Instituto Politécnico Nacional.
Escuela Superior de Cómputo.

Práctica 3:
"Perceptrón"

Asignatura:
Neural Networks.

Profesor: Marco Antonio Moreno Armendáriz.

Grupo: 3CM1.

Fecha de Entrega: 12/03/19

Alumno:

- Morales Flores Víctor Leonel.

Neural Networks

Práctica 3: Perceptrón.

Introducción:

Los seres vivos aprendemos desde el primer día de vida. Para nosotros todo este proceso es transparente y en ocasiones ni siquiera nos damos cuenta de que hemos aprendido algo nuevo. El problema surge cuando queremos hacer que una computadora "aprenda". ¿Cómo le enseñamos a una computadora? Para responder esto primero tendríamos que definir qué es el aprendizaje.

Según la Real Academia Española, aprendizaje se define como la "adquisición por la práctica de una conducta duradera". Esta definición es muy ambigua como para poder llevarla directo a los ordenadores.

La aparición de la célula de McCulloch-Pitts abrió un nuevo mundo en el área del cómputo, con ello se logró que las computadoras pudieran "aprender" a clasificar objetos; no obstante, esta célula no aprendía de forma automática. El usuario tenía que proponer valores y comprobar si al propagar los datos, las salidas de la red coincidían con los valores esperados. En caso de que esto no sucediera, se tenían que proponer otros valores nuevos(aleatorios si el usuario no sabía el funcionamiento interno de la célula). Esto quedó en el pasado con la aparición del perceptrón ya que éste contaba con una regla de aprendizaje.

En términos muy generales, una regla de aprendizaje permite a uan red neuronal calcular nuevos valores de pesos y bias comparando el valor esperado a la salida(targets) con la salida de la red con los actuales. En el caso del perceptrón, la regla de aprendizaje se basa en la señal del error. El cálculo de la señal del error para este modelo es muy simple pero importante para aplicar correctamente su regla de aprendizaje.

Volviendo a la pregunta de ¿Cómo le enseñamos a una computadora?, la respuesta no es la misma que la forma en que aprendemos los seres vivos. La computadora no tiene la capacidad de nuestro cerebro y aunque se llamen "redes neuronales", estas no son redes neuronales reales, sino que una aproximación con lo poco que sabemos hasta la fecha ed su funcionamiento.

La respuesta a la pregunta está en las matemáticas, para implementar una red neuronal se requiere de su modelo matemático y para lograr que su aprendizaje sea automático requerimos de una regla de aprendizaje que está definida en ecuaciones matemáticas. Ahora, una parte de la pregunta ya está contestada pero lograr que la computadora "aprenda" no es tan fácil. Se tienen que encontrar ecuaciones que satisfagan el propósito de forma adecuada y eficiente.

A lo largo de esta práctica se busca explicar la arquitectura de un perceptrón, la definición de su modelo matemático; así como la regla de aprendizaje que lo rige para que pueda hacer los ajustes de forma automática. También se proporcionará la lógica necesaria para implementarlo, el código fuente usando MATLAB, se abordarán sus limitaciones de forma general y algunas pruebas del funcionamiento de este tipo de redes para la clasificación de dos, cuatro o seis clases considerando entradas a la red de dos dimensiones.

Marco Teórico:

El perceptrón simple o perceptrón simplemente es una arquitectura propuesta en el año de 1958 por Frank Ronsenblant. Este modelo está basado en la célula de McCulloch-Pitts y una regla de aprendizaje.

Una de las características que hizo que este modelo llamara la atención fue su capacidad de "aprender" a partir de un conjunto de entrenamiento. Esto se realiza a través de la definición de una regla de aprendizaje que se basa en recalcular los valores de pesos(W) y bias en base a la señal de error que se genera a la salida de la red en la fase de entrenamiento.

Un perceptrón funciona en modo clasificador, eso quiere decir que con el uso de un perceptrón podemos separar los datos a través del uso de fronteras.

La arquitectura:

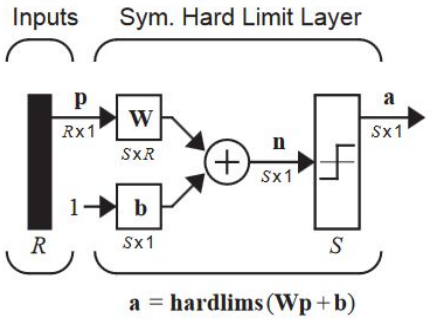

Figura 1: Arquitectura básica de un perceptrón simple.

Un perceptrón simple consta únicamente de una capa; sin embargo, el número de neuronas será determinado por el número de clases que se deseen clasificar ya que por la naturaleza de las funciones de activación usadas para esta arquitectura nuestras salidas en cada neurona sólo pueden ser 0's, 1's o -1's dependiendo de cuál escojamos.

Como podemos apreciar en la Figura 1, la arquitectura está formada por nuestra matriz de pesos, nuestro bias(aunque no es obligatorio siempre), nuestra función de activación o transferencia y las entradas a la red.

Un perceptrón tiene la capacidad de separar las clases siempre y cuando estas sean linealmente separables. En caso de que nuestro problema no cumpla con el principio de superposición se tendrá que hacer uso de otro tipo de red para cumplir con la tarea de clasificación.

Funciones de transferencia válidas:

Como se mencionó anteriormente, la salida de cada neurona de nuestro modelo solo puede tener a la salida valores de “uno-cero” o “uno-menos uno”. Esto es debido a las funciones de activación que son válidas para el sistema.

Las funciones de activación permitidas son ***hard limit*** o ***symmetric hard limit*** que son funciones de tipo escalón.

Para el caso de la función de transferencia ***hard limit*** obtendremos a la salida un valor “+1” si n es mayor o igual a cero; en caso contrario la salida de la función de activación será “0”(cero). Esto se puede entender mejor a través de la figura 2 y la definición de la función como se puede observar en la figura 3:

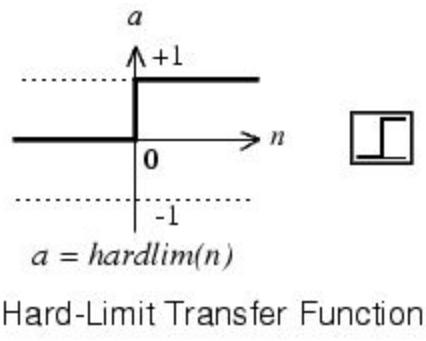

Figura 2: Gráfica de la función de transferencia ***hard limit(hardlim)***

$$a = \begin{cases} 1, & con\ n \geq 0 \\ 0, & en\ otro\ caso \end{cases}$$

Figura 3: Definición de la función de transferencia ***hard limit(hardlim)*** a trozos.

Si decidimos usar la función de transferencia ***symmetric hard limit*** entonces obtendremos a la salida un valor “+1” si n es mayor o igual a cero; en otro caso la salida de la función de activación será “-1”(menos uno). La figura 4 muestra el comportamiento de la función de transferencia de forma gráfica y la figura 5 muestra cómo está definida la función a trozos.

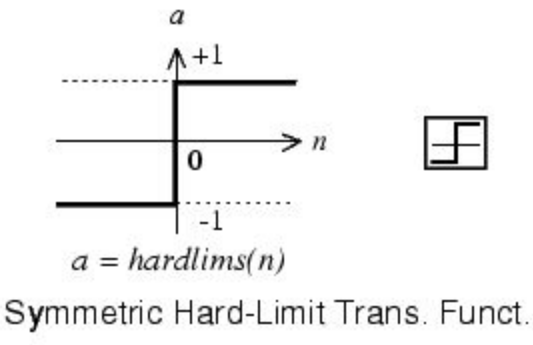

Figura 4: Gráfica de la función de transferencia ***symmetric hard limit(hardlims)***

$$a = \begin{cases} +1, & con\ n \geq 0 \\ -1, & en\ otro\ caso \end{cases}$$

Figura 5: Definición de la función de transferencia ***symmetric hard limit(hardlim)*** a trozos.

El modelo matemático:

El modelo matemático por el que se rige el perceptrón es muy sencillo debido a que solo cuenta con una capa feed-forward. Dependiendo de la función de activación que escojamos, el modelo matemático podría ser el que se muestra en la figura 6 dependiendo de nuestra función de activación

$$a = haardlim(Wp + b) \quad ó \quad a = haardlims(Wp + b)$$

donde:

a: salida de la red

W: Matríz de pesos (S filas y R columnas)

p: entrada a la red (R filas y 1 columna)

b: bias (S filas y 1 columna)

S: número de neuronas

R: Tamaño del vector de entrada a la red

Figura 7: Definición del modelo matemático del perceptrón.

Dependiendo de la implementación que se haga, la matriz de pesos(*W*) tendrá una forma similar a la descrita en la figura 8. Para hacer más fácil y rápida la obtención del i-ésimo elemento de la salida de nuestra red esta matriz de pesos podrá ser definida a través de un vector compuesto de la i-ésima fila de la matriz(figura 9) para posteriormente representar nuestra matriz a través de un vector como se aprecia en la figura 10.

$$W = \begin{pmatrix} w_{11} & w_{12} & w_{13} & \cdots & w_{1R} \\ w_{21} & w_{22} & w_{23} & \cdots & w_{2R} \\ w_{31} & w_{32} & w_{33} & \cdots & w_{3R} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{S1} & w_{S2} & w_{S3} & \cdots & w_{SR} \end{pmatrix}$$

Figura 8: Matriz de pesos con S neuronas y R entradas

$$w_i = \begin{bmatrix} w_{i1} \\ w_{i2} \\ w_{i3} \\ \vdots \\ w_{is} \end{bmatrix}$$

Figura 9: Elementos de la i-ésima fila de la matriz de pesos.

$$W = \begin{bmatrix} w_1{}^T \\ w_2{}^T \\ \vdots \\ w_s{}^T \end{bmatrix}$$

Figura 10: Representación de la matriz de pesos en forma de vector.

Usando las figuras anteriores nuestro modelo matemático

Regla de aprendizaje:

Para el desarrollo del perceptrón Frank Rosenblatt tuvo como aportación clave la regla de aprendizaje que dotaba a la arquitectura de la capacidad de ser entrenada para poder solucionar problemas de reconocimiento de patrones. Rosenblatt logró probar que si existían los pesos sinápticos y bias para resolver el problema, su regla de aprendizaje lograría converger a ellos de forma automática y simple, incluso si las condiciones iniciales del perceptrón eran generadas de forma aleatoria.

Esta arquitectura del perceptrón simple contiene limitaciones que fueron publicadas en el libro *Perceptron* por Marvin Minskey pero fueron solucionados posteriormente con el perceptrón multicapa que será estudio de otra práctica.

En términos simples una regla de aprendizaje es un procedimiento en el que se ajustan los pesos(*W*) y bias. En el campo de las redes neuronales encontraremos tres categorías básicas las redes supervisadas, no supervisadas o las de aprendizaje reforzado. Para el caso de esta red el aprendizaje será supervisado, eso quiere decir que tendremos un conjunto de entrenamiento(valores de entrada a la red para propagar junto con la salida esperada).

Para poder aplicar nuestra regla de aprendizaje es necesario calcular nuestra señal de error. Primero recordemos que nuestro caso ideal tienda a cero; sin embargo, en ocasiones esto podría conllevar a un costo computacional elevado ya que requerirá de más iteraciones para poder converger a este valor. Por ello, en la práctica se busca que la señal de error sea muy pequeña. En la figura 11 se muestra cómo calcular la señal de error.

$$Caso\ ideal:$$

$$e \rightarrow 0$$

$$En\ la\ práctica\ (\ ejemplo):$$

$$e \rightarrow 1x10^{-3}$$

$$e = t - a$$

$$donde:$$

$$t: target\ de\ nuestro\ conjunto\ de\ entrenamiento$$

$$a: salida\ de\ la\ red$$

Figura 11: Cálculo de la señal de error

Vectores de pesos y fronteras de decisión:

Para este apartado primero debemos considerar que el número de fronteras de decisión estará determinado en base al número de neuronas que se tengan. Cada neurona es capaz de dibujar sobre nuestro espacio una línea recta. El número de neuronas está a su vez determinado por el número de clases que se quiere separar, por lo tanto, dependiendo de la tarea que realizará la red es el número de neuronas que se requerirán obedeciendo la siguiente expresión(figura 12):

$$2^S = C$$

$$\therefore S = \lceil \log_2(C) \rceil$$

$$donde:$$

$$S: Número\ de\ neuronas$$

$$C: número\ de\ clases\ a\ clasif icar$$

Figura 12: Cálculo del número de neuronas necesarias para la clasificación.

Con todas las ecuaciones mostradas hasta este punto ya estamos en condiciones de poder determinar la frontera de decisión. Para ello haremos los cálculos para cada neurona considerando un espacio de dos dimensiones. Recordando el modelo matemático del perceptrón(figura 7) lo desarrollaremos un poco más de la siguiente forma:

$$a = hardlim(Wp + b)$$

$$considerando\ una\ neurona\ y\ un\ tamañ o\ de\ entrada\ 2\ tenemos:$$

$$a = hardlim\left(\begin{pmatrix} w_{11} & w_{12} \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} + b \right)$$

$$desarrollando\ la\ expresió n\ anterior;$$

$$a = hardlim\left(w_{11} p_1 + w_{12}\ p_2 + b \right)$$

Figura 13: Desarrollo del modelo del perceptrón para una neurona.

Ahora calcularemos dos-puntos que pertenezcan a nuestra frontera de decisión. Para hacer eso igualaremos *n* con cero ya que ese valor de *n* es donde cruza nuestra frontera de decisión.

Ya que igualamos a cero encontraremos en qué puntos la frontera de decisión intersecta nuestro plano P_1 y P_2. Esto lo lograremos igualando nuestros puntos p_1 y p_2 con cero. Este desarrollo lo podemos observar en nuestra figura 14:

$$\textit{Igualamos } n \textit{ con cero}:$$

$$n = 0$$

$$\textit{pero } n = w_{11} p_1 + w_{12}\ p_2 + b$$

$$\therefore\ w_{11} p_1 + w_{12}\ p_2 + b = 0$$

$$\textit{Ahora encontraremos el punto donde la frontera intersecta al plano } P_1 \textit{ igualando } p_1 = 0$$

$$\left(w_{11}\, x\, 0\right) + w_{12}\ p_2 + b = 0$$

$$w_{12}\ p_2 + b = 0$$

$$w_{12}\ p_2 = -b$$

$$p_2 = -\frac{b}{w_{12}}$$

$$\textit{Tras calcular el valor de } p_2 \textit{ tenemos el primer punto de la frontera}$$

$$\textit{Primer punto de la frontera}: \left(0,\ p_2\right)$$

$$\textit{Para encontrar el punto donde la frontera intersecta al plano } P_2 \textit{ igualando } p_2 = 0$$

$$w_{11} p_1 + \left(w_{12}\, x\, 0\right)\ + b = 0$$

$$w_{11}\ p_1 + b = 0$$

$$w_{11}\ p_1 = -b$$

$$p_1 = -\frac{b}{w_{11}}$$

$$\textit{Tras calcular el valor de } p_1 \textit{ tenemos el segundo punto de la frontera}$$

$$\textit{Segundo punto de la frontera}: \left(p_1,\ 0\right)$$

Figura 14: Obtención de los puntos de la frontera de decisión para una neurona.

Para cada neurona se hará este procedimiento con el fin de obtener cada una de las fronteras de todo el modelo.

Por último, es importante mencionar algunas propiedades del vector de pesos, ya que este nos auxilia a determinar de qué lado de la frontera se encuentra nuestra clase '+1' :

- El vector de pesos permite determinar dónde está la clase '+1'
- El vector de pesos se dibuja haciendo uso de los valores de los pesos sinápticos.

- Siempre parte del origen(0,0)
- Lo importante del vector de pesos es su dirección y no su sentido.

Diagrama de Flujo:

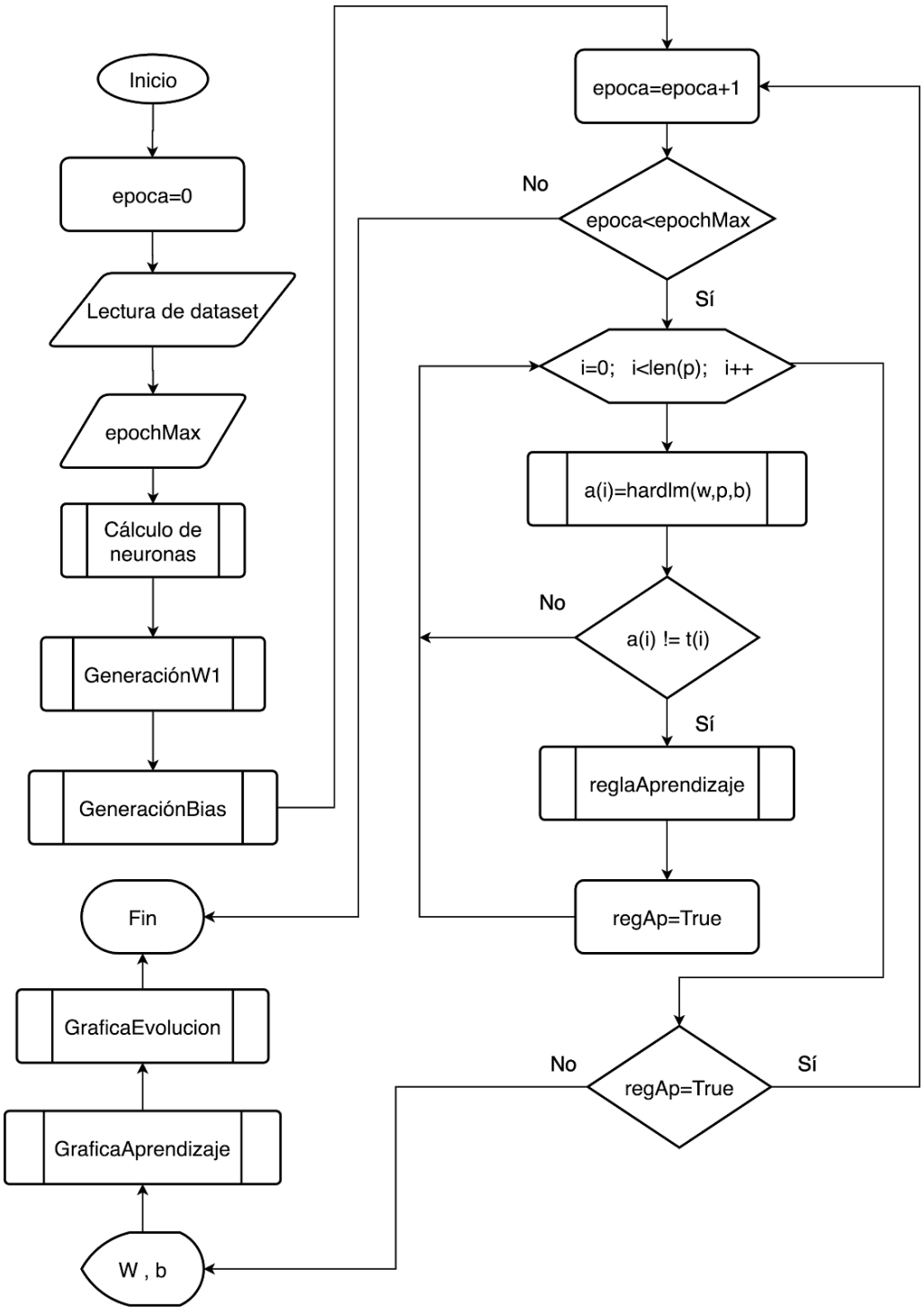

Figura 15: Diagrama de flujo del programa

Experimentos:

Para las pruebas realizadas a este perceptrón simple se hará uso de 3 datasheet. El primero será el comportamiento de la compuerta AND, el segundo será un conjunto de 4 clases y el tercero contendrá un total de seis clases. Con estas pruebas podremos tomar valores como el total de épocas que se requieren para que la red converja.
Con cada datasheet se harán 2-pruebas y al final se colocarán las notas correspondientes de los datos obtenidos durante el aprendizaje de la red.

A) Compuerta AND:

Para esta prueba el conjunto de entrenamiento es el siguiente:

p1	p2	t
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 1: Conjunto de entrenamiento de la compuerta AND.

$$W(0) = \begin{pmatrix} 0.8147 & 0.9058 \end{pmatrix} \qquad b(0) = 0.1270$$

$$W(6) = \begin{pmatrix} 2.8147 & 0.9058 \end{pmatrix} \qquad b(6) = -2.8730$$

Figura 16:Condiciones iniciales y finales en la prueba 1

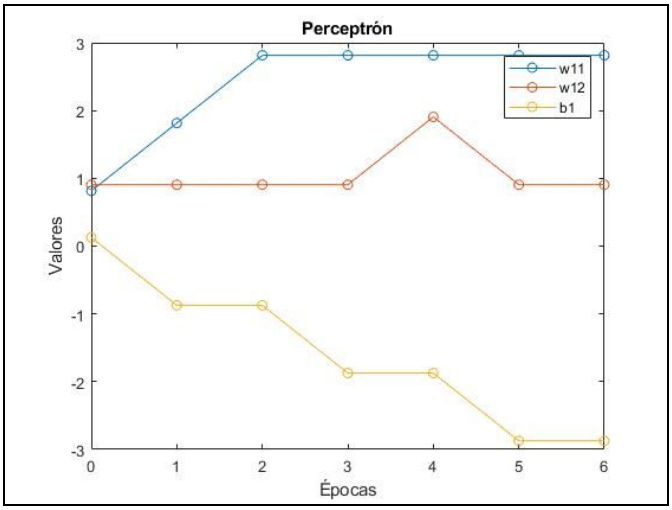

Figura 17: Evolución de pesos y bias en prueba 1 .

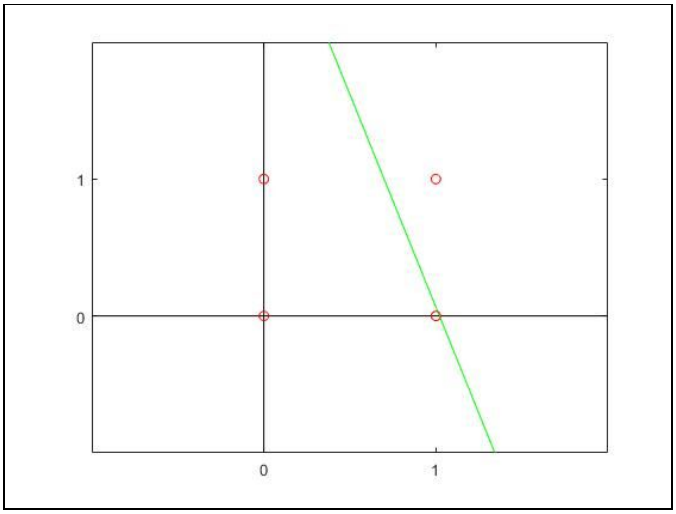

Figura 18: Gráfica de clases y frontera de decisión en prueba 1.

$$W(0) = \begin{pmatrix} 0.1189 & 0.4983 \end{pmatrix} \qquad b(0) = 0.9597$$

$$W(8) = \begin{pmatrix} 2.1189 & 1.4983 \end{pmatrix} \qquad b(8) = -3.0402$$

Figura 19:Condiciones iniciales y finales en la prueba 2

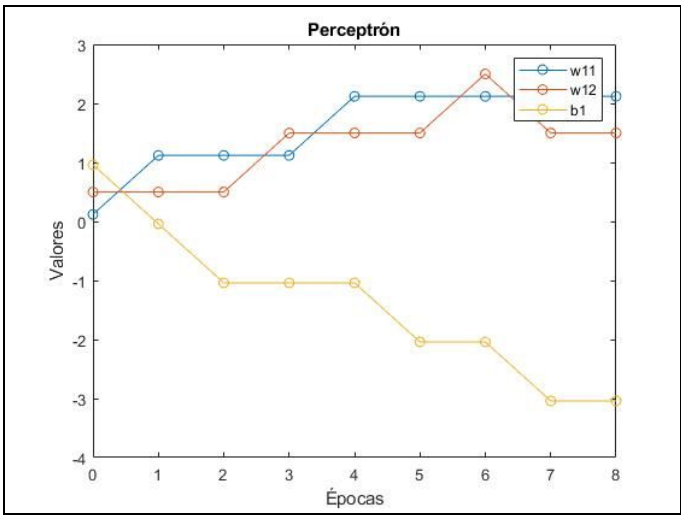

Figura 20: Evolución de pesos y bias en prueba 2 .

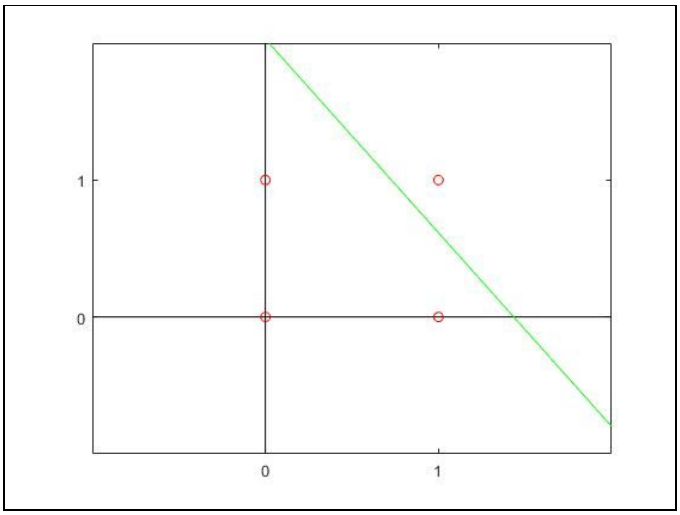

Figura 21: Gráfica de clases y frontera de decisión en prueba 2.

NOTAS DE LAS PRUEBAS CON LA COMPUERTA AND:

De estas pruebas podemos resaltar datos importantes que son más fáciles de percibir que cuando pasemos a clasificar conjuntos con más clases.

El primer punto interesante es la frontera de decisión. Como podemos ver, las fronteras de decisión en la figura 18 y la figura 21, las fronteras de decisión no son las mismas, cambian demasiado una de otra pero ambas cumplen con la función de clasificar los puntos dados de la forma correcta. Esto nos puede dar pistas de dos cosas:
a)Hay más de una frontera de decisión tal que puedan separar de forma adecuada el conjunto de entrenamiento.
b)Dependiendo de la frontera de decisión es la clasificación de la red y algunas podrían no clasificar correctamente para nuevos valores. Para este ejemplo si nosotros ingresáramos un punto más en (0,1.2) sería clasificado de forma diferente para cada una de las pruebas. Mientras que para la prueba 1(figura 18) el punto mencionado pertenecería a la clase "+1", en la prueba 2(figura 21) este valor pertenecería a la clase "0".
Para este caso, que es la compuerta AND, nosotros sabemos que la clase correcta es "0". Sabemos que los únicos puntos válidos son los presentados en el conjunto de entrenamiento a nivel lógico(ya que no existe un valor lógico "1.2") pero nos sirve analizar este caso para intuir que entre más valores se tengan en el conjunto de entrenamiento, entonces más exacta será la clasificación.

El segundo punto de esta prueba es el valor de los pesos y bias; así como su importancia. El programa genera los valores de forma aleatoria pero estos valores tienen asociado un costo computacional. En el caso de la prueba 1 podemos observar que se necesitaron de seis épocas para el aprendizaje de la red(figura 17); sin embargo, en la prueba número dos no se logró el aprendizaje en esa época, sino que se logró el aprendizaje hasta la época ocho(figura 20). Para este ejemplo sencillo el costo computacional entre cada uno de los casos fue poco(sólo dos épocas más) pero con problemas de clasificación más complejos el costo de una mala elección de pesos puede costar mucho más.

Conjunto de entrenamiento con 4 clases:

p1	p2	t1	t2
1	1	0	0
1	2	0	0
2	-1	0	1
2	0	0	1
-1	2	1	0
-2	1	1	0
-1	-1	1	1
-2	-2	1	1

Tabla 2: Conjunto de entrenamiento para 4 clases

$$W(0) = \begin{pmatrix} 0.9649 & 0.9706 \\ 0.1576 & 0.9572 \end{pmatrix} \qquad b(0) = \begin{pmatrix} 0.4853 \\ 0.8002 \end{pmatrix}$$

$$W(3) = \begin{pmatrix} -3.0351 & -0.0294 \\ 0.1576 & -2.0428 \end{pmatrix} \qquad b(3) = \begin{pmatrix} 0.4853 \\ -0.1997 \end{pmatrix}$$

Figura 22:Condiciones iniciales y finales en la prueba 3

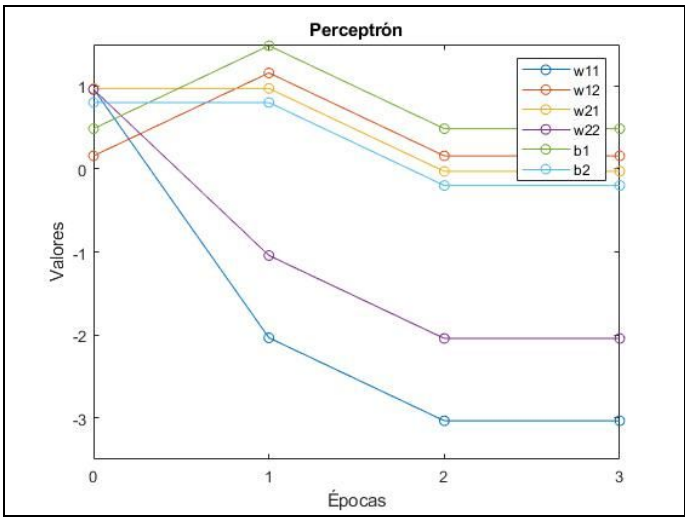

Figura 23: Evolución de pesos y bias en prueba 3.

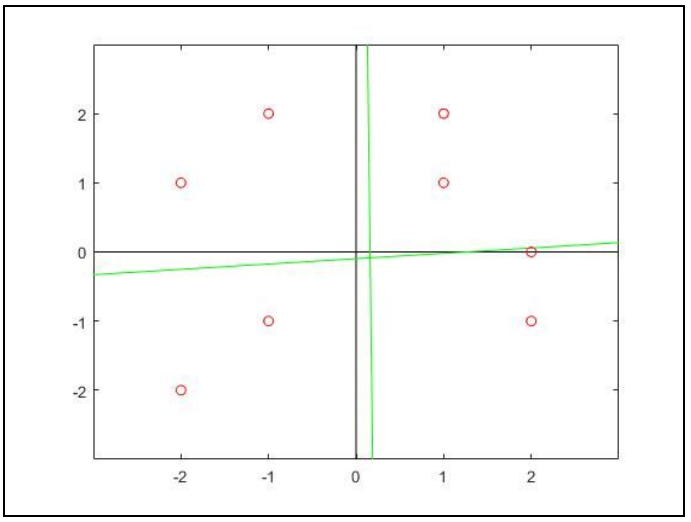

Figura 24: Gráfica de clases y frontera de decisión en prueba 3.

$$W(0) = \begin{pmatrix} 0.0357 & 0.9340 \\ 0.8491 & 0.6787 \end{pmatrix} \qquad b(0) = \begin{pmatrix} 0.7577 \\ 0.7431 \end{pmatrix}$$

$$W(2) = \begin{pmatrix} -0.9643 & -0.0660 \\ -0.1509 & -3.3212 \end{pmatrix} \qquad b(2) = \begin{pmatrix} -0.2422 \\ 1.7431 \end{pmatrix}$$

Figura 25:Condiciones iniciales y finales en la prueba 4.

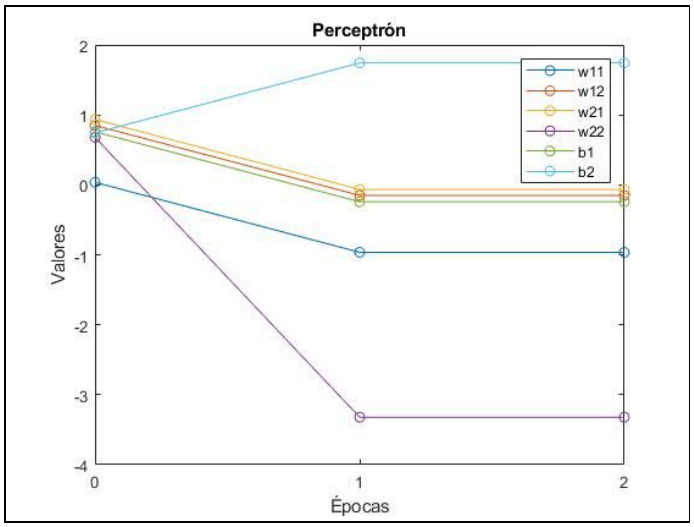

Figura 26: Evolución de pesos y bias en prueba 4.

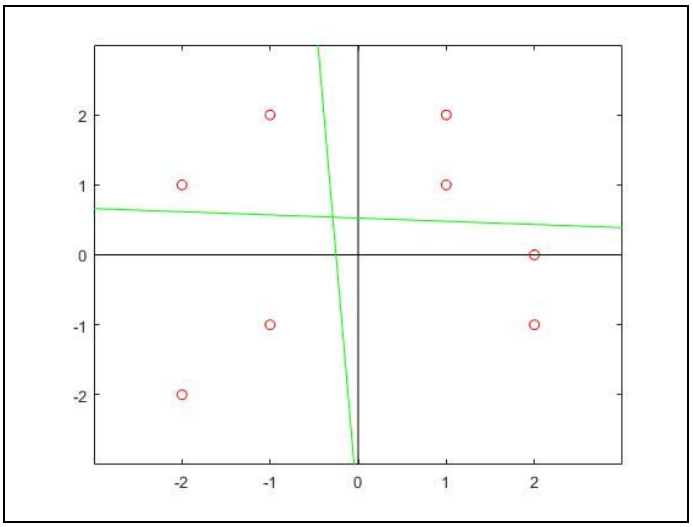

Figura 27: Gráfica de clases y frontera de decisión en prueba 4.

NOTAS DE LAS PRUEBAS CON 4 CLASES:

Comparada con el ejercicio anterior se pueden reforzar algunos puntos. Como se puede apreciar a simple vista en los datasets de cada uno de los ejemplos, entre más classes se tengan que clasificar se necesitarán de más valores en el conjunto de entrenamiento para cumplir con el propósito. Otro dato curioso es que aunque en este ejemplo se tienen más clases, el número de épocas necesarias para converger fue menor. En las pruebas anteriores el mínimo de épocas fue de seis mientras que para estas pruebas en ambos casos sólo se requirieron de tres épocas para lograr el aprendizaje de la red.

El otro dato importante de esta prueba se encuentra en nuestra matriz de pesos, bias y nuestro número de clases. En el marco teórico se presentó una ecuación(figura 12) para saber el número de neuronas necesarias. En esta prueba abordaremos uno de los casos que es un valor que es potencia de dos exacta. Así como se estableció en la expresión, el número de neuronas requeridas para clasificar este conjunto es de dos. Al ver la dimensión de nuestra matriz de pesos, nuestro bias y el número de nuestras fronteras de decisión podemos percatarnos de eso(figuras 22,24,25 y 27).

Recordemos que el número de filas en la matriz de pesos es igual al número de neuronas de la capa; también es bueno recordar que una neurona puede dibujar a lo sumo una frontera de

decisión en el espacio. En el ejemplo tenemos dos fronteras de decisión que nos permiten intuir de forma gráfica el número de neuronas en nuestra red.

Conjunto de entrenamiento con 6 clases:

p1	p2	t1	t2	t3
0	8	1	1	1
2	8	1	1	1
4	4	0	1	1
6	2	0	1	1
-6	2	1	0	0
-6	-2	1	0	0
-6	6	1	1	0
-6	4	1	1	0
-4	-4	0	0	0
-2	-4	0	0	0
2	2	0	0	1
2	1	0	0	1

Tabla 3: Conjunto de entrenamiento para 6 clases

$$W(0) = \begin{pmatrix} 0.3922 & 0.7060 \\ 0.6554 & 0.0318 \\ 0.1712 & 0.2769 \end{pmatrix} \qquad b(0) = \begin{pmatrix} 0.0461 \\ 0.0971 \\ 0.8234 \end{pmatrix}$$

$$W(55) = \begin{pmatrix} -3.6077 & 4.7060 \\ 0.6554 & 12.0318 \\ 12.1711 & 0.2769 \end{pmatrix} \quad b(55) = \begin{pmatrix} -4.9538 \\ -26.9028 \\ -0.1765 \end{pmatrix}$$

Figura 28:Condiciones iniciales y finales en la prueba 5.

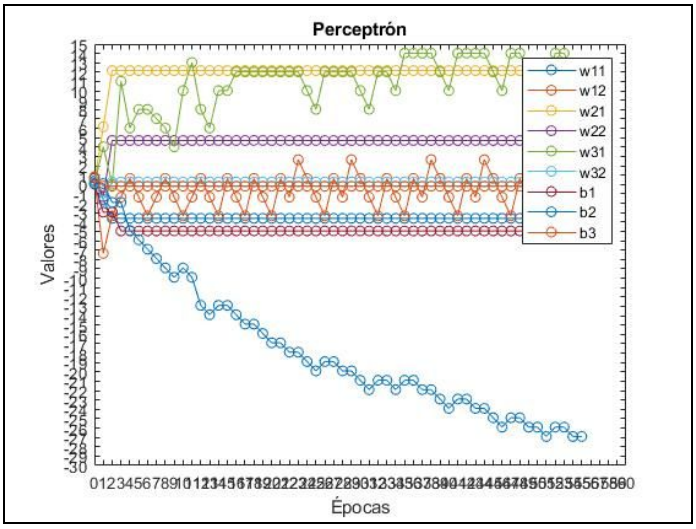

Figura 29: Evolución de pesos y bias en prueba 5.

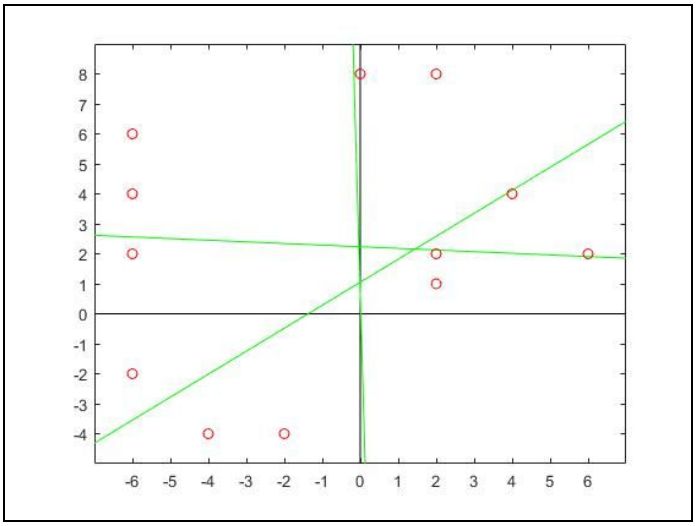

Figura 30: Gráfica de clases y frontera de decisión en prueba 5.

$$W(0) = \begin{pmatrix} 0.4898 & 0.7093 \\ 0.4455 & 0.7547 \\ 0.6463 & 0.2760 \end{pmatrix} \qquad b(0) = \begin{pmatrix} 0.6797 \\ 0.6550 \\ 0.1626 \end{pmatrix}$$

$$W(40) = \begin{pmatrix} -3.5102 & 4.7093 \\ -1.5544 & 15.7546 \\ 0.6463 & 0.2760 \end{pmatrix} \quad b(40) = \begin{pmatrix} -6.3203 \\ -25.3449 \\ 0.1626 \end{pmatrix}$$

Figura 31:Condiciones iniciales y finales en la prueba 6.

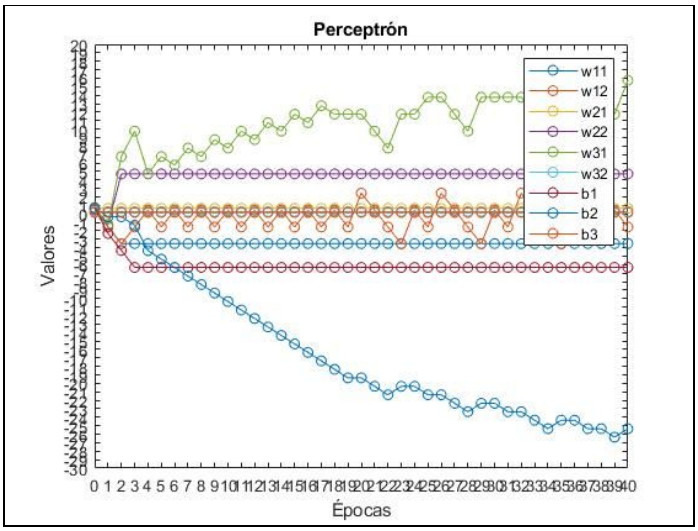

Figura 32: Evolución de pesos y bias en prueba 6.

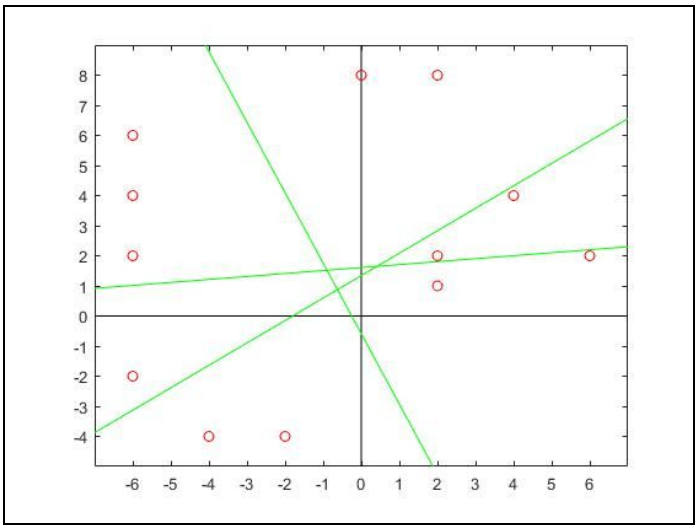

Figura 33: Gráfica de clases y frontera de decisión en prueba 6.

NOTAS DE LAS PRUEBAS CON 6 CLASES:

En esta prueba nos enfrentamos con varias situaciones, la primera es que seis no es una potencia exacta de dos. Por esta razón es muy importante que le prestemos atención nuevamente a nuestra ecuación para obtener el número de neuronas(figura 12). En la ecuación está marcado que se usará la función techo, este quiere decir que después de calcular el logaritmo, tomaremos el valor inmediato siguiente. Esto porque si hacemos uso de la función piso el número de neuronas calculadas no será suficiente para satisfacer la tarea.

Para este caso concreto al aplicar la ecuación, el resultado que nos dará será de 2.5849. Como no es un valor exacto tomaremos el siguiente valor entero que será tres. Esto quiere decir que para clasificar seis clases requerimos de tres neuronas.

El siguiente punto relevante es el número de iteraciones que se requirió para cumplir con la tarea. Para estas pruebas la red no completó su aprendizaje antes de la época cincuenta y cinco(figura 29); este valor está muy alejado del número de épocas necesarias en las pruebas anteriores(figuras 17, 20,23 y 26). Las razones por las que sucede esto es el valor inicial de la matriz de pesos, bias pero también influye demasiado qué tan alejados o cercanos estén los puntos de distintas clases en el espacio. Entre más cercanos estén unos de otros más difícil será obtener una frontera de decisión porque se requerirá de mayor precisión.

Por último está el caso menos deseado que es el hecho de que la red no aprenda de forma adecuada. Para ello, en la prueba seis se colocó un máximo de épocas permitidas de cuarenta. Con ello la red llegó a esta época sin poder clasificar de forma correcta los diversos puntos en el espacio. Prueba de esto está en el punto (-6,2), (2,2) y (6,2) de la figura 33. Estos puntos aún no están clasificados de la forma correcta(figura 30 y tabla 3) por lo que no podemos decir que nuestra red haya tenido un aprendizaje exitoso. Si prestamos atención veremos que el aprendizaje de la red para esta época(época 40) ya solo depende de una frontera de decisión porque las otras dos fronteras ya clasifican de forma adecuada.

Discusión:

El perceptrón tiene ciertas características que lo volvieron interesante para su época como el hecho de ser el primero en incorporar una regla de aprendizaje. Otra característica del perceptrón es la simplicidad de la arquitectura ya que sólo está conformado por una capa, esto lo hace más fácil de entender e implementar.

No obstante, el perceptrón simple tiene algunas limitaciones importantes para la clasificación. Una de ellas es que sólo puede ser usado para clasificar sistemas que sean linealmente separables. Otra limitante es su función de activación. Las funciones de activación válidas para el modelo sólo permiten que a la salida de una neurona tengamos dos valores; esto nos lleva a que para clasificar más de dos clases requerimos ir aumentando el número de neuronas y por consecuente, se requerirán más operaciones por parte del ordenador para encontrar los pesos y bias adecuados.

Por ello, para clasificar muchas clases probablemente este no sea el módelo más óptimo. Esto no quiere decir que esté descontinuado, aún en la actualidad este modelo sigue siendo referente y aplicado para cumplir ciertas tareas.

Conclusiones:

Cuando se habla de redes neuronales se suele escuchar que los valores de pesos y bias se ajustan de forma "mágica", se suele referir al proceso de aprendizaje como una caja negra pero esto no es cierto. Detrás de cada arquitectura de red neuronal se tienen reglas de aprendizaje que permiten que a lo que le llamamos "aprendizaje" se pueda lograr. Es interesante porque cuando escuchamos hablar de "aprendizaje" de la computadora lo asociamos con algo complejo pero detrás de esto solo se realizan operaciones que incluso pueden consistir en simples sumas y multiplicaciones, algo que las máquinas logran hacer desde hace años sin mayor problema.

El problema aquí es el desarrollar reglas de aprendizaje adecuadas, que sean eficientes pero a su vez fiables para cumplir con el propósito para el que fueron diseñadas.

El perceptrón fue el primer modelo que contemplaba su propia regla de aprendizaje y por el primero que ajustaba sus valores de pesos y bias de forma automática. Esto permite que encontrar estos valores sea mucho más rápido a intentar encontrarlos de forma manual proponiendo valores aleatorios.

Referencias:

Hagan, M., Demuth, H., Beale, M., & De Jesús, O. (2016). *Neural network design* (2nd ed.). s. n.].

Anexos:

Archivo: P03_PerceptronAprendizaje_2014081038.m

```
function [] = P03_PerceptronAprendizaje()
%P3: Aprendizaje del perceptron
%   Implementacion de un perceptron con lectura de dataset
%   desde archivo txt, introduccion de numero de iteraciones por consola y
%   graficacion de la evolución de bias y pesos
%Fecha de elaboración: 2019/04/06
%Autor: Morales Flores Victor Leonel
%Asignatura: Neural Networks
%Escuela: ESCOM-IPN(MX)
	epoca=0;
	w=zeros(1,1);
	b=zeros(1,1);
	e=0;
	archivoDS=input('Ingrese el nombre del archivo que contiene el dataset(sin
extension .txt): ','s');
	maxEpocas=input('Ingrese el maximo de épocas: ');
	[pn,targets,s,r]=lecturaDataSet(archivoDS);
	%targets
	w=generacionW(s,r);
	b=generacionBias(s);
	vecesAprendizaje=1;
	GuardarArchivo(epoca,w,b,"w");
	[numEntradas,tamEntrada]=size(pn);
	while(vecesAprendizaje>0)
	vecesAprendizaje=0;

	epoca=epoca+1;
	if(epoca>maxEpocas)
		fprintf("\n¡¡¡El programa no logró converger en %d
épocas!!!\n",maxEpocas)
		break;
	end
	fprintf("\n>>>>>Epoca: %d\n",epoca)
	for i=1:numEntradas
		p=pn(i,:).';
		n=w*p+b;
		a=hardlim(n);
		t=targets(i,:).';
		[e,reglaAprendizaje]=errorAprendizaje(t,a);
		if reglaAprendizaje
		vecesAprendizaje=vecesAprendizaje+1;
```

```
            [w,b]=reglaAprendizajePerceptron(w,b,e,p);
            %fprintf("\nSe aplica regla de aprendizaje");
            end

        end
        GuardarArchivo(epoca,w,b,"a");
        end
        graficaFronteras(pn,b,w);
        GraficarEvolucion(s,epoca)
        if(vecesAprendizaje==0)
        w
        b
        end
end
```

Archivo: generacionW.m

```
function w = generacionW(s,r)
%P3: Aprendizaje del perceptron
%   Genera matriz de pesos W con valores aleatorios en base de dimension
%   S,R
%Fecha de elaboración: 2019/04/06
%Autor: Morales Flores Victor Leonel
%Asignatura: Neural Networks
%Escuela: ESCOM-IPN(MX)
    w=rand(s,r);
end
```

Archivo: generacionBias.m

```
function bias = generacionBias(s)
%P3: Aprendizaje del perceptron
%   Genera matriz de pesos W con valores aleatorios en base de dimension
%   S,R
%Fecha de elaboración: 2019/04/06
%Autor: Morales Flores Victor Leonel
%Asignatura: Neural Networks
%Escuela: ESCOM-IPN(MX)
       bias=rand(s,1);
end
```

Archivo: errorAprendizaje.m

```
function [e,he] = errorAprendizaje(t,a)
%P3: Aprendizaje del perceptron
%   Calcula el valor de error y lo devuelve junto con true en caso de que
%   exista error(sum(e)~=0)
%Fecha de elaboración: 2019/04/06
%Autor: Morales Flores Victor Leonel
```

```
%Asignatura: Neural Networks
%Escuela: ESCOM-IPN(MX)
       format long
       e=t-a;
       he=false;
       if sum(e)~=0
            he=true;
       end
end
```

Archivo: graficaFronteras.m

```
function [] = graficaFronteras(puntos,bias,w)
%P3: Aprendizaje del perceptron
%  Grafica los puntos, las fronteras de decision y los vectores de pesos.
%Fecha de elaboración: 2019/04/06
%Autor: Morales Flores Victor Leonel
%Asignatura: Neural Networks
%Escuela: ESCOM-IPN(MX)
       figure(1)
       xpuntos=puntos(:,1);
       ypuntos=puntos(:,2);
       xejeP1=[min(xpuntos)-1,max(xpuntos)+1];
       yejeP1=zeros(1,2);
       xejeP2=zeros(1,2);
       yejeP2=[min(ypuntos)-1,max(ypuntos)+1];
       plot(xejeP1,yejeP1,"k-");
       hold on;
       plot(xejeP2,yejeP2,"k-");
       hold on;
       [f1,c1]=size(w);
       for i=1:f1
       wp=w(i,:);
       bp=bias(i,:);
       [xfrontera,yfrontera,m]=rectaFrontera(bp,wp,puntos);
       plot(xfrontera,yfrontera,"g-");
       hold on;
       %[xvector,yvector]=vectorW(m,wp);
       %quiver(xvector(1),yvector(1),xvector(2),yvector(2),1);
       %hold on;
       end
       axis([min(xpuntos)-1 max(xpuntos)+1 min(ypuntos)-1 max(ypuntos)+1])
       plot(xpuntos,ypuntos,"ro");
       xticks(min(xpuntos):1:max(xpuntos));
       yticks(min(ypuntos):1:max(ypuntos));
end
```

Archivo: GraficaEvolucion.m

```
function GraficarEvolucion (S,epocas)
%P3: Aprendizaje del perceptron
%Funcion que permite graficar todos los valores obtenidos durante el aprendizaje
%. Recibe S para poder leer de forma adecuada el archivo que contiene los datos.
%Fecha de elaboración: 2019/04/11
%Autor: Morales Flores Victor Leonel
%Asignatura: Neural Networks
%Escuela: ESCOM-IPN(MX)
	etiquetas="";
	%Abrimos el archivo archivo results todos los valores que se crearon en
	%las n iteraciones
	results = fopen('results.txt','r');
	%Formato de lectura del archivo
	filas=S*3+1;
	sizeRes=[filas Inf];
	%Se recupera en Res la matriz de resultados y se cierra el archivo
	%results.txt
	Res=fscanf(results,"%f",sizeRes);
	fclose(results);
	Res=Res.';
	[f,c]=size(Res);
	epoca=Res(:,1);
	w1=Res(:,2);
	aux=1;
	fila=1;
	figure(2)
	for i=2:2:c-S
	plot(epoca,Res(:,i),"o-");
	hold on
	etiquetas(aux)=strcat("w",string(fila),string(1));
	aux=aux+1;
	plot(epoca,Res(:,i+1),"o-");
	hold on
	etiquetas(aux)=strcat("w",string(fila),string(2));
	aux=aux+1;
	fila=fila+1;
	end
	fila=1;
	for i=c-S+1:c
	plot(epoca,Res(:,i),"o-");
	hold on
	etiquetas(aux)=strcat("b",string(fila));
	aux=aux+1;
	fila=fila+1;
```

```
        end
        %Configuracion de la grafica a mostrar
        xticks(0:1:100);
        yticks(-150:150);
        title('Perceptrón');
        xlabel('Épocas');
        ylabel('Valores');
        legend(etiquetas);
end
```

Archivo: GuardarArchivo.m

```
function GuardarArchivo(i,w,b,opc)
%P3: Aprendizaje del perceptron
%   Con el fin de un código mas legible se crea esta función encargará
%   de escribir en un archivo de texto 'results.txt' todos los valoers
%   a2 junto con 'i' que corresponde al número de epoca en la que
%   se generaron esos valores. opc permite que decidamos entre 'w' para
%   reescribir el archivo en caso que exista o 'a' para escribir al final
%   del mismo si es que existe.
%Fecha de elaboración: 2019/03/07
%Autor: Morales Flores Victor Leonel
%Asignatura: Neural Networks
%Escuela: ESCOM-IPN(MX)
        [f,c]=size(w);
        %Adapta el peso en solo una fila
        warc=reshape(w,[1,2*f]);
        barc=reshape(b,[1,f]);
        results=fopen("results.txt",opc);
        fprintf(results,"%d ",i);
        for j=1:f*2
        fprintf(results,"%d ",warc(1,j));
        end
        for j=1:f
        fprintf(results,"%d ",barc(1,j));
        end
        fprintf(results,"\n");
        fclose(results);
end
```

Archivo: hardlim.m

```
function a = hardlim(n)
%P3: Aprendizaje del perceptron
%   Implementacion de funcion de transferencia hardlim para una matriz de
%   mxn. Donde: Si i(m,n)<0, entonces i(m,n)=0; en otro caso i(m,n)=1
%Fecha de elaboración: 2019/04/06
%Autor: Morales Flores Victor Leonel
```

```
%Asignatura: Neural Networks
%Escuela: ESCOM-IPN(MX)
       [f,c]=size(n);
       a=zeros(f,c);
       for i=1:f
          for j=1:c
              if n(i,j)>0
                 a(i,j)=1;
              else
                 n(i,j)=0;
              end
          end
       end
end
```

Archivo: lecturaDataSet.m

```
function [p,targets,s,r] = lecturaDataSet(nomArc)
%P3: Aprendizaje del perceptron
%   Funcion para la lectura del documento que contiene el dataet y que
%   devielve las entradas a la red, los targets y el numero de neuronas de
%   la red.
%Fecha de elaboración: 2019/04/06
%Autor: Morales Flores Victor Leonel
%Asignatura: Neural Networks
%Escuela: ESCOM-IPN(MX)
       arc = fopen(strcat(nomArc,'.txt'),'r');
       info=fscanf(arc,"%d");
       fclose(arc);

       clases=info(end);
       r=2;
       s=ceil(log2(clases));
       info=info(1:end-1,:);
       columnas=2+s;
       filas=size(info)/columnas;
       info=reshape(info,columnas,[]);
       info=info.';
       p=info(:,1:2);
       targets=info(:,3:end);
end
```

Archivo: rectaFrontera.m

```
function [x,y,m] = rectaFrontera(bias,w,p)
```

```
%P3: Aprendizaje del perceptron
%  Se realizan los calculos necesarios para poder graficar la frontera de
%  decision
%Fecha de elaboración: 2019/04/06
%Autor: Morales Flores Victor Leonel
%Asignatura: Neural Networks
%Escuela: ESCOM-IPN(MX)
       xmax=max(p(:,1));
       xmin=min(p(:,1));
       x=[xmin-1,xmax+1];
       x1=0;
       format long
       y1=-(bias(1,1))/w(1,2);
       format long
       x2=-(bias(1,1))/w(1,1);
       format long
       y2=0;
       format long
       m=w(1,2)/w(1,1);
       format long
       mi=(-1)/m;
       format long
       b=y2-(mi*x2);
       [f,c]=size(x);
       for i=1:c
          format long
          y(i)=mi*x(i)+b;
       end
end
```

Archivo: reglaAprendizajePerceptron.m

```
function [wp,bp] = reglaAprendizajePerceptron(w,b,e,p)
%P3: Aprendizaje del perceptron
%   Hace calculo de la nueva matriz de pesos y del nuevo bias
%Fecha de elaboración: 2019/04/06
%Autor: Morales Flores Victor Leonel
%Asignatura: Neural Networks
%Escuela: ESCOM-IPN(MX)
       format long
       wp=w+e*p.';
       format long
       bp=b+e;

end
```

Archivo: vectorW.m

```
function [x,y] = vectorW(m,w)
%P3: Aprendizaje del perceptron
%  Recibe la pendiente y los pesos sinapticos para obtener las cooredenadas
%  de nuestro vector para que pueda ser graficado correctamente.
%Fecha de elaboración: 2019/04/06
%Autor: Morales Flores Victor Leonel
%Asignatura: Neural Networks
%Escuela: ESCOM-IPN(MX)
       x=zeros(1,2);
       y=zeros(1,2);
       x(1,2)=w(1,1);
       y(1,2)=w(1,2);
end
```