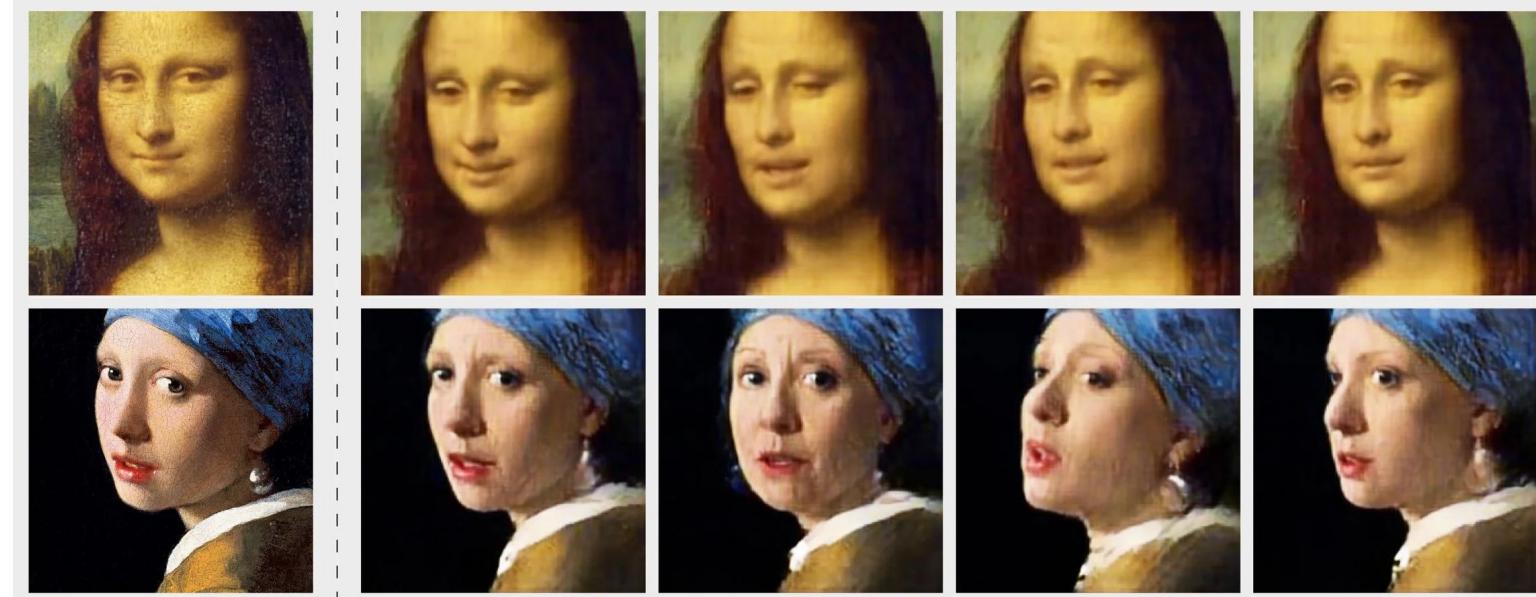




# Deep Learning Python



# Introducción al Deep Learning (aprendizaje profundo)





# ¿Qué es “Deep Learning”?

Inteligencia artificial

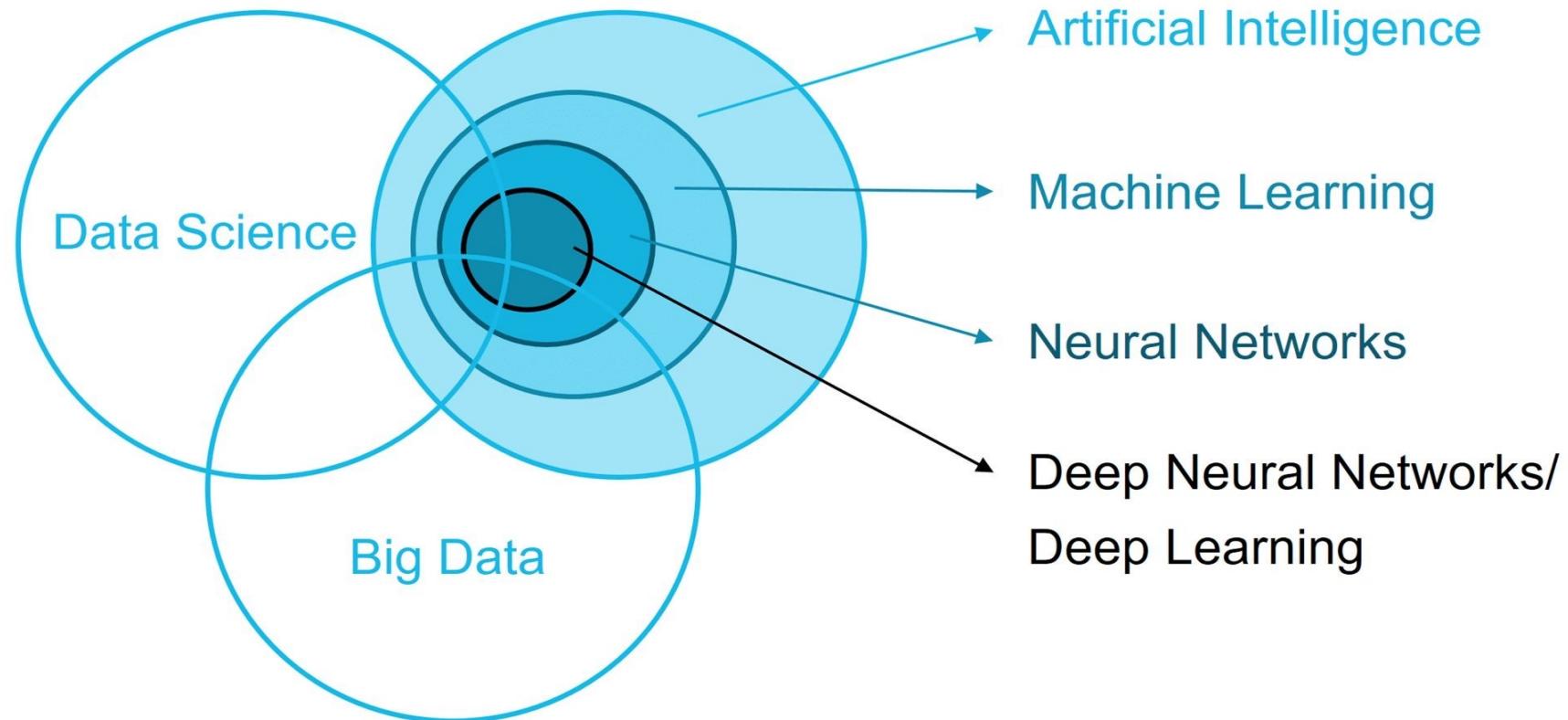
Big Data

Machine Learning

Deep learning

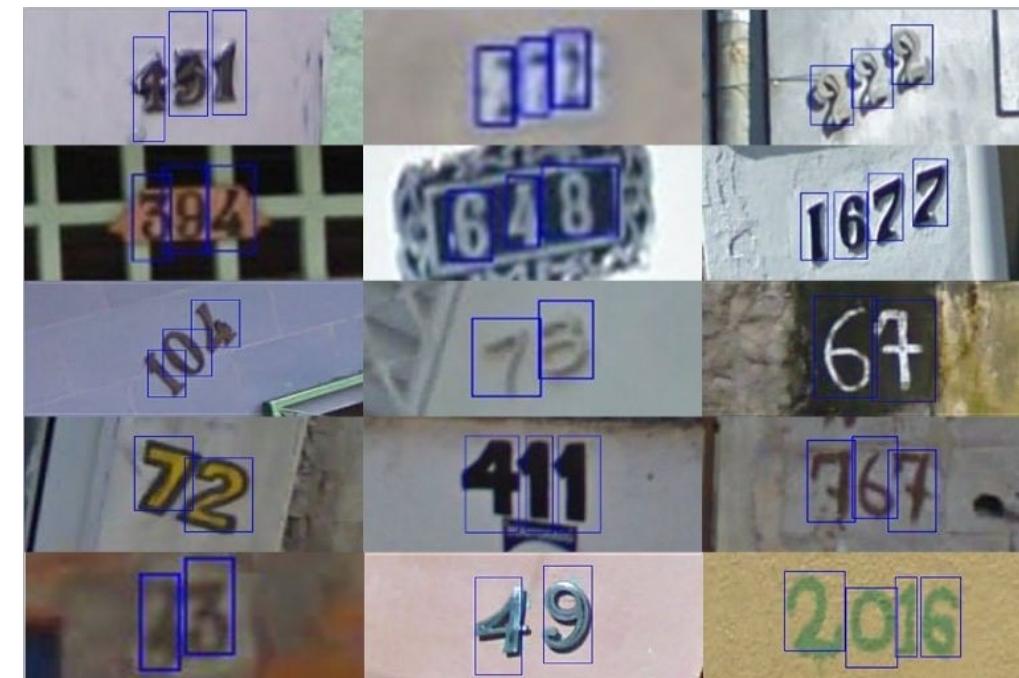
Ciencia de datos

# ¿Qué es “Deep Learning”?

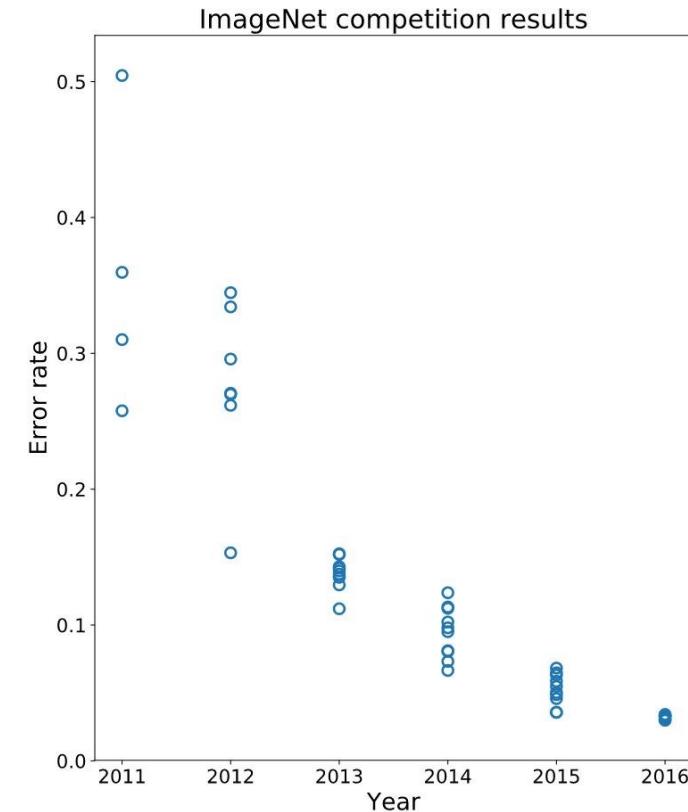
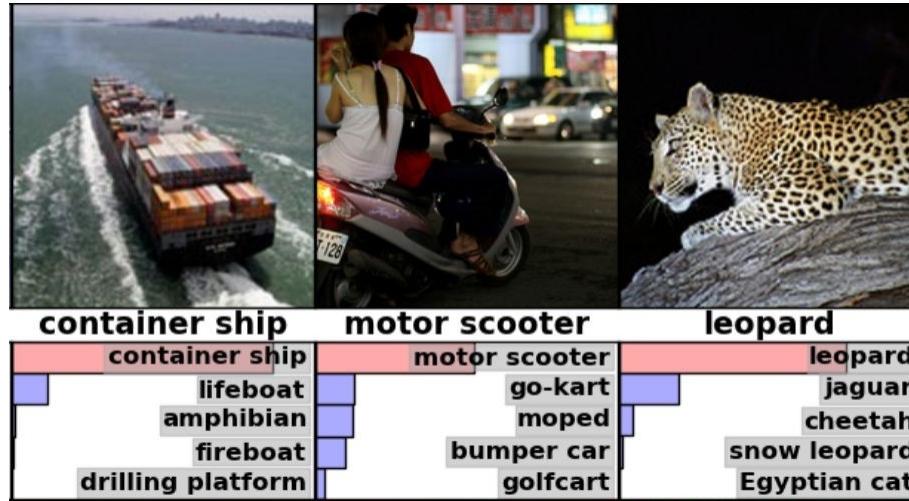


# 1. Reconocimiento óptico de caracteres

La visión por computadora es uno de los primeros lugares donde el Deep Learning comenzó a eclipsar otros métodos.



## 2. Clasificación de imágenes



### 3. Traducción de texto



# 4. Generación de imágenes





# 5. Generación de texto

**Here's a poem by Dr. Seuss about Elon Musk launching rockets with SpaceX, building electric cars at Tesla, selling flame throwers, dating Grimes, fighting the SEC, crushing short sellers, building brain chips to fight AI, and colonizing mars, and tweeting at all hours of day and night. Musk wants to make humans interplanetary to prevent extinction. The poem rhymes every other line with an ABAB structure. The rhyme structure is typical of Dr. Seuss nursery rhymes.**

**Elon Musk  
By Dr. Seuss**

Once there was a man  
who really was a Musk.  
He liked to build robots  
and rocket ships and such.

He said, "I'm building a car  
that's electric and cool.  
I'll bet it outsells those  
Gasoline-burning clunkers soon!" flamethrowers, tunnels, and such.

They sell cool cars  
that are electric and clean.  
And then he wanted to go  
even farther, you see.

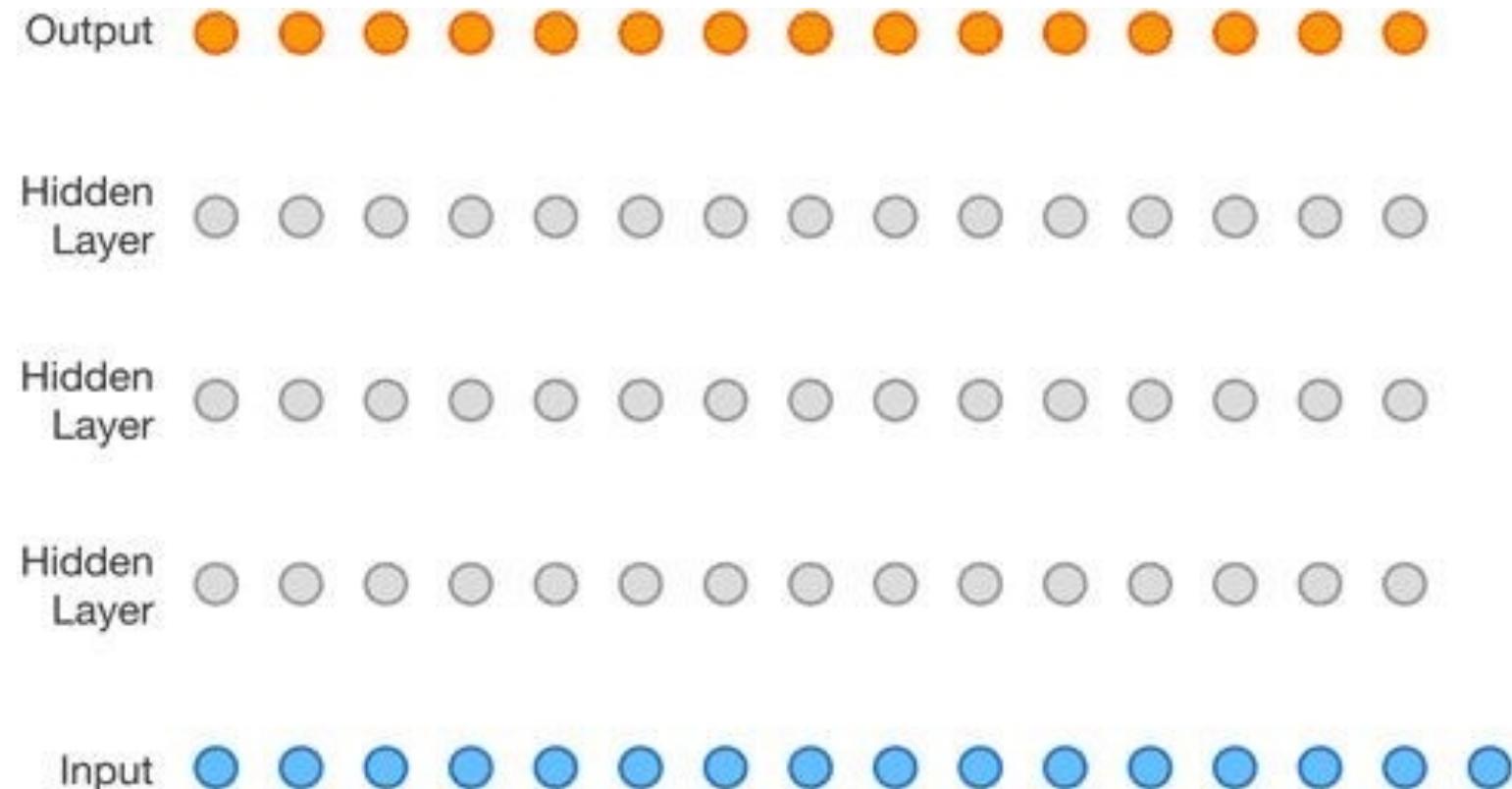
The first Tesla, Roadster,  
Was just the start of a bigger dream.  
He said, "I'm building a plane  
that's electric and green.

That's not all!  
That man is also a Musk.  
He likes electric cars,  
flamethrowers, tunnels, and such.

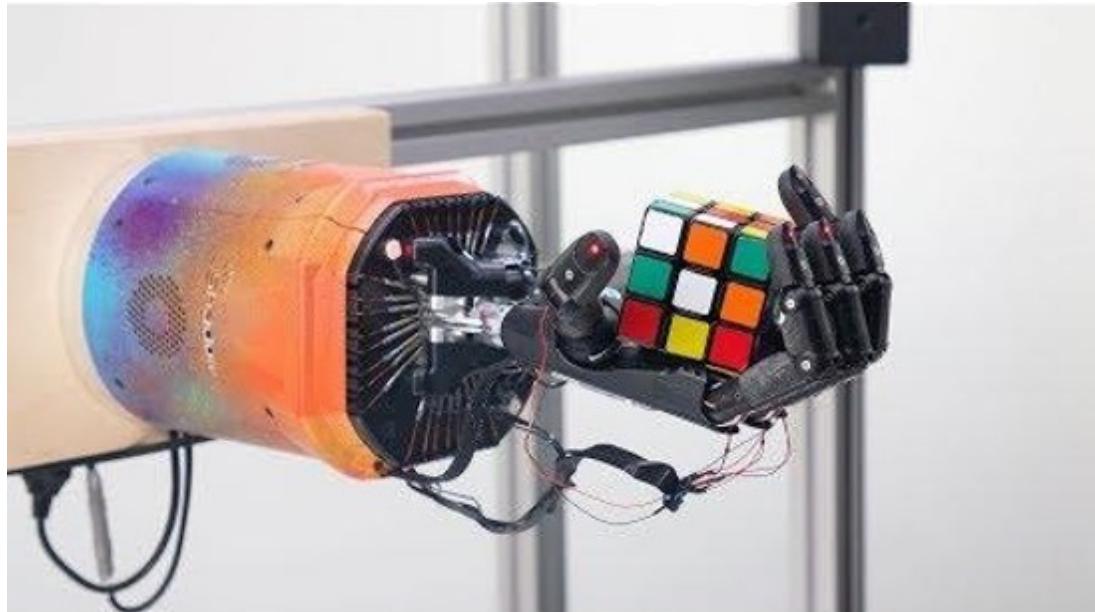
But then he thought, "If we make  
a very small, small chip,  
We'll implant it in our heads,  
And then connect our minds to the  
Internet!"

...

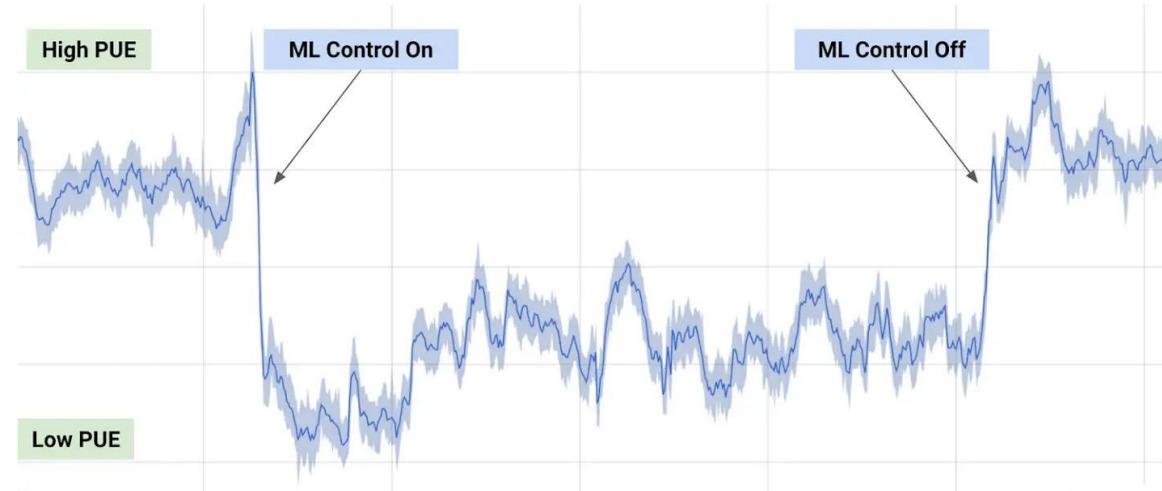
# 6. Generación de audio



# 7. Aprendizaje por refuerzo



# 8. Energía



MENU ▾

**nature**

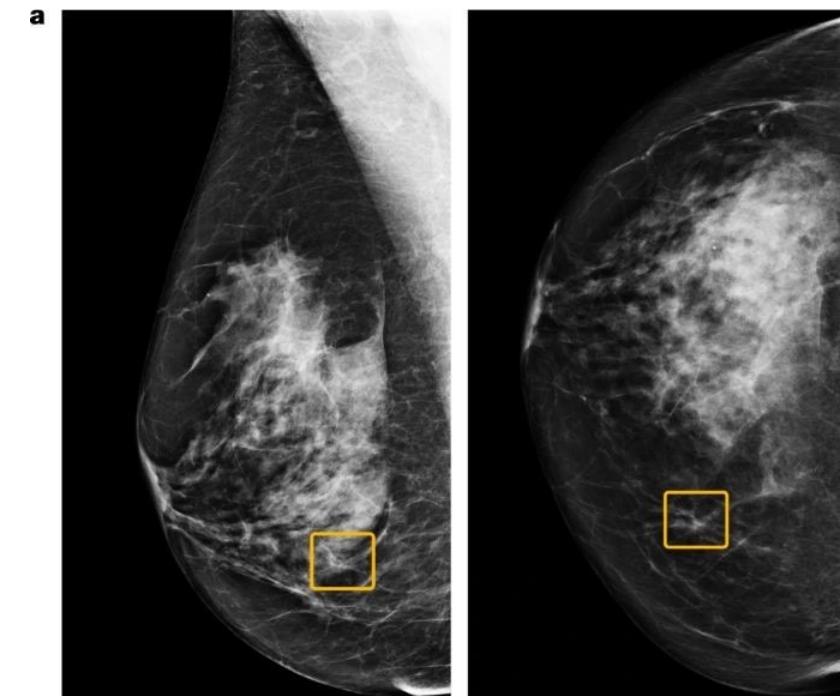
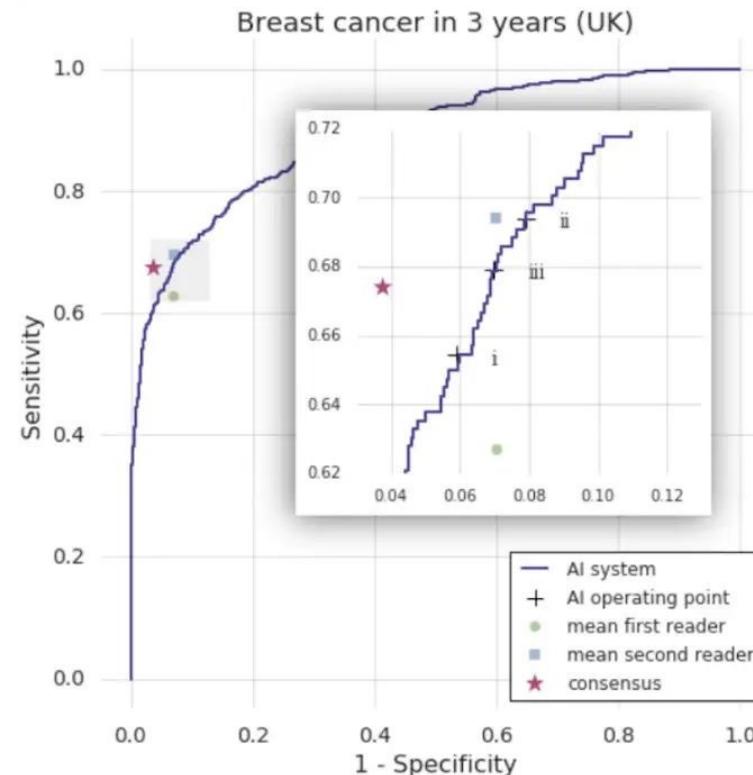
Article | Published: 19 February 2020

## Closed-loop optimization of fast-charging protocols for batteries with machine learning

Peter M. Attia, Aditya Grover, Norman Jin, Kristen A. Severson, Todor M. Markov, Yang-Hung Liao, Michael H. Chen, Bryan Cheong, Nicholas Perkins, Zi Yang, Patrick K. Herring, Muratahan Aykol, Stephen J. Harris, Richard D. Braatz✉, Stefano Ermon✉ & William C. Chueh✉

# 9. Salud

## IA de Deep Mind para la detección del cáncer de mama

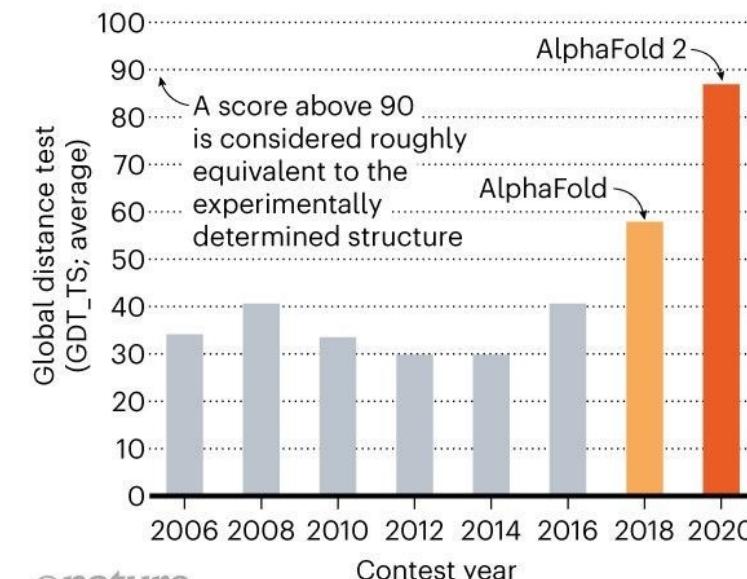


# 10. Biología



## STRUCTURE SOLVER

DeepMind's AlphaFold 2 algorithm significantly outperformed other teams at the CASP14 protein-folding contest — and its previous version's performance at the last CASP.



©nature

# 11. Desarrollo sostenible

Estimación de la pobreza mediante el aprendizaje de transferencia y las luces nocturnas



# 12. Invertir

Predecir el precio de las acciones a partir de los fundamentos de la empresa



# 13. Vigilancia

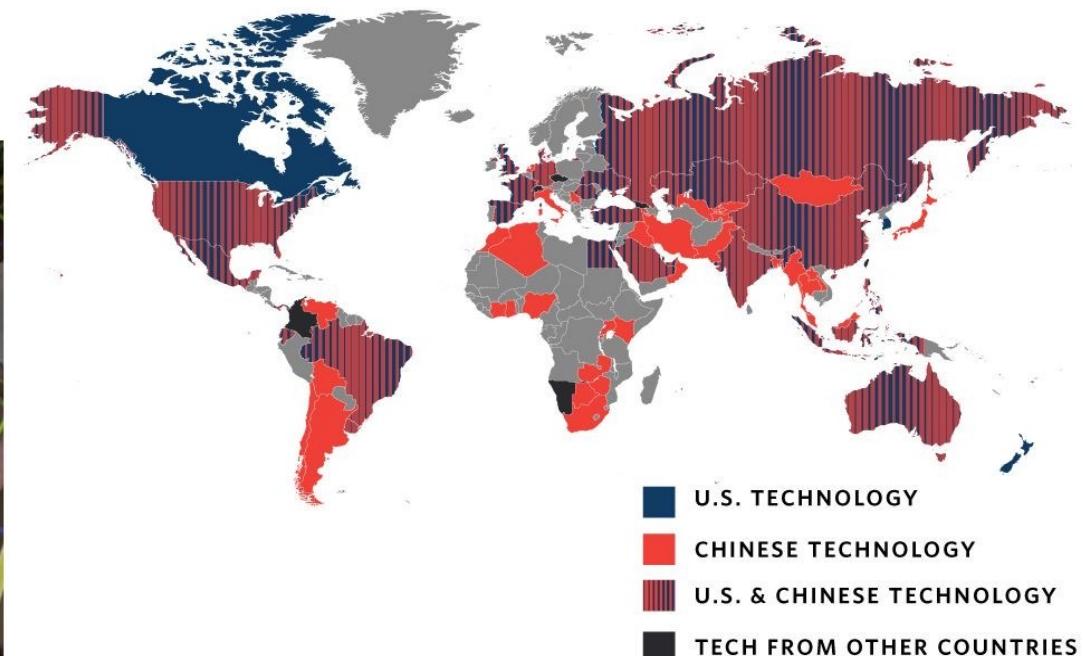
The New York Times

## *San Francisco Bans Facial Recognition Technology*



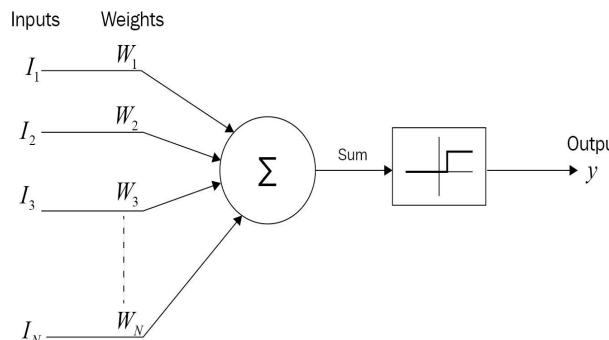
Attendees interacting with a facial recognition demonstration at this year's CES in Las Vegas. Joe Buglewicz for The New York Times

MAP 1  
AI Surveillance Technology Origin

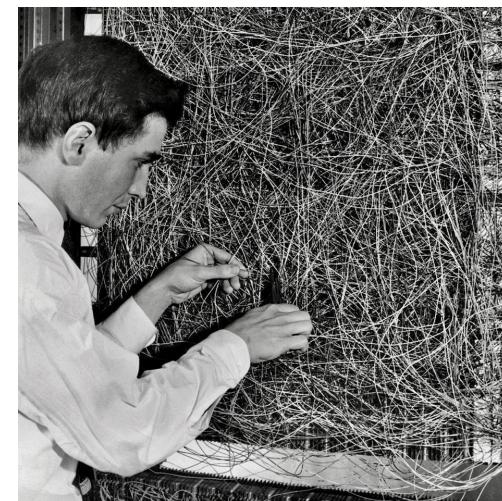


# Historia del Deep Learning

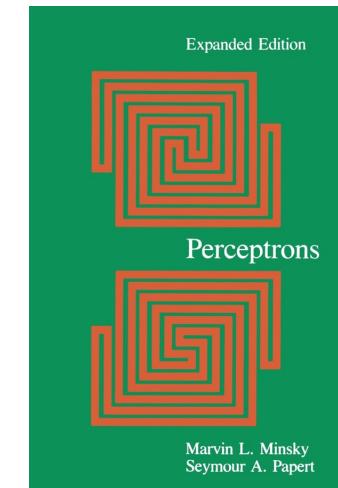
1943: McCulloch y Pitts desarrollan un modelo matemático de una neurona.



1957: Rosenblatt inventa el algoritmo perceptrón.



1969: Minsky y Papert mejoran el perceptrón y demuestran que no puede modelar problemas no lineales como XOR.

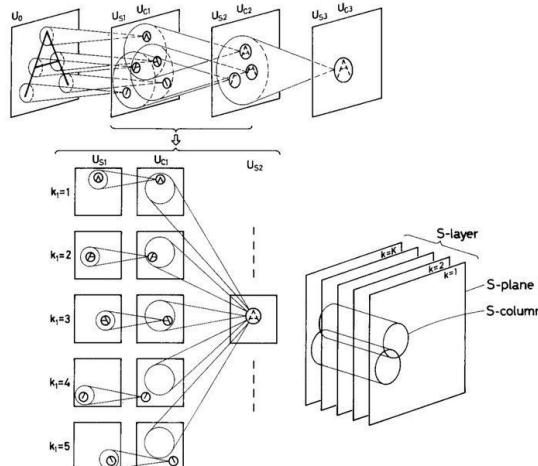


1974–1980: Primera Generación de la Robótica.



# Historia del Deep Learning

1980: Fukushima crea el "neocognitron", introduciendo capas convolucionales y de reducción de resolución.

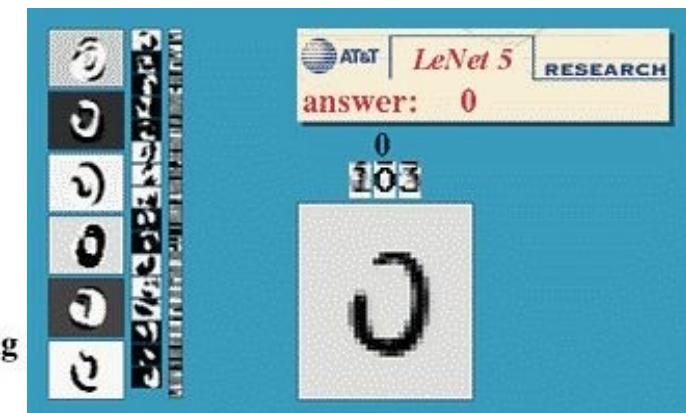


1986: Rumelhart, Hinton y Williams aplican la retropropagación para entrenar redes neuronales.



D.E. Rumelhart, G.E. Hinton, R.J. Williams  
**Learning representation by back-propagating errors.** *Nature*, 323 (1986), pp. 533–536

1989: LeCun utiliza backprop para entrenar redes neuronales convolucionales para leer dígitos.



1987–1994: Segunda Generación en Robótica.

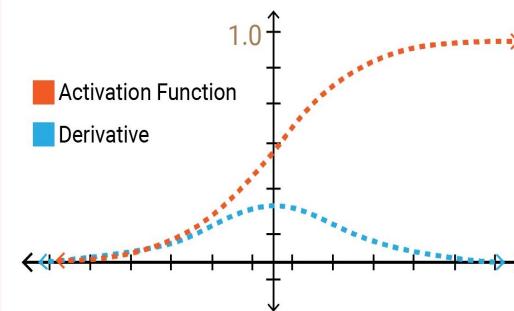


# Historia del Deep Learning

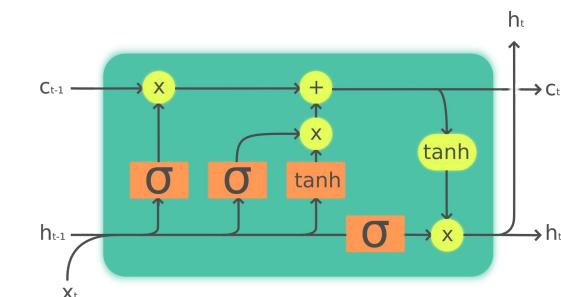
1990s: Las computadoras se vuelven más rápidas, las GPUs son desarrolladas.



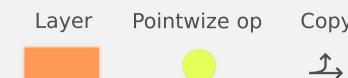
1991: Se identifica el problema de la caída gradiente.



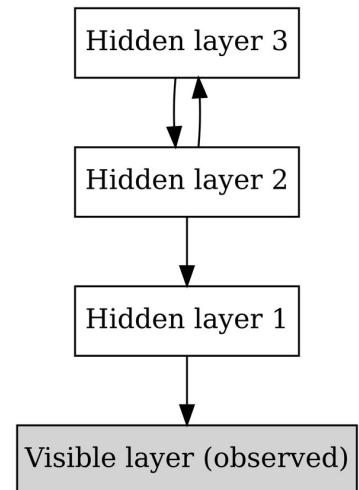
1997: Hochreiter y Schmidhuber desarrollan el algoritmo de memoria a corto plazo.



Legend:

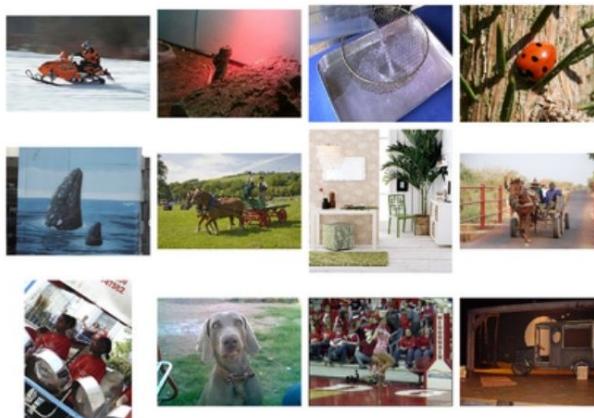


2006: Hinton y otros muestran que las redes más profundas se pueden entrenar una capa a la vez.

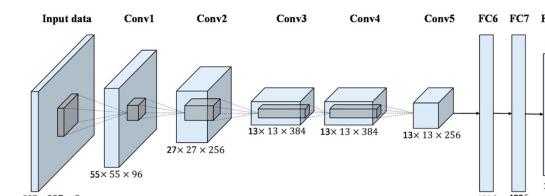


# Historia del Deep Learning

**2009:** Fei-fei Li lanza ImageNet, un conjunto de datos de clasificación de 14 millones de imágenes etiquetadas.



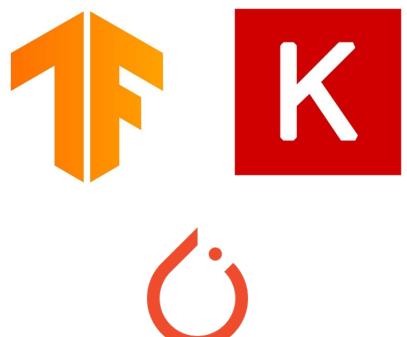
**2012:** AlexNet, una red neuronal tipo CNN, gana el Desafío de Reconocimiento Visual a Gran Escala.



**2014:** Goodfellow diseña redes adversariales generativas para la generación de datos.



**2015-2016:** Paquetes de Deep Learning como TensorFlow, Keras, and PyTorch son desarrollados.





# Fundamentos Matemáticos: Deep Learning

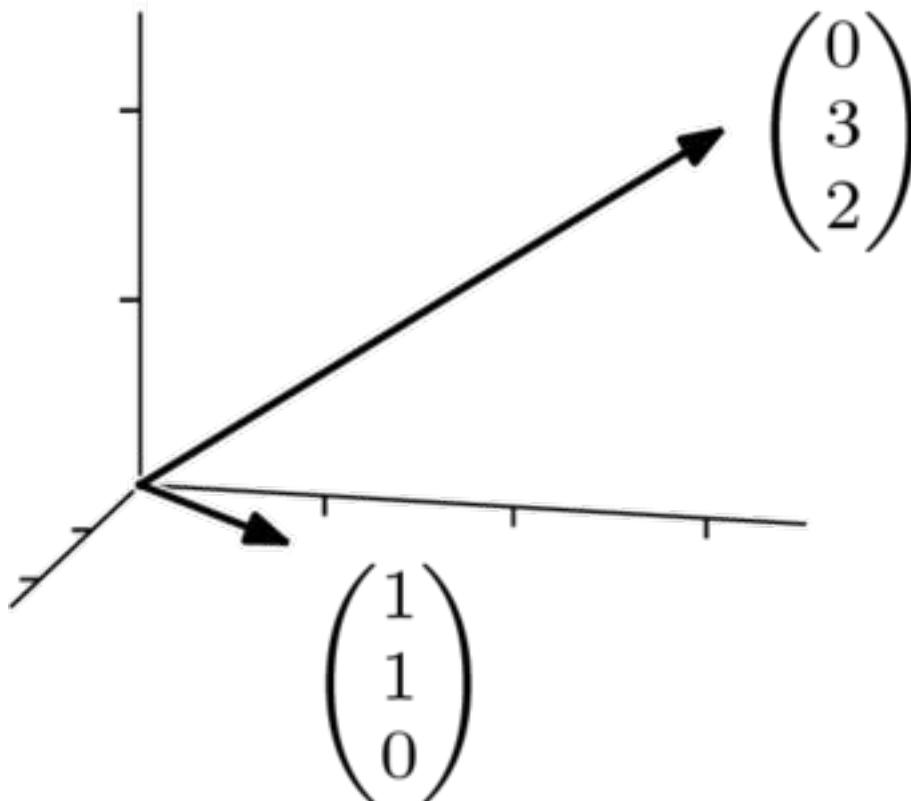
# ¿Qué es un vector?

$$\begin{bmatrix} 2 \\ 7 \\ 3 \end{bmatrix}$$


Dimensión = 3

$$\vec{v} \in \mathbb{R}^3$$

# ¿Qué es un vector?



Además del álgebra lineal, es posible que haya aprendido sobre vectores en cursos de física como entidades con dirección y magnitud.

Aquí hay dos vectores más de dimensión 3, visualizados en el espacio euclíadiano.

# Operaciones vectoriales

## 1. Multiplicación escalar

$$4 * \begin{pmatrix} 2 \\ 7 \\ 3 \end{pmatrix} = \begin{pmatrix} 8 \\ 28 \\ 12 \end{pmatrix}$$

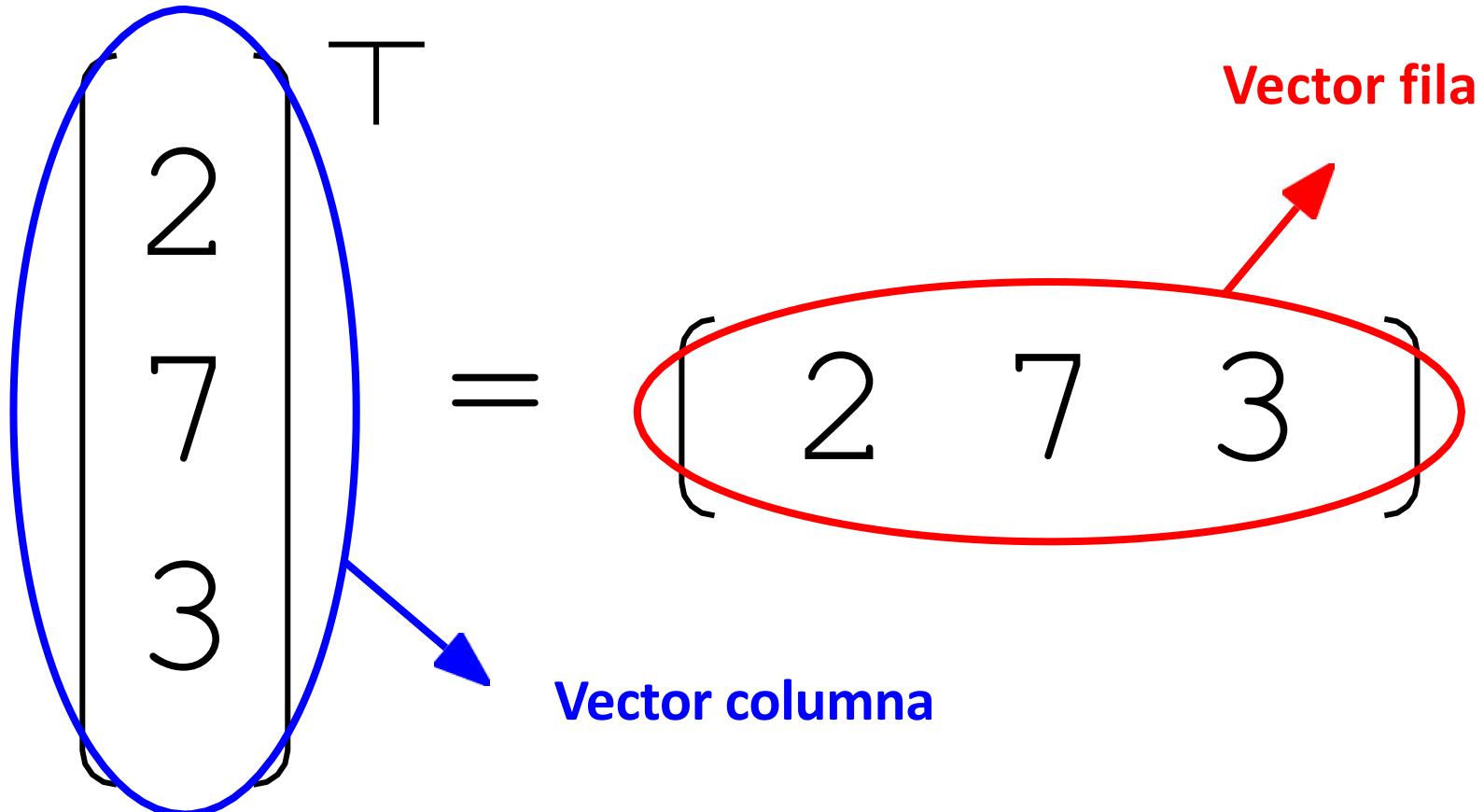
# Operaciones vectoriales

## 2. Suma

$$\begin{bmatrix} 2 \\ 7 \\ 3 \end{bmatrix} + \begin{bmatrix} 1 \\ 3 \\ 6 \end{bmatrix} = \begin{bmatrix} 3 \\ 10 \\ 9 \end{bmatrix}$$

# Operaciones vectoriales

## 3. Transponer



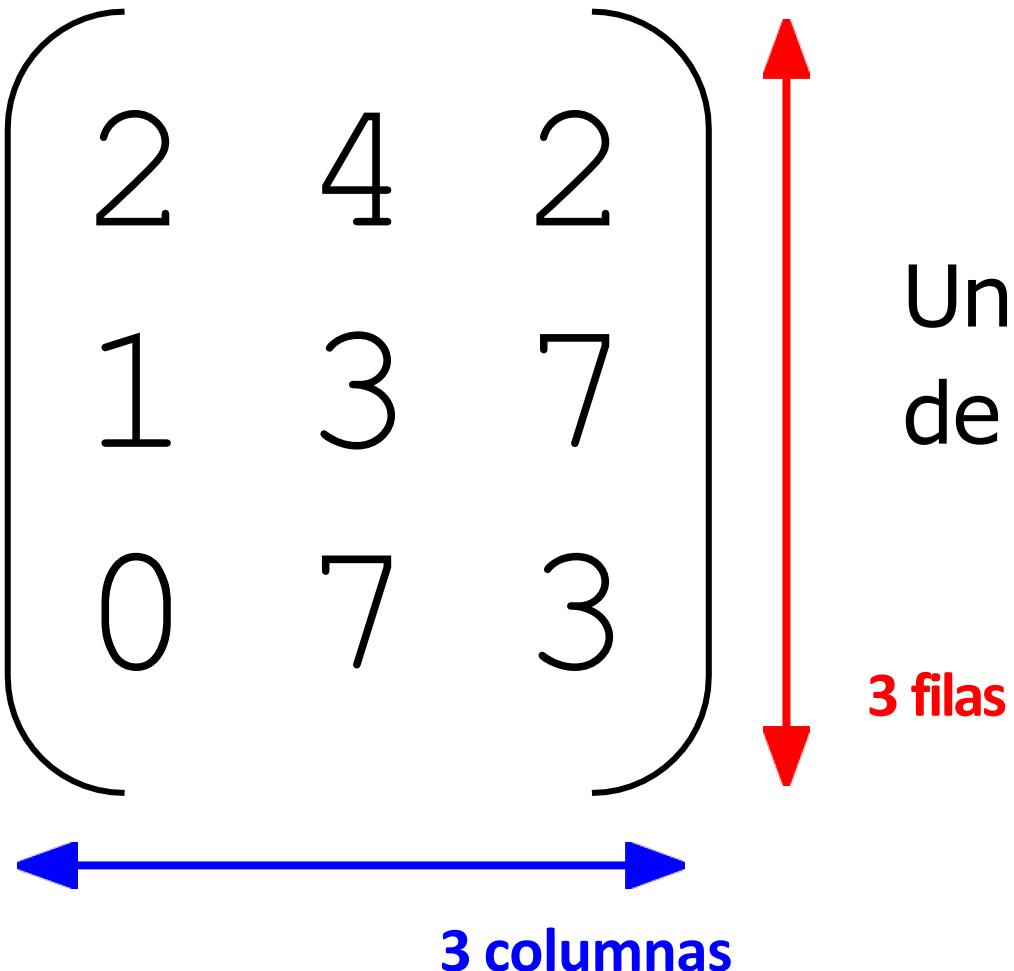
# Operaciones vectoriales

## 4. Producto interno

$$\begin{pmatrix} 2 & 7 & 3 \end{pmatrix} \begin{pmatrix} 1 \\ 3 \\ 6 \end{pmatrix} = 2 * 1 + 7 * 3 + 3 * 6 = 41$$

$\vec{u}^\top \vec{v} = w$

# ¿Qué es una matriz?



Una matriz rectangular (o tabla) de números.

# ¿Qué es una matriz?

$$\begin{matrix} & \begin{matrix} 1 & 2 & \dots & n \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ \vdots \\ m \end{matrix} & \left[ \begin{matrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{matrix} \right] \end{matrix} \quad \mathbf{A} \in \mathbb{R}^{m \times n}$$

# ¿Qué es una matriz?

$$\begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 4 \\ 3 \\ 7 \end{pmatrix} \begin{pmatrix} 2 \\ 7 \\ 3 \end{pmatrix}$$

Se puede pensar en una matriz como un vector de filas de vectores de columna.

# ¿Qué es una matriz?

$$\begin{pmatrix} 2 & 4 & 2 \\ 1 & 3 & 7 \\ 0 & 7 & 3 \end{pmatrix}$$

... o como un vector de  
columna de vectores de fila.

# Operaciones matriciales

1. Multiplicación por un escalar.
2. Suma.

$$2 * \begin{pmatrix} 2 & 4 & 2 \\ 1 & 3 & 7 \\ 0 & 7 & 3 \end{pmatrix} = \begin{pmatrix} 4 & 8 & 4 \\ 2 & 6 & 14 \\ 0 & 14 & 6 \end{pmatrix}$$

# Operaciones matriciales

## 3. Transponer

$$\begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 4 \\ 3 \\ 7 \end{pmatrix} \begin{pmatrix} 2 \\ 7 \\ 3 \end{pmatrix}^\top = \begin{pmatrix} 2 & 1 & 0 \\ 4 & 3 & 7 \\ 2 & 7 & 3 \end{pmatrix}$$

# Operaciones matriciales

## 4. Multiplicación con un vector.

$$\begin{pmatrix} 2 & 4 & 2 \\ 1 & 3 & 7 \\ 0 & 7 & 3 \end{pmatrix} \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 18 \\ 24 \\ 27 \end{pmatrix}$$

# Operaciones matriciales

## 4. Multiplicación con un vector.

$$\begin{bmatrix} \vec{a}_1^T \\ \vec{a}_2^T \\ \vec{a}_3^T \end{bmatrix} \vec{b} = \begin{bmatrix} 18 \\ 24 \\ 27 \end{bmatrix}$$

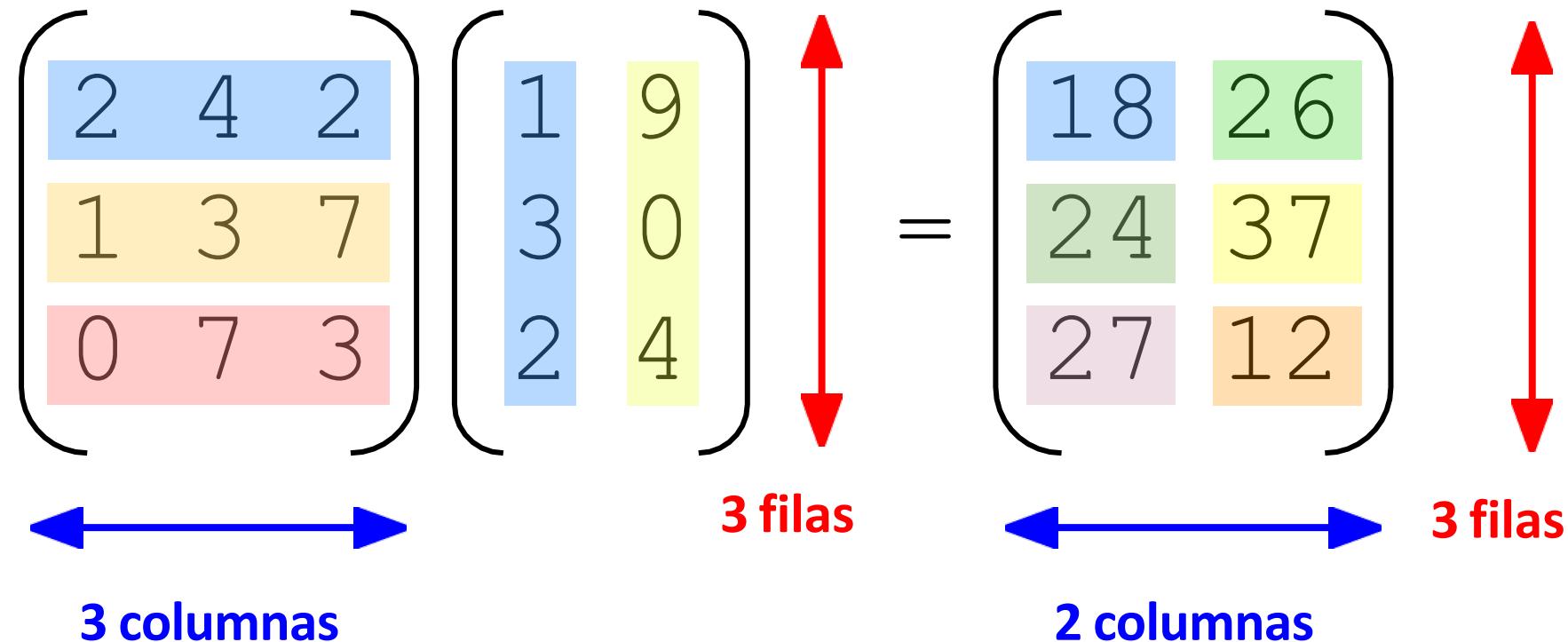
# Operaciones matriciales

## 4. Multiplicación con un vector.

$$\begin{pmatrix} \vec{a}_1^\top \vec{b} \\ \vec{a}_2^\top \vec{b} \\ \vec{a}_3^\top \vec{b} \end{pmatrix} = \begin{pmatrix} 18 \\ 24 \\ 27 \end{pmatrix}$$

# Operaciones matriciales

## 5. Multiplicación con otra matriz.



# ¿Qué es un tensor?

A tensor is an N-dimensional array of data



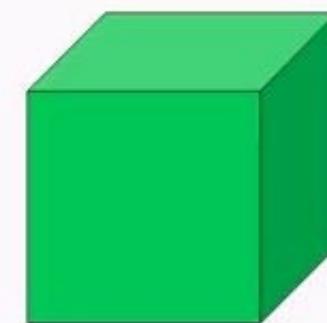
Rank 0  
Tensor  
scalar



Rank 1  
Tensor  
vector



Rank 2  
Tensor  
matrix



Rank 3  
Tensor



Rank 4  
Tensor



# Fundamentos Matemáticos del “Deep Learning”

- Los datos se almacenan en arreglos Numpy multidimensionales, también conocidos como tensores. Pero, ¿qué es un tensor?
- Un tensor es un contenedor de datos, casi siempre numérico. Las matrices son tensores en 2D; los tensores son una generalización de las matrices a un número arbitrario de dimensiones (en tensores a las dimensiones se les llaman ejes). Al número de ejes de un tensor también se le llama **rango**.



- Escalares (Tensores 0D)

Un tensor que contenga solo un número es llamado **escalar**. En Numpy, un número float32 o float64 es un tensor escalar (o arreglo escalar). Podemos mostrar el número de ejes de un tensor Numpy mediante el atributo `ndim`; un tensor escalar tiene 0 ejes.

Aquí hay un escalar Numpy:

```
>>> import numpy as np
>>> x = np.array(12)
>>> x
array(12)
>>> x.ndim
0
```

- Vectores (Tensores 1D)

Un arreglo de números se llama vector, o tensor 1D. Un tensor 1D tiene exactamente un eje.

Aquí hay un vector Numpy:

```
>>> x = np.array([12, 3, 6, 14])
>>> x
array([12, 3, 6, 14])
>>> x.ndim
1
```

Este vector tiene cuatro entradas, por eso se llama vector de 4 dimensiones. No hay que confundir un vector 4D con un tensor 4D. Un vector 4D tiene solo un eje y cuatro dimensiones a lo largo de su eje, mientras que un tensor 4D tiene 4 ejes y cualquier número de dimensiones a lo largo de cada eje.

- Matrices (tensores 2D)

Un arreglo de vectores es una matriz, o tensor 2D. Una matriz tiene dos ejes (filas y columnas). Podemos visualizarla como una cuadrícula de rectangular de números.

Esto es una matriz Numpy:

```
>>> x = np.array([[7, 80, 4, 38, 2],[4, 77, 1, 33, 3],[5, 78, 2, 34, 1]])  
>>> x.ndim  
2
```

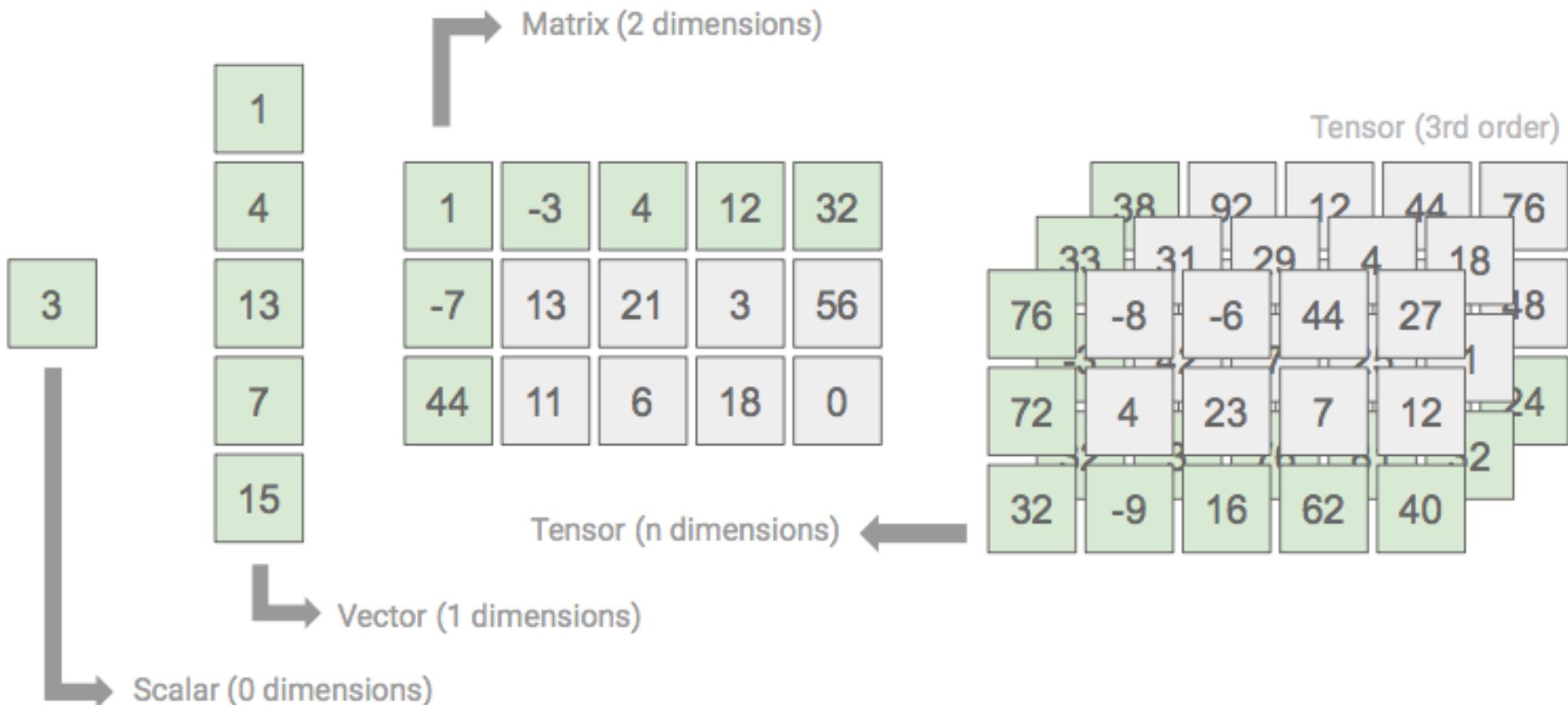
Las entradas del primer eje son las **filas**, y las entradas del segundo eje son las **columnas**. En este ejemplo, [7, 80, 4, 38, 2] es la primera fila de x, y [7, 4, 5] la primera columna.

- Tensores 3D y tensores de más dimensiones

Si empaquetamos dichas matrices en una nueva matriz, obtenemos un tensor 3D, que se puede interpretar visualmente como un cubo de números. A continuación, un tensor 3D de Numpy:

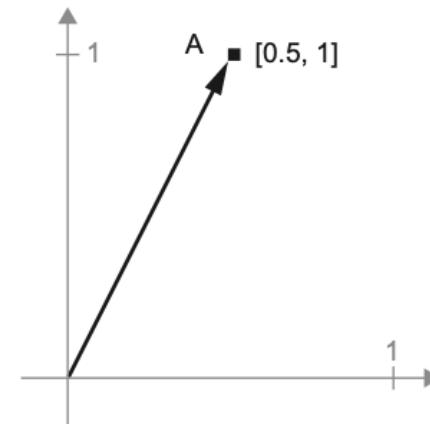
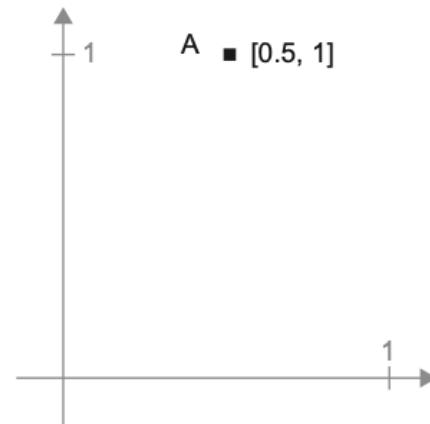
```
>>> x = np.array([[[7, 80, 4, 38, 2],[4, 77, 1, 33, 3],[5, 78, 2, 34, 1]],  
... [[7, 80, 4, 38, 2],[4, 77, 1, 33, 3],[5, 78, 2, 34, 1]],  
... [[7, 80, 4, 38, 2],[4, 77, 1, 33, 3],[5, 78, 2, 34, 1]]])  
>>>  
>>> x.ndim  
3
```

Empaquetando tensores 3D en una matriz, creamos tensores 4D, y así sucesivamente. En Deep Learning, generalmente manipularemos tensores desde 0D a 4D, aunque podríamos subir a 5D si se procesan datos de video.



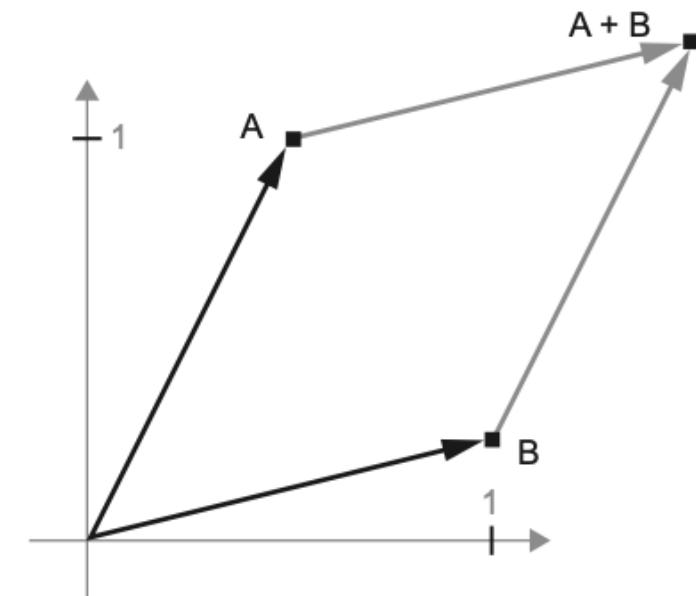
# Interpretación geométrica de operaciones entre tensores

- A es un punto en un espacio bidimensional. Podemos imaginar un vector como una flecha uniendo el origen al punto.



# Interpretación geométrica de operaciones entre tensores

- Ahora consideremos un nuevo punto B, el cual sumaremos a A. Esto geométricamente se hace encadenando las dos flechas de vectores, la nueva ubicación resultante será la representación vectorial de la suma de ambos vectores.



# Funciones de activación

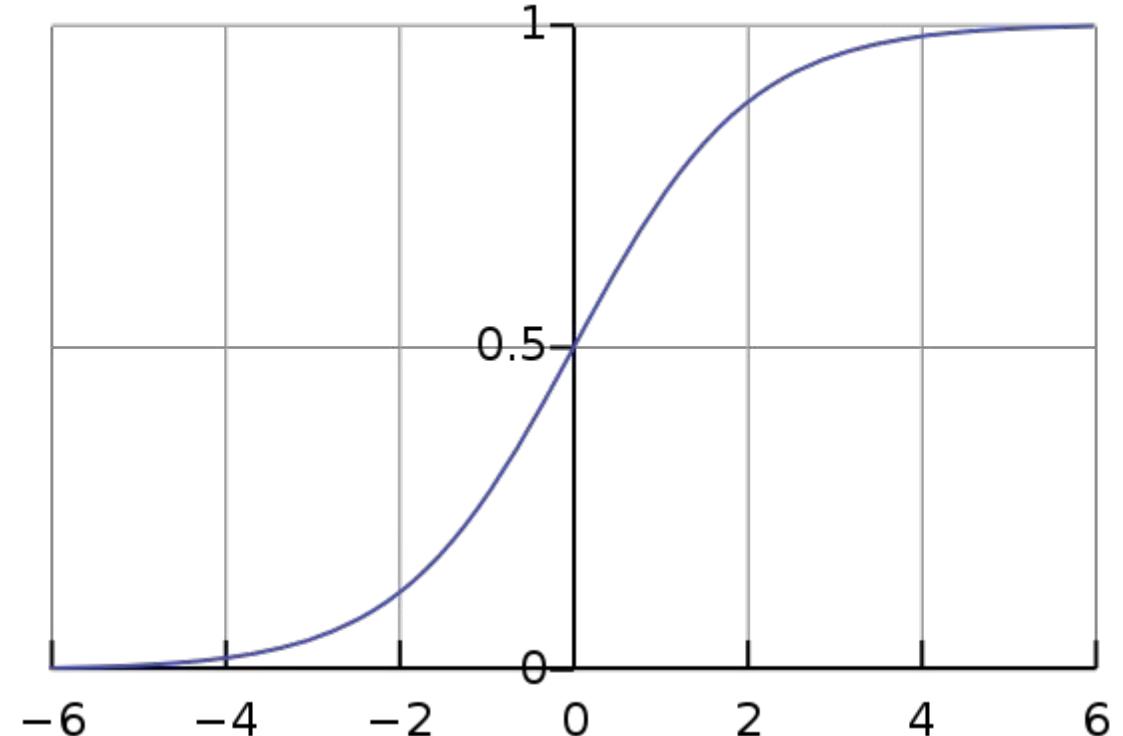
- Existen una serie de funciones de activación, algunas de las cuales fueron diseñadas con las neuronas en mente
- ¿Cuál es el propósito?
- La mayoría de los fenómenos no son lineales, por lo que se introduce no linealidad en las redes neuronales.

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = \max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

Copyright © Sebastian Raschka 2016  
(<http://sebastianraschka.com>)

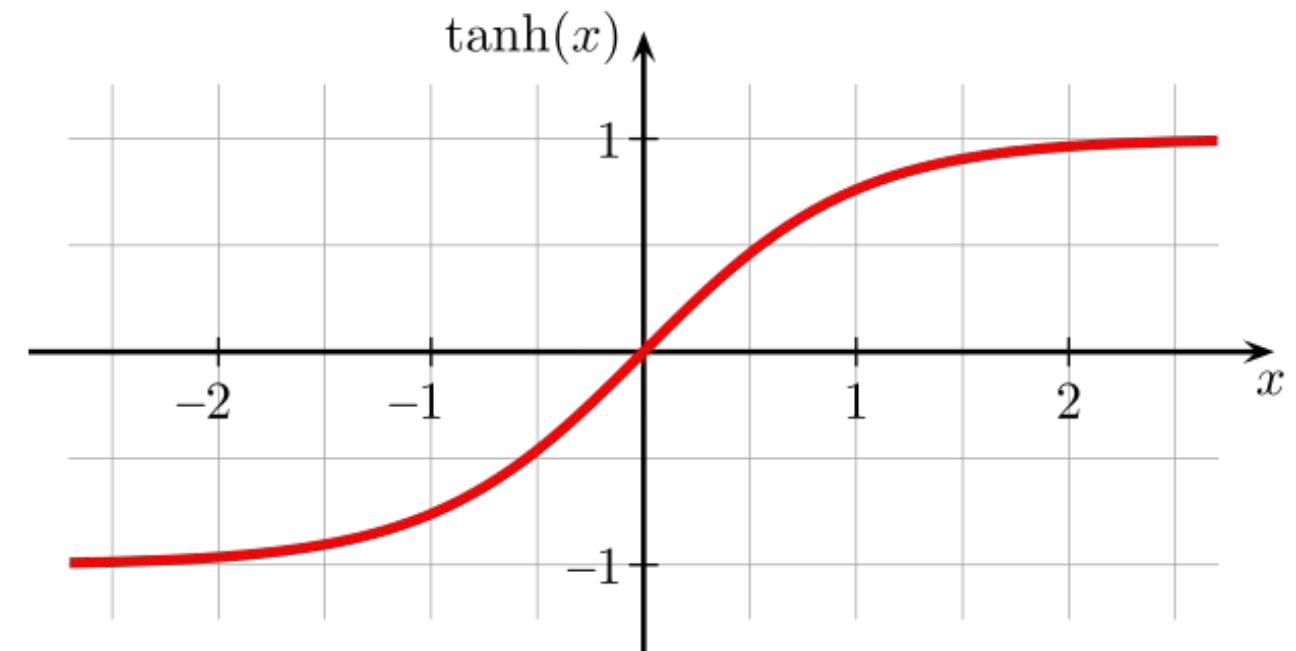
# Sigmoid (logistic)

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$



# Tangente Hiperbólico

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



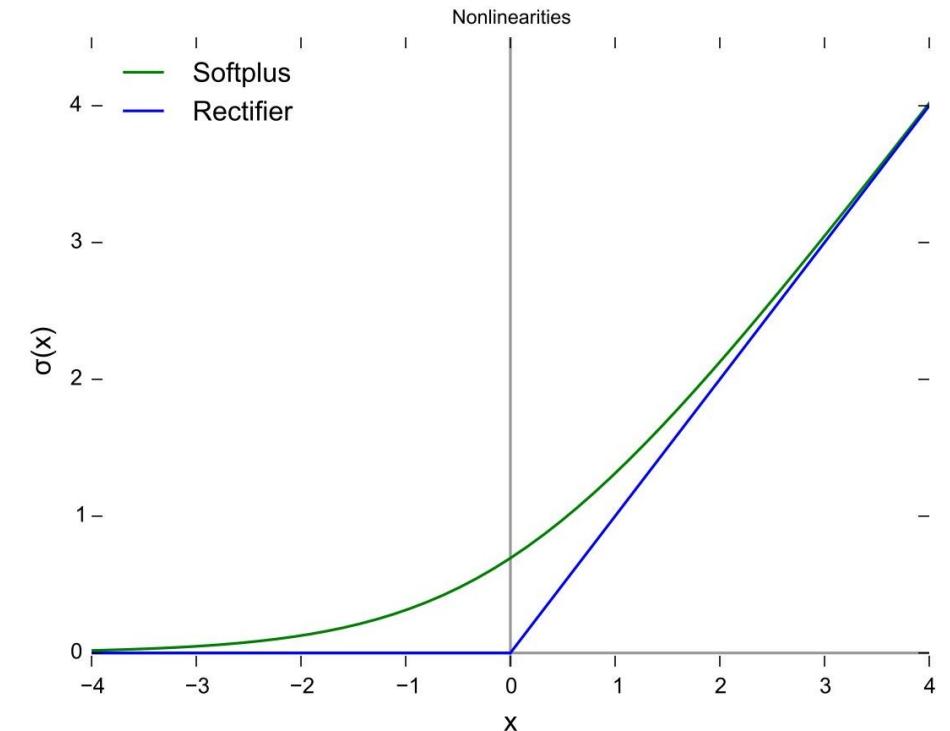
# La función ReLU

Son las siglas de “Rectified Linear Unit”  
(Unidad Lineal Rectificada)

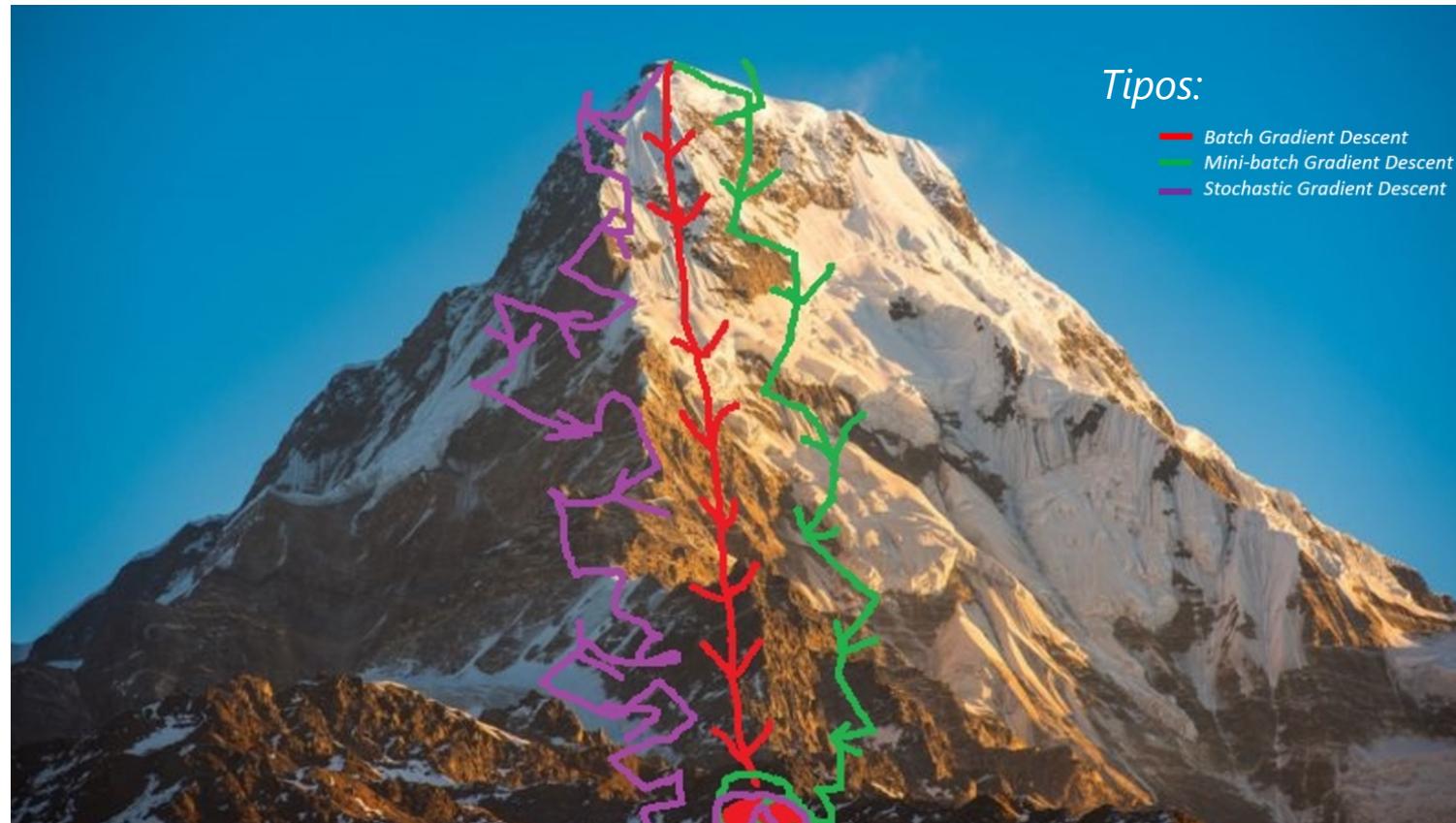
$$\text{ReLU}(x) = \max(0, x)$$

Una versión suave de ReLU es softplus

$$\text{softplus}(x) = \ln(1 + e^x)$$



# Descenso de Gradiente

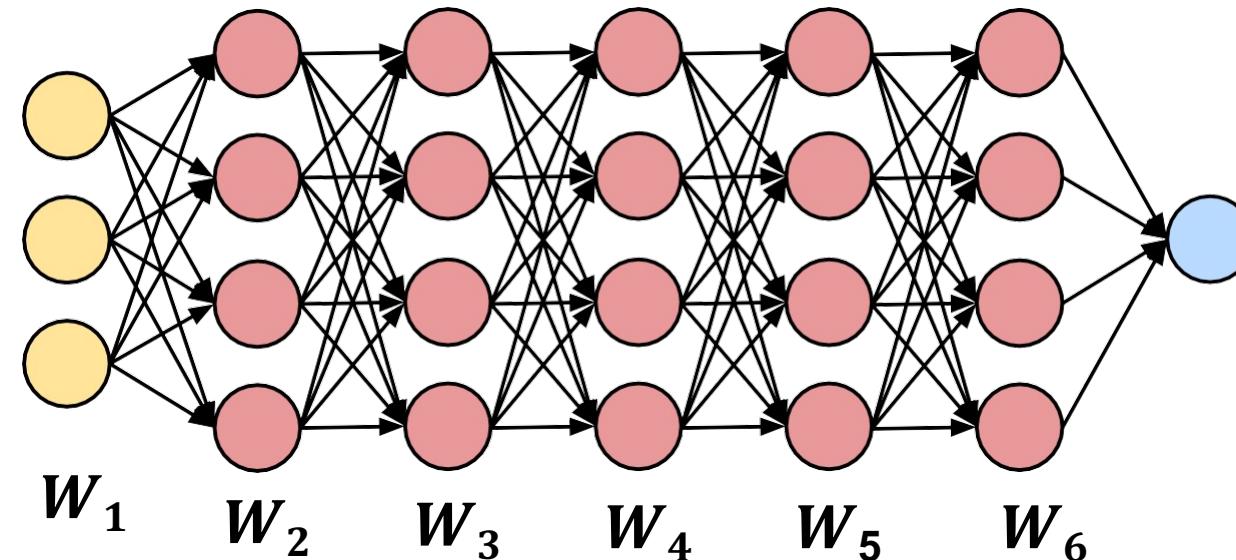


# OPTIMIZACIÓN BASADA EN GRADIENTES

- Las Redes Neuronales Profundas (Deep Learning) realizan este mapeo de entrada a la salida a través de una secuencia profunda de transformaciones de datos simples (**capas**) y estas transformaciones de datos se aprenden mediante la exposición a ejemplos.
- Lo que una capa hace a sus datos de entrada se almacena en los pesos de la capa. Se podría decir que la *transformación implementada por una capa es parametrizada por sus pesos*. En este contexto, “**learning**” significa encontrar un conjunto de valores para los pesos de todas las capas en una red, tales que la red mapee correctamente ejemplos de entradas a sus objetivos asociados.

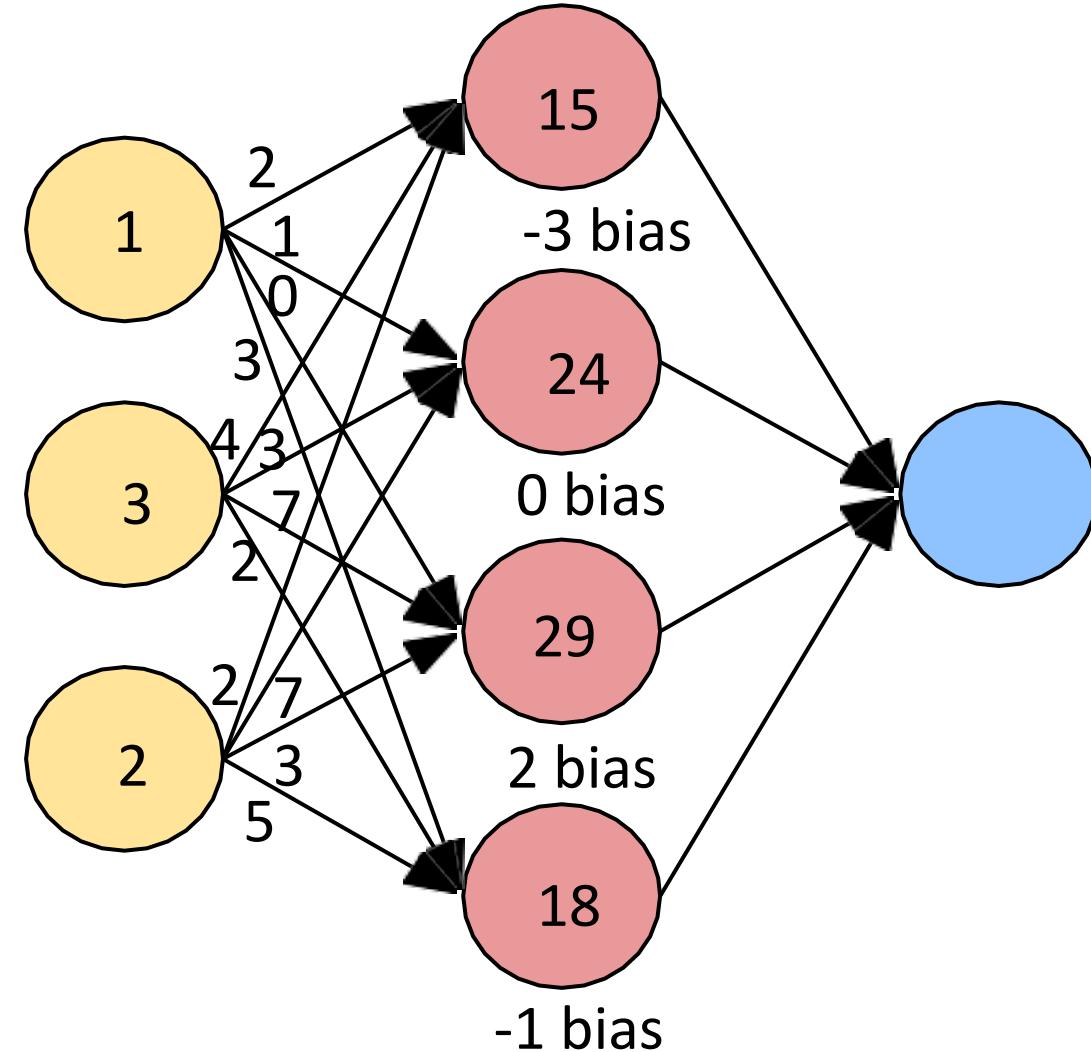
# Apilamiento de varias capas

*Se convierte en una red neuronal profunda (Deep) cuando se utilizan muchas capas:*



Con el método de gradiente se calcula los pesos para cada neurona.

# Apilamiento de varias capas



# OPTIMIZACIÓN BASADA EN GRADIENTES

- Lo que pasa, es que una red neuronal puede contener, por ejemplo, 10 millones de parámetros (pesos). Encontrar el valor correcto para todos puede ser una *tarea difícil*, en especial por que modificar el parámetro de una puede afectar el comportamiento de las otras.
- Para controlar la salida de la red neuronal, necesitamos ser capaces de medir qué tan distante está la salida de lo que esperamos. Este es el trabajo de la **función de pérdida**. Esta función toma las predicciones de la red y el objetivo real y calcula una **puntuación de distancia**, capturando qué tan bien ha funcionado la red en este caso específico.

# OPTIMIZACIÓN BASADA EN GRADIENTES

- Los pesos de una capa de una red neuronal contienen información aprendida por la red durante el entrenamiento. Inicialmente estas matrices de pesos son llenadas con pequeños **valores aleatorios** (inicialización aleatoria).
- Aunque los resultados no tengan mucho sentido, sí representan un punto de partida, ya que con base a estos iremos ajustando los pesos, basándonos en la **retroalimentación**.
- Este ajuste es lo que se conoce como *entrenamiento*, es decir, el *aprendizaje del Machine Learning*.
- Este ajuste es el trabajo del **optimizador**, que implementa lo que se llama el algoritmo de **Retropropagación**: el algoritmo central en **Deep Learning**.



# OPTIMIZACIÓN BASADA EN GRADIENTES

- Todo esto sucede dentro de un ciclo de entrenamiento (training loop), repetiremos estos pasos tanto como sea necesario:
  1. Divida los datos en muestras de entrenamiento  $x$  y prueba  $y$ .
  2. Ejecute la red en  $x$  (un paso llamado pase directo) para obtener predicciones  $y_{pred}$ .
  3. Calcule la pérdida (puntuación de pérdida) de la red en el lote, una medida de error entre  $y_{pred}$  y  $y$ .
  4. Actualice todos los pesos de la red de manera que reduzca ligeramente el error en este lote.

# OPTIMIZACIÓN BASADA EN GRADIENTES

- Eventualmente acabaremos teniendo una red con un error muy bajo. La red habrá “aprendido” a mapear sus entradas a los objetivos buscados. Aquí el verdadero reto consistirá en encontrar los pesos adecuados para la red.
- ¿Cómo saber si los pesos deben aumentar o disminuir?
- Un buen método es aprovechar el hecho de que todas las operaciones utilizadas en la red son *diferenciables* (derivables) y calcular el **gradiente** de la pérdida con respecto a los coeficientes de la red. Luego se puede mover los coeficientes en la dirección opuesta al gradiente, disminuyendo así la pérdida.



# ¿Qué es una derivada?

Considere una función continua y suave (derivable)  $f(x) = y'$  en  $\mathbb{R}$ . Como la función es *continua*, un pequeño cambio *épsilon* en  $x$ , genera un pequeño cambio *épsilon* en  $y$ .

$$f(x + \text{epsilon}_x) = y + \text{epsilon}_y$$

Como la función es *suave*, es decir, no tiene pendientes muy pronunciadas, cuando *epsilon\_x* es lo suficientemente pequeño, alrededor de un cierto punto  $p$ , es posible aproximar  $f$  como una función lineal de pendiente  $a$ , de modo que *epsilon\_y* se convierte en  $a * \text{epsilon}_x$ .

$$f(x + \text{epsilon}_x) = y + a * \text{epsilon}_x$$

# ¿Qué es una derivada?

- Está pendiente es la derivada de  $f$  en  $p$ . Si  $a$  es *negativa*, esto quiere decir que un pequeño cambio en  $x$  alrededor de  $p$  resultaría en una *disminución* de  $f(x)$ . Si por otra parte,  $a$  fuera *positiva*, un pequeño cambio en  $x$  resultaría en un *aumento* de  $f(x)$ .
- Además, el valor absoluto de  $a$  (la magnitud de la derivada) indicará qué tan rápido ocurrirá este aumento o disminución.
- Cabe resaltar, que la aproximación es válida solo cuando  $x$  está lo suficientemente cerca de  $p$ .





# ¿Qué es una derivada?

- Para cada función diferenciable  $f(x)$  (que se puede derivar), existe una función derivada  $f'(x)$  que mapea valores de  $x$  a la pendiente de la aproximación lineal local de  $f$  en esos puntos.
- Por ejemplo, la derivada de  $\sin(x)$  es  $\cos(x)$ , la derivada de  $f(x) = ax$  es  $f'(x) = a$ .
- En conclusión, la derivada describe como  $f(x)$  evoluciona al cambiar  $x$ . Si se quiere minimizar  $f(x)$ , solo debemos mover  $x$  en la dirección opuesta a la derivada (donde la derivada decrece).

# ¿Qué es una derivada?

- Para la función  $x^2$ , tenemos que su derivada es  $2x$ , ¿pero esto qué quiere decir geométricamente?
- Observe que para el punto  $x=2$  su derivada (su aproximación lineal) es 4 y para el punto  $x=3$  su derivada es 6.

x	f(x)	f'(x)
2	4	4
3	9	6
4	16	8



# Tabla de derivadas

Función	Derivada
$f(x) = n$	$f'(x) = 0$
$f(x) = x^n$	$f'(x) = nx^{n-1}$
$f(x) = \frac{1}{x}$	$f'(x) = \frac{-1}{x^2}$
$f(x) = u \cdot v$	$f'(x) = u' \cdot v + u \cdot v'$
$f(x) = \frac{u}{v}$	$f'(x) = \frac{u' \cdot v - u \cdot v'}{v^2}$
$f(x) = \sqrt{x}$	$f'(x) = \frac{1}{2\sqrt{x}}$
$f(x) = \sqrt[n]{x}$	$f'(x) = \frac{1}{n\sqrt[n]{x^{n-1}}}$



# El Algoritmo del Descenso del Gradiente

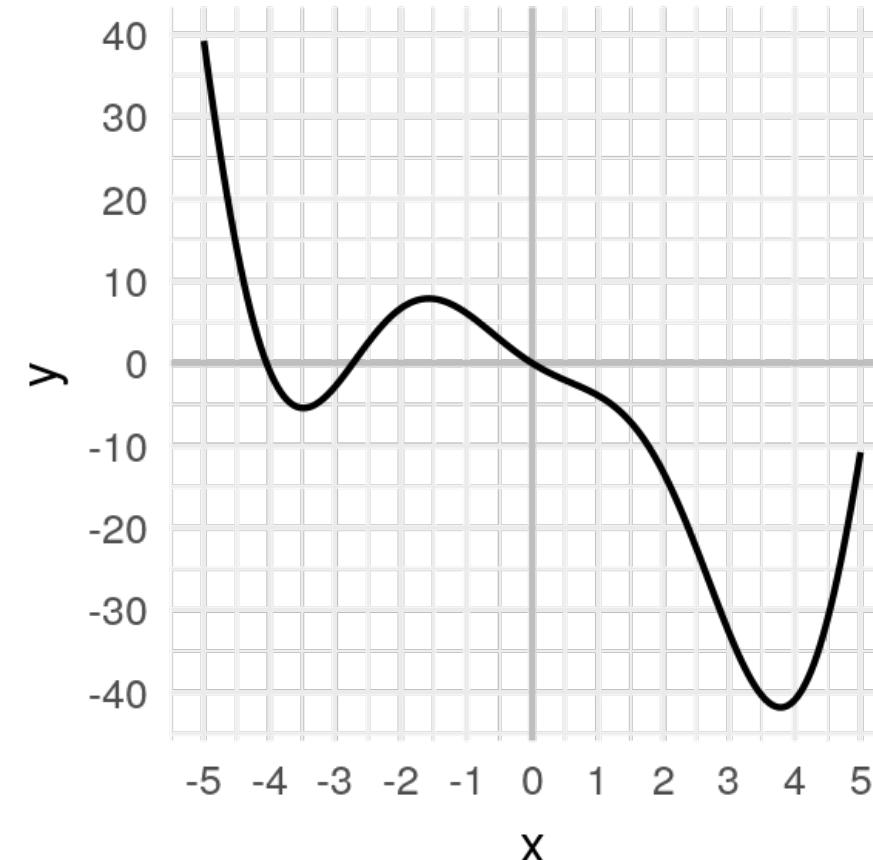
1. Tome un punto al azar  $x_0$ .
2. Calcule el valor de la pendiente (derivada)  $f'(x_0)$ .
3. Camine en dirección opuesta a la pendiente:

$$x_1 = x_0 - \eta \cdot f'(x_0)$$

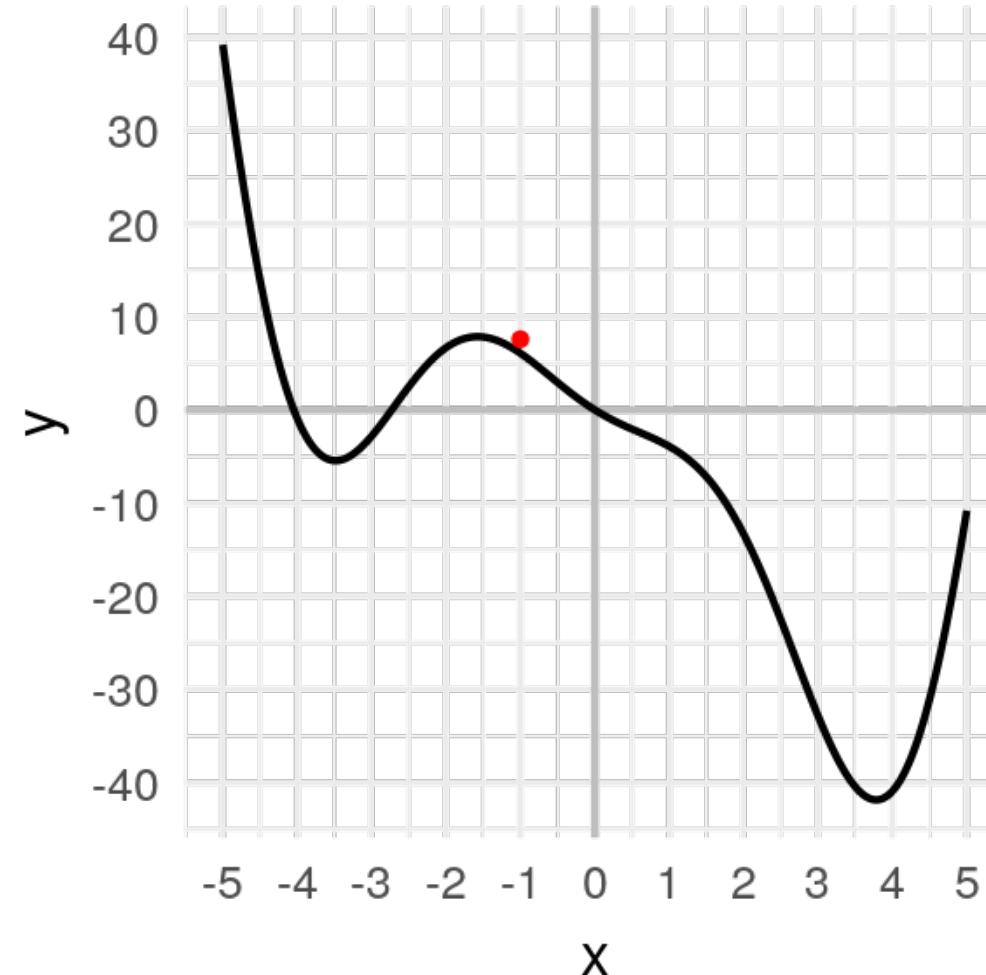
- Aquí,  $\eta$  es la tasa de aprendizaje que mencionamos anteriormente. Y el signo menos nos permite ir en la dirección opuesta.

# Ejemplo:

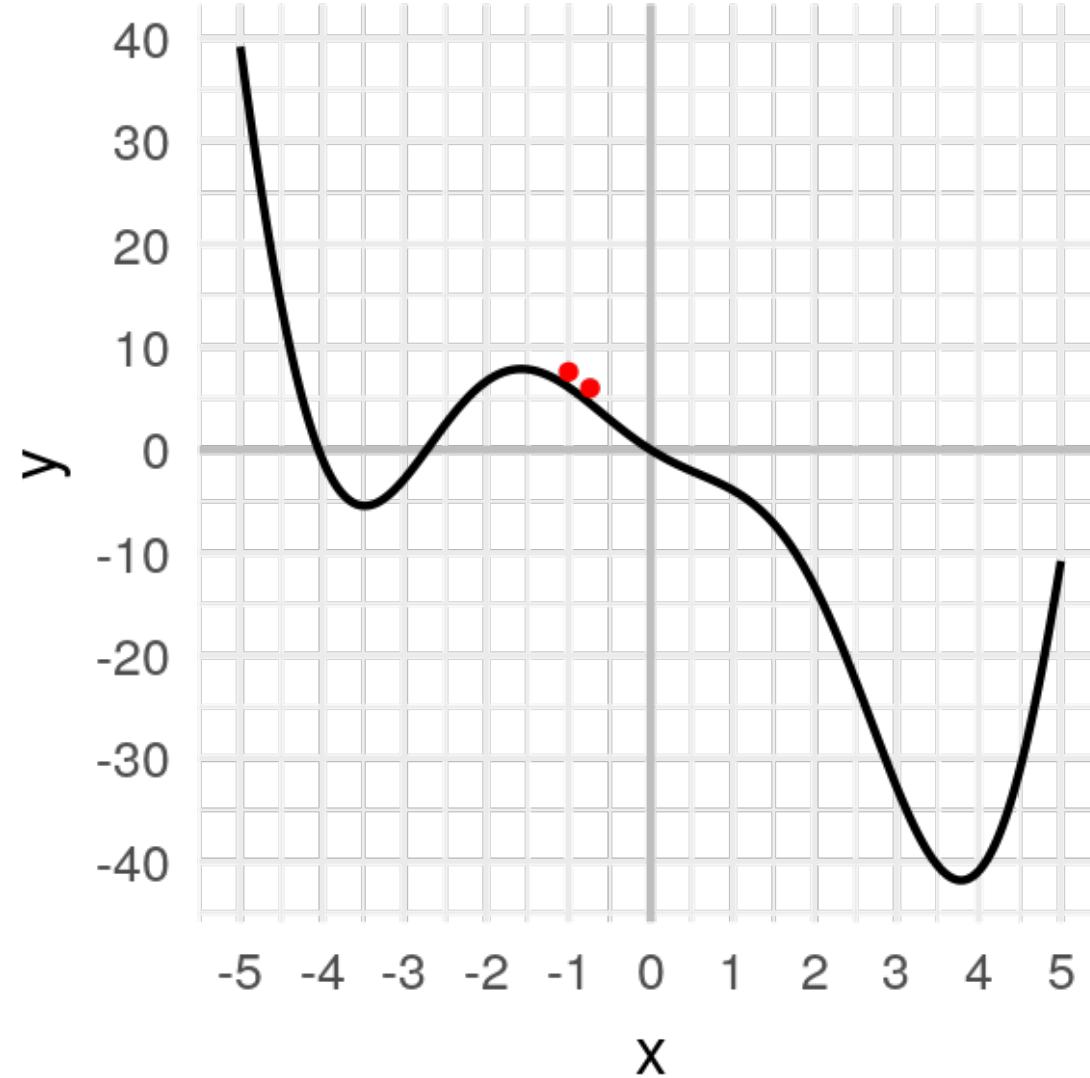
- Sea  $f(x)=2x^2\cos(x)-5x$  en el intervalo  $[-5,5]$



- **Paso 1:** Tome al azar  $x_0 = -1$ . Esto nos da  $f(x_0) = 6.08$ :



- **Paso 2:** Calculamos la derivada (pendiente), note que  $f'(x)=4x\cos(x)-2x^2\sin(x)-5$ , de donde la pendiente es  $f'(x_0) = -5.478$  (*pendiente negativa*).
- **Paso 3:** Si tomamos  $\eta=0.05$  entonces:
  - $x_1=x_0-\eta \cdot f'(x_0)=-0.726$ ,
  - Así se tiene que  $f(x_1)=4.419$ .
  - Graficamos el par  $(x_1, f(x_1))$  como sigue:



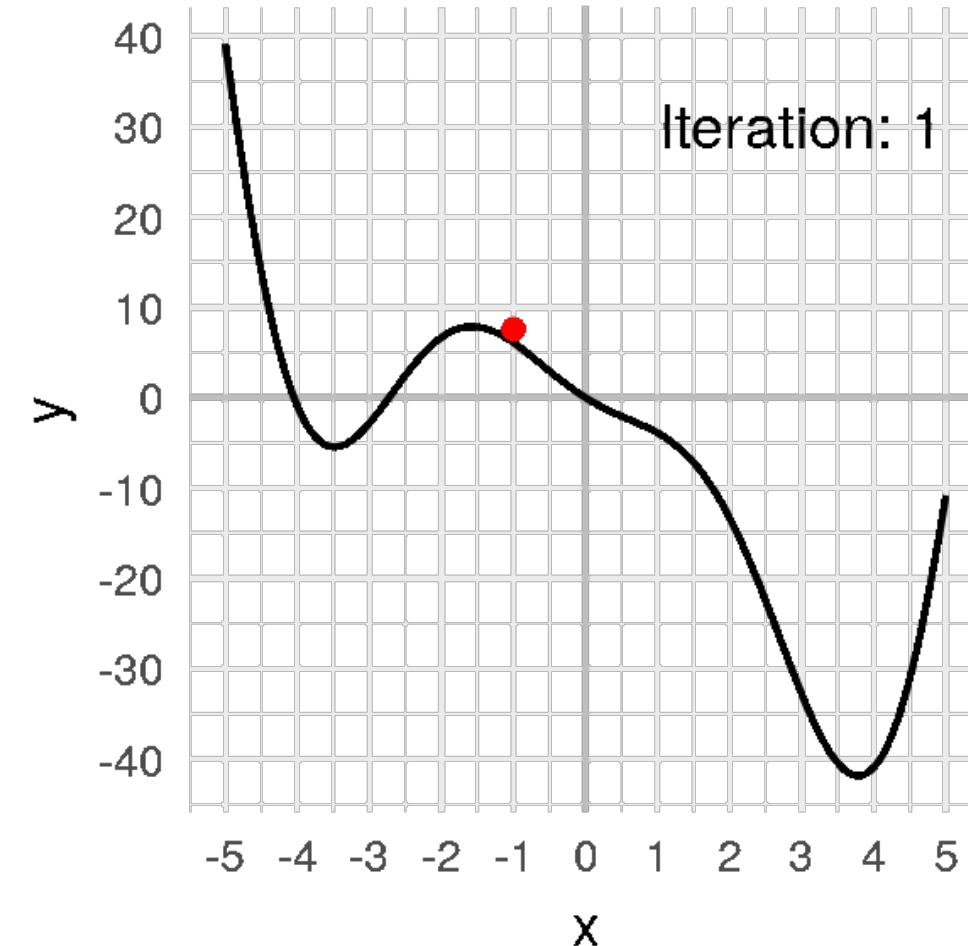


## Repetimos pasos 2 y 3:

- **Paso 2:** La pendiente es  $f'(x_1) = -5.478$  (*pendiente negativa*).
- **Paso 3:** Si tomamos  $\eta = 0.05$  entonces:
  - $x_2 = x_1 - \eta \cdot f'(x_0) = -0.402$ ,
  - Así se tiene que  $f(x_2) = 2.311$ .
  - Graficamos el par  $(x_2, f(x_2))$  como sigue:

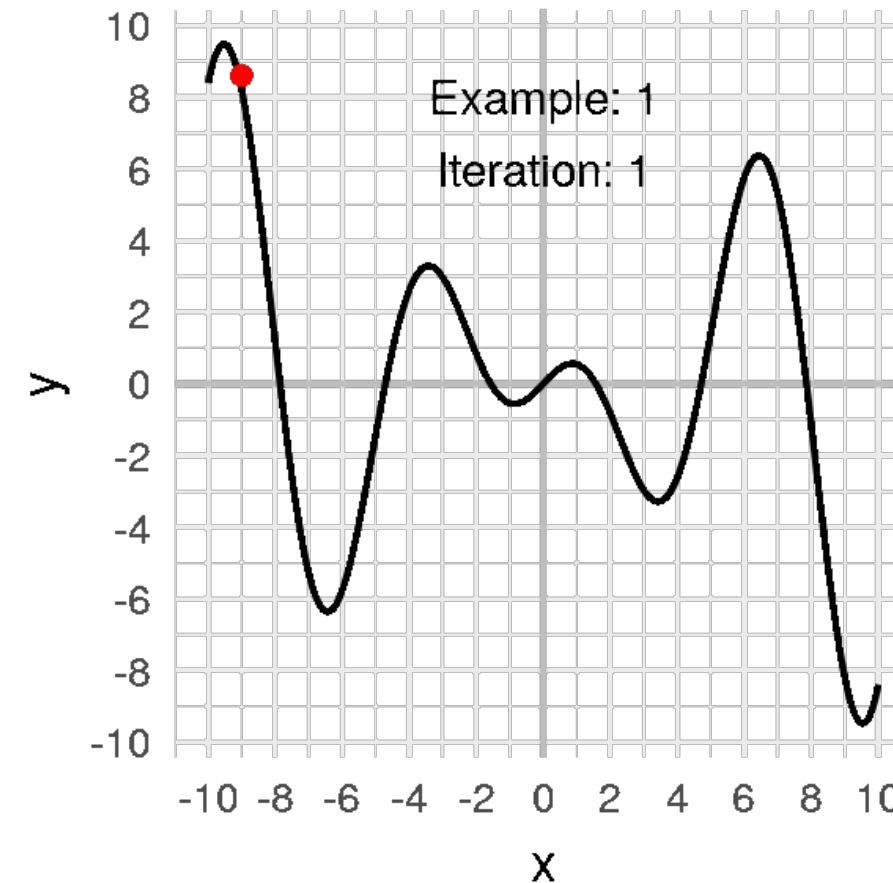


# Iterando 20 veces:



# ¿Qué hacer en caso de mínimos locales?

- Considere:  $f(x)=x \cos(x)$ .





# Derivada de una operación tensorial: El Gradiente

- El *gradiente* es la derivada de una operación tensorial. Es la generalización del concepto de derivadas a funciones que toman tensores como entradas.
- Considere un vector de entrada  $x$ , una matriz  $W$ , un vector objetivo  $y$  y una función de pérdida *loss\_value*. Podemos utilizar  $W$  para calcular un  $y_{pred}$  y calcular la pérdida o la diferencia entre la predicción  $y_{pred}$  y el valor real  $y$ .

```
y_pred = dot(W, x)
```

```
loss_value = loss(y_pred, y)
```



# Derivada de una operación tensorial: El Gradiente

- Digamos que el valor actual de  $W$  es  $W_0$ . Entonces la derivada de  $f$  en el punto  $W_0$  es un tensor  $\text{gradiente}(f)(W_0) = \nabla f(W_0)$  con la misma forma que  $W$ , donde cada coeficiente del  $\text{gradiente}(f)(W_0)[i, j]$  indica la dirección y magnitud del cambio en *loss\_value* que se observa al modificar  $W_0[i, j]$ .
- Anteriormente vimos que la derivada de una función  $f(x)$  de un coeficiente simple se puede interpretar como la pendiente de la curva de  $f$ . Asimismo,  $\nabla f(W_0)$  se puede interpretar como el tensor que describe la *curvatura* de  $f(W)$  alrededor de  $W_0$ .



# Derivada de una operación tensorial: El Gradiente

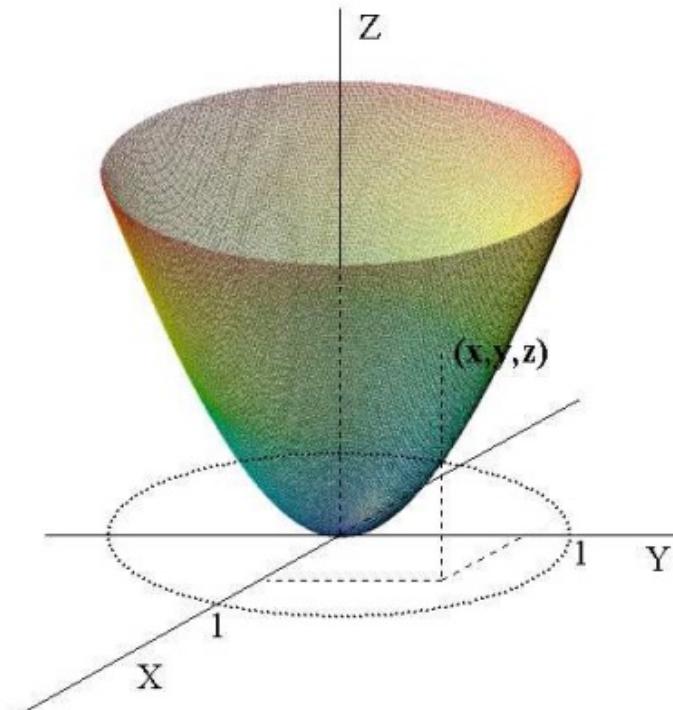
- Por esta razón, de la misma manera que, para una función  $f(x)$ , se puede reducir el valor de  $f(x)$  moviendo  $x$  un poco en la dirección opuesta a la derivada (sentido de decrecimiento), con una función  $f(W)$  de tensor, puede reducir  $f(W)$  moviendo  $W$  en la dirección opuesta al gradiente.
- Por ejemplo,  $W_1 = W_0 - \text{paso} \cdot \nabla f(W_0)$  (donde **paso** =  $\eta$  es un factor de escala pequeño).
- Eso significa ir en contra de la curvatura, que intuitivamente debería ubicarte más abajo en la curva. Tenga en cuenta que el paso del factor de escala es necesario porque  $\nabla f(W_0)$  solo se aproxima a la curvatura cuando está cerca de  $W_0$ , por lo que no quiere alejarse demasiado de  $W_0$ .

# Derivada de una operación tensorial: El Gradiente

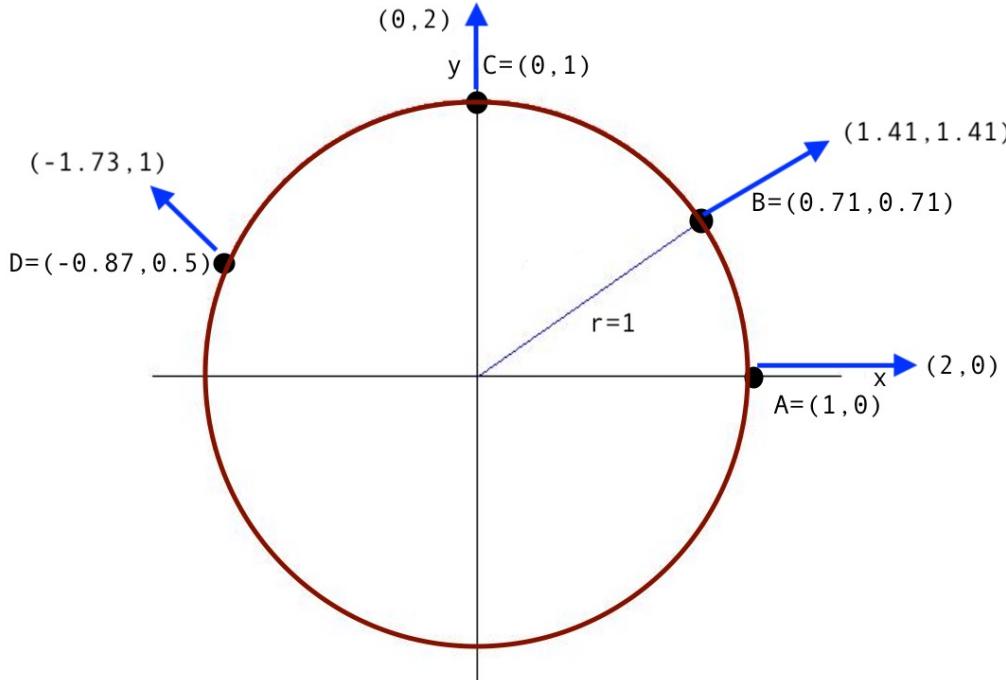
Tome ahora el ejemplo de la función  $f(x,y) = x^2 + y^2$  y vea que su gradiente es igual a:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

Para comprender esto mejor, veámoslo en el plano  $xy$  con radio igual a 1.



# Derivada de una operación tensorial: El Gradiente



- $\nabla f(A) = (2, 0)$
- $\nabla f(B) = (\sqrt{2}, \sqrt{2})$
- $\nabla f(C) = (0, 2)$
- $\nabla f(D) = (\sqrt{3}, 1)$

Note que los nuevos puntos buscan alejarse del origen (el punto más bajo del paraboloide). Por esto, debo ir en dirección contraria al gradiente (donde decrece), para encontrar el punto más bajo.



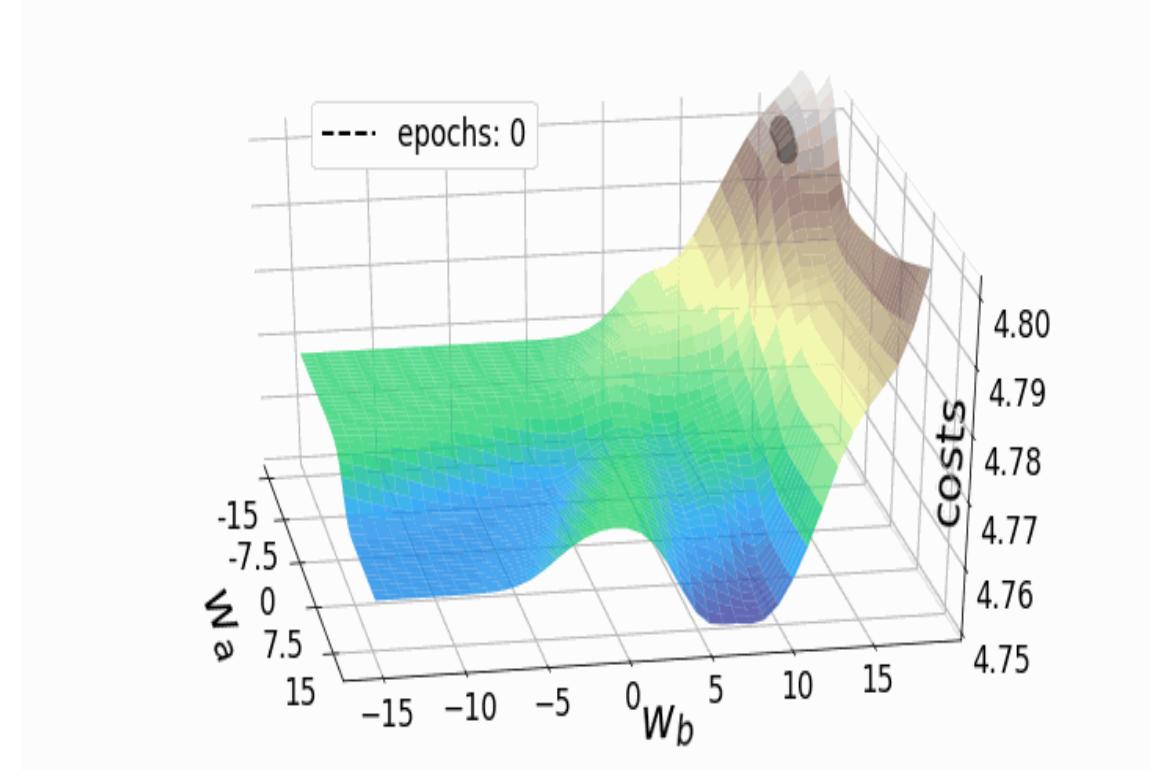
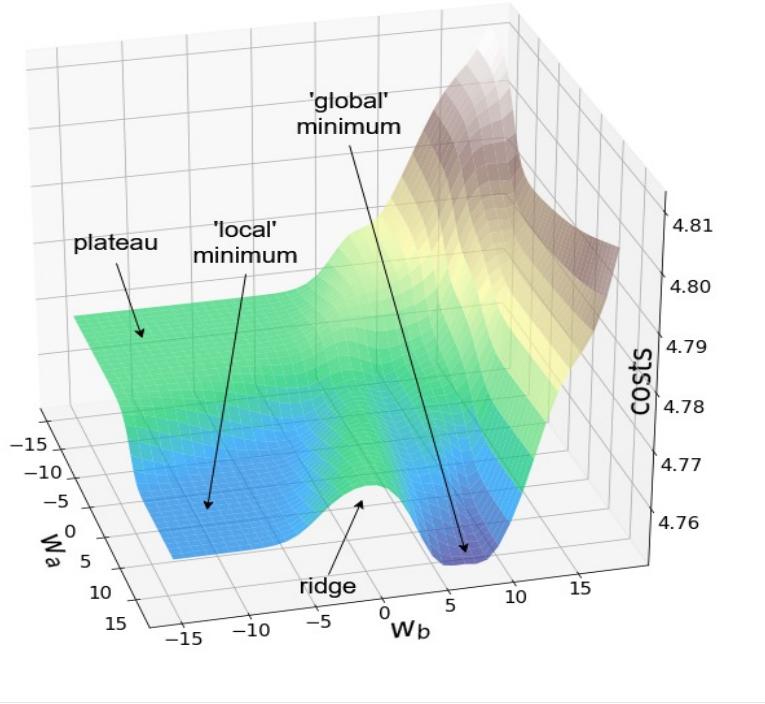
# Interpretando el gradiente en tres dimensiones

Supongamos que  $\mathbf{f}$  está en un plano tridimensional, su gradiente  $\nabla\mathbf{f}$  contiene toda la información de sus derivadas parciales en un vector.

$$\nabla\mathbf{f} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{bmatrix}$$

Es decir,  $\mathbf{f}$  es una función vectorial. Ahora sea  $(x_0, y_0, z_0)$  un punto en el espacio. ¿Qué nos dice  $\nabla\mathbf{f}(x_0, y_0, z_0)$ ? El gradiente entonces apuntará en la dirección para incrementar  $\mathbf{f}$ , en dirección a la cima de la función.

# Interpretando el gradiente en tres dimensiones

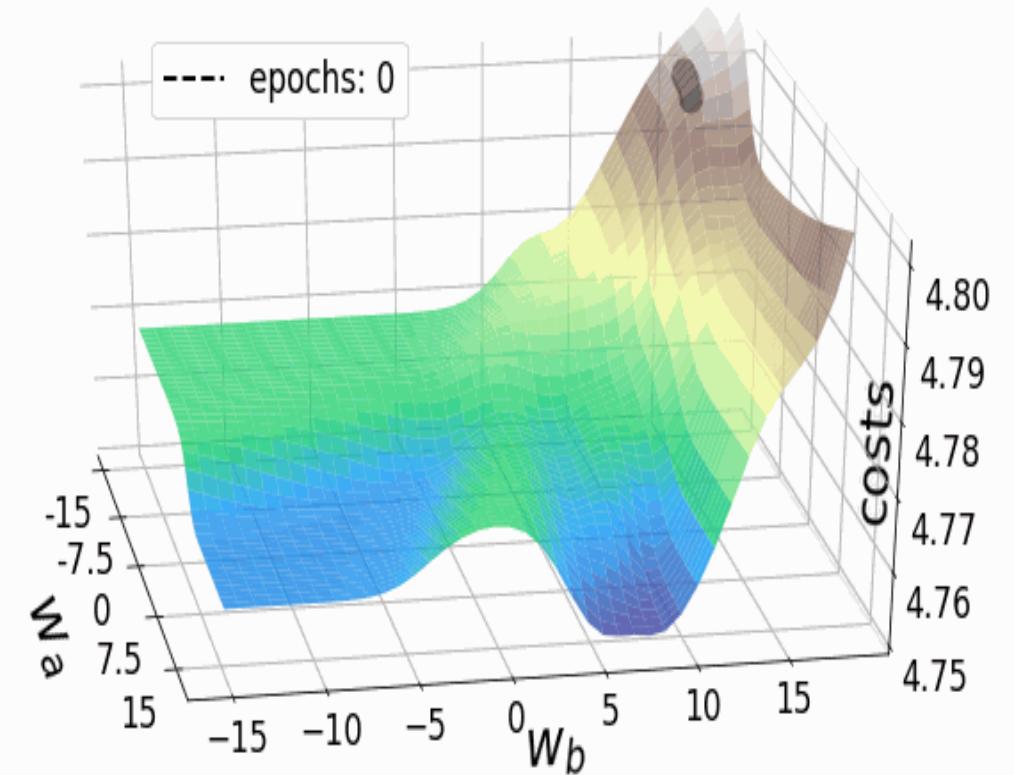


**epochs:** Término utilizado en Machine Learning para indicar la cantidad de ciclos que el conjunto de training ha completado.

# Ejemplo

Si  $f(x,y,z) = x^2 - 5xy + z$ , entonces el  $\nabla f$  es el siguiente:

$$\nabla f = \begin{bmatrix} 2x - 5y \\ -5x \\ 1 \end{bmatrix}$$



# Formalmente:

- ¿Qué es el *gradiente*?
- Dada una función  $f: D \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$  diferenciable en  $x$  en  $D$ , se define el *gradiente* como el vector
$$\nabla f(x) := \left( \frac{\partial f(x)}{\partial x_1}, \dots, \frac{\partial f(x)}{\partial x_n} \right).$$
- Es un vector normal al hiperplano tangente en  $x$  la hipersuperficie de  $f(x) = 0$ .
- Apunta en la dirección de máximo crecimiento de la función  $f$ .

# El Algoritmo del Descenso del Gradiente

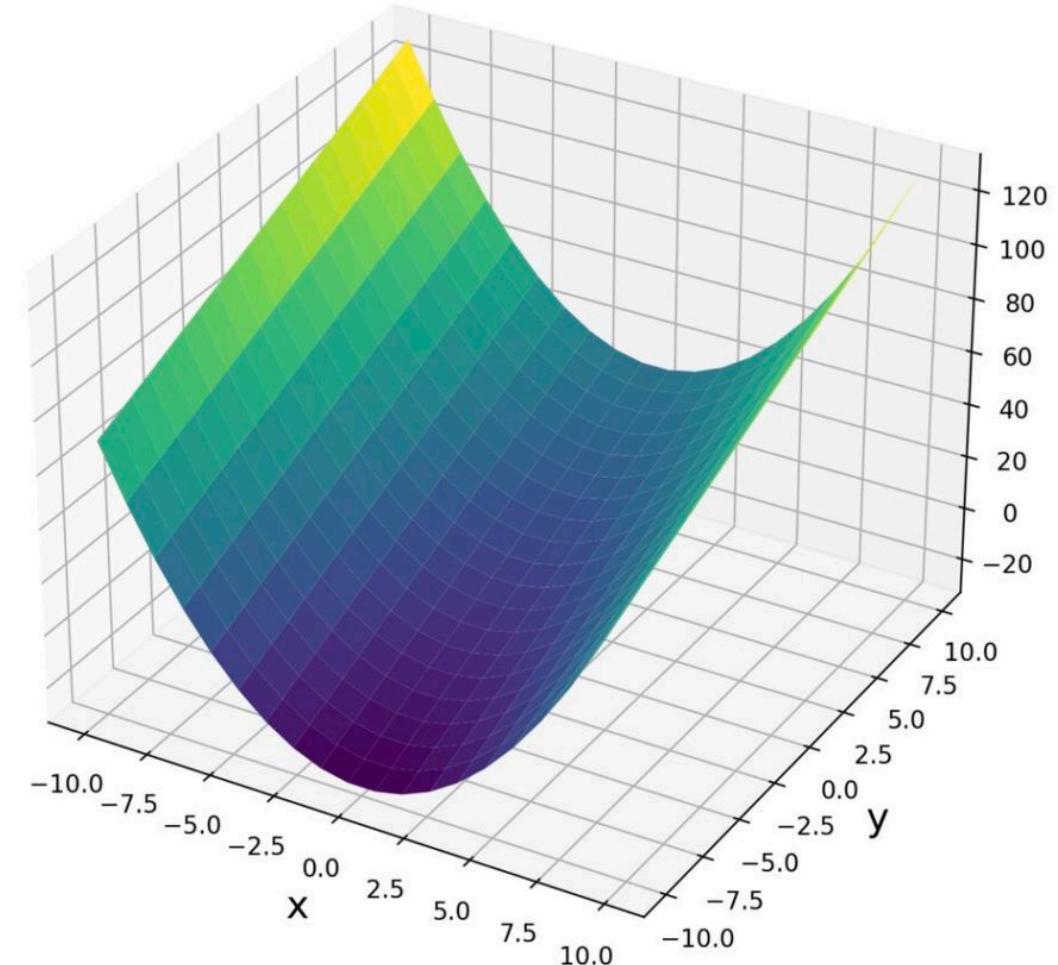
1. Tome un punto al azar  $x_0$ .
2. Calcule el gradiente de  $f(x)$  en  $x_0$ , dado por:  $\nabla f(x_0)$
3. Camine en dirección opuesta a la pendiente:

$$x_1 = x_0 - \eta \cdot \nabla f(x_0)$$

- Aquí,  $\eta$  es la tasa de aprendizaje que mencionamos anteriormente. Y el signo menos nos permite ir en la dirección opuesta.

# Ejemplo en 3D

$$f(x, y) = x^2 + 3y$$



# Ejemplo

Si  $f(x,y) = x^2 + 3y$ , entonces el  $\nabla f$  es el siguiente:

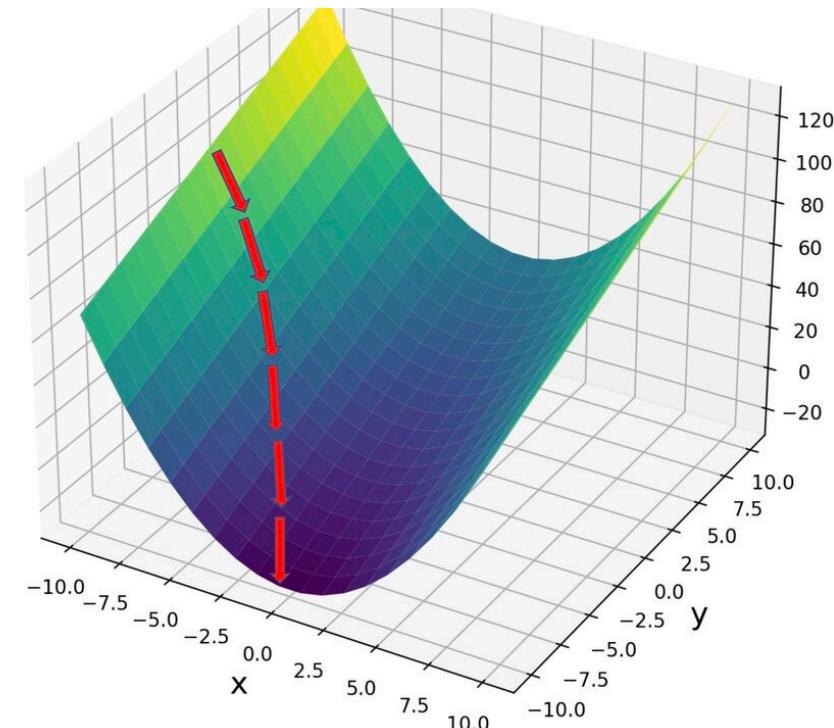
$$\nabla f = \begin{bmatrix} 2x \\ 3 \end{bmatrix}$$

Sea  $x_0 = \begin{bmatrix} -10 \\ 10 \end{bmatrix}$  entonces  $f(x_0) = 100 + 30 = \mathbf{130}$ .

Tomamos  $\eta = 0.1$  entonces:

$$x_1 = \begin{bmatrix} -10 \\ 10 \end{bmatrix} - 0.1 * \begin{bmatrix} -20 \\ 3 \end{bmatrix} = \begin{bmatrix} -8 \\ 9.7 \end{bmatrix}$$

entonces  $f(x_1) = 64 + 29.1 = \mathbf{93.1}$ .



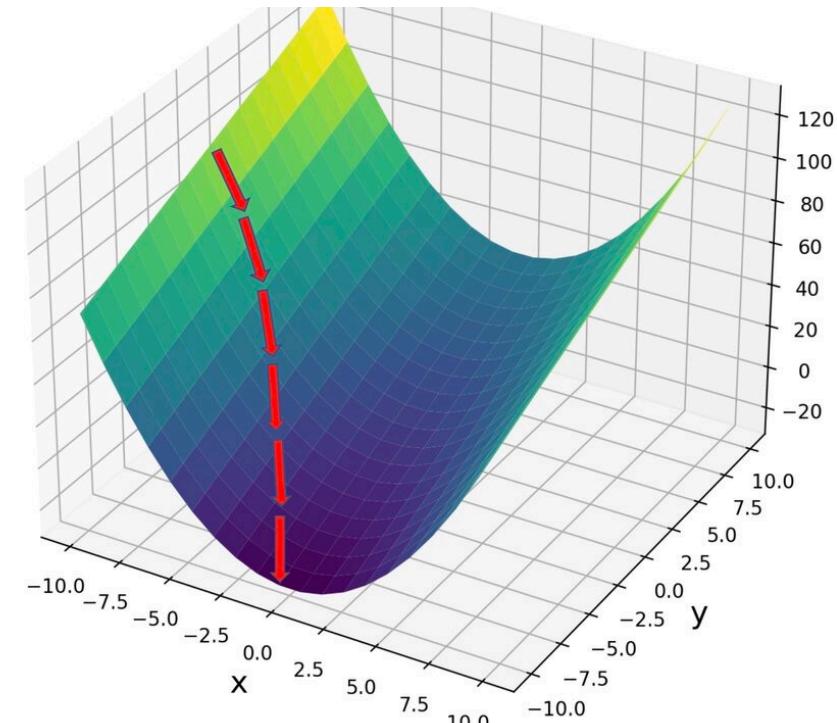
# Ejemplo

$$x_2 = \begin{bmatrix} -8 \\ 9.7 \end{bmatrix} - 0.1 * \begin{bmatrix} -16 \\ 3 \end{bmatrix} = \begin{bmatrix} -6.4 \\ 9.4 \end{bmatrix}$$

entonces  $f(x_2) = 40.96 + 28.2 = \mathbf{69.16}$ .

$$x_3 = \begin{bmatrix} -6.4 \\ 9.4 \end{bmatrix} - 0.1 * \begin{bmatrix} -12.8 \\ 3 \end{bmatrix} = \begin{bmatrix} -5.12 \\ 9.1 \end{bmatrix}$$

entonces  $f(x_2) = 26.21 + 27.3 = \mathbf{53.51}$ .

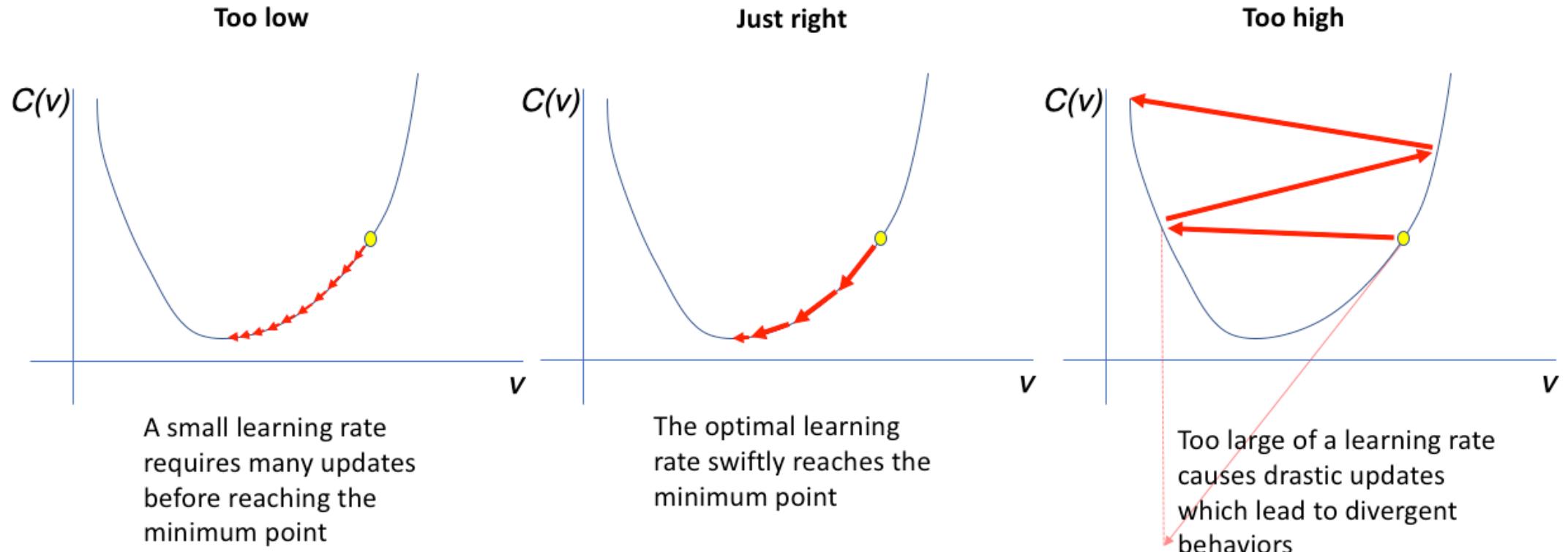




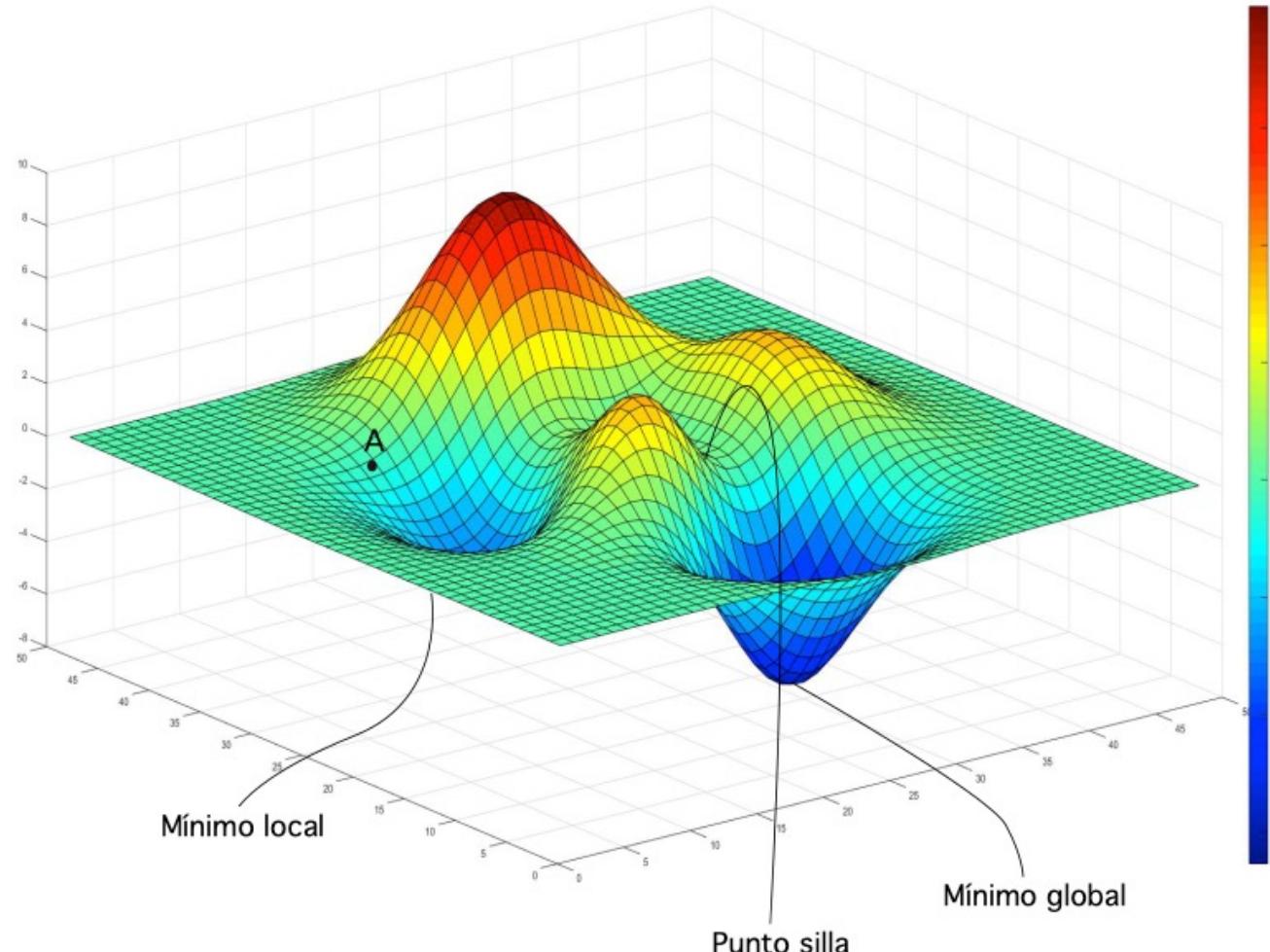
# Problemas del Método

1. Tasa de aprendizaje.
2. Mínimos locales.
3. Puntos Silla.
4. Valor inicial  $v^*$ .

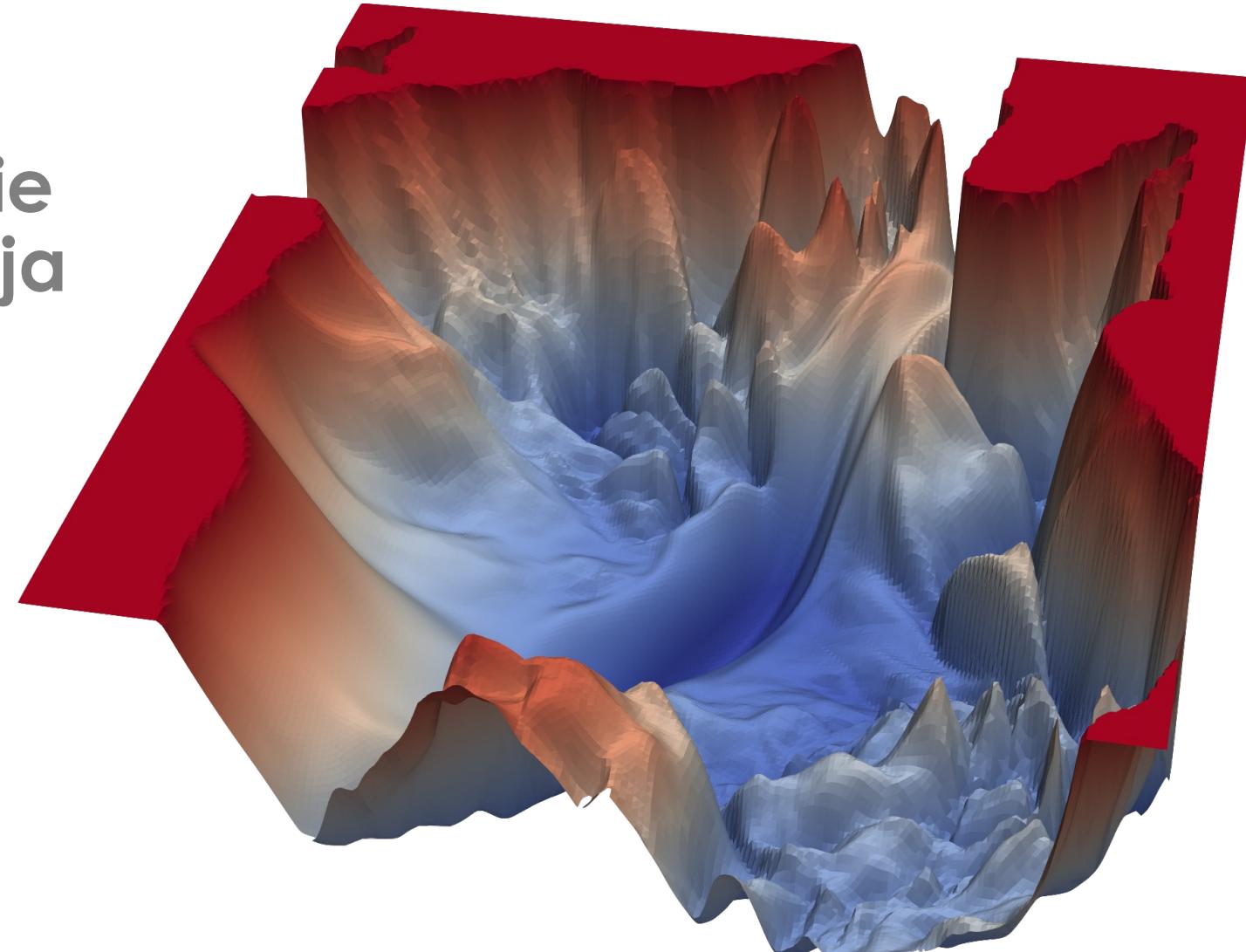
# Tasa de Aprendizaje



# Mínimos Locales



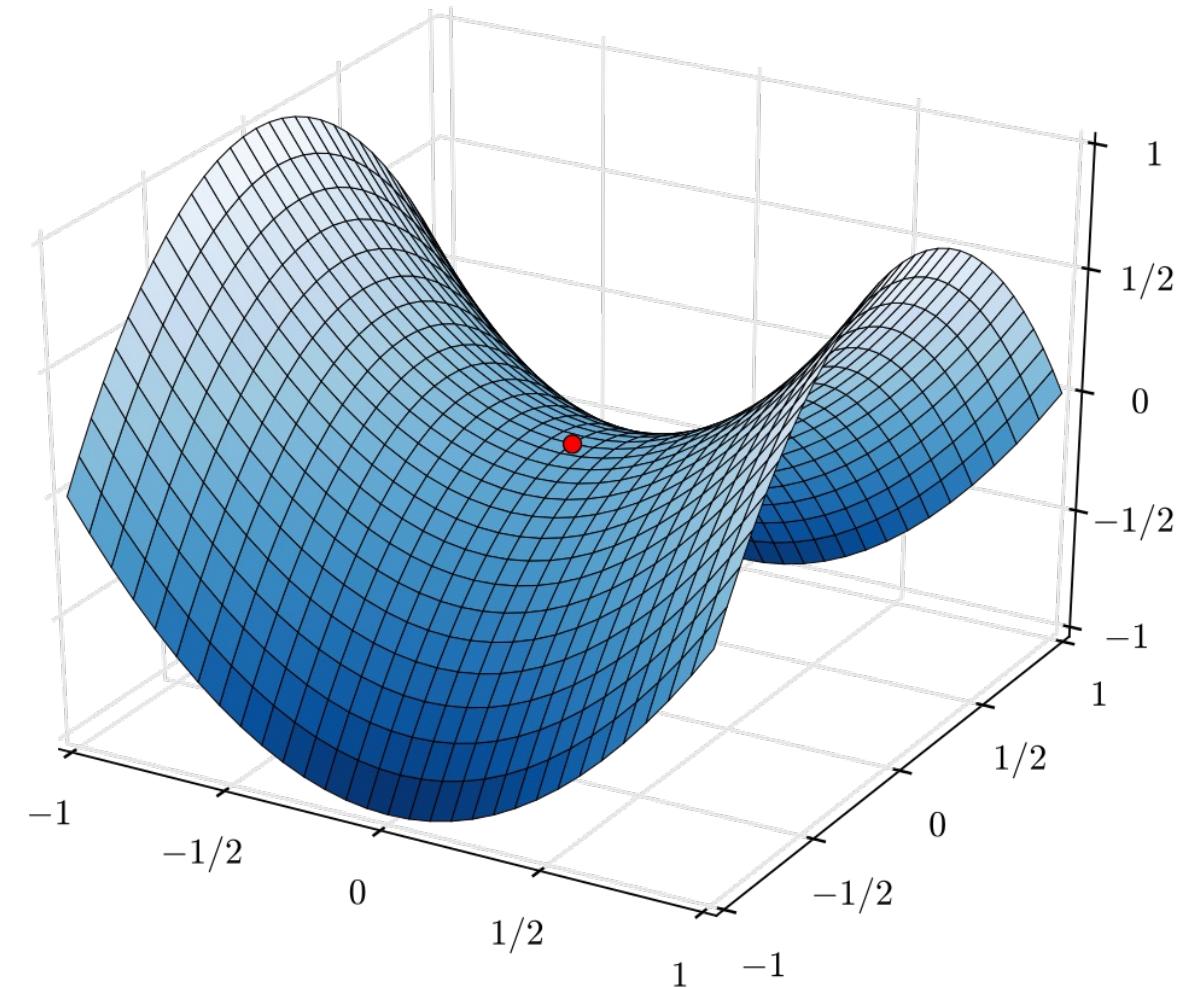
# Una Superficie más Compleja



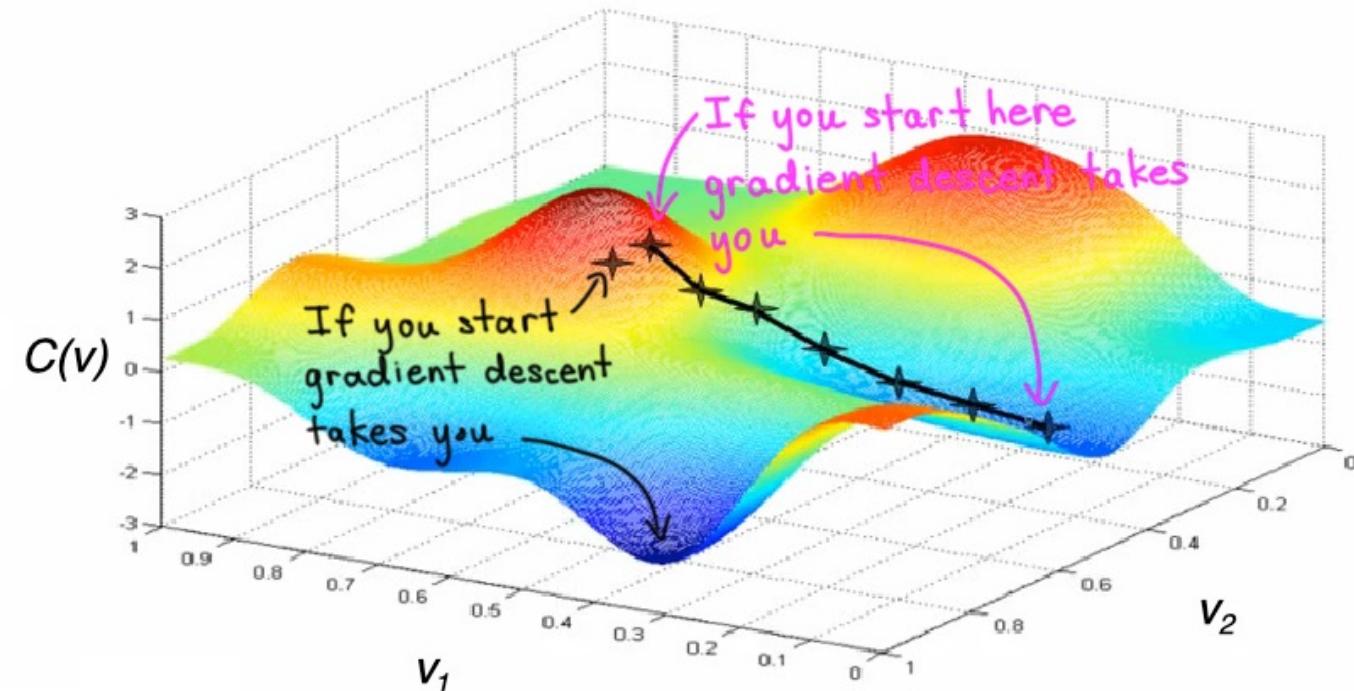
Tomada de:

<https://www.cs.umd.edu/~tomg/projects/landscapes/>

# Puntos Silla



# Valor Inicial $v^*$

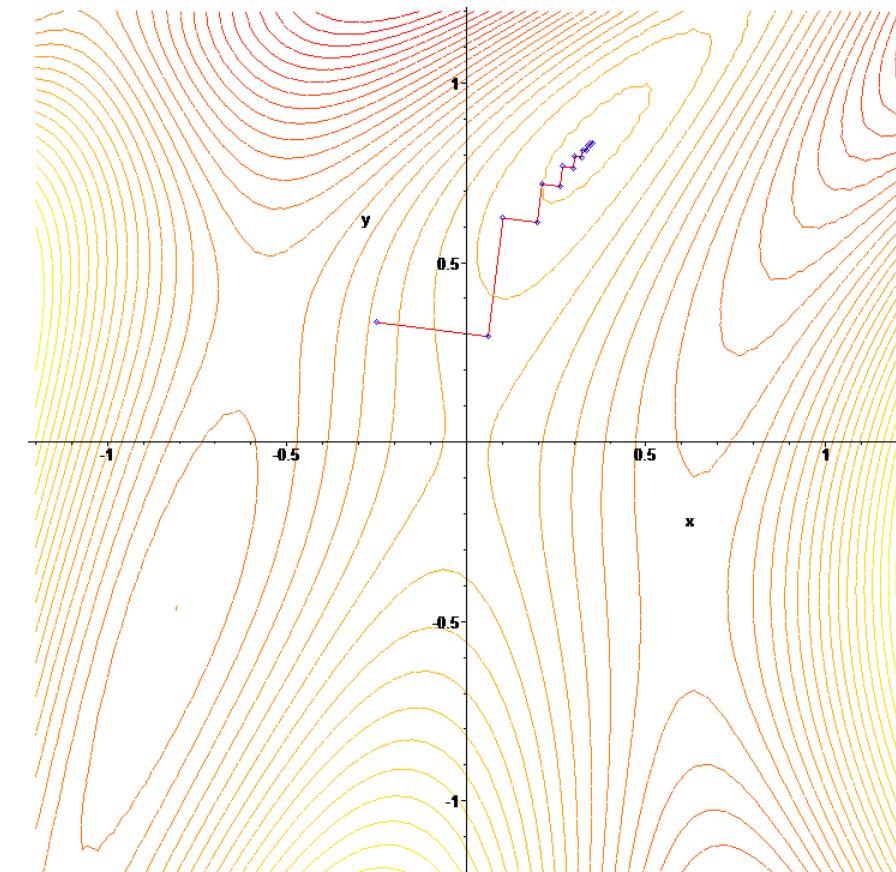
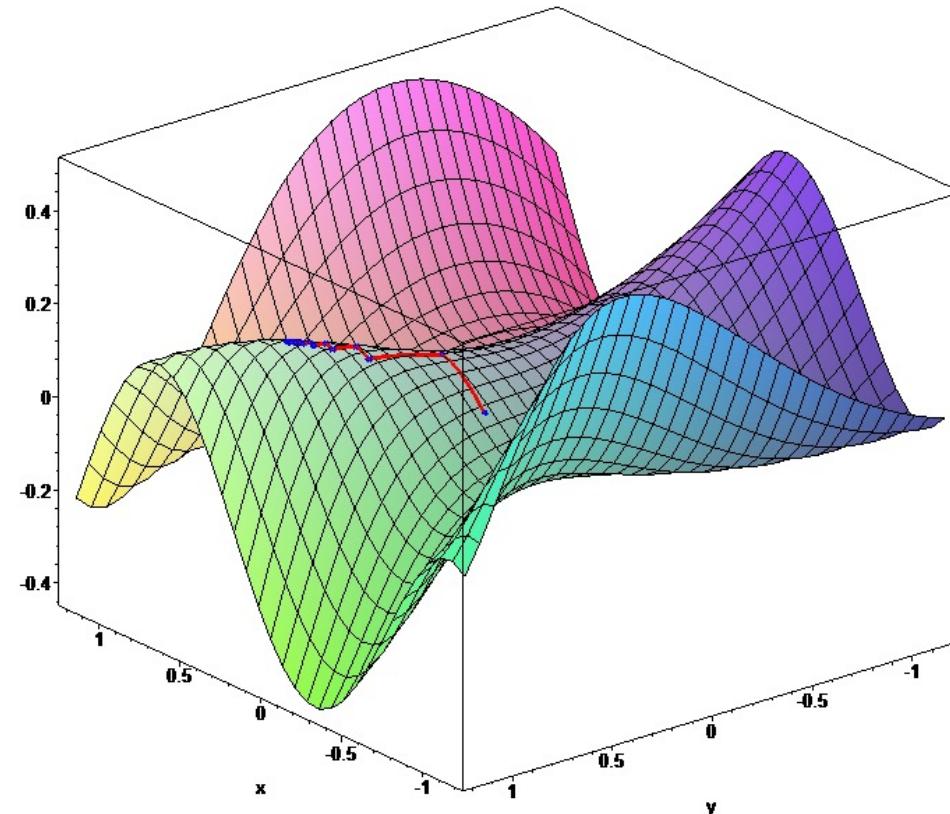


# Possible solución a estos problemas

- Para 1: Usar una tasa de aprendizaje variable que disminuye cada cierto número de iteraciones.
- Para 2, 3 y 4: Usar *descenso de gradiente estocástico* que usa en cada iteración una función de costo para una muestra aleatoria de individuos.

¡Estos son temas más avanzados!

$$F(x, y) = \sin\left(\frac{1}{2}x^2 - \frac{1}{4}y^2 + 3\right) \cos(2x + 1 - e^y).$$



# Aplicado a una Función de Costo

- Buscamos un algoritmo que encuentre los pesos y el bias de forma que el output aproxime  $y(x)$  para todo  $x$  de entrenamiento, con  $y(x)$  es la variable a predecir.
- Para cuantificar qué tan bien alcanzamos el objetivo definimos:

$$C(\mathbf{w}, \mathbf{b}) = \frac{1}{2n} \sum_x \|y(x) - a\|^2,$$

donde  $w$  son los pesos,  $b$  los bias,  $n$  cantidad individuos de training,  $a = a(x, \mathbf{w}, \mathbf{b})$  el output de la red donde  $x$  es el input y.  $\|v - u\|$  denota distancia entre vectores.

- $C$  se llama función de costo o función de pérdida.



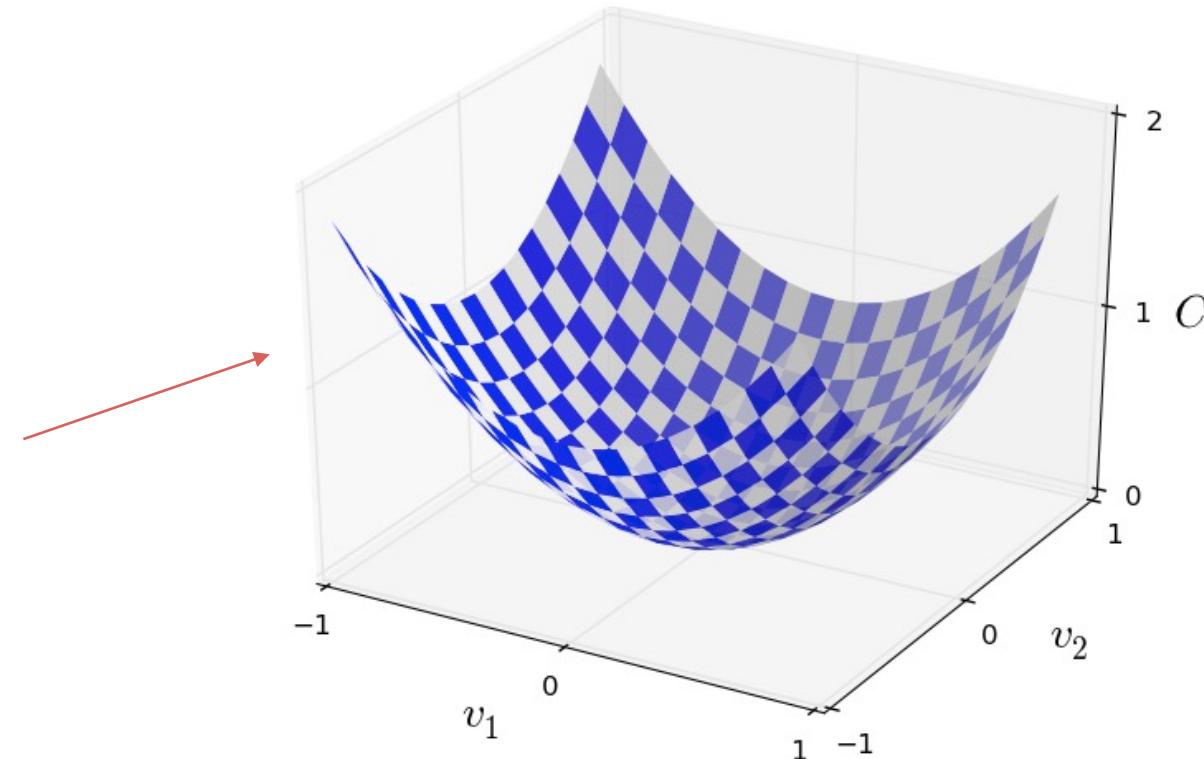
# Función de Costo

- El algoritmo ha entrenado bien cuando  $C(w, b)$  es pequeño.
- El algoritmo ha entrenado mal cuando  $C(w, b)$  es grande.
- El objetivo es encontrar un conjunto de pesos  $w$  y bias  $b$  que minimicen la función  $C(w, b)$ . ¿Será fácil?
- Esto se logra con el algoritmo de *Descenso de Gradiente*.

# Descenso de Gradiente

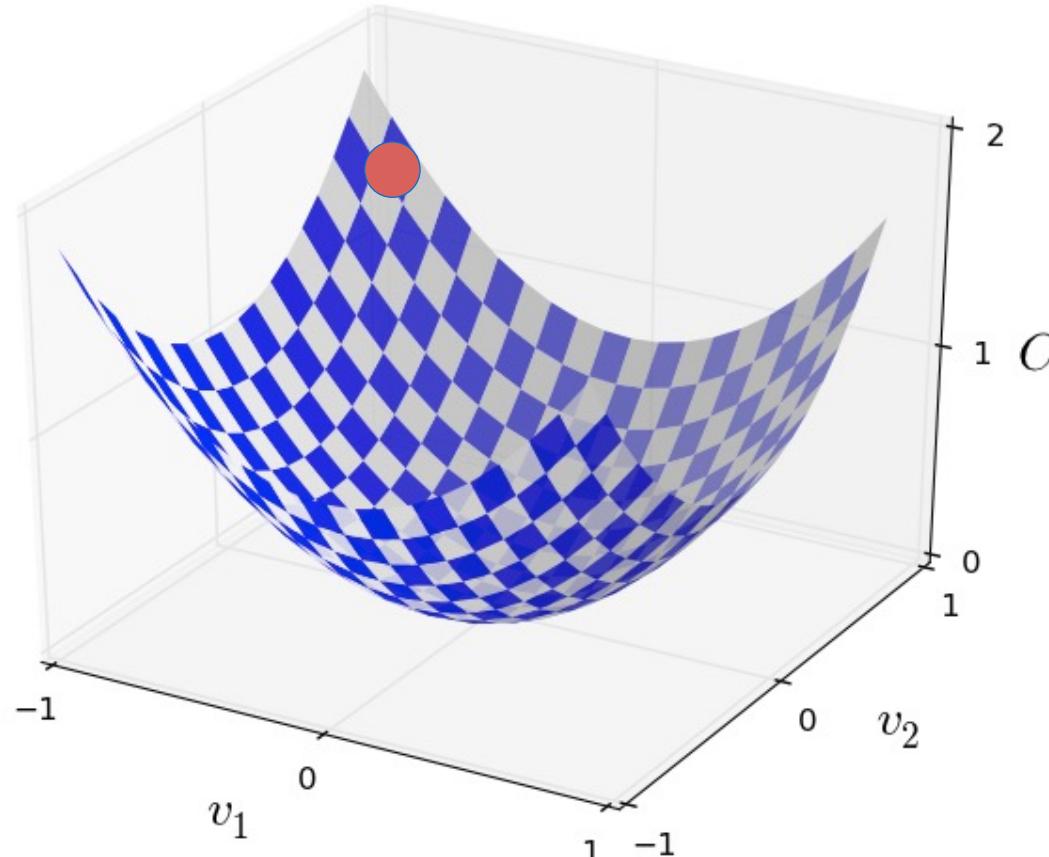
- Queremos minimizar la función *real en varias variables*  $C(v)$ , con  $v = (v_1, v_2, \dots, v_r)$ .
- Usemos un ejemplo simple con  $r = 2$ . ¡Las ideas aplican en general!

Suponga que  
 $C(v)$  tiene esta  
forma:



# Descenso de Gradiente

- Podríamos usar otras técnicas de optimización para minimizar  $C$ .
- Quizá funcionaría cuando  $C$  sea una función simple y de pocas variables.
- Para redes neuronales normalmente  $C$  es función de miles de parámetros  $w$ .
- ¡Usar cálculo no es viable! Debemos usar métodos computacionales como el *Descenso de Gradiente* .



- Imagine que de un punto aleatorio hacemos rodar una pelota en la superficie de  $C$ .
- *¿Qué pasaría?*
- La pelota eventualmente llegará al punto mínimo.
- Podemos simular este comportamiento con un algoritmo en una computadora y usando algunas **derivadas** de  $C$ .

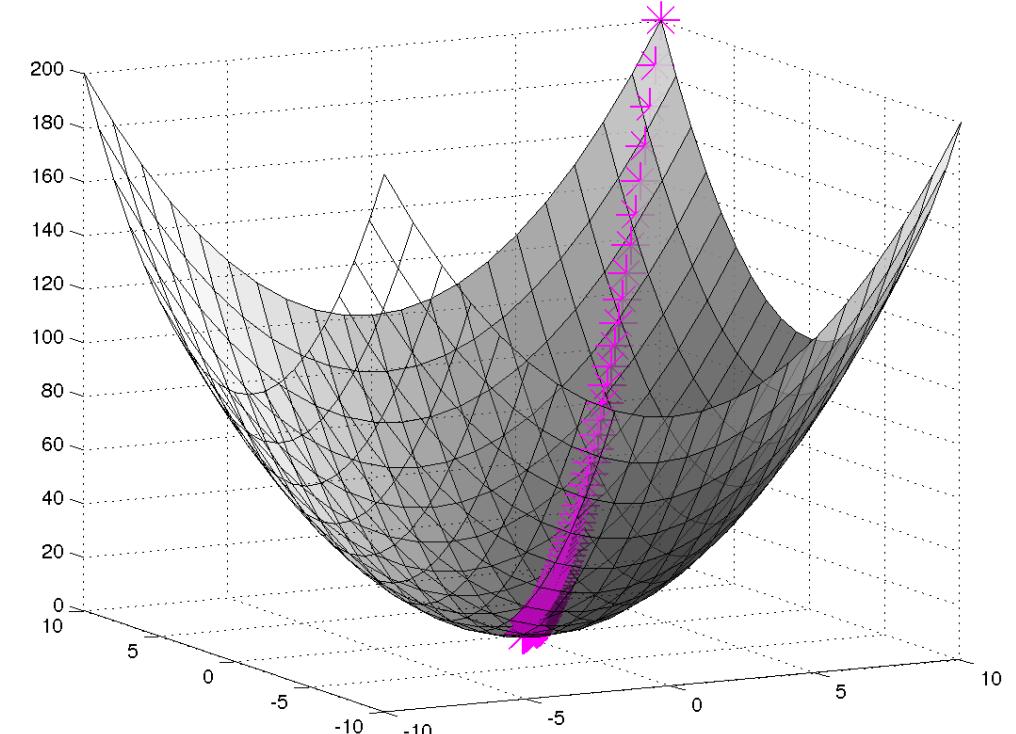
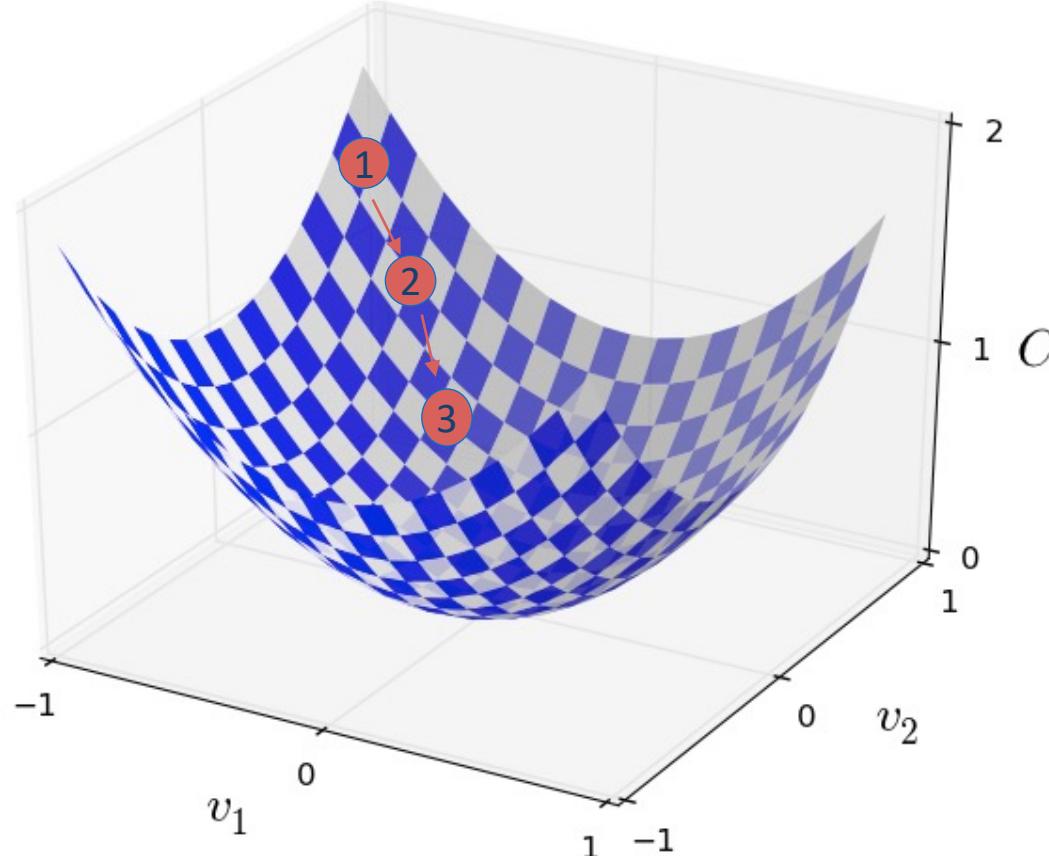
# El Algoritmo del Descenso del Gradiente

1. Tome un punto al azar  $x_0$ .
2. Calcule el gradiente de  $C(w)$  en  $x_0$ , dado por:  $\nabla C(x_0)$
3. Camine en dirección opuesta a la pendiente:

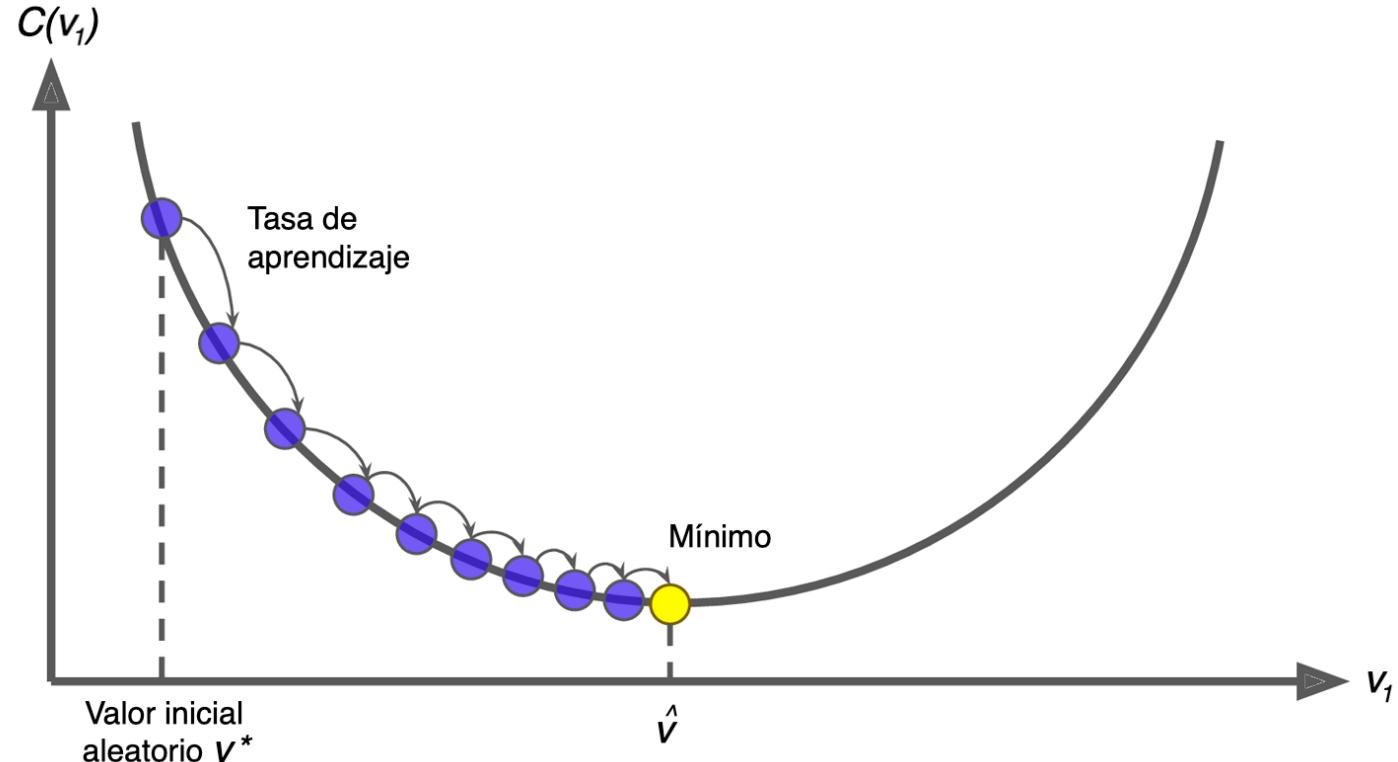
$$x_1 = x_0 - \eta \cdot \nabla C(x_0)$$

- Aquí,  $\eta$  es la tasa de aprendizaje que mencionamos anteriormente. Y el signo menos nos permite ir en la dirección opuesta.

# Gráficamente: $v = (v_1, v_2)$



# Gráficamente: $v = v_1$





# Agregando un “Momentum” (Impulso)

Cuando usamos el descenso de gradiente, nos encontramos con los siguientes problemas:

1. Quedarse atrapado en un mínimo local.
2. Sobrepasar y perder el óptimo global, esto es un resultado directo de moverse demasiado rápido a lo largo de la dirección del gradiente
3. Oscilación, este es un fenómeno que ocurre cuando el valor de la función no cambia significativamente sin importar la dirección en la que avanza. Puedes pensar en ello como navegar por una meseta, estás a la misma altura sin importar a dónde vayas

***Para combatir estos problemas, un término de impulso  $\alpha$  se agrega a para estabilizar la tasa de aprendizaje al avanzar hacia el valor óptimo global, como se muestra en la siguiente versión del algoritmo.***



# El Algoritmo del Descenso del Gradiente

1. Tome un punto al azar  $x_0$ .
2. Calcule el gradiente de  $C(w)$  en  $x_0$ , dado por:  $\nabla C(x_0)$
3. Camine en dirección opuesta a la pendiente:

$$x_1 = \alpha \cdot x_0 - \eta \cdot \nabla C(x_0)$$

- Aquí,  $\eta$  es la tasa de aprendizaje que mencionamos anteriormente. Y el signo menos nos permite ir en la dirección opuesta.  $\alpha$  se *llama Momentum o Impulso*.
- **$\alpha$  y  $\eta$  deben estar en el intervalo  $[0, 1]$ .**



# Ejemplo con una Red Neuronal de una capa

- Supongamos tenemos una tabla de entrenamiento con  $n$  individuos y  $m$  variables.
- Vamos a usar una Red Neuronal de una capa (regresión), o sea, que para  $i = 1, 2, \dots, n$  se tiene:

$$a = a(x_i, \mathbf{w}) = o_i = w_0 + w_1 x_{i1} + w_2 x_{i2} + \dots + w_m x_{im}.$$

- donde  $x_i = (x_{i1}, x_{i2}, \dots, x_{im})$  y  $\mathbf{w} = (w_0, w_1, w_2, \dots, w_m)$  pesos de la red.
- Entonces función de costo es:  $C(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (t_i - o_i)^2$  donde  $t_i = y(x_i)$  la variable a predecir aplicada en  $x_i$ .

# Ejemplo con una Red Neuronal de una capa

- Entonces el gradiente de la función de costo es:

$$\nabla C(\mathbf{w}) = - \sum_{i=1}^n (t_i - o_i) x_i$$

- donde  $t_i = y(x_i)$  la variable a predecir aplicada en  $x_i$ ,  $\mathbf{w} = (w_0, w_1, w_2, \dots, w_m)$  y  $o_i = w_0 + w_1 x_{i1} + w_2 x_{i2} + \dots + w_m x_{im}$
- Ver ejecución en Python.*



# GRACIAS....