

Notas Para el Curso de Introducción a la Ingeniería de Software y Datos

POR JUAN CAMILO MACÍAS

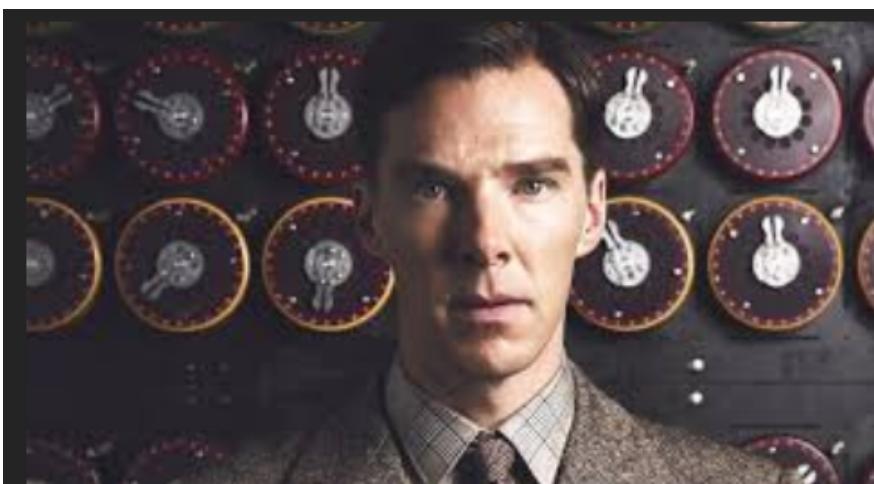
Facultad de Ingeniería y Ciencias Agropecuarias
IUDigital de Antioquia
2024

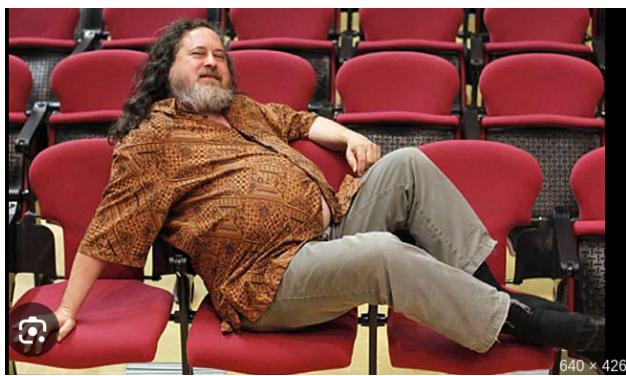
Índice

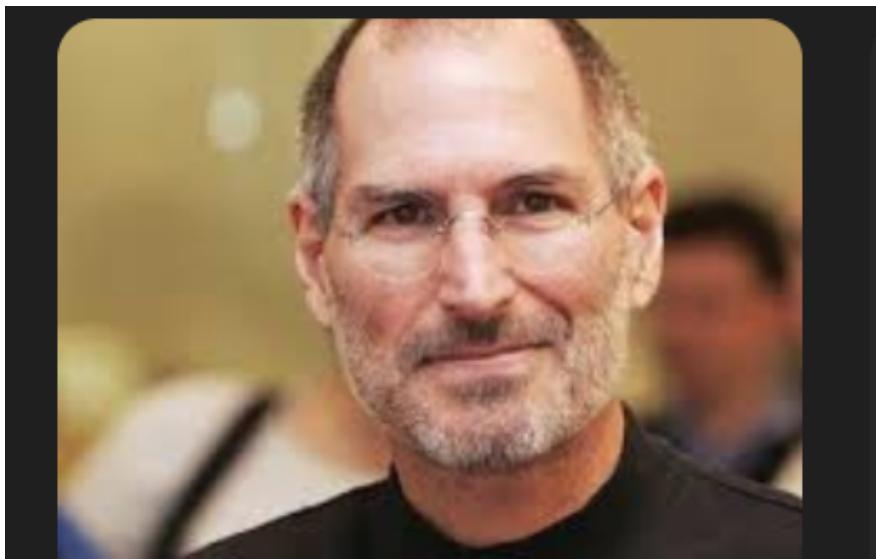
1 Hardware	1
1 El ábaco	7
1.1 En base 10	8
1.2 En base 2	8
2 Lógica de las Operaciones matemáticas	10
Reto 1 :	11
Reto 2:	11
Reto 3:	12
Pregunta:	12
Compuerta NOT	12
2.1 Actividad 4. Construyo las compuertas usando transistores y protoboards	13
2.2 Actividad 5. Usando el computador para simular las compuertas lógicas	13
Reto 1	13
Reto 2	14
2.3 Actividad 6. Sumas de 2 bits	14
Reto:	15
Respuesta	15
Reto	16
2.3.1 Construyendo un Sumador de 2 Bits	16
Un paso al frente: Llevado en cuenta lo que sobra	16
Reto:	16
2 Software	16
3 Datos	18

1 Actividad de conocimientos previos

Levanta la mano si reconoces estos personajes:













2 Hardware

Antes de comenzar, es fundamental que comprendamos mejor nuestra herramienta de trabajo: el computador. La máquina que tienes a tu disposición, por más modesta que sea, es un dispositivo de altísima tecnología, mucho más poderoso que las computadoras utilizadas para los cálculos de la primera bomba atómica o el proyecto que llevó al hombre a la luna. Los pioneros de la ciencia de la computación y la ingeniería de software no contaron con las computadoras que hoy se pueden adquirir en cualquier tienda. Incluso la más básica de las computadoras actuales habría sido un sueño para ellos.

Sin embargo, las limitaciones de las computadoras a las que los programadores tenían acceso en las décadas de los 70, 80 y 90, exigían un profundo entendimiento del funcionamiento interno de estas máquinas, algo que muchos programadores actuales no poseen.

Hoy en día, aprendemos sobre paradigmas de programación, estructuras de datos y lenguajes de programación, pero a menudo sin comprender completamente por qué fueron diseñados de la manera en que lo fueron. Las herramientas modernas facilitan enormemente nuestro trabajo diario, pero también convierten

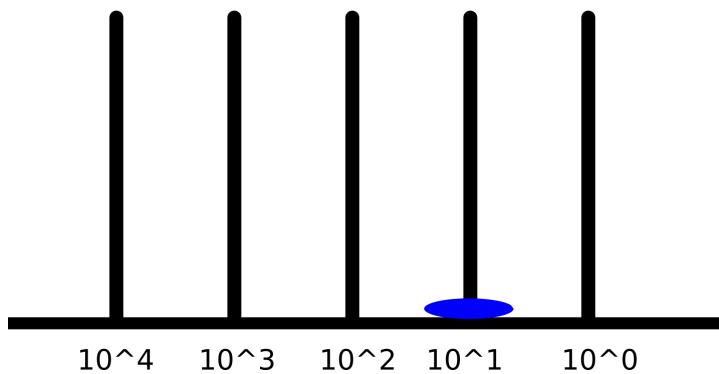
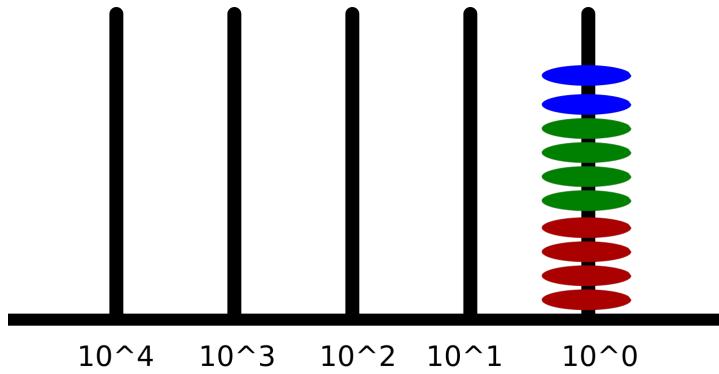
al computador en una caja negra. La capacidad de innovar y proponer nuevos paradigmas o enfoques revolucionarios se encuentra oculta bajo capas y más capas de abstracción, apiladas sobre la más fundamental de todas: el hardware.

Comenzaremos nuestra introducción al mundo de la ingeniería de software y datos desde esta primera capa.

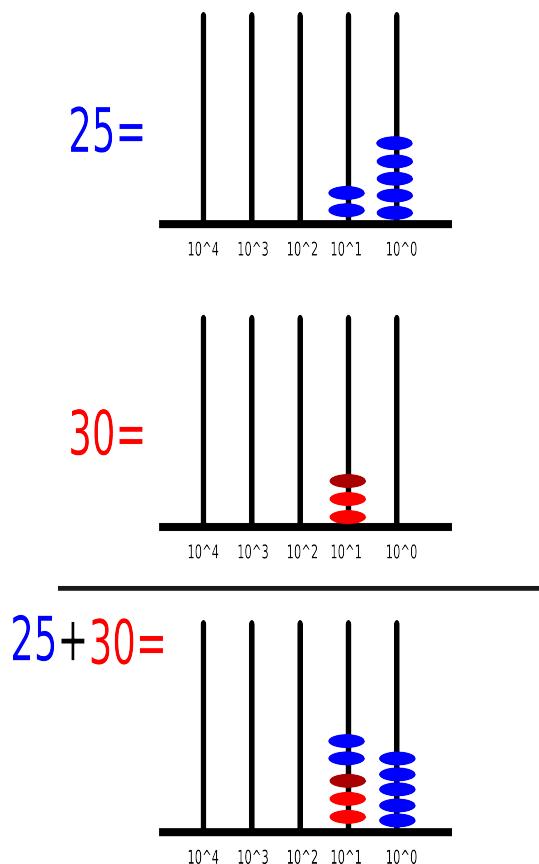
1 El ábaco

1.1 En base 10

Para entender como se hacen cálculos matemáticos con variables recurriremos al ábaco. El ábaco de la figura representamos por ejemplo el número 10:

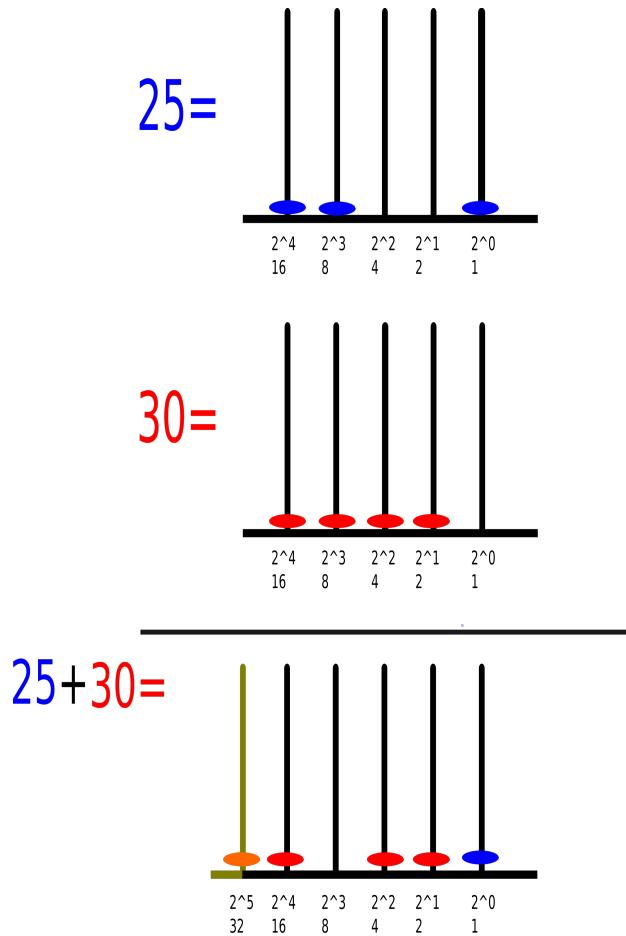


Cada unidad en el asta, representada como $10^0 = 1$, tiene un valor individual de 1. Por lo tanto, las 10 unidades en el primer lugar pueden ser reemplazadas por una sola en el siguiente donde cuenta como $10^1 = 10$. Sucesivamente podemos representar otros números, y su suma, en base 10. Por ejemplo:



1.2 En base 2

Si construimos nuestro computador en base 2, necesitaremos más ‘bits’ para representar los mismos números que usamos en base 10. En el ejemplo anterior, tendríamos que diseñar un computador con un mayor número de ástas.

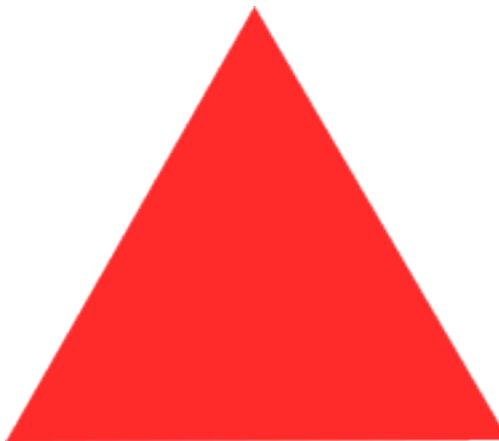


¿Por qué, entonces, los computadores utilizan base 2 si requieren más memoria? Lo veremos en nuestro próximo encuentro. Por el momento la moraleja es la siguiente:

Podemos representar números en como objetos físicos ocupando lugares con significado abstracto.

2 Lógica de las Operaciones matemáticas

Analicemos los siguientes retos:

**Figura 1.**

Reto 1 : Responde VERDADERO o FALSO:

1. ¿El bloque lógico de la figura es rojo AND es triangular?
 - a) VERDADERO
 - b) FALSO
2. ¿El bloque lógico de la figura es rojo AND NO es triangular?
 - a) VERDADERO
 - b) FALSO
3. ¿El bloque lógico de la figura NO es rojo AND es triangular?
 - a) VERDADERO
 - b) FALSO
4. ¿El bloque lógico de la figura NO es rojo AND NO es triangular?
 - a) VERDADERO
 - b) FALSO

Si respondiste las preguntas estás listo para el siguiente reto:

Reto 2: Dadas las expresiones

$$A = \text{El bloque de la figura es rojo.}$$

$$B = \text{El bloque de la figura es triangular.}$$

Si asignamos el valor 1 para indicar que A o B son verdaderos, y el valor 0 para indicar que son falsos, completa la siguiente tabla:

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 1. 1 significa que VERDADERO, 0 significa FALSO.

Reto 3: El circuito contiene 2 interruptores A y B. Representamos un interruptor abierto por 0 y uno cerrado por 1. Completa la tabla:

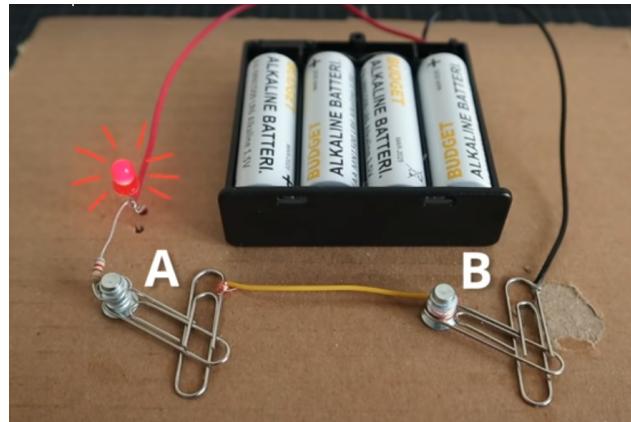


Figura 2. Aquí tenemos un circuito simple con dos entradas A y B. La luz solamente enciende cuando las dos entradas están cerradas. La foto de la izquierda, el circuito pude construirse usando switch de lámparas que venden en las ferreterías. En Yolombó se pueden comprar en «ElectroAlejo» en el parque principal.

A	B	Luz
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 2. Recorda: luz encendida es 1 y apagada es 0.

Pregunta: A qué tabla de verdad se te parece la tabla 2?

Reto 4 Para el circuito de la figura completa la tabla:

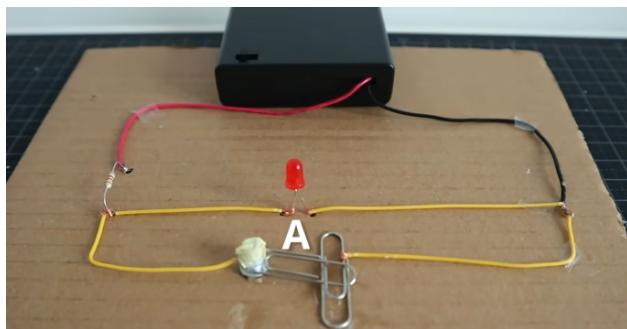


Figura 3.

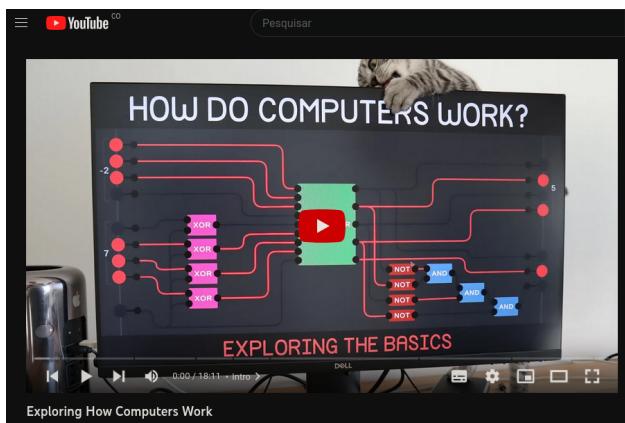
A	Luz
0	1
1	0

Tabla 3.

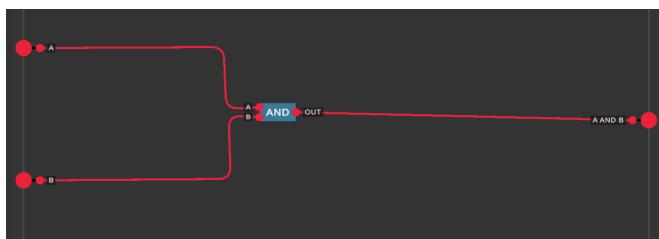
Qué nombre le darías a este operador?

2.1 Usando el computador para simular las compuertas lógicas

Descarga el simulador de compuertas lógicas, <https://github.com/DigitalLogicSimCommunity/Digital-Logic-Sim-CE/releases/tag/v0.39.1>. Puedes explorar el contenido del video de YouTube <https://www.youtube.com/watch?v=QZwneRb-zqA&list=PLn20Vn487hvyG983yv1xMQBe6F7kdDRkK&index=6> para entender como funciona el simulador de compuertas lógicas.

**Figura 4.**

El simulador nos permite experimentar con la compuerta AND

**Figura 5.**

La salida solo está encendida cuando A y B están encendidas.

2.1.1 Compuerta NOT

Otra compuerta interesante es la compuerta NOT, que es simplemente la negación de un valor de verdad. En este caso si A es verdadero su negación NOT A es falso y reciprocamente, si A es falso, NOT A es

verdadero. Entonces la tabla de verdad se vería como:

A	NOT A
1	0
0	1

Tabla 4.

Un circuito que emula este comportamiento es mostrado en la figura:

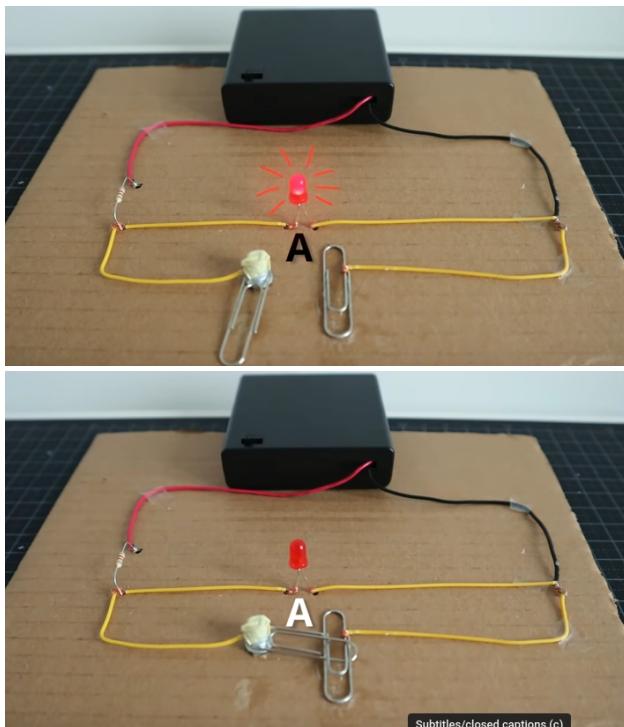


Figura 6.

Reto 1

Usando las compuertas AND and NOT. Construir la compuerta NAND que tiene el comportamiento de la tabla

A	B	Luz
0	0	1
0	1	1
1	0	1
1	1	0

Tabla 5. La luz solamente está apagada cuando las dos compuertas están abiertas. Esta es la tabla de verdad de la conjunción o NAND en Inglés, por ser la negación de AND

Solución:

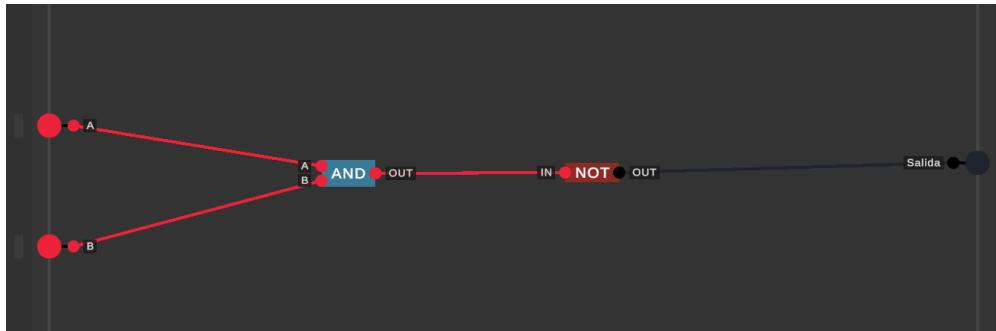


Figura 7.

Este arreglo de compuertas puede resumirse en el circuito integrado NAND



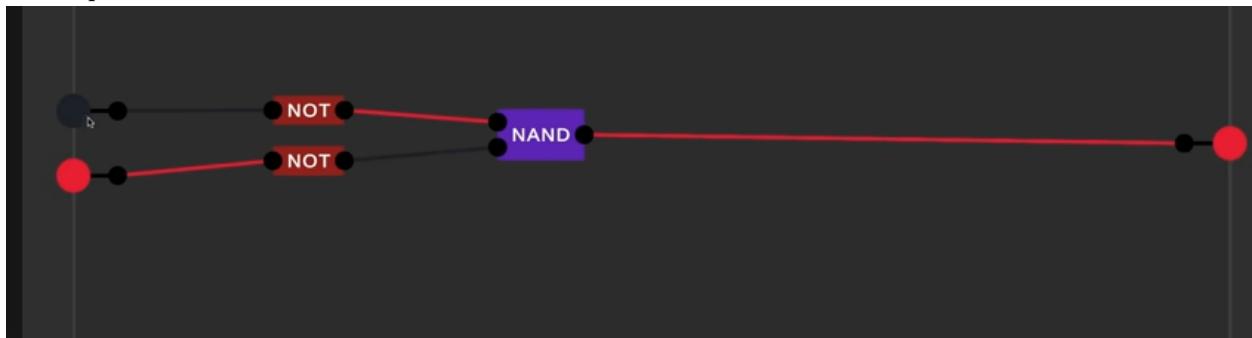
Reto 2

Usando las compuertas AND and NOT. Construir la compuerta OR que tiene el comportamiento de la tabla

A	B	Luz
0	0	0
0	1	1
1	0	1
1	1	1

Tabla 6. La luz solamente está apagada cuando las dos compuertas están cerradas. Esta es la tabla de verdad de la disyunción u OR en Inglés.

Respuesta:



2.2 Sumas de 2 bits

Haciendo uso del ábaco sumamamos $10001 + 10101$. El objetivo de esta actividad es representar números en binario y ver su relación con las compuertas lógicas.

Reto: Despues de lo que descubriste en el ábaco, completa la siguiente tabla con 1 o 0

A	B	Reciduo	Suma
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Tabla 7.

Respuesta

A	B	Reciduo	Suma
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Tabla 8.

Reto : A qué compuerta lígica se te parece el resultado del reciduo?

Respuesta: Es la compuerta AND

2.2.1 Construyendo un Sumador de 2 Bits

Puede verifica que esta es la tabla de verdad de la suma de 2 Bits

A	B	Lo que sobra	Suma
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Tabla 9. Tabla de verdad de una suma de 2 Bits. Cuando las dos entradas son 1, el resultado de la suma es 0, pero sobra 1.

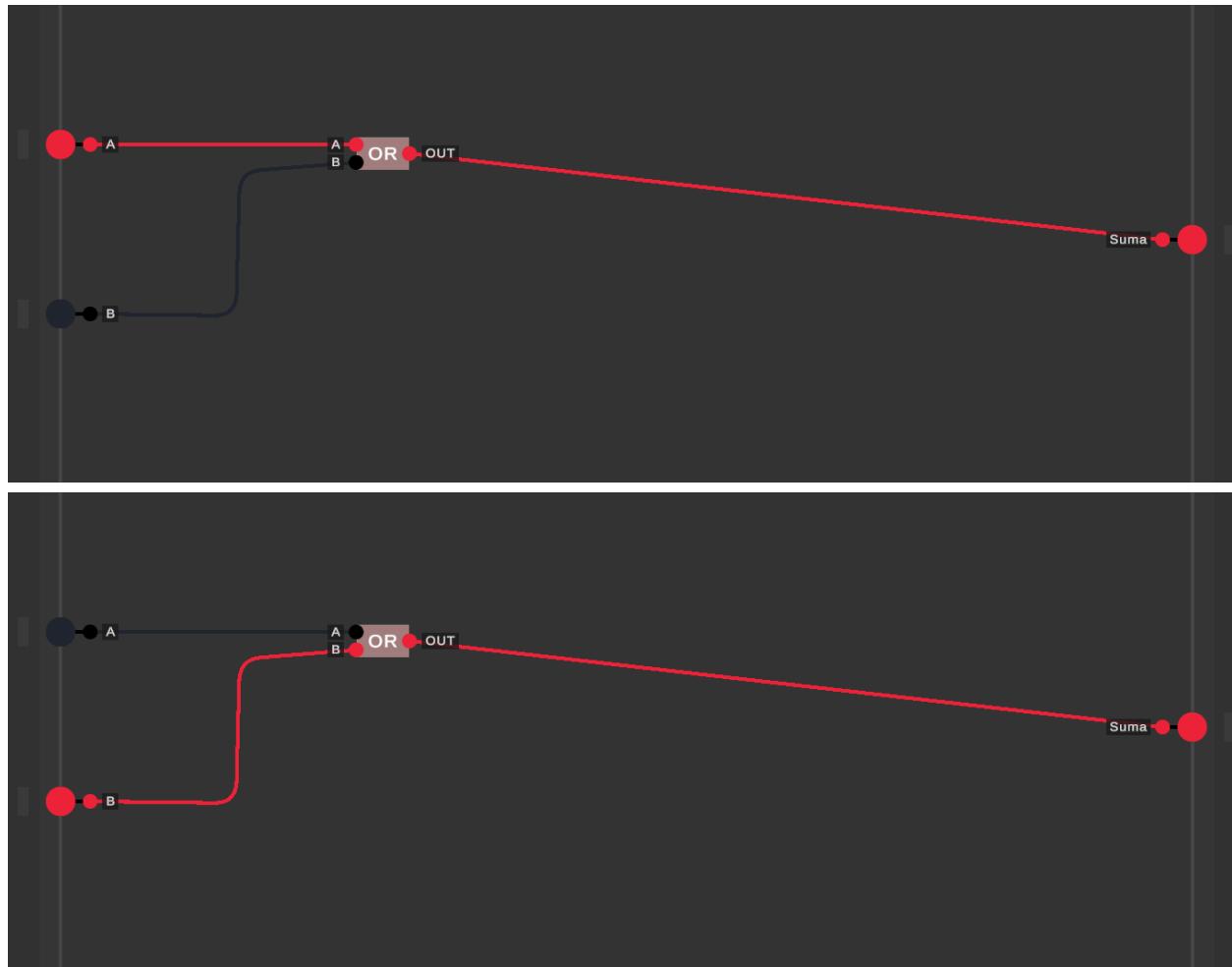
Reto: Observa la columna de lo que sobra. A que operacidor lógico se te parece?

Respuesta: El el operador AND.

Reto: Observa la columna **Suma** a que operador lógico se te parece?

Respuesta: Se parece al operador OR. Sólo que en vez de tener un 1 en la última fila, tiene un cero.

Teneindo en cuenta tu respuestas a los dos retos anteriores, tomemos la compuerta OR como punto de partida

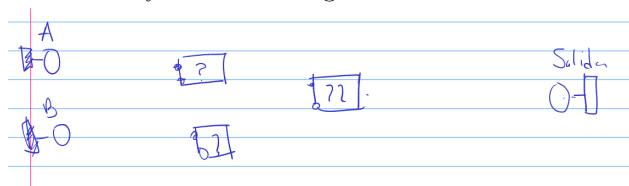


La suma es 1 cuando alguno de los as dos entradas es 1. Pero queremos replicar este comportamiento:

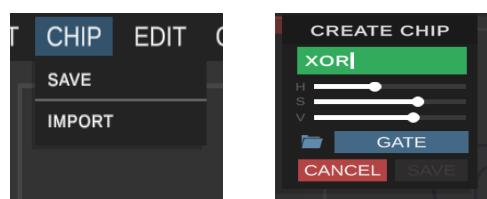
Suma
0
1
1
0

Cómo lo harías?

Reto: Ensayar varias configuraciones en el simulador para lograr el comportamiento de la suma.



De hecho este arreglo de compuertas tiene un nombre especial es llamado el OR exclusivo (XOR). Puedes usar el menú CHIP -> SAVE para resumir esta configuración en un solo chip integrado.



Un paso al frente: Llevado en cuenta lo sobra

Volvamos a la suma, supongamos que queremos sumar dos números binarios. Por ejemplo:

$$\begin{array}{r} 100011 \\ + 000111 \\ \hline \end{array}$$

Sumamos las columnas como lo hacemos desde primero de escuela, solo que en binario. Entonces , la primera columna nos da $1 + 1 = 10$ pongo 0 y llevo el 1:

$$\begin{array}{r}
 & | & \rightarrow \text{Lo que sobra} \\
 100011 & \\
 + 000111 & \\
 \hline
 0 & \rightarrow \text{Suma.}
 \end{array}$$

En la siguiente columna tenemos $1 + 1 + 1 = 11$, pongo el 1 y llevo 1 y así sucesivamente con las demás columnas

$$\begin{array}{r}
 & | & | & | & \rightarrow \text{Lo que sobra} \\
 100011 & \\
 + 000111 & \\
 \hline
 101010 & \rightarrow \text{Suma.}
 \end{array}$$

¡Excelente! ¡Hemos descubierto cómo construir un chip sumador que cumple con los requisitos de la columna ‘Suma’ en la tabla! Ahora el desafío consiste en diseñar un chip que tenga una salida adicional para representar ‘lo que sobra’ y que esta nueva salida se comporte de acuerdo a la columna ‘Lo que sobra’. ¡Manos a la obra!

Lo que sobra	Suma
0	0
0	1
0	1
1	0

Reto: Observa el siguiente chip incompleto



Aquí tienes un emocionante desafío! Si decides aceptarlo, tu misión es utilizar el chip que acabas de crear, junto con otros que ya conoces, para lograr que la salida correspondiente a “lo que sobra” se comporte de acuerdo a la tabla que descubriste para la suma de 2 bits. ¡Demuestra tu ingenio y habilidades para resolver este desafío! ¡Adelante!

A	B	Lo que sobra	Suma
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Tabla 10. Tabla de verdad de una suma de 2 Bits. Cuando las dos entradas son 1, el resultado de la suma es 0, pero sobra 1.

3 Computadores de tubos al vacío

La conexión entre los circuitos eléctricos y la lógica fue establecida por Claude Shannon en 1937, quien se basó en los trabajos de George Boole de 1864 específicamente en *An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities*. Shannon fue el primero en reconocer que las operaciones booleanas podían representarse mediante circuitos electrónicos.

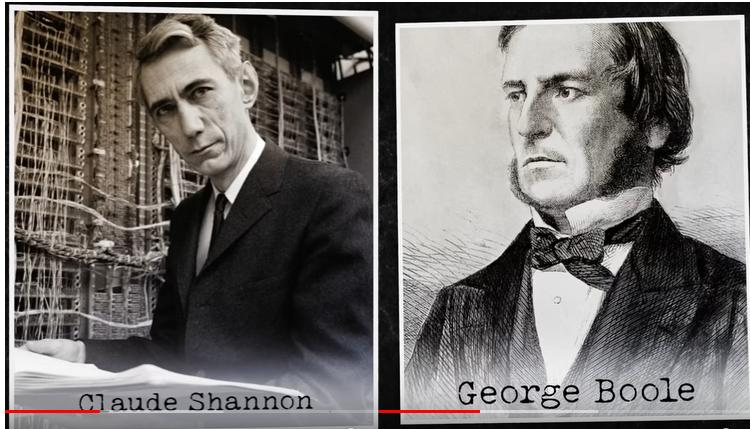


Figura 8.

Shannon observó lo que hemos explorado en las secciones anteriores: es posible reconstruir operaciones lógicas en circuitos mediante la activación y desactivación de interruptores eléctricos dispuestos de manera ingeniosa. Sin embargo, estos interruptores debían ser activados manualmente. Al principio, se intentó automatizar este proceso utilizando interruptores magnéticos, pero con el tiempo surgieron tecnologías más eficientes, como los tubos de vacío y, más recientemente, los transistores. Para introducir lo que hacen los transistores, la figura muestra la compuerta NOT que estudiamos en la clase anterior, donde el interruptor ha sido reemplazado por un pequeño transistor.

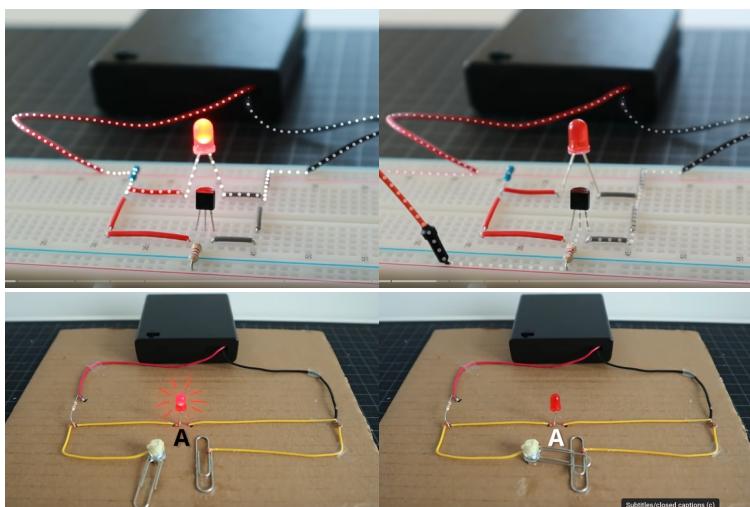


Figura 9.

En el panel derecho de la figura, una pequeña corriente hace que el transistor conduzca energía, impiendiéndolo así que la energía fluya a través del bombillo, lo que provoca que este se apague. La física del transistor permite que funcione como un circuito cerrado en presencia de la corriente y abierto en su ausencia. Para comprender mejor su origen y funcionamiento, es útil estudiar cómo evolucionaron a partir de los tubos de vacío.

Los tubos de vacío eran, en esencia, bombillos. Thomas Alva Edison fue el primero en notar que sus bombillos siempre se oscurecían en un lado. Esto se debe a que, cuando el filamento dentro de un tubo de vacío se calienta al aplicar una corriente eléctrica, su temperatura aumenta significativamente. Este filamento, generalmente hecho de tungsteno, alcanza temperaturas tan altas que algunos de sus electrones adquieren suficiente energía para liberarse de los átomos que los mantienen unidos. Este proceso, conocido como **emisión termoiónica**, ocurre porque la agitación térmica proporciona a los electrones la energía necesaria para superar la barrera de potencial que normalmente los mantiene ligados al núcleo del átomo.

Como las bombillas de Edison funcionaban con corriente continua, se generaba un campo eléctrico en el espacio entre el cátodo (negativo) y el ánodo (positivo). Dado que los electrones son partículas cargadas negativamente, son atraídos hacia el ánodo cargado positivamente. Algunos de estos electrones impactaban contra el vidrio del tubo, causando su oscurecimiento gradual.

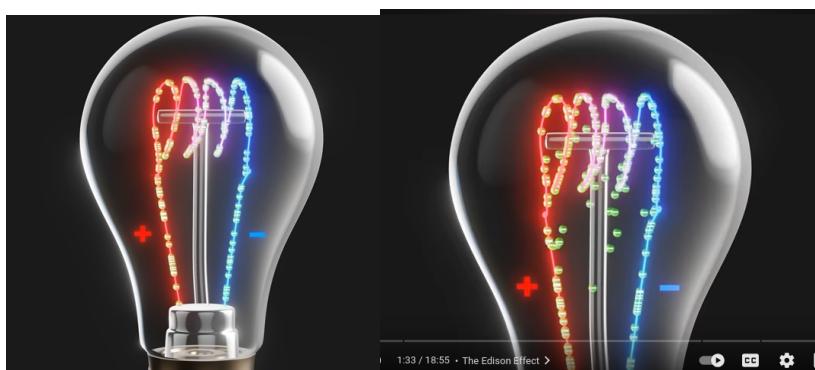


Figura 10. https://www.youtube.com/watch?v=FU_YFpfDqqA

Este flujo de electrones, que ocurre naturalmente en las bombillas de Edison solo en una dirección, puede invertirse si se agrega un pequeño filamento en el centro, como se muestra en la figura.

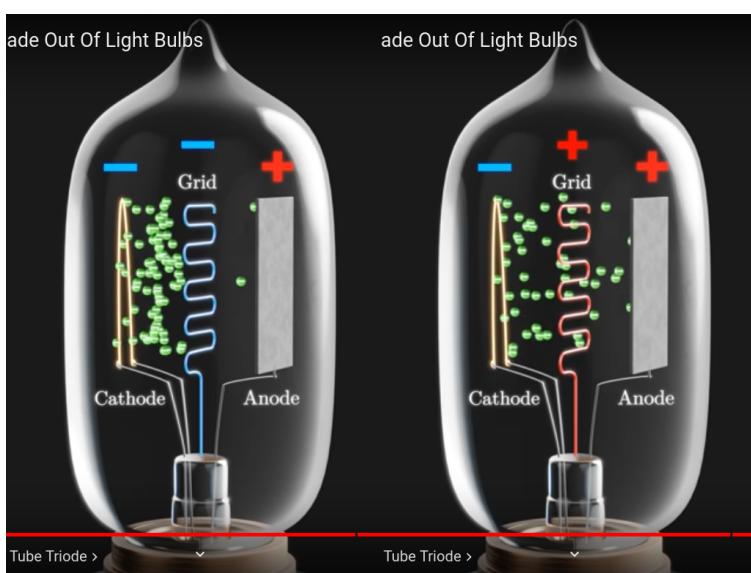


Figura 11.

Este invento, inspirado por el *efecto Edison*, se conoció como el triodo y fue el avance clave que dio origen a la electrónica moderna. La versión original fue patentada en 1904 por John Ambrose Fleming y perfeccionada en 1906 por Lee de Forest. En el triodo, un electrodo en forma de rejilla o espiral se coloca entre el cátodo y el ánodo, de ahí su nombre.

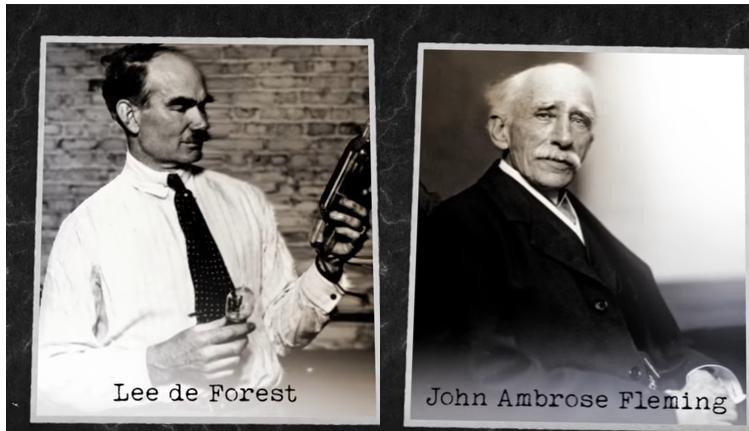


Figura 12.

Un potencial puede aplicarse entre el ánodo y el cátodo, pero el número de electrones que fluye entre ellos es controlado por el voltaje en la rejilla central. Si la rejilla se carga negativamente, repele a los electrones hacia el cátodo y cancela la corriente. Si, por el contrario, se carga positivamente, acelera a los electrones hacia el ánodo, permitiendo que un pequeño voltaje en la rejilla controle el flujo de corriente en el circuito, es decir, puede encender o apagar el circuito.

Este aspecto, el de poder controlar el flujo de electrones con un pequeño voltaje, implica que este pequeño voltaje puede estar asociado con las oscilaciones de una onda de radio o la señal de un teléfono. En estos casos, el triodo funciona como un amplificador de la señal. Este avance dio origen a las llamadas de larga distancia por teléfono, así como a radios y televisores, todos basados en tubos de vacío.



Figura 13.

Lo más interesante del triodo es que también funciona como un interruptor. Recordemos las compuertas NOT en la figura (mencionar figura anterior). El trabajo del interruptor puede ser realizado por un triodo al que se le aplica una pequeña corriente para abrir o cerrar el circuito, como se muestra en la figura.

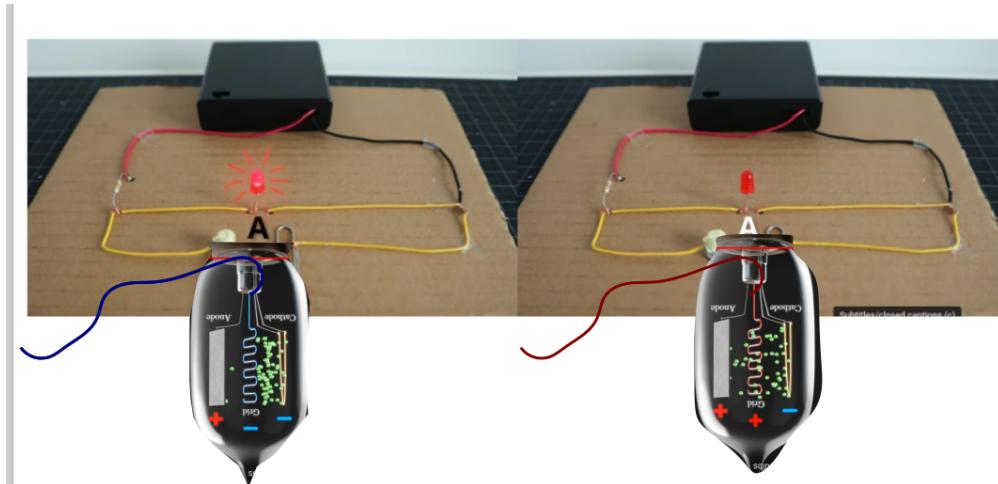
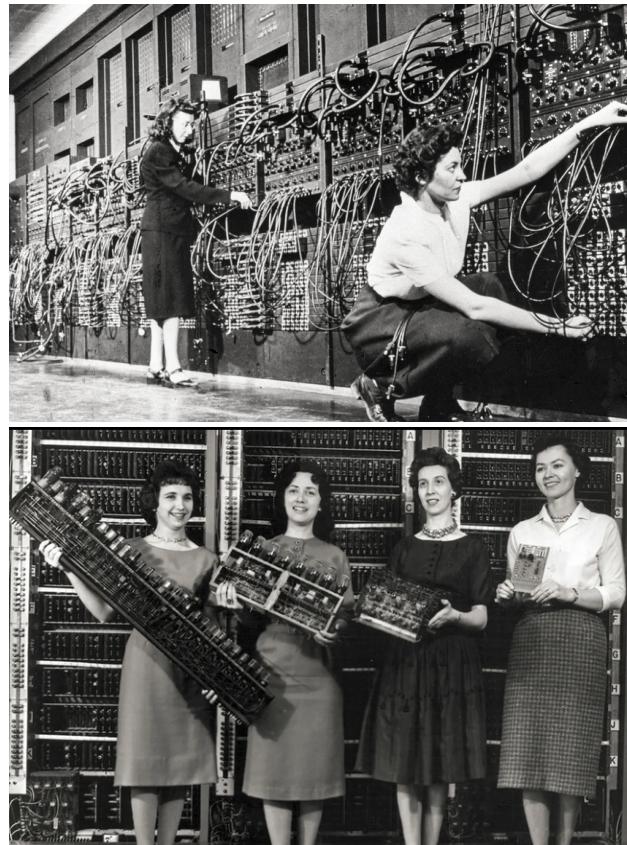


Figura 14.

En otras palabras, ¡el triodo es un interruptor electrónico! Esto abrió la posibilidad de implementar las *Laws of Thought on Which are Founded the Mathematical Theories of Logic*, es decir, de realizar operaciones matemáticas utilizando álgebra booleana y circuitos eléctricos. Así fue como se creó el primer computador electrónico programable, el ENIAC (*Electronic Numerical Integrator And Computer*), que comenzó a operar por primera vez en diciembre de 1945.



Patsy Simmers, holding ENIAC board; Gail Taylor, holding EDVAC board; Milly Beck, holding ORDVAC board; and Norma Stec, holding BRLESC-I board. | U.S. Army/ARL Technical Library Archives

Figura 15.

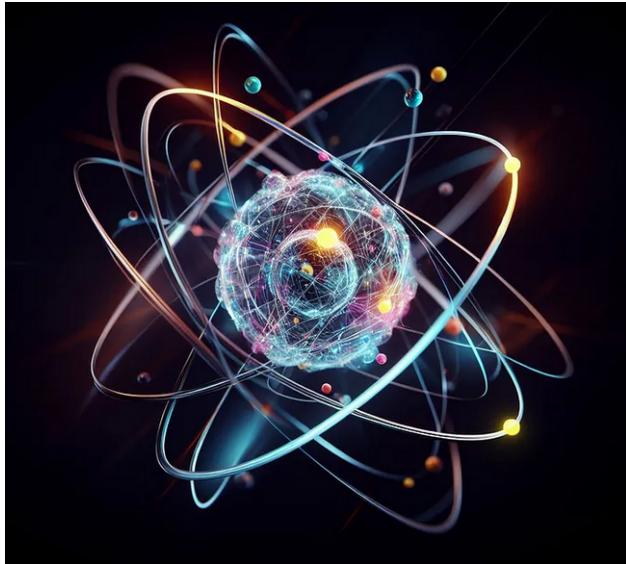
<https://www.digitaltrends.com/computing/remembering-eniac-and-the-women-who-programmed-it/>

3.1 El transistor

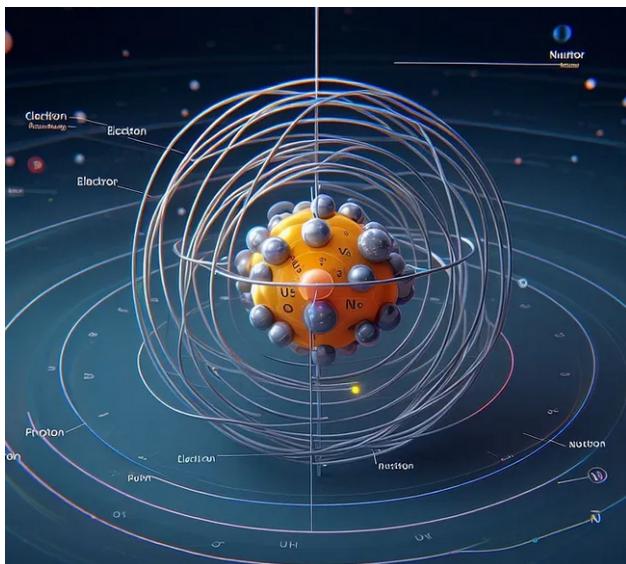
<https://www.youtube.com/watch?v=7ukDKVHnac4>

<https://www.youtube.com/watch?v=0wS9aTE2Go4&t=14s>

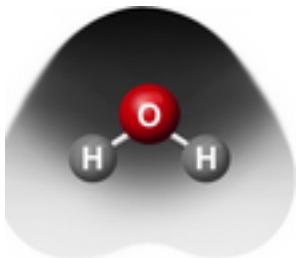
Es hora de profundizar un poco en la física que permite las herramientas que estamos estudiando:
Esto es un átomo



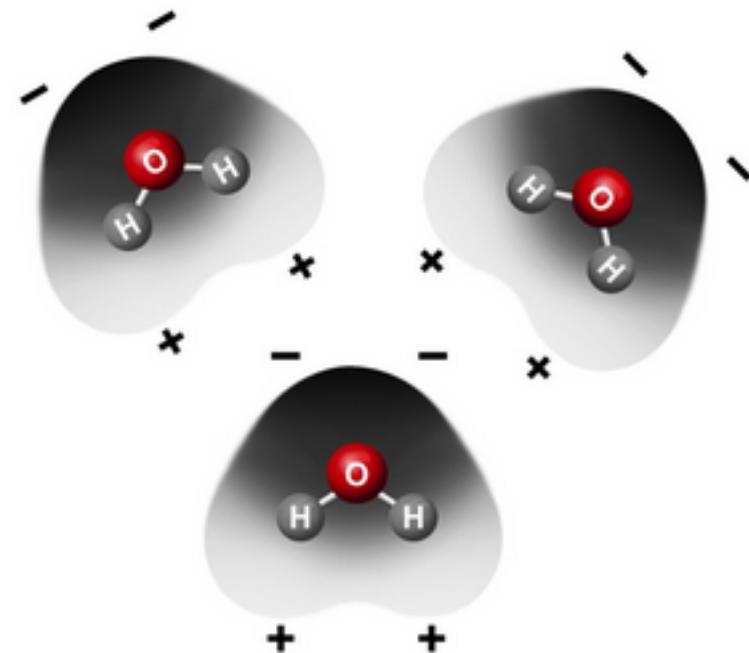
Este es el núcleo



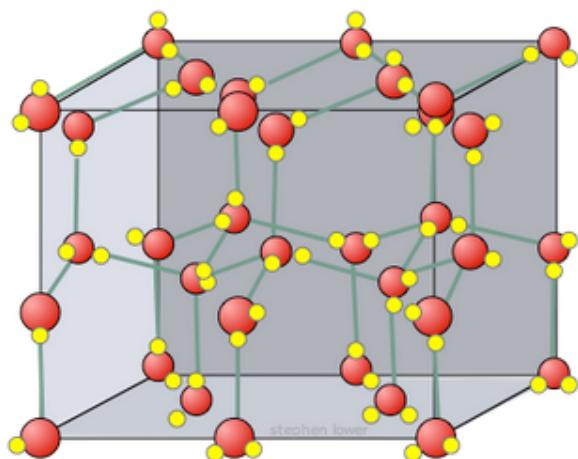
Esta es una molécula de agua



Y esto es agua líquida:



Este es un Pedazo de hielo

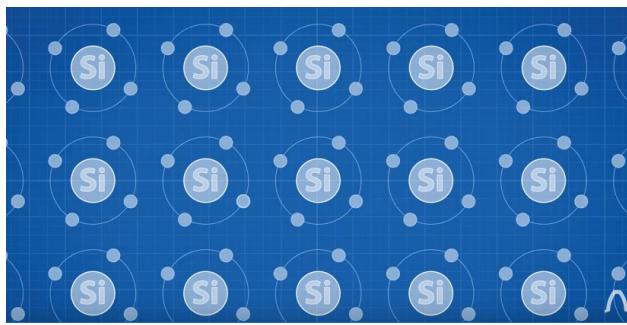


Este es un átomo de Silicio

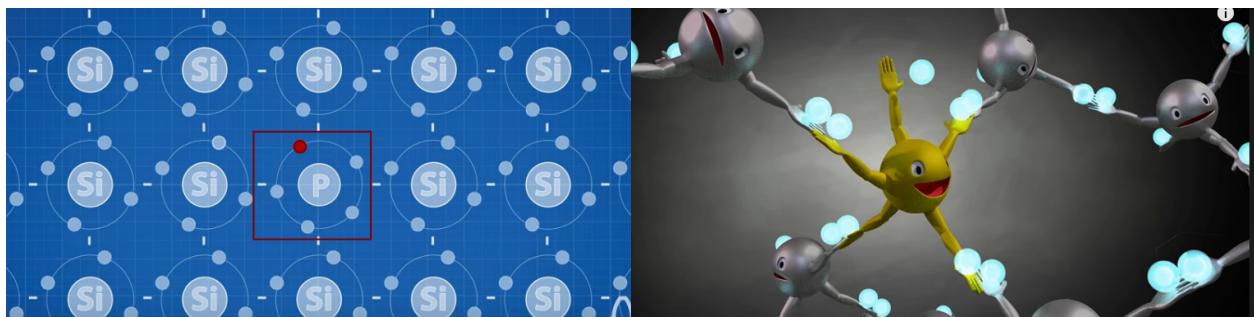


<https://www.youtube.com/watch?v=0wS9aTE2Go4&t=14s>

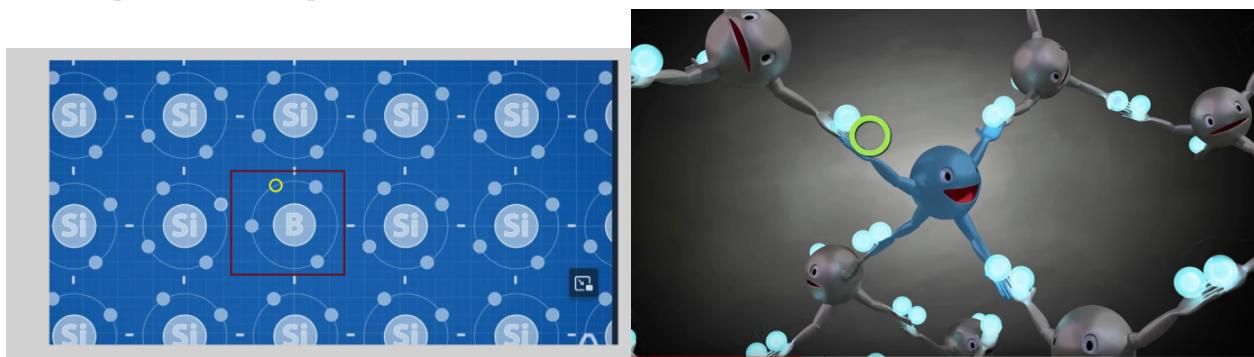
Esto es una red cristalina de silicio



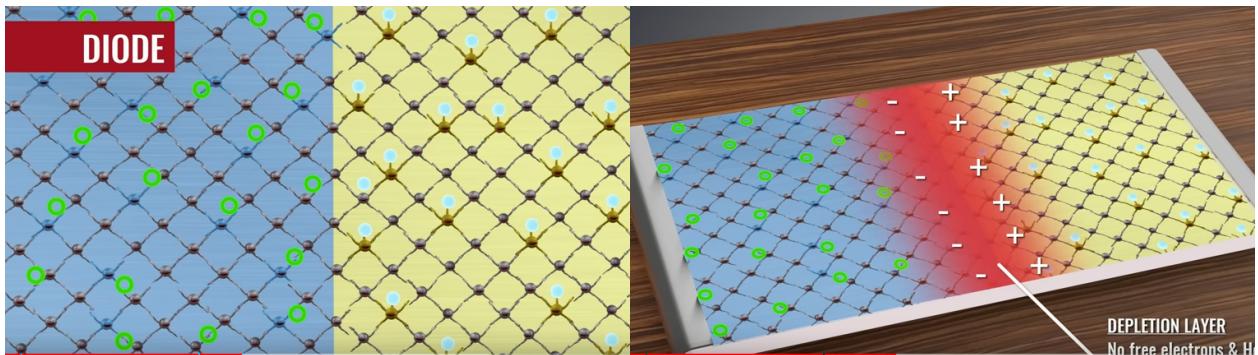
Silicio dopado con Fósforo, que tiene un electrón de valencia más:



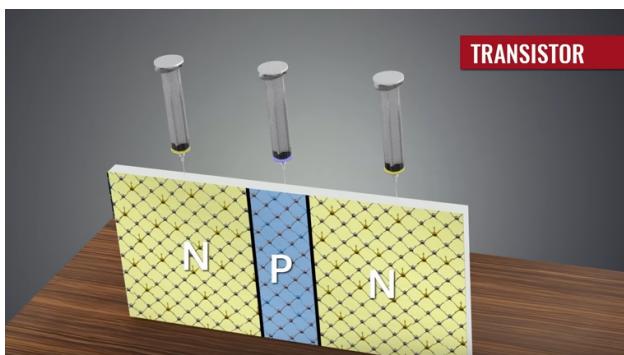
Silicio dopado con Boro, que tiene un electrón menos



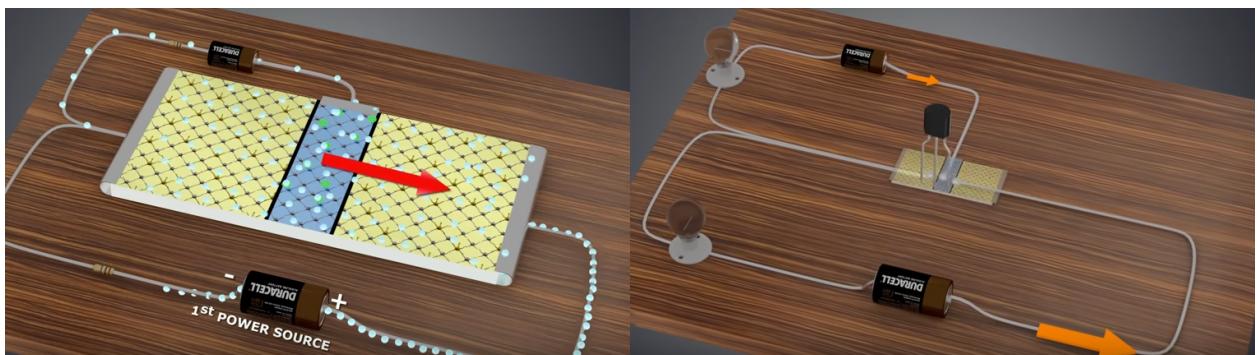
Diodo:



Transistor



Activado



Ahora te invito a que veas este interesante video sobre como los transistores leen código
<https://www.youtube.com/watch?v=HjneAhCy2N4>

3 Software

En la sesión anterior aprendimos que un transistor es básicamente un interruptor. En electrónica se representa por el símbolo

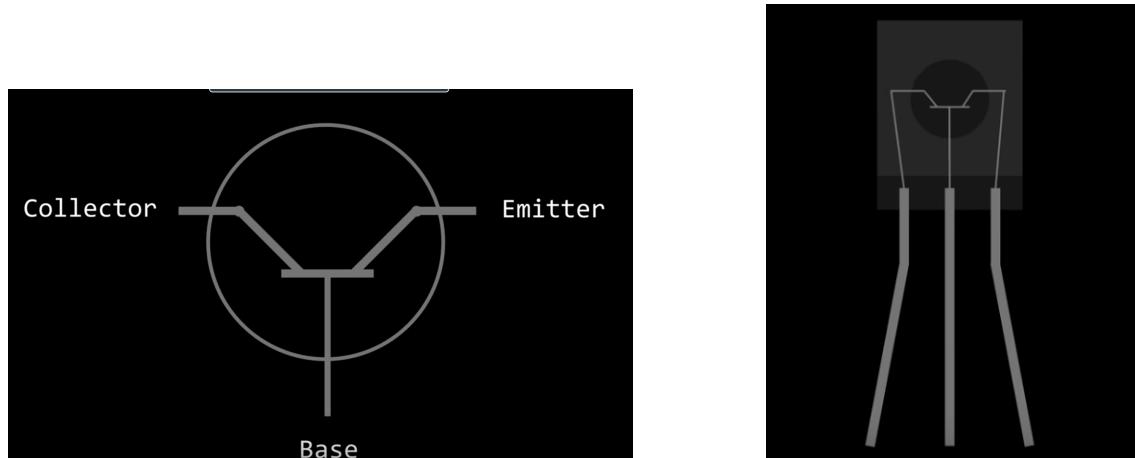


Figura 16.

Cuando no existe corriente en la base el transistor evita el paso de la corriente en el circuito, y una pequeña corriente en la base hace que el interruptor se abra:

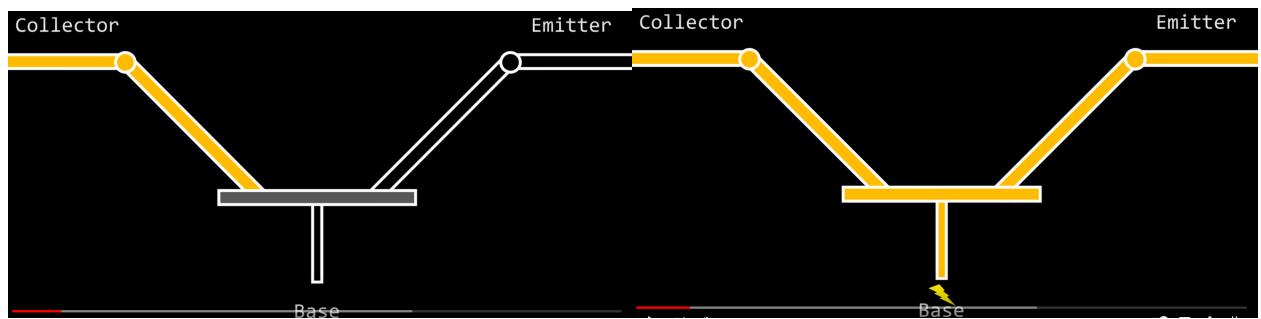
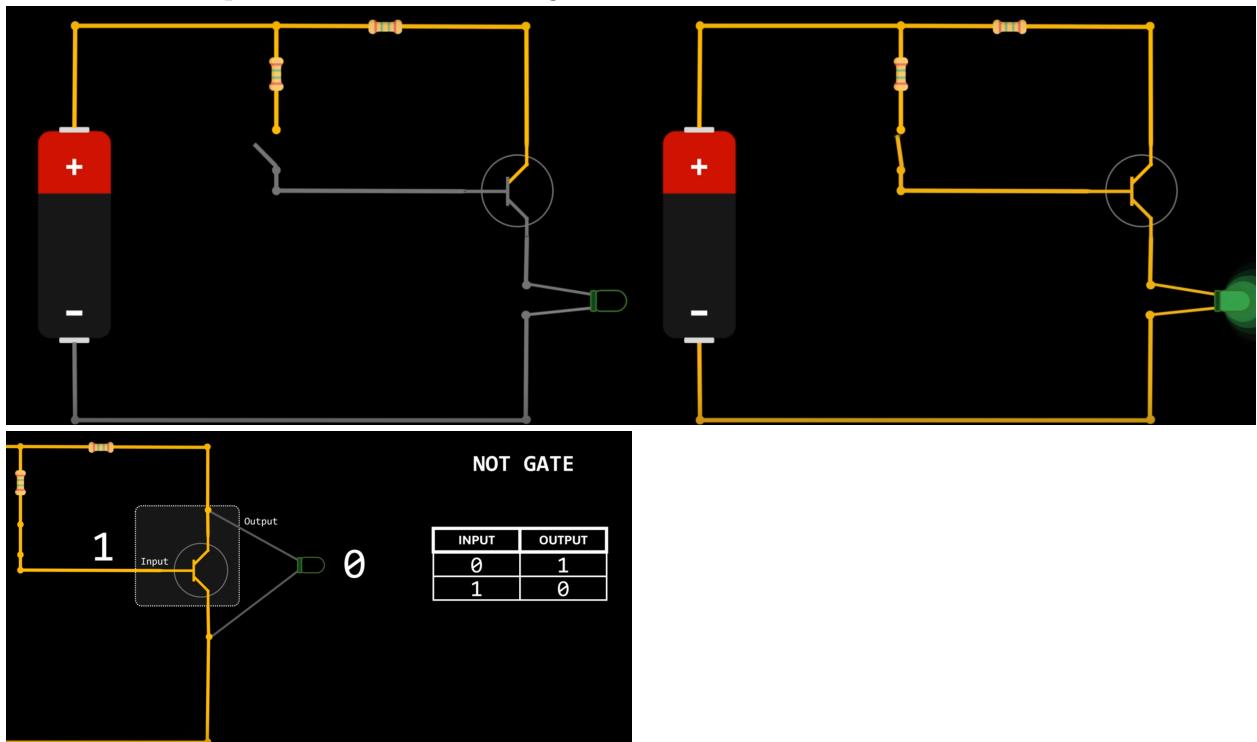


Figura 17.

En el caso de la copuerta NOT tenemos el diagrama



La compuesta AND consiste en dos transistores en paralelo:

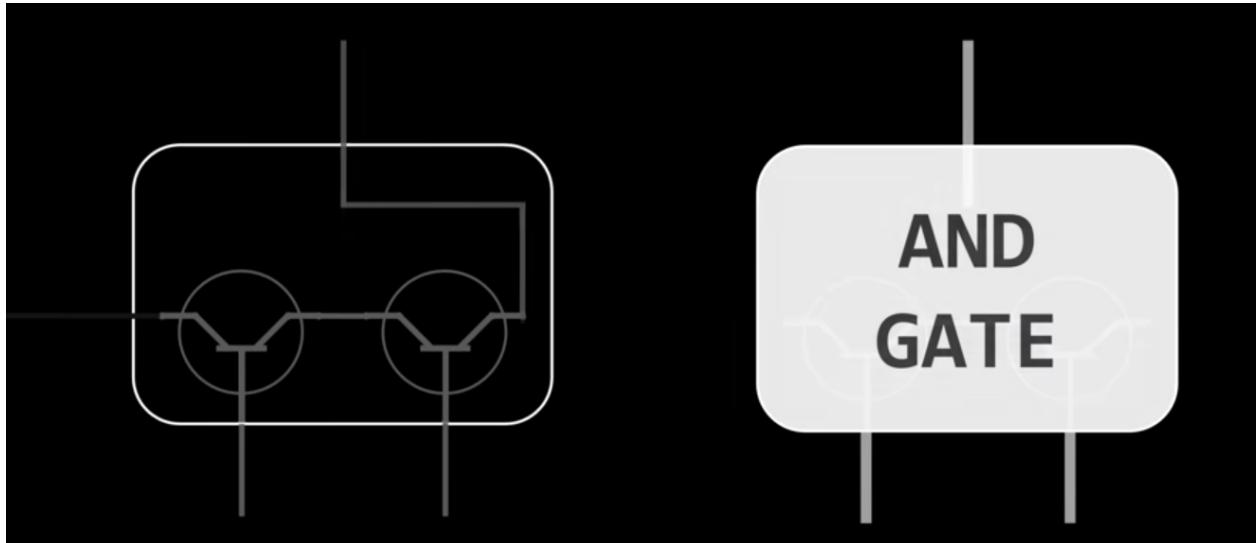
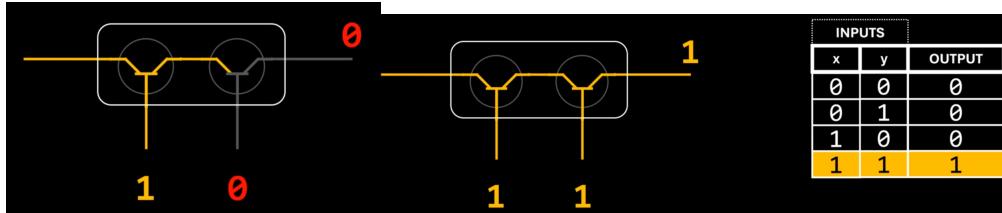


Figura 18.

Si conectamos los transistores en paralelo.

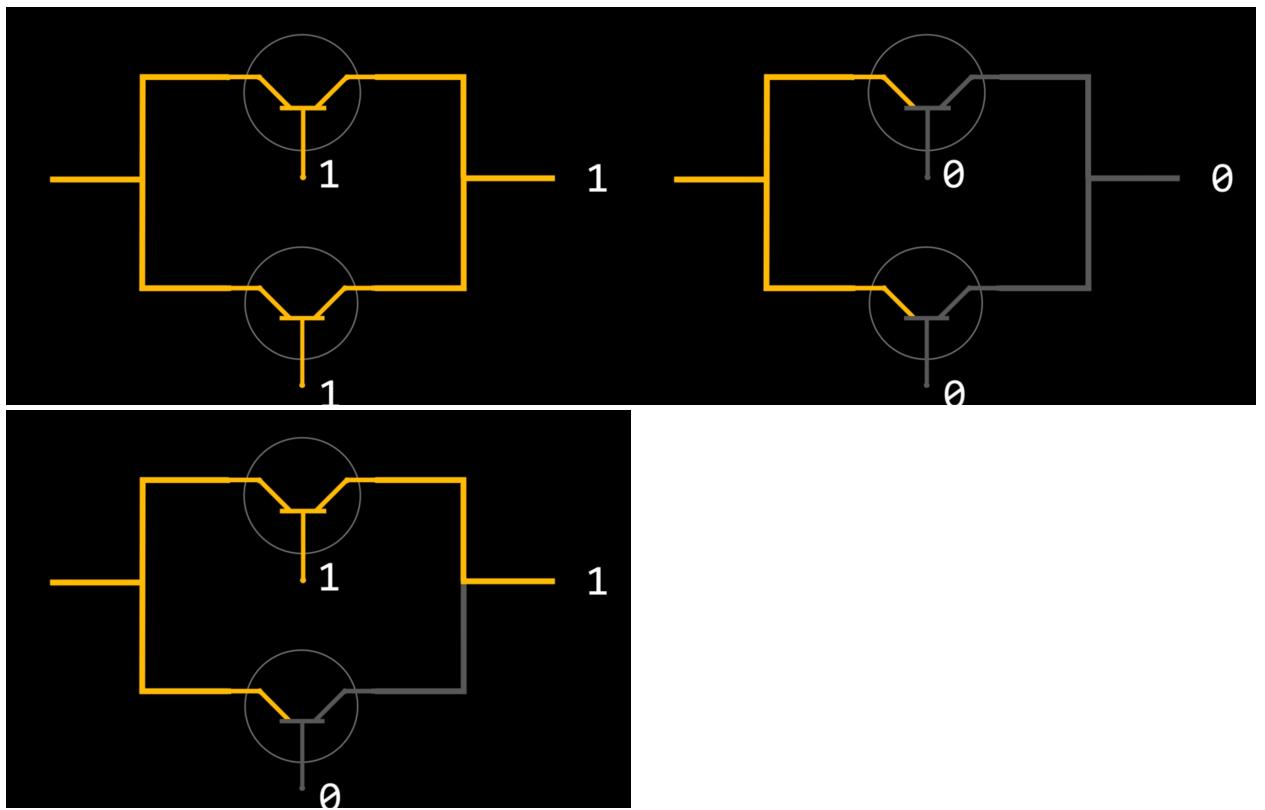


Figura 19.

Lo importante que podemos abstraer los circuitos a una caja. Por ejemplo:

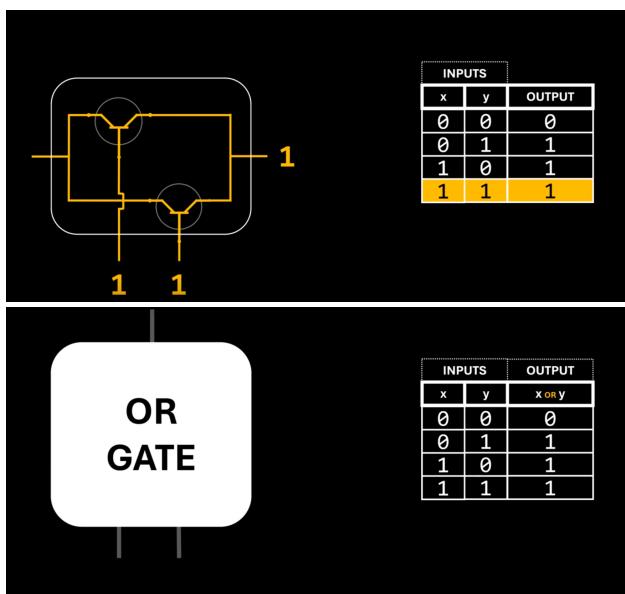


Figura 20.

Estas compuertas lógicas son tan fundamentales que tienen su propio símbolos

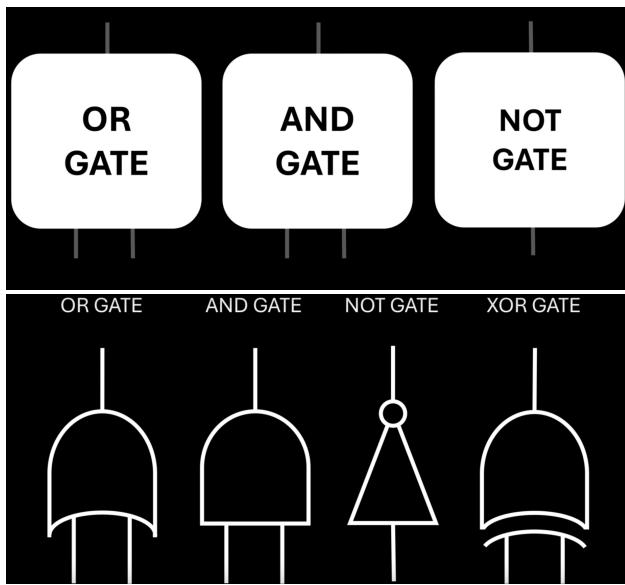


Figura 21.

Tambien vimos que usamos estas copuertas para construir comportamientos mas complejos. Por ejemplo un circuito que implemente la suma de dos bits, llamado un ADDER debe tener el comportamiento

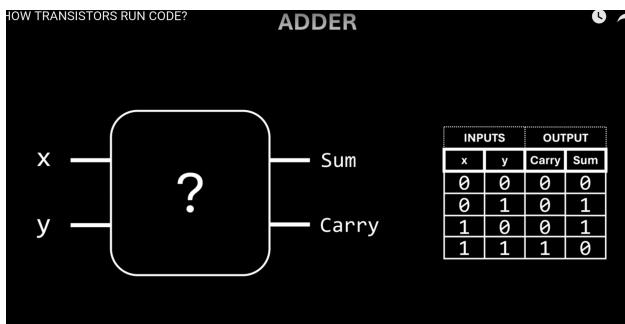


Figura 22.

Y vismo que este comportamiento es implementado por el circuito:

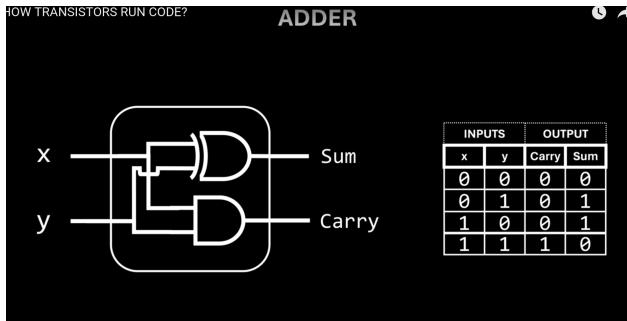


Figura 23.

Un sumadore de 3 bits es llamado un FULL ADDER y es implementado por el circuito

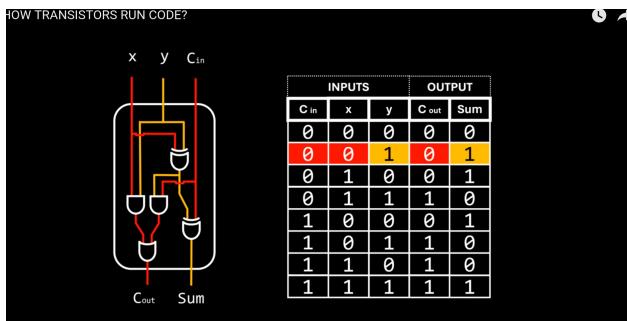


Figura 24.

La figura muestra el proceso de sumar 3 dígitos usando 3 FULL ADDERS

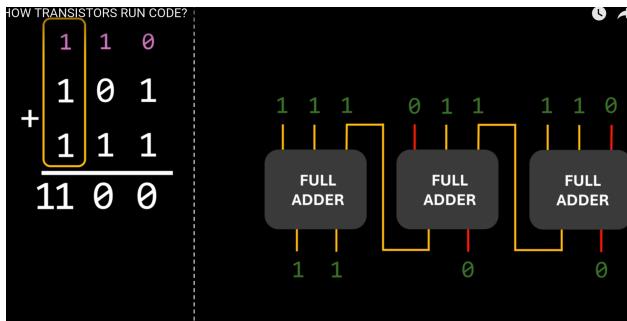


Figura 25.

Mientras que para 8 bits el gráfigra necesitaría 8 FULL ADDERS y así sucesivamente

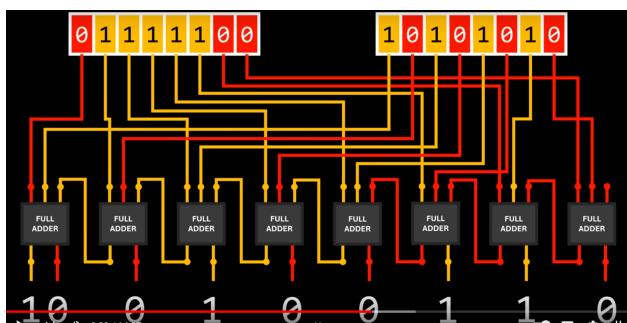


Figura 26.

Y nuevamente este circuito puede ser encapsulado en un solo componente conocido como el 8-BIT ADDER

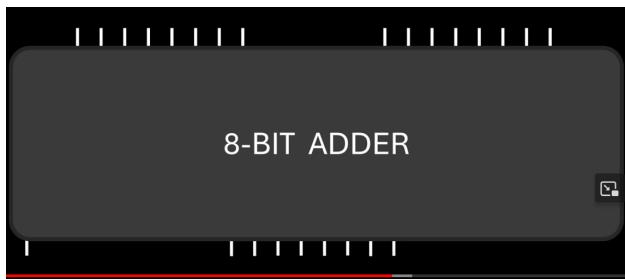


Figura 27.

A partir de estas mismas compuertas lógicas se pueden construir componentes con las instrucciones que entiende una CPU.

1 Lenguajes de programación

La figura muestra una unidad lógica de una CPU y el código en **lenguaje ensamblador** (o **assembly**). El lenguaje ensamblador es un lenguaje de bajo nivel que se utiliza para escribir instrucciones que son directamente ejecutables por la CPU. En este caso:

- LOAD [a] R0: Carga el valor almacenado en la dirección de memoria a en el registro R0.
- LOAD [b] R1: Carga el valor almacenado en la dirección de memoria b en el registro R1.
- ADD R0 R1: Suma los valores de los registros R0 y R1, y usualmente almacena el resultado en uno de los registros, en este caso parece ser R0.
- STORE R0 [a]: Guarda el valor del registro R0 en la dirección de memoria a.

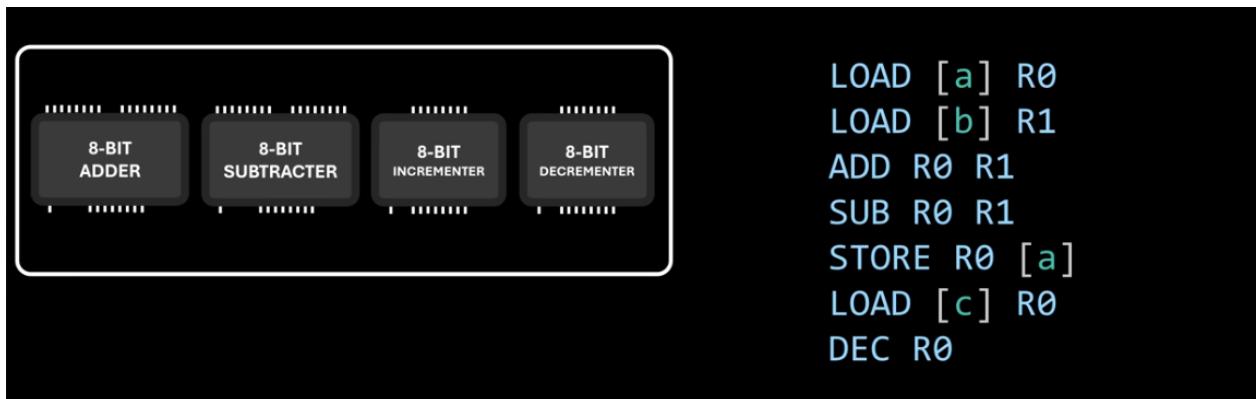


Figura 28.

```

LOAD [a] R0 ;Load value of memory address [a] in Register_0
LOAD [b] R1 ;Load value of memory address [b] in Register_1
ADD R0 R1 ;Add Register_0 with Register_1 and write the result in Register_0
SUB R0 R1 ;Subtract Register_0 with Register_1 and the write result in Register_0
STORE R0 [a] ;Store value in Register_0 in [a] memory address
LOAD [c] R0 ;Load value of [c] in Register_1
DEC R0 ;Decrement value in Register_0

```

Pero quién controla que parte del circuito ejecuta la instrucción?

1.1 Decodificadores Binarios

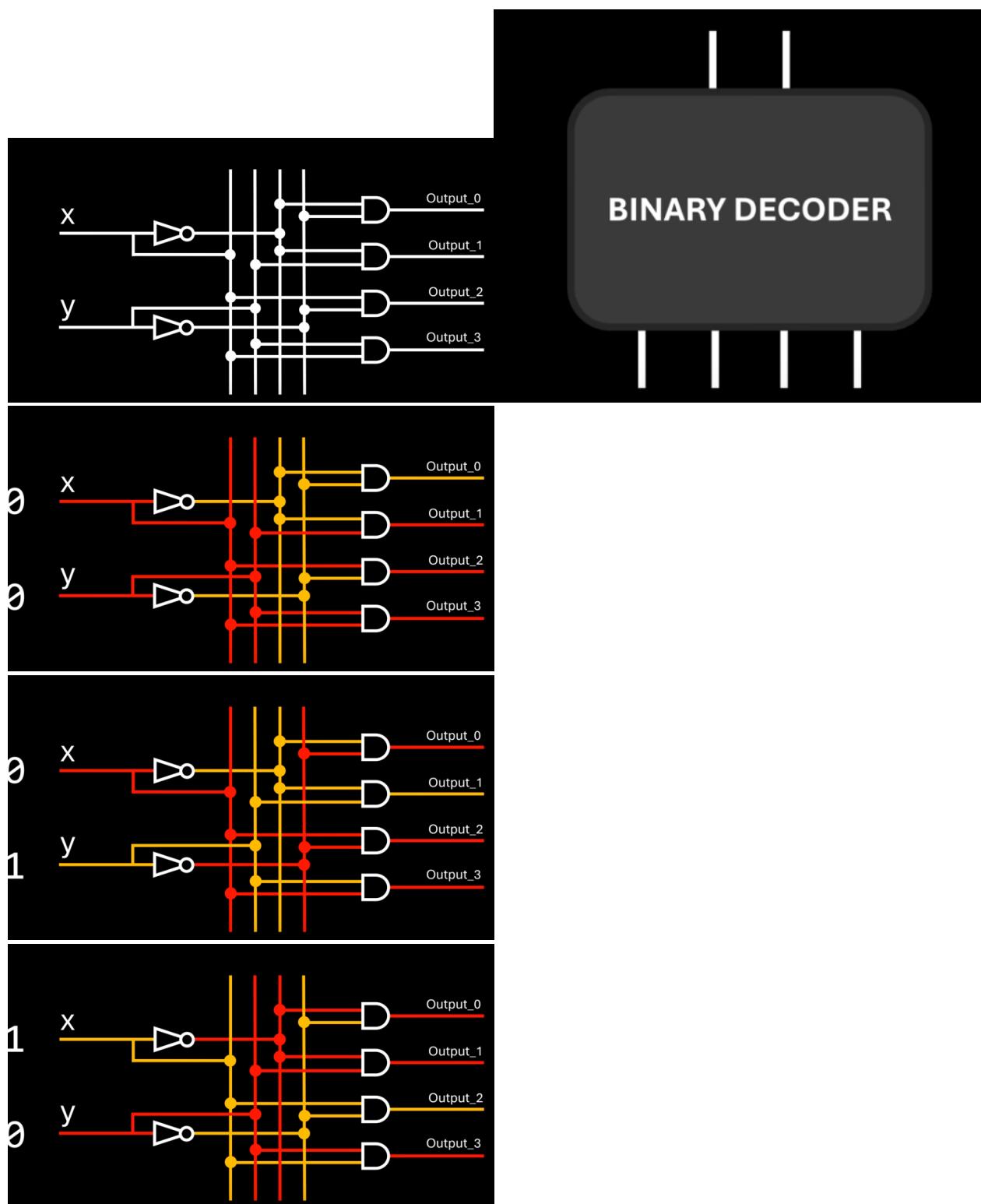


Figura 29.

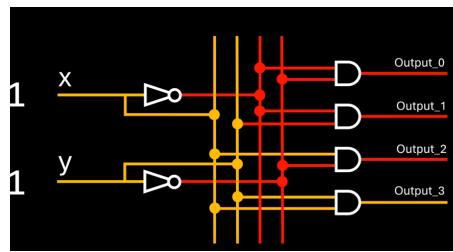


Figura 30.

Un **decodificador binario** (o **binary decoder**) es un circuito digital que convierte una entrada codificada en binario en una única salida activa. En otras palabras, toma un conjunto de bits de entrada y activa una de las muchas posibles salidas en función de esa entrada.

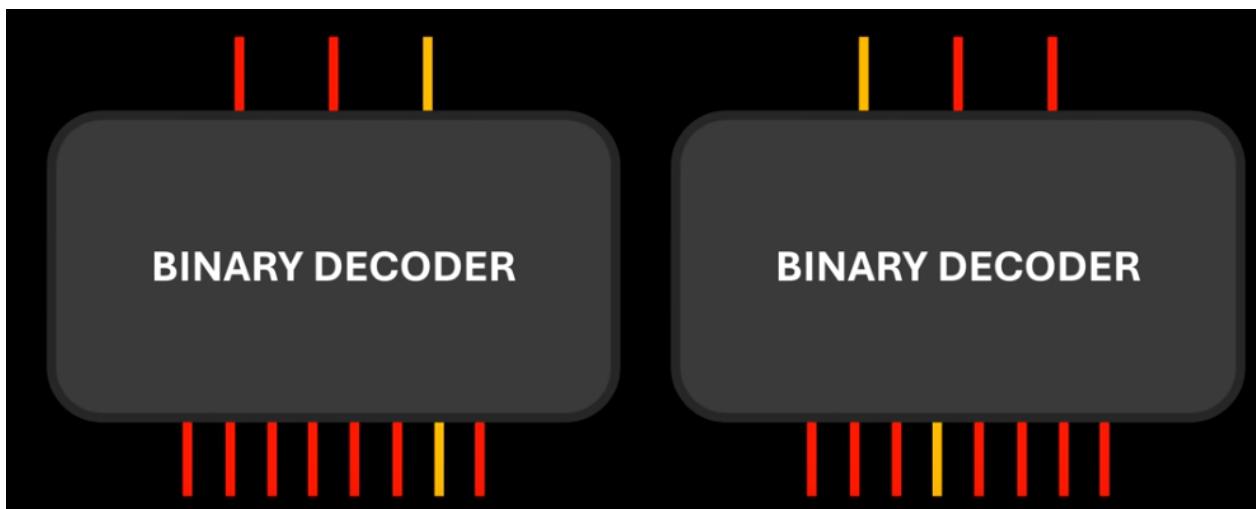


Figura 31.

En general con n inputs, podemos controlar 2^n outputs.

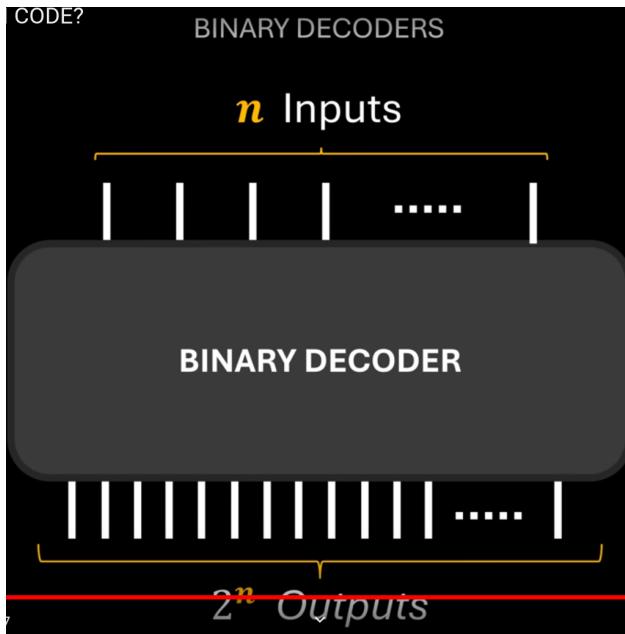


Figura 32.

Esta es la base del software, porque significa que podemos construir circuitos que nos permitan escoger entre múltiples opciones:

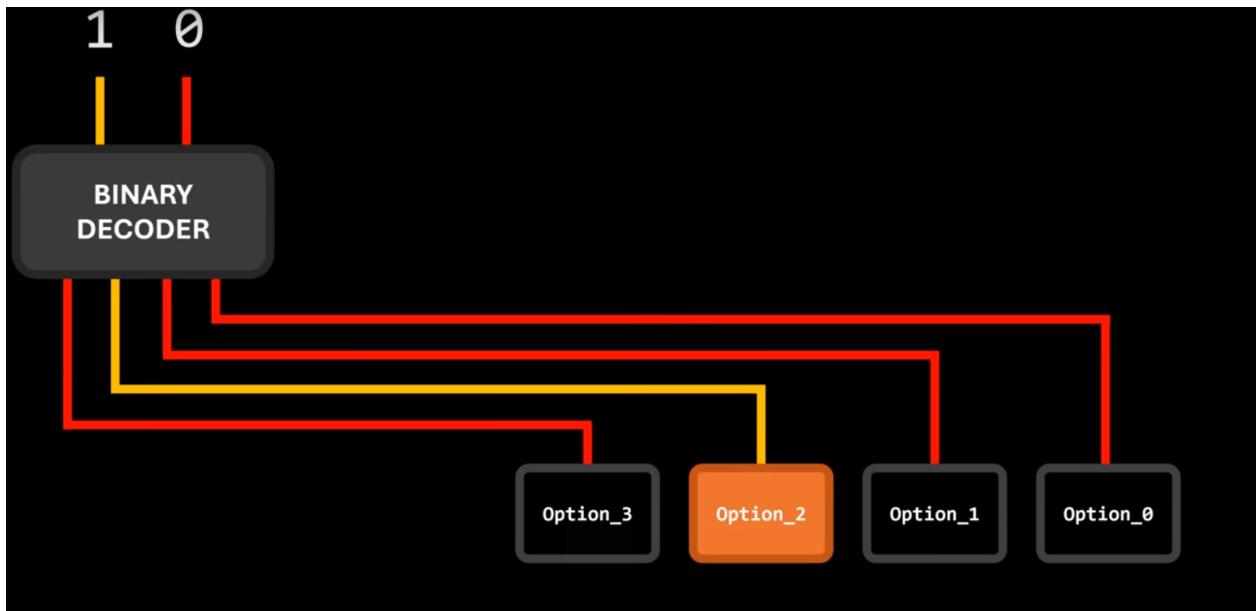
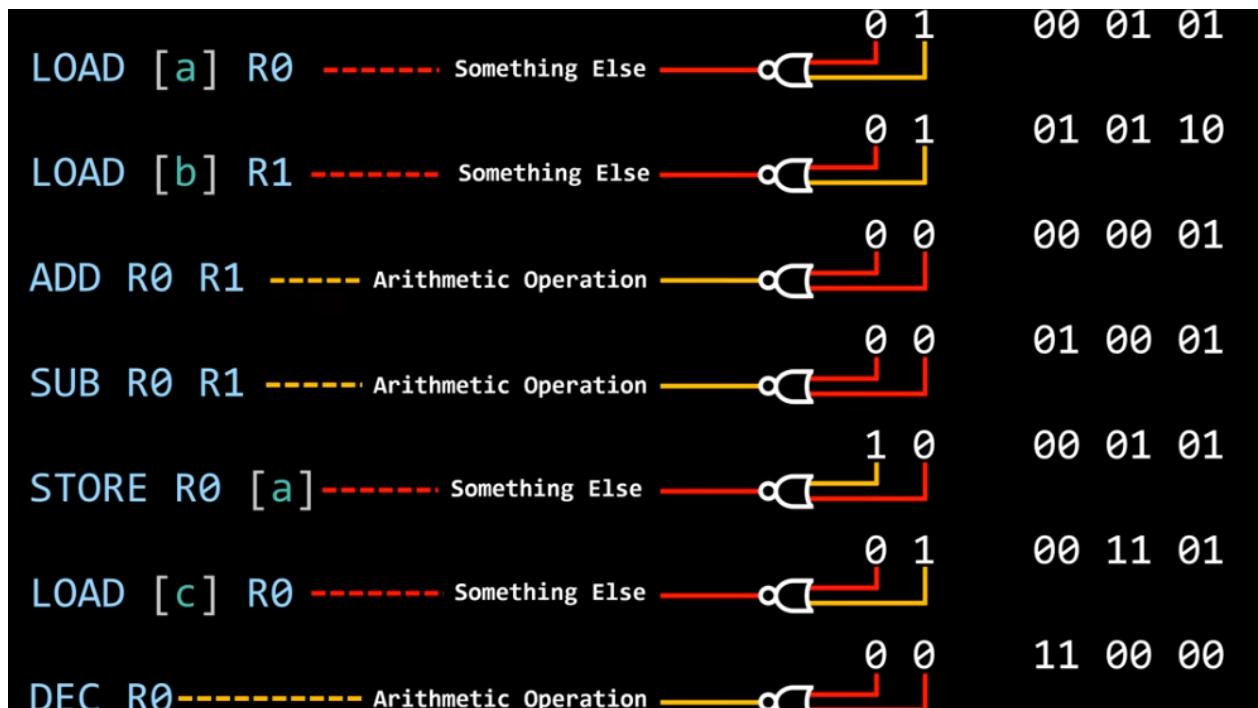


Figura 33.

De hecho el código assembly que acabamos de ver es simplemente una representación más legible de combinaciones de 1 y 0 para que el codificador binario:

Ejemplo:



Concentrémonos en operaciones aritméticas:

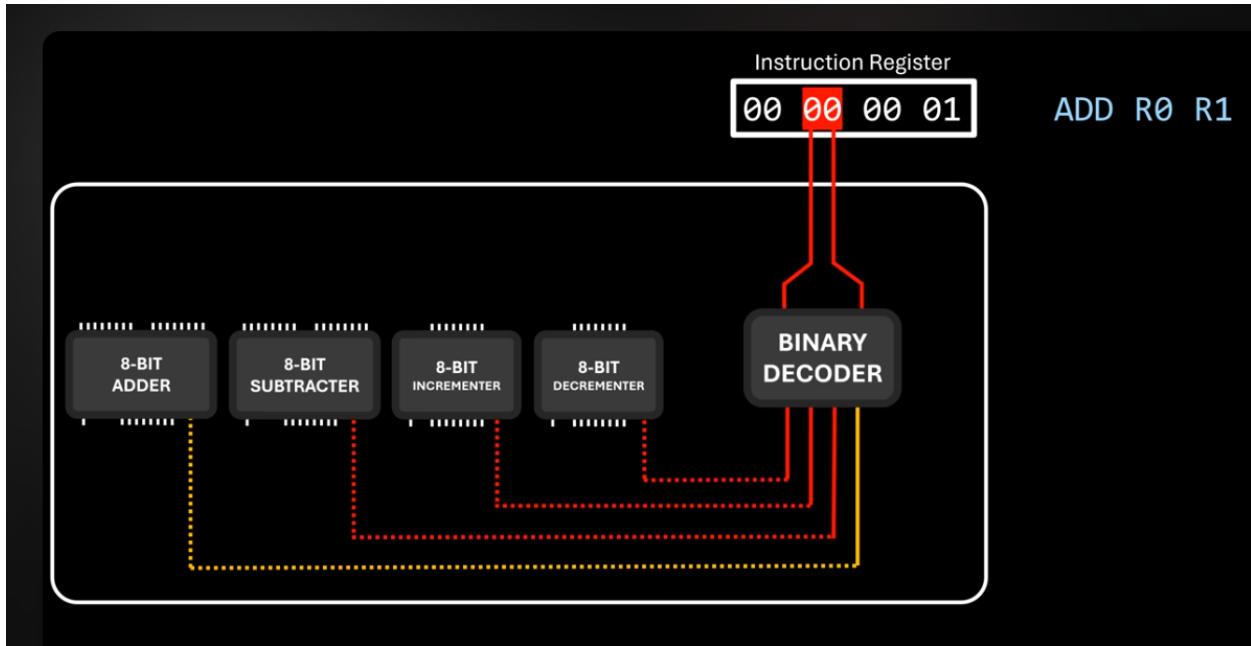
ADD R0 R1	00 00 00 01
SUB R0 R1	00 01 00 01
DEC R0	00 11 00 00

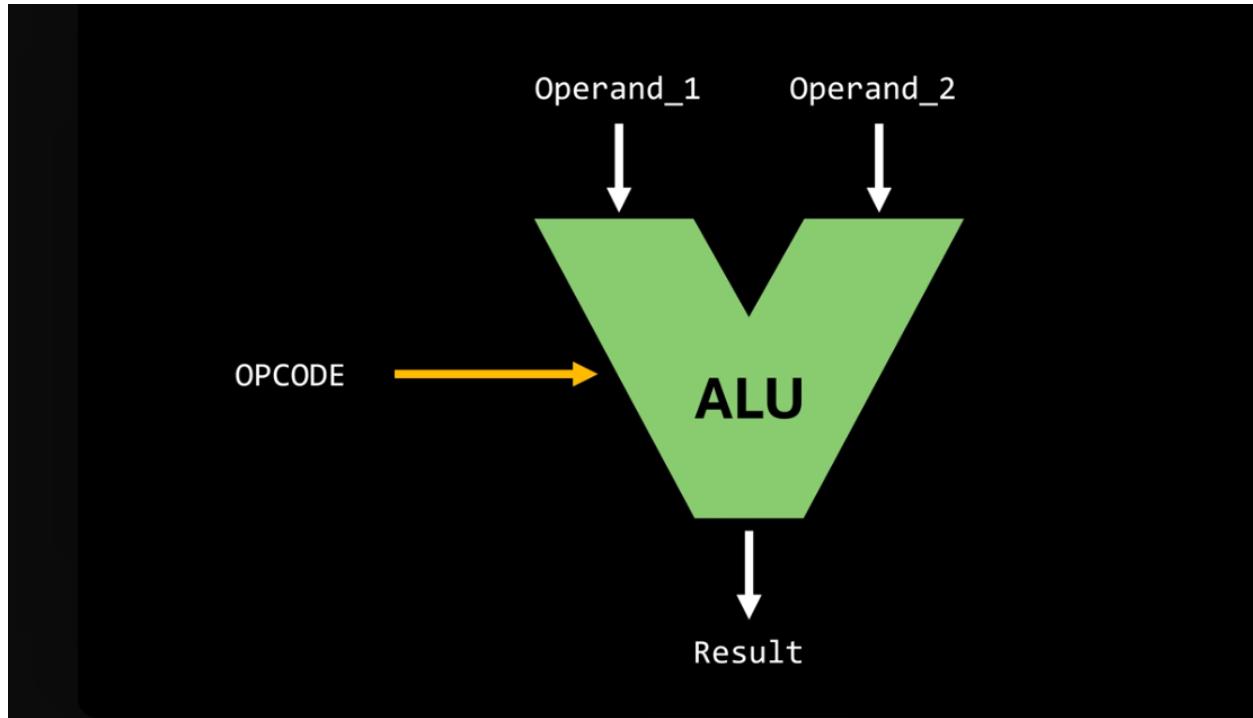
Figura 34.

ADD R0 R1	00	00	00 01
SUB R0 R1	00	01	00 01
DEC R0	00	11	00 00

ADD R0 R1	00	00	00	01
SUB R0 R1	00	01	00	01
DEC R0	00	11	00	00
ADD R0 R1	00	00	00	01
SUB R0 R1	00	01	00	01
DEC R0	00	11	00	00

OPCODE				
ADD R0 R1	00	00	00	01
SUB R0 R1	00	01	00	01
DEC R0	00	11	00	00





1.2 Breve historia del software

Los computadores que utilizaban tubos de vacío marcan el origen de los computadores digitales modernos, con la construcción del ENIAC en 1945. Simultáneamente, el matemático John von Neumann realizó importantes contribuciones que cambiaron la manera de entender cómo los computadores deben ser organizados y construidos. Basándose en los trabajos de Alan Turing, von Neumann mejoró los conceptos de memoria y direcciones en los computadores. Supervisó la construcción del sucesor del ENIAC, el EDVAC, completado en 1950, que se convirtió en el primer computador capaz de almacenar instrucciones y programas en su memoria.



Figura 35.

Este hito marcó el comienzo de la evolución de las ciencias de la computación como un campo de estudio independiente, dejando de ser solo una rama de las matemáticas, la física o la ingeniería pues pasamos de computadores mecánicos, como el ábaco, a computadores digitales basados en tubos de vacío, que podían realizar cálculos en microsegundos y almacenar instrucciones y programas. En 1949, los Laboratorios Bell lograron un avance significativo con la construcción del primer transistor de silicio. Posteriormente, en 1954, se construyó el primer computador basado en transistores, conocido como el **TX-0**. Este cambio de los tubos de vacío a los transistores permitió a los computadores volverse más rápidos, eficientes y pequeños.

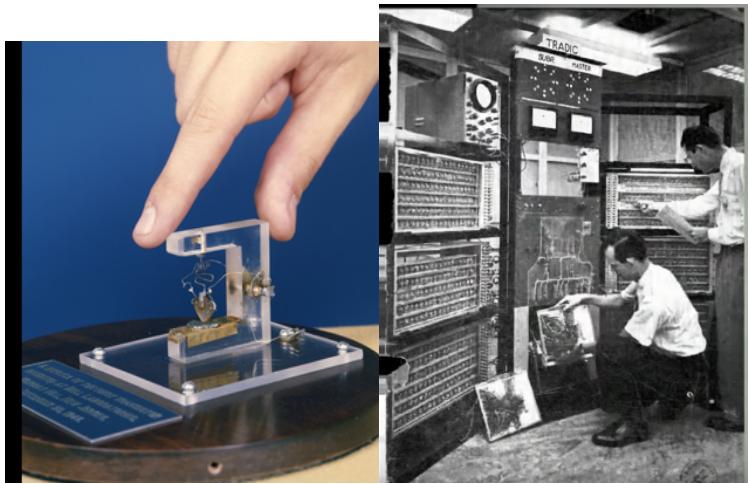


Figura 36.

Es en esta época cuando comenzaron a aparecer los primeros tratados sobre programación de computadores. Entre ellos destaca el trabajo de Donald Knuth, autor de la serie de libros *The Art of Computer Programming*, que formó a las primeras generaciones de programadores modernos. Puedes escuchar su historia aquí.

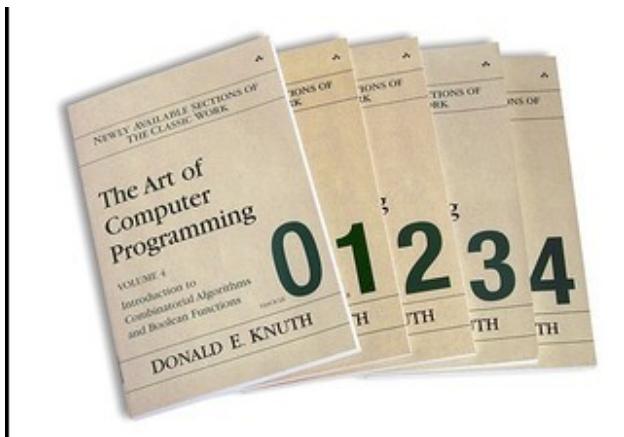


Figura 37.

El lenguaje ensamblador (Assembly) fue el primer lenguaje de programación introducido en 1949, permitiendo a los programadores comunicarse directamente con el hardware de los computadores a través de un código más amigable con el ser humano.

```

; Example of IBM PC assembly language
; Accepts a number in register AX;
; subtracts 32 if it is in the range 97-122;
; otherwise leaves it unchanged.

SUB32 PROC      ; procedure begins here
    CMP AX,97   ; compare AX to 97
    JL DONE     ; if less, jump to DONE
    CMP AX,122   ; compare AX to 122
    JG DONE     ; if greater, jump to DONE
    SUB AX,32   ; subtract 32 from AX
DONE: RET        ; return to main program
SUB32 ENDP      ; procedure ends here

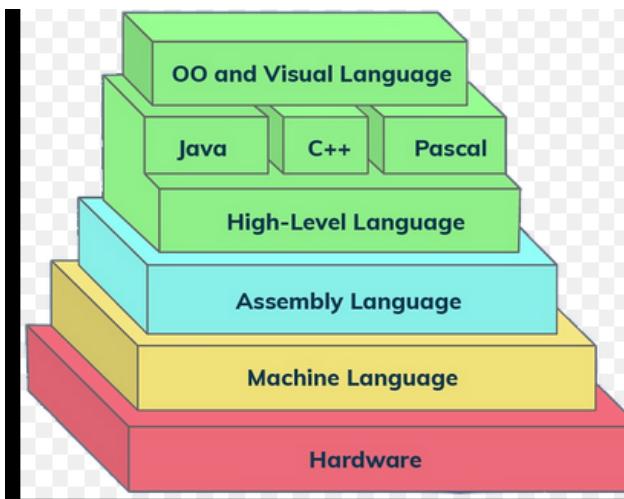
```

Figura 38.

- * **Assembly (1949):** Código cercano al hardware, eficiente para sistemas embebidos.
- * **FORTRAN (1957):** Cálculos científicos y de ingeniería.
- * **LISP (1958):** Pionero en inteligencia artificial y procesamiento simbólico.
- * **COBOL (1959):** Aplicaciones de negocios y sistemas de gestión.
- * **BASIC (1964):** Lenguaje accesible para principiantes.
- * **Pascal (1970):** Educación y desarrollo estructurado.
- * **C (1972):** Desarrollo de sistemas y aplicaciones de alto rendimiento.
- * **Prolog (1972):** Resolución de problemas lógicos y IA.
- * **Ada (1980):** Aplicaciones de misión crítica.
- * **C++ (1983):** Programación orientada a objetos y sistemas complejos.
- * **Perl (1987):** Procesamiento de texto y administración de sistemas.
- * **Python (1991):** Ciencia de datos, IA, desarrollo web.
- * **Java (1995):** Aplicaciones empresariales y móviles.
- * **JavaScript (1995):** Desarrollo web frontend.
- * **C# (2000):** Aplicaciones de escritorio, videojuegos, desarrollo empresarial.

2006 – Rust

Rust fue creado por Mozilla como un lenguaje de sistemas seguro y eficiente, diseñado para prevenir errores de memoria comunes en lenguajes como C y C++, a través de un sistema de gestión de memoria sin recolección de basura.



1.3 Git

Git es un sistema de control de versiones distribuido que permite gestionar el código fuente y colaborar en proyectos de manera eficiente. A través de comandos, Git rastrea los cambios, almacena versiones anteriores y facilita la colaboración en equipo.

1.3.1 Instalación de Git

En Linux:

Para instalar Git en distribuciones basadas en Debian/Ubuntu, ejecuta:

```
sudo apt update
sudo apt install git
```

En Windows:

1. Descarga [Git para Windows](#).
2. Ejecuta el instalador y sigue las instrucciones.
3. Abre “Git Bash” para usar Git en una terminal similar a Linux.

Configurar Git

Una vez instalado, debes configurar tu nombre de usuario y correo electrónico globalmente (estos aparecerán en cada commit).

```
git config --global user.name "Tu Nombre"
git config --global user.email "tuemail@ejemplo.com"
```

Inicializar un repositorio

Un repositorio es donde Git almacena tus archivos y sus versiones.

Crear un nuevo repositorio:

Para inicializar un repositorio de Git en un directorio existente, usa:

Inicializar un repositorio

Un repositorio es donde Git almacena tus archivos y sus versiones.

Crear un nuevo repositorio:

Para inicializar un repositorio de Git en un directorio existente, usa:

```
git init
```

Esto creará un subdirectorio `.git` en la carpeta actual, donde Git mantendrá el historial.

Preparar archivos: `git add` y la área de “staging”

Git usa una “área de staging” para preparar archivos antes de comprometerlos al historial del repositorio.

Agregar archivos al área de staging:

```
git add archivo.txt
```

O, para agregar todos los archivos modificados:

```
git add .
```

El comando `git add` toma una instantánea de los archivos y los mueve a la zona de “staging”. Estos archivos están listos para ser confirmados (`commit`), pero no han sido guardados permanentemente en el historial.

Guardar cambios: `git commit`

Una vez que los archivos están en la “staging”, debes confirmarlos con `git commit`. Esto guarda una versión permanente de los cambios en el historial de Git.

```
git commit -m "Mensaje que describe los cambios"
```

Branches (Ramas) y Cambiar entre ellas

Git permite trabajar en diferentes ramas para mantener un desarrollo organizado, facilitando la creación de características nuevas o corrección de errores sin afectar el código principal.

```
git checkout -b nueva-rama
```

Esto crea una nueva rama llamada `nueva-rama` y automáticamente te cambia a esa rama.

Si ya tienes una rama existente, puedes cambiarte a ella:

```
git checkout nombre-de-la-rama
```

Fusionar Branches: `git merge`

Cuando terminas de trabajar en una rama, puedes fusionarla con otra (normalmente con la rama principal `main` o `master`).

Por ejemplo, cambiamos a la rama `main`

```
git checkout main
```

y una vez allí podemos fusionar el contenido de alguna de las otras ramas:

```
git merge nombre-de-la-rama
```

Esto combinará los cambios de la `nombre-de-la-rama` en la rama actual.

Recuerda que puedes usar `git status` para ver el estado actual de tu repositorio, mostrándote qué archivos han sido modificados, cuáles están listos para ser confirmados (en el área de “staging”) y cuáles no se están rastreando aún.

1.4 GitHub

Git, como ya hemos visto, es una herramienta fundamental para el control de versiones de software, permitiendo a los desarrolladores gestionar cambios en sus proyectos de manera local y colaborativa. Sin embargo, cuando trabajamos en equipo o queremos almacenar nuestros proyectos en la nube, **GitHub** entra en escena como la extensión perfecta de Git.

1.4.1 Estableciendo la conexión

Una **llave pública SSH** es un componente de la criptografía de clave pública que se utiliza en el protocolo **SSH** (Secure Shell) para autenticar de manera segura la comunicación entre dos sistemas, generalmente para conectarse a un servidor remoto sin necesidad de introducir una contraseña en cada ocasión. Lo primero que haremos será generar una llave en nuestro computador:

Ejecuta el comando para generar la llave:

```
ssh-keygen -t ed25519 -C "tuemail@ejemplo.com"
```

Este comando genera una nueva llave SSH utilizando el algoritmo `ed25519` (recomendado). Asegúrate de reemplazar `tuemail@ejemplo.com` con el correo electrónico asociado a tu cuenta de GitHub.

Cuando se te pregunte dónde guardar la llave, presiona **Enter** para aceptar la ubicación predeterminada (`~/.ssh/id_ed25519`).

Te pedirá ingresar una frase de seguridad. Puedes dejarlo en blanco si no quieres una contraseña, pero agregar una frase de seguridad es una buena práctica de seguridad.

Añadir la llave SSH al agente SSH

El agente SSH se encarga de gestionar tus llaves SSH, permitiéndote no tener que ingresar la frase de seguridad cada vez que la uses.

Iniciar el agente SSH:

```
eval "$(ssh-agent -s)"
```

Añadir tu nueva llave SSH al agente:

```
ssh-add ~/.ssh/id_ed25519
```

Ahora, necesitas copiar la llave pública para añadirla a tu cuenta de GitHub.

Usa el siguiente comando para copiar el contenido de tu llave pública al portapapeles:

```
cat ~/.ssh/id_ed25519.pub
```

Esto mostrará el contenido de la llave pública en la terminal. Selecciona y copia todo el texto.

Añadir la llave SSH a GitHub

1. Inicia sesión en tu cuenta de GitHub.

2. Dirígete a **Settings** (Configuración) en la parte superior derecha.

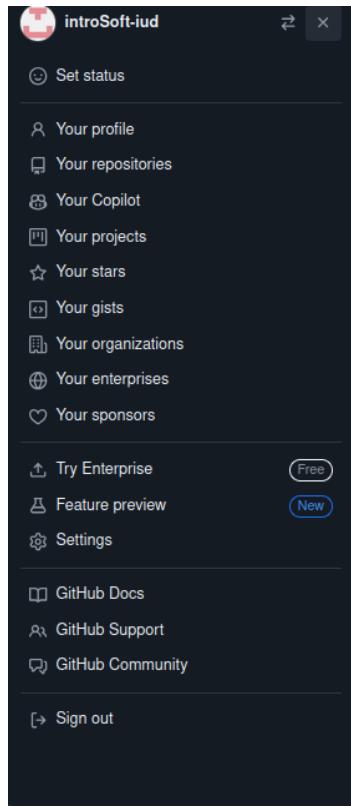


Figura 39.

3. En el menú de la izquierda, selecciona **SSH and GPG keys**.

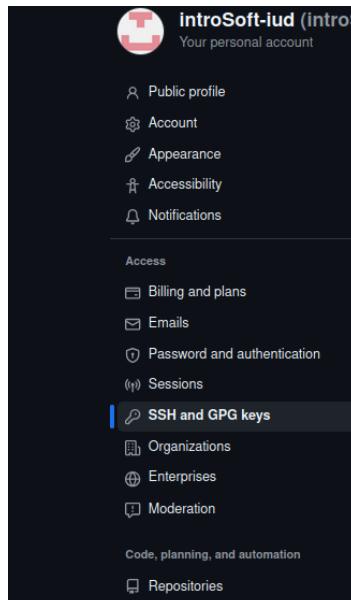


Figura 40.

4. Haz clic en el botón **New SSH Key**.



Figura 41.

5. Pega la llave pública que copiaste en el campo correspondiente y dale un nombre descriptivo (por ejemplo, "computador_vaio_casa")

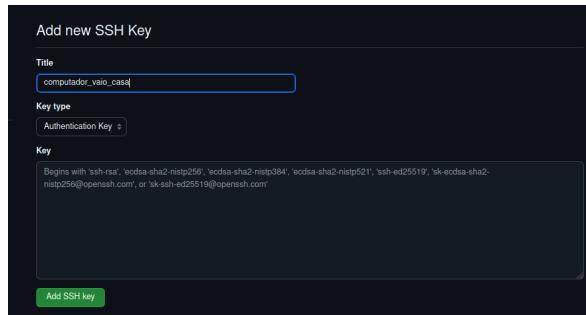


Figura 42.

6. Haz clic en **Add SSH Key**.

6. Probar la conexión

Una vez añadida la llave SSH a tu cuenta de GitHub, puedes verificar que todo está funcionando correctamente.

6.1 Ejecuta este comando para probar la conexión:

```
cat ~/.ssh/id_ed25519.pub
```

```
cat ~/.ssh/id_ed25519.pub
```

2 El flujo de trabajo Git Fork

El flujo de trabajo **Git Fork** es una estrategia comúnmente utilizada cuando varios desarrolladores colaboran en un proyecto de código abierto o en cualquier repositorio al que no tienen acceso directo para hacer cambios. Como lo aplicarás con tu grupo de trabajo para desarrollar la tarea 3, a continuación se explica cómo funciona:

2.1 Clonar un repositorio mediante un "fork"

El flujo de trabajo comienza con la creación de un **fork**. Un *fork* es una copia completa del repositorio que te permite trabajar de forma independiente en tu propia cuenta de GitHub (u otra plataforma Git). Este paso es esencial cuando no tienes permisos para hacer *push* directamente al repositorio principal (generalmente llamado el **repositorio upstream**).

Ejemplo: Si encuentras un proyecto interesante en GitHub pero no tienes permisos para modificarlo, puedes hacer un *fork*. Esto crea una copia del repositorio en tu cuenta.

2.2 Clonar tu repositorio fork a tu máquina local

Una vez que has hecho un *fork* del proyecto en tu cuenta, clonas tu copia del repositorio a tu máquina local para empezar a trabajar en los cambios.

```
git clone https://github.com/tu-usuario/tu-fork-del-proyecto.git
cd tu-fork-del-proyecto
```

2.3 Crear una rama para los cambios

Antes de comenzar a modificar el código, es una buena práctica crear una nueva rama para trabajar en los cambios. Esto ayuda a mantener el repositorio organizado y a evitar modificar directamente la rama principal (*main* o *master*).

```
git checkout -b mi-nueva-rama
```

2.4 Hacer cambios y commits

Realizas los cambios necesarios en tu copia local del repositorio. Después de realizar los cambios, puedes agregar y hacer *commits*.

```
git add .
git commit -m "Descripción de los cambios"
```

2.5 Subir los cambios a tu fork en GitHub

Los cambios se suben a tu repositorio *fork* en GitHub o GitLab (o la plataforma de Git que estés utilizando).

```
git push origin mi-nueva-rama
```

2.6 Crear un "Pull Request" (PR)

Una vez que los cambios están en tu repositorio *fork* en GitHub, puedes solicitar que se integren al repositorio original (*upstream*). Esto se hace mediante un **Pull Request** (PR). En el PR, los mantenedores del proyecto pueden revisar tus cambios y decidir si los aceptan o no.

Pull Request: Es una solicitud para que los mantenedores del repositorio original revisen y fusionen los cambios propuestos desde tu *fork*.

Ejemplo: Si has corregido un error o añadido una nueva funcionalidad al proyecto original, puedes enviar un PR para que esos cambios sean revisados y potencialmente aceptados en el código base.

2.7 Sincronizar tu fork con el repositorio original (*upstream*)

A medida que el repositorio original sigue evolucionando, debes mantener tu *fork* actualizado. Para ello, primero necesitas agregar el repositorio original como un **remoto** (usualmente llamado *upstream*).

```
git remote add upstream https://github.com/usuario-del-proyecto-original/proyecto-original.git
```

Luego, puedes obtener los cambios del repositorio original y fusionarlos con tu copia local:

```
git fetch upstream
git merge upstream/main # o 'master', según la rama que uses
```

4 Datos

1 La internet

<https://www.youtube.com/watch?v=x3c1ih2NJEg>

Cable coaxial



Figura 43.

1.1 Fibra óptica

1.2 Cómo los datos son transmitidos

1.3 Cómo los datos son encriptados

1.3.1 El Algoritmo de Diffie-Hellman

En 1976 Withfield Hiffie y Martin Hellman desarrollaron las bases del algoritmo de encriptado de mensajes más usado actualmente. El problema puede ser descrito de la siguiente manera:

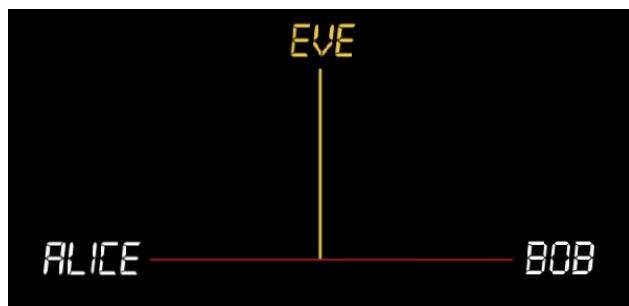


Figura 44.

Alice y Bob quieren compartir mensajes encriptados sin que eve descubra el mensaje. La solución es basada en las llamadas funciones de una sola vía:

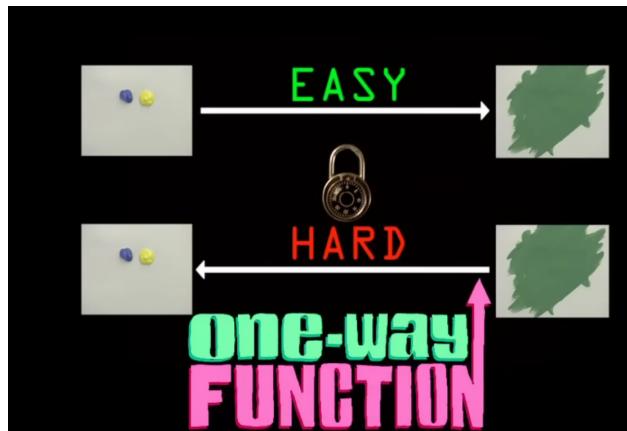


Figura 45.

La solución funciona de la siguiente manera:

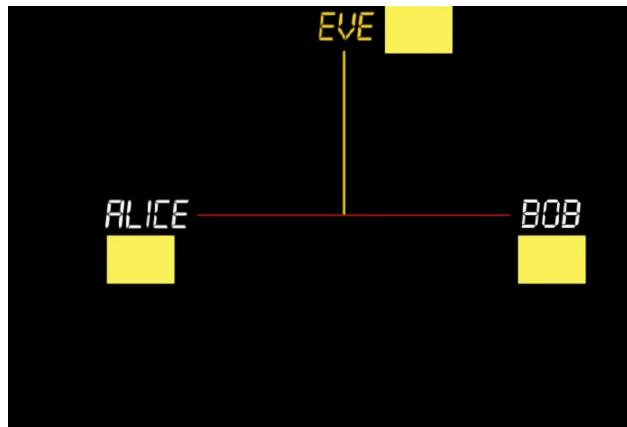


Figura 46.

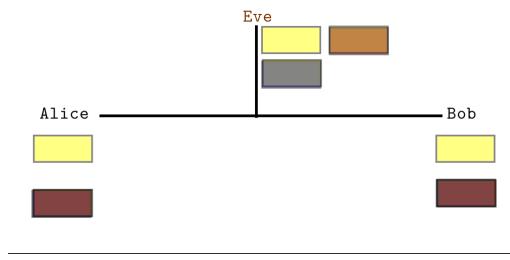


Figura 47.

https://www.youtube.com/watch?v=YEBfamv_-do

5 Introducción: Qué es Machine Learning?

The field of study that gives computers the ability to learn without being explicitly programmed”

Arthur Samuel (1959)

El Machine Learning, o Aprendizaje Automático, es un campo de estudio que dota a las computadoras de la capacidad de aprender sin ser explícitamente programadas para realizar tareas específicas. Esta definición, propuesta por Arthur Samuel en 1959, captura la esencia de lo que hace que el Machine Learning sea una disciplina revolucionaria dentro de la ciencia de la computación y la inteligencia artificial.

Sabias qué, Para aprender más:

Arthur Samuel es una figura histórica en el campo de la Inteligencia Artificial y el Machine Learning, y su trabajo pionero en la década de 1950 y 1960 sentó las bases para muchas de las técnicas y enfoques que hoy son comunes en esta disciplina. Su contribución más destacada fue en el ámbito del aprendizaje automático, donde desarrolló uno de los primeros programas capaces de aprender y mejorar con el tiempo: un programa de juego de damas.

El programa de damas de Samuel utilizaba lo que ahora se considera una forma temprana de algoritmos de aprendizaje: técnicas de búsqueda y optimización que permitían al programa mejorar su juego al competir contra sí mismo y analizar cuáles estrategias conducían a la victoria con mayor frecuencia. A través de este proceso, el programa aprendía y ajustaba sus tácticas, demostrando un aumento significativo en su habilidad para jugar damas.

A diferencia de la programación tradicional, donde los algoritmos siguen instrucciones precisas para ejecutar tareas, el Machine Learning permite a las máquinas mejorar su rendimiento y tomar decisiones basándose en el análisis de datos. A través del estudio de patrones y la aplicación de modelos estadísticos, las computadoras pueden predecir resultados, adaptarse a nuevos escenarios y aprender de las experiencias previas sin intervención humana directa. Esta capacidad ha impulsado el desarrollo de aplicaciones en una amplia gama de áreas, como la detección de fraudes, la personalización de contenido en línea, diagnósticos médicos y la automatización de vehículos.

1 Categorías de Sistemas en Aprendizaje Automático

Los sistemas de aprendizaje automático se clasifican en diversas categorías, dependiendo de su método de aprendizaje y la forma en que realizan predicciones o crean contenido. A continuación, se describen las principales categorías:

1. **Aprendizaje Supervisado:** Esta modalidad se basa en el aprendizaje a partir de ejemplos previamente etiquetados. El sistema intenta aprender una función que mapea entradas a salidas, utilizando un conjunto de pares de entrada-salida conocidos. Su principal objetivo es predecir la salida para nuevas entradas después de haber sido entrenado en un conjunto de datos de entrenamiento.
2. **Aprendizaje No Supervisado:** A diferencia del aprendizaje supervisado, en el aprendizaje no supervisado, los datos de entrada no están etiquetados. El sistema intenta aprender la estructura o patrones subyacentes de los datos sin instrucciones explícitas, agrupando la información en categorías basadas en similitudes o diferencias entre los elementos del conjunto de datos.
3. **Aprendizaje por Refuerzo:** En el aprendizaje por refuerzo, el sistema aprende a tomar decisiones a través de la experimentación y la interacción con un entorno. Se basa en el concepto de recompensas y penalizaciones: el sistema realiza acciones y recibe retroalimentación en forma de recompensas (positivas o negativas), ajustando sus estrategias para maximizar la suma total de recompensas obtenidas.
4. **Inteligencia Artificial Generativa (Generative AI):** Los sistemas de IA generativa se enfocan en crear contenido nuevo y original que sea indistinguible de los ejemplos reales. Utilizan técnicas avanzadas para generar datos que no existían previamente en el conjunto de entrenamiento, encontrando aplicaciones en la creación de imágenes, texto, música y más.

Cada una de estas categorías abarca una amplia gama de algoritmos y técnicas específicas diseñadas para resolver distintos tipos de problemas. La elección del tipo de sistema de ML adecuado depende del problema específico a resolver, la naturaleza de los datos disponibles y los objetivos del proyecto.

Para tener un panorama de como está evolucionando este campo en la actualidad, te invito a que veas la conferencia del profesor Andrew Ng sobre *Opportunities in AI* <https://www.youtube.com/watch?v=5p248yoa3oE>

En la actualidad, una de las herramientas más usadas es la del aprendizaje supervisado debido a su solidez en la realización de tareas de clasificación, mientras que la innovación en IA generativa promete revolucionar los sectores creativos.

Esta configuración está lejos de ser estática. Se anticipa una evolución constante, impulsada por avances tecnológicos y nuevas exigencias del mercado.

Así es como Andrew Ng estima, en el video recomendado, la participación de las tecnologías de AI en la actualidad:

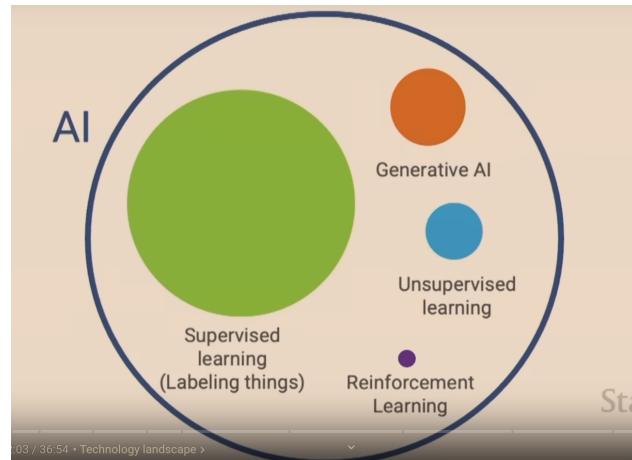


Figura 48. Cuota de mercado de diversas tecnologías de IA, destacando la posición predominante de ciertas metodologías sobre otras en el contexto actual.

Y así es como estima que será de aquí a algunos años cuando las tecnologías de AI generativas maduren un poco:

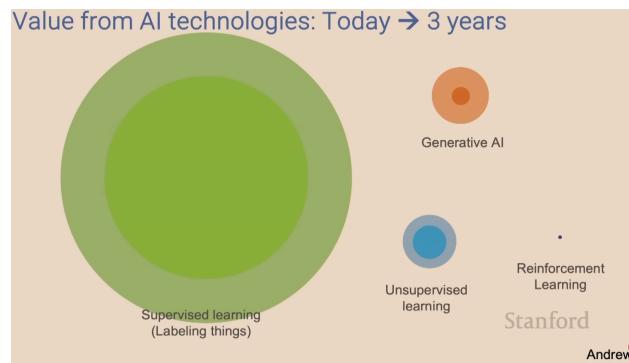


Figura 49. Proyección futura que visualiza la evolución esperada en la adopción de tecnologías de IA.

En esta, los círculos más oscuros en el centro simbolizan la participación actual, mientras que los círculos periféricos y más claros indican el potencial de crecimiento de cada tecnología.

Si las predicciones del Dr. Ng se confirman, el aprendizaje supervisado seguirá desempeñando un papel crucial en el campo de la inteligencia artificial. Por esta razón, nuestro curso iniciará con un enfoque en los algoritmos más fundamentales y relevantes del aprendizaje supervisado en la Unidad 1. A continuación, en la Unidad 2, abordaremos diversos algoritmos de aprendizaje no supervisado. Finalmente, en la Unidad 3, nos adentraremos en las herramientas de IA generativa, las cuales están cobrando gran relevancia en la actualidad. ¡Comencemos!

2 Aprendizaje Supervisado: Etiquetar cosas

Los algoritmos de aprendizaje supervisado aprenden a partir de conjuntos de datos extensos que contienen ejemplos etiquetados con las respuestas correctas. Se denominan “supervisados” porque durante el proceso de entrenamiento, al sistema se le suministran tanto los datos de entrada como las salidas correctas asociadas. Esta metodología permite que el algoritmo identifique patrones y relaciones dentro de los datos. Una vez entrenado con un número suficiente de ejemplos correctos, el sistema es capaz de hacer predicciones o clasificaciones sobre nuevos conjuntos de datos para los cuales no se conocen las respuestas.

En términos sencillos, los algoritmos de aprendizaje supervisado consisten en sistemas diseñados para aprender mapeos (una función f) entre variables de entrada (x) y variables de salida (y).

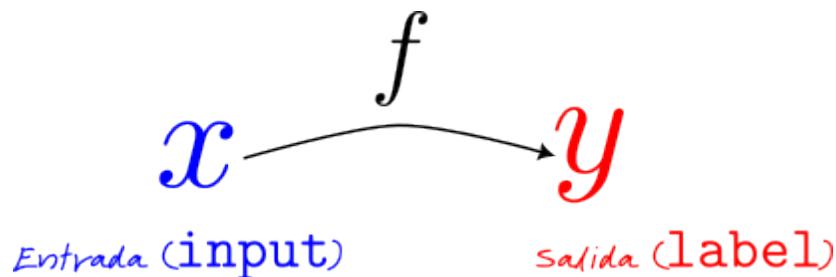


Figura 50. Ilustración gráfica del mapeo en aprendizaje supervisado

El aprendizaje supervisado se fundamenta en los siguientes conceptos:

- **Datos**
- **Modelo**
- **Entrenamiento**
- **Evaluación**
- **Inferencia**

Los Datos son el punto de partida del Machine Learning. En las figuras 3 y 4, presentamos dos ejemplos de *data sets* habituales en problemas de aprendizaje supervisado. Cada *data set* se organiza en Ejemplos (Examples, representados por las 'filas') que incluyen Características (**Features**) y una Etiqueta (**Label**). Las **Features** son los datos de entrada que se utilizan para hacer predicciones o tomar decisiones; pueden ser numéricas, como el tamaño de una casa en metros cuadrados en la figura 52, o categóricas, como los síntomas en la figura 51. Por otro lado, los **Labels** son las respuestas o resultados correctos para cada ejemplo; en un contexto de clasificación, podrían ser categorías como “spam” o “no spam” para un filtro de correo electrónico, mientras que en un contexto de regresión, podrían ser valores continuos, como el precio de una casa.

	Features	Labels
Examples		
	Datos de Entrada (Síntomas)	Etiqueta (Diagnóstico)
	Fiebre, tos, dificultad para respirar	COVID-19
	Dolor de cabeza, fiebre, rigidez de nuca	Meningitis
	Sed excesiva, hambre intensa, pérdida de peso	Diabetes Tipo 1
	Fatiga, debilidad, palidez	Anemia

Figura 51. Ejemplos de datos categóricos para clasificación con algoritmos de aprendizaje supervisado

Features		Features	Labels
Tamaño de la Casa (m ²)	Edad de la Casa (Años)	Precio de Venta (miles de \$)	
100	5	300	
150	10	400	
200	15	500	
250	20	600	
300	25	700	

Figura 52. Ejemplos de datos numéricos para regresión lineal aplicando algoritmos de aprendizaje supervisado en la predicción de precios inmobiliarios

El Modelo es una representación matemática de la relación entre las Features y los Labels. Este puede tomar diversas formas, desde simples líneas rectas en la regresión lineal hasta estructuras complejas en redes neuronales profundas. La elección del modelo depende de la naturaleza del problema, la complejidad de los datos y el tipo de predicción que se desea realizar.

El Entrenamiento implica ajustar los parámetros del modelo utilizando los datos. Durante esta fase, el modelo aprende gradualmente la relación entre las Features y los Labels, minimizando el error entre las predicciones y los valores reales. Este proceso se realiza a través de algoritmos de optimización, como gradiente descendente.

La Evaluación del modelo se lleva a cabo después del entrenamiento, utilizando un conjunto de datos diferente, conocido como el conjunto de prueba. Esta fase mide la capacidad del modelo para generalizar su aprendizaje a nuevos datos, proporcionando una estimación de su rendimiento en el mundo real.

La Inferencia es el uso final del modelo entrenado y evaluado para hacer predicciones sobre datos nuevos y no vistos. En esta etapa, el modelo aplica lo que ha aprendido para proporcionar resultados útiles, como clasificar correos electrónicos en “spam” o “no spam” o predecir precios de casas con base en sus características.

2.1 Regresión & Clasificación

Un modelo de regresión predice valores continuos. Por ejemplo, los modelos de regresión realizan predicciones que responden a preguntas como:

- ¿Cuál es el valor de una casa en el barrio Laureles, basado en su tamaño y la edad de la construcción?

Por otro lado, un modelo de clasificación predice valores discretos. Por ejemplo, los modelos de clasificación realizan predicciones que responden a preguntas como:

- Dados ciertos síntomas, ¿el diagnóstico es gripe, COVID-19 o una alergia?

2.1.1 Regresión

Retomemos el análisis de la relación entre el tamaño de una casa y su precio de venta, tal como se muestra en la Tabla 11

Tamaño de la Casa (m ²)	Precio de Venta (miles de \$)
100	288.29
125	428.96
150	438.37
175	426.53
200	527.13
225	526.83
250	576.71
275	662.1
300	604.34

Tabla 11. Relación entre el Tamaño de la Casa y el Precio de Venta

Supongamos que un amigo está interesado en vendernos su casa, que tiene un área de 210 m². Dados los datos anteriores, nos preguntamos: ¿cuál sería un precio de venta justo de acuerdo con el mercado para su propiedad?

La distribución de los datos se visualiza en la Figura 53.

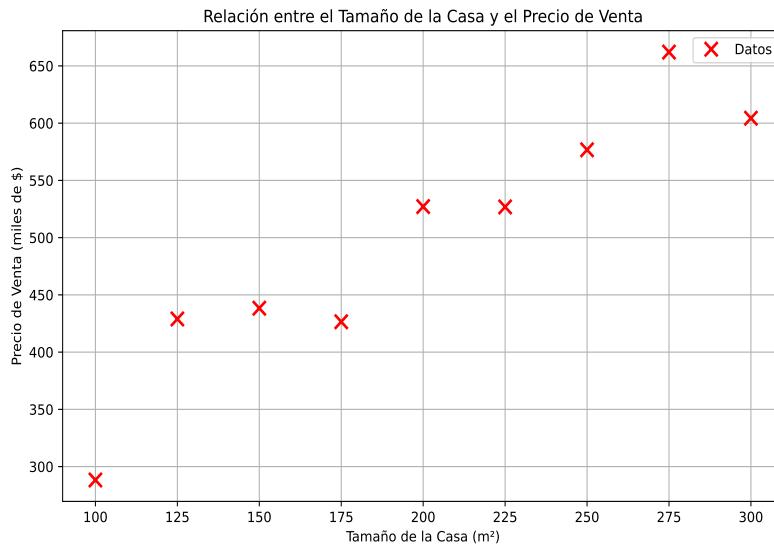


Figura 53. Relación entre el Tamaño de la Casa y el Precio de Venta

Un enfoque que podría tomar un algoritmo de machine learning sería ajustar una línea recta a los datos, como se muestra en la Figura 54 , Con este modelo, podríamos sugerir a nuestro amigo que su casa tiene un valor de mercado de aproximadamente 518 k.

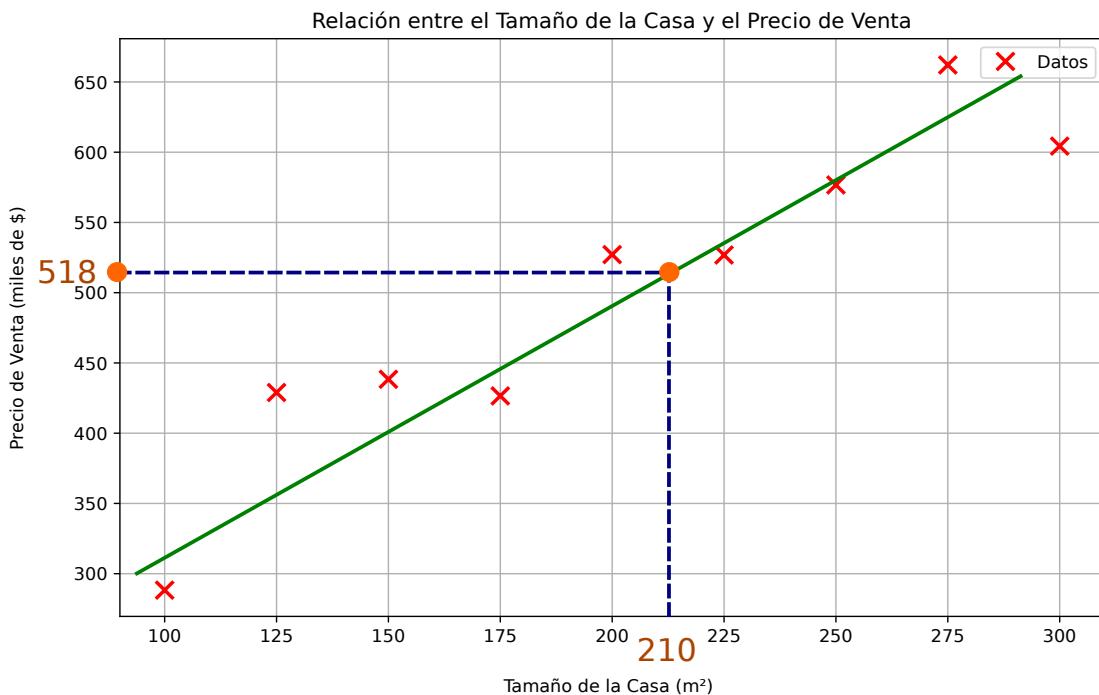


Figura 54. Relación entre el Tamaño de la Casa y el Precio de Venta ajustado por una recta.

No obstante, nuestro amigo considera que la línea recta quizás no sea el mejor modelo y sugiere explorar uno más complejo, como la curva representada en la Figura 55.

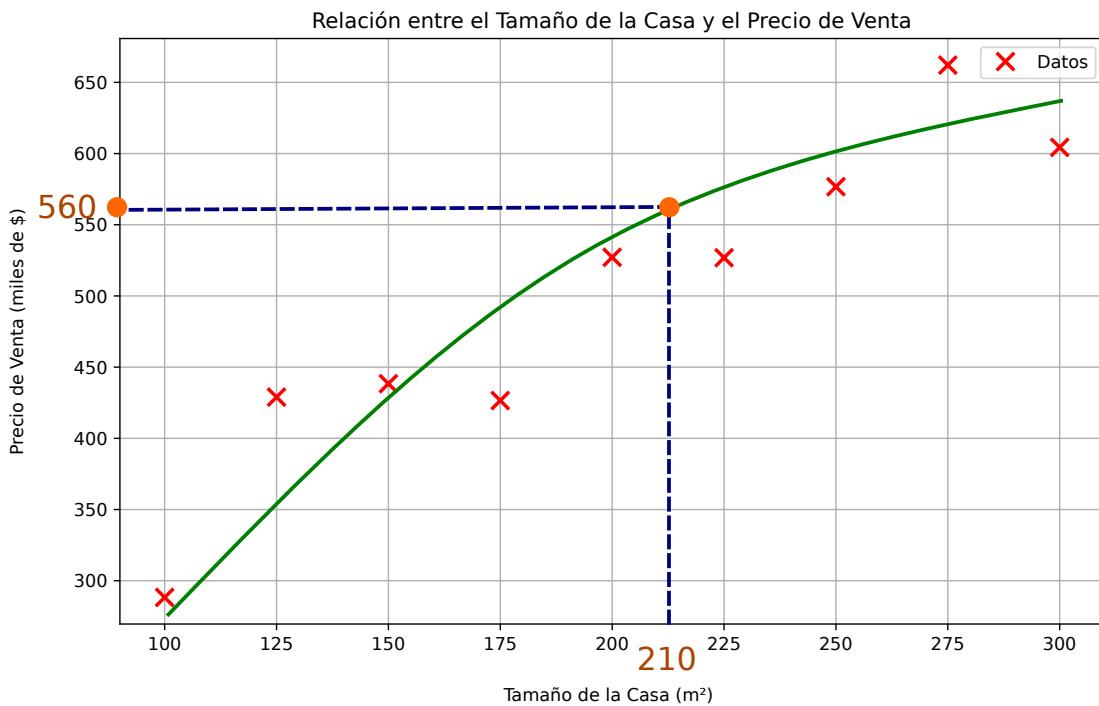


Figura 55. Relación entre el Tamaño de la Casa y el Precio de Venta ajustado por una curva.,

Según este modelo más complejo, la casa de nuestro amigo tendría un valor de 560 k.

¿Cómo determinamos cuál es el modelo más adecuado para establecer un precio justo?

Lo veremos más adelante.

Este tipo de situación ejemplifica un escenario típico de **aprendizaje supervisado**. Suministramos al algoritmo un conjunto de datos y le asignamos la tarea de definir la etiqueta (**Label**); específicamente, identificar el precio más apropiado para la casa de nuestro amigo. Esta modalidad específica de aprendizaje supervisado se conoce como **regresión**.

2.1.2 Clasificación

Los algoritmos de clasificación constituyen una de las categorías fundamentales del aprendizaje supervisado. En contraste con los algoritmos de regresión, que se orientan hacia la predicción de valores continuos, los algoritmos de clasificación apuntan a la identificación de categorías específicas. Dichas categorías no necesariamente adoptan forma numérica; pueden, por ejemplo, discernir si una fotografía muestra un perro o un gato, o diagnosticar si una serie de síntomas se asocian a condiciones médicas tales como diabetes o cáncer. Sin embargo, es posible que las categorías se representen mediante números (como 1, 2 o 3), pero, a diferencia de la regresión, estos números se interpretan como distintas clases discretas sin consideración de los valores intermedios entre ellos.

Imaginemos que contamos con información sobre transacciones bancarias clasificadas como legítimas (triángulos) o fraudulentas('x'), basadas en su frecuencia (número de transacciones por hora) y el monto involucrado. Los datos recopilados se distribuyen tal como se muestra en la Figura 56

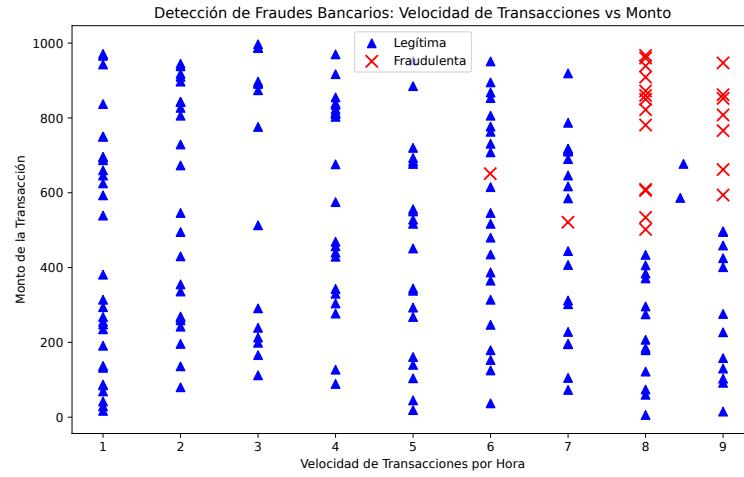


Figura 56. Distribución de transacciones fraudulentas y legítimas. Los triángulos representan transacciones legítimas y las 'x' indican aquellas consideradas fraudulentas.

Un algoritmo de clasificación sería capaz de establecer un límite de separación entre estos dos conjuntos de transacciones, como se ilustra con una línea punteada en la Figura 57. De esta manera, cuando se efectúe una nueva transacción, representada por el rombo naranja en la figura, el sistema podría alertarnos si se trata de una operación potencialmente sospechosa, facilitando así la detección temprana y prevención del fraude bancario.

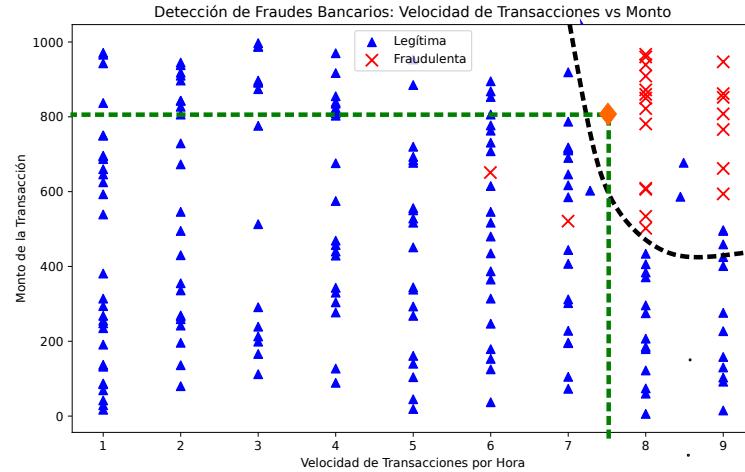


Figura 57. Clasificación de Transacciones Bancarias: Distribución de transacciones bancarias, diferenciando entre las legítimas (representadas por triángulos) y las fraudulentas (indicadas con 'x'). Una línea punteada marca el límite de separación propuesto por el algoritmo de clasificación, permitiendo identificar nuevas transacciones, como la señalada con un rombo naranja, y evaluar si son sospechosas de fraude.

3 La regresión lineal

En la sección 2.1.1, exploramos cómo determinar el modelo más adecuado para establecer un precio justo para la compra de la casa de nuestro amigo, es decir, cuál función f sería la más adecuada para realizar el mapeo más preciso entre las características (features) y las etiquetas (labels), como se ilustra en la Figura 58

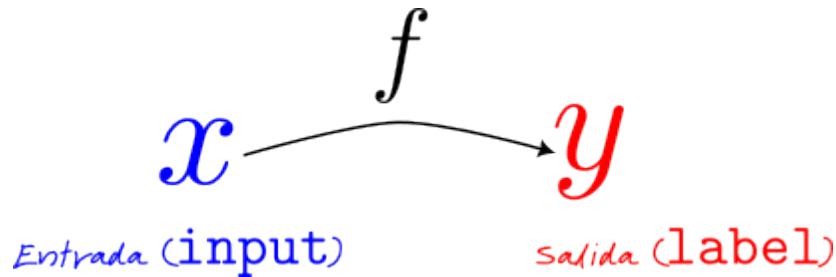


Figura 58. Ilustración gráfica del mapeo en aprendizaje supervisado

Una opción simple podría ser una función lineal. El modelo de regresión lineal es, posiblemente, el algoritmo de aprendizaje automático más utilizado en la actualidad. Además, muchos de los conceptos que abordaremos aquí son aplicables a otros modelos de aprendizaje de máquinas.

Primeramente, recordemos que una línea recta se describe mediante la ecuación:

$$y = mx + b. \quad (1)$$

A manera de ejemplo, la Figura 59 muestra la recta

$$y = 0.5x + 4$$

donde $m = 0.5$ y $b = 4$.



Figura 59. Ejemplo de una recta definida por $y = 0.5x + 4$, donde $m = 0.5$ y $b = 4$.

Esta ecuación establece un mapeo entre x y y , donde m es la pendiente de la recta y b es el punto en el eje y cuando $x = 0$, es decir, el intercepto con el eje y . En términos de funciones, al eje y se le denomina ‘las imágenes de x ’ o $f(x)$ ¹.

Retomemos el ejemplo donde intentamos estimar el valor de una casa basado en su tamaño (ver Figura 60).

1. Léase ‘imágenes de x ’ o simplemente $f(x)$

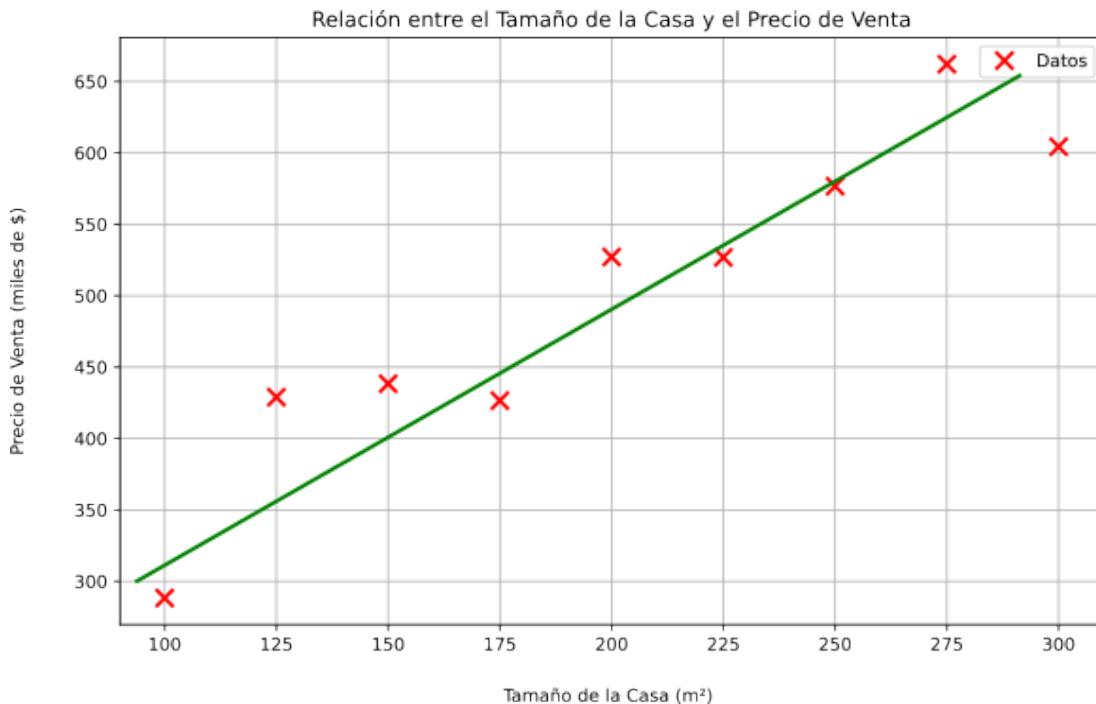


Figura 60.

La línea recta no necesariamente pasa por todos los puntos, pero establece claramente una relación entre el precio de la casa y su tamaño.

Por convención, en Machine Learning, escribimos la ecuación para el modelo de regresión lineal con una notación ligeramente diferente:

$$f_{b,w}(x) = b + w x \quad (2)$$

Donde:

- $f_{b,w}(x) = \hat{y}$ es el label o salida esperada.
- b es el sesgo, a veces llamado w_0 .
- w es el peso de la característica x (es decir, la pendiente de la recta).
- x es una característica (una entrada conocida). También llamada variable ‘feature’

Recordemos nuestro conjunto de datos presentado en la Tabla 11

	Features (x)	Labels (y)
	Tamaño de la Casa (m ²)	Precio de Venta (miles de \$)
Examples	100	288.29
	125	428.96
	150	438.37
	175	426.53
	200	527.13
	225	526.83
	250	576.71
	275	662.1
	300	604.34

$(x^{(i)}, y^{(i)})$

Figura 61. Conjunto de Datos de Ejemplo: Representación de un conjunto de datos que consta de pares de valores (x, y) , donde cada par representa un ejemplo específico, como el tamaño de una casa (x^i) y su precio correspondiente (y^i).

El conjunto de datos consta de un número n de filas (denominadas ejemplos') compuestas por pares (x, y) . En la figura, denotamos el i -ésimo par como $(x^{(i)}, y^{(i)})$. Una vez que hemos establecido los parámetros b y w que definen nuestro modelo f , podemos utilizarlo para estimar el valor \hat{y} dado un valor de x que no esté presente en nuestro conjunto de datos, como se muestra en la fórmula:

$$f_{b,w}(x) = \hat{y}$$

En el contexto de nuestro ejemplo de la casa, como se ilustra en la Figura 54, el valor estimado de \hat{y} es 518k para una casa de 510 m^2 en el barrio Laureles. Es decir,

$$f_{b,w}(510) = w \times 510 + b = 518,$$

como se muestra en la Figura 62.

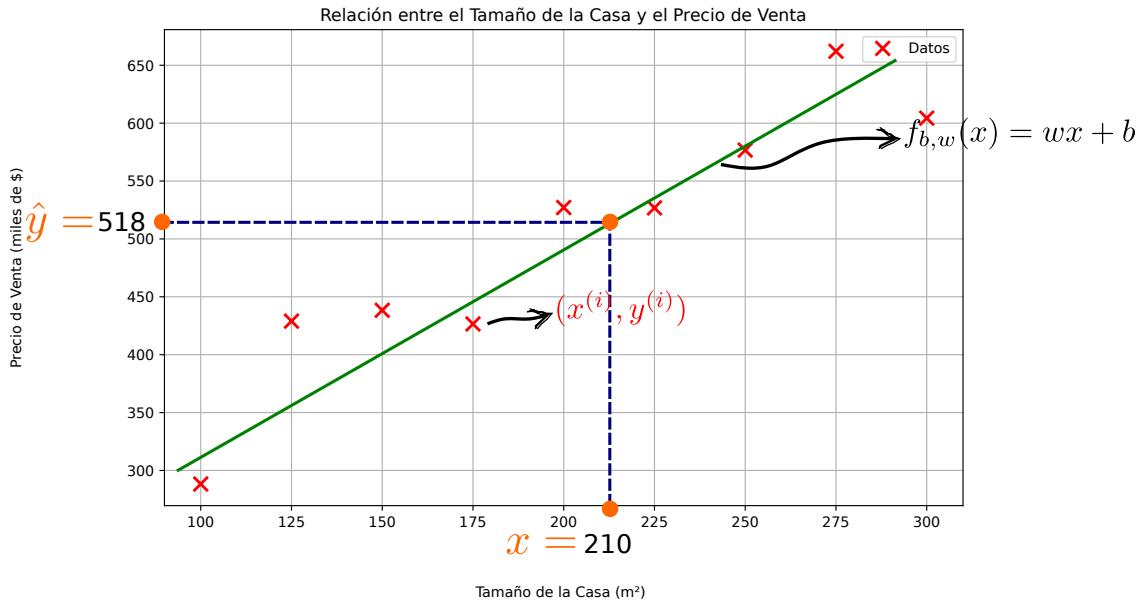


Figura 62. Estimación de Precios de Casas: Ilustración de la aplicación del modelo de regresión lineal para estimar el precio de una casa basado en su tamaño. La línea representa la función de regresión $f_{b,w}(x) = wx + b$, adaptada a los datos. El punto específico donde $x = 510\text{ m}^2$ muestra la estimación del precio ($y = 518\text{k}$) de una casa en el barrio Laureles.

3.1 Más de una feature

El precio de la casa no está determinado, necesariamente, por una única variable conocida de entrada (o feature) como $x = \text{Tamaño en m}^2$. Otras variables de entrada que podría tener nuestro modelo pueden ser $x_2 = \text{Años de antiguedad}$ o $x_3 = \text{Número de habitaciones}$, etc. En general, una regresión lineal es una predicción basada en una suma ponderada de n variables conocidas de entrada x_1, x_2, \dots, x_n , entonces, haciendo $b = w_0$, en la Eq. 2, el valor predicho por el modelo es

$$\hat{y} = w_0 + w_1 x_1 + \dots + w_n x_n. \quad (3)$$

Recordando nuestro curso de álgebra lineal, observamos que podemos escribirnos esta ecuación como el producto vectorial

$$\hat{y} = \mathbf{w} \cdot \mathbf{x}. \quad (4)$$

donde $\mathbf{w} = (w_0, w_1, \dots, w_n)$ es vector de parámetros del modelo, que contiene además el intersepto $w_0 = b$ y $\mathbf{x} = (1, x_1, \dots, x_n)$ es el i -ésimo vector de features de entrada conteniendo a x_0, x_1, \dots, x_n con x_1 siempre igual a 1.

De manera que en notación matricial el modelo de regresión lineal es escrito como

$$\begin{aligned}\hat{y} &= \mathbf{w}^\top \mathbf{x} \\ &= (w_0, w_1, \dots, w_n) \begin{pmatrix} x_0 \\ \vdots \\ x_n \end{pmatrix}\end{aligned}$$

Nota: En nuestro curso dejamos este tipo de ecuaciones para que nuestro cerebro se vaya «entrenando» y familiarizando con nociones un poco más avanzadas y completas.

3.2 Entrenamiento: la función de costo

Los parámetros w y b son las variables que ajustamos para mejorar la eficacia del modelo, a este proceso se le conoce como el **entrenamiento**. En ciertos contextos, estos parámetros también se denominan coeficientes o pesos. Analicemos el papel que juegan estos parámetros: dependiendo de los valores asignados a w y b , obtenemos diferentes resultados para f . La Figura 63 ilustra algunos ejemplos de funciones f generadas a partir de diversos valores de w y b .

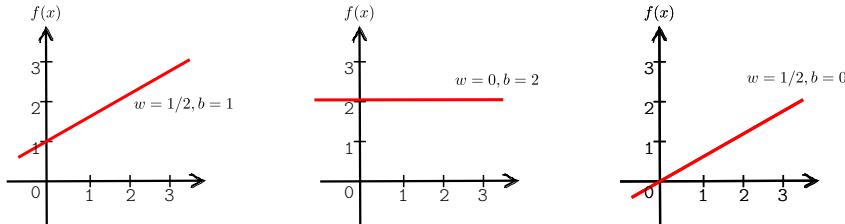


Figura 63. Ejemplos de Funciones Lineales con Diferentes Parámetros: Esta figura ilustra cómo las variaciones en los parámetros w y b afectan la forma de la función lineal $f(x) = wx + b$. Cada línea3a representa una configuración única de w (la pendiente) y b (el intersepto con el eje y), demostrando la influencia directa de estos parámetros en la posición y orientación de la recta en el plano cartesiano. Las intersecciones con los ejes y las variaciones de pendiente ilustran visualmente la diversidad de comportamientos que el modelo puede adoptar en función de sus parámetros.

Para evaluar la eficacia de nuestro modelo f , es decir, para encontrar los mejores valores de b y w , utilizamos la denominada ‘función de costo’².

Considerando un conjunto de datos con m ejemplos de entrenamiento y un par de datos $(x^{(i)}, y^{(i)})$, la función de costo evalúa cuán diferentes son los valores reales $y^{(i)}$ de las predicciones del modelo $\hat{y}^{(i)} = f(x^{(i)})$. Una de las funciones de costo más comunes es la función de costo cuadrático medio $J(w, b)$, expresada como:

$$J_{wb}(x) = \frac{1}{2m} \sum_{i=1}^m [\hat{y}^{(i)} - y^{(i)}]^2 \quad (5)$$

$$= \frac{1}{2m} \sum_{i=1}^m [f(x^{(i)}) - y^{(i)}]^2. \quad (6)$$

2. En inglés se la menciona como *cost function* o, en ocasiones, *loss function*

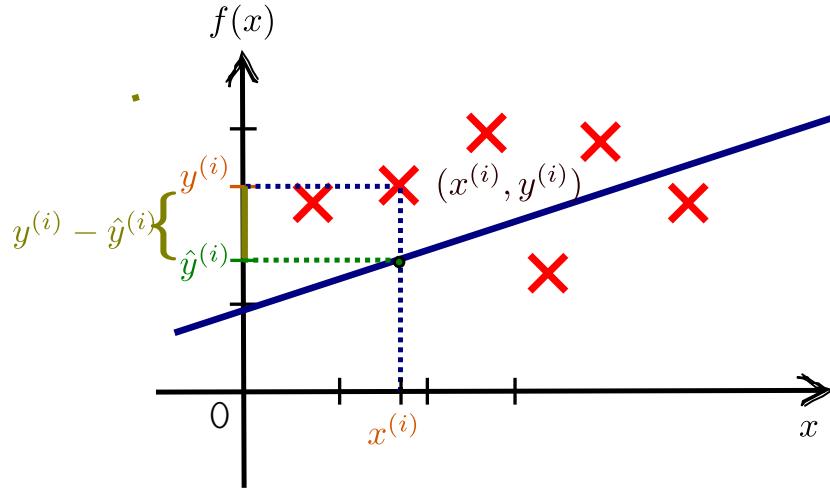


Figura 64.

La esencia de esta función de costo es calcular un número proporcional a la distancia entre el valor de la etiqueta $y^{(i)}$ y la predicción del modelo $\hat{y}^{(i)}$.

En particular, calcula el valor promedio del cuadrado de esta distancia. Los valores óptimos de w y b son aquellos para los cuales J es mínimo, es decir, los valores que minimizan la discrepancia entre las predicciones del modelo y los ejemplos etiquetados.

Si bien la función de costo puede parecer compleja, es importante no distraerse con su apariencia. Por ejemplo, está normalizada por un factor de $1/2m$, y la diferencia entre el ejemplo etiquetado $y^{(i)}$ y la predicción $\hat{y}^{(i)}$ se eleva al cuadrado. Estas particularidades se introducen en la función para facilitar y agilizar el proceso de minimización de J .

El objetivo ahora es minimizar J como una función de w y b

3.3 Entrnamiento y costo: minimizando $J(w, b)$

Queremos encontrar los valores de w y b que hacen que J sea lo más pequeño posible. Para apreciar que significa esto vamos a simplificar el problema haciendo $b=0$ por el momento. Esto resulta en una versión simplificada del modelo de regresión lineal

$$f_w(x) = wx$$

Así tendríamos solamente el parámetro w y la función de costo luciría como

$$J(w) = \frac{1}{2m} \sum_{i=1}^m [f_w(x^{(i)}) - y^{(i)}]^2 \quad (7)$$

$$= \frac{1}{2m} \sum_{i=1}^m [wx^{(i)} - y^{(i)}]^2 \quad (8)$$

Bajo este modelo simplificado, buscamos el valor de w que minimice $J(w)$.

El panel izquierdo de la Figura 65 muestra cuatro puntos representativos de un conjunto de entrenamiento $(x, y) = \{(0, 0), (1, 1), (2, 2), (3, 3)\}$. El panel derecho muestra el valor de la función de costo $J(w)$ para distintos valores de w .

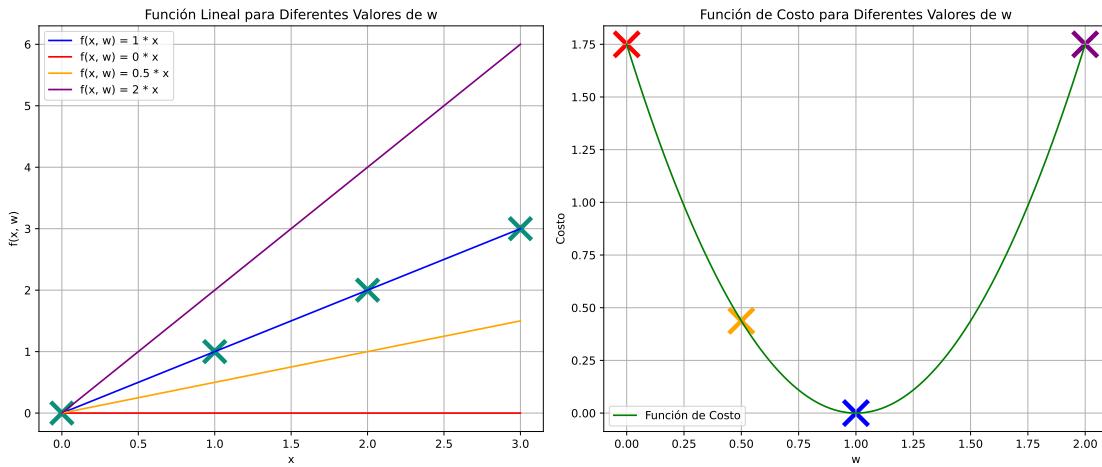


Figura 65.

Por ejemplo, para

$w = 1$:

$$J(1) = \frac{1}{8} [(1 \times 0 - 0)^2 + (1 \times 1 - 1)^2 + (1 \times 2 - 2)^2 + (1 \times 3 - 3)^2] = \frac{1}{8} [0 + 0 + 0 + 0] = 0$$

corresponde a la x azul en el panel derecho y la línea azul en el panel izquierdo.

De manera similar, podemos calcular los valores de la función de costo para otros valores de w :

Para

$w = 0$:

$$J(0) = \frac{1}{8} [(0 \times 0 - 0)^2 + (0 \times 1 - 1)^2 + (0 \times 2 - 2)^2 + (0 \times 3 - 3)^2] = \frac{1}{8} [0 + 1 + 4 + 9] = 1.75$$

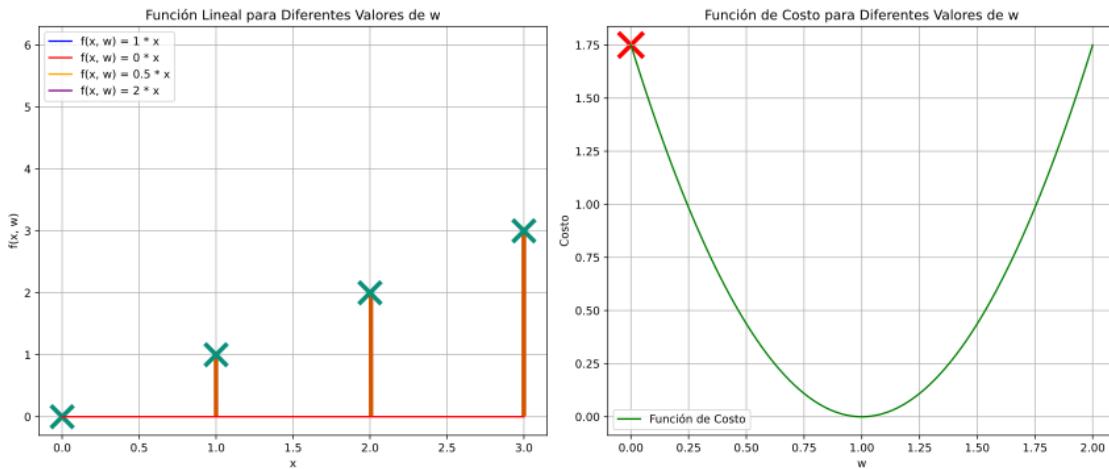


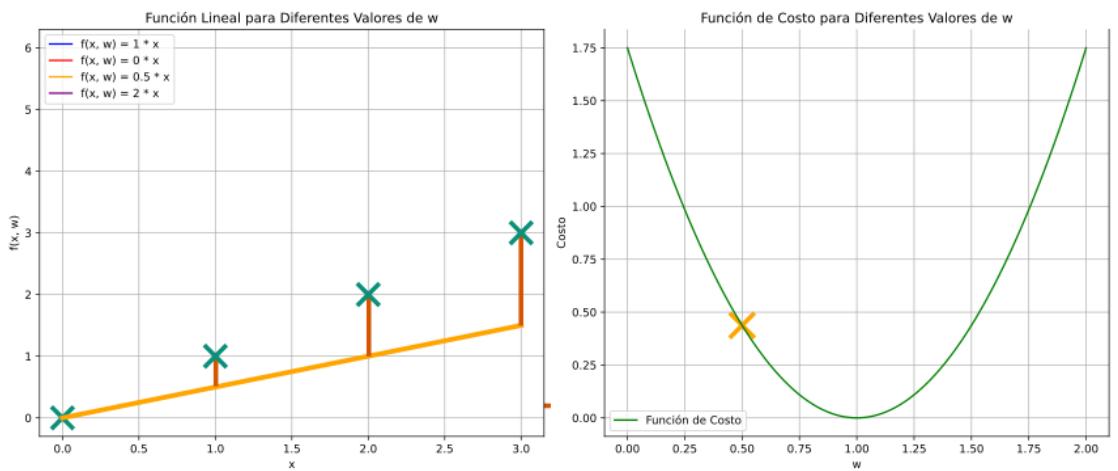
Figura 66.

Que corresponde a la **x** roja en el panel derecho

Para

$w = 0.5$:

$$J(0.5) = \frac{1}{8} [(0.5 \times 0 - 0)^2 + (0.5 \times 1 - 1)^2 + (0.5 \times 2 - 2)^2 + (0.5 \times 3 - 3)^2] = \frac{1}{8} [0 + 0.25 + 1 + 2.25] = 0.4375$$



Que corresponde a la **x** amarilla en el panel derecho

Para

$w = 2$:

$$J(2) = \frac{1}{8} [(2 \times 0 - 0)^2 + (2 \times 1 - 1)^2 + (2 \times 2 - 2)^2 + (2 \times 3 - 3)^2] = \frac{1}{8} [0 + 1 + 4 + 9] = 1.75$$

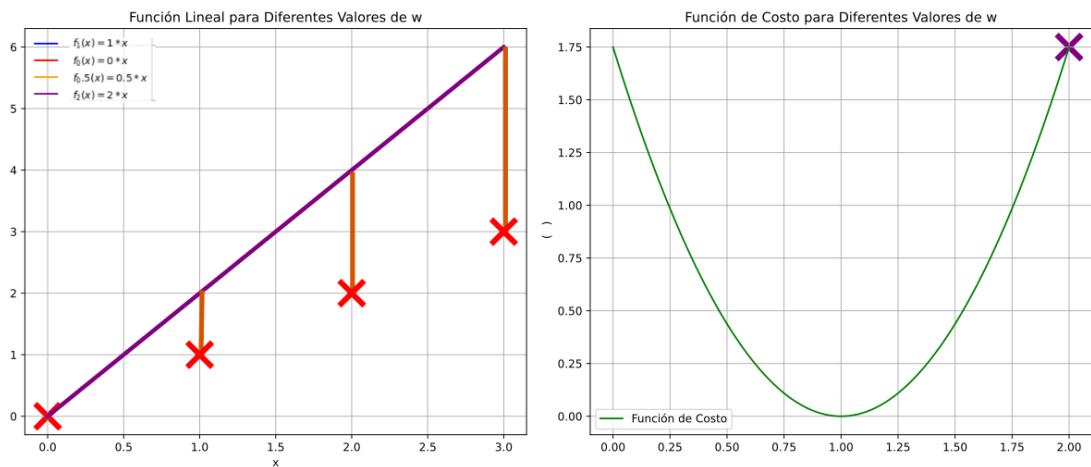


Figura 67.

Que corresponde a la **x** púrpura en el panel derecho.

Observamos entonces que el mínimo de la función de costo corresponde a $w = 1$, y observamos en el panel izquierdo que este modelo, la línea azul, ajusta los datos de entrenamiento perfectamente.

En el caso más general tendremos también el parámetro b en nuestra función de costo y tendríremos que encontrar el valor mínimo de una función 3-dimesional como en la Figura 68, donde, en el panel izquierdo, son presentados datos de entrenamiento aleatoriamente distribuidos en junto con las líneas rectas que podrían ajustarlos. Y la función de costo correspondiente en el panel derecho.

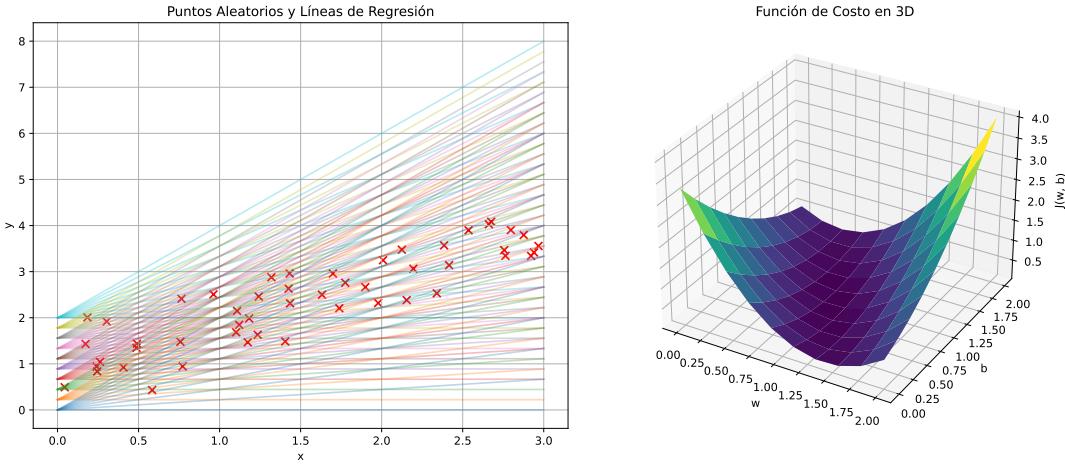


Figura 68.

En general, en el caso con múltiples features la función de costo cuadrático medio está dada por

$$\begin{aligned} J(\mathbf{w}) &= \frac{1}{2m} \sum_{i=1}^m (\mathbf{w}^\top \mathbf{x}^{(i)} - y^{(i)})^2. \\ &= \frac{1}{2m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2. \end{aligned}$$

Donde $J(\mathbf{w})$ representa la función de costo MSE, m es el número de ejemplos en el conjunto de datos de entrenamiento, $\mathbf{w}^\top \mathbf{x}^{(i)}$ es la predicción del modelo para la i -ésima instancia, $y^{(i)}$ es el valor real correspondiente a la i -ésima instancia, y la suma acumula el error cuadrado para todas las instancias del conjunto de datos.

3.4 La Ecuación Normal

Para identificar el valor de \mathbf{w} , que reduce al mínimo el error cuadrático medio (MSE), se dispone de una solución de forma cerrada; es decir, una expresión matemática que proporciona el resultado de manera directa. A esta se le conoce como la ecuación normal.

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \quad (9)$$

Aquí, \mathbf{X} es la matriz de características con filas $\mathbf{x}^{(i)}$ (cada una de ellas incluyendo un término de sesgo), y \mathbf{y} es el vector de valores objetivo. Esta ecuación proporciona la solución de forma cerrada para los pesos \mathbf{w} que minimizan la función de costo.

Esta ecuación viene de minimizar la función de costo MSE para encontrar los parámetros óptimos del modelo. Básicamente, se parte de la función de costo, se calcula su derivada respecto a los pesos, se iguala a cero y se resuelve para los pesos.

3.4.1 Demostración de la Ecuación normal. (Opcional)

Supongamos que tenemos m ejemplos de entrenamiento $(\mathbf{x}^{(i)}, y^{(i)})$ donde

$$\mathbf{x}^{(i)} = (x_1^{(i)}, \dots, x_n^{(i)})^\top$$

el i -ésimo ejemplo de los datos de entrenamiento conteniendo n características. Cambiamos la notación para hacerlo más amigable con la que se usa en el álgebra lineal:

$$\mathbf{x}^{(i)} = (x_1^{(i)}, \dots, x_n^{(i)})^\top \equiv (x_{i1}, \dots, x_{in})^\top.$$

Podemos poner estas m filas en forma de matriz como

$$X = \begin{bmatrix} \mathbf{x}_1^\top \\ \vdots \\ \mathbf{x}_m^\top \end{bmatrix} = \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{bmatrix}$$

y los valores observados como

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

Nota que generalmente hay una característica adicional $\mathbf{x}_{i0}=1$ - la intersección, entonces $\mathbf{x}_i \in \mathbb{R}^{n+1}$

$$X = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_m^\top \end{bmatrix} = \begin{bmatrix} x_{00} & x_{01} & x_{02} & \cdots & x_{0n} \\ x_{10} & x_{11} & x_{12} & \cdots & x_{1n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m0} & x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} \in \mathbb{R}^{m \times (n+1)}$$

Por lo tanto, podemos escribir nuestra ecuación lineal como

$$X\mathbf{w} = \mathbf{y} \tag{10}$$

Si no hay una solución exacta para \mathbf{w} , la solución aproximada se obtiene al minimizar una función de error. En particular para minimizar la función costo dada por

$$J(\mathbf{w}) = \|\mathbf{y} - X\mathbf{w}\|^2.$$

Minimizar J , como aprendimos en cálculo, consiste en encontrar el valor de \mathbf{w} que hace que la derivada de J sea cero. En matrices, la derivada respecto a \mathbf{w} se escribe como

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = 0 \tag{11}$$

donde para efectos prácticos entenderemos el símbolo $\nabla_{\mathbf{w}}$ como una manera resumida de escribir ‘derivada respecto a \mathbf{w} de la matriz’. Para calcular la derivada primero expandamos $J(\mathbf{w})$:

$$\begin{aligned} J(\mathbf{w}) &= (\mathbf{y} - X\mathbf{w})^\top (\mathbf{y} - X\mathbf{w}) \\ &= (\mathbf{y}^\top - \mathbf{w}^\top X^\top)(\mathbf{y} - X\mathbf{w}) \\ &= \mathbf{y}^\top \mathbf{y} - \mathbf{y}^\top X\mathbf{w} - \underbrace{\mathbf{w}^\top X^\top}_{\mathbf{y}^\top X\mathbf{w}} \mathbf{y} + \mathbf{w}^\top X^\top X\mathbf{w} \\ &= \mathbf{y}^\top \mathbf{y} - 2\mathbf{y}^\top X\mathbf{w} + \mathbf{w}^\top X^\top X\mathbf{w} \end{aligned}$$

Usando las reglas de derivación de matrices:

$$\nabla_{\mathbf{w}} \mathbf{y}^\top \mathbf{y} = 0$$

pues no depende explícitamente de \mathbf{w} . Por otro lado

$$\nabla_{\mathbf{w}}(2\mathbf{y}^\top X\mathbf{w}) = 2X^\top \mathbf{y}$$

el último término contiene el factor $X^\top X$ que no es más que un escalar entonces $\mathbf{w}^\top X^\top X\mathbf{w} = X^\top X\mathbf{w}^\top \mathbf{w}$, entonces

$$\begin{aligned} \nabla_{\mathbf{w}}(\mathbf{w}^\top X^\top X\mathbf{w}) &= X^\top X \nabla_{\mathbf{w}}(\mathbf{w}^\top \mathbf{w}) \\ &= 2X^\top X\mathbf{w} \end{aligned}$$

Por tanto

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = -2X^\top \mathbf{y} + 2X^\top X\mathbf{w}$$

Igualando a cero

$$\begin{aligned} -2X^\top \mathbf{y} + 2X^\top X\mathbf{w} &= 0 \\ -X^\top \mathbf{y} + X^\top X\mathbf{w} &= 0 \\ X^\top X\mathbf{w} &= X^\top \mathbf{y} \\ (X^\top X)^{-1}X^\top X\mathbf{w} &= (X^\top X)^{-1}X^\top \mathbf{y} \\ \mathbf{w} &= (X^\top X)^{-1}X^\top \mathbf{y} \end{aligned}$$

Fin!

http://mlwiki.org/index.php/Normal_Equation

4 El gradiente descendente

Hasta el momento, hemos estudiado dos de los elementos comunes en casi todos los proyectos de ML o análisis de datos. El primero es el conjunto de datos X , y el segundo, el modelo $f(\mathbf{w})$, que es función de los parámetros \mathbf{w} . Además, hemos examinado la función de costo $J(\mathbf{w}) = J(\mathbf{w}, f(\mathbf{w}))$, que nos permite entender qué tan bien nuestro modelo $f(\mathbf{w})$ describe los datos X . El modelo se ajusta encontrando los parámetros \mathbf{w} que minimizan la función de costo. En esta sección, discutiremos brevemente la versión más simplificada de la familia de métodos de descenso por gradiente que realizan esta minimización.

4.1 Intuición sobre el gradiente descendente

Recurriremos nuevamente a nuestra versión simplificada de la función de costo en la ecuación 7

$$J(w) = \frac{1}{2m} \sum_{i=1}^m [f_w(x^{(i)}) - y^{(i)}]^2$$

en la que hemos hecho $b=0$. Vimos en que este caso la función de costo es una función convexa como en la figura

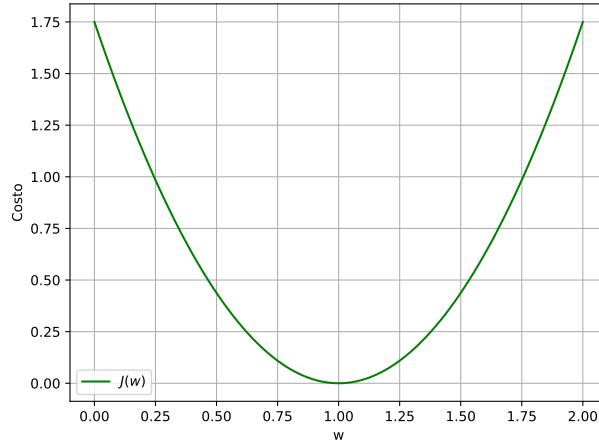


Figura 69.

La idea es bastante simple. Queremos encontrar el mínimo de la función y para eso, partimos de cualquier punto en w_0 e iterativamente ajustamos el parámetro w en la dirección en que la función de costo decrece.

El Gradiende Descendente ajusta iterativamente el parámetros w , intentando encontrar el valor más bajo de la función de costo $J(w)$. El término "gradiente" se refiere al gradiente de la función de costo, el cual indica la dirección del ascenso más empinado en cualquier punto de la función. Para minimizar J , es necesario moverse en la dirección opuesta a este gradiente.

4.2 Algoritmo

En dos dimensiones, el gradiente de $J(w, b)$ puede describirse mediante las derivadas parciales de J con respecto a cada parámetro:

$\frac{\partial J}{\partial w}$ nos indica cómo cambia J con cambios en w , manteniendo b constante.

$\frac{\partial J}{\partial b}$ nos indica cómo cambia J con cambios en b , manteniendo w constante.

El algoritmo de gradiente descendente actualiza w y b restando una fracción α del gradiente en el punto actual. A α se le denomina la *tasa de aprendizaje*, y no es más que un pequeño escalar positivo que determina el tamaño del paso. Así:

1. Inicializar w y b a cualquier valor. Estos podrían ser cero, valores aleatorios, o resultados de una ejecución anterior.

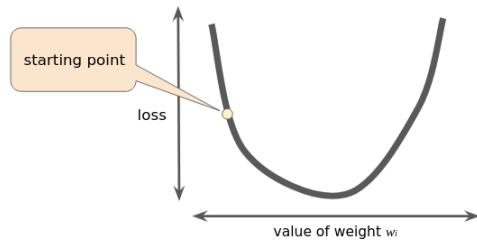


Figure 3. A starting point for gradient descent.

2. Calcular el gradiente actual:

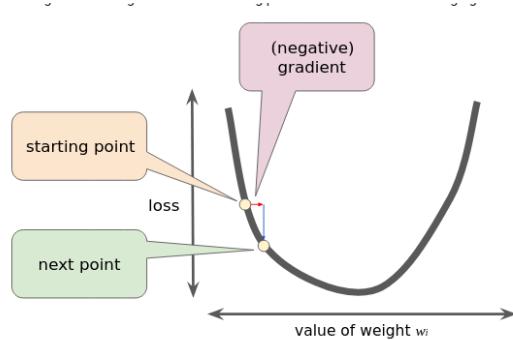
$$\text{grad}_w = \frac{\partial J}{\partial w}(w, b)$$

$$\text{grad}_b = \frac{\partial J}{\partial b}(w, b)$$

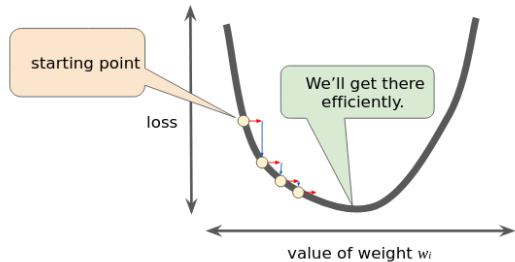
3. Actualizar w y b :

$$w := w - \alpha \cdot \text{grad}_w$$

$$b := b - \alpha \cdot \text{grad}_b$$



4. Repetir hasta la convergencia (o hasta alcanzar un número máximo de iteraciones)



4.3 Para practicar

Felicitades por llegar al final de nuestra primera lección te invito a que practiques los conceptos apredidos en el el liguiente link <https://developers.google.com/machine-learning/crash-course/fitter/graph?authuser=1>

6 Semana 2. Regresión logística

1 Ajustes no lineales

En la figura 70, se presentan datos ficticios sobre el crecimiento de una población de bacterias y el aumento a lo largo de los años del PIB de China. En ninguno de estos dos conjuntos de datos parece razonable realizar un ajuste lineal.

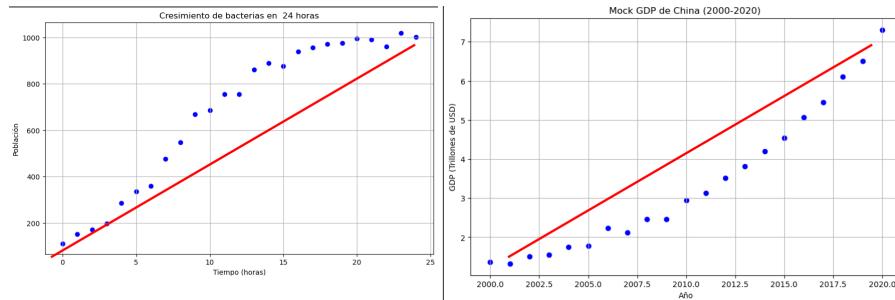


Figura 70. Datos ficticios que ilustran el crecimiento de la población de bacterias y el crecimiento del PIB de China a lo largo de los años. Los patrones observados sugieren que un ajuste lineal no es apropiado para estos conjuntos de datos

En particular, el crecimiento de las bacterias es mejor modelado por una función ‘sigmoide’, y el PIB de China puede ser mejor representado por una función exponencial, como se muestra en la figura 71

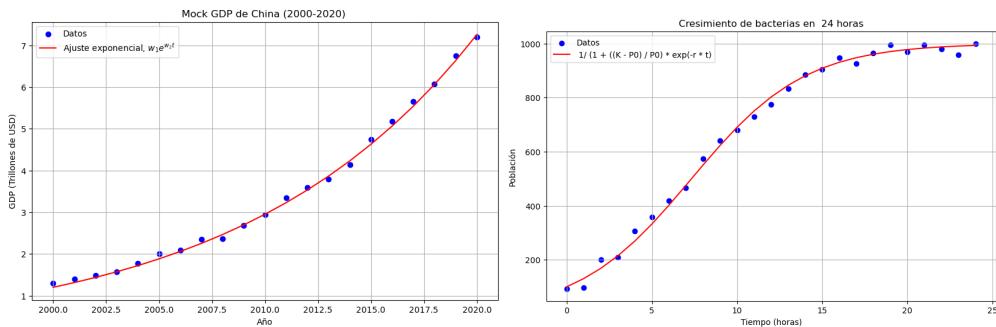


Figura 71. Representación gráfica de ajustes no lineales. Se utiliza una función sigmoidal para modelar el crecimiento de la población de bacterias y una función exponencial para el crecimiento del PIB de China

De hecho, existen muchos tipos de regresiones que se pueden usar dependiendo de las características de los datos como se muestra en la figura 72, donde se presentan varios tipos de regresiones polinómicas.

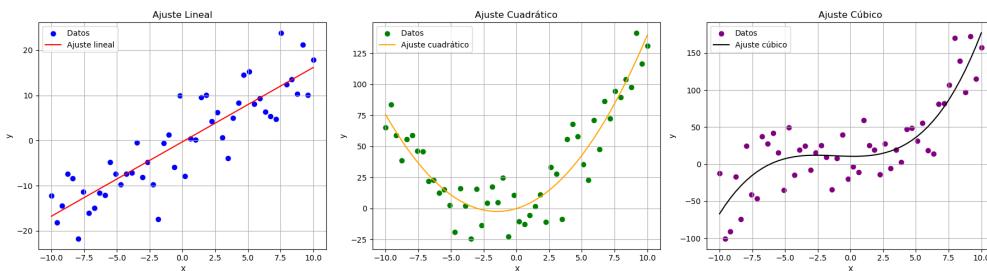


Figura 72.

Una regresión polinómica ajusta una curva a los datos. Por ejemplo, un polinomio de grado 3 se describe con la ecuación

$$\hat{y} = b + w_1x + w_2x^2 + w_3x^3$$

donde los w_i son los pesos o parámetros estimados para ajustar nuestros datos. Sin embargo, las regresiones polinomiales pueden transformarse en regresiones lineales si definimos nuevas variables, como $x_1 = x$, $x_2 = x^2$, $x_3 = x^3$ entonces

$$\begin{aligned}\hat{y} &= b + w_1x_1 + w_2x_2 + w_3x_3 \\ &= b + \mathbf{w}^\top \mathbf{x}\end{aligned}$$

de este modo, el modelo es lineal en los pesos w_i , razón por la cual la regresión polinómica se considera un caso especial de la regresión lineal múltiple. Por tanto, podemos utilizar las mismas técnicas estudiadas en la regresión lineal como la minimización del error cuadrático medio. Lo importante es que \hat{y} sea lineal en los pesos, no en las características x_i .

Para un modelo ser considerado no lineal debemos observar una relación no lineal entre \hat{y} y los pesos w_i como en las ecuaciones

$$y = w_0 + w^2x$$

y en

$$\hat{y} = w_0 e^{w_1 x}$$

Estos ejemplos muestran cómo las ecuaciones pueden describir relaciones complejas y no lineales entre la variable dependiente y los pesos. En estos casos, no podemos usar técnicas como la minimización del error cuadrático medio y la estimación de los parámetros no es tan fácil.

1.1 El problema del sobreajuste (Overfitting)

Supongamos que observamos una variable de entrada de valor real x y deseamos usar esta observación para predecir el valor de una variable objetivo y . Los datos para este ejemplo los generaremos a partir de la función $\cos(2\pi x)$ agregando ruido gaussiano. De esta manera, queremos ajustar los datos de la figura que constituyen nuestro conjunto de entrenamiento que consta de m observaciones de x , denotadas por $x \equiv (x_1, \dots, x_N)^T$, junto con las observaciones correspondientes de los valores de y , que se denotan por $y \equiv (y_1, \dots, y_m)^T$

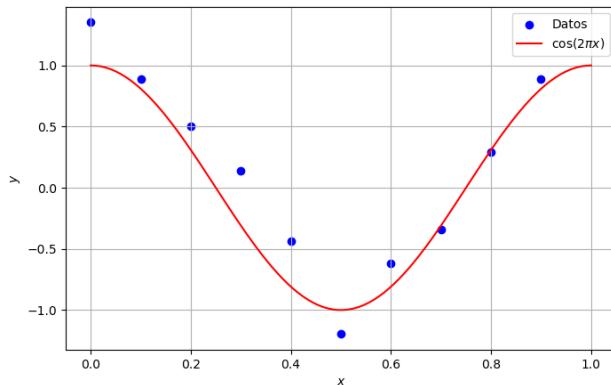
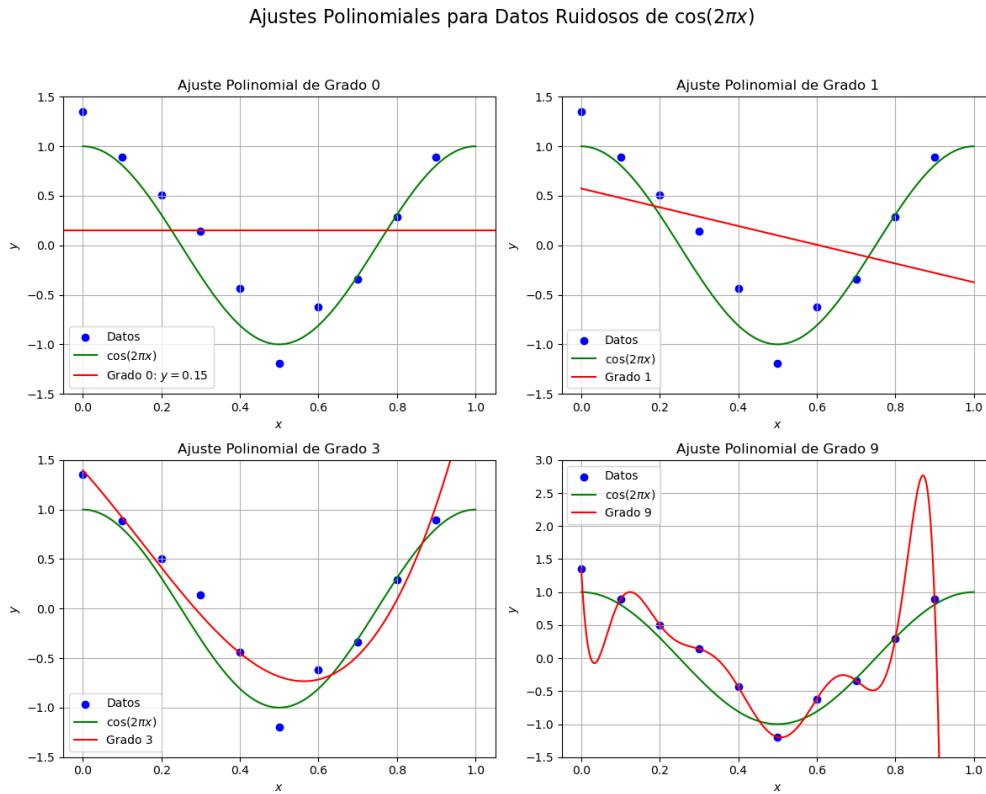
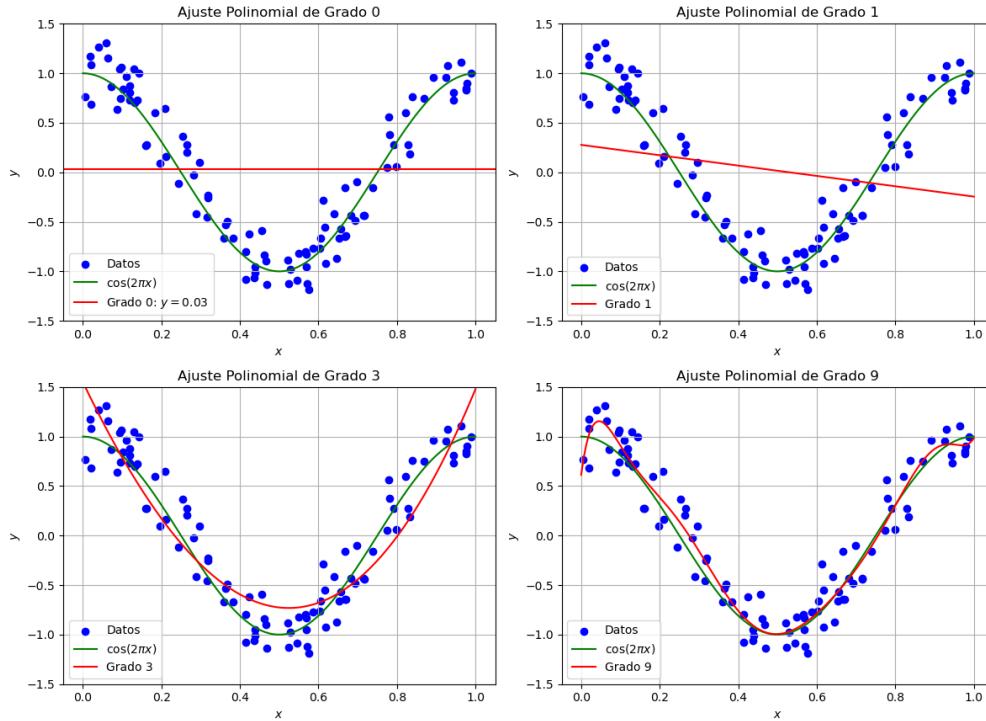


Figura 73. Ver código en Apéndice ?

**Figura 74.**

Ajustes Polinomiales para Datos Ruidosos de $\cos(2\pi x)$ con $m = 100$ ejemplos

**Figura 75.** Aumentar el número de datos reduce el sobreajuste

Otra alternativa para corregir el sobreajuste es usar la regularización como

$$J(\mathbf{w}) = \frac{1}{2m} \sum_{i=1}^m (\mathbf{w}^\top \mathbf{x}^{(i)} - y^{(i)})^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

Puedes ampliar este tema en la referencia [?].

2 Regresión Logística

La regresión logística es un algoritmo de clasificación para variables categóricas. A diferencia de la regresión lineal, que se utiliza para predecir resultados continuos (por ejemplo, precios de bienes raíces, presión arterial de pacientes o eficiencia de combustible de vehículos), la regresión logística se enfoca en resultados binarios como 'Sí/No', 'VERDADERO/FALSO', "exito/fracaso" o 'émbarazada/no embarazada', los cuales se codifican típicamente como 0 o 1. La regresión logística es un excelente modelo para los casos en los que deseamos clasificar datos categóricos binarios y predecir no solo la clase a la que pertenece un caso, sino también la probabilidad asociada a esa clasificación.

Algunos ejemplo en los que se recomienda aplicar la regresión logística son

- Predicción de la presencia o ausencia de una enfermedad (ej. cáncer, diabetes) basada en variables clínicas y biométricas de los pacientes.
- Determinar si un solicitante debe recibir un crédito bancario o no, basándose en su historial crediticio, ingresos, y otras variables financieras.
- Predecir si un cliente potencial realizará una compra o se suscribirá a un servicio después de una campaña de marketing, utilizando datos como historial de compras, actividad en la web y demografía.
- Clasificar a los candidatos como aptos o no aptos para un puesto basado en sus respuestas en tests psicométricos y antecedentes laborales.
- Identificar transacciones fraudulentas en servicios bancarios y tarjetas de crédito, analizando patrones de comportamiento y transacciones anómalas.
- Evaluar si asegurar a un individuo o propiedad es un riesgo alto o bajo, basado en su historial de reclamaciones y otros factores de riesgo.
- Predecir si un cliente dejará de usar un servicio, como puede ser la baja de un servicio de telecomunicaciones o una suscripción a un software, utilizando datos de uso y satisfacción del cliente.
- Pronosticar si un estudiante pasará o fallará en un curso o examen, basándose en su desempeño previo, asistencia y otras métricas educativas.
- Prever si un votante apoyará a un candidato o medida política en una elección, usando datos demográficos y resultados de encuestas.
- Clasificar productos en categorías de pasar/fallar al final de una línea de producción, basado en mediciones de control de calidad.

El objetivo de la regresión logística es construir un modelo que pueda predecir la categoría de cada instancia, así como la probabilidad de que dicha instancia pertenezca a una categoría específica. Para comenzar, definamos estructuralmente el problema. Consideremos X como nuestro conjunto de datos, que consiste en m características o dimensiones y n registros, donde $X \in \mathbb{R}^{m \times n}$.

La variable y indica la clase que deseamos predecir, pudiendo ser 0 o 1. Idealmente, un modelo de regresión logística, \hat{y} , podría predecir que la clase de una instancia es 1 basándose en sus características x , es decir,

$$P(y=1|x) = 1.$$

Por tanto, la probabilidad de que la instancia pertenezca a la clase 0 es simplemente el complemento de que pertenezca a la clase 1. Es decir,

$$P(y=0|x) = 1 - P(y=1|x). \quad (12)$$

Observemos el caso en que nuestro conjunto de datos está dado por datos sobre los clientes de una compañía de telecomunicaciones. Un ejemplo de este tipo de dataset es el llamado churn dataset que puedes descargar del IBM store

```
wget -O ChurnData.csv
https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%203/data/ChurnData.csv
```

En este conjunto de datos esta hipotética compañía de telecomunicaciones tiene información sobre:

- Clientes que han dejado el servicio en el último mes, una variable designada como "Churn", que indica si un cliente va a permanecer con los servicios de la compañía (0) o si la dejó (1).
- La gama de servicios a los que cada cliente está suscrito, incluyendo servicios telefónicos, opciones de múltiples líneas, acceso a internet, seguridad en línea, copia de seguridad en línea, protección de dispositivos, soporte técnico, así como servicios de streaming para televisión y películas.
- Detalles de la cuenta de cada cliente, que incluyen la duración del servicio, tipo de contrato, métodos de pago, si utilizan facturación sin papel, y los cargos mensuales y totales.
- Información demográfica sobre los clientes, como género, rango de edad, y si tienen parejas y dependientes.

	tenure	age	address	income	ed	employ	equip	callcard	wireless	longmon	...	pager	internet	callwait	confer	ebill	loglong	logtoll	lninc	custcat	churn
0	11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0	4.40	...	1.0	0.0	1.0	1.0	0.0	1.482	3.033	4.913	4.0	1.0
1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0	9.45	...	0.0	0.0	0.0	0.0	0.0	2.246	3.240	3.497	1.0	1.0
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0	6.30	...	0.0	0.0	0.0	1.0	0.0	1.841	3.240	3.401	3.0	0.0
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0	6.05	...	1.0	1.0	1.0	1.0	1.0	1.800	3.807	4.331	4.0	0.0
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	7.10	...	0.0	0.0	1.0	1.0	0.0	1.960	3.091	4.382	3.0	0.0
5	68.0	52.0	17.0	120.0	1.0	24.0	0.0	1.0	0.0	20.70	...	0.0	0.0	0.0	0.0	0.0	3.030	3.240	4.787	1.0	0.0
6	42.0	40.0	7.0	37.0	2.0	8.0	1.0	1.0	1.0	8.25	...	0.0	1.0	1.0	1.0	1.0	2.110	3.157	3.611	4.0	0.0
7	9.0	21.0	1.0	17.0	2.0	2.0	0.0	0.0	0.0	2.90	...	0.0	0.0	0.0	0.0	0.0	1.065	3.240	2.833	1.0	0.0
8	35.0	50.0	26.0	140.0	2.0	21.0	0.0	1.0	0.0	6.50	...	0.0	0.0	1.0	1.0	0.0	1.872	3.314	4.942	3.0	0.0
9	49.0	51.0	27.0	63.0	4.0	19.0	0.0	1.0	0.0	12.85	...	0.0	1.0	1.0	0.0	1.0	2.553	3.248	4.143	2.0	0.0

Figura 76.

Nuestro objetivo es calcular la probabilidad de que un cliente permanezca con los servicios de la compañía, por lo tanto, 'churn' será nuestra variable dependiente y . Seleccionamos nuestras variables independientes X como

```
cdf = df[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip', 'callcard', 'wireless', 'churn']]
churn_df['churn'] = churn_df['churn'].astype(int)
```

```
churn_df.head()
```

Y Obtenemos el dataframe

	X										y
	tenure	age	address	income	ed	employ	equip	callcard	wireless	churn	
0	11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0	1.0	
1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0	1.0	
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0	0.0	
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0	0.0	
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	0.0	

Queremos construir un modelo \hat{y} que nos indique la probabilidad de obtener $y=1$ dadas las características X , es decir, $\hat{y}=P(y=1|x)$

$$\hat{y}=P(y=1|x).$$

Para tener una mejor idea de por qué usar la regresión logística, supongamos que queremos predecir \hat{y} a través de una regresión lineal basada en la edad de los clientes. Podemos representar nuestros datos con un gráfico de dispersión::

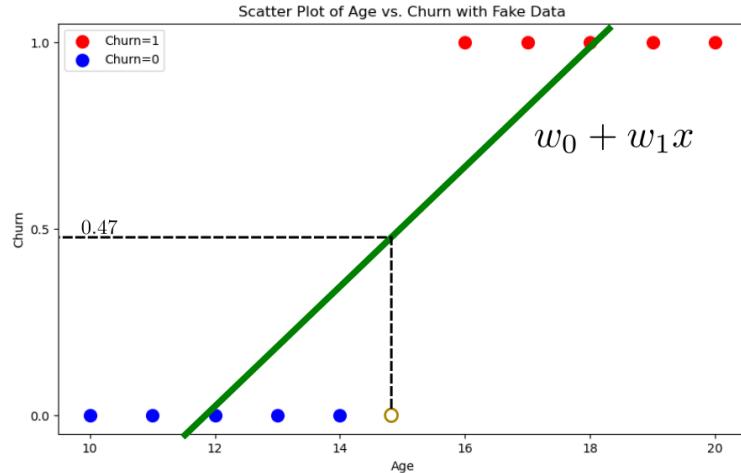


Figura 77.

En este caso, nuestra línea recta está dada por el modelo:

$$\hat{y}=\mathbf{w}^\top \mathbf{X} = w_0 + w_1x$$

Supongamos que el valor predicho para un nuevo cliente de 15 años es $\hat{y}=0.47$, como en la figura 77. En este caso, 0.47 no nos indica con certeza si nuestro cliente continuará o no con los servicios. Entonces, inventamos una regla que nos permita decidir a qué clase pertenece el cliente. Por ejemplo:

$$\hat{y} = \begin{cases} 0 & \text{si } \mathbf{w}^\top \mathbf{X} < 0.5 \\ 1 & \text{si } \mathbf{w}^\top \mathbf{X} \geq 0.5 \end{cases} \quad (13)$$

Si el valor de $\mathbf{w}^\top \mathbf{X}$ es menor que 0.5, la clase es 0; de lo contrario, es 1. Dado que el valor del cliente es menor que 0.5, podemos afirmar que pertenece a la clase 0. La función descrita anteriormente actúa como un umbral y, si se grafica, se asemeja a la línea azul en la figura 78..

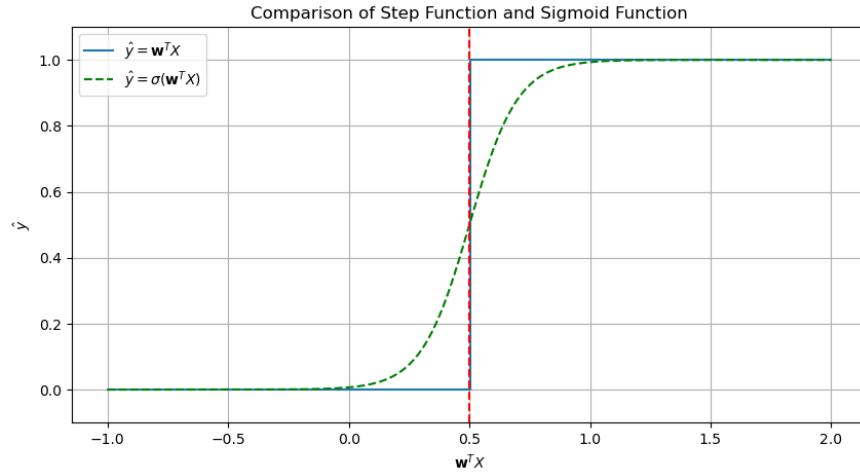


Figura 78. Representación gráfica de la función sigmoide $\sigma(x) = 1 / (1 + e^{-x})$. Esta curva muestra cómo la función transforma el input lineal $\mathbf{w}^\top \mathbf{X}$ en una probabilidad entre 0 y 1, ideal para la clasificación binaria en regresión logística.

La línea verde representa la aplicación de una función especial conocida como la función *sigmoide*, por su forma de ‘S’, denotada como $\sigma(x)$ y definida por:

$$\sigma(x) = \frac{1}{1 + e^{-x}},$$

verificamos una propiedad importante de la función sigmoide sumando $\sigma(x)$ y $\sigma(-x)$:

$$\begin{aligned}\sigma(x) + \sigma(-x) &= \frac{1}{1 + e^{-x}} + \frac{1}{1 + e^x} \\ &= \frac{1}{1 + e^{-x}} \cdot \frac{e^x}{e^x} + \frac{1}{1 + e^x} \\ &= \frac{e^x}{e^x + 1} + \frac{1}{1 + e^x} \\ &= \frac{e^x + 1}{e^x + 1} \\ &= 1\end{aligned}$$

Así, la función sigmoide $\sigma(x)$ también cumple con la propiedad de probabilidad en la ecuación 12 $\sigma(x) = 1 - \sigma(-x)$, lo que nos permite interpretar $\sigma(\mathbf{w}^\top \mathbf{X})$ como la probabilidad $P(y = 1 | x)$ de que una instancia pertenezca a la clase 1, dadas las características X . Esto se expresa como:

$$\sigma(\mathbf{w}^\top \mathbf{X}) = P(y = 1 | x),$$

mientras que

$$1 - \sigma(\mathbf{w}^\top \mathbf{X}) = P(y = 0 | x).$$

Donde \mathbf{w}^\top debe ser encontrado en el proceso de entrenamiento del modelo de regresión $\hat{y} = \sigma(\mathbf{w}^\top \mathbf{X})$ siguiendo el mismo algoritmo que estudiamos para la regresión lineal:

1. Inicializa en un valor cualquiera \mathbf{w} .

2. Calcula $\hat{y} = \sigma(\mathbf{w}^\top X)$.
3. Compara la salida \hat{y} el valor real etiquetado y .
4. Calcula el error para todos los valores etiquetados y y suma los errores. El error total es la función de costo de tu modelo y es calculada por la función de costo.
5. Actualiza w para minimizar el costo $J(w)$ y vuelve al paso 2.

2.1 La función de costo

El proceso para minimizar el costo en modelos de regresión logística comúnmente emplea el método del gradiente descendente. La función de costo utilizada es la siguiente:

$$J(w) = -\frac{1}{m} \sum_{i=1}^m [y^i \log(\hat{y}^i) + (1 - y^i) \log(1 - \hat{y}^i)]$$

Esta función de costo es adecuada porque penaliza significativamente las predicciones incorrectas. Como lo muestra la figura, la expresión $-y^i \log(\hat{y}^i)$ es clave; el logaritmo negativo $-\log(t)$ se incrementa considerablemente cuando t se aproxima a 0, lo que implica un costo alto si el modelo estima una probabilidad cercana a 0 para una instancia positiva. Por otro lado, $(1 - y^i) \log(1 - \hat{y}^i)$ actúa para las instancias negativas; $-\log(t)$ es cercano a 0 cuando t es cercano a 1, resultando en un costo casi nulo si la probabilidad estimada es cercana a 0 para una instancia negativa o cercana a 1 para una positiva, que es exactamente lo que deseamos.

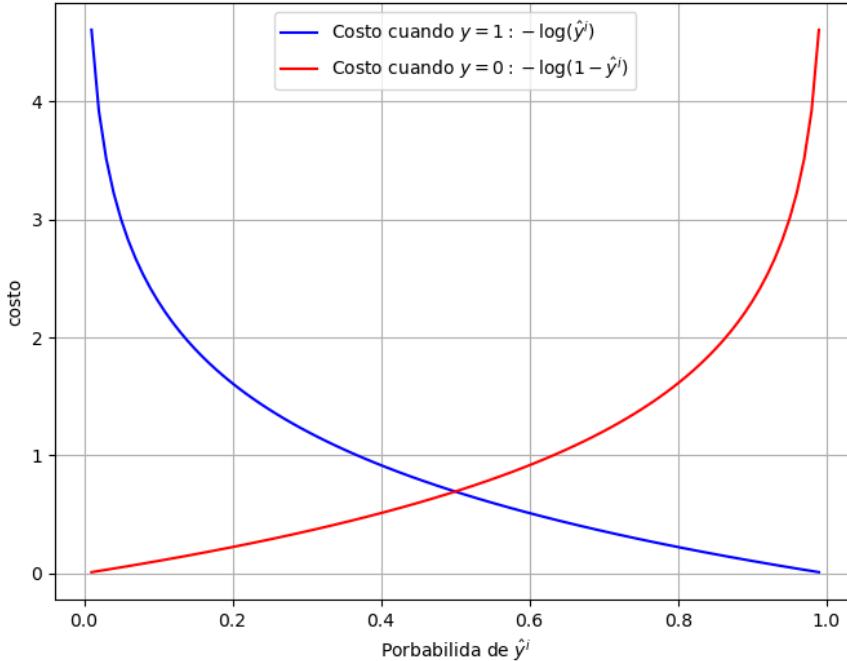


Figura 79.

La eficacia de esta función radica en que a mayor discrepancia entre la predicción \hat{y}^i y el valor real y^i , mayor es el logaritmo del error, incrementando así el costo. Esto incentiva al modelo a ajustar los parámetros w de manera que se minimice el error en las predicciones.

Para ampliar este tema y otros te invito a revisar el excelente curso en la referencia [?]

7 Semana 3. Deep Learning Parte 1 (Versión preliminar)

1 Redes Neuronales

1.1 Gráficos computacionales de funciones

Una función, puede entenderse como la representación abstracta de una máquina que recibe un valor de entrada y lo transforma en un valor de salida según una regla determinada. Por ejemplo, la función

$$f(x) = x^2,$$

devuelve el cuadrado de todo lo que recibe:

$$\begin{aligned} f(2) &= 2^2, \\ f(5) &= 5^2, \end{aligned}$$

o incluso

$$f(\heartsuit) = \heartsuit^2.$$

Sin embargo, $f(x) = x^2$ no es la única manera de representar esta función. También podemos pensarla como un gráfico computacional. Entonces, la función $f(x) = x^2$ puede entenderse a través del gráfico

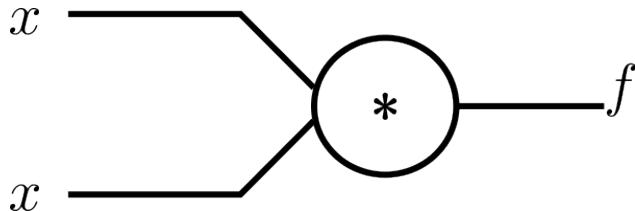


Figura 80.

En esta representación, las entradas son ‘links’ y las operaciones son nodos.

Asignemos algunos valores a nuestro gráfico computacional. Por ejemplo, $x = 2$

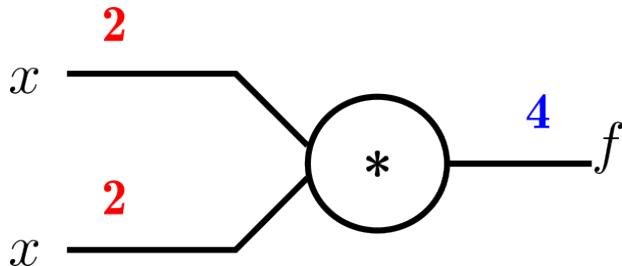


Figura 81.

Los valores de $x=2$ se propagan a través de los enlaces y son transformados en el nodo ‘*’ , de donde sale el valor de salida 4.

Reto: Qué función representa el gráfico de la figura?

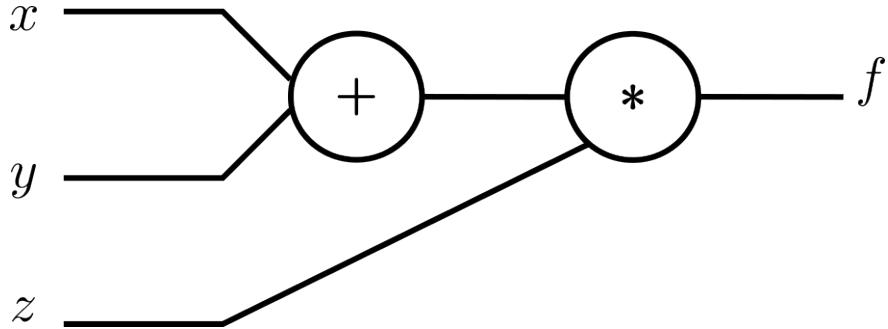
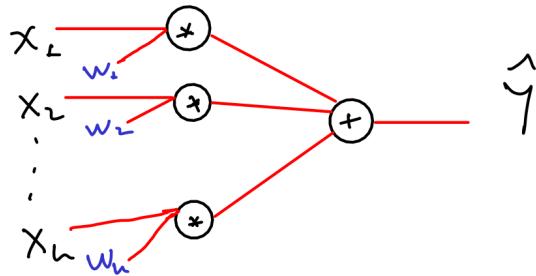


Figura 82.

1.1.1 Gráfico computacional del modelo de regresión lineal

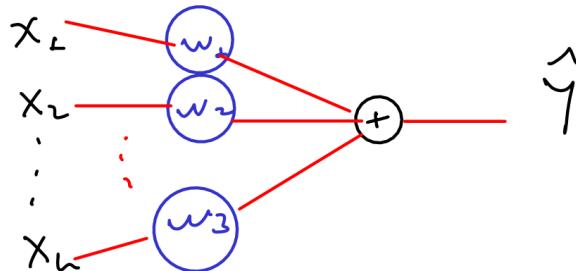
Cómo podríamos representar la función de regresión lineal $\hat{y} = w_0 + w_1x_1 + \dots + w_nx_n$? Una opción sería la figura



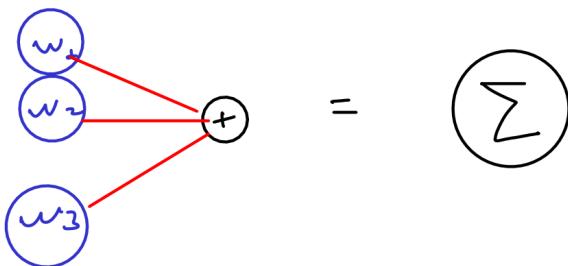
Podemos hacer un diagrama más simple si hacemos la equivalencia

$$\begin{array}{c} \text{---} \\ | \\ \text{---} \end{array} \stackrel{=} {\text{---}} \quad \text{---} \quad \text{---}$$

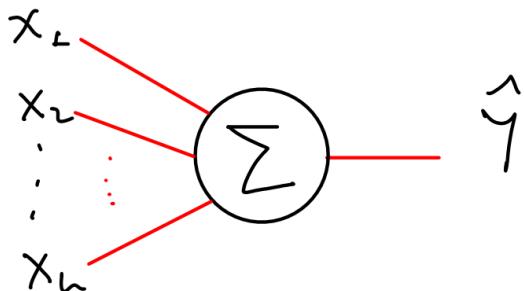
Entonces nuestro diagrama queda reducido a



Los pesos w_i son representados por los links. Podemos simplificar aún más juntando links y nodo en un solo símbolo

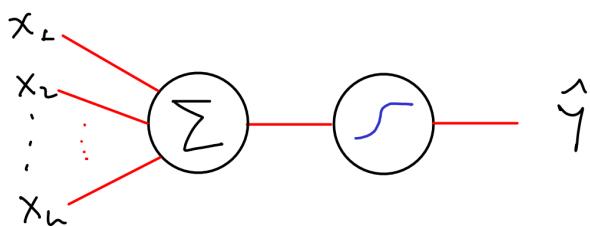


Obteniendo el diagrama



1.1.2 Gráfico computacional del modelo de regresión logística

El modelo de regresión logística consiste en comprimir la suma ponderada del modelo de regresión lineal a través de la función σ . Entonces una representación adecuada sería el diagrama de regresión lineal, seguido de un diagrama de compresión logística. El primer candidato sería el diagrama



Si juntamos los dos nodos en 1 llegamos al diagrama

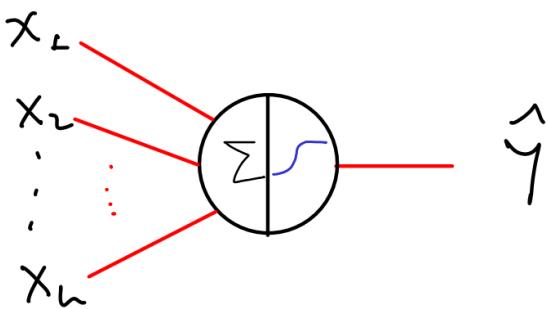


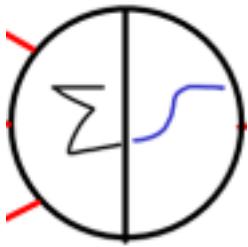
Figura 83.

Las propiedades computacionales de este tipo de diagramas han sido resumidas en modelos matemáticos muy simples conocidos como *redes neuronales artificiales*. En general, este tipo de modelos se basa en las propiedades de una sola neurona realizando combinaciones lineales de las salidas de otras neuronas que es después transformada usando la función no lineal (no necesariamente σ). En general estos modelos pueden ser descritos matemáticamente como

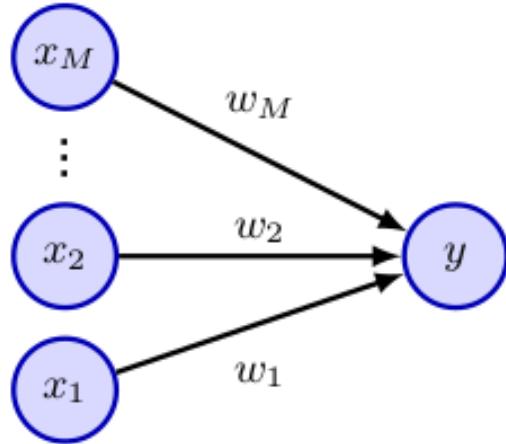
$$y = f\left(\sum_{i=1}^m w_i x_i\right) \quad (14)$$

donde x_1, \dots, x_m representan las m entradas correspondientes a las actividades de otras neuronas, que envían conexiones a esta neurona. A la cantidad $a = \sum_{i=1}^m w_i x_i$, se le llama la *pre-activación*, mientras que a la función no lineal $f(\cdot)$ se le llama la *función de activación*, y a la salida y , se le llama la *activación*.

La simple formulación matemática presentada en la ecuación 14 es la versión general del diagrama en la figura 83 y es la base de modelo de redes neuronales. Es usual que en la literatura se omita el símbolo que inventamos



y una neurona se representa simplemente por el diagrama



donde cada link tiene el significado implícito de la transformación representada por el diagrama:

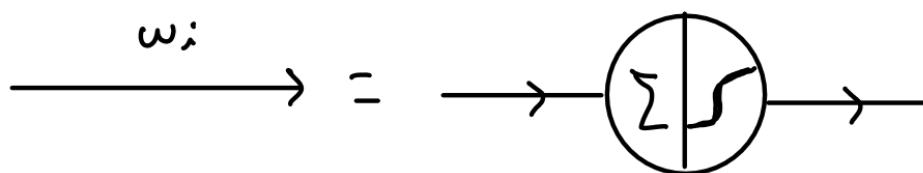
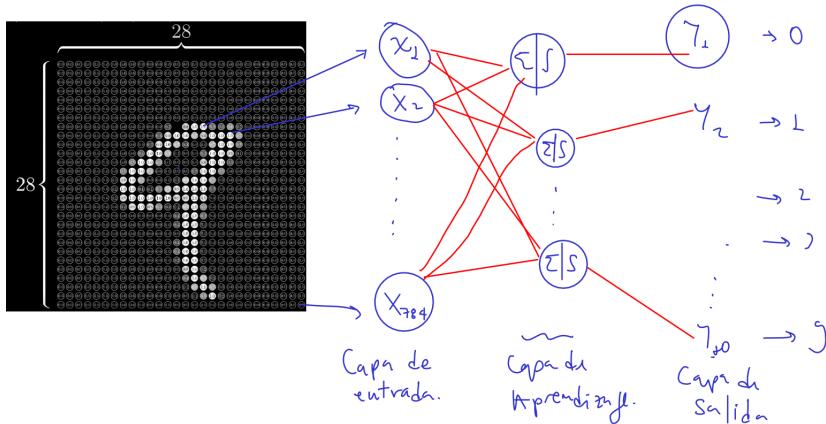


Figura 84.

2 Redes neuronales de una sola capa: El perceptrón

El diagrama que acabamos de construir para el modelo de regresión logísitca en la figura 83 es conocido como la ***linear threshold unit*** (LTU). Por tanto, una LTU no es más que la representación de una función que realiza clasificaciones lineales binarias. Calcula una combinación lineal de las variables de entrada, si el resultado excede un cierto valor crítico entrega un 1 y de lo contrario un 0. Igual que en la regresión logísitca, entrenar una LTU consiste en encontrar valores para w_i usando los algoritmos que estudiamos. Lo novedoso, es que las LTU son las piezas de lego ('neuronas') de uno de las arquitecturas más simples de Redes Neuronales Artificiales (ANN) llamado el ***perceptrón***. El perceptrón fué propuesto por en 1957 por Frank Rosenblatt [?] y es un arreglo de LTU, con cada TLU conectada a todos los valores de entrada. En su forma original, el perceptrón fué propuesto como un sistema con una sola capa de neuronas en las que los parámetros son ajustados a partir de los datos.

Para hacer menos abstracta la implementación del perceptrón, supongamos que lo usaremos para hacer procesamiento de imágenes. En particular queremos reconocer números escritos a mano a partir de imágenes de 28×28 pixeles como se muestra en la figura:



Cada píxel puede ser utilizado como una característica de entrada x_i en la red neuronal. Dado que hay un total de 784 píxeles (28 multiplicado por 28), la capa de entrada constará de 784 nodos. Cada nodo en esta capa de entrada recibirá el brillo (generalmente normalizado) de un píxel de la imagen. Esta configuración permite que la red neuronal aprenda del arreglo espacial de los brillos de los píxeles para reconocer los dígitos, ya que cada nodo de entrada corresponde a un píxel. Mientras que en la capa de salida tendremos 10 nodos una para cada dígito del 0 al 9 (cada nodo representando una clase de dígito). Aplicando la notación de la figura 84 el perceptrón es simplemente representado por el diagrama de la figura 38

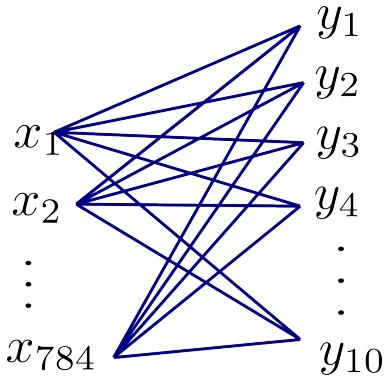


Figura 85.

A pesar de que la capa de aprendizaje del perceptrón contiene varias capas de procesamiento, esta es la única capa en la que se ajustan los parámetros del modelo. Por esta razón el perceptrón es considerado una red neuronal de una sola capa. En 1969 Minsky and Papert [?] demostraron las limitaciones de las redes neuronales de una sola capa debido a esto, se creyó por mucho tiempo que estas mismas limitaciones se

extendían a las redes neuronales con múltiples capas de aprendizaje en parte porque no existían en la época algoritmos efectivos para entrenarlas.

3 Entrenamiento y Backpropagation

Las dificultades para entrenar redes neuronales de múltiples capas con varios pesos comenzaron a ser resueltas en la década de los 80 mediante el uso de técnicas derivadas del cálculo diferencial y métodos de optimización basados en el gradiente. Para entrenar una red de estas características, los parámetros son inicializados aleatoriamente y después son actualizados usando métodos de optimización basados en el gradiente. Para esto, es necesario evaluar las derivadas de la función de error en un proceso conocido como *propagación de error* o *retropropagación* (Backpropagation).

Consideremos la red que representa la función

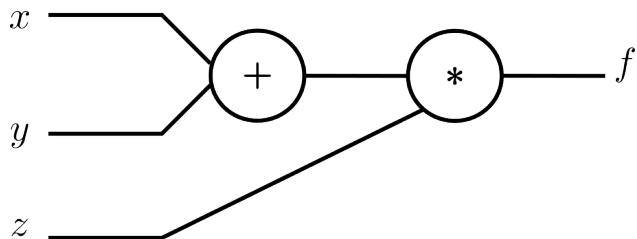


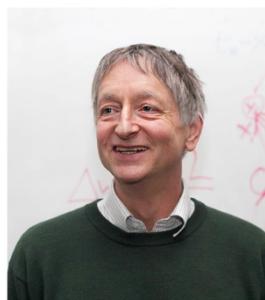
Figura 86.

<https://www.youtube.com/watch?v=Qf1XxNfMCKo>



<https://www.psychologicalscience.org/observer/david-rumelhart>

David Rumelhart



<https://www.frontiersofknowledgeawards-fbva.es/galardonados/geoffrey-hinton-2/>

Geoffrey Hinton

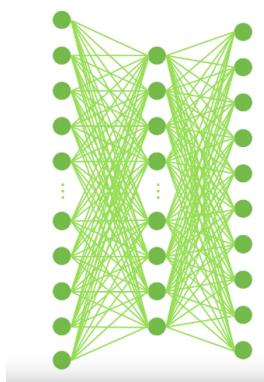


<https://www.ccs.neu.edu/home/rjw/>

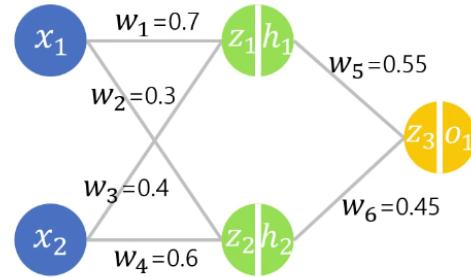
Ronald J. Williams

Finding the best weights requires hundreds to thousands of iterations.

- Number of weights:
 $784 \times 100 + 100 \times 10 = 79,400$
- Number of biases:
 $100 + 10 = 110$
- Loss calculation for one image :
79,400 computation (only weights)
- To update one parameter:
 $4,764,000,000 (= 79,400 \times 60,000)$
- We should update all parameters
 $4,767,000,000 \times 79,511 (= 79,400 + 110 + 1) = 378,790,404,000,000$
- $378,790,404,000,000 / 850,000,000 = 445,635 \text{ sec} = 123 \text{ h}$



Neural network training using backpropagation consists of three main steps.



1st step: Feedforward



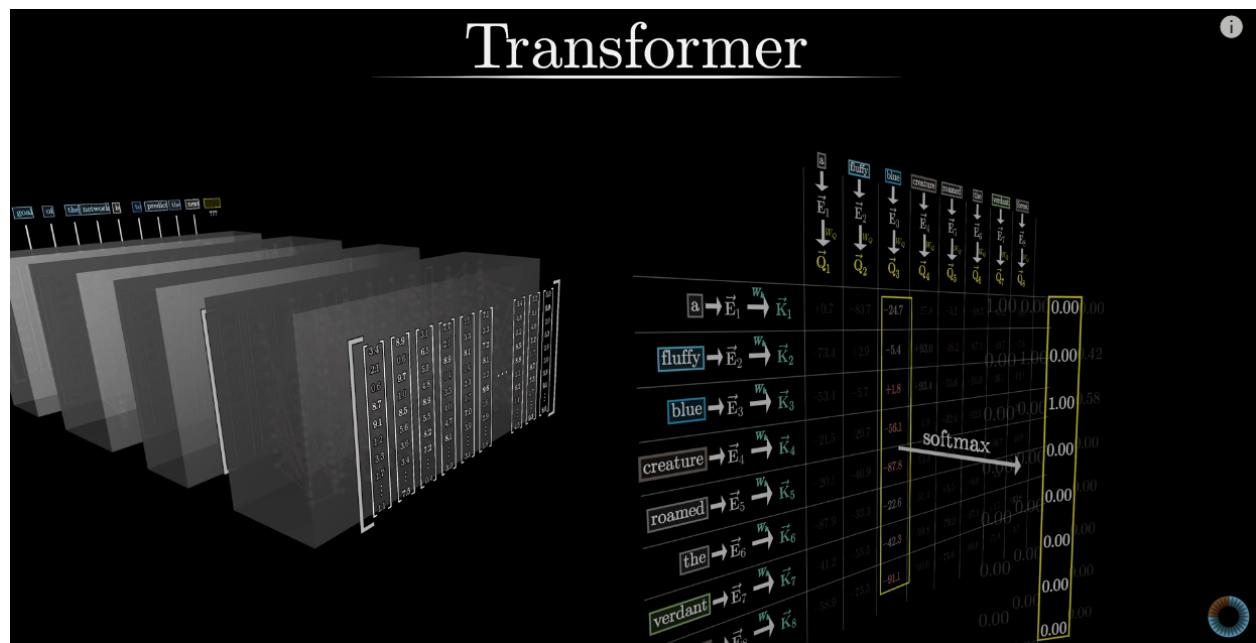
2nd step: Loss calculation

3rd step: Backpropagation

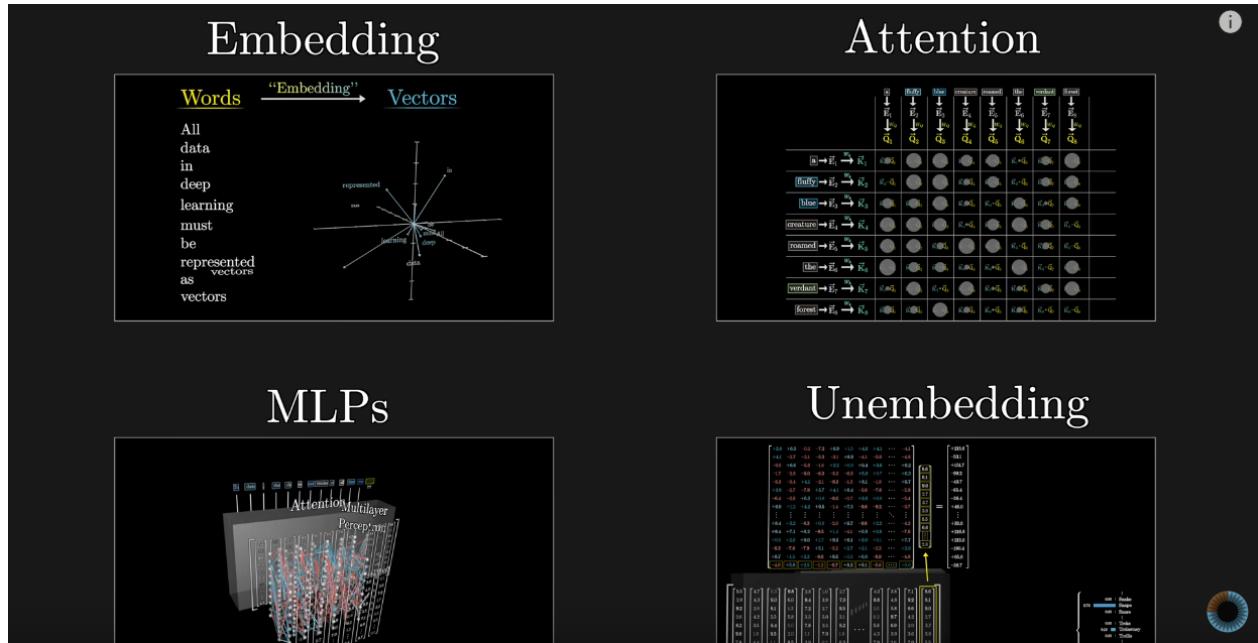


Videos recomendados: <https://www.youtube.com/watch?v=aircAruvnKk>
<https://www.youtube.com/watch?v=Ilg3gGewQ5U>

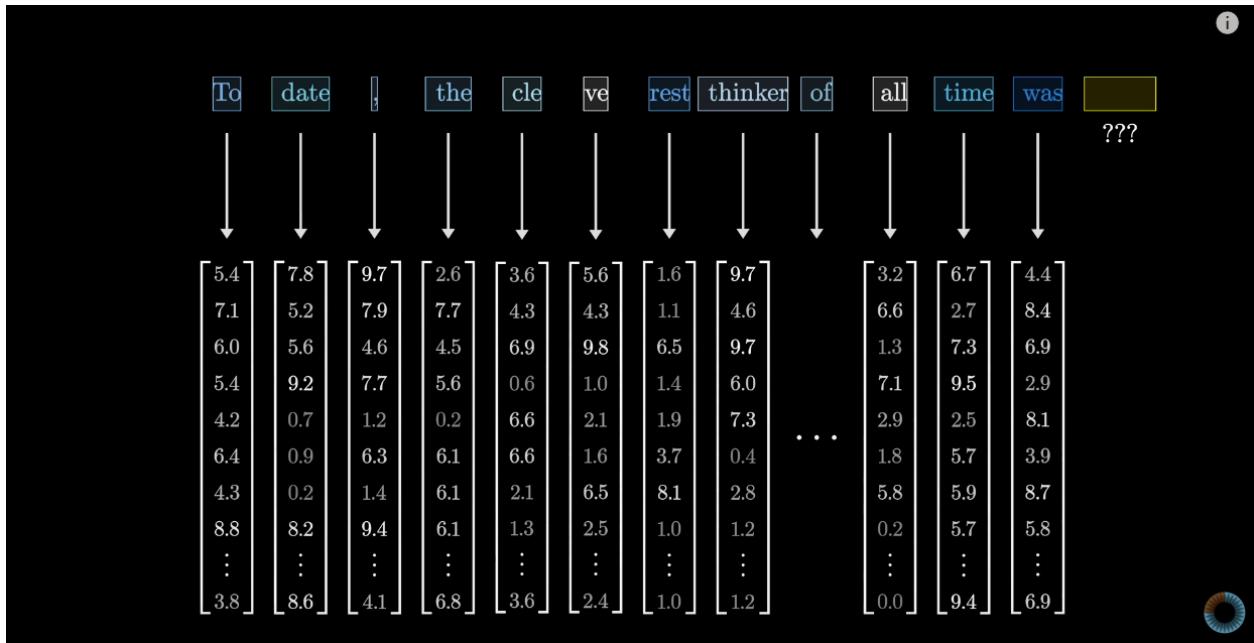
4 Transformers

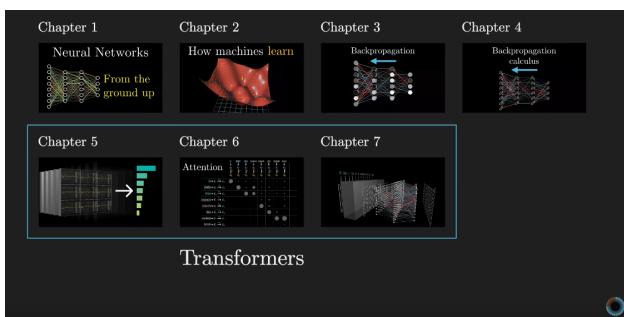
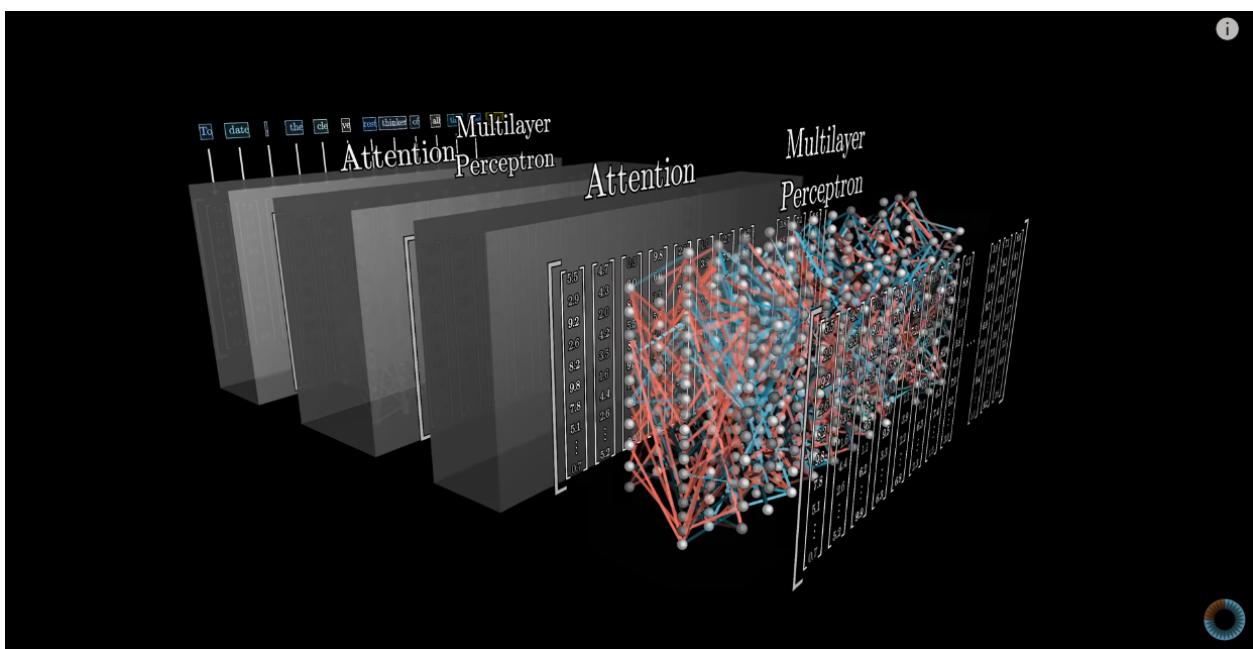
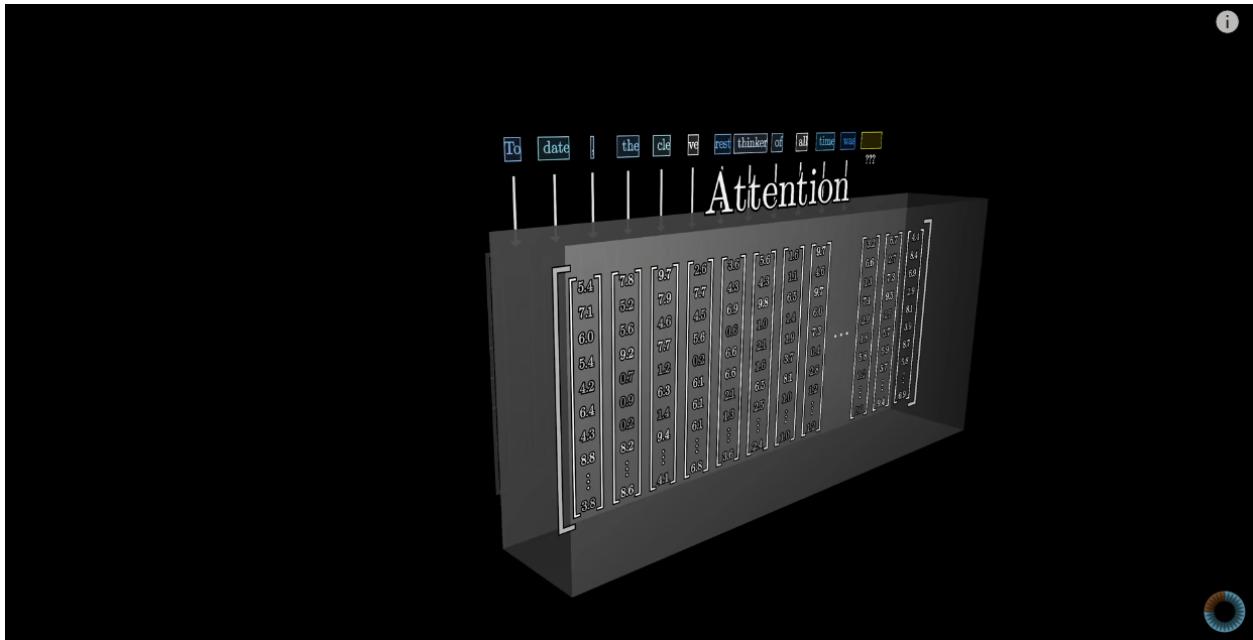


Estos son los pasos que vamos a seguir

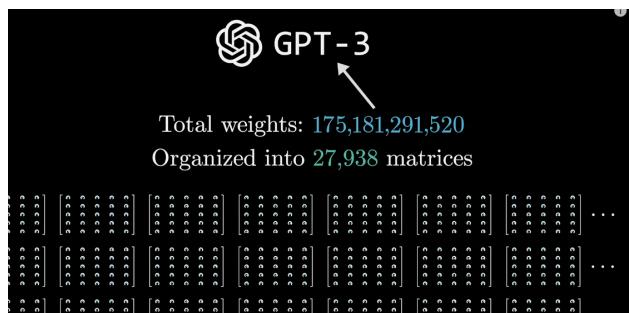


4.1 Tokens





https://www.youtube.com/playlist?list=PLZHQB0WTQDNU6R1_67000Dx_ZCJB-3pi



27 mil matrices divididas en 6 categorías:

Total weights: 175,181,291,520		GPT-3
Organized into 27,938 matrices		
Embedding	[...]	
Key	[...]	
Query	[...]	
Value	[...]	
Output	[...]	
Up-projection	[...]	
Down-projection	[...]	
Unembedding	[...]	

We will break this down

Total weights: 175,181,291,520		GPT-3
Organized into 27,938 matrices		
Embedding	$12,288 \times 50,257$	$= 617,558,016$
Key	$128 \times 12,288 \times 96 \times 96$	$= 14,495,514,624$
Query	$128 \times 12,288 \times 96 \times 96$	$= 14,495,514,624$
Value	$128 \times 12,288 \times 96 \times 96$	$= 14,495,514,624$
Output	$12,288 \times 128 \times 96 \times 96$	$= 14,495,514,624$
Up-projection	$n_{\text{neurons}} \times 12,288 \times 96$	$= 57,982,058,496$
Down-projection	$12,288 \times n_{\text{neurons}} \times 96$	$= 57,982,058,496$
Unembedding	$50,257 \times 12,288$	$= 617,558,016$

<https://transformer-circuits.pub/2021/framework/index.html>