Redes neuronales artificiales

Las *redes neuronales artificiales* difieren drásticamente de las biológicas. Estas pretenden imitar el comportamiento de las biológicas, pero reduciendo todo a números y operaciones matemáticas que pueden ser realizadas con sencillez por los *ordenadores.*

Las *RRNN* son la base de la *Inteligencia Artificial.* Las herramientas de *IA* que han ganado popularidad últimamente, como *ChatGPT,* están compuestas por grandes redes neuronales que han sido entrenadas para operar con los números de tal forma que parezca que saben entender y usar el lenguaje natural.

Pero este tipo de *RRNN* tienen una gran desventaja: son mucho más simples que las biólogicas, y por ello, son capaces de transmitir y manejar menos información por neurona.

Aunque esto no tiene gran importancia en tareas básicas, ya que puede ser solventado si aumentamos el número de neuronas artificiales; limita en gran medida la inteligencia máxima de la red neuronal, pero todavía no se ha llegado a ese límite.

Origen

La *inteligencia* es una *virtud* que tenemos los seres humanos. Somos capaces de pensar, comprender y asimilar como ningún ser es capaz. Gracias a esta habilidad que tenemos, podemos construir, investigar, hipotetizar, inventar, aprender.

Aunque hay unos pocos animales que tienen esa facultad también, como los *gorilas,* somos la única especie que conocemos con esa capacidad tan desarrollada.

Pero, ¿y si pudiésemos *crear* inteligencia? ¿y si pudiésemos inventar algo que pudiese pensar por sí mismo? Aristóteles pensaba que la capacidad de razonar era exclusiva del ser humano, pero ¿y si no lo fuese?

Los científicos de la década de los 40 ya habían probado a crear programas que tuviesen *fórmulas* muy complejas o que se aplicasen *reiteradamente,* pero no habían conseguido crear un sistema que tuviese la habilidad de usar la lógica como lo hacemos nosotros.

Pero entonces, en 1943, unos científicos tomaron un enfoque radicalmente diferente para intentar solventar este problema. Podrían, en vez de intentar dar con un *algoritmo complejo* que simulase la lógica, intentar recrear un *modelo matemático* de cómo nosotros, los seres humanos, pensamos.

Warren McCulloch y *Walter Pitts* fueron los pioneros que se adentraron en el estudio de la *neurociencia computacional,* al publicar en ese año *A Logical Calculus of the Ideas Immanent in Nervous Activity.*

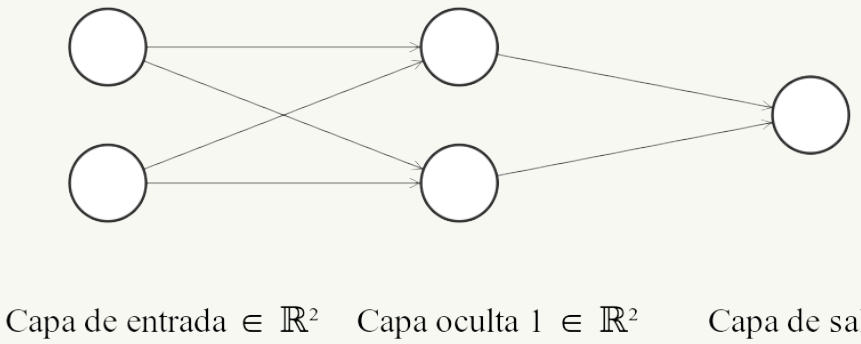
El modelo que ellos presentaron ha sufrido muchos cambios con los años, y difiere en gran medida de cómo funcionan las redes neuronales artificiales actuales, pero sentó

las bases y el interés de la población por las capacidades de esta nueva tecnología.
Eso quería decir que el mecanismo de razonamiento humano podía ser simplificado a esa simulación de *neuronas* y *enlaces.* A este primer modelo, la primera red neuronal artificial de la *Historia,* se le denominó *La neurona de McCulloch-Pitts.*

Neuronas

Ahora nos adentraremos en el funcionamiento de las *RRNN artificiales.*

La



Representación gráfica de una red neuronal artificial.
Figura 1.

figura 1 muestra la representación de una RRNN artificial. Lo primero que llama la atención son unos *grandes círculos:* las neuronas.
Las neuronas son *entidades* que contienen un número, denominado *sesgo.* Se encargan de recibir información por los *enlaces* detrás de ellas (a la izquierda), modificar esa información y transmitirla para delante.
En la mayoría de las redes neuronales artificiales, a diferencia de las biológicas, las neuronas que están adelante no pueden comunicar información a las de atrás, pero sí *vice versa.* Por esa razón, una vez que la información llega al final de la red neuronal, su actividad cesa por completo y no hay nada que hacer.
Aunque sí que hay redes neuronales más avanzadas, como las que usa *ChatGPT o3-mini,* que puede devolver información a neuronas previas, esta *tecnología* presenta muchos problemas (relacionados con la coordinación y la causalidad) por lo que no hablaremos de ella.
Las neuronas, como puedes observar en la ilustración, están organizadas en *capas* verticales que pueden tener un número *arbitrario* de neuronas. Existen tres tipos de capas:

1. Capa *de entrada:* las primeras neuronas, que no tienen enlaces previos, se activan cuando se les da información de entrada como si viniese de enlaces.
2. Capas *ocultas*: las neuronas en estas capas transformarán la información original en otra, que nos resulta más valiosa. Puede haber un número indefinido de capas ocultas.

3. Capa *de salida:* las últimas neuronas, una vez que han transformado la información, dan como resultado, como salida, esa información.

Las redes neuronales necesitan un proceso de aprendizaje para aprender qué información de toda la información de entrada es la valiosa, para devolverla en las neuronas de salida.

Al final, una red neuronal artificial de este tipo, denominado *perceptrón multicapa,* no es más que una *gran función matemática.* Tú das unos valores de entrada (como los argumentos en una función), por dentro se realizan cálculos, y se devuelven unos datos de salida.

Es por esa razón, por que es una gran función matemática, que los *ordenadores* pueden simular este tipo de redes neuronales rápidamente.

Enlaces

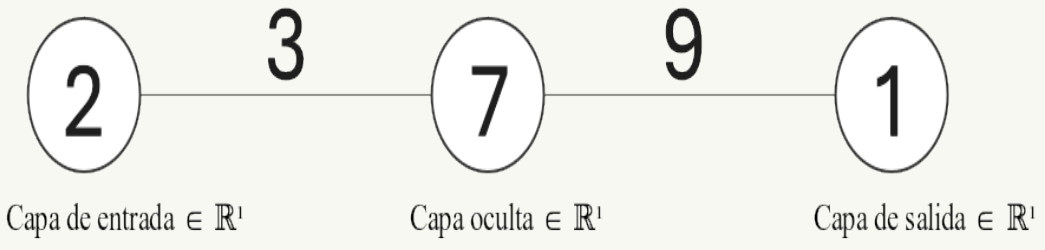
Las conexiones, o enlaces, son las *líneas* que conectan las neuronas. Estos enlaces transportan la información de una neurona transmisora a una receptora. Para los perceptrones multicapa hay algunas restricciones sobre a dónde pueden ir estos enlaces:

1. Los enlaces no pueden volver a neuronas previas.
2. Los enlaces no pueden comunicar dos neuronas en la misma capa.
3. Los enlaces no pueden transmitir información a través de otras capas de neuronas.
4. Los enlaces deben comunicar todas las neuronas de una capa con la siguiente.

   Estos enlaces, aparte de transmitir información, también la modifican. Las conexiones almacenan un número, denominado *peso,* que modificará el valor que recibe de la neurona transmisora y se lo dará a la receptora.

Cálculo simple

Observa la siguiente red neuronal.



RRNN artificial simple con valores.
Figura 2.

En la *figura 2* he añadido los valores de sesgo de las neuronas dentro de las neuronas y los valores de peso de los enlaces encima de ellos.

Antes de calcular nada, tenemos que conocer las operaciones que usaremos para calcular el resultado de la red neuronal:

1. Los sesgos: los sesgos en las neuronas sumarán el valor entrante con su valor de sesgo y se lo transmitirán a todos los enlaces salientes.
2. Los pesos: los pesos en los enlaces multiplican el número entrante por su valor de peso y se lo transmiten a la neurona receptora.

Anotaremos todos los valores, para no tener que estar recurriendo a la imagen y para poder usar nomenclatura específica:

$$b_0^{(0)} = 2 \qquad b_0^{(1)} = 7 \qquad b_0^{(2)} = 1$$

$$w_{0,0}^{(1)} = 3 \qquad w_{0,0}^{(2)} = 9$$

Los sesgos se nombran con la nomenclatura $b_{\text{nº de neurona}}^{(\text{nº de capa})}$. Tanto el número de capa como el de neurona empieza en 0, siendo cero la primera capa o la primera neurona de la capa. Los pesos se nombran de la siguiente forma: $w_{\text{nº de receptora, " " transmisora}}^{(\text{nº de capa receptora})}$. En el caso de los sesgos y los pesos, el superíndice se escribe entre paréntensis para diferenciarlo de los exponentes.

Vamos a operar esta red neuronal con un número cualquiera, como x. Para arrancar la red neuronal, daremos este valor de entrada a la única neurona de la capa de entrada.

Lo primero que sucede es que al llegar a la neurona, x se encuentra con el valor $b_0^{(0)}$ y por tanto se suma:

$$r(x) = x + b_0^{(0)}$$

He llamado también a la *transformación* de la red neuronal como la *función* $r(x)$. Acto seguido, el resultado $x + b_0^{(0)}$ se propaga por el único enlace, y se multiplica por su peso:

$$r(x) = (x + b_0^{(0)}) \cdot w_{0,0}^{(1)}$$

y al llegar a la siguiente neurona se le suma su sesgo:

$$r(x) = (x + b_0^{(0)}) \cdot w_{0,0}^{(1)} + b_0^{(1)}$$

siguiente conexión:

$$r(x) = ((x + b_0^{(0)}) \cdot w_{0,0}^{(1)} + b_0^{(1)}) \cdot w_{0,0}^{(2)}$$

hasta la última neurona, la neurona de salida:

$$r(x) = ((x + b_0^{(0)}) \cdot w_{0,0}^{(1)} + b_0^{(1)}) \cdot w_{0,0}^{(2)} + b_0^{(2)}$$

Podríamos simplificar, pero no es necesario. Esta función matemática modela los resultados de nuestra red neuronal. Podemos darle algunos valores:

$r(-5) = -17$ $r(-1) = 91$ $r(0) = 118$ $r(1) = 145$ $r(5) = 253$

$r(-100) = -2582$ $r(-10) = -152$ $r(10) = 388$ $r(100) = 2818$

> Si representamos gráficamente esta función, veremos que es una función lineal con pendiente 27, y es que todas las RRNN artificiales con una sola neurona por capa se simplifican a una función de *grado* 1. En cambio, una RRNN con dos neuronas por capa puede llegar a simular cualquier función continua.

En realidad, las redes neuronales artificiales no funcionan de esta forma por dentro. Generalmente cuentan con unas *funciones de activación* que limitan el valor que las neuronas pueden transmitir.
La más utilizada se denomina $ReLU()$ (por sus siglas en inglés, *Rectified Linear Unit*). Esta función se define como:

$$ReLU(x) = \max(0, x)$$

y pretende simular (de forma simplificada) la activación y desactivación de las neuronas biológicas, que solo se *disparan* en caso de que se alcance un *umbral de activación.*
Nuestra *fórmula* para la RRNN quedaría así con la función de activación:

$$r(x) = ReLU(ReLU(x + b_0^{(0)}) \cdot w_{0,0}^{(1)} + b_0^{(1)}) \cdot w_{0,0}^{(2)} + b_0^{(2)}$$

A la última neurona no se le aplica la *función de activación* para mantener el conjunto de resultados posibles. Así, obtenemos resultados diferentes:

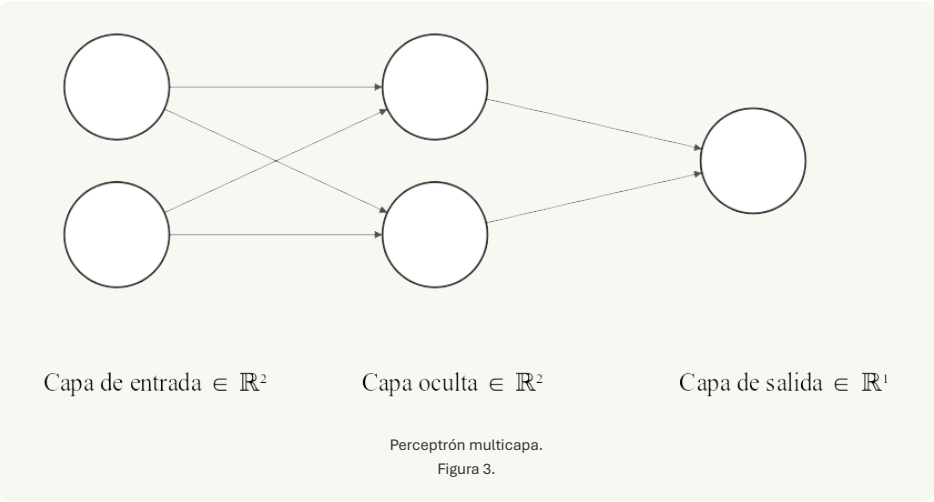
$r(-5) = 64$ $r(-1) = 91$ $r(0) = 118$ $r(1) = 145$ $r(5) = 253$

$r(-100) = 64$ $r(-10) = 64$ $r(10) = 388$ $r(100) = 2818$

Cálculo

Si tenemos más de una neurona por capa, recurriremos a *conceptos matemáticos* más avanzados para operar nuestra red neuronal.

Vamos a nombrar los sesgos y pesos que hay en esta red neuronal. Como puedes comprobar en la *figura 3,* aunque solo hemos aumentado el número total de neuronas en dos, el número de variables ha aumentado drásticamente. Empezaremos nombrando los sesgos:



Perceptrón multicapa.
Figura 3.

$$\underbrace{b_0^{(0)} \quad b_1^{(0)}}_{\text{Sesgos capa 0}} \qquad \underbrace{b_0^{(1)} \quad b_1^{(1)}}_{\text{Sesgos capa 1}} \qquad \underbrace{b_0^{(2)}}_{\text{Sesgo capa 2}}$$

y ahora los pesos:

$$\underbrace{w_{0,0}^{(1)} \quad w_{1,0}^{(1)}}_{\text{Enlaces neurona 0 capa 0}} \qquad \underbrace{w_{0,1}^{(1)} \quad w_{1,1}^{(1)}}_{\text{Enlaces neurona 1 capa 0}}$$

$$\underbrace{w_{0,0}^{(2)} \quad w_{0,1}^{(2)}}_{\text{Enlaces neuronas capa 1}}$$

Ya no podemos computar el resultado de forma *lineal* como hicimos antes y hacerlo resultaría tedioso y lioso. No podemos hacerlo con variables. Pero sí podemos hacerlo si usamos *álgebra lineal.* Para ello, reuniremos todos los valores de sesgo de cada capa en un vector:

$$b^{(0)} = \begin{pmatrix} b_0^{(0)} \\ b_1^{(0)} \end{pmatrix} \qquad b^{(1)} = \begin{pmatrix} b_0^{(1)} \\ b_1^{(1)} \end{pmatrix} \qquad b^{(2)} = \begin{pmatrix} b_0^{(2)} \end{pmatrix}$$

Aunque estemos organizando todos estos valores en vectores, realmente no existe un *significado geométrico* para lo que estamos haciendo. Los vectores aquí son únicamente *listas de números.* Algo similar haremos con los pesos, pero los organizaremos en *matrices:*

$$w^{(1)} = \begin{pmatrix} w_{0,0}^{(1)} & w_{0,1}^{(1)} \\ w_{1,0}^{(1)} & w_{1,1}^{(1)} \end{pmatrix} \qquad w^{(2)} = \begin{pmatrix} w_{0,0}^{(2)} & w_{0,1}^{(2)} \end{pmatrix}$$

Una *matriz* es una lista de vectores. Como ya hemos considerado que un vector es una

lista, una matriz sigue la idea de ser una lista de listas. Dentro de esta matriz, la primera lista (columna vertical) es aquella que involucra todas las conexiones de la primera neurona, la segunda involucra todas de la segunda neurona...
Además, podemos *multiplicar* vectores y matrices con facilidad, como veremos en la siguiente sección.
Ahora vamos a construir la función que nos dará el resultado de esta red neuronal, lo haremos sin profundizar en los valores que hay dentro de las matrices y los vectores. Primero denotaremos a la variable de entrada como X, en mayúscula para simbolizar que es un vector:

$$f(X) = X = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$$

Lo primero que se encuentra el vector X al activar la red neuronal, como vemos en la imagen, es el vector de sesgos de la capa 0, por lo que se lo añadimos y aplicamos la función de activación:

$$f(X) = ReLU(X + b^{(0)})$$

La función de activación $ReLU()$ de un vector aplica a cada componente, es decir:

$$Y = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix} \longrightarrow ReLU(Y) = \begin{pmatrix} ReLU(Y_1) \\ ReLU(Y_2) \\ \vdots \\ ReLU(Y_n) \end{pmatrix}$$

Lo siguiente que se encuentra en nuestra red neuronal son los pesos de la capa 0 a la capa 1, es decir, $w^{(1)}$. Multiplicamos:

$$f(X) = ReLU(X + b^{(0)}) \cdot w^{(1)}$$

Únicamente nos queda por aplicar, en este orden; $b^{(1)}$, $ReLU()$, $w^{(2)}$ y $b^{(2)}$:

$$f(X) = ReLU(ReLU(X + b^{(0)}) \cdot w^{(1)} + b^{(1)}) \cdot w^{(2)} + b^{(2)}$$

Multiplicaciones vector-matriz

Para poder multiplicar un vector por una matriz, se tiene que cumplir la siguiente *condición:* la longitud de la columna del vector tiene que ser igual a la longitud de la fila de la matriz.
Si se cumple esa *condición,* el resultado de la multiplicación es un vector de la misma longitud que la columna de la matriz. Veamos como se operaría el siguiente ejemplo:

$$\begin{pmatrix} x \\ y \end{pmatrix} \cdot \begin{pmatrix} a & d \\ b & e \\ c & f \end{pmatrix}$$

Primero, comprobaremos si la multiplicación está definida: la longitud de columna del vector (2) sí es igual a la longitud de fila de la matriz (2) por lo que podemos operar.
Para cada fila de la matriz multiplicaremos cada término de la columna del vector por su correspondiente en orden de la fila de la matriz y los sumamos. De esta forma obtenemos tres resultados, que los colocaremos en un vector resultante, y su posición responde al número de fila de la matriz que se ha escogido para calcular el resultado:

$$\begin{pmatrix} x \\ y \end{pmatrix} \cdot \begin{pmatrix} a & d \\ b & e \\ c & f \end{pmatrix} = \begin{pmatrix} ax + dy \\ bx + ey \\ cx + fy \end{pmatrix}$$

El vector resultante, como podemos comprobar, tiene la longitud de la columna de la matriz original. Desde el punto de vista de una red neuronal esto tiene sentido, ya que partíamos de dos neuronas que han pasado por una matriz de pesos en la que cada neurona original tenía tres pesos (a tres neuronas); por lo que la cantidad de neuronas finales tenía que ser tres.

Inteligencia artificial

Hemos cubierto los fundamentos de las redes neuronales y cómo funcionan por dentro, pero es posible que te quede la duda de cómo esto se relaciona con la inteligencia artificial que nosotros conocemos.
Como mencionamos en la primera sección, las redes neuronales necesitan de un proceso de aprendizaje en el que aprenden a filtrar la información para devolver la información que nosotros queremos, la que nos resulta valiosa.
Esto lo hacen mediante un proceso muy similar al de aprendizaje humano: mediante prueba, fallo y ajuste de los pesos y los sesgos.
Al principio, las variables internas de la red neuronal son números aleatorios, pero mediante un algoritmo de aprendizaje llamado *backpropagation,* se ajustan los valores de las variables para que el resultado de la red neuronal sea lo más cercano posible a los valores de la información que queremos obtener.
El campo que estudia cómo las redes neuronales *aprenden* (y por tanto, prácticamente toda la *Inteligencia Artificial*) es el *machine learning,* que en los últimos años ha conseguido reducir significativamente el costo de aprendizaje con algoritmos de optimización como *RMSProp.*
Todo esto ha contribuído a que en los últimos años la *IA* esté disponible para todo el mundo de forma gratuita.