**PROYECTO PERCEPTRON SIMPLE (CÓDIGO C++)**

**INTRODUCCIÓN**

En 1936 Alan Turing estudio el cerebro como una forma de ver el mundo de la computación. Sin embargo, los primeros teóricos que concibieron los fundamentos de la computación neuronal fueron Warren McCulloch un neurofisiología y Walter Pitts un matemático quienes en 1943 lanzaron una teoría acerca de la forma de trabajar de las neuronas. Ellos modelaron una red neuronal simple mediante circuitos eléctricos. Años más tarde en 1949 Donald Hebb escribió un importante libro en el que explicaba los procesos del aprendizaje desde un punto de vista psicológico, desarrollando una regla de como el aprendizaje ocurría. Su idea fue que el aprendizaje ocurría cuando ciertos cambios en una neurona eran activados. Los trabajos de Hebb formaron las bases de la Teoría de las Redes Neuronales.

Luego en 1957 Frank Rosenblatt desarrolla el Perceptrón, Esta es la red neuronal más antigua utilizándose hoy en día para aplicación como reconocedor de patrones. Este modelo era capaz de generalizar, es decir, después de haber aprendido una serie de patrones podía reconocer otros similares, aunque no se le hubiesen presentado anteriormente. Sin embargo, tenía una serie de limitaciones, por ejemplo, su incapacidad para resolver el problema de la función OR-exclusiva y, en general, era incapaz de clasificar clases no separables linealmente.

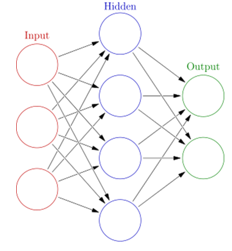
Diez años más tarde Stephen Grossberg realizó una red llamada Avalancha, que consistía en elementos discretos con actividad que varía en el tiempo que satisface ecuaciones diferenciales continuas, para resolver actividades como reconocimiento continúo de habla y aprendizaje de los brazos de un robot. En 1969 Marvin Minsky y Seymour Papert probaron (matemáticamente) que el Perceptrón no era capaz de resolver problemas relativamente fáciles, tales como el aprendizaje de una función no-lineal. Esto demostró que el Perceptrón era muy débil, dado que las funciones no-lineales son extensamente empleadas en computación y en los problemas del mundo real. Después de varios avances en 1980 Kunihiko Fukushima desarrolló un modelo neuronal para el reconocimiento de patrones visuales. Al instante en 1986 - David Rumelhart y G. Hinton redescubrieron el algoritmo de aprendizaje de propagación hacia atrás backpropagation.

En la actualidad, las redes neuronales se componen de numerosas capas de procesamiento, en las cuales se emplean métodos para obtener modelos de representación de los datos de entrada, a través de módulos simples no lineales de representación, que se encargan de extraer las características y patrones de los datos en bruto y transferirlas a capas posteriores más abstractas. Esta forma de trabajar, dota a la red la capacidad de deducir por sí misma, la mejor manera de representar los datos, sin necesidad de haber sido programada explícitamente para ello.

La forma más frecuente de aprendizaje automático, es el aprendizaje supervisado, en el cual el algoritmo requiere de una función objetivo, para ajustar los pesos de modo tal que las salidas del algoritmo se acerquen a la meta. De tal forma, las redes neuronales de aprendizaje supervisado, necesitan de una función objetivo, la cual es comparada con las salidas de la red, para calcular el valor de la función de costos, y así poder ajustar los pesos de las capas anteriores, de modo tal que en cada iteración se disminuya el error.

Debido a que una red neuronal profunda puede tener millones de neuronas, y centenares de millones de conexiones (incluso miles de millones), resulta impráctico ajustar uno a uno los pesos, incluso se puede intentar calcular el error mediante un procedimiento analítico empleando derivadas, pero dado el elevado número de variables de la red el proceso sería extremadamente complicado. Es por eso que, el efecto de aprendizaje (reducción del error) se consigue empleando el método de gradiente descendente.

Una red neuronal artificial es un grupo interconectado de nodos similar a la vasta red de neuronas en un cerebro biológico. Cada nodo circular representa una neurona artificial y cada flecha representa una conexión desde la salida de una neurona a la entrada de otra.

[](https://4.bp.blogspot.com/-AzmKfkChWMY/Wx8N73qO08I/AAAAAAAABIo/UZzmWZashE06ZfSn7HFBwyCxQRsyKvezACLcBGAs/s1600/Captura1.PNG)

Cada unidad neuronal está conectada con muchas otras y los enlaces entre ellas pueden incrementar o inhibir el estado de activación de las neuronas adyacentes. Cada unidad neuronal, de forma individual, opera empleando funciones de suma. Puede existir una función limitadora o umbral en cada conexión y en la propia unidad, de tal modo que la señal debe sobrepasar un límite antes de propagarse a otra neurona. Estos sistemas aprenden y se forman a sí mismos, en lugar de ser programados de forma explícita, y sobresalen en áreas donde la detección de soluciones o características es difícil de expresar con la programación convencional.

Las redes neuronales manejan dos tipos de información. La primera, es la información volátil, que se refiere a los datos que se están usando y varían con la dinámica de la computación de la red, se encuentra almacenada en el estado dinámico de las neuronas, el segundo tipo de información que manejan las redes neuronales, es la información no volátil que se mantiene para recordar los patrones aprendidos y se encuentra almacenada en los pesos sinápticos.

El aprendizaje de las redes neuronales, es el proceso de presentar los patrones a aprender, a la red y el cambio de los pesos de las conexiones sinápticas usando una regla de aprendizaje.

La regla de aprendizaje consiste en algoritmos basados en fórmulas matemáticas, que usando técnicas como minimización del error o la optimización de alguna "función de energía", modifican el valor de los pesos sinápticos en función de las entradas disponibles y con ello optimizan la respuesta de la red a las salidas que deseamos.

Podemos distinguir tres tipos de aprendizaje, el modo más intuitivo es el Aprendizaje supervisado, que consiste en que la red dispone de los patrones de entrada y los patrones de salida que deseamos para esa entrada y en función de ellos se modifican los pesos de las sinapsis para ajustar la entrada a esa salida.

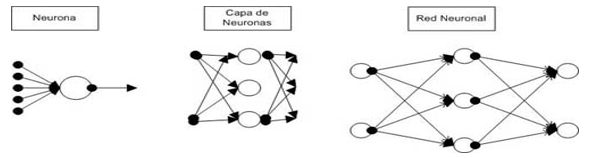
Otro modo de aprendizaje, Aprendizaje no supervisado, consiste en no presentar patrones objetivos, sino solo patrones de entrada, y dejar a la red clasificar dichos patrones en función de las características comunes de los patrones.

Y por último el aprendizaje reforzado, que usa una formula híbrida, el supervisor no enseña patrones objetivos si no que solo le dice si acierta o falla en su respuesta ante un patrón de entrada.

La topología o arquitectura de una red consiste en la organización y disposición de las neuronas en la red. Las neuronas se agrupan formando capas, que pueden tener muy distintas características.

Además las capas se organizan para formar la estructura de la red. Se puede observar en la siguiente figura.

La jerarquía de las redes neuronales:

[](https://3.bp.blogspot.com/-HX96igkZSyE/Wx8OKc2ZDnI/AAAAAAAABIs/-Tqfv8F0Shwg1oCwtSb5RwXn-LOL2ErTgCLcBGAs/s1600/Captura3.PNG)

Donde podemos ver que las neuronas se agrupan para formar capas y las capas se unen entre ellas formando redes neuronales. Para clasificar por la topología usaremos el número de capas en Redes Monocapas o Redes Multicapas.

Las redes monocapa son redes con una sola capa, para unirse las neuronas crean conexiones laterales para conectar con otras neuronas de su capa. Las redes más representativas son la red de Hopfield, la red BRAIN-STATE-IN-A-BOX o memoria asociativa y las maquinas estocásticas de Botzmann y Cauchy.  
  
Entre las redes neuronales monocapa, existen algunas que permiten que las neuronas tengan conexiones a sí mismas y se denominan auto recurrentes.

Las redes multicapa están formadas por varias capas de neuronas (2,3...). Estas redes se pueden a su vez clasificar atendiendo a la manera en que se conexionan sus capas.  
  
Usualmente, las capas están ordenadas por el orden en que reciben la señal desde la entrada hasta la salida y están unidas en ese orden. Ese tipo de conexiones se denominan conexiones feedforward o hacia delante.  
  
Por el contrario existen algunas redes en que las capas aparte del orden normal algunas capas están también unidas desde la salida hasta la entrada en el orden inverso en que viajan las señales de información. Las conexiones de este tipo se llaman conexiones hacia atrás, feedback o retroalimentadas.

**ANÁLISIS**

El perceptrón es una red de alimentación directa, esto es la información fluye desde la capa de entrada hacia la capa de salida. Fue desarrollado por F. Rosenblatt hacia final de la década de los cincuenta basándose en la regla de aprendizaje de hebb y de los modelos de neuronas biológicas de McCulloch y Pitts.

El Perceptrón es un clasificador, asigna a un vector de N valores un valor binario, usando una transformación no lineal. Así cada vector pertenece a una de las particiones que crea el perceptrón.

El perceptrón simple consta de dos capas de neuronas.

Esta red admite valores binarios o bipolares como entrada para los sensores y los valores de su salida están en el mismo rango que los de entrada.

El funcionamiento para ejecutar un patrón de la red es el siguiente:

**1.** Se establece el patrón de entrada en los sensores, la capa de entrada.

**2.** Se actualizan las neuronas de la capa de Salida.

Los pasos para que la red aprenda una lista de patrones son los siguientes:

**1.** Tomar un patrón al azar de la lista.

**2.** Se establece el patrón de entrada en los sensores, la capa de entrada.

**3.** Se establecen los valores deseados en las neuronas de la capa de salida

**4.** Se actualizan las neuronas de la capa de Salida.

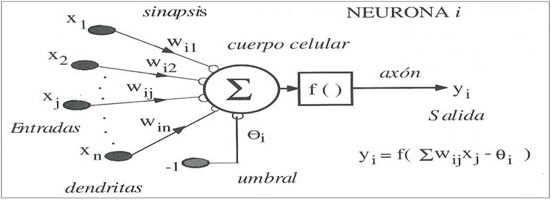
**5.** Solicitar que aprendan todas las sinapsis.

**6.** Si las sinapsis han cambiado volver al paso 1.

Si no han cambiado la red se ha estabilizado y paramos.

Hay que destacar que el perceptrón aprende solo cuando se equivoca al clasificar el patrón.

**Diseño**

[](https://4.bp.blogspot.com/-WGjFqDPvbA4/Wx8MiGzRsTI/AAAAAAAABIc/YRJvaVZeA8g1mb4DjvOOW0zVJDud9eIigCLcBGAs/s1600/Captura.PNG)

**Implementación**

Se tiene una neurona para clasificar frutas en comestibles y no comestibles, nuestra neurona estará en la capacidad de decirnos por medio de las entradas que frutas son comestibles y cuales no son comestibles.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Banano** | **pera** | **limón** | **mora** | **maracuyá** | **manzana** |
| **Piel** | **-1** | **1** | **-1** | **1** | **-1** | **1** |
| **Semilla** | **1** | **-1** | **-1** | **1** | **1** | **-1** |
| **Sabor** | **1** | **1** | **-1** | **-1** | **-1** | **1** |
| **Color** | **1** | **1** | **1** | **-1** | **1** | **1** |
| **Estado** | **1** | **1** | **1** | **1** | **1** | **1** |
| **Olor** | **1** | **1** | **1** | **1** | **1** | **1** |
| **Resultado** | **1** | **1** | **-1** | **-1** | **-1** | **1** |

Cabe aclarar que 1 es si y -1 es no.

El primer paso será declarar las entradas de cada fruta y la salida final en base de si es comestible o no lo siguiente que haremos será declarar el umbral que podrá ir de 1 a -1 de forma aleatoria, los rangos posibles de los pesos será tomado de manera aleatoria para cada entrada y el nivel de aprendizaje hará lo mismo dependiendo de cada época.

Luego entramos al entrenamiento del perceptron para lo cual tomaremos cada entrada la multiplicaremos por el peso y las sumaremos si el resultado es diferente a la salida esperada se ejecutara las veces necesarias hasta que el perceptron este entrenado.

**Codigo C++ (frutas)**

**#include <stdio.h>**

**#include <time.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**void Perceptron(){**

**srand(time(NULL));**

**float intervalos[] = {-0.98,-0.87,-0.65,-0.75,0.45,0.95,-0.32,-0.28,-0.11,-0.06,0.023,0.23,0.23,0.46,0.67,0.78, 0.75,0.82,0.98};**

**float aprendizaje[] = {0.023,0.23,0.233,0.467,0.675,0.788, 0.752,0.823,0.987};**

**float Umbral,defumbral,factor;**

**int i;**

**int x1[] = {-1,1,-1,1,-1,1};**

**int x2[] = {1,-1,-1,1,1,-1};**

**int x3[] = {1,1,-1,-1,-1,1};**

**int x4[] = {1,1,1,-1,1,1};**

**int x5[] = {1,1,1,1,1,1};**

**int x6[] = {1,1,1,1,1,1};**

**int result[] = {1,1,-1,-1,-1,1};**

**float w[6] = {intervalos[rand() % 18],intervalos[rand() % 18],intervalos[rand() % 18],intervalos[rand() % 18],intervalos[rand() % 18],intervalos[rand() % 18]};**

**factor = intervalos[rand() % 18];**

**Umbral = intervalos[rand() % 18];**

**defumbral = 1;**

**int verdad = 0;**

**int op;**

**int cont = 0;**

**int n = 0;**

**//////////// ENTRENAMIENTO DEL PERCEPTRON;**

**while(verdad == 0){**

**n++;**

**for(i=0;i<6;i++){**

**op = ((x1[i]\*w[0])+(x2[i]\*w[1])+(x3[i]\*w[2])+(x4[i]\*w[3])+(x5[i]\*w[4])+(x6[i]\*w[5])) + (defumbral\*Umbral);**

**if(op >= 0){**

**op = 1;**

**}**

**else{**

**op = -1;**

**}**

**if(op != result[i]){**

**w[0] = w[0] + (6\*factor)\*(x1[i]\*result[i]);**

**w[1] = w[1] + (6\*factor)\*(x2[i]\*result[i]);**

**w[2] = w[2] + (6\*factor)\*(x3[i]\*result[i]);**

**w[3] = w[3] + (6\*factor)\*(x4[i]\*result[i]);**

**w[4] = w[4] + (6\*factor)\*(x5[i]\*result[i]);**

**w[5] = w[5] + (6\*factor)\*(x6[i]\*result[i]);**

**Umbral = Umbral + (6\*factor)\*(defumbral\*result[i]);**

**}**

**}**

**for(i=0;i<6;i++){**

**op = ((x1[i]\*w[0])+(x2[i]\*w[1])+(x3[i]\*w[2])+(x4[i]\*w[3])+(x5[i]\*w[4])+(x6[i]\*w[5])) + (defumbral\*Umbral);**

**if(op == result[i]){**

**cont++;**

**}**

**if(cont == 6){**

**verdad = 1;**

**}**

**}**

**if(n > 900000000){**

**printf("[!] Demasiadas epocas realizadas!\n");**

**printf("[!] Intente nuevamente con otros pesos.\n");**

**exit(1);**

**}**

**}**

**/////////// MOSTRAR RESULTADOS FINALES**

**//system("clear");**

**printf("\n[\*] FINALIZANDO ENTRENAMIENTO...\n");**

**printf("[\*] PERCEPTRON SIMPLE [ENTRENADO]\n");**

**printf("\n--------------- DATOS FINALES --------------------------------\n\n");**

**printf("[\*] Total de epocas: (%i)\n",n);**

**printf("[\*] Peso 1 w[1]\t\t--> %2.2f\n",w[0]);**

**printf("[\*] Peso 2 w[2]\t\t--> %2.2f\n",w[1]);**

**printf("[\*] Peso 3 w[3]\t\t--> %2.2f\n",w[2]);**

**printf("[\*] Peso 4 w[4]\t\t--> %2.2f\n",w[3]);**

**printf("[\*] Peso 5 w[5]\t\t--> %2.2f\n",w[4]);**

**printf("[\*] Peso 6 w[6]\t\t--> %2.2f\n",w[5]);**

**printf("[\*] Umbral(Polarizacion)\t\t--> %2.2f\n",Umbral);**

**printf("\n\n--------------- VERFICIACION DE ENTRENAMIENTO ------------------\n\n");**

**for(i=0;i<6;i++){**

**op = ((x1[i]\*w[0])+(x2[i]\*w[1])+(x3[i]\*w[2])+(x4[i]\*w[3])+(x5[i]\*w[4])+(x6[i]\*w[5])) + (defumbral\*Umbral);**

**if(op >= 0){**

**op = 1;**

**}**

**else{**

**op = -1;**

**}**

**printf("\n[\*] Entradas: (%2i,%2i,%2i,%2i,%2i,%2i) --> Salida: %2i",x1[i],x2[i],x3[i],x4[i],x5[i],x6[i],op);**

**if(op != result[i]){**

**//system("clear");**

**printf("\n[!] Demasiadas epocas realizadas!\n");**

**printf("[!] Intente nuevamente con otros pesos.\n");**

**exit(1);**

**Perceptron();**

**}**

**}**

**}**

**int main(int argc, char \*\*argv)**

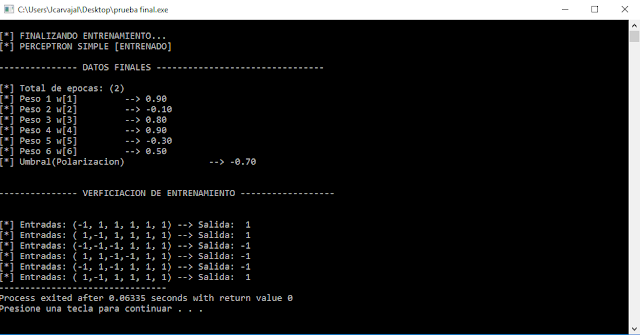
**{**

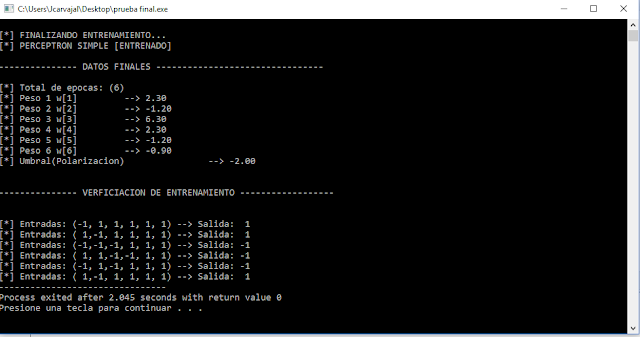
**Perceptron();**

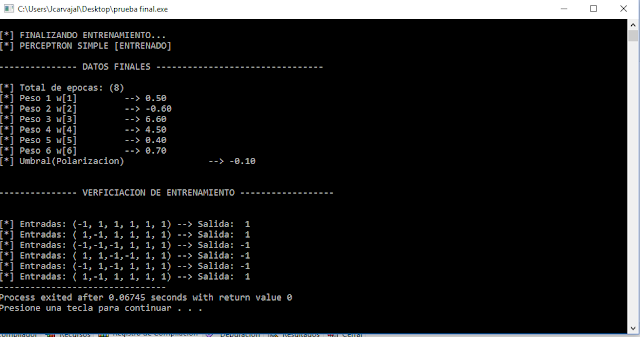
**return 0;**

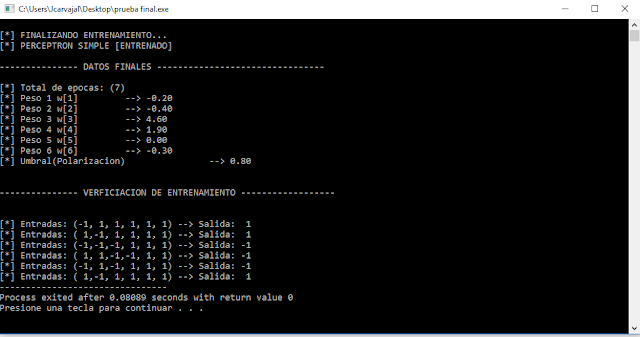
**}**

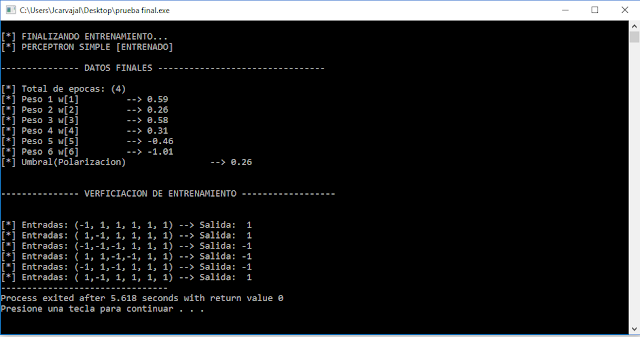
**Resultados**

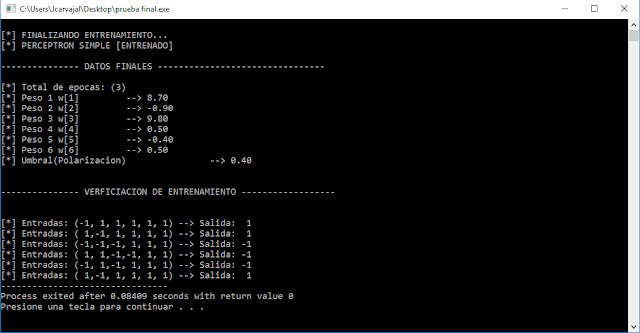
[](https://3.bp.blogspot.com/-izNhD-zfVRs/Wx8Q388cGbI/AAAAAAAABJQ/AxFWKF9cuqQZ3tKUhTA4NgMStWTa5VdxQCLcBGAs/s1600/cuarta%2Bejecucion.PNG)

[](https://4.bp.blogspot.com/-5osbhbfUFsg/Wx8Q3xRJIyI/AAAAAAAABJM/5B1_0uKTqqESwWi2Lyb3INT5tL_7Pcn9wCLcBGAs/s1600/primera%2Bejecucion.PNG)

[](https://4.bp.blogspot.com/-_x6wdiPiXqE/Wx8Q35_CRdI/AAAAAAAABJU/JlZFYcyZAckwlutkH_72yZN8T4sQigUhgCLcBGAs/s1600/quinta%2Bejecucion.PNG)

[](https://4.bp.blogspot.com/-TGm_6KYDPA4/Wx8Q4RdMXgI/AAAAAAAABJY/55IO3uqGnJwtnYwmc-0fblsgI_LF3HF1gCLcBGAs/s1600/segunda%2Bejecucion.PNG)

[](https://3.bp.blogspot.com/-ONTH7bMyoFw/Wx8Q4ufkX2I/AAAAAAAABJc/QCm6FDxWr2g0gI5cgcNxtykIJKMMeF2AACLcBGAs/s1600/sexta%2Bejecucion.PNG)

[](https://4.bp.blogspot.com/-9YITqq8q8no/Wx8Q5GP4zjI/AAAAAAAABJg/xatBl9Tu-RAj9BegTUD4PLCy6is2fjv5ACLcBGAs/s1600/tercera%2Bejecucion.PNG)