

Curso de

Introducción a Machine Learning

Adrián Catalán

Conceptos básicos de Machine Learning

Introducción

"IT'S AS **LARGE AS LIFE**
AND **TWICE AS NATURAL**".

~ Lewis Carroll





AI
Inteligencia Artificial



ML
Aprendizaje automático



DL
Aprendizaje profundo

Inteligencia Artificial

AI se refiere a la capacidad de una máquina de realizar tareas que requieren de la inteligencia de un ser humano, cumpliéndolas al mismo nivel o siendo mejor que una persona.

Machine Learning

Subcampo de AI que involucra brindar a máquinas, la habilidad a de realizar una tarea específica, sin necesidad de que esté programada explícitamente para realizarla.

Principales tipos de ML



Supervised learning

Aprendizaje supervisado

Unsupervised learning

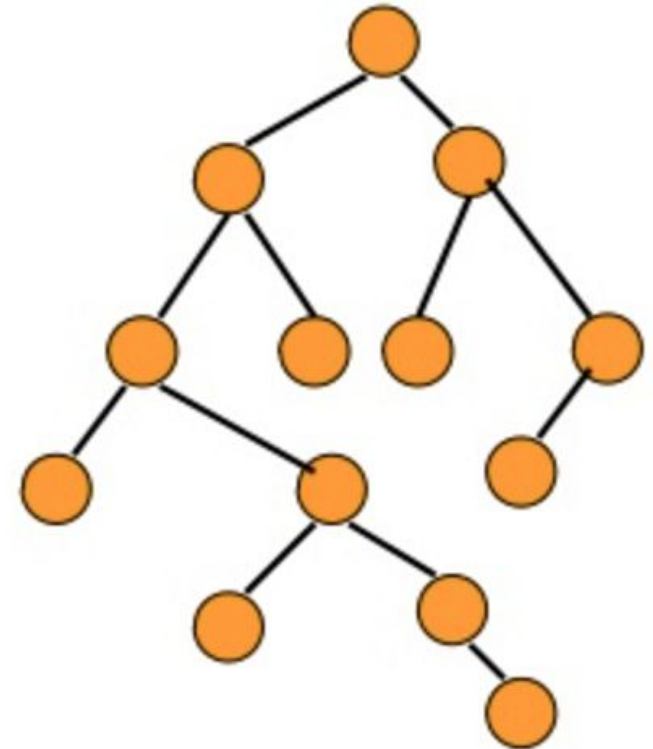
Aprendizaje sin supervisión

Reinforcement learning

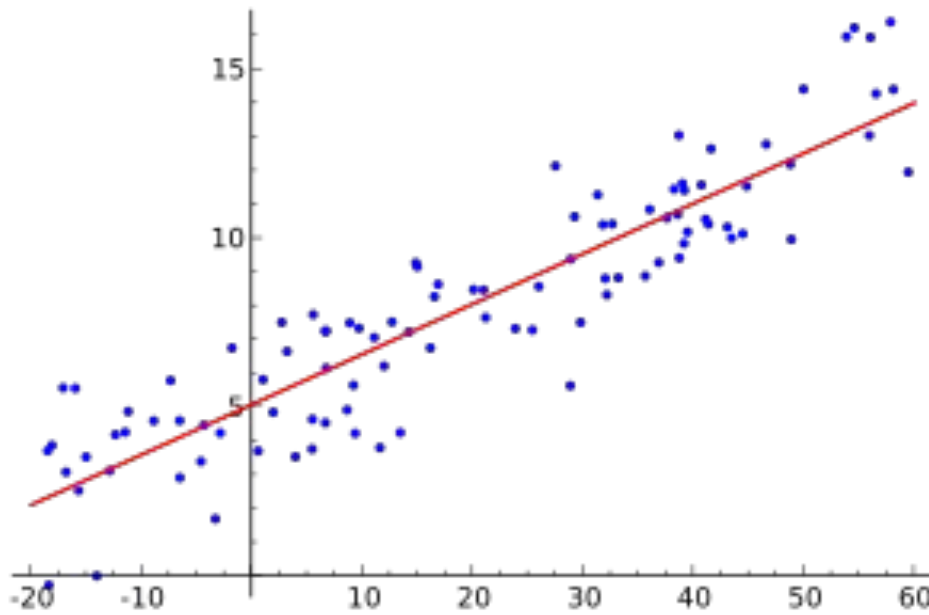
Aprendizaje por refuerzo autónomo

¿Debería comer tacos?

- ¿Tengo hambre?
- ¿Estoy solo o acompañado?
- ¿Cuál es mi presupuesto?



¿Cuál es el precio de un taco?



$$y = f(\sum w_i x_i + b)$$

- Zona
- Tipo de restaurante
- Ingredientes

¿Dónde están los mejores tacos?

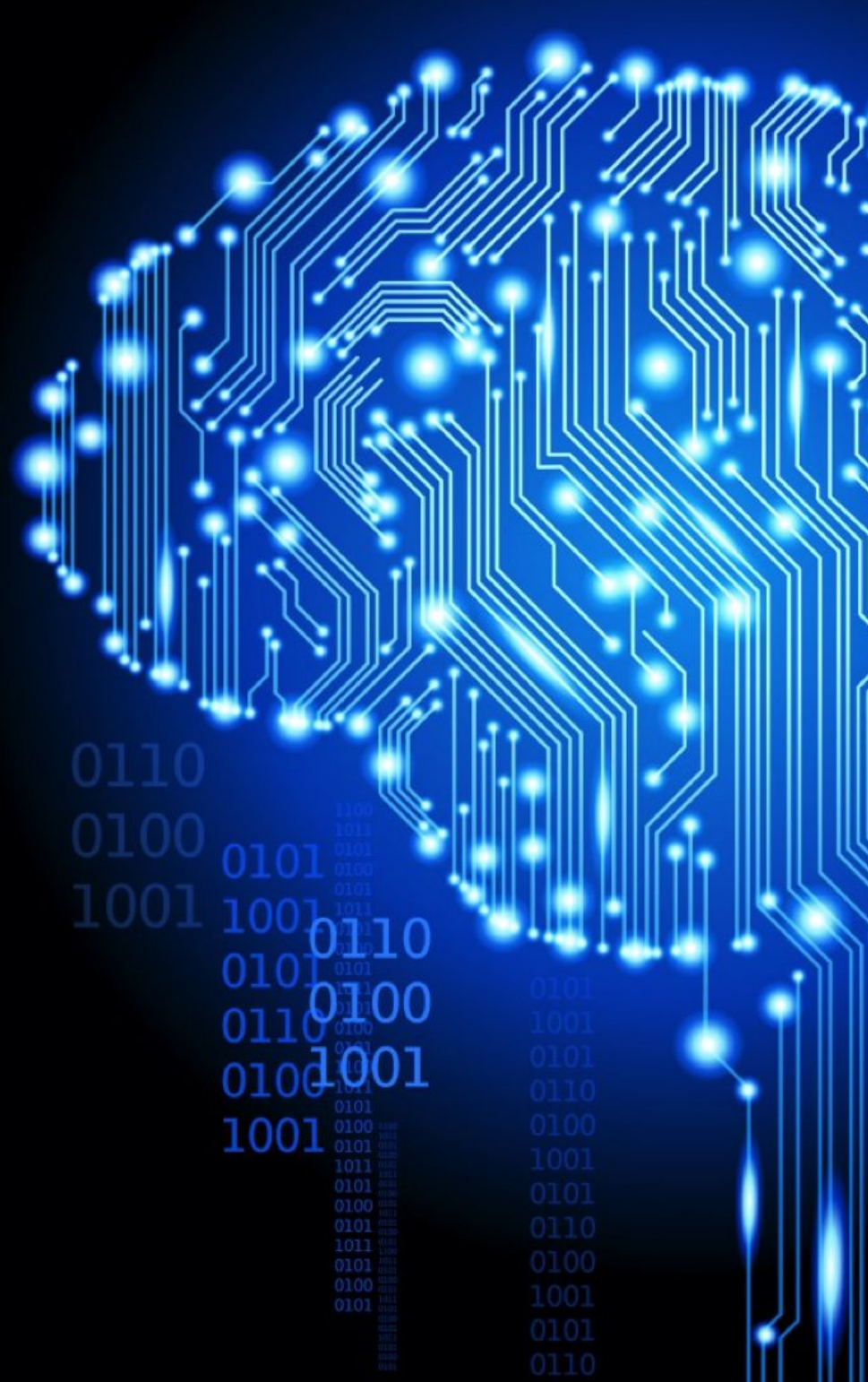
Supongamos que no tengo etiquetas sino solo input data. Es posible agrupar y buscar patrones.

¿Cuántos tacos debería comer?

- Tomar acciones para maximizar la recompensa en una situación específica.
- Aprendizaje basado en experiencia.

Deep Learning

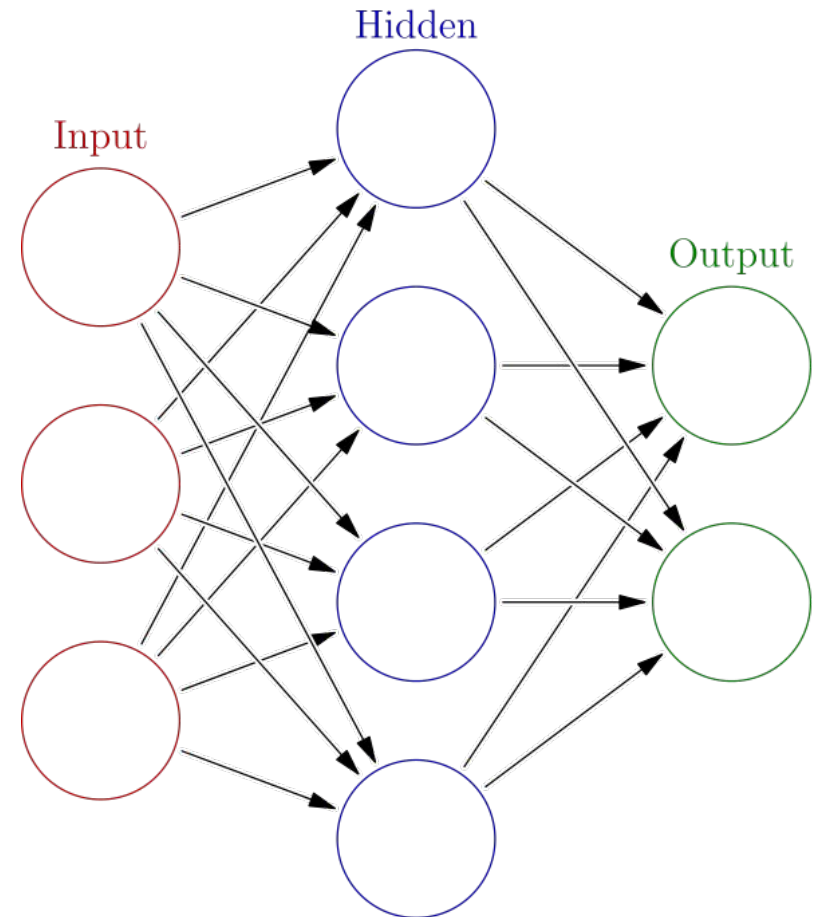
Un tipo de ML que utiliza redes neuronales (artificial neural networks) que pueden aprender asociaciones entre sus entradas y salidas



Artificial Neural Network

Sistema de software construído por nodos interconectados.

Puede ser una transformación lineal(peso + bias) o no-lineal(función de activación).






Diferencia entre AI, ML, DL y ANNs



Tipos principales de ML
Supervised, Unsupervised & Reinforcement learning



Nuestro curso cubrirá aprendizaje supervisado
Trabajaremos un proyecto de clasificación de imágenes

Conceptos básicos de Machine Learning

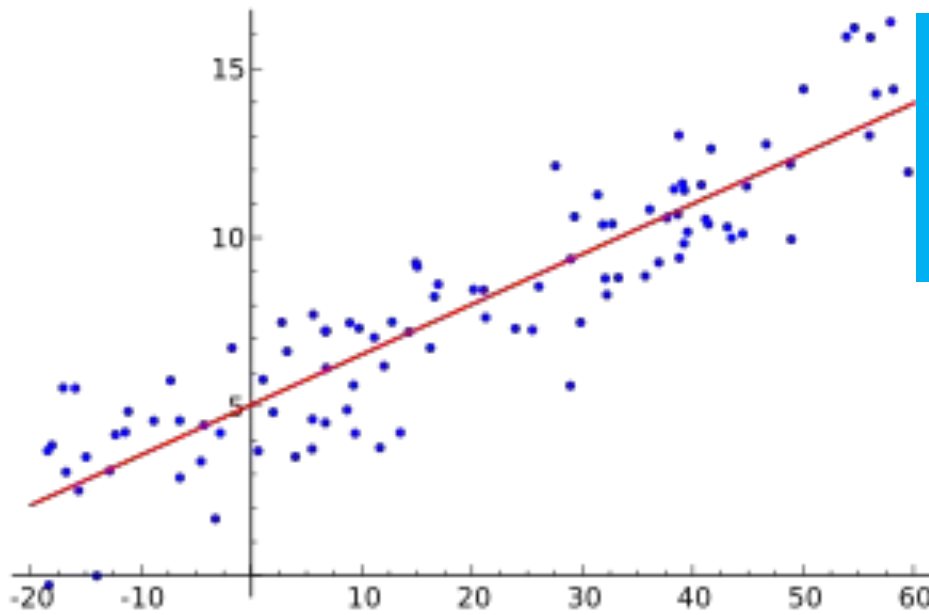
Terminología y regresión lineal

"AS **SOON** AS IT WORKS,
NO ONE CALLS IT **AI** ANYMORE."

~ John McCarthy



¿Cuál es el precio de un taco?

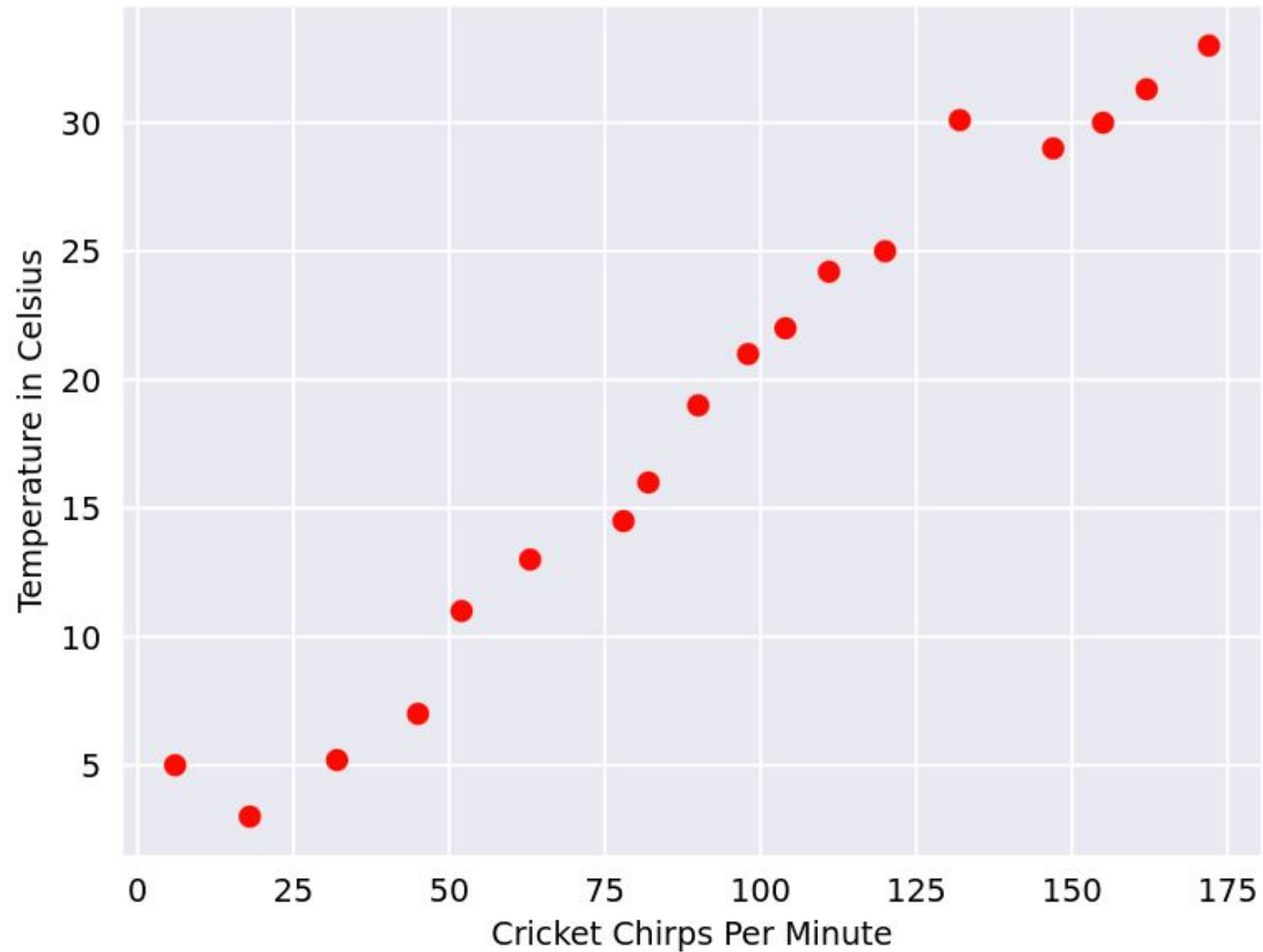


Este es un problema de regresión.

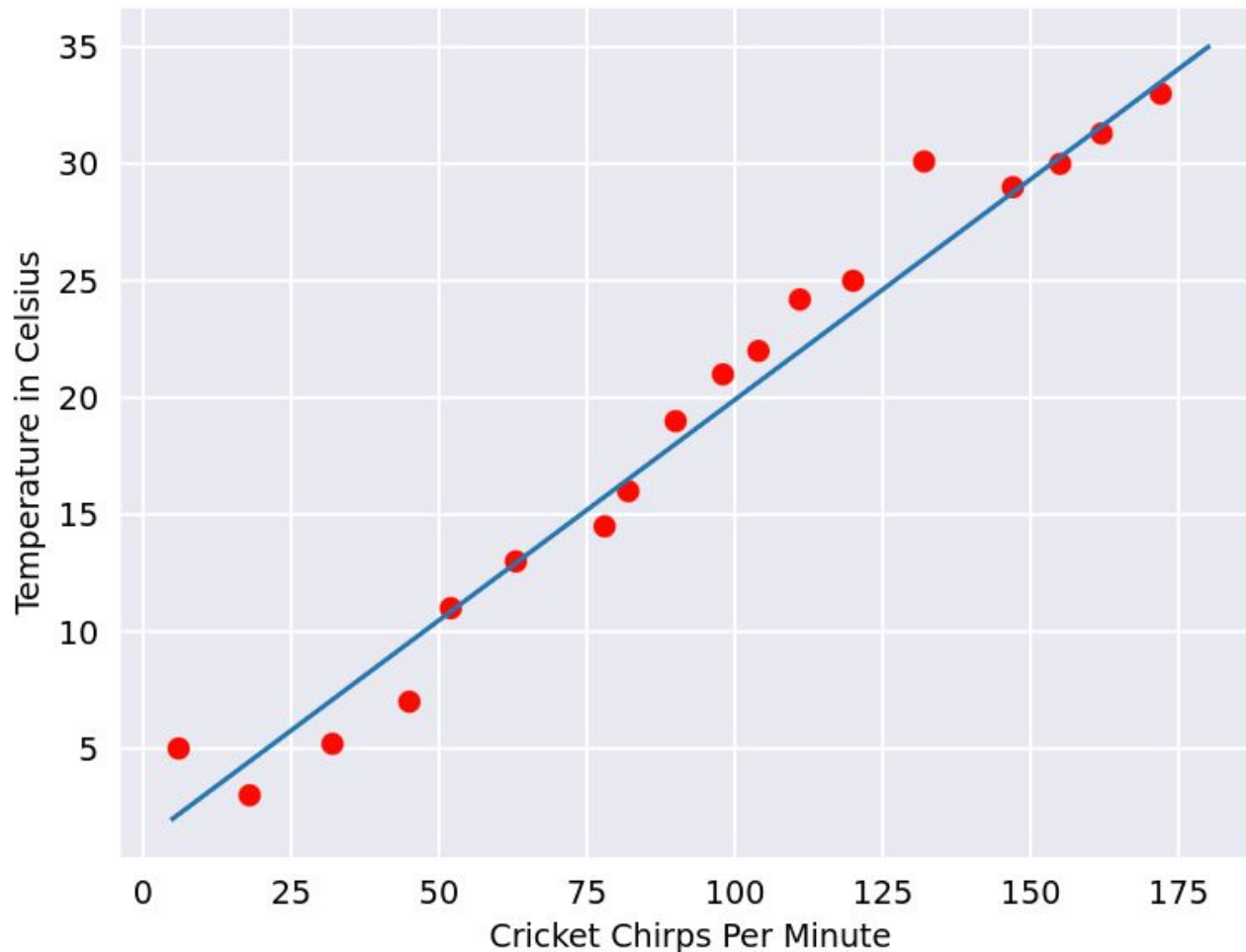
$$y = f(\sum w_i x_i + b)$$

- Zona
- Tipo de restaurante
- Ingredientes

Representando los datos



Modelando la relación



$$y = mx + b$$

$$y' = b + w_1 x_1$$

- $y = f(\sum w_i x_i + b)$

- **Label (y)**
Lo que estamos prediciendo

- **Feature (x)**
Una variable de entrada

Precios de ~~tacos~~ casas

| Be ns | et | Nei hood | S nce |
|------------------------|-----------|---------------------------|------------------------|
| 3 | 2000 | Normaltown | \$250,000 |
| 2 | 800 | Hipsterton | \$300,000 |
| 2 | 850 | Normaltown | \$150,000 |
| 1 | 550 | Normaltown | \$78,000 |
| 4 | 2000 | Skid Row | \$150,000 |



Modelo

Define una relación entre features y labels



Training

Darle un dataset al modelo y permitirle aprender de datos con label



Inference

Usar el modelo para realizar predicciones

Construyendo un modelo

Usando las variables (es necesario normalizar una, para tener representación numérica) construimos una ecuación para predecir

$$\text{precio} = \text{habitaciones} * w_0 + \text{tamaño} * w_1 + \text{ubicación} * w_2$$

Predicciones

| Bedrooms | Sq. feet | Neighborhood | Sale price | My Guess |
|----------|----------|--------------|------------|-----------|
| 3 | 2000 | Normaltown | \$250,000 | \$178,000 |
| 2 | 800 | Hipsterton | \$300,000 | \$371,000 |
| 2 | 850 | Normaltown | \$150,000 | \$148,000 |
| 1 | 550 | Normaltown | \$78,000 | \$101,000 |
| 4 | 2000 | Skid Row | \$150,000 | \$121,000 |

¿Y ahora?

Repetimos para todos los posibles valores de los pesos

¿Cuál es nuestro objetivo?

¿Cómo lo logramos?



Cómo modelar la relación de variables
features y labels



Proceso de aprendizaje
Varias iteraciones, para reducir loss

Conceptos básicos de Machine Learning

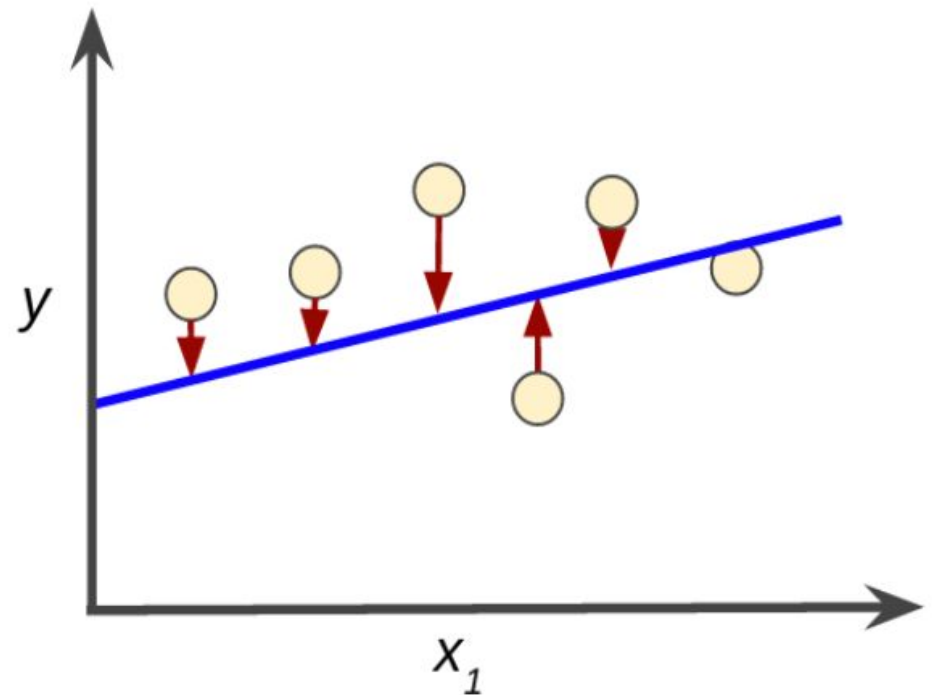
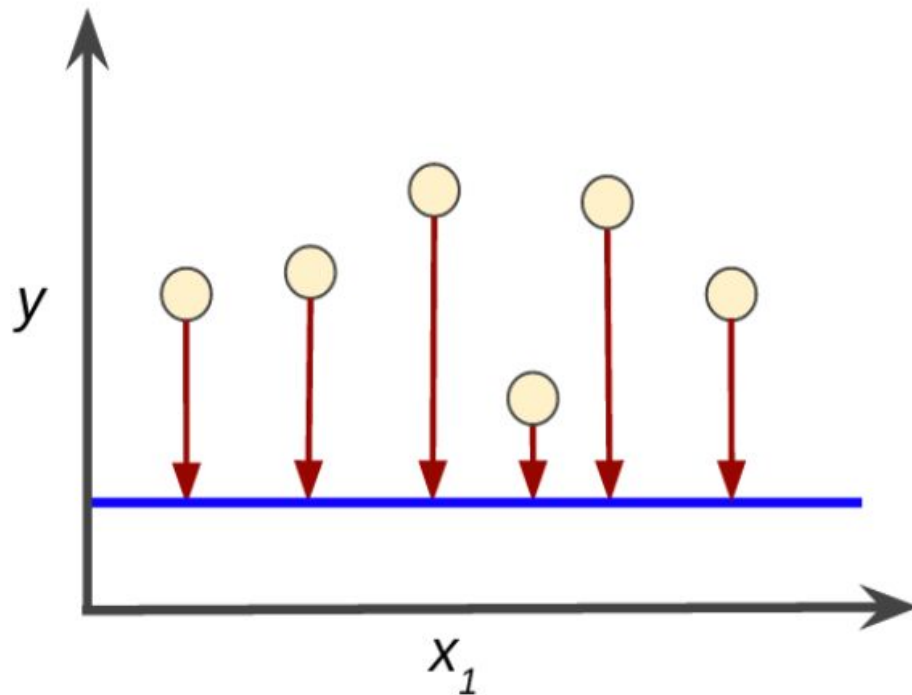
Training & loss



"WELL, NOW THAT WE HAVE SEEN
EACH OTHER", SAID THE UNICORN,
"IF YOU'LL BELIEVE IN ME,
I'LL BELIEVE IN YOU".

~ Lewis Carroll

Predicciones y valores reales



Predicciones

| Bedrooms | Sq. feet | Neighborhood | Sale price | My Guess |
|----------|----------|--------------|------------|-----------|
| 3 | 2000 | Normaltown | \$250,000 | \$178,000 |
| 2 | 800 | Hipsterton | \$300,000 | \$371,000 |
| 2 | 850 | Normaltown | \$150,000 | \$148,000 |
| 1 | 550 | Normaltown | \$78,000 | \$101,000 |
| 4 | 2000 | Skid Row | \$150,000 | \$121,000 |

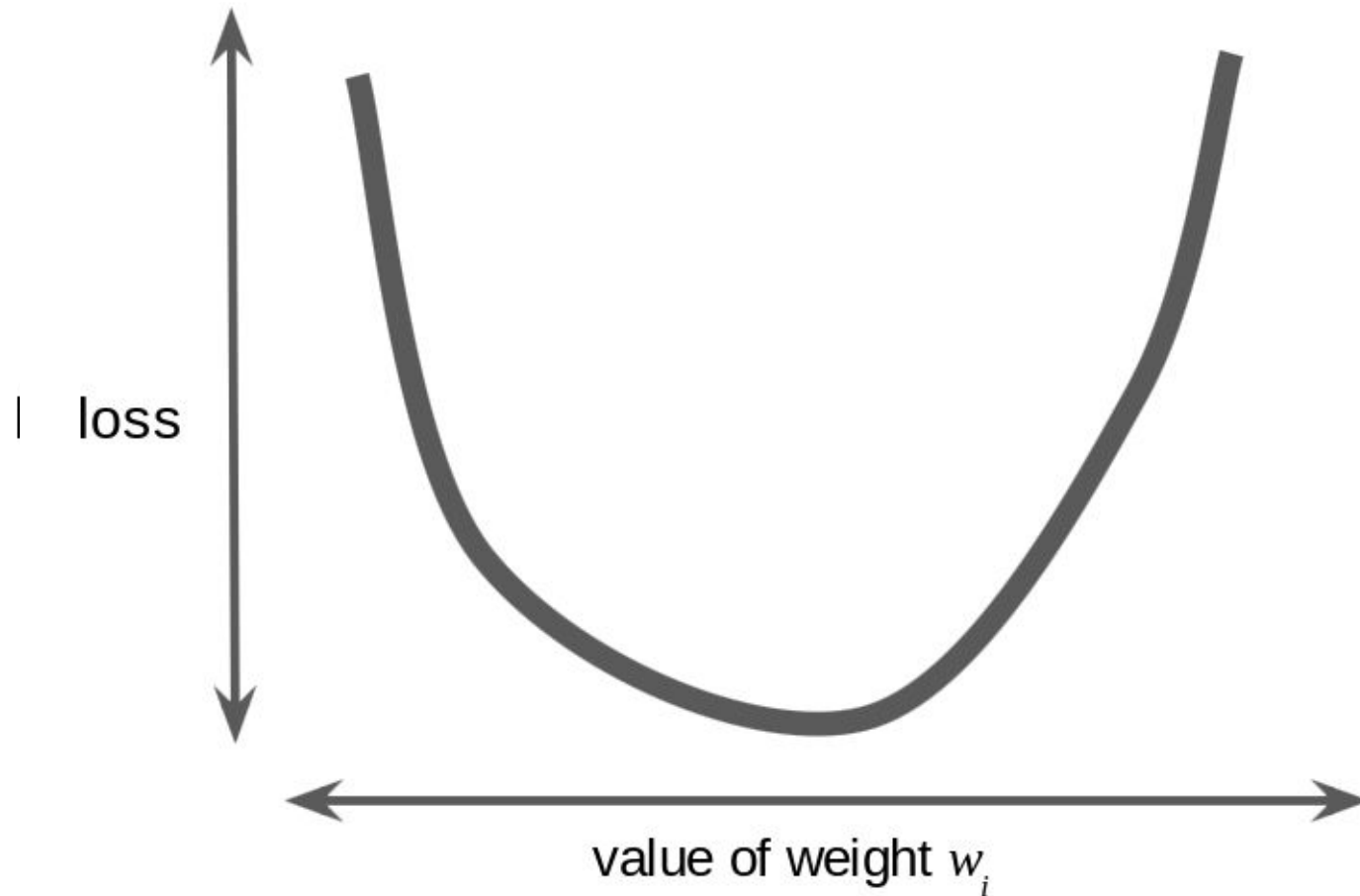
Cálculo de error

$$\text{Cost} = \frac{\sum_{i=1}^{500} (\text{MyGuess}(i) - \text{RealAnswer}(i))^2}{500 \cdot 2}$$

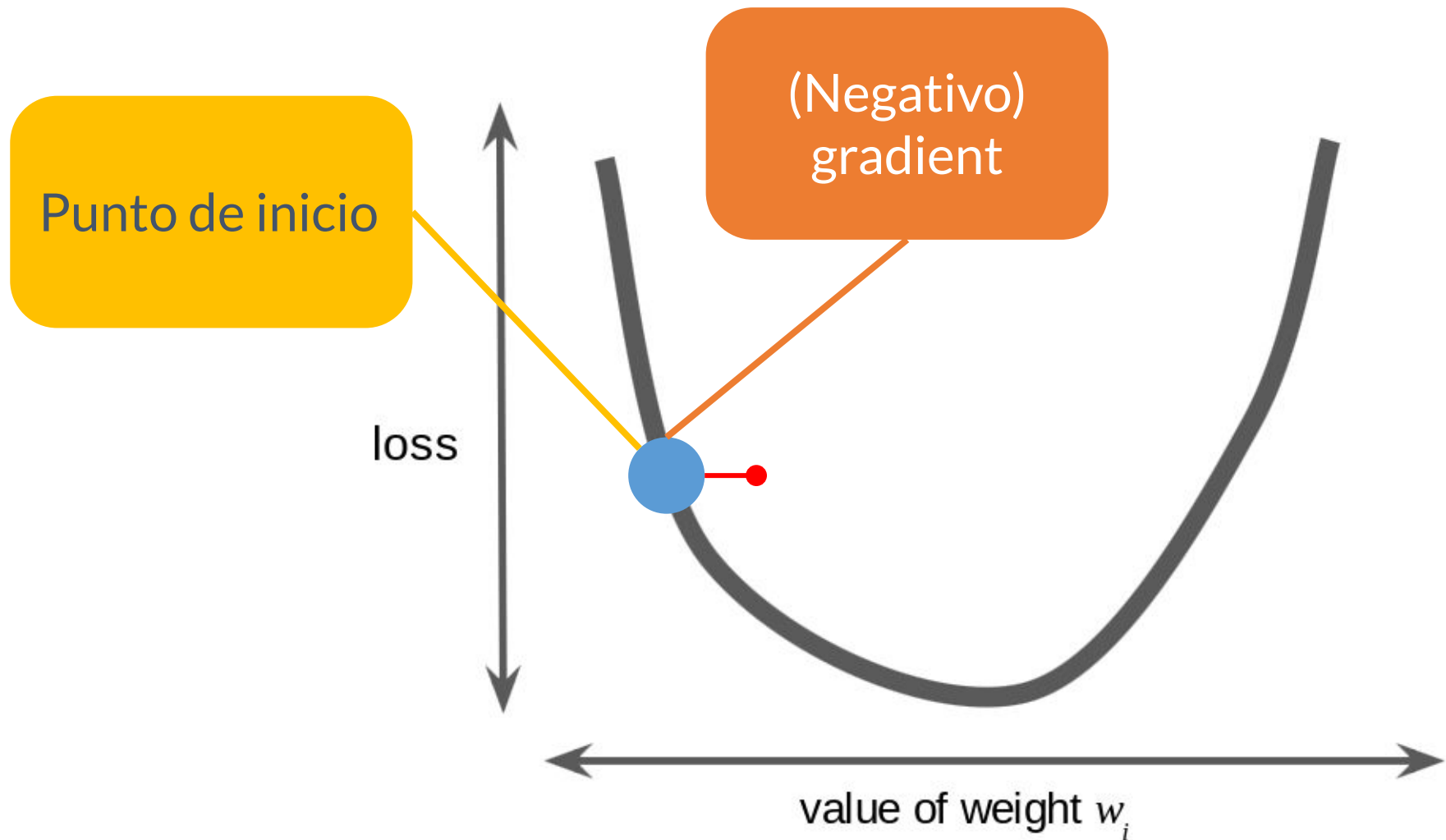
MSE (Mean Squared Error)

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m \left(h_{\theta}(x^{(i)}) - y^{(i)} \right)^2$$

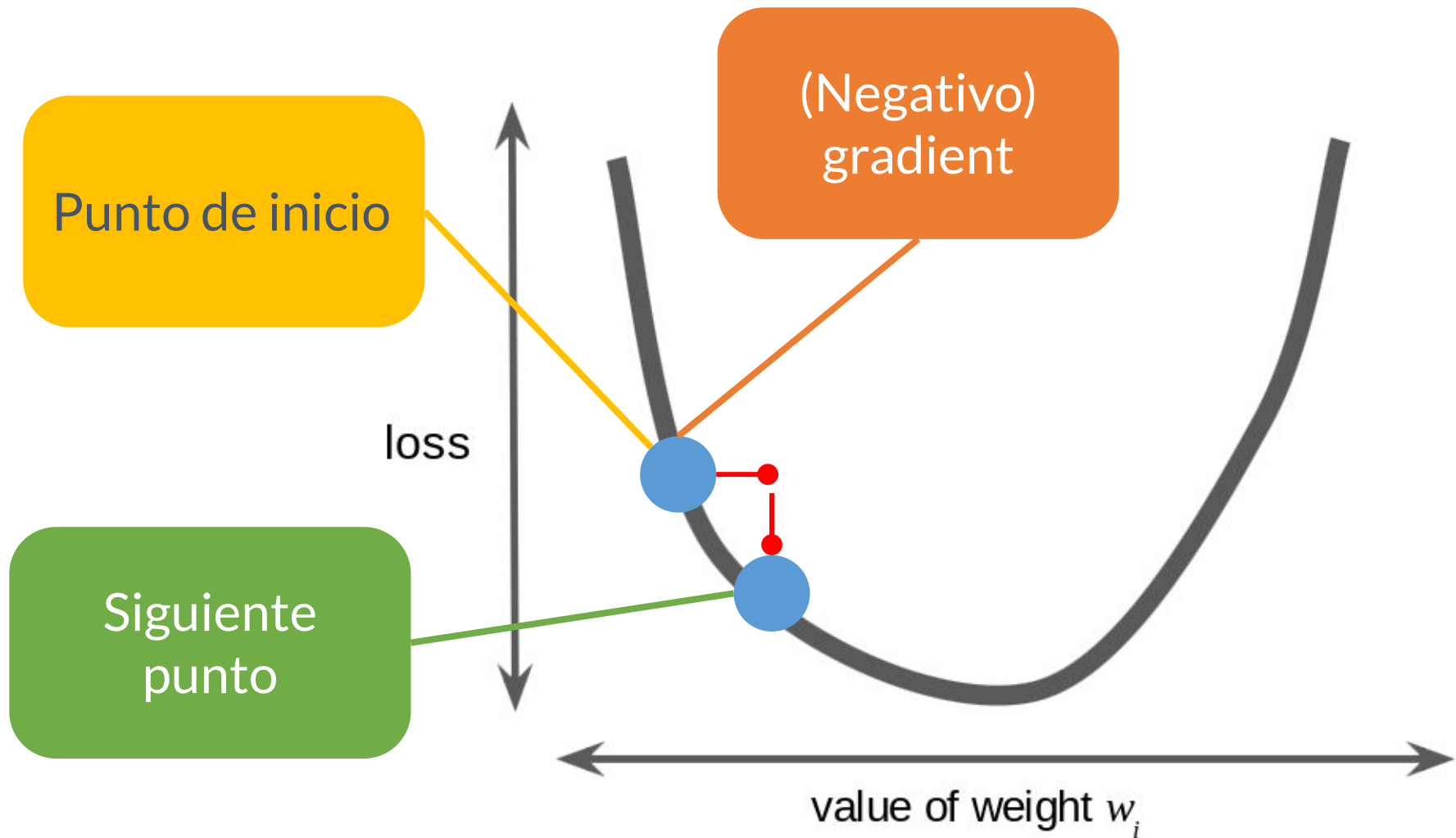
Gradient descent



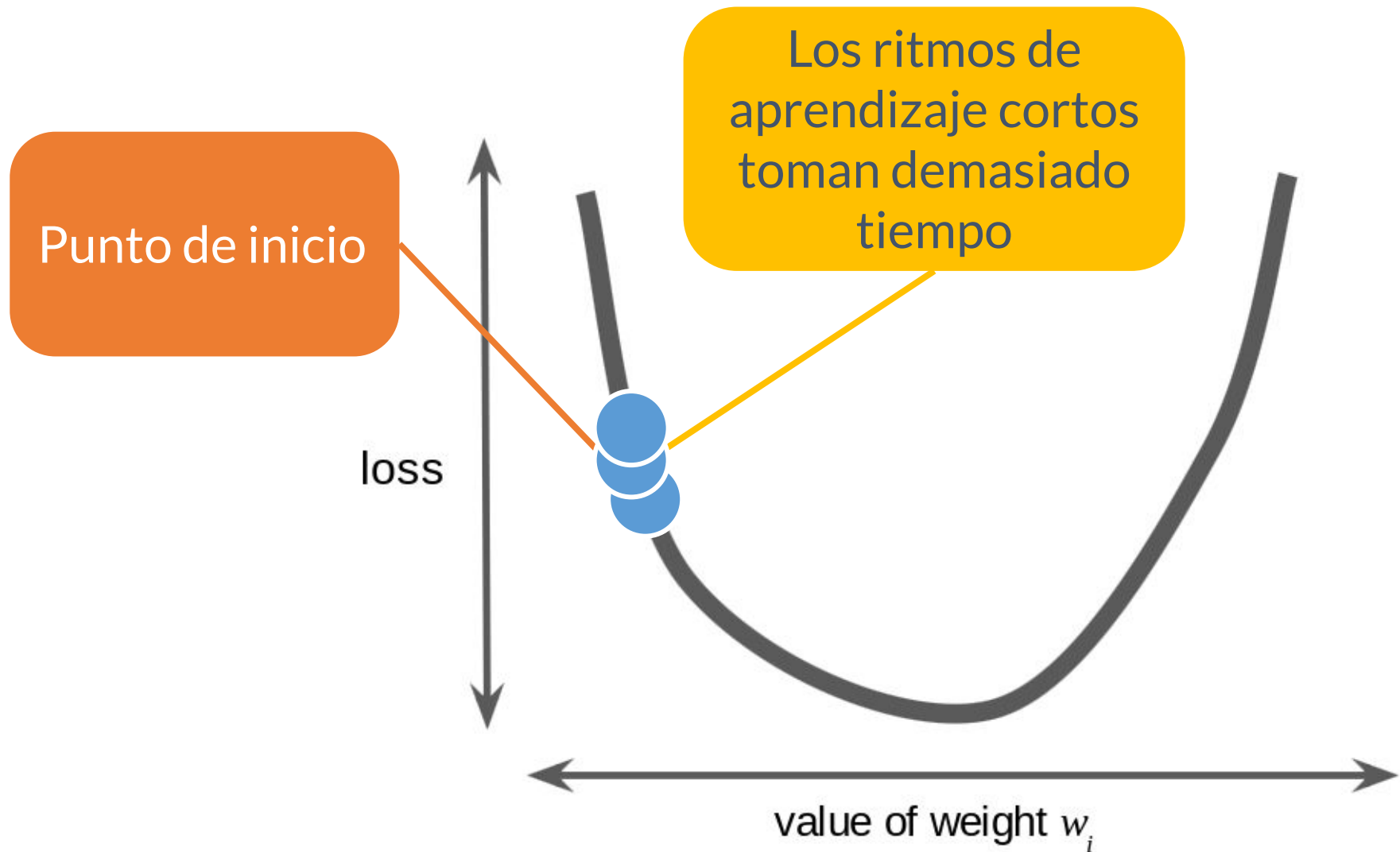
Gradient descent



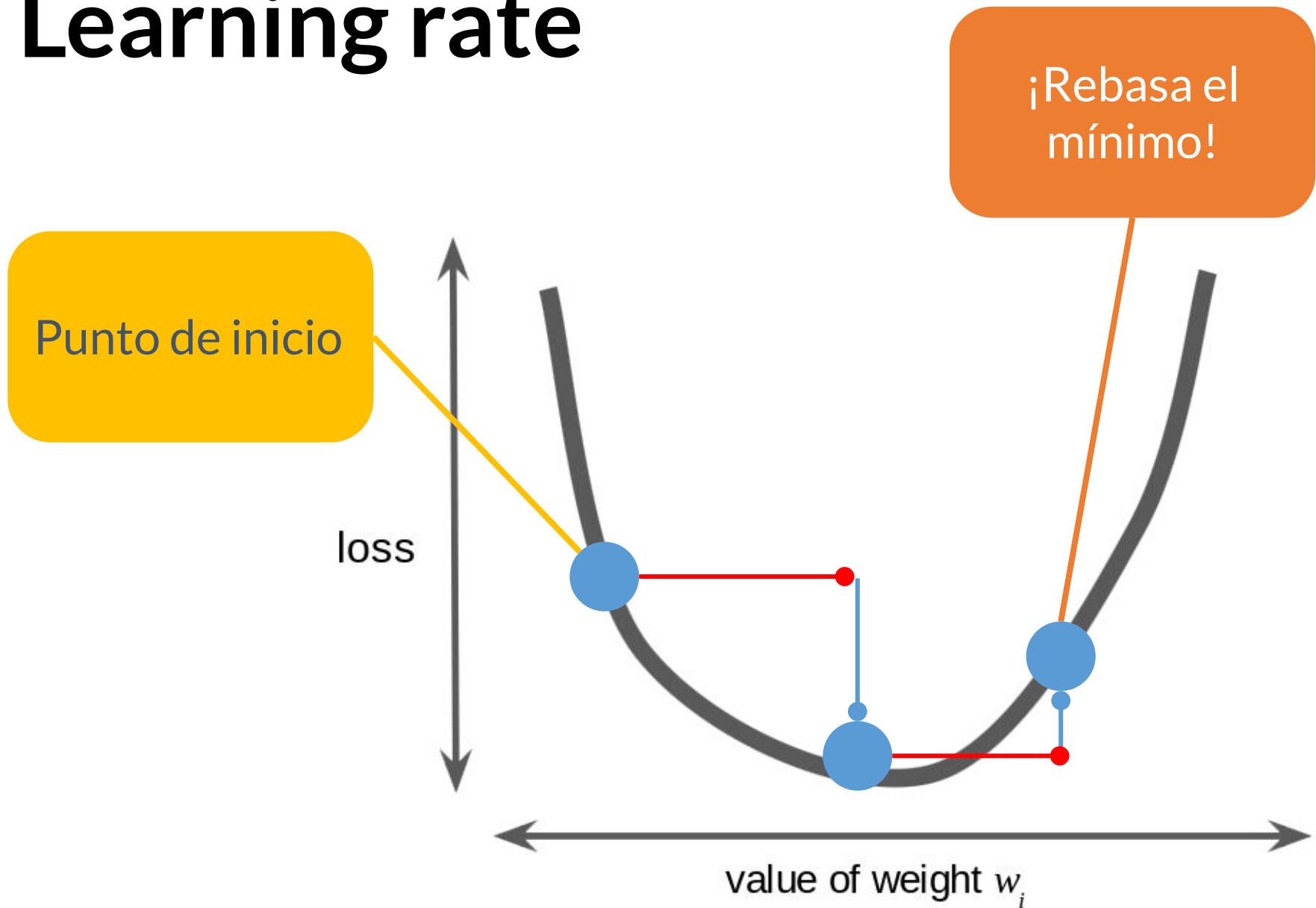
Gradient descent



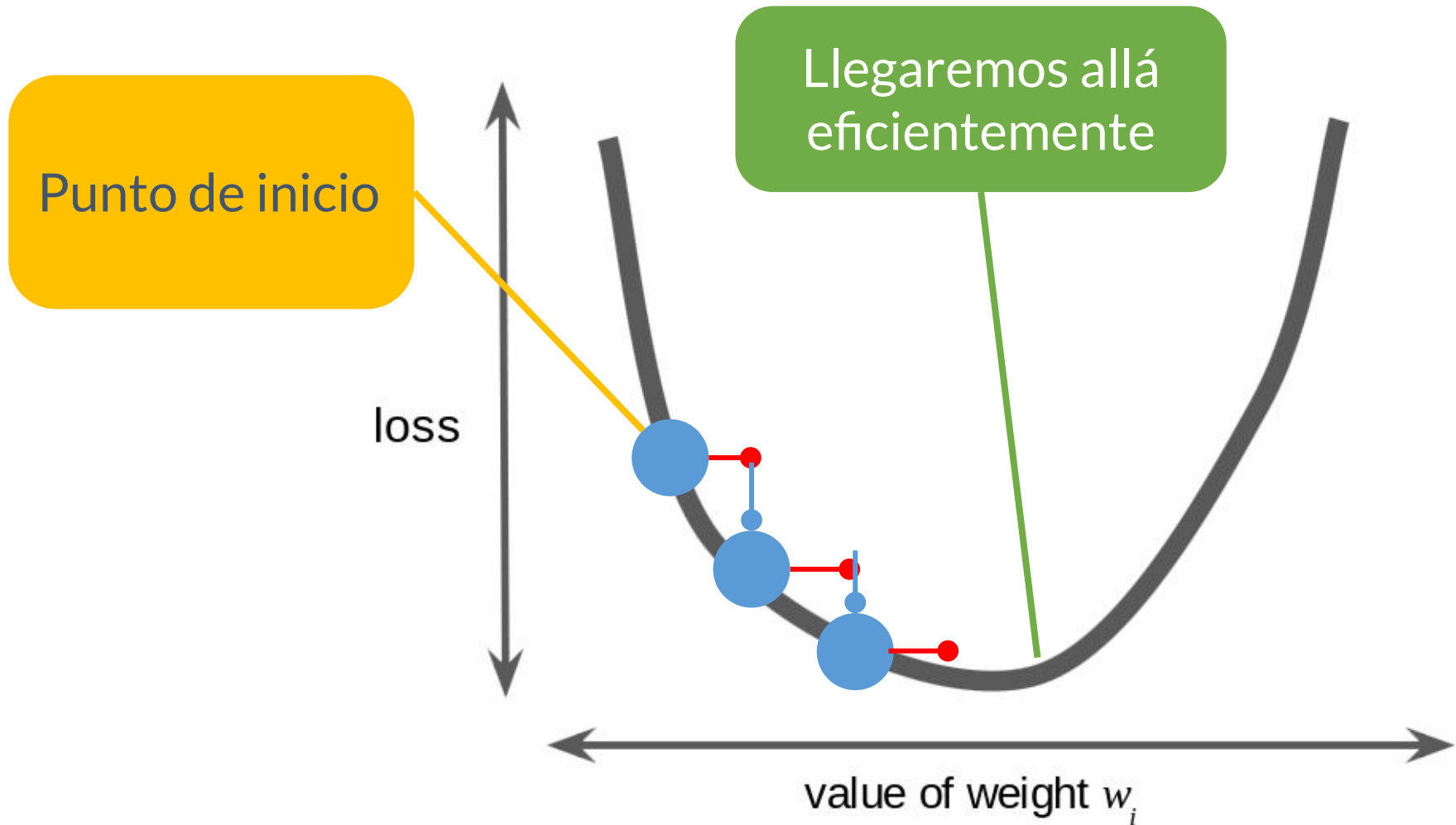
Learning rate



Learning rate



Learning rate



SGD (Stochastic Gradient Descent)

- ¿Deberíamos calcular el gradiente de todo el dataset?
- Tal vez sea más eficiente elegir ejemplos aleatorios.
- Un solo ejemplo es muy poco, trabajamos con un batch.

Proceso de aprendizaje

- Inicializamos los valores de los pesos y utilizamos el modelo para predecir.
- Calculamos loss y evaluamos el desempeño del modelo.
- Usando el gradiente, recalculamos pesos e iteramos.



La fase de entrenamiento es un proceso iterativo



Al calcular el error(loss), el gradiente nos ayuda a buscar el mínimo



Es necesario calibrar el valor de learning rate

Trabajando con PyTorch

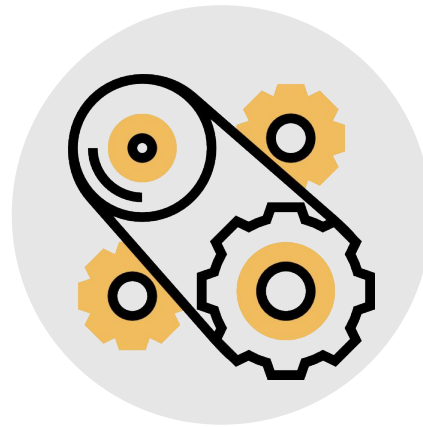
Introducción a PyTorch



PyTorch



Frameworks



Ejecución
inmediata o
postergada

Proceso de aprendizaje

- Forward pass
- Backpropagation
- Optimización

Tensores

Scalar

Vector

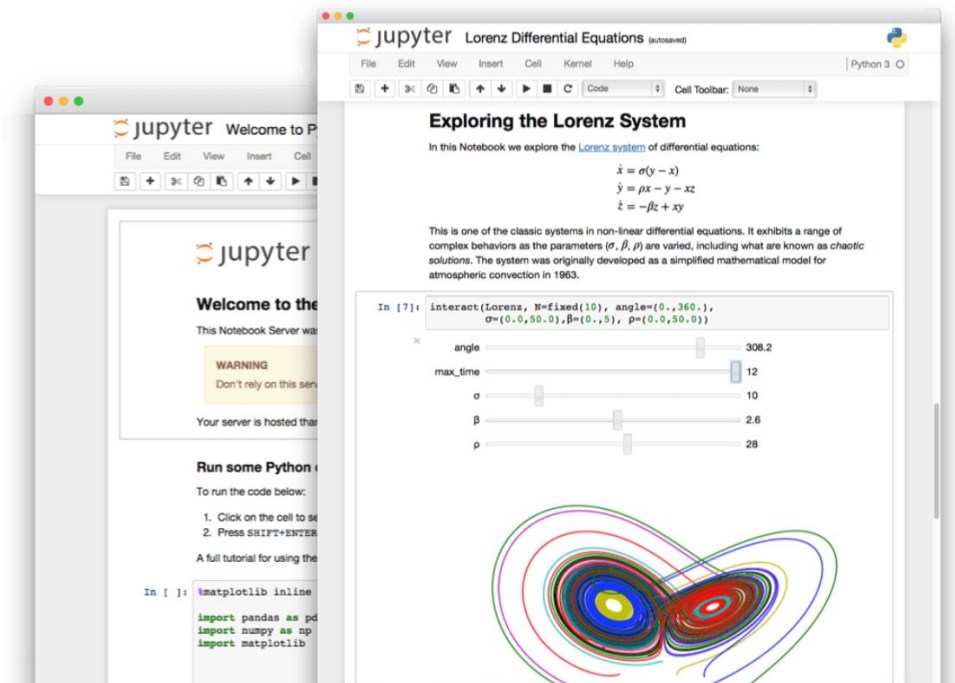
Matrix

Tensor

1

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$
$$\begin{bmatrix} \begin{bmatrix} 1 & 2 \end{bmatrix} & \begin{bmatrix} 1 & 2 \end{bmatrix} \\ \begin{bmatrix} 1 & 7 \end{bmatrix} & \begin{bmatrix} 5 & 4 \end{bmatrix} \end{bmatrix}$$

Jupyter Notebooks




The Jupyter Notebook


The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Try it in your browser


Install the Notebook






Google Colaboratory



Welcome To Colaboratory 

File Edit View Insert Runtime Tools Help

 SHARE [Sign in](#)

 CODE  TEXT  CELL  CELL  COPY TO DRIVE




CONNECT  EDITING 


Table of contents Code snippets Files 



Introducing Colaboratory

Getting Started

More Resources


Machine Learning Examples: Seedbank

 SECTION



 **Welcome to Colaboratory!** 

Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud.

With Colaboratory you can write and execute code, save and share your analyses, and access powerful computing resources, all for free from your browser.

 **Introducing Colaboratory**

This 3-minute video gives an overview of the key features of Colaboratory:

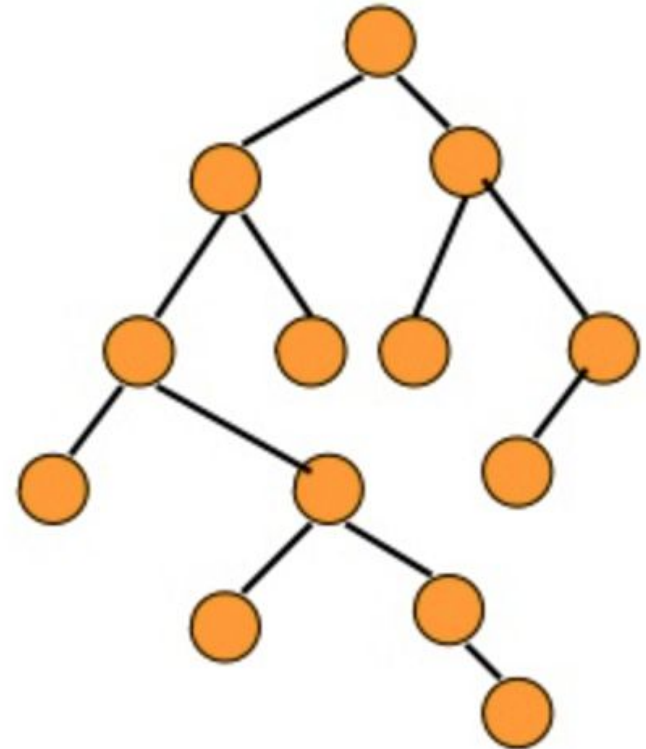


Implementaciones de algoritmos de
Machine Learning en PyTorch

Regresión logística

¿Debería comer tacos?

- ¿Tengo hambre?
- ¿Estoy solo o acompañado?
- ¿Cuál es mi presupuesto?



Clasificación

- Para resolver algunos problemas, es necesario predecir probabilidad.
- Spam v No spam
- Aprobado v Reprobado

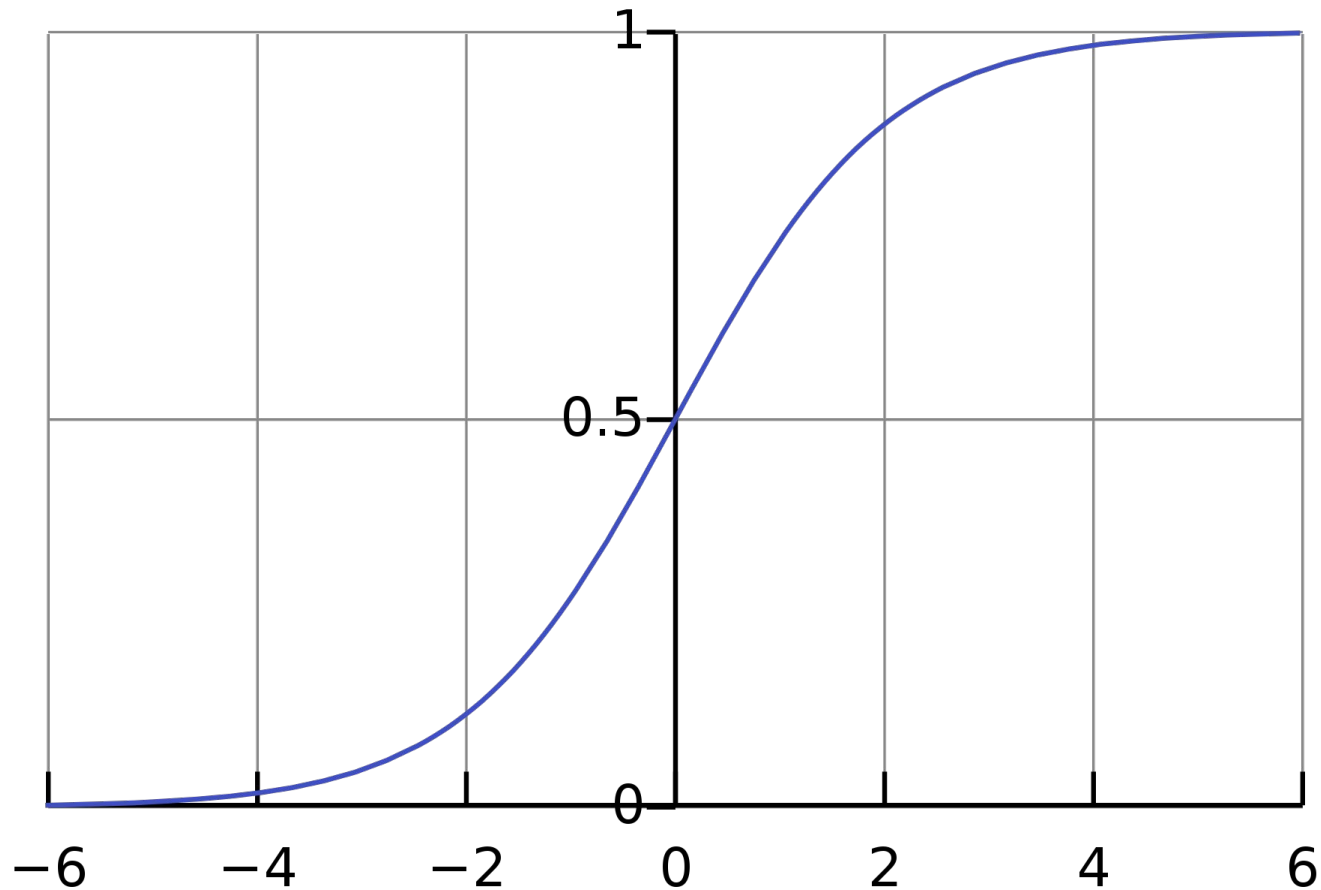
Regresión logística

Son un mecanismo eficiente para calcular probabilidades. El resultado puede utilizarse tal cual o convertirlo a una categoría binaria (para clasificar).

Garantizando resultado $[0,1]$ entre cero y uno

Para una clasificación binaria, nos apoyamos en una función matemática llamada *Sigmoide*. Si en caso la clasificación tuviera más parámetros, haríamos uso de la función *Softmax*.

Función Sigmoid

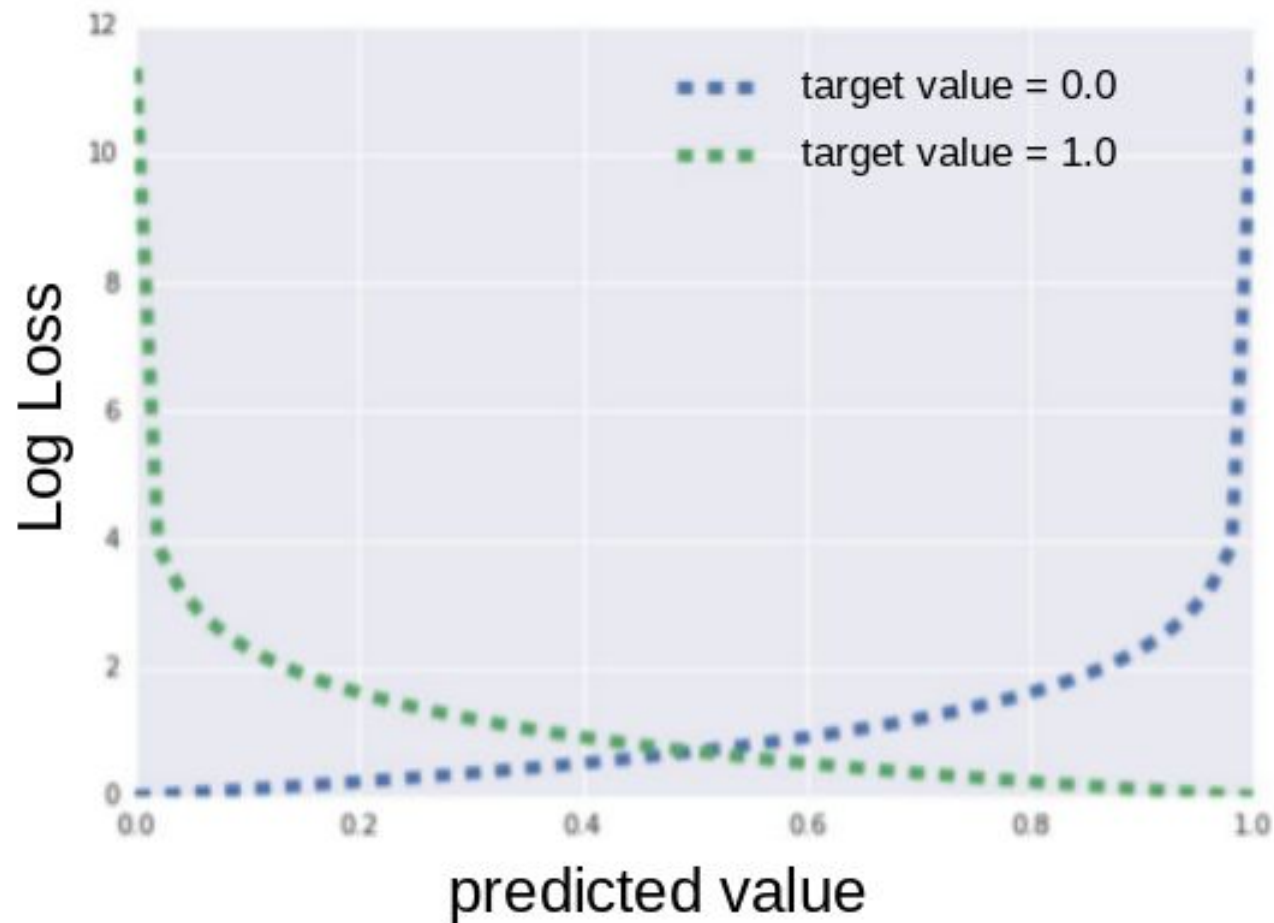


$$y = \frac{1}{1+e^{-z}}$$

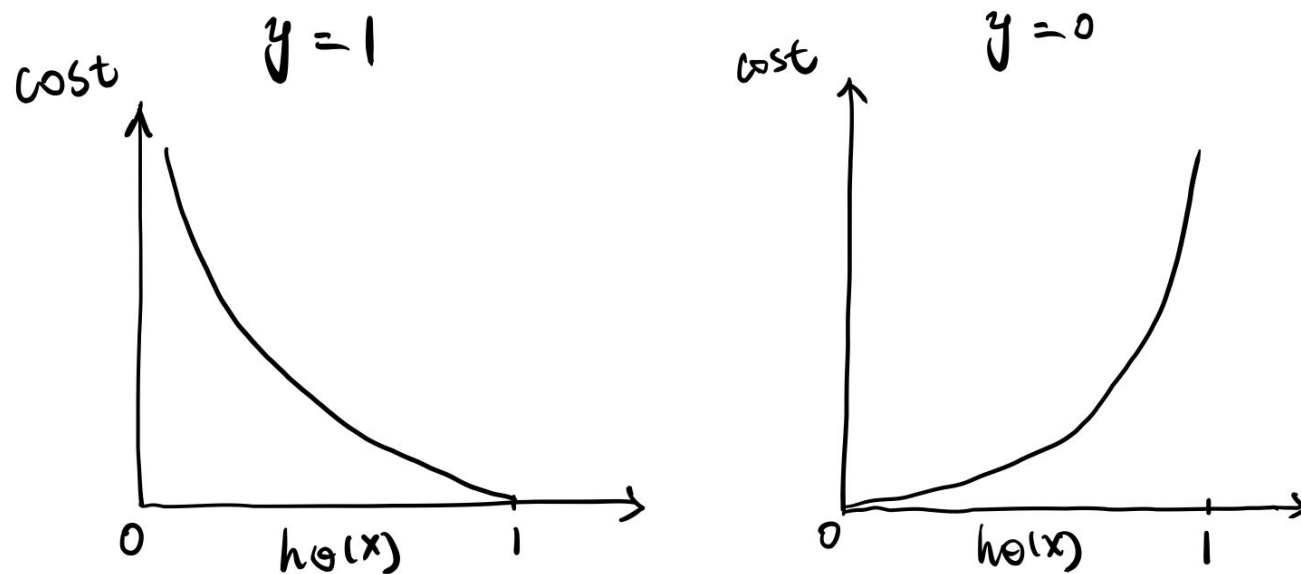
Loss

La aproximación intuitiva es *castigar* cuando el valor es 0 y la predicción resulta en 1 (o viceversa)


Loss




Binary Cross Entropy/Log Loss



$$Cost(h_{\theta}(x), y) = \begin{cases} -\log(h_{\theta}(x)) & \text{if } y = 1 \\ -\log(1 - h_{\theta}(x)) & \text{if } y = 0 \end{cases}$$



Para problemas de probabilidad, utilizamos una regresión logística.

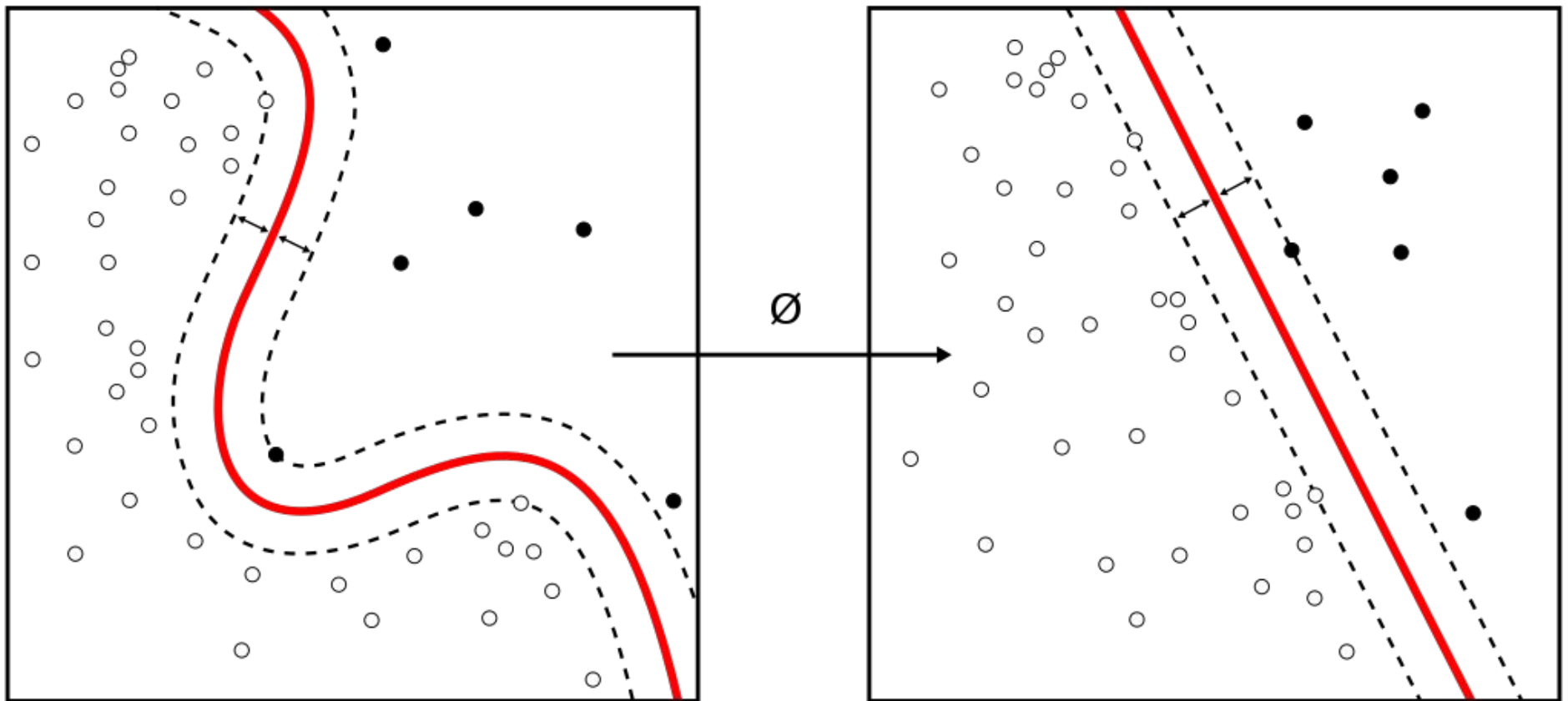


Para calcular el error(loss), nos basamos en la entropía pero el gradiente sigue siendo útil.

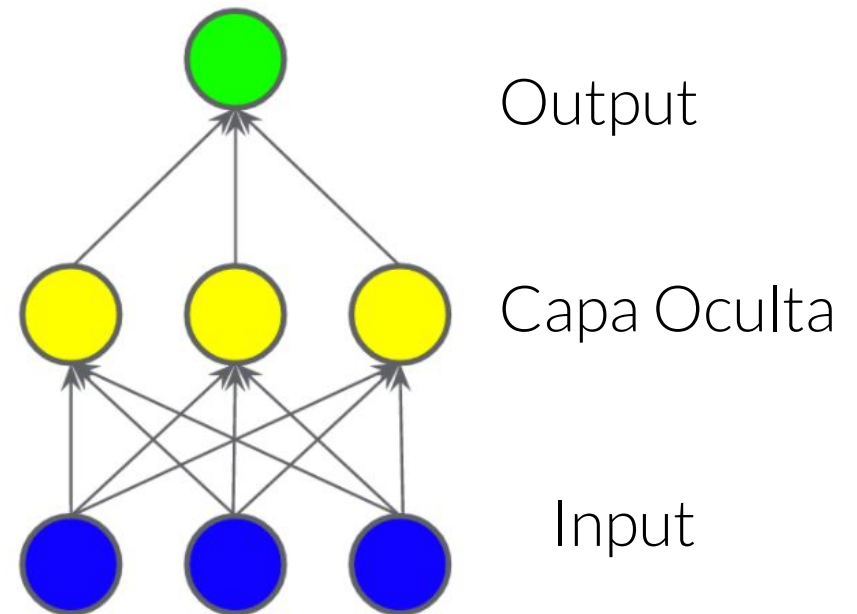
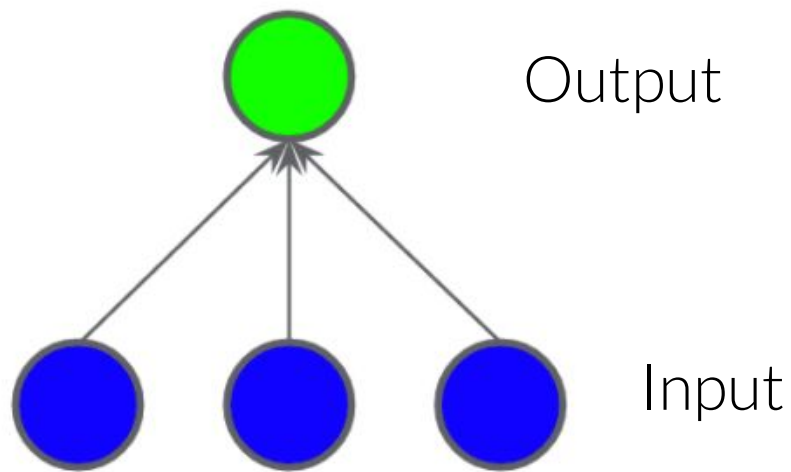
Redes Neuronales y reconocimiento de
imágenes

Neuronas y función de activación

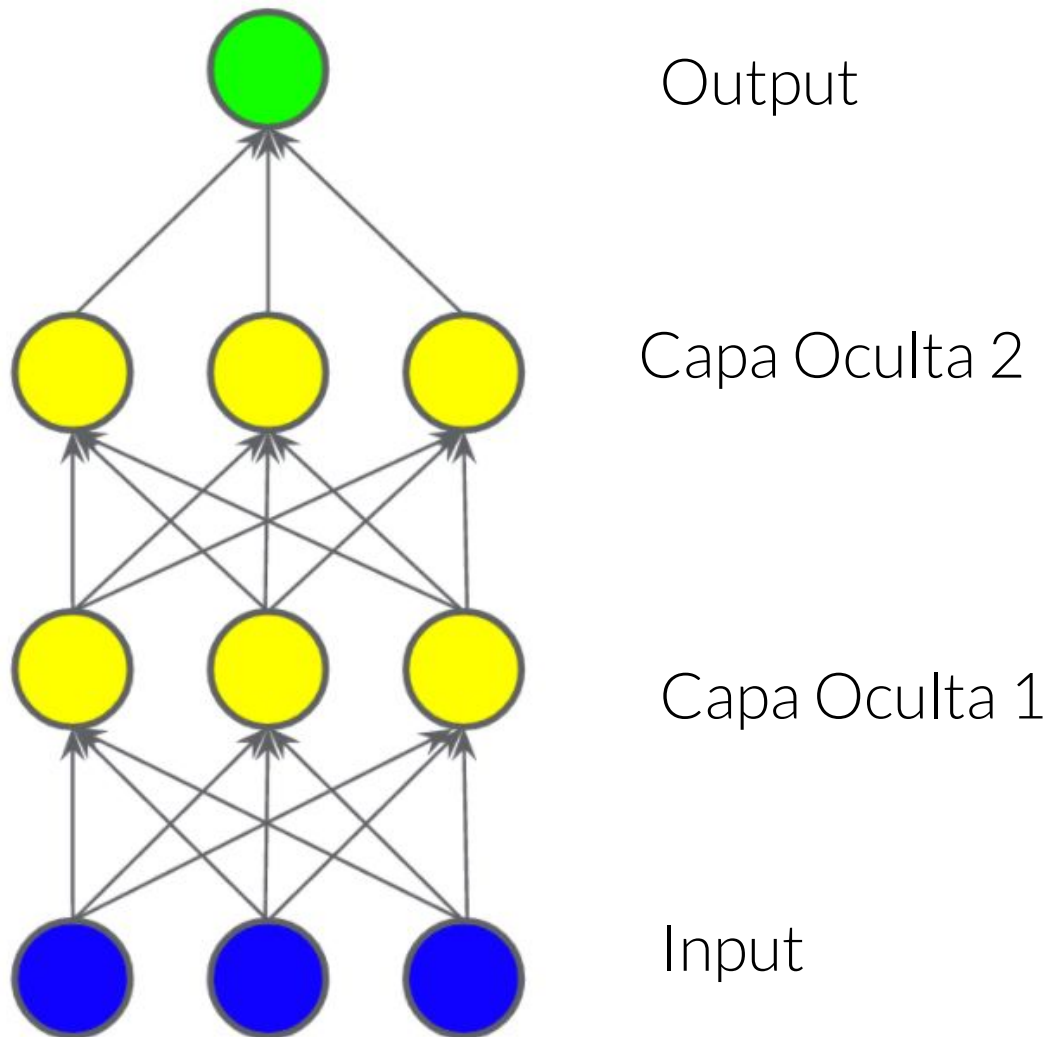
Artificial Neural Network



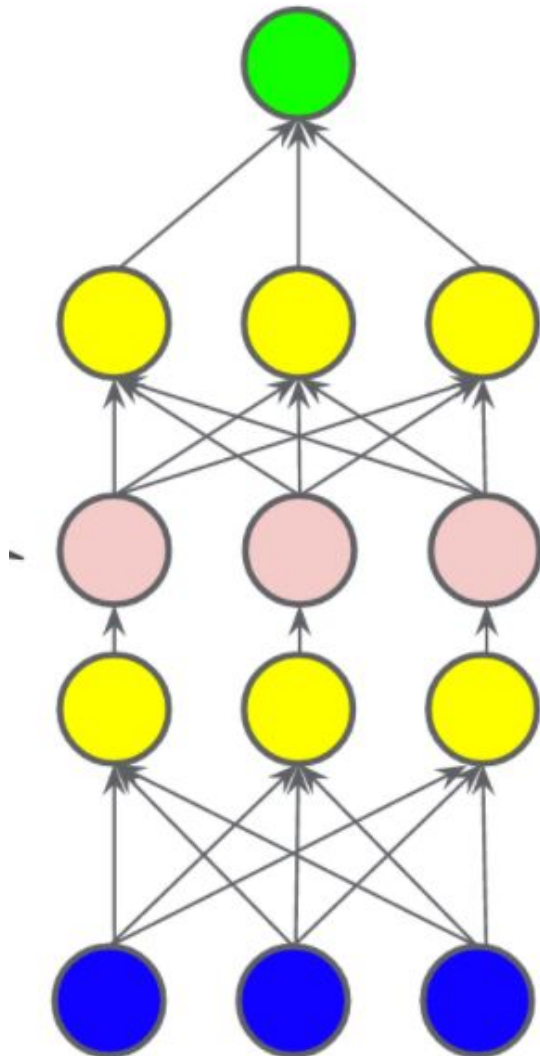
Resolviendo problemas no-lineales



Agregando capas



Función de activación



Output

Capa Oculta 2

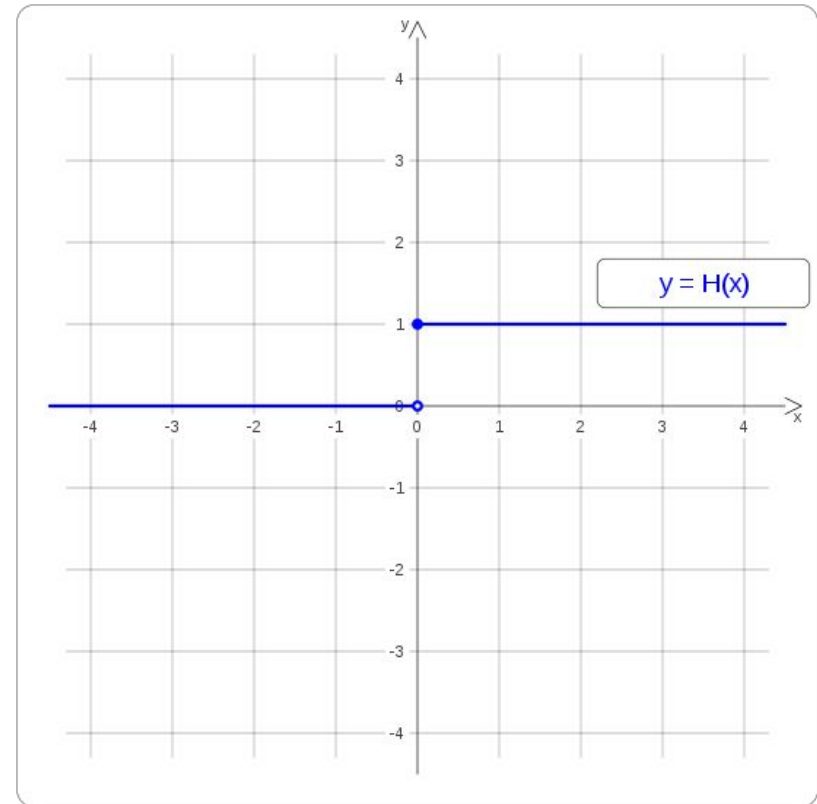
Transformación no lineal
(función de activación)

Capa Oculta 1

Input

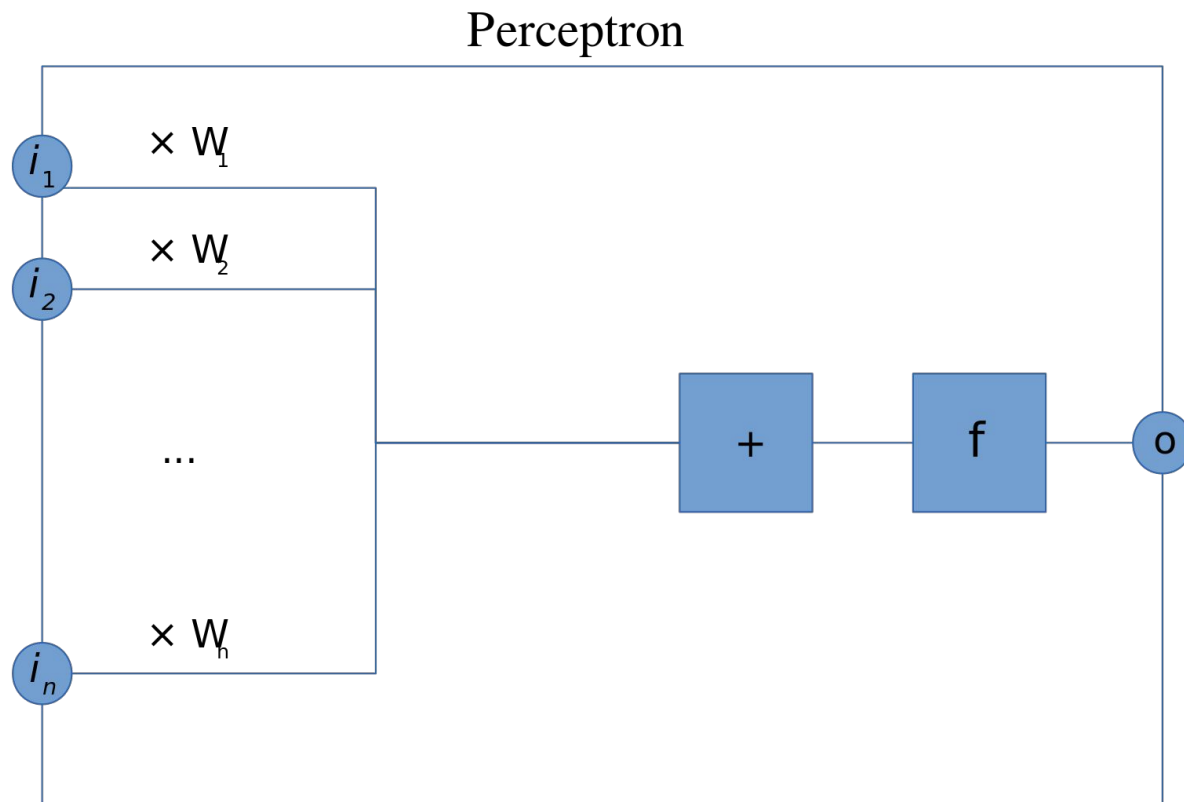
Perceptrón

Neurona básica, toma datos como input y se le aplica un función de activación (normalmente escalón de Heaviside)



$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$

Perceptrón



$$o = f\left(\sum_{k=1}^n i_k \cdot W_k\right)$$

- **Sigmoid**

- **Tanh**

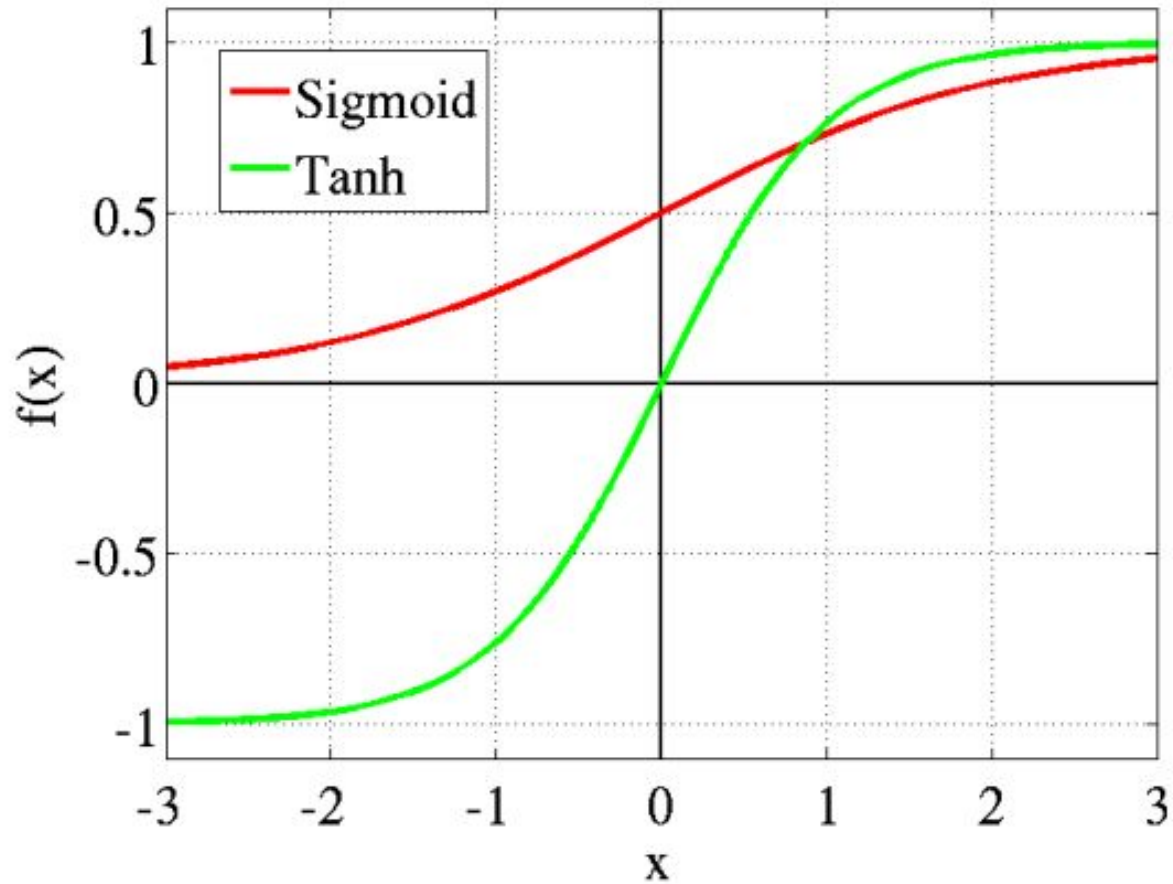
- **ReLUs**

Tanh

$$\tanh(x) = 2\sigma(2x) - 1$$

Sigmoid *escalado*, con output $[-1,1]$. Puede fallar cuando la activación se satura cerca de 0 o 1 con gradientes cercanos a cero.

Tanh



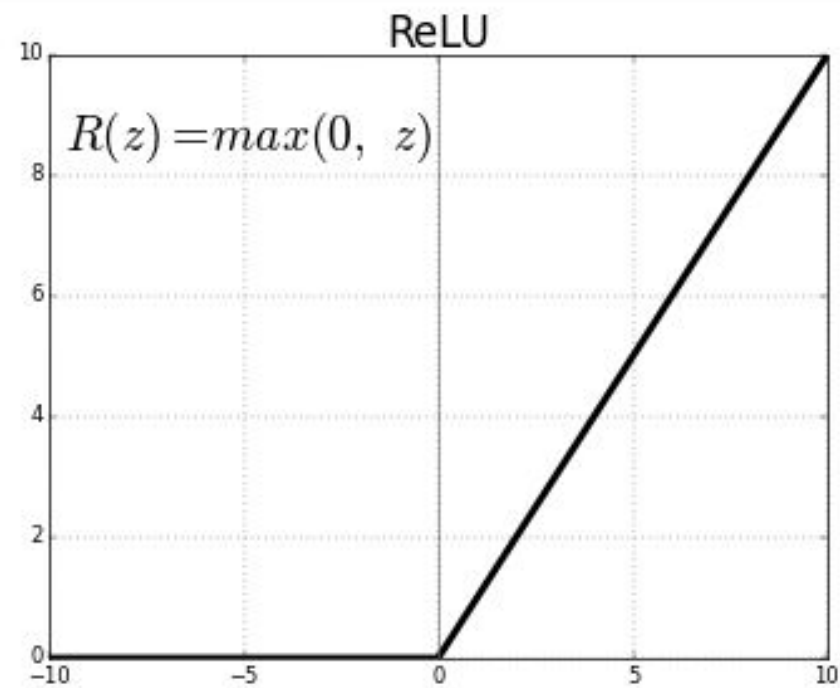
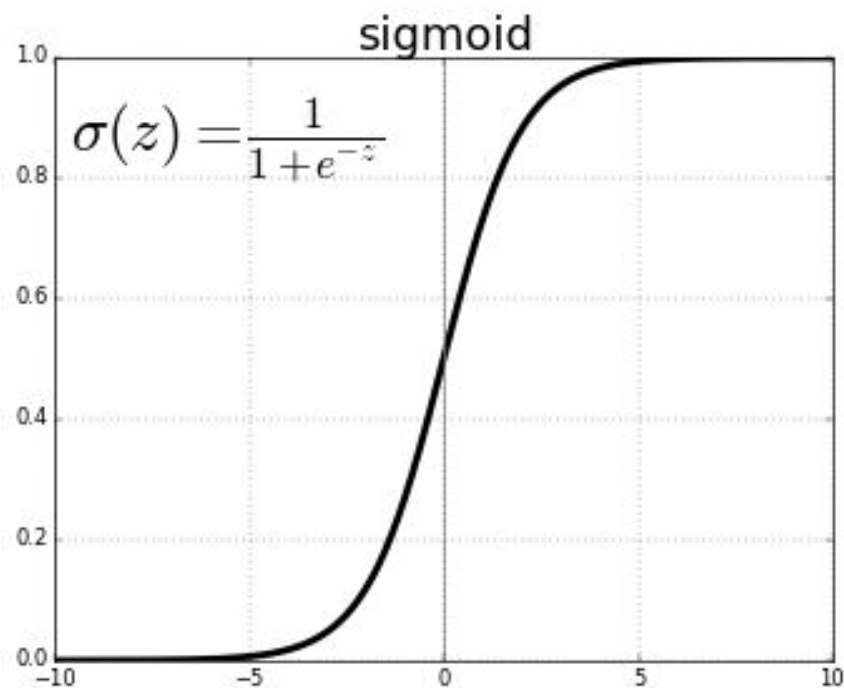
$$\tanh(x) = 2\sigma(2x) - 1$$

ReLU

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

Evita el problema de *vanishing gradient* pero solo puede utilizarse en las hidden layers de una NN. Existen variantes para evitar algunos de los problemas más comunes (como neuronas muertas).


ReLU



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



NN nos permiten solucionar problemas no-lineales.



Es necesaria una función de activación y existen varias opciones.

Estado actual de Machine Learning

"MACHINES WILL BE **CAPABLE**,
WITHIN TWENTY YEARS,
OF DOING **ANY** WORK
A **MAN** CAN DO."

~ Herbert Simon (1956)



Principales tipos de ML

- **Supervised learning**

Aprendizaje supervisado

- **Unsupervised learning**

Aprendizaje sin supervisión

- **Reinforcement learning**

Aprendizaje por refuerzo autónomo

Aviación



Agricultura



Dónde se está usando

- Salud
- Arte
- Campo

DensePose



DensePose

[Home](#)

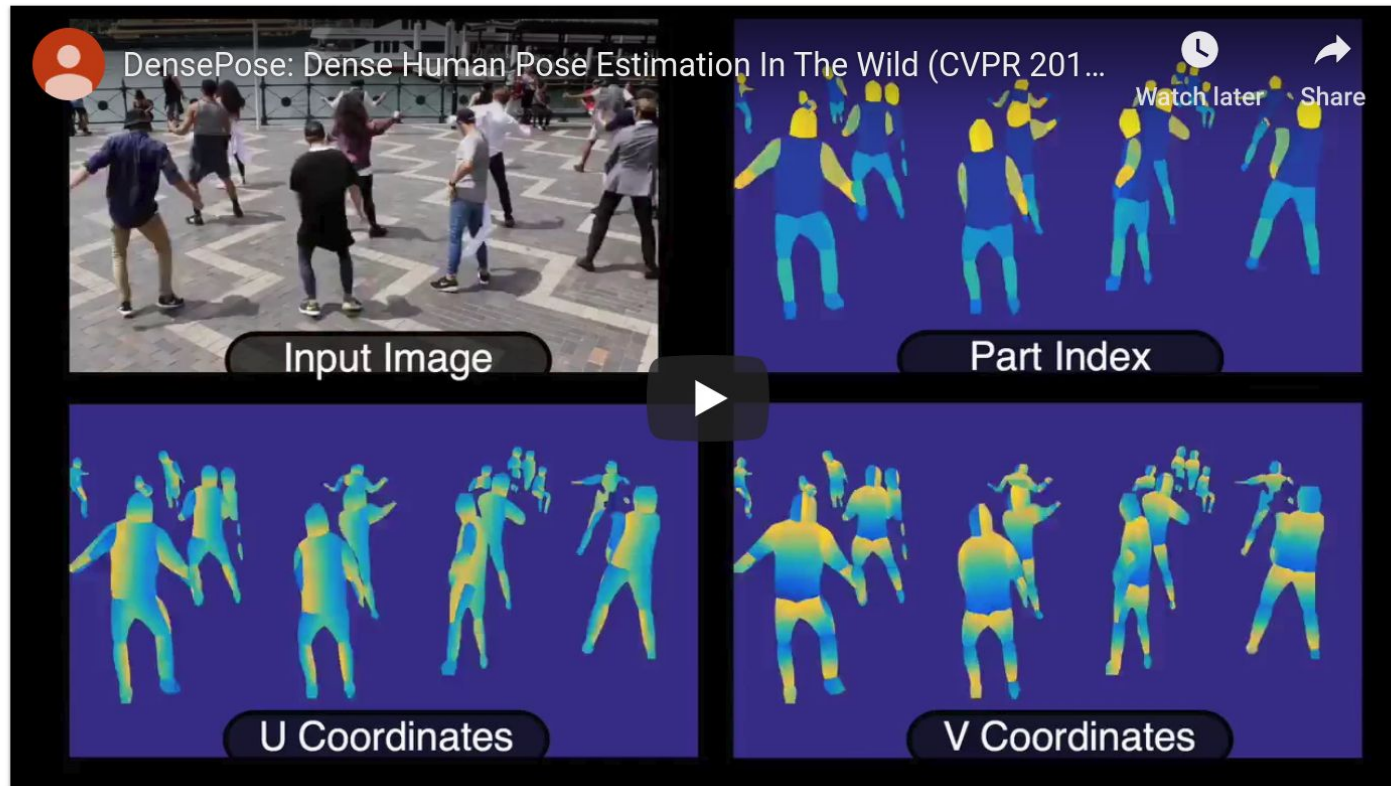
[Dataset](#)

[Method](#)

[Team](#)

• DensePose is a part of the PoseTrack Challenge at ECCV 2018.

See the challenge description page and the evaluation server.



End-to-End Negotiator

End-to-End Negotiator

This is a [PyTorch](#) implementation of research paper [Deal or No Deal? End-to-End Learning for Negotiation Dialogues](#) developed by [Facebook AI Research](#).

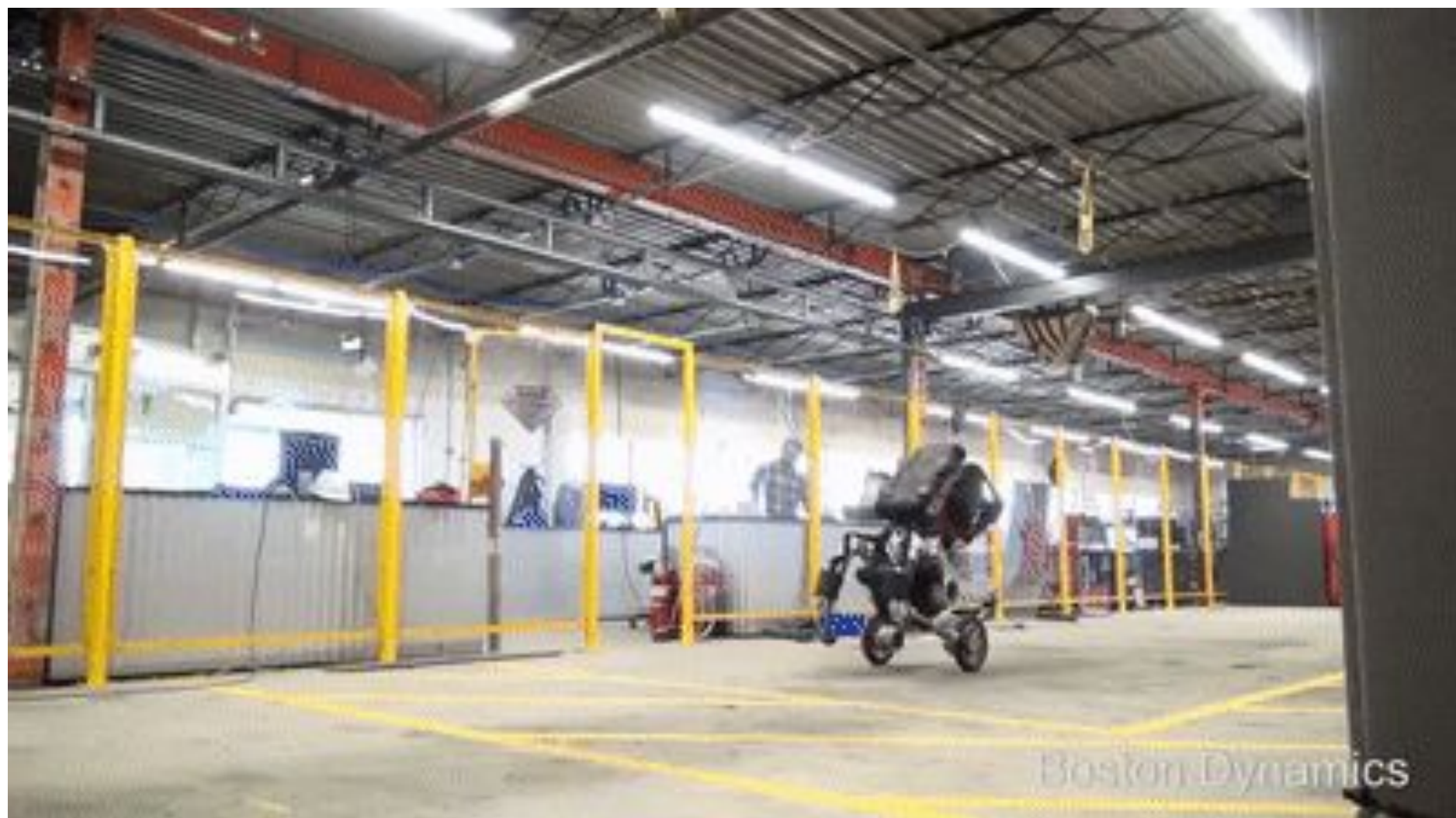
The code trains neural networks to hold negotiations in natural language, and allows reinforcement learning self-play and rollout-based planning.

Robots

Teaching robots to learn how to walk on their own











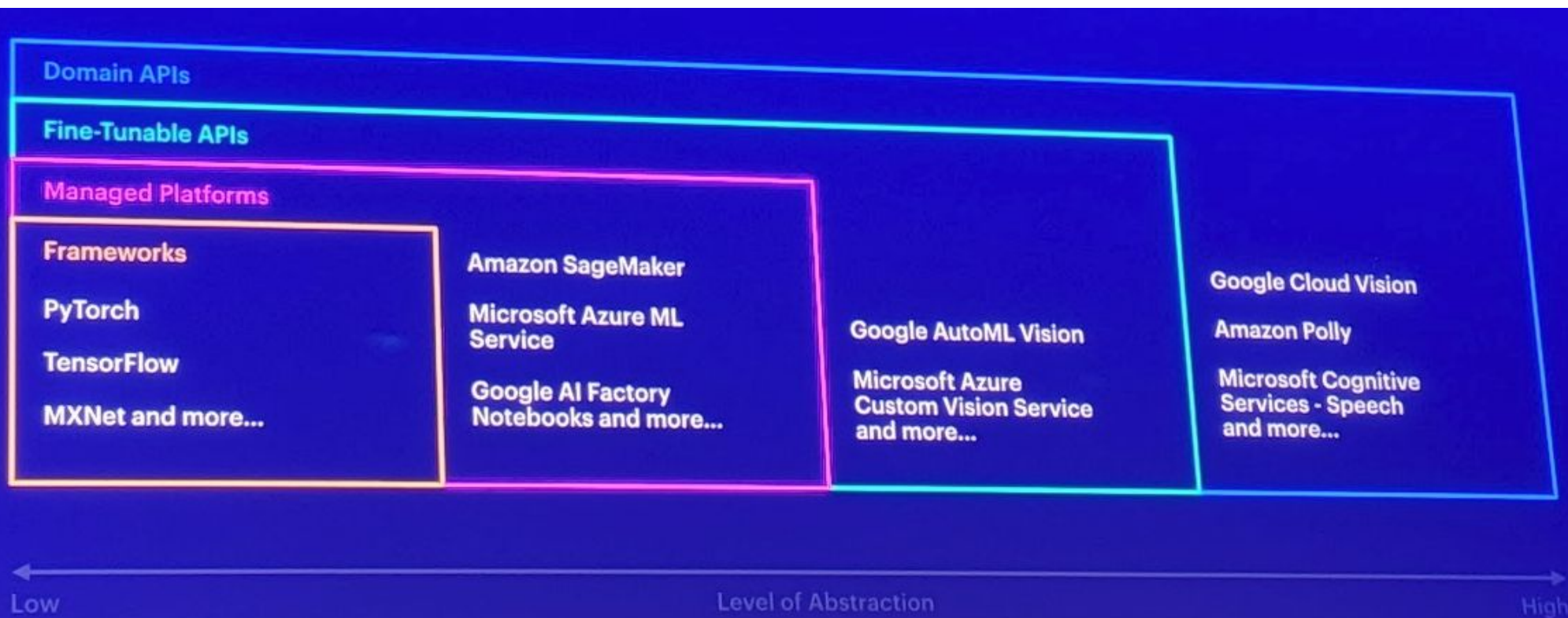
Empezando com ML

- Trabajar con APIs
- Re-entrenar un modelo
- Desarrollar un modelo nuevo

Workflow para ML

- Elegir enfoque y preparar el dataset
- Construir y entrenar un modelo
- Deploy, uso para predicciones & escalar

Herramientas



Herramientas



PyTorch

PyTorch is an open source deep learning framework built to be flexible and modular for research, with the stability and support needed for production deployment. It enables fast, flexible experimentation through a tape-based autograd system designed for immediate and python-like execution.

[GitHub](#)  [Overview](#) >



ONNX

ONNX is an open format for deep learning models, allowing AI developers to easily move between state-of-the-art tools.

[GitHub](#)  [Overview](#) >



Tensor Comprehensions

Tensor Comprehensions accelerates development by automatically generating code from high-level mathematical

[GitHub](#)  [Overview](#) >

Convolución en reconocimiento de imágenes

Necesidad de convolución

Es posible trabajar con *fully connected networks* pero introducen demasiados parámetros.

Con imágenes de 100x100, tendremos 10,000 *weights* por neurona para la siguiente capa. Con convolución, dividimos la imagen en subregiones de 5x5 que requieren trabajo con 25 parámetros

Convolución

En matemática, una convolución es una función aplicada sobre el output de otra función.

En este caso, aplicaremos una multiplicación de matrices, con un filtro aplicado a cada píxel, a través de subregiones de una imagen.

Filtros

| | | |
|----|----|----|
| 0 | -1 | 0 |
| -1 | 5 | -1 |
| 0 | -1 | 0 |

sharpen ▼



The **sharpen** kernel emphasizes differences in adjacent pixel values. This makes the image look more vivid.

ConvNet

Una ConvNet o CNN es capaz de recordar información espacial(color y forma), puede “ver” una imagen completa o en parches, analizando grupos de pixels.

Receptive field

Las neuronas reciben input de una subregión restringida de la imagen. En una capa fully connected, el input es la capa completa, en una capa convolucional, el input es solo una parte (5x5)

Convolution Layer

La capa de convolución aplica distintos filtros(kernels) a lo largo de estos parches.

Input es un tensor con la forma (número de imágenes) * ancho * alto * profundidad.

Pooling

Se usan capas de pooling para reducir las dimensiones combinando los outputs de las neuronas de una capa en una sola neurona para la conexión a la siguiente capa. Los pools locales son de pequeños clusters, normalmente 2x2.