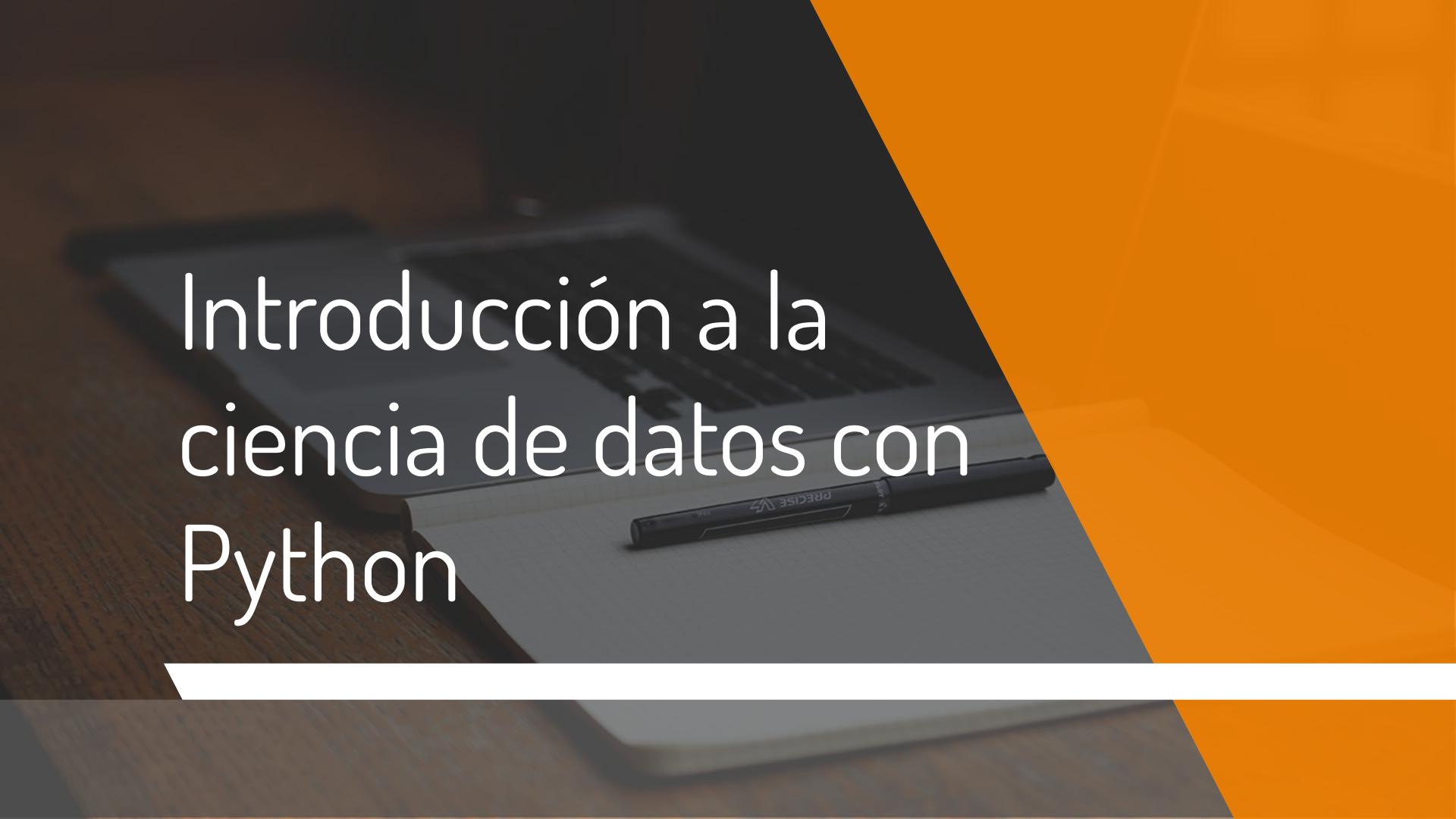


Introducción a la ciencia de datos con Python





deepdaemon.org



¡Hola!

Soy Carlos Duchanoy

Me pueden contactar en
carlos.duchanoy@deepdaemon.com
duchduchanoy@gmail.com



¿Qué es la ciencia de datos?

¿Por qué es útil para mí? ¿Por qué usar Python?

¿Qué aprenderán este curso?

En este curso se les explicará los fundamentos teóricos y las herramientas prácticas para que puedan empezar a implementar algoritmos de Inteligencia Artificial



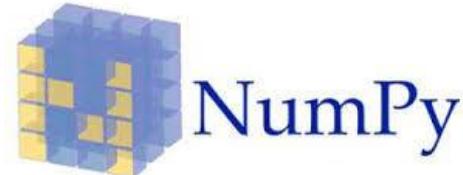
¿Qué se espera de ustedes?



Introducción al ambiente de programación

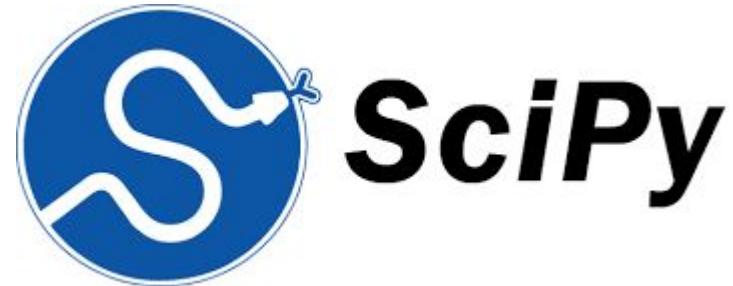
Software a usar durante el curso:

- ▶ **Python 3.6+**
- ▶ **Jupyter notebook:** Consola de ipython
- ▶ **NumPy:** Librería para cómputo científico (manejo de matrices)



Introducción al ambiente de programación

- ▶ **SciPy:** Librería fundamental para cómputo científico



- ▶ **Matplotlib:** Librería para graficar



Anaconda: Ambientes virtuales

Anaconda permite el uso de ambientes virtuales, con lo que es posible tener múltiples instalaciones de Python en la misma computadora, así se pueden tener:

- ▶ Múltiples versiones de Python
- ▶ Si se daña una instalación, se puede reparar sin problemas

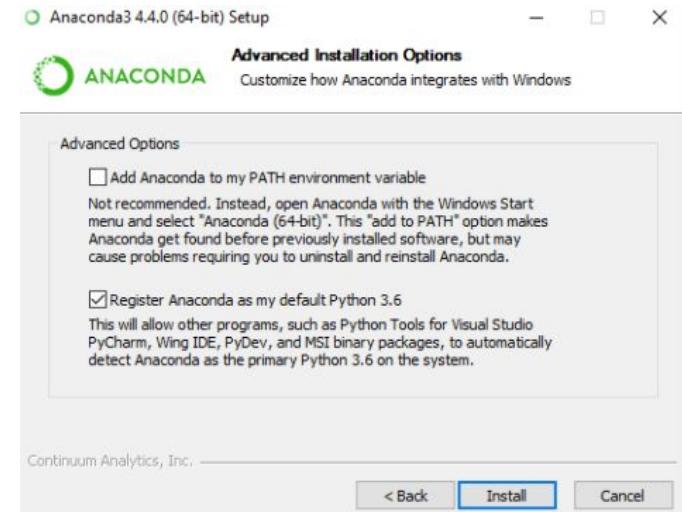


Instalar el software

Instalar Anaconda

<https://www.anaconda.com/download/>

Nota: Para el correcto funcionamiento de los paquetes de desarrollo se recomienda que Anaconda sea la distribución principal de Python en la computadora



Anaconda incluye:

- ▶ Jupyter notebook <http://jupyter.org/>
- ▶ NumPy : <http://www.numpy.org/>
- ▶ Scikit-learn: <http://scikit-learn.org/stable/>
- ▶ SciPy: <https://www.scipy.org>

Uso de anaconda

- ▶ Crear ambientes

`conda create -n tensorflowCPU python=3.5`

- ▶ Activar ambientes

Linux ó Mac: `source activate tensorflowCPU`

Windows: `activate tensorflowCPU`

Instalar matplotlib

Matplotlib es una librería para graficar en 2D y es compatible con ipython

<https://matplotlib.org/>

Usar instalación con pip:

`pip install matplotlib`

Nota: Antes de instalar se debe abrir el ambiente deseado

Instalar TensorFlow TF

Guía de instalación: <https://www.tensorflow.org/install/>

Determinar qué versión de tensorflow instalar:

1. **Tensorflow CPU:** Esta es la versión básica de Tensorflow recomendada cuando no se cuenta con un GPU Nvidia Compatible con Tensorflow
2. **Tensorflow GPU:** Versión que se encuentra implementada para realizar en entrenamiento en GPU compatible, esta versión mejora considerablemente el desempeño de software por eso se recomienda su uso.

Prerrequisitos para instalar Tensorflow GPU

- 1) **Tarjeta GPU NVIDIA:** Con un poder de computo (Compute Capability) de 3.0 or superior, en el siguiente link se puede checar la compatibilidad de las tarjetas <https://developer.nvidia.com/cuda-gpus>
- 2) **Contar con CUDA toolkit 8.0:** Es importante que la versión sea la 8.0 ya que la 9.0 no es compatible adicionalmente es necesario garantizar que se agreguen las direcciones relevantes a cuda en la variable de entorno PATH

<https://developer.nvidia.com/cuda-80-ga2-download-archive>

Prerrequisitos para instalar Tensorflow GPU

- 3) Contar Con **cuDNN V6.1** Es importante agregar a dirección de la carpeta donde se instala cuDNN a la variable de entorno Path
<https://developer.nvidia.com/cudnn>

Preparación para programar

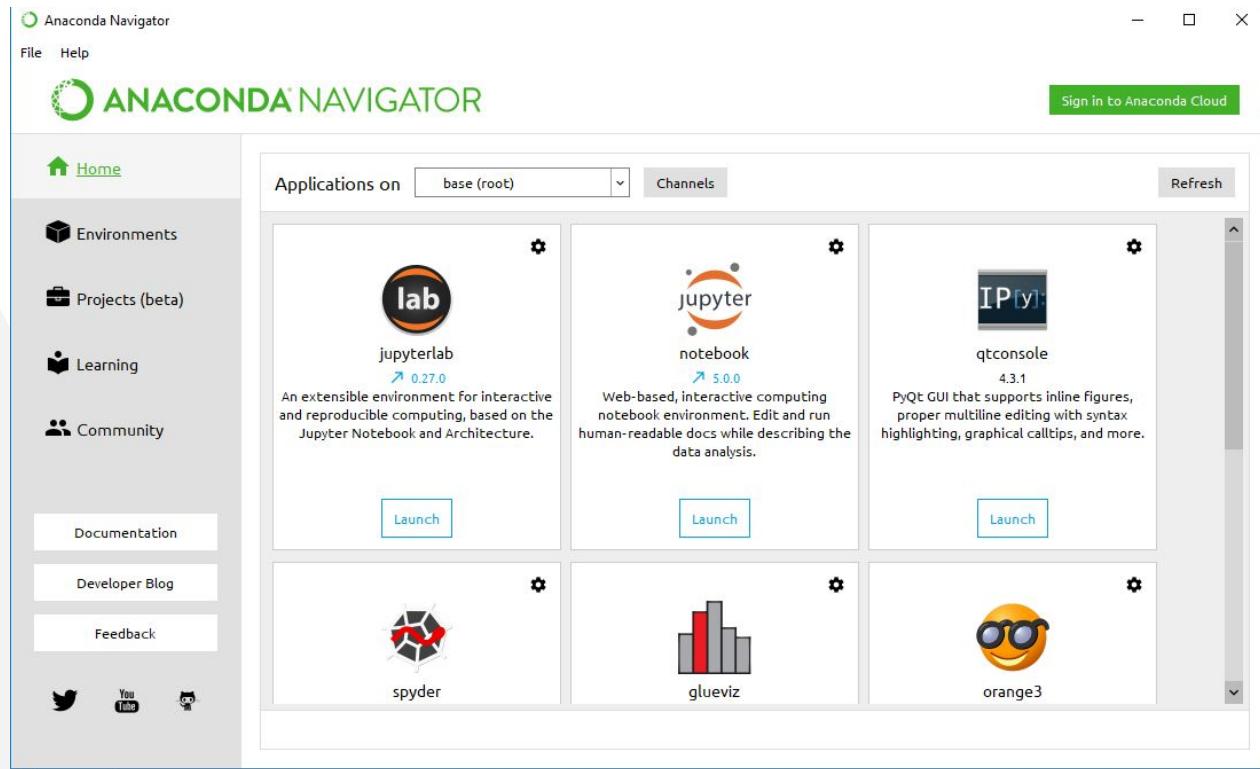
1. Abrir Anaconda Navigator



Alternativa: Abrir una consola

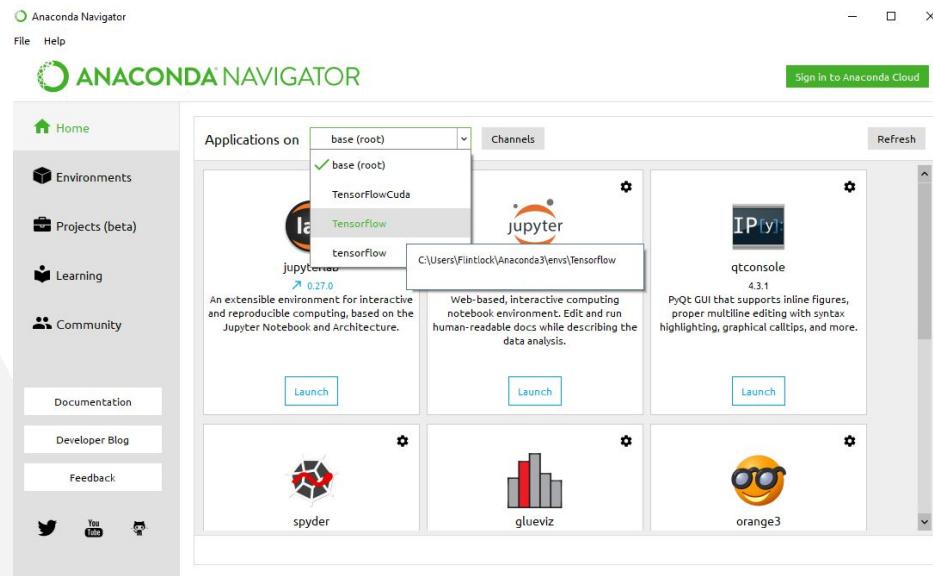


Anaconda Navigator



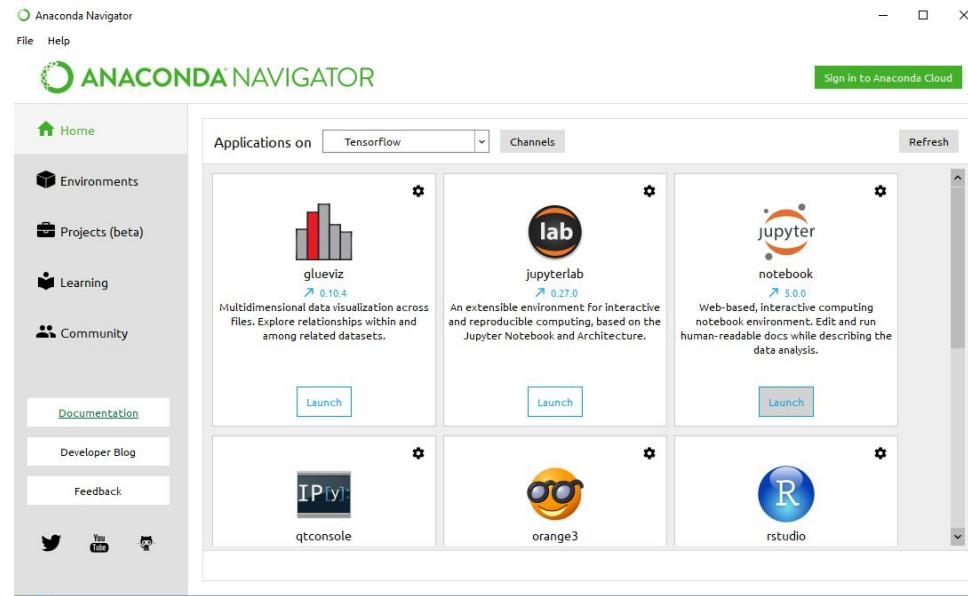
Anaconda Navigator

2. Abrir el ambiente deseado

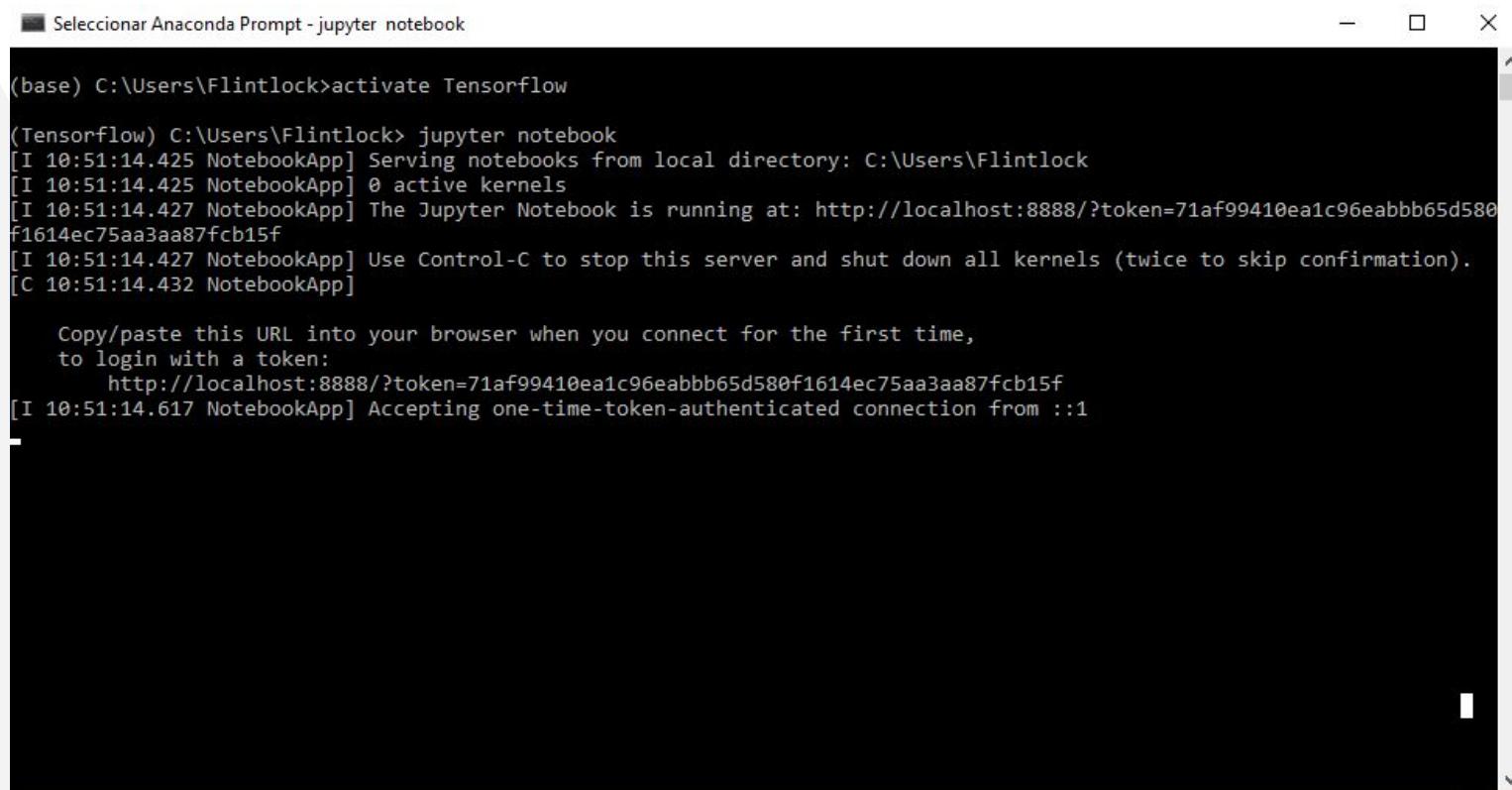


Anaconda Navigator

3. Abrir el ambiente jupyter



Consola ó Anaconda prompt



The screenshot shows a Windows-style terminal window titled "Seleccionar Anaconda Prompt - jupyter notebook". The terminal is running on a Windows operating system, as evidenced by the taskbar icons at the bottom.

```
(base) C:\Users\Flintlock>activate Tensorflow
(Tensorflow) C:\Users\Flintlock> jupyter notebook
[I 10:51:14.425 NotebookApp] Serving notebooks from local directory: C:\Users\Flintlock
[I 10:51:14.425 NotebookApp] 0 active kernels
[I 10:51:14.427 NotebookApp] The Jupyter Notebook is running at: http://localhost:8888/?token=71af99410ea1c96eabbb65d580f1614ec75aa3aa87fc15f
[I 10:51:14.427 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 10:51:14.432 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
http://localhost:8888/?token=71af99410ea1c96eabbb65d580f1614ec75aa3aa87fc15f
[I 10:51:14.617 NotebookApp] Accepting one-time-token-authenticated connection from ::1
```

Elementos básicos de Python

1. Uso de la Jupyter Console
2. Fundamentos de Python
3. NumPy
4. Matplotlib

Abrir: [Apuntes de Python Básico del curso introducción a la ciencia de datos con Python](#)

NumPy

1. NumPy (numpy, NP) es una librería de álgebra lineal para Python; es una librería muy importante porque es la base de todas las demás librerías de cómputo científico.
2. Permite un procesamiento de datos a una velocidad superior, ya que parte de esta librería se encuentra en lenguaje C.
3. NumPy permite el uso de arreglos y matrices.

Abrir: Apuntes Numpy del curso introducción a la ciencia de datos con Python

Matplotlib

Es una herramienta para graficar datos en Python. Sus principales ventajas son:

- ▶ Permite graficar fácilmente
- ▶ Tiene soporte para etiquetas en las gráficas
- ▶ Gran control de cada elemento en la figura
- ▶ Permite exportar la figura en varios formatos
- ▶ Se puede ajustar a muchas necesidades

Recursos: <http://matplotlib.org/>

1.

Proyecto Chatbot

Empecemos con lo básico de la Inteligencia Artificial

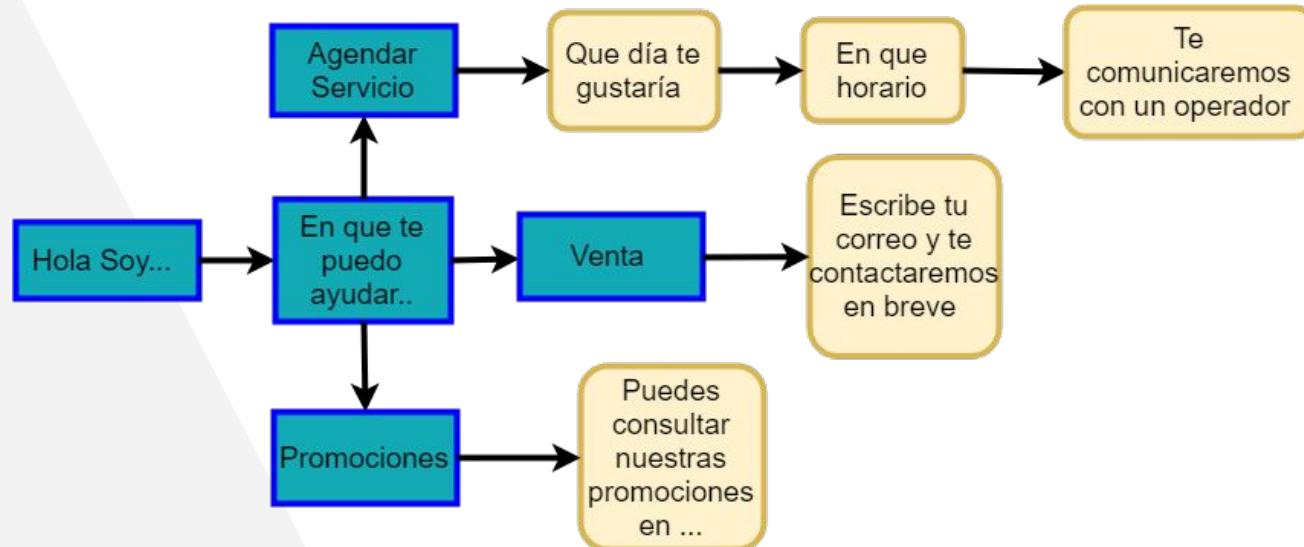


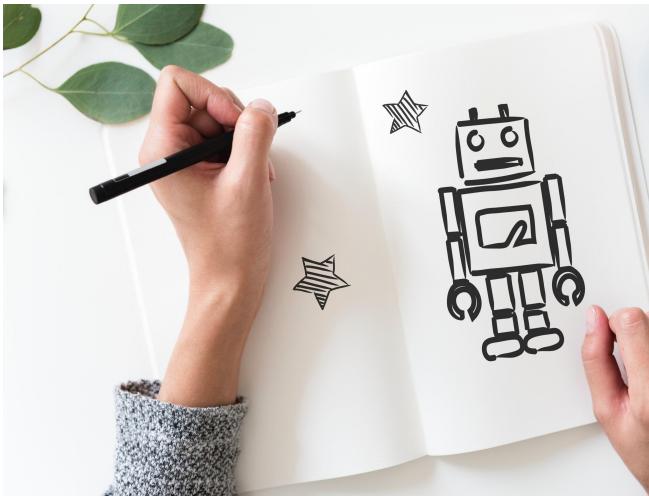
Asistente Ford

Tratemos de hacer un pequeño asistente vía chat para una agencia Ford

Diagrama del asistente

Para realizar un chatbot sencillo es necesario conocer la lógica con la que va a operar, por ejemplo:





¿Creen que este chatbot puede considerarse como Inteligencia Artificial?

Introducción al aprendizaje máquina tradicional

“

Definiciones de Inteligencia Artificial

Teoría y el desarrollo de sistemas informáticos capaces de realizar tareas que normalmente requieren inteligencia humana, como la percepción visual, el reconocimiento del lenguaje, la toma de decisiones y la traducción entre idiomas

English Oxford Living Dictionary

”

“

Definiciones de Inteligencia Artificial

Inteligencia Artificial (IA), es la capacidad de una computadora o un robot para realizar tareas comúnmente asociadas con seres inteligentes.

Encyclopedia Britannica

”

¿Qué es Inteligencia Artificial?

1. Rama de la computación relacionada con la simulación del comportamiento inteligente en las computadoras.

2. La capacidad de una máquina para imitar el comportamiento humano.

La Inteligencia Artificial incluye muchos algoritmos

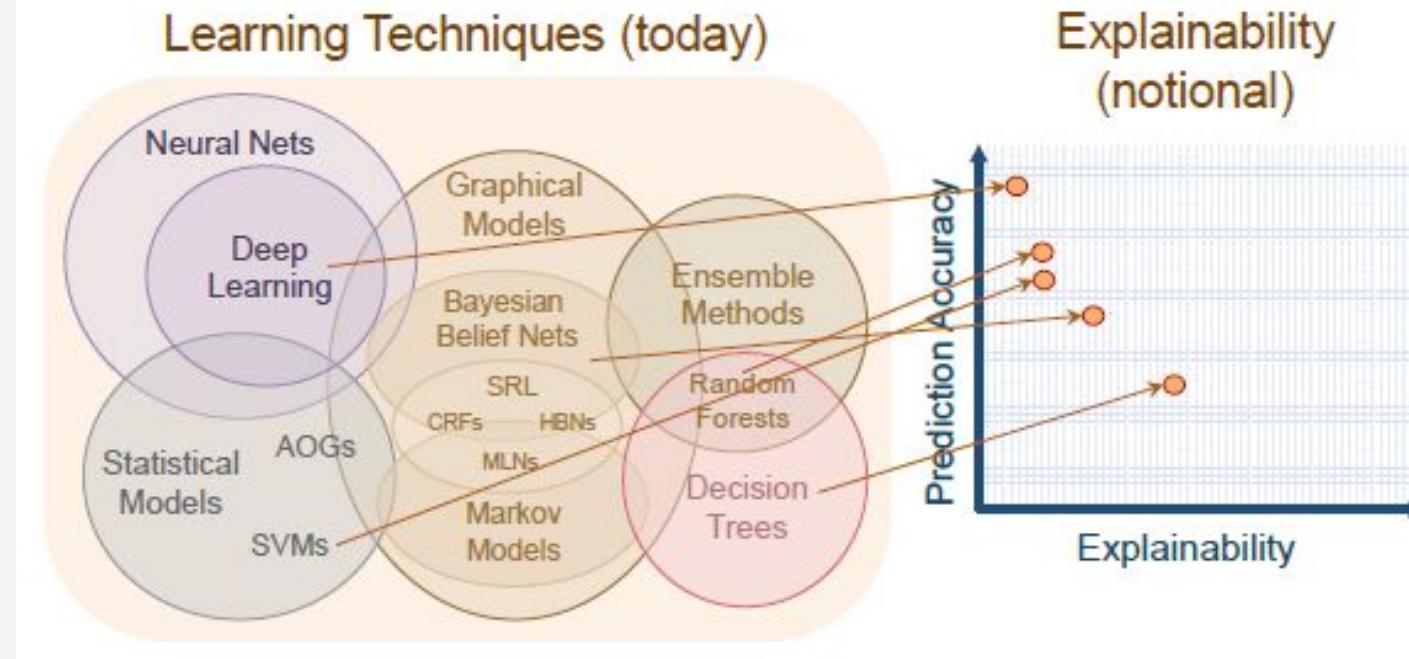


Imagen tomada de Explainable Artificial Intelligence (XAI) DARPA-BAA-16-53-2016 [1]

¿Inteligencia Artificial es igual a Aprendizaje Máquina ?

Inteligencia
Artificial



Capacidad de pensar
como humanos

Aprendizaje
Maquina



Aprender sin ser
programados
explicitamente

Aprendizaje Profundo



Capacidad de extraer
información de datos
no estructurados

¿Qué es el Aprendizaje Máquina?

- ▶ Es un método de análisis de datos que automatiza la creación de modelos.
- ▶ Estos modelos pueden ser usados para muchas cosas entre las que destacan: Regresión, clasificación, reducción de dimensionalidad y agrupamiento

¿Cuál es la ventaja del Aprendizaje Máquina?

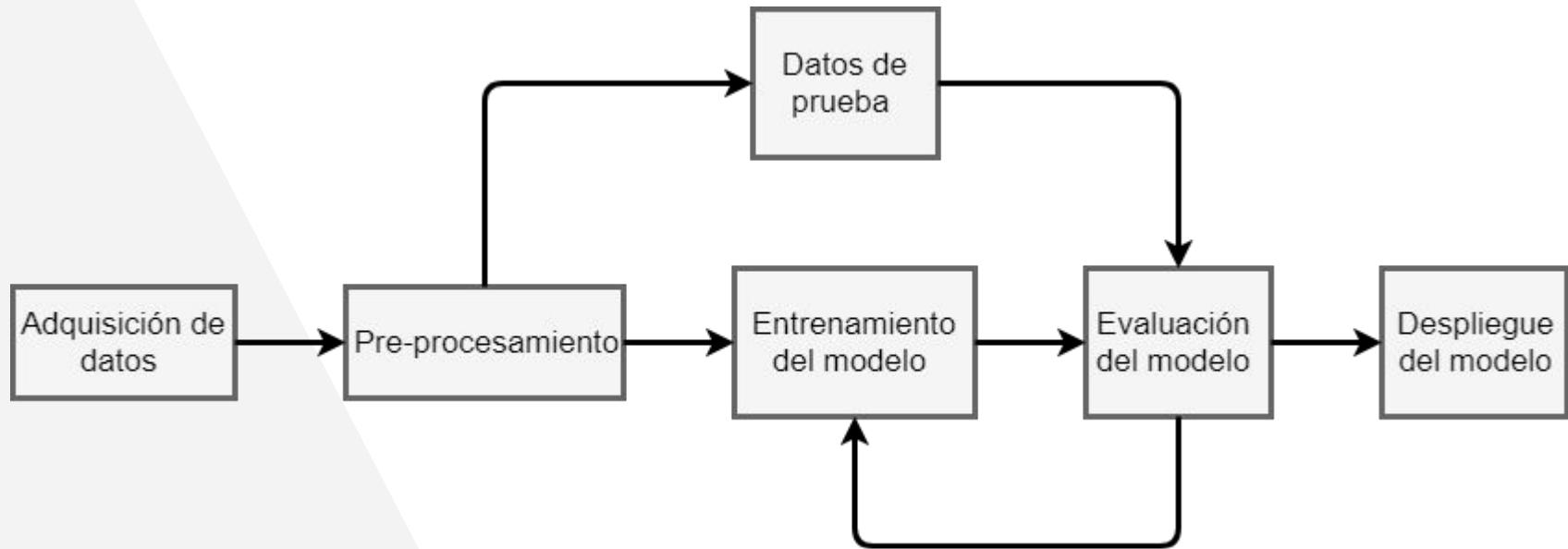
Los modelos se ajustan por medio de algoritmos iterativos que aprenden de los datos, permitiéndole a las computadoras encontrar las relaciones entre estos sin ser explícitamente programadas. Algunos ejemplos donde se usa son:

- ▶ Detección de fraudes
- ▶ Segmentación de clientes
- ▶ Reconocimiento de patrones
- ▶ Sistemas de recomendación
- ▶ Análisis de sentimiento
- ▶ Análisis crediticio

¿En qué me ayuda el Aprendizaje Máquina?

1. Reducir el tiempo para programar soluciones (corrector ortográfico)
2. Facilita escalar soluciones (múltiples idiomas)
3. Permite resolver cosas que de otra forma se consideran imposibles de programar (reconocer sentimientos)

Proceso general del Aprendizaje Máquina



Tipos de Aprendizaje Máquina

Existen tres:

- ▶ Supervisado
- ▶ No-supervisado
- ▶ Reforzado



Aprendizaje supervisado

Los algoritmos son entrenados usando datos etiquetados



Aprendizaje supervisado

Los algoritmos son entrenados usando datos etiquetados



Carro



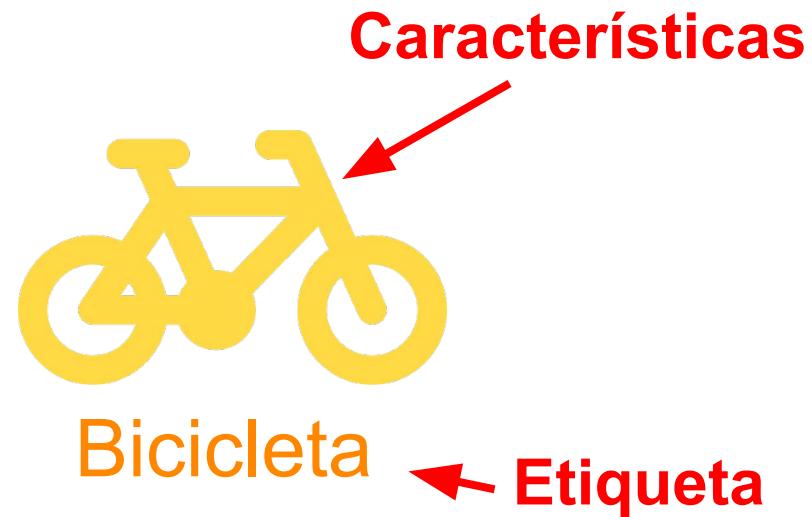
Bicicleta

Aprendizaje supervisado

Los algoritmos son entrenados usando datos etiquetados



Carro

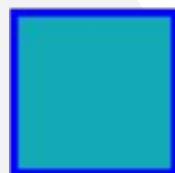
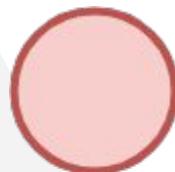


Aprendizaje supervisado

- ▶ Este algoritmo recibe un conjunto de entradas y su correspondiente salida deseada
- ▶ El algoritmo aprende por medio de comparar la salida del modelo con la salida correcta
- ▶ El modelo se ajusta de acuerdo a este error

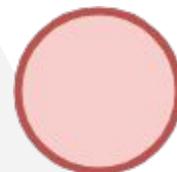
Aprendizaje supervisado: Ejemplo

- 1) Mostrar las entradas y pedir una salida

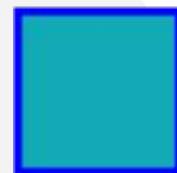


Aprendizaje supervisado: Ejemplo

- 2) Mostrar las salidas correctas y ajustar el modelo



Clase A



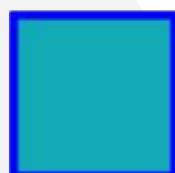
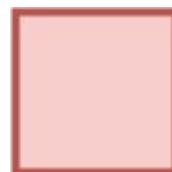
Clase B

Aprendizaje supervisado: Ejemplo

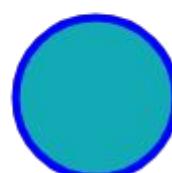
- 1) Mostrar nuevas entradas y pedir una salida



Clase A

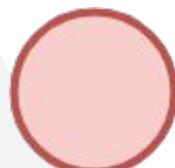


Clase B

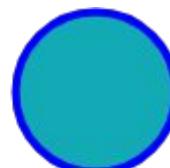


Aprendizaje supervisado: Ejemplo

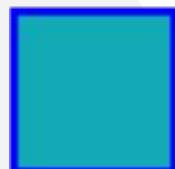
- 2) Mostrar las salidas correctas y ajustar el modelo



Clase A



Clase A



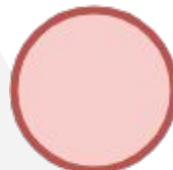
Clase B



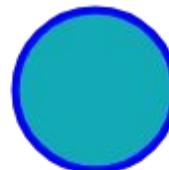
Clase B

Aprendizaje supervisado: Ejemplo

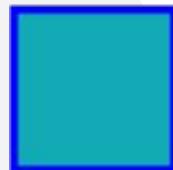
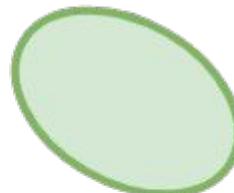
- 3) Una vez que se ha entrenado la red con todos los ejemplos, se realiza una prueba



Clase A



Clase A



Clase B



Clase B



Aprendizaje supervisado: Ejemplo

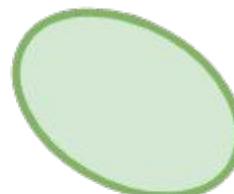
- 4) Se verifican los resultados. Si fueron satisfactorios se usa el sistema, si no se entrena con más ejemplos



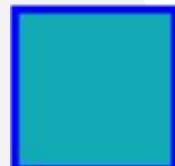
Clase A



Clase A



Clase A



Clase B



Clase B



Clase B

¿Para qué sirve el aprendizaje supervisado?

- ▶ El aprendizaje supervisado se usa para predecir los valores (etiquetas) de datos no vistos por un humano
- ▶ Es muy usado en tareas donde se puede predecir un resultado futuro a partir de datos históricos

Ejemplos de aprendizaje supervisado

- ▶ Anticipar si una transacción de tarjeta de crédito es fraudulenta
- ▶ ¿Qué tan probable es que un asegurado cobre su seguro?
- ▶ El precio de una propiedad dada su información histórica

Aprendizaje no supervisado

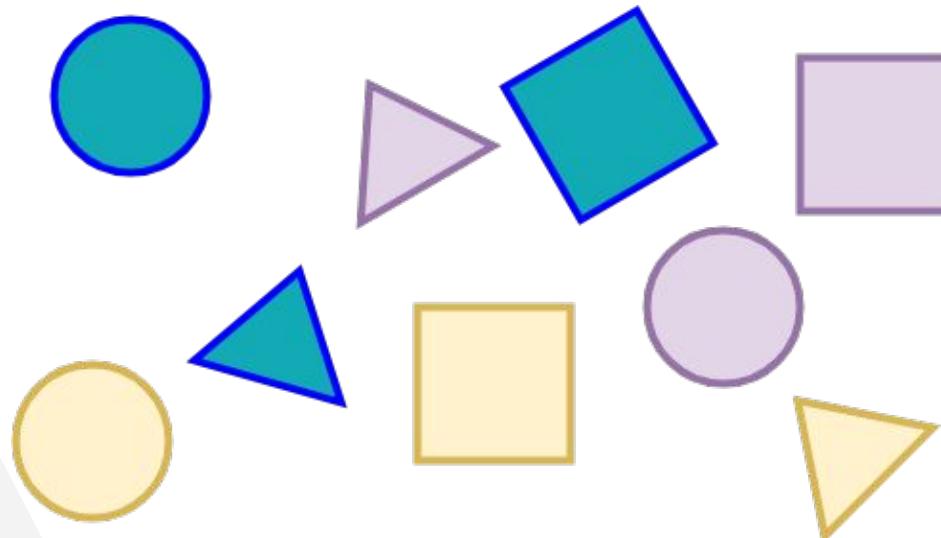
- ▶ Este tipo de aprendizaje utiliza un conjunto de datos que no se encuentran etiquetados
- ▶ El objetivo es encontrar una lógica o estructura en los datos

Aprendizaje no supervisado

- ▶ Aquí no se da la “**respuesta correcta**”, los algoritmos deben de determinar por sí solos qué hacer con la información proporcionada
- ▶ Esto permite encontrar los mejores atributos o separar entre elementos

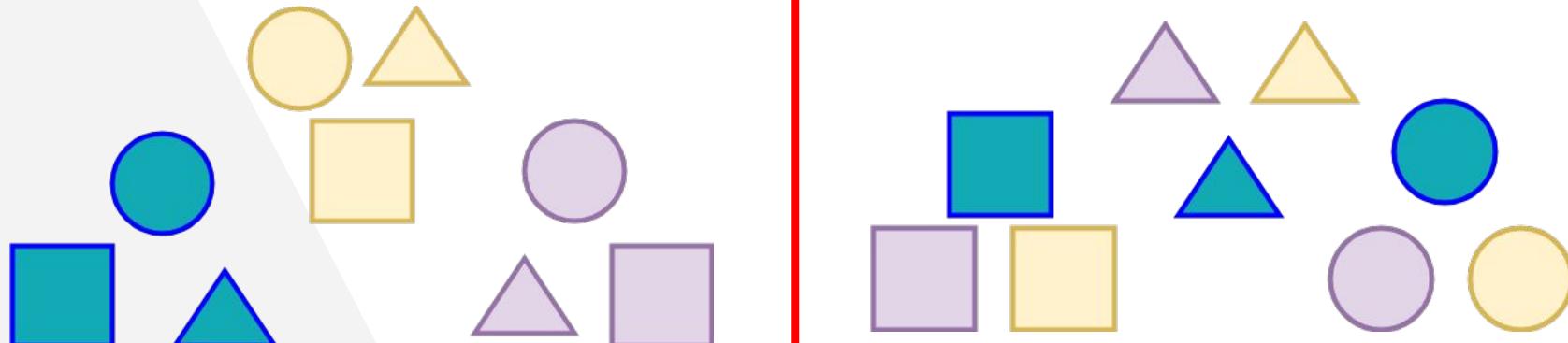
Aprendizaje no supervisado: Ejemplo

1. Se proporcionan los datos y el algoritmo busca ordenar la información



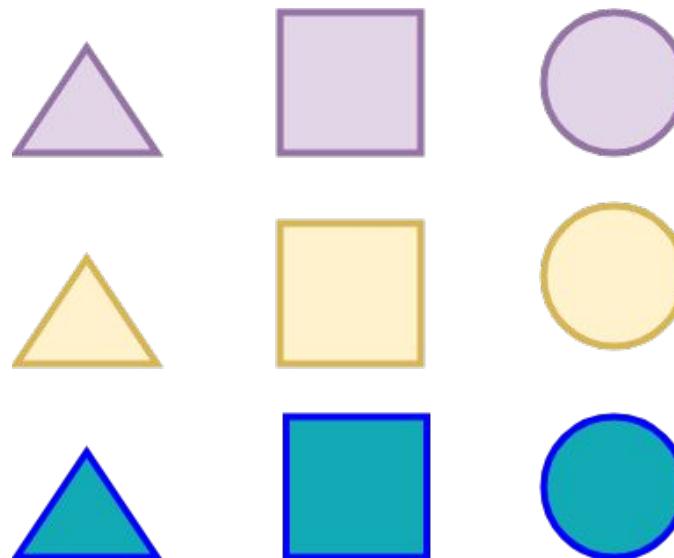
Aprendizaje no supervisado: Ejemplo

2. El algoritmo busca relaciones entre los elementos



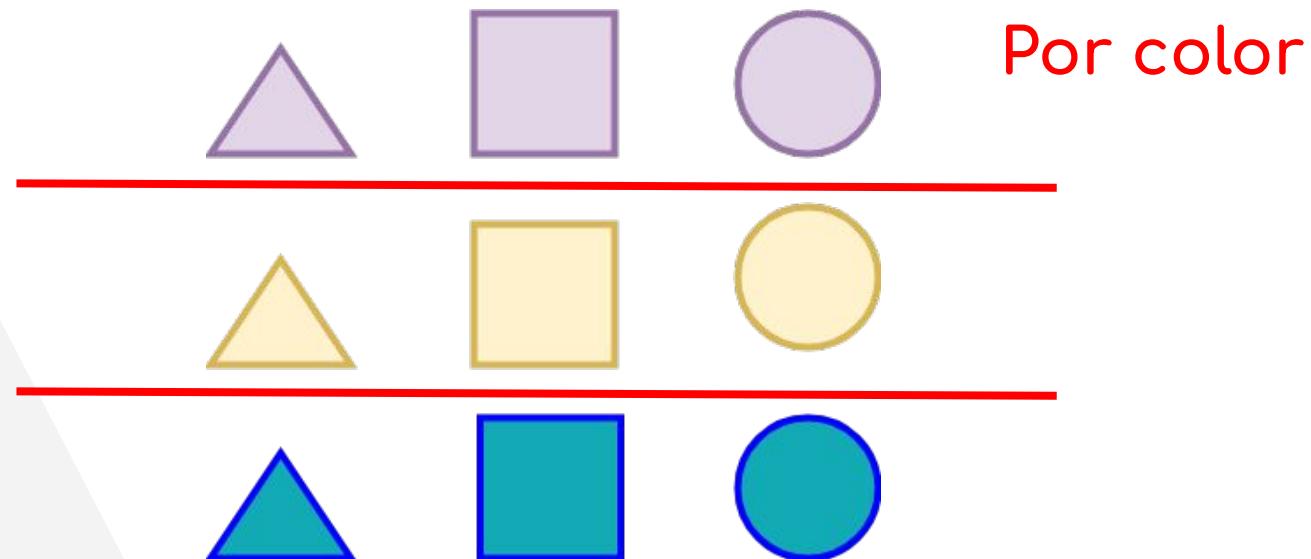
Aprendizaje no supervisado: Ejemplo

Se pueden tener diversos ordenamientos “correctos”. Sin embargo, su utilidad varía dependiendo del problema que queramos resolver



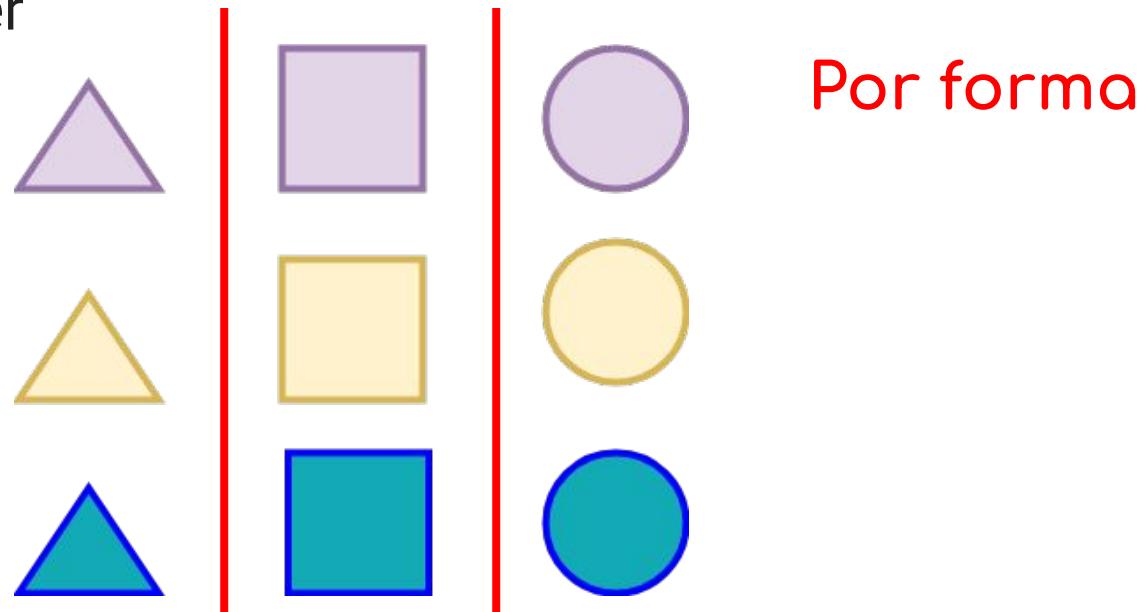
Aprendizaje no supervisado: Ejemplo

Se pueden tener diversos ordenamientos “correctos”. Sin embargo, su utilidad varía dependiendo del problema que queramos resolver



Aprendizaje no supervisado: Ejemplo

Se pueden tener diversos ordenamientos “correctos”. Sin embargo, su utilidad varía dependiendo del problema que queramos resolver



¿Para qué sirve el aprendizaje no supervisado?

- ▶ Analiza cómo se relacionan los datos
- ▶ Permite encontrar diferentes maneras para segmentar los datos
- ▶ Identifica automáticamente cuales son las características más significativas de los datos

Ejemplos de aplicaciones de aprendizaje no supervisado

- ▶ Segmentación de clientes
- ▶ Análisis de tópicos en un texto
- ▶ Recomendación de productos
- ▶ Identificación de anomalías

Aprendizaje reforzado

Este tipo de aprendizaje tiene tres componentes principales:

1. **El agente** (algoritmo que está aprendiendo)
2. **El ambiente** (aquellos que interactúan con el agente)
3. **Las acciones** (lo que puede hacer el agente)

Aprendizaje reforzado

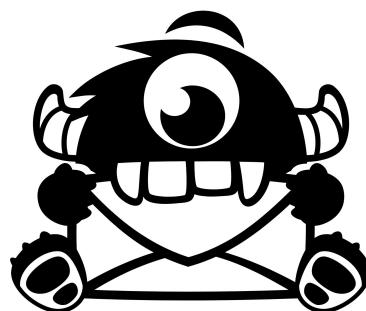
- ▶ Este tipo de aprendizaje descubre por medio de prueba y error, qué acción provee la mayor recompensa.
- ▶ El objetivo es que el agente escoja las acciones más adecuadas para mejorar la recompensa.

Aprendizaje reforzado: Ejemplo



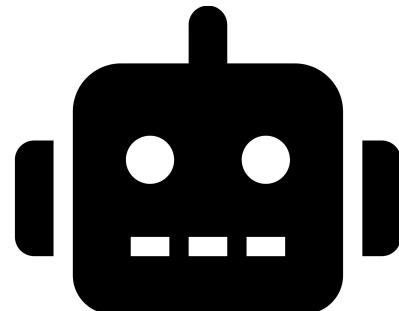
¿Para qué sirve el aprendizaje reforzado?

- ▶ Se usa en casos en que es más fácil instrumentar y realizar pruebas físicas que generar modelos u obtener datos históricos.



Ejemplos de aplicaciones de aprendizaje reforzado

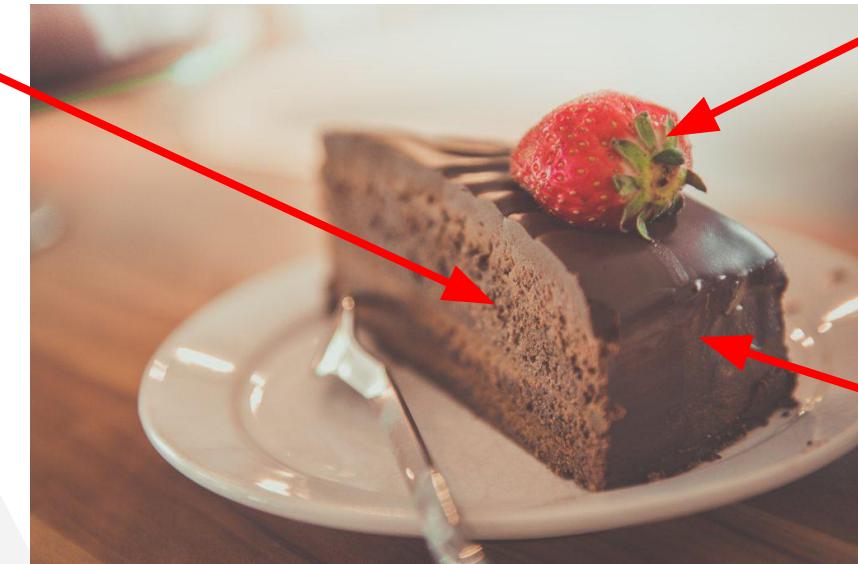
- ▶ Control de robots
- ▶ Chatbots
- ▶ Autos autónomos



¿Cómo escoger un tipo de aprendizaje?

No supervisado

La máquina analiza millones de datos y ordena la información (conocer el mundo)



Reforzado

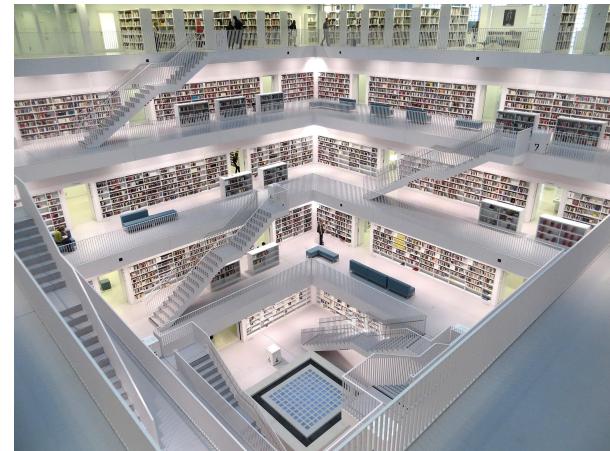
La máquina perfecciona sus habilidades (superación)

Supervisado

La máquina se prepara para realizar una tarea en específico (capacitación)

¿Qué permite realizar el aprendizaje máquina?

- ▶ Regresión
- ▶ Clasificación
- ▶ Reducción dimensional
- ▶ Agrupación



<http://www-bcf.usc.edu/~gareth/ISL/ISLR%20Sixth%20Printing.pdf>

Aprendizaje Máquina y Python (Scikit-learn)

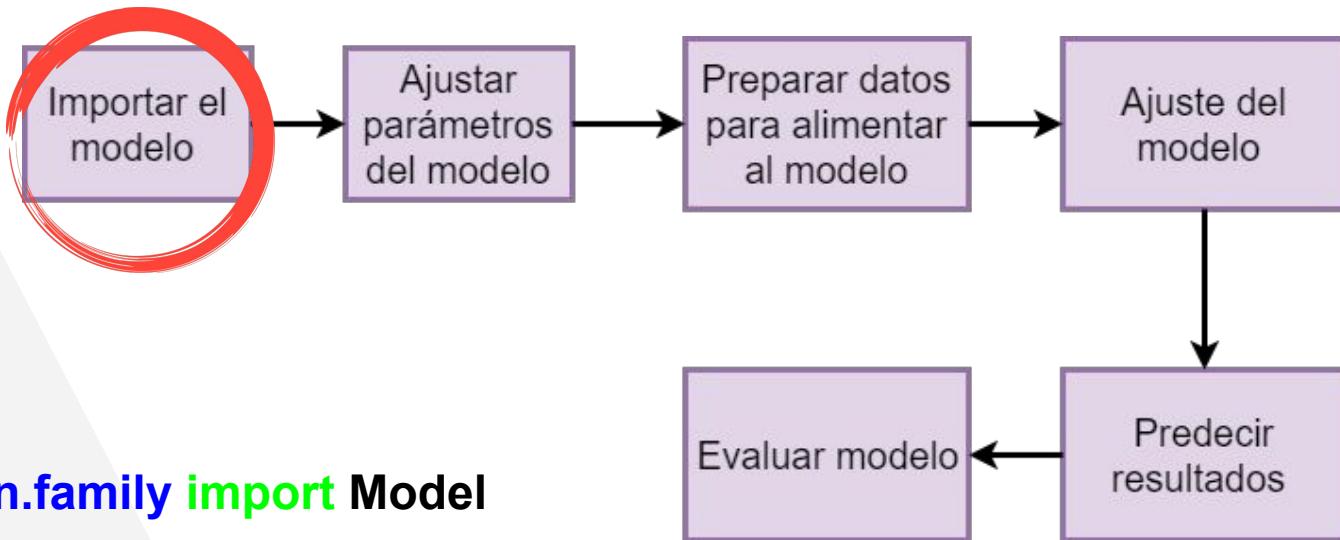
Scikit-learn es la librería más popular para el uso de aprendizaje máquina en Python

conda install scikit-learn

ó

pip install scikit.learn

Scikit-learn: Importar el modelo

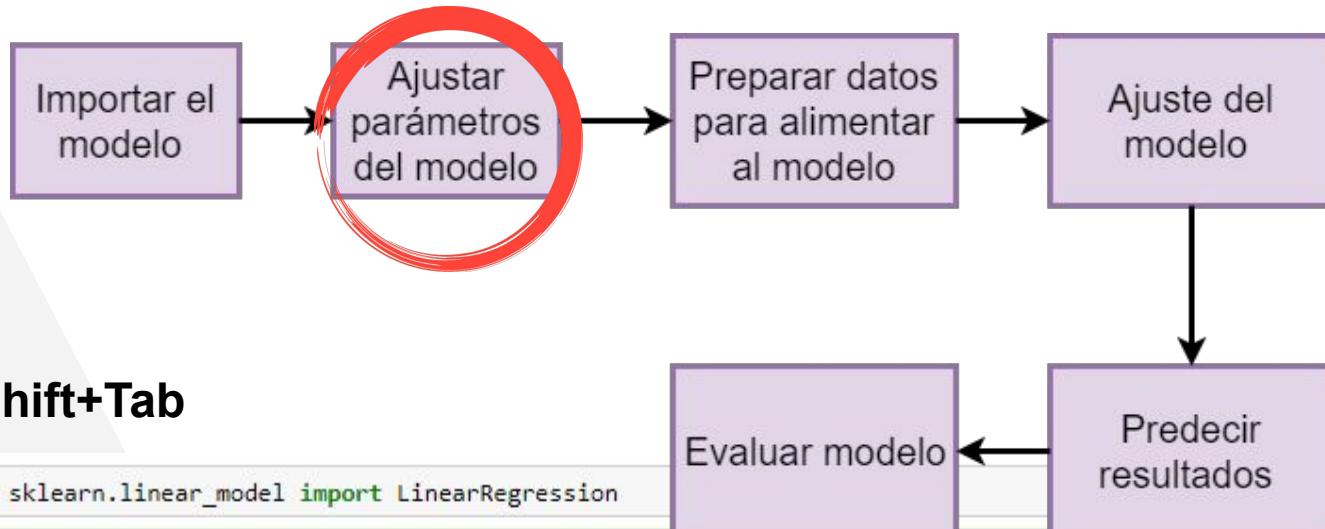


From sklearn.family import Model

Ejemplo:

From sklearn.linear_model import LinearRegression

Scikit-learn: Parámetros del modelo



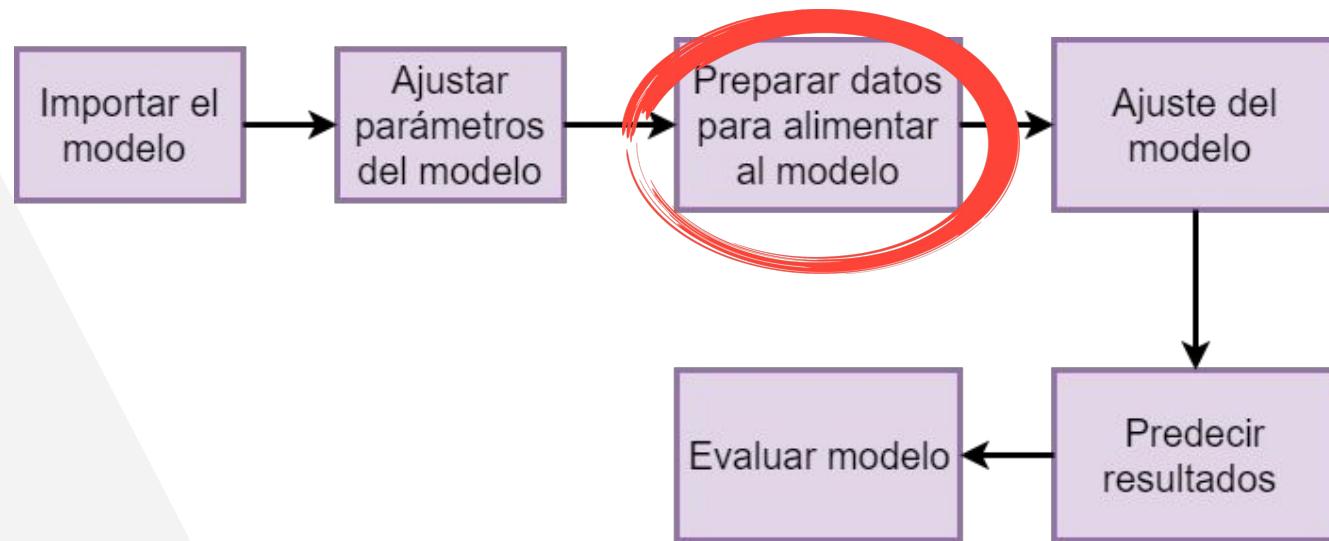
Ejemplo: Shift+Tab

```
In [4]: from sklearn.linear_model import LinearRegression
```

```
In [ ]: model=LinearRegression(
```

```
Init signature: LinearRegression(fit_intercept=True, normalize=False, copy_X=True, n_jobs=1)
Docstring:
Ordinary least squares Linear Regression.
```

Scikit-learn: Preparar los datos



Separar los datos en conjunto de entrenamiento y conjunto de pruebas

Scikit-learn: Separar los datos

```
import numpy as np
from sklearn.model_selection import train_test_split
```

```
X, y = np.arange(10).reshape((5, 2)), range(5)
X
```

```
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
```

```
list(y)
```

```
[0, 1, 2, 3, 4]
```

Scikit-learn: Separar los datos

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

```
X_train
```

```
array([[4, 5],  
       [0, 1],  
       [6, 7]])
```

```
y_train
```

```
[2, 0, 3]
```

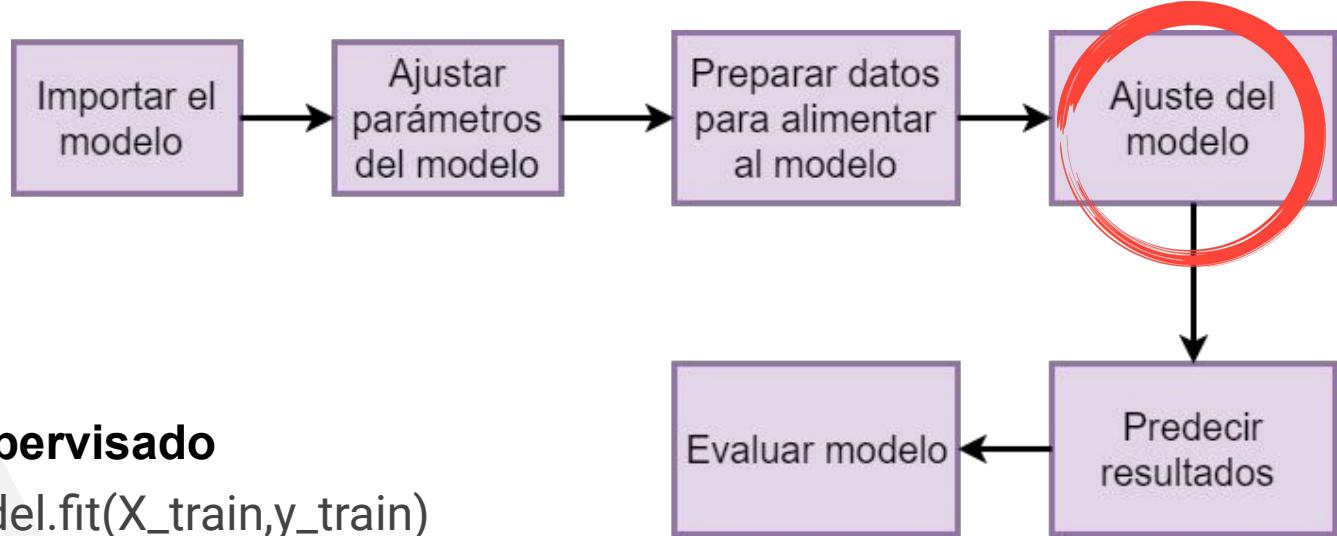
```
X_test
```

```
array([[2, 3],  
       [8, 9]])
```

```
y_test
```

```
[1, 4]
```

Scikit-learn: Ajustar el modelo



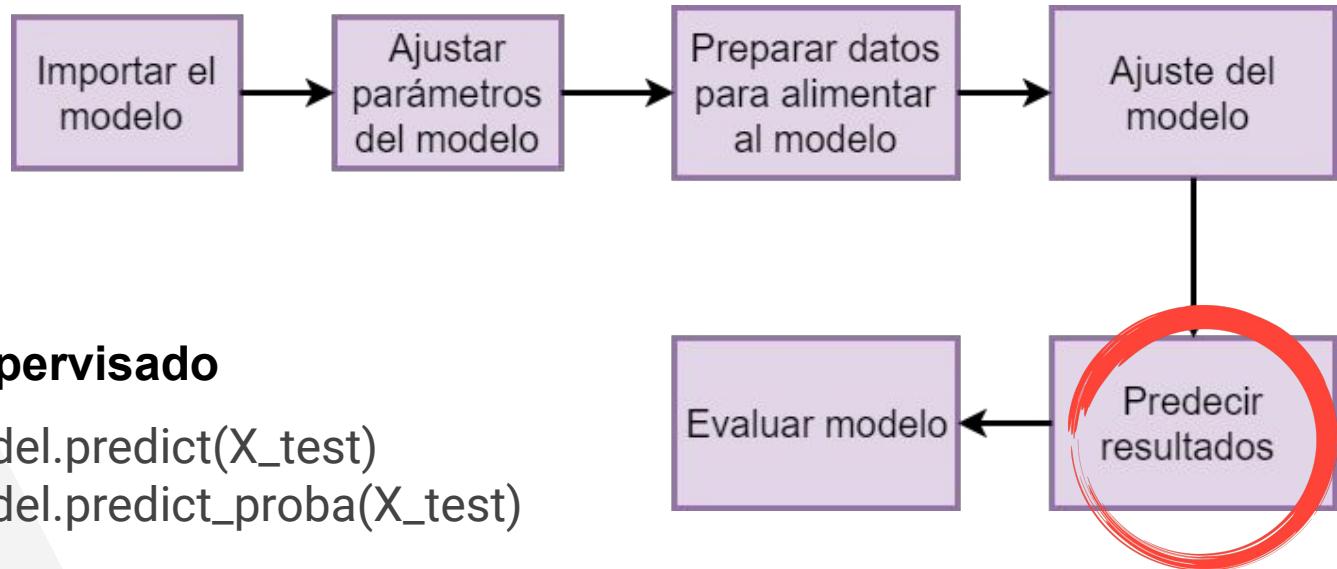
Supervisado

```
model.fit(X_train,y_train)
```

No-supervisado

```
model.fit(X_train)
```

Scikit-learn: Predecir resultados



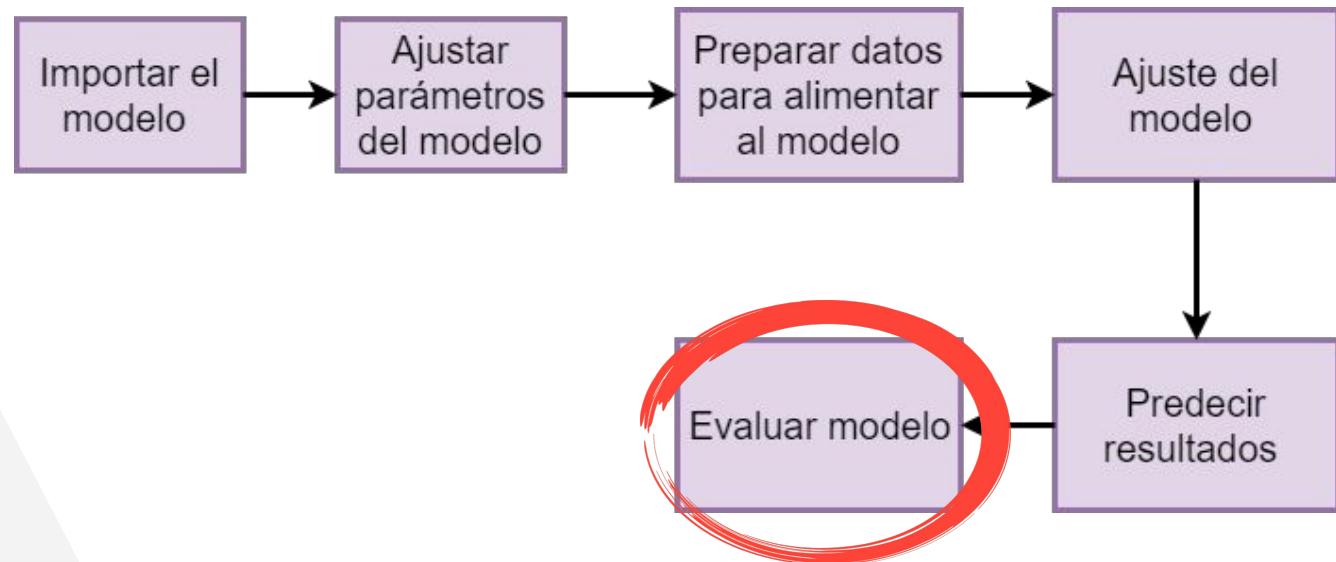
Supervisado

```
model.predict(X_test)  
model.predict_proba(X_test)
```

No-supervisado

```
model.predict(X_test)
```

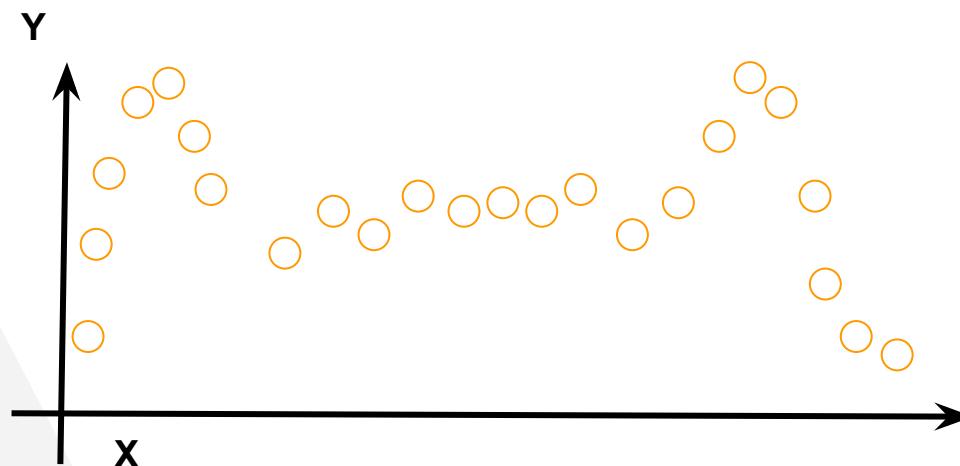
Scikit-learn: Evaluar modelo



Este paso depende del tipo de modelo que se está usando y lo que se quiere conseguir

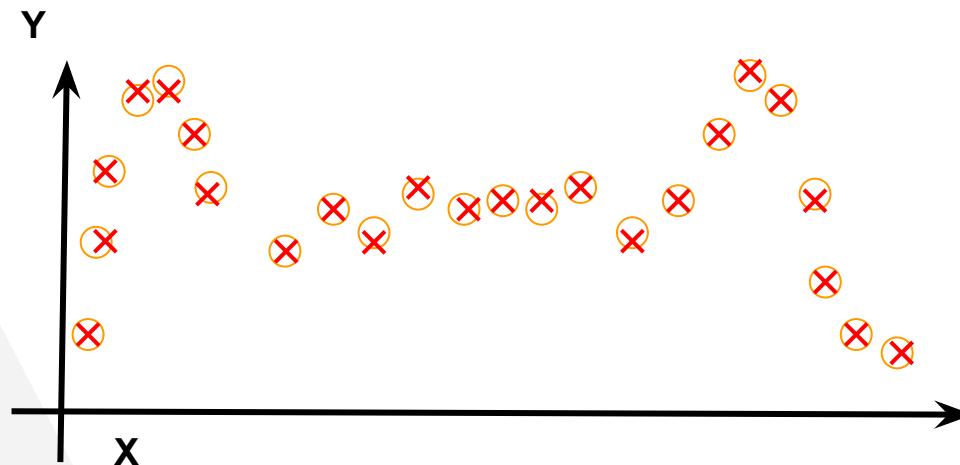
Regresión

Un problema de regresión consiste en un modelo que relaciona un conjunto de características con una respuesta numérica



Regresión

Un problema de regresión consiste en un modelo que relaciona un conjunto de características con una respuesta numérica



Ejemplos de aplicaciones de regresión

- ▶ Predicción de la temperatura de un día
- ▶ Estimar la probabilidad de que un usuario vea un anuncio publicitario
- ▶ Diseño de sensores suaves
- ▶ Predicción de fallas en una línea de producción

¿Qué se necesita para plantear un problema de regresión?

- ▶ El problema de regresión requiere predecir una cantidad numérica
- ▶ Las entradas pueden ser reales o discretas
- ▶ Puede tener muchas variables de entrada (**regresión multivariable**)
- ▶ Puede tener variables de entrada ordenadas temporalmente (**predicción ó *forecasting***)

¿Qué nos permite hacer una regresión?

- ▶ Identificar qué variables influyen para llegar a un resultado
- ▶ Predecir cuál será el resultado dado un conjunto de variables

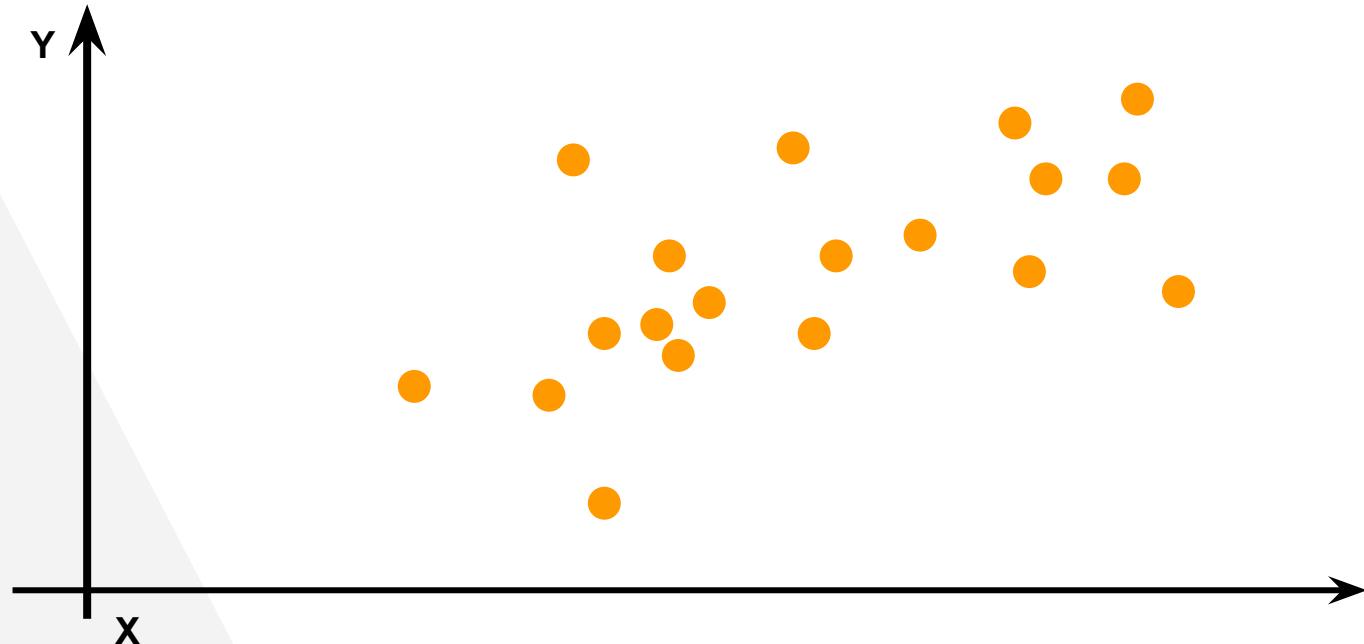
¿Qué se necesita para plantear un problema de regresión?

- ▶ El problema de regresión requiere predecir una cantidad numérica (**aprendizaje supervisado**)
- ▶ Las entradas pueden ser reales o discretas
- ▶ Puede tener muchas variables de entrada (**regresión multivariable**)
- ▶ Puede tener variables de entrada ordenadas temporalmente (**predicción ó forecasting**)

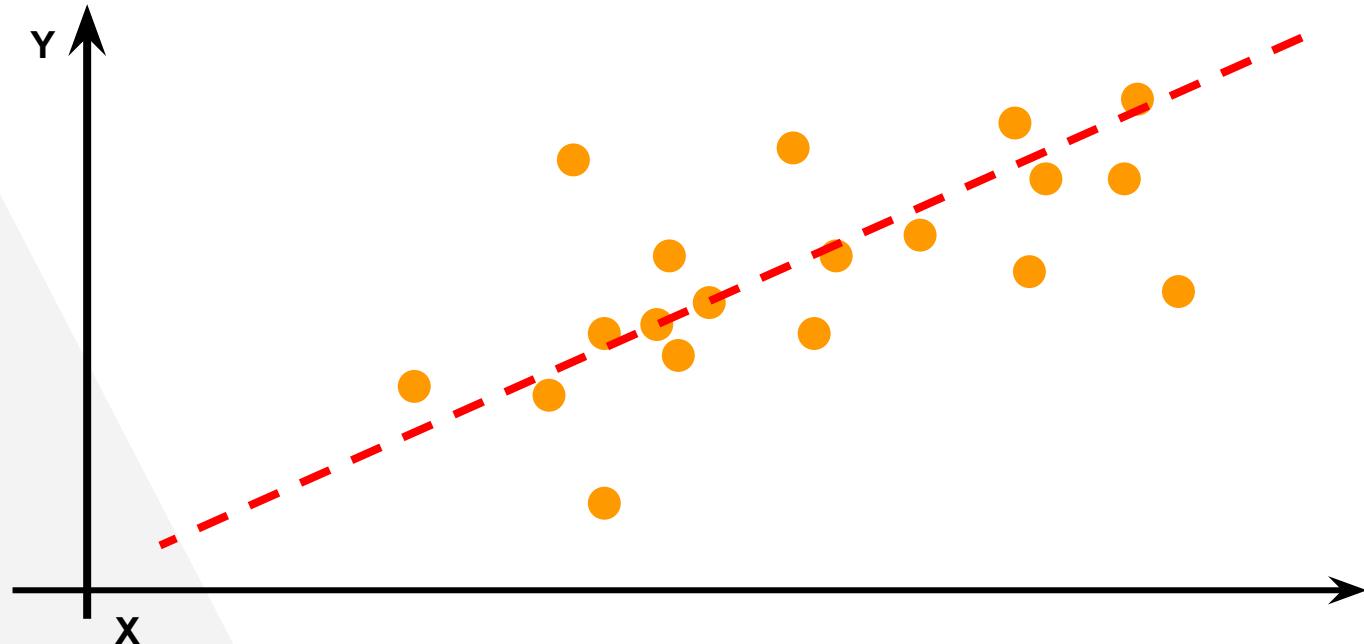
¿Qué nos permite hacer una regresión?

- ▶ Identificar qué variables influyen para llegar a un resultado
- ▶ Predecir cuál será el resultado dado un conjunto de variables

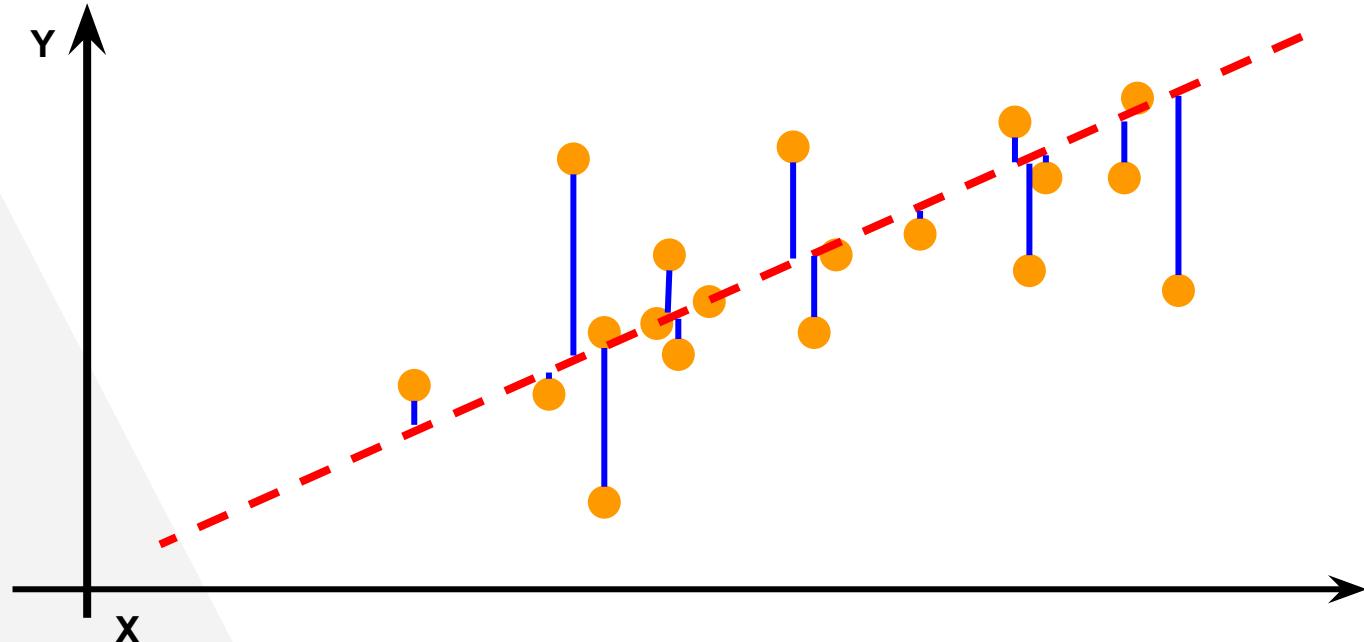
Ejemplo: Regresor lineal



Ejemplo: Regresor lineal



¿Cómo medir el desempeño de un regresor?



[Abrir: Apuntes regresión lineal del curso aprendizaje máquina](#)

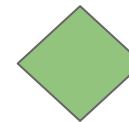
¿Qué regresores puedo usar?

```
from sklearn.ensemble import RandomForestRegressor  
from sklearn.neighbors import KNeighborsRegressor  
from sklearn.linear_model import LinearRegression  
from sklearn.neural_network import MLPRegressor
```



Clasificación

Un problema de clasificación consiste en realizar un modelo que sea capaz de separar elementos en conjuntos a partir de sus características



Aplicaciones: Ejemplos de clasificación

- ▶ Saber si un correo es spam o no
- ▶ Saber si un producto cumple la calidad o no
- ▶ Segmentación de clientes
- ▶ Análisis de sentimiento
- ▶ Clasificación de paquetes

¿Qué se necesita para plantear un problema de clasificación?

- ▶ El problema requiere ejemplos clasificados en dos o más clases (**aprendizaje supervisado**)
- ▶ Puede tener entradas con valores reales o discretos

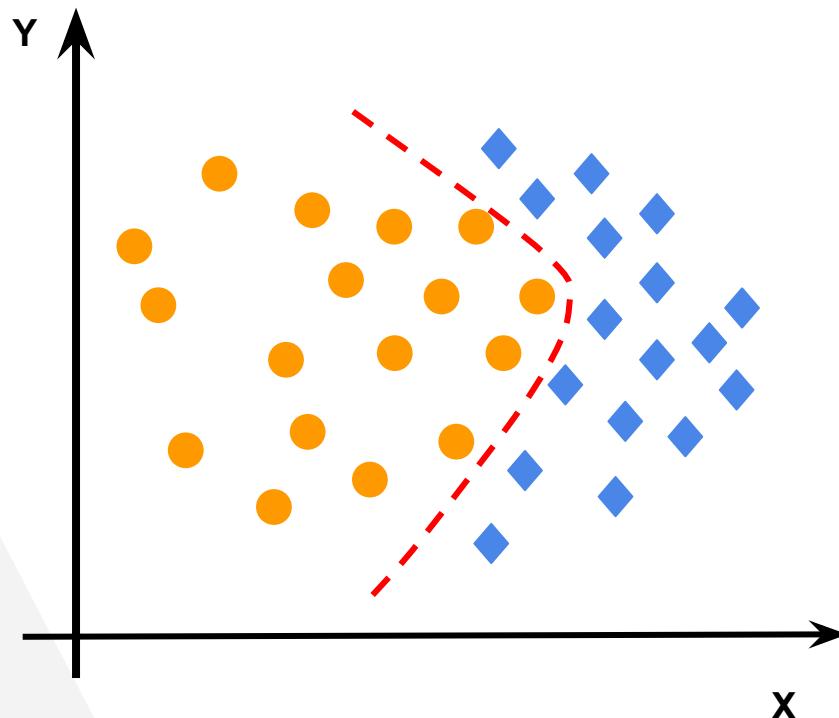
Tipos de problemas de clasificación

- ▶ Un problema con dos clases se denomina **binario**
- ▶ Un problema con muchas clases se denomina **multi-clase**
- ▶ Cuando un elemento pertenece a múltiples clases se le denomina problema **multi-etiqueta**

¿Qué nos permite hacer una clasificación?

- ▶ Identificar qué variables se deben considerar para separar las clases
- ▶ Predecir a qué clase corresponde un elemento que no haya sido clasificado

Ejemplo de clasificación



¿Cómo medir el desempeño de un clasificador?

La exactitud es la manera más común de medir el desempeño de un clasificador

$$A_{cc} = \frac{N_c}{T_p}$$

Donde:

A_{cc} Exactitud

N_c Número de predicciones correctas

T_p Número total de predicciones

Ejemplo: Clasificar la intención del pastor

Fábula del joven pastor y el lobo

- ▶ El joven pastor se aburre de cuidar el rebaño del pueblo y, para divertirse, grita: "¡Lobo!"
- ▶ El joven pastor ve un lobo real acercándose al rebaño y grita: "¡Lobo!"

Hagamos un clasificador que decida si le creemos o no al pastor

¿La exactitud es una buena medida de desempeño?

Supongamos que analizamos el desempeño del clasificador de intención del pastor y se obtiene la siguiente exactitud:

$$A_{cc} = 99.00\%$$

Nuestro clasificador es **bueno**, ¿cierto?

Analizando el problema a detalle: Matriz de confusión

<p>Verdadero Positivo (VP)</p> <ul style="list-style-type: none">• En realidad hay un lobo• El pastor gritó lobo y le creyeron <p>Resultado: El pastor es un héroe</p>	<p>Falso Positivo (FP)</p> <ul style="list-style-type: none">• En Realidad No hay un lobo• El pastor gritó lobo y le creyeron <p>Resultado: Los vecinos son despertados por nada y están molestos</p>
<p>Falso Negativo (FN)</p> <ul style="list-style-type: none">• En Realidad hay un lobo• El pastor gritó lobo y No le creyeron <p>Resultado: El lobo se comió al pastor y a las ovejas</p>	<p>Verdadero Negativo (VN)</p> <ul style="list-style-type: none">• En realidad No hay un lobo• El pastor gritó lobo y No le creyeron <p>Resultado: Todo mundo duerme plácidamente</p>

Matriz de confusión del para el caso del joven pastor

Supongamos que el pastor gritó **100 veces**, recuerden nuestro clasificador obtuvo 99% de exactitud

Verdadero Positivo (VP) <ul style="list-style-type: none">• En realidad hay un lobo• El pastor gritó lobo y le creyeron 0	Falso Positivo (FP) <ul style="list-style-type: none">• En realidad No hay un lobo• El pastor gritó lobo y le creyeron 0
Falso Negativo (FN) <ul style="list-style-type: none">• En Realidad hay un lobo• El pastor gritó lobo y No le creyeron 1	Verdadero Negativo (VN) <ul style="list-style-type: none">• En realidad No hay un lobo• El pastor gritó lobo y No le creyeron 99

¿Nuestro clasificador es **bueno**?

Recuerden, no todos los errores son igual de costosos

Veamos un caso más real hagamos un sistema que decida si debemos o no autorizar un préstamo

<p>Verdadero Positivo (VP)</p> <ul style="list-style-type: none"> • Se aprueba el préstamo • El acreedor paga el préstamo <p>Resultado: tenemos una ganancia </p>	<p>Falso Positivo (FP)</p> <ul style="list-style-type: none"> • Se aprueba el préstamo • El acreedor No paga el préstamo <p>Resultado: Perdemos todo el dinero prestado </p>
<p>Falso Negativo (FN)</p> <ul style="list-style-type: none"> • No se aprueba el préstamo • El acreedor era capaz de pagar el préstamo <p>Resultado: Perdemos la ganancia del préstamo </p>	<p>Verdadero Negativo (VN)</p> <ul style="list-style-type: none"> • No se aprueba el préstamo • El acreedor No era capaz de pagar el préstamo <p>Resultado: Todo bien</p>

Abrir: Apuntes Clasificador MSV del curso aprendizaje máquina

¿Qué clasificadores puedo usar?

```
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.svm import SVC, LinearSVC, NuSVC  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import RandomForestClassifier,  
    AdaBoostClassifier, GradientBoostingClassifier  
from sklearn.naive_bayes import GaussianNB  
from sklearn.neural_network import MLPClassifier
```

<https://www.kaggle.com/jeffd23/10-classifier-showdown-in-scikit-learn>

Reducción dimensional

Un problema de reducción dimensional busca resumir o describir los datos usando menos información



Aplicaciones de reducción dimensional

- ▶ Visualizar información multidimensional
- ▶ Identificar qué datos influyen más para la toma de decisiones
- ▶ Prepara los datos para otros algoritmos de aprendizaje máquina
- ▶ Reducir el espacio de las bases de datos

¿Qué se necesita para plantear un problema de reducción dimensional?

- ▶ El problema requiere datos no etiquetados (**aprendizaje no-supervisado**)
- ▶ Puede tener entradas con valores reales o discretos

¿Qué reducción dimensional puedo usar?

```
from sklearn.decomposition import PCA
```

```
from sklearn.random_projection import GaussianRandomProjection
```

```
from sklearn.manifold import TSNE
```

```
from sklearn.decomposition import FastICA
```



Preprocesamiento de señales

1. **Estandarización (Standardization)**: genera datos con una media cero y una varianza de una unidad
2. **Normalización (Normalization)**: proceso de lleva los datos a tener una norma de máxima de una unidad

<http://scikit-learn.org/stable/modules/preprocessing.html>

Agrupación

Un problema de agrupamiento busca encontrar relaciones entre los datos para ordenarlos en grupos



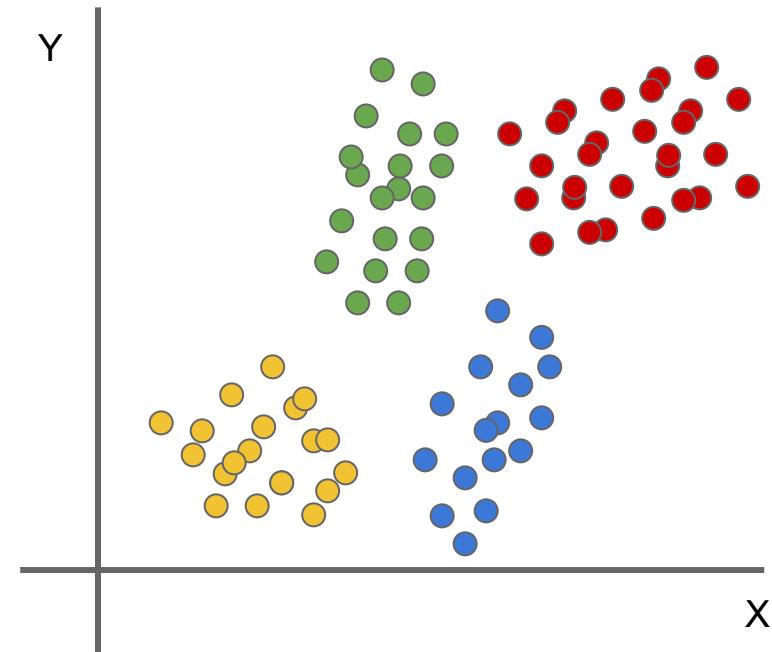
Aplicaciones de agrupación

- ▶ Preparación de datos para otro método de machine learning
- ▶ Detección de anomalías
- ▶ Segmentación de datos (**segmentación de clientes**)
- ▶ Análisis de datos

¿Qué se necesita para plantear un problema de agrupación?

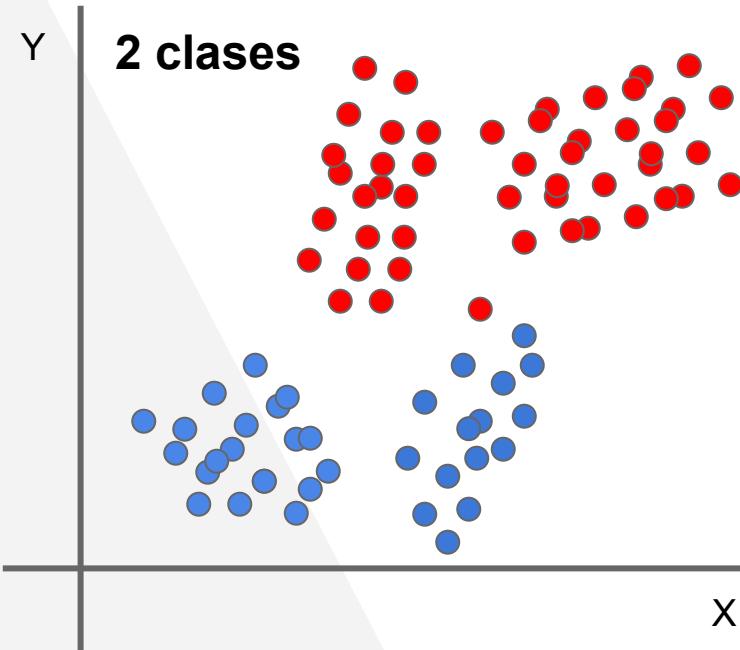
- ▶ El problema requiere datos no etiquetados
(aprendizaje no-supervisado)
- ▶ Puede tener entradas con valores reales o discretos

Ejemplo: Agrupación

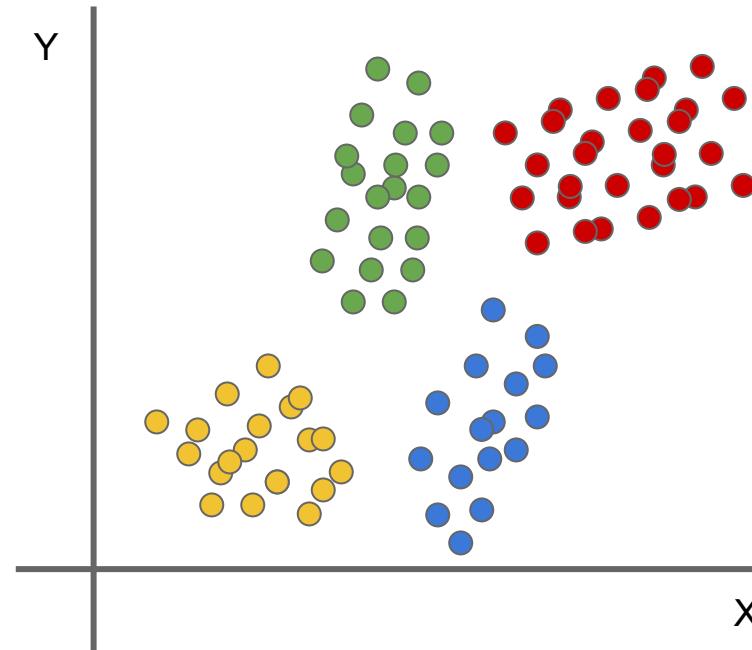


Tipos de agrupamiento

Por número de clases



Por cercanía o similitud



[Abrir: Apuntes de agrupación del curso aprendizaje máquina](#)

¿Qué métodos de agrupación puedo usar?

```
from sklearn.cluster import KMeans
```

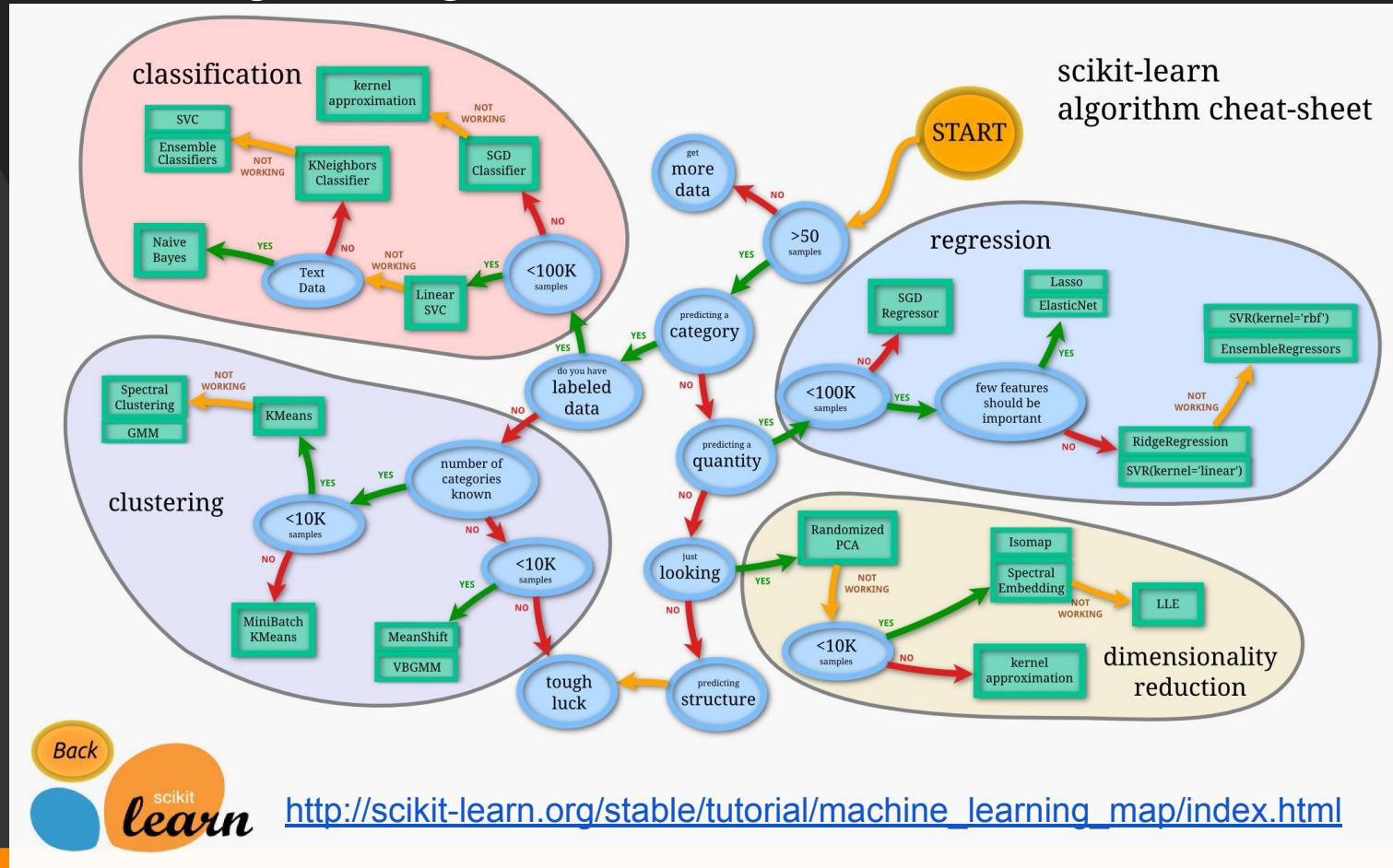
```
from sklearn.cluster import SpectralClustering
```

```
from sklearn.cluster import DBSCAN
```

```
from sklearn.cluster import Birch
```

```
from sklearn.cluster import AgglomerativeClustering
```

¿Cómo escoger un algoritmo?



Cursos de Machine Learning

1. <https://developers.google.com/machine-learning/crash-course/>

2. <https://pythonprogramming.net/machine-learning-tutorial-python-introduction/>

¿Por qué enfocar el resto del curso en las redes neuronales?

- ▶ Ustedes quieren obtener soluciones

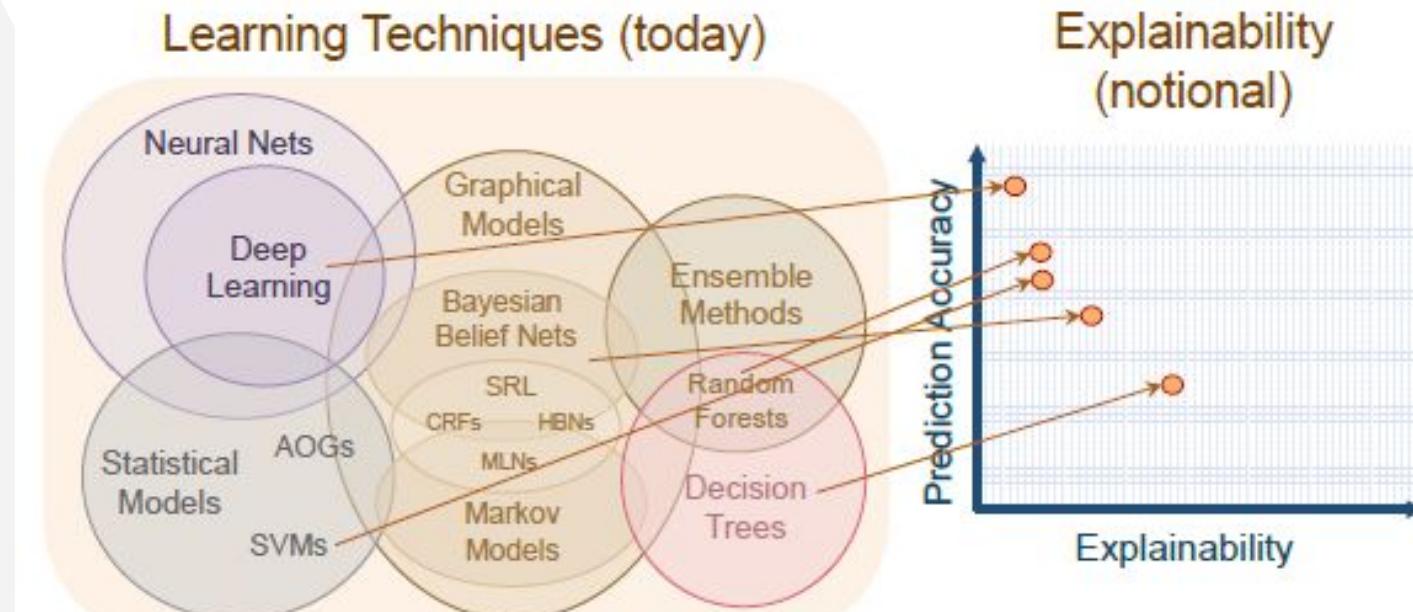


Imagen tomada de Explainable Artificial Intelligence (XAI) DARPA-BAA-16-53-2016 [1]

¿Qué ventajas tienen las redes neuronales?

1. Tienen la capacidad de modelar relaciones complejas no lineales.
2. Son capaces de generalizar, ya que después de aprender con un conjunto de datos acotado pueden inferir relaciones en datos nunca antes vistos.

¿Qué ventajas tienen las redes neuronales?

3. A diferencia de otros métodos de aprendizaje máquina las redes neuronales no tienen ninguna restricción en los tipos de datos de entrada que se pueden usar.



Tipos de problemas en donde se pueden usar las redes neuronales

En el teorema de Cybenko^[1], se demuestra que una red neuronal es un aproximador universal, por que se puede usar para problemas de:

- ▶ Regresión
- ▶ Clasificación
- ▶ Reducción dimensional
- ▶ Agrupación

[1]Cybenko, G. 1989. Approximation by superpositions of a sigmoidal function Mathematics of Control, Signals, and Systems, 2(4), 303–314.

Tipos de problemas en donde se pueden usar las redes neuronales

En el teorema de Cybenko^[1], se demuestra que una red neuronal es un aproximador universal, por que se puede usar para problemas de:

- ▶ Regresión 
- ▶ Clasificación 
- ▶ Reducción dimensional 
- ▶ Agrupación 

^[1]Cybenko, G. 1989. Approximation by superpositions of a sigmoidal function Mathematics of Control, Signals, and Systems, 2(4), 303–314.

¿Qué tipos de problemas deben resolver con las redes neuronales?

Problemas que son difíciles para los humanos pero sencillos para las computadoras

Problemas que son fáciles de resolver para humanos y difíciles para las computadoras

¿Qué tipos de problemas deben resolver con las redes neuronales?

Problemas que son
difíciles para los
humanos pero
sencillos para las
computadoras

Problemas que son
fáciles de resolver
para humanos y
difíciles para las
computadoras

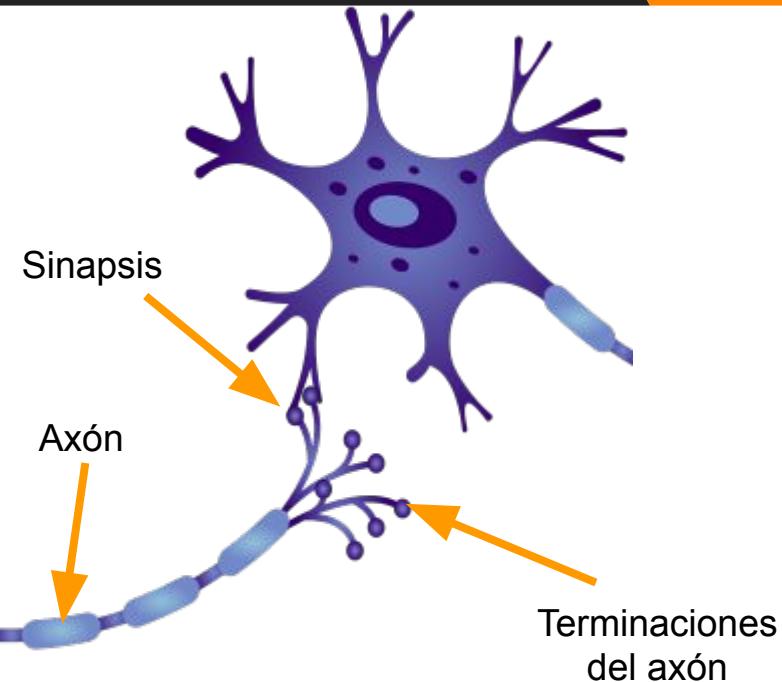
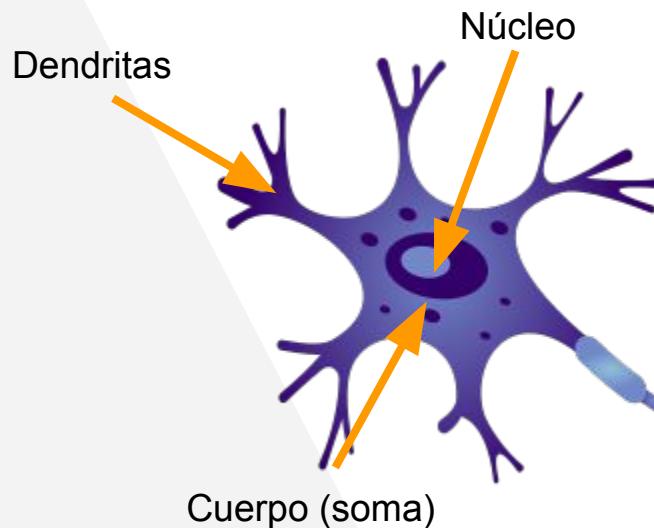
¿Qué significa esto para ustedes?

Simple

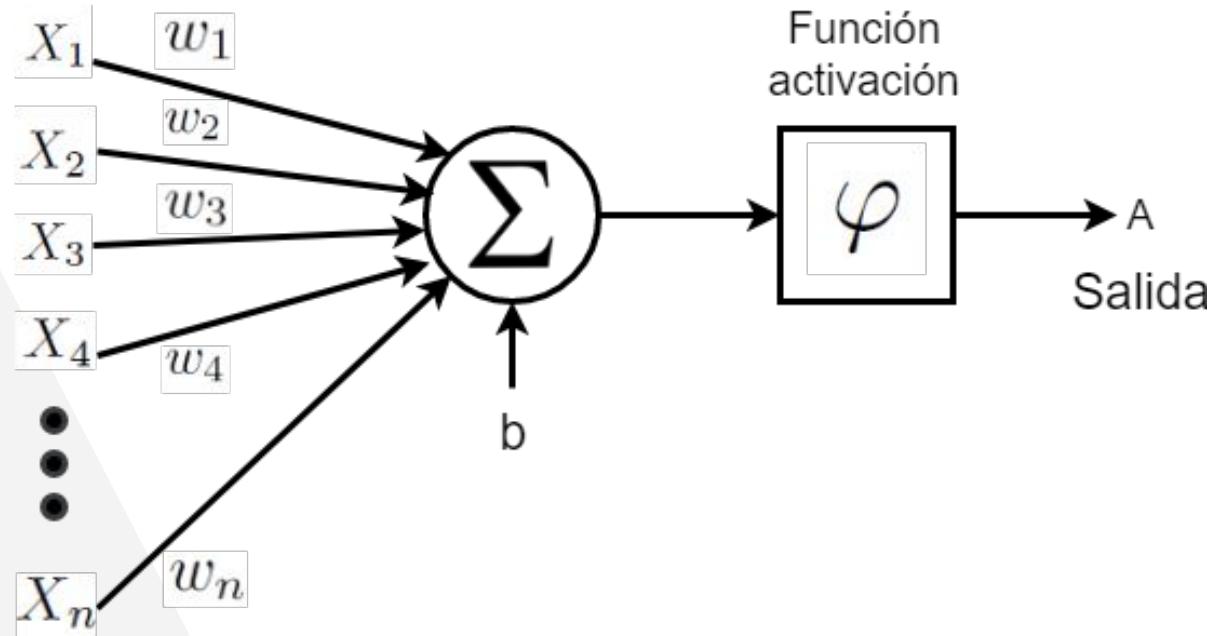
- ▶ Para poder realizar aplicaciones exitosas de redes neuronales es necesario que ustedes comprendan el problema a resolver y puedan resolverlo a mano
- ▶ Es altamente improbable que una red neuronal entrenada por ustedes resuelva un problema que ustedes no pueden resolver

Entendamos las redes neuronales

Neurona artificial



¿Cómo funciona una neurona artificial?



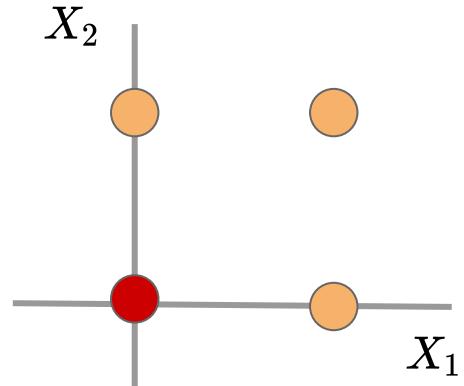
$$A = \varphi(X_1w_1 + X_2w_2 + X_3w_3 + X_4w_4 + \dots + X_nw_n + b)$$

Red neuronal de una sola neurona

Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T
0	0	0
0	1	1
1	0	1
1	1	1

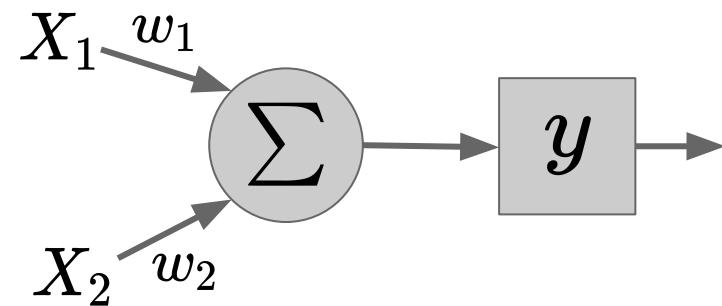
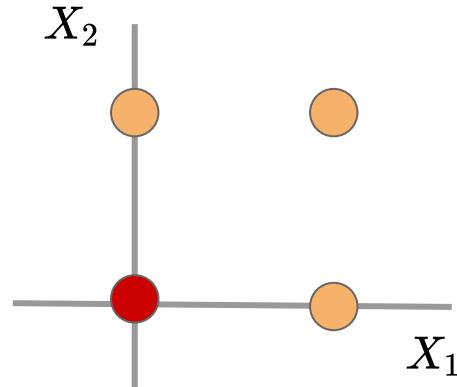


Red neuronal de una sola neurona

Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T
0	0	0
0	1	1
1	0	1
1	1	1



$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i}$$

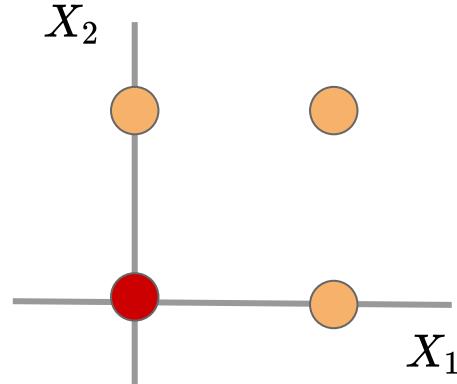
$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases}$$

Red neuronal de una sola neurona

Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T
0	0	0
0	1	1
1	0	1
1	1	1



Describiendo la frontera de decisión

$$0 = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i}$$

$$x_2 = -\frac{w_1}{w_2}x_1$$

$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i}$$

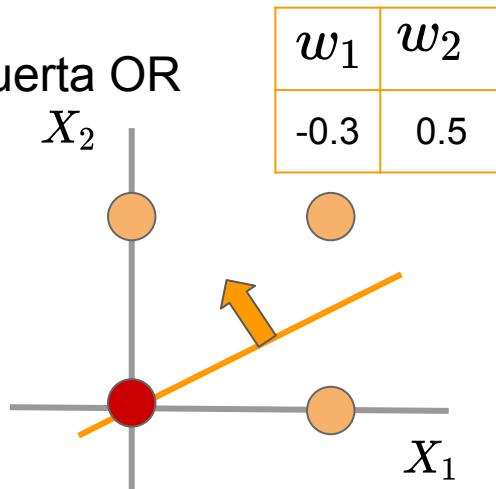
$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases}$$

Red neuronal de una sola neurona

Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T
0	0	0
0	1	1
1	0	1
1	1	1



$$0 = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i}$$

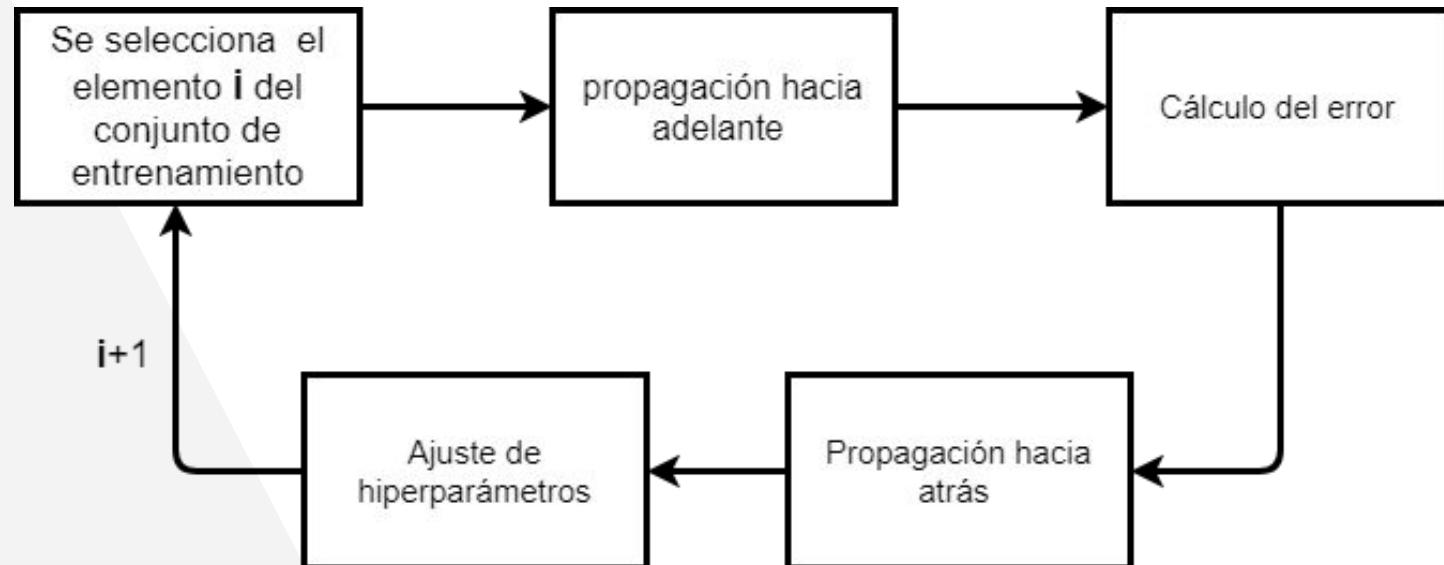
$$x_2 = -\frac{w_1}{w_2} x_1$$

$$x_2 = 0.6x_1$$

$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i}$$

$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases}$$

Entrenando a una sola neurona

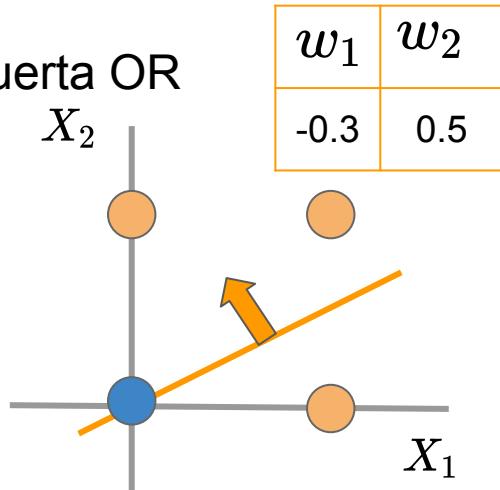


Entrenando a una sola neurona: Seleccionando el elemento $i=1$

Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T	y
0	0	0	
0	1	1	
1	0	1	
1	1	1	



$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i}$$

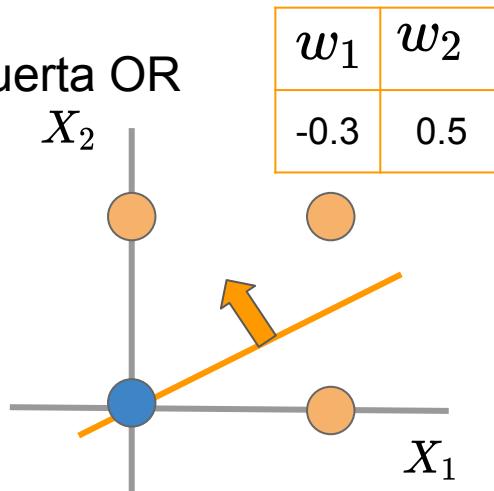
$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases}$$

Entrenando a una sola neurona: Propagación para adelante

Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T	y
0	0	0	0
0	1	1	
1	0	1	
1	1	1	



$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i} = 0$$

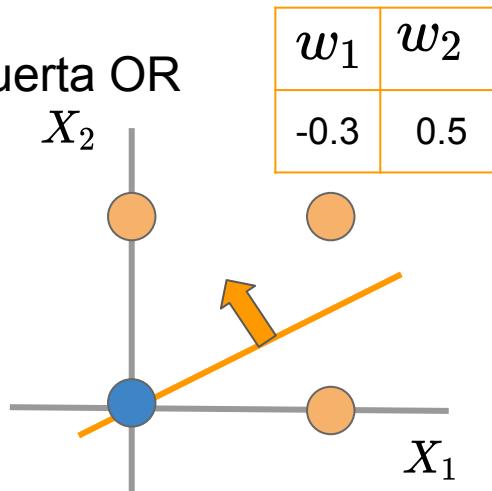
$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases} = 0$$

Entrenando a una sola neurona: Cálculo del error

Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T	y	e
0	0	0	0	0
0	1	1		
1	0	1		
1	1	1		



Cálculo del error

$$e = T - y$$

$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i} = 0$$

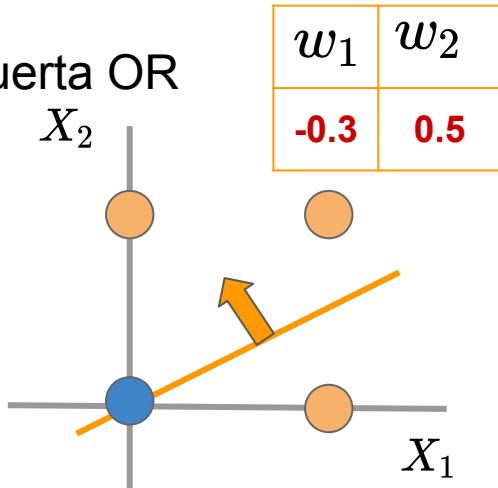
$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases} = 0$$

Entrenando a una sola neurona: Propagación hacia atrás y cálculo del error

Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T	y	e
0	0	0	0	0
0	1	1		
1	0	1		
1	1	1		



$$w_{j_{new}} = w_{j_{old}} + \Delta w_{j_{old}}$$

$$\Delta w_j = \eta(e)x_j \quad \eta = 0.5$$

$$j = 1$$

$$w_1 = -0.3 + 0.5 * (0) * 0$$

$$j = 2$$

$$w_2 = 0.5 + 0.5 * (0) * 0$$

$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i} = 0$$

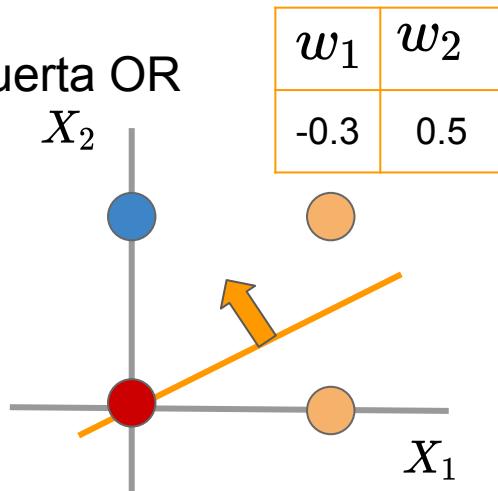
$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases} = 0$$

Entrenando a una sola neurona: Seleccionando el elemento $i=2$

Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T	y	e
0	0	0	0	0
0	1	1		
1	0	1		
1	1	1		



$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i}$$

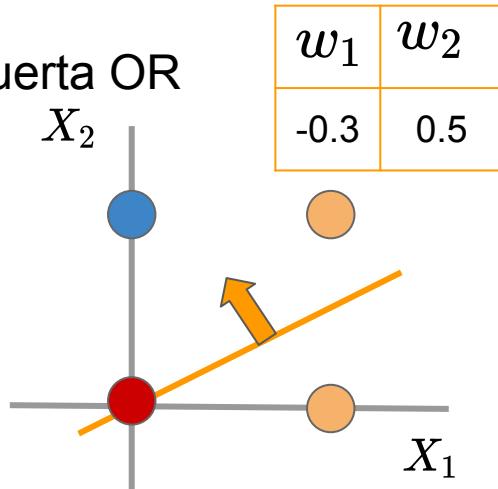
$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases}$$

Entrenando a una sola neurona: Propagación para adelante

Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T	y	e
0	0	0	0	0
0	1	1	1	
1	0	1		
1	1	1		



$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i} = 0.5$$

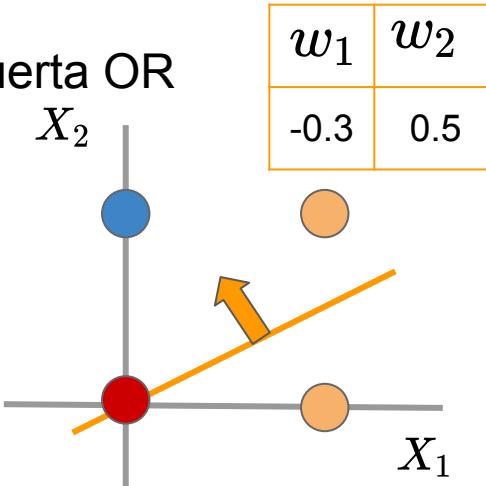
$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases} = 1$$

Entrenando a una sola neurona: Cálculo del error

Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T	y	e
0	0	0	0	0
0	1	1	1	0
1	0	1		
1	1	1		



Cálculo del error

$$e = T - y$$

$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i} = 0.5$$

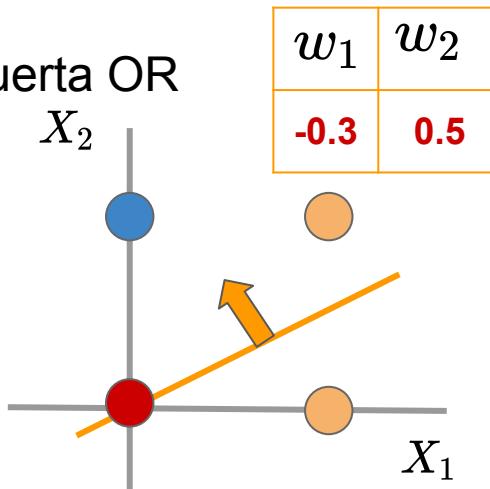
$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases} = 1$$

Entrenando a una sola neurona: Propagación hacia atrás y cálculo del error

Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T	y	e
0	0	0	0	0
0	1	1	1	0
1	0	1		
1	1	1		



$$w_{j_{new}} = w_{j_{old}} + \Delta w_{j_{old}}$$

$$\Delta w_j = \eta(e)x_j \quad \eta = 0.5$$

$$j = 1$$

$$w_1 = -0.3 + 0.5 * (0) * 0$$

$$j = 2$$

$$w_2 = 0.5 + 0.5 * (0) * 0$$

$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i} = 0.5$$

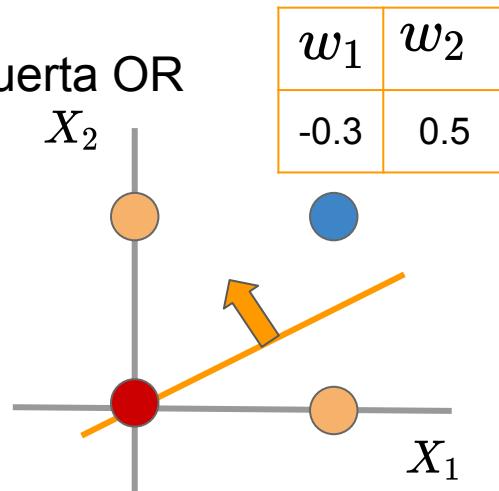
$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases} = 1$$

Entrenando a una sola neurona: Seleccionando el elemento $i=3$

Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T	y	e
0	0	0	0	0
0	1	1	1	0
1	0	1		
1	1	1		



$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i}$$

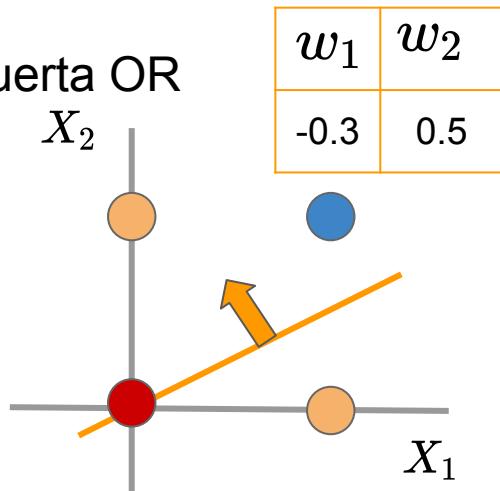
$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases}$$

Entrenando a una sola neurona: Propagación para adelante

Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T	y	e
0	0	0	0	0
0	1	1	1	0
1	0	1	0	
1	1	1		



$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i} = -0.3$$

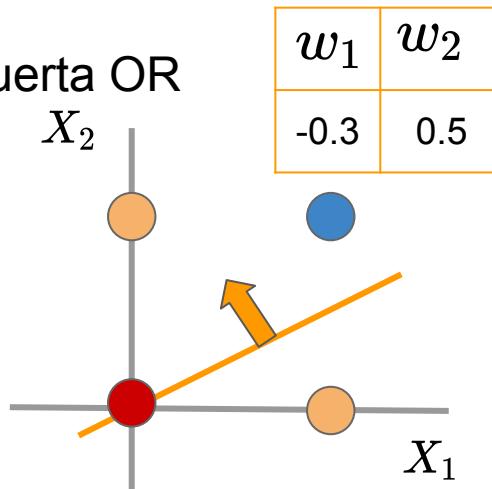
$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases} = 0$$

Entrenando a una sola neurona: Cálculo del error

Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T	y	e
0	0	0	0	0
0	1	1	1	0
1	0	1	0	1
1	1	1		



Cálculo del error

$$e = T - y$$

$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i} = -0.3$$

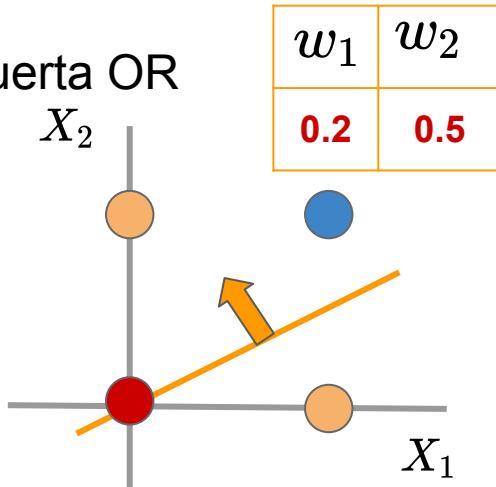
$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases} = 0$$

Entrenando a una sola neurona: Propagación hacia atrás y cálculo del error

Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T	y	e
0	0	0	0	0
0	1	1	1	0
1	0	1	0	1
1	1	1		



w_1	w_2
0.2	0.5

$$w_{j_{new}} = w_{j_{old}} + \Delta w_{j_{old}}$$

$$\Delta w_j = \eta(e)x_j \quad \eta = 0.5$$

$$j = 1$$

$$w_1 = -0.3 + 0.5 * (1) * 1 = 0.2$$

$$j = 2$$

$$w_2 = 0.5 + 0.5 * (1) * 0 = 0.5$$

$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i} = -0.3$$

$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases} = 0$$

Entrenando a una sola neurona: Ajuste de pendiente

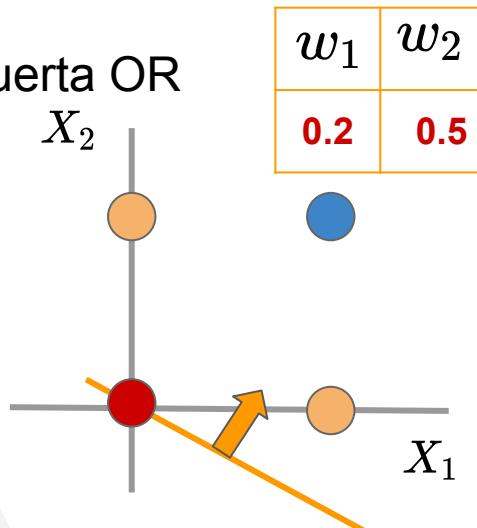
Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T	y	e
0	0	0	0	0
0	1	1	1	0
1	0	1	0	1
1	1	1		

$$x_2 = -\frac{w_1}{w_2}x_1$$

$$x_2 = -0.4x_1$$



w_1	w_2
0.2	0.5

$$w_{j_{new}} = w_{j_{old}} + \Delta w_{j_{old}}$$

$$\Delta w_j = \eta(e)x_j \quad \eta = 0.5$$

$$j = 1$$

$$w_1 = -0.3 + 0.5 * (1) * 1 = 0.2$$

$$j = 2$$

$$w_2 = 0.5 + 0.5 * (1) * 0 = 0.5$$

$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i} = -0.3$$

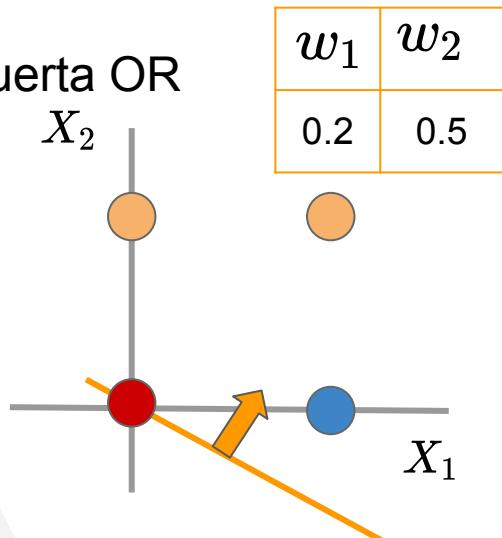
$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases} = 0$$

Entrenando a una sola neurona: Seleccionando el elemento $i=4$

Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T	y	e
0	0	0	0	0
0	1	1	1	0
1	0	1	0	1
1	1	1		



$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i}$$

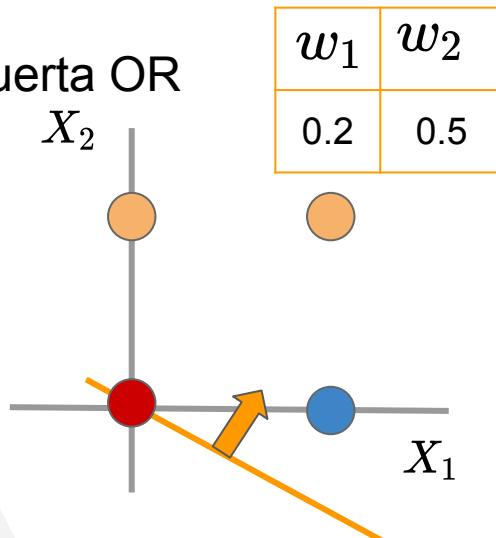
$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases}$$

Entrenando a una sola neurona: Propagación para adelante

Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T	y	e
0	0	0	0	0
0	1	1	1	0
1	0	1	0	1
1	1	1	1	



$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i} = 0.7$$

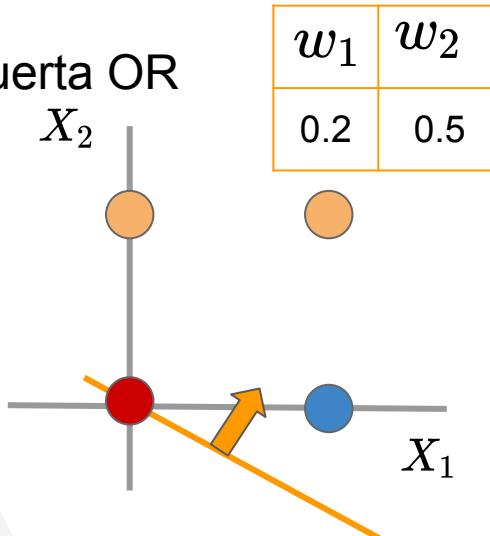
$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases} = 1$$

Entrenando a una sola neurona: Cálculo del error

Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T	y	e
0	0	0	0	0
0	1	1	1	0
1	0	1	0	1
1	1	1	1	0



Cálculo del error

$$e = T - y$$

$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i} = 0.7$$

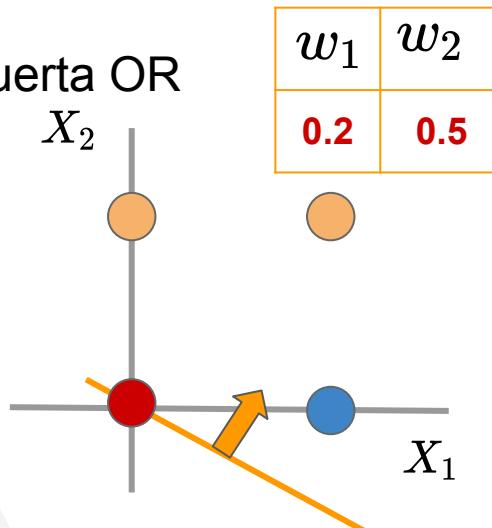
$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases} = 1$$

Entrenando a una sola neurona: Propagación hacia atrás y cálculo del error

Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T	y	e
0	0	0	0	0
0	1	1	1	0
1	0	1	0	1
1	1	1	1	0



$$w_{j_{new}} = w_{j_{old}} + \Delta w_{j_{old}}$$

$$\Delta w_j = \eta(e)x_j \quad \eta = 0.5$$

$$j = 1$$

$$w_1 = 0.2 + 0.5 * (0)1$$

$$j = 2$$

$$w_2 = 0.5 + 0.5 * (0)1$$

$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i} = -0.3$$

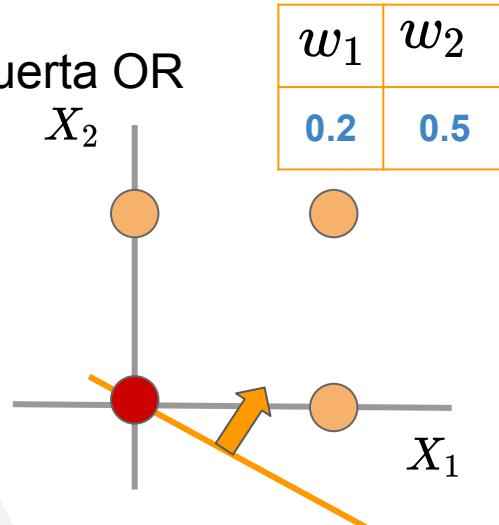
$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases} = 0$$

Probando la red neuronal

Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T	y	e
0	0	0	0	0
0	1	1	1	0
1	0	1	1	0
1	1	1	1	0



$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i}$$

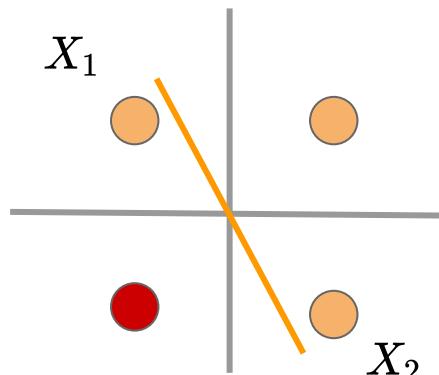
$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases}$$

Probando los límites de una neurona

Ejemplo: Compuerta OR

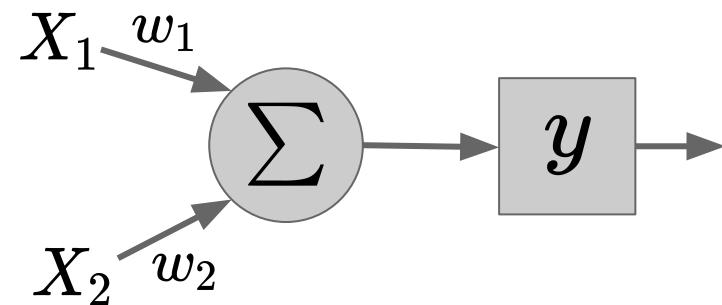
Entradas Salidas

X_1	X_2	T
-1	-1	0
-1	1	1
1	-1	1
1	1	1



$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases}$$

$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i}$$



Describiendo la frontera de decisión

$$0 = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i}$$

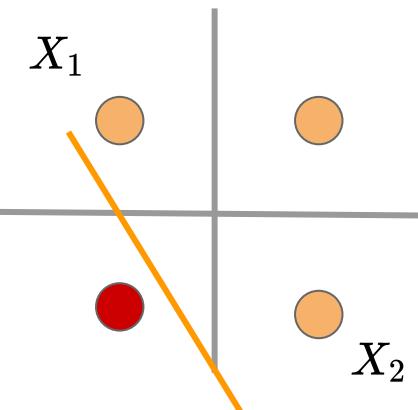
$$x_2 = -\frac{w_1}{w_2} x_1$$

Agregando el bias a la neurona

Ejemplo: Compuerta OR

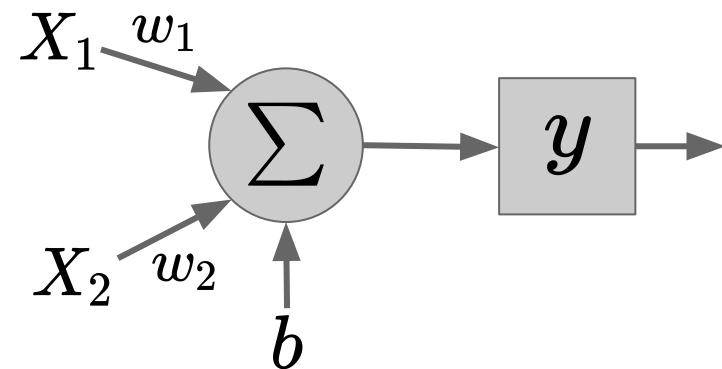
Entradas Salidas

<u>X₁</u>	<u>X₂</u>	<u>T</u>
-1	-1	0
-1	1	1
1	-1	1
1	1	1



$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases}$$

$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i} + b$$



Describiendo la frontera de decisión

$$0 = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i} + b$$

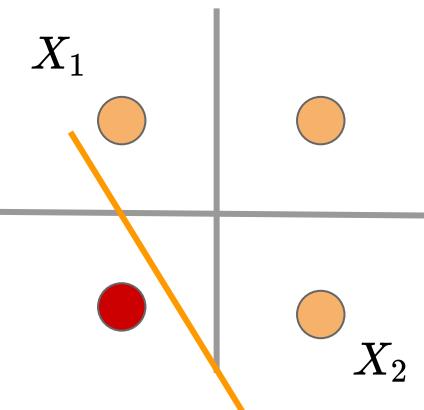
$$x_2 = -\frac{w_1}{w_2}x_1 - b$$

Reglas de entrenamiento con bias

Ejemplo: Compuerta OR

Entradas Salidas

X_1	X_2	T
-1	-1	0
-1	1	1
1	-1	1
1	1	1



$$w_{j_{new}} = w_{j_{old}} + \Delta w_{j_{old}}$$

$$\Delta w_j = \eta(e)x_j$$

$$b_{new} = b_{old} + \eta(e)$$

$$y = \begin{cases} +1 & if \quad f(x_i) > 0 \\ 0 & if \quad f(x_i) \leq 0 \end{cases}$$

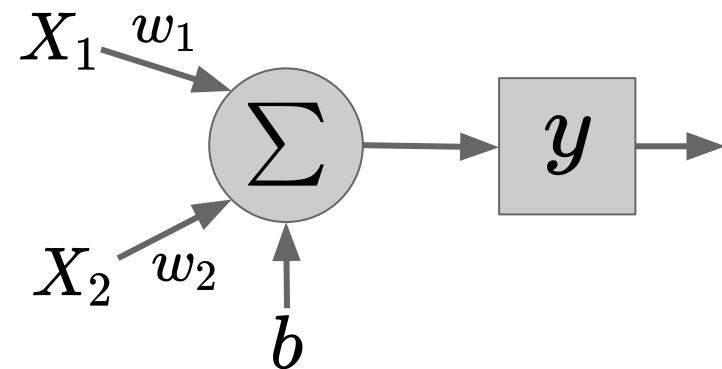
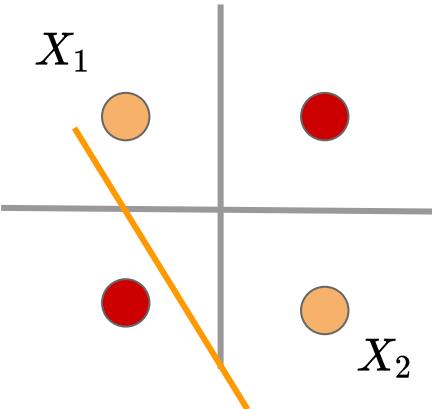
$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i} + b$$

¿Una neurona es suficiente?

Ejemplo: Compuerta XOR

Entradas Salidas

X_1	X_2	T
-1	-1	0
-1	1	1
1	-1	1
1	1	0



$$y = \begin{cases} +1 & \text{if } f(x_i) > 0 \\ 0 & \text{if } f(x_i) \leq 0 \end{cases}$$

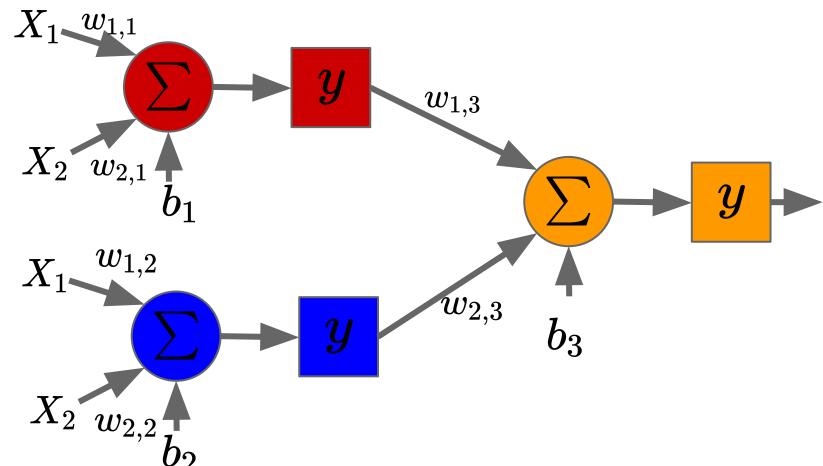
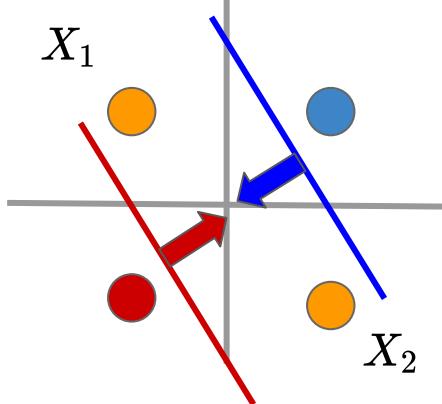
$$f(x_i) = x_{1,i} * w_{1,i} + x_{2,i} * w_{2,i} + b$$

Dividiendo el problema

Ejemplo: Compuerta XOR

Entradas Salidas

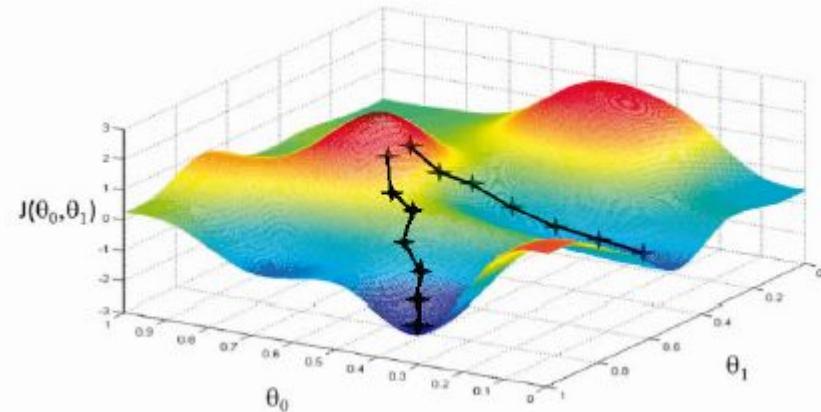
X_1	X_2	T
-1	-1	0
-1	1	1
1	-1	1
1	1	0



Con más neuronas se pueden resolver problemas más complejos

Optimización determinística: gradiente descendente

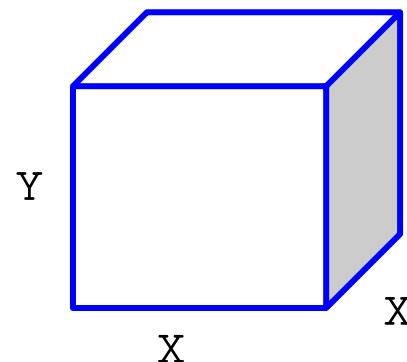
Otro método basado en el gradiente es el gradiente descendiente este consiste en calcular el gradiente en un punto específico inicial y de ese punto calcular el gradiente con el cual se determinará la dirección en la cual se continuará la búsqueda.



Ejemplo: Grad-des

Minimizar la superficie

Una caja rectangular con base cuadrada y sin tapa debe albergar un volumen de 216 cm^3 cuál es la configuración de la caja que ocupa la menor área de material



Ejemplo: Grad-des Minimizar la superficie

l

pro

Objetivo

$$\text{Min } A(xy) = 4xy + x^2$$

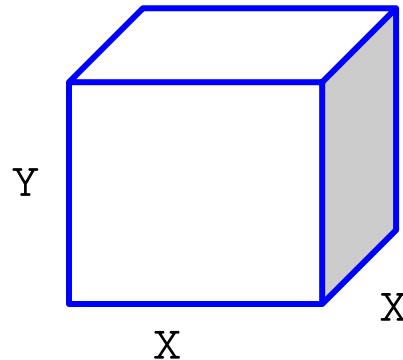
Modelo

$$x^2y = 216$$

Restricciones

$$0 < x < 50$$

$$0 < y < 50$$



Ejemplo: Grad-des

Minimizar la superficie (planteamiento del problema)

Objetivo

$$\text{Min } A(xy) = 4xy + x^2$$

Modelo

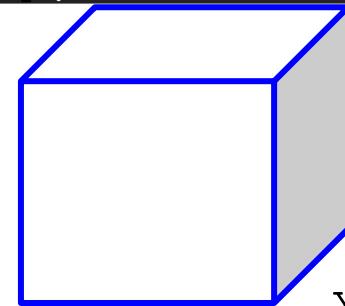
$$x^2y = 216$$

Restricciones

$$0 < x < 50$$

$$0 < y < 50$$

$$y = \frac{216}{x^2}$$



X
El paso uno es proponer un punto inicial donde comenzar la búsqueda para esto primero haremos una sustitución con la cual buscamos garantizar que se respeten las constantes

$$A(x) = \frac{864}{x} + x^2$$

Ejemplo: Grad-des

Minimizar la superficie

que se pro

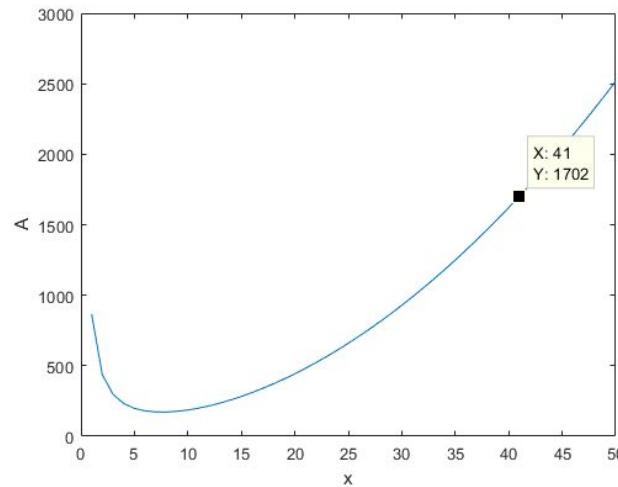
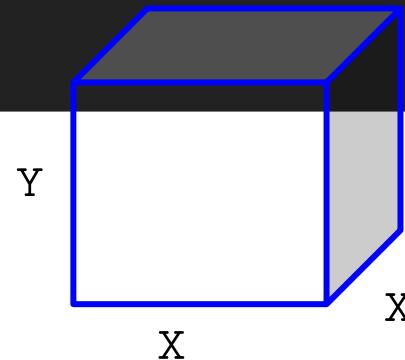
Objetivo

$$\text{Min } A(x) = \frac{864}{x} + x^2$$

Restricciones

$$0 < x < 50$$

Se selecciona un punto inicial de donde se empezará la búsqueda



Ejemplo: Grad-des

Minimización de costos

Objetivo

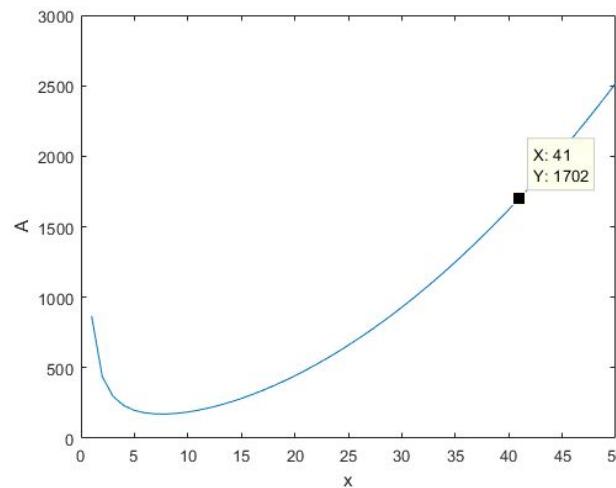
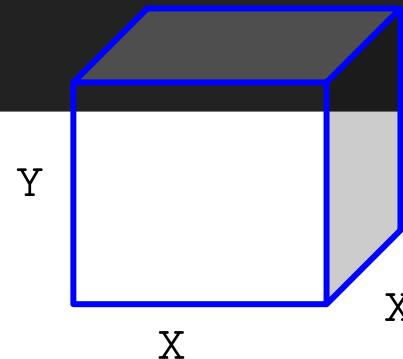
$$\text{Min } A(x) = \frac{864}{x} + x^2$$

Restricciones

$$0 < x < 50$$

Se calcula el gradiente de la función objetivo con respecto a la variable de diseño

$$A(x) = \frac{864}{x^2} + 2x$$



Ejemplo: Grad-des

Minimizar la función:

Objetivo

$$\text{Min } A(x) = \frac{864}{x} + x^2$$

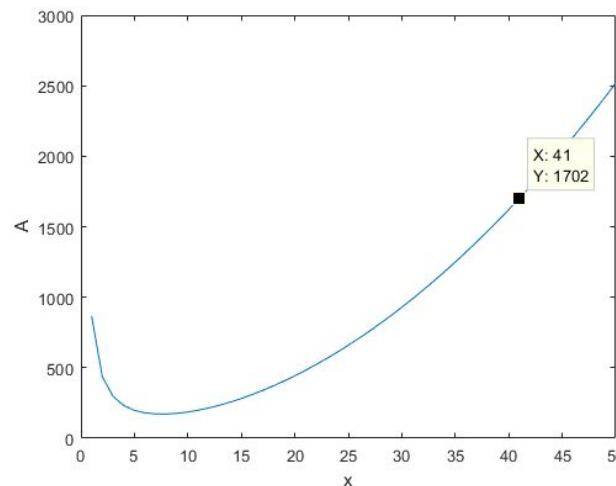
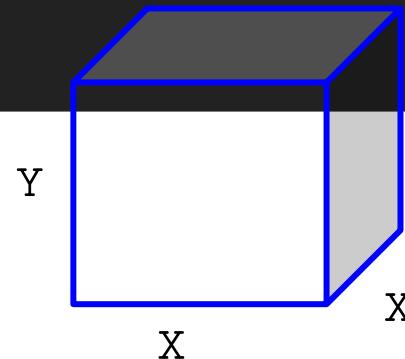
Restricciones

$$0 < x < 50$$

Se evalúa la derivada en el punto inicial seleccionado

$$\dot{A}(x) = \frac{864}{x^2} + 2x$$

$$\dot{A}(41) = \frac{864}{41^2} + 2 * 41 = 82.5140$$



Ejemplo: Grad-des

Minimizar la función:

Objetivo

$$\text{Min } A(x) = \frac{864}{x} + x^2$$

Restricciones

$$0 < x < 50$$

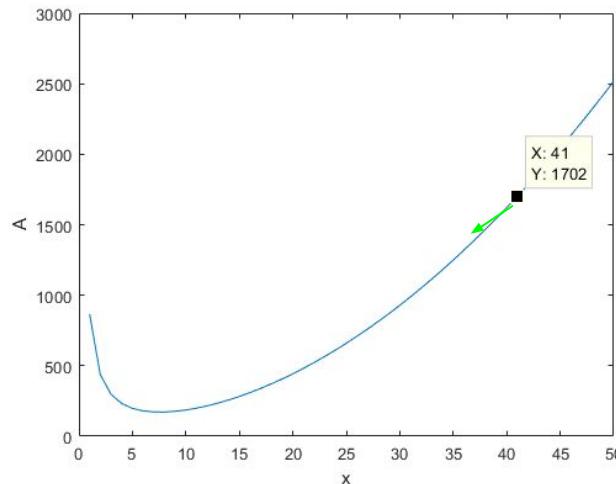
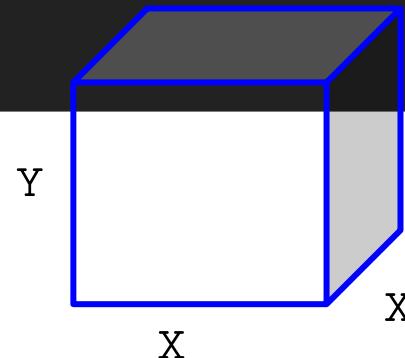
Se calcula un nuevo valor por medio del gradiente

$$\dot{A}(x) = \frac{864}{x^2} + 2x$$

$$A(41) = \frac{864}{41^2} + 2 * 41 = 82.5140$$

$$x_{new} = x_{old} - \mu \Delta f(x)$$

$$x_{new} = 41 - 0.05 * 82.5140 = 36.873$$



Ejemplo: Grad-des

Minimizar la función:

Objetivo

$$\text{Min } A(x) = \frac{864}{x} + x^2$$

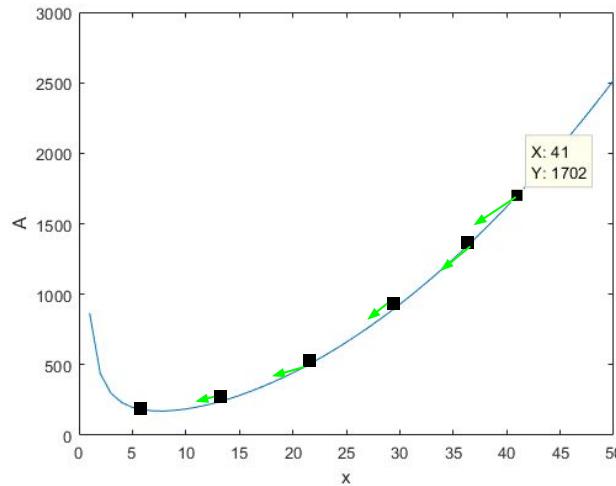
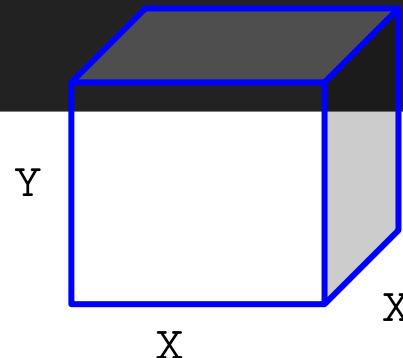
Restricciones

$$0 < x < 50$$

El proceso se repite cuantas veces sea necesario

$$\dot{A}(x) = \frac{864}{x^2} + 2x$$

$$x_{new} = x_{old} - \mu \Delta f(x)$$



Algoritmo del gradiente descendiente

Dado un problema de optimización

Objetivo

$$\text{Min } F_1(X)$$

Modelo

$$g_j(X) = 0 \quad j = 1, 2, 3, \dots, J$$

$$h_k(X) = 0 \quad k = 1, 2, 3, \dots, K$$

Restricciones

$$x_i^L \leq x_i \leq x_i^U = 0 \quad i = 1, 2, 3, \dots, N$$

Para cada variable x_i

Asignar condición inicial $x_i = k$

Por cada iteración

Para cada variable x_i en X

Calcular el gradiente del objetivo en función de la variable x_i

$$\Delta x_i = \Delta f(x_i)$$

Ajustar el valor de x_i

$$x_i = x_i - \mu \Delta f(x_i)$$

Fin del procedimiento

Repaso de regla de la cadena

$$f(x) = (3x - 2x^2)^3$$

$$\frac{df(x)}{dx} = \frac{df(x)}{du} \frac{du}{dx}$$

$$u = 3x - 2x^2$$

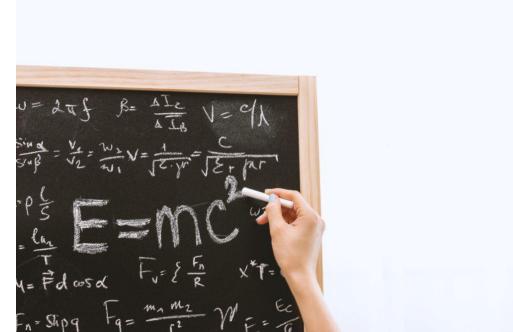
$$\frac{df(x)}{dx} = 3u^2 * 3 - 4x$$

$$f(x) = u^3$$

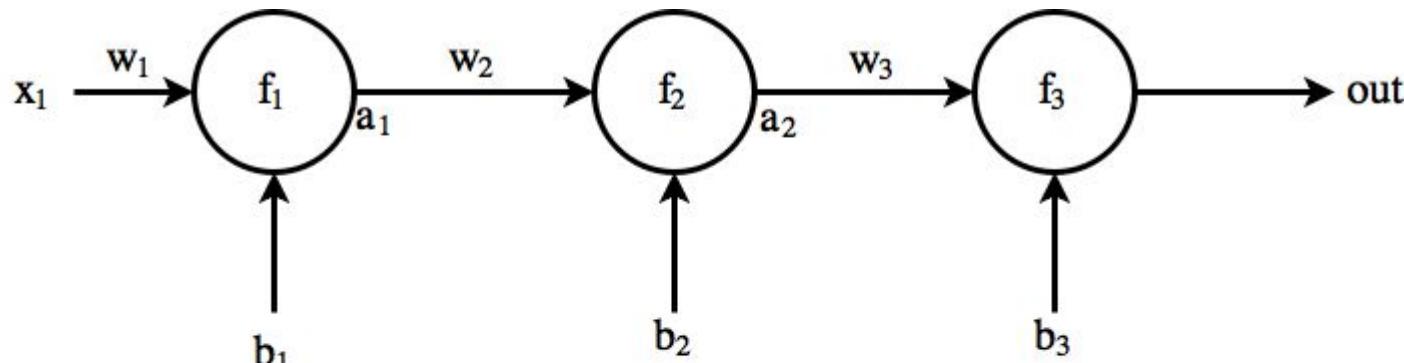
$$\frac{df(x)}{dx} = 3(3x - 2x^2)^2 * 3 - 4x$$

$$\frac{df(x)}{du} = 3u^2$$

$$\frac{du}{dx} = 3 - 4x$$



Entrenando una red más grande



$$a_1 = f_1(w_1 * x_1 + b_1)$$

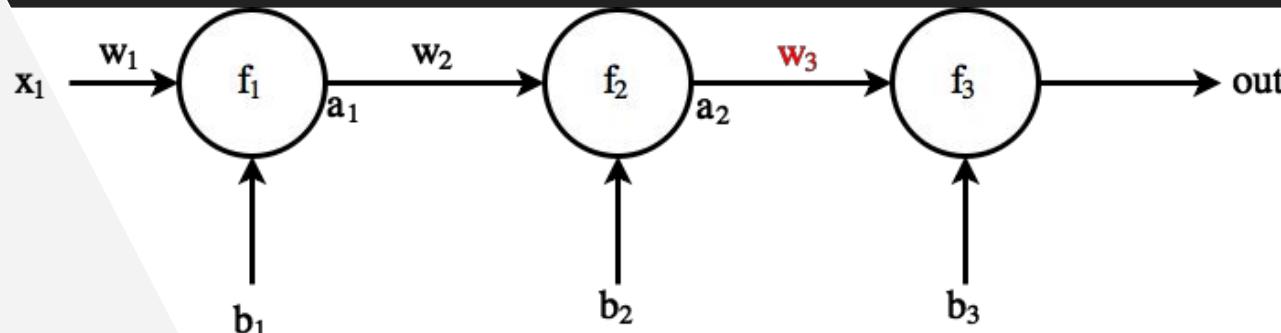
$$a_2 = f_2(w_2 * a_1 + b_2)$$

$$\text{out} = f_3(w_3 * a_2 + b_3)$$

$$\text{out} = f_3(f_2(f_1(w_1 * x + b_1) + b_2) + b_3)$$

$$w_{j_{new}} = w_{j_{old}} + \Delta w_{j_{old}}$$

Propagación hacia atrás (backpropagation)



$$a_1 = f_1(w_1 * x_1 + b_1)$$

$$a_2 = f_2(w_2 * a_1 + b_2)$$

$$out = f_3(w_3 * a_2 + b_3)$$

$$out = f_3(f_2(f_1(w_1 * x + b_1) + b_2) + b_3)$$

$$C = (out - y)^2$$

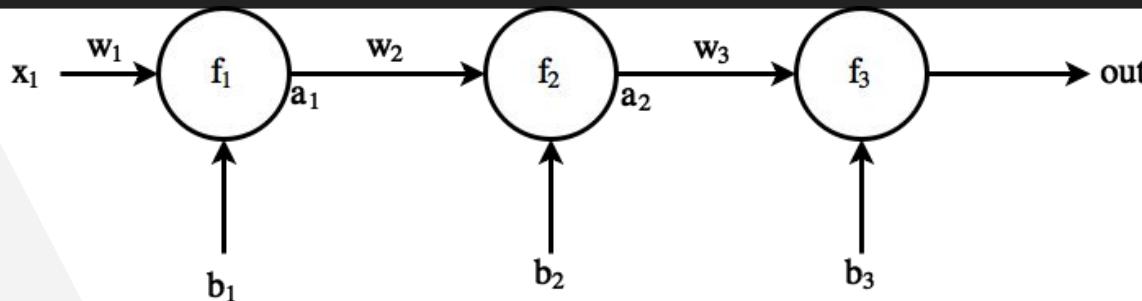
$$C = (out - y)^2$$

$$\frac{d}{dw_3} C = \frac{d}{dout} C * \frac{d}{dn_3} f_3 * \frac{d}{dw_3} n_3$$

$$out = f_3(n_3)$$

$$n_3 = w_3 * a_2 + b_3$$

Propagación hacia atrás (backpropagation)



$$a_1 = f_1(w_1 * x_1 + b_1)$$

$$a_2 = f_2(w_2 * a_1 + b_2)$$

$$out = f_3(w_3 * a_2 + b_3)$$

$$out = f_3(f_2(f_1(w_1 * x + b_1)) + b_2) + b_3$$

$$C = (out - y)^2$$

$$C = (out - y)^2$$

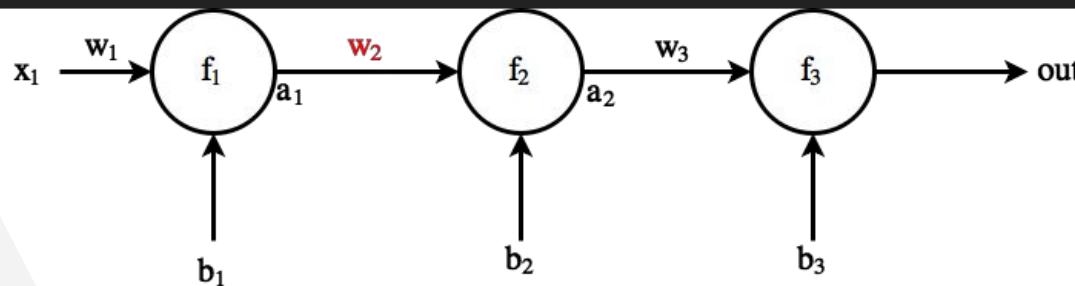
$$\frac{d}{dw_3} C = \frac{d}{dout} C * \frac{d}{dn_3} f_3 * \frac{d}{dw_3} n_3$$

$$out = f_3(n_3)$$

$$n_3 = w_3 * a_2 + b_3$$

$$\Delta w_3 = -\eta \frac{d}{dw_3} C$$

Propagación hacia atrás (backpropagation)



$$a_1 = f_1(w_1 * x_1 + b_1)$$

$$a_2 = f_2(w_2 * a_1 + b_2)$$

$$out = f_3(w_3 * a_2 + b_3)$$

$$out = f_3(f_2(f_1(w_1 * x + b_1)) + b_2) + b_3$$

$$C = (out - y)^2$$

$$C = (out - y)^2$$

$$out = f_3(n_3)$$

$$\frac{d}{dw_2} C = \frac{d}{dout} C * \frac{d}{dn_3} f_3 * \frac{d}{da_2} n_3 * \frac{d}{dn_2} a_2 * \frac{d}{dw_2} n_2$$

$$n_3 = w_3 * a_2 + b_3$$

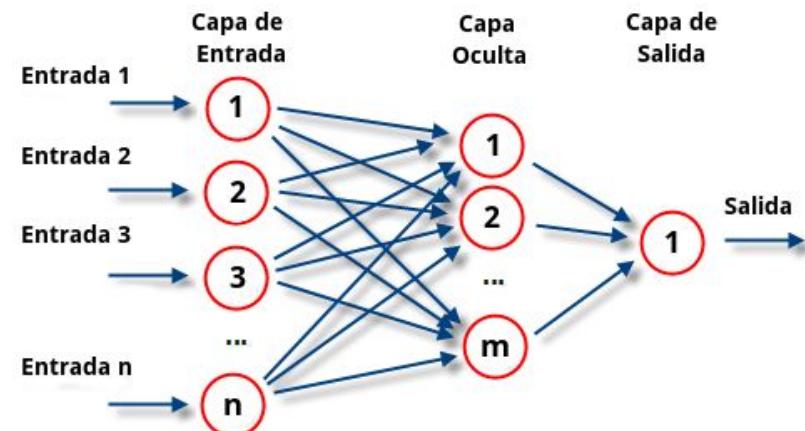
$$a_2 = f_2(n_2)$$

$$n_2 = w_2 * a_1 + b_2$$

Red perceptron multicapa (MLP)

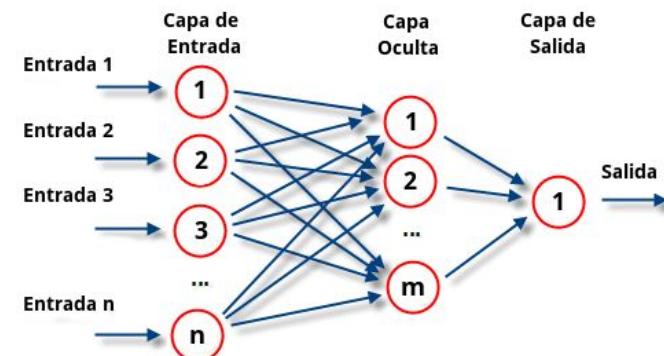
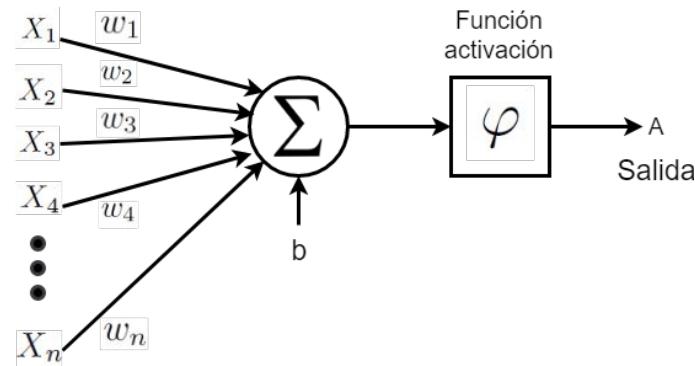
Es un tipo red neuronal

- ▶ Propagación hacia adelante
- ▶ Consiste de un mínimo de 3 capas de nodos
- ▶ Cada nodo consiste de una neurona
- ▶ Se entrena de manera supervisada con Back propagation



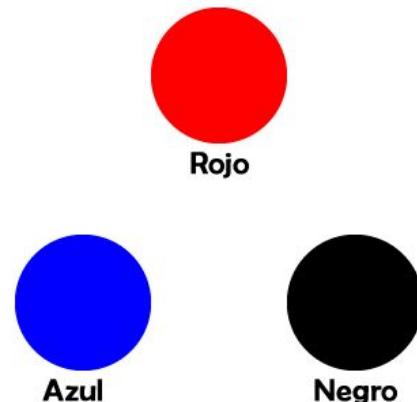
Terminología de las redes

- ▶ Arquitectura
- ▶ Función de activación
- ▶ Capas
- ▶ Aprendizaje
- ▶ Entradas
- ▶ Salidas
- ▶ Completamente conectada
- ▶ Parcialmente conectada
- ▶ Función de costo
- ▶ Optimizador

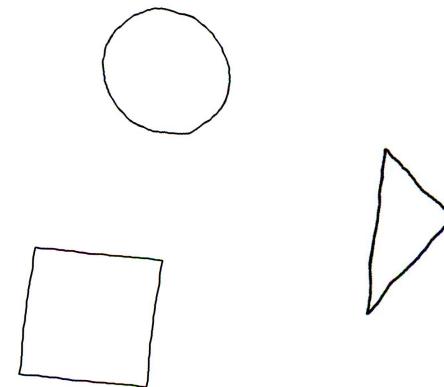


Clasificador MLP: Problema de extracción de características

Clasificador de color



Clasificador de forma



Abrir: Apuntes Color Hu del curso aprendizaje máquina

Aprendizaje profundo con TensorFlow

“

La inteligencia es la habilidad de adaptarse a los cambios

Stephen Hawking

”

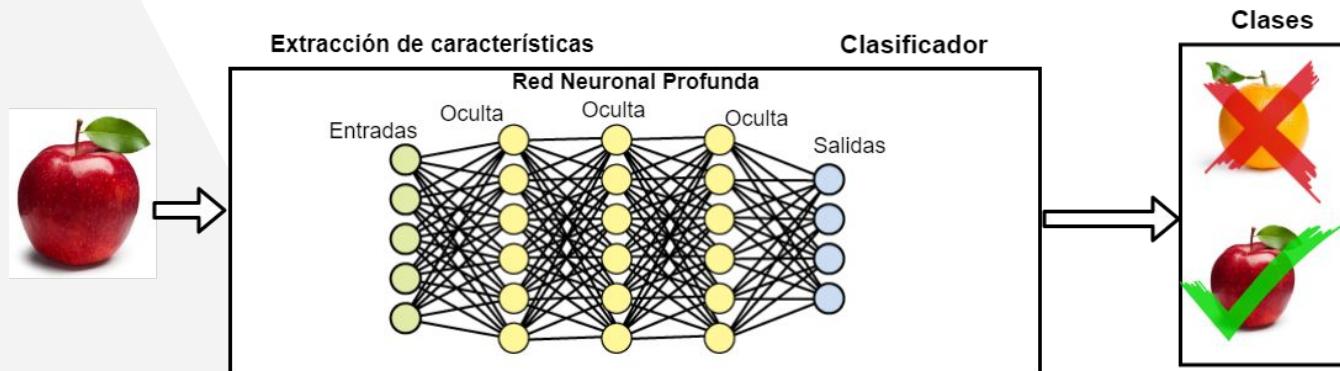
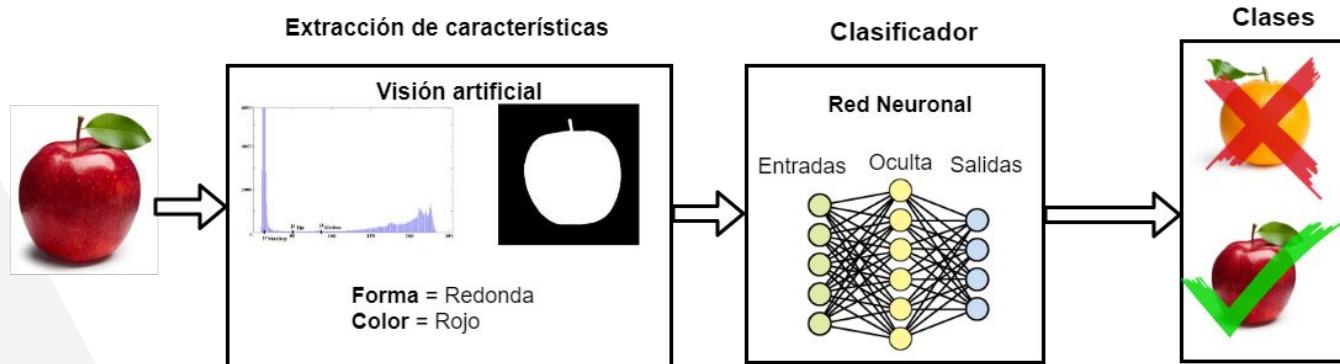
¿Limitantes para aplicar el aprendizaje máquina tradicional?

Para poder realizar aprendizaje máquina tradicional es necesario ser capaz de describir numéricamente toda la información.

- ▶ Imágenes (visión por computadora)
- ▶ Audio (procesamiento de señales)
- ▶ Predicciones económicas (estadísticas)

Problema de extracción de características

¿Qué hacen las Redes Neuronales Profundas (DNN)?



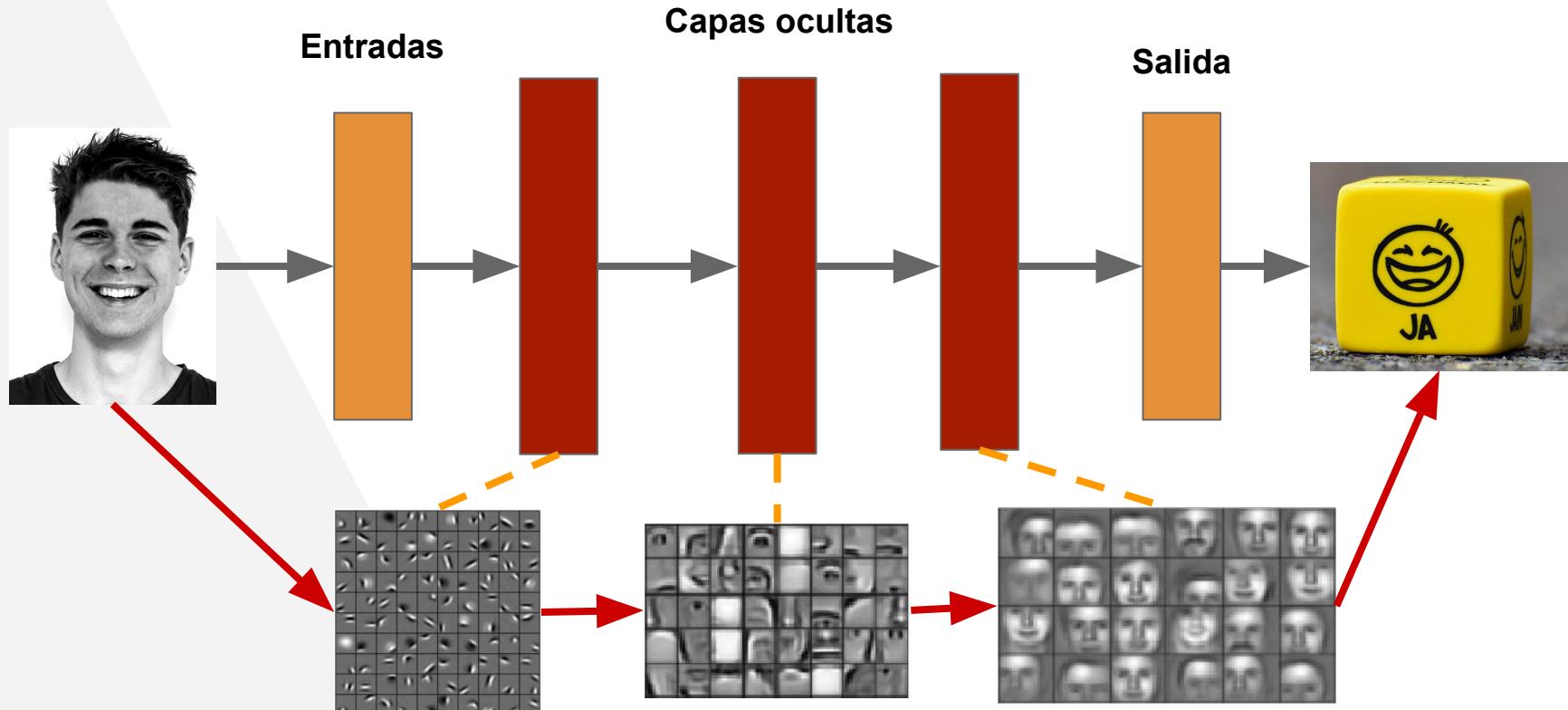
¿Qué problemas puede resolver una DNN?

- ▶ Problemas que puede resolver un humano (**texto a voz**)
- ▶ Problemas que requieren mucha concentración por mucho tiempo de parte del humano para ser resueltos (**Alpha Go**)
- ▶ Problemas que requieren un conocimiento específico para ser resueltos (**traducción de textos**)
- ▶ Problemas cuya resolución requieren de un análisis de un gran volumen de datos (**IBM Watson Health**)
- ▶ Problemas repetitivos (**fabricación a gran escala**)

¿Qué hace a un modelo profundo?

- ▶ **Redes neuronales con una capa oculta no son profundos:** No existe extracción de características
- ▶ **Una RN con dos capas ocultas no es profundo:** No existe una jerarquía de características
- ▶ **Las máquinas de soporte vectorial y los métodos de kernel no son profundos:** No extraen características
- ▶ **Los árboles de clasificación:** No son profundos, todas las decisiones se toman a partir de la entrada

¿Qué hay dentro de una DNN?



¿Por qué usar redes neuronales profundas?

- ▶ **Permiten hacer la extracción de características**
 - ▷ Las características de la red están basadas en los datos que puede visualizar
 - ▷ Puede extraer características complejas que son fáciles de apreciar por un humano pero difíciles de extraer por métodos convencionales
 - ▷ Permite hacer una búsqueda exhaustiva de las características deseadas, lo cual le puede permitir superar a un humano
 - ▷ Las características son adecuadas para resolver el problema propuesto
- ▶ **Permiten resolver problemas que no puede resolver otra metodología**

Profundizando en las redes neuronales

Perceptrón multicapa

- ▶ Requiere datos reducidos a un conjunto de características
- ▶ No requiere mucho poder de cómputo

Perceptrón multicapa profunda

- ▶ Extrae información de datos en crudo (estado natural)
- ▶ Requiere equipos de cómputo especializados

“Se puede obtener mejores resultados torturando los datos”

Profundizando en las redes neuronales

Perceptrón multicapa

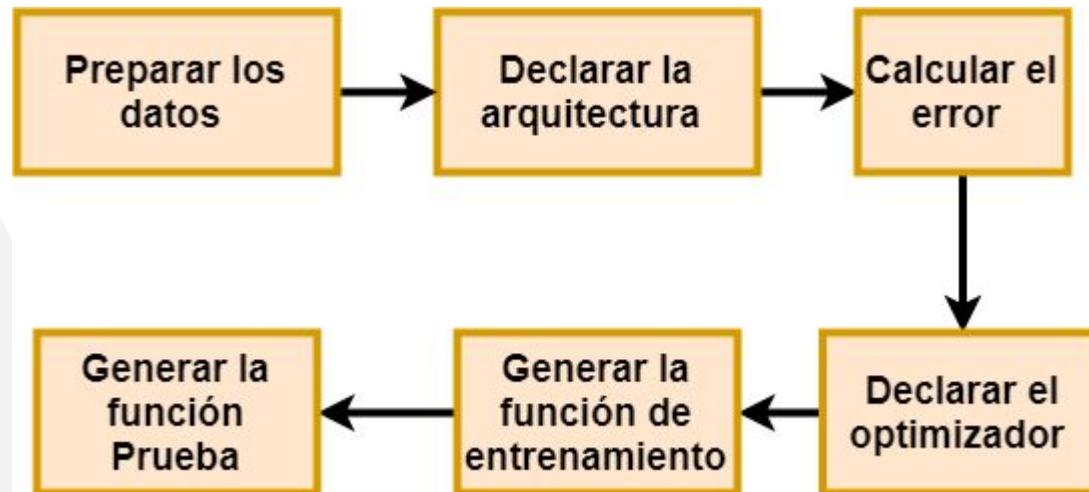
- ▶ Su entrenamiento se realiza por medio de gradiente descendiente
- ▶ Requiere una cantidad moderada de datos
- ▶ Su entrenamiento puede tomar horas

Perceptrón multicapa profunda

- ▶ Se entranan por medio de gradiente estocástico descendiente
- ▶ Requiere grandes cantidades de datos
- ▶ Su entrenamiento puede tomar días

Frameworks

Pasos para implementar una red profunda en TensorFlow

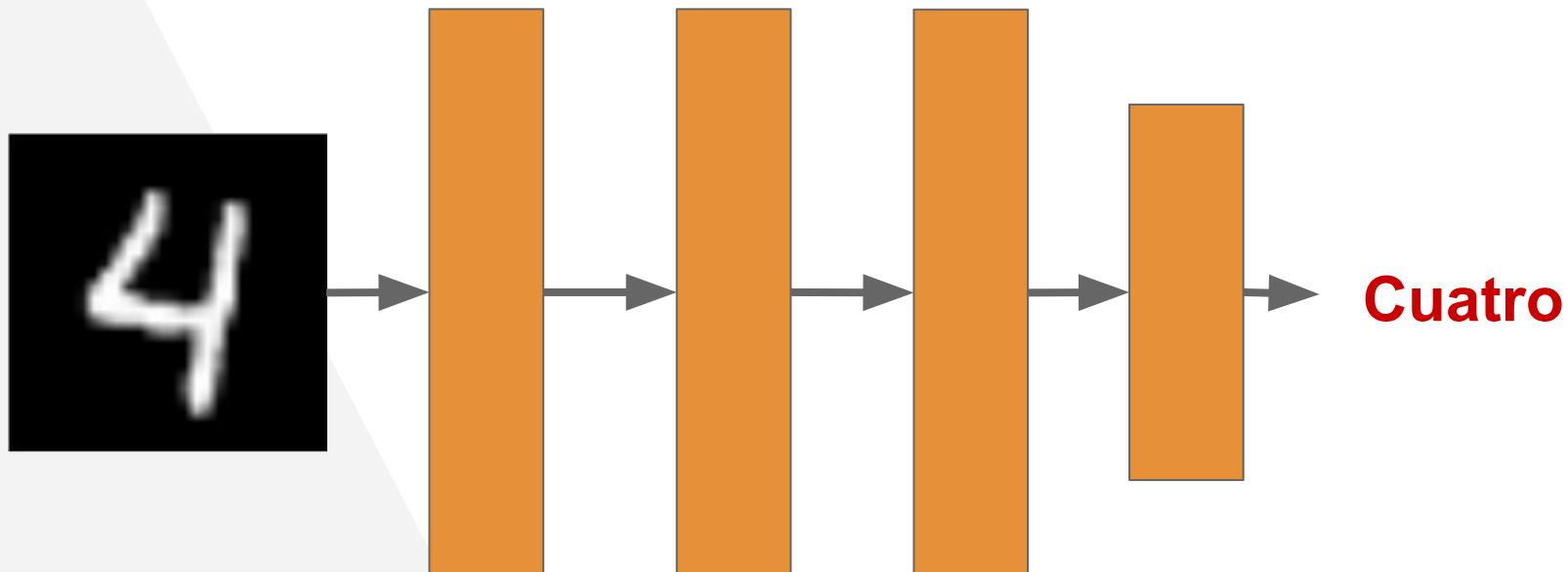


Red perceptrón profunda (DNN, Red Vanilla)

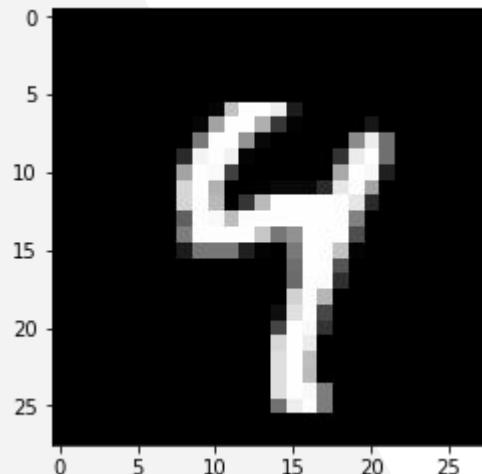
Ejemplo: Reconocimiento de caracteres

Conjunto MNIST

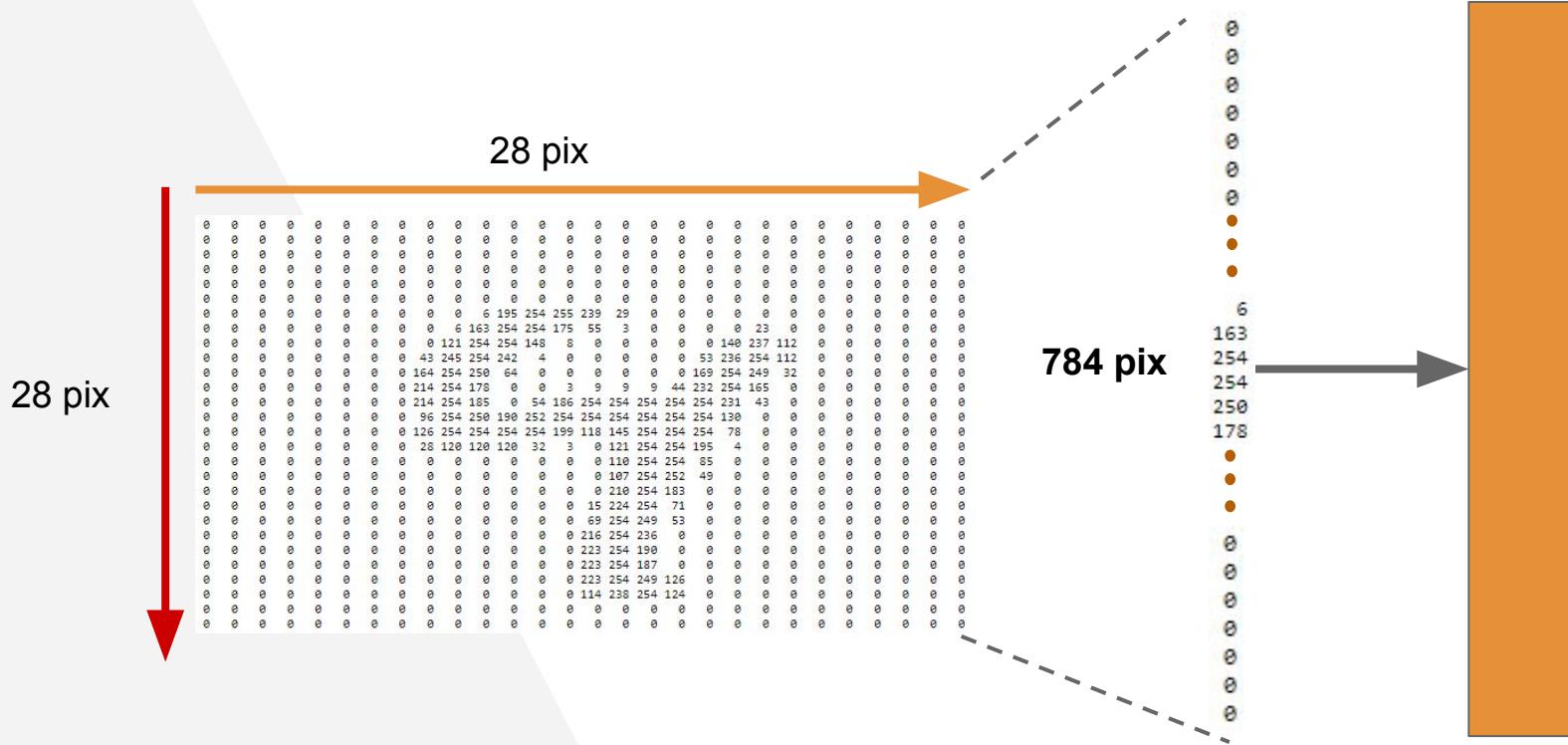
504 / 92



Ejemplo MNIST: Definiendo las entradas al sistema



Ejemplo MNIST: Definiendo las entradas al sistema



Ejemplo MNIST: Definiendo las salidas

Codificaciones:



Valor numérico real: **3**

Codificación binaria:

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

Codificación one hot:

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

Ejemplo MNIST: Definiendo las salidas

Codificaciones:



Valor numérico real: **3**

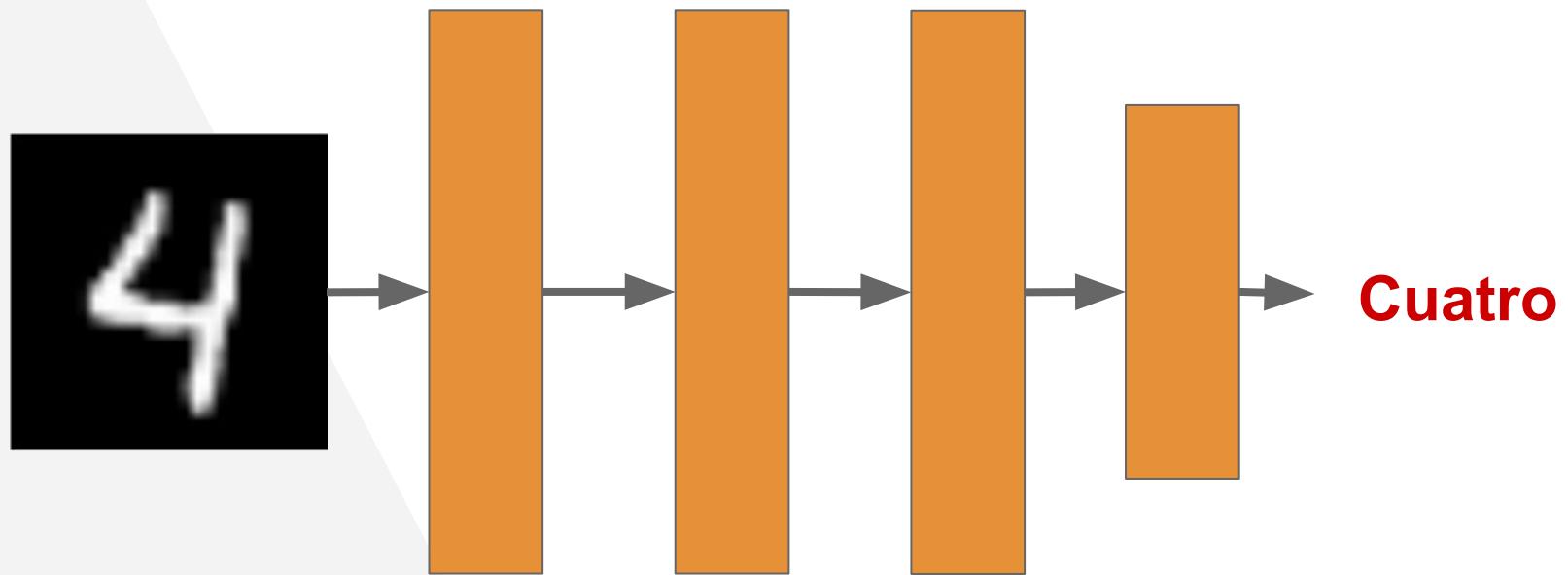
Codificación binaria:

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

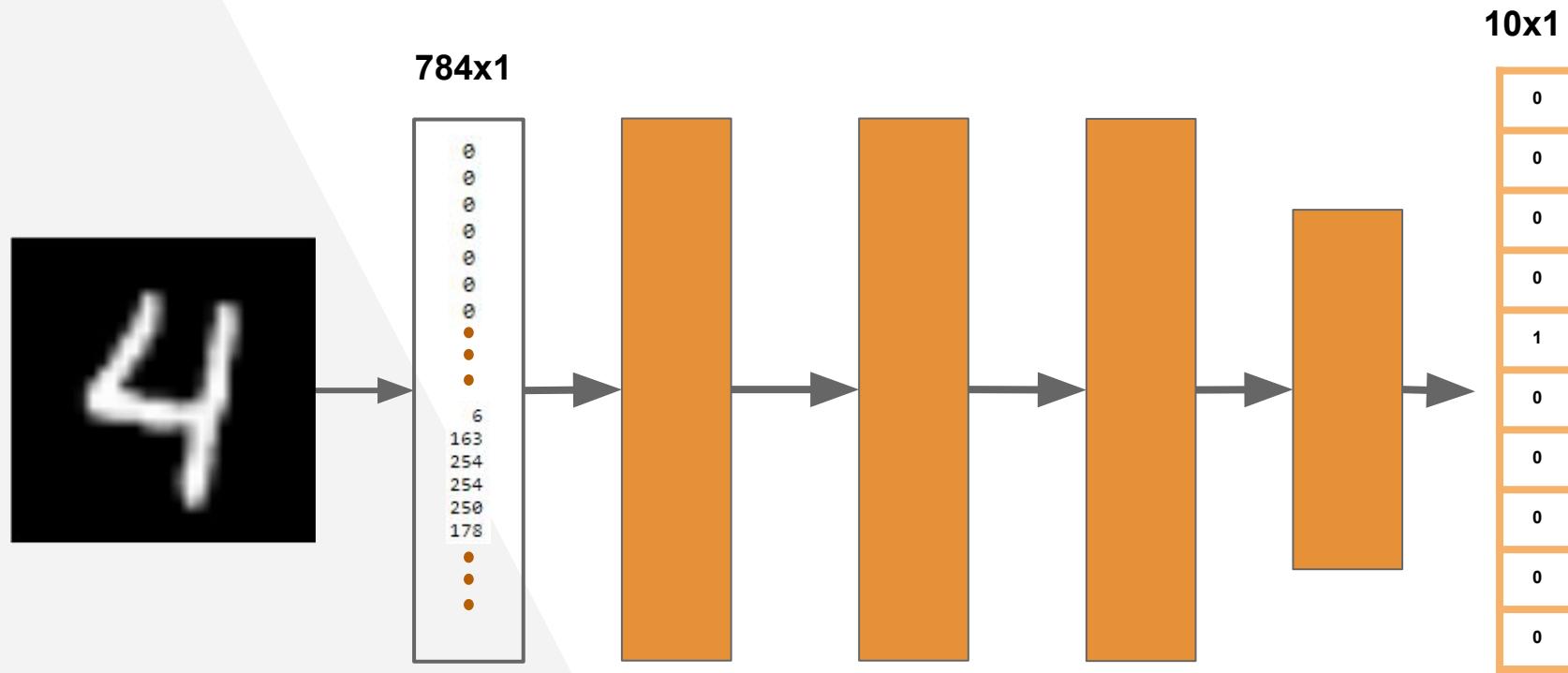
Codificación one hot:

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

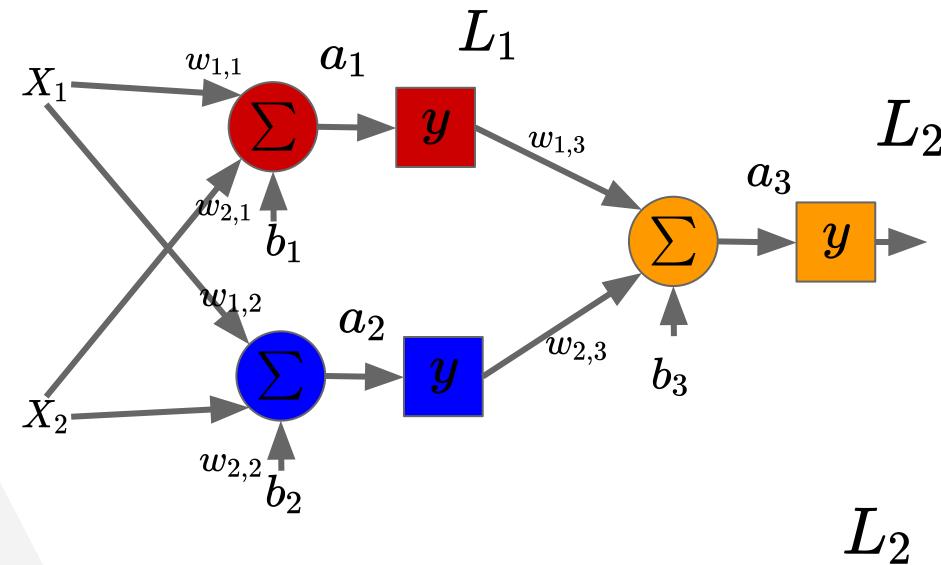
Analizando la arquitectura de una red neuronal perceptrón profunda



Analizando la arquitectura de una red neuronal perceptrón profunda



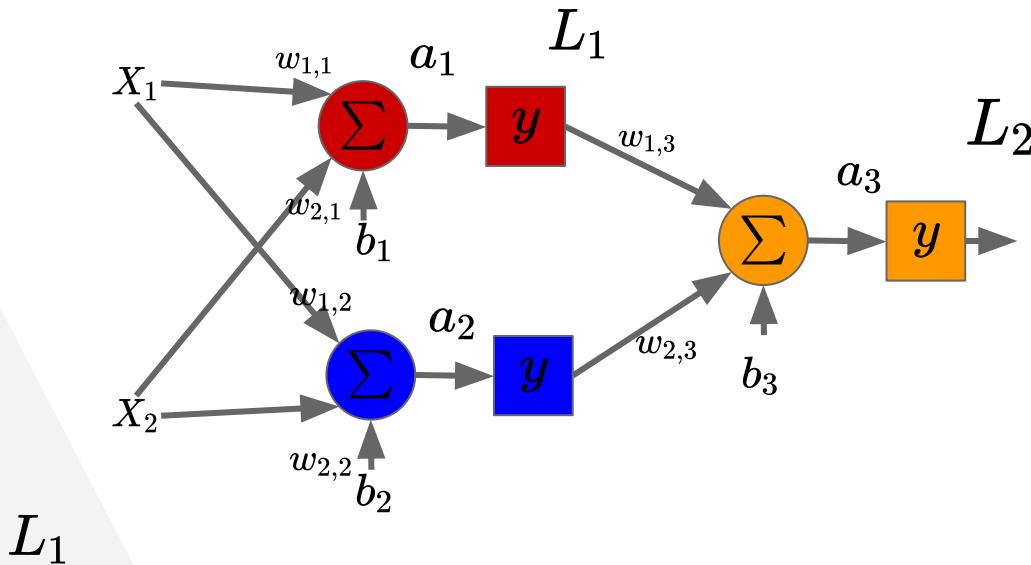
Interpretación matricial de una red neuronal



$$\begin{aligned} a_1 &= X_1 * w_{1,1} + X_2 * w_{2,1} + b_1 \\ a_2 &= X_1 * w_{1,2} + X_2 * w_{2,2} + b_2 \end{aligned}$$

$$a_3 = y(a_1) * w_{1,3} + y(a_2) * w_{2,3} + b_3$$

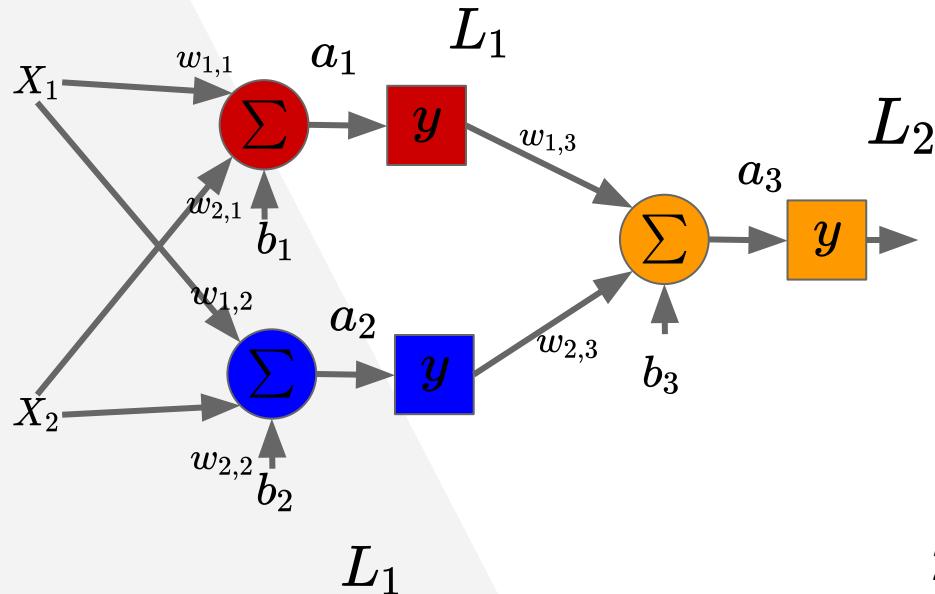
Interpretación matricial de una red neuronal



$$\begin{aligned} a_1 &= X_1 * w_{1,1} + X_2 * w_{2,1} + b_1 \\ a_2 &= X_1 * w_{1,2} + X_2 * w_{2,2} + b_2 \end{aligned}$$

$$\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{2,1} \\ w_{1,2} & w_{2,2} \end{bmatrix} * \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

Interpretación matricial de una red neuronal

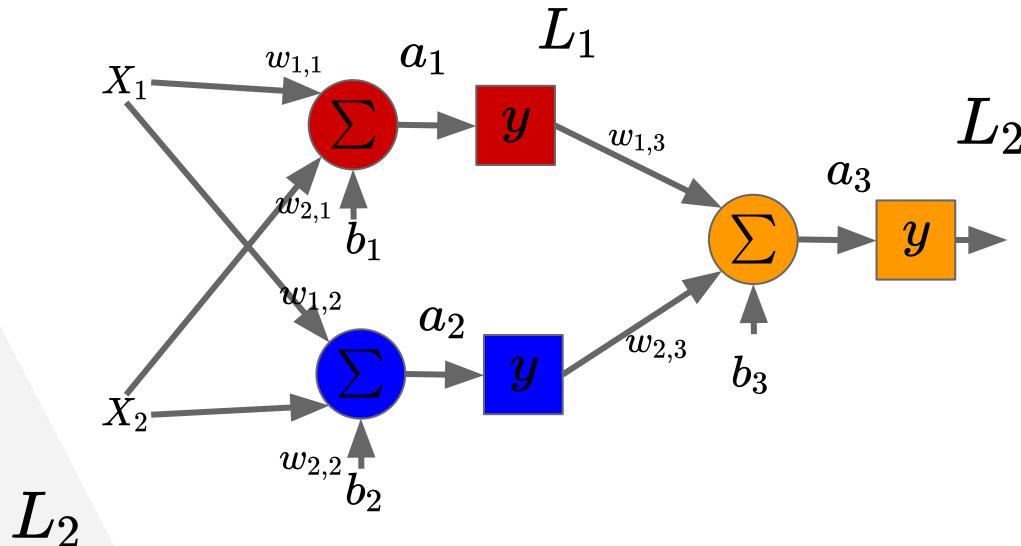


$$\begin{aligned} L_1 \\ a_1 &= X_1 * w_{1,1} + X_2 * w_{2,1} + b_1 \\ a_2 &= X_1 * w_{1,2} + X_2 * w_{2,2} + b_2 \end{aligned}$$

$$A = X * W + B$$

$$\begin{matrix} 2 \times 1 \\ \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \end{matrix} = \begin{bmatrix} w_{1,1} & w_{2,1} \\ w_{1,2} & w_{2,2} \end{bmatrix} * \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

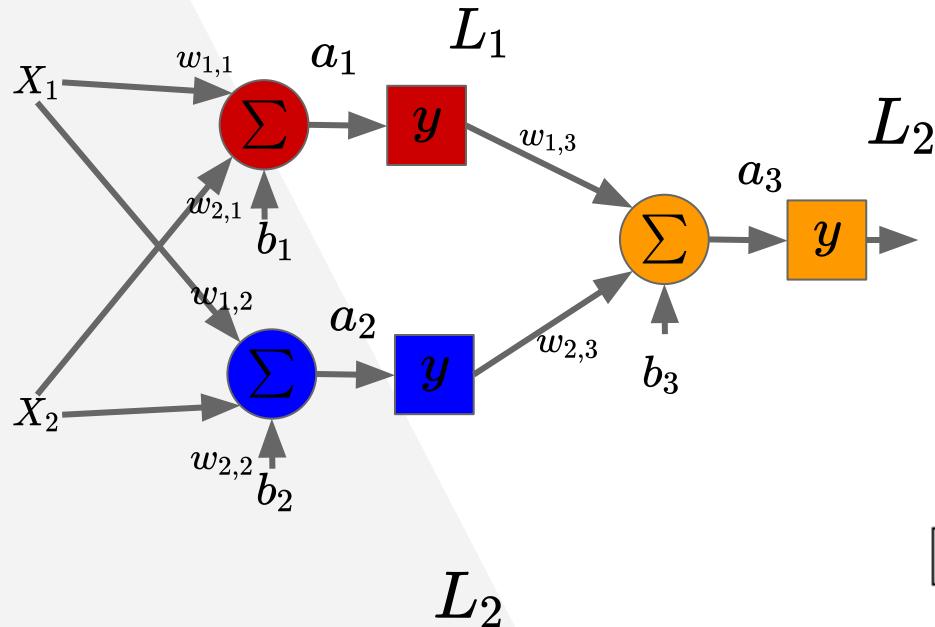
Interpretación matricial de una red neuronal



$$a_3 = y(a_1) * w_{1,3} + y(a_2) * w_{2,3} + b_3$$

$$[a_3] = [w_{1,3} \quad w_{2,3}] * \begin{bmatrix} y(a_1) \\ y(a_2) \end{bmatrix} + [b_3]$$

Interpretación matricial de una red neuronal

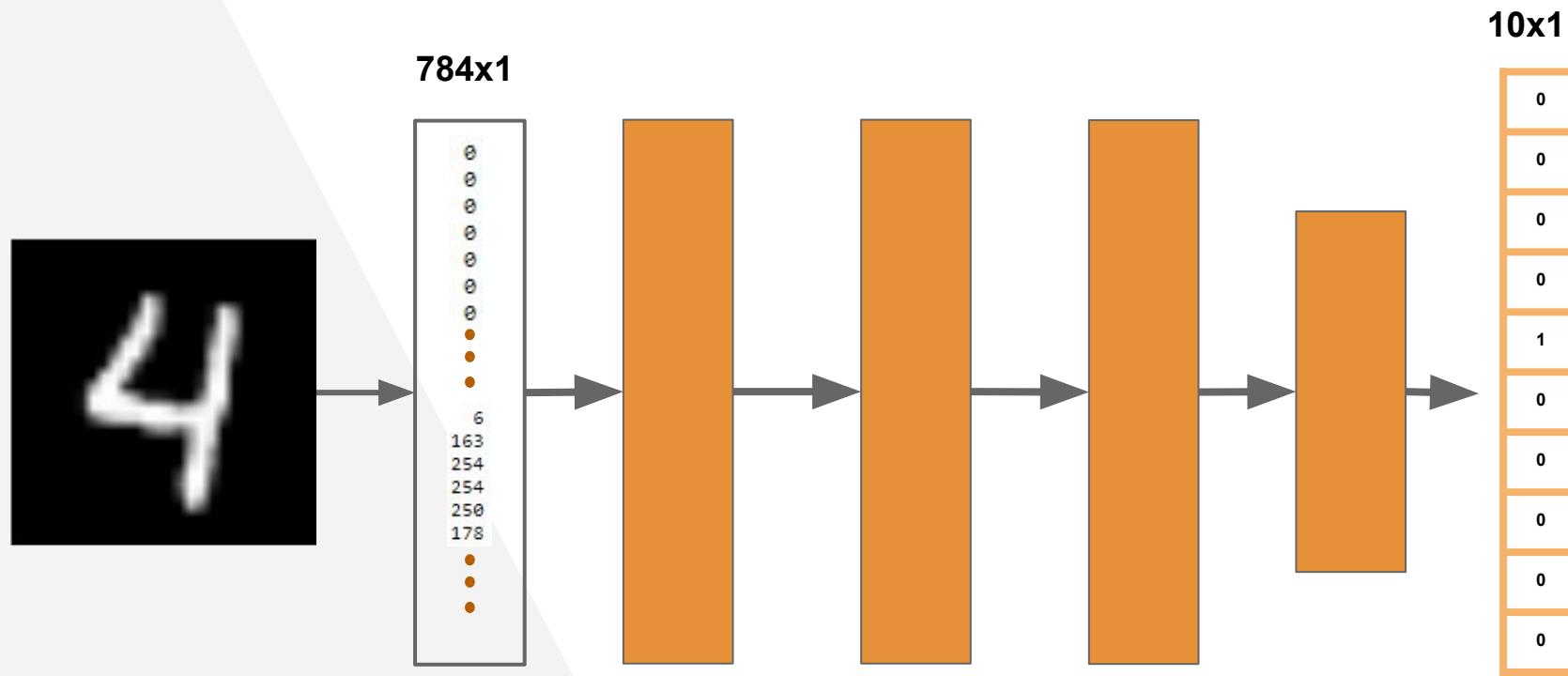


$$A = X * W + B$$

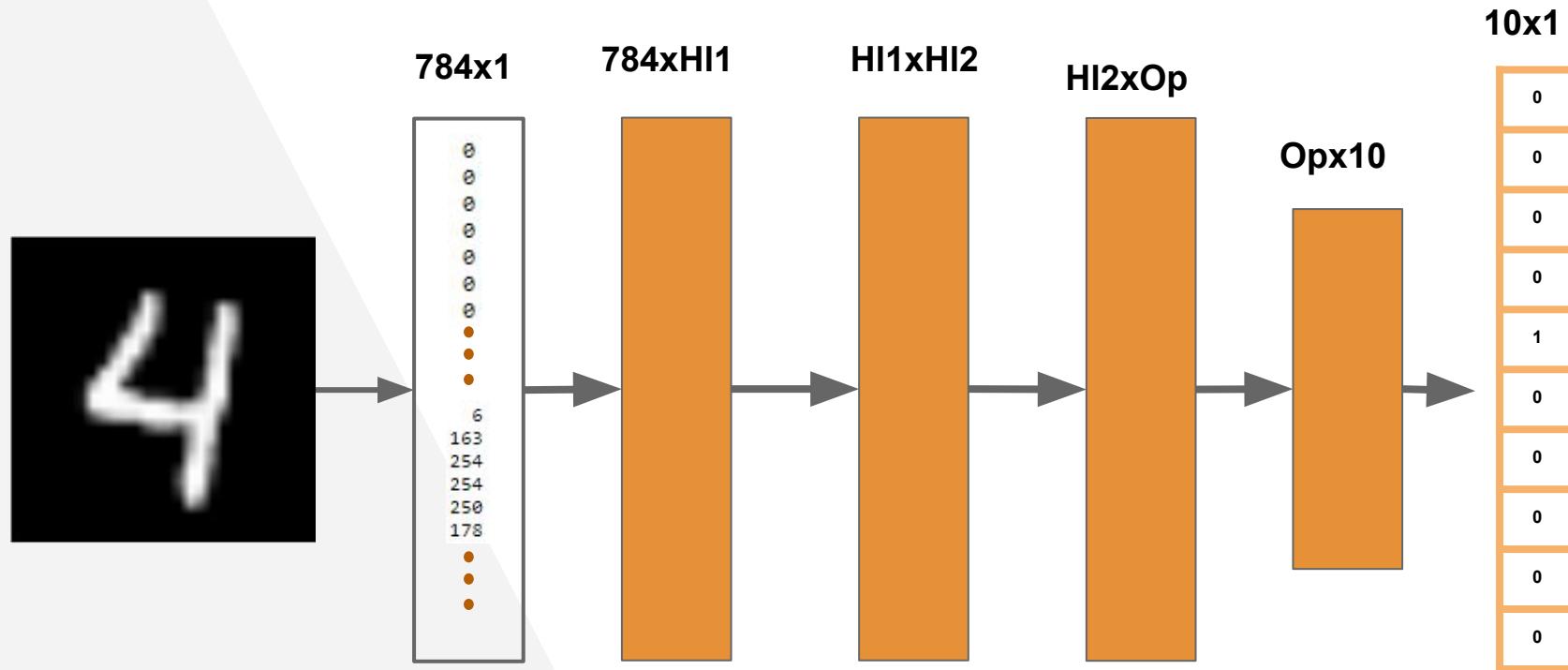
$$\begin{matrix} \mathbf{1x1} & \mathbf{1x2} & \mathbf{2x1} \\ \left[a_3 \right] = \left[w_{1,3} \quad w_{2,3} \right] * \begin{bmatrix} y(a_1) \\ y(a_2) \end{bmatrix} + [b_3] & & \end{matrix}$$

$$a_3 = y(a_1) * w_{1,3} + y(a_2) * w_{2,3} + b_3$$

Analizando la arquitectura de una red neuronal perceptrón profunda



Analizando la arquitectura de una red neuronal perceptrón profunda



Abrir: Apuntes Red_vanilla del curso aprendizaje profundo

¿Comó se entrena la red neuronal profunda?



Analizemos el problema desde el punto de vista de la red neuronal

Ejemplo: Queremos una red que identifique si existe la característica **Ragamuffin** en una imagen

Usemos la red neuronal más poderosa conocida por el hombre: El cerebro humano

Ustedes serán la red neuronal

Conjunto de entrenamiento: Aprendizaje supervisado



Predecir resultados



¿Cuál era la tarea?

Ragamuffin (sustantivo) viene del inglés medieval derivó en la palabra **rag** (*trapo*)

En español harapiento
ó con ropa rasgada



Evaluando los resultados



¿Cuál fue el problema ?

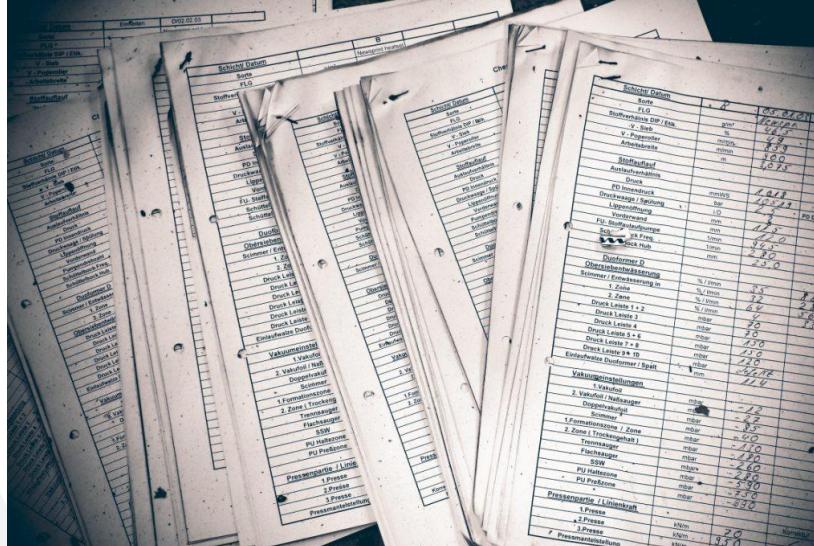
La arquitectura de la red neuronal

¿Cierto?



¿Cuál fue el problema?

El problema fueron los datos



¿Cuáles fueron los problemas en los datos?

Había **múltiples características en común** con el grupo de imágenes que califican como **Ragamuffin**: **todas eran mujeres, todas usaban lentes oscuros, tenían pelo largo, todas se encontraban en un ambiente de dentro de la ciudad, tenían más o menos el mismo rango de edad, estaban en la misma posición, entre otras.**



¿Cuáles fueron los problemas en los datos?

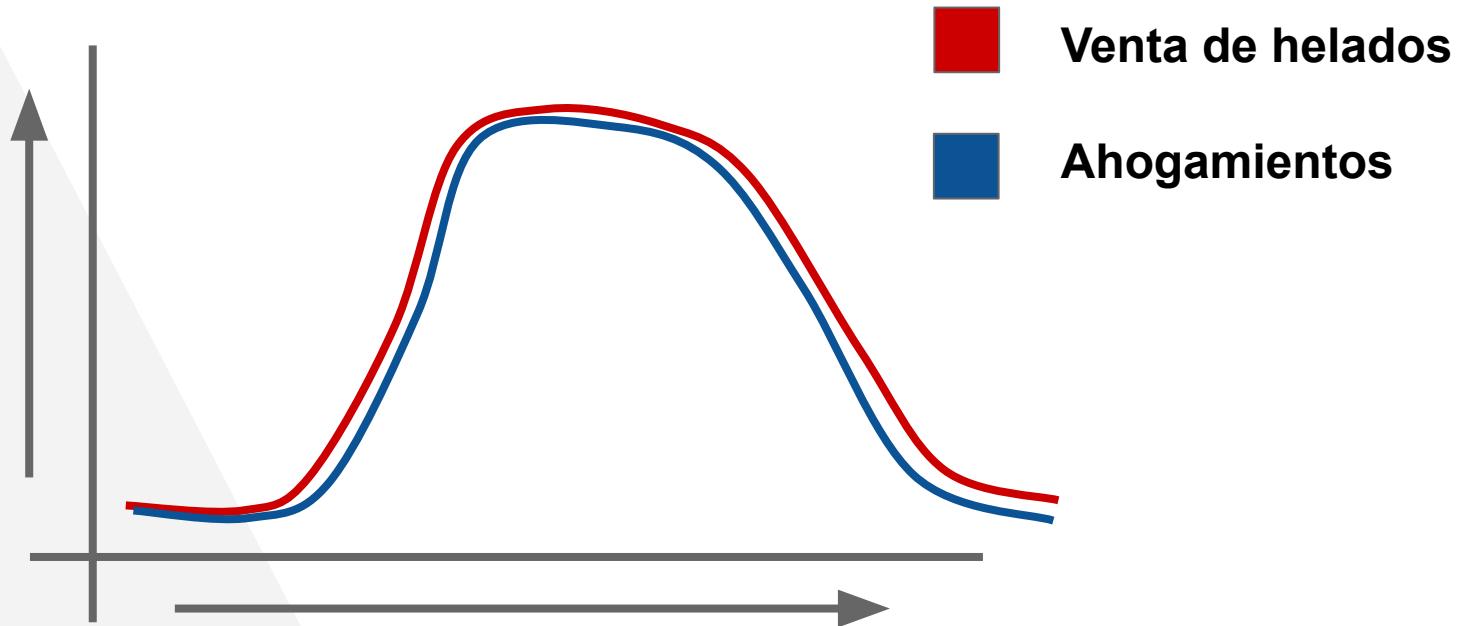


Habían **múltiples características en común** en el grupo que no califican como **Ragamuffin**, eran **hombres**, se encontraban en un **ambiente fuera de la ciudad**, se encontraban en **diferentes posiciones**, tienen **diferente edad**, entre otras.

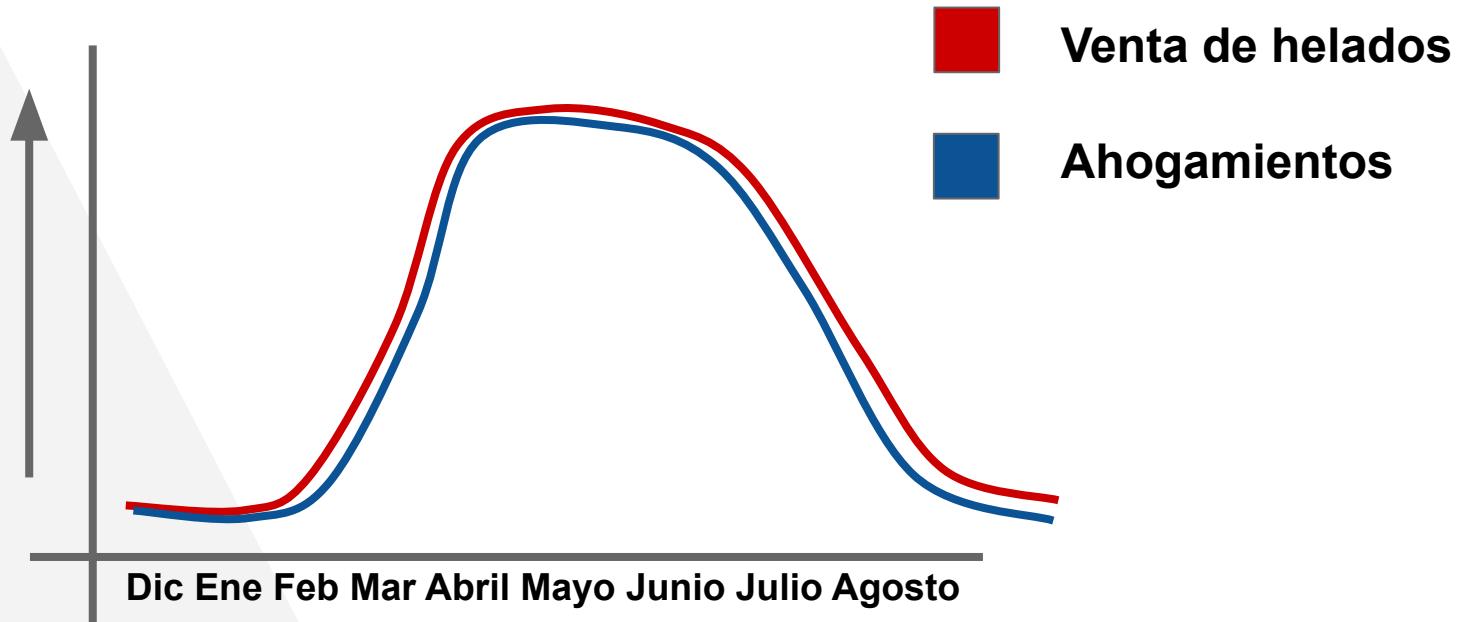
¿Cuáles fueron los problemas en los datos?

- ▶ Hay muchas cosas que no comparten los dos grupos. Cualquiera de estas podría ser un criterio para separarlos.
- ▶ El conjunto no fue lo suficientemente claro para ayudar a la red a identificar las características de interés, es decir, el conjunto de datos no fue representativo.

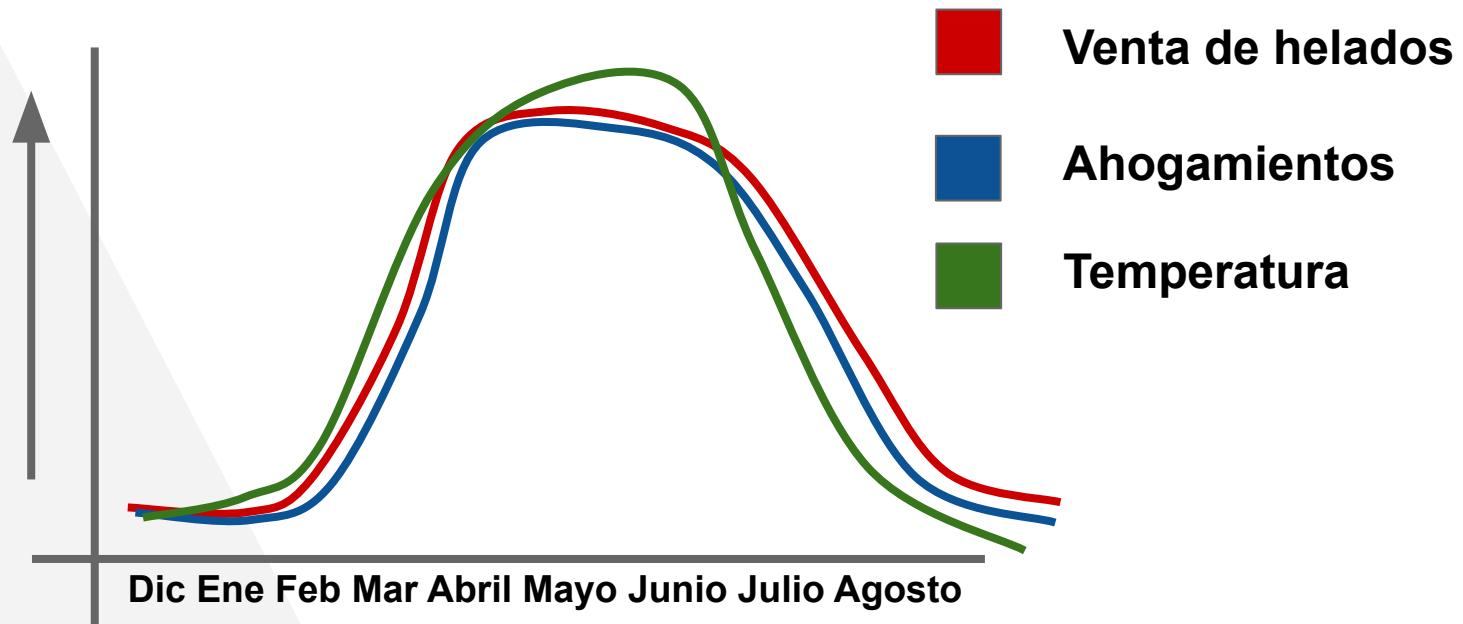
La red neuronal no es capaz de encontrar relaciones que no están presentes en los datos



La red neuronal no es capaz de encontrar relaciones que no están presentes en los datos



La red neuronal no es capaz de encontrar relaciones que no están presentes en los datos



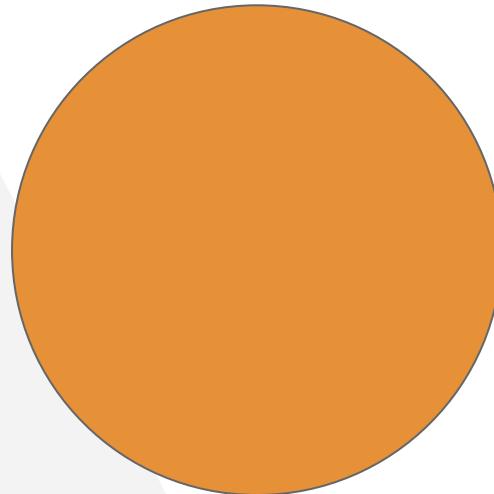
Recuerden: Los resultados son tan buenos como sus datos



https://www.ted.com/talks/tricia_wang_the_human_insights_missing_from_big_data

¿Qué importancia tienen los datos?

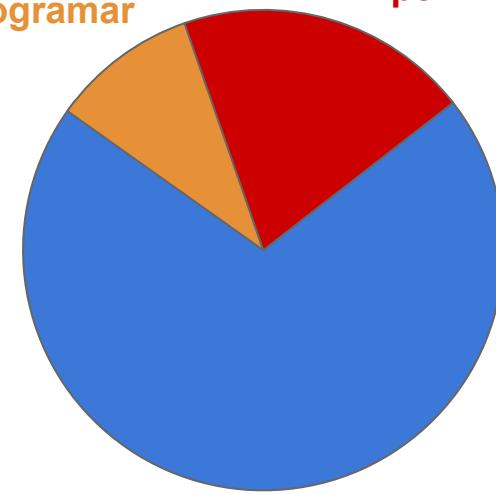
Programar



**Lo que las personas creen
que es Inteligencia Artificial**

Programar

Experimentar



Obtener datos para el
entrenamiento

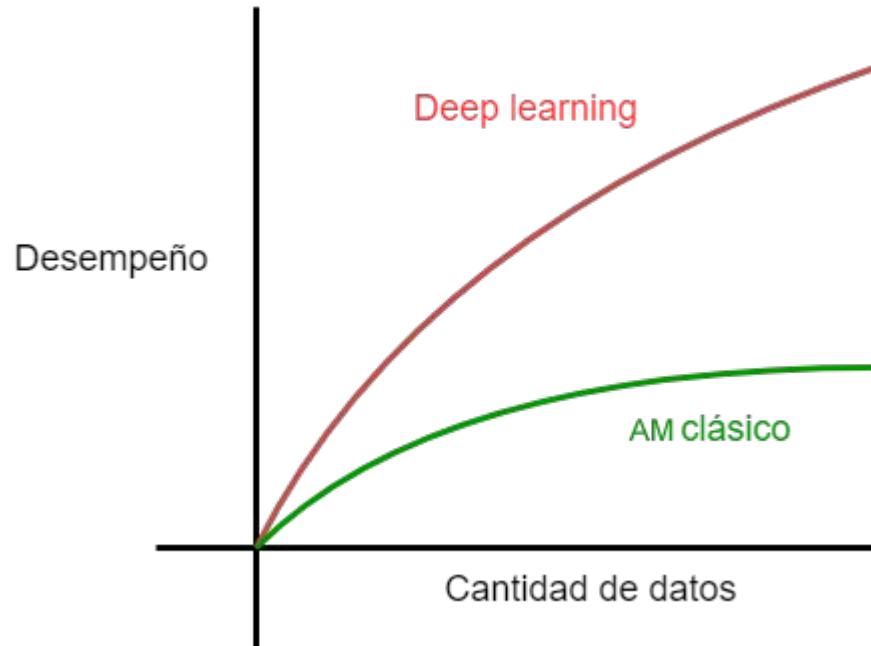
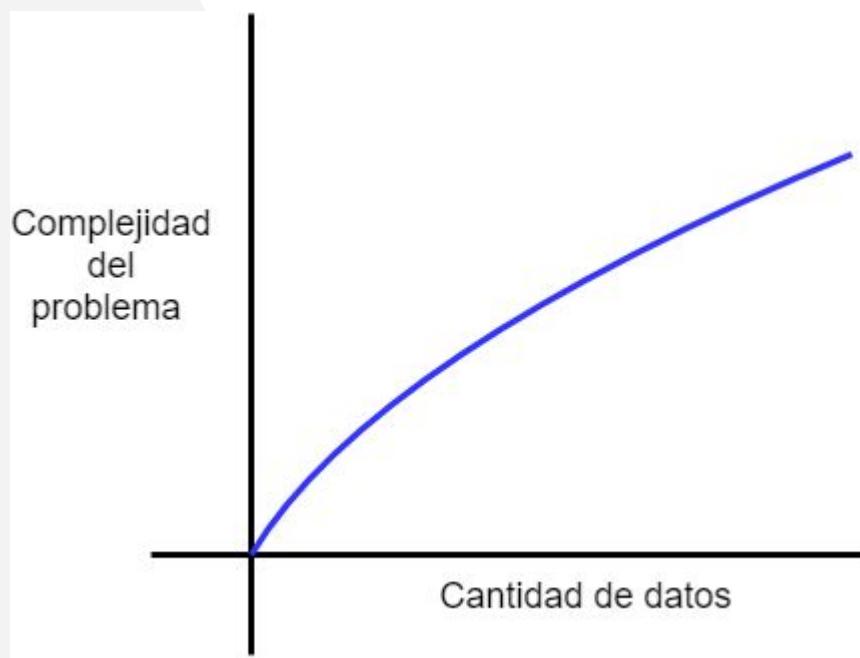
La realidad

En este momento deberían estar pensando en datos
y no tanto en algoritmos

1. Para determinar qué algoritmo necesitas, es vital que tengas los datos con los que vas resolver el problema
2. Si no sabes qué datos necesitas, no has estudiado tu problema. Por lo tanto, no importa cuántos algoritmos conozcas, no lo vas a resolver



¿Cuáles y cuántos datos necesito?



¿Dónde puedo encontrar mis datos?

- ▶ **Datos libres al público:** Fáciles de encontrar, pero ya fueron usados por muchas personas, lo que implica que no son suficientes para resolver el problema deseado (generalmente, no resuelven tu problema, pero pueden llegar a ayudar en ocasiones)

- ▶ **Datos artificiales:** Si los puedes generar significa que ya existe un modelo que representa adecuadamente tu problema y su solución. Así que, usa ese modelo y no quieras hacer uno profundo (solo son útiles cuando el modelo no es del todo correcto, pero da una idea de cómo resolver el problema. Sirve para hacer un pre entrenamiento)

¿Dónde puedo encontrar mis datos?

- ▶ **Web:** Muchos datos útiles, difíciles de buscar y casi imposible de encontrarlos correctamente etiquetados
- ▶ **Datos anotados previamente por alguien más:** Hay compañías que venden sus datos, pero son costosos. Como alternativa suele haber información que alguien usó para otro propósito en la red y que puede ayudar a solucionar el problema
- ▶ **Datos autogenerados:** Si quieres algo bien hecho, hazlo tú

Definiendo el conjunto de datos

Para definir el conjunto, es necesario **entender el problema**

¿Cuál es el objetivo del sistema?

Definiendo el conjunto de datos

Para definir el conjunto es necesario **entender el problema**

¿Cuál es el objetivo del sistema?

¿Cuál es la tarea a realizar?

Definiendo el conjunto de datos

Para definir el conjunto es necesario **entender el problema**

¿Cuál es el objetivo del sistema?

¿Cuál es la tarea a realizar?

¿Cuáles son las entradas y salidas del sistema?

Definiendo el conjunto de datos

Para definir el conjunto es necesario **entender el problema**

¿Cuál es el objetivo del sistema? **¿De qué forma se ingresan los datos?**

¿Cuál es la tarea a realizar?

¿Cuáles son las entradas y salidas del sistema?

Definiendo el conjunto de datos

Para definir el conjunto es necesario **entender el problema**

¿Cuál es el objetivo del sistema?

¿Cuál es la tarea a realizar?

¿Cuáles son las entradas y salidas del sistema?

¿De qué forma se ingresan los datos?

¿Qué información debe regresar el sistema?

Definiendo el conjunto de datos

Para definir el conjunto es necesario **entender el problema**

¿Cuál es el objetivo del sistema?

¿Cuál es la tarea a realizar?

¿Cuáles son las entradas y salidas del sistema?

¿De qué forma se ingresan los datos?

¿Qué información debe regresar el sistema?

¿En qué condiciones debe funcionar el sistema?

Analizar las condiciones bajo las que se va a operar el sistema

- ▶ Imágenes (tipos de iluminación, tipos de fondo, condiciones posición, translación, etc.)
- ▶ Audio (ruido ambiental, duración del audio, volumen, tipo de voz, etc.)
- ▶ **Ejemplo:** Clasificación de aves



Analizar las condiciones bajo las que se va a operar el sistema

- ▶ Imágenes (tipos de iluminación, tipos de fondo, condiciones posición, translación, etc.)
- ▶ Audio (ruido ambiental, duración del audio, volumen, tipo de voz, etc.)
- ▶ **Ejemplo:** Clasificación de aves



¿Cómo uso mis datos para resolver mi problema?

Generalmente, juntar los datos no es suficiente, ya que se encuentran en un estado en el que no pueden ser usados

- ▶ Datos Ruidosos
- ▶ Datos insuficientes
- ▶ Pre procesamiento de los datos
- ▶ Manejo de memoria

¿Cómo uso mis datos para resolver mi problema?

Generalmente, juntar los datos no es suficiente, ya que se encuentran en un estado en el que no pueden ser usados

- ▶ Datos Ruidosos (**Método semi-supervisado**)
- ▶ Datos insuficientes
- ▶ Pre procesamiento de los datos
- ▶ Manejo de memoria

Datos insuficientes

1. **Obtener más datos:** Evidentemente, es la solución más adecuada. Por eso, es necesario agotar cualquier medio posible para recolectar más datos
2. **Generar datos sintéticos:** Si no puedes encontrar los datos, fabrícalos
3. **Metodologías de aumentación de datos:** Haz pequeños cambios a tus datos, para que proporcionen información adicional
4. **Uso de modelos pre-entrenados:** Si no tienes datos suficientes te puedes apoyar en los entrenamientos de otras investigaciones, ya que eso permite que la red se apoye en los datos con los que fue entrenada previamente

Preprocesamiento de los datos

Las redes neuronales requieren entradas numéricas. Sin embargo, son capaces de trabajar con imágenes, audio y texto.

Por esto, es necesario procesar los datos de tal manera que se puedan expresar de forma numérica y así sean entradas de la red.

Manejo de memoria

El problema es que son **muchos**. Estamos hablando de unas 55000 imágenes para visión artificial (**50gb**), la Wikipedia para procesamiento de texto (**38gb**), 60 hrs de video (**120 gb**)

¿Quién tiene tanto espacio en memoria RAM?

Manejo de mini lotes

No podemos procesar todos los datos de un solo golpe. Por eso, los procesamos en pequeños lotes, después podemos guardar los lotes. Y finalmente, usarlos en el entrenamiento

Tenemos 2 opciones

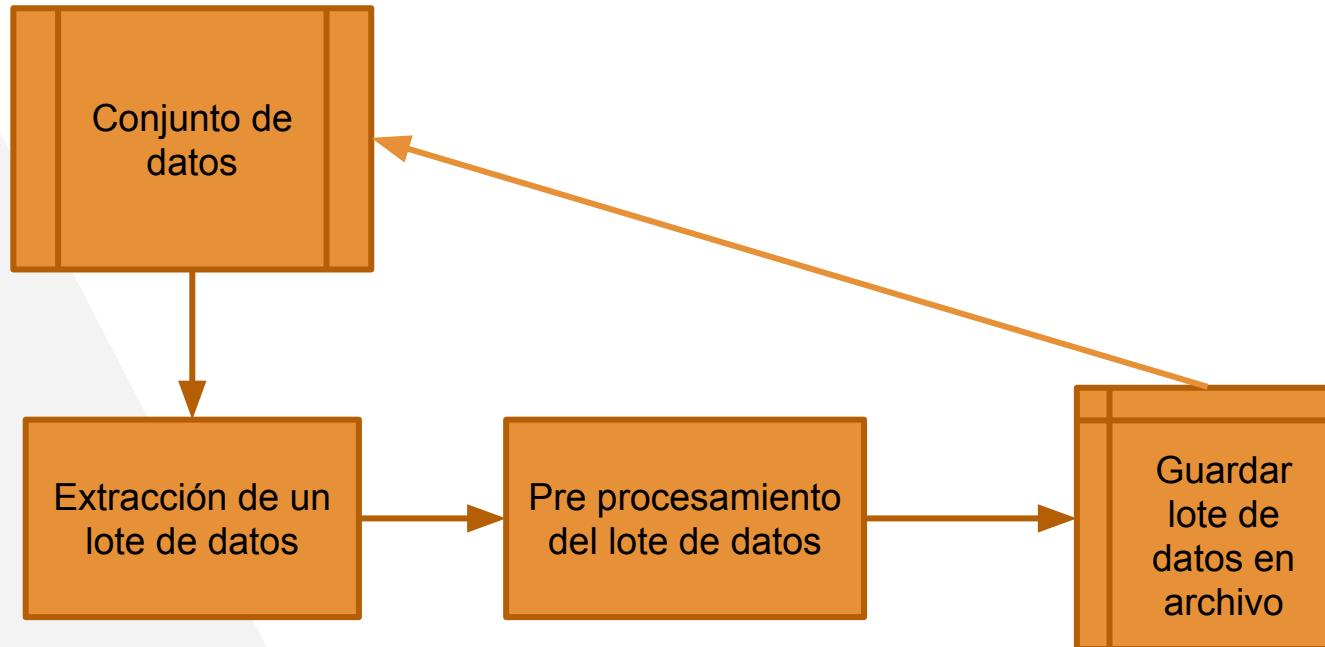
NumPy

np.save()
np.load()

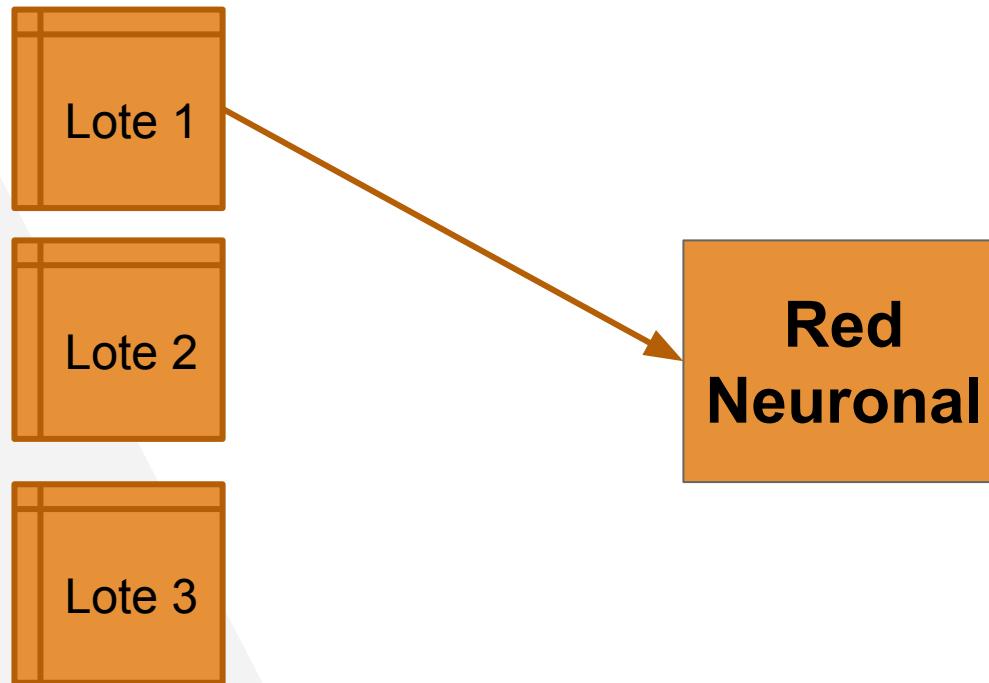
Pickle

pickle.dump()
pickle.load()

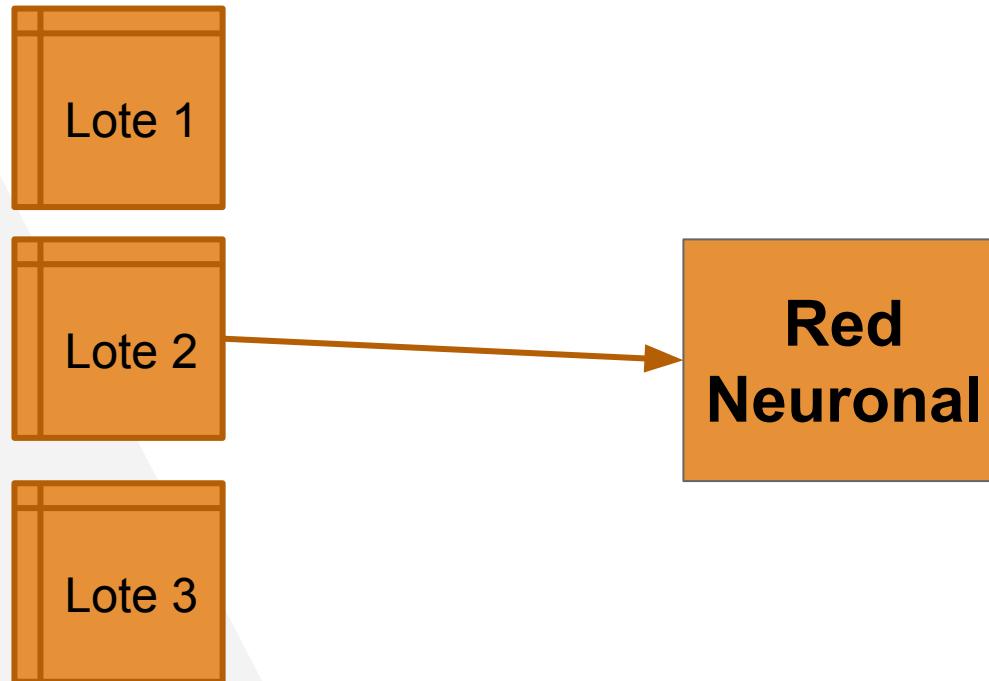
Manejo de mini lotes: Procesando los datos en lotes



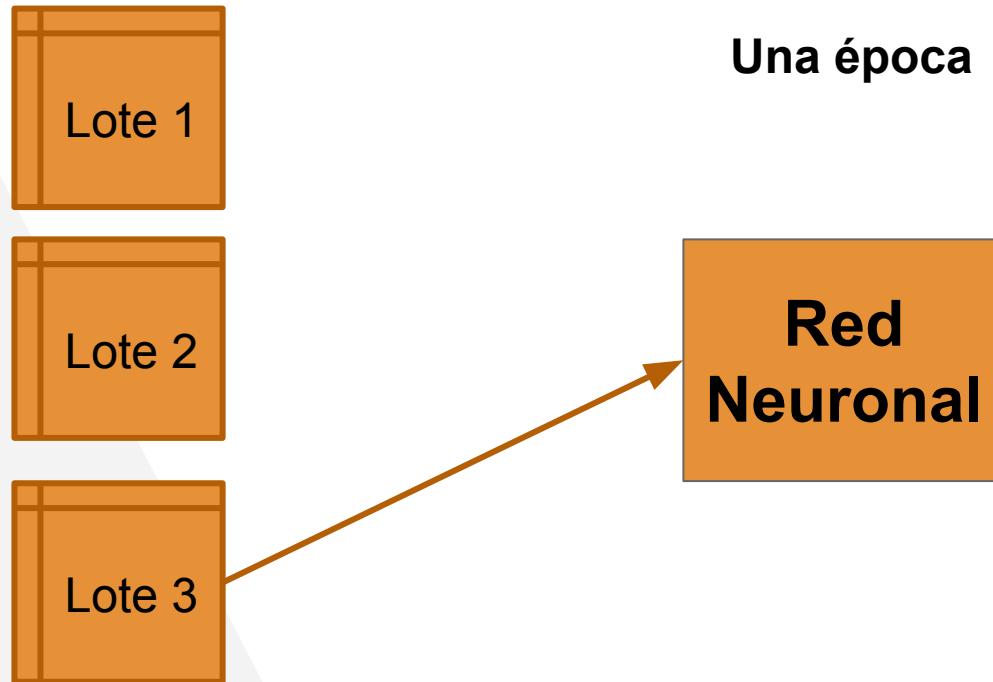
Manejo de mini lotes: Entrenando la red con lotes



Manejo de mini lotes: Entrenando la red con lotes



Manejo de mini lotes: Entrenando la red con lotes



Alternativa a mini lotes HDF5

Permite usar el disco duro en lugar de la memoria RAM



<http://www.h5py.org/>

Manejando datos propios: Análisis de sentimiento

Negativo

- ▶ simplistic , silly and tedious.
- ▶ the movie is a mess from start to finish.

Positivo

- ▶ take care of my cat offers a refreshingly different slice of asian cinema.
- ▶ this is a film well worth seeing , talking and singing heads and all.

Abrir: Apuntes Análisis de sentimiento del curso aprendizaje profundo

¿Cómo se ajusta una red neuronal para resolver un problema?

Entrenamiento-(Parámetros del modelo)

- ▶ Pesos
- ▶ Bias

Diseño-(Hyper-Parámetros del modelo)

- ▶ Función de costo
- ▶ Método de optimización
- ▶ Velocidad de aprendizaje
- ▶ Tamaño del lote (batch)
- ▶ Función de activación
- ▶ Tipo de entrenamiento
- ▶ Arquitectura

¿Cómo se ajusta una red neuronal para resolver un problema?

Entrenamiento-(Parámetros del modelo)

- ▶ Pesos
- ▶ Bias

Diseño-(Hyper-Parámetros del modelo)

- ▶ **Función de costo**
- ▶ Método de optimización
- ▶ Velocidad de aprendizaje
- ▶ Tamaño del lote (batch)
- ▶ Función de activación
- ▶ Tipo de entrenamiento
- ▶ Arquitectura

Hyper-Parámetros: Funciones de costo

Error cuadrático medio: (*Mean Squared Error MSE*) En el caso de la función de activación sigmoide converge muy lentamente

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$$

Error cuadrático medio logarítmico: Se usa cuando no se quiere penalizar diferencias muy grandes entre valores numéricamente grandes

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^n (\log(y^{(i)} + 1) - \log(\hat{y}^{(i)} + 1))^2$$

Hyper-Parámetros: Funciones de costo

Norma L2: (L2) Típicamente, usada para problemas de regresión en donde se quiere evitar que los pesos crezcan rápidamente

$$\mathcal{L} = \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$$

Error absoluto medio: (*mean absolute error*) Este error es más robusto en la presencia de valores atípicos (*outliers*), es útil para ocultar errores grandes con consecuencias grandes

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^n |y^{(i)} - \hat{y}^{(i)}|$$

Hyper-Parámetros: Funciones de costo

Porcentaje de error absoluto medio: (*Mean Absolute Percentage Error MAPE*) En teoría, se suena bien, pero en la práctica no funciona igual. No puede ser usado si algún valor es cero

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^n \left| \frac{y^{(i)} - \hat{y}^{(i)}}{y^{(i)}} \right| \cdot 100$$

Norma L1: (absolute error L1)

$$\mathcal{L} = \sum_{i=1}^n |y^{(i)} - \hat{y}^{(i)}|$$

Hyper-Parámetros: Funciones de costo

Entropía cruzada : (**cross entropy o binary classification**) Se ocupa para evaluar el error, en casos de clasificación que usan el método one hot para codificar la salida. Comparado con el error cuadrático medio, converge más rápidamente y es más factible que llegue al mínimo global

$$\mathcal{L} = -\frac{1}{n} \sum_{i=1}^n [y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

Hyper-Parámetros: Funciones de costo

Proximidad coseno: (*cosine proximity*) No toma en cuenta la magnitud de los vectores solo el ángulo existente entre ellos. Funciona para regresión

$$\mathcal{L} = -\frac{\mathbf{y} \cdot \hat{\mathbf{y}}}{\|\mathbf{y}\|_2 \cdot \|\hat{\mathbf{y}}\|_2} = -\frac{\sum_{i=1}^n y^{(i)} \cdot \hat{y}^{(i)}}{\sqrt{\sum_{i=1}^n (y^{(i)})^2} \cdot \sqrt{\sum_{i=1}^n (\hat{y}^{(i)})^2}}$$

Guía rápida funciones de costo

https://isaacchanghau.github.io/post/loss_functions/

Implementación en TensorFlow

https://www.tensorflow.org/api_docs/python/tf/losses

¿Cómo se ajusta una red neuronal para resolver un problema?

Entrenamiento-(Parámetros del modelo)

- ▶ Pesos
- ▶ Bias

Diseño-(Hyper-Parámetros del modelo)

- ▶ Función de costo
- ▶ **Método de optimización**
- ▶ Velocidad de aprendizaje
- ▶ Tamaño del lote (batch)
- ▶ Función de activación
- ▶ Tipo de entrenamiento
- ▶ Arquitectura

Hyper-Parámetros: Optimizador Gradiente descendente

- ▶ **Vanilla gradient descent:** Se busca minimizar la función de costo $\mathcal{L}(\theta)$ con respecto al objetivo $\theta \in \mathbb{R}^n$, donde este objetivo puede ser cualquier parámetro de la red neuronal

$$\theta := \theta - \eta \cdot \nabla_{\theta} \mathcal{L}(\theta)$$

Nota: El gradiente ∇_{θ} se calcula por medio de *backpropagation*, como se vio anteriormente

Problemas: Esta fórmula debe de ser calculada para cada variable del conjunto de entrenamiento, lo cual consume muchísimos recursos
Solamente garantiza la convergencia al mínimo global en problemas convexos

Hyper-Parámetros: Optimizador Gradiente estocástico descendente

- ▶ ***Stochastic gradient descent***: A diferencia del gradiente descendente, este propaga un conjunto de datos y calcula el gradiente promedio para ellos

$$\theta := \theta - \eta \cdot \nabla_{\theta} \mathcal{L}_i(\theta)$$

Nota: Este es mucho más rápido que el gradiente vainilla, ya que calcula por todos los datos

Problemas: Esta fórmula hace que el gradiente fluctúe bruscamente, lo cual puede ayudar a salir de mínimos locales. Pero también genera que no se pueda llegar a un mínimo exacto

Métodos de optimización para redes neuronales

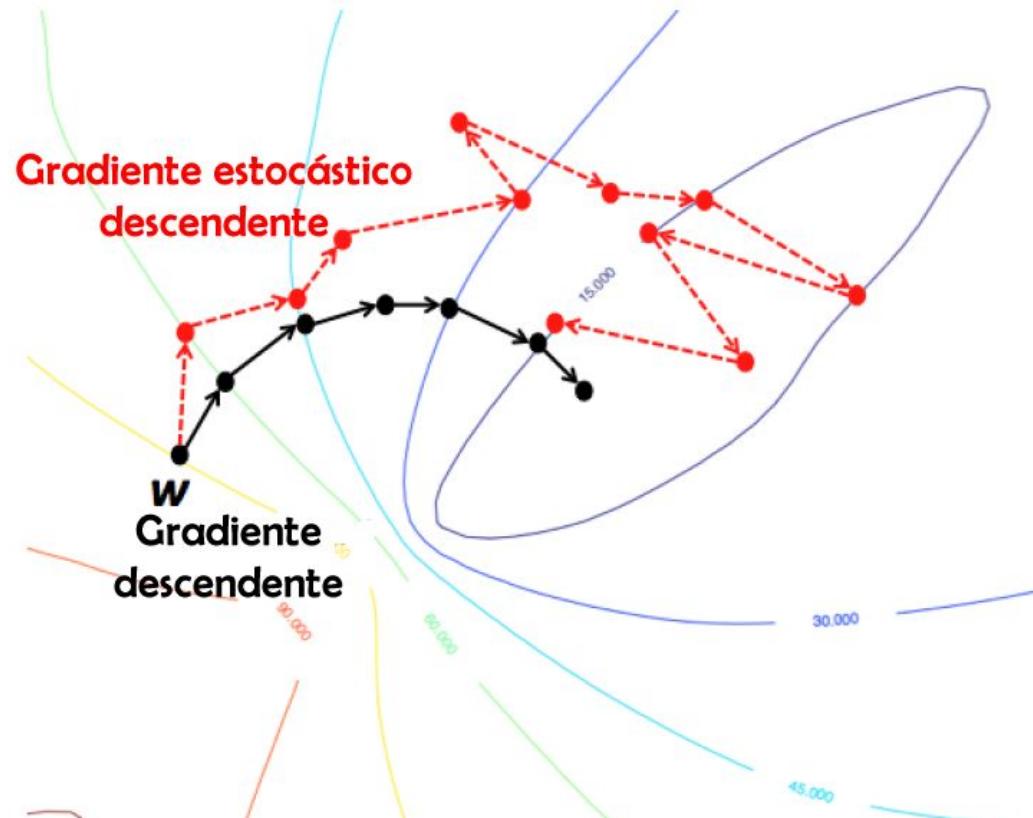
Percepción multicapa

- ▶ Gradiente descendente
- ▶ Entrenamiento de un elemento en un elemento
- ▶ Consume mucho tiempo
- ▶ Es muy preciso
- ▶ Es susceptible a mínimos locales

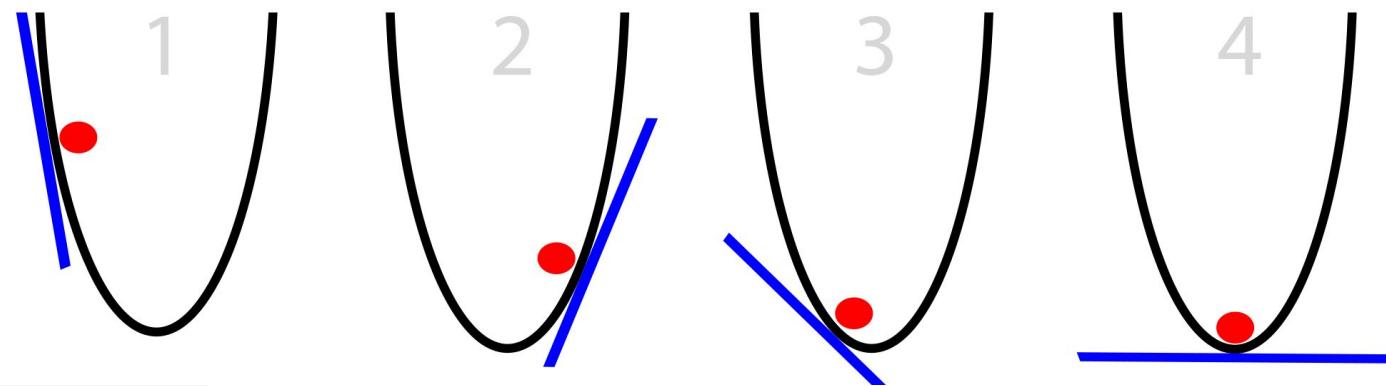
Percepción multicapa profunda

- ▶ Gradiente estocástico descendente
- ▶ Entrenamiento por lotes de elementos
- ▶ Consume un tiempo moderado
- ▶ Tiene una precisión adecuada
- ▶ Tiene cierta tolerancia a mínimos locales

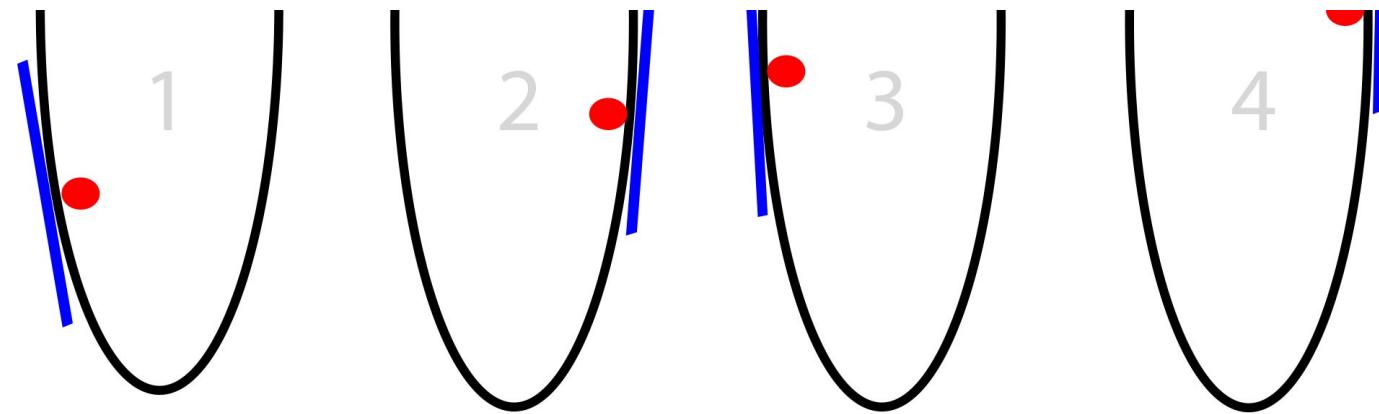
Comparativo GED vs. GD



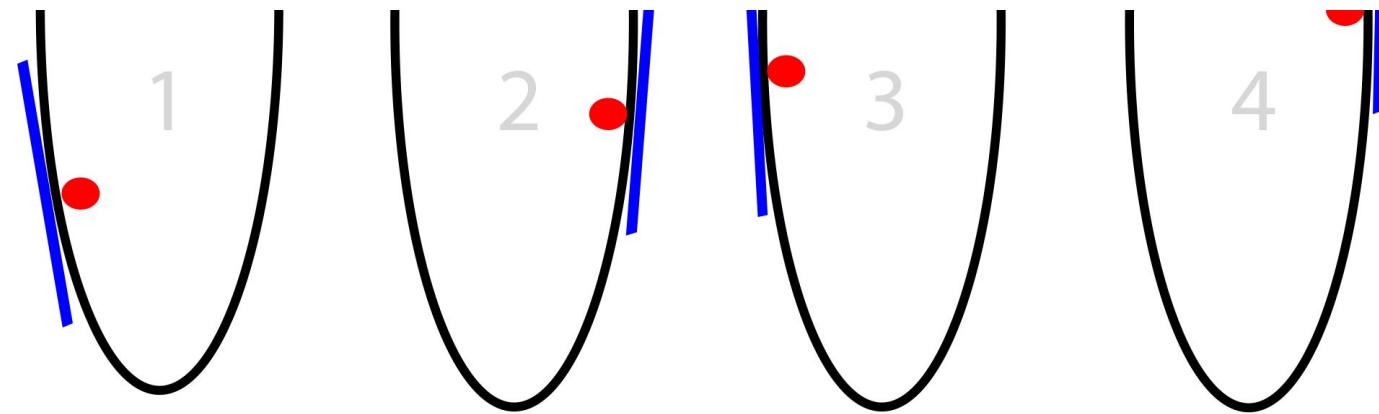
¿Cómo funciona la optimización por medio del gradiente?



Problema de las pendientes pronunciadas

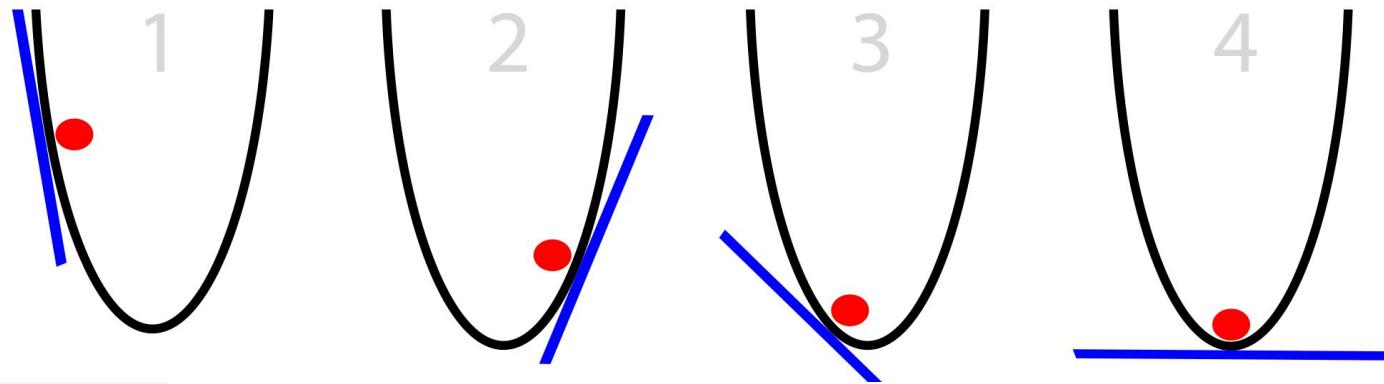


Problema de las pendientes pronunciadas



$$\theta := \theta - \eta \cdot \nabla_{\theta} \mathcal{L}(\theta)$$

Problema de las pendientes pronunciadas

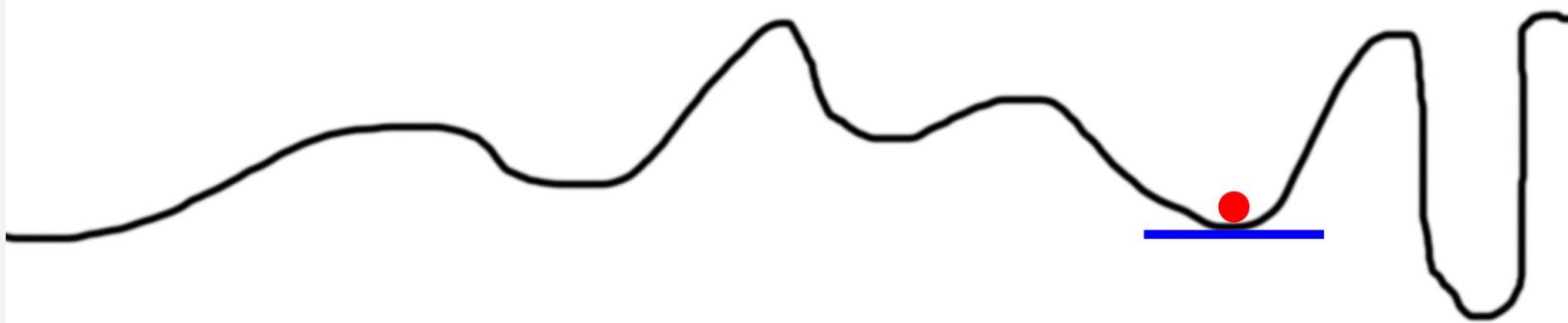


$$\theta := \theta - \eta \cdot \nabla_{\theta} \mathcal{L}(\theta)$$

La variable η debe tener un valor entre 0 y 1

Usualmente: **0.01**

Problema de los mínimos locales



Problema de pendientes pequeñas



Hyper-Parámetros: Problemas del gradiente descendente

- ▶ Es muy difícil ajustar la velocidad de aprendizaje η (**learning rate**)
- ▶ También, puede quedarse atorado en mínimos locales presentes en problemas no convexos

“

*Deep learning es la pesadilla de un teórico, ya que todos los problemas interesantes son **no convexos***

Yann LeCun

”

Métodos paramétricos para optimización

Momentum: Ya que el GED tiene problema navegando los mínimos locales se desarrolló la modificación de momento. Este parámetro permite ayudar al gradiente a salir de mínimos locales, el momento recuerda la modificación anterior Δw en cada iteración y la ocupa para modificar la siguiente iteración

$$\begin{aligned}\Delta w_t &:= \alpha \Delta w_{t-1} + \eta \nabla_{\theta} \mathcal{L}(\theta) \\ \theta &:= \theta - \Delta w_t\end{aligned}$$

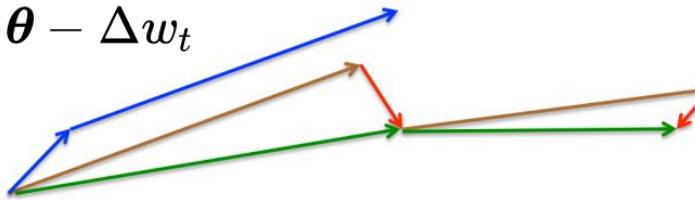
Donde: α es un hyper-parámetro que controla la contribución a cada vector. Generalmente, es de un valor de 0.9, como resultado mejora la convergencia

Métodos paramétricos para optimización

Nesterov momentum: Mejor conocido como *Nesterov accelerated gradient NAG*, le permite al gradiente conservar un conocimiento de la dirección que lleva para evitar que siga ciegamente las pendientes

$$\Delta w_t := \alpha \Delta w_{t-1} + \eta \nabla_{\theta} \mathcal{L}(\theta - \alpha \Delta w_{t-1})$$

$$\theta := \theta - \Delta w_t$$



Momento: Primero se calcula el gradiente (azul chico), gradiente acumulado(azul grande)

NGA: Gradiente acumulado (café grande) mide el gradiente y corrige (verde) útil para redes neuronales recurrentes (RNN)

Métodos paramétricos para optimización

AdaGrad: (gradiente adaptativo) versión modificada del gradiente estocástico descendente, en el cual se modifica la velocidad de aprendizaje en cada paso para cada parámetro η , basado en los gradientes previamente calculados para θ_i

$$\theta_t := \theta_{t-1} - \frac{\eta \cdot \nabla_{\theta} \mathcal{L}(\theta)}{\sqrt{G_{t-1} + \epsilon}}$$

Incrementa: Parámetros dispersos y decrementa en caso contrario

Permite eliminar la necesidad de ajustar manualmente la velocidad de aprendizaje

Su principal desventaja es que cuando el proceso de entrenamiento es muy largo, el gradiente se acumula y la velocidad de aprendizaje decrece hasta que deja de contribuir

Métodos paramétricos para optimización

AdaDelta: Es una modificación de AdaGrad, que busca limitar el crecimiento monotónico de la velocidad de entrenamiento. Por eso, para su ajuste solo considera una ventana de valores de gradiente, en lugar de todo el histórico

$$\Delta w_t = - \frac{RMS[\Delta w]_{t-1}}{RMS[g]_t} \cdot g_t$$

$$\theta_{t+1} = \theta_t + \Delta w_t$$

Métodos paramétricos para optimización

Adam: Estimador de momento adaptativo, es otro método que adapta la velocidad de aprendizaje. Este método es el más completo y generalmente, entrega los mejores resultados

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\eta \cdot \hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$$

Adam combina la capacidad de *AdaGrad*, pero también es bueno para el caso en el que el objetivo varía continuamente, lo cual lo hace la mejor opción para problemas convexos

Información adicional

Guía rápida optimizadores

https://isaacchanghau.github.io/post/parameters_update/

Implementación en TensorFlow

https://www.tensorflow.org/api_guides/python/train

¿Cómo se ajusta una red neuronal para resolver un problema?

Entrenamiento-(Parámetros del modelo)

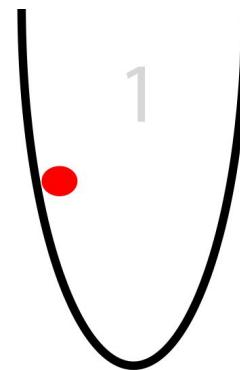
- ▶ Pesos
- ▶ Bias

Diseño-(Hyper-Parámetros del modelo)

- ▶ Función de costo
- ▶ Método de optimización
- ▶ **Velocidad de aprendizaje**
- ▶ Tamaño del lote (batch)
- ▶ Función de activación
- ▶ Tipo de entrenamiento
- ▶ Arquitectura

Hyper-Parámetros: Velocidad de aprendizaje

- ▶ Muy pequeño lleva a una lenta y dolorosa convergencia
- ▶ Muy grande lleva a fluctuaciones alrededor del mínimo, lo hace que la respuesta no converja



$$\eta = 0.1$$

$$\eta = 0.001$$

$$\eta = 0.00001$$

¿Cómo se ajusta una red neuronal para resolver un problema?

Entrenamiento-(Parámetros del modelo)

- ▶ Pesos
- ▶ Bias

Diseño-(Hyper-Parámetros del modelo)

- ▶ Función de costo
- ▶ Método de optimización
- ▶ Velocidad de aprendizaje
- ▶ **Tamaño del lote (batch)**
- ▶ Función de activación
- ▶ Tipo de entrenamiento
- ▶ Arquitectura

Hyper-Parámetros: Tamaño del lote

Mini-batch (stochastic) gradient descent: Este gradiente descendente se calcula para pequeños lotes de n datos, cuando $n=1$ tenemos el gradiente vainilla

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \eta \cdot \nabla_{\boldsymbol{\theta}} \mathcal{L}_{i:i+n}(\boldsymbol{\theta})$$

Este gradiente es el que se suele usar en redes neuronales profundas, y al que se refieren en general cuando se menciona el gradiente estocástico descendente

Generalmente, el **tamaño del lote** se define por el **espacio en memoria** y la precisión del algoritmo

Hyper-Parámetros: Tamaño del lote

Lote Grande

- ▶ Tiempo de entrenamiento reducido
- ▶ Precisión reducida
- ▶ Requiere más memoria RAM
- ▶ Mejor respuesta ante el problema de mínimos locales
- ▶ Mejor generalización

Lote pequeño

- ▶ Tiempo de entrenamiento largo
- ▶ Precisión más grande
- ▶ Requiere poca memoria
- ▶ Propenso a mínimos locales
- ▶ Propenso a sobre entrenamiento

$$1 < \text{Batch} < \text{Memory}$$

¿Cómo se ajusta una red neuronal para resolver un problema?

Entrenamiento-(Parámetros del modelo)

- ▶ Pesos
- ▶ Bias

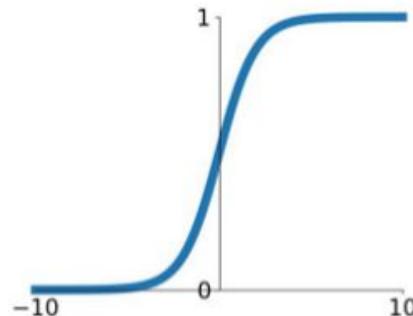
Diseño-(Hyper-Parámetros del modelo)

- ▶ Función de costo
- ▶ Método de optimización
- ▶ Velocidad de aprendizaje
- ▶ Tamaño del lote (batch)
- ▶ **Función de activación**
- ▶ Tipo de entrenamiento
- ▶ Arquitectura

Hyper-Parámetros: Función de activación

Función Sigmóide: Esta función se utiliza generalmente en las capas intermedias, toma un valor real y lo acota entre 0 y 1

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



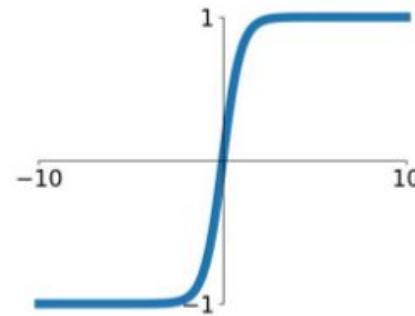
Las sigmoides saturan y eliminan los gradientes

Las salidas de las sigmoides no se centran en cero

Hyper-Parámetros: Función de activación

Función Tangente Hiperbólica: Esta función se utiliza generalmente en las capas intermedias alternativa a la función sigmoide, toma un valor real y lo acota entre -1 y 1

$$\tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$



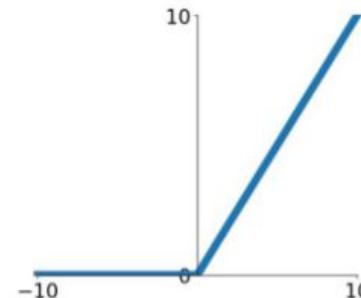
Las tangentes hiperbólicas saturan y eliminan los gradientes

Las salidas de la tangente está centrada en cero

Hyper-Parámetros: Función de activación

Unidades lineales rectificadas (RELU): Esta función se encuentra acotada en cero para valores negativos, con un rango de 0 a ∞

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



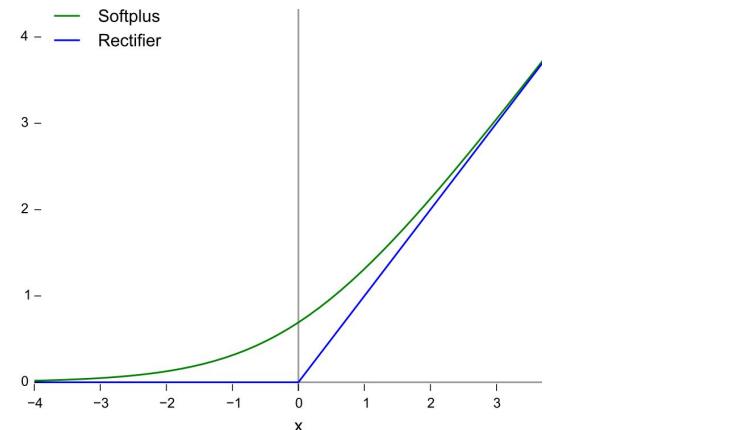
Se ha comprobado que acelera la convergencia cuando se usa gradiente estocástico descendente en comparación con la sigmoidal y tangente hiperbólica. ([Krizhevsky et al](#))

Como desventaja, en algunos casos se pueden morir durante el entrenamiento

Hyper-Parámetros: Función de activación

Softplus: Esta función es una versión suave de la **ReLU**. La salida se encuentra acotada en cero para valores negativos, con un rango de 0 a ∞

$$f(x) = \ln(1 + e^x)$$

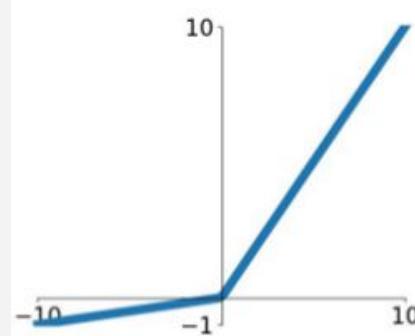


Tiene un comportamiento muy parecido a la **ReLU**, solo que es más suave en la transición, pero con un cómputo más lento

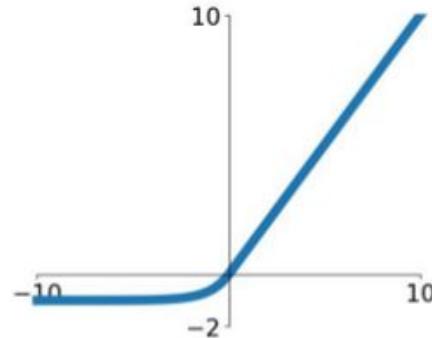
Hyper-Parámetros: Función de activación

Leaky RELU y ELU: Estas funciones buscan arreglar el problema de la “muerte de la RELU” evitando que las funciones se saturen en cero

Leaky RELU



ELU



$$\begin{cases} f(x) = \alpha x, & (x < 0) \\ f(x) = x, & (x \geq 0) \end{cases}$$

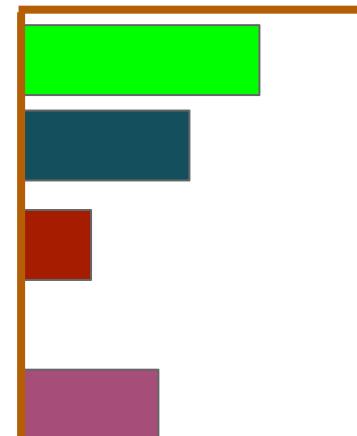
$$\begin{cases} f(x) = \alpha(e^x - 1), & (x < 0) \\ f(x) = x, & (x \geq 0) \end{cases}$$

Hyper-Parámetros: Función de activación

Softmax: Esta función se usa para clasificación, permitiendo determinar un intervalo de confianza. Ajusta la salida a un vector de k dimensiones de valores arbitrarios a valores acotados entre 0 y 1. La suma de todos es igual a 1

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}, j = 1, 2, \dots, K$$

Probabilidades



Información adicional

Guía rápida funciones de activación

https://isaacchanghau.github.io/post/activation_functions/

Implementación en TensorFlow

https://www.tensorflow.org/versions/r1.0/api_guides/python/nn

¿Cómo se ajusta una red neuronal para resolver un problema?

Entrenamiento-(Parámetros del modelo)

- ▶ Pesos
- ▶ Bias

Diseño-(Hyper-Parámetros del modelo)

- ▶ Función de costo
- ▶ Método de optimización
- ▶ Velocidad de aprendizaje
- ▶ Tamaño del lote (batch)
- ▶ Número de épocas
- ▶ **Tipo de entrenamiento**
- ▶ Arquitectura

Combinando los tipos de Aprendizaje Máquina

Existen tres:

- ▶ Supervisado
- ▶ No-supervisado
- ▶ Reforzado



Combinado los tipos de Aprendizaje Máquina

Existen tres:

- ▶ Supervisado (Caso MNIST, Sentimiento)
- ▶ No-supervisado
- ▶ Reforzado

Descubriendo una estructura en datos de muchas dimensiones

La hipótesis manifold:

- ▶ La información natural se encuentra en pocas dimensiones
- ▶ Esto se debe a las variables reales en la información natural

Ejemplo: Reconocimiento facial

- ▶ **Representación original**
 - ▶ Imagen de 1000X1000 píxeles = 1,000,000 dimensiones
- ▶ **Datos físicos**
 - ▶ Ubicación 3 coordenadas cartesianas y 3 ángulos de Euler
 - ▶ Los humanos tienen menos de 50 músculos en la cara
 - ▶ Por lo tanto, la expresión facial requiere menos de 56 dimensiones



Extracción de características



Ustedes deciden qué características son las importantes



Dependiendo de las características es el resultado

Buenas



Malas



Validando su desempeño



¿Cómo funciona el sistema?



Crimen



Testigo



Retratista



Retrato

¿Cómo le enseñarían al testigo a obtener
mejores características?

TEACH

Red neuronal autoencoder

- ▶ El autoencoder es una red neuronal usada para el aprendizaje no-supervisado
- ▶ Su objetivo es encontrar una representación de las características principales de un conjunto de datos
- ▶ Típicamente, se usa para obtener una reducción dimensional del problema
- ▶ Permite el entrenamiento en casos donde no se cuente con datos etiquetados

¿Cómo funciona el sistema?



Características



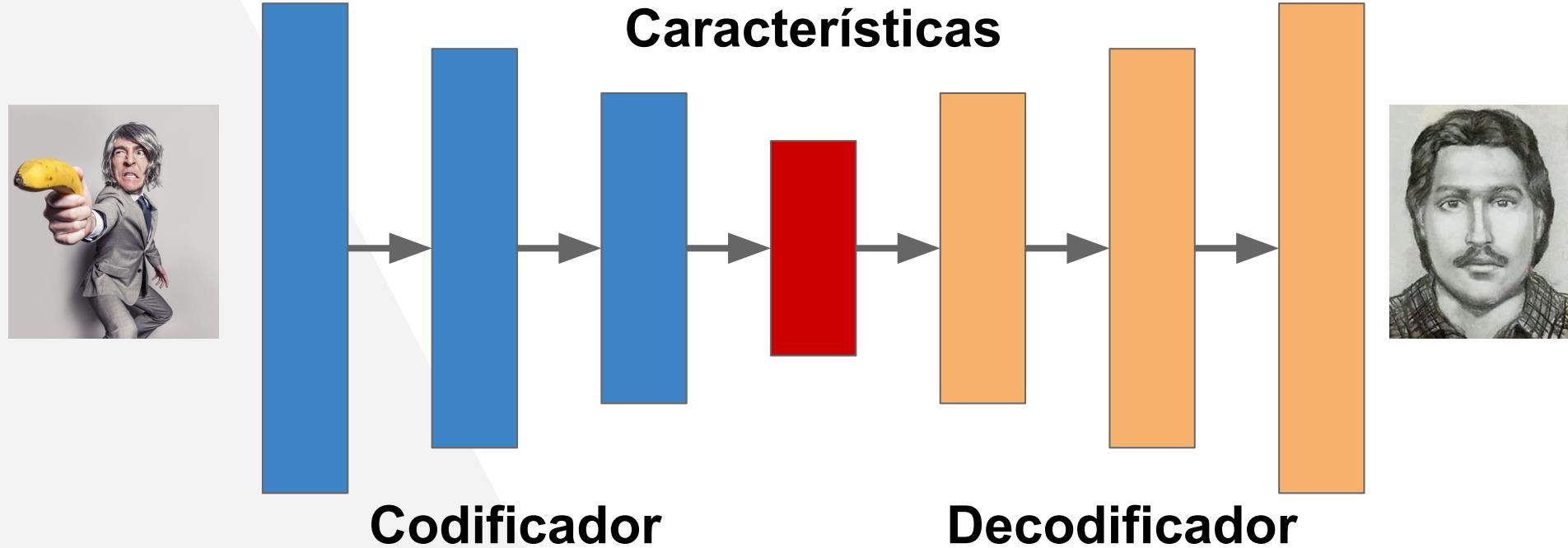
Crimen

Codificador

Decodificador

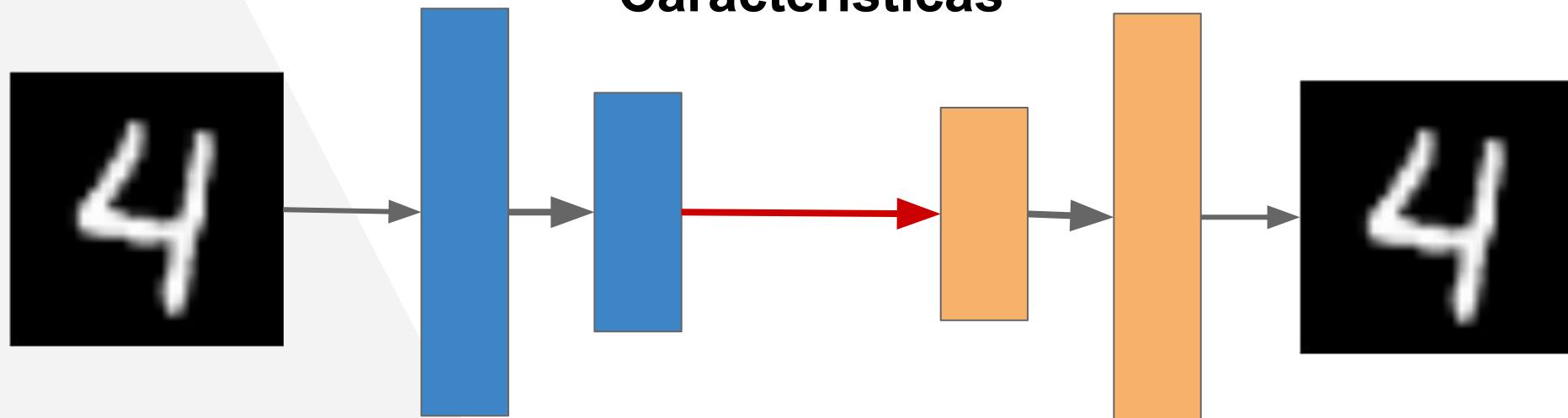
Retrato

Autoencoder usando una perceptrón profunda



Autoencoder usando una perceptrón profunda

Características

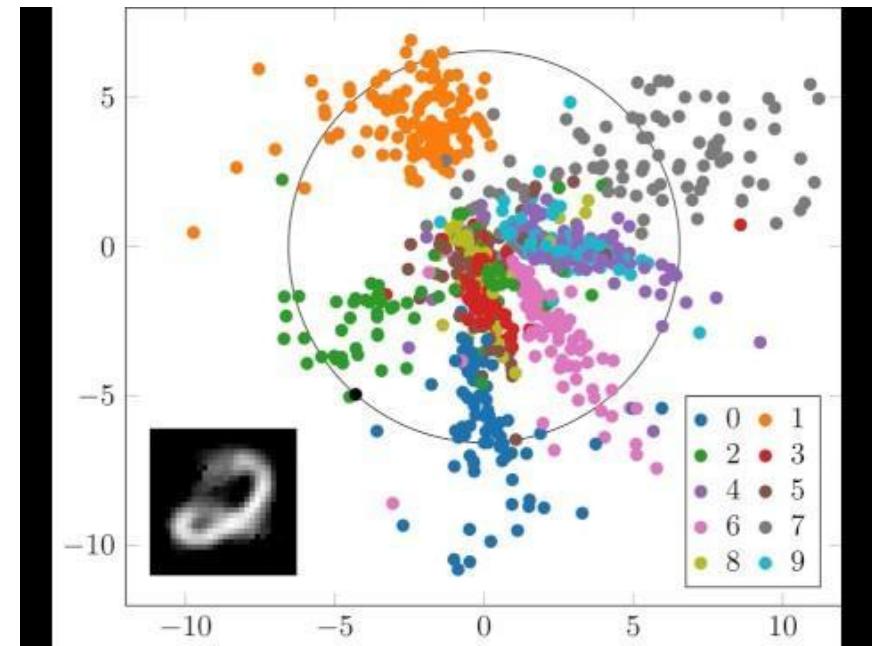
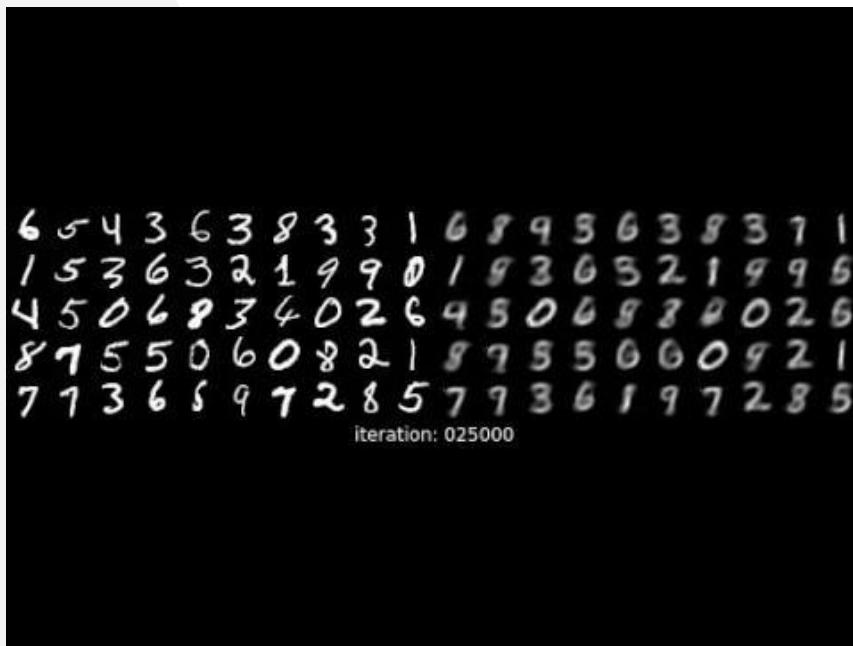


Codificador

Abrir: Apuntes Autoencoder_DNN del curso aprendizaje profundo

Decodificador

Autoencoder usando una perceptrón profundo



Combinado los tipos de Aprendizaje Máquina

Existen tres:

- ▶ Supervisado
- ▶ No-supervisado
- ▶ Reforzado

Semi-supervisado

- ▶ Extractor de características no supervisado + clasificador supervisado
- ▶ Extractor de características no supervisado + entrenamiento global supervisado
- ▶ Transferencia de conocimiento

¿Qué características tienen mis datos?

¿Están etiquetados o los puedo etiquetar?

- ▶ Correctamente etiquetados
- ▶ Parcialmente etiquetados

¿Todos están etiquetados?

- ▶ Si
- ▶ Algunos
- ▶ Ninguno

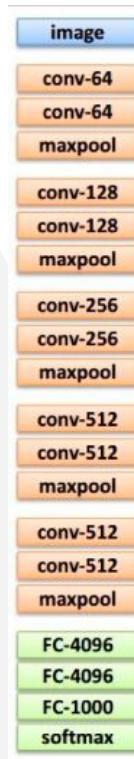
¿Cómo entreno a la red neuronal?

- ▶ **Puramente supervisado:** Cuando los datos están completamente etiquetados
- ▶ **Extractor de características no supervisado + clasificador supervisado:** Cuando se cuenta con pocos datos etiquetados correctamente y el resto no está etiquetado
- ▶ **Extractor de características no supervisado + entrenamiento global supervisado:** Cuando se tienen datos pobemente etiquetados
- ▶ **Transferencia de conocimiento:** Cuando se tiene una red entrenada para un problema similar y pocos datos etiquetados

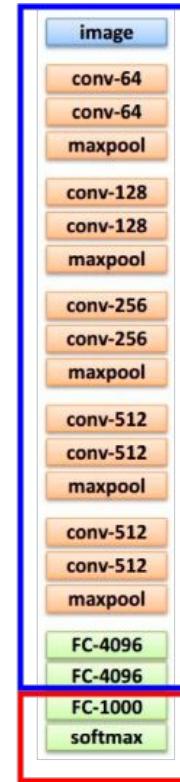
Transferencia de conocimiento

- ▶ Se denomina transferencia de conocimiento al proceso de ocupar modelos pre entrenados
- ▶ Estos modelos ya han sido entrenados con grandes cantidades de datos. Generalmente, con buenos resultados, es decir, la red ya tiene un conocimiento base que puede ser utilizado como referencia para hacer otra tarea.

Transferencia de conocimiento



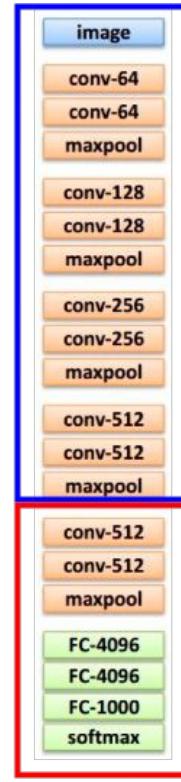
Red neuronal
entrenada con
(image net)



Aplicación basada
en imágenes
Conjunto de datos
pequeño

Conservar
estos
pesos

Entrenar
esto

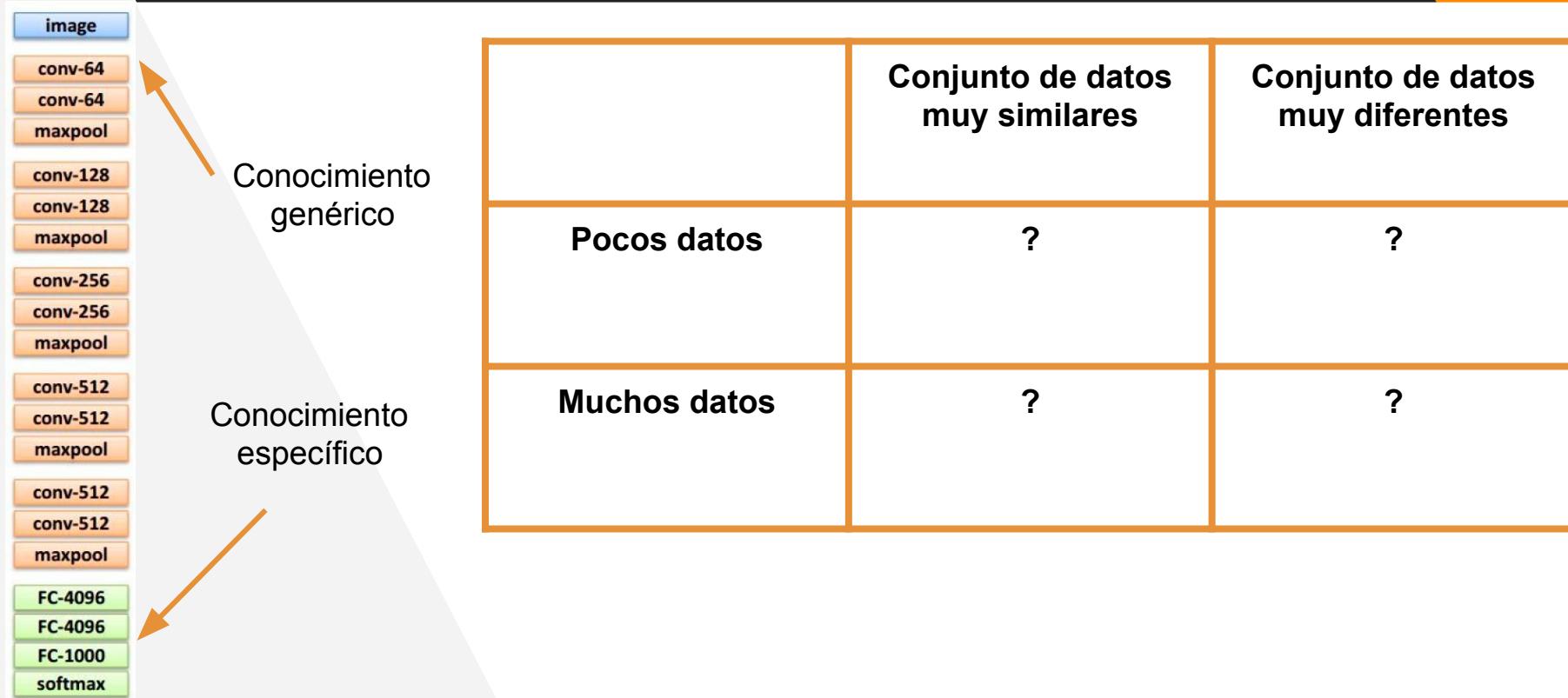


Aplicación basada
en imágenes
Conjunto de datos
mediano

Conservar
estos
pesos

Entrenar
esto

Transferencia de conocimiento



Transferencia de conocimiento

image		
conv-64		Conjunto de datos muy similares
conv-64		Conjunto de datos muy diferentes
maxpool		
conv-128		
conv-128	Pocos datos	Solo entrenar el clasificador lineal
maxpool		?
conv-256		
conv-256		
maxpool		
conv-512	Muchos datos	Se puede realizar una extracción de características específica
conv-512		?
maxpool		
conv-512		
conv-512		
maxpool		
FC-4096		
FC-4096		
FC-1000		
softmax		

Conocimiento genérico

Conocimiento específico

Transferencia de conocimiento

image		
conv-64		Conjunto de datos muy similares
conv-64		Conjunto de datos muy diferentes
maxpool		
conv-128		
conv-128	Pocos datos	Solo entrenar el clasificador lineal
maxpool		Estás en problemas busca otra metodología
conv-256		
conv-256	Muchos datos	Se puede realizar una extracción de características específica
maxpool		Entrena de nuevo la red completa
conv-512		
conv-512		
maxpool		
conv-512		
conv-512		
maxpool		
FC-4096		
FC-4096		
FC-1000		
softmax		

¿Cómo se ajusta una red neuronal para resolver un problema?

Entrenamiento-(Parámetros del modelo)

- ▶ Pesos
- ▶ Bias

Diseño-(Hyper-Parámetros del modelo)

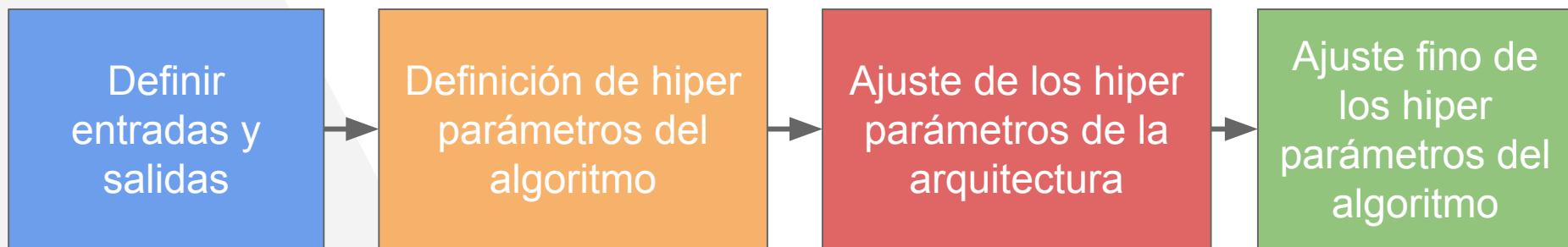
- ▶ Función de costo
- ▶ Método de optimización
- ▶ Velocidad de aprendizaje
- ▶ Tamaño del lote (batch)
- ▶ Número de épocas
- ▶ Tipo de entrenamiento
- ▶ **Arquitectura**

Arquitectura de una red neuronal

- ▶ Todas las redes neuronales son un conjunto de neuronas conectadas.
- ▶ Sin embargo, la forma en la que se conectan impacta drásticamente en su comportamiento

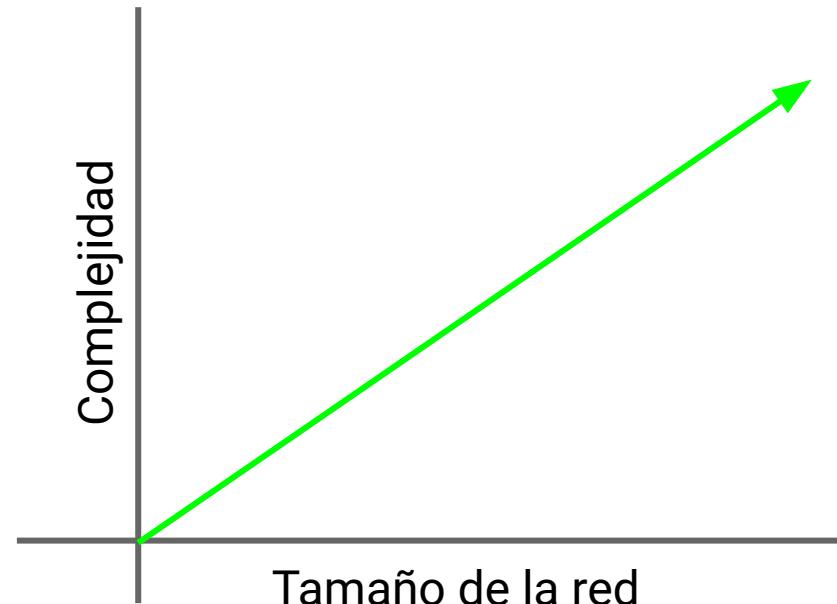
A la forma en la que se conecta una red neuronal se le denomina arquitectura

Proceso de diseño de una red neuronal



Profundidad de la arquitectura de una red

No existe un manual para definir la profundidad que debe tener una arquitectura para resolver un problema



Profundidad de la arquitectura de una red

Dado un problema específico

**Red con neuronas
insuficientes para
resolver el
problema**

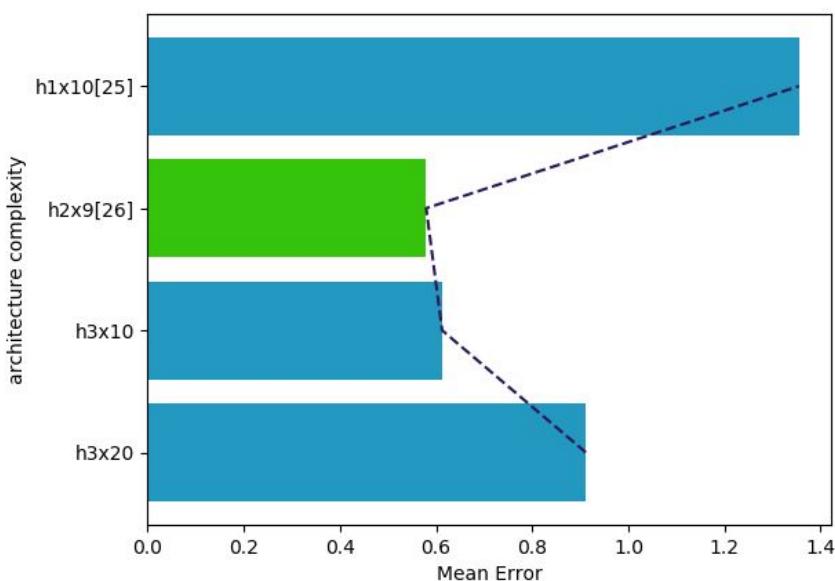
**Tamaño adecuado
de red**

**Red muy grande
para resolver el
problema**

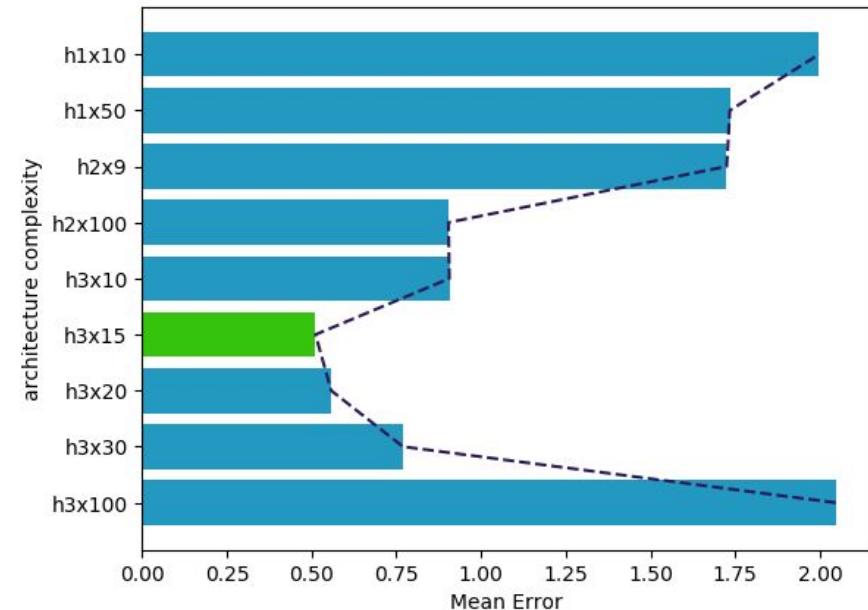
Tamaño de la red



Profundidad de la arquitectura de una red

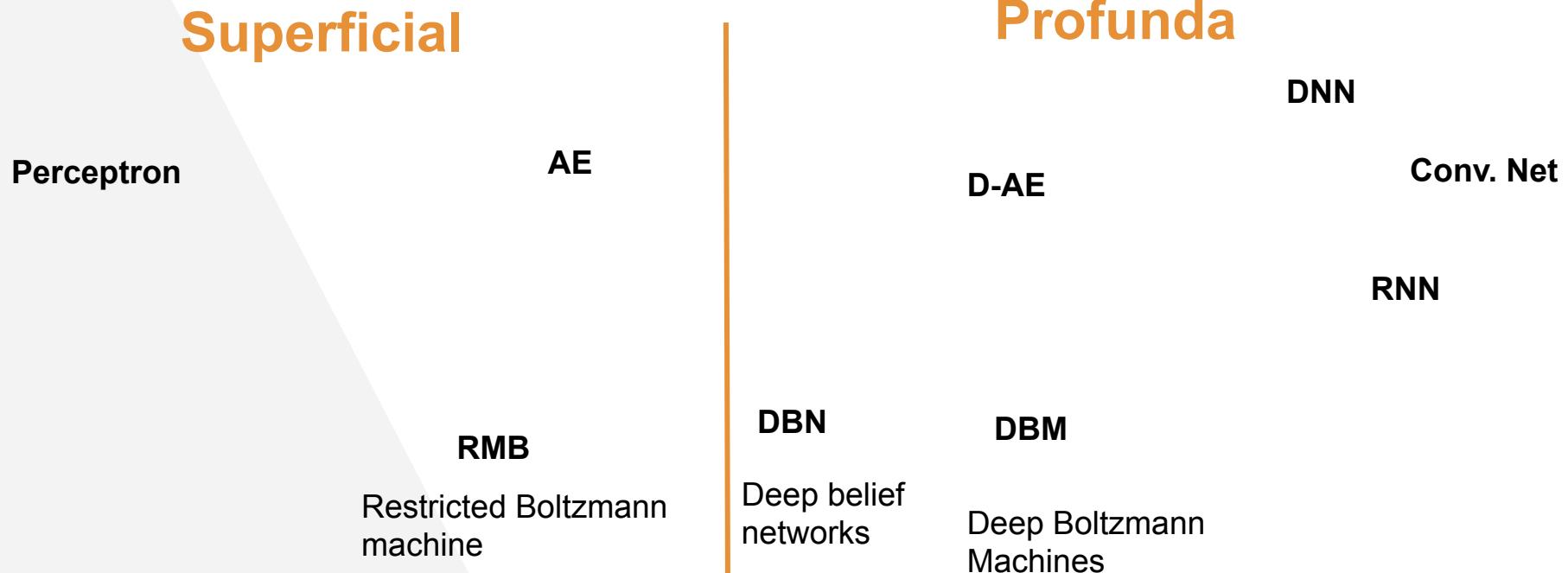


Problema simple

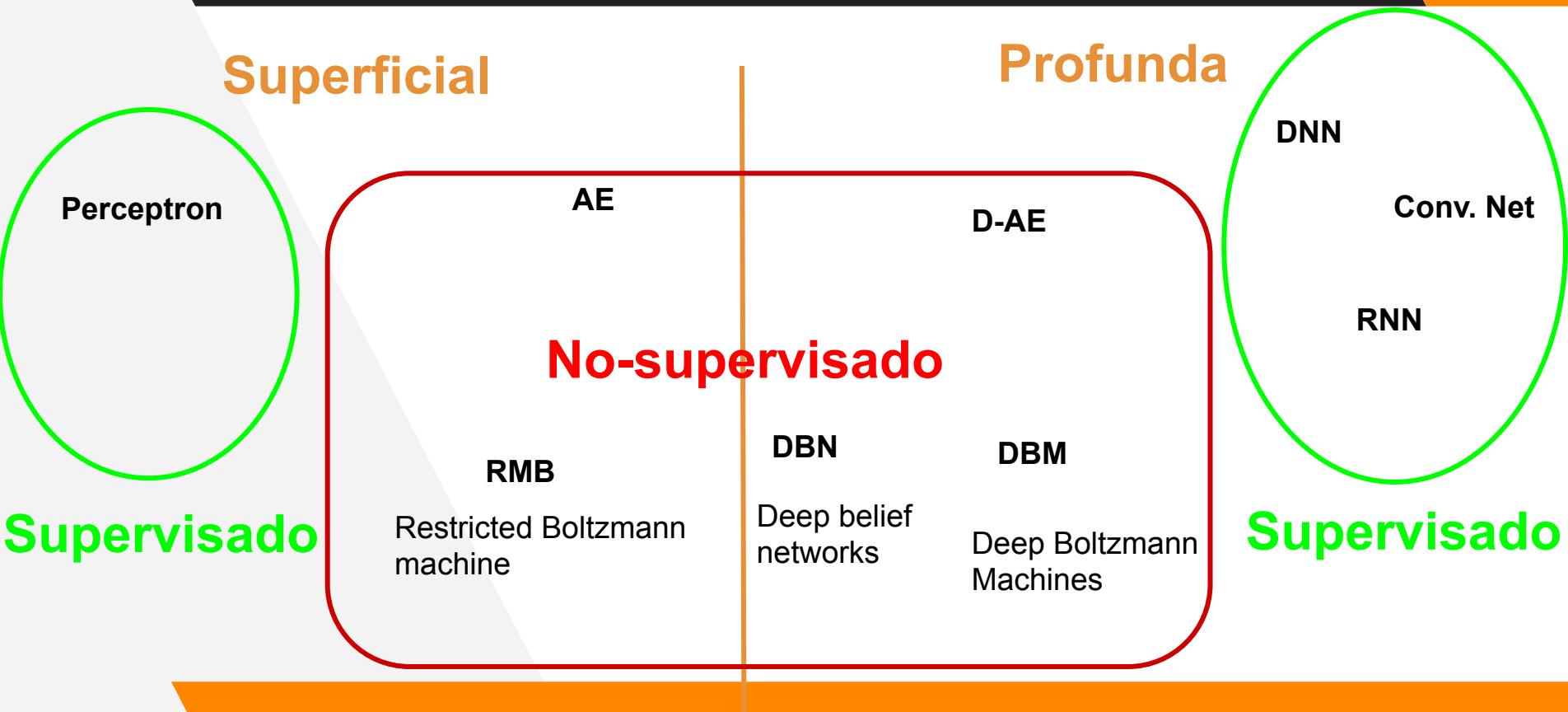


Problema complejo

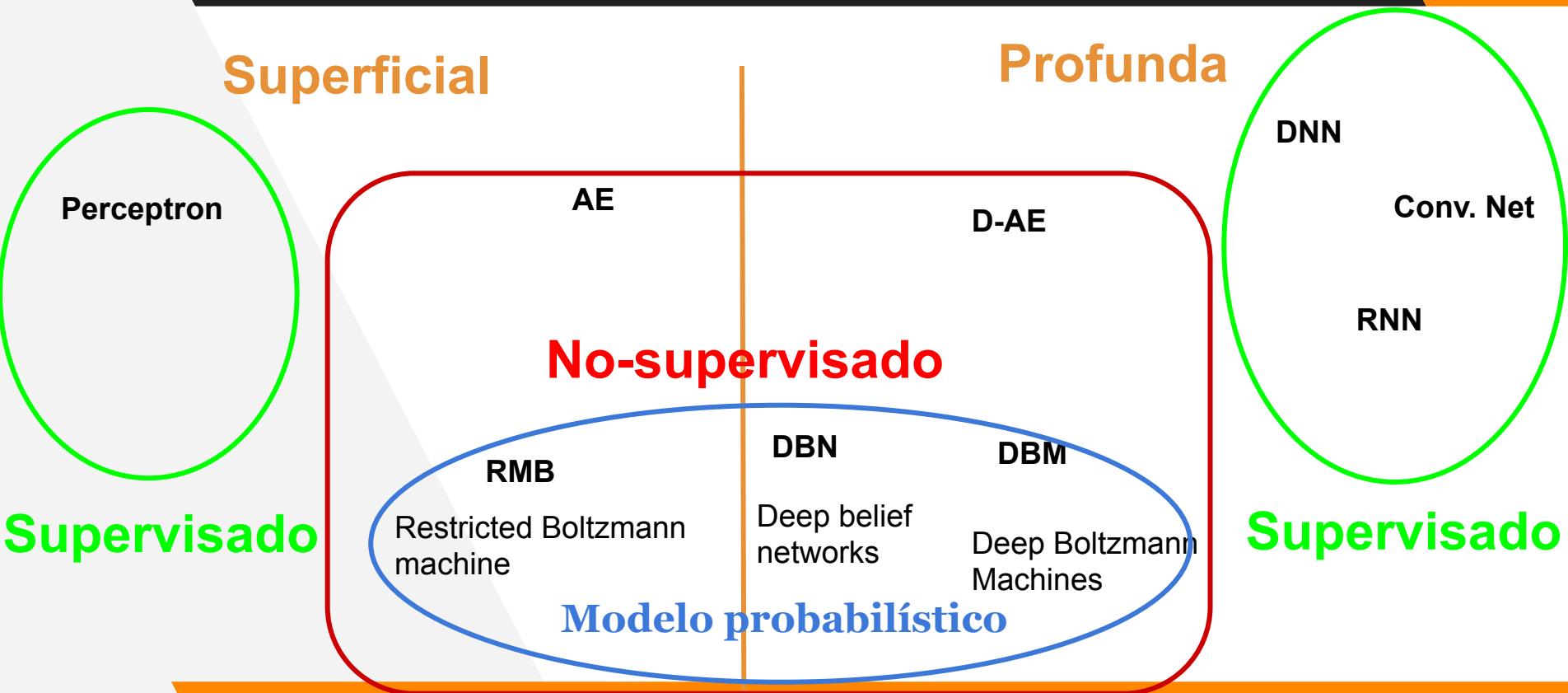
¿Qué arquitecturas profundas existen?



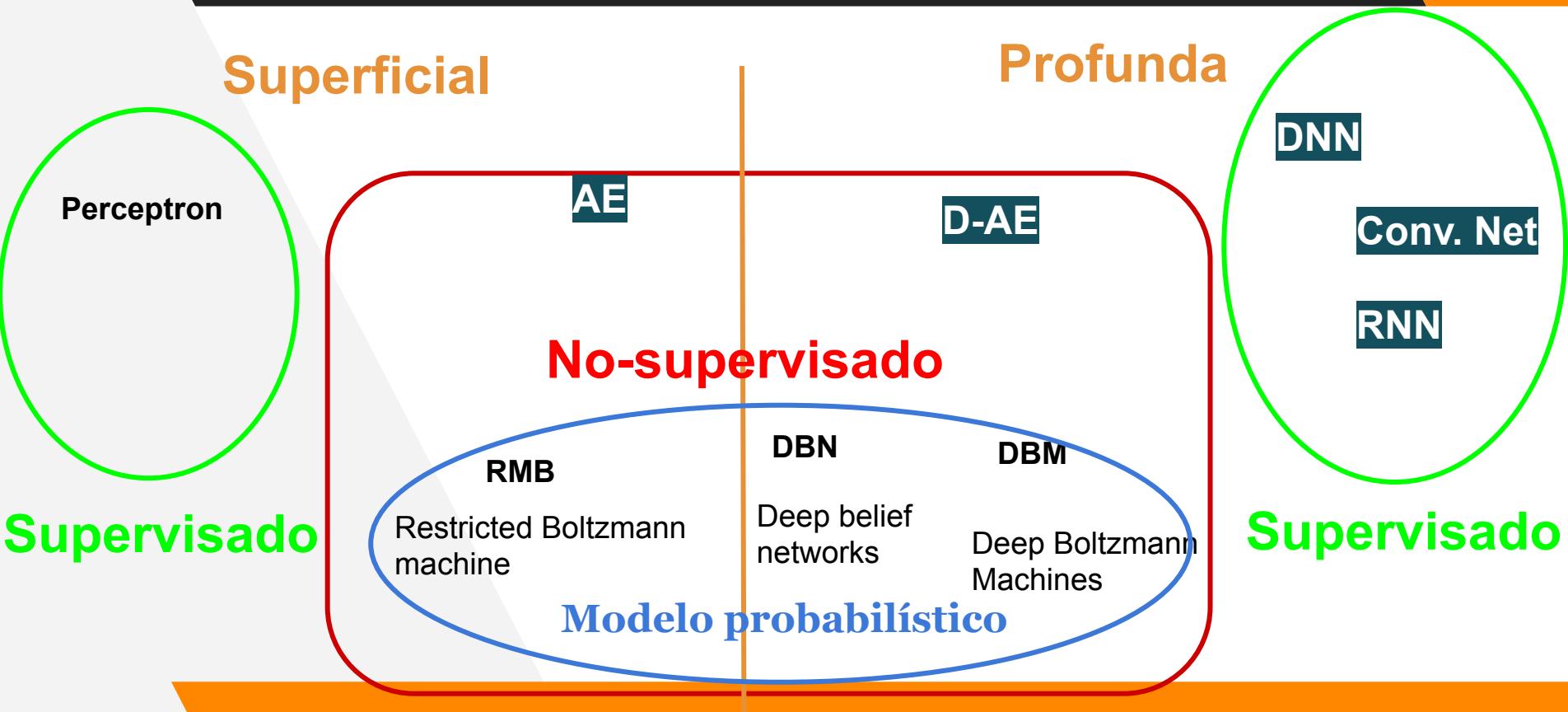
¿Qué arquitecturas profundas existen?



¿Qué arquitecturas profundas existen?



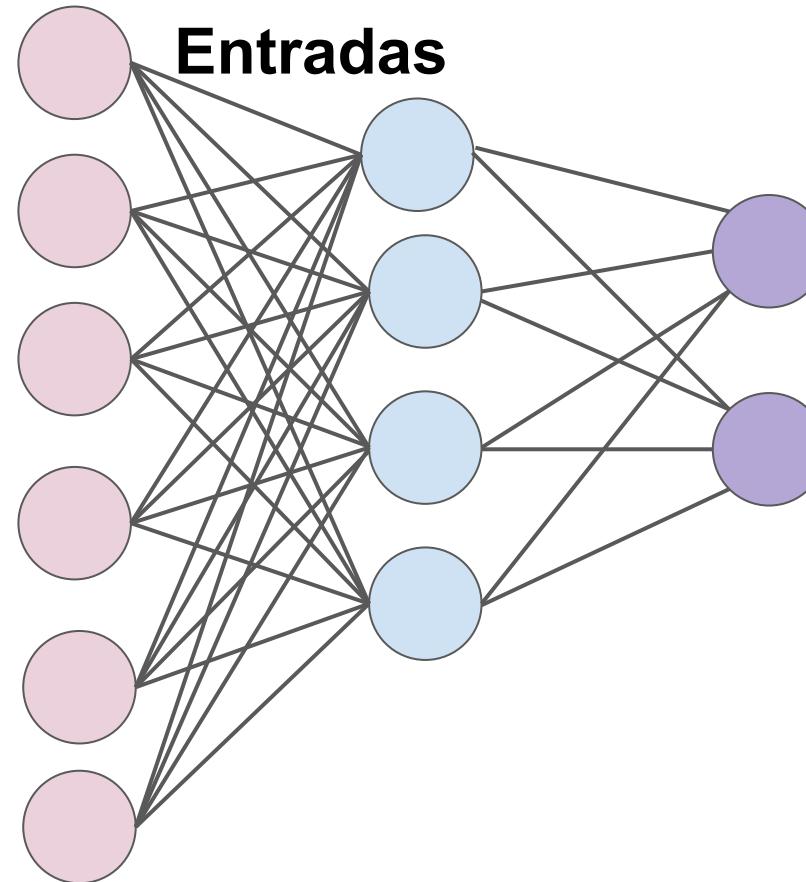
¿Qué arquitecturas profundas existen?



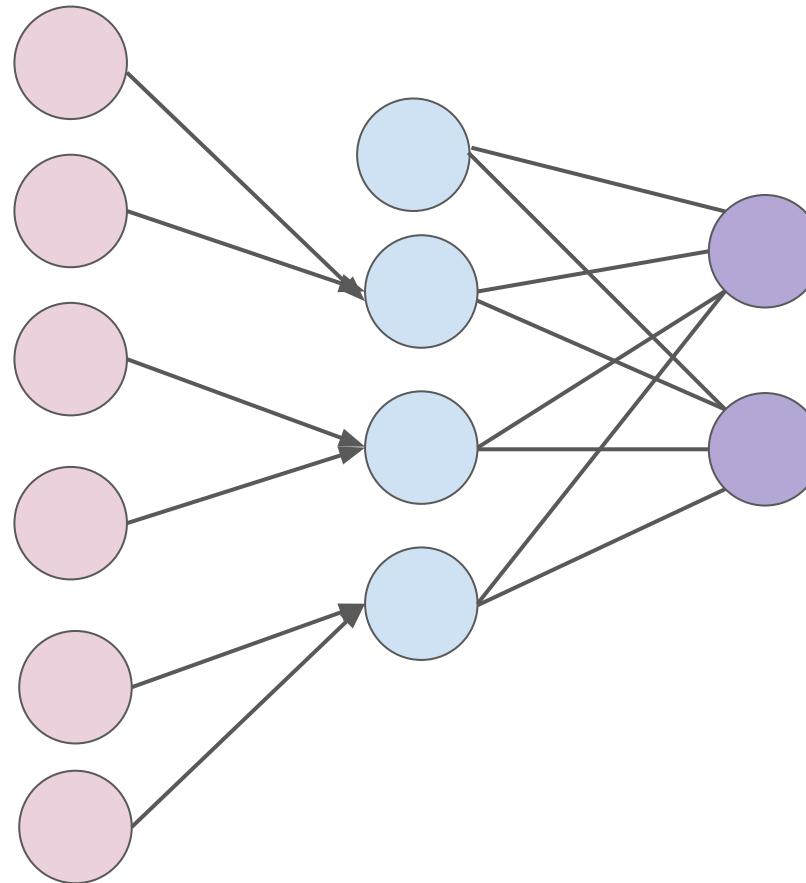
Redes neuronales convolucionales

La investigación de la corteza visual ha revelado que en esta zona las neuronas se conectan en pequeños bloques en donde comparten una pequeña cantidad de información local.

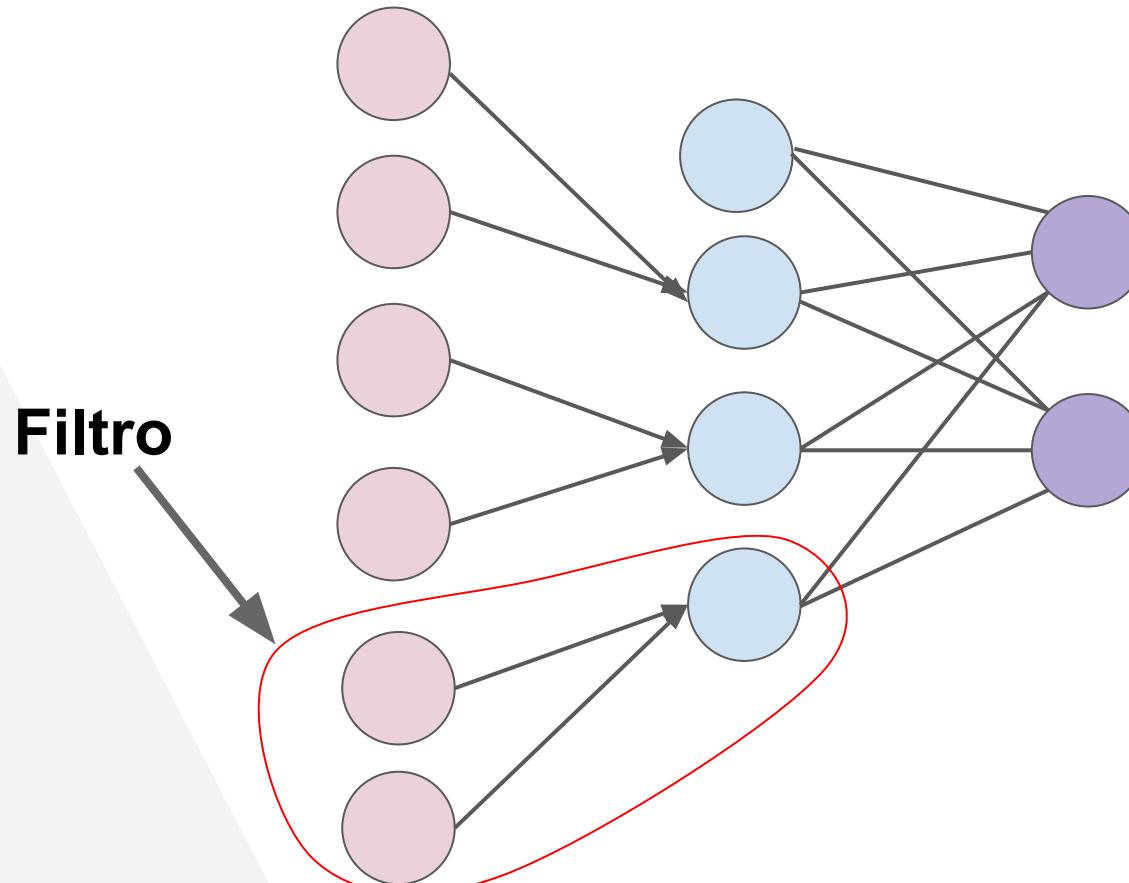
Del perceptrón multicapa a convolucional



Convolucional 1D para señales



Convolucional 1D para señales

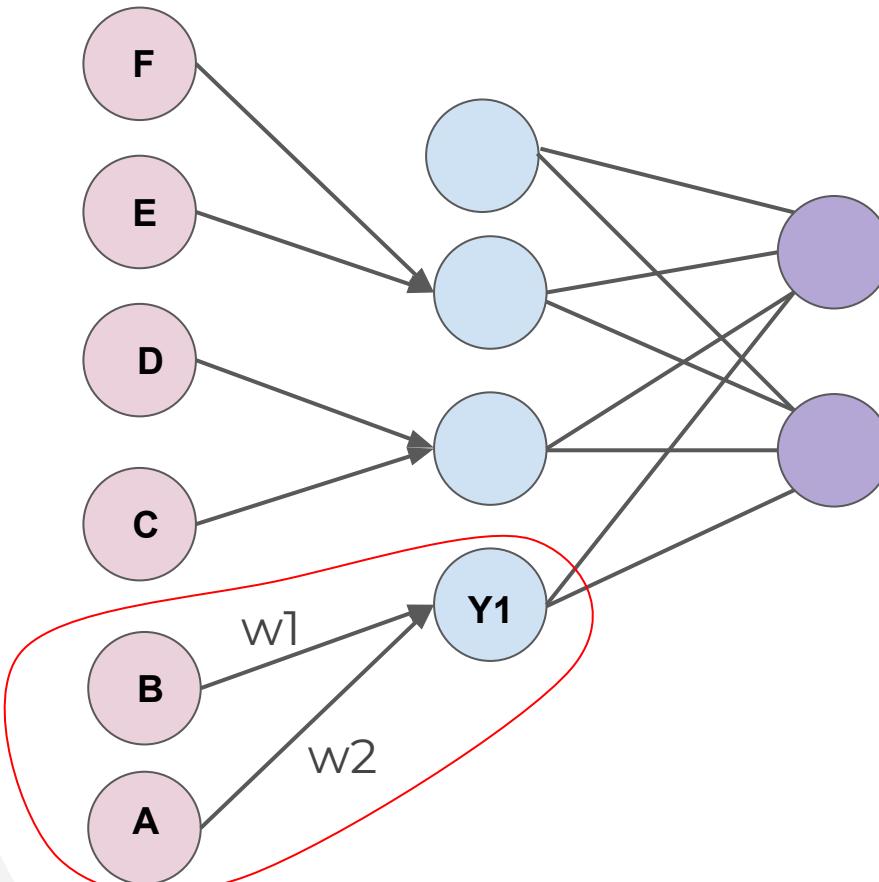


Entradas

Cálculo del filtro

$$Y_1 = W_2 \cdot A + W_1 \cdot B$$

$$\text{abs}(y1(A, B)) \in (1, 0)$$

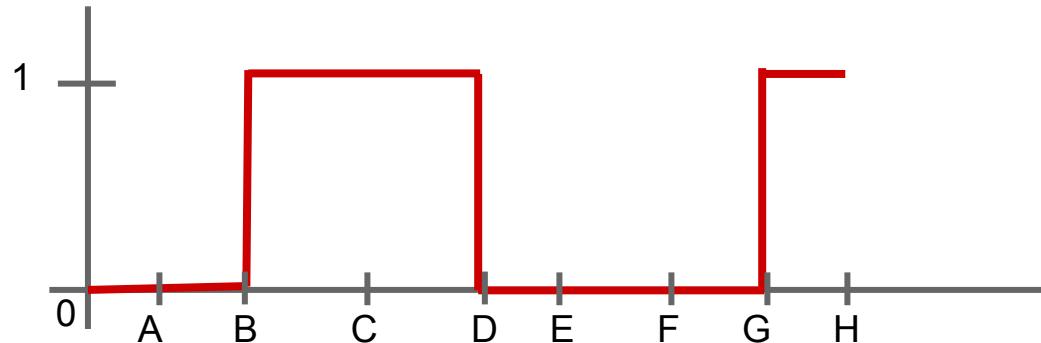


Filtro detector de transiciones

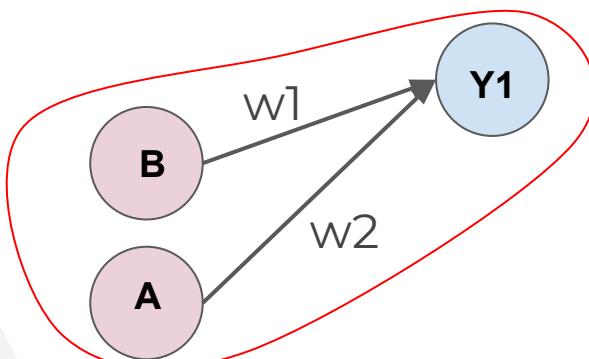
Cálculo del filtro

$$Y_1 = W_2 \cdot A + W_1 \cdot B$$

$$\text{abs}(y_1(A, B)) \in (1, 0)$$



$$(w_1, w_2) = (1, -1)$$

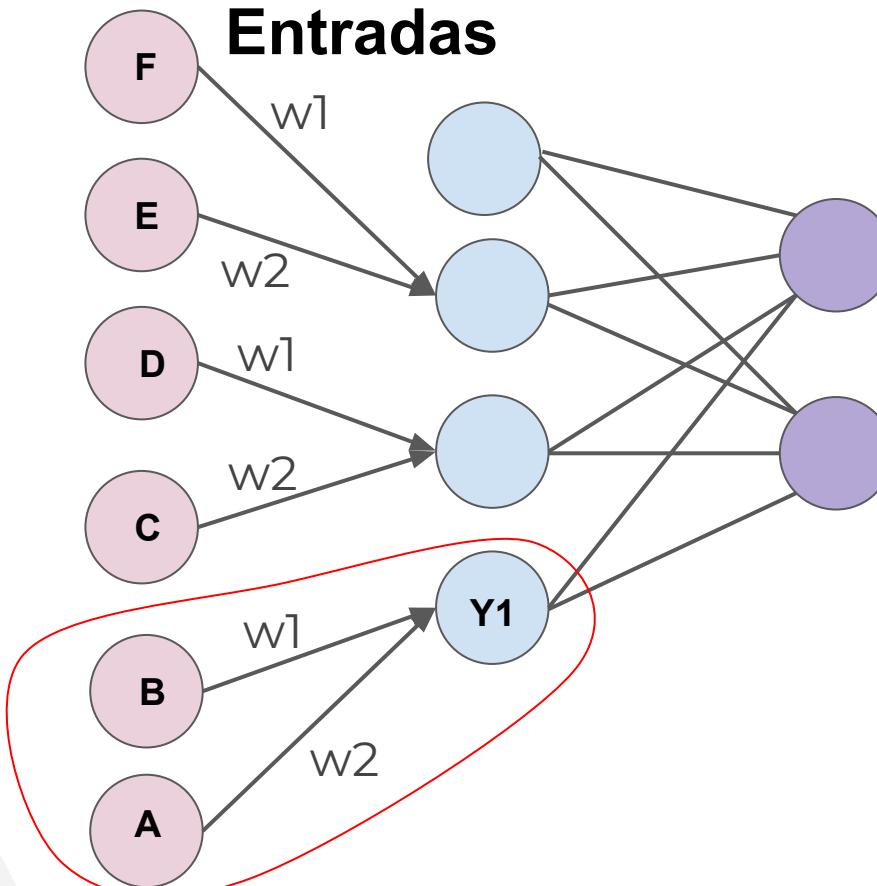


Filtro detector de transiciones

Cálculo del filtro

$$Y_1 = W_2 \cdot A + W_1 \cdot B$$

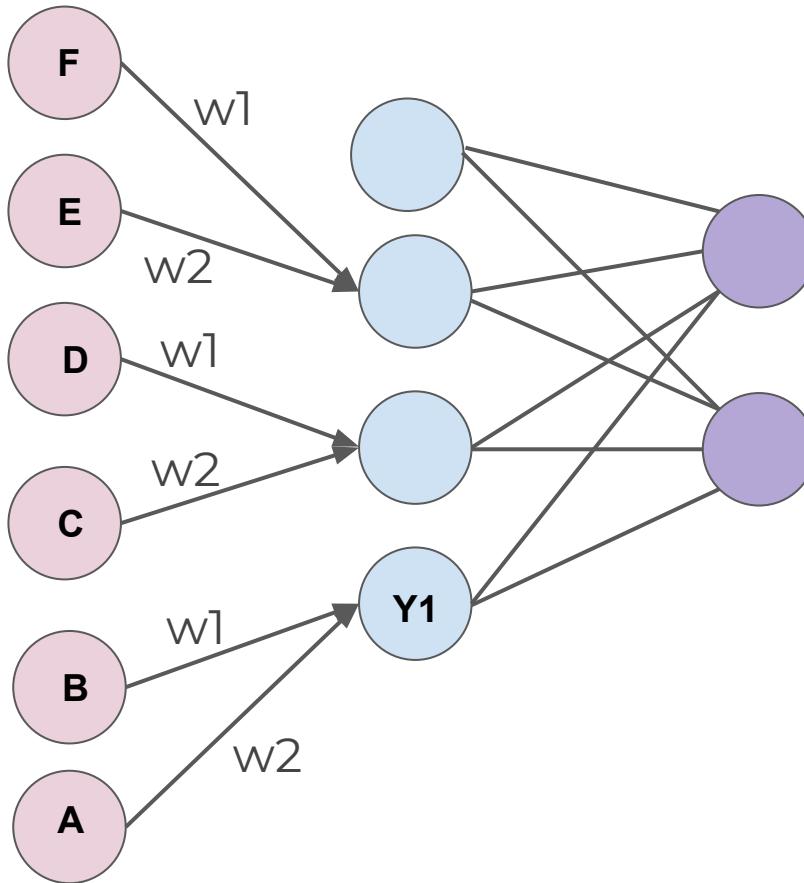
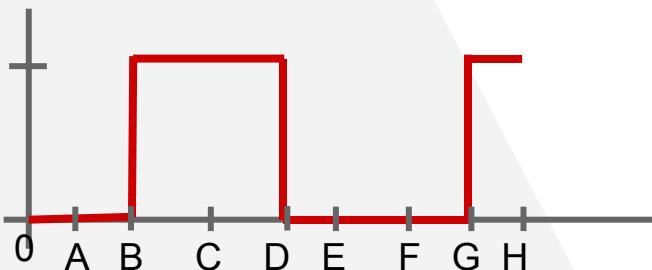
$$\text{abs}(y1(A, B)) \in (1, 0)$$



Entradas

Tipo de Filtro

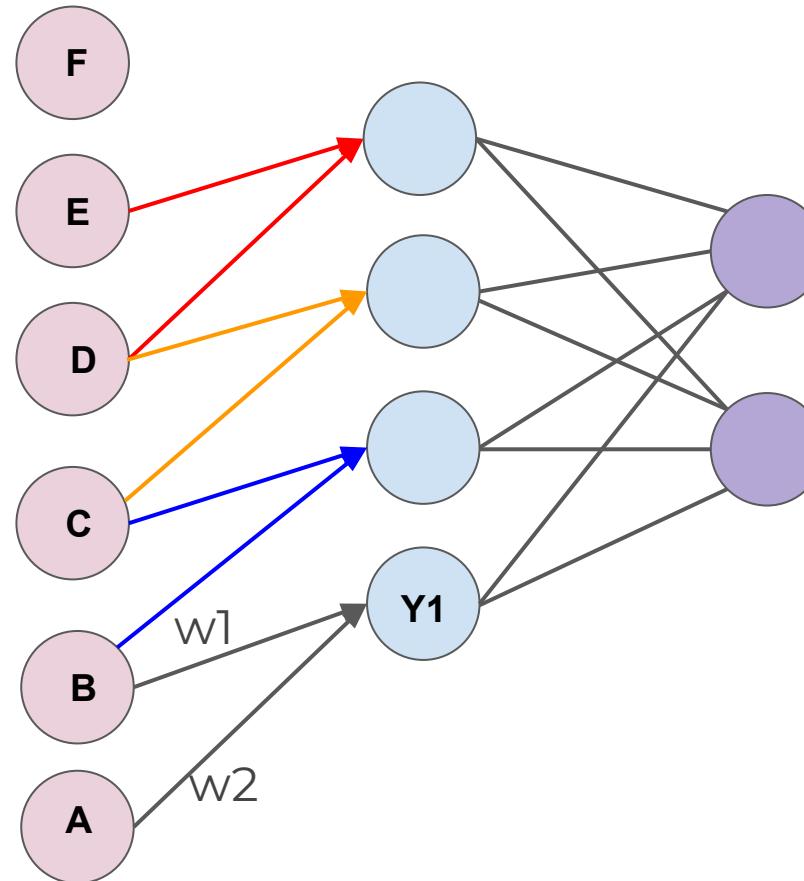
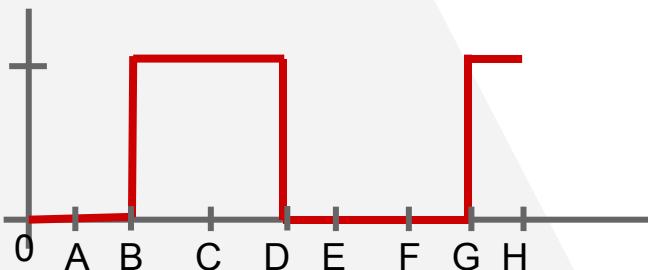
- ▶ Número de filtros = 1
- ▶ Tamaño del filtro = 2
- ▶ Paso del filtro = 2



Entradas

Tipo de Filtro

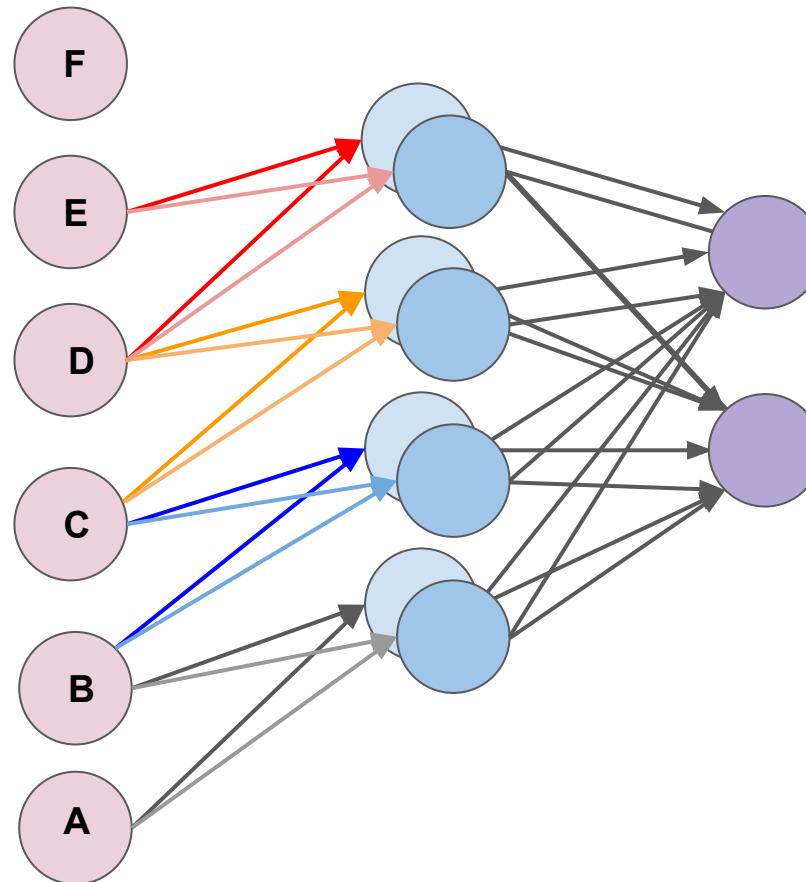
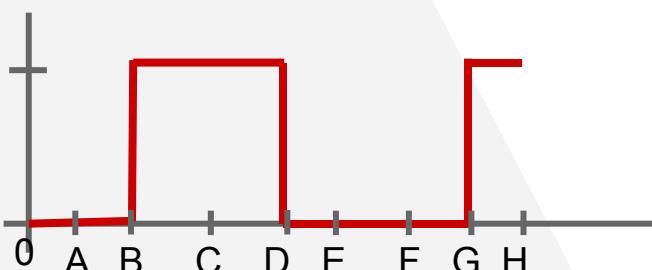
- ▶ Número de filtros = 1
- ▶ Tamaño del filtro = 2
- ▶ Paso del filtro = 1



Entradas

Tipo de Filtro

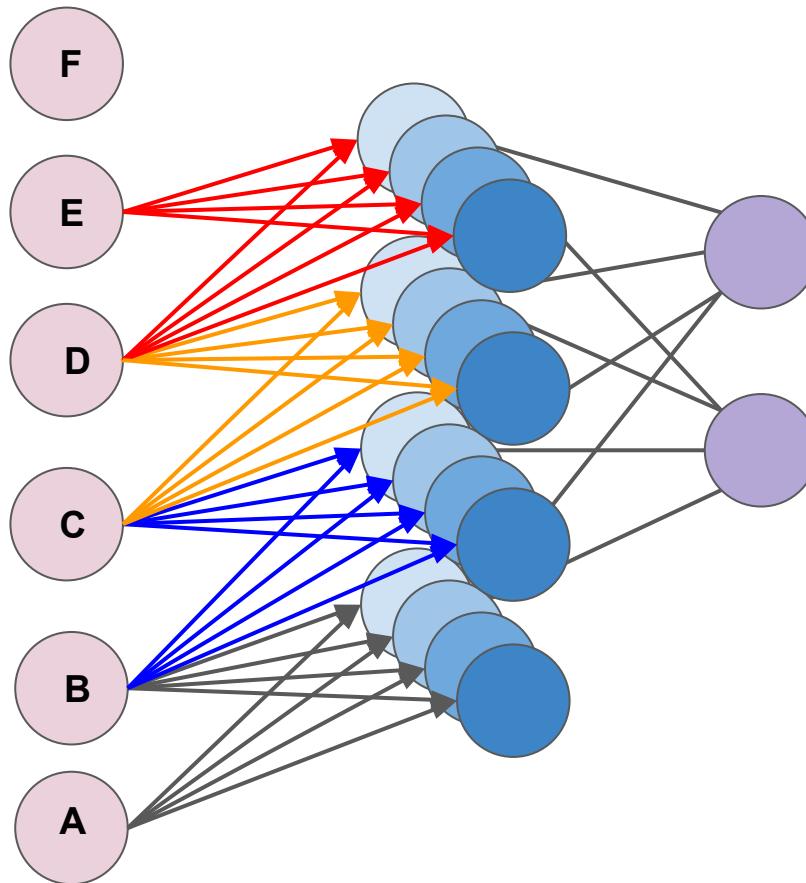
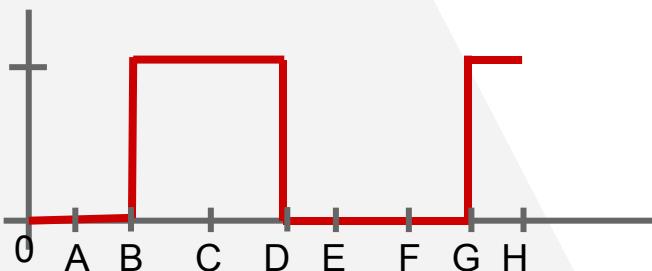
- ▶ Número de filtros = 2
- ▶ Tamaño del filtro = 2
- ▶ Paso del filtro = 1



Entradas

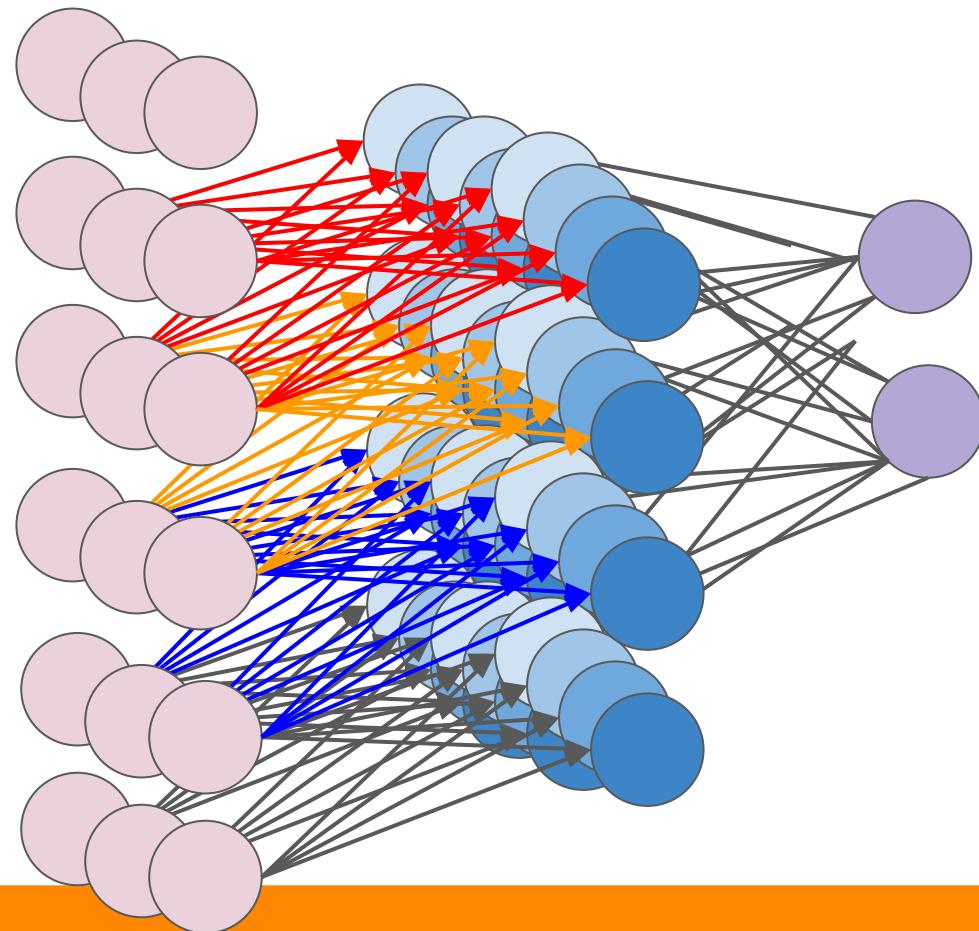
Tipo de Filtro

- ▶ Número de filtros = 4
- ▶ Tamaño del filtro = 2
- ▶ Paso del filtro = 1



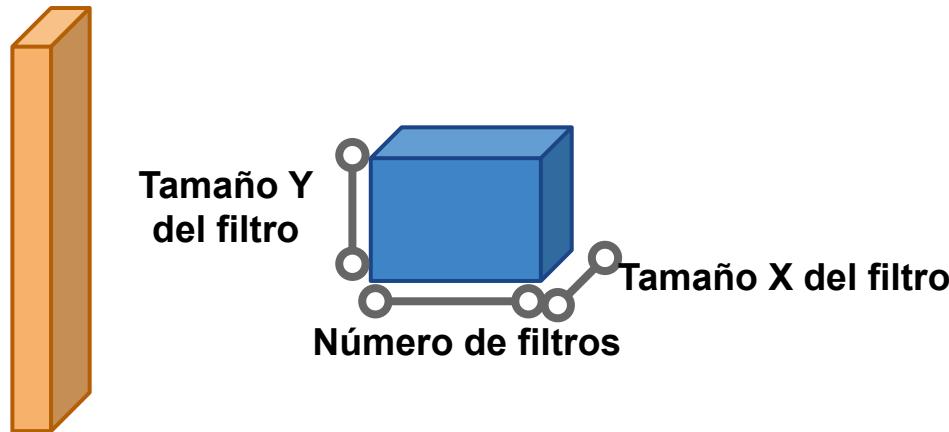
Tipo de Filtro

- ▶ Número de filtros = 4
- ▶ Tamaño del filtro = 2x3
- ▶ Paso del filtro = 1



Cambio de nomenclatura

Imagen



Filtros convolucionales en imágenes

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

0	0	0
0	-1	1
0	1	-1

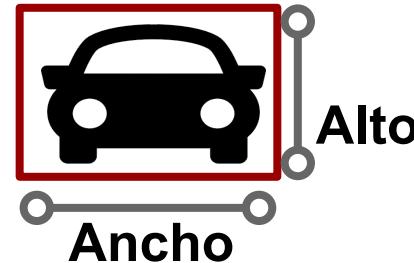
3 x 3 Filtro

0x0	0x0	0x0
0x0	1x-1	1x1
0x0	1x1	1x-1

Multiplicación de los pesos

0	0	0
0	-1	1
0	1	-1

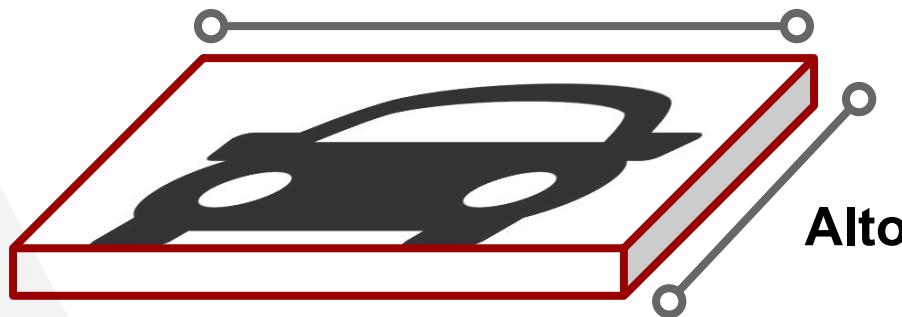
Se suma el resultado



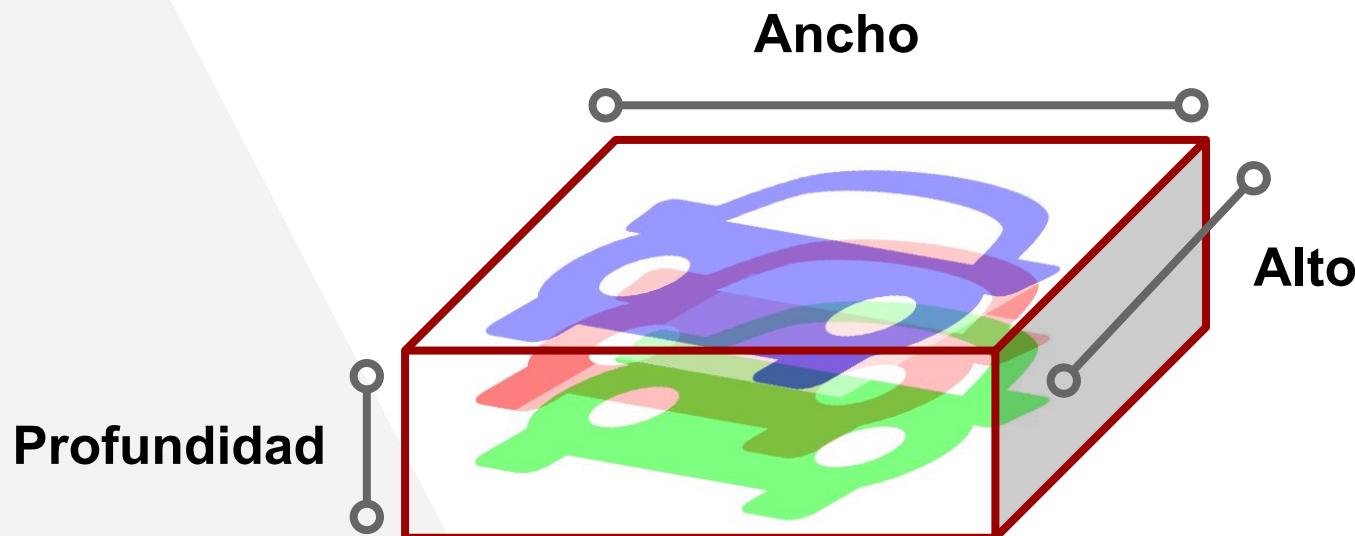
<http://setosa.io/ev/image-kernels/>

Analizando una imagen

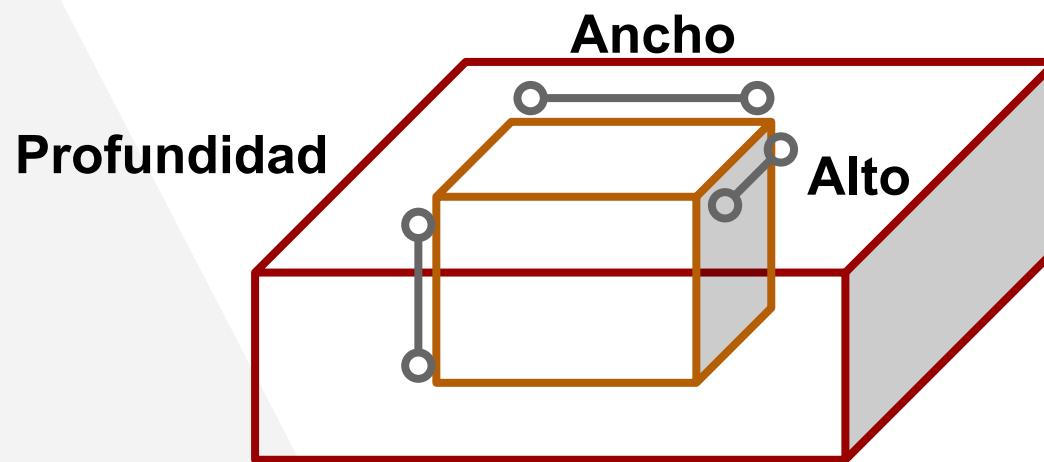
Ancho



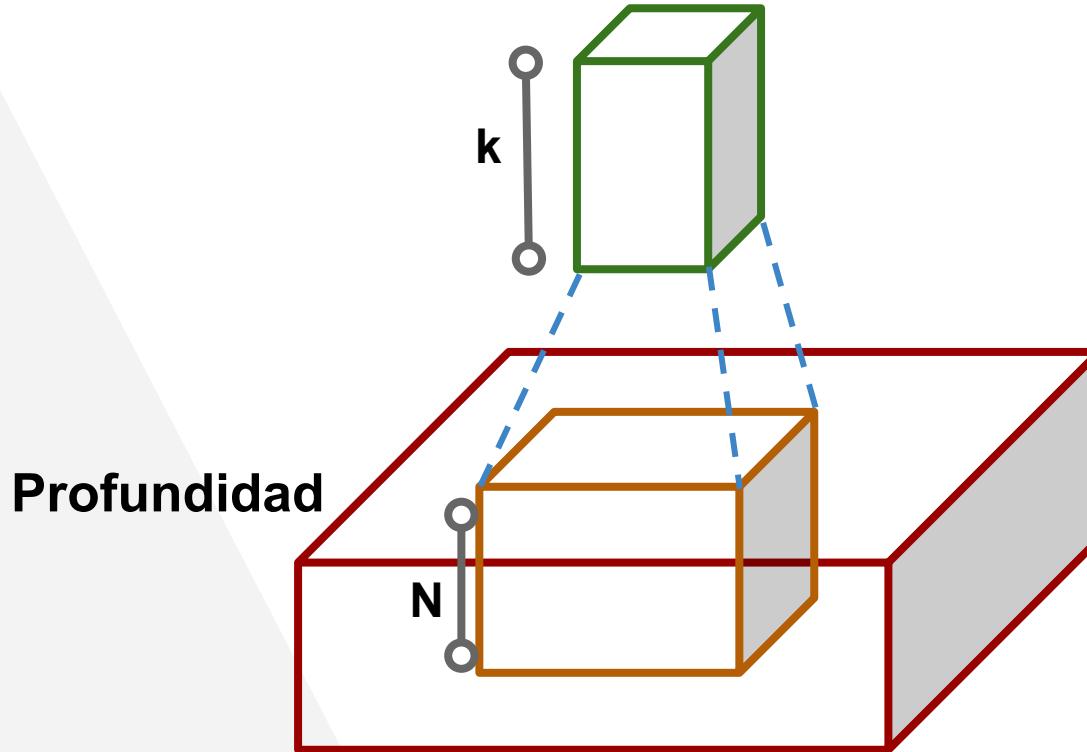
Profundidad de la imagen



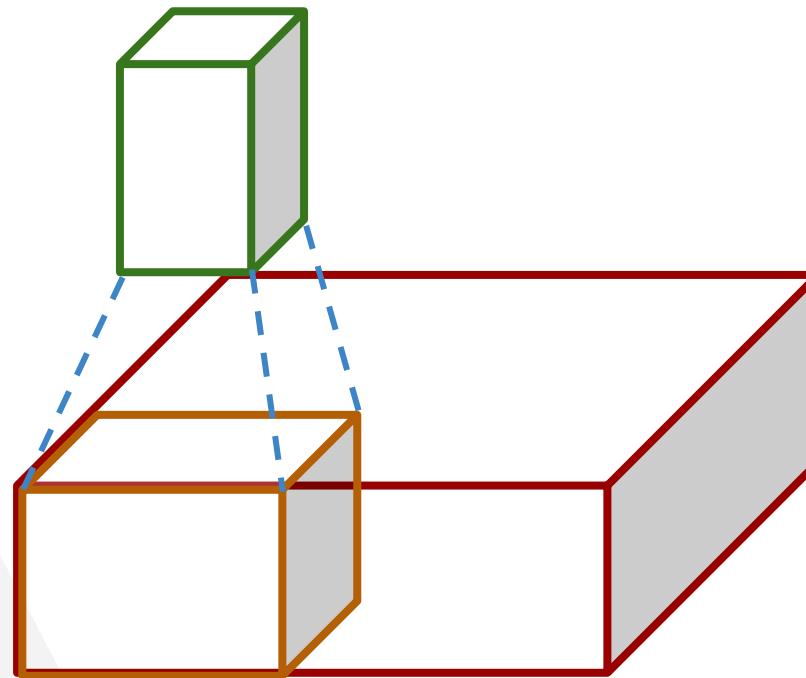
Declarando el kernel



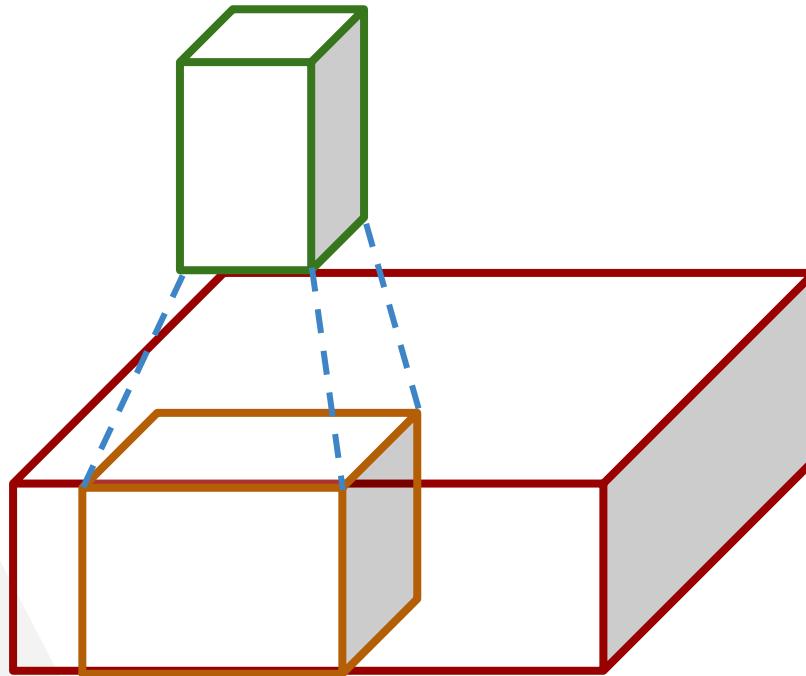
Red convolucional barriendo la imagen



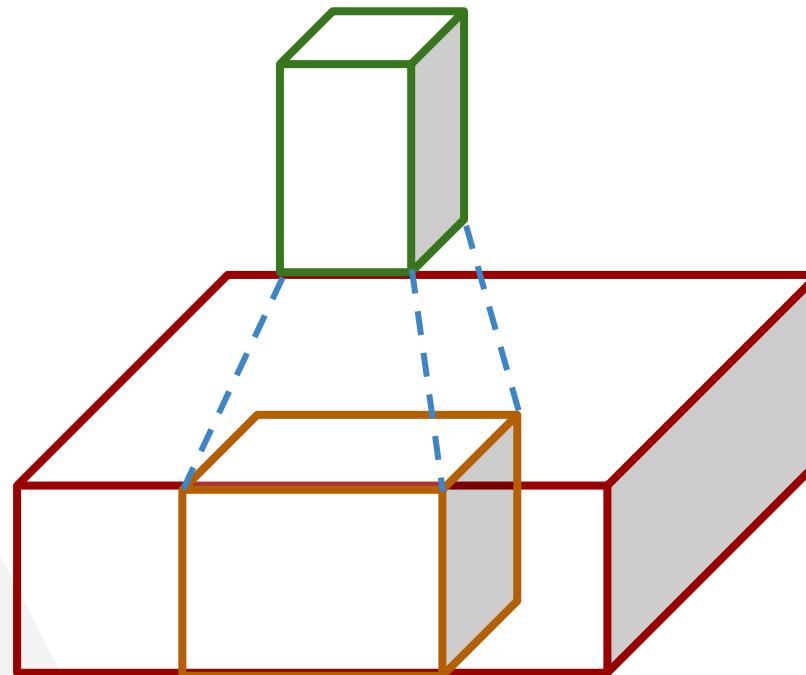
Red convolucional barriendo la imagen



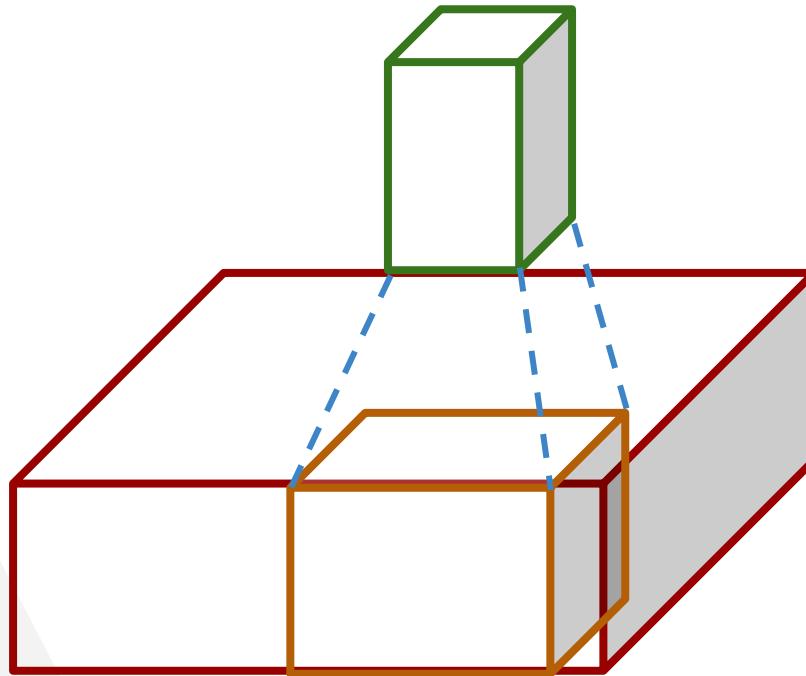
Red convolucional barriendo la imagen



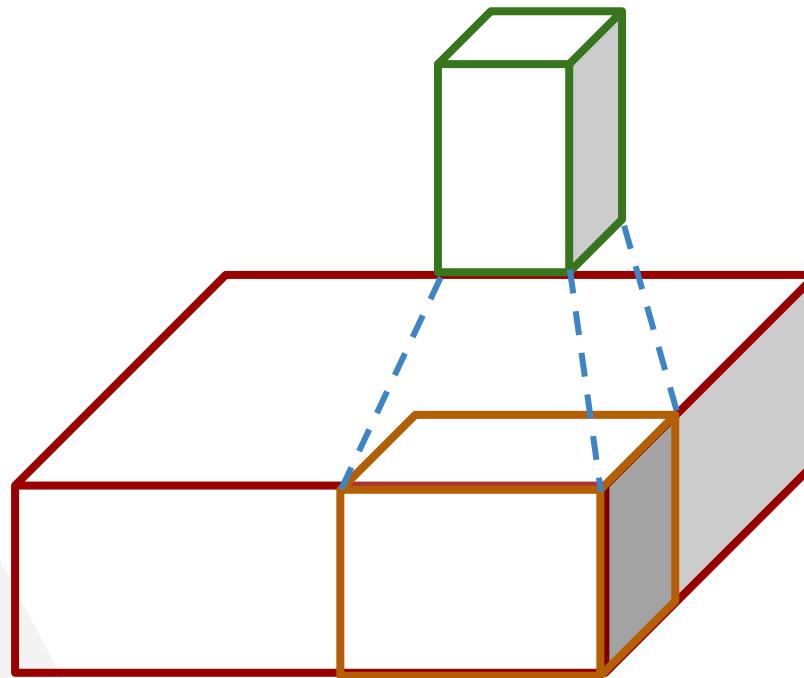
Red convolucional barriendo la imagen



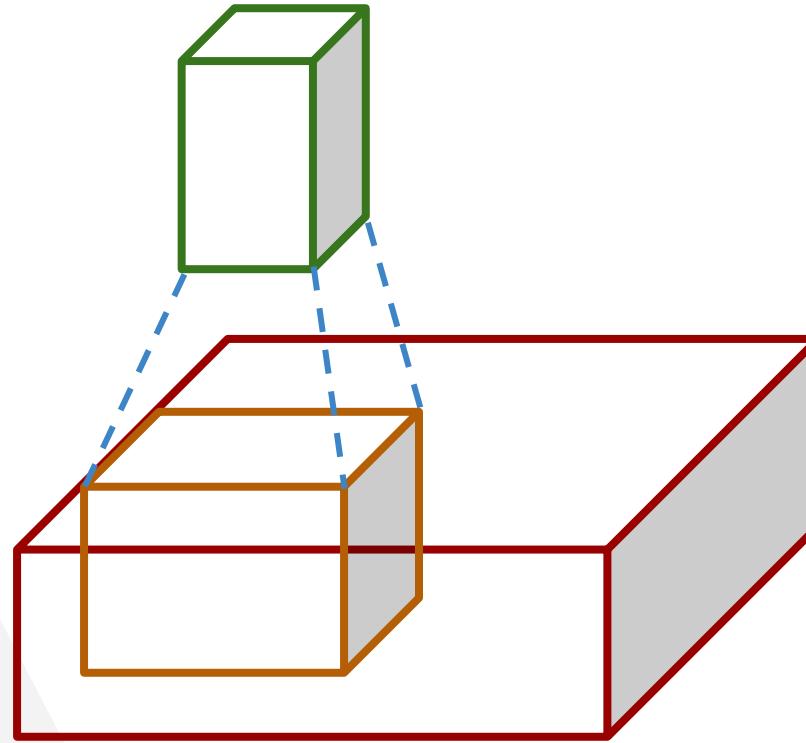
Red convolucional barriendo la imagen



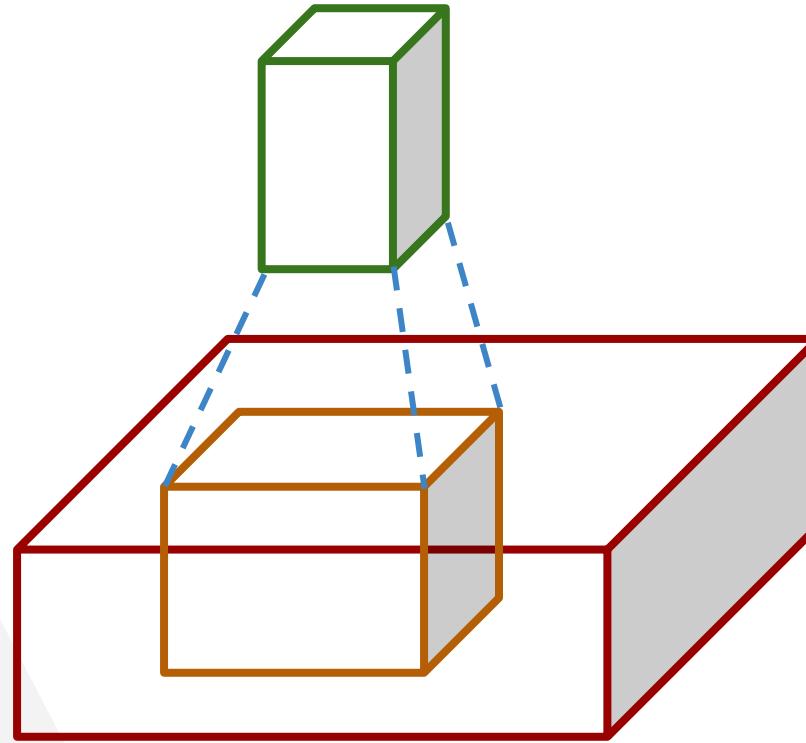
Red convolucional barriendo la imagen



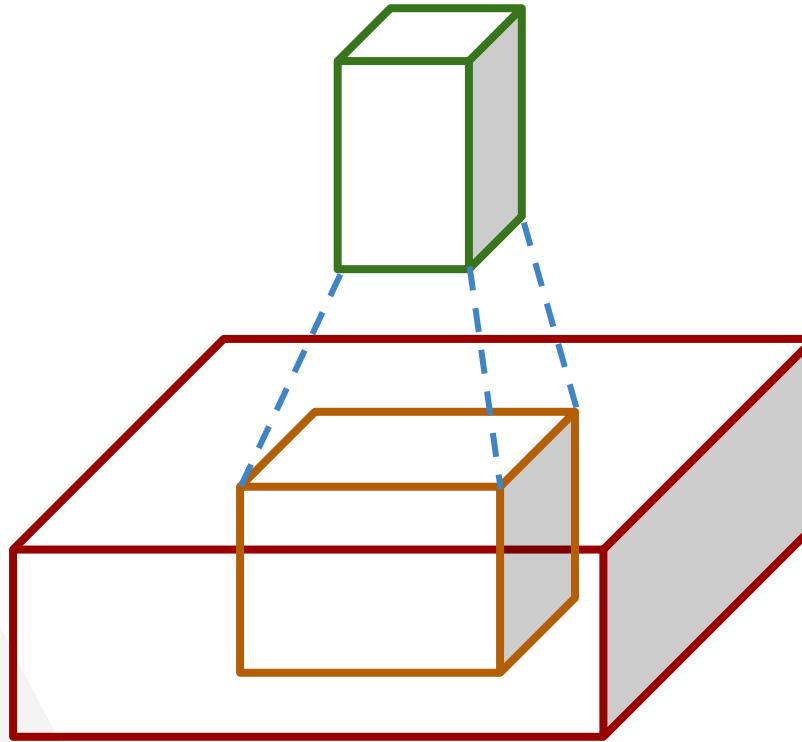
Red convolucional barriendo la imagen



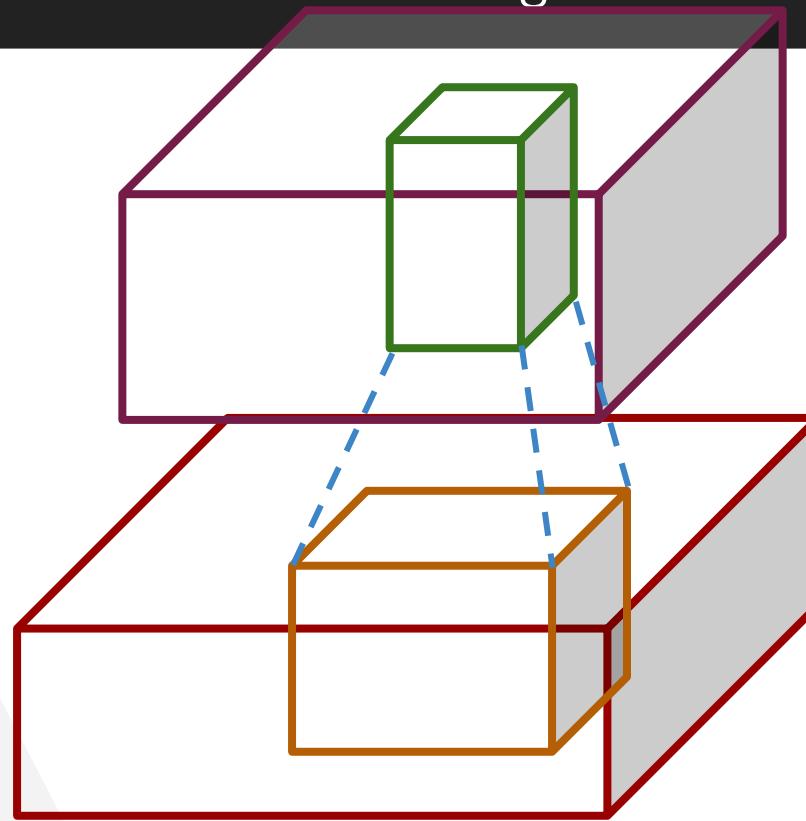
Red convolucional barriendo la imagen



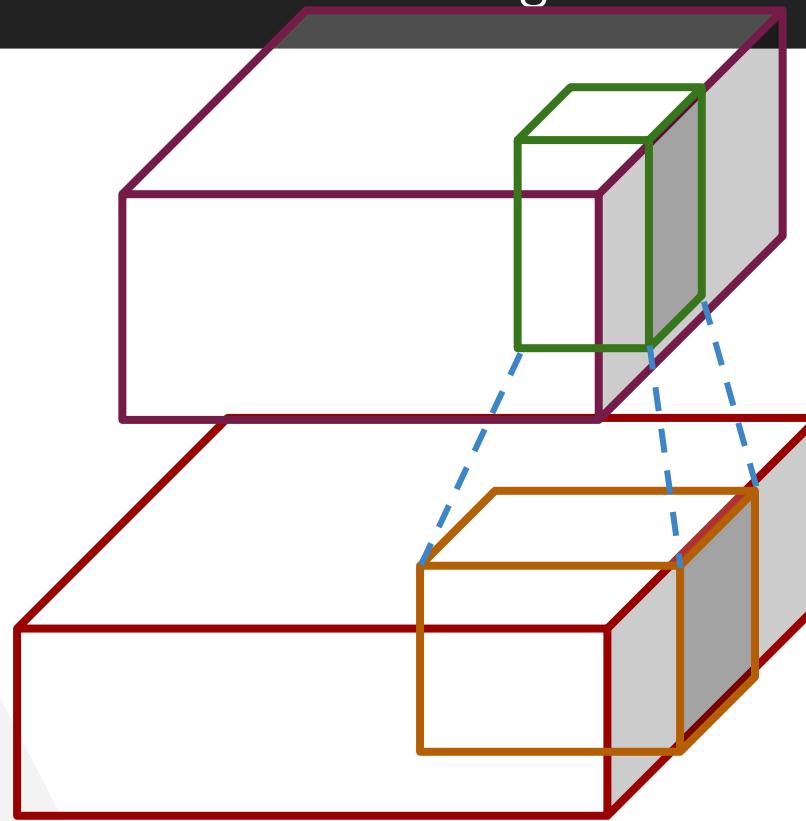
Red convolucional barriendo la imagen



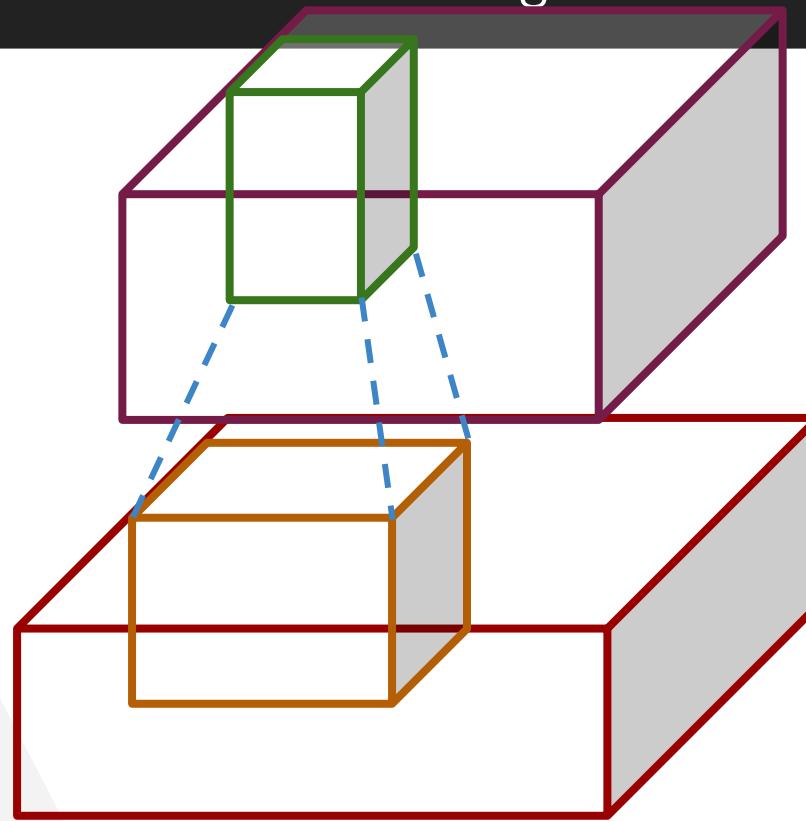
Red convolucional barriendo la imagen



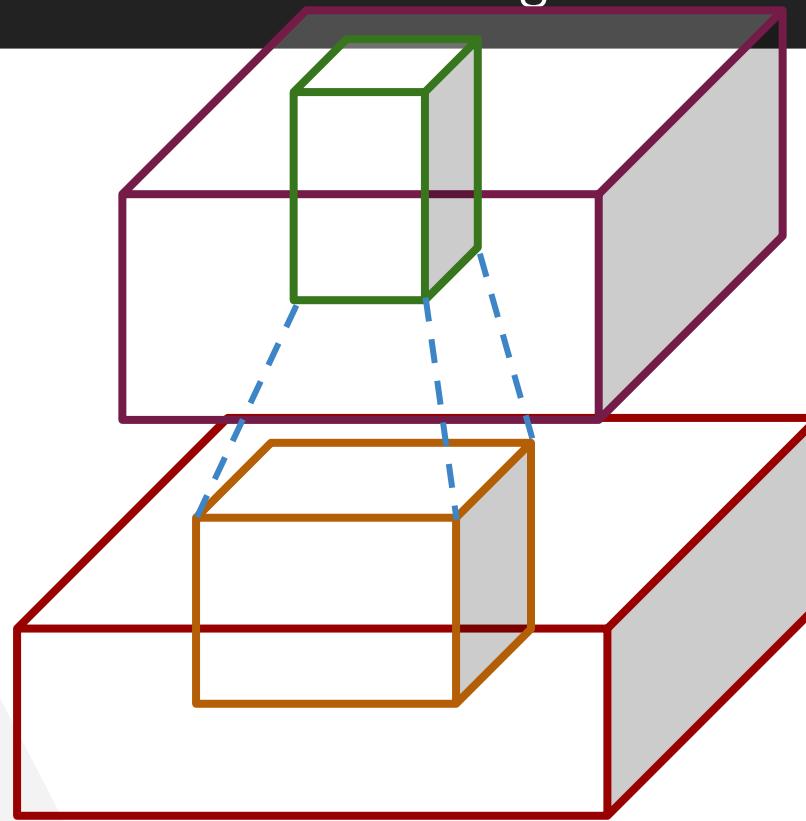
Red convolucional barriendo la imagen



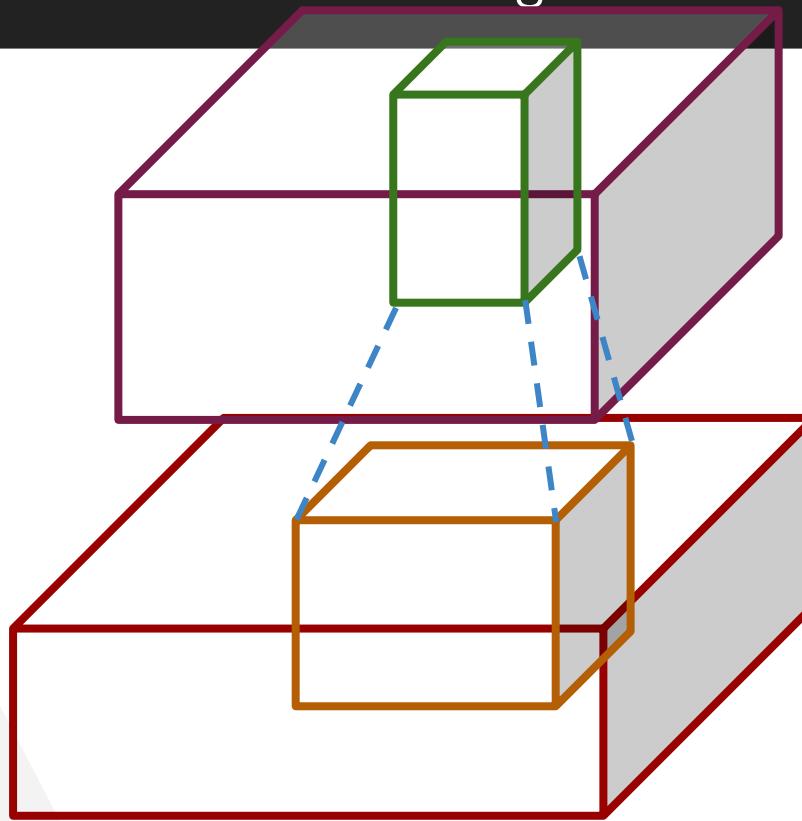
Red convolucional barriendo la imagen



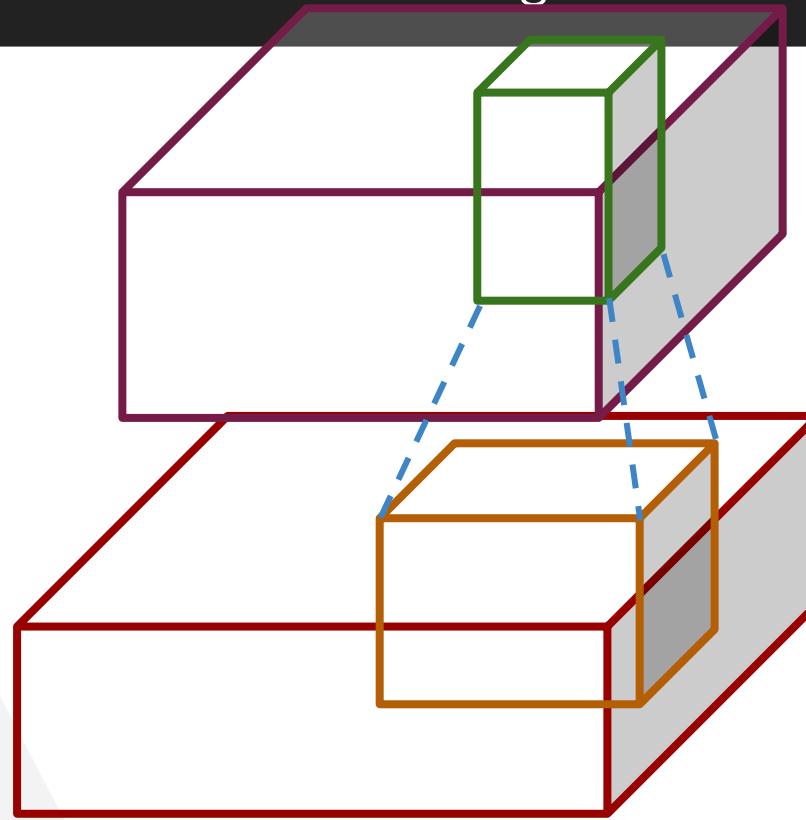
Red convolucional barriendo la imagen



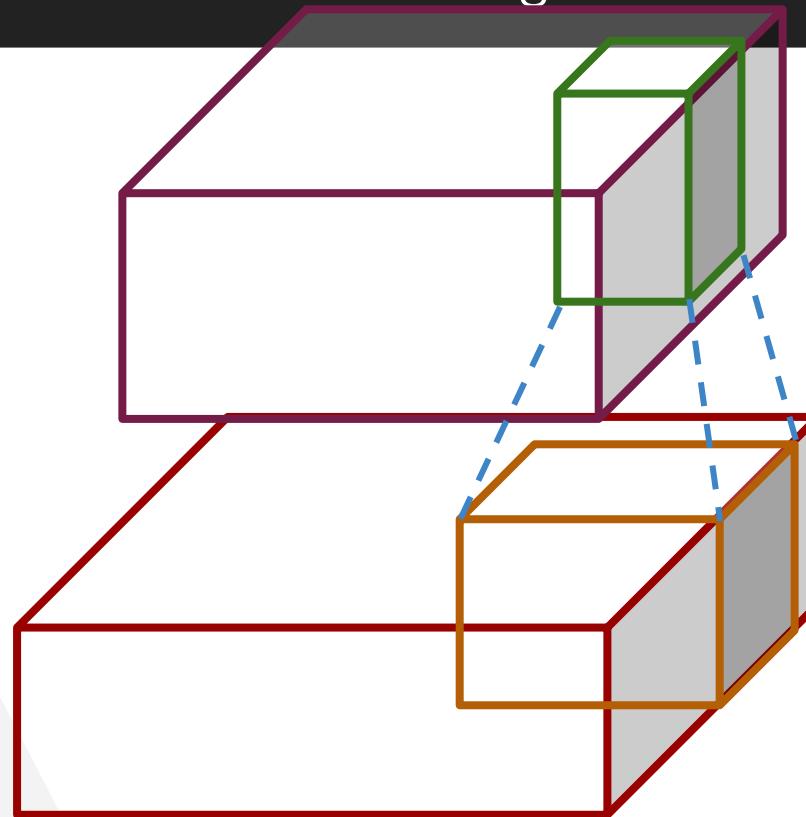
Red convolucional barriendo la imagen



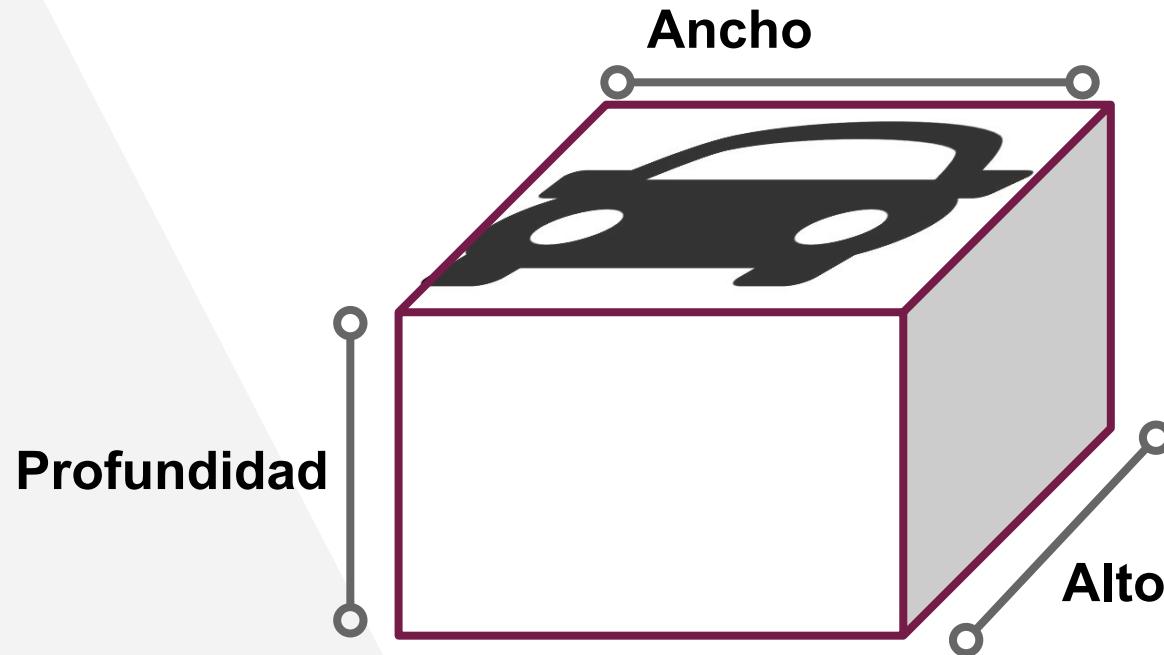
Red convolucional barriendo la imagen



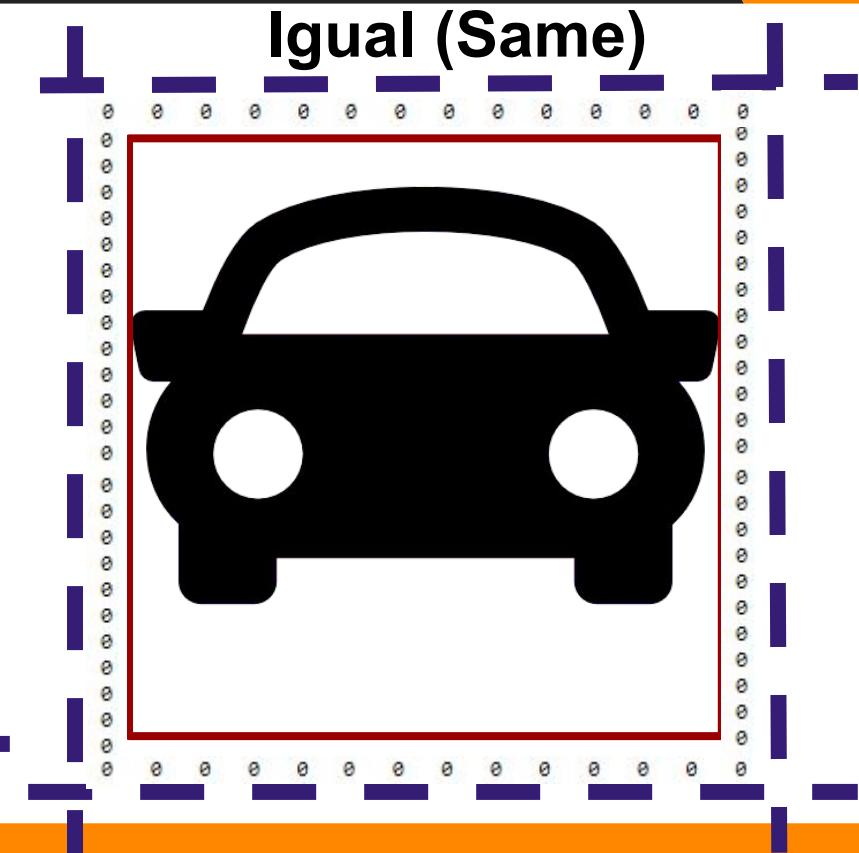
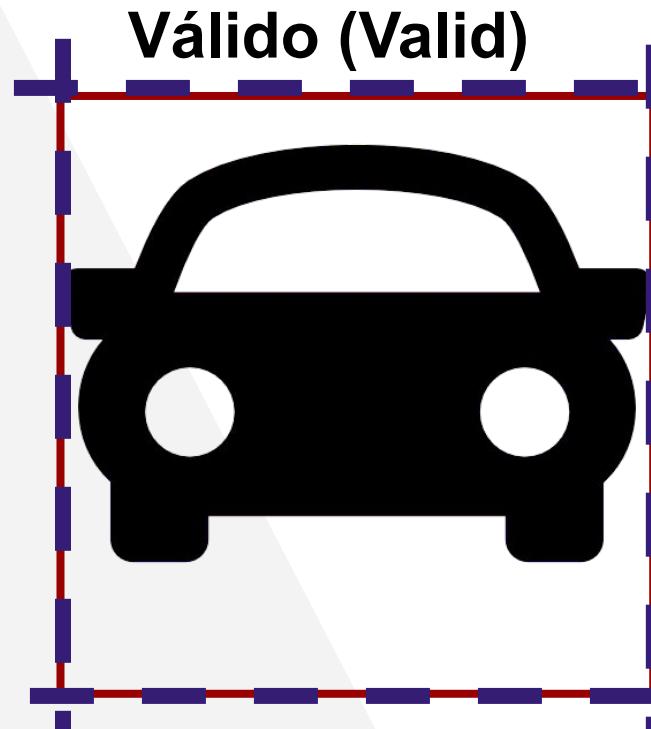
Red convolucional barriendo la imagen



Salida de la convolucional



Relleno de una imagen (padding)

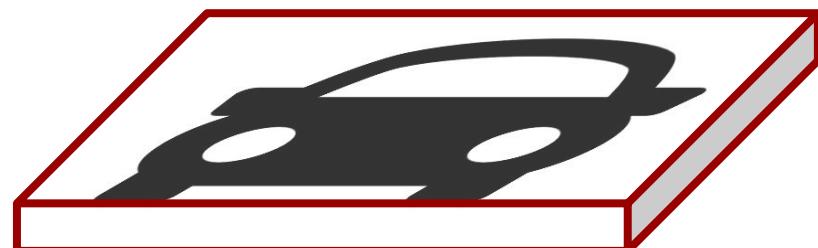
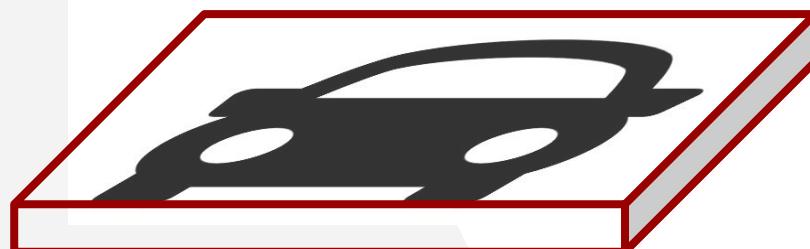


Relleno de una imagen (padding)

Válido (Valid)



Igual (Same)



Relleno de una imagen (padding) kernel 3X3

Válido (Valid)

.7	.8	.9	.5	.4	.2
.8	.9	.5	.4	.2	.2
1	1	1	1	.2	.3
1	0	0	1	.3	.4
1	0	0	1	.4	.4
1	1	1	1	.4	.1

Igual (Same)

0	.8	.9	.5	.4	.2
0	1	1	1	1	.2
0	1	0	0	1	.3
0	1	0	0	1	.4
0	1	1	1	1	.4
0	0	0	0	0	0

Relleno de una imagen (padding) kernel 5X5

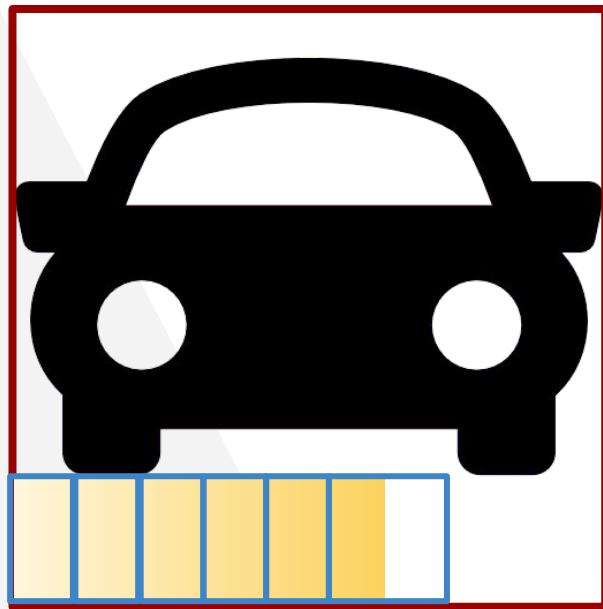
Válido (Valid)

.7	.8	.9	.5	.4	.2
.8	.9	.5	.4	.2	.2
1	1	1	1	.2	.3
1	0	0	1	.3	.4
1	0	0	1	.4	.4
1	1	1	1	.4	.1

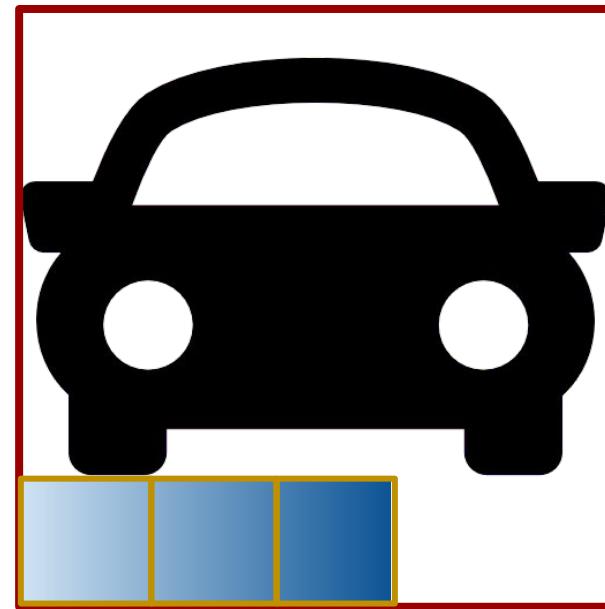
Igual (Same)

0	0	1	1	1	1
0	0	1	0	0	1
0	0	1	0	0	1
0	0	1	1	1	1
0	0	0	0	0	0
0	0	0	0	0	0

Avance de una imagen (stride)

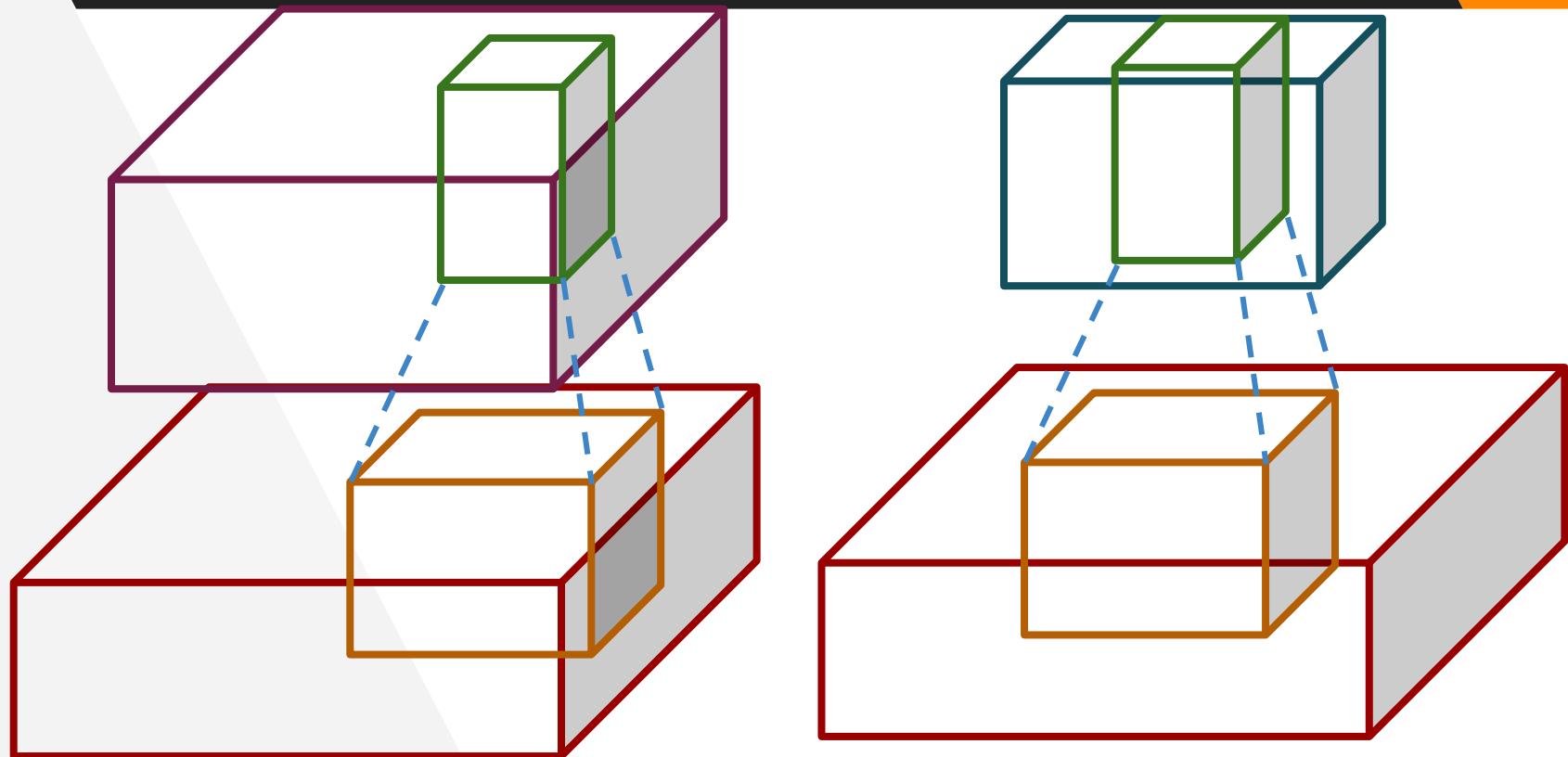


Stride 1

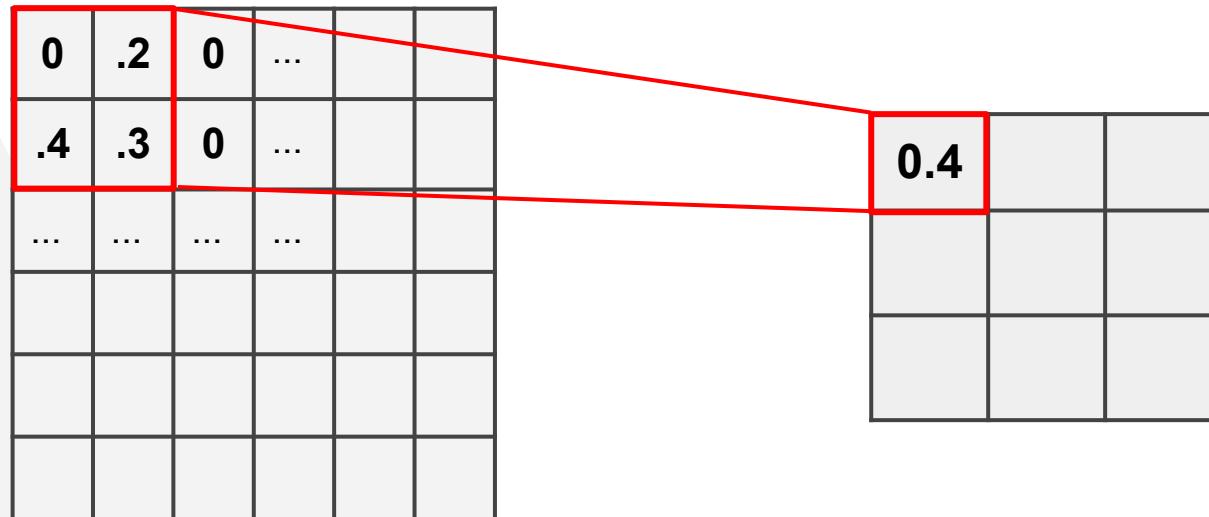


Stride 2

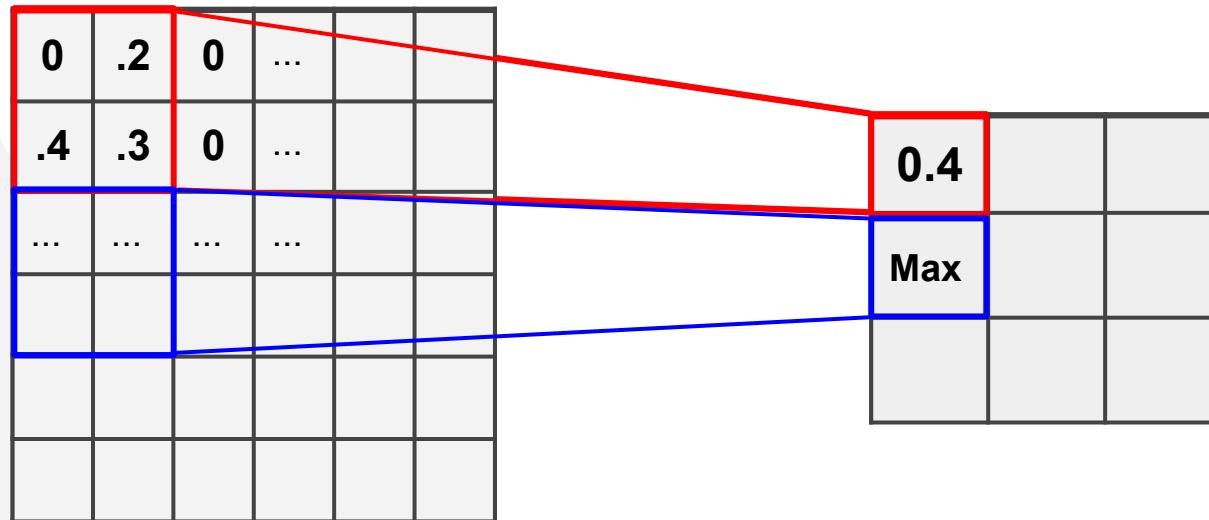
Tamaño de la imagen resultante



Reducción dimensional: Max Pooling



Reducción dimensional: Max Pooling



Conectando redes neuronales convolucionales

256x256xRGB

128x128x16

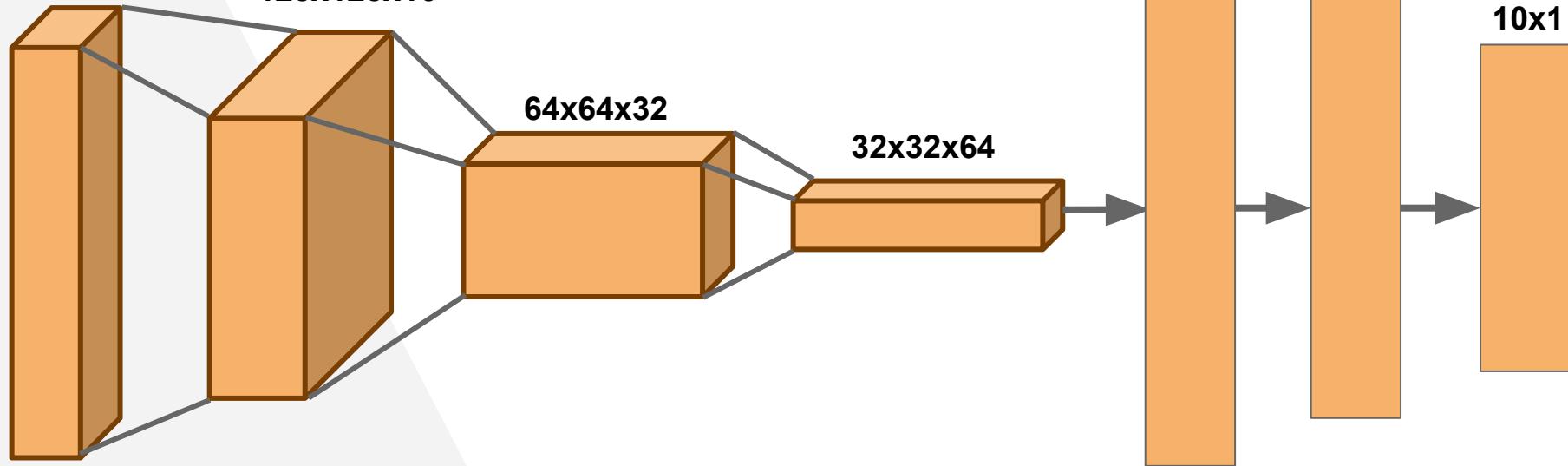
64x64x32

32x32x64

65536x1

1000x1

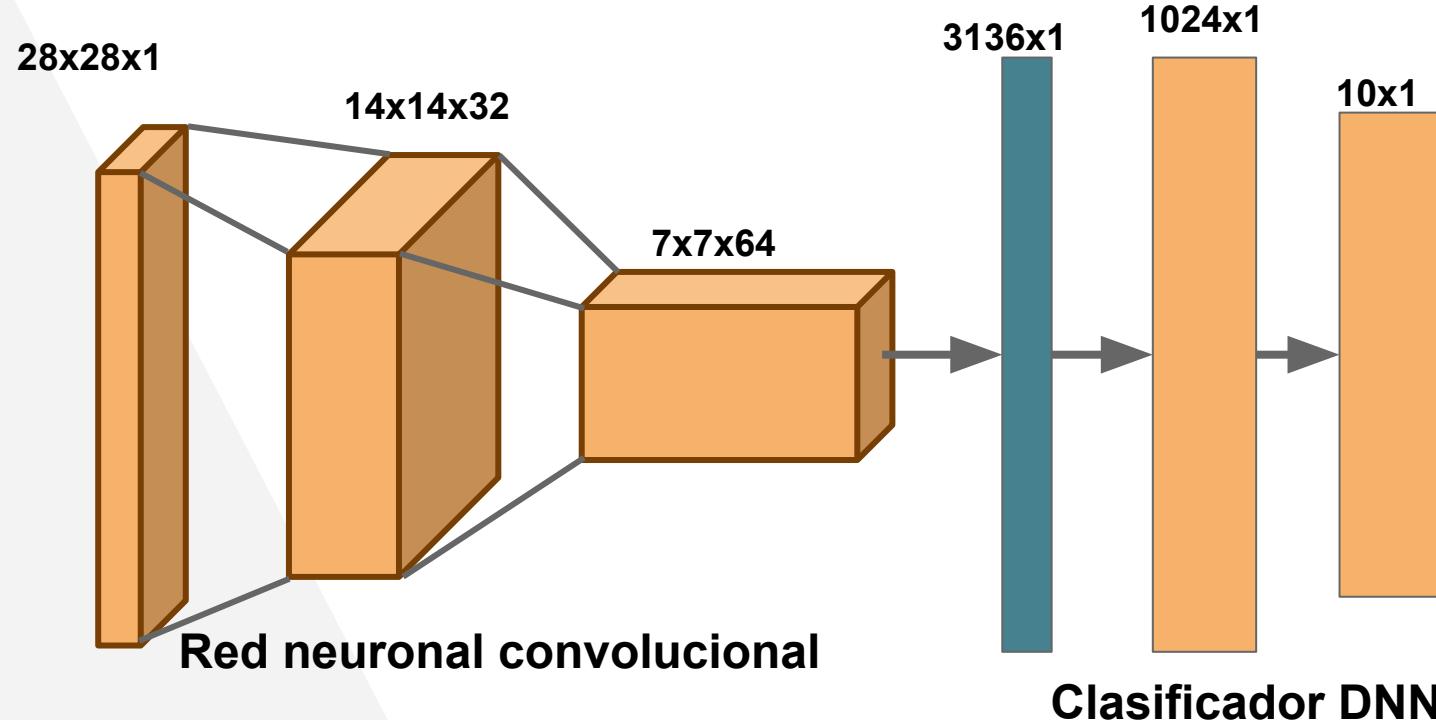
10x1



Red neuronal convolucional

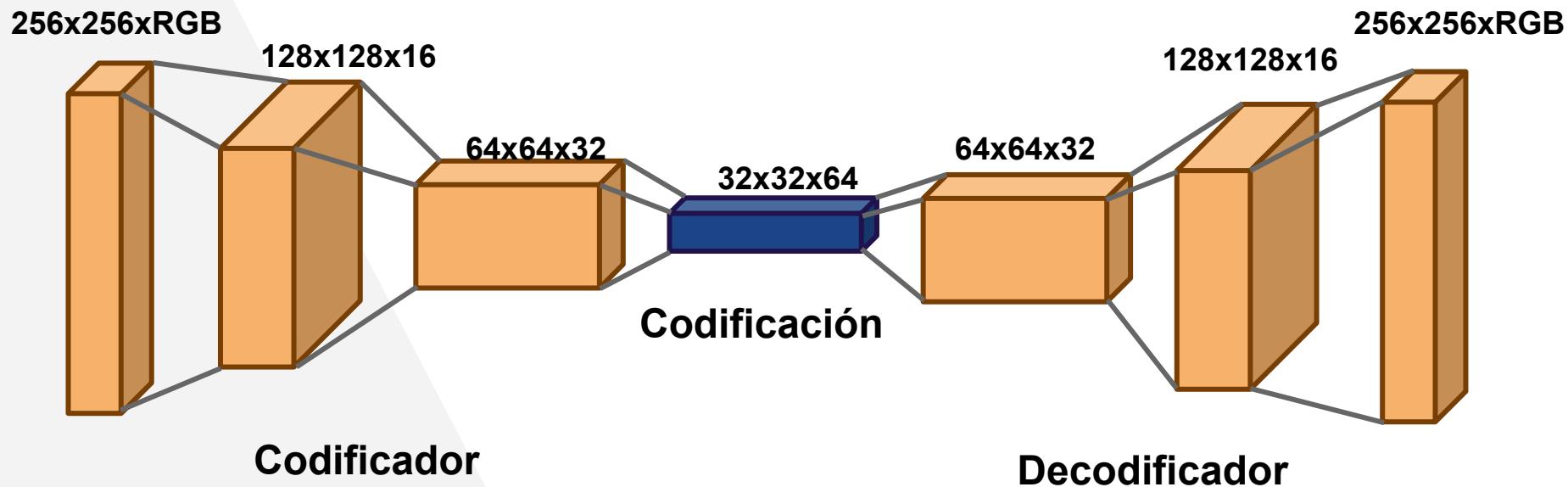
Clasificador DNN

Ejemplo MNIST convolucional

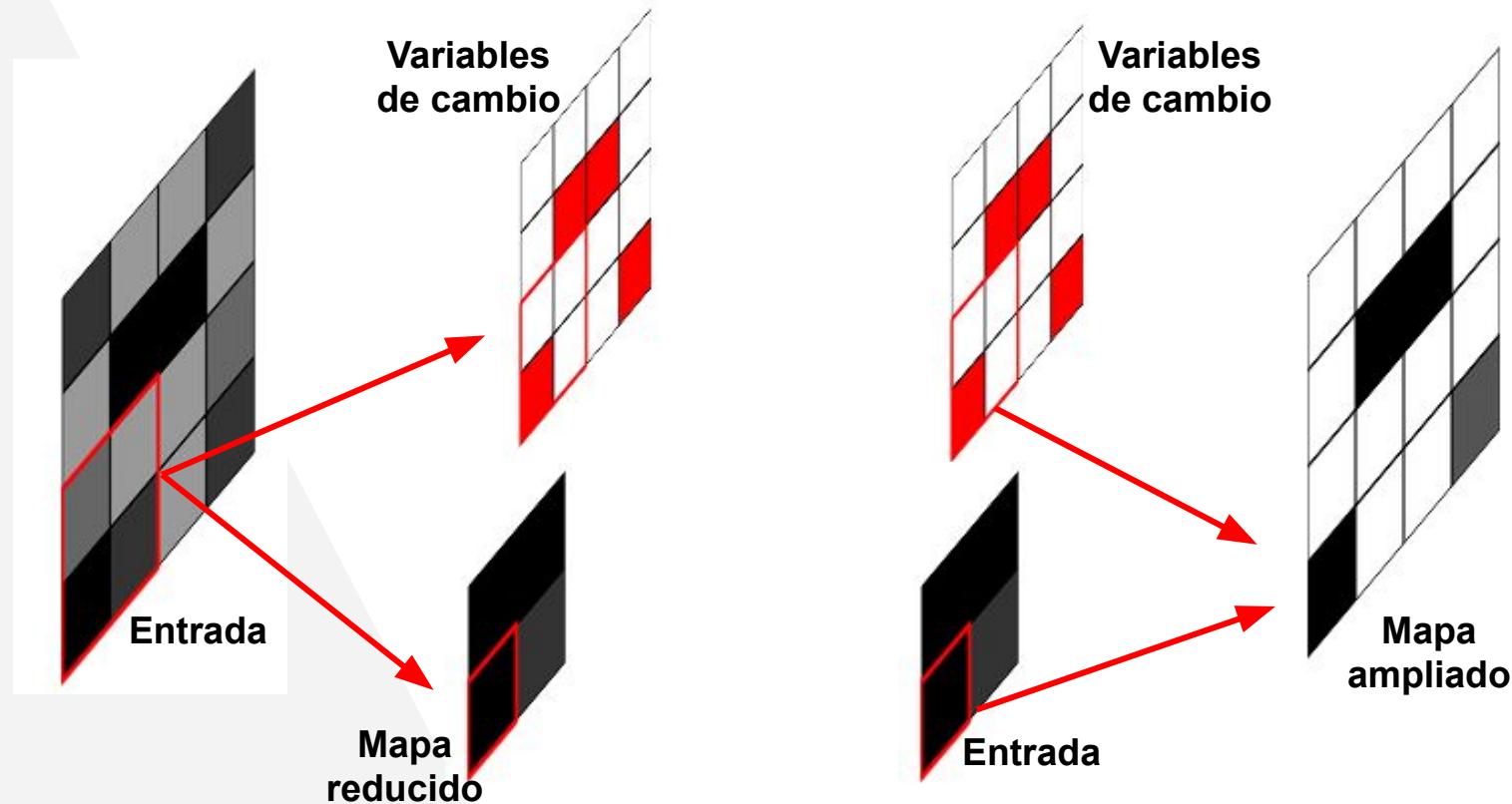


Abrir: Apuntes Red convolucional del curso aprendizaje profundo

Autoencoder convolucional

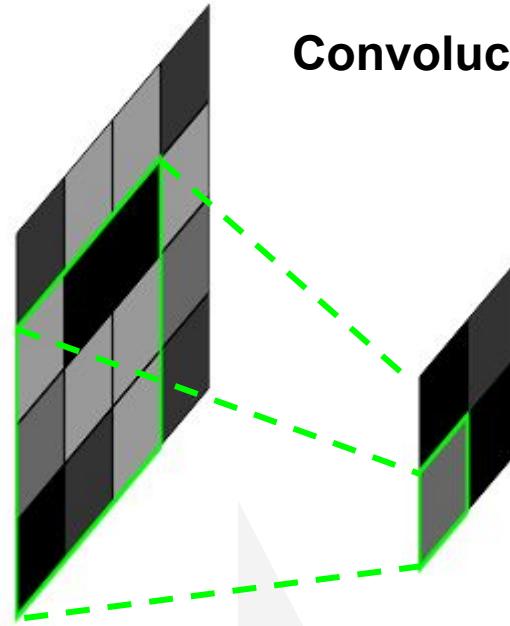


Compresión y descompresión (pooling, unpooling)

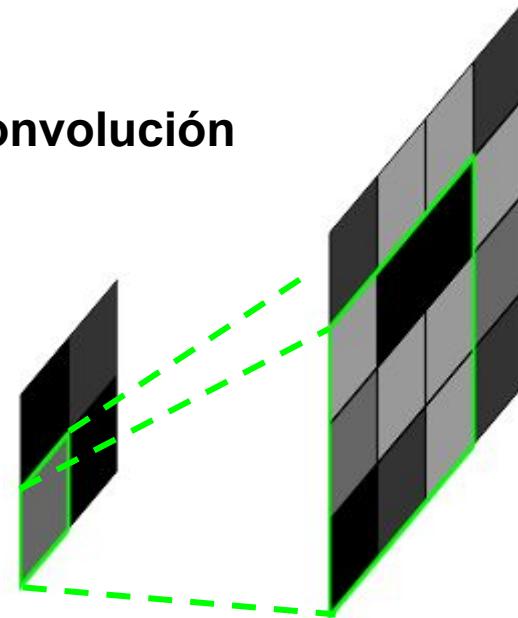


Compresión y descompresión (convolución, deconvolución)

Convolución



Deconvolución



[Abrir: Apuntes AE convolucional del curso aprendizaje profundo](#)

Redes neuronales recurrentes (RNN)

De perceptrón a recurrente



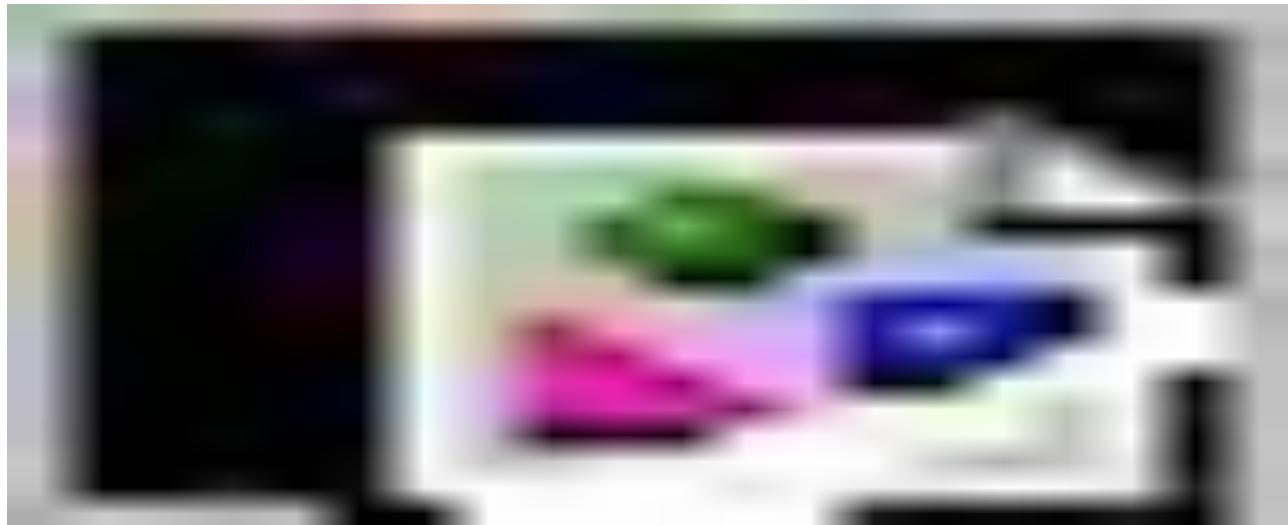
Redes neuronales recurrentes (RNN)

Recurrente simple



Redes neuronales recurrentes (RNN)

Recurrente simple

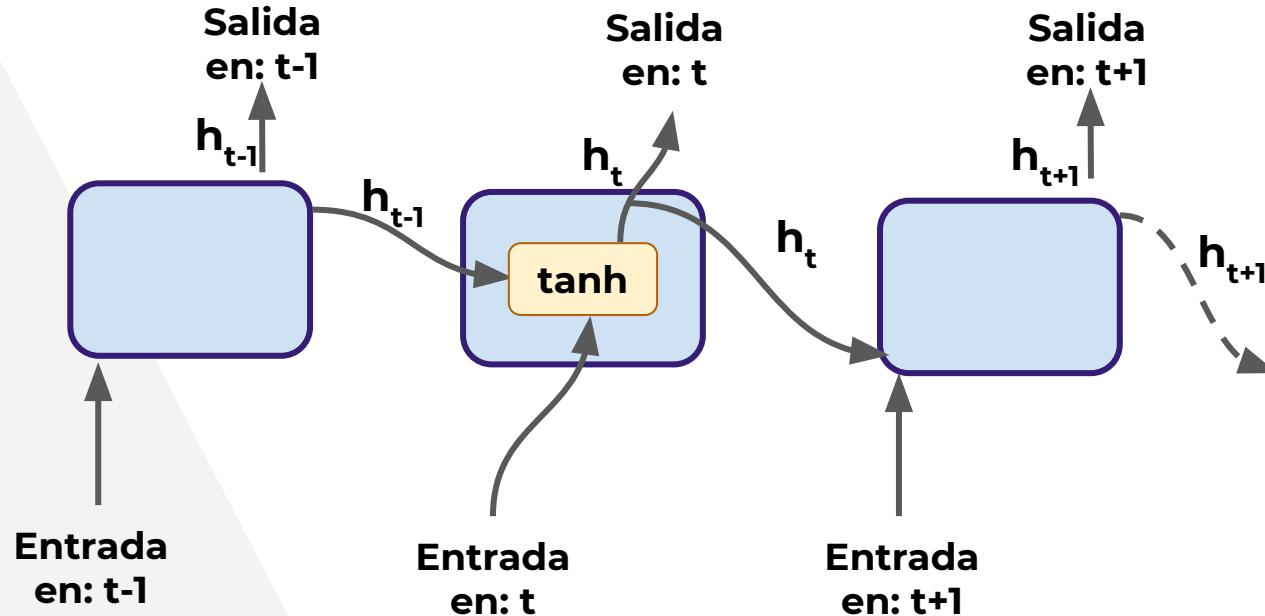


Redes neuronales recurrentes (RNN)

Recurrente simple



Red neuronal recurrente básica



Redes neuronales recurrentes

Estas redes a diferencia de las anteriores incorporan la capacidad de recordar información de sus interacciones pasadas (cuentan con memoria)

La memoria les permite analizar datos que de otra forma no podrían ser analizados. Principalmente, aquellos que dependen de una secuencia para generar sentido

- ▶ **Análisis de Texto**
- ▶ **Predicción Temporal**

Ejemplo: Utilidad de recurrencia en NLP

Todos los pulgares son dedos

↓
Representacion one Hot

todos	pulgares	dedos	son	los										
0	1	0	0	1	0	0	1	0	0	1	0	0	1	0

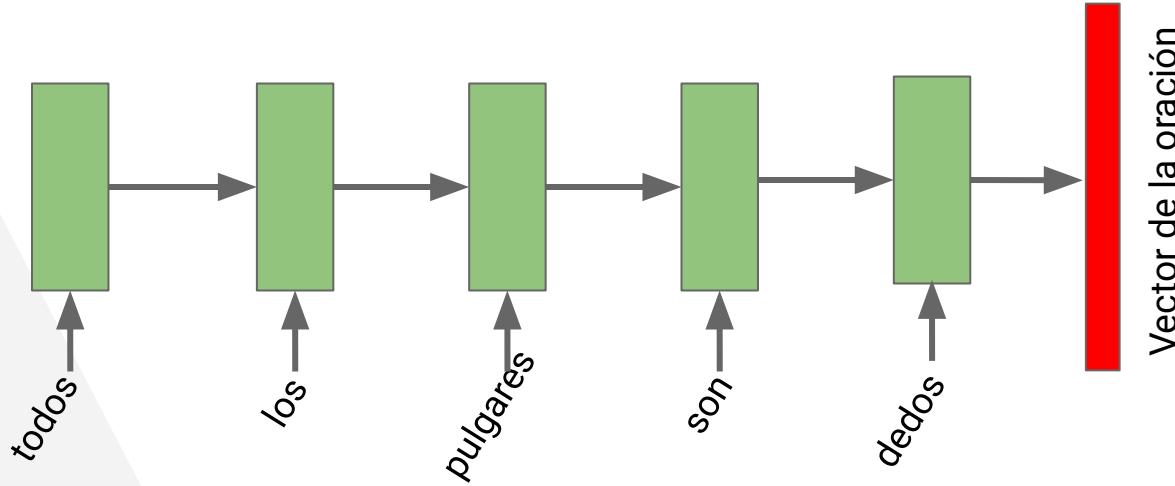
Ejemplo: Utilidad de recurrencia en NLP

Todos los pulgares son dedos

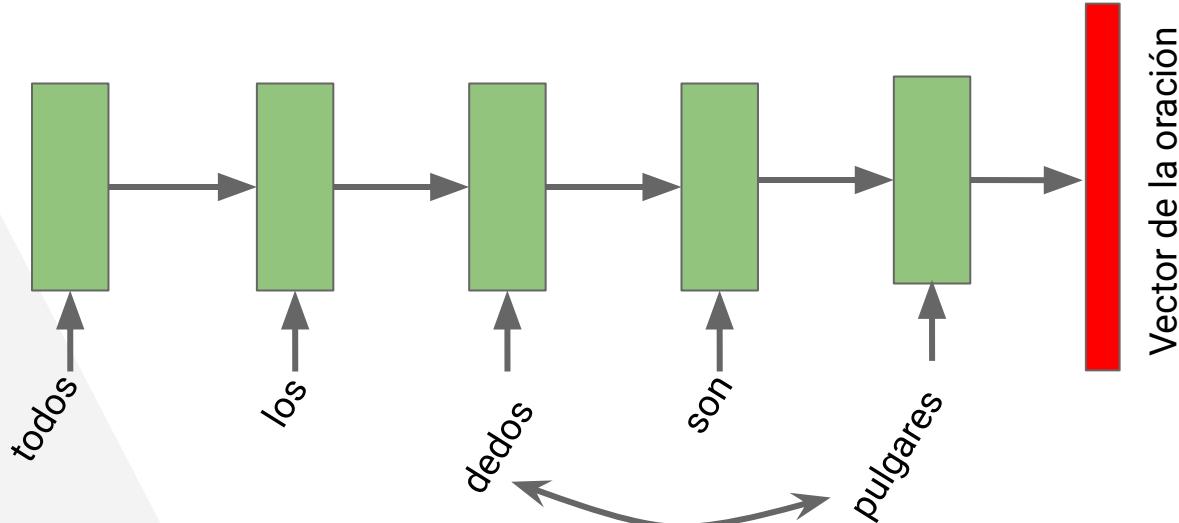
todos	pulgares	dedos	son	los										
0	1	0	0	1	0	0	1	0	0	1	0	0	1	0

Todos los dedos son pulgares

todos	pulgares	dedos	son	los										
0	1	0	0	1	0	0	1	0	0	1	0	0	1	0



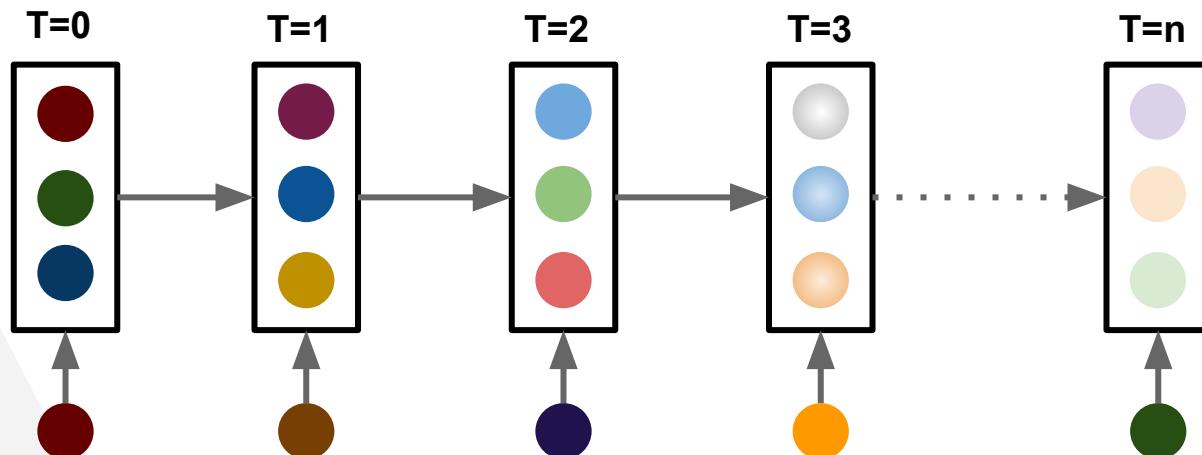
Todos los pulgares son dedos



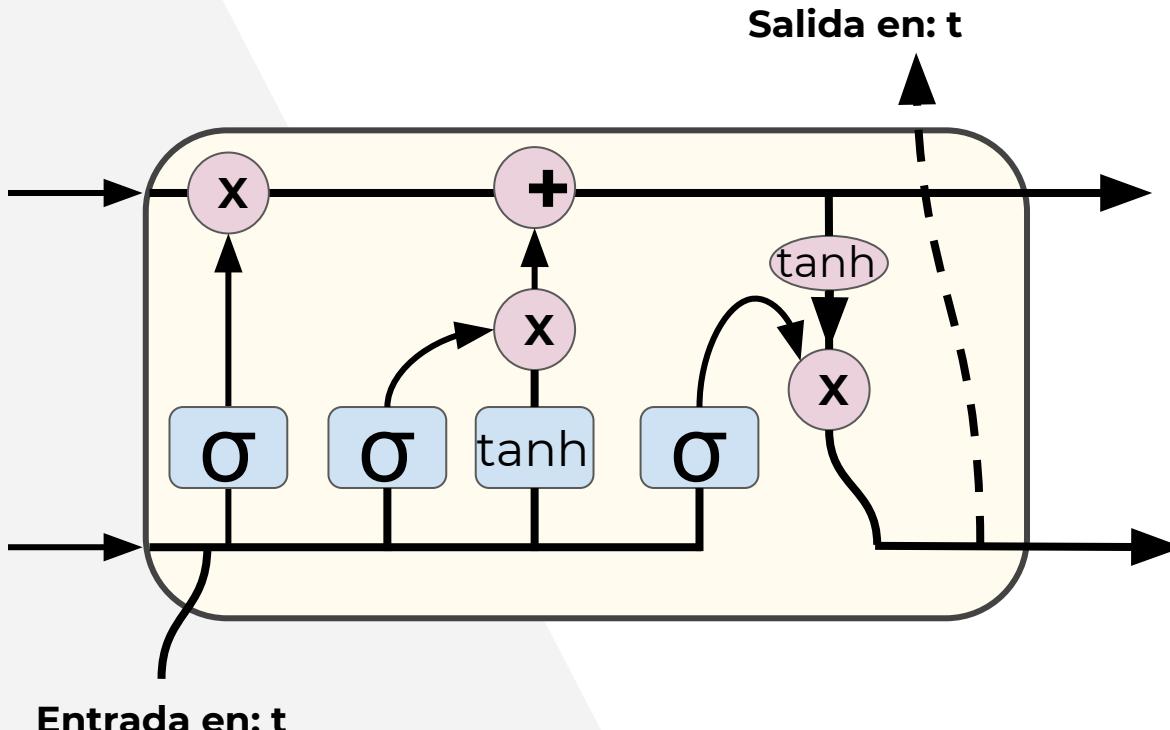
Todos los dedos son pulgares

Problema del gradiente desvaneciente

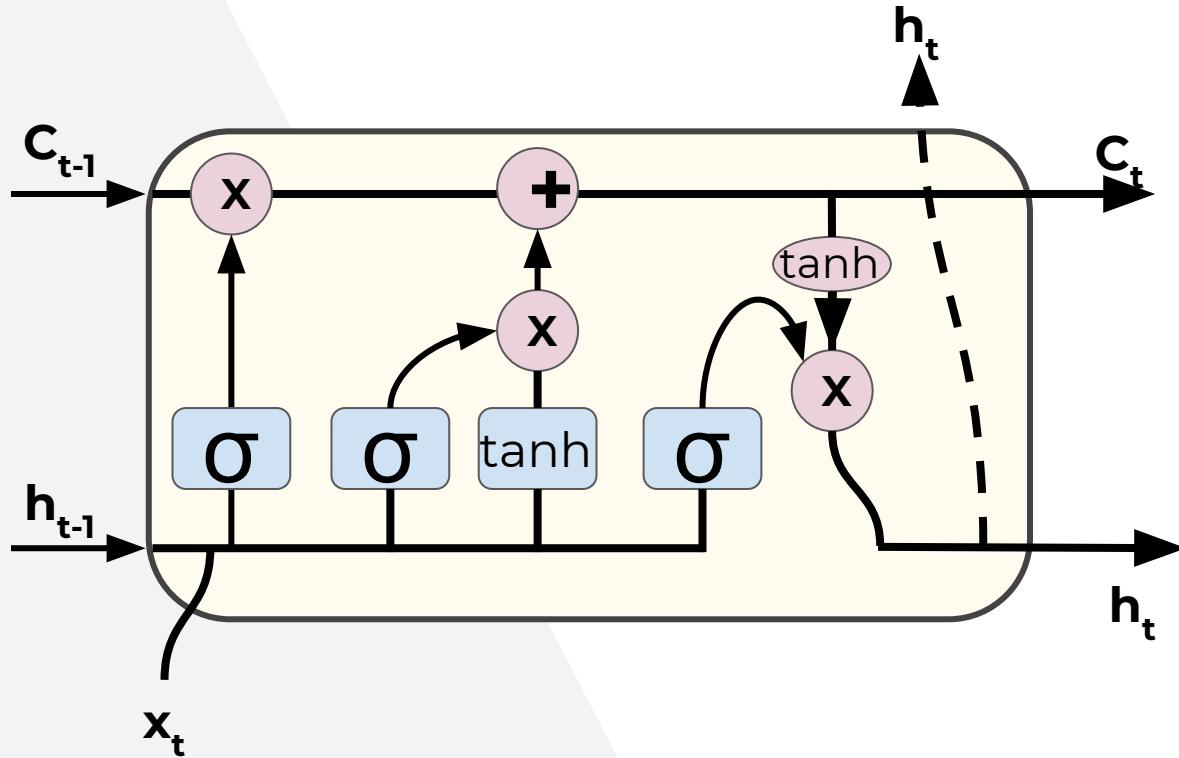
La información decae con el tiempo



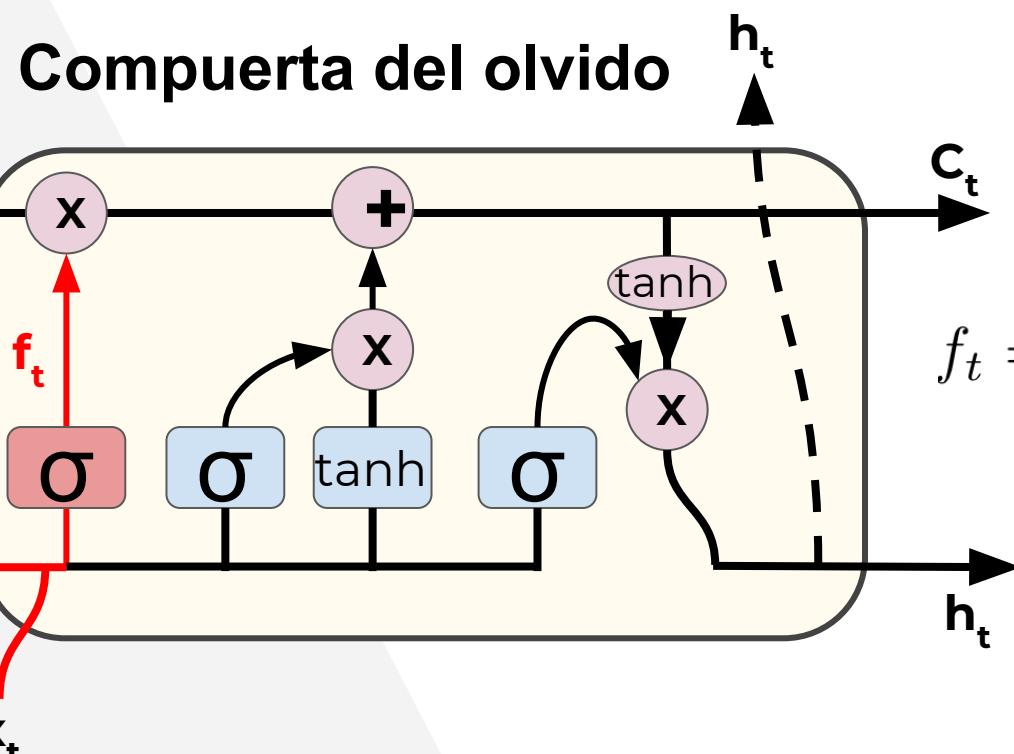
LSTM (Long-Short-Term Memory) RNN



LSTM (Long-Short-Term Memory) RNN

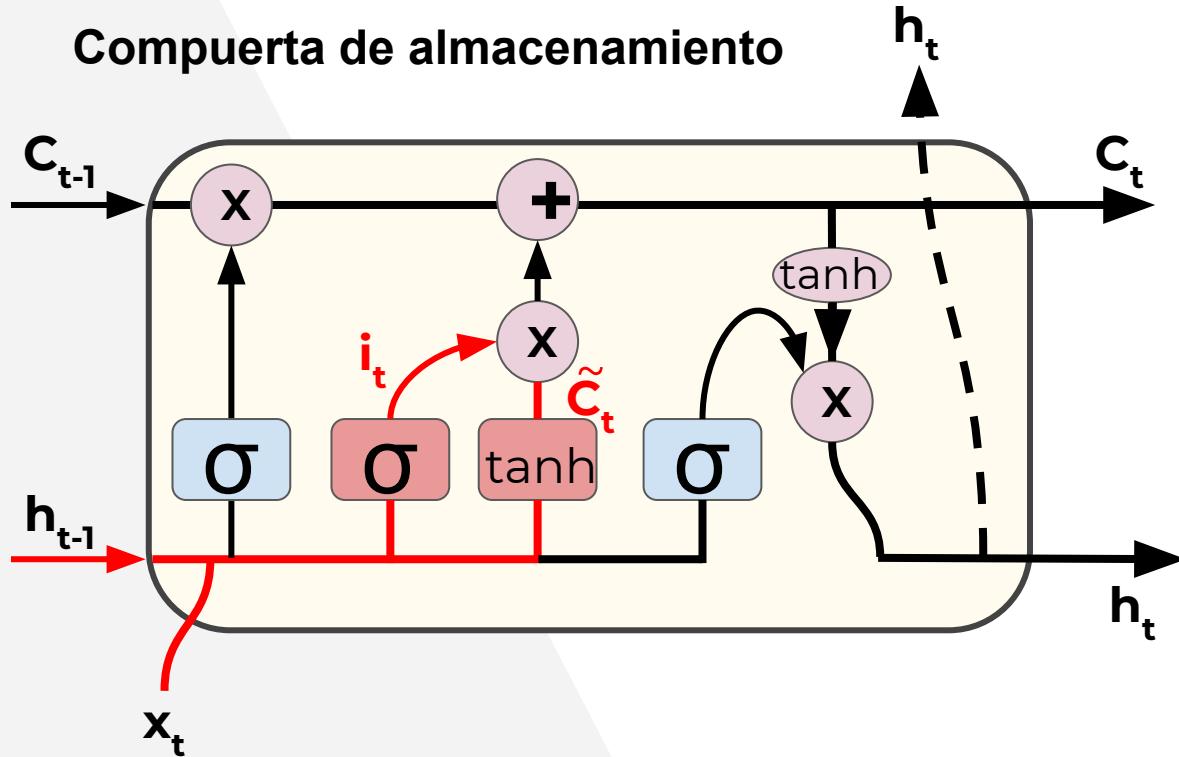


LSTM (Long-Short-Term Memory) RNN



LSTM (Long-Short-Term Memory) RNN

Compuerta de almacenamiento

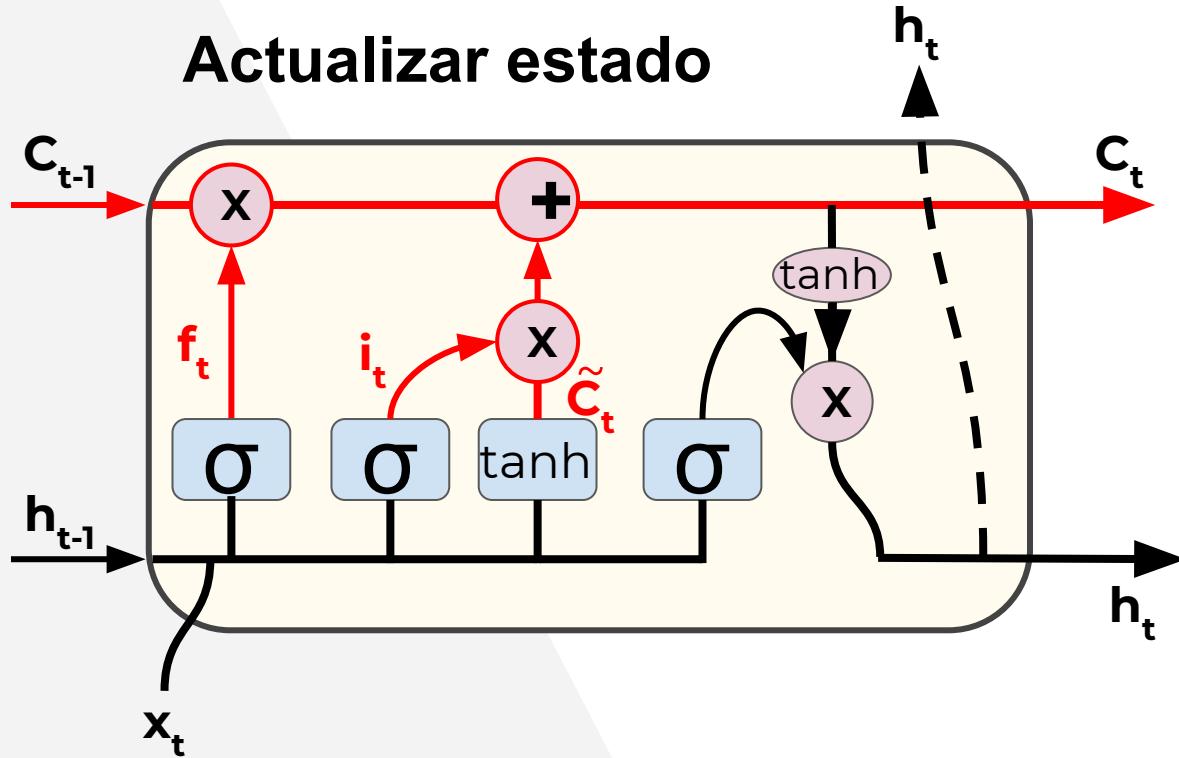


$$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

LSTM (Long-Short-Term Memory) RNN

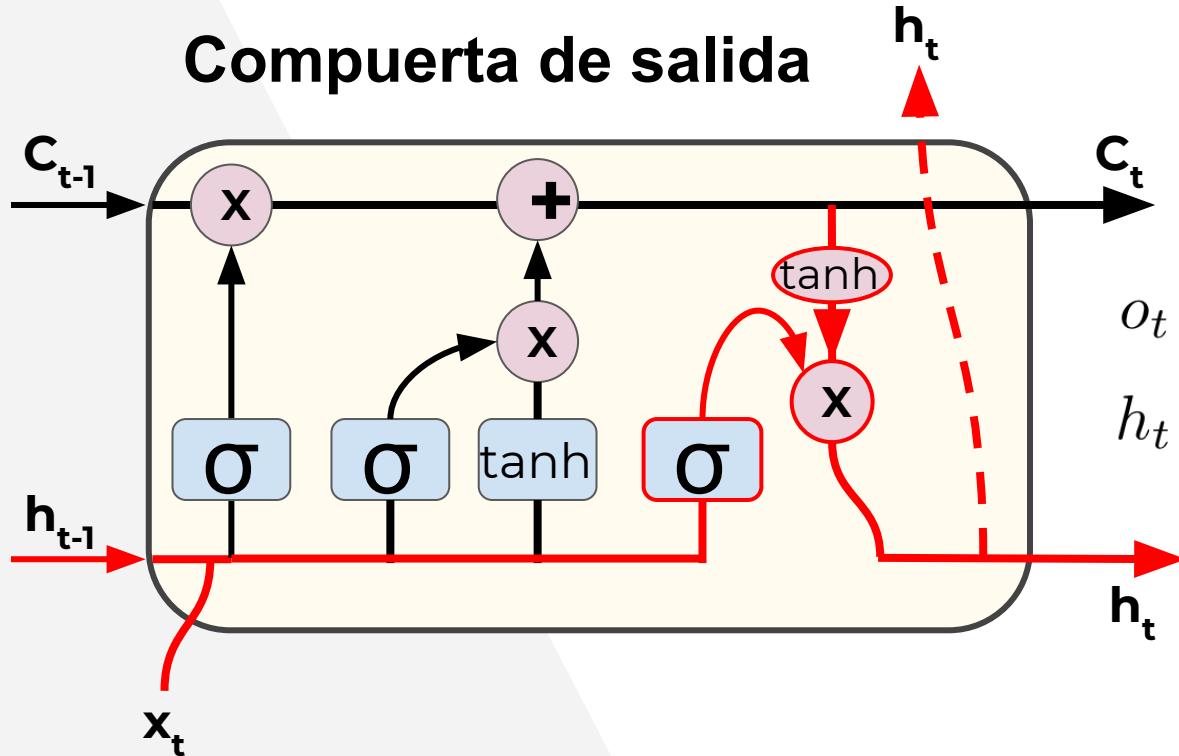
Actualizar estado



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

LSTM (Long-Short-Term Memory) RNN

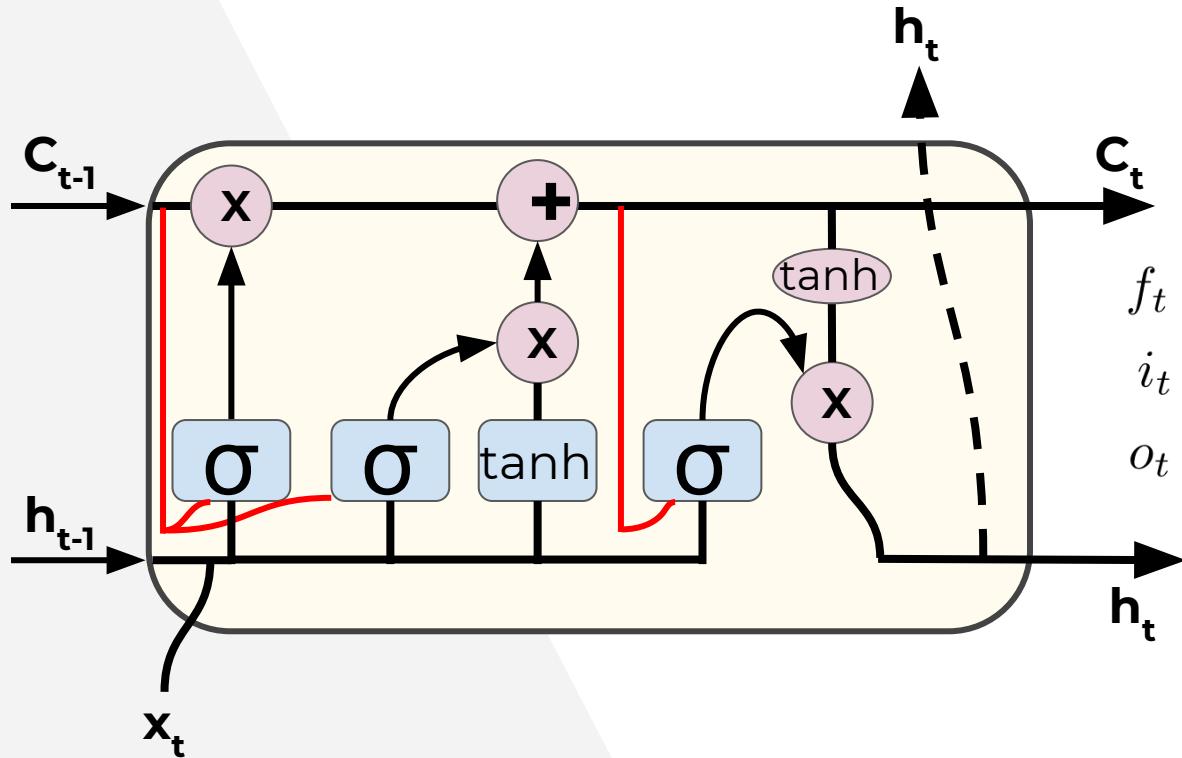
Compuerta de salida



$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

LSTM con “peepholes”

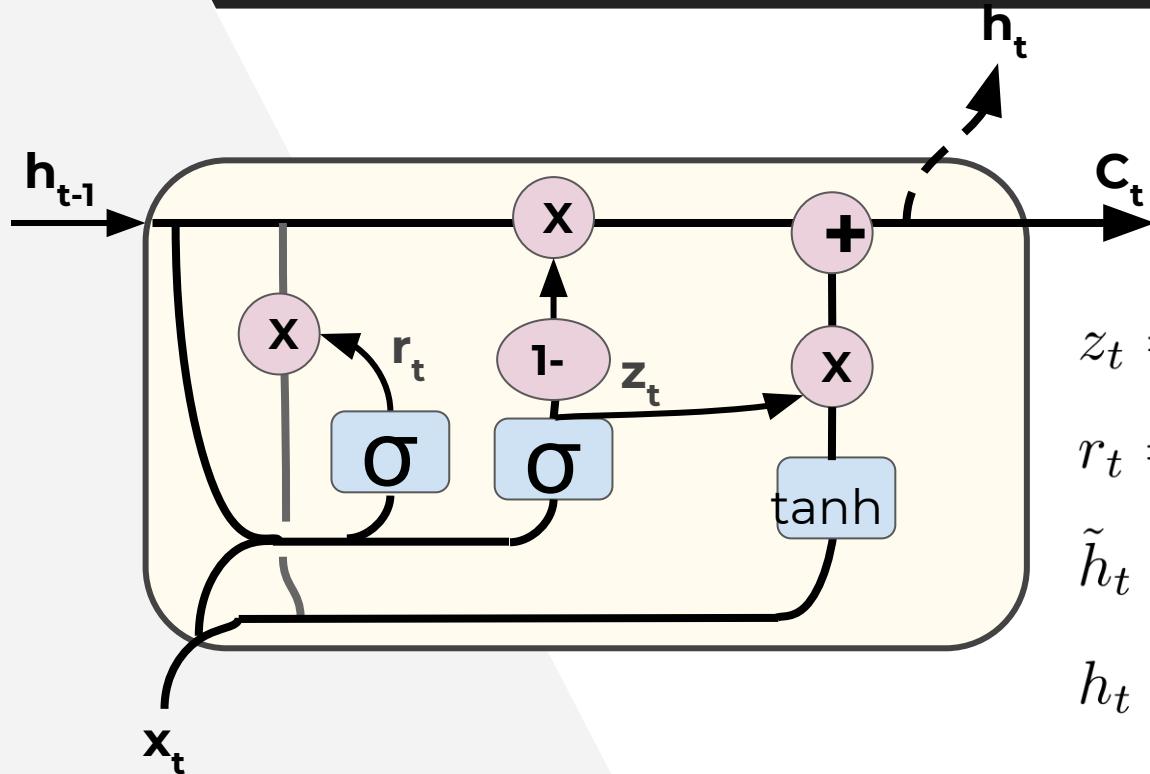


$$f_t = \sigma (W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma (W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i)$$

$$o_t = \sigma (W_o \cdot [C_t, h_{t-1}, x_t] + b_o)$$

Gated Recurrent Unit (GRU)



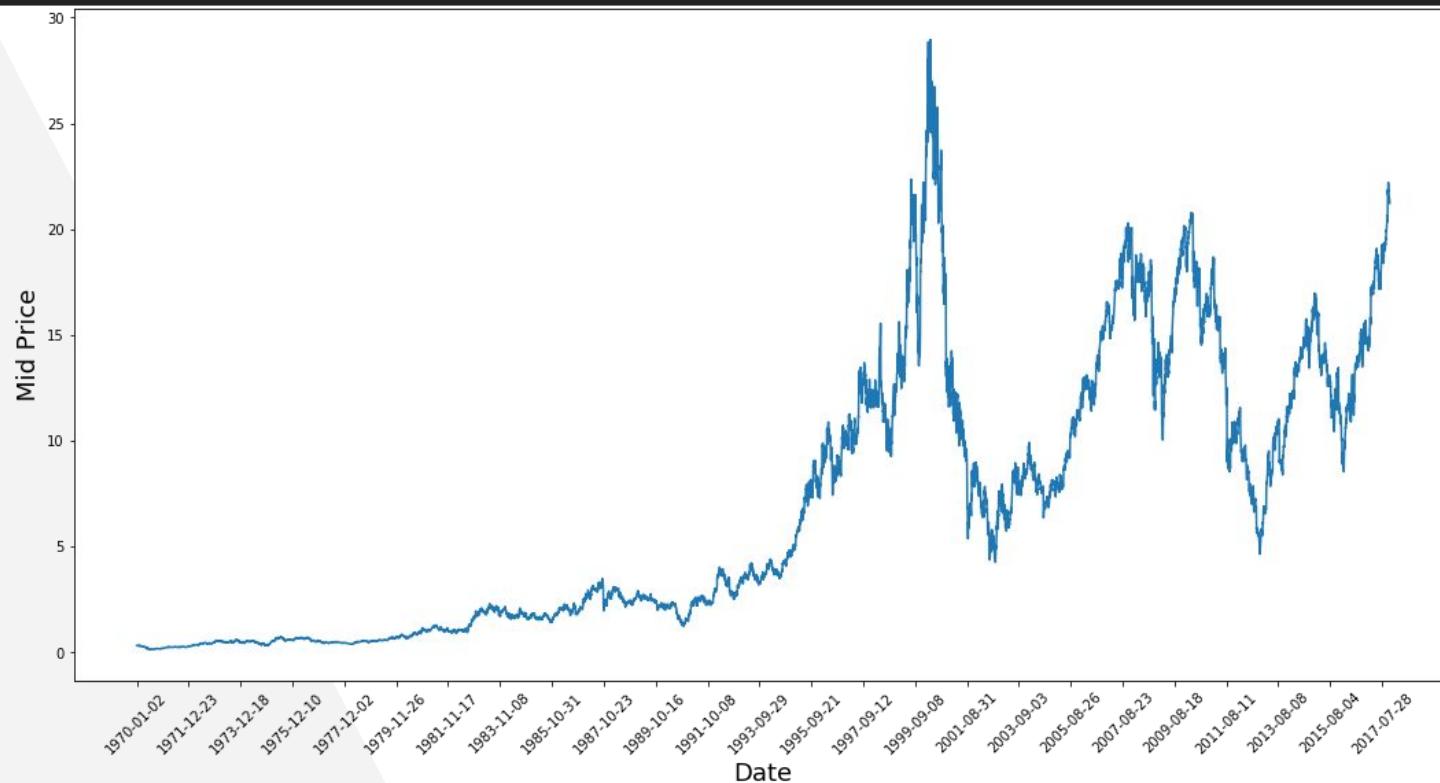
$$z_t = \sigma (W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma (W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh (W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Ejemplo: Predicción del valor de las acciones



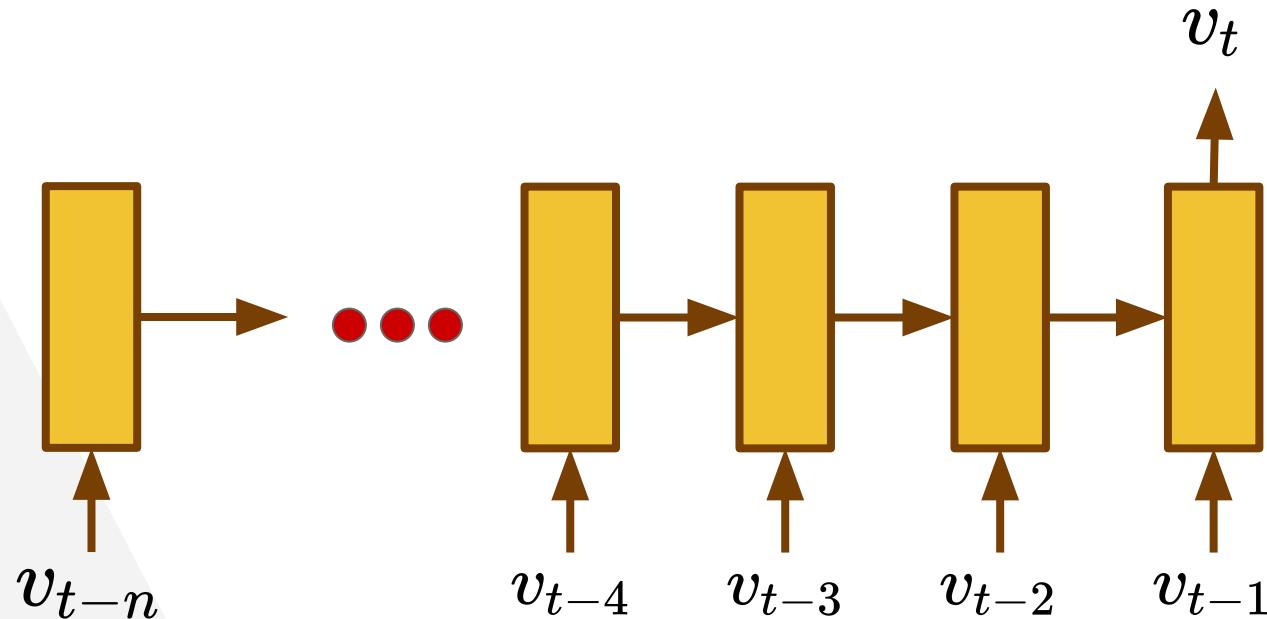
<https://finance.yahoo.com/quote/%5EGSPC/history?p=%5EGSPC>

Analizando el problema como series de tiempo

0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

Arquitectura :Predicción del valor de las acciones



Abrir: Apuntes red recurrente del curso aprendizaje profundo

¿Cómo escojo la arquitectura?

Red neuronal perceptron multicapa

- Extracción de características
- Clasificador
- Regresor
- Generador de secuencias

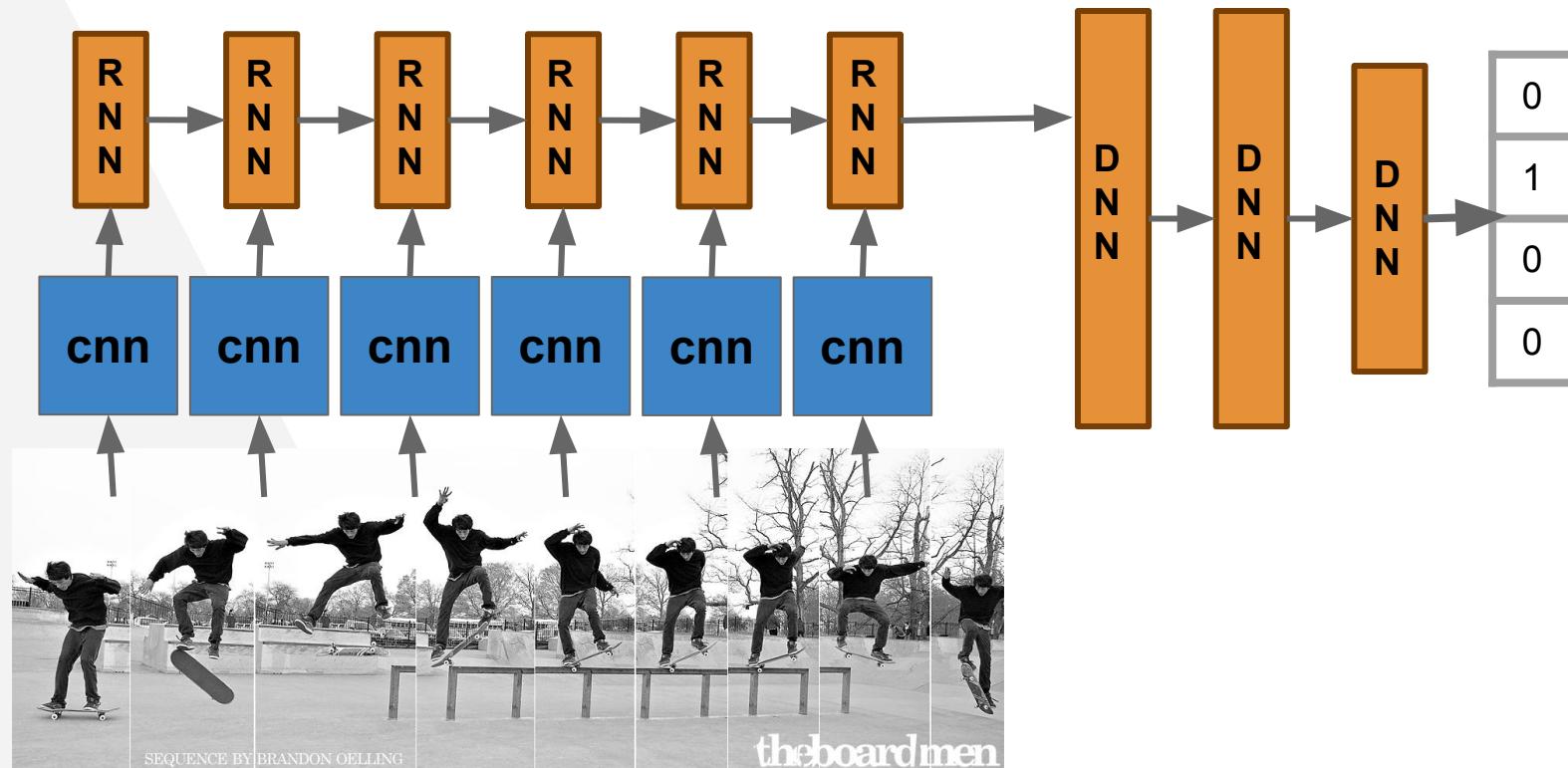
Red neuronal Convolucional

- Extracción de características
- Clasificador
- Regresor
- Generador de secuencias

Red neuronal Recurrente

- Extracción de características
- Clasificador
- Regresor
- Generador de secuencias

Las redes neuronales son Modulares

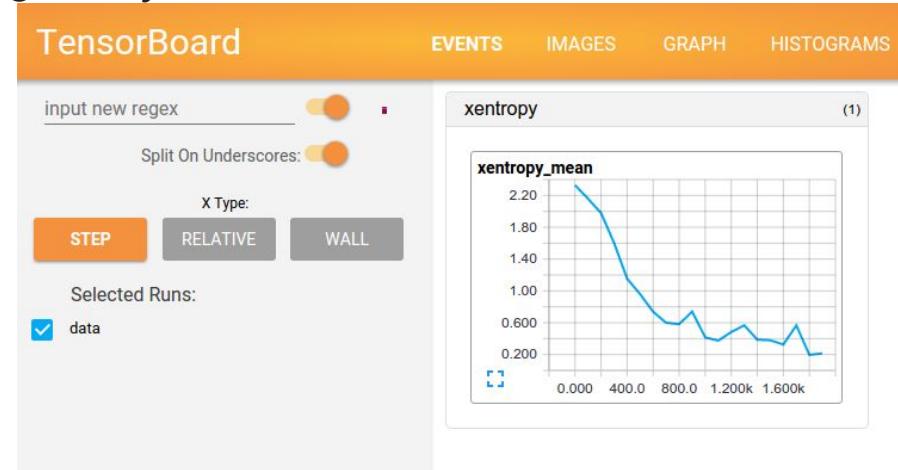


Determinando si existe un problema

Monitorear el entrenamiento

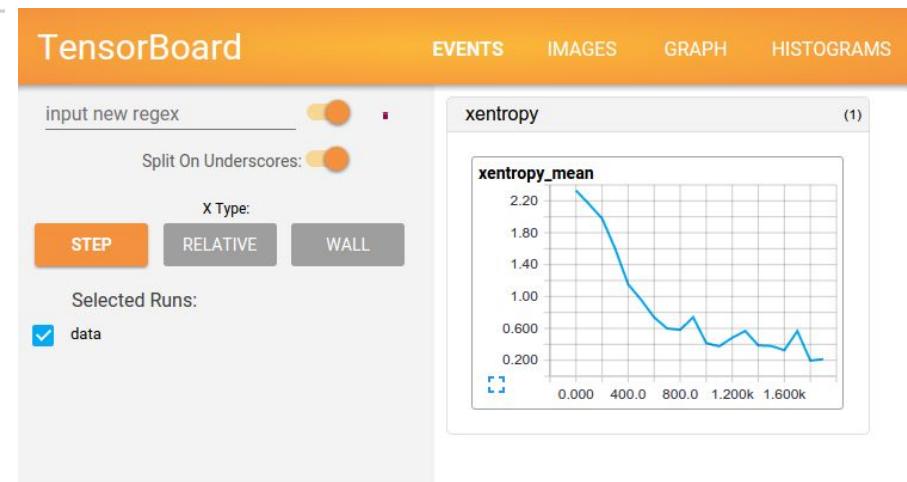
TensorBoard: TF herramienta de registro y análisis

1. Visualizar la arquitectura
2. Graficar métricas
3. Mostrar información adicional



Cálculo del error

```
Epoch 0 completed out of 10 loss: 1710127.14067
Epoch 1 completed out of 10 loss: 396040.798613
Epoch 2 completed out of 10 loss: 217499.373378
Epoch 3 completed out of 10 loss: 131378.394984
Epoch 4 completed out of 10 loss: 76742.2280202
Epoch 5 completed out of 10 loss: 49436.3336062
Epoch 6 completed out of 10 loss: 34864.258816
Epoch 7 completed out of 10 loss: 25578.158231
Epoch 8 completed out of 10 loss: 22059.0476563
Epoch 9 completed out of 10 loss: 19165.6416646
Accuracy: 0.98
```



5 pasos para poder usar TensorBoard

1. Anotar los elementos que se quieren analizar
2. Unir todas las anotaciones en un documento
3. Crear archivo para almacenar los documentos
4. Ejecutar los anotadores
5. Analizar en consola

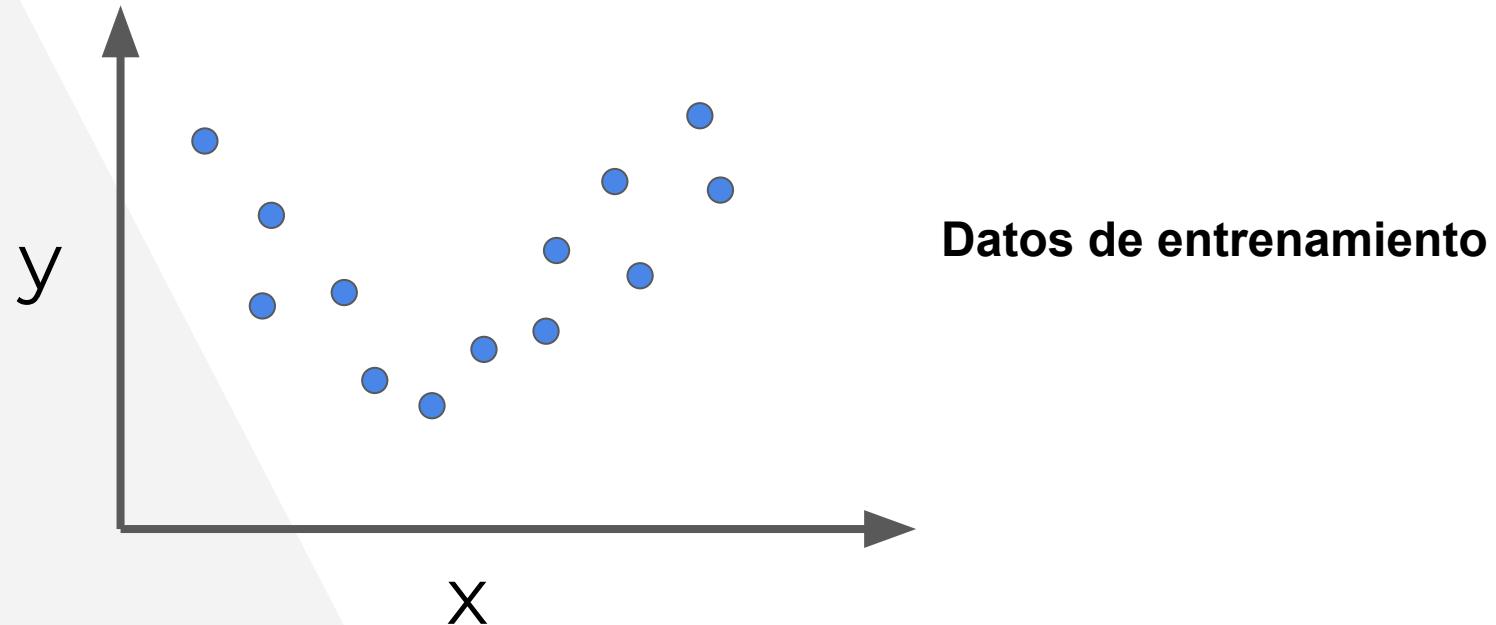
Abrir: Apuntes Tensorboard del curso aprendizaje profunda

¿Qué puedo hacer si mi red no entrena?

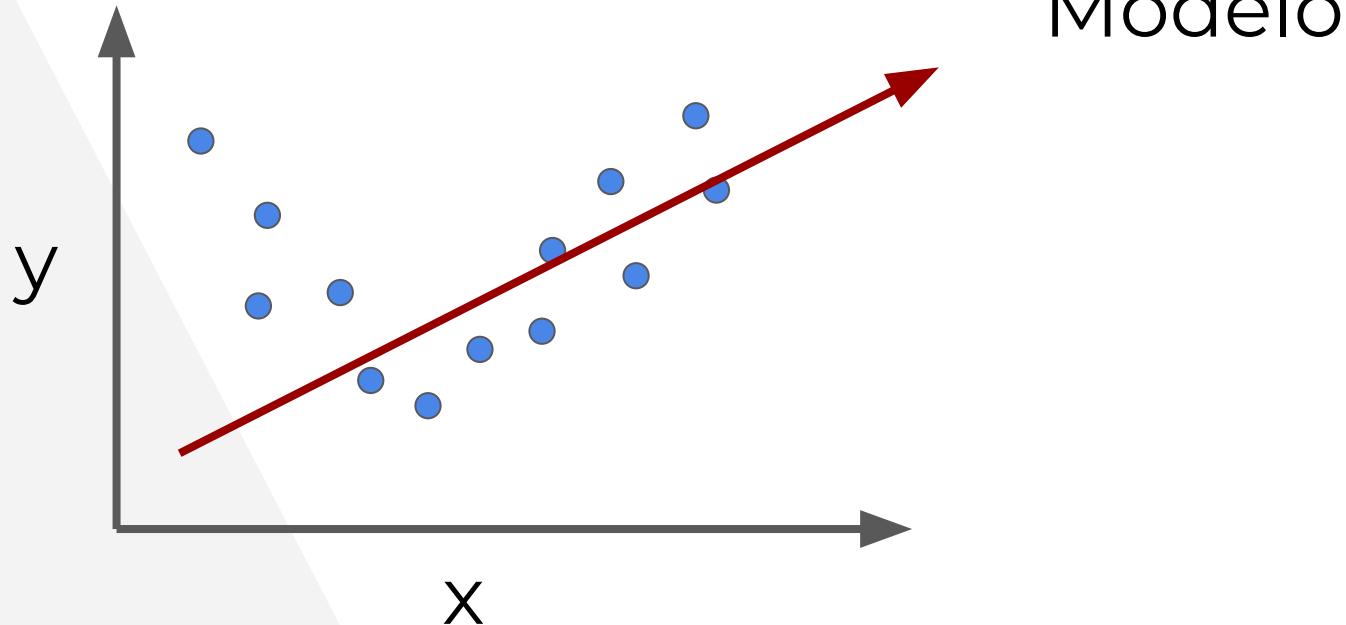
Problemas en el entrenamiento

- ▶ Problemas con los datos
 - ▷ Escasez de datos
 - ▷ Datos no balanceados
- ▶ Bajo entrenamiento y sobre entrenamiento
 - ▷ Monitorear el entrenamiento
 - ▷ Ajuste de los datos
 - ▷ Ajuste de algoritmos
 - ▷ Ajuste del modelo

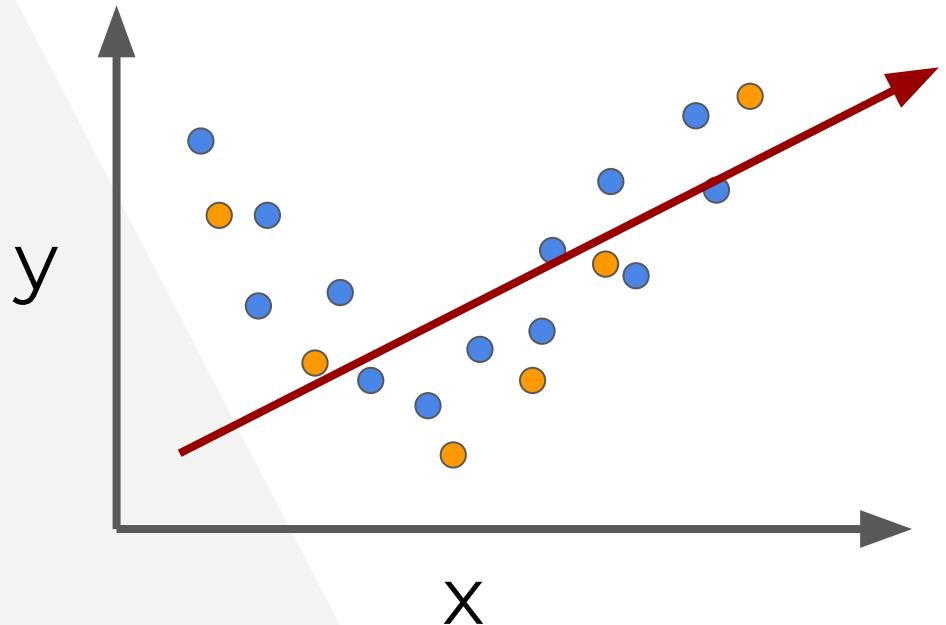
Sobre entrenamiento y bajo entrenamiento



Bajo entrenamiento



Bajo entrenamiento



Error en
prueba

Soluciones por medio de los datos

1. Preparar los datos
 - ▶ Normalizar (de 0 a 1)
 - ▶ Reescalar (de -1 a 1)
 - ▶ Estandarizar
2. Transformar los datos
 - ▶ ¿Toda la información es útil?
 - ▶ Probar una reducción dimensional (PCA)
 - ▶ ¿Hay alguna información adicional que le puedas proveer?
 - ▶ ¿Existe alguna relación entre los datos que valga la pena resaltar?

Replantear el problema

Existe una mejor manera de resolver el problema

- ▶ Se pueden incorporar elementos temporales que permitan complementar al problema
- ▶ Se puede abordar el problema de clasificación como un problema de regresión o viceversa
- ▶ Se puede cambiar la codificación de la salida a binaria, softmax, continua
- ▶ Se puede dividir el problema general en pequeños subproblemas para que sea más fácil la solución

No free lunch theorem

La metáfora que da nombre a este algoritmo consiste en decir que existen una serie de restaurantes cada uno especializado en un tipo de comida. Cada restaurante ofrece un menú con diferentes precios para cada platillo pero con un precio menor para el platillo en el que se especializan

Bajo este paradigma cuando un cliente quiere comer puede ir al restaurante que venda el platillo de su interés más barato. Pero no existe un restaurante que de todos los platillos más baratos Es decir no existe un restaurante que de toda la carta gratis

Solución para mejorar el desempeño modificando el algoritmo

Escoger el mejor algoritmo para resolver tu problema

No existe un algoritmo que se desempeñe mejor en todos los casos

“No free lunch theorem”

1. Implementar de la literatura
2. Buscar soluciones a problemas similares
3. Probar con los algoritmos más comunes

Ajustando el algoritmo

- 1. Inicializar los pesos con valores aleatorios pequeños**
- 2. Ajustar la velocidad de aprendizaje**
 - a. Probar valores grandes (0.2, 0.3)
 - b. Probar con valores pequeños ($1e-4$, $1e-6$)
 - c. Tratar de reducir el valor cada n generaciones

Nota: La velocidad de aprendizaje debe de ser más grande en redes neuronales profundas, que a su vez tardan más generaciones en entrenar

Ajustando el algoritmo

3. Lotes y épocas

- a. La configuración general es usar lotes pequeños con un gran número de épocas
 - i. Intenten con el lote más grande que entre en memoria
 - ii. Intentar con un lote de uno
 - iii. Probar con lotes en diferentes dimensiones 8, 16, 32, ...
- b. Las épocas suelen ser pequeñas no recurrir a muchas épocas al menos que el error siga bajando

Ajustando el algoritmo

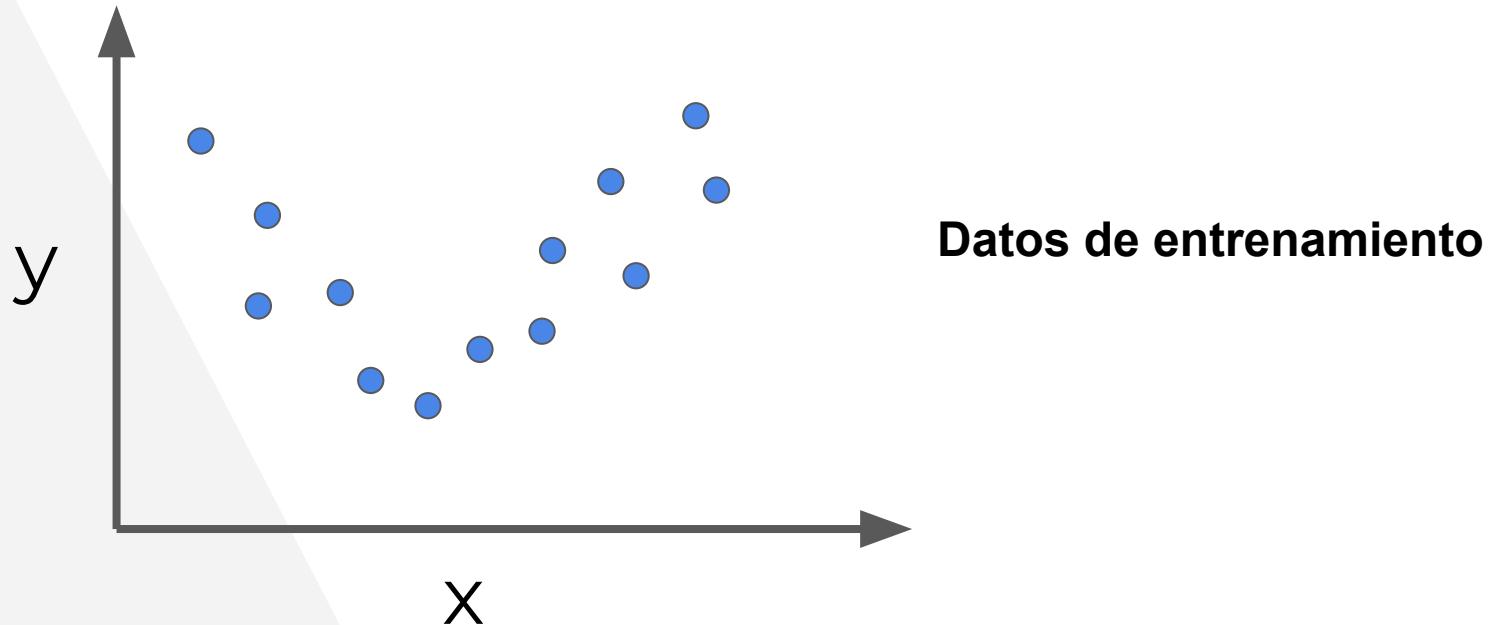
4. Cambiar las funciones de activación

- a. En general las funciones rectificadoras funcionan mejor
- b. Si no funciona, tratar con las funciones sigmoide y tangente hiperbólica, solo con datos normalizados
- c. A la salida se puede usar la función softmax o lineal

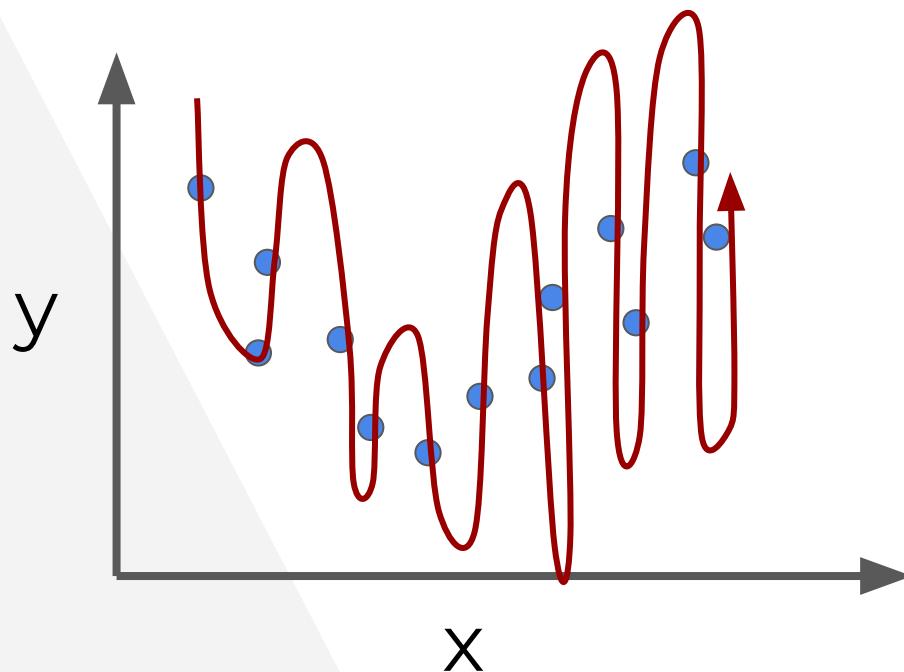
5. Función de optimización y costos

- a. En general, seleccionar el optimizador adam y la función de costo que más se adapte a tu problema

Sobre entrenamiento

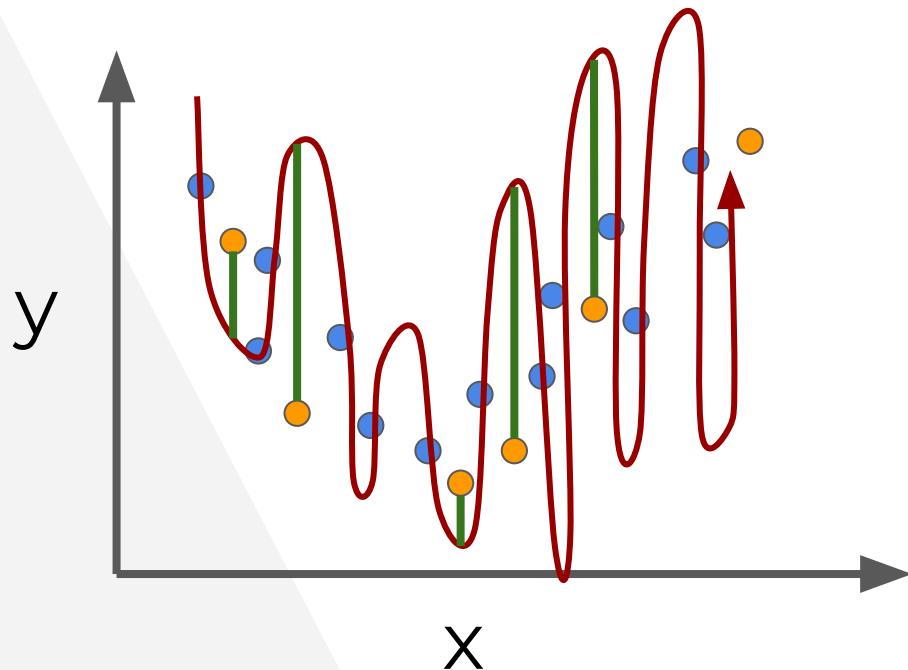


Sobre entrenamiento



Error muy bajo en el conjunto de entrenamiento

Sobre entrenamiento



Error
grande en
el conjunto
de prueba

Regularization: La regularización es un proceso que limita el aprendizaje del modelo para reducir el sobreajuste. La forma más utilizada de regularización en aprendizaje profundo es a través del Dropout.

BalanceClasses: Clases variadas y equilibradas en cantidad. Si el número de muestras no está balanceado puede causar que el modelo tienda a favor de alguna clase dominante al momento de evaluar el modelo.

ParametersTunning: Reducir o aumentar el número de capas y neuronas que contiene la red.

FeaturesReduction: Reducir la cantidad de características extraídas para entrenar el modelo.

GatherData: Obtener más conjuntos de datos para el entrenamiento y pruebas. Esto puede ayudar a que la red tenga suficientes muestras para generalizar y no aprender de memoria los datos con los que fue entrenado.

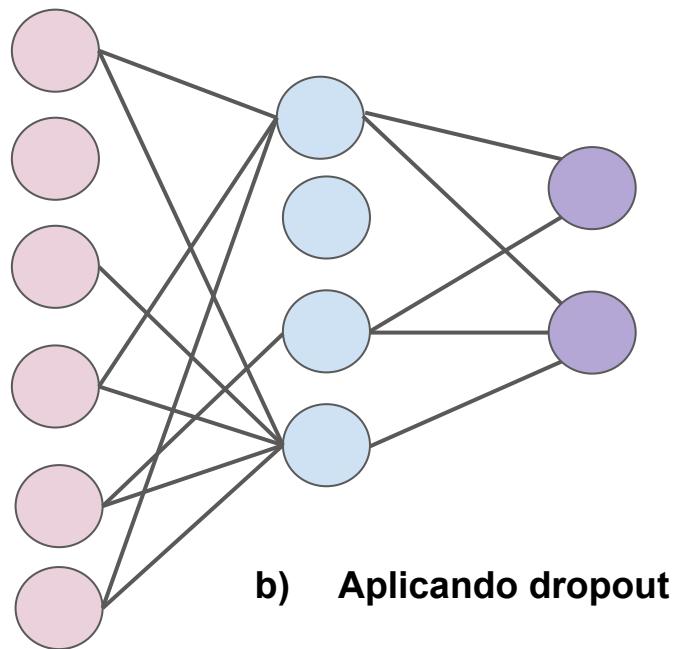
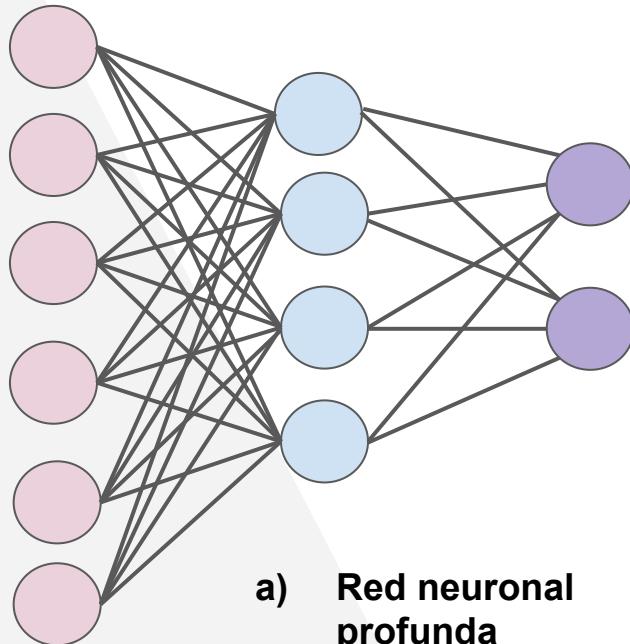
Regularización

La regularización sirve para combatir el sobreentrenamiento. Esto se logra evitando que la red aprenda de una manera muy rígida y evite la generalización

- ▶ Dropout
- ▶ Penalización en los pesos

Dropout

El dropout permite desconectar neuronas de una manera aleatoria, cambiando los canales donde se realiza la sinapsis



Regularización por penalización en los pesos

- ▶ Esta es la manera clásica de regularización. Se ha aplicado en muchos casos
- ▶ En general, pesos pequeños son mejores que pesos grandes. Cuando un peso empieza a crecer desmedidamente se genera una relación en la red neuronal que es imposible de olvidar lo que, en general, lleva a un sobre entrenamiento.
- ▶ Las penalidades evitan que los pesos crezcan y los mantiene cercanos a cero.

Normas para regularización

Norma L1

$$\text{NormaCosto} = \text{Costo} + \lambda \sum_{i=1}^k |w_i|$$

Norma L2

$$\text{NormaCosto} = \text{Costo} + \lambda \sum_{i=1}^k w_i^2$$

[Abrir: Apuntes Regularización del curso aprendizaje máquina](#)

Mejorando el desempeño con ensambles

Si después de intentar ajustar un algoritmo no se tiene el desempeño deseado se pueden probar otras soluciones que involucran modificar la arquitectura de la red

- ▶ Combinar diferentes arquitecturas
- ▶ Dividir el problema en problemas más pequeños
- ▶ Unir predicciones

Simplificando el código

Como pudieron ver a lo largo del curso, TensorFlow es una herramienta bastante poderosa para el desarrollo de redes neuronales profundas.

Sin embargo, puede llegar a ser difícil crear modelos, ya que es necesario considerar los tamaños y las conexiones cuidadosamente

Por esta razón, se han creado librerías de alto nivel que proveen una forma limpia y conveniente de crear modelos de aprendizaje profundo

- ▶ **TFlearn**
- ▶ **Keras**

Introducción a Keras

Keras es una librería minimalista que corre arriba de TensorFlow

Esta librería fue generada por el MIT y corre en CPU o GPU, dada la instalación de TensorFlow

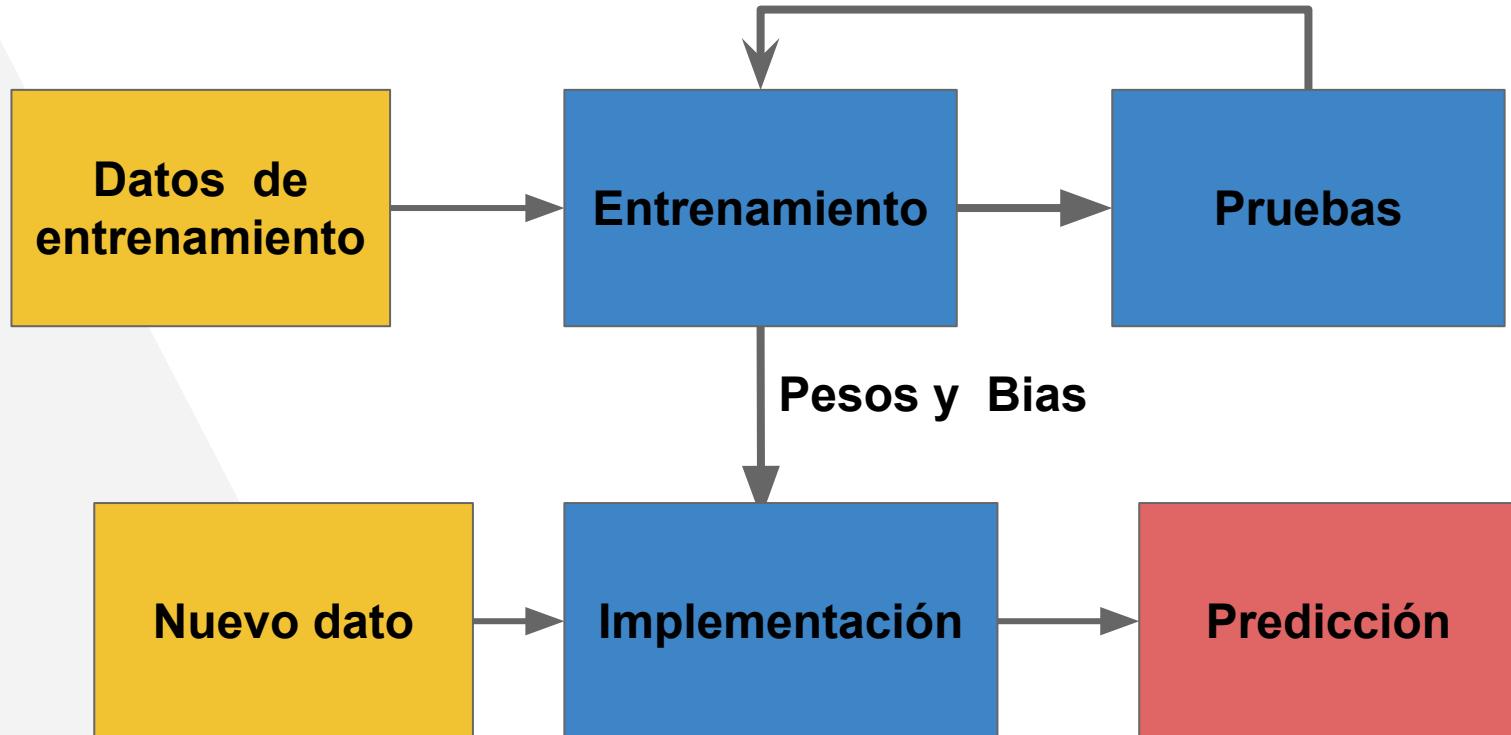
- ▶ Modular
- ▶ Minimalista
- ▶ Extensible
- ▶ Python nativo

Construyendo modelos con keras

1. Definir el modelo
2. Compilar el modelo
3. Entrenar/ajustar el modelo
4. Hacer predicciones/probar el modelo

[Abrir: Apuntes Vanilla keras del curso aprendizaje profundo](#)

Proceso de implementación de una red neuronal



Abrir: [Apuntes Vanilla keras y Implementar_vanilla_keras](#) del curso aprendizaje profundo

Arquitecturas para casos especiales

Al ser modulares las redes neuronales
pueden ser usadas en diferentes
configuraciones