REDES NEURONALES EVOLUCIONARIAS
1. INTRODUCCIÓN

José del Carmen Rodríguez Santamaría
https://www.ejristos.com
Bogotá, Colombia

November 21, 2023

Abstract

Trabajamos varios modelos de una neurona y sus redes. Mostramos en primera instancia un mecanismo que puede reconocer un patrón. Es un circuito booleano compuesto de compuertas booleanas. Luego, dotamos a dichas compuertas de estabilidad frente al ruido y obtenemos un perceptrón. A continuación, proponemos un mecanismo que puede reconocer un patrón incluso en presencia de ruido y variabilidad. Es una red neuronal (RN). Para finalizar, mostramos cómo vincular la evolución a una RN para producir un mecanismo que aprende a reconocer un patrón en medio del ruido y la variabilidad. Es un Entorno Evolutivo para Redes Neuronales (EERN). Todo se muestra sobre un modelo de juguete. Esta introducción está dirigida a las personas que deseen entender tanto los por qué como los cómo de las RNs Evolucionarias.

Contents

INTRODUCCIÓN

Reconocer patrones está en la raíz de nuestra cultura humana. Una vez le pedí a una niña que compartiera conmigo la torta de su cumpleaños pero ella respondió: `No te conozco`. Esto significa que, inesperadamente para mí, yo no figuraba en sus registros de patrones de personas. Como consecuencia, no fui admitido a su fiesta.

Nuestro propósito general es automatizar el reconocimiento de patrones. Nuestra idea de partida es utilizar el lenguaje pero restringiéndonos a expresiones booleanas, que son aquellas que se refieren simplemente a ser o no ser. Ejemplo: sobre un artículo científico, que sea o no interesante, que esté o no claro. Así, el reconocimiento de patrones se convierte en una cuestión de lógica proposicional que puede automatizarse fácilmente gracias a compuertas booleanas, las que tratan con 1 y 0, VERDADERO y FALSO. Esto se ilustra en un ejemplo en la *Sección 1: Reconocimiento de patrones booleanos*.

Este tipo de reconocimiento no tiene demasiado que ver con lo que somos: nuestra pequeña invitaría a su mamá a su fiesta a pesar del vestido nuevo que hoy luce. Así, ella puede reconocer a su madre independientemente del ruido. Por ello, mostraremos que el reconocimiento booleano se puede modificar con éxito gracias al estudio de las neuronas biológicas. Pueden reconocer patrones incluso en condiciones de ruido y variabilidad. Esto es posible gracias a una abstracción o modelo matemático de neuronas que se llamó **perceptrón** y que generaliza las compuertas booleanas pero que además son estables en presencia de ruido y variabilidad. Los circuitos de perceptrones forman una Red Neural (RN) y mostraremos en la *Sección 2, Ruido y variabilidad*, cómo usarlos para el reconocimiento de patrones.

UNA RED NEURONAL CALCULA UNA FUNCIÓN, UNA ASIGNACIÓN DE UN CONJUNTO A OTRO.

1 *Ejemplo.* *Para reconocer imágenes en un conjunto de datos, la función generada por la red asocia un nombre a un subconjunto dado de imágenes, por ejemplo, a una imagen de un león, la RN asocia la palabra o cadena* LEÓN.

La función que define el objetivo de una RN se llama **función de especificación o espec**.

Otro desafío es que todos aprendemos incluso a edades muy avanzadas. Esto significa que, de ser muy malos clasificadores, podemos recibir un entrenamiento que mejore nuestras capacidades de reconocimiento. Así, *el materialismo impulsó a los ingenieros a buscar un mecanismo capaz de lograr el aprendizaje*. Esto se resolvió mediante el **Algoritmo de Propagación Reversa**, que modificó gradualmente las RNs para implementar su función espec según lo deseado por el diseñador. No implementamos el algoritmo de Propagación Reversa. Más bien, mostramos, en la *Sección 3, Aprendizaje*, cómo utilizar la Evolución para modificar una población de RNs de tal manera que el más apto, el que mejor resuelve la tarea de reconocimiento, sea clonado para formar la población de la generación siguiente. De esa manera, se espera que la población mejore sus capacidades de clasificación cuando corran las generaciones. Sobre una arquitectura muy simple, un modelo de juguete, ponemos a prueba esta expectativa.

1 RECONOCIMIENTO DE PATRONES BOOLEANOS

Las neuronas fueron en un principio una idea formulada en 1740 por Emmanuel Swedishborg (Fodstad [3] 2001).

Purkinje dio en 1837 la primera descripción de las neuronas(Purkyně [10]).

Más de 200 años de arduo trabajo en el estudio de las propiedades electrofisiológicas de las células nerviosas se pueden resumir de manera sucinta: `las neuronas son máquinas electroquímicas`.

Reciben nutrientes y oxígeno y los transforman en potencial de membrana. Esta se desestabiliza cuando la neurona se excita, aparece un pico eléctrico que luego se transmite como una señal, la mayoría de las veces bioquímica, a otras células (Piccolino, [9] 1998).

Se trata de materialismo en colores plenos y brillantes, aunque los detalles aún están en discusión (Ghosh et al, [4] 2022).

Para convertir una creencia en Ciencia, el Método Científico exige hacer predicciones. Lo más inmediato y necesario con respecto a las neuronas es que si se hace que una máquina realice las propiedades abstractas de las neuronas, la máquina debe pensar. Warren McCullock y Walter Pitts aceptaron en 1943 a este desafío y propusieron la neurona McCullock-Pitts. En esta sección veremos cómo resolver una tarea de reconocimiento de patrones con este modelo. Necesitamos algunas definiciones:

Una tarea de reconocimiento de patrones se resuelve mediante una función spec que asigna un nombre, un número o una etiqueta a un subconjunto determinado de imágenes, sonidos o textos, o instancias de un patrón complejo. Digamos que a una imagen de un número escrito a mano que fue dibujada por un adulto, la función asigna su nombre, digamos cinco, o, eventualmente, su representación arábica. Dicha función spec debe implementarse en un mecanismo.

Un clasificador es un mecanismo que resuelve una tarea de reconocimiento de patrones.

Un valor booleano puede ser solo 0 o 1, VERDADERO o FALSO, estar ENCENDIDO o APAGADO.

Un mapa de bits es una secuencia de valores booleanos o bits que codifican una imagen.

2 ***Ejemplo.*** *Una imagen digital cuyos píxeles están en negro y en blanco genera un mapa de bits porque todos los píxeles colocados en fila generan una secuencia binaria.*

Un clasificador booleano es un clasificador que acepta un

mapa de bits como entrada y su salida viene dada por un conjunto de lámparas numeradas que se encienden si y sólo si se ha detectado la clase correspondiente.

3 *Ejemplo.* *En la tarea de reconocer números escritos a mano, tenemos una imagen con píxeles en negro y en blanco. Cada píxel genera un valor booleano y la imagen completa se presenta como un mapa de bits, una secuencia o vector de valores booleanos o bits. Es la entrada. Como salida, tenemos lámparas en fila numeradas 0, 1,... 9, de modo que cuando el sistema detecta, digamos, un 5, se enciende la lámpara correspondiente. Cada lámpara puede estar ENCENDIDA o APAGADA, por lo que representa una salida booleana. Desarrollaremos un ejemplo que reconoce 1 ó 0. Este es nuestro modelo de juguete a lo largo de este artículo. En consecuencia, tenemos dos lámparas seguidas, la primera se enciende cuando el sistema detecta un 1 pero permanece apagada en caso contrario. La segunda lámpara se enciende si y sólo si el sistema detecta un 0. La entrada a la función es un mapa de bits, la salida de cada perceptrón es un valor booleano y la salida de una RN es una secuencia binaria con los valores binarios de todos los perceptrones.*

1.1 Nuestra tarea

Definamos que *nuestra tarea es reconocer mapas de bits de imágenes de números que representan 0 o 1*. Cada píxel de cada imagen se puede representar en blanco o negro y los mapas de bits correspondientes son secuencias de ceros o unos. Un píxel de color negro se codifica como uno, pero si el píxel está en blanco, se codifica como cero. En este artículo discriminaremos solo entre dos números, 1 y 0. Cada mapa de bits se compara con los dos arquetipos o imágenes de 1 y 0 y el sistema responderá: el mapa de bits es 1, o el mapa de bits es 0, o nada, que corresponde a DESCONOCIDO. Nuestras imágenes se dibujarán en una cuadrícula binaria 3×3, cada píxel en negro y en blanco. Ver Figura 1.

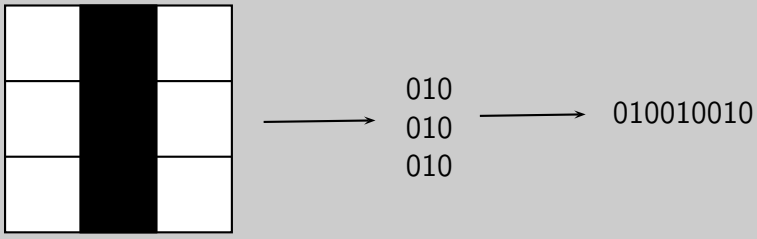

Figure 1: El número 1 se dibuja en una cuadrícula de 3×3. Un píxel en negro se codifica como 1 y un píxel en blanco como 0. El número 1 se codifica por el texto binario 010 010 010 Para la computadora se convierte en la secuencia binaria 010010010. Al adoptar esta simplificación, ignoramos la importancia de la geometría.

4 ***Ejercicio.*** *Dibuje un cero en la cuadrícula antes mencionada y encuentre la codificación binaria correspondiente.* Respuesta

5 ***Definición.*** *La tarea de reconocer patrones binarios cualesquiera sobre una cuadrícula* 3×3 *que codifica un cero o un uno se denomina aquí* ***problema-0-ó-1****.*

1.2 La arquitectura del clasificador booleano

Nuestra primera abstracción de un cerebro fue un conglomerado de neuronas que están modeladas como compuertas booleanas y que tienen entre ellas muchas interconexiones. En casos importantes, como en la retina y la corteza neural, las neuronas se ubican en capas. Estas ideas simples nos permitirán en un futuro proponer una arquitectura general para un clasificador. Por ahora, inventaremos una solución simple para la tarea del clasificador 0 o 1. Su arquitectura esquemática puede verse en la figura 2.

Pasemos ahora al circuito concreto que resuelve nuestro cometido.

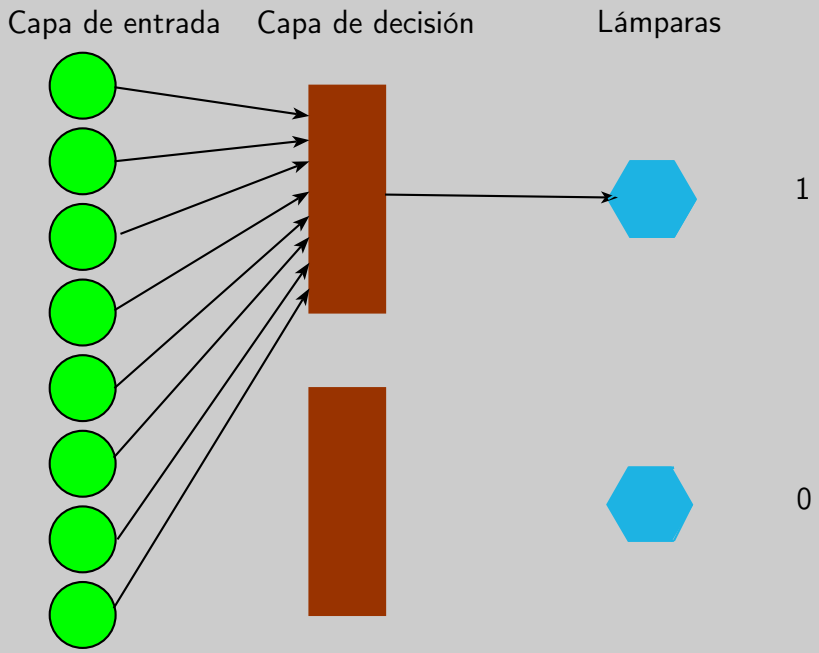

Figure 2: La cuadrícula 3×3 tiene 9 píxeles que están representados por una secuencia binaria con 9 bits. Cada píxel es leído por una neurona dedicada de la primera capa o capa de entrada. La máquina en general tiene dos salidas que podemos imaginar como lámparas. La lámpara superior se enciende cuando se detecta un 1. La lámpara inferior se enciende cuando se detecta un 0. Para encenderse, cada lámpara recibe un bit con un valor de 1 o verdadero. Para encender las lámparas para cumplir la tarea de clasificación, la capa intermedia de neuronas se divide en dos módulos, el primero debe activar la lámpara superior y el segundo la inferior. Esta arquitectura tiene una virtud: es muy sencilla e intuitiva. Su inconveniente: es demasiado costosa porque tiene dos módulos que suenan a repetición. Por tanto, podrían explorarse otras arquitecturas.

1.3 El clasificador booleano

Si observamos la imagen del número 1 y su representación binaria, podemos inferir que para discriminar imágenes lo único que debemos hacer es comprobar que los píxeles deben ser 1 en algunos sitios concretos y 0 en el resto. Y esto es cierto para cualquier tarea. Por eso podemos automatizar la tarea de discriminación considerando circuitos booleanos. Usaremos compuertas booleanas para diseñar nuestro discriminador. Elegimos la familia compuesta por AND, OR y NOT. La compuerta AND tiene dos entradas y una salida. Su salida es 1 o VERDADERA si y solo si ambas entradas son VERDADERAS. Y la compuerta OR tiene dos entradas y una salida. Su salida es 0 o FALSA si y sólo si ambas entradas son FALSAS. Por lo tanto, genera 1 cuando al menos una entrada es 1 o VERDADERA. La compuerta NOT tiene una entrada y una salida. Genera FALSO cuando la entrada es VERDADERA y VERDADERO cuando la entrada es FALSA. Estos operadores se representan de la siguiente manera: AND con $\wedge$, OR con $\vee$, y NOT con $\sim (p)$ (tutorialspoint [14] 2023; Xin [16] 2023).

Tenga en cuenta que nuestras imágenes de números son perfectas, pero sólo en este ejemplo introductorio. El módulo superior de nuestra capa de decisión debe verificar si la entrada es la representación binaria de la imagen de 1, es decir, 010010010. Por lo tanto, debemos idear un circuito booleano que genere uno cuando llegue esta secuencia y cero en caso contrario. Denotemos por n_i el valor verdadero o binario de la compuerta i de la capa de entrada, donde los números van del 1 al 9.

La tarea de reconocer 1 está representada por una fórmula booleana. Todo lo que tenemos que hacer es traducir la secuencia 010010010 a una fórmula booleana en la cual el cero corresponda a falso, una negación, y el uno a verdadero, una afirmación:

$\sim n_1 \wedge n_2 \wedge \sim n_3 \wedge \sim n_4 \wedge n_5 \wedge \sim n_6 \wedge \sim n_7 \wedge n_8 \wedge \sim n_9$

que dice que para generar 1, el valor de n_1 debe ser falso, una

negación, y el de n_2 debe ser verdadero, una afirmación, y el de n_3 falso, ..., y el de n_9 falso.

Para traducir esta fórmula a un circuito lógico, debemos tener en cuenta que las compuertas AND y OR son binarias, admitiendo dos entradas, mientras que NOT es uniaria, admitiendo una entrada. Por lo tanto, lo reescribimos tal como

$[((\sim n_1) \wedge n_2) \wedge$
$((\sim n_3) \wedge (\sim n_4))] \wedge$
$\{[(n_5 \wedge (\sim n_6)) \wedge$
$((\sim n_7) \wedge n_8)] \wedge$
$(\sim n_9)\}$

Su representación gráfica está en la figura 3.

6 *Ejercicio.* *Verificar que este circuito cumple con la tarea predefinida.* Respuesta

7 *Desafío.* *Escriba un programa en JS que calcule la tabla de verdad de la fórmula citada para verificar que responde 1 cuando y solo cuando la entrada es 010010010.*

8 *Ejercicio.* *Construya otro circuito que implemente la misma función que el ejemplo anterior. Infiera que muchos circuitos resuelven una función específica dada.* Respuesta

9 *Ejercicio.* *Diseñe un circuito que genere UNO o VERDADERO cuando detecte un cero. Con esto completamos el clasificador booleano que enciende la luz superior cuando se recibe UNO como entrada y la inferior cuando la entrada es CERO.* Respuesta

10 *Ejercicio.* *¿Cuál es la función espec del sistema completo que detecta un 1 o un 0?* Respuesta

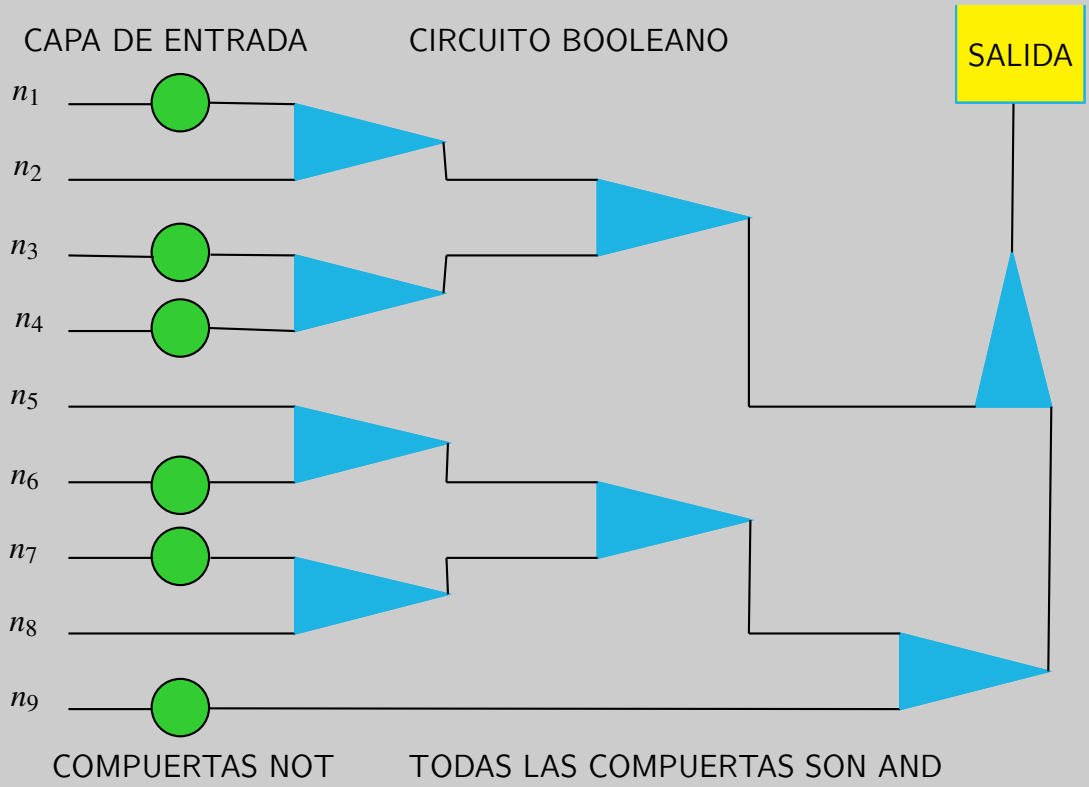

Figure 3: Este circuito booleano consta de compuertas NOT y AND. Responde con VERDADERO o UNO si y sólo si la capa de entrada recibe la cadena binaria 010010010. Para cualquier otra secuencia, la respuesta es FALSO o CERO. Aunque preferimos interpretar 0 como DESCONOCIDO.

2 RUIDO Y VARIABILIDAD

En esta sección, exploraremos la siguiente idea: *Las neuronas son circuitos booleanos estables que pueden realizar clasificaciones correctamente a pesar del ruido y la variabilidad.* **Debemos convertir esta idea en un programa científico que construya abstracciones, las pruebe, mida variables y extraiga conclusiones**. Comencemos.

Hemos considerado que nuestras imágenes sean perfectas y sin ruidos. ¿Qué podría pasar si introducimos ruido y variabilidad?

En nuestro marco en blanco o negro, el ruido cambia el color de un píxel en cualquier lugar de la cuadrícula. Un 0 se convierte en 1, o un 1 en 0. Por otro lado, la variabilidad aparece en los textos escritos a mano cuando cada persona dibuja signos de diferentes formas. El consiguiente problema de reconocimiento de patrones es tan engorroso que de vez en cuando todo el mundo elabora letras que nadie puede entender, ni siquiera el que las escribió.

Esto plantea un terrible problema de ingeniería:

11 ***Desafío.*** *Diseñe la representación de imagen del conjunto de números 0 y 1 que permita el máximo poder de discriminación cuando se permiten ruido y variabilidad. El autor supone que la representación habitual, que también se adoptó en el texto, es óptima en este sentido: tiene el mínimo número de casos* `no se sabe` *y de clasificación errónea. ¿Pero, qué podrá pasar con todo el conjunto de 10 dígitos?*

Pasemos ahora a proponer algunas soluciones al problema del ruido que también implica variabilidad.

2.1 Solución con circuitos booleanos

Probemos que con circuitos booleanos se puede implementar un circuito discriminador para imágenes con ruido. La mejor manera de

probar esto es desarrollar un método para construir el circuito. Hagámoslo.

Ya sabemos cómo hacer un circuito que detecte un mapa de bits específico. Entonces, juntemos todas las imágenes que están clasificadas como un patrón dado, digamos, aquellas que representan 1. Así, para cada mapa de bits de la clase resultante, construimos un módulo de reconocimiento y los unimos todos con compuertas OR. Entonces, si se encuentra alguna variante, todo el subsistema responderá VERDADERO.

12 *Ejercicio.* *¿Recomendaría utilizar la solución recién propuesta para manejar datos reales, como los de letras escritas a mano, cuyos caracteres están codificados en imágenes de* 28×28 *píxeles?* *Respuesta*

Dado que las compuertas booleanas no ofrecen ninguna posibilidad real para el tratamiento del ruido, consideremos las máquinas de Turing o la programación ordinaria. Pero cuidado, la idea primordial de las puertas booleanas no puede abandonarse porque de todas formas todo problema de reconocimiento puede formularse y resolverse en términos de operaciones booleanas, ser o no ser.

2.2 Solución con máquinas de Turing

El problema de reconocimiento en presencia de ruido tiene una respuesta inmediata con las Máquinas de Turing o la informática ordinaria de la siguiente manera:

Supongamos que la tarea es discriminar entre imágenes cuyos mapas de bits son 010010010 y 111101111 que corresponden a 1 y 0 respectivamente. El ruido produce 110111101. ¿Debe interpretarse como un 0 o como un 1? Para decidir, calculamos la distancia de la secuencia observada a cada objetivo, encontramos la mínima y seleccionamos el objetivo correspondiente como la interpretación correcta. Ahora bien, ¿cómo calculamos la distancia?

13 ***Definición.*** *La distancia Bit por Bit (distancia BpB or BPB) entre dos secuencias binarias o cadenas de igual longitud es el número de sitios o bits en los que difieren las dos secuencias.*

14 ***Ejemplo.*** *Estamos utilizando cadenas binarias que representan mapas de bits, descripciones binarias de una imagen. La distancia BpB cuenta el número de sitios en los que el patrón observado difiere del objetivo:*

```
Objetivo 1, 010010010, uno
Observado   110111101
Distancia   *  * **** 6.



Objetivo 2, 111101111, cero
Observado   110111101
Distancia     * *  * 3



La distancia mínima es 3,
y decidimos que
la imagen observada representa un cero.
```

Vemos que la distancia de la cadena observada al mapa de bits de 1 es 6, y la distancia al mapa de bits de 0 es 3. Por lo tanto, decidimos que la cadena observada o de prueba representa un 0.

15 ***Ejercicio.*** *Implemente la idea antes mencionada de la distancia BpB en un programa JavaScript.* *Respuesta*

16 ***Crítica.*** *Por razonable que sea el criterio de clasificación de la Distancia BpB, no es incondicionalmente satisfactorio. Considere, por ejemplo, el siguiente resultado posible:*

.

```
101000010 Secuencia aleatoria
010010010 UNO
Distancia a UNO = 4

101000010 Secuencia aleatoria
111101111 Cero
Distancia al cero = 5.

ES UNO
```

Vemos una secuencia binaria que se clasifica como 1. Pero, después de dibujarlo, mi intuición prefiere catalogarlo como cero:

```
101
000
010
```

17 ***Desafío.*** *Estúdiese para detectar situaciones conflictivas y desarrolle e implemente un criterio en JS que coincida con su intuición.*

La rápida solución del problema del ruido mediante la distancia BpB plantea una pregunta: ¿por qué son importantes las RNs, dado que la informática ordinaria es tan simple, intuitiva y elegante?

El problema es la complejidad. Eso significa que no existe una solución rápida para todos los casos. Más bien, hay muchos casos en los que irremediablemente surgen en gran cantidad situaciones conflictivas cuya solución exige un estudio aparte. Es aquí donde entran las RNs: no es necesario programarlas para resolver cada caso concreto. En cambio, reciben todos los casos y luego aprenden a resolverlos. De todos modos, estemos atentos a la evaluación del rendimiento de las RNs: ¿resuelven perfectamente todos los problemas y a qué precio?

Mientras tanto, consideremos ahora la solución biológica al problema del reconocimiento a prueba de ruido.

2.3 Abstrayendo la neurona

Las neuronas son complejas, tanto en estructura como en función. Algunas características se han abstraído de los modelos que han impulsado la Inteligencia Artificial. La primera es su estabilidad frente al ruido: si una señal es débil, se desvanece y se olvida, pero si la señal supera un umbral determinado, se propaga. Esta propiedad nos permite idear **neuronas booleanas** que son compuertas booleanas modificadas y que son a prueba de ruido. Es necesario recuperar las propiedades lógicas habituales de las compuertas pero, además, los circuitos deben ser estables frente al ruido.

18 ***Ejemplo.*** *Sea T el umbral necesario para que una entrada active una neurona.*

Una neurona OR es una compuerta con varias entradas. Si cada entrada es menor que T, la neurona no produce nada, un comportamiento que se interpreta como CERO o FALSO o DESCONOCIDO. Pero si la entrada supera T, la compuerta se activa y responde 1 o VERDADERO.

19 ***Ejercicio.*** *Defina una neurona AND.* Respuesta

20 ***Ejercicio.*** *Defina una neurona NOT.* Respuesta

Con estas construcciones vemos que el estudio de las neuronas es prometedor.

2.4 El perceptrón

Según el materialismo, pensamos con el cerebro. Este está compuesto por neuronas. Así, **el Método Científico predice que una**

máquina hecha de abstracciones claras de la neurona debe ser capaz de pensar.

Una abstracción fiable de las propiedades de la neurona se llamó **perceptrón** y fue presentada por Frank Rosenblatt en 1958.

Las propiedades de las neuronas que son abstraídas por el perceptrón son:

- Estabilidad frente al ruido: si una señal es débil, se desvanece y se olvida, pero si la señal está por encima de un umbral determinado, la señal se propaga.
- Las neuronas pueden asignar diferentes pesos a diferentes canales. Así, un estímulo cerca del cuerpo de la neurona podría ser más efectivo que otro con la misma fuerza pero en una región distante.
- Las neuronas pueden tanto activar como inhibir otras neuronas.

Un perceptrón naturalmente puede interpretarse como una máquina que toma decisiones. Ver la figura 4. Podría recibir entradas binarias que indiquen si existe un determinado factor, 0 cuando no, 1 si existe. Cada entrada se multiplica por un número que representa su importancia. Los resultados de las multiplicaciones se suman. Si la suma supera un determinado umbral, la decisión es afirmativa, en caso contrario es negativa. De esto deducimos inmediatamente que las neuronas también pueden tomar decisiones. Por lo tanto, cada organismo con una única neurona es eventualmente capaz de tomar decisiones por sí solo.

Este es un punto bueno para el materialismo, cuyo propósito es impulsar la creencia de que pensamos con el cerebro. Pero esto es falso: pensamos con el espíritu (Rodriguez [12] 2023).

21 ***Ejemplo.*** *Hay que decidir si se continúa leyendo un artículo. Los factores importantes son* `lectura fácil, interesante, buenos gráficos, instructivo, abundantemente citado`

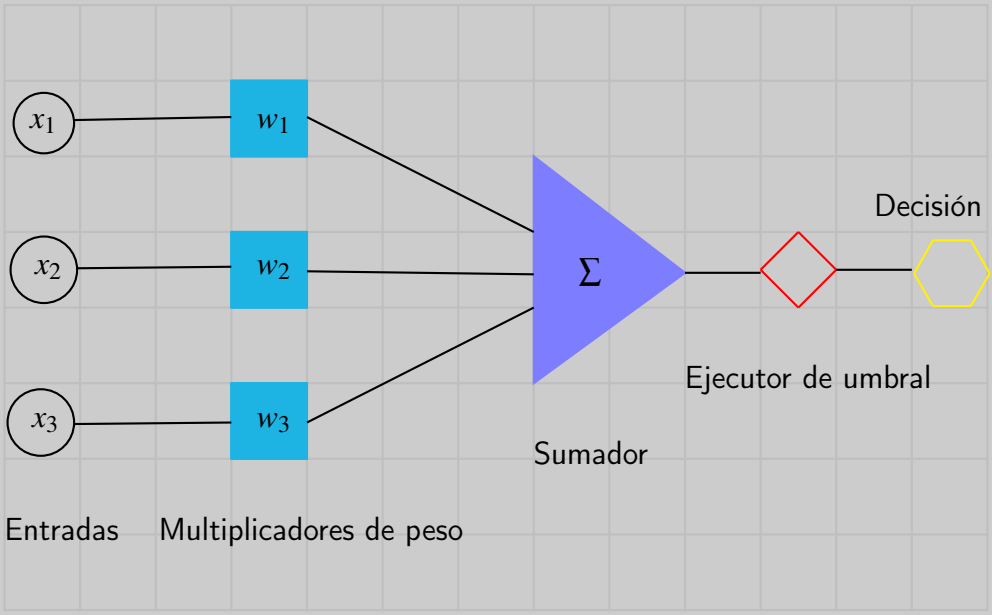

Figure 4: Este es un perceptrón. Es una abstracción de una neurona que naturalmente puede interpretarse como una máquina que toma decisiones. Supongamos que necesitamos decidir si debemos continuar leyendo o no un artículo. Tenemos como entrada al perceptrón la presencia o ausencia de algunos factores, digamos, parece muy importante para construir una cosmovisión, y es interesante. Los factores tienen diferentes pesos para influir en la decisión. Las entradas se multiplican por sus pesos y sus valores multiplicados se suman. Si la suma supera un determinado umbral, la decisión es afirmativa, en caso contrario es negativa. Para mejorar el poder del perceptrón, el ejecutor de umbral se divide en dos partes, la primera desplaza la suma en una constante b y la segunda compara el valor resultante S con el umbral T. Si $S \geq T$ la decisión es afirmativa, en caso contrario es negativa.

`y escrito por un chat`. *El artículo que nos ocupa tiene las siguientes características:* $[0,1,0,1,0,1]$. *Una notación que significa: el artículo no es fácil de leer, es interesante, carece de gráficos, es muy instructivo pero no está citado y ha sido escrito por un chat. El conjunto de valores de importancia de estos factores es respectivamente* $[0.2,0.8,0.6,0.9,0.2,-2]$, *lo que significa que ser instructivo es la cualidad más importante de un artículo bajo la condición de que esté escrito por un ser humano porque ningún chat es responsable de lo que dice. La decisión será afirmativa si el valor global supera 0,8. ¿Seguimos leyendo?*

Multiplicamos cada factor por su peso y sumamos los resultados: $0\times 0.2+1\times 0.8+0\times 0.6+1\times 0.9+0\times 0.2+1\times(-2)=0+0.8+0+0.9+0-2=-0.3$

Como la suma es menor que el umbral, el perceptrón nos aconseja que dejemos de leer. Vemos que la irresponsabilidad es el factor dominante que nos impulsa a dejar de leer.

Con una pequeña corrección, los perceptrones pueden convertirse en compuertas universales a prueba de ruido, con la capacidad de simular cualquier otra. Para lograr esto, restamos un valor b de la suma. De esa manera, obtenemos $S=\sum w_i x_i - b$ como la entrada al ejecutor de umbral que verifica el valor de S contra el umbral T: si $S\geq T$ la decisión es afirmativa, en caso contrario es negativa. En la literatura, al valor b se llama **sesgo**.

22 ***Ejercicio.*** *Demuestre oficialmente que un perceptrón calcula una función.* Respuesta

23 ***Ejemplo.*** *Codifiquemos el perceptrón en JS.*

Para habilitar la posibilidad de ejecutar nuestro programa en cualquier navegador y teléfono celular, incrustamos el código JS dentro de una plantilla HTML:

```
<!DOCTYPE html>
<html>


<head>


<meta charset="utf-8" />
<title>Perceptron.html </title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<meta name="generator" content="Bluefish 2.2.12, Visual Studio Code">
<meta name="author" content="jose">
<meta name="date" content="2023-06-23T13:13:00-0500">
<meta name="copyright" content="Licence = unconditional">
<meta name="keywords" content="Perceptron, AI, artificial intelligence">
<meta name="description" content=" This program calculates the output of a Perceptron.">
<meta name="ROBOTS" content="NOINDEX, NOFOLLOW">
<meta http-equiv="content-type" content="text/html; charset=UTF-8">


</head>


<body>


<div id="myDoc"></div>




<script>
var
message = "<h1>Perceptron <br> Shall one continue reading? </h1>";
bias = 0.2; //a small temptation to continue reading
factors = " easy reading, interesting, good graphics, " +
"<br> instructive, abundantly cited, written by GPT-chat";
inputs = [0, 1, 0, 1, 0, 1];
weights = [0.2, 0.8, 0.6, 0.9, 0.2, -2];
threshold = 0.8;
advice = "";
message += " <h3><br>Parameters <br> bias = " + bias
+ " //a small temptation to continue reading"
+ "<br> factors: " + factors
+ "<br> inputs = " + inputs
+ "<br> weights = " + weights
+ "<br> threshold = " + threshold;



let sum = bias;
for (let i = 0; i < inputs.length; i++) {
sum = sum + inputs[i] * weights[i];
}



//Print real numbers with two decimals only.
sum = 1.0 * Math.round(sum * 100) / 100;
message += "<br> sum = " + sum + "</h3> <br>";


if (sum > threshold) {
advice = "<h1>sum > threshold <br> Advice: Continue reading!</h1>";
} else {
advice = "<h1>sum < threshold <br> Advice: Stop reading!</h1>";
}
message += advice;
document.getElementById("myDoc").innerHTML = message;
</script>



</body>


</html>
```

24 ***Ejercicio sin respuesta.*** *Ejecute en un navegador el programa* `Perceptron.html`. *Compruebe que todo esté en orden. Su gentileza lo preparará para ejecutar otros*

programas que acompañan a este artículo. Para ello, puede copiar el código al portapapeles, pegarlo en un procesador de textos y guardarlo con el nombre `Perceptron.html`. *Para ejecutarlo: en una nueva pestaña de su navegador, presione Control + O para abrir un cuadro de diálogo para abrir archivos, seleccione el programa y se ejecutará automáticamente. También puede descargar el archivo comprimido* `evolNNintro.zip`, *descomprimirlo, seleccionar el programa y ejecutarlo en un navegador. Por favor, deje de asustarse y experimente hasta que pueda ver el resultado del programa.*

25 ***Desafío.*** *Ejecute en un navegador el programa de un perceptrón de W3Schools ([15], 2023). Compárelo con nuestra versión.*

26 ***Ejercicio sin respuesta.*** *Codificar es difícil. Por eso existen herramientas para facilitar el trabajo. Por ejemplo, todos, desde principiantes hasta expertos, aprecian los editores con textos coloridos y discriminación sintáctica. Se puede utilizar, por ejemplo, Visual Studio Code, Eclipse o NetBeans con la versión adecuada para el lenguaje utilizado. Entonces, elija uno y abra este archivo para ver qué tan bien se ve. Para desarrollar un programa, se escribe en el editor y se guarda. A continuación, se abre en un navegador (pruebe CNTRL + O), que lo ejecuta automáticamente después de abrirlo. JS es particularmente difícil porque muchos errores de ejecución no se publican. Aprender a desarrollar programas es importante porque uno debe preguntarse: ¿Por qué cometo tantos bugs o errores, y en su mayoría tan estúpidos? A continuación, podría ser inteligente preguntarse: dado que nuestro genoma es software, ¿por qué todo el mundo cree que somos un producto de la evolución cuando la abundancia de errores obligatorios no está por ninguna parte?*

Pasemos ahora a justificar una afirmación importante.

2.5 El perceptrón generaliza las compuertas booleanas y las neuronas.

Los perceptrones son importantes porque generalizan las compuertas booleanas y, por tanto, son útiles para el reconocimiento de patrones.

27 ***Ejemplo.***

La COMPUERTA-OR puede ser simulada por un perceptrón de la siguiente manera: Dos entradas que admiten valores booleanos, 0 o 1. Dos pesos, ambos iguales a 1. Valor de sesgo igual a 0. El valor umbral es igual a 1. Si al menos un valor de entrada es VERDADERO, la suma que llega al ejecutor de umbral es

igual o mayor que el umbral: el perceptrón genera VERDADERO. Si ambos valores de entrada son cero, la suma es cero y el perceptrón genera FALSO porque no se ha superado el umbral.

Para obtener una NEURONA-OR que sea estable frente al ruido, se puede cambiar el valor umbral en la COMPUERTA-OR anterior a 0,8 en lugar de 1. Con este nuevo valor, el perceptrón responde VERDADERO incluso cuando las señales están atenuadas. Pero no responde VERDADERO en otros casos si sólo el ruido es moderado. Porque si el ruido es extremo, los valores cercanos a cero podrían convertirse en 0,97 y, por tanto, podrían aparecer falsos positivos.

28 ***Ejercicio.*** *Especifique los valores de los parámetros de un perceptrón para simular compuertas y neuronas AND. Respuesta*

29 ***Ejercicio.*** *Especifique los valores de los parámetros de un perceptrón para simular compuertas y neuronas NOT. Respuesta*

30 ***Ejemplo.*** *En nuestra vida real, la complejidad se puede caracterizar por una simple receta:* `haz esto pero no aquello`*. Las neuronas naturales pueden encargarse directamente de este tipo de tareas (Yoder [18], 1989). Demostremos que un único perceptrón puede simular un circuito booleano, sin ruido, que codifica* `haz esto pero no aquello`

Un perceptrón puede simular un circuito booleano de la forma $p \wedge \sim q$: debe tener dos entradas, una para p con peso 1, otra para q con peso -1, $sesgo = 1$ y un umbral de 2. La suma es $S = p - q + 1$. El perceptrón generará 1 o VERDADERO cuando $S \geq 2$. Esto sucede si y solo si $p = 1$ y $q = 0$, lo que significa que p es verdadero pero q es falso.

31 ***Ejercicio.*** *Demuestre que un solo perceptrón puede implementar la función de especificación* `hacer esto pero no aquello` *y que es estable frente al ruido. Respuesta*

32 ***Desafío.*** *Diseñe sus simulaciones personales de compuertas booleanas y neuronales sobre un perceptrón. Compárelos con nuestras soluciones y con las de la literatura (Banoula, [2], 2023). ¿Cómo decidiría cuál es la mejor?*

Implicaciones para la neurología. Los perceptrones tienen una apariencia exterior compuesta de entradas+pesos+sesgo. Es igual en todas partes, pero debido a diferencias internas, un perceptrón puede cumplir una función muy diferente de otro. Es atractivo sugerir que nuestro cerebro es exactamente igual: lleno de neuronas que parecen iguales para un observador exterior pero que son bastante diferentes en sus funciones en la red cerebral. Convertir esta bonita idea

en Ciencia me parece extremadamente difícil: mientras una población significativa de neuronas se compone de unos pocos tipos, la población completa de neuronas ahora puede clasificarse en miles de tipos (Kwon [7] 2023). ¿Cuál es su función y por qué son necesarios tantos tipos?

Hemos visto que los perceptrones son realmente poderosos de forma aislada. ¿Qué pueden hacer cuando trabajan en conjunto?

2.6 Redes neuronales (RNs)

A principios del siglo XIX, Golgi y Cajal tiñeron células nerviosas y pudieron distinguir las sinapsis donde una neurona comunica algo a otra (Glickstein [5] 2000). **Este descubrimiento lleva a la idea de que las neuronas forman circuitos.**

Paralelamente a esta observación, y dado que los perceptrones son compuertas neuronales universales, nuestras abstracciones nos llevan a la siguiente definición:

33 ***Definición.*** *Una RED NEURAL (RN) es un circuito formado por perceptrones. A la literatura le encanta resaltar que este tipo de RN es Artificial.*

Los perceptrones son direccionales desde las entradas hasta las salidas. Por lo tanto, naturalmente pueden unirse o componerse de modo que una RN tenga entradas y salidas. Guiados por la estructura de los circuitos booleanos y de la anatomía del cerebro, digamos de la retina y la neocorteza, nuestros circuitos consistirían en capas y varias lámparas de salida que se activan cuando se descubre algún rasgo en los datos de entrada. En este artículo experimentamos con las RNs más simples, con 2 perceptrones o células aparte de las que reciben las entradas.

Para calcular la salida de una red, debemos reconocer que una RN calcula una función. Entonces, una red se calcula de acuerdo con las reglas de composición de funciones.

2.7 Construyendo simplicidad

Ya sabemos cómo utilizar compuertas booleanas para crear un circuito que reconozca un patrón. Para diseñar una RN que haga lo mismo, podemos reemplazar cada compuerta booleana por el perceptrón correspondiente, una compuerta AND por un perceptrón AND, y así sucesivamente. La RN resultante debe ser estable frente al ruido, es decir, debe reconocer el patrón objetivo incluso cuando esté ligeramente distorsionado.

34 ***Desafío.*** *Pruebe la afirmación antes mencionada.*

Salgamos de este procedimiento natural. En lugar de ello, experimentemos con las posibilidades de los perceptrones para crear simplicidad.

35 ***Ejemplo.*** *Diseñemos una RN para reconocer la función específica de 1 en el* ***problema 0 o 1****.*

El mapa de bits de 1, 010010010, induce la fórmula booleana o función espec
$\sim n_1 \wedge n_2 \wedge \sim n_3 \wedge \sim n_4 \wedge n_5 \wedge \sim n_6 \wedge \sim n_7 \wedge n_8 \wedge \sim n_9$
Esta fórmula es una función: para cada asignación de longitud 9 de valores booleanos, genera VERDADERO o FALSO.

Ahora, dado que todos los conectores son AND, en lugar de 9 perceptrones AND, construimos un perceptrón AND que admite 9 entradas, algunas de las cuales están previamente negadas, un hecho que se representa mediante un peso negativo más un sesgo conveniente. Y eso es todo. Explícitamente:

La función espec 010010010, el mapa de bits de 1, debe determinar los parámetros del perceptrón:

- La fórmula tiene 9 entradas AND. Para ser VERDADERO, cada sitio debe ser VERDADERO. Entonces el umbral es 9.
- Por cada VERDADERO o 1 en la entrada, el peso debe ser 1.
- Por cada FALSO o 0 en la entrada, debe haber una negación. La de y es $1 - y$. Entonces, el peso correspondiente es -1.
- El valor del sesgo se determina así: hay seis negaciones, por lo que hay que sumar 6 al sesgo.
- Para conseguir estabilidad, restemos 1,8 al umbral. Esto equivale a sumar 1,8 al sesgo. Sesgo total = 6 + 1,8 = 7,8.

El código está en el programa `PerceptronOne.html`. El código JS correspondiente, sin la envoltura HTML, tiene este aspecto:

```
<script>
//This perceptron shall recognize (output TRUE or 1) the bitmap
//010010010
//as the bitmap of the image of 1 in a 3x3 grid.
var
message = "<h1>Perceptron <br> Do you recognize 010010010 as the bitmap of 1? </h1>";


factors = "Each pixel is a factor, 9 in total. " ;
//This is the
inputs = [0, 1, 0, 0, 1, 0, 0, 1, 0 ];
weights = [-1, 1, -1, -1, 1,-1,-1, 1,-1];
//To know the value of the threshold:
//The evaluation of the formula over the spec bitmap
//shall be equal to 9 TRUE values
```

```
//because everything is TRUE. So, the threshold is 9.
threshold = 9;
//The value of the bias is determined so:
//The sum evaluated over 0100100010 must be equal to the threshold
//Each negation of y is given by -y + 1.
//Since there are 3 TRUE values in 0100100010, the bias must be 6.
//To achieve some stability against noise,
//let us subtract 1.8 from the threshold, i.e., 0.2 for each bit of the input.
//This is equivalent to adding 1.8 to the bias.
//Total bias = 6+1.8 = .7.8.
bias = 7.8; //
verdict = "";
message += " <h3><br>Parameters <br> bias = " + bias
+ " //to deal with negations and stability"
+ "<br> factors: " + factors
+ "<br> inputs = " + inputs
+ "<br> weights = " + weights
+ "<br> threshold = " + threshold;



let sum = bias;
for (let i = 0; i < inputs.length; i++) {
sum = sum + inputs[i] * weights[i];
}



//Print real numbers with two decimals only.
sum = 1.0 * Math.round(sum * 100) / 100;
message += "<br> sum = " + sum + "</h3> <br>";


if (sum > threshold) {
verdict = "<h1>sum > threshold <br> Verdict: a 1 is recognized!</h1>";
} else {
verdict = "<h1>sum < threshold <br> Verdict: the bitmap es not that of 1!</h1>";
}
message += verdict;
document.getElementById("myDoc").innerHTML = message;
</script>
```

36 ***Ejercicio sin respuesta.*** *Ejecute el programa* `PerceptronOne.html` *para probar la afirmación de que este perceptrón clasifica correctamente el mapa de bits de 1.*

37 ***Ejercicio.*** *Diseñe un perceptrón para detectar un 0 sobre el* ***problema-0-ó-1*** *Respuesta*

38 ***Ejercicio.*** *Tenemos dos perceptrones aislados, uno para detectar el mapa de bits de 1 y el otro el de 0. ¿Cómo deben ensamblarse en una RN que resuelva el* ***problema-0-ó-1*** *? Respuesta*

Hemos creado un circuito con perceptrones que resuelve el **problema-0-ó-1**. Pero hasta ahora no tenemos idea de la estabilidad de su reacción al ruido. Digamos que los mapas de bits ligeramente diferentes al de 1 aún se clasificarán como UNO, pero los demasiado diferentes deben clasificarse como NO-UNO. ¿Es eso cierto? Para saberlo, usemos la distancia BpB para conocer la cercanía de los mapa de bits a los objetivos en el **problema-0-ó-1**. Nuestro problema es que todavía tenemos un vacío:

39 ***Ejemplo.*** *Ideemos un método para producir todos los mapas de bits posibles. Implementémoslo en JS e incluyamos una clasificación de cada mapa de bits como 0 o 1 según la distancia BpB. Hagamos que nuestro programa genere los mapas de bits ordenados por distancia.*

En primer lugar, necesitamos determinar el número de mapas de bits. Una imagen $3 \times 3 = 9$ cuyos píxeles son blancos o negros genera $2^9 = 512$ posibles mapas de bits o secuencias binarias. Para generar todos los mapas de bits, enumeramos todos los números del 0 al 512, transformamos cada número en notación binaria y rellenamos los espacios vacíos con 0.

40 ***Ejemplo.*** *El número 5 es 101 en binario. Dado que nuestros mapas de bits tienen 9 bits, nos faltan 6 caracteres que deben ser rellenados con 0's. La cadena resultante es 000000101. Este es un mapa de bits que este procedimiento no repetirá. El código está en el programa* `allBitmaps.html`*, que clasifica cada mapa de bits como una distorsión de 1 ó 0. Para ello utiliza la distancia BpB: la distancia mínima decide la identidad. Al final de los resultados, vemos una lista de todos los mapas de bits ordenados por la distancia al objetivo más cercano, ya sea el mapa de bits de 1 o de 0.*

41 ***Ejercicio.*** *En el programa* `allBitmaps.html`*, los mapas de bits se presentaban como cadenas. Desarrollar un programa que los reporte como dibujos, en formato de 3 líneas de 3 x 3 píxeles. Así, el mapa de bits 111000111 debe verse así:*

```
111
000
111
```

Ahora es natural decir que este mapa de bits corresponde a 0. ¿Está de acuerdo su sentido común con la clasificación que ofrece la distancia BpB? *Respuesta*

Una vez que sabemos cómo producir todos los mapas de bits, podemos proceder a juzgar directamente el comportamiento de nuestro circuito bajo variabilidad. Esto se hace en el programa `CompareTwoClassifications.html`. Este programa compara la clasificación de los mapas de bits según la distancia BpB con la dada por nuestro circuito de dos perceptrones.

Nuestros resultados son:

La RN de dos perceptrones, el primero que detecta 1 y el segundo 0, reacciona ante un pequeño ruido de forma debida: cuando hay un cambio en un solo sitio,

y la distancia BpB es 1, todos los mapas de bits mutados se clasificaron correctamente, ya sean deformaciones de 1 ó 0. El espacio de estabilidad constaba de 10 mapas de bits que se clasificaron correctamente, correspondientes al mapa de bits original más 9 mutantes, uno para cada sitio en la cadena binaria. Ejemplo: mapa de bits original de cero = 111101111, mapa de bits mutante 111111111 que el Perceptron0 clasificó correctamente como cero y el Perceptron1 como DESCONOCIDO.

42 ***Ejercicio.*** *En el programa* `CompareTwoClassifications.html` *cambie los sesgos de los dos perceptrones para mejorar su tolerancia sin perder confiabilidad. Esta significa un número máximo de coincidencias pero sin desajustes o errores de clasificación.* Respuesta

RETROSPECTIVA: Al observar los resultados positivos de nuestra teoría que se ha implementado en nuestros programas, afirmamos que son correctos. ¡Sigamos adelante hacia nuestro objetivo final!

Nuestro propósito final es involucrar a la Evolución en nuestras discusiones para mejorar al máximo la tolerancia pero sin cometer errores de clasificación. Es por eso que necesitamos poblaciones de nuestros perceptrones y RNs. Por tanto, nuestra primera tarea es encapsularlos en clases JS.

2.8 Perceptrón como clase JS

Usaremos perceptrones como componentes básicos de las RNs. Entonces, encapsulémoslos como una clase JS para tener la posibilidad de crear muchas instancias para usarlas como celdas.

43 ***Ejemplo.*** *Transformamos el programa* `Perceptron.html` *en una clase y reproducimos su salida. Nuestra clase es el Perceptrón. Una clase es la abstracción de un conjunto de datos y sus correspondientes métodos o procedimientos para procesarlos y publicar resultados. Una clase es similar al plano de una casa. Para convertir el plano en realidad, necesitamos una instanciación. La creación de instancias se realiza mediante el modificador* `new` *seguido del nombre de la clase:*

```
let myPerceptron = new Perceptron(0.2, factors, [0, 1, 0, 1, 0, 1], [
```

44 ***Ejemplo.*** *El código primordial de la clase Perceptron JS es el siguiente,* `PerceptronClass.html:`

```
<!DOCTYPE html>
<html>


<head>


<meta charset="utf-8" />
<title>PerceptronClass.html. </title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<meta name="generator" content="Bluefish 2.2.12, Visual Studio Code">
<meta name="author" content="jose">
<meta name="date" content="2023-07-05T13:13:00-0500">
<meta name="copyright" content="Licence = unconditional">
<meta name="keywords" content="Perceptron, AI, artificial intelligence">
<meta name="description" content=" This program defines the perceptron class with getters and calculates its output.">
<meta name="ROBOTS" content="NOINDEX, NOFOLLOW">
<meta http-equiv="content-type" content="text/html; charset=UTF-8">


</head>


<body>


<div id="myDoc"></div>




<script>
// Declaration of the class
class Perceptron {
//Declaration of the parameters of the perceptron
bias;
factors;
inputs;
weights;
threshold;



//This definition of the class is similar to the drawing of a plan for a house.
//It is not the house, it is just a plan.
//With a plan, many houses can be made.
//With a class many instances can be created,
//one by one,
//thanks to the constructor method.
//When one instance is created,
//all parameters can be initialized at once:
constructor(bias, factors, inputs, weights, threshold) {
this.bias = bias;
this.factors = factors;
this.inputs = inputs;
this.weights = weights;
this.threshold = threshold;
}


//Methods to retrieve information
get bias() {
return this.bias;
}


get factors() {
return this.factors;
}


get inputs() {
return this.inputs;
}


get weights() {
return this.weights;
}
```

```
// Summ Getter
get sum() {
return this.calcSum();
}
// Method
calcSum() {
let sum = this.bias;
for (let i = 0; i < this.inputs.length; i++) {
sum = sum + this.inputs[i] * this.weights[i];
}
return sum;
}



//Threshold getter
get threshold() {
return this.threshold;
}


*getData() {
yield this.bias;
yield this.factors;
yield this.inputs;
yield this.weights;
yield this.threshold;
}


}//End of class



var
message = "<h1>Perceptron <br> Shall one continue reading? </h1>";
/*
bias = 0.2; //a small temptation to continue reading
factors = " easy reading, interesting, good graphics, " +
"<br> instructive, abundantly cited, written by GPT-chat";
inputs = [0, 1, 0, 1, 0, 1];
weights = [0.2, 0.8, 0.6, 0.9, 0.2, -2];
threshold = 0.8;


*/


const factors = " easy reading, interesting, good graphics, " +
"<br> instructive, abundantly cited, written by GPT-chat";
let myPerceptron = new Perceptron(0.2, factors, [0, 1, 0, 1, 0, 1], [0.2, 0.8, 0.6, 0.9, 0.2, -2], 0.8, -1);




message += "<br> threshold = " + myPerceptron.threshold;
// document.getElementById("myDoc").innerHTML = message;


message += " <h3><br>Parameters <br> bias = " + myPerceptron.bias
+ " //a small temptation to continue reading"
+ "<br> factors: " + myPerceptron.factors
+ "<br> inputs = " + myPerceptron.inputs
+ "<br> weights = " + myPerceptron.weights
+ "<br> threshold = " + myPerceptron.threshold
+ "<br> <br> All data = " + [...myPerceptron.getData()];


document.getElementById("myDoc").innerHTML = message;



message += "<br> <br> sum = " + myPerceptron.sum + "</h3> <br>";



document.getElementById("myDoc").innerHTML = message;


let advise = "";
if (myPerceptron.sum > myPerceptron.threshold) {
advice = "<h1>sum > threshold <br> Advice: YES!</h1>";
```

```
} else {
advice = "<h1>sum < threshold <br> Advice: NO!</h1>";
}
message += advice;
document.getElementById("myDoc").innerHTML = message;
</script>



</body>


</html>
```

45 ***Ejercicio sin respuesta.*** *Ejecute este programa* `PerceptronClass.html`.

El programa `PerceptronClass.html` todavía no es útil para la Evolución porque los valores de cada instancia de la clase están congelados.

46 ***Ejemplo.*** *Habilitemos la clase Perceptron con la posibilidad de modificar sus valores. Esto se hace mediante el siguiente código,* `PerceptronClass2.html`:

```
<!DOCTYPE html>
<html>


<head>


<meta charset="utf-8" />
<title>PerceptronClass2. </title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<meta name="generator" content="Bluefish 2.2.12, Visual Studio Code">
<meta name="author" content="jose">
<meta name="date" content="2023-07-08T13:13:00-0500">
<meta name="copyright" content="Licence = unconditional">
<meta name="keywords" content="Perceptron, AI, artificial intelligence">
<meta name="description"
content="This program defines the perceptron class with setters and getters and re-calculates its output.">
<meta name="ROBOTS" content="NOINDEX, NOFOLLOW">
<meta http-equiv="content-type" content="text/html; charset=UTF-8">


</head>


<body>


<div id="myDoc"></div>




<script>


//================================
//====== The perceptron class ====
//================================


// Declaration of the class
class Perceptron {
//Declaration of the parameters of the perceptron
bias;
factors;
inputs;
weights;
threshold;
```

```
//This definition of the class is similar to the drawing of a plan for a house.
//It is not the house, it is just a plan.
//With a plan, many houses can be made.
//With a class many instances can be created,
//one by one,
//thanks to the constructor method.
//When one instance is created,
//all parameters can be initialized at once:
constructor(bias, factors, inputs, weights, threshold) {
this.bias = bias;
this.factors = factors;
this.inputs = inputs;
this.weights = weights;
this.threshold = threshold;


}


// GETTERS: methods to retrieve information


get bias() {
return this.bias;
}


get factors() {
return this.factors;
}


get inputs() {
return this.inputs;
}


get weights() {
return this.weights;
}



// Summ Getter
get sum() {
return this.calcSum();
}
// Method
calcSum() {
let sum = this.bias;
for (let i = 0; i < this.inputs.length; i++) {
sum = sum + this.inputs[i] * this.weights[i];
}
return sum;
}



//Threshold getter
get threshold() {
return this.threshold;
}



get decision() {
return this.calcDecision();
}



calcDecision() {
let d = -1;
if (this.sum > this.threshold) {
d = 1;
} else {
d = 0;
}
return d;
}
```

```
*getData() {
yield this.bias;
yield this.factors;
yield this.inputs;
yield this.weights;
yield this.threshold;
}


// Summ Getter
get message() {
return this.getMessage();
}
// Method
getMessage() {
let message = "<h1>Perceptron <br> Shall one continue reading? </h1>"
+ " <h3><br>Parameters <br> bias = " + this.bias
+ " //a small temptation to continue reading"
+ "<br> factors: " + this.factors
+ "<br> inputs = " + this.inputs
+ "<br> weights = " + this.weights
+ "<br> threshold = " + this.threshold
+ "<br> <br> All data = " + [...this.getData()];
return message;


}



//SETTERS, methods to redefine information


set bias(x) {
this.bias = x;
}


set factors(x) {
this.factors = x;
}


set inputs(x) {
this.inputs = x;
}


set weights(x) {
this.weights = x;
}
set threshold(x) {
this.threshold = x;
}
set decision(x) {
this.decision = x;
}


}//End of class




//Evolutionary Environments for Neural Networks



/* Factors to decide whether to continue reading.
bias = 0.2; //a small temptation to continue reading
factors = " easy reading, interesting, good graphics, " +
"<br> instructive, abundantly cited, written by GPT-chat";
inputs = [0, 1, 0, 1, 0, 1];
weights = [0.2, 0.8, 0.6, 0.9, 0.2, -2];
threshold = 0.8;
*/
var
factors = " easy reading, interesting, good graphics, " +
"<br> instructive, abundantly cited, written by GPT-chat";
myPerceptron = new Perceptron(0.2, factors, [0, 1, 0, 1, 0, 1], [0.2, 0.8, 0.6, 0.9, 0.2, -2], 0.8, -1);
```

```
message = myPerceptron.getMessage();
message += "<br> <br> sum = " + myPerceptron.sum + "</h3> <br>";


let advise = "";
if (myPerceptron.sum > myPerceptron.threshold) {
advice = "<h1>sum > threshold <br> Advice: YES!</h1>";
} else {
advice = "<h1>sum < threshold <br> Advice: NO!</h1>";
}
message += advice;
document.getElementById("myDoc").innerHTML = message;


//==========REDEFINITION=============


message += "<br> <h1> REDEFINITION</h1> <br> ";


//Some values are redefined
myPerceptron.threshold = 0.0001;
myPerceptron.inputs = [0, 0, 0, 0, 0, 0];
myPerceptron.inputs[0] = 1;


message += myPerceptron.getMessage();


message += "<br> <br> sum = " + myPerceptron.sum + "</h3> <br>";



if (myPerceptron.decision) {
advice = "<h1>sum > threshold <br> Advice: YES!</h1>";
} else {
advice = "<h1>sum < threshold <br> Advice: NO!</h1>";
}
message += advice;
document.getElementById("myDoc").innerHTML = message;



</script>



</body>


</html>
```

47 ***Ejercicio sin respuesta.*** *Ejecute este programa* `PerceptronClass2.html`. *La diferencia entre estas dos versiones es la siguiente: en el primer programa incluimos toda la maquinaria necesaria para recuperar información de cualquier instancia de la clase. Esto se hizo con captadores. Pero para modificar el contenido de la información, necesitamos configuradores. Se prueba la posibilidad de cambiar los valores y los resultados se muestran en la salida del programa. Esto permite la Evolución, que es cambio sobre cambio. Posiblemente fuimos exageradamente legalistas y el código podría ser mucho más simple.*

Con la clase Perceptron, podemos crear muchas instancias de ella para que tengamos muchas celdas que formen un circuito o RN. Y dado que necesitamos poblaciones de RNs para programar la Evolución, debemos encapsular una RN como una clase JS.

2.9 El código para una RN como clase JS

Reformulemos el programa `CompareTwoClassifications.html` que representa una RN con dos perceptrones en el lenguaje de clases JS. Esto se hace en el programa `CompareTwoClassifications2.html`. Observe que la declaración de la RN como clase involucra su arquitectura. Esto implica que no tenemos una única declaración para todas las RNs sino que debemos definir una clase específica para cada arquitectura. Por eso debemos entender muy claramente el modus operandi del código.

La arquitectura de la RN actual consta de: 9 entradas se corresponden con los 9 bits de cada mapa de bits. Estos ingresan como datos brutos a cada una de las dos celdas de la única capa que toma decisiones finales. Esta capa se llama capa de decisión. El primer perceptrón emite 1 cuando se detecta UNO. El segundo genera 1 cuando se detecta CERO. Interpretamos la salida 0 como DESCONOCIDO.

48 ***Ejercicio.*** *Agregue algunas estadísticas al programa* `CompareTwoClassifications2.ht` *para ver el papel del grado de tolerancia sobre la confiabilidad. Ponga a prueba la conclusión del autor:* ***la RN empleada puede alcanzar 260 coincidencias sin ningún error. Este es nuestro top que será tomado como la perfección.*** *Respuesta*

Necesitamos estos resultados para compararlos con los de la Evolución que se investigan en la siguiente sección.

3 APRENDIZAJE

Una RN es una abstracción del sistema neuronal. Las RNs artificiales han captado la atención de todo el mundo porque no solo pueden ejecutar cálculos sino que también pueden conectarse a una instalación externa que en conjunto simula el aprendizaje.

Prestemos atención a un hecho muy simple que nos ayudará a no confundir conceptos: una RN es un circuito especial que siempre responde a cualquier entrada. Una RN calcula una función matemática que a cualquier entrada dada permitida asocia la salida resultante. Esta función es la **función generada**. Una RN puede existir por sí misma, está congelada y no puede aprender. Pero podemos crear un entorno para la RN, en el que el aprendizaje se produzca de forma natural.

La idea del aprendizaje es realmente simple: la función generada y la espec pueden diferir y se puede medir la diferencia o el error. La siguiente es una forma común, pero no la única, :

$$Error = \sum_{entradas}(f_{espec}(entrada) - f_{generada}(entrada))^2$$

En general el error no es nulo. **El aprendizaje ocurre cuando uno hace cambios que sistemáticamente disminuyen el error.** Así, las redes neuronales se enfrentan a una prueba en la que se proporciona un subconjunto de patrones para su clasificación. Si las respuestas coinciden con las correctas que ya se conocen no se realiza ningún cambio, pero si se detecta una discrepancia se proponen cambios para mejorar el rendimiento, es decir, disminuir en futuras pruebas el error total de la RN.

Para minimizar el error, nos enfrentamos a la tarea de encontrar el mínimo de una cáscara de huevo o paraboloide multidimensional. El error se denomina en la literatura **función de costo**, expresión tomada de la jerga de los problemas de optimización.

Existen muchos métodos para encontrar buenas soluciones aproximadas a problemas de optimización. Para las RNs, el siguiente es el estándar.

3.1 Algoritmo de propagación reversa

Cuando se puede utilizar una medida de error cuadrática, minimizarla corresponde a encontrar la base de un paraboloide. Este mínimo se puede alcanzar en el aprendizaje cuando se realizan pequeños cambios en la dirección en la que fluye el agua para llegar al fondo de la cáscara del huevo. Este es el famoso Algoritmo de Propagación Reversa. Esta metodología es intuitiva, sencilla y -en mi opinión- extremadamente eficiente.

49 ***Ejercicio sin respuesta.*** *Disfrute el excelente video de 3Blue1Brown, ([1] 2018) sobre el método de propagación reversa.*

50 ***Ejercicio para aquellos que quieren ser expertos.*** *Lea la maravillosa exposición con demostraciones interactivas sobre RNs por Nielsen ([8] 2018). El algoritmo de propagación reversa se puede encontrar en el Capítulo 2. En el Capítulo 3 se encuentra una prueba informal y comprensible de que las RNs pueden aproximar cualquier función.*

51 ***Desafío.*** *En el enfoque habitual de las RNs, se debe minimizar un paraboloide. Esa es una tarea fácil porque sólo tiene un mínimo. Pero todo el mundo dice que las RNs son muy complejas. Desmantele esta contradicción.*

En un proceso de aprendizaje, se compara el objetivo con el estado real del sistema y se ejecutan cambios con la intención de minimizar la discrepancia

entre ambos. Este es un problema de optimización. Se pueden dar buenas aproximaciones a este tipo de problemas mediante diversos métodos. El nuestro es la Evolución.

3.2 Entornos Evolutivos para Redes Neuronales (EERN)

Nuestra implementación evolutiva del aprendizaje consiste en someter a una población de RNs a sufrir mutaciones y recombinaciones que producen cambios al azar. La instancia que mejor realiza la tarea predefinida se selecciona para clonarla y formar la siguiente generación. Y así sucesivamente, hasta que el emparejamiento sea perfecto o aparezca una meseta o un proceso pseudo-cíclico sin mayor evolución.

Es interesante que el método de propagación reversa esté inspirado en la geometría directa e implemente cambios deterministas. Por el contrario, la Evolución utiliza cambios aleatorios y sus recombinaciones y aquellos que mejoran la función son seleccionados para reproducción. ¿Puede la Evolución superar otros métodos para minimizar el error de una RN?

Si ese fuera el caso, sería inesperado. El problema es encontrar el mínimo de un paraboloide para el cual ya existe un método muy intuitivo y eficiente: el algoritmo de retropropagación. Sin embargo, la aplicación de la Evolución al aprendizaje de RNs ya estaba bien establecida en 1993 (Radcliffe ([11] 1993, Yao [17] 1993) para optimizar los pesos de conexión de una NN, su arquitectura y también su reglas de aprendizaje. Por lo tanto, este tema es una de las propuestas más fructíferas en el campo porque los dos últimos objetivos están por fuera del Algoritmo de propagación reversa. Aquí aprenderemos sobre un modelo de juguete el modus operandi de este enfoque. Es simple, natural y ha sido profundamente estudiado (Kenneth, [6] 2019).

Además, y lo que es más importante para nosotros, este enfoque combina dos elementos importantes de la ciencia moderna: la evolución y las RNs, ambas relacionadas con nuestra comprensión de la naturaleza del hombre. Por tanto, nuestra tarea es estudiar este enfoque para abordar cuestiones de alto nivel. Comencemos.

52 ***Definición. Un Entorno Evolutivo para Redes Neuronales (EERN)*** *es una población de redes neuronales que sufren recurrentemente cambios ciegos y su recombinación y cuya aptitud o rendimiento para optimizar un objetivo determinado determina el procedimiento de clonación.*

Pasemos ahora a la aplicación más sencilla de la Evolución: hemos diseñado una RN que decide si una imagen corresponde a UNO ó CERO. Hemos encontrado una manera de ampliar la estabilidad de la RNs cambiando el sesgo de cada

perceptrón o célula. Usemos la Evolución para responder una pregunta simple: ¿Cómo y cuánto se puede mover el sesgo para maximizar el rendimiento y la confiabilidad de la RN?

Veamos el algoritmo para simular la Evolución que utiliza nuestro EERN.

3.2.1 Implementación

Nuestro problema es hacer que la Evolución encuentre los valores de los sesgos de dos perceptrones que maximicen la capacidad de la RN para reconocer un UNO o un CERO incluso cuando sus imágenes están distorsionadas.

Nuestro algoritmo comienza con una población de redes neuronales generadas al azar. A continuación, la población está sujeta a un ciclo recurrente que consiste en

1. Evaluación del puntaje o idoneidad de cada RN que viene dada por el número de coincidencias correctas menos un castigo por sus errores de clasificación. Cada RN se prueba en todos los mapas de bits y se otorga la puntuación correspondiente, que determina directamente su aptitud.

2. Se clona al más apto para generar una nueva población con el mismo número de individuos.

3. La población está sujeta a mutación: se realizan algunos cambios en los clones.

4. La población se somete a recombinación o intercambio de información.

¿Qué esperamos de nuestro EERN? Dado que el más apto es el único que se reproduce, se espera que la población aprenda a clasificar correctamente los mapas de bits a medida que avanzan las generaciones. El programa correspondiente es `ENN1.html`. Está incluido en el archivo comprimido que acompaña esta publicación. Todos los programas se pueden ejecutar en un navegador web una vez que se ha descomprimido este archivo. En el celular todo fluye por instinto.

Debemos decidir antes de continuar si toleraremos errores que ocurren cuando la RN clasifica una imagen como UNO, pero la distancia BBB dice que es CERO, o viceversa, o cuando las dos percepciones se contradicen -un evento que nunca sucede.

53 ***Ejemplo.*** *Si la distancia BBB dice que el mapa de bits es 1 pero la RN dice que es 0, tenemos una contradicción. Esto significa que el perceptrón 1 respondió 0 (DESCONOCIDO) mientras que el perceptrón 0 respondió 1 (el mapa de bits representa 0). Si la distancia BBB dice que el mapa de bits es*

0, pero la RN dice que es 1, obtenemos una contradicción. En este caso, el perceptrón 1 respondió 1 (es 1) y el perceptrón 0 respondió 0 (DESCONOCIDO). Si el perceptrón 1 responde 1 (es 1) y también el perceptrón 0 (es 0), obtenemos otra contradicción -algo que nunca sucede. Si ambos perceptrones responden 0 (DESCONOCIDO), no hay contradicción, ni tampoco punto bueno.

Entonces, probamos cada asignación de sesgo dándole a la RN correspondiente una prueba de clasificación de un número determinado de mapas de bits. Definamos que existe acuerdo o coincidencia cuando la RN coincide con la distancia BBB en la clasificación de un mapa de bits determinado, y desacuerdo o desajuste cuando no. Para nosotros, la BBB es la ley. Entonces, si el número de coincidencias es m y el de no coincidencias es n, la aptitud del NN es

$Aptitud = m - n$

Esta medida adolece de un problema: una coincidencia pesa tanto como un desajuste. Esto suena mal porque dice que no es difícil compensar los desajustes. Sin embargo, es barato fabricar errores de clasificación. El azar es más que suficiente. Presentamos un ejemplo más abajo. El azar es capaz incluso de producir 256 aciertos y 256 desaciertos en un test con 512 mapas de bits.

Por eso, para salvar nuestro honor, debemos evitar las clasificaciones erróneas a cualquier precio. Nuestra sugerencia es cambiar la función de aptitud de la siguiente manera:

$Aptitud = m - kn$

Hemos descubierto que $k = 5$ parece funcionar bien para nuestros ejemplos. Esto se implementó en el programa `EvolNN.html`.

54 ***Ejercicio sin respuesta.*** *Ejecute el programa `EvolNN.html` y verifique que, con alta frecuencia, 20 individuos evolucionando durante un máximo de 70 generaciones son suficientes para encontrar el valor de los sesgos de los dos perceptrones que generan el mayor número de coincidencias (260 según el programa `CompareTwoClassifications3.html`) y sin discrepancias. La tasa de mutación fue 0,1 y la tasa de recombinación fue 0,2. Esto demuestra que la implementación más simple de la Evolución funciona perfectamente. Advertencia: este programa y todos los siguientes consumen mucha memoria. Necesita 8 Gigabytes de RAM, cerrar todas las aplicaciones y utilizar una ventana privada dedicada de su navegador. Correrá durante unos 4 minutos antes de detenerse. Para detener un programa en Firefox, haga clic dentro de su ventana y arrastre hacia abajo. Aparecerá un cuadro de diálogo con la opción de detener el programa.*

55 ***Ejercicio sin respuesta.*** *El desarrollo de software genera demasiados errores o bugs. La depuración de errores de programación debe ser de dominio*

público. Nuestra directiva es que ver lo que hace un programa debe estar abierto a todos. La implementación en JS exige mantener en memoria las salidas que se quieren ver y que podrían publicarse una vez que el programa se detenga por sí solo. Esta tarea consume demasiada memoria. Por eso,los programas se deben ejecutar sobre ejemplos pequeños. Por lo tanto, modifique, alrededor de la línea 300, el número de generaciones que evolucionarán a un máximo de 3. Seleccione la línea 1383 en lugar de 1382. De lo contrario, el programa puede provocar que su sistema operativo colapse. Active las banderas de impresión, a partir de la línea 318 y siguientes, una por una. Si enciende una bandera, apague todas las demás. Luego, puede verificar el procesamiento de su método seleccionado.

56 ***Ejercicio.*** *Añada al programa* `EvolNN.html` *la posibilidad de ver el número de coincidencias junto con el número de desajustes. Verifique que no se produzcan desaciertos para una puntuación de 260, tal como se prometió. De lo contrario, cree e implemente una solución.* *Respuesta*

57 ***Ejercicio.*** *Demuestre que por mera aleatoriedad, cualquier RN puede cumplir algunos valores de la función de especificación aunque también comete errores de clasificación. Haga un programa que pruebe esto sobre la tarea de encontrar los valores óptimos del sesgo en el problema 0-ó-1.* *Respuesta*

58 ***Desafío.*** *Basándonos en el ejercicio anterior, hemos hecho una terrible generalización: la aleatoriedad es incapaz de producir RNs de alta calidad. Es terrible porque niega la posibilidad del azar para lograr cualquier cosa posible. Remedio: fabricar un montón de funciones y poner a prueba la generalización antes mencionada. Sugerencia: La distancia BpB divide el conjunto completo de mapas de bits en dos subconjuntos separados. En general, cada separación del conjunto universal en dos clases distintas también define una función: a un mapa de bits en un subconjunto, la función asocia, digamos, A, y a mapas de bits en el otro, B. Para resolver el desafío, haga tantos cortes como sea posible y corra y ejecute las estadísticas correspondientes.*

59 ***Ejercicio.*** *Ponga a prueba el poder de la Evolución para ajustar solo los valores de umbral de los dos perceptrones de la RN que fue diseñada por un humano y se muestra en el Programa* `EvolNN.html` *y* `EvolNN2.html`*.* *Respuesta*

60 ***Desafío.*** *Resuelva el siguiente enigma: Nos parece que el costo de ajustar los sesgos es mucho menor que el de ajustar los umbrales. ¿Es eso cierto o simplemente un efecto de muestreo o, tal vez, un efecto secundario de la discontinuidad de la codificación? Si esto es un hecho real, ¿cómo se puede explicar?*

Consejo para disminuir el gasto de memoria: estudiar cómo se gestionó la impresión de diversos resultados en el programa `EvolNN3.html` *y adaptarlo al programa* `EvolNN2.html` *para silenciar la impresión detallada. Esto aceleraría la velocidad de la computación unas 100 veces y al mismo tiempo mantendría el uso de memoria bastante bajo. Así se podría reunir suficientes datos para poder hacer estadísticas.*

61 ***Ejercicio.*** *Ponga a prueba el poder de la Evolución para ajustar solo los valores de los pesos de los dos perceptrones de la RN que fue diseñado por un humano y se muestra en el Programa* `EvolNN.html` *y* `EvolNN2.html`*. Compare sus resultados con los del autor: 100.000 generaciones no fueron suficientes para realizar la tarea, que era alcanzar la máxima puntuación posible que interpretamos como perfección.* Respuesta

¿Cuál podría ser la causa de nuestro fracaso en lograr que la evolución tuviera éxito? Posiblemente se debió a una falta de recombinación con nuevas ideas porque la nueva generación se formó clonando al campeón. Para remediar esto, sustituimos la clonación del campeón por un método en el que la puntuación o aptitud de cada genoma se interpretaba como una probabilidad de reproducirse mediante clonación. El comportamiento fue exactamente el mismo. Esto se hizo con el programa `EvolNN4C.html`. En un experimento intensivo, este programa se ejecutó con 100 genomas, tasa de mutación = 0,1, tasa de recombinación = 0,1. Se observó un comportamiento similar: la Evolución alcanza una buena calidad bastante rápido pero cae en un mar absorbente de mediocridad del que no se encontró salida a lo largo de 100.000 generaciones. Por lo tanto, se descubrió que **la firma inconfundible de la Evolución era una persistente submediocridad**.

62 ***Ejercicio.*** *¿Podemos afirmar que la Evolución no puede alcanzar la perfección y que por tanto es falsa?* Respuesta

63 ***Ejercicio.*** *¿Podemos afirmar que la persistente submediocridad, la firma inconfundible de la evolución, es por sí sola suficiente para falsificar la teoría de la evolución?* Respuesta

64 ***Graduación. Programe y estudie la batalla de la Evolución contra la complejidad para una NN con un solo perceptrón.***

4 CONCLUSIÓN

Hemos hecho todo lo posible para demostrar que una neurona funciona como una compuerta booleana generalizada tolerante al ruido. Para conseguirlo, hemos

simulado a una neurona mediante un perceptrón, introducido en 1958, y a una red neuronal mediante una red neuronal artificial. Utilizando un computador ordinario con 8 Gigabytes de RAM, también hemos demostrado que la Evolución puede ajustar algunos parámetros aislados de los perceptrones de un NN con 2 perceptrones para maximizar las coincidencias pero sin un solo error de clasificación. Este resultado fue válido para sesgos y umbrales, pero fallamos con los valores de peso. En el último caso, el desempeño se mantuvo preferentemente por debajo de la mediocridad.

Hemos comprobado que **la Evolución de las RNS con una capa y dos perceptrones tiene una firma inequívoca: debería estar empantanada en un mar de sub-mediocridad**. Esta humilde conclusión demuestra que la Evolución como dogma para explicar nuestra existencia es una vergüenza, pero que es maravillosa como investigación a la luz del Método Científico. Más claramente: *Las generalizaciones sobre la Evolución sin pruebas detalladas y predicciones reproducibles son mera palabrería*.

5 RESPUESTAS A LOS PROBLEMAS

4, página 7. El cero recorre el borde de la cuadrícula 3×3. Está representado por el texto binario.

```
111
101
111
```

eso se convierte en la secuencia binaria 111101111.

6, page 10. Si evaluamos la fórmula

$\sim n_1 \wedge n_2 \wedge \sim n_3 \wedge \sim n_4 \wedge n_5 \wedge \sim n_6 \wedge \sim n_7 \wedge n_8 \wedge \sim n_9$

sobre la cadena 010010010, obtenemos

$\sim FALSO \wedge VERDADERO \wedge \sim FALSO \wedge \sim FALSO \wedge VERDADERO \wedge \sim FALSO \wedge \sim FALSO \wedge VERDADERO \wedge \sim FALSO$

que se reduce a

$VERDADERO \wedge VERDADERO \wedge VERDADERO \wedge VERDADERO \wedge VERDADERO \wedge VERDADERO \wedge VERDADERO \wedge VERDADERO \wedge VERDADERO$

eso se simplifica a VERDADERO. Para cualquier otra cadena, hay un cero o FALSO en algún lugar y, por lo tanto, la fórmula general representa cero o FALSO.

8, página 10. Otro circuito resulta si aplicamos la propiedad asociativa de las compuertas AND:

$p \wedge q \wedge r = ((p \wedge q) \wedge r) = (p \wedge (q \wedge r))$.

Aparecen más circuitos si aplicamos la ley de Morgan:

$p \wedge q = \sim (\sim p \vee \sim q)$

9, página 10. El módulo inferior debe detectar la secuencia 111101111 que codifica 0. Esto se hace mediante un circuito que realiza la fórmula booleana.

$n_1 \wedge n_2 \wedge n_3 \wedge n_4 \wedge \sim n_5 \wedge n_6 \wedge n_7 \wedge n_8 \wedge n_9$

10, página 10. La función espec del discriminador completo es: la primera lámpara debe emitir 1 para la secuencia 010010010 y 0 para cualquier otro valor, mientras que la segunda lámpara debe responder 1 a 111101111 y 0 para cualquier otra entrada. La fórmula que detecta 0 debe permanecer separada de la de 1 porque cada módulo produce su salida. Entonces, la salida del discriminador es una cadena con dos bits, el primero es 1 cuando se detecta 1 y el segundo es 1 cuando se detecta 0. De lo contrario, son cero.

12, página 13. Recomendamos no utilizar circuitos booleanos para resolver el problema del ruido con datos reales debido a una explosión exponencial del número de compuertas necesarias. Solo consideremos que con imágenes en una cuadrícula con $28 \times 28 = 784$ píxeles, como es habitual en la vida real, podemos codificar $2^{784} = \infty$ diferentes mapas de bits.

15, página 14. Nuestro código para la Distancia BpB se utilizará abundantemente en los programas que siguen. Es el siguiente:

```
// Bit-by-bit distance between two binary bitmaps
//that are 9 chars long.
//For each discrepancy, one is added to the distance.
function distance(s, t) {
let d = 0;
let n = s.length;
console.log("s=" + s + "s.length = " + n);
for (let i = 0; i < n; i++) {
if (s.charAt(i) === t.charAt(i)) {
d++;
}
}
let distance = n - d;
console.log("s = " + s + " t = " + t + " d = " + distance);
```

```
return distance;
}
```

Hay 512 mapas de bits posibles y de ellos 256 fueron clasificados por el método de la distancia como 1 y los otros 256 como 0. Por tanto, la probabilidad de cada evento es $1/2$.

19, página 16. Una neurona AND es una compuerta que tiene varias entradas, digamos k. Sea T un número algo menor que 1. Si la suma de todas las entradas supera kT, la neurona se activa y responde 1 o VERDADERO. De lo contrario, no dispara. Por lo tanto, una neurona AND es una puerta AND modificada que responde VERDADERO cuando la entrada global es lo suficientemente fuerte, un poco menor que k haciéndola algo estable ante el ruido de atenuación.

20, página 16. UNA NOT-NEURONA es una compuerta que cuando la entrada es menor que un umbral determinado cercano a cero, se activa y responde con 1 o VERDADERO, pero si la entrada supera el umbral, la neurona no genera nada, CERO o FALSO o DESCONOCIDO.

22, página 19. Encontraremos la expresión matemática de la función calculada por un perceptrón.

Sea P un perceptrón que acepta n factores, cuyas entradas son e_j tales que $j = 0, \ldots, n$ (no incluido), con pesos w_j, un peso para cada factor, sesgo b, suma total $S = \sum w_j \times e_j + b$ y umbral T, luego P calcula la función f:

$$f(S) = \begin{cases} 1 & \text{si} \quad S < T \\ 0 & \text{si} \quad S \geq T \end{cases} \tag{1}$$

28, página 22. Especifiquemos el valor de los parámetros de un perceptrón para simular una compuerta AND booleana y una neurona AND booleana.

La compuerta AND puede ser simulada por un perceptrón de la siguiente manera: Dos entradas que admiten valores booleanos, 0 o 1. Dos pesos, ambos iguales a 1. Valor de sesgo igual a 0. El valor umbral es igual a 2. Si ambos valores de entrada son VERDADEROS, la suma que llega al ejecutor de umbral es 2, lo que coincide con el umbral: el perceptrón genera VERDADERO. En cualquier otro caso, el perceptrón genera FALSO.

Para obtener una neurona AND que sea estable frente al ruido, se puede cambiar el valor umbral en la neurona AND anterior de 2 a 1,8. Con este nuevo valor, el perceptrón responde VERDADERO incluso cuando las señales están atenuadas. Pero no responde VERDADERO en otros casos si sólo el ruido es

moderado. Porque si el ruido es extremo, los valores cercanos a cero podrían convertirse en 0.97 y, por tanto, podrían aparecer falsos positivos.

29, página 22. Para simular una NOT-GATE o compuerta NOT, podemos aprovechar el hecho de que si $y = 1 - x = -x + 1$, entonces si $x = 1$, $y = 0$, y también $y = 1$ cuando $x = 0$. Por lo tanto, el perceptrón correspondiente tiene una entrada, un peso igual a -1, un valor de sesgo igual a 1 y un umbral igual a 1. Obsérvese el papel determinante del sesgo.

La NEURONA-NOT correspondiente puede tener un umbral igual a $0,5$. Entonces, si la entrada x es $x < 0.5$, lo que representa un cero, $1 - x \geq 0.5$ y el perceptrón responde VERDADERO. Pero si la entrada $x >= 0.5$, que representa 1, $1 - x < 0.5$ y el perceptrón responde FALSO.

31, página 22. Para implementar con un perceptrón la función spec `haz esto pero no aquello`, podemos probar con los siguientes valores para los parámetros del perceptrón: debe tener dos entradas, una para p con peso 1, otra para q con peso -1, $sesgo = 1$ y un umbral de 1.5. La suma es $S = p - q + 1$. El perceptrón generará 1 o VERDADERO cuando $S \geq 1.5$. Esto sucede si y solo si $p - q + 1 \geq 1.5$ o $p > 1.5 + q - 1$ o $p > q + 0.5$. Suponiendo que q y p son positivos y menores que 1, q debe ser menor que 0.5, lo que representa FALSO, no hagas eso, y $p > 0.5$, que representa VERDADERO, haz esto.

37, página 25. Esta es una ligera variación del programa anterior. El código está contenido en el programa `PerceptronZero.html`.

38, página 25. Los dos perceptrones reciben la misma entrada pero producen salidas diferentes que deben mantenerse separadas. La arquitectura resultante se muestra en la figura 5.

41, página 26.

El programa `allBitmaps2.html` reporta los mapas de bits en modo visual. El Autor se identifica con la clasificación presentada que ofrece la distancia BpB para las distancias 0, 1 y 2. Para la distancia 3 surgen dudas que se vuelven más fuertes para las distancias 4 y 5. En caso de fuerte duda, sería preferible clasificar el mapa de bits como DESCONOCIDO.

42, página 27. Si se aumenta moderadamente la tolerancia de cada perceptrón, todo está bien. De lo contrario, aparecen errores y contradicciones.

48, página 34. En el programa `CompareTwoClassifications3.html` medimos el rendimiento de cada par de sesgos que determinan la tolerancia de

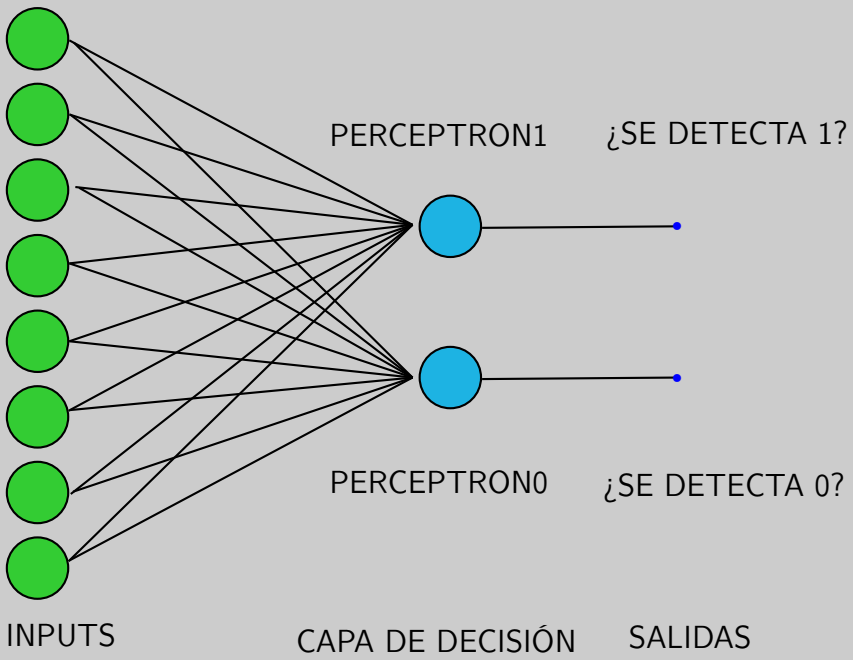

Figure 5: Esta RN fue diseñada por la inteligencia humana. Su tarea es reconocer imágenes ligeramente distorsionadas de 0 o 1. ¿Cómo funciona? Tiene 9 entradas, una para cada píxel en la cuadrícula binaria 3×3. Las entradas representan un mapa de bits posiblemente distorsionado de las imágenes de 0 o 1. Todas las entradas están conectadas a cada celda o perceptrón de la capa de decision, que contiene solo 2 celdas. La celda superior debe generar 1 cuando se presenta una imagen de 1 y cero en caso contrario. La celda inferior debe generar 1 cuando se presenta 0 y 0 en caso contrario. Las imágenes pueden estar un poco distorsionadas por el ruido y/o la variabilidad. El output global es una cadena que tiene el output del perceptron 1 seguido del output del perceptron 0.

los perceptrones. Reportamos el número de coincidencias, desajustes y una puntuación que se define como:

$$puntaje = \#Coincidencias - \#Discrepancias$$

Se produce una coincidencia cuando un perceptrón coincide con la distancia BBB y el otro dice que no sabe. Cuando ambos perceptrones informan DESCONOCIDO, no hay coincidencia ni discrepancia. En cualquier otro caso se produce una falta de coincidencia y denota una contradicción entre un perceptrón y la distancia BBB. Nuestras estadísticas dicen:

```
Sesgo del perceptrón P1 = 7,8
Sesgo del perceptrón P0 = 2,8
Puntuación = 20
númeroDeCoincidencias 20
númeroDeErrores 0
númeroDeIncógnitas 492
Total 512
```

```
Sesgo del perceptrón P1 = 8,8
Sesgo del perceptrón P0 = 3,8
Puntuación = 92
númeroDeCoincidencias 92
númeroDeErrores 0
númeroDeIncógnitas 420
Total 512
```

```
Sesgo del perceptrón P1 = 9,8
Sesgo del perceptrón P0 = 4,8
Puntuación = 260
número de coincidencias 260
númeroDeErrores 0
númeroDeIncógnitas 252
Total 512
```

```
Sesgo del perceptrón P1 = 10,8
```

```
Sesgo del perceptrón P0 = 5,8
Puntuación = 302
númeroDeCoincidencias 372
número de discrepancias 70
númeroDeIncógnitas 70
Total 512


Sesgo del perceptrón P1 = 11,8
Sesgo del perceptrón P0 = 6,8
Puntuación = 8
número de coincidencias 260
número de discrepancias 252
númeroDeDesconocidos 0
Total 512


Sesgo del perceptrón P1 = 12,8
Sesgo del perceptrón P0 = 7,8
Puntuación = -328
númeroDeCoincidencias 92
númeroDeErrores 420
númeroDeDesconocidos 0
Total 512
```

Se ve que la tolerancia se puede ampliar sin conseguir contradicciones hasta los valores del sesgo de $P1 = 9.8$ y $P0 = 4.8$ cuando se encuentran 260 coincidencias entre 512 posibilidades. Una mayor expansión crea contradicciones aunque la puntuación aumenta. Luego, las contradicciones superan las coincidencias.

56, página 39. El programa `EvolNN2.html` muestra que nuestras expectativas se han cumplido: no se producen discrepancias cuando se encuentran 260 coincidencias, que es el mayor número de coincidencias limpias. La población tenía 20 individuos, la tasa de mutación fue de 0,2 y la tasa de recombinación fue de 0,2. Aumentamos la tasa de mutación porque, al observar los resultados publicados en la consola, descubrimos que había un punto de estancamiento creado por la discontinuidad en la codificación de números:

El programa debe encontrar los valores correctos de dos sesgos, uno alrededor de 9 y el otro alrededor de 4. Entonces, cuando el programa ofreció 10 y 4, 10 no se pudo modificar porque se debería haber cambiado dos valores al mismo tiempo, poniendo 0 en el primer dígito y 9 en el segundo. Dado que la tasa de

mutación era demasiado baja, esto sucedía raramente, retardando el flujo de la evolución. Por eso, aumentamos un poco los valores para obtener los que tienen actualmente, una tasa de mutación igual a 0,2 y una tasa de recombinación igual a 0,2 para una población de 20 individuos. Con esto, el programa completó la tarea de conseguir 260 coincidencias y ningún desajuste en un número variable de generaciones: 4, 11, 10, 11, 47, 13, 10.

57, página 39. El programa `RandomBiases.html` genera los sesgos de una RN con valores al azar. Se le asigna la tarea de clasificar 512 mapas de bits y se reporta el número de coincidencias con la distancia BbB. Todo el experimento se repite algunas veces y se presenta una distribución de frecuencia absoluta. Nuestro resultado final dice: **Con una frecuencia relativamente alta, la aleatoriedad puede producir RNs que tienen 256 coincidencias y 256 discrepancias, o 130 coincidencias y 382 errores de clasificación.**

59, página 39. El programa `EvolNN3.html` utiliza el poder de la Evolución para ajustar los valores umbral de dos perceptrones de la RN habitual. Se realizó con 20 individuos, la tasa de mutación fue de 0,4 y la tasa de recombinación fue de 0,4. La tarea se completó en las generaciones 28, 6, 88 y 190. Con una tasa de mutación igual a 0,3 y una recombinación de 0,5, el número de generaciones para completar la tarea fue: 13, 26, 57, 11, 327, 360, 61, 46, 302, 392, 3, 226.

61, página 40. El programa `EvolNN4.html` muestra que la aleatoriedad puede proponer valores para los pesos que producen RNs que a la vez hacen tanto aciertos como desaciertos. A partir de esto, la Evolución lucha por disminuir los desajustes y luego aumentar el número de coincidencias. Muy pronto, la Evolución puede llegar incluso a 232 puntos de 260 y sin desajustes, pero simplemente para caer a vagar en un mar de submediocridad que se ve interrumpido por excursiones mucho más allá de la mediocridad, pero no hasta la perfección. Esto es independiente de los valores de tasa de mutación y recombinación.

Tal comportamiento se observó, por ejemplo, en otro experimento con una tasa de mutación igual a 0,1 y una tasa de recombinación igual a 0,1, una población de 100 genomas. Más exactamente, el programa se ejecutó durante 100 generaciones y los puntajes máximos por generación fueron: 64 48 40, 40, 48 56, 56, 48 116 56 116, 116, 116, 96, 96, 96, 96, 96, 96, 96 , 96, 96, -40 56, 56 64 56 74 64 116, 116 74 116, 116, 116, 116, 116, 116 32 8, 8, 64 56 96 56, 56, 56 116 96 116, 116, 11 6, 116, 116, 96, 96, 96, 96, 117 118, 118 116 132 16 56 116, 116 130 116, 116, 116, 118 116 132 116, 116, 116, 116 48 64, 64 48 64 116 56 96 104 112 152 116 128 116, 116, 116, 116, 96, 96, 96, 96, 56. Recordemos que la puntuación máxima alcanzable es 260.

La perfección no se alcanzó ni siquiera cuando se ejecutaron 100.000 generaciones con una población de 20 genomas, una tasa de mutación igual a 0,15 y una tasa de recombinación de 0,35.

62, página 40. Hemos probado la Evolución durante 100.000 generaciones y no se encontró la perfección. Pero nadie puede excluir que en la generación 100001 la tarea no esté resuelta.

63, página 40. Hemos puesto la Evolución a prueba a lo largo de 100.000 generaciones y hemos descubierto que la Evolución vaga en un mar de submediocridad sin dar ninguna señal de ir a una salida. Entonces, tomamos esto como la firma inequívoca de la Evolución de la Evolución de las NNs con una capa y dos perceptrones. Como vemos, nuestra ventana de observación ha sido demasiado estrecha para sacar una conclusión más seria.

6 BIBLIOGRAFÍA

References

[1] 3Blue1Brown 2018
But what is a neural network? . | Chapter 1 of four, Deep learning, Youtube Neural Network Tutorial in four parts.
https://www.youtube.com/watch?v=aircAruvnKk 35

[2] Mayank Banoula Last updated on May 10, 2023
What is Perceptron: A Beginners Guide for Perceptron .
https://www.simplilearn.com/tutorials/deep-learning-tutorial/perc
22

[3] Fodstad H.
The neuron theory. Stereotact Funct Neurosurg. 2001;77(1-4):20-4. doi: 10.1159/000064596. PMID: 12378051. .
https://pubmed.ncbi.nlm.nih.gov/12378051/
4

[4] Subrata Ghosh, Pushpendra Singh, Jhimli Manna, Komal Saxena, Pathik Sahoo, Soami Daya Krishnanda, Kanad Ray, Jonathan P. Hill & Anirban Bandyopadhyay (2022)
The century-old picture of a nerve spike is wrong: filaments fire, before membrane, . Communicative & Integrative Biology, 15:1, 115-120, DOI:

10.1080/19420889.2022.2071101
https://www.tandfonline.com/doi/full/10.1080/19420889.2022.207110
5

[5] Mitch Glickstein (2000)
Golgi and Cajal: The neuron doctrine and the 100th anniversary of the 1906 Nobel Prize .
https://www.cell.com/current-biology/pdf/S0960-9822(06)01203-6.pd
23

[6] Kenneth O. Stanley , Jeff Clune, Joel Lehman, and Risto Miikkulaine
Designing neural networks through neuroevolution . Nature Machine intelligence | VOL 1 | JANUARY 2019 | 24–35 |
https://www.nature.com/articles/s42256-018-0006-z
36

[7] Diana Kwon (2023)
The quest to map the mouse brain. Nature | Vol 620 | 17 August 2023 | 686-687
https://www.nature.com/articles/d41586-023-02559-9
23

[8] Michael Nielsen (2018)
Neural Networks and Deep Learning .Determination Press 2015.
http://neuralnetworksanddeeplearning.com/
Code samples for "Neural Networks and Deep Learning"
https://github.com/mnielsen/neural-networks-and-deep-learning
35

[9] Piccolino M. 1998
Animal electricity and the birth of electrophysiology: the legacy of Luigi Galvani. . Brain Res Bull. 1998 Jul 15;46(5):381-407. doi: 10.1016/s0361-9230(98)00026-4. PMID: 9739001.
https://pubmed.ncbi.nlm.nih.gov/9739001/
5

[10] Jan Evangelista Purkyně. (2023, August 6)
In Wikipedia. .
https://en.wikipedia.org/wiki/Jan_Evangelista_Purkyn%C4%9B
4

[11] Radcliffe, N.J. 1993
Genetic set recombination and its application to neural network topology optimisation. Neural Comput & Applic 1, 67–90
https://link.springer.com/article/10.1007/BF01411376
36

[12] Rodríguez José 2023
Christianity and the Avatar Interpretation of the Brain v2 .
https://www.ejristos.com/eternity/files/AvatarCerebro.pdf
https://www.ejristos.com/eternity/files/AvatarBrain.pdf
17

[13] F. Rosenblatt, 1960
Perceptron Simulation Experiments . in Proceedings of the IRE, vol. 48, no. 3, pp. 301-309, March 1960, doi: 10.1109/JRPROC.1960.287598.
https://ieeexplore.ieee.org/abstract/document/4066017

[14] tutorialspoint. Undated. Active by 2023.
Logic Gates .
https://www.tutorialspoint.com/computer_logical_organization/logi
9

[15] W3schools (2023)
Perceptrons .
https://www.w3schools.com/ai/ai_perceptrons.asp
21

[16] Xin He (University at Buffalo) undated, active by 2023
Propositional Logic .
https://cse.buffalo.edu/~xinhe/cse191/Classnotes/note01-1x2.pdf
9

[17] Xin Yao 1993
A Review of Evolutionary Artificial Neural Networks . International Journal of Intelligent Systems Vol 8: 539-567
https://onlinelibrary.wiley.com/doi/epdf/10.1002/int.4550080406
36

[18] Yoder, Lane. (2009).
Explicit Logic Circuits Discriminate Neural States. PloS one. 4. e4154. 10.1371/journal.pone.0004154.

https://journals.plos.org/plosone/article?id=10.1371/journal.pone
22

Todos los links citados estaban activos el 8/XI/2023