



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



## SEMINARIO DE INTELIGENCIA ARTIFICIAL II

### PRÁCTICA 1 – PERCEPTRÓN SIMPLE

**PROFESOR:** CIBRIAN DECENA MARÍA ISABEL  
**SECCIÓN:** D02

**ALUMNO:** CARLOS ANDRÉS FERNÁNDEZ JALOMO  
**CODIGO:** 210711118  
**CARRERA:** ING. EN COMPUTACIÓN  
**CICLO:** 2017<sup>a</sup>

## Contenido

Introducción .....	3
Aprendizaje y usos .....	3
Ejercicio 1 .....	4
Tabla de verdad .....	4
Construcción del perceptrón en MATLAB .....	4
Poniendo a prueba el perceptrón .....	7
Pesos finales y BIAS .....	9
Código en MATLAB .....	9
Ejercicio 2 .....	10
Tabla de datos de entrada .....	10
Construcción del perceptrón en MATLAB .....	10
Poniendo a prueba el perceptrón .....	13
Pesos finales y BIAS .....	14
Código en MATLAB .....	15

# Perceptrón Simple

Representación y entrenamiento mediante MATLAB®

*Práctica 1 del Seminario de Inteligencia Artificial II: Perceptrón Simple usando MATLAB®*

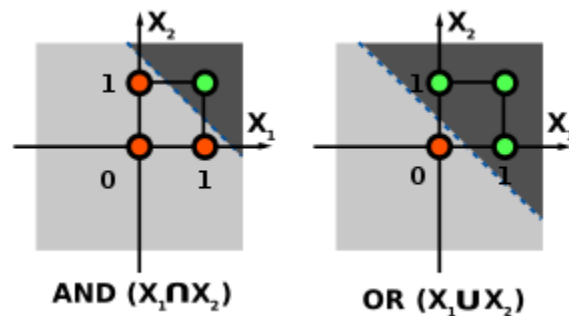
## Introducción

Un Perceptrón dentro del campo de las redes neuronales tiene dos conceptualizaciones, la primera la red artificial que fue desarrollada por Frank Rosenblatt, entendida como una neurona artificial o unidad básica de inferencia como modelado de discriminador lineal, que genera un criterio de selección de un sub-grupo. Y la otra es la teoría perceptiva sobre el auto-aprendizaje de las máquinas establecida igualmente en la teoría de Rosenblatt. El Perceptrón puede combinarse con otros perceptrones o tipos de neuronas artificiales, para formar redes neuronales más complejas, pero más eficientes.

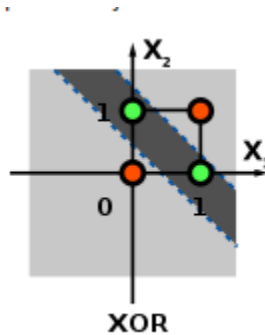
Es la representación del modelo biológico más simple, el modelo matemático de la neurona humana es el Perceptrón, la neurona tiene canales de entrada llamados dendrita y el canal de salida llamado axón, las dendritas operan como sensores, que derivan la información al cuerpo de la neurona y que mediante la reacción sináptica envía respuestas al cerebro; una neurona sola y aislada carece de razón de existencia, pero se torna interesante cuando se asocia con más y más neuronas, formando una red, el principio se sigue en el modelo matemático, donde tanto las neuronas artificiales como naturales, tienen  $N$  entradas y generan 1 salida, salida que a su vez puede pertenecer al subgrupo de  $N$  entradas de otra neurona. El perceptrón usa una matriz para representar las redes neuronales y es un discriminador terciario que traza su entrada  $x$  (un vector binario) a un único valor de salida  $f(x)$  (un solo valor binario) a través de dicha matriz.

## Aprendizaje y usos

Lo más interesante es la capacidad de separar linealmente los problemas con tal de clasificarlos de mejor manera, esto es que, en un plano coordenado, el perceptrón puede generar una línea divisoria entre los casos para saber a qué subgrupo de salida pertenece la combinación recibida por sus entradas, por ejemplo: considerando las funciones AND y OR lógicas, estas son linealmente separables y, por tanto, potencialmente aprendidas por un perceptrón:



Sin embargo, hay otras como la función XOR, que no puede ser aprendida por un perceptrón puesto que requiere al menos dos líneas para separar los casos (0,1):



Pero si puede ser aprendida por 2 perceptrones, y con una capa adicional interna para procesarlo.

### Ejercicio 1

Dada la tabla de verdad de la función lógica  $A \cup (B \cap C)$  generar el perceptrón que ubique los resultados y pueda clasificar la salida correctamente siendo:

La **conjunción lógica** ( $\cap$ ) y la **disyunción lógica** ( $\cup$ ) los elementos a trabajar

Tabla de verdad

$A$	$B$	$C$	$(B \cap C)$	$A \cup (B \cap C)$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

### Construcción del perceptrón en MATLAB

Primero modelamos la tabla de verdad en una matriz y un vector en MATLAB:

```
1 -   entradas = [0 0 0 0 1 1 1 1;
2           0 0 1 1 0 0 1 1;
3           0 1 0 1 0 1 0 1];
4 -   sal_des = [0 0 0 1 1 1 1 1];
```

Lo cual, nos generaría los elementos de las 3 entradas y 1 salida del perceptrón, para entrenarlo. Posteriormente debemos modelar gráficamente las entradas y las salidas con el comando `plotpv`, tal y como se muestra en la figura 1.

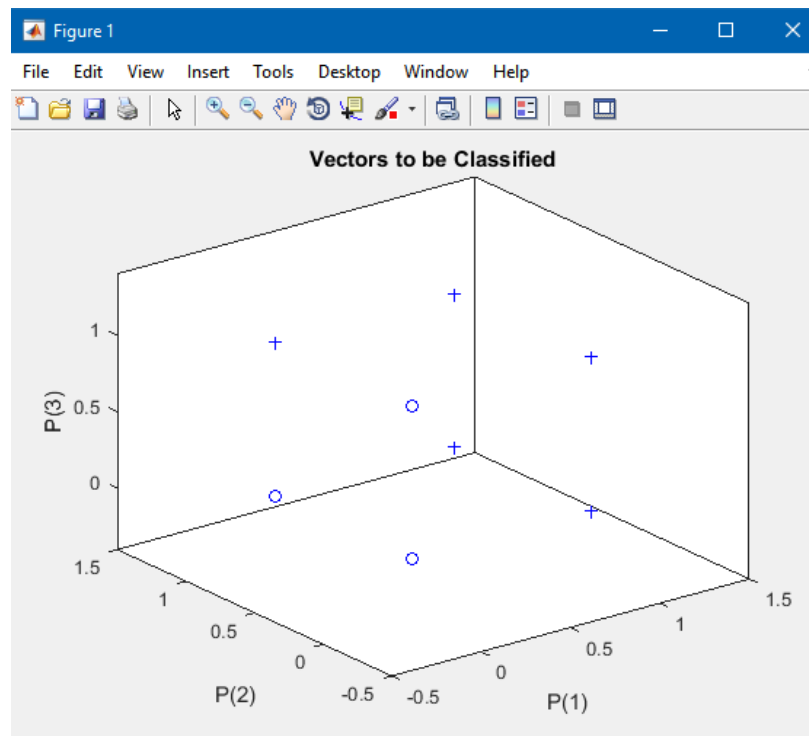


Figura 1 Uso de `plotpv` para graficar las entradas y salidas

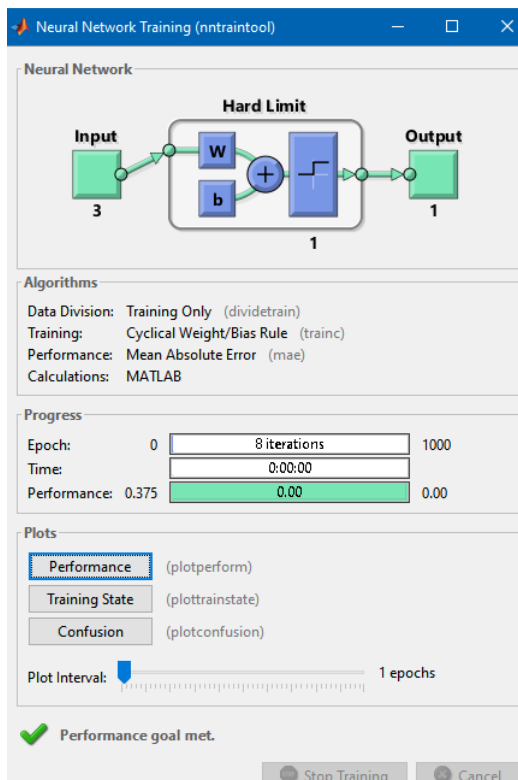


Figura 2. Datos del Perceptrón

Ahora procederemos a crear el perceptrón, con el comando `perceptron` y almacenándolo en una variable para guardar el objeto perceptrón para su posterior uso, y asignándole una vez más la función `train(net,entradas,sal_des)` que será la que entrene al perceptrón con nuestras entradas y salidas deseadas, tal como se ve a continuación:

```
plotpv(entradas,sal_des);
net = perceptron;
net = train(net,entradas,sal_des);
plotpc(net.IW{1},net.b{1});
```

La instrucción `train` también sirve para visualizar la red neuronal y cómo fue entrenada, así como otros datos relevantes que MATLAB considera que pueden ser útiles, tal como se muestra en la figura 2.

Dónde `plotpc(net.IW{1},net.b{1})` sirve para graficar los resultados del perceptrón, es decir la línea que considera es la óptima para clasificar, y como estamos usando un plano 3D, su división es más una pared que una línea como podemos ver en la figura 3.

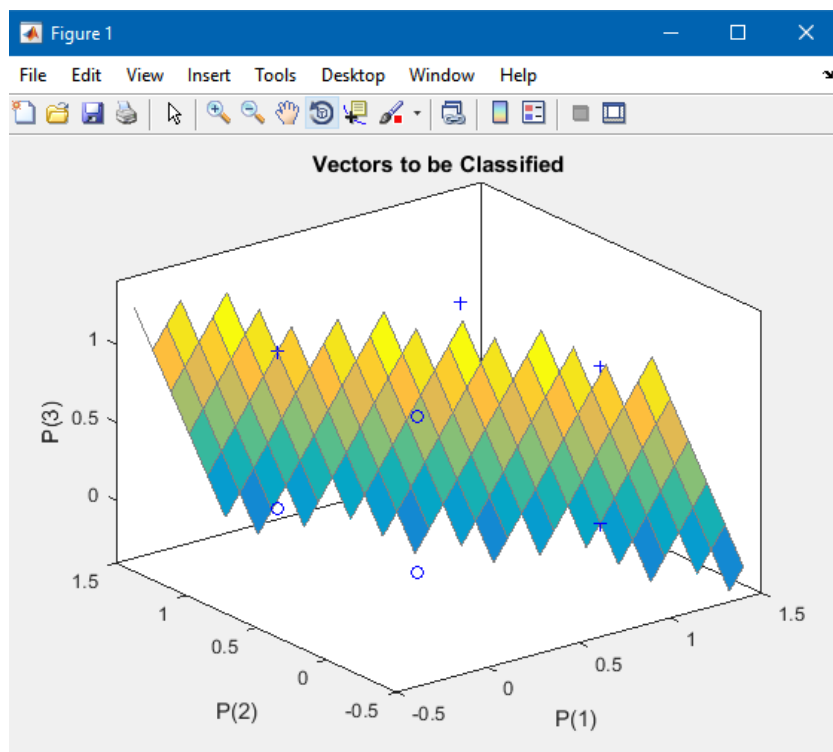


Figura 3. Graficación de los resultados del perceptrón y la línea de separación de salidas

Ahora si usamos la herramienta de giro, podemos movernos en el cubo y ver como la línea separa los datos y crea esa pared de separación, creada por el perceptrón, como se ve en la figura 4.

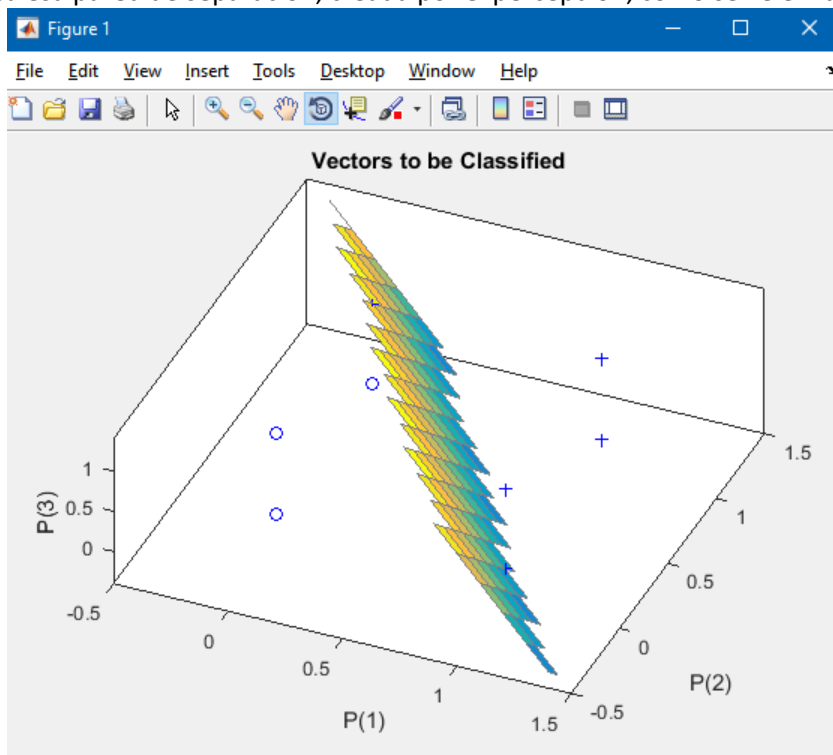


Figura 4. Graficación después de girar, mostrando su posición y figura 3D en el plano

### Poniendo a prueba el perceptrón

Ahora que ya está entrenado nuestro perceptrón, este debería poder clasificar los datos con base en la línea de separación y su ubicación en el plano 3D. Para esto le daremos datos de entrada distintos a los que fueron los entrenamientos y veremos su comportamiento en este plano, como hicimos anteriormente creamos una matriz de datos de entrada:

```
prueba = [0.9 0.1 0.4 0.5;
          0.9 0.2 0.3 0.7;
          0.8 0.1 0.5 0.6];
```

Pero ahora no daremos datos de salida porque le daremos la prueba al perceptrón y este nos arrojará la salida esperada, dependiendo de su entrenamiento, mediante el comando sim:

```
salida2 = sim(net,prueba);
```

Una vez que tengamos los datos de la simulación, procederemos a graficarlos y ver los resultados:

```
figure();
plotpv(prueba,salida2);
hold on; plotpc(net.IW{1},net.b{1});
```

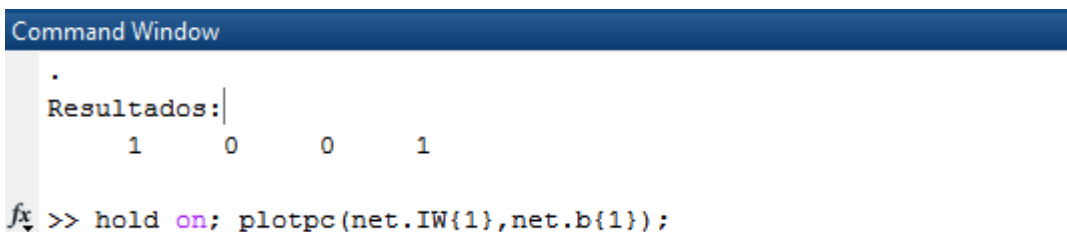
Se esperarían estos datos de salida:

A	B	C	Salida esperada
0.9	0.9	0.8	1
0.1	0.2	0.1	0
0.4	0.3	0.5	0
0.5	0.7	0.6	1

Ahora al ejecutar todo el código, obtendríamos los resultados, sin embargo para facilitarnos esto, usaremos el comando `disp(<>)` que permite mostrar los valores de una variable, en este caso queremos saber la de `salida2`, que son los resultados de clasificación del perceptrón:

```
disp(salida2);
```

Lo que nos daría esto:



```
Command Window
.
Resultados:
      1      0      0      1

fx >> hold on; plotpc(net.IW{1},net.b{1});
```

Confirmando que el perceptrón entrenó bien y pudo clasificar adecuadamente los datos de entrada.

Ahora lo graficaremos para poder visualizar su movimiento y ubicación en el plano, esto en la figura 5 y girado espacialmente en la figura 6.

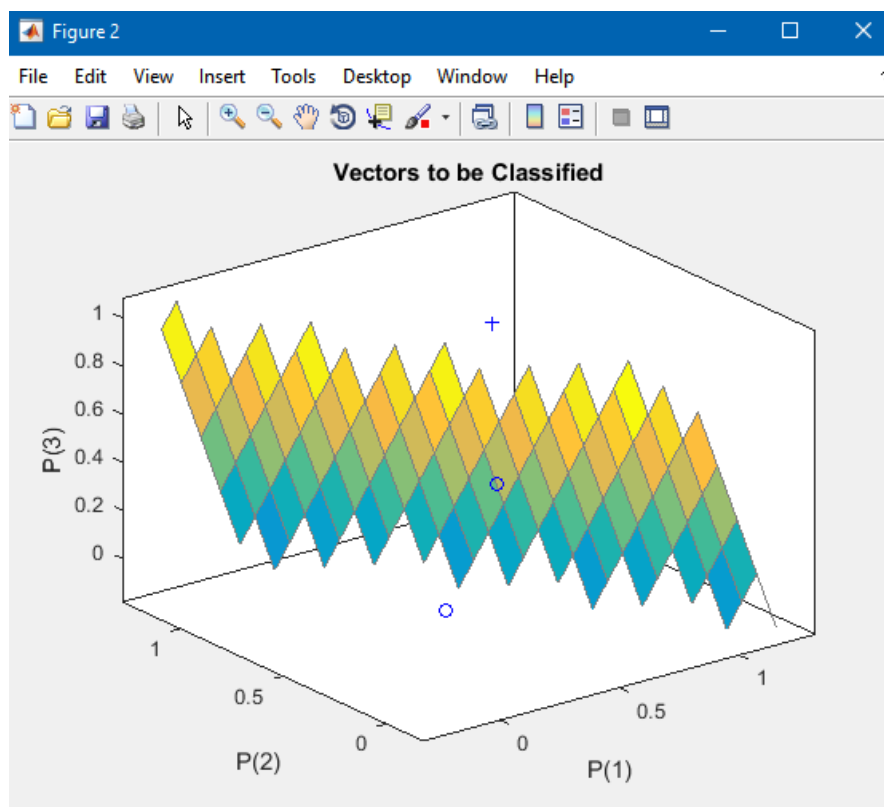


Figura 5. Graficación de los resultados, separados linealmente por el perceptrón y clasificados

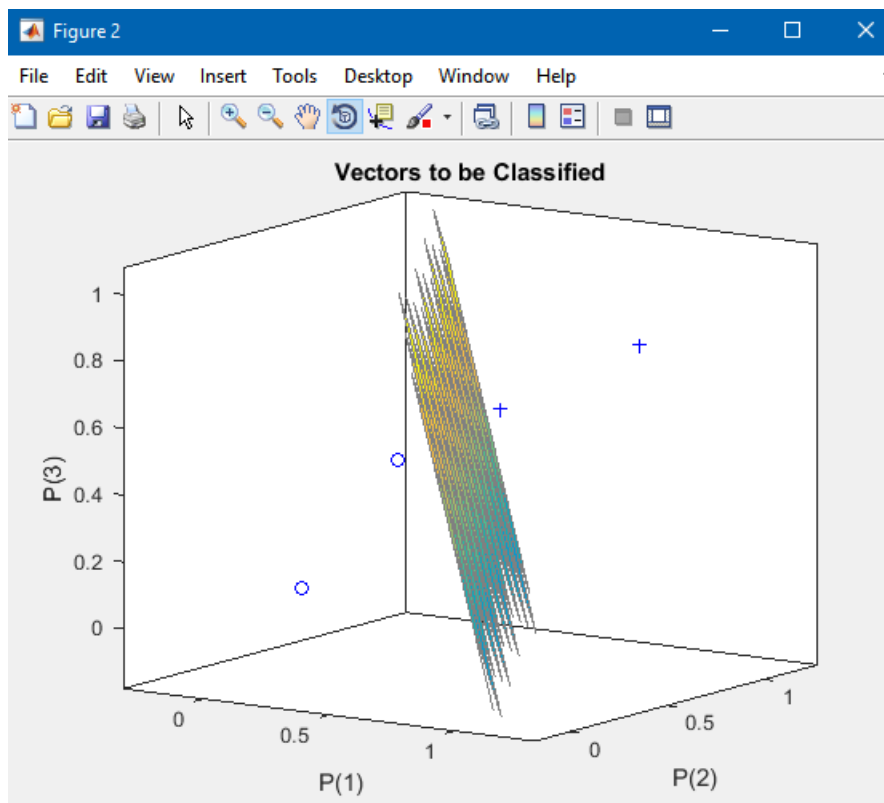


Figura 6. Resultados girados, visualizando su separación por grupos



## Pesos finales y BIAS

```

Resultados:
      1      0      0      1

Pesos:
      3      2      1
|
Bias:
     -3

```

## Código en MATLAB

```

entradas = [0 0 0 0 1 1 1 1;
            0 0 1 1 0 0 1 1;
            0 1 0 1 0 1 0 1];
sal_des = [0 0 0 1 1 1 1 1];

plotpv(entradas,sal_des);
net = perceptron;
net = train(net,entradas,sal_des);
plotpc(net.IW{1},net.b{1});

prueba = [0.9 0.1 0.4 0.5;
          0.9 0.2 0.3 0.7;
          0.8 0.1 0.5 0.6];
salida2 = sim(net,prueba);
figure();
plotpv(prueba,salida2);
hold on; plotpc(net.IW{1},net.b{1});
disp('Resultados:');
disp(salida2);
pesos = net.iw{1,1};
bias = net.b{1};
disp('Pesos: ');
disp(pesos);
disp('Bias: ');
disp(bias);

```

## Ejercicio 2

Dados los siguientes datos, fabricar un perceptrón de clasificación de 4 grupos.

Tabla de datos de entrada

A	B	Salida
0.5	1.4	Grupo 1
0.7	1.8	Grupo 1
0.8	1.6	Grupo 1

A	B	Salida
1.5	0.8	Grupo 2
2.0	1.0	Grupo 2

A	B	Salida
0.3	0.5	Grupo 3
0.0	0.2	Grupo 3
-0.3	0.8	Grupo 3

A	B	Salida
-0.5	-1.5	Grupo 4
-1.5	-2.2	Grupo 4

## Construcción del perceptrón en MATLAB

Primero modelamos las tablas en 4 matrices en MATLAB:

```

2 - Grupo1 = [0.5 0.7 0.8;
3           1.4 1.8 1.6];
4 - Grupo2 = [1.5 2.0;
5           0.8 1.0];
6 - Grupo3 = [0.3 0.0 -0.3;
7           0.5 0.2 0.8];
8 - Grupo4 = [-0.5 -1.5;
9           -1.5 -2.2];

```

Lo cual, nos generaría los elementos entrada del perceptrón, para entrenarlo. Posteriormente debemos modelar gráficamente las entradas y las salidas con el comando `plottpv`, y añadiendo los siguientes comandos para modelarlo más visualmente y estableciendo cada grupo con un símbolo y color particular, quedando tal y como se muestra en la figura 7.

```

plot(Grupo1(1,:), Grupo1(2,:), 'bo')
hold on; grid on

```

```

plot(Grupo2(1,:),Grupo2(2,:), 'r*')
plot(Grupo3(1,:),Grupo3(2,:), 'g+')
plot(Grupo4(1,:),Grupo4(2,:), 'mx')
text(0,1.5, 'Grupo 1')
text(1.5,0, 'Grupo 2')
text(-1,0.5, 'Grupo 3')
text(0,-1, 'Grupo 4')
title('Elementos')
xlabel('P(X)')
ylabel('P(Y)')

```

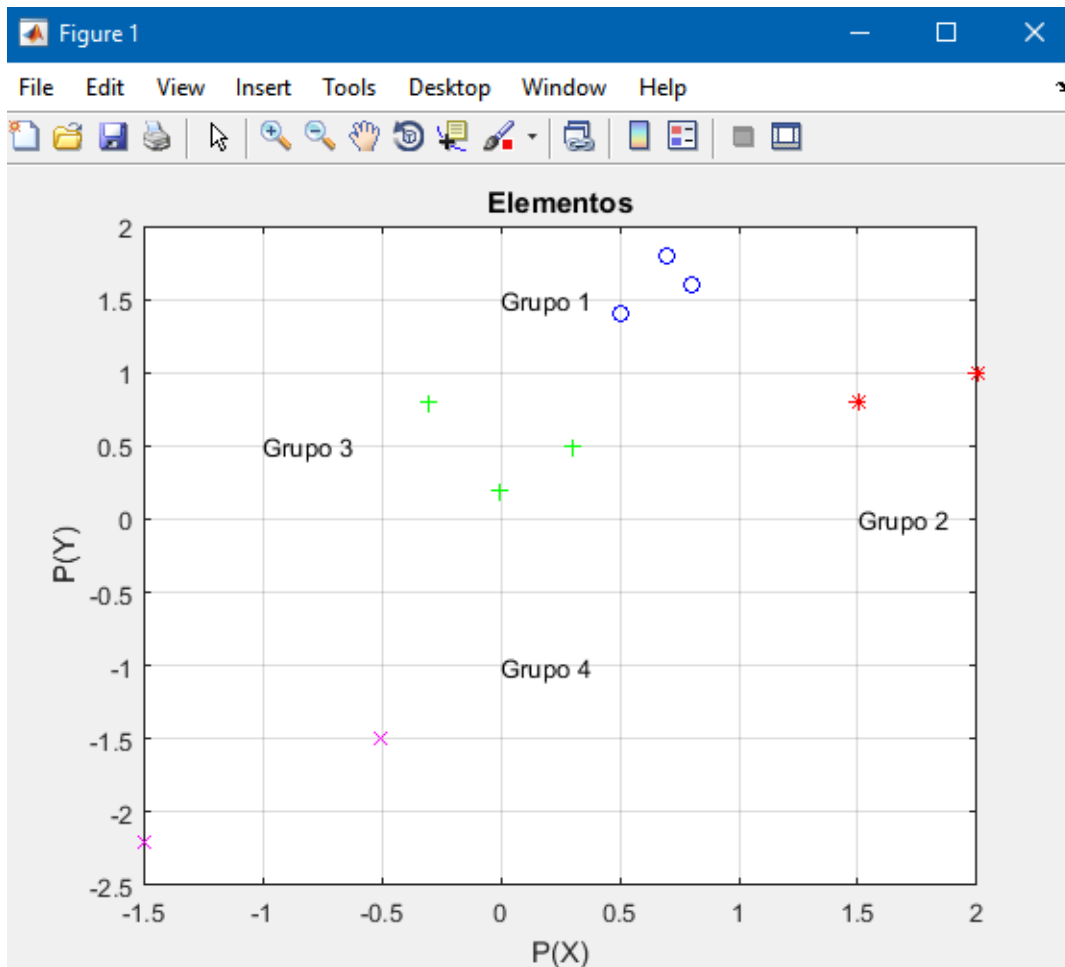


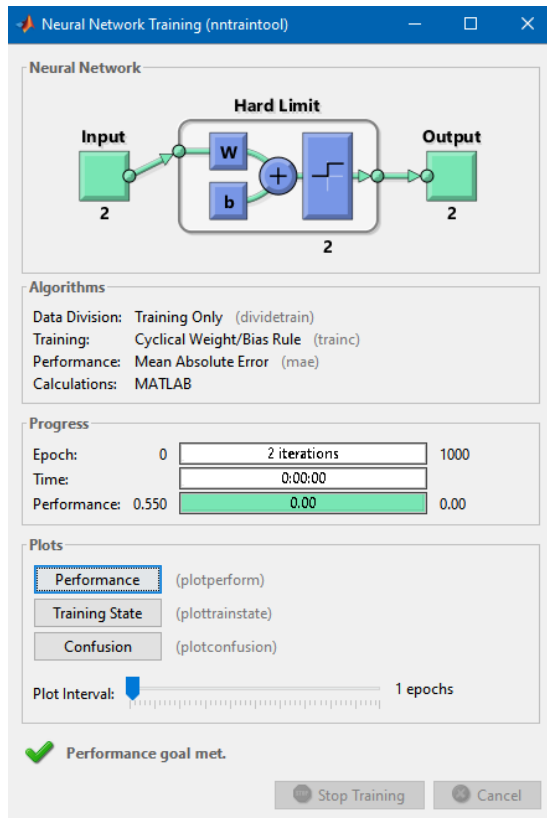
Figura 7. Elemento de los 4 grupos en el plano coordinado

A diferencia del caso anterior, en este tendremos que preparar los datos de entrada del perceptrón para que sean válidos ante el procesamiento de este, primero indicamos el vector de salida de cada grupo ante el perceptrón:

```

a = [0 0]';
b = [0 1]';
c = [1 0]';
d = [1 1]';
P = [Grupo1 Grupo2 Grupo3 Grupo4];
T = [repmat(a,1,length(Grupo1)) repmat(b,1,length(Grupo2)) ...
     repmat(c,1,length(Grupo3)) repmat(d,1,length(Grupo4)) ];

```



Una vez hecho esto, procederemos a crear el perceptrón, con el comando `perceptron` y almacenándolo en una variable para guardar el objeto perceptrón para su posterior uso, y asignándole una vez más la función `train(net,entradas,sal_des)` que será la que entrene al perceptrón con nuestras entradas y salidas deseadas, y usamos el comando `plotpc` para visualizar la estructura del perceptrón, tal como se ve en la figura 8.

Como podemos apreciar el perceptrón ha terminado de analizar las entradas y determinó que las líneas de separación son las mostradas en la figura 9, lo que permite clasificar puntos siguientes con base en esto y en los entrenamientos que ha realizado.

Figura 8. Estructura interna del perceptrón con sus 2 neuronas intermedias y salidas de 2 elementos (00,01,10,11)

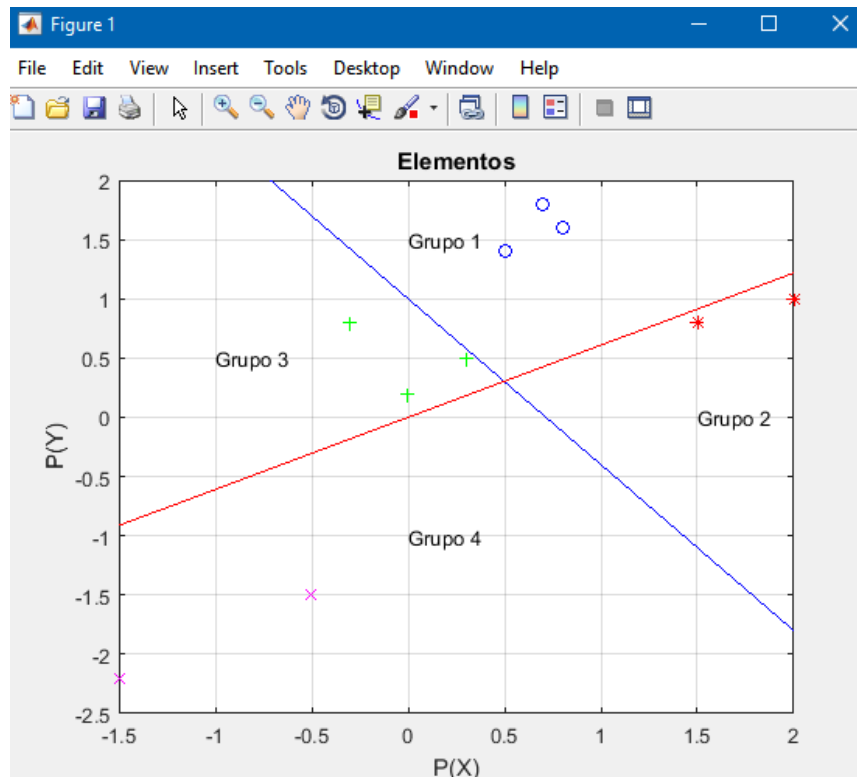


Figura 9. Líneas de clasificación del perceptrón para los 4 grupos

### Poniendo a prueba el perceptrón

Ahora que ya está entrenado nuestro perceptrón, este debería poder clasificar los datos con base en la línea de separación y su ubicación en el plano 2D. Para esto le daremos datos de entrada distintos a los que fueron los entrenamientos y veremos su comportamiento en este plano, como hicimos anteriormente creamos una matriz de datos de entrada:

```
p = [0.5 0.43 0.5 0.58;
     0.36 0.3 0.26 0.28];
```

Pero ahora no daremos datos de salida porque le daremos la prueba al perceptrón y este nos arrojará la salida esperada, dependiendo de su entrenamiento, mediante el comando `sim`:

```
salida2 = sim(net,p);
```

Una vez que tengamos los datos de la simulación, procederemos obtener información de la red neuronal, a graficarlos y ver los resultados:

```
disp(salida2)
pesos = net.iw{1,1};
bias = net.b{1};
disp('Pesos: ');
disp(pesos);
disp('Bias: ');
disp(bias);
figure('Name','Resultados de prueba','NumberTitle','off')
plotpv(p,salida2);
title('Resultados de prueba')
xlabel('P(X)')
ylabel('P(Y)')
hold on; plotpc(pesos,bias);
```

Se esperarían estos datos de salida:

A	B	Salida
0.5	0.36	Grupo 1 (00)
0.43	0.3	Grupo 3 (10)
0.5	0.26	Grupo 4 (11)
0.58	0.28	Grupo 2 (01)

Ahora al ejecutar todo el código, obtendríamos los resultados, sin embargo para facilitarnos esto, usaremos el comando `disp(<>)` que son los resultados de clasificación del perceptrón:

```
disp(salida2);
```

Lo que nos daría esto:

Command Window				
<b>Resultados:</b>				
0	1	1	0	
0	0	1	1	

Confirmando que el perceptrón entrenó bien y pudo clasificar adecuadamente los datos de entrada.

Ahora lo graficaremos para poder visualizar su ubicación en el plano, esto en la figura 10.

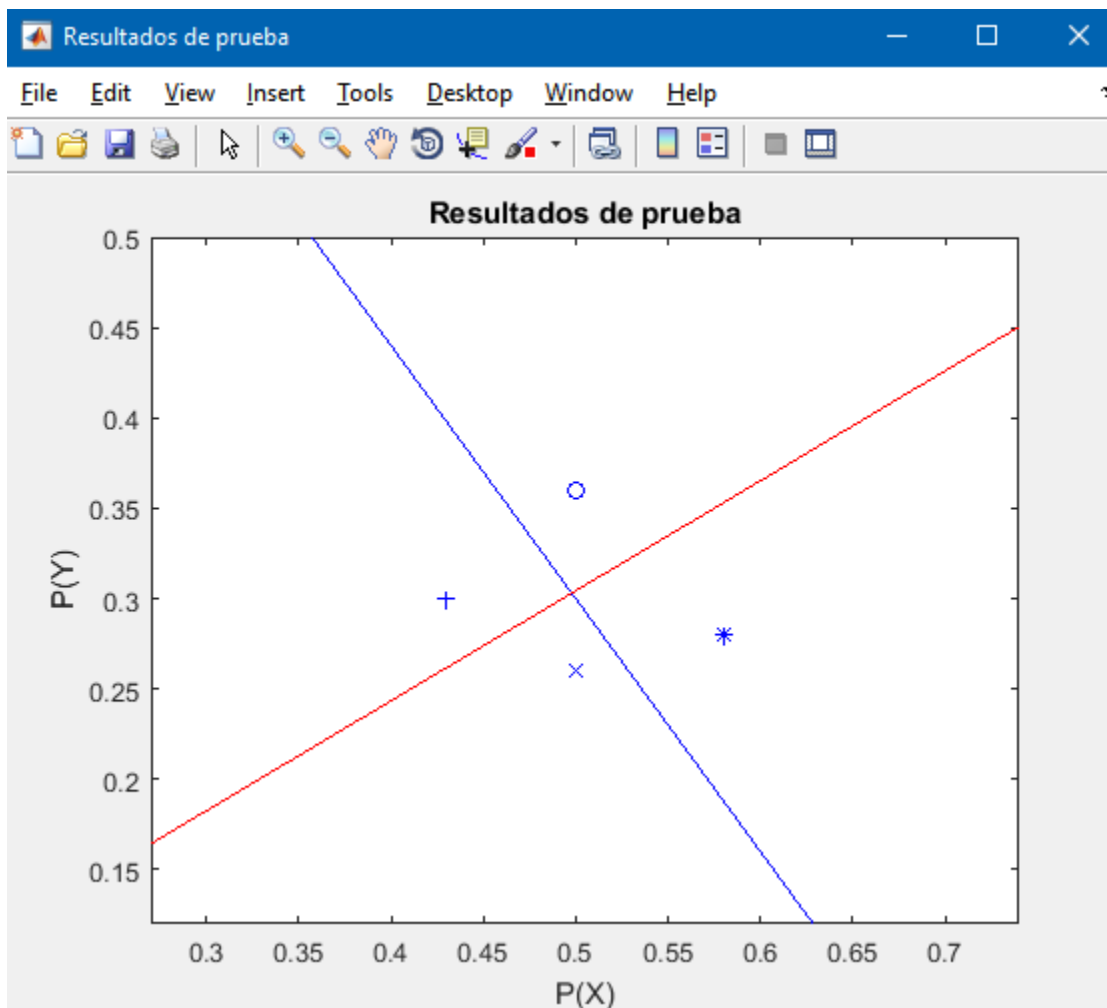


Figura 10. Nuevos puntos clasificados por la simulación del perceptrón

En esta Graficación podemos ver nuevos puntos que están en el plano coordenado, y que de acuerdo a dónde están ubicados pertenecen a un grupo diferente, comprobando así que la clasificación ha sido exitosa.

Pesos finales y BIAS

**Pesos:**

-1.4000	-1.0000
1.4000	-2.3000

**Bias:** |

1
0

## Código en MATLAB

```

Grupo1 = [0.5 0.7 0.8;
          1.4 1.8 1.6];
Grupo2 = [1.5 2.0;
          0.8 1.0];
Grupo3 = [0.3 0.0 -0.3;
          0.5 0.2 0.8];
Grupo4 = [-0.5 -1.5;
          -1.5 -2.2];

plot(Grupo1(1,:),Grupo1(2,:), 'bo')
hold on
grid on
plot(Grupo2(1,:),Grupo2(2,:), 'r*')
plot(Grupo3(1,:),Grupo3(2,:), 'g+')
plot(Grupo4(1,:),Grupo4(2,:), 'mx')

text(0,1.5, 'Grupo 1')
text(1.5,0, 'Grupo 2')
text(-1,0.5, 'Grupo 3')
text(0,-1, 'Grupo 4')
title('Elementos')
xlabel('P(X)')
ylabel('P(Y)')

a = [0 0]';
b = [0 1]';
c = [1 0]';
d = [1 1]';
P = [Grupo1 Grupo2 Grupo3 Grupo4];
T = [repmat(a,1,length(Grupo1)) repmat(b,1,length(Grupo2)) ...
     repmat(c,1,length(Grupo3)) repmat(d,1,length(Grupo4)) ];
%plotpv(P,T);
net = perceptron;
net = train(net,P,T);
plotpc(net.IW{1},net.b{1});

%Prueba
p = [0.5 0.43 0.5 0.58;
     0.36 0.3 0.26 0.28];
salida2 = sim(net,p);
disp('Resultados: ');
disp(salida2)
pesos = net.iw{1,1};
bias = net.b{1};
disp('Pesos: ');
disp(pesos);
disp('Bias: ');
disp(bias);
figure('Name','Resultados de prueba','NumberTitle','off')
plotpv(p,salida2);
title('Resultados de prueba')
xlabel('P(X)')
ylabel('P(Y)')
hold on; plotpc(pesos,bias);

```