada, comprobar la implementación de la regularización

fprintf('\nComprobando J con regularización ... \n')

lambda = 1; %utilizaremos incialmente este valor de lambda

J = CosteRedNeuronal(theta_in, n, sl, num_clases,...
                X, y, lambda);

fprintf(['Para los parámetros theta_in previamente establecidos y con lamda= 1,'...
         '\nel valor de la función de coste es J= %f'...
        '\n(debe ser igual a 0.383770)\n'], J);

fprintf('Pausa. Presione enter para continuar.\n');
pause;


%% ================ Parte 4: Inicialización de Parámetros ================
%  En esta parte del algoritmo debe implementar una función para
%  inicializar aleatoriamente los parámetros de aprendizaje Theta de cada
%  capa

in_Theta1 = InicializarTheta(n, sl);
in_Theta2 = InicializarTheta(sl, num_clases);

% Poner todos los parámetros en un vector
rand_Theta_in = [in_Theta1(:) ; in_Theta2(:)];


% %% ========= Parte 5: Implementar Backpropagation y comprobarlo ==========
% %  Una vez comprobado que funciona la implementación de la función de
% %  coste, implementar el algoritmo de BP para obtener la derivada de la
% %  función de coste y comprobar que funcione correctamente
% %  En esta parte deberá implementar el algoritmo de comprobación del
% %  gradiente por métodos numéricos en la función GradienteNumerico
%
```

```
% fprintf('\nComprobando Backpropagation... \n');
%
% %  Comprobar el gradiente de BP con ComprobarGradiente
% ComprobarGradiente;
%
% fprintf('Pausa. Presione enter para continuar.\n');
% pause;
% %==================no se hizo================
%
% %% ====== Parte 6: Implementar Regularización para el gradiente ===========
% %  Si BP es correcta, continuar implementado regularización
%
% fprintf('\nComprobando BP con Regularización ... \n')
%
% %  Comprobar el gradiente de BP con ComprobarGradiente
% lambda = 3;
% ComprobarGradiente(lambda);
%
% % comprobar tambien la función de coste
% debug_J  = CosteRedNeuronal(theta_in, n, sl, num_clases,...
%                   X, y, lambda);
%
% fprintf(['\n\nCosto en los parámetros preestablecidos (con lambda=3): %f ' ...
%         '\n(este valor debe ser 0.576051)\n\n'], debug_J);
%
% fprintf('Pausa. Presione enter para continuar.\n');
% pause;
% %================no se hizo==========

%% ================== Parte 7: Entrenar la RN ==================
%  Una vez hechas las comprobaciones correspondientes, entrenar la red
%  neuronal. Para esto puede utilizar la función fminunc que es un
%  algoritmo de optimización avanzado

fprintf('\nEntrenando la Red Neuronal... \n')

% En este caso vamos a utilizar 100 iteraciones.
load opciones_opt.mat;

%  Debera probar diferentes valores de lambda
lambda = input('Entre valor de lambda= ');
while ~isnumeric(lambda)
```

```
    fprintf('\nlambda debe ser un valor numérico');
    lambda = input('Entre valor de lambda= ');
end
it = input('Entre # de iteraciones= ');
while ~isnumeric(it)
    fprintf('\nEl número de iteraciones debe ser un valor numérico');
    lambda = input('Entre # de iteraciones= ');
end
options.MaxIter = it;
% Crear una variable auxiliar para la FC
costoRed = @(p) CosteRedNeuronal(p, ...
                                 n, ...
                                 sl, ...
                                 num_clases, X, y, lambda);


%[Theta, costo] = fmincg(costoRed, rand_Theta_in, options);
%Theta1=reshape(theta_in(1:(n+1)*sl),[sl n+1]);
%Theta2=reshape(theta_in(((n+1)*sl)+1:end),[num_clases sl+1]);
%rand_Theta_in=[Theta1(:);Theta2(:)];
[Theta, costo] = fmincg(costoRed, rand_Theta_in, options);
% Obtener Theta1 y Theta2 a partir de Theta
Theta1=reshape(Theta(1:(n+1)*sl),[sl n+1]);
Theta2=reshape(Theta(((n+1)*sl)+1:end),[num_clases sl+1]);
% Theta1=in_Theta1;
% Theta2=in_Theta2;
fprintf('Pausa. Presione enter para continuar.\n');
pause;

%% ================ Parte 8: Implementar Predicción ================
%  Una vez entrenada la red neuronal, los parámetros Theta nos sirven para
%  predecir nuevos valores que le sean presentados.
%  En esta parte implementará la función Predecir  para predecir a que
%  dígito pertenece cada una de las imágenes de los datos de entrenamiento
%  (si hubiera un conjunto de validación, podría utilizarla para predecir
%  la clase de esto datos). A partir de los valores predecidos, puede
%  calcular el rendimiento de la red neuronal
%  Puede utilizar la función PropAdelante para este propósito

pred = PropAdelante(Theta1, Theta2, X);
PredTotal=mean(double(pred == y)) ;
```

```
fprintf(['\nPara labmda= ' num2str(lambda) ' y ' num2str(options.MaxIter) ' iteraciones, se obtiene:'])
fprintf('\nPorcentaje de aciertos del conjunto de entrenamiento: %f\n', PredTotal * 100);
```

PropAdelante

```
function clase_NN = PropAdelante (ThetaFF1, ThetaFF2, X)
        % tt=ones(1,5000);
        % un=(tt ,X);
        X0=ones(5000,1);
        X=[X0 X];
        ZS1=X*ThetaFF1';
        h1 = hipoLog (ZS1);
        Z0=ones(5000,1);
        ZS1=[Z0 h1];
        ZS2=ZS1*ThetaFF2';
        h2 = hipoLog (ZS2);
        [C,I]=max(h2');
        clase_NN=I';
end
```

Función de coste

```
function[J, grad] = CosteRedNeuronal (Theta, n, sl, num_clases, X, y, lambda)
%dividir los thetas
Theta1=reshape(Theta(1:(n+1)*sl),[sl n+1]);
Theta2=reshape(Theta(((n+1)*sl)+1:end),[num_clases sl+1]);
%se binariza la salida
yz=[zeros(size(X, 1)/num_clases,num_clases-1) ones(size(X, 1)/num_clases,1); ones(size(X, 1)/num_clases,1) zeros(size(X, 1)/num_clases,num_clases-1);zeros(size(X, 1)/num_clases,1) ones(size(X, 1)/num_clases,1) zeros(size(X, 1)/num_clases,num_clases-2);zeros(size(X, 1)/num_clases,2) ones(size(X, 1)/num_clases,1) zeros(size(X, 1)/num_clases,num_clases-3);zeros(size(X, 1)/num_clases,3) ones(size(X, 1)/num_clases,1) zeros(size(X, 1)/num_clases,num_clases-4); zeros(size(X, 1)/num_clases,4) ones(size(X, 1)/num_clases,1) zeros(size(X, 1)/num_clases,num_clases-5);zeros(size(X, 1)/num_clases,5) ones(size(X, 1)/num_clases,1) zeros(size(X, 1)/num_clases,num_clases-6);zeros(size(X, 1)/num_clases,6) ones(size(X, 1)/num_clases,1) zeros(size(X, 1)/num_clases,num_clases-7);zeros(size(X, 1)/num_clases,7) ones(size(X, 1)/num_clases,1) zeros(size(X, 1)/num_clases,num_clases-8); zeros(size(X, 1)/num_clases,8) ones(size(X, 1)/num_clases,1) zeros(size(X, 1)/num_clases,num_clases-9)];
%
m=length(X(:,1));
```

```
% for i=1:m
%     yz(i,y(i))=1;
% end

X0=ones(m,1);
X=[X0 X];
ZS1=X*Theta1';
%PRIMERACAPA
h1 = hipoLog (ZS1);
Z0=ones(m,1);
ZS1=[Z0 h1];
ZS2=ZS1*Theta2';
%segunda capa
h2 = hipoLog (ZS2);

%FUNCIÓN DE COSTE
J=-(1/m)*((sum(sum((yz.*log(h2))+((1-yz).*log(1-h2))))))+(lambda/(2*m))*((sum(sum((The
ta1(:).^2)))+(sum(sum(Theta2(:).^2)))));

%gradiente
%1. errores
error1=zeros(size(Theta1));
error2=zeros(size(Theta2));
for i=1:m
   %a1=X(i,:);
   %error por capas de salida
   %s3=a-y
   %s2=theta*s*dg
   ss3=h2(i,:)-yz(i,:);
   %funcion de activación
   dg=LogDer(ZS1(i,:));
   ss2=(Theta2'*ss3').*dg';
   %formulas de el error acumulado
   error1= error1 + ss2(2:end)*X(i,:);
   error2= error2 + (ss3'*ZS1(i,:));
end
%2. gradiente
%como se calculo en clase
% grad1=(1/2)*error1;
% grad2=(1/2)*error2;
%con regularización, diapositivas
grad1=(1/m)*(error1 +lambda*Theta1);
```

```
grad2=(1/m)*(error2 +lambda*Theta2);
% se pone (:) para que se vuelva vector
grad=[grad1(:); grad2(:)];
end
```

InicializarTheta

```
function Theta0 = InicializarTheta (p, q)
omega=sqrt(6)/sqrt(p+q);
Theta0=rand(q,p)*(2*omega)-omega;
u=Theta0(:,1);
k=ones(length(u),1);
Theta0=[k Theta0];
end
```