Redes Neuronales

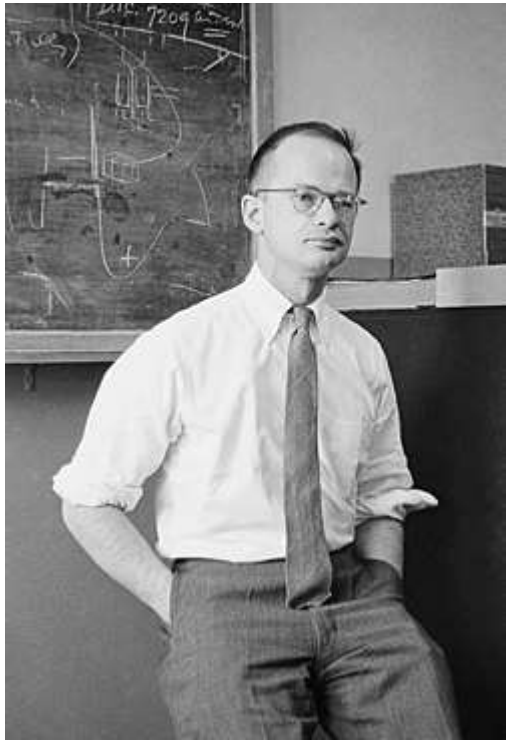
Módulo 4 - parte 2

Actualizado el 31/10/25

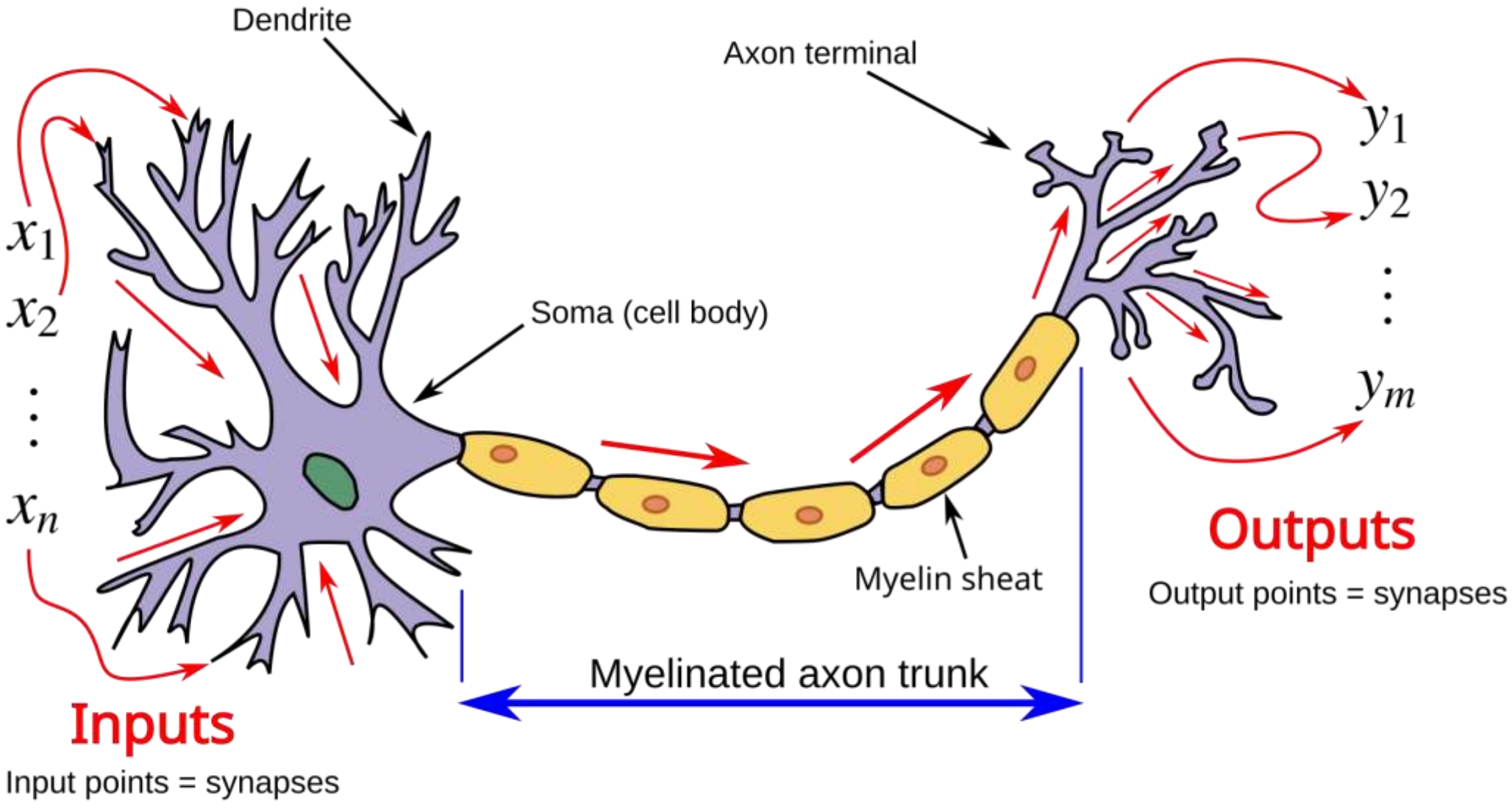
La Neurona Artificial

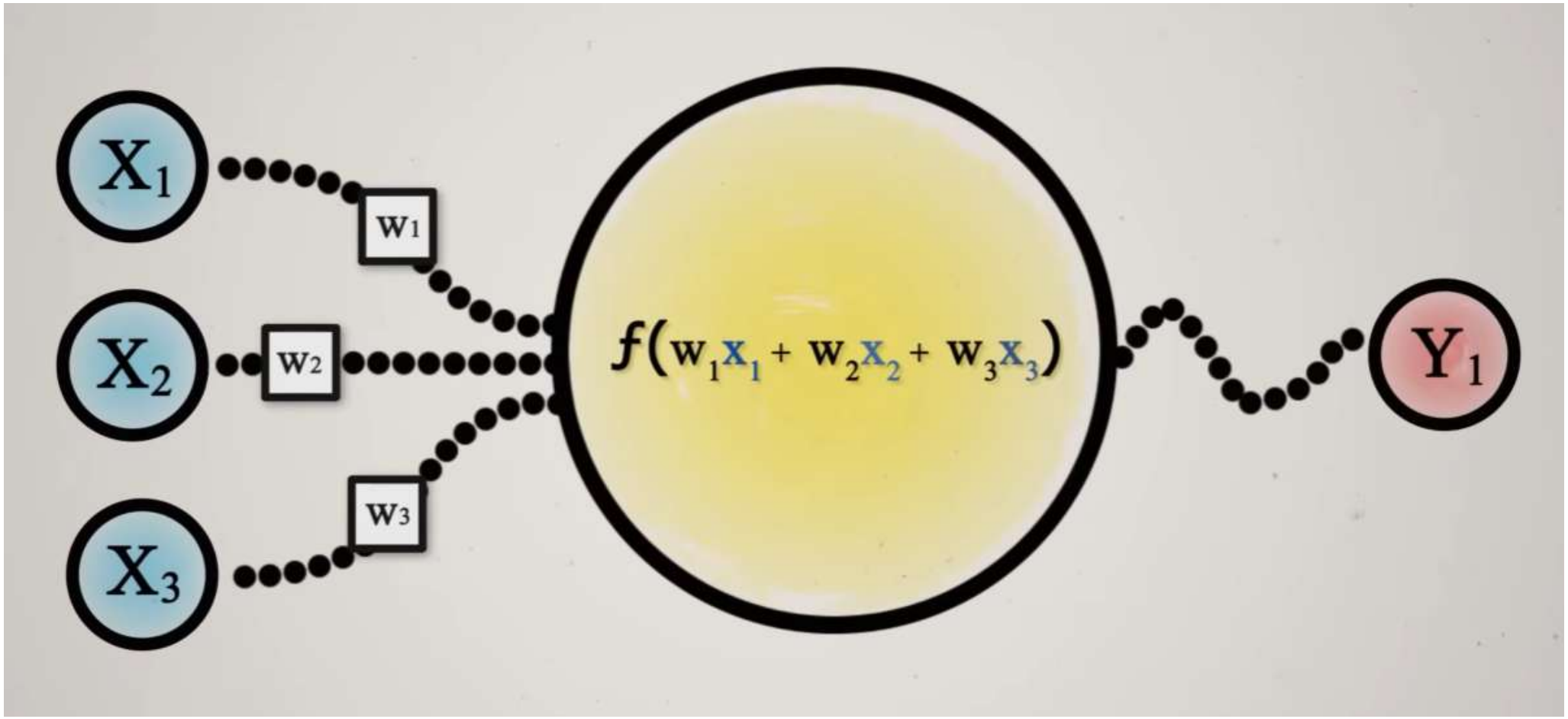
La primera neurona artifical

- 1943 – McCulloch y Pitts
- Es un **clasificador lineal binario** capaz de separar datos en dos categorías si estos son linealmente separables
- Características:
  - Entradas
  - Pesos (fijos) **No había aprendizaje**
  - Salida
  - Función de activación: escalón
- Podía emular las lógica AND, OR entre otras
- Fue un modelo teórico



Pitts en el MIT





¿Cómo funciona?

Funciona tomando múltiples **entradas**, multiplicándolas por sus respectivos "**pesos**" para asignarles importancia, sumando estos productos y luego aplicando una **función de activación** (con un umbral) para producir una **salida** binaria, usualmente un "1" o un "0".

Las matemáticas de la Neurona Artificial

Suma ponderada:

$$z = \sum_{i=1}^{n} w_i x_i$$

$x = entradas$
$w = pesos$

Función de activación:

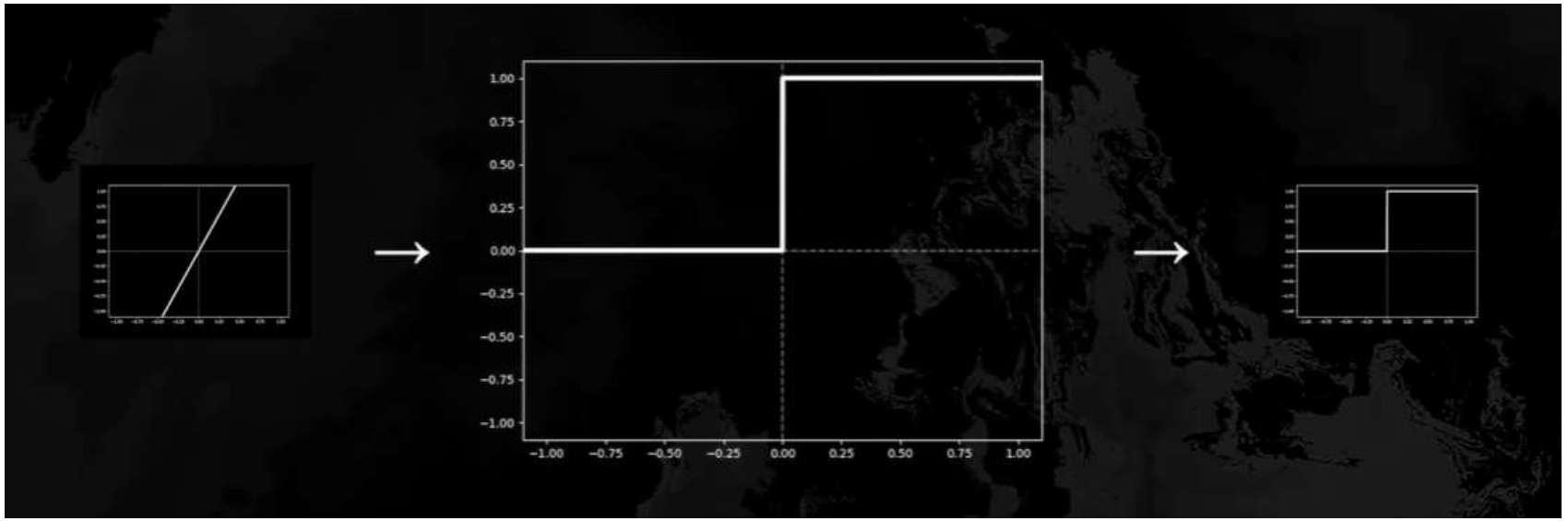
$$f(z) = \begin{cases} 1 \; si \; z \geq \alpha \\ 0 \; si \; z < \alpha \end{cases}$$

α = umbral que puede variar, generalmente es 0

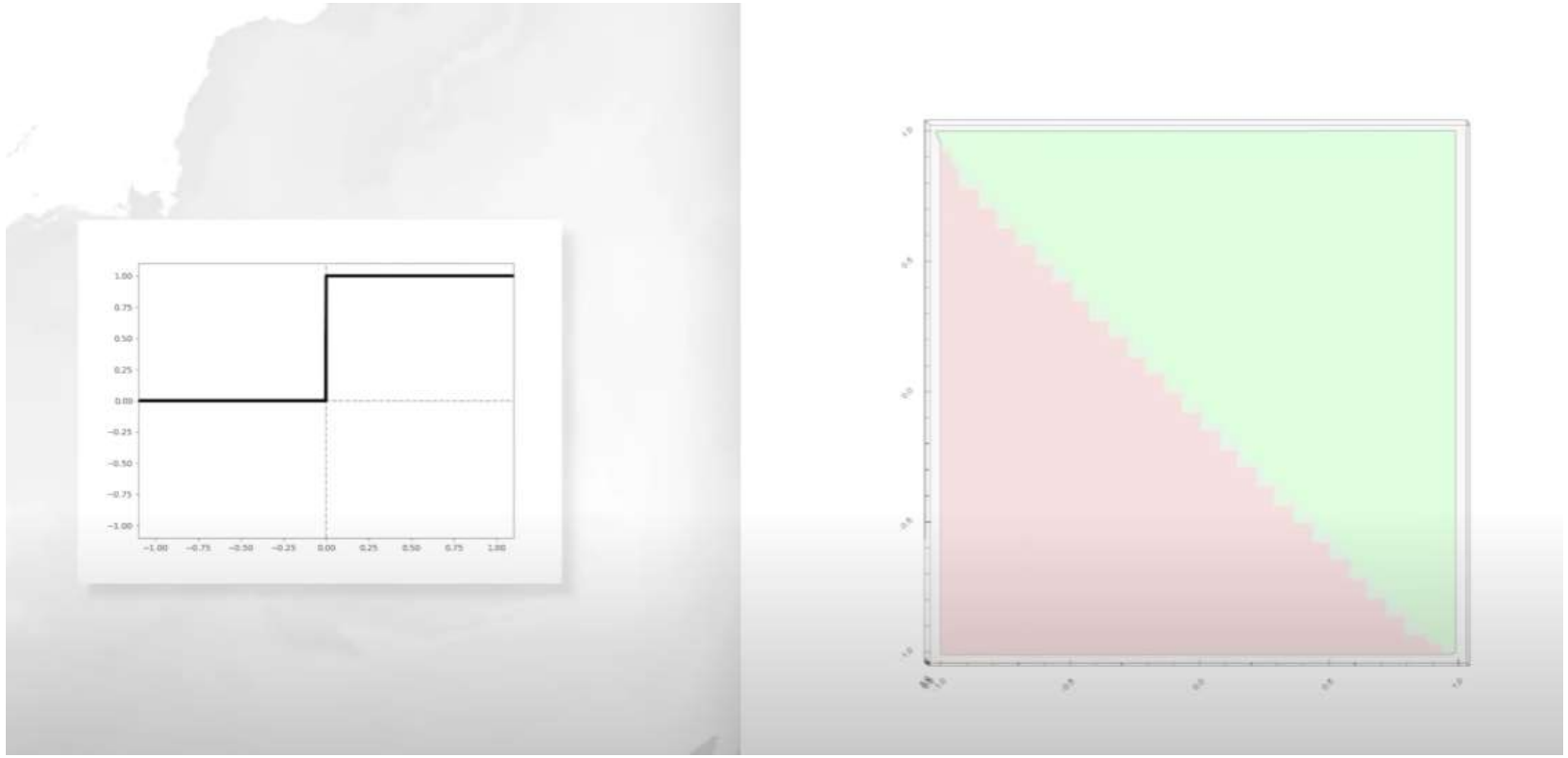
Salida:

$$y = f(z)$$

La función de activación: Escalonada

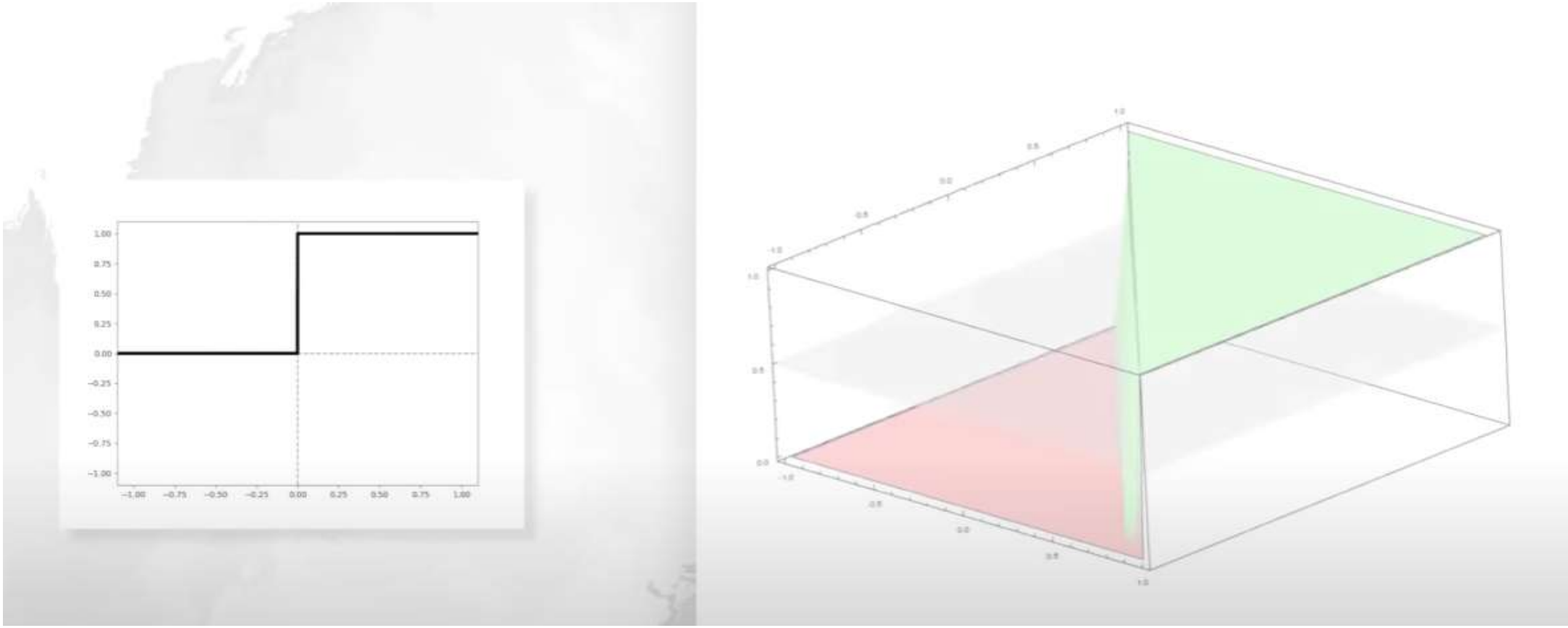


La función de activación: Escalonada



La función de activación distorciona el plano generado por la neurona.

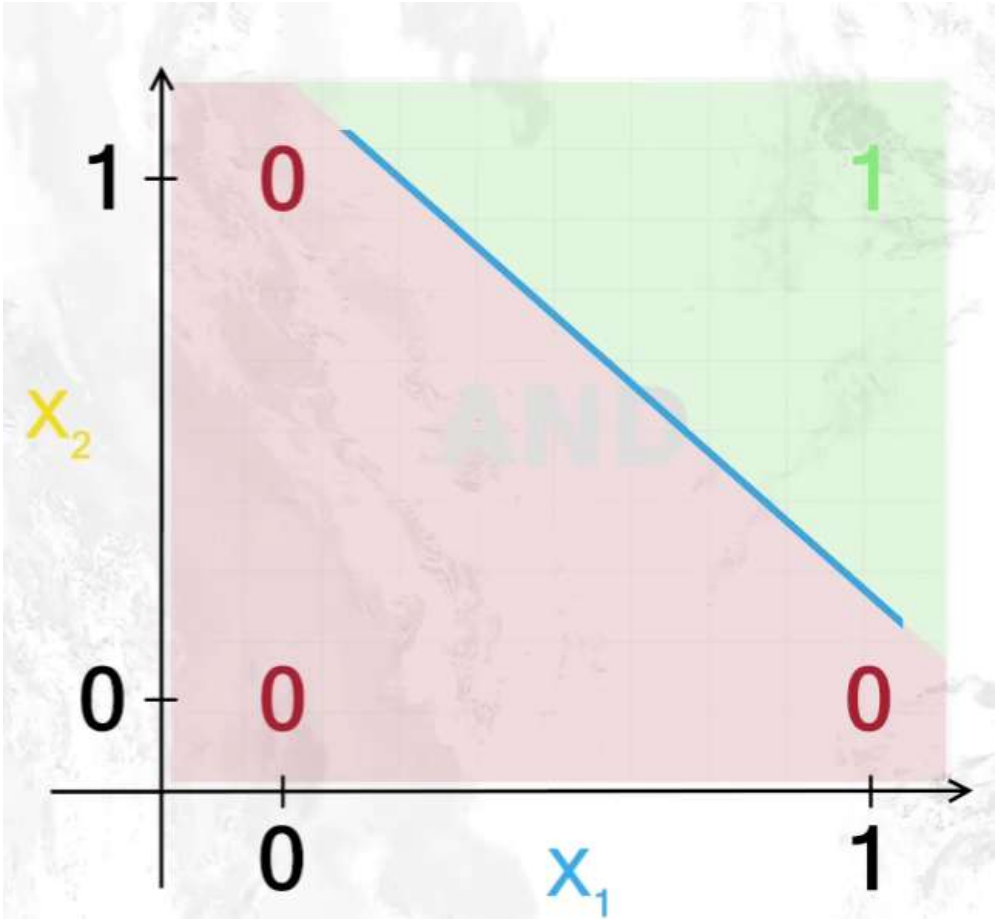
La función de activación: Escalonada



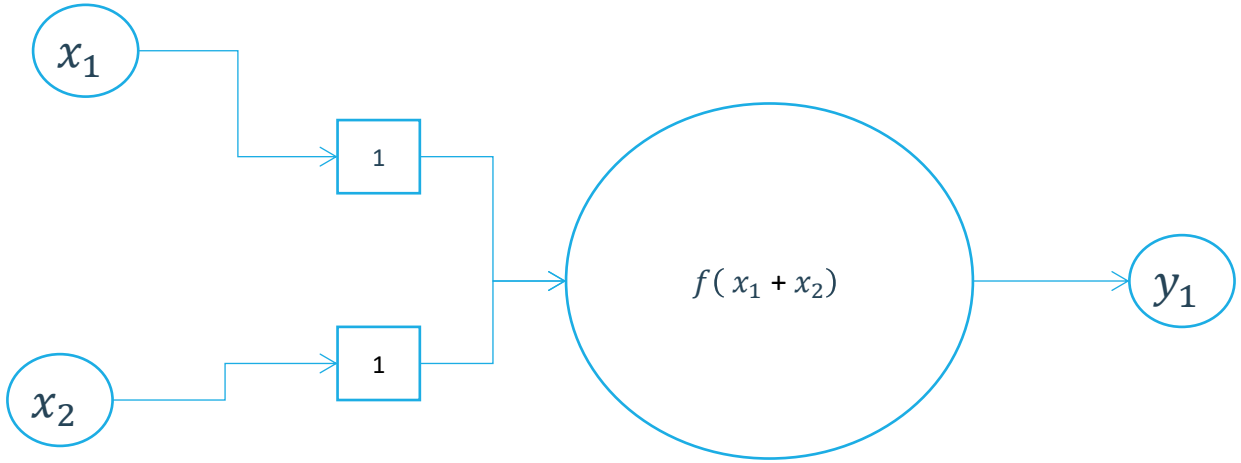
La función de activación distorciona el plano generado por la neurona.

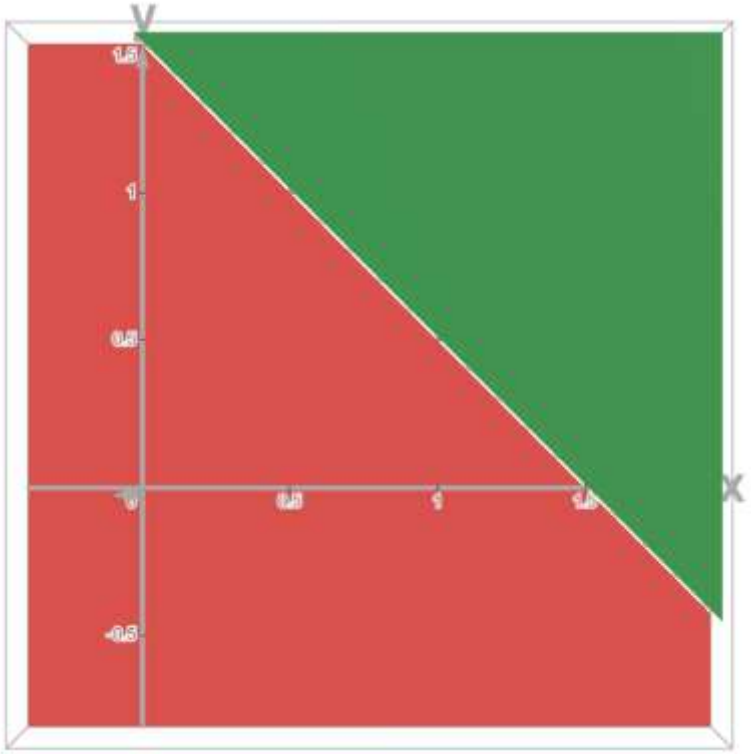
Ejemplo AND

a	b	AND
0	0	0
0	1	0
1	0	0
1	1	1



Ejemplo AND

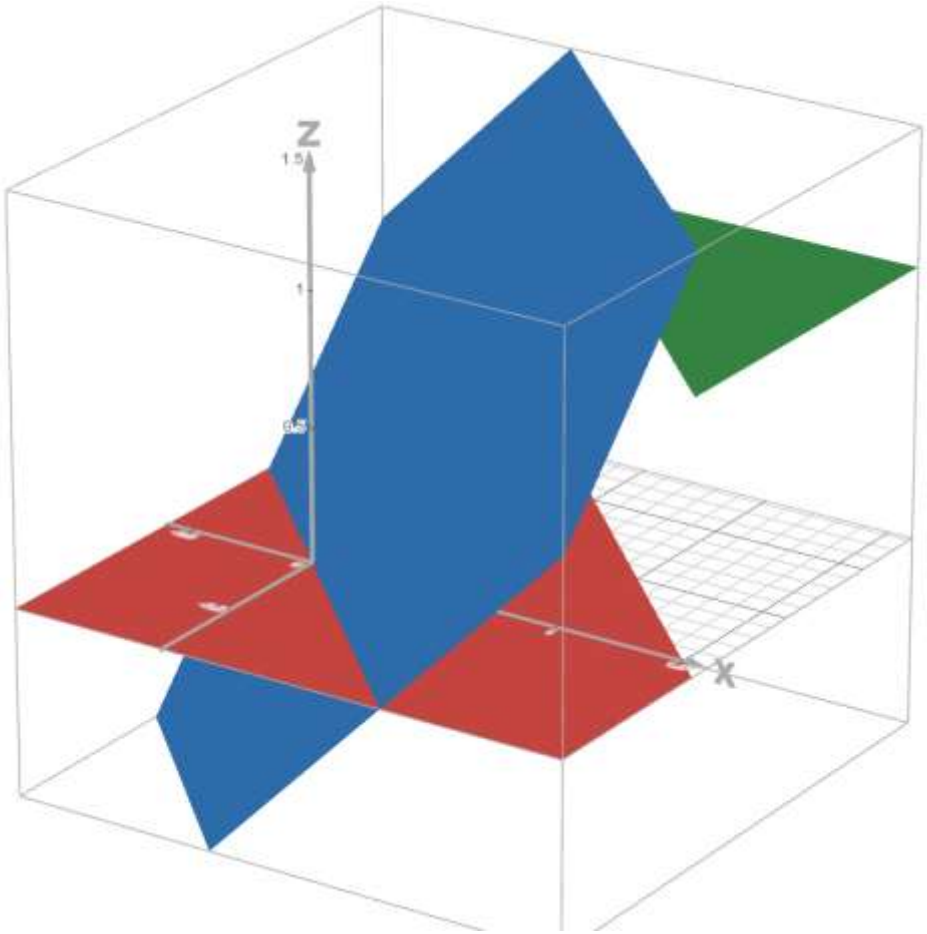




X1	X2	W1	W2	z = Suma	y = f(z)	Salida deseada		Parámetros:	
0	0	1	1	0	0	0		Umbral	1,5
1	0			1	0	0			
0	1			1	0	0			
1	1			2	1	1			

El ejemplo en 3D

1. $z = x + y$
2. $z = \{x + y \geq 1.5 : 1\}$
3. $z = \{x + y < 1.5 : 0\}$



Ejercicios

- Con Excel crear una red neuronal que resuelva la compuerta lógica OR
- Crear código en Python para emular una neurona artificial

Limitaciones

- Entradas y salidas binarias
- Pesos fijos e iguales lo cual implica que no “aprende”
- Función de activación escalón
- Solo funciones linealmente separables

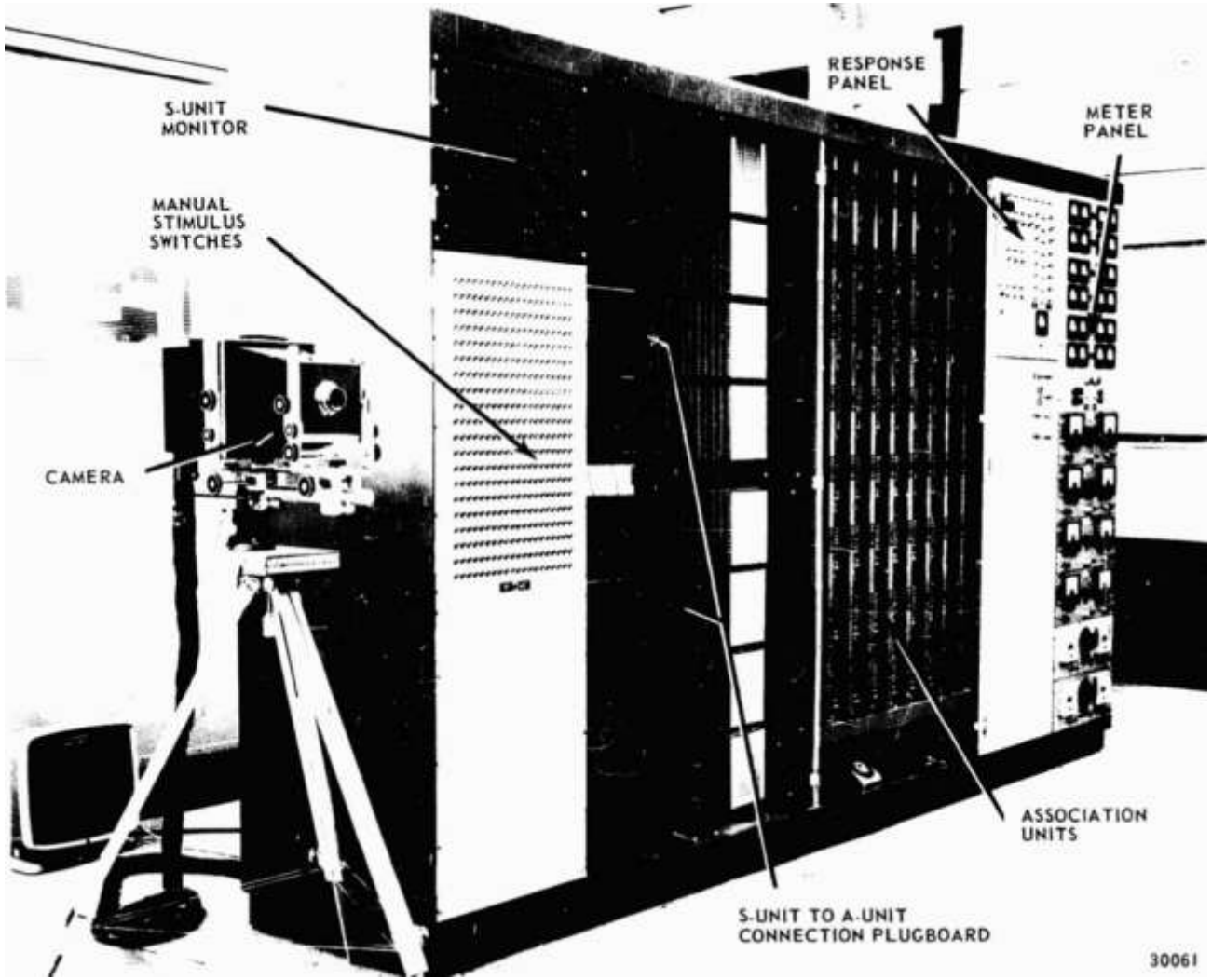
El Perceptrón

El Perceptrón

- 1957 – Frank Rosenblatt
- Es un clasificador lineal capaz de separar datos en dos categorías si estos son linealmente separables
- Características:
  - Una neurona
  - **Pesos que se aprenden con la "regla de aprendizaje"**
  - Función escalón o Sign function
- Reconocimiento de patrones (carácteres, clasificación)
- Clasificación lineal binaria inicialmente o también contínua si la función de activación es la identidad (o sea, como si no estuviera)
- Se creo hardware para resolver problemas prácticos

El Perceptrón

Frank Rosenblatt

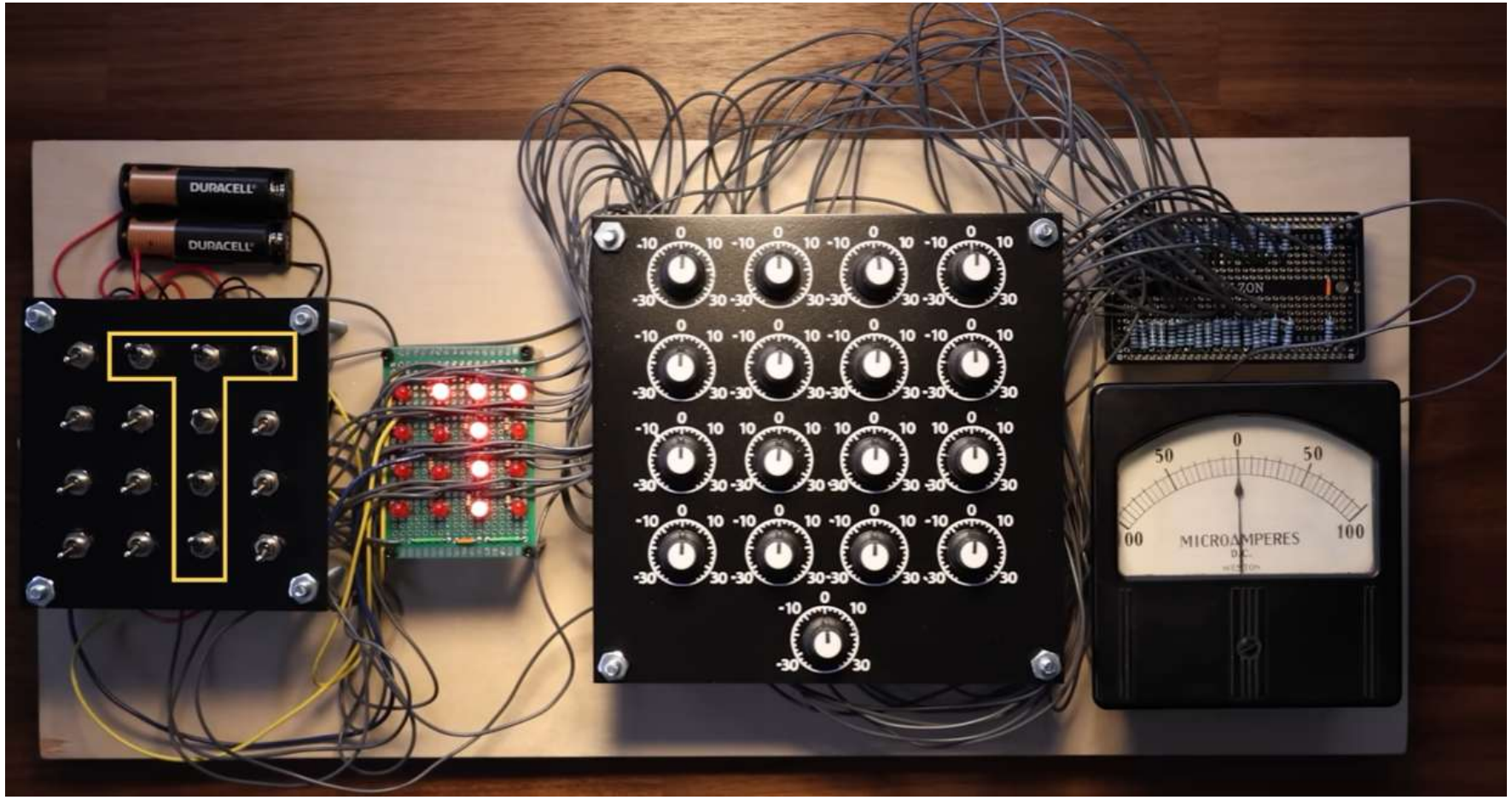


Mark I Perceptron

¿Cómo funciona?

Funciona tomando múltiples **entradas**, multiplicándolas por sus respectivos "**pesos**" para asignarles importancia, sumando estos productos y luego aplicando una **función de activación** (con un umbral) para producir una **salida** binaria, usualmente un "sí" o un "no".

El Perceptrón



Las matemáticas del Perceptrón

Suma ponderada:

$$z = \sum_{i=1}^{n} w_i x_i + b\, w_0$$

$x = entradas$
$w = pesos$
$b = sesgo$

Función de activación:

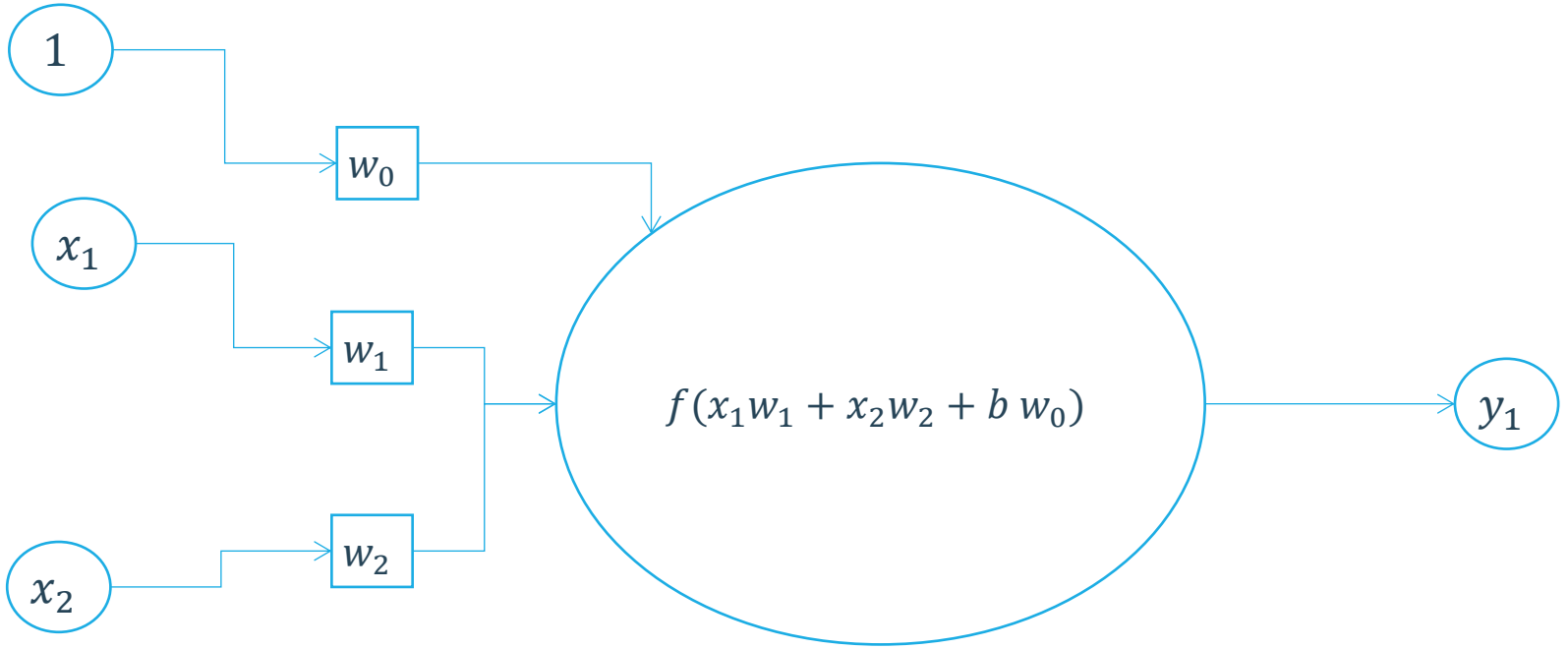
$$f(z) = \begin{Bmatrix} 1\ si\ z \geq \alpha \\ 0\ si\ z < \alpha \end{Bmatrix}$$

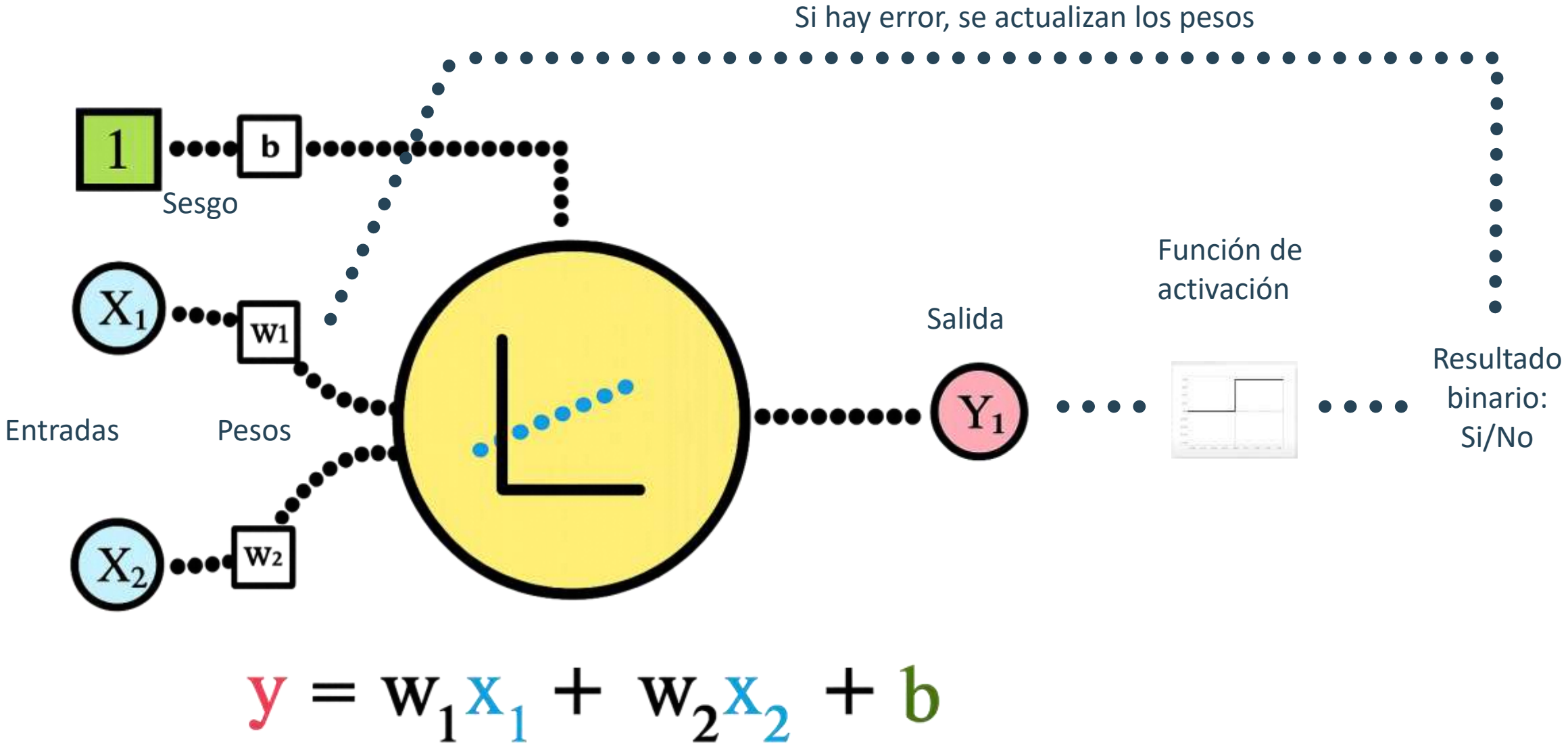
α = umbral que puede variar, generalmente es 0

Salida:

$$y = f(z)$$

El Perceptrón





$$y = w_1 x_1 + w_2 x_2 + b$$

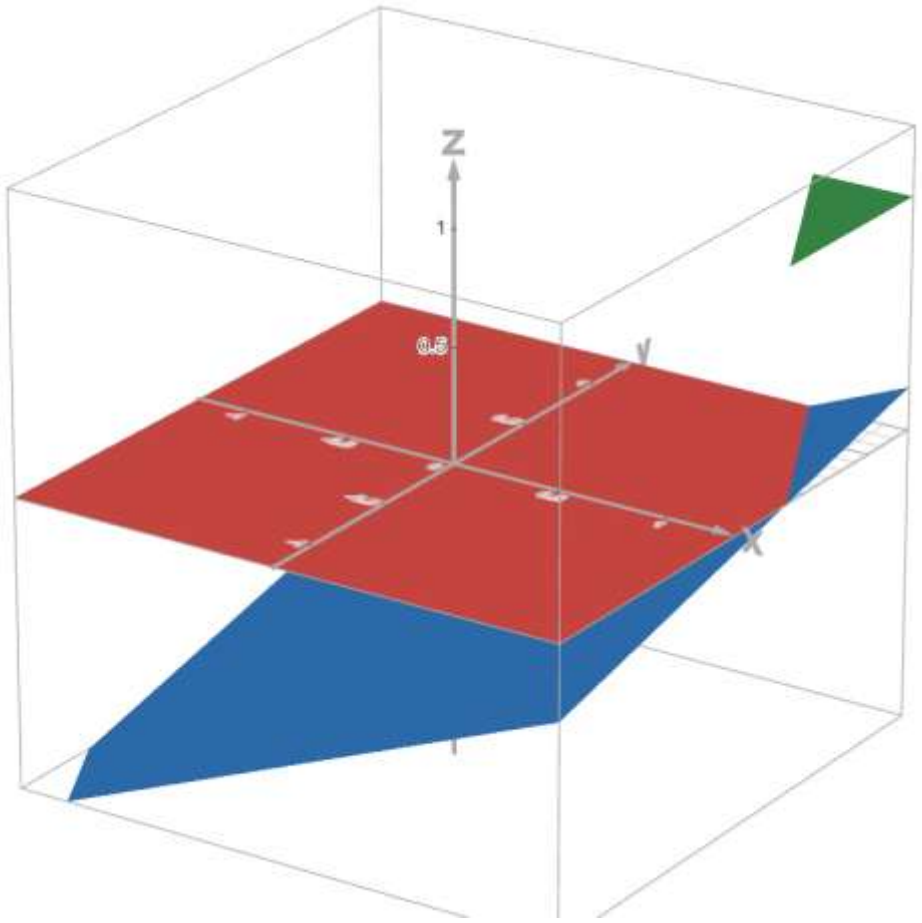
La regla de aprendizaje: el entrenamiento

1. **Inicializar:** Asignar pesos y sesgo aleatorios pequeños (o cero).
2. **Iterar:** Repetir hasta que clasifique correctamente **todas** las entradas del conjunto de entrenamiento:
   a) Calcular la suma ponderada z
   b) Calcular la salida del perceptrón $y=f(z)$ usando la función de activación.
   c) Calcular el error $e = d - y$. (d es la salida deseada, y es la salida predicha)
   d) Calcular la variación Δw que aplicaremos a los siguientes pesos: $\Delta w = \eta * e * x_i$ (η es la tasa de aprendizaje)
   e) Ajustar los pesos:

$$w_{i\ nuevo} = w_{i\ antiguo} + \Delta w_i$$

El Perceptrón

1. $z = 0.4x + 0.2y - 0.6$
2. $z = \{0.4x + 0.2y - 0.6 \geq 0 : 1\}$
3. $z = \{0.4x + 0.2y - 0.6 < 0 : 0\}$



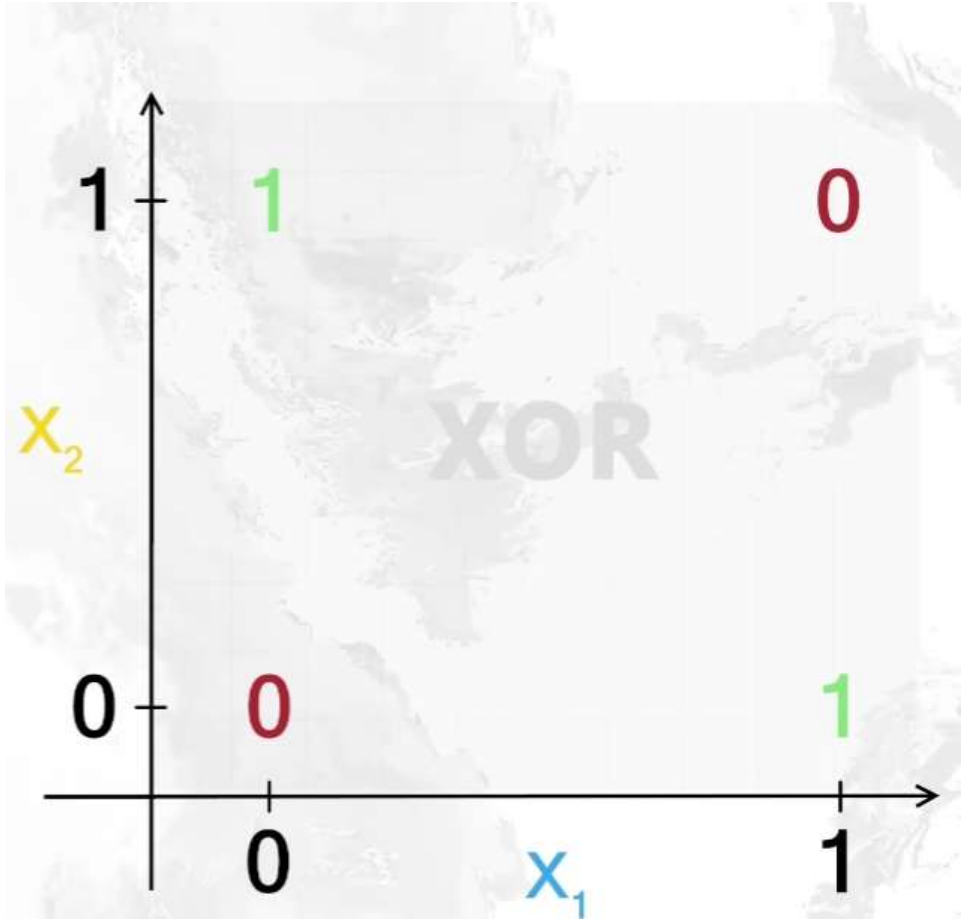
El Perceptrón en la práctica

- Con Excel

Ejercicios

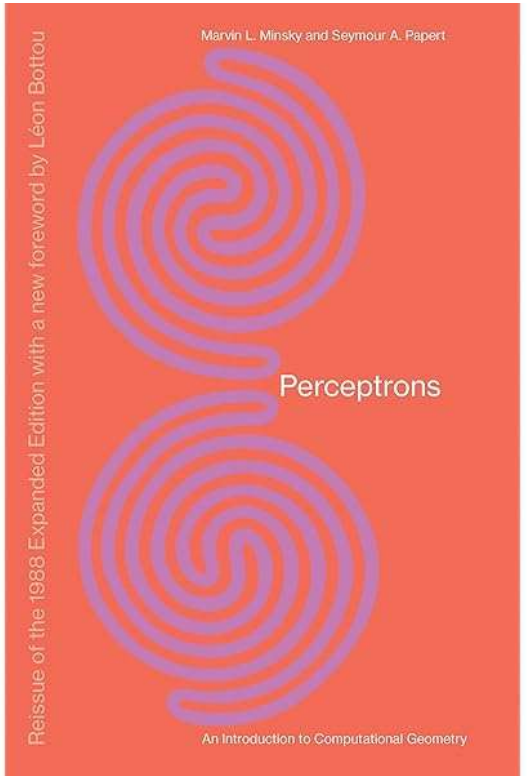
- Implementar el perceptrón en Python
- Hacer un Perceptrón que lea números de una matriz de 3 x 5 en Excel. Entrenarlo para que reconozca el número 7, añadiendo el 1,3,4 y 5 como negativos.
- Implementarlo también en Python

El problema de la compuerta lógica XOR



El problema del Perceptrón

- Con una sola neurona no se puede resolver una compuerta XOR
- Solo puede resolver problemas **linealmente** separables
- En 1969 se publica el libro “Perceptrons” dando incio al primer “inverno de la IA”



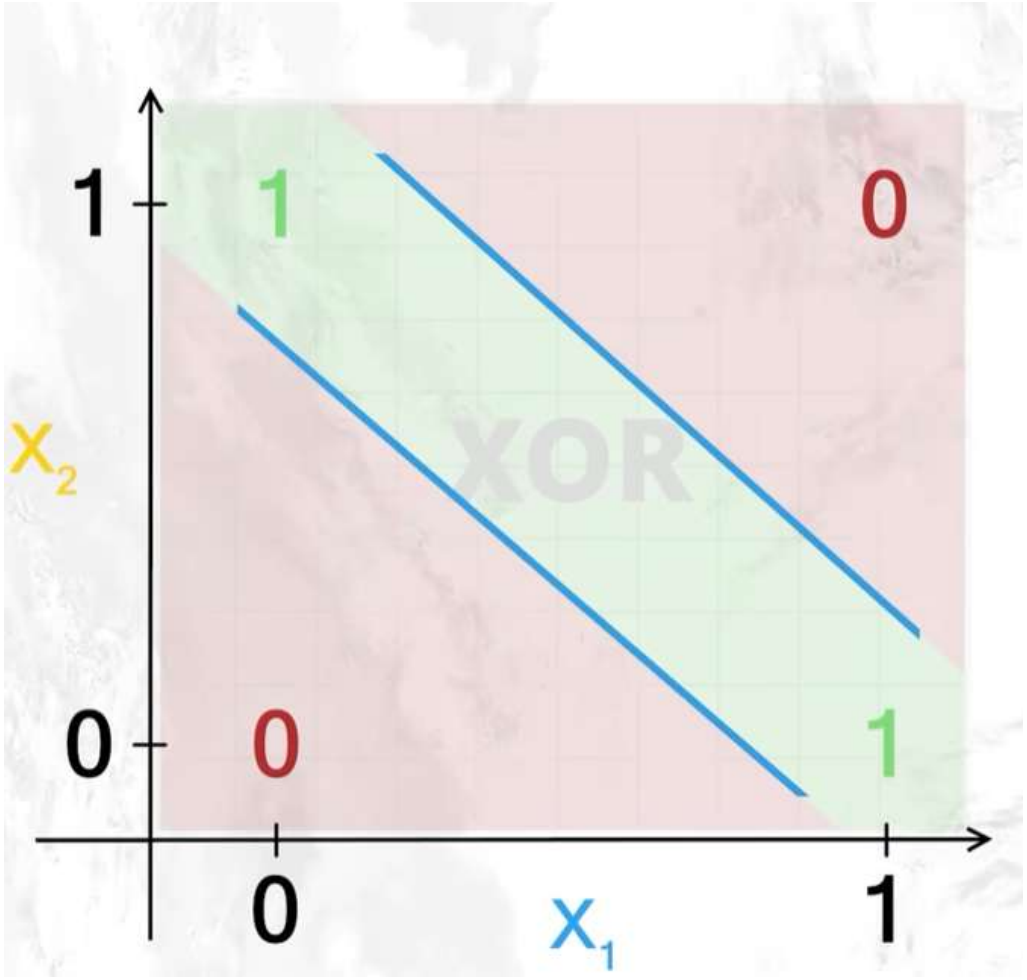
El Perceptrón Multicapa

La solución al problema XOR

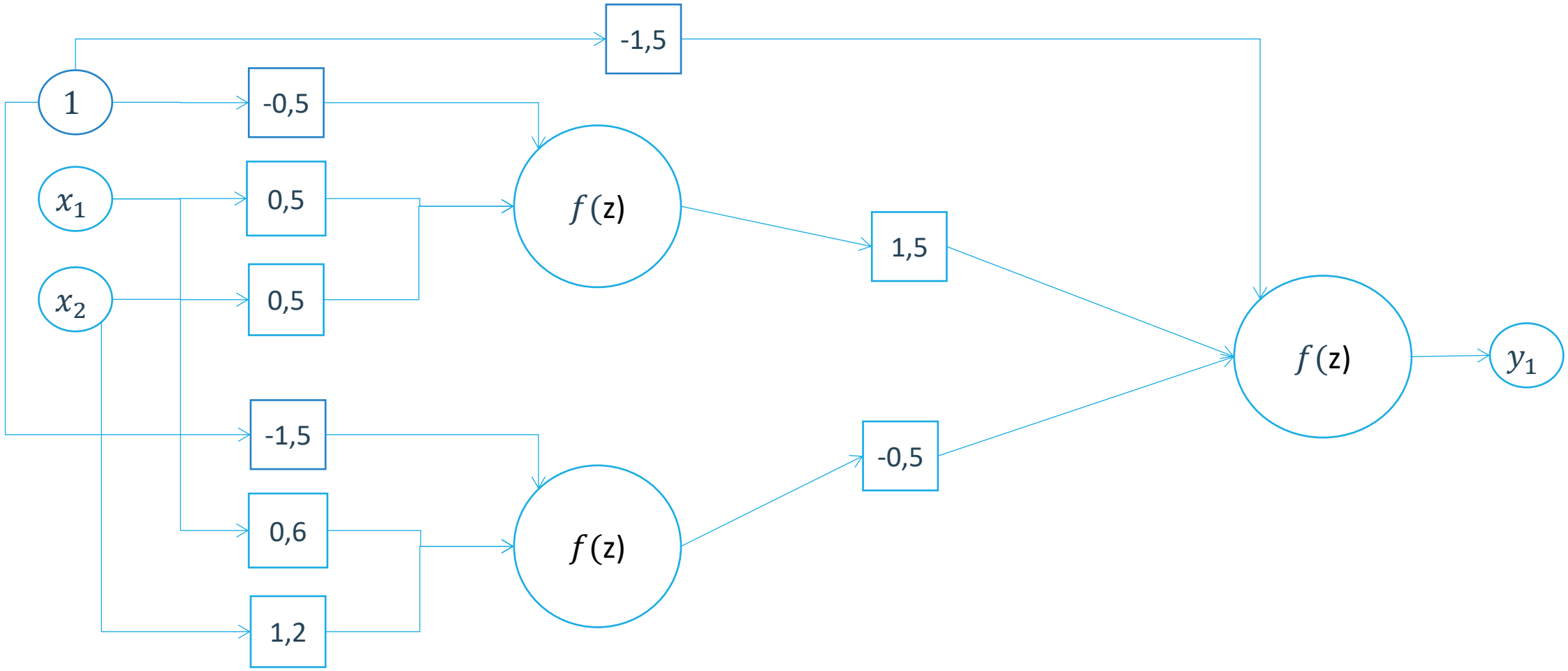
El Perceptrón Multicapa

- Desarrollo: 1965 por Alexey Ivakhnenko
- Más neuronas y más de una capa permiten modelar funciones más complejas
- Más funciones de activación (Sigmoide, ReLu, etc.)

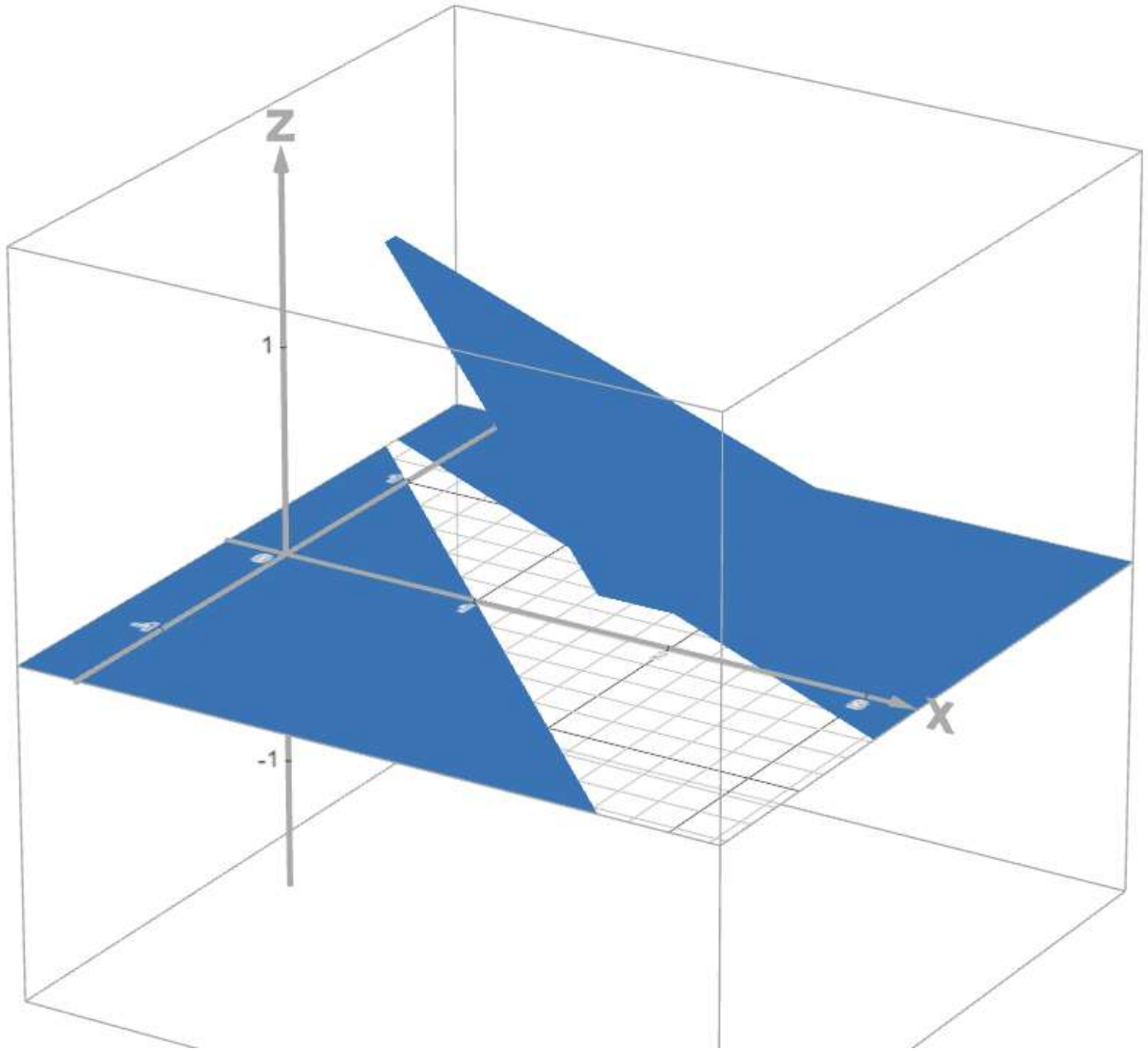
La solución



La solución usando Excel



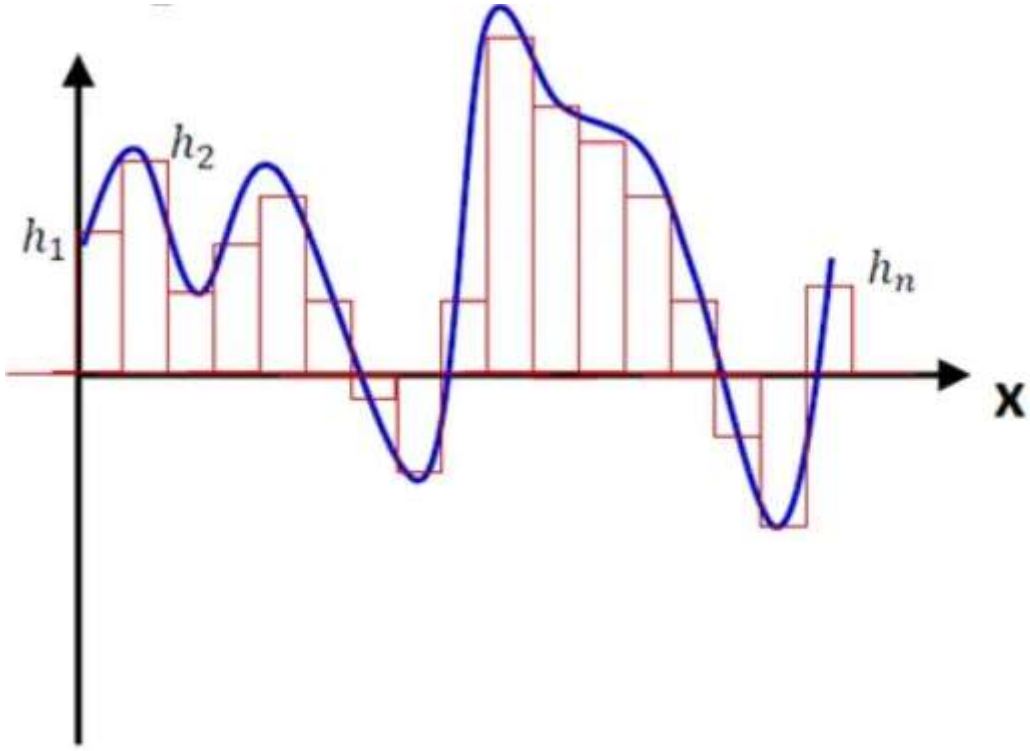
La solución



Teorema de aproximación universal

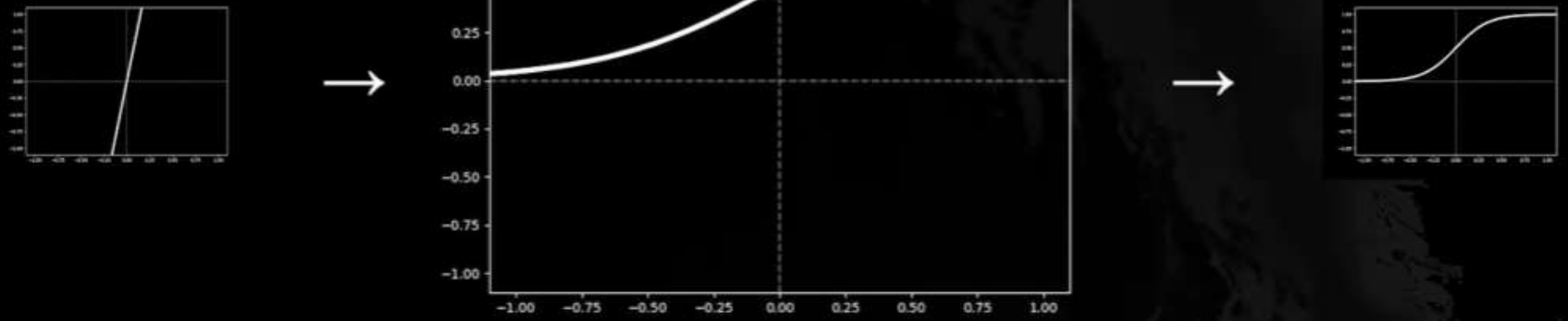
Una red neuronal de tipo "feedforward" (o alimentación hacia adelante) con al menos una capa oculta, y con un número finito de neuronas, puede **aproximar cualquier función continua** a un grado arbitrario de precisión, siempre que la función de activación utilizada en las neuronas de la capa oculta sea no lineal (sino sería como una sola) y continua (como la función sigmoide, ReLU, tanh, etc.).

Teorema de aproximación universal



FUNCION DE ACTIVACION

SIGMOIDE

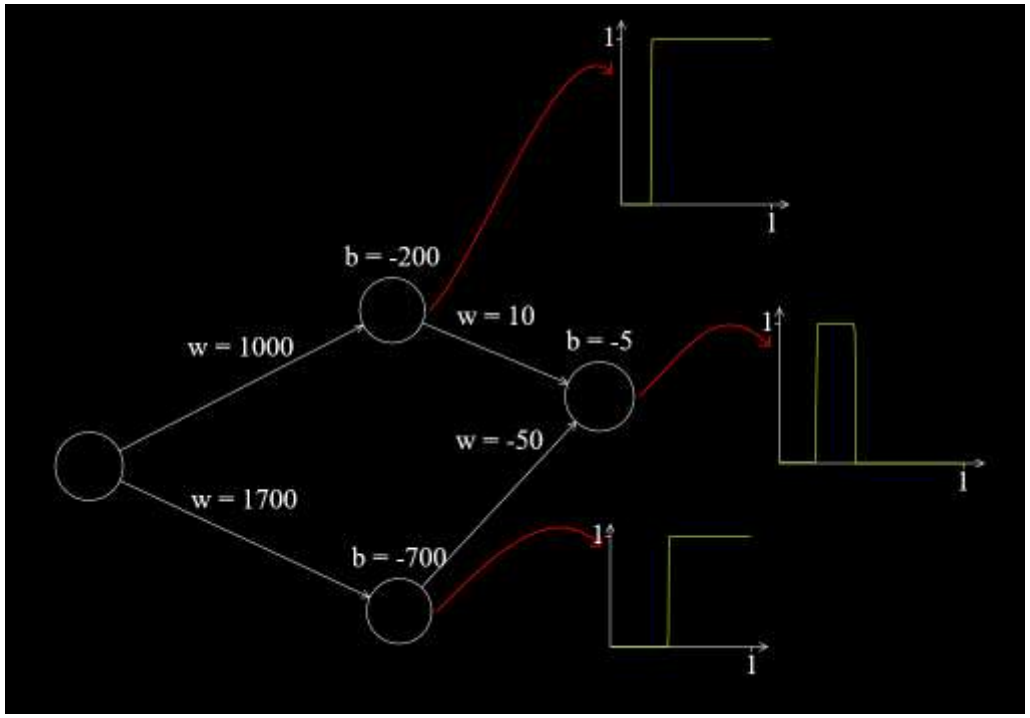


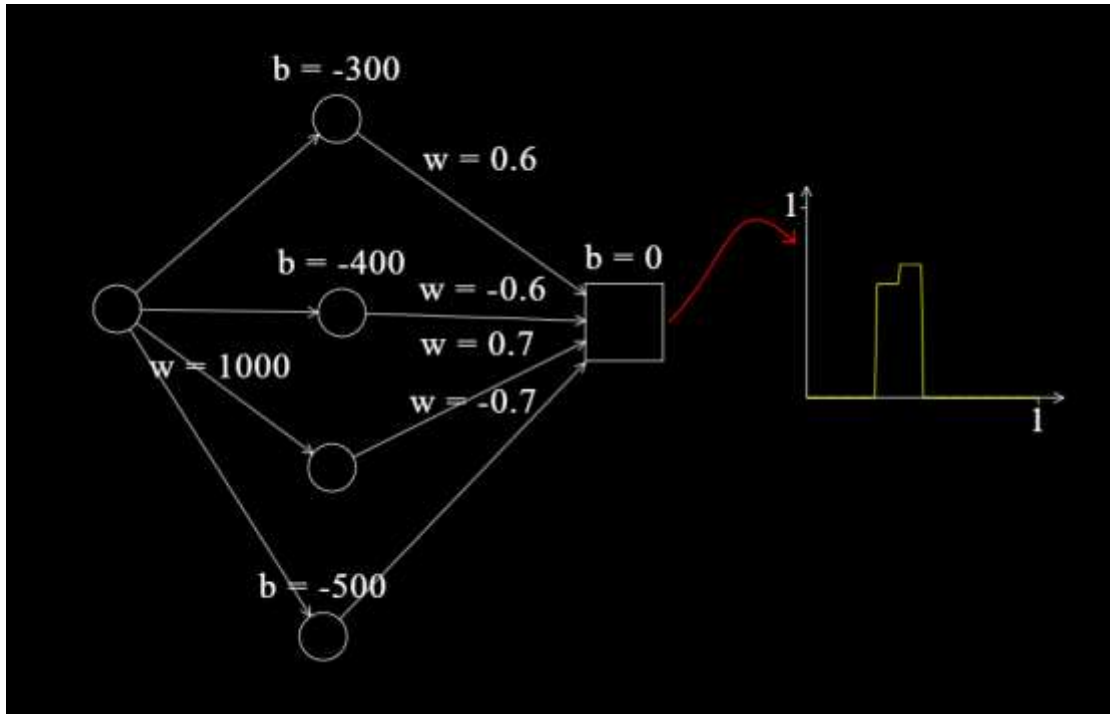
$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

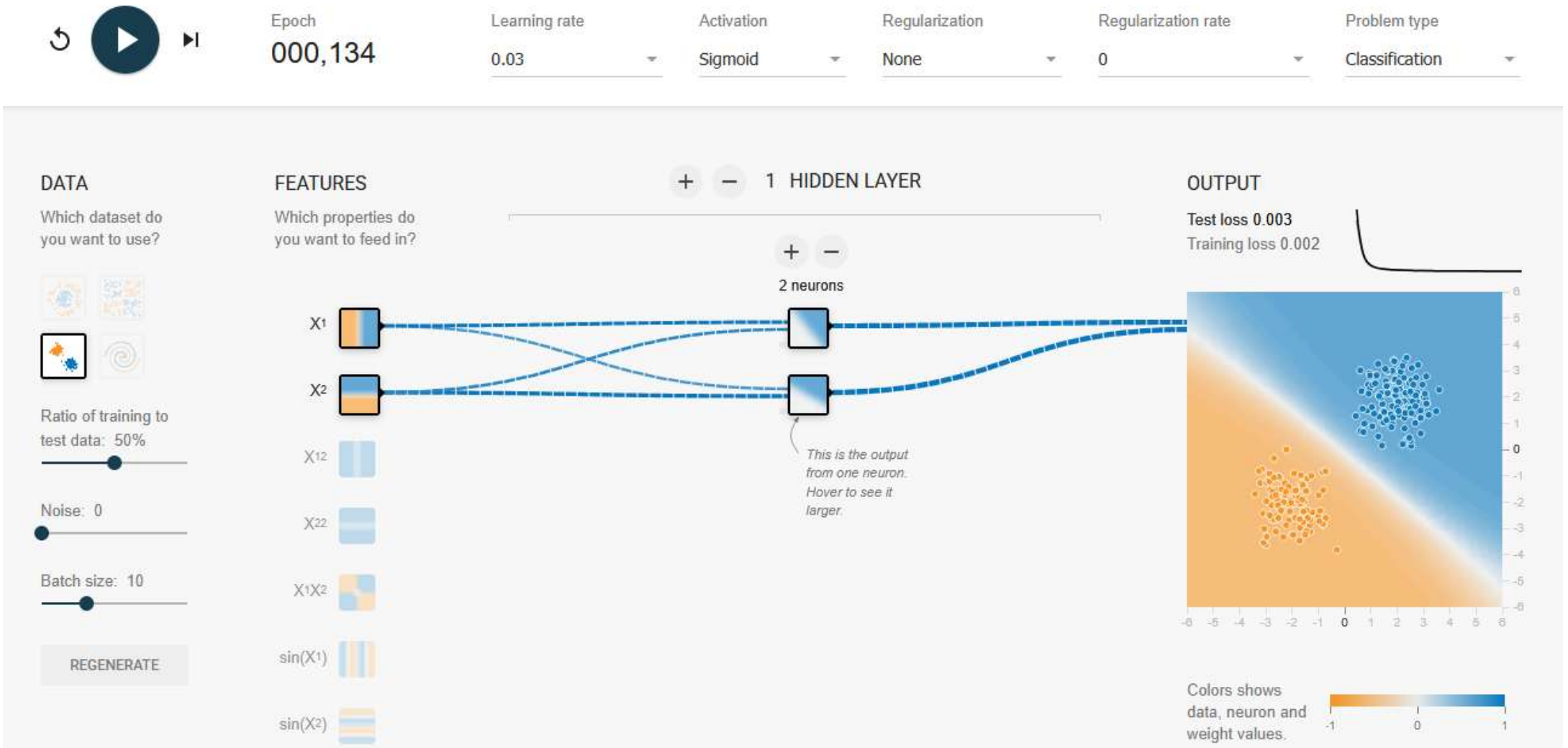
1. Función Sigmoide (Logística)

- **Fórmula:** $f(x) = \frac{1}{1+e^{-x}}$
- **Rango de Salida:** $(0, 1)$

Teorema de aproximación universal







Ejercicio

Crear código en Python que resuelva la compuerta XOR, dadas la entradas AND y OR

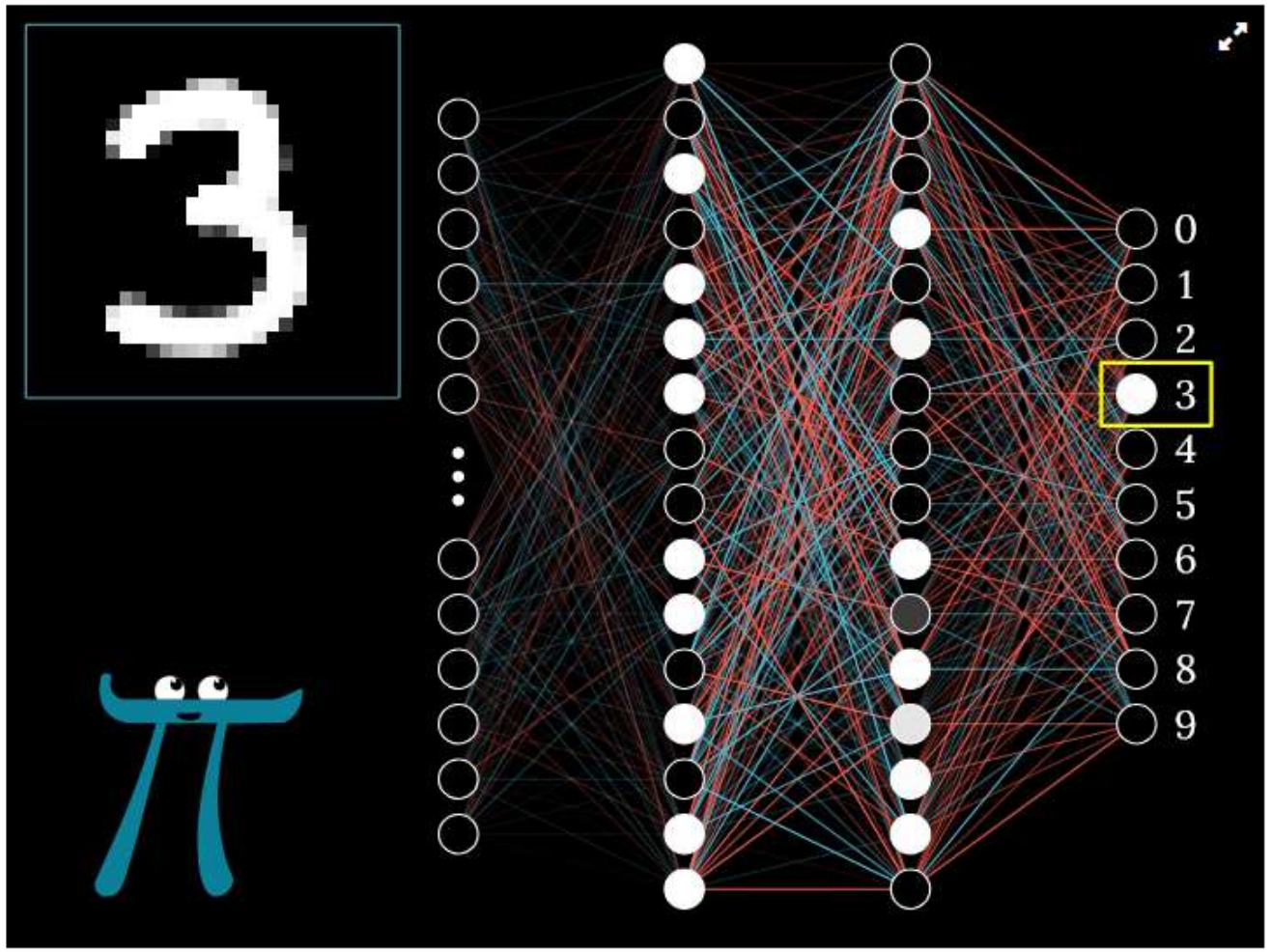
Problemas del MLP

- No existía un método efectivo para entrenar más de una capa
- Comenzó el segundo “invierno de la IA”

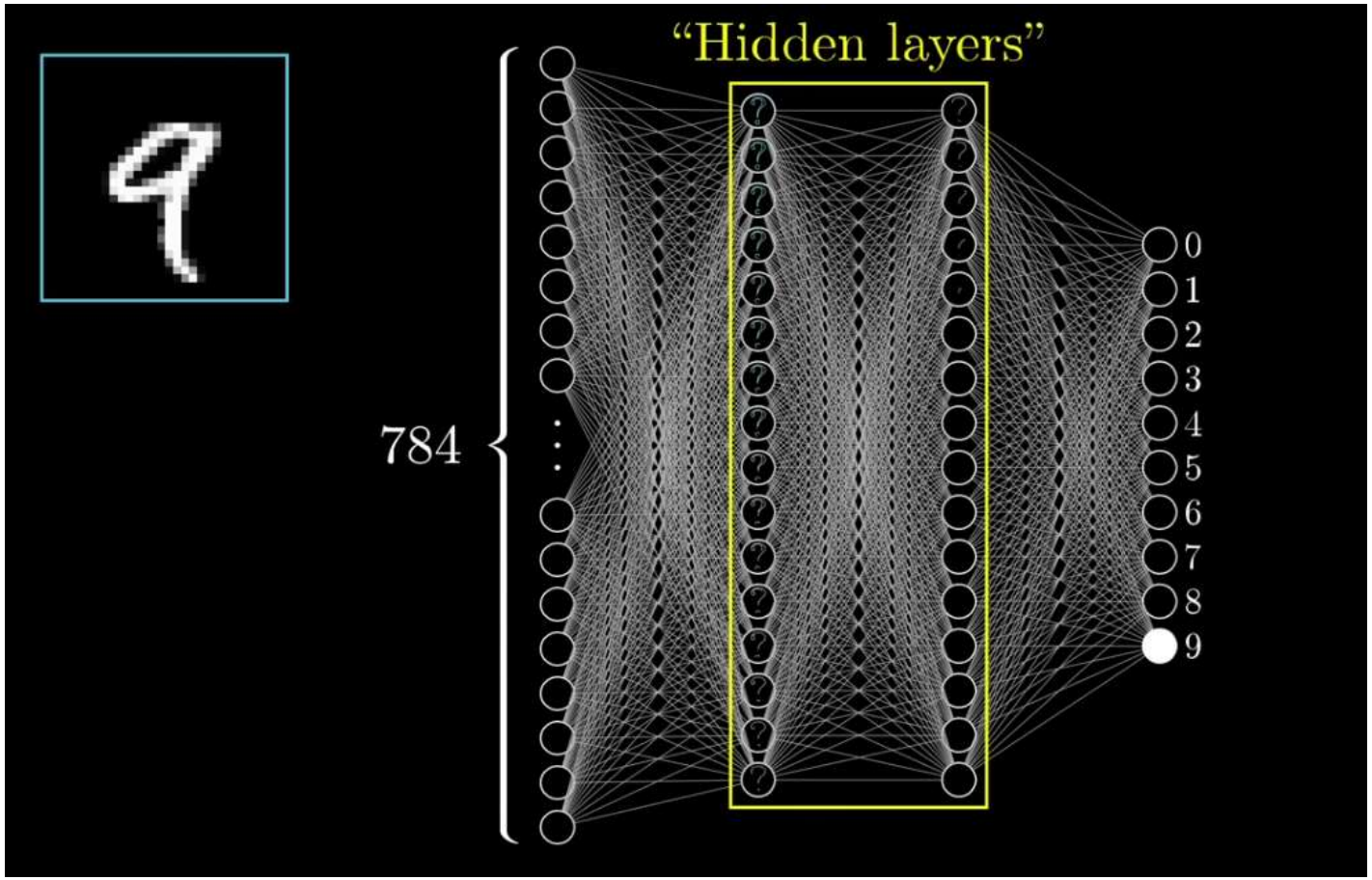
Red Neuronal

Con varias capas

Demo

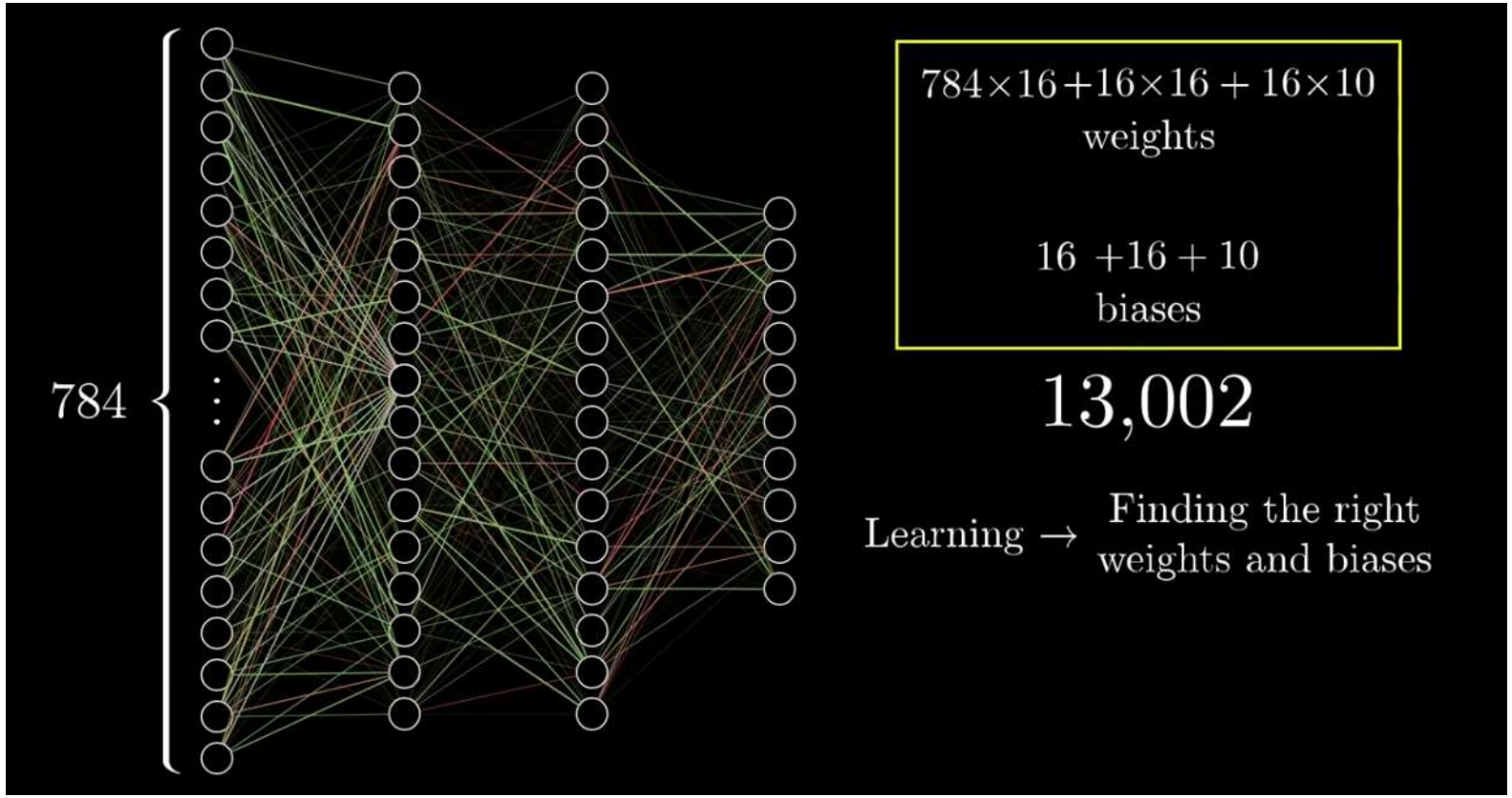


Ejemplo de Red neuronal



Ejemplo de Red neuronal

¿Qué es el entrenamiento?



La Retropropagación

Entrenando a las redes neuronales

La Retropropagación

1986: El MLP resurgió y se popularizó gracias a que **David Rumelhart**, **Geoffrey Hinton** y **Ronald Williams redescubrieron** y aplicaron el algoritmo de Retropropagación (Backpropagation), permitiendo por fin el entrenamiento eficiente de las redes multicapa.

La Retropropagación

Es un algoritmo esencial en el entrenamiento de redes neuronales que **calcula eficientemente los gradientes** de la **función de pérdida** (el error) con respecto a **todos los pesos y sesgos** de la red. Lo hace propagando el error desde la capa de salida hacia atrás, a través de las capas ocultas, utilizando la **regla de la cadena** del cálculo diferencial, permitiendo así determinar cómo cada ajuste individual en cualquier capa contribuye al error global y, en consecuencia, cómo deben ser modificados por un algoritmo de optimización como el **descenso del gradiente**.

La Retropropagación

- Descenso del gradiente
  - Pendiente de una recta
  - La derivada
  - El gradiente
  - Derivadas parciales
  - Vector gradiente
  - El cálculo del descenso del gradiente
  - Aplicado a las redes neuronales
  - La función de coste
- Función compuesta
- La regla de la cadena

Descenso del gradiente

Cálculo de mínimos de funciones

Descenso del gradiente

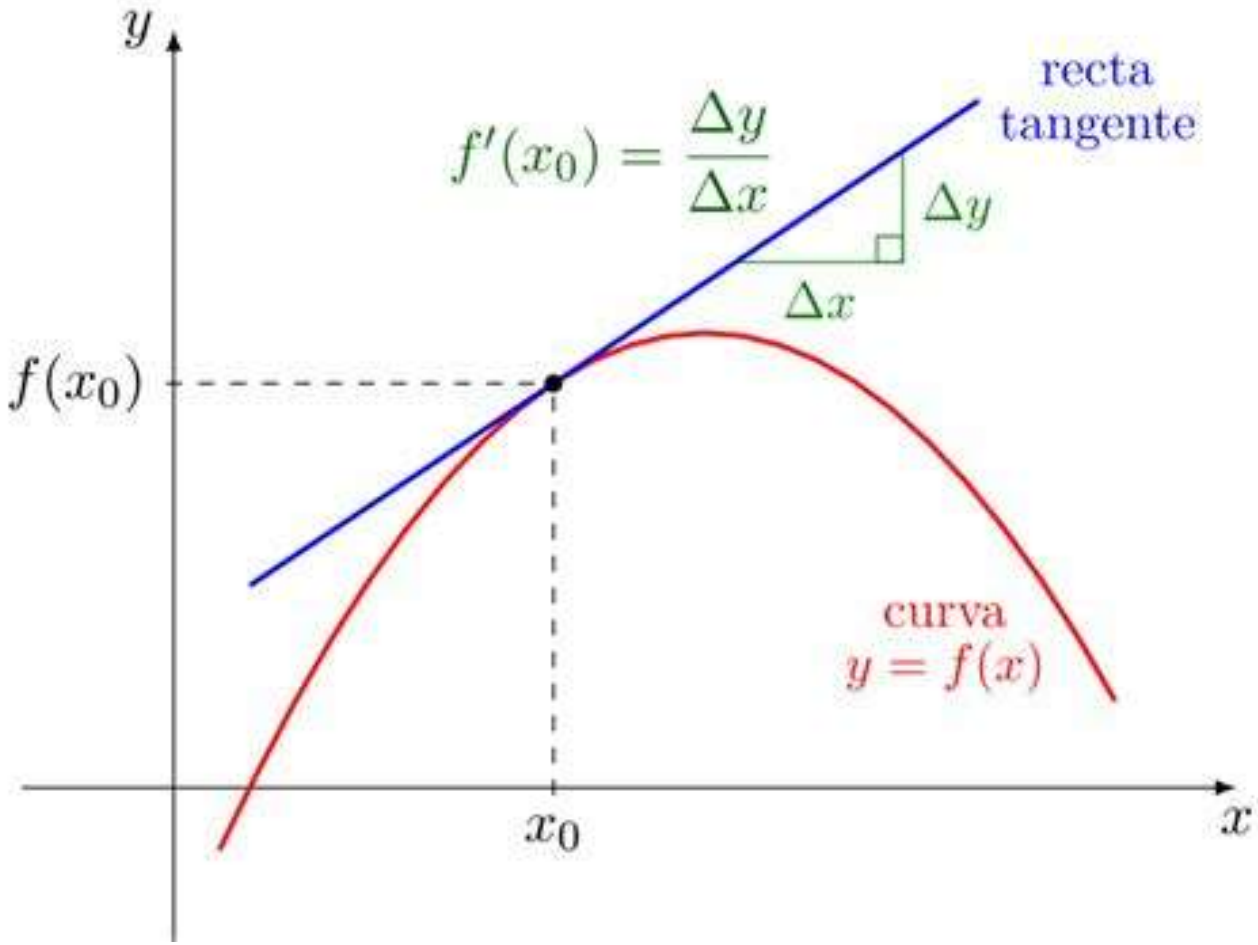
Es un algoritmo de optimización iterativo que permite **encontrar el mínimo** en una función **diferenciable**. La idea es tomar pasos de manera repetida en dirección contraria al gradiente.

Conceptos previos

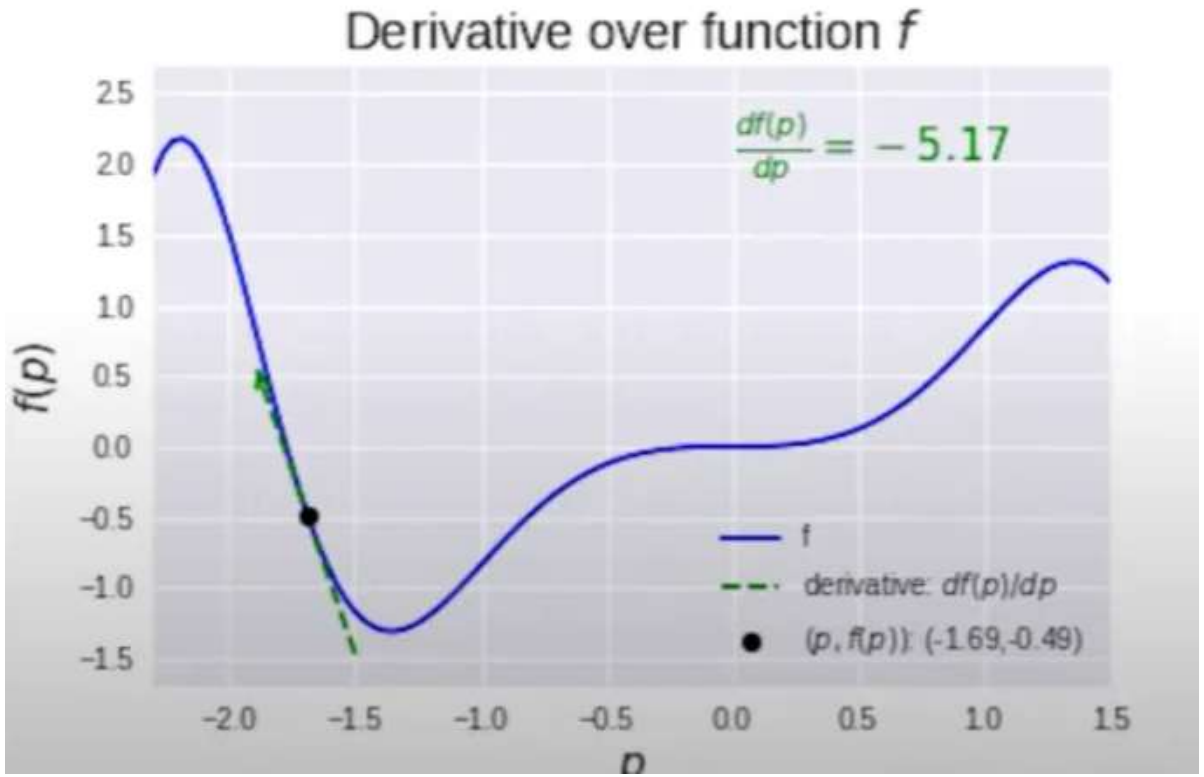
- Derivada
- Gradiente

La Derivada

El valor de la derivada de una función en un punto puede interpretarse geométricamente, ya que se corresponde con la pendiente de la recta tangente a la gráfica de la función en dicho punto.



La Derivada



La Derivada

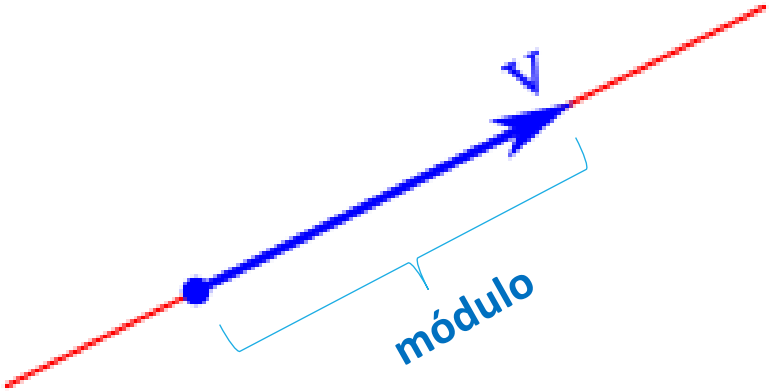
$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Derivadas parciales

La derivada parcial de una función de **varias variables** es la derivada con respecto a cada una de esas variables manteniendo las otras como **constantes**.

Vector

Es un objeto matemático y físico que posee **magnitud** (tamaño, longitud o módulo) y **dirección**.



El Gradiente

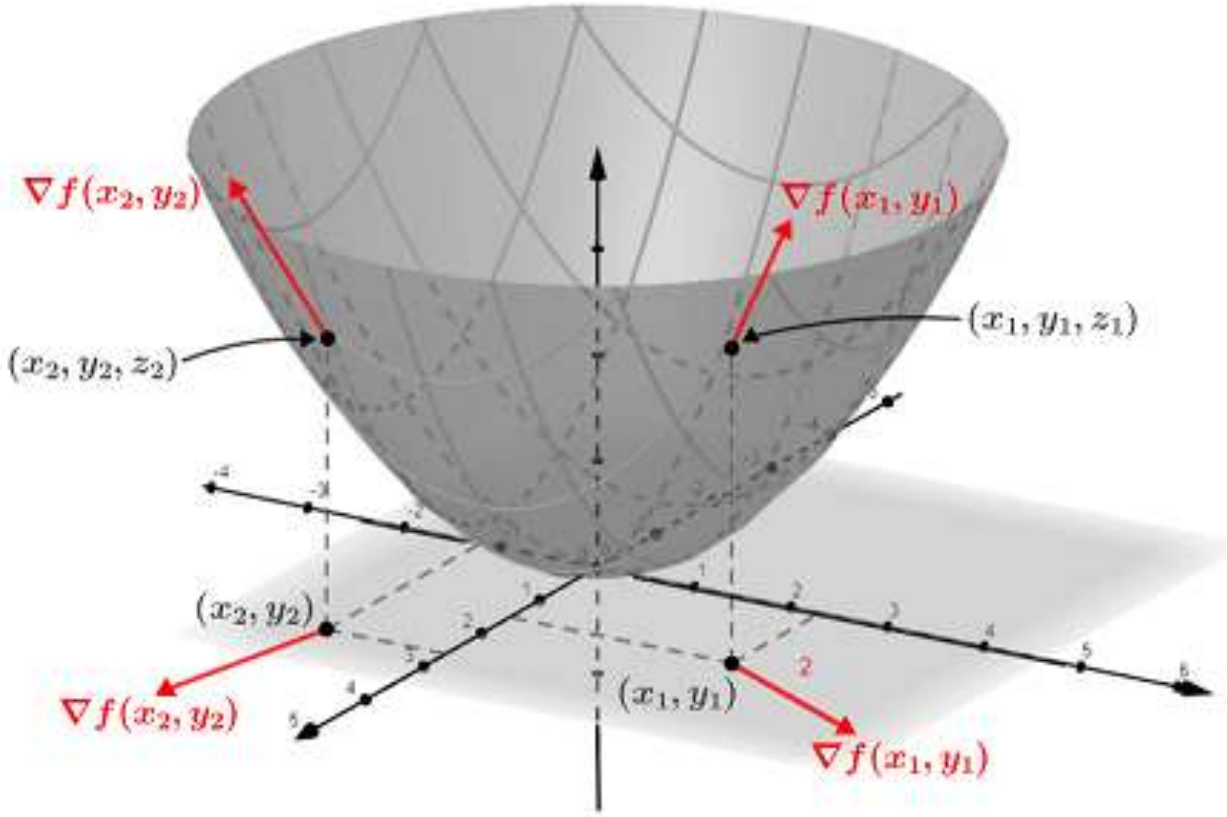
El vector gradiente de *f* evaluado en un punto genérico $\boldsymbol{x}$ del dominio de *f* **indica la dirección** en la cual el campo *f* **varía más rápidamente** y su **módulo representa el ritmo de variación** de *f* en la dirección de dicho vector gradiente.

$$\Delta f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$$

El Gradiente

En matemáticas, el “gradiente” es una **generalización multivariable** de la **derivada**. Mientras que una derivada se puede definir solo en funciones de una sola variable, para funciones de varias variables, el gradiente toma su lugar. El gradiente es una **función de valor vectorial**, a diferencia de una derivada, que es una función de valor escalar.

El Vector gradiente

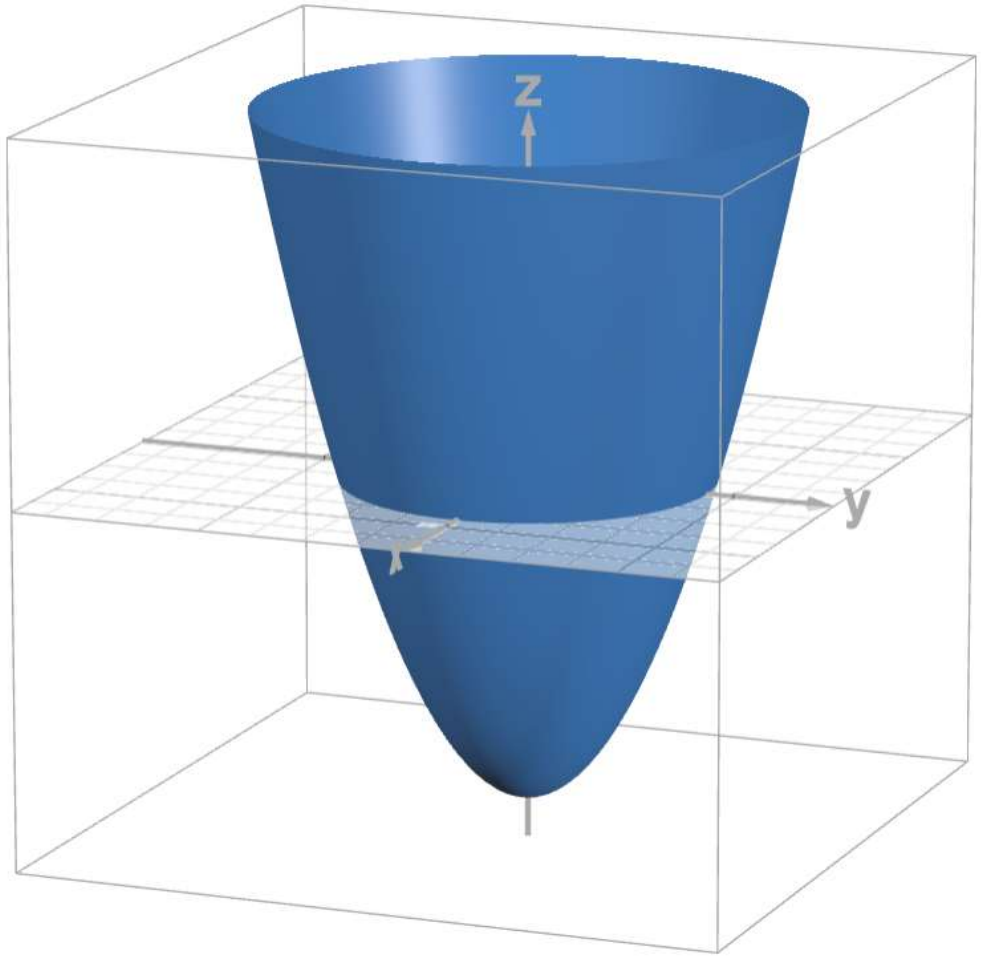


Cálculo del Descenso del gradiente

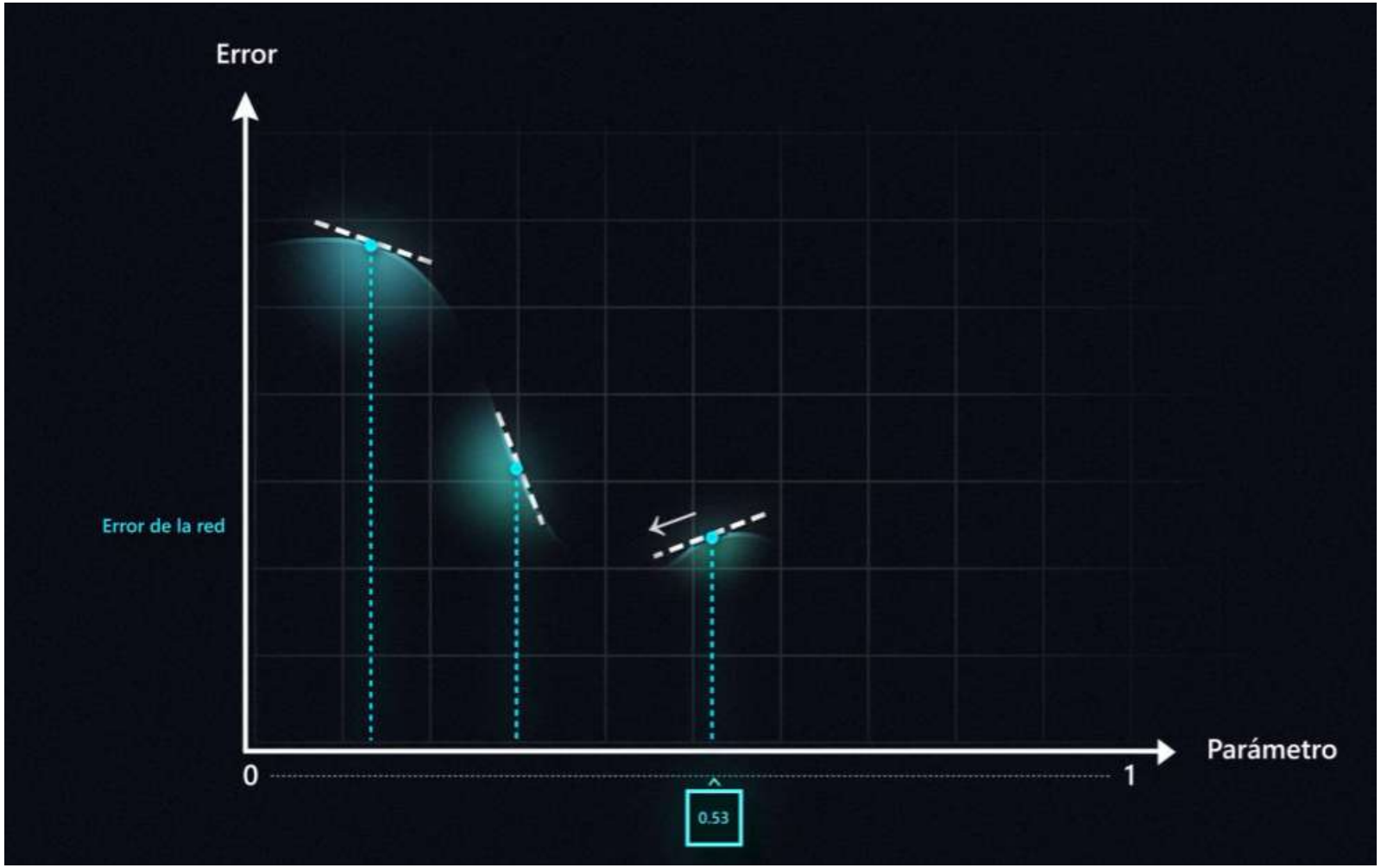
El gradiente descendiente calcula iterativamente el siguiente punto utilizando el gradiente del punto actual. Este punto es escalado con una **razón de aprendizaje γ**. Se tiene entonces que el gradiente descendiente utiliza la siguiente expresión:

$$p_{n+1} = p_n - \gamma \cdot \Delta f(p_n)$$

Ejemplo



$$z = x^2 + y^2 - 3$$



Ejemplo

- Mirar ejemplo en Excel y en desmos
- Crear el código en Python

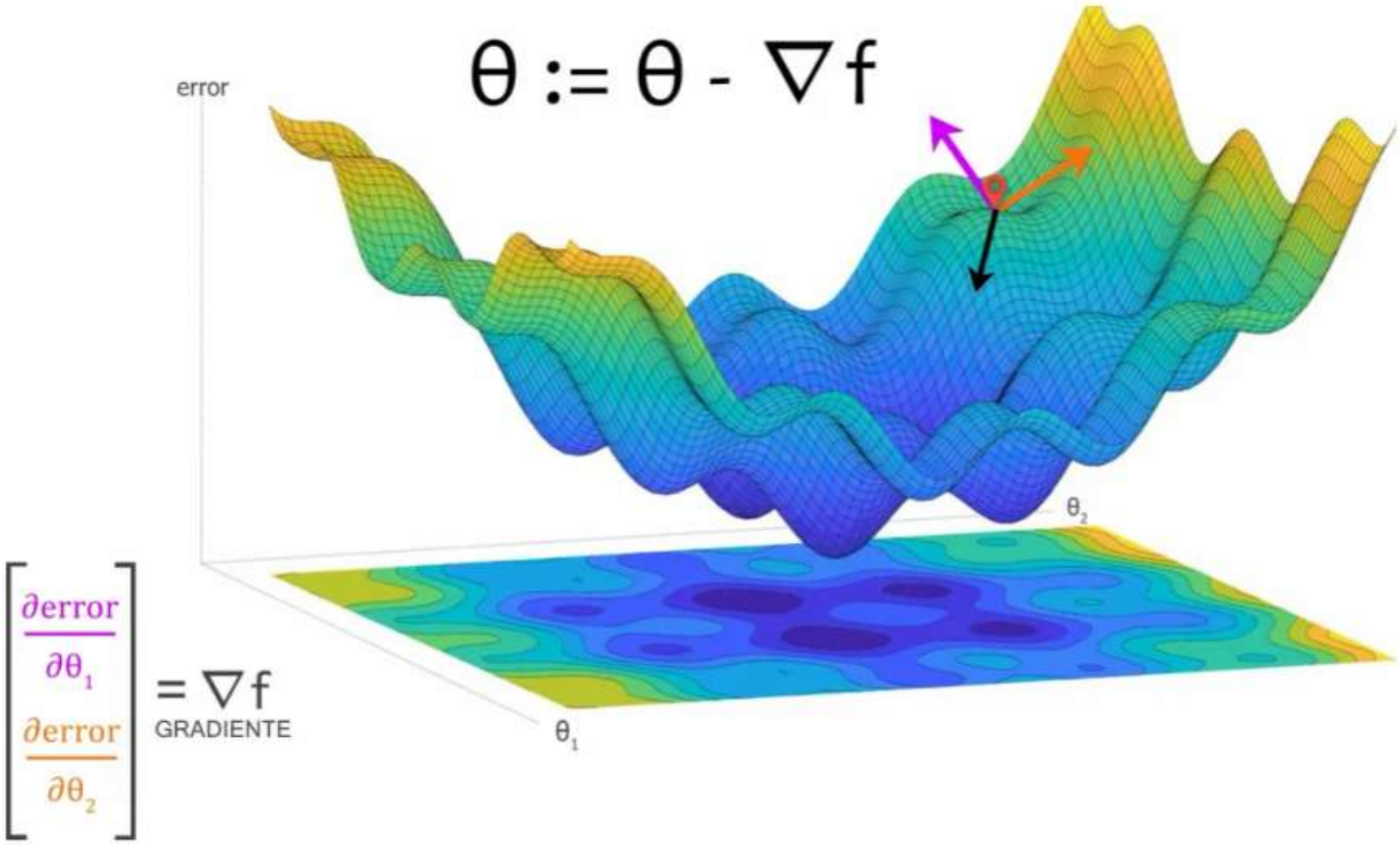
Descenso del gradiente

En el backpropagation para entrenar redes neuronales

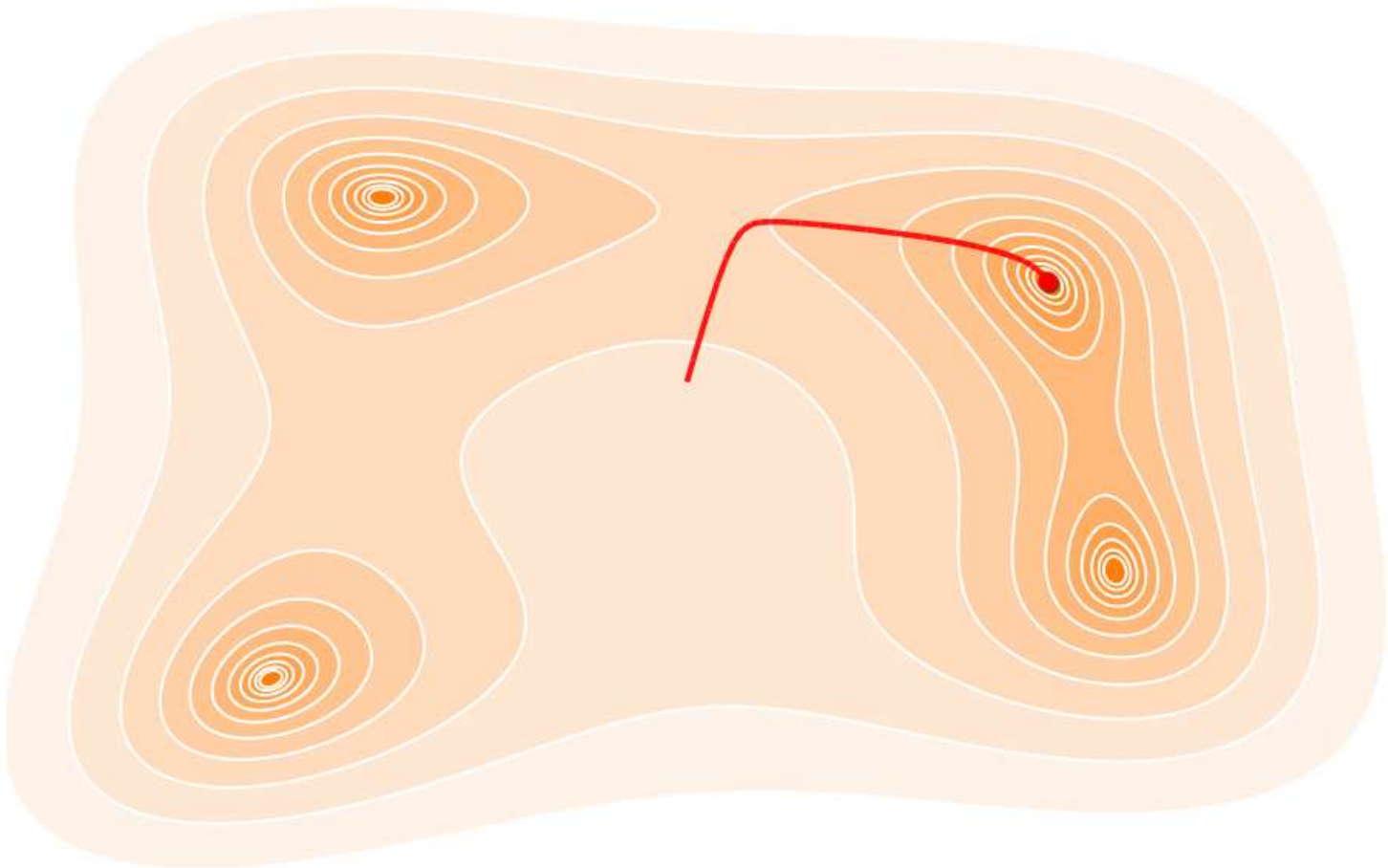
Descenso del gradiente

Aplicado a redes neuronales permite ajustar los **pesos y sesgos** de manera gradual para **reducir el error** que comete la red al hacer predicciones.

Descenso del gradiente



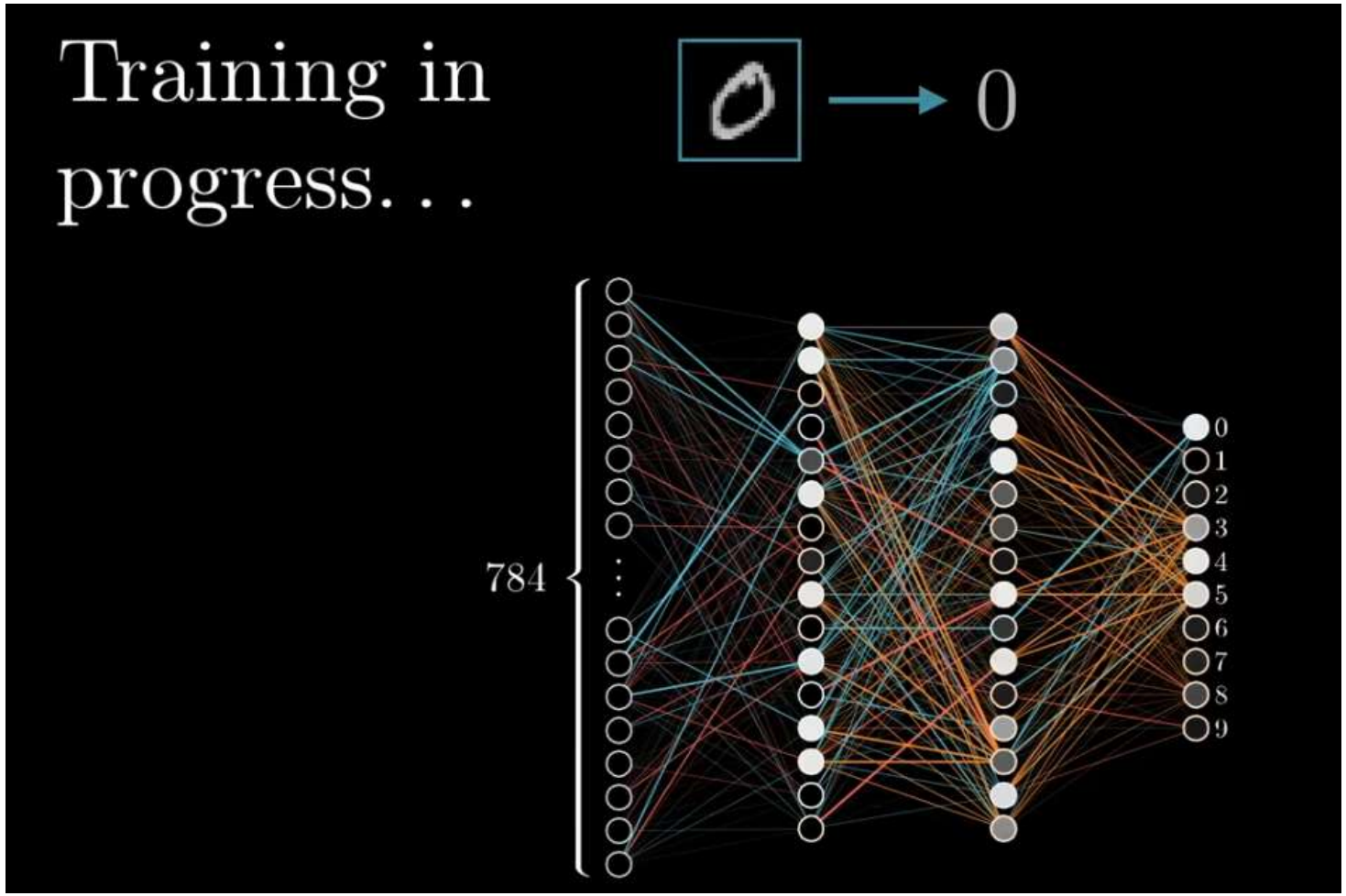
$f(x,y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$



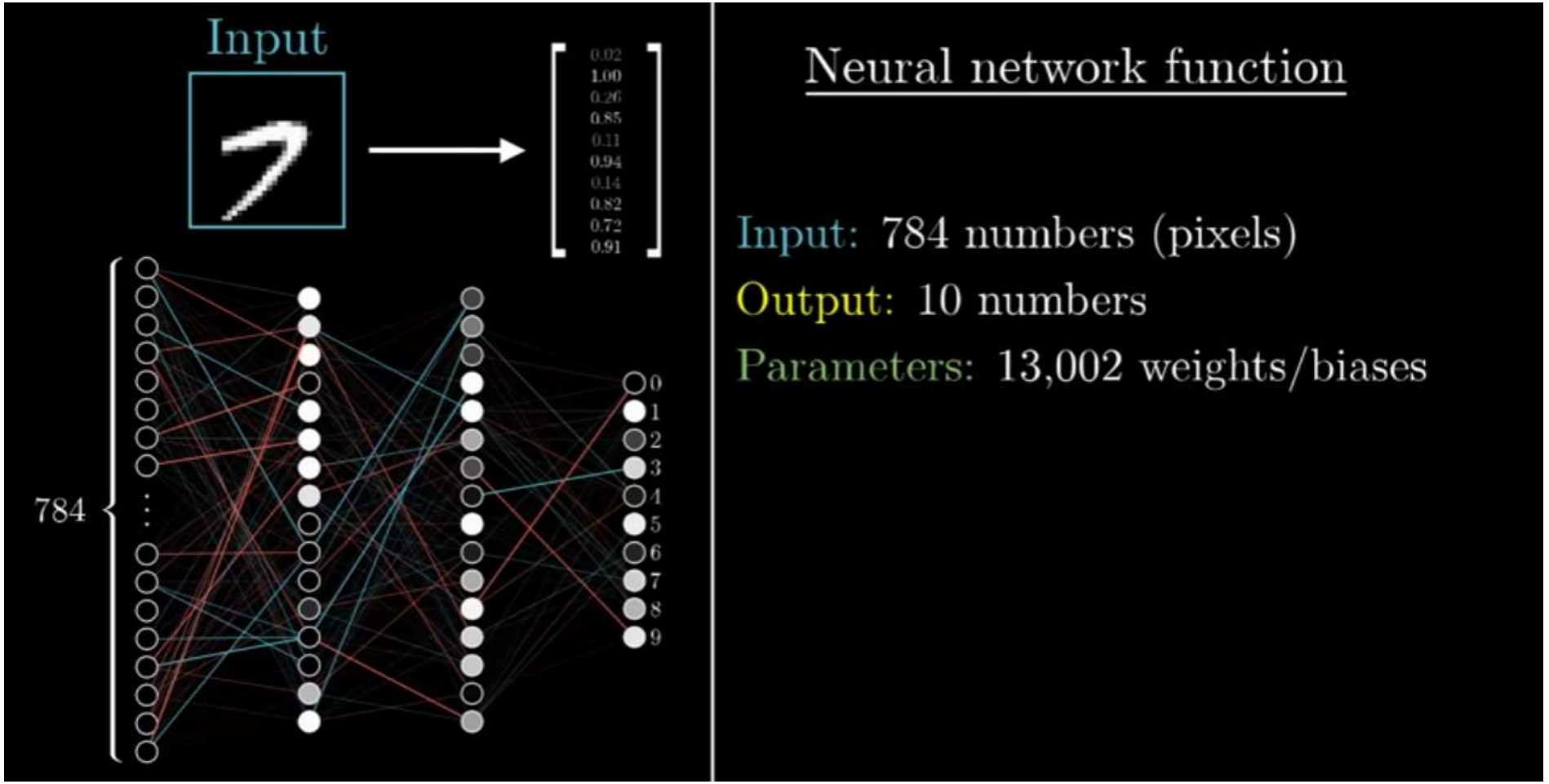
Learning Rate α = 0.0075

.0001 0.001 0.01 0.1 1

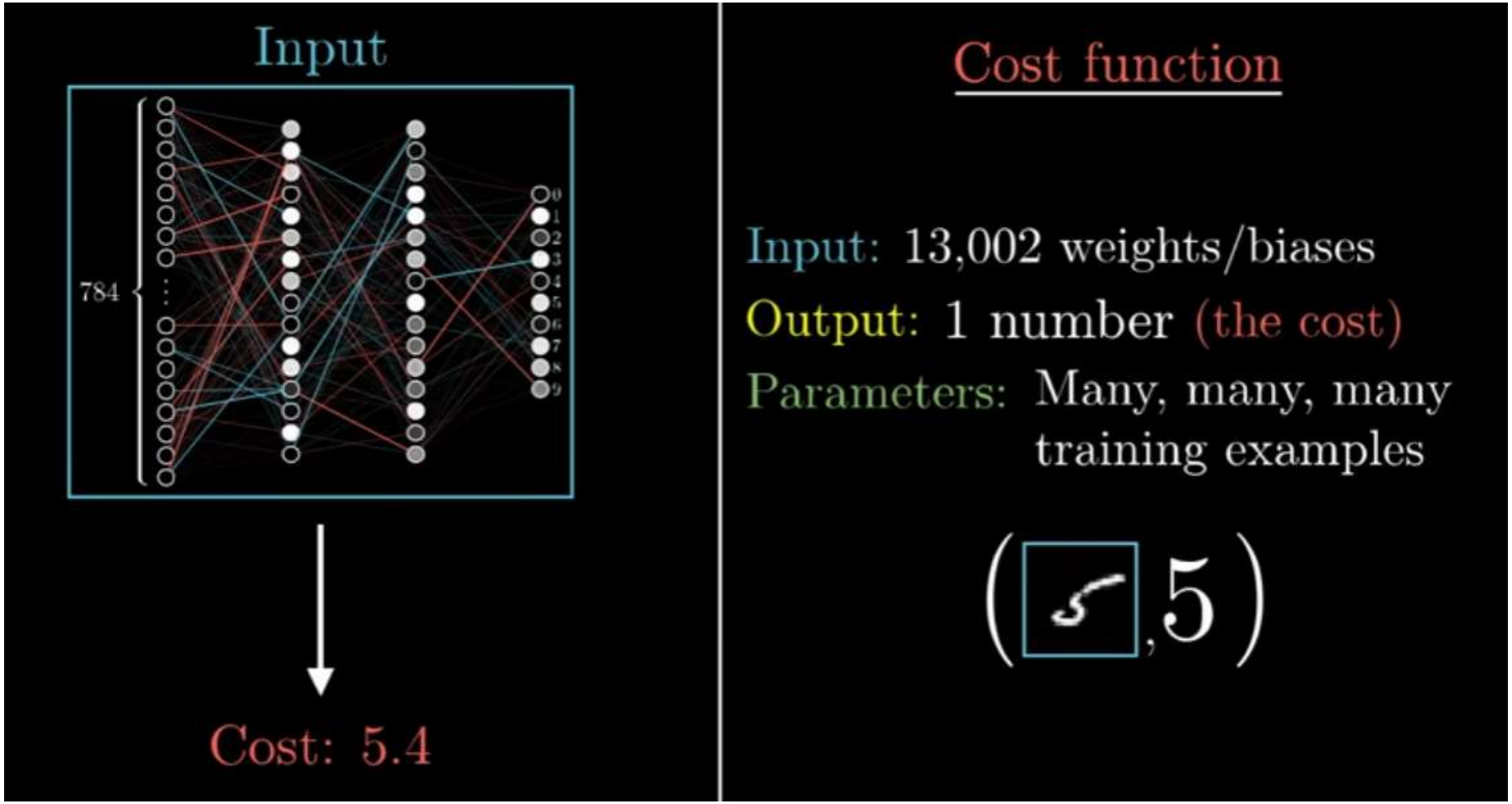
Descenso del gradiente



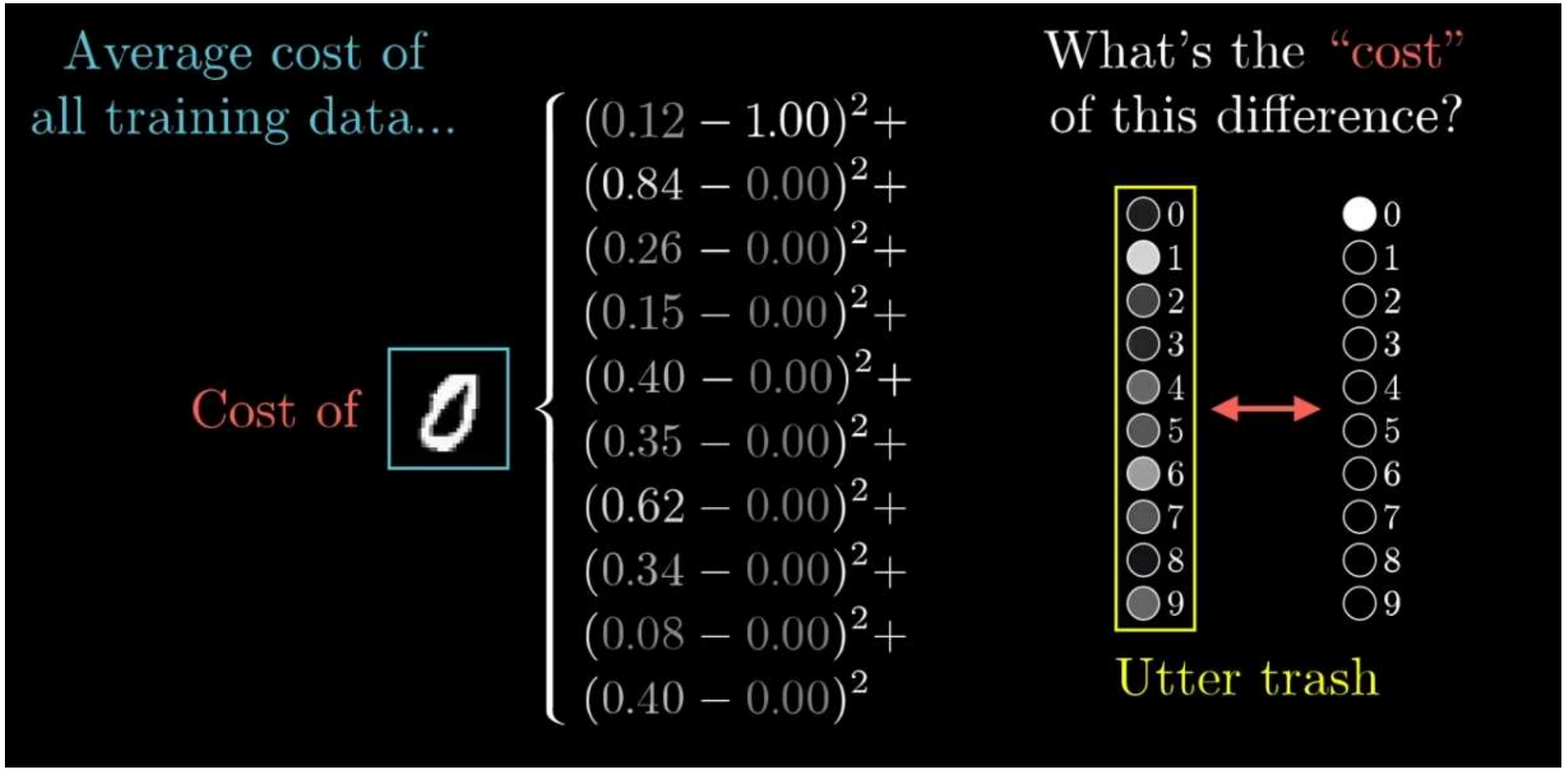
Descenso del gradiente



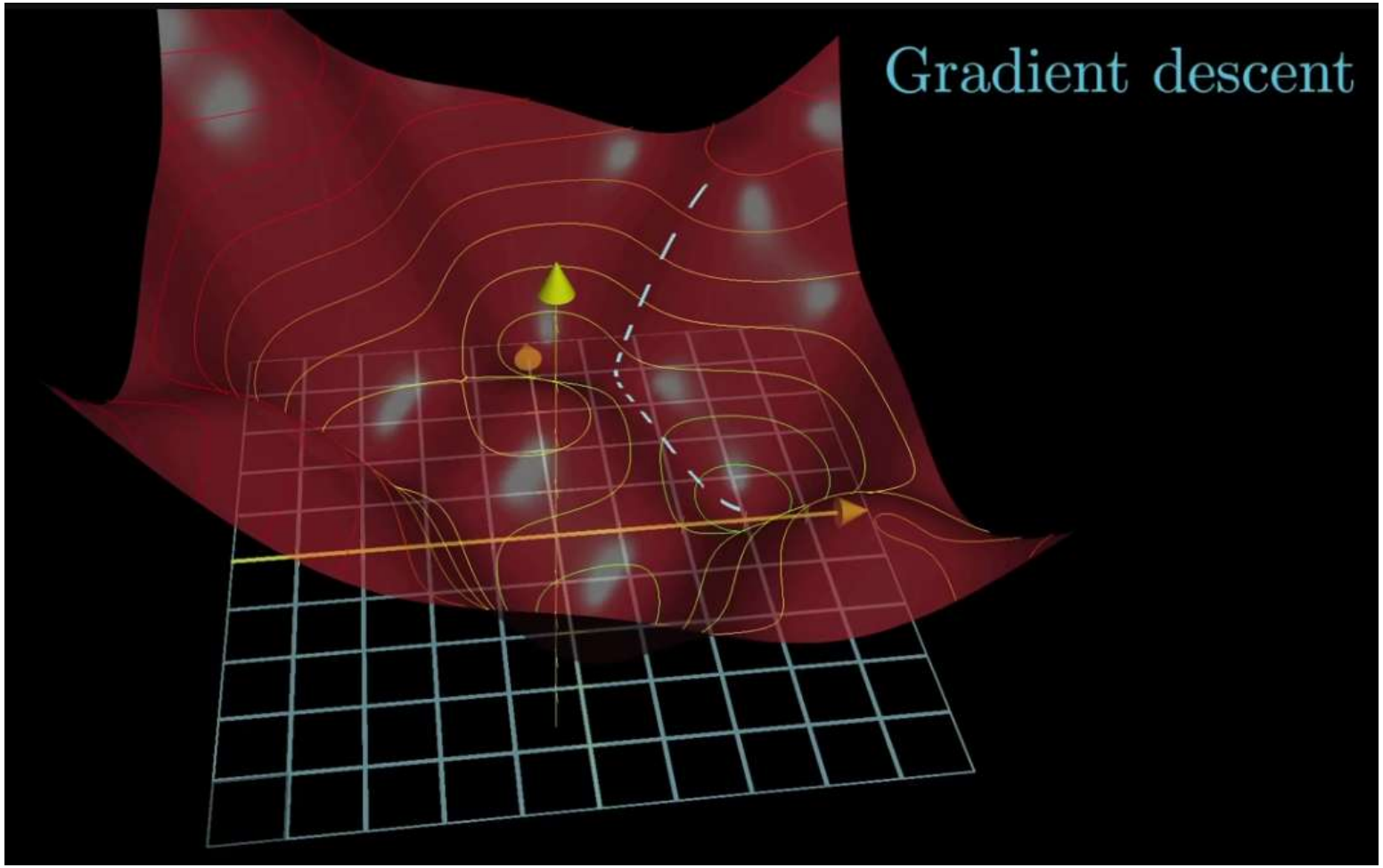
Descenso del gradiente



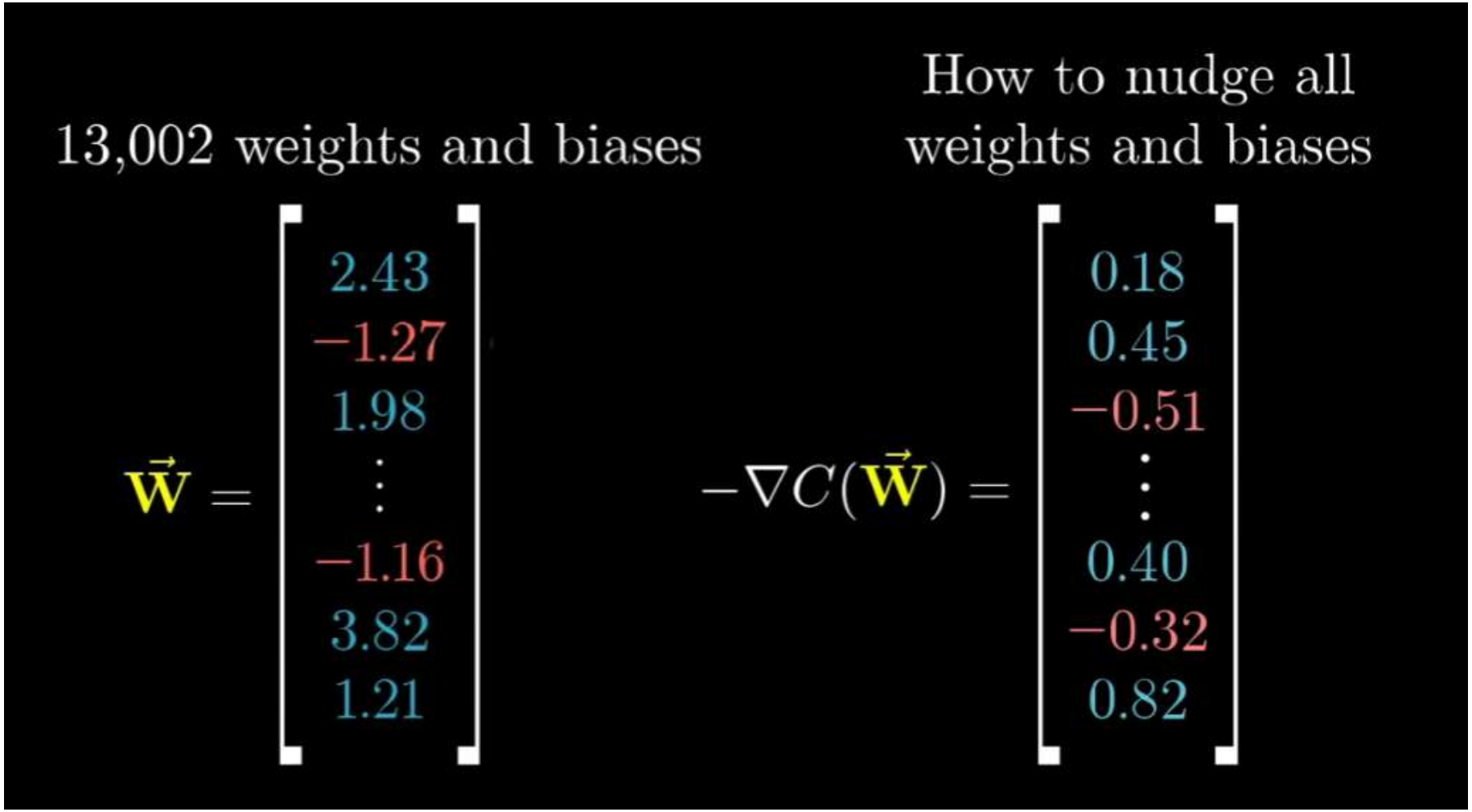
Descenso del gradiente



Descenso del gradiente



Descenso del gradiente



La Retropropagación

1. Propagación Hacia Adelante (Forward Pass)
   - La red toma datos de entrada, los procesa capa por capa aplicando funciones de activación, y produce una salida (predicción).
2. Propagación Hacia Atrás (Backward Pass)
   - Cálculo del Error: Se compara la predicción con el valor real usando una función de pérdida para cuantificar el error.
   - Cálculo de Gradientes: El error se propaga hacia atrás, calculando cómo cada peso y sesgo contribuye a ese error (usando la **regla de la cadena**).
   - Actualización de Parámetros: Los pesos y sesgos se ajustan en la dirección opuesta al gradiente (mediante descenso de gradiente y una tasa de aprendizaje) para disminuir el error.

Este ciclo se repite iterativamente (en épocas) para que la red aprenda y mejore su precisión.

La Función de coste

La función de coste en el algoritmo de retropropagación es una **medida del error** entre lo que la red neuronal predice y lo que debería predecir. Su objetivo es **cuantificar qué tan mal se está desempeñando la red**. La retropropagación utiliza esta medida de error para **ajustar los pesos y sesgos de la red** de forma iterativa, buscando **minimizar ese error** y así mejorar la precisión del modelo.

Función compuesta

Una **función compuesta** es una función que se forma al aplicar una función al resultado de otra función. En otras palabras, es como "encadenar" dos o más funciones, donde la salida de una función se convierte en la entrada de la siguiente.

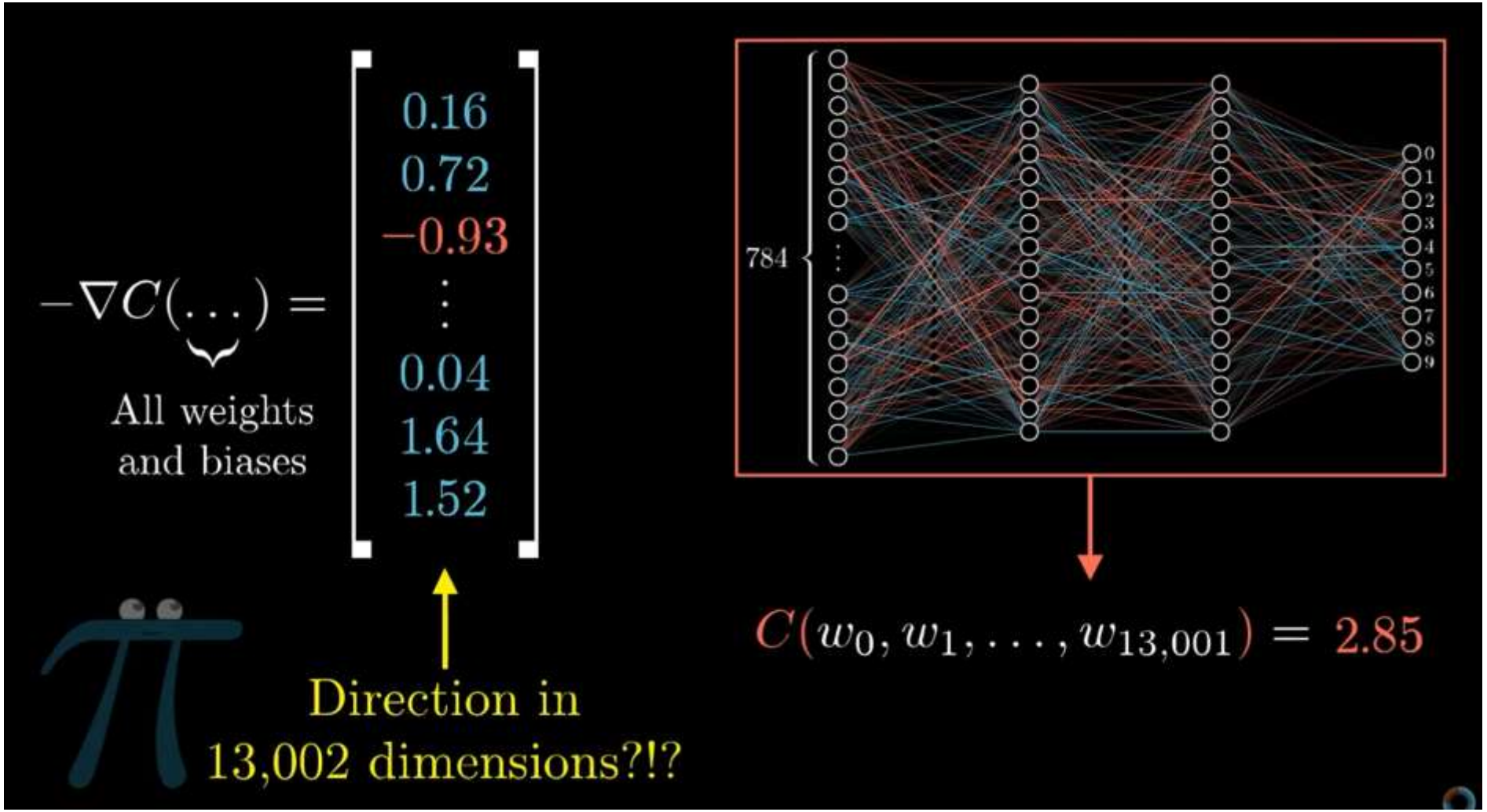
$$g(f(x))$$

La Regla de la cadena

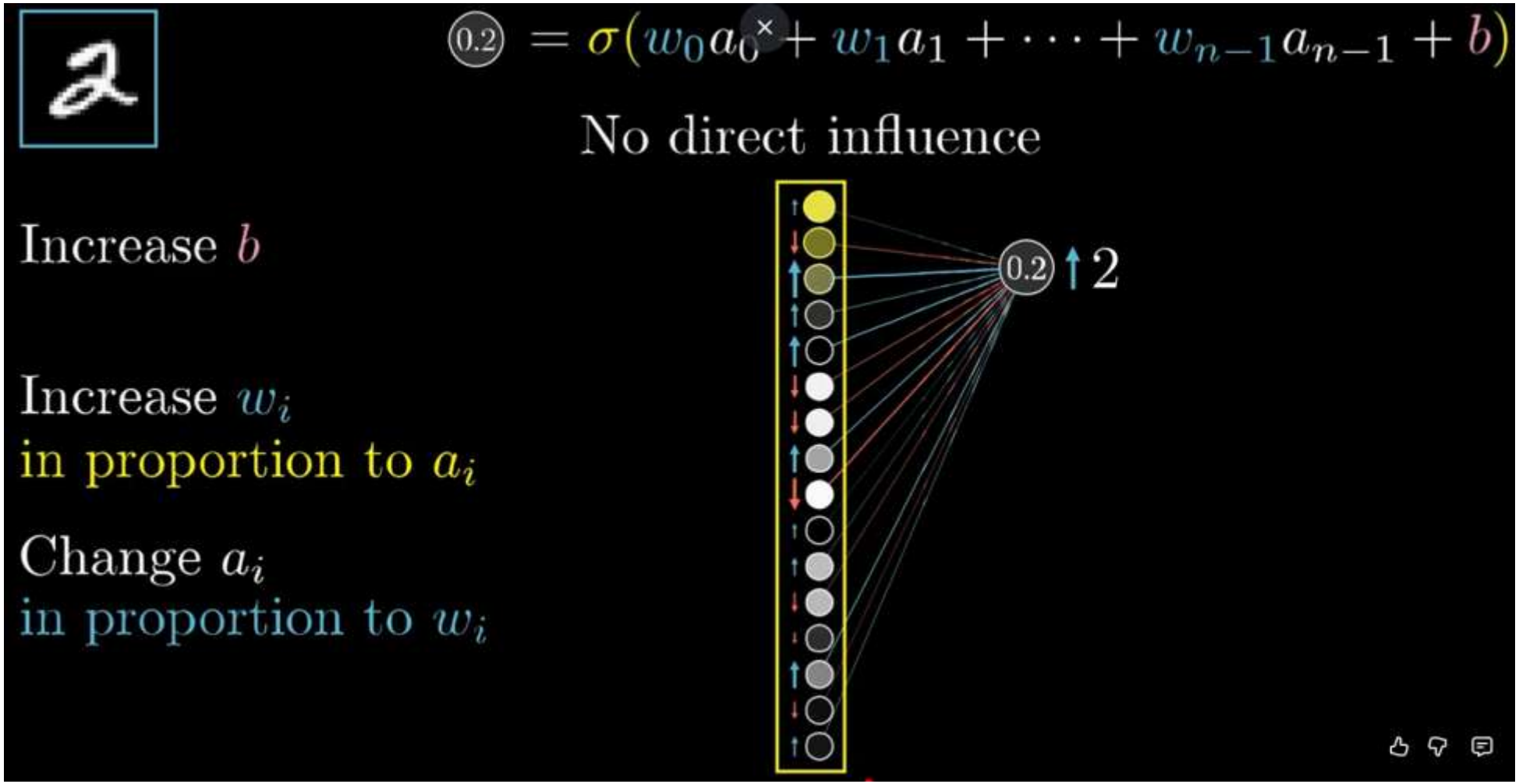
- Es una fórmula explícita de la derivada de una función compuesta por dos funciones derivables.
- Esta regla permite conocer la *j-ésima* derivada parcial de la *i-ésima* aplicación parcial de la composición de dos funciones de varias variables. Esquemáticamente, si una variable y *depende de una segunda variable u* , la cual depende de una variable x , la tasa de cambio de y respecto a x se calcula como el producto de la tasa de cambio de y respecto a *u* y de la tasa de cambio de *u* respecto a x , esto es:

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{\mathrm{d}y}{\mathrm{d}u} \cdot \frac{\mathrm{d}u}{\mathrm{d}x}$$

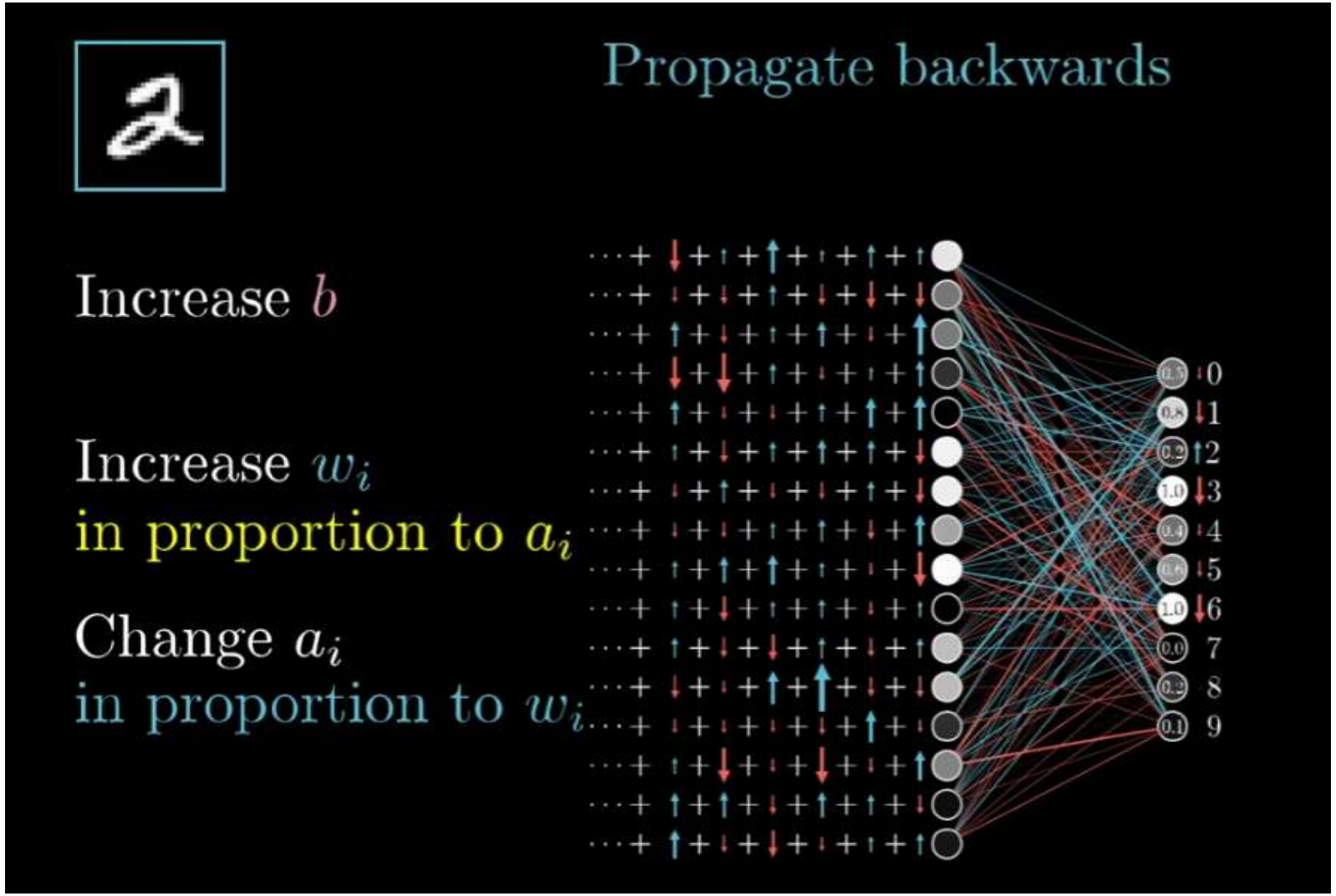
Retropropagación



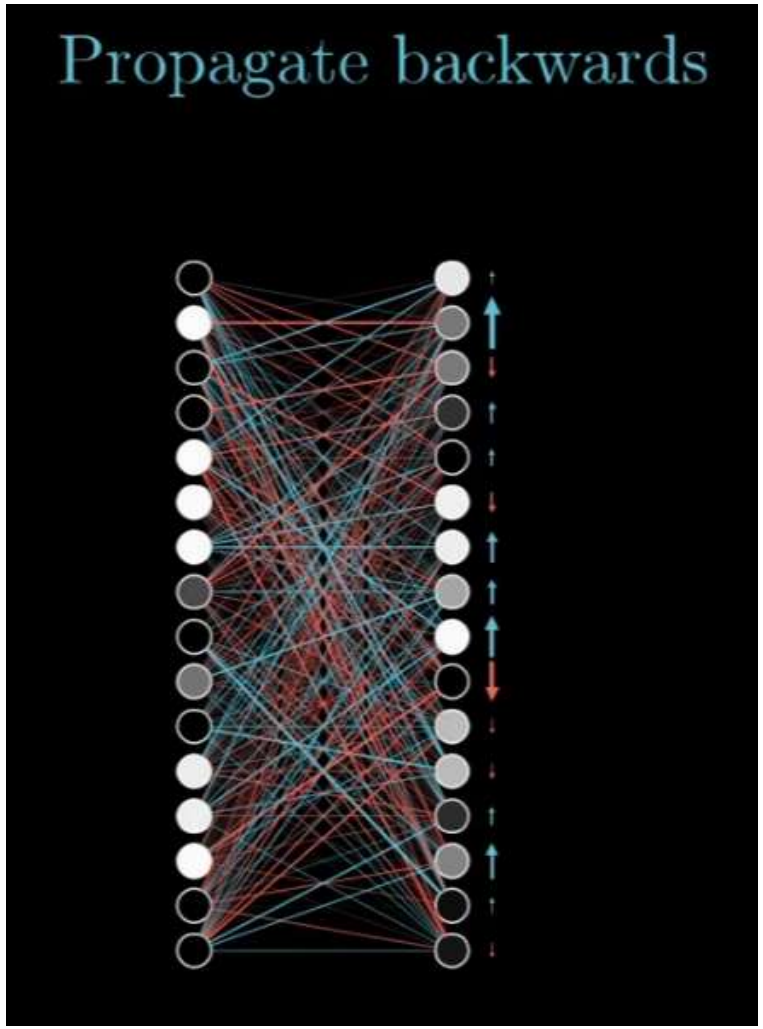
Retropropagación



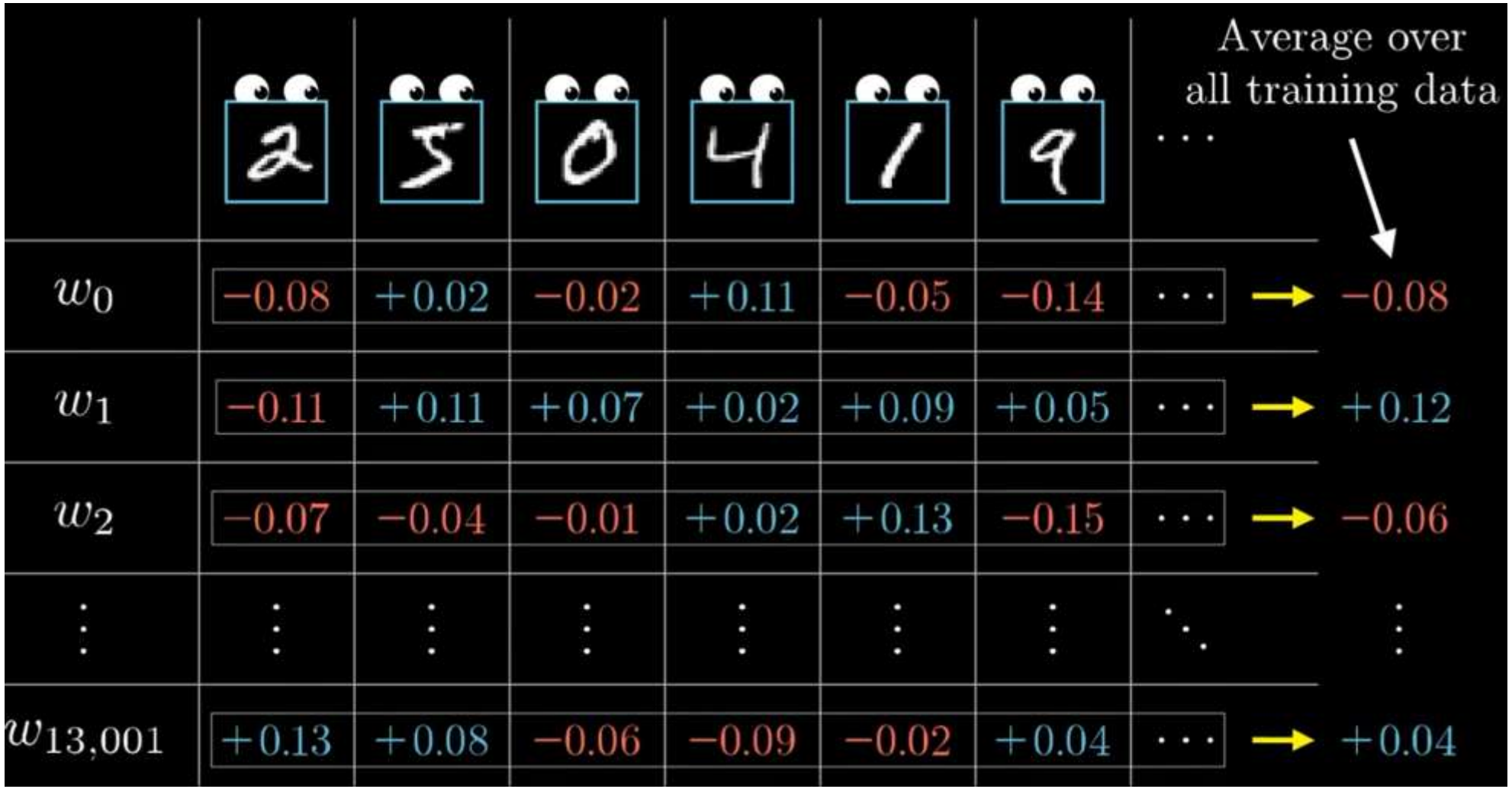
Retropropagación



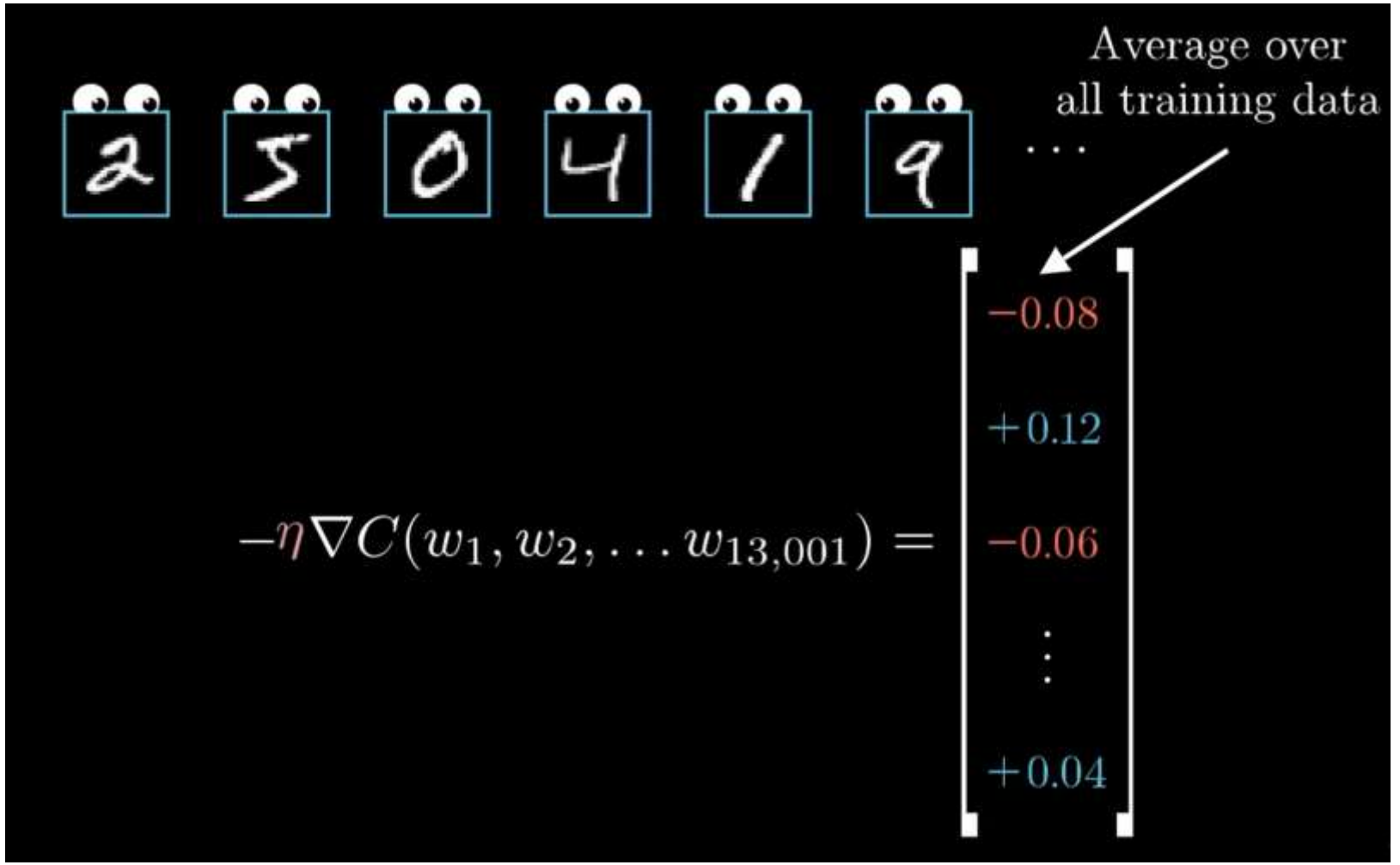
Retropropagación



Retropropagación



Retropropagación



Las fórmulas de la retropropagación

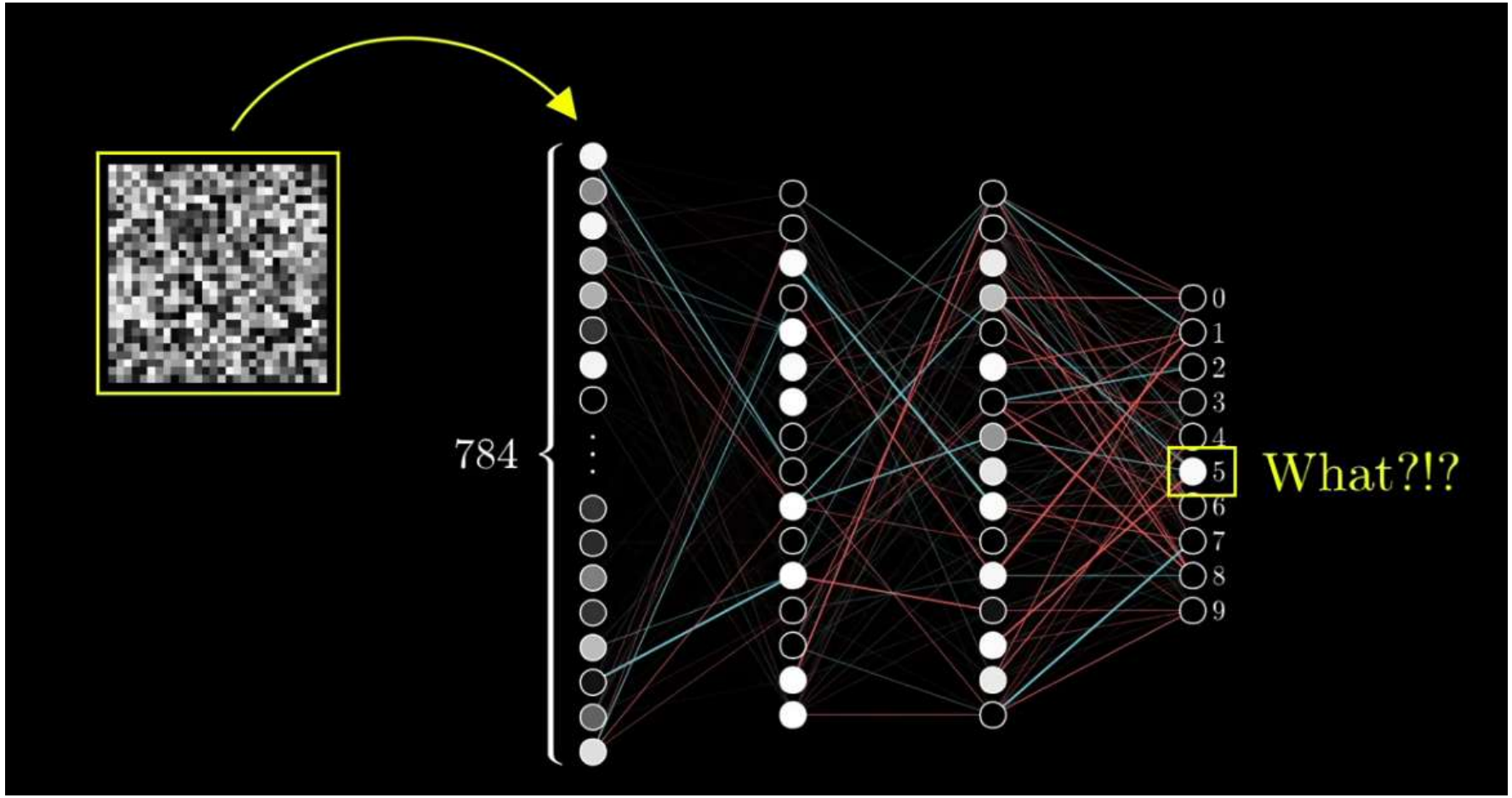
$$\Delta w = -k\frac{\partial C}{\partial w} \quad \frac{\partial C}{\partial w} = \frac{\partial C}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w} \quad \Delta b = -k\frac{\partial C}{\partial b} \quad \frac{\partial C}{\partial b} = \frac{\partial C}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial b}$$

$$\frac{\partial C}{\partial a} = \underbrace{2\,(a - y)}_{amount\ of\ error} \qquad \frac{\partial a}{\partial z} = \underbrace{\sigma'(z) = \frac{e^z}{(1 + e^z)^2}}_{slope\ of\ activation}$$

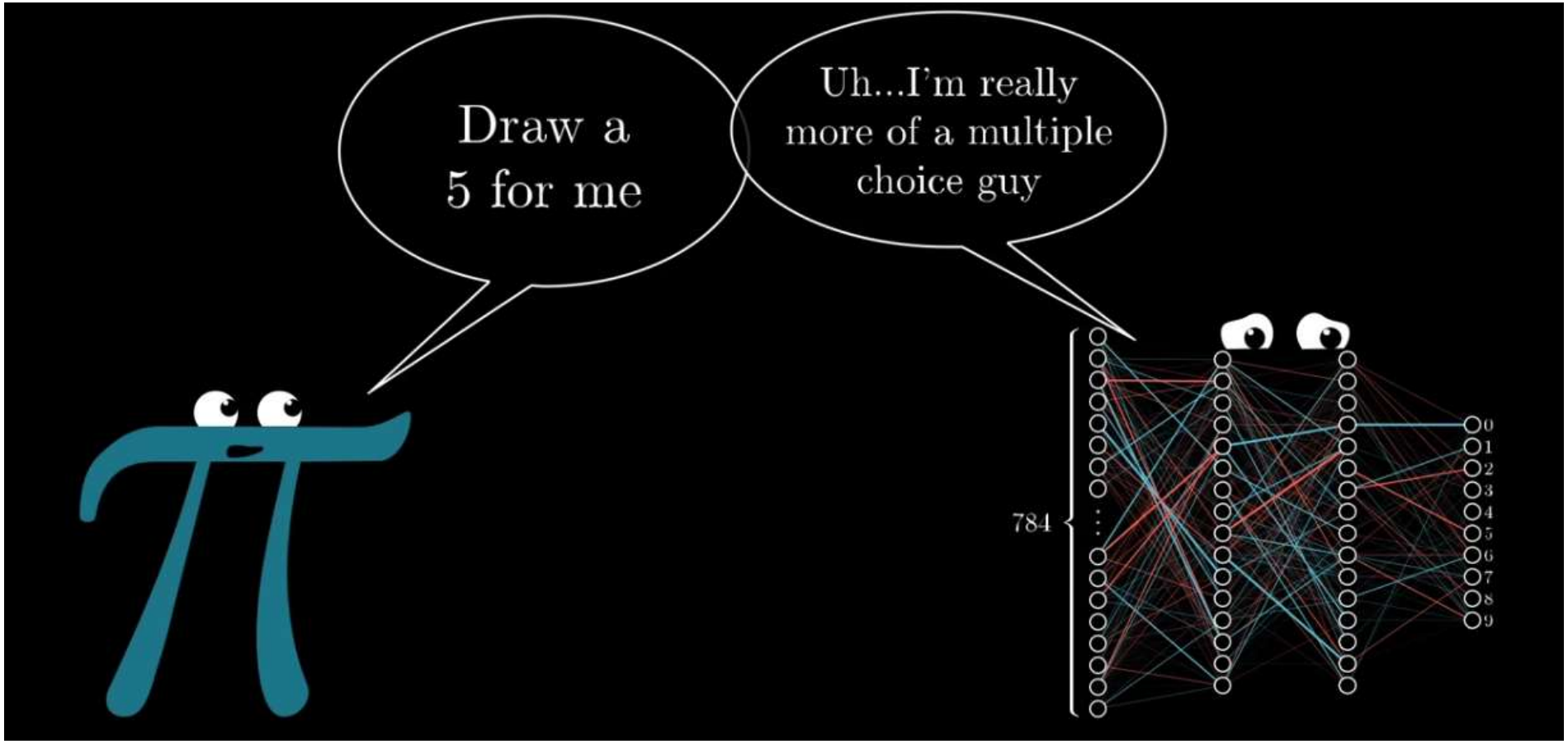
$$\frac{\partial z}{\partial w} = \underbrace{x}_{input} \qquad \frac{\partial z}{\partial b} = 1$$

Problemas en las redes neuronales

Ataques



Limitaciones



Problemas en el reconocimiento de imágenes

1) Destrucción de la Estructura Espacial

- **Problema:** Un MLP requiere que la entrada sea un **vector 1D**. Para alimentar una imagen a un MLP, primero se debe **aplanar** (o vectorizar).
- **Consecuencia:** Se **pierde la información crucial de la vecindad local** (la relación espacial entre un píxel y sus vecinos inmediatos). Esta relación es esencial para identificar características visuales como bordes, líneas y texturas. El MLP trata a cada píxel como una característica independiente y desordenada.

2) **Explosión de Parámetros**

- **Problema:** Un MLP utiliza **conexión total** (*fully connected*).
- **Consecuencia**: El número de parámetros (pesos) crece exponencialmente, especialmente con imágenes de alta resolución. Ejemplo: Una imagen pequeña de 100 x 100 x 3 (color) tiene 30.000 píxeles de entrada. Si la primera capa oculta tiene solo 1.000 neuronas, ya hay 30 millones de pesos que aprender solo en esa capa.

Problemas en el reconocimiento de imágenes

3) Ausencia de Invariancia Traslacional

- **Problema**: Los pesos en un MLP están ligados a posiciones específicas en el vector de entrada.
- **Consecuencia**: Si la red aprende a reconocer un objeto (ej. un círculo) en la esquina superior izquierda, tendrá que aprenderlo de nuevo con un conjunto de pesos completamente diferente si el círculo aparece en la esquina inferior derecha.

Ejemplo

- Abrir el archivo Red neuronal.ipynb
- Hacer cambios en los hiper-parámetros de la red para mejorar la precisión.
  - Tasa de Aprendizaje
  - Número de Épocas
  - Tamaño del Lote
- Modernización de Componentes Clave
  - Optimizador: Reemplace el optimizador SGD por alternativas modernas como Adam o RMSprop.
  - Activación: Cambie la función de activación de las capas ocultas de Sigmoide a ReLU
  - Función de Costo: (Si usó MSE inicialmente) Cambie a la Entropía Cruzada Categórica.

Ejemplo

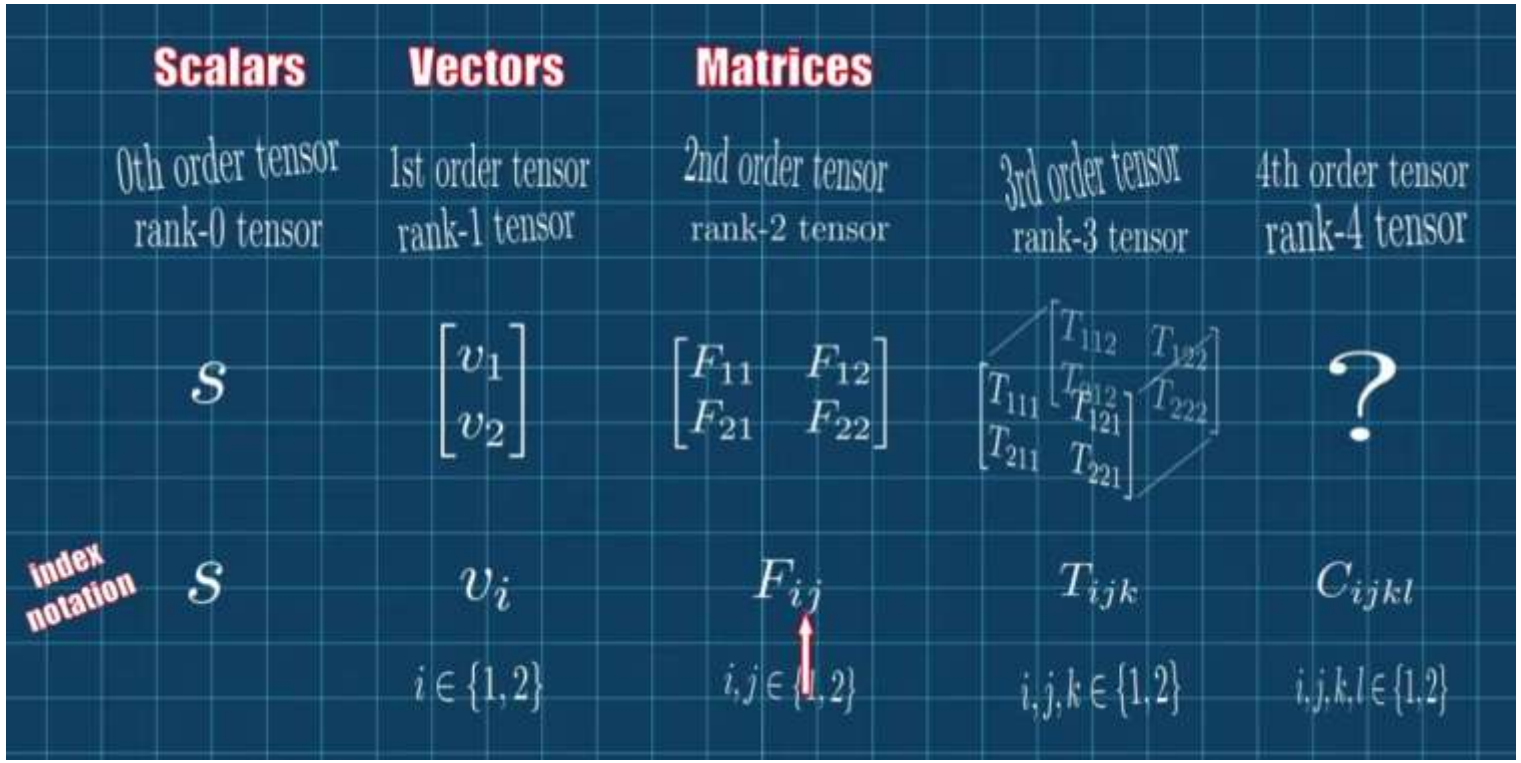
- Modificaciones Arquitectónicas
  - Profundidad y Ancho: Experimente con la topología de la red. Pruebe añadiendo una o dos capas ocultas adicionales, o ajustando el número de neuronas en cada capa (ej. de 16 a 64).

Ejercicios

- Clasificación Binaria: clasificar solo si la imagen es el dígito “5” o “no es el 5”
- Detección par/impar: Entrena la red para clasificar si un dígito es par o impar.
- Detección de Números Grandes/Pequeños:Problema: Clasifica los dígitos en dos grupos: "pequeños" (0, 1, 2, 3, 4) o "grandes" (5, 6, 7, 8, 9).

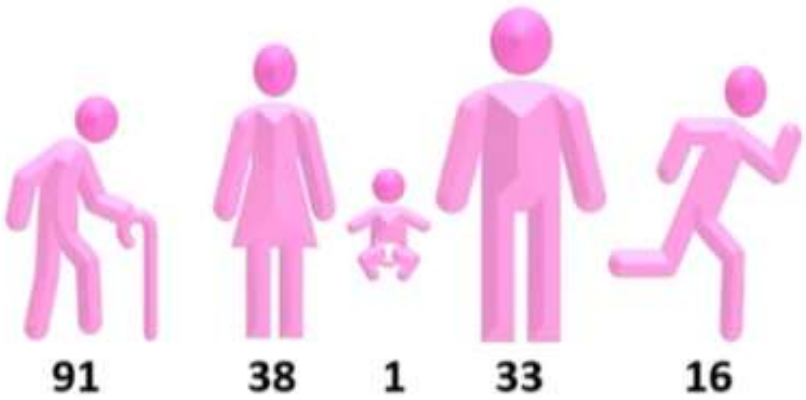
Qué es un Tensor

Es una **generalización** de los conceptos de **escalar** (tensor de orden 0), **vector** (tensor de orden 1) y **matriz** (tensor de orden 2) a **cualquier número de dimensiones** u **orden** (*rank*).



La entropía: una medida de la incertidumbre

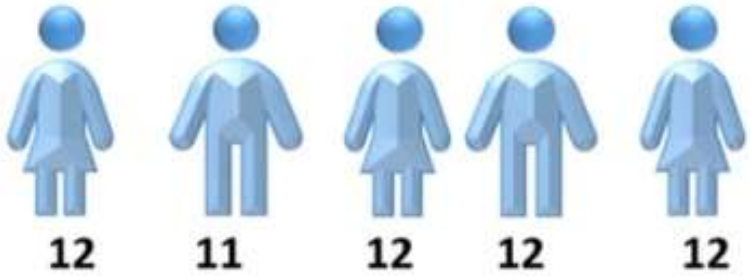
Edad
población
general



ALTA

Incertidumbre

Edad
niños de
6° de primaria



BAJA

Incertidumbre

CNN

Redes Neuronales Convolucionales

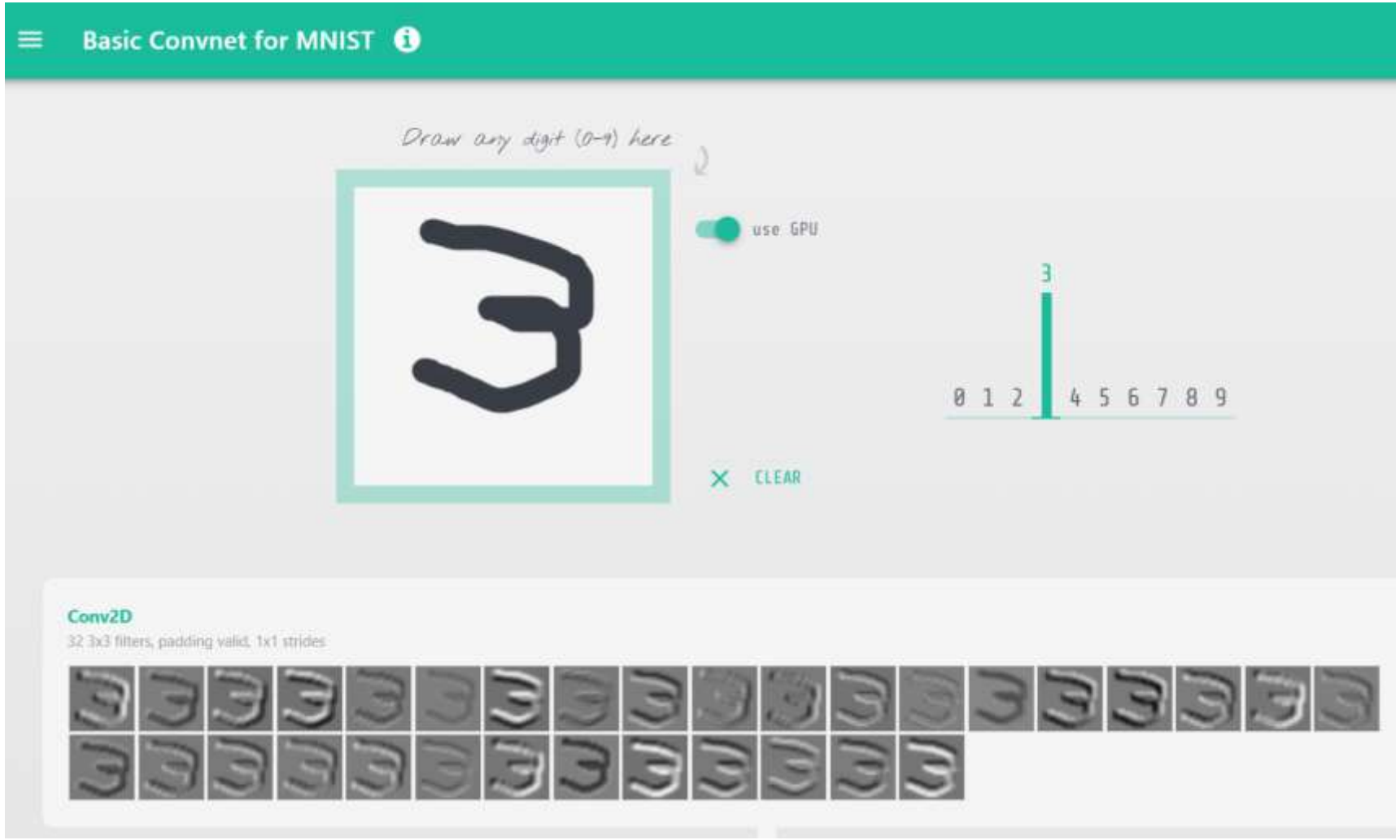
Temario

- Ventajas de una CNN
- Demo
- Breve historia
- Definición
- Arquitectura
- Funcionamiento de cada parte
- Ejemplo
- Ejercicio

Definición

Una Red Neuronal Convolucional (CNN), del inglés *Convolutional Neural Network*, es un tipo de red neuronal artificial de aprendizaje profundo (Deep Learning) diseñada específicamente para procesar datos que tienen una topología similar a una cuadrícula, como **imágenes**, donde las **relaciones espaciales** entre los datos son cruciales.

Demo



Ventajas de una CNN

A. Compartición de Pesos

- **Mecanismo:** El **kernel,** se desliza y se aplica a toda la imagen de entrada.
- **Beneficio**: Un **único conjunto de pesos** se reutiliza para escanear miles de ubicaciones. Esto reduce drásticamente el número total de parámetros que el modelo debe aprender.

B. Extracción de Características Locales (Estructura Espacial)

- **Mecanismo**: La operación de convolución garantiza que cada neurona de la capa convolucional solo esté conectada a una **pequeña región** local de la imagen de entrada (el llamado campo receptivo).
- **Beneficio**: La red aprende primero las **relaciones entre píxeles vecinos** (características locales como bordes o líneas) y **luego las combina** en capas posteriores para formar patrones complejos (jerarquía de características).

C. Invariancia Traslacional

- Gracias a la compartición de pesos y a las capas de pooling, la CNN puede reconocer un patrón **independientemente** de su ubicación exacta en la imagen.

Breve historia

La historia de las Redes Neuronales Convolucionales (CNN) se extiende por varias décadas, fusionando la neurociencia con la informática

1) Raíces Biológicas (Décadas de 1950 y 1960)

El concepto fundamental de las CNNs se inspira en el funcionamiento de la corteza visual de los mamíferos:

- **1959 - Descubrimiento Clave:** Neurofisiólogos publicaron un trabajo sobre el cerebro de los gatos. Descubrieron que las neuronas en la corteza visual responden solo a **estímulos en una región específica** del campo visual (el *campo receptivo*) y que existen dos tipos principales de células:
  - **Células Simples:** Responden a orientaciones específicas (bordes, líneas).
  - **Células Complejas:** Responden a las mismas características, pero con cierta **invarianza a la posición** (el objeto aún se detecta aunque se mueva ligeramente)

Breve historia

2) Modelos Pioneros (Década de 1980)

Los hallazgos biológicos se tradujeron en modelos matemáticos iniciales:

- **1980 - El Neocognitrón: Kunihiko Fukushima** creó el **Neocognitrón**, la primera arquitectura que implementó capas de convolución y *pooling* de forma jerárquica. Este modelo podía reconocer patrones visuales a pesar de ligeras distorsiones, pero **no utilizaba el algoritmo de retropropagación** para el entrenamiento.

3) El Nacimiento de la CNN Moderna (Década de 1990)

La introducción de la retropropagación impulsó la primera CNN completamente funcional:

- **1989 - Retropropagación: Yann LeCun** aplicó la retropropagación para entrenar redes convolucionales en el reconocimiento de dígitos. (Vídeo)
- **1998 - LeNet-5:** LeCun y sus colegas presentaron **LeNet-5**, la primera CNN que combinó la **convolución, el *pooling*** y el **entrenamiento por retropropagación** de forma efectiva. Se convirtió en el **modelo base** de las CNNs modernas.

Breve historia

4) El Boom del *Deep Learning* (Década de 2010 en adelante)

A pesar de LeNet, las CNNs permanecieron relativamente inactivas hasta que el aumento de la potencia de cómputo (GPUs) y los datos disponibles permitieron entrenar modelos mucho más profundos:

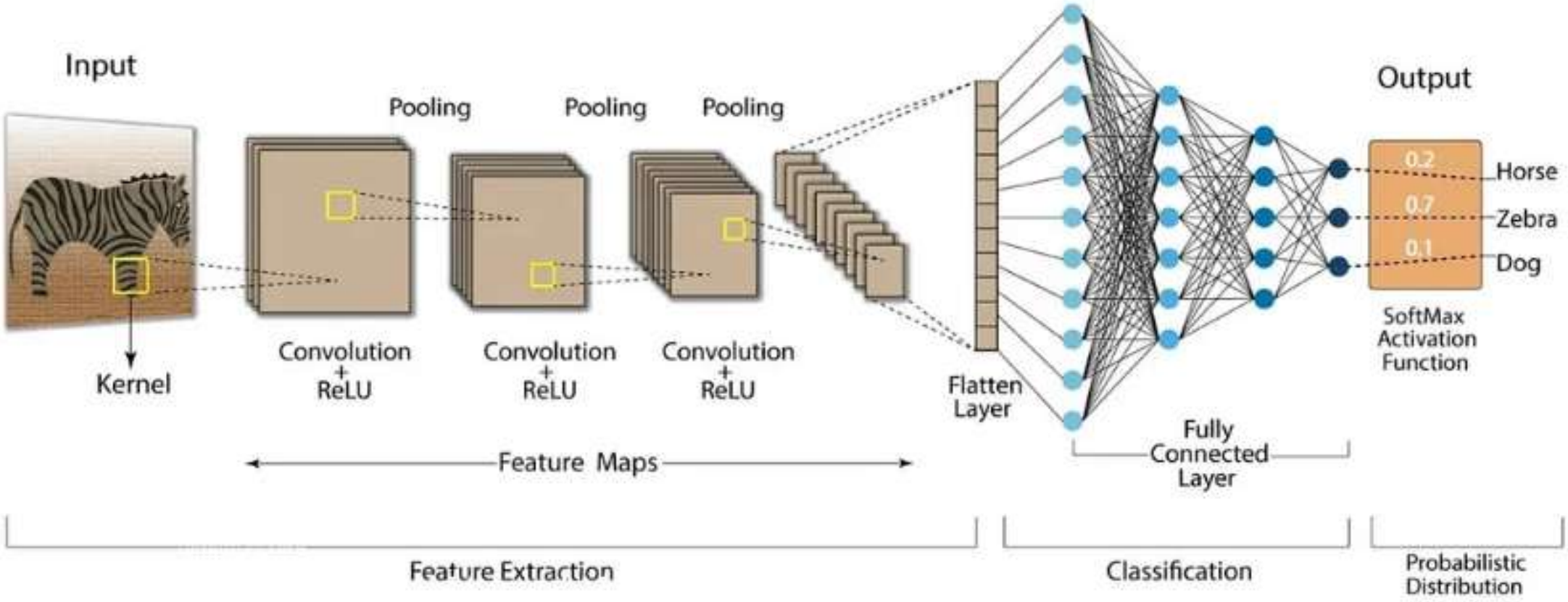
- **2012 - AlexNet y el "Momento ImageNet":** Alex Krizhevsky, Ilya Sutskever y Geoffrey Hinton ganaron el concurso de clasificación de imágenes **ImageNet (ILSVRC)** con **AlexNet**. Esta red era mucho más profunda que sus predecesoras, estaba entrenada en GPUs y utilizaba la función de activación **ReLU**. Su victoria marcó un punto de inflexión, demostrando que las CNNs profundas superaban drásticamente a los métodos de visión artificial tradicionales.
- **Evolución Arquitectónica:** A partir de entonces, se sucedieron arquitecturas cada vez más profundas y eficientes, consolidando a las CNNs como la tecnología dominante en la visión por computadora.

Elementos Distintivos de una CNN

A diferencia de un Perceptrón Multicapa (MLP) que requiere aplanar la entrada, una CNN **preserva la estructura espacial** de los datos mediante capas especializadas:

- Capas Convolucionales (Conv)
  - Aplican pequeños filtros llamados **kernels** que se deslizan sobre la imagen de entrada para detectar patrones locales específicos, como bordes, líneas o texturas.
- Capas de Agrupamiento (Pooling)
  - Reducir el tamaño espacial de los mapas de características generados por la capa convolucional (submuestreo). Aumenta la **invariancia traslacional** y reduce la complejidad computacional.
- Capas Totalmente Conectadas (Dense)
  - Realiza la **clasificación** final de las categorías de la imagen.

Arquitectura



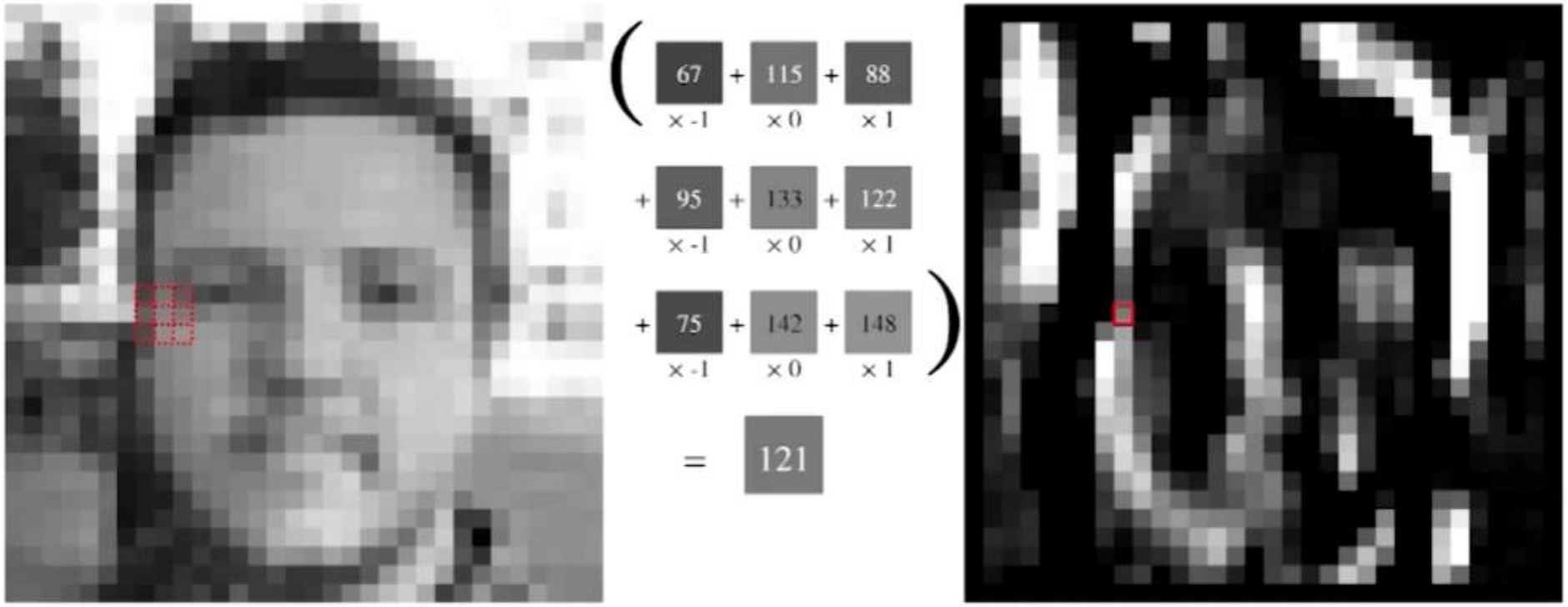
El Kernel

Es una **pequeña matriz de pesos** que se utiliza en la capa convolucional para **extraer características** específicas de la entrada (como bordes o texturas).

Funciones Clave del Kernel

- **Extractor de Características:** Actúa como un detector de patrones. Cuando los valores del kernel coinciden con los valores de un parche de la imagen, produce una **activación alta** en el mapa de características.
- **Pesos Compartidos:** El mismo kernel se aplica a **toda la imagen** mediante un desplazamiento, lo que garantiza que los patrones se reconozcan independientemente de su posición.
- **Parámetro Aprendible:** Los valores dentro de la matriz del kernel se **ajustan automáticamente** durante el entrenamiento mediante la retropropagación para minimizar el error de la red.

El Kernel



FILTRO

-1	0	1
-1	0	1
-1	0	1

La convolución

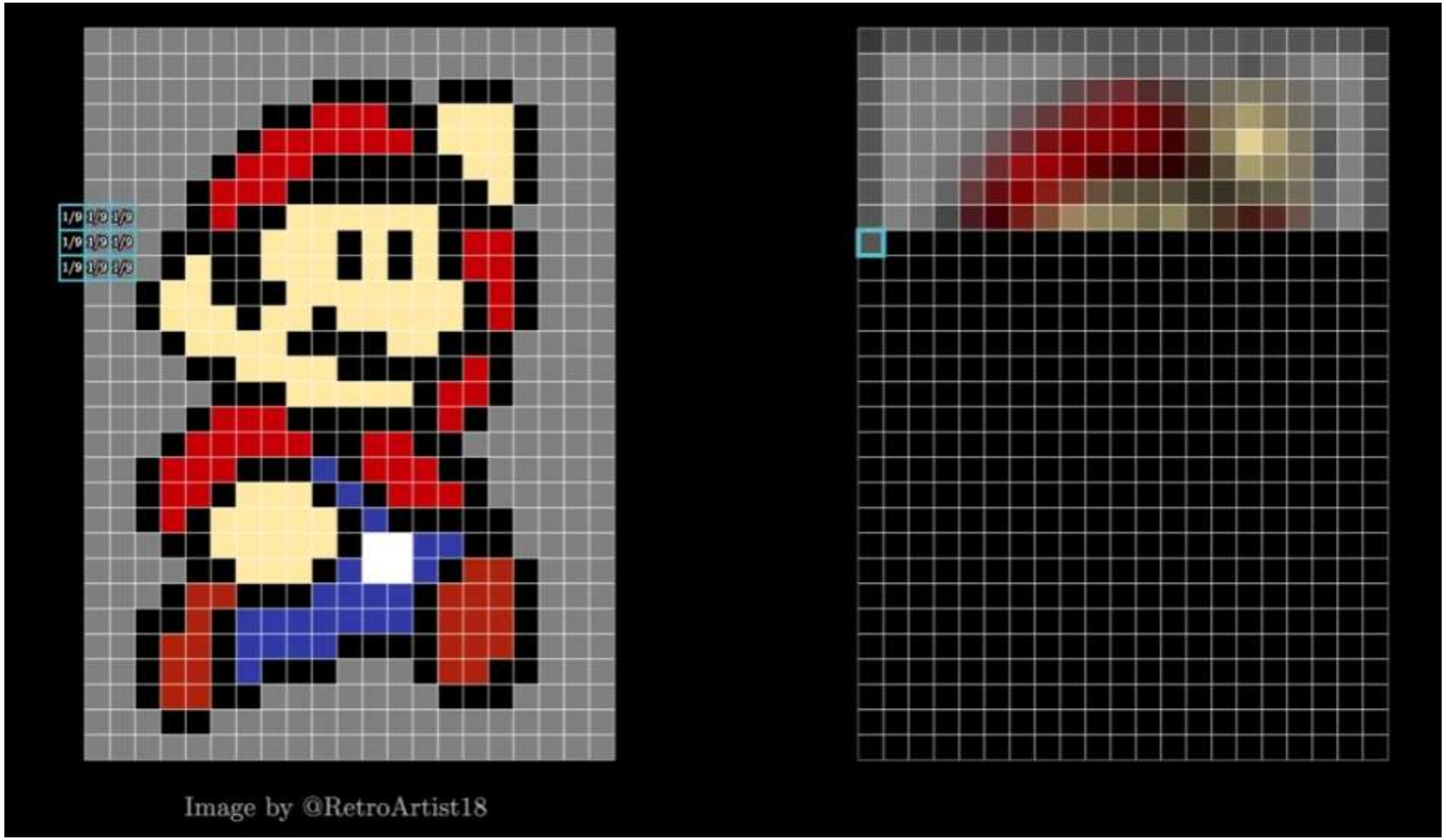
La **convolución** es la **operación matemática** por la cual un **kernel** se **desliza** sobre la entrada (imagen o mapa de características) y calcula la **suma ponderada** de los valores de entrada cubiertos por el kernel, produciendo una única salida.

Esta operación tiene dos funciones principales:

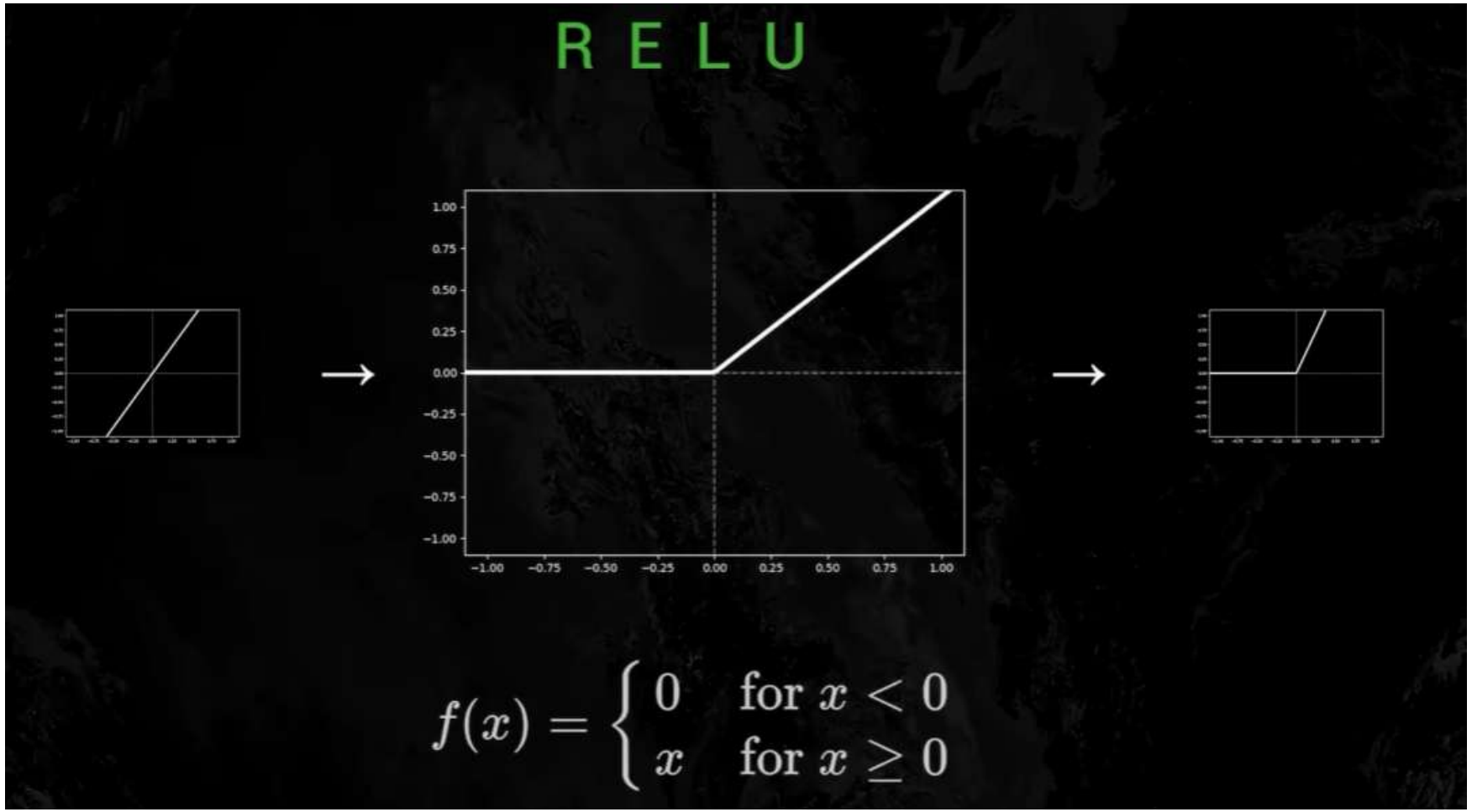
1) **Extracción de Características:** Multiplica los pesos del kernel por los valores de los píxeles de la entrada y los suma, destacando cuándo el patrón del kernel es detectado.
2) **Generación de Mapas de Características:** El resultado de aplicar el kernel en todas las posiciones es un nuevo tensor llamado **mapa de características** (*feature map*), que representa la presencia y fuerza de ese patrón específico a lo largo de toda la entrada.

La convolución

La convolución



Función de activación



Ejemplo

- Abrir Keras-CNN.ipynb

Ejercicio

- Hacer cambios en la arquitectura de la red para mejorar la precisión o disminuir el tiempo que emplea.