UNIVERSIDAD DE GUADALAJARA
Centro Universitario de Ciencias Exactas e Ingenierías
División de Electrónica y Computación

SEMINARIO DE INTELIGENCIA ARTIFICIAL II

PROYECTO 1 – PERCEPTRÓN MULTICAPA

PROFESOR: CIBRIAN DECENA MARÍA ISABEL
SECCIÓN: D02

INTEGRANTES DEL EQUIPO:
PAÚL ESQUEDA GONZÁLES
CARLOS ANDRÉS FERNÁNDEZ JALOMO
SAMUEL GALLEGOS GÓMEZ
DIEGO OLVERA GUTIERRÉZ
ARTURO PLASCENCIA MATA

CODIGO: 210711118
CARRERA: ING. EN COMPUTACIÓN
CICLO: 2017ª

Contenido

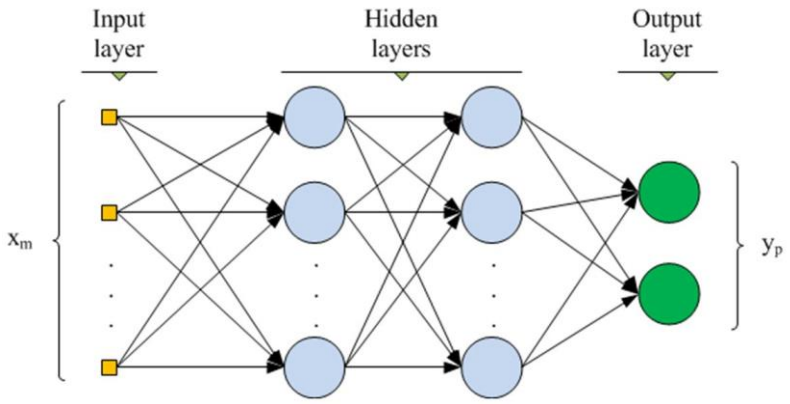
Perceptrón Multicapa (MLP)

Perceptrón de capa oculta, y análisis de patrones

Proyecto 1 del Seminario de Inteligencia Artificial II: Reconocimiento de patrones en imágenes mediante un MLP usando MATLAB®

Introducción

Perceptrón multicapa es un tipo de red neuronal artificial (RNA) que es formada por la conglomeración de múltiples capas, esto le permite resolver problemas que no son linealmente separables, lo cuál es la principal limitación del perceptrón simple. El perceptrón multicapa puede ser totalmente o localmente conectado.



Las capas pueden clasificarse en tres tipos:

- Capa de entrada: Constituida por aquellas neuronas que introducen los patrones de entrada en la red. En estas neuronas no se produce procesamiento.
- Capas ocultas: Formada por aquellas neuronas cuyas entradas provienen de capas anteriores y cuyas salidas pasan a neuronas de capas posteriores.
- Capa de salida: Neuronas cuyas salidas corresponden con las salidas de toda la red.

La propagación hacia atrás (backpropagation), es un algoritmo utilizado en el entrenamiento de estas redes, por ello, el perceptrón multicapa también es conocido como red de retropropagación.

Limitaciones

- El Perceptrón Multicapa no extrapola bien, es decir, si la red se entrena mal o de manera insuficiente, las salidas pueden ser imprecisas.
- La existencia de mínimos locales en la función de error dificulta considerablemente el entrenamiento, pues una vez alcanzado un mínimo el entrenamiento se detiene, aunque no se haya alcanzado la tasa de convergencia fijada

Imágenes como entrenamientos

La capacidad de análisis que proveen los MLP son la causa por la que son usados en ambientes más prácticos y no simplemente basados en datos planos de tablas como los perceptrones simples, eso nos lleva a querer implementarlo en el análisis de contenido multimedia, enfocado a reconocimiento de patrones en imágenes.

Como bien sabemos las imágenes pueden ser transportadas a mapas de datos que se traducen en matrices de valores en la escala RGB (255,255,255), pero por simpleza en el análisis de las imágenes, podemos convertirlas a valores que no sean tan enormes o distantes como la escala de grises, un subconjunto de colores más reducido, ya que el RGB provee 16777216 colores distintos para su uso, mientras que en la escala de grises solo provee 512.

Primero vamos a leer las imágenes en png para que sean almacenados como matrices de datos 8uint en MATLAB:

```
clc
clear all %#ok<*CLALL>
RGB1 = imread('c_0.png');        %Ejemplo inicial Clase "Circulo"
RGB2 = imread('t_0.png');        %Ejemplo inicial Clase "Tache/Cruz"
```

Ahora si procedemos a convertirlo en a escala de grises, esto lo haremos mediante un proceso de procesamiento de imágenes que ya provee MATLAB:

```
I = rgb2gray(RGB1);              %Convierte a matriz B/N
G = rgb2gray(RGB2);              %Convierte a matriz B/N
```

Por motivos de interpretación, al ser matrices de 20x20, MATLAB las detecta como 20 ejemplos de 20 datos cada ejemplo, cuando la totalidad de los 20x20 es una sola imagen, por lo que se tendrán que convertir a un vector, de 1 ejemplo x 400 elementos (la imagen 20x20 en un vector).

Esto mediante el siguiente código:

```
I = double(I(:));                %Convierte a vector analizable (1 sample)
G = double(G(:));                %Convierte a vector analizable (1 sample)
```

Preparación de los datos

Ahora procederemos a preparar los datos de entrenamiento y los de salidas esperadas, así como el resto de las imágenes con las que alimentaremos la red neuronal que estamos creando, primero declarando los contenedores de las entradas y salidas, así como de los datos de asignación de etiquetas:

```
entradas = [I G];
salidas = [0 1];
data = ['Circulo';'Tache  '];
```

Ahora cargamos el resto de los ejemplos, para este proyecto usamos 45 de cada clase, lo que vamos a clasificar son que, si las imágenes contienen “CIRCULOS” o “TACHES” como contenido, y el

clasificador, nos dirá a cuál pertenecen, por lo que entrenamos primero con los casos ideales y así la RNA sepa qué es una clase y qué es otra.

Aquí cargamos primero los 45 ejemplos de círculos:

```
%Lectura de los 45 ejemplos de entrenamiento de los círculos
for i = 1:45
    A = imread(['c_',num2str(i),'.png']);
    A = rgb2gray(A);
    A = double(A(:));
    entradas = [entradas A]; %#ok<*AGROW>
    salidas = [salidas 0];
    data = [data;'Circulo'];
end
```

Y luego los 45 de taches:

```
%Lectura de los 45 ejemplos de entrenamiento de las cruces
for i = 1:45
    A = imread(['t_',num2str(i),'.png']);
    A = rgb2gray(A);
    A = double(A(:));
    entradas = [entradas A];
    salidas = [salidas 1];
    data = [data;'Tache  '];
end
```

Los entrenamientos son como los siguientes:

Y convertimos en celdas el vector de etiquetas:

```
celldata = cellstr(data);
```

Construcción del MLP en MATLAB

Una vez que tenemos ya preparados los datos de entrada, salida y entrenamientos de la red neuronal, procederemos a crearla, antes, podías crear una red neuronal multicapa mediante el comando `newff` (entradas, salidas), sin embargo, desde MATLAB 2015b, se crearon varios tipos de redes neuronales multicapa optimizadas para cierto tipo de uso, por lo que surgieron las siguientes, en este proyecto por el objetivo, se optó por el uso de `patternnet`, que es una red neuronal multicapa optimizada al reconocimiento de patrones:

```
%net = cascadeforwardnet(10,'traingd');
%net = fitnet(10,'traingd','mse');
%net = feedforwardnet(10,'traingd');        %También se puede usar esta
net = patternnet([10,10],'traingd','mse');  %traingd es para usar
gradiente descendiente con backpropagation
```

Dónde el vector [10,10] que le pasamos a la `patternnet`, indica que tendrá 2 capas ocultas, de 10 neuronas internas cada una.

`'traingd'` es la función de entrenamiento y el algoritmo de resolución de errores, siendo esta la que indica "Entrenamiento por Gradiente Descendiente con Backpropagation", tal como lo indica la documentación de la misma:

traingd

Gradient descent backpropagation

Syntax

```
net.trainFcn = 'traingd'
[net,tr] = train(net,...)
```

Description

`traingd` is a network training function that updates weight and bias values according to gradient descent.

Dónde también indica el algoritmo de evaluación que es Backpropagation:

More About

▼ Algorithms

`traingd` can train any network as long as its weight, net input, and transfer functions have derivative functions.

Backpropagation is used to calculate derivatives of performance `perf` with respect to the weight and bias variables `X`. Each variable is adjusted according to gradient descent:

```
dX = lr * dperf/dX
```

Training stops when any of these conditions occurs:

- The maximum number of `epochs` (repetitions) is reached.
- The maximum amount of `time` is exceeded.
- Performance is minimized to the `goal`.
- The performance gradient falls below `min_grad`.
- Validation performance has increased more than `max_fail` times since the last time it decreased (when using validation).

Y en la detección de errores usaremos `'mse'`, ya que el error cuadrático medio incorpora la varianza del estimado, así como el sesgo, es decir la diferencia entre el estimador y el estimado, proporciona un poco más de precisión debido a que al ser una función de riesgo, evalúa la aleatoriedad y trata de disminuirla.

Con esto tendríamos la RNA creada, pero faltaría configurarla.

Configuración del MLP

Para configurar la red neuronal, tenemos que indicar la función de activación, el algoritmo de entrenamiento, así como las funciones de transferencia y la cantidad mínima de iteraciones (Epochs), esto mediante el siguiente código:

```
net.layers{1}.transferFcn = 'tansig';        %Función de transferencia de
la primera capa en tangente
net.layers{2}.transferFcn = 'logsig';        %Función de transferencia de
la primera capa en logaritmo
net.trainParam.epochs = 2000;                %Epochs de entrenamiento
```

Anteriormente ya configuramos el algoritmo de activación que había sido `'traingd'`, ahora usando `transferFcn` dónde esta es la función de transferencia o activación que en estos casos usamos para la primera capa oculta, una función de activación sigmoide hiperbólica tangente, por su capacidad de detectar cambios bruscos en las texturas y la capacidad de reconocimiento de patrones no lineales. Mientras que en la segunda usamos una función sigmoide logarítmica puesto que basa la escala de resultados de 0 a 1, que es lo que nos ayudaría a la pertenencia de ambas clases de clasificación.

Simulación y fase de pruebas

Ahora que ya tenemos lista y configurada nuestro MLP usaremos `train(net,entradas,salidas)` para entrenarla y otras funciones propias de la herramienta de controles neuronales que ofrece MATLAB, como se muestra a continuación:

```
[net,entrenamiento] = train(net,entradas,salidas);
resultados = net(entradas);
errores = gsubtract(salidas,resultados);
rendimiento = perform(net,salidas,resultados);
view(net)
figure, plotroc(salidas,resultados)
```

La instrucción `plotroc(salidas,resultados)` nos permite graficar la curva ROC que se generó en la red al entrenarla, lo cual es muy útil pero que abordaremos más adelante, para indagar en sus beneficios.

Ahora como al inicio, tomaremos una imagen de prueba, distinta a las de entrenamiento para ver si el clasificador nos arroja los resultados correctos o al menos puede identificar si una imagen es o no una de las figuras y con qué certeza lo asegura, las imágenes de prueba serán un intento de circulo, como se ve en la figura 1, y un intento de tache como se ve en la figura 2, para ver qué tan bien puede clasificar elementos no tan perfectos como el entrenamiento:

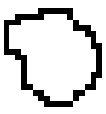
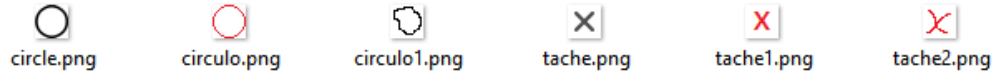


Figura 1. Circulo de prueba

Figura 2. Tache de prueba

Donde las pruebas para ver qué tan bueno es son las siguientes:



Y el código para probarlas serían los siguientes:

```
RGB3 = imread('circulo1.png');     %Prueba de simulación
H = rgb2gray(RGB3);
H = double(H(:));
resultado = sim(net,H);
fprintf('Resultados: %.4f\n',resultado);
if(resultado<0.5)
    disp('Es un circulo');
else
    disp('Es una tache');
end
pesos = net.iw{1,1};
bias = net.b{1};
```

Al ejecutar el código que hemos tenido nos muestra la estructura de la RNA, así como sus opciones de Graficación de elementos, como se ve en la figura 3:

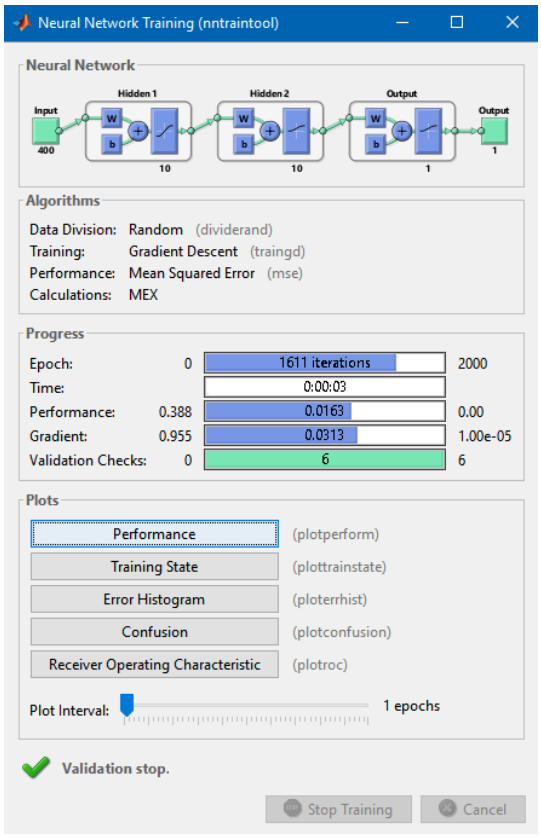


Figura 3. Panel de la herramienta de redes neuronales de MATLAB

Siendo la estructura del MLP, como se aprecia en la figura 4, donde vemos las 2 capas ocultas, la capa de entrada y la capa de salida:

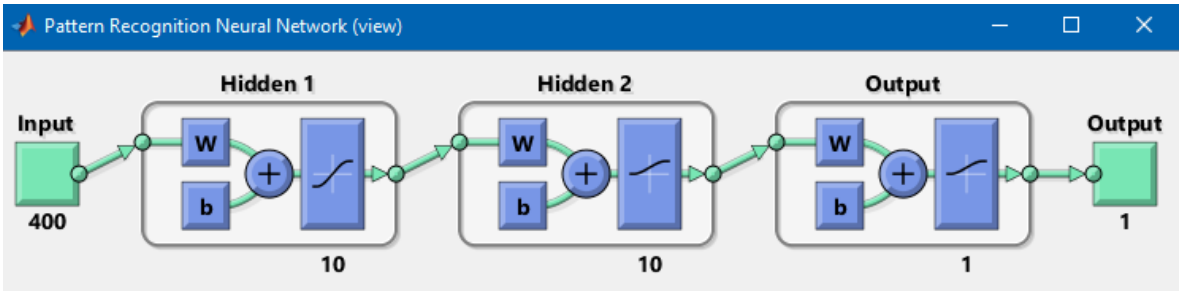


Figura 4. Estructura del MLP

Y el resultado nos dice que cuando usamos el archivo 'circulo1.png' es:

```
Command Window
  Resultados: 0.1452
  Es un circulo
```

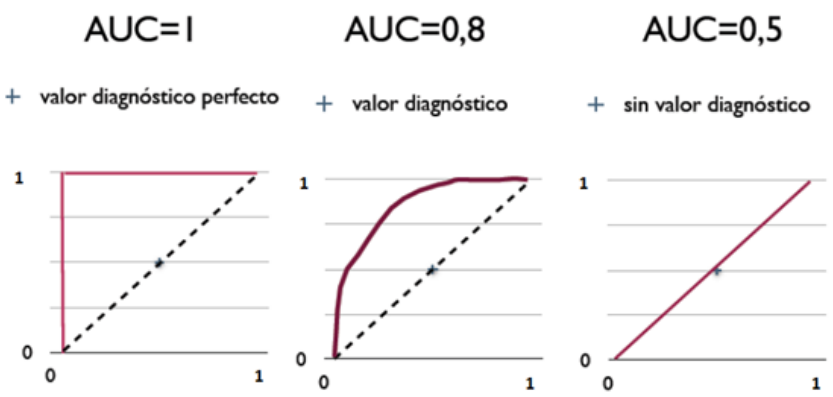
Y cuando usamos el archivo 'tache2.png' es:

```
Command Window
  Resultados: 0.6626
  Es una tache
```

Podemos decir que entrenó bien y clasifica relativamente bien, lo cual quedará probada en la siguiente sección.

Curva ROC

La curva ROC que es acrónimo para Receiver Operating Characteristic o Característica Operativa del Receptor, es una representación gráfica de la sensibilidad frente a la especificidad para un sistema clasificador binario según se varía el umbral de discriminación e indica que tan bueno es un clasificador en nuestro caso, siendo que, si el valor es 1 o se acerca bastante a él, el clasificador es muy bueno o excelente, y mientras más se acerque a 0 es peor:



En nuestro caso, al momento de generar la curva ROC, nos muestra que nuestro clasificador en un 85% de los casos (ya que cada entrenamiento que se da es distinto, y puede que a veces genere un ROC irregular y malo), es excelente y puede clasificar correctamente, como se ve en la figura 5.

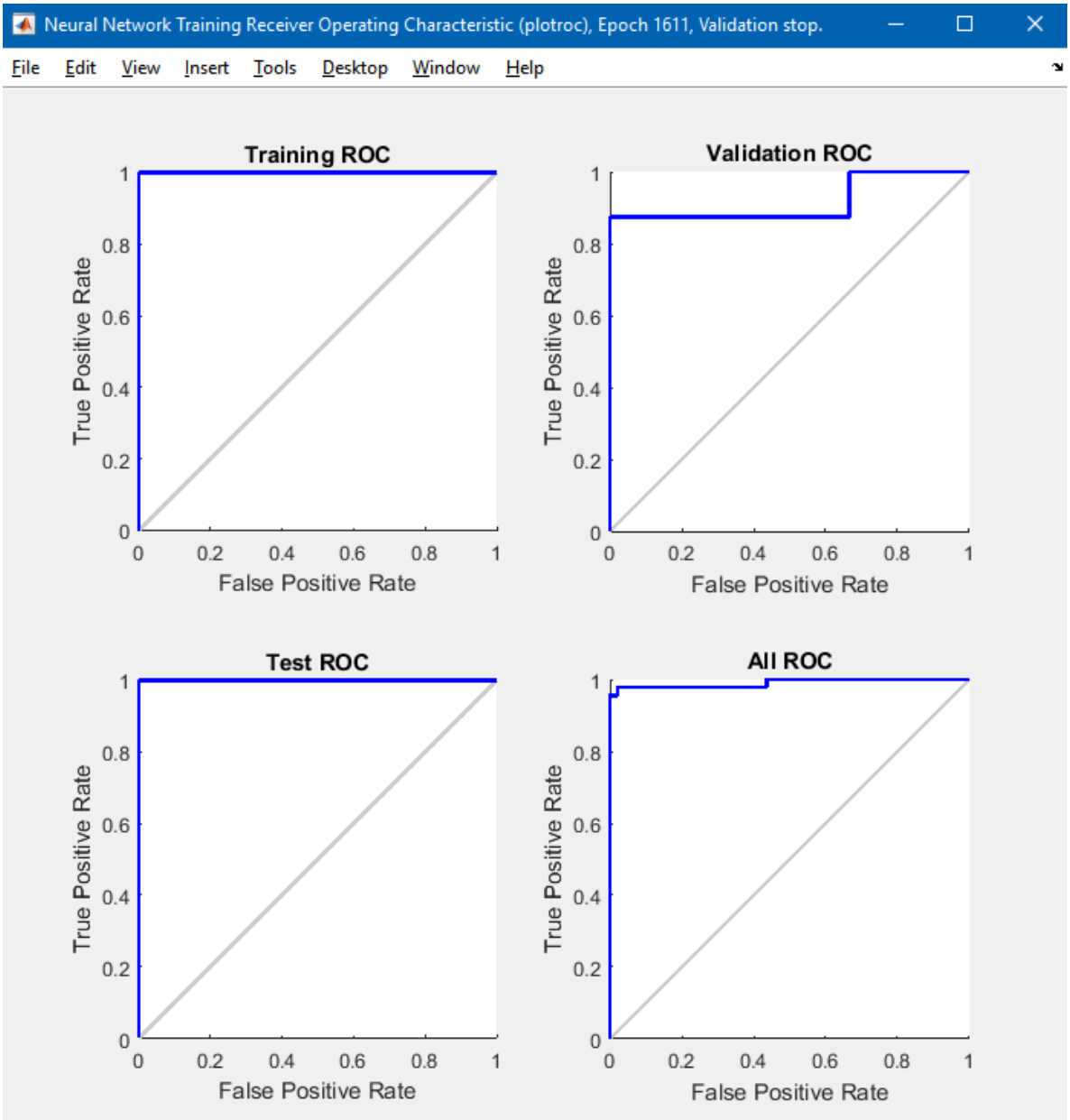


Figura 5. Curva ROC del sistema clasificador de 'O' y 'X'

También MATLAB nos entrega el resultado del ROC, así como de los Verdaderos positivos (TPR), los Falsos positivos (FPR), los Verdaderos Negativos (TNR) y los Falsos Negativos (FNR), así como la sensibilidad y especificidad del sistema, que son dados por las siguientes ecuaciones:

$$\text{Sensibilidad} = \frac{Verdaderos\ Positivos}{Todos\ los\ realmente\ positivos} \qquad \text{Especificidad} = \frac{Verdaderos\ Negativos}{Todos\ los\ realmente\ negativos}$$

Y El código en MATLAB es el siguiente:

```
[c,cm,ind,per] = confusion(salidas,resultados);
FNR = cm(1,2)/sum(cm(:,2));
FPR = cm(2,1)/sum(cm(:,1));
TPR = cm(1,1)/sum(cm(:,1));
TNR = cm(2,2)/sum(cm(:,2));
Sensibilidad = cm(1,1)/sum(cm(1,:));
Especificidad = cm(2,2)/sum(cm(2,:));
[X,Y,T,AUC] = perfcurve(celldata,resultados,'Tache  ');
fprintf('FNR: %.4f | FPR: %.4f | TPR: %.4f | TNR:
%.4f\n',FNR,FPR,TPR,TNR);
fprintf('Sensibilidad: %.4f\n',Sensibilidad);
fprintf('Especificidad: %.4f\n',Especificidad);
fprintf('El valor del AUC es: %.4f\n',AUC);
```

Y los resultados finales siendo:

```
Command Window

  Resultados: 0.1452
  Es un circulo
  FNR: 0.0217 | FPR: 0.0217 | TPR: 0.9783 | TNR: 0.9783
  Sensibilidad: 0.9783
  Especificidad: 0.9783
  El valor del AUC es: 0.9901
```

También podemos observar su matriz de confusión del sistema, la cual nos dice que no tuvo mayor problema, como se puede apreciar en la figura 6.

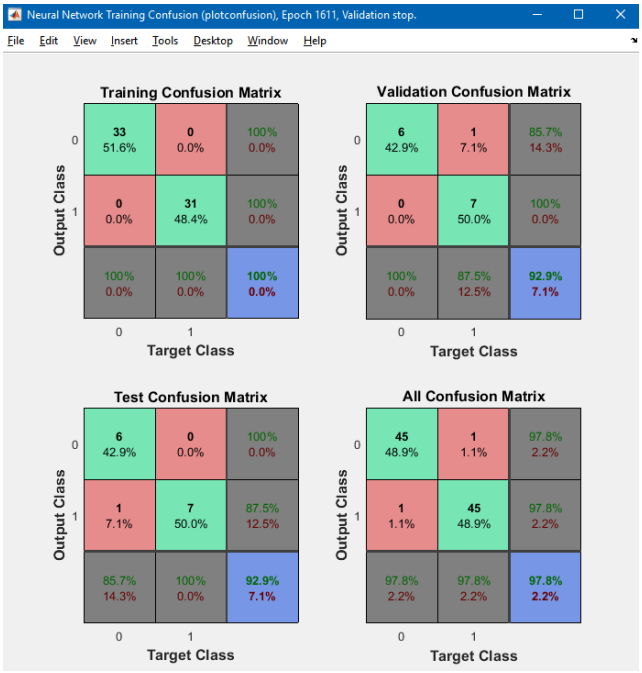


Figura 6. Matrices de Confusión del sistema

Resultados generales

Cuando es un deseable circulo:

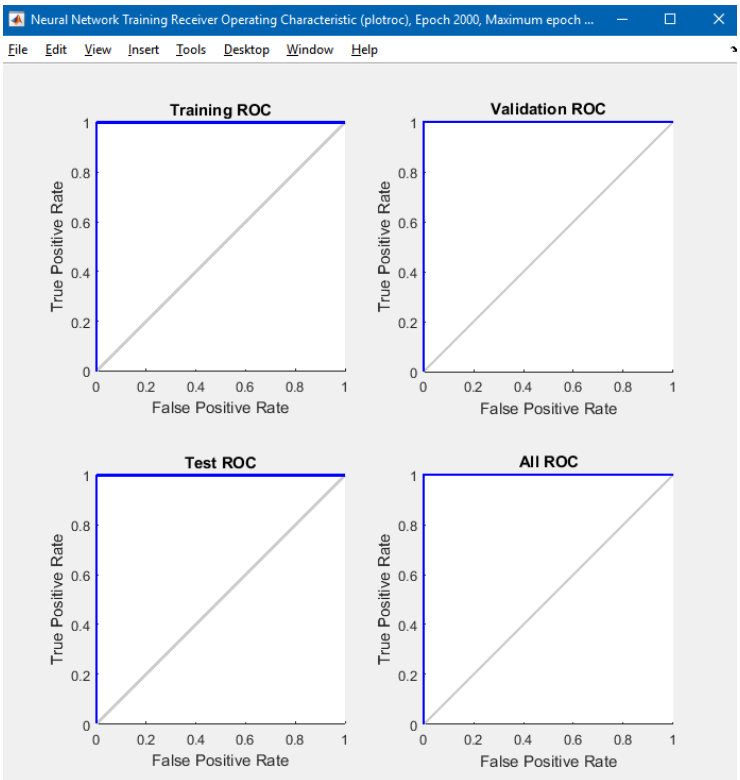
```
Command Window
  Resultados: 0.1452
  Es un circulo
  FNR: 0.0217 | FPR: 0.0217 | TPR: 0.9783 | TNR: 0.9783
  Sensibilidad: 0.9783
  Especificidad: 0.9783
  El valor del AUC es: 0.9901
```

Cuando es una deseable tache:

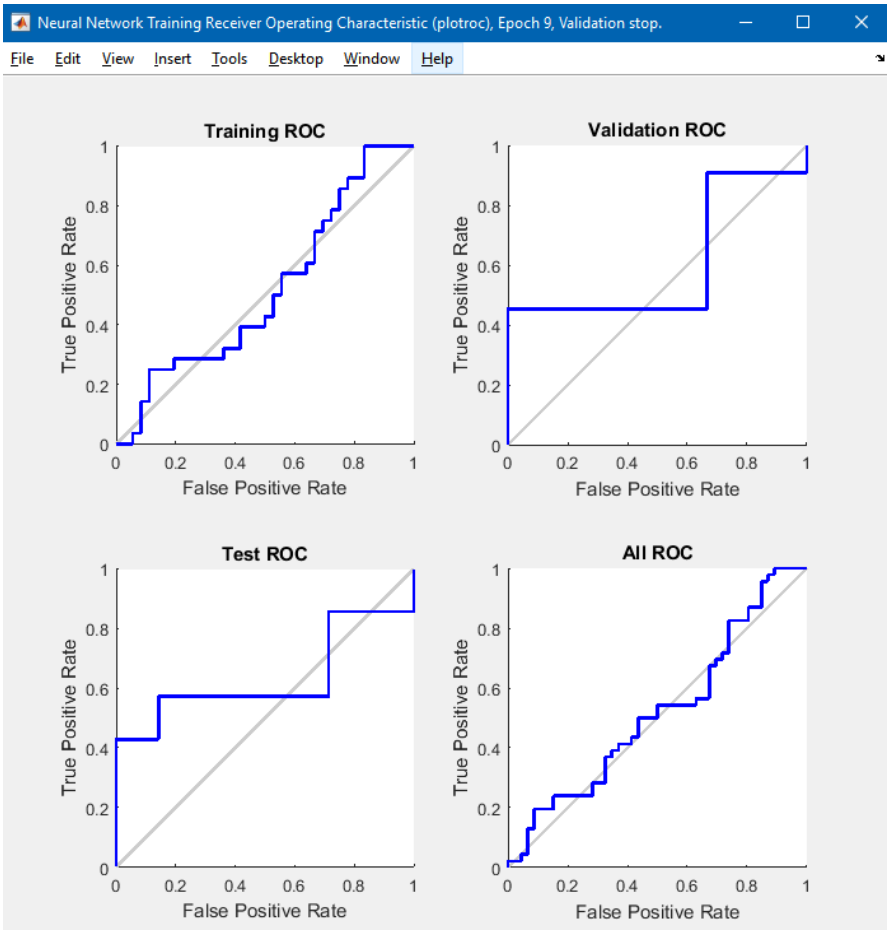
```
Command Window
  Resultados: 0.6626
  Es una tache
  FNR: 0.0800 | FPR: 0.0000 | TPR: 1.0000 | TNR: 0.9200
  Sensibilidad: 0.9130
  Especificidad: 1.0000
  El valor del AUC es: 0.9948
```

Notas

Al depender del entrenamiento del momento y cómo la RNA gestione su conocimiento, puede que aparezcan casos perfectos como este:



Donde en el anterior el AUC era un perfecto 1, pero también pueden existir terribles como este:



Dónde el AUC es de 0.52.

Código en MATLAB

```
%-------------------------------------------------------
%   LECTURA DE DATOS DE ENTRENAMIENTO
%-------------------------------------------------------
clc
clear all %#ok<*CLALL>
RGB1 = imread('c_0.png');       %Ejemplo inicial Clase "Circulo"
RGB2 = imread('t_0.png');       %Ejemplo inicial Clase "Tache/Cruz"
I = rgb2gray(RGB1);             %Convierte a matriz B/N
G = rgb2gray(RGB2);             %Convierte a matriz B/N
I = double(I(:));               %Convierte a vector analizable (1 sample)
G = double(G(:));               %Convierte a vector analizable (1 sample)
entradas = [I G];
salidas = [0 1];
data = ['Circulo';'Tache  '];
%Lectura de los 45 ejemplos de entrenamiento de los círculos
for i = 1:45
    A = imread(['c_',num2str(i),'.png']);
    A = rgb2gray(A);
    A = double(A(:));
    entradas = [entradas A]; %#ok<*AGROW>
    salidas = [salidas 0];
    data = [data;'Circulo'];
end
%Lectura de los 45 ejemplos de entrenamiento de las cruces
for i = 1:45
    A = imread(['t_',num2str(i),'.png']);
    A = rgb2gray(A);
    A = double(A(:));
    entradas = [entradas A];
    salidas = [salidas 1];
    data = [data;'Tache  '];
end
celldata = cellstr(data);

%-------------------------------------------------------
%   CREACIÓN Y CONFIGURACIÓN DE LA RNA
%-------------------------------------------------------
%Declaración de la red neuronal multicapa
%net = cascadeforwardnet(10,'traingd');
%net = fitnet(10,'traingd','mse');
%net = feedforwardnet(10,'traingd');        %También se puede usar esta
net = patternnet([10,10],'traingd','mse');  %traingd es para usar
gradiente descendiente con backpropagation
net.layers{1}.transferFcn = 'tansig';       %Función de transferencia de
la primera capa en tangente
net.layers{2}.transferFcn = 'logsig';       %Función de transferencia de
la primera capa en logaritmo
net.trainParam.epochs = 2000;               %Epochs de entrenamiento

[net,entrenamiento] = train(net,entradas,salidas);
resultados = net(entradas);
errores = gsubtract(salidas,resultados);
rendimiento = perform(net,salidas,resultados);
```

```
view(net)
figure, plotroc(salidas,resultados)

%-------------------------------------------------------
%   SIMULACIÓN, CLASIFICACIÓN Y RESULTADOS
%-------------------------------------------------------
RGB3 = imread('circulo.png');    %Prueba de simulación
H = rgb2gray(RGB3);
H = double(H(:));
resultado = sim(net,H);
fprintf('Resultados: %.4f\n',resultado);
if(resultado<0.5)
    disp('Es un circulo');
else
    disp('Es una tache');
end
pesos = net.iw{1,1};
bias = net.b{1};

%-------------------------------------------------------
%   ESPECIFICIDAD, SENSIBILIDAD Y ROC
%-------------------------------------------------------
[c,cm,ind,per] = confusion(salidas,resultados);
FNR = cm(1,2)/sum(cm(:,2));
FPR = cm(2,1)/sum(cm(:,1));
TPR = cm(1,1)/sum(cm(:,1));
TNR = cm(2,2)/sum(cm(:,2));
Sensibilidad = cm(1,1)/sum(cm(1,:));
Especificidad = cm(2,2)/sum(cm(2,:));
[X,Y,T,AUC] = perfcurve(celldata,resultados,'Tache  ');
fprintf('FNR: %.4f | FPR: %.4f | TPR: %.4f | TNR:
%.4f\n',FNR,FPR,TPR,TNR);
fprintf('Sensibilidad: %.4f\n',Sensibilidad);
fprintf('Especificidad: %.4f\n',Especificidad);
fprintf('El valor del AUC es: %.4f\n',AUC);
```

Conclusiones

Al utilizar el perceptrón multicapa se puede obtener un mejor análisis de los datos, sin embargo, una deficiencia que observo es el entrenamiento dependiendo también de como este programado el Perceptrón puede necesitar más entrenamientos. Al estar basadas, obviamente, en las neuronas biológicas también estas requieren de varios aprendizajes para un mejor razonamiento. El MLP puede resultar muy útil en las ciencias para analizar patrones, un ejemplo analizar la migración de los bivalvos en las costas del lado de océano pacifico y detectar que especie es.

-Esqueda González, Paúl

El uso, conocimiento e implementación de los MLP, hacen que uno comprende en mejor medida cómo es que las RNA pueden aplicarse en la vida real, ya que no solo son prácticas con datos de prueba, sino que son análisis reales de imágenes, cosa que puede ser aplicada a N cantidad de proyectos, aunque siento que va enfocado más a las áreas biomédicas o de análisis de Big Data o Minería de datos, el algoritmo de Backpropagation me pareció increíble de usar y el cómo provee una nueva gestión del conocimiento a la red neuronal.

-Fernández Jalomo, Carlos Andrés

Un uso de un perceptrón multicapa es para el reconocimiento óptico. El perceptrón que se creó en este proyecto fue entrenado con el algoritmo de Backpropagation, analizando como entrada muchas imágenes de 20x20, con la intención de que luego de ser entrenado, sea capaz de reconocer patrones y dar como respuesta un resultado confiable.

-Gallegos Gómez, Samuel**

Hemos trabajo desde el principio del semestre con perceptrones de una sola capa, pero en ciertos problemas que resuelve la inteligencia artificial suele no ser suficientes para lograr lo esperado, por eso existen los multi nivel o también conocidos multicapas, que con llevan mejores en los algoritmos de aprendizaje y la regla de Backpropagation, este tipo de redes neuronales facilitan enormemente el procesamiento de imágenes como lo mostrado con los caracteres ‘X’ y ‘O’.

-Olvera Gutiérrez, Diego

En este proyecto se tenía que analizar como entrada imágenes de 20x20 que Matlab lo interpretaría como un vector, por su nivel de complejidad y tipo de análisis se trabajó con el perceptrón multicapa con el algoritmo Backpropagation, al hacer este tipo de trabajos nos damos una idea de lo útiles y eficaces que pueden ser los perceptrones multicapa a la hora de detectar patrones y del porque son utilizados para investigaciones médicas, aparte de ser útiles nos pueden dar información acerca de que tan confiable es su misma evaluación lo que es importante en este tipo de áreas.

-Plascencia Mata, Arturo