



# MACHINE LEARNING – UN VISTAZO A LA IA

M. en T.A. Angel Hernández

Correo: [amhrdz.1001@gmail.com](mailto:amhrdz.1001@gmail.com)

Linkedin: Angel Moisés Hernández Ponce

Telegram: @amhrdz.1001

# Contenido

1. ¿Qué es Python?
2. ¿Cómo instalar y usar Python?
  1. *Python IDE / CMD / Terminal*
  2. *Jupyter Notebook y Anaconda*
  3. *Google Colabs*
3. Introducción a Google Colabs
4. Tipos de objetos y estructura de datos
  1. *Números*
  2. *Strings*
  3. *Listas*
  4. *Tuplas*
  5. *Diccionarios*
  6. *Booleanos*
5. Operadores de comparación
6. Declaraciones (statements) en Python
  - *Sintaxis*
  - *If y elif*
  - *Ciclos for*
  - *Ciclos while*
  - *Iterando a través de listas*
7. Expandiendo Python con liberías
8. Funciones y definiciones
  - *Estructura*
  - *Palabras clave*
  - *Cómo invocarlas*

# Contenido

 IA y Machine Learning ¿Qué diferencia hay?

 Tensorflow y Keras

 Conceptos básicos de ML

 Redes neuronales

 Practicas

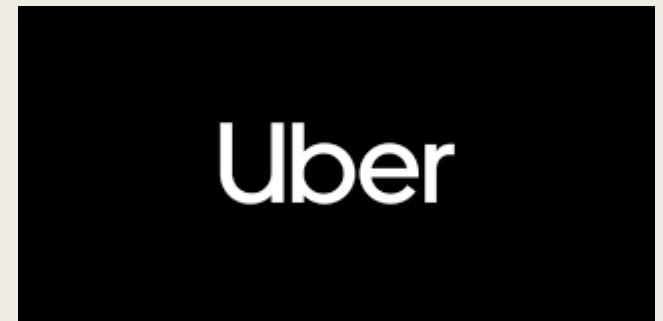
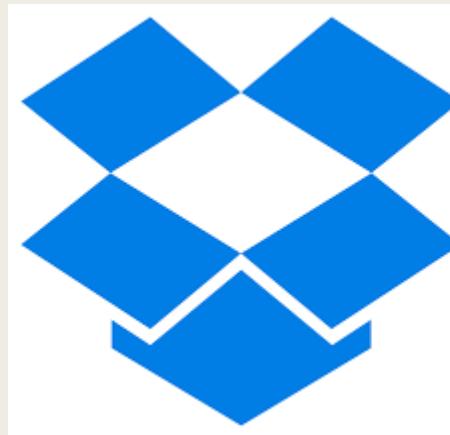
 Redes normales vs Redes convolucionales

 Proyecto final

# ¿QUÉ ES PYTHON?

- Python es un lenguaje de programación no interpretado cuya filosofía se basa en una sintaxis ligera que favorezca un código legible.
- Es de código libre
- Corre en todos los sistemas operativos (Windows, MacOS y Linux).
- Es fácil de aprender.
- Puede expandirse a través de librerías.

# ¿Quién usa Python?



# ¿Para qué se usa Python?

- Python se puede usar para las siguientes aplicaciones:
  - *Machine learning*
  - *Desarrollo de interfaces gráficos de usuario*
  - *Framework para páginas web como Django*
  - *Procesamiento digital de imágenes*
  - *Computación científica*

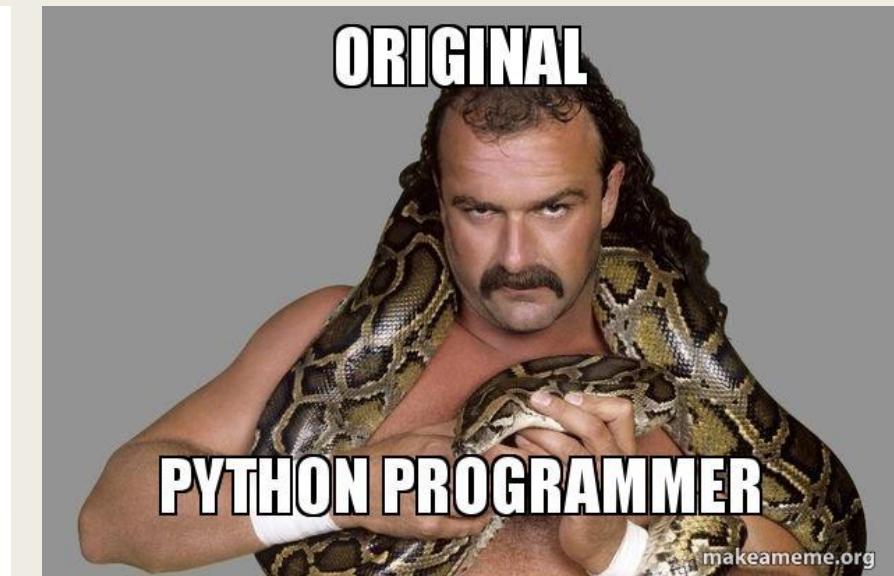
**PUTS A ; IN PYTHON**



**When you switch  
from C++ to Python**



**ORIGINAL**



**SI HARRY POTTER  
HABLA PARCEL...**



**ENTONCES,  
PROGRAMA EN PYTHON?**

# C++

# UNIX SHELL

Tú pedí una copia,  
no cuatrocientas.



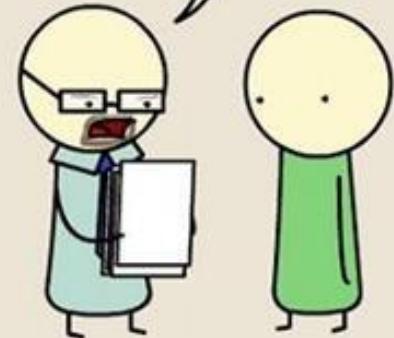
No tengo permiso  
para leer esto.



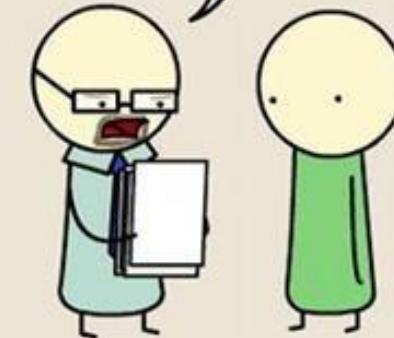
# PYTHON

# JAVA

Esto es plagio. No puedes hacer  
"import ensayo;" sin más.



Llevo dos páginas y todavía no  
sé qué es lo que quieres decir.



# LATEX

# HTML

Tu artículo no tiene el menor  
sentido, pero es la cosa más bonita  
que he visto en mi vida.



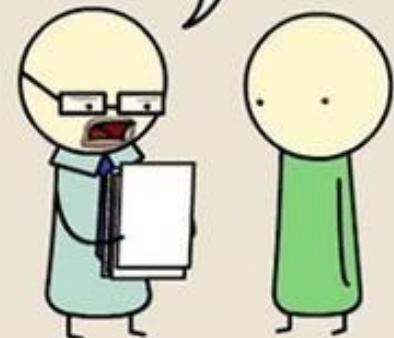
Esto es una maceta



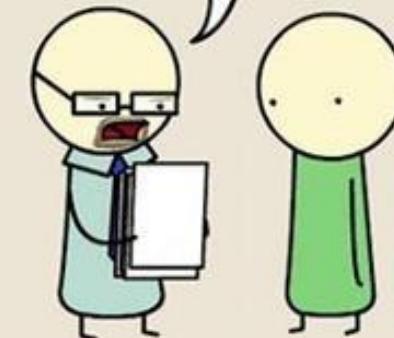
# ENSAMBLADOR

# C

¿En serio era necesario redefinir  
todas las palabras del español?



Esto es estupendo, pero has olvidado  
terminar con el carácter nulo. Ahora  
estoy leyendo un montón de basura.



# Python vs the world

C

```
#include  
  
int main(void)  
{  
puts("Hola mundo");  
}
```

C++

```
#include  
  
int main()  
{  
std::cout << "Hola mundo!";  
return 0;  
}
```

# Python vs the world

## Java

```
import javax.swing.JFrame;
import javax.swing.JLabel;
{
    public static void main(String[] args) {
        JFrame frame = new JFrame();          //Creating frame
        frame.setTitle("Hi!");
        frame.add(new JLabel("Hola mundo!"));
        frame.pack();
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }
}
```

## C#

```
using System;
class Program
{
    public static void Main(string[] args)
    {
        Console.WriteLine("Hello, world!");
    }
}
```

# Python vs the world

```
print('Hola mundo')
```

# CONOCIENDO PYTHON

# Instalando Python

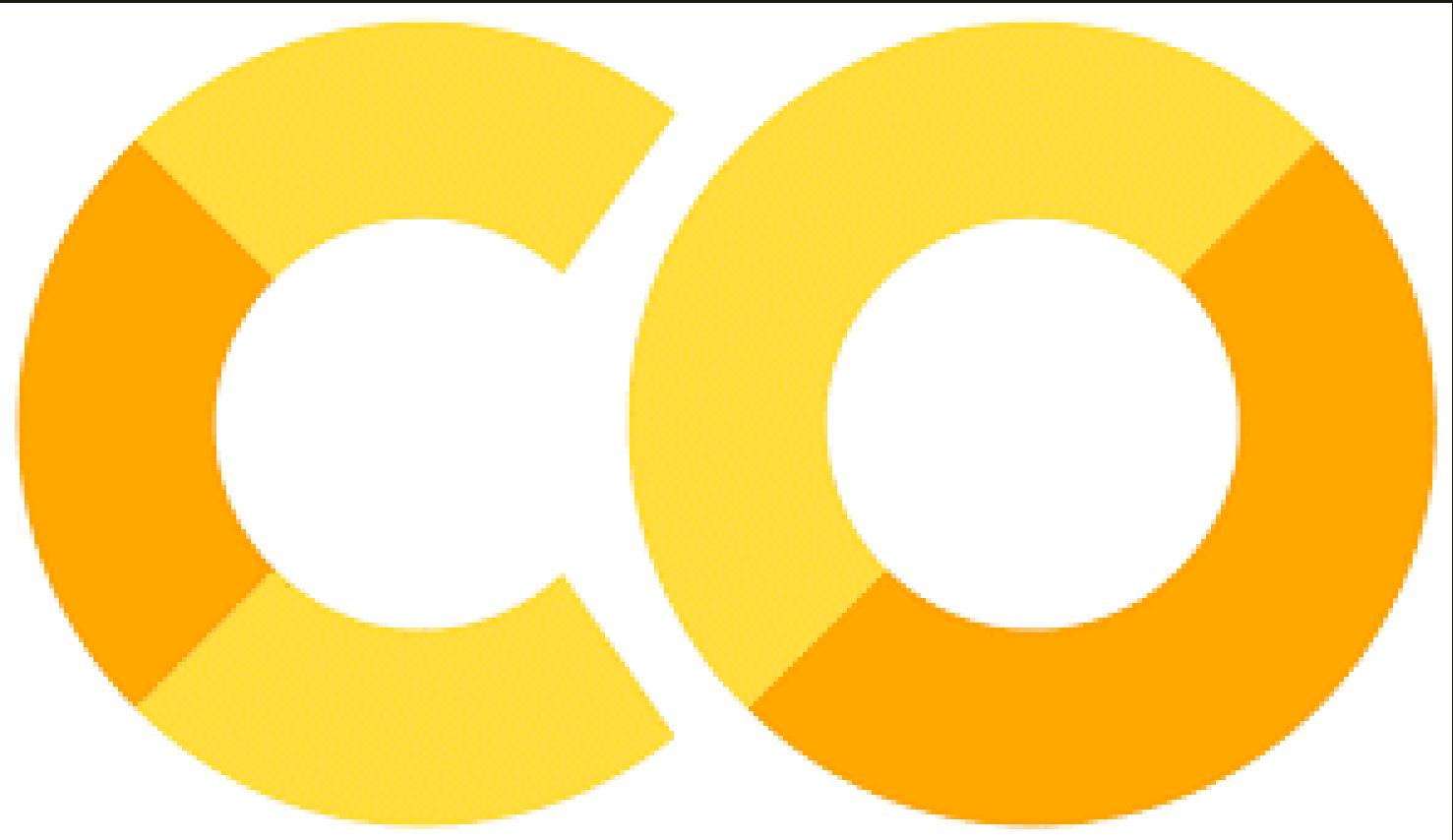
---

## De forma nativa

- Pueden instalar Python en su PC buscando el instalador en su página web.
- Ya viene preinstalado en Linux y MacOS.
- Interfaz poco amigable.

## En un entorno virtual

- Permite instalar distintas versiones y librerías.
- Entornos específicos para proyectos específicos.
- Mejor control de librerías.
- La forma más común de instalarlo es a través de Jupyter Notebooks y Anaconda.
- Interfaz sencilla y permite añadir texto complementario.



Google Colabs

# Google Colabs

- Colaboratory es una herramienta de investigación para la educación y la exploración del aprendizaje automático.
- Funciona a modo de bloc de notas de Jupyter que se puede usar sin ninguna configuración previa.
- Ya no hay que instalar algo.
- Viene con bastantes librerías externas precargadas.
- Funciona y se guarda en la nube, a través de Google Drive.

## Nombre del archivo

## Monitor de recursos

Untitled0.ipynb

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda

+ Código + Texto

Comentar

Compartir



RAM  
Disco

Edite

[1] 1+1

2

[2] print('Hola mundo')

Hola mundo

[3] print('En serio esto es gratis?')

En serio esto es gratis?

[ ]

Esto es una linea de texto, aquí no corre ningún código Como en la mayoría de editores de texto podemos dar formato al texto como **negritas** o *itálicas* entre otros

Líneas de código

Recursos

## Celda de texto

# Las reglas de Python

- Al igual que en los lenguajes humanos, los lenguajes de programación tienen sus propias reglas.
- Estas reglas se debe seguir si desean que sus programas se ejecuten sin errores.
- Por suerte estas reglas son pocas y muy sencillas.

```
32     self.fingerprints = set()
33     self.logdups = True
34     self.debug = debug
35     self.logger = logging.getLogger(__name__)
36
37     if path:
38         self.file = open(os.path.join(path,
39                         'fingerprint.log'), 'w')
40         self.file.seek(0)
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool('general.debug')
45         return cls(job_dir(settings), debug)
```

## Sintaxis básica de Python

```
46     def request_seen(self, request):
47         fp = self.request_fingerprint(request)
48         if fp in self.fingerprints:
49             return True
50         self.fingerprints.add(fp)
51
52         if self.file:
53             self.file.write(fp + os.linesep)
```

# OBJETOS Y ESTRUCTURA DE DATOS

# ASIGNACIÓN DE VARIABLES

# Qué es una variable

- En Python, como la mayoría de los lenguajes de programación, se puede asignar un valor arbitrario a una variable.
- Técnicamente hablando, una variable es un espacio reservado de memoria que guarda su valor asignado; este valor puede ser llamado desde cualquier parte del programa para su uso o manipulación.
- Las variables pueden contener cualquier tipo de objeto o estructuras.
- En Python no es necesario indicar qué tipo de dato contendrá una variable.

# Algunas restricciones para las variables

- No pueden comenzar con números.
  - *2x, 20pesos, 1.5litros*
- No puede haber espacios en el nombre, es recomendable usar guion (-) o guion bajo (\_) en su lugar.
- No puede contener ninguno de estos símbolos: “”<>/\?|()@\$%&\*+-
- No se pueden utilizar palabras clave como: for, while, if, else, help, or, and.

And	As	Assert	Break	Class
Continue	Def	Del	Elif	Else
Except	False	Finally	For	From
Global	If	Import	In	Is
Lambda	None	Nonlocal	Not	Or
Pass	Raise	Return	True	Try
While	With	yield		

# Palabras clave

# Otra cosa más...

Python usa una filosofía de tecleado dinámico, por lo que los valores de las variables pueden ser reasignados sobre la marcha.

Tener cuidado de repetir los nombres de una variable, puede ser que pierdan información valiosa.

```
>>> x = 5
>>> y = 10
>>> c = x + y
>>> print(c)
15

>>> x = 10
>>> print(c)
15

>>> c = x + y
>>> print(c)
20
```

# Tipos de datos

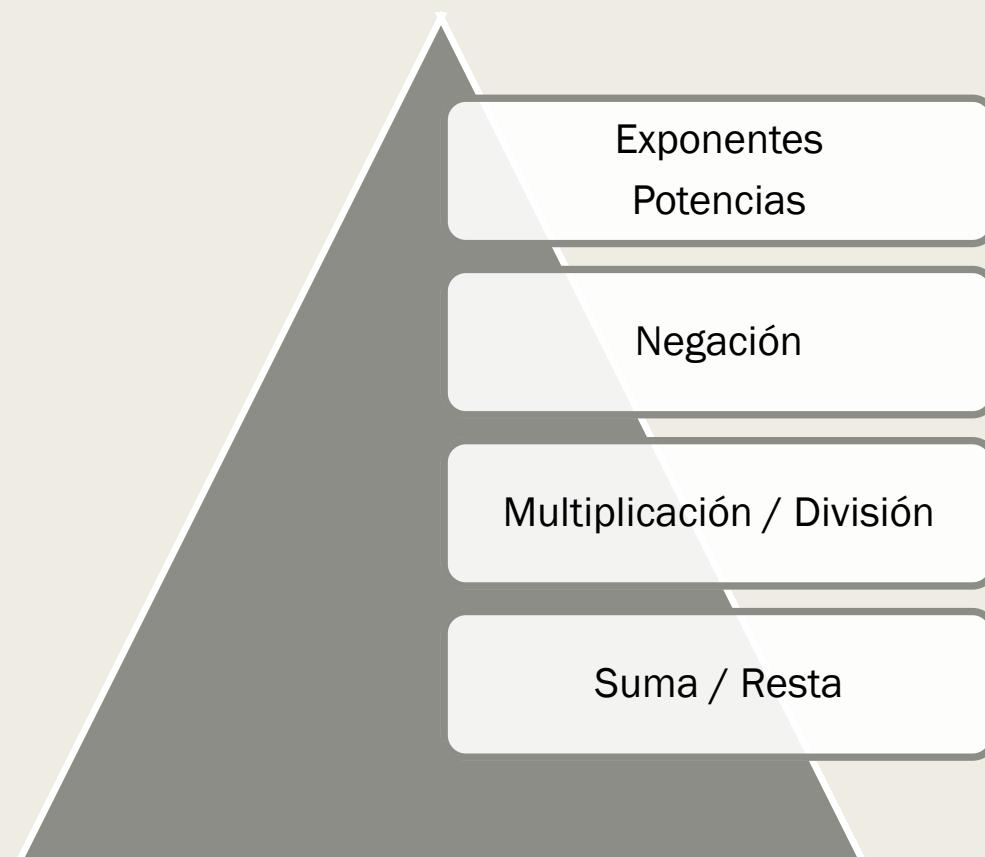
Nombre	Tipo	Descripción	Ejemplo
Enteros	int	Cualquier número entero positivo o negativo	3, 1000, -2
Flotantes	float	Números con punto decimal	5.8, 0.00001, 1.0
Texto/ Caracteres	str	Secuencia de caracteres, pueden ser números, letras o ambos	“Hola”, “Python”, ‘100’
Listas	list	Secuencia de objetos, puede contener números, strings o ambos	[10, ‘hola’, 100.5, ‘25’]
Diccionarios	dict	Secuencia de objetos, no ordenada, dados en pares	{"c1": "valor", "nombre": "Juan"}
Tuplas	tup	Secuencia de objetos ordenada no inmutable	(10, “hola”, 100.5, ‘25’)
Sets	set	Colección de datos no ordenada y no indexada	{"fruta", "comida", "platos"}
Booleano	bool	Valores lógicos	True & False

# Números

- Puede ser cualquier número.
- Los números con decimales son considerados flotantes (**float**).
- Se puede realizar cualquier operación aritmética.
- En otras palabras, Python puede funcionar como una calculadora.

Operador	Nombre	Ejemplo
+	Suma	$1 + 1$
-	Resta	$3 - 2$
*	Multiplicación	$3 * 4$
/	División	$10 / 4$
%	Módulo	$20 \% 5$
**	Potencia	$10^{**3}$
//	División entera ( <i>floor división</i> )	$15 // 2$

# Jerarquía de operaciones



```
31     def __init__(self, file=None, fingerprints=None, logduplicates=True, debug=False):
32         self.file = file
33         self.fingerprints = fingerprints
34         self.logduplicates = logduplicates
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, "request.log"), "w")
39             self.file.seek(0)
40             self.fingerprints.update([os.linesep])
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool("SUPERVISOR_DEBUG")
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)
```

## Práctica 1: Usando Python como calculadora

# Strings

- Las strings (**str**) se componen de cualquier carácter (ya sea número, letra o símbolo) siempre y cuando estén entre “” o ‘’.
- Para visualizar una string se puede usar la función *print*, o también se puede asignar a una variable.
  - *print('Hola mundo')*
  - *msg = 'hola mundo'*
- Si se quiere ingresar una string multilínea se debe usar tres comillas dobles.
  - *msg = """El veloz murciélagó hindú comía cardillo y kiwi"""*
  - *Si la string que se utilizará contiene un apostrofe (') se debe usar las comillas (""").*
    - *print("I don't do that")*

# Operaciones con strings

- Podemos realizar algunas operaciones con el tipo de dato string.
- Las únicas operaciones válidas son la suma y la multiplicación.
- Sin embargo, hay otras formas de manejar a las strings.

```
>>> a = 'hola'
>>> a+a
holahola'
>>> a*2
```

# Dando formato a nuestras strings

- Ya vimos como trabajar con las strings, sin embargo, el texto que ingresamos se despliega tal como está.
- Esto es funcional más no presentable, por suerte, existe una forma de dar formato a las strings.
- La forma más sencilla de hacer esto es usando *placeholders*.
- Un *placeholder* se declara con el operador modulo (%), este símbolo indicará a Python que en esa posición habrá un objeto.

Operador	Tipo de dato
%d	Enteros
%f	Flotantes
%b	Binarios
%o	Octal
%x	Octal Hexadecimal
%s	Strings
%e	Números exponenciales

# Manipulación de strings

- En ciertas ocasiones puede resultar conveniente obtener un solo carácter o una parte de la string, por ejemplo. Obtener la primera letra o valor.
- Para este caso se utilizan las operaciones de indexación (indexing) y slicing.
  - *Indexación*: tomar solo un carácter de la string.
  - *Slicing*: tomar una subsecuencia de caracteres de la string.
- La cuenta inicia desde 0, no desde 1.
- También se puede consultar si existe un carácter en la string.
- Nota: los espacios cuentan.

```
31     def __init__(self, path=None, debug=False):
32         self.file = None
33         self.fingerprints = set()
34         self.logduplicates = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, "fingerprint.log"), "a+")
39             self.file.seek(0)
40             self.fingerprints.update(self._read_file())
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool("GENERAL.DEBUG")
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)
```

## Práctica 2: Manipulación y formateo de strings

# ARRAYS

# Listas

- Una lista es una colección de objetos que es ordenada y puede cambiar el orden de sus elementos.
- Las listas pueden tener todos los tipos de datos.
- Se puede realizar operaciones aritméticas con ellas como la suma o multiplicación.

```
>>> lst = [1, 2, 3, 4.5, 5.5]
>>> lst
[1, 2, 3, 4.5, 5.5]
>>> lst2 = ['a', 1, 'b', 2, 'c', 3]
>>> lst2
['a', 1, 'b', 2, 'c', 3]
>>> l_bool = [True, False, True, True]
>>> l_bool
[True, False, True, True]
>>> n_lst = lst + lst2 + l_bool
>>> n_lst
[1, 2, 3, 4.5, 5.5, 'a', 1, 'b', 2, 'c', 3, True, False, True, True]
>>> lst[1]
2
>>> n_lst[5]
'a'
>>> lst[3] = 4
>>> lst
[1, 2, 3, 4, 5.5]
```

# Manipulando Listas

- Los elementos de las listas se pueden agregar, remover o modificar.
- Estas operaciones se ejecutan mediante los métodos de Python.
- La sintaxis es:  
variable.método()

Método	Descripción
append()	Agrega un elemento al final de la lista
clear()	Quita todos los elementos de la lista
copy()	Regresa una copia de la lista
remove()	Quita el primer elemento del valor especificado
pop()	Quita el elemento en el índice especificado

```
>>> n_lst.append("agregado")
>>> n_lst
[1, 2, 3, 4.5, 5.5, 'a', 1, 'b', 2, 'c', 3, True, False, True, True, 'agregado']
>>> n_lst.remove(1)
>>> n_lst
[2, 3, 4.5, 5.5, 'a', 1, 'b', 2, 'c', 3, True, False, True, True, 'agregado']
>>> n_lst.pop()
'agregado'
>>> n_lst
[2, 3, 4.5, 5.5, 'a', 1, 'b', 2, 'c', 3, True, False, True]
>>> n_lst.pop(-1)
True
>>> n_lst
[2, 3, 4.5, 5.5, 'a', 1, 'b', 2, 'c', 3, True, False, True]
```

```
31     def __init__(self, path=None, debug=False):
32         self.file = None
33         self.fingerprints = set()
34         self.logduplicates = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, "fingerprint.log"), "a+")
39             self.file.seek(0)
40             self.fingerprints.update(self._read_file())
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool("GENERAL.DEBUG")
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)
```

# Tuplas

- Una **tupla** es una colección de datos que están ordenados y **no son intercambiables**.
- Las tuplas son declaradas con paréntesis () .
- Se pueden acceder a los elementos al igual que las listas.

```
>>> tupla = ("manzana","fresa","kiwi","pera")
>>> print(tupla)
('manzana', 'fresa', 'kiwi', 'pera')
>>> print(tupla[1]) # Mostramos el segundo elemento
fresa
>>> print(tupla[-1]) # Desplegamos el último elemento
pera
>>> print(tupla[1:4])
('fresa', 'kiwi', 'pera')
>>> "manzana" in tupla
True
>>> "limon" in tupla
False
>>> len(tupla) # Muestra la longitud de la tupla
4
```

# Diccionarios

- Es una colección de datos que no tiene un orden pero si un **identificador**.
- Sus elementos pueden **cambiar** y ser **indexados**.
- Se declaran mediante laves { }.
- Para cada elemento **debe** existir una **key** indicando qué elemento se está haciendo referencia.
- Se pueden manipular al igual que las listas.

```
1 # Declaramos un diccionario
2 orquesta = {
3     "vientos": "flauta",
4     "metales": "trompeta",
5     "cuerdas": "violin",
6     "percusiones": "tambor"
7 }
```

```
1 # Diccionario con valores repetidos
2 planetas = {
3     "rocosos": "mercurio, venus, tierra, marte",
4     "gaseosos": "neptuno, jupiter, urano, saturno",
5     "anillados": "saturno, urano, neptuno, jupiter"
6 }
7
8 print(planetas)
```



```
1 ### Referenciaremos los valores de cada diccionario
2 print(orquesta["metales"])
3 print(planetas["rocosos"])
```



trompeta  
mercurio, venus, tierra, marte

```
[ ] 1 ### También se pueden crear diccionarios que contengan otros diccionarios
2
3 info1 = {
4     "artista": "acdc", "genero": "rock"
5 }
6
7 info2 = {
8     "artista": "abba", "genero": "pop"
9 }
10
11 info3 = {
12     "artista": "daft punk", "genero": "electronica"
13 }
14
15 musica = {
16     "artista 1": info1,
17     "artista 2": info2,
18     "artista 3": info3
19 }
20
21 print(musica)

{'artista 1': {'artista': 'acdc', 'genero': 'rock'}, 'artista 2': {'artista': 'abba', 'genero': 'pop'}, 'artista 3': {'artista': 'daft punk', 'genero': 'electronica'}}
```

# OPERADORES BOOLEANOS Y DE COMPARACIÓN

# Operadores de comparación

- Sirven para evaluar si dos valores o sentencias con iguales, mayores, menores o distintos.
- Pueden establecer relaciones para manipular el flujo de un programa.
- Pueden ser aplicado a distintos tipos de datos.
- El resultado siempre será del tipo booleano.

Operador	Nombre	Ejemplo
<code>==</code>	Es igual a	<code>x == y</code>
<code>!=</code>	No es igual. Es distinto de	<code>x != y</code>
<code>&gt;</code>	Mayor que	<code>x &gt; y</code>
<code>&lt;</code>	Menor que	<code>x &lt; y</code>
<code>&gt;=</code>	Mayor o igual que	<code>x &gt;= y</code>
<code>&lt;=</code>	Menor o igual que	<code>X &lt;= y</code>

# Operadores booleanos

- Los operadores booleanos vienen desde el álgebra booleana.
- Agrupan términos en combinaciones lógicas.
- En esencia sólo son tres: **and**, **or** y **not**.
- El resultado de una operación booleana puede regresar dos valores: **True** o **False**.

# Tablas de verdad

Compuerta Or

Entrada		Salida
A	B	S
0	0	0
0	1	1
1	0	1
1	1	1

Compuerta And

Entrada		Salida
A	B	S
0	0	0
0	1	0
1	0	0
1	1	1

```
31     def __init__(self, path=None, debug=False):
32         self.file = None
33         self.fingerprints = set()
34         self.logduplicates = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, "fingerprint.log"), "a+")
39             self.file.seek(0)
40             self.fingerprints.update(self._read_file())
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool("GENERAL.DEBUG")
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)
```



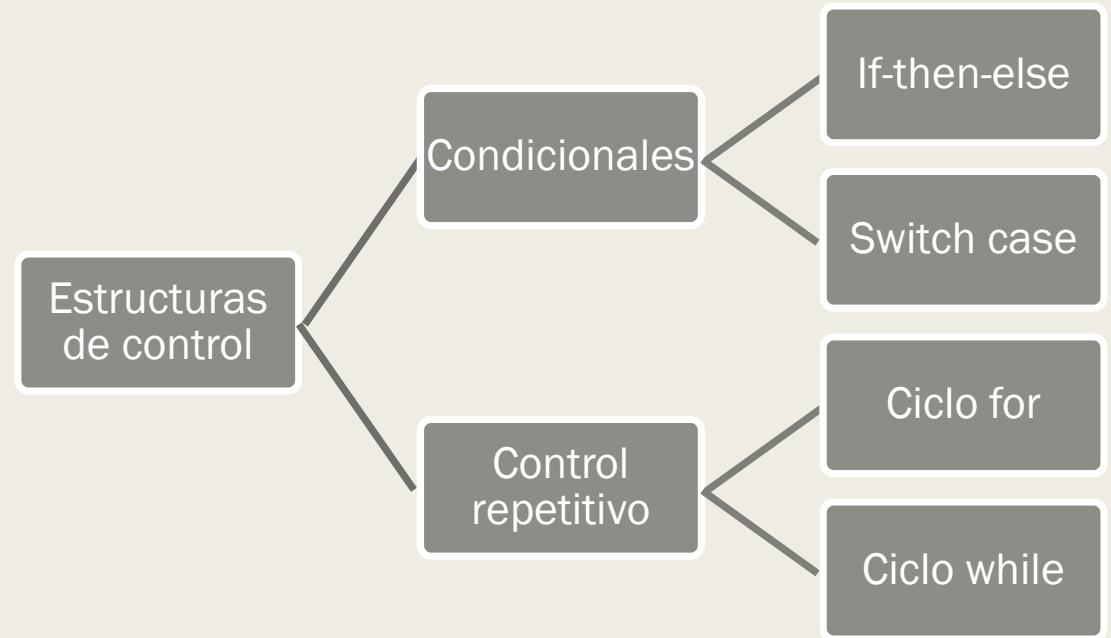
ArmaTuCoso.com

Examen sorpresa!

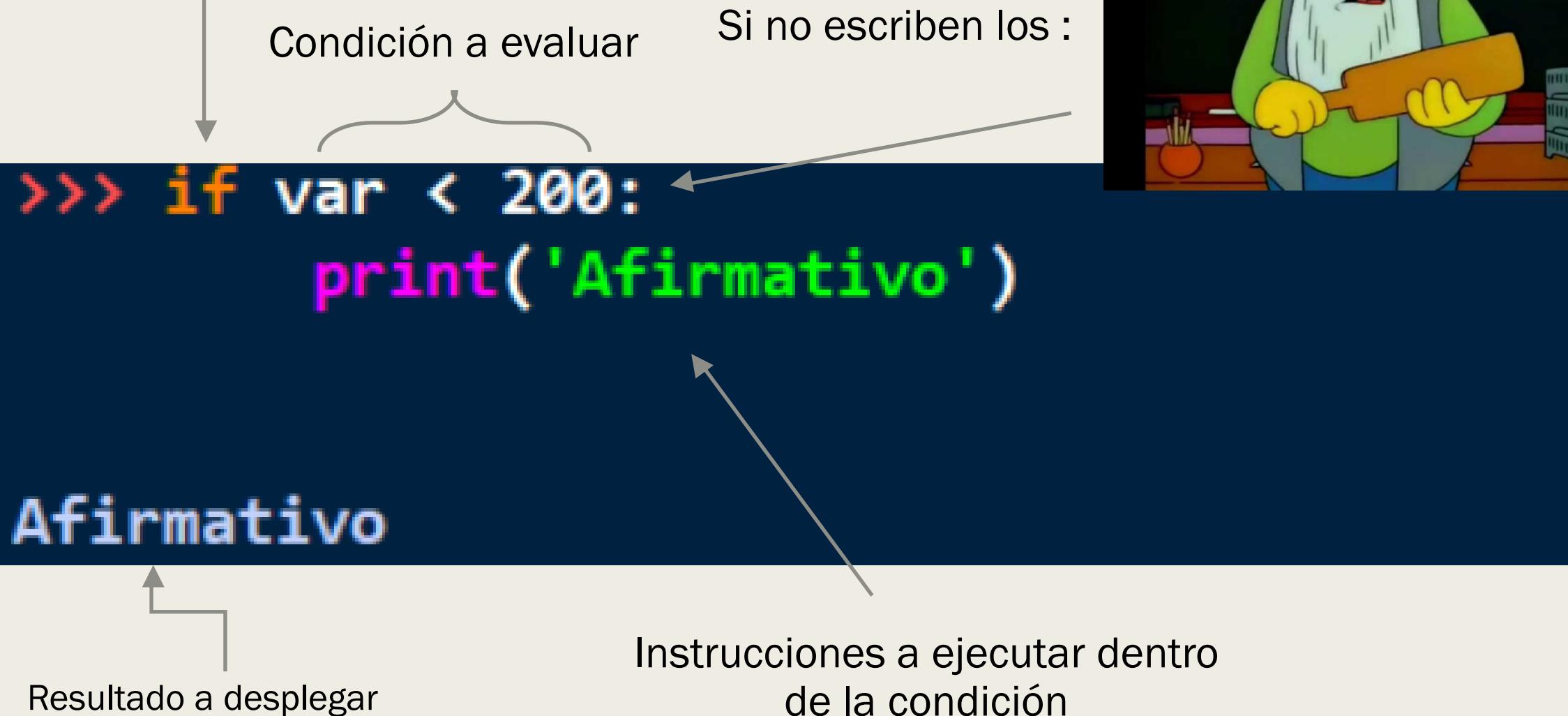
# ESTRUCTURAS DE CONTROL

# Condicionales

- En programación, se conoce como **estructuras de control** a las herramientas que permiten modificar el flujo de trabajo del programa.
- Son útiles para dirigir al programa hacia donde queramos.
- Se pueden dividir en dos grupos: **condicionales** y **control repetitivo**.



## Palabra clave *if*



# If & Elif

- Python soporta todos los operadores de comparación.
- Hay que realizar la indentación después de la sentencia.
- Se puede usar la palabra clave **elif** para reevaluar una condición. Esto se interpretaría como *si la condición previa no se cumple, entonces intentar esto.*
- La palabra clave **else** indica realizar una acción si la condición no se cumplió.

```
[1] a = 33  
    b = 33  
  
[2] if b > a:  
    print("b es mayor que a")  
elif a == b:  
    print("a y b son iguales").
```

↳ a y b son iguales

Lógica de programación hijos!!!



# Un poco de filosofía...

- Si la idea es fácil de explicar entonces es una buena idea.
- Si la idea es difícil de explicar entonces es una mala idea.
- Dividan y vencerán.
- Sigan la regla KISS (Keep It Single Stupid).
- La solución más sencilla casi siempre será la solución correcta.

# Control repetitivo

---

## Ciclo **for**

- Se utiliza para iterar sobre una secuencia de datos.
- A diferencia del *while* éste ciclo itera sobre todos los elementos de la colección, no mientras la condición sea verdadera.
- Se pueden ejecutar líneas de código mientras el ciclo esté activo.
- Si se quiere iterar con números enteros se debe añadir la palabra clave **range()** antes del número.

## Ciclo **while**

- Este ciclo se ejecuta siempre y cuando se cumpla la condición.
- Se puede interrumpir usando el comando **break**.
- Se debe agregar o quitar un valor a la variable contadora.

¿Qué pasa si no agregamos la variable contadora?



**TU CICLO  
WHILE TODO HERMOSO**

**SIN I++**



Recuerden amigos, si no escriben bien la sintaxis les marcará error



Hasta la próxima!!!

```
31     def __init__(self, path=None, debug=False):
32         self.file = None
33         self.fingerprints = set()
34         self.logduplicates = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, "fingerprint.log"), "a+")
39             self.file.seek(0)
40             self.fingerprints.update(self._read_file())
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool("GENERAL.DEBUG")
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)
```

```
31     def __init__(self, path=None, debug=False):
32         self.file = None
33         self.fingerprints = set()
34         self.logduplicates = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, "fingerprint.log"), "a+")
39             self.file.seek(0)
40             self.fingerprints.update(self._read_file())
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool("GENERAL.DEBUG")
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)
```

# EXPANDIENDO PYTHON CON LIBRERÍAS

# Cómo usar e instalar librerías

- Python puede usar una gran cantidad de librerías con las cuales puede adquirir más flexibilidad y poder de procesamiento.
- En nuestro caso, Google Colabs cuenta con bastantes librerías precargadas.
- Para instalar una librería se utiliza el comando *pip* y para conocer qué librerías tenemos instaladas se utiliza la instrucción *pip list*.
- Para indicar a Python que queremos utilizar una o varias librerías es necesario usar la palabra clave *import* seguida del nombre de la misma.

**YOU FOOL**

**THIS ISN'T EVEN MY  
FINAL FORM**

# Numpy

- Librería especializada para cómputo numérico científico y avanzado.
- Ideal para trabajar con matrices y arreglos de datos (arrays).
- Manejo de operaciones de álgebra lineal, matricial, transformadas de Fourier, entre otras.
- Su manejo (e incluso algunas funciones) es similar al software Matlab.

# Matplotlib

- Colección de comandos para realizar gráficas elegantes y entendibles.
- Se pueden hacer gráficas de barras, histogramas, de pastel, entre otras.
- Se pueden editar aspectos como los ejes, la escala, el título de la gráfica, color, leyendas.

# Pandas

- Es una biblioteca de software especializada en manejo de base de datos y su análisis.
- Ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales.
- Trabaja principalmente con objetos conocidos como DataFrames, que permiten almacenar y manipular los datos contenidos a modo de tablas.
- Permite crear bases de datos desde cero hasta importar archivos (hojas de cálculo, blocs de notas, csv) completos para su trabajo.
- Permite crear bases de datos alternas a la original.

# SkLearn

- Es una herramienta para realizar análisis de datos y algunas técnicas de machine learning.
- También ofrece formas de evaluar modelos de aprendizaje automático.
- Sirve para preprocesamiento de datos y aplicar normalizaciones.
- También tiene precargadas algunas bases de datos para utilizarlas.

# FUNCIONES, DEFINICIONES Y MÉTODOS

# Funciones propias de Python

- Python trae incluidas varias funciones que pueden ser útiles.
- Para usar estas funciones debemos seguir la siguiente sintaxis:
  - *funcion(variable)*

# Algunos ejemplos...

Nombre	Funcionamiento	Ejemplo
abs()	Devuelve el valor absoluto de cualquier número	abs(x)
bin(x)	Convierte un entero a binario añadiendo un prefijo '0b'	bin(x)
enumerate()	Devuelve un objeto enumerado.	enumerate(x)
len()	Devuelve la longitud (número de elementos) de un objeto.	len(x)
max(), min(x)	Devuelve el valor máximo o mínimo de un objeto	max(x), min(x)
print()	Despliega la información contenida en los paréntesis	print(x)
range()	Crea un secuencia de números. Se puede indicar el inicio, el fin y los incrementos.	range(x,y) range(x,y,z)

# Funciones personalizadas (Scripts)

- Un **script** son varias líneas de código que se ejecutan **una sola vez** al ser llamadas.
- Un script puede contener una o varias **funciones**.
- Una **función** siempre regresa un **resultado**.
- Para declarar un función se usa la palabra clave **def**.

Palabra clave **def**

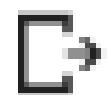
Nombre de la función

Parámetros

```
[19] def my_function():
        print("Hola mundo desde una función")
```

```
[20] my_function()
```

Instrucciones de la función



Hola mundo desde una función

# Parámetros

- Se puede pasar información a la función a través de **parámetros**.
- Son especificados después del nombre de la función, dentro de los **paréntesis**.
- También se puede asignar un parámetro **default**.

```
[24] def my_function(fname):  
    print(fname + " Hernández")
```

```
[25] my_function("Angel")  
my_function("Moisés")  
my_function("Tomás")  
my_function("Luis")
```

⇒ Angel Hernández  
Moisés Hernández  
Tomás Hernández  
Luis Hernández

```
[26] def nacionalidad(pais= "México"):  
    print("Yo soy de " + pais)
```

```
[27] nacionalidad("EEUU")  
nacionalidad("Francia")  
nacionalidad("Japón")  
nacionalidad()
```

⇒ Yo soy de EEUU  
Yo soy de Francia  
Yo soy de Japón  
Yo soy de México

# La palabra clave return()

```
1 x = 100
2 y = 300
3
4 def suma1(x,y):
5     print('El resultado de la suma es: ', x+y)
6
7 def suma2(x,y):
8     s = x + y
9     return(s)

1 suma1(x,y)
El resultado de la suma es: 400

1 suma2(x,y)
400
```

# \*args

- Es una **sintaxis especial** que nos permite pasar cualquier número de parámetros a una función.
- Por convención se suele usar `*args`, pero puede ser reemplazada por cualquier palabra, siempre y cuando esté después de un **asterisco(\*)**.

```
>>> def myfunc(*args):
    print(args)

>>> myfunc('hola','mundo','estoy','vivo','muajajaja')
('hola', 'mundo', 'estoy', 'vivo', 'muajajaja')
>>> def myfunc2(*args):
    for arg in args:
        print(arg)

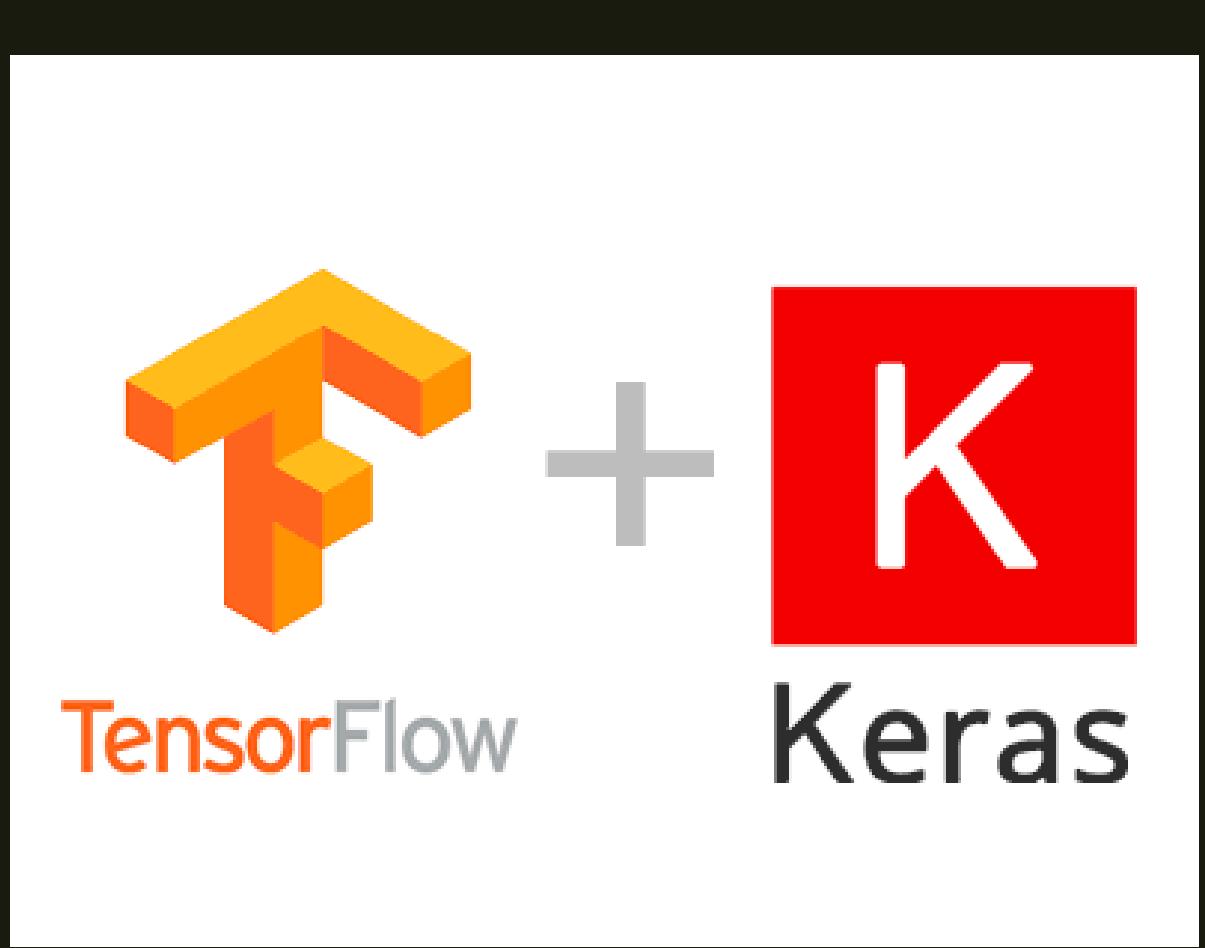
>>> myfunc2('hola','mundo','estoy','vivo','muajajaja')
hola
mundo
estoy
vivo
muajajaja
>>> def suma(*args):
    print(sum(args))

>>> suma(1,2,3,4,5,6,7,8,9)
45
```

# Algunos tips...

- Definan muy bien y antes de comenzar qué hará exactamente su función.
- Procuren dar siempre una descripción del funcionamiento y ejemplos; nunca sabremos quién podría leer nuestro código.
- Elijan un nombre acorde al funcionamiento de la función.
- Si su función hará dos o mas cosas distintas, mejor divídanlas. **Dividan y vencerán.**

```
31     def __init__(self, path=None, debug=False):
32         self.file = None
33         self.fingerprints = set()
34         self.logduplicates = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, "fingerprint.log"), "a+")
39             self.file.seek(0)
40             self.fingerprints.update(self._read_file())
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool("GENERAL.DEBUG")
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)
```



# TENSORFLOW & KERAS

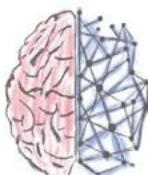
Un vistazo a la  
inteligencia artificial y  
machine learning

# ¿Qué es la inteligencia artificial

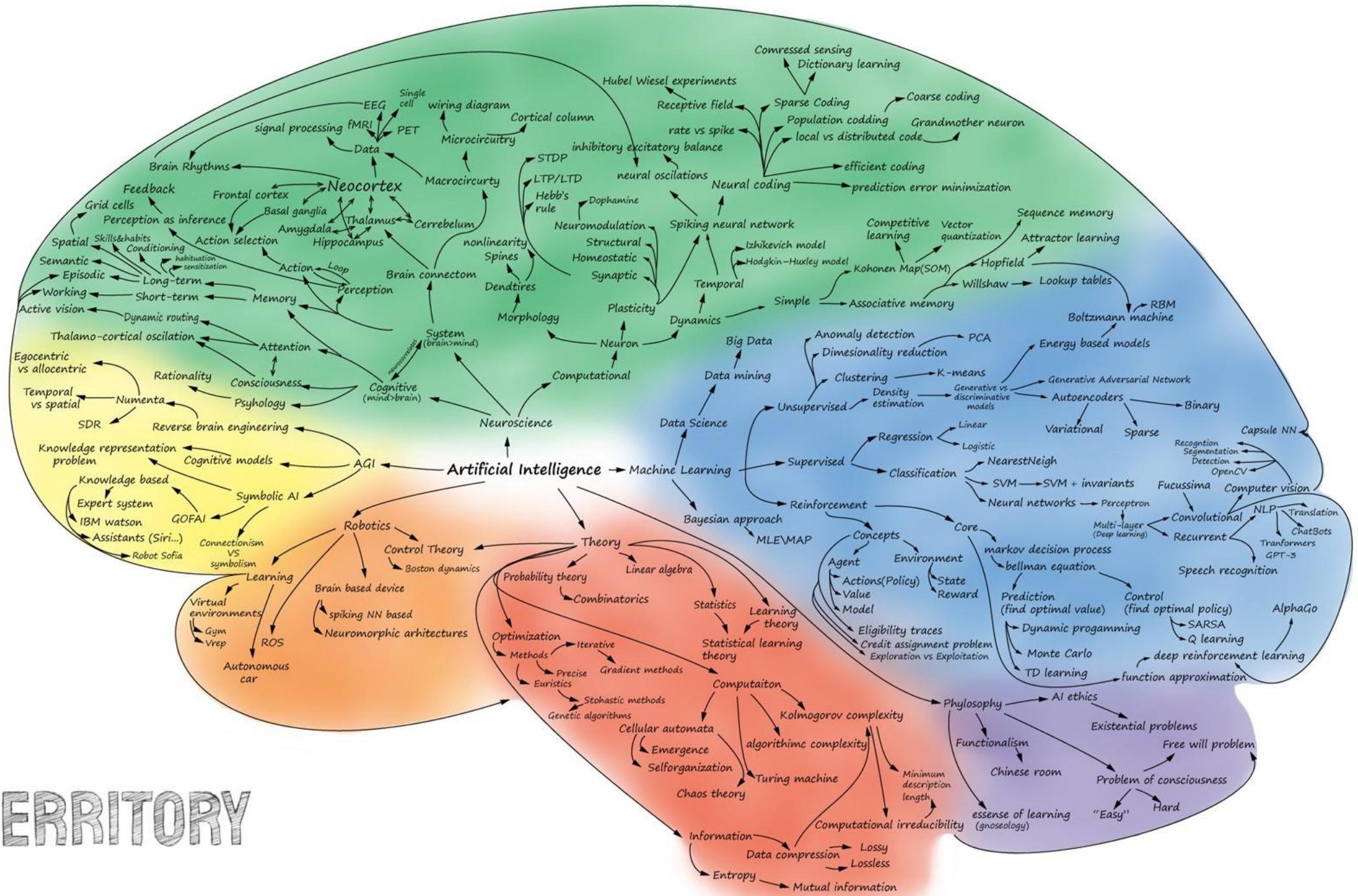
- “La ciencia e ingeniería de hacer máquinas inteligentes” — John McCarthy
- “Rama de las ciencias de la computación que se encarga de la simulación de comportamientos inteligentes en las computadoras”
- “La capacidad de las máquinas de imitar el comportamiento humano inteligente”

# Tipos de inteligencia artificial

- Autómatas
- Máquinas de memoria limitada
- Máquinas reactivas
- Sistemas expertos
- Machine Learning



# AI TERRITORY



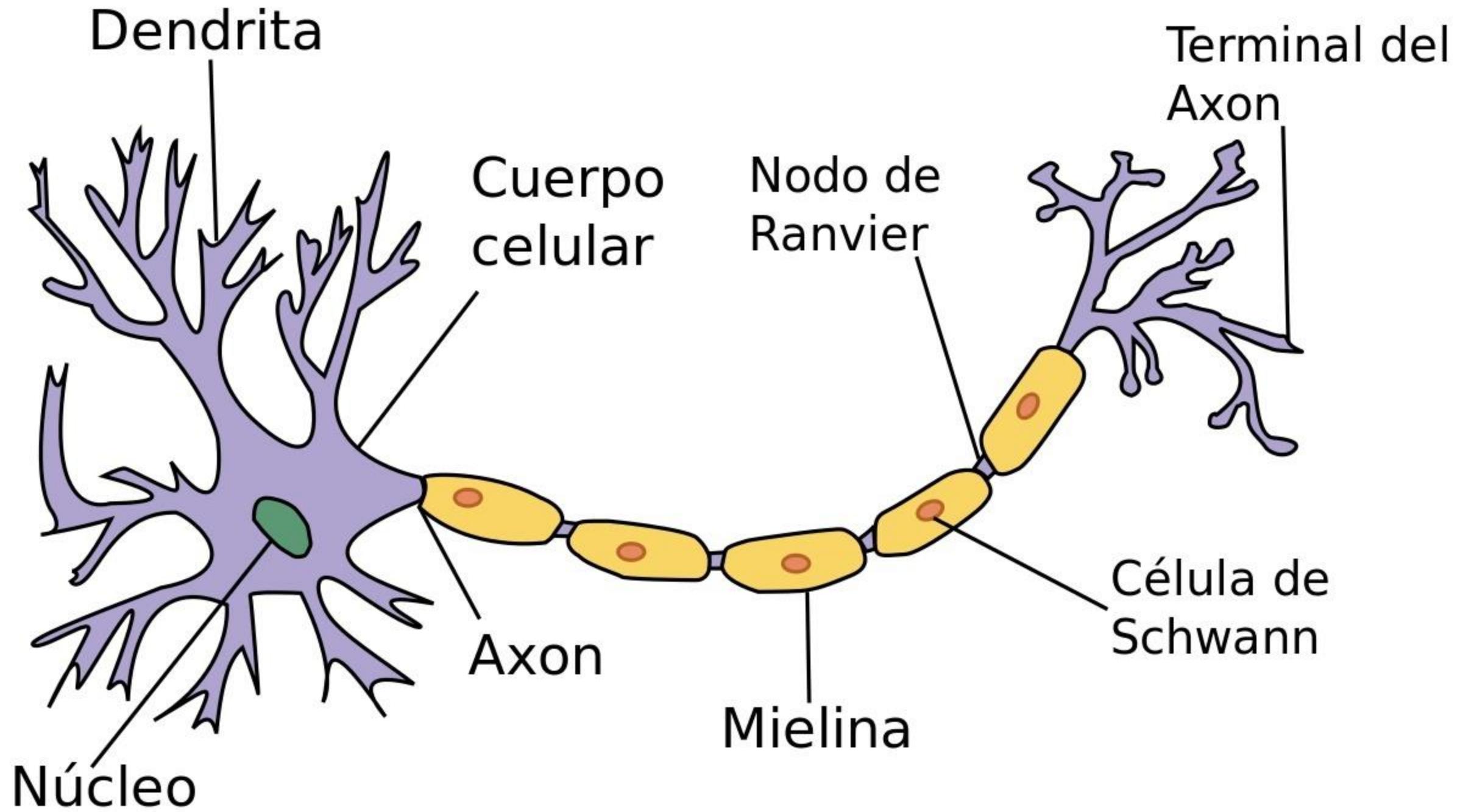
# ¿Qué es machine learning?

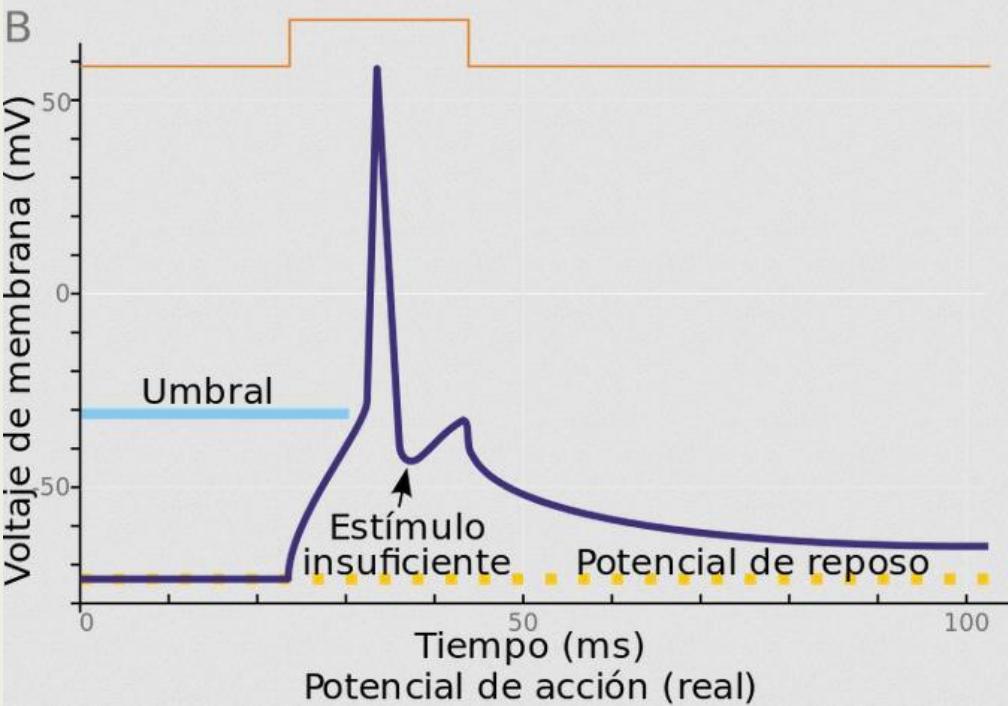
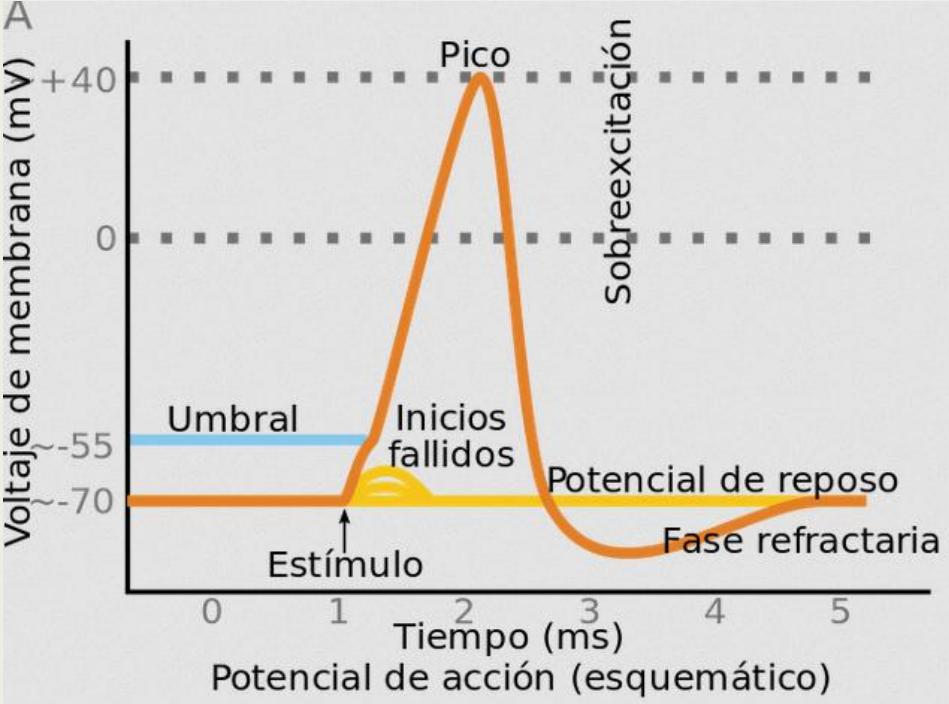
- Rama de la inteligencia artificial
- Es el campo de estudio que dota a las computadoras la habilidad de “aprender” sin ser expertamente programable.
- Modelos matemáticos dinámicos.
- Fuertemente basados en modelos estadísticos.
- Se pueden “adaptar” a las circunstancias.

*Todo el machine learning es  
inteligencia artificial, pero no toda  
la inteligencia artificial cuenta como  
machine learning*

# ¿Cómo aprende el ser humano?

- Dentro del cerebro hay cerca de 10 millones de neuronas.
- Cada neurona percibe las señales de nuestro entorno y las procesa de forma que las podamos entender.
- Cada conexión entre neuronas se llama sinapsis.
- Cuando el cerebro aprende, toma decisiones, siente emociones libera químicos y pulsos eléctricos que transportan esas señales.





# COMPARANDO PARADIGMAS DE MACHINE LEARNING

# Support Vector Machine (SVM)

- Es de aprendizaje supervisado
- Se puede usar para problemas de clasificación como de regresión.
- El método de SVM intenta dibujar un hiperplano que pueda separar óptimamente los grupos.
- Si no hay “separabilidad” en los grupos, se pueden proyectar las características (atributos) a un nuevo “espacio” donde sean separables.
- Pros
  - *Ahorra memoria*
  - *Ideal para problemas de clasificación*
  - *Eficiente cuando los grupos tienen muchos atributos*
- Contras
  - *Poco eficiente en datasets largos o con muchos elementos*
  - *Tiempo de entrenamiento largo*
  - *Propenso a los outliers*

# Decision Trees

- Es un método de lenguaje supervisado.
- Se organiza de manera jerárquica y se constituye de ramas y hojas.
- El primer nodo (decisión) se le conoce como raíz.
- Puede trabajar con datos numéricos y categóricos.
- Pros
  - *Fácil de interpretar*
  - *Trabaja bien con datasets largos*
  - *No sensible a outliers*
  - *Flexibilidad con los datos*
- Contras
  - *Overfitting constante*
  - *El resultado final depende de la primer decisión.*
  - *Puede que la precisión sea muy baja en algunos casos*

# kNN

- Algoritmo de aprendizaje no supervisado.
- Se usa para problemas de clasificación y predicción.
- Usa la cercanía de grupos cercanos (vecinos) para agrupar los datos.
- Pros
  - *Fácil de interpretar*
  - *Útil para clasificación multiclase*
  - *Bueno para datasets con pocos atributos*
- Contras
  - *Sensible a outliers*
  - *Lento computacionalmente cuando hay muchos ejemplos con muchos atributos*
  - *Se debe declarar un número de vecinos k*

# Redes neuronales artificiales (ANN)

- El aprendizaje puede ser supervisado o no supervisado.
- Funciona con base a modelos matemáticos y probabilísticos.
- Usa el perceptrón como unidad básica de aprendizaje
- Se puede adaptar a varios problemas con pocos ajustes.
- Pros
  - *Adaptabilidad.*
  - *No es sensible a outliers.*
  - *Puede trabajar con imágenes, series de tiempo, audio, entre otras.*
- Contras
  - *Necesita enormes cantidades de datos para obtener resultados “aceptables”.*
  - *Alto costo computacional*
  - *Tiempos de entrenamiento largos cuando trabaja con datos especiales.*
  - *El funcionamiento es una “caja negra”*



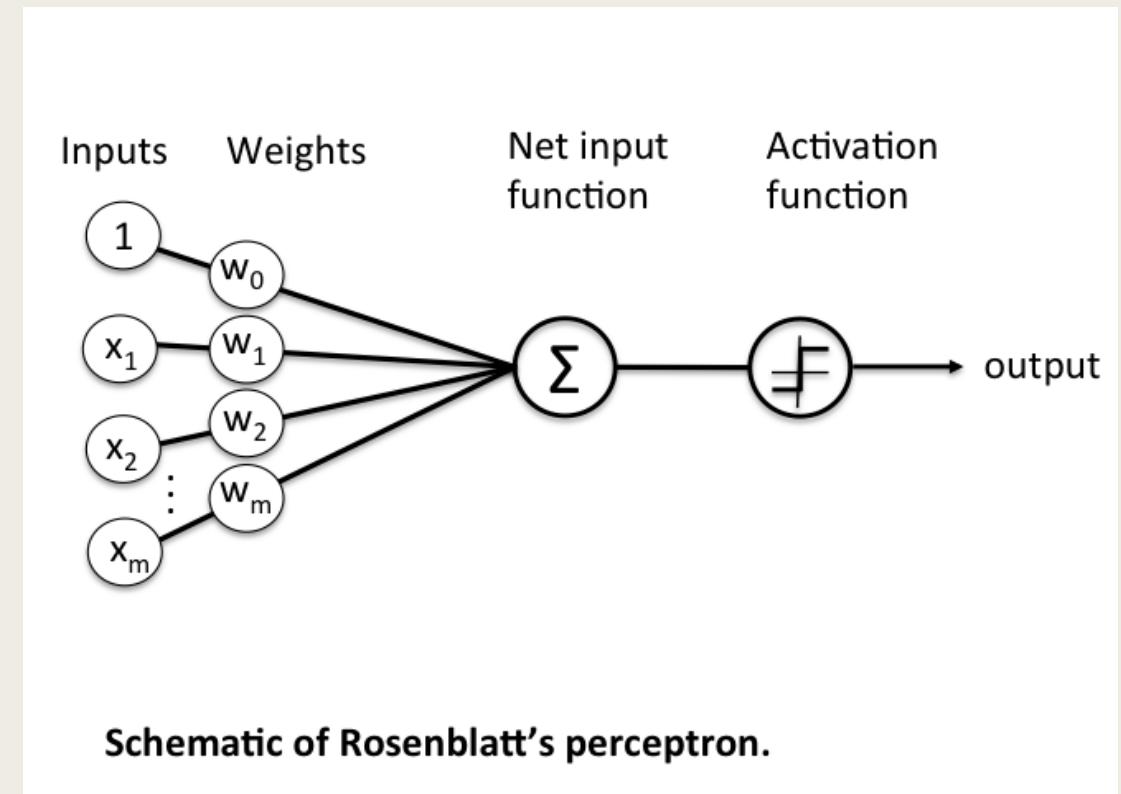
# REDES NEURONALES ARTIFICIALES

# Redes neuronales artificiales (ANN)

- Las redes neuronales (neural networks, NN por sus siglas en inglés) son un tipo de machine learning, que usan aproximaciones de funciones matemáticas para resolver problemas.
- Están basadas en el modelo biológico de una neurona humana.
- La forma básica usada es el perceptrón.
- Algunos tipos de redes neuronales son:
  - *Convolucionales (Convolutional Neural Networks)*
  - *Recurrentes (Recurrent Neural Networks)*
  - *GAN's (Generative Adversarial)*
  - *LSTM's (Long-Short Term Memory)*

# Estructura de un perceptrón

- La unidad básica de una NN.
- Consta de:
  - Entradas
  - Pesos
  - Sesgos (bias)
  - Función de activación
  - Salida



Modelo de perceptrón de Rosenblat, 1959

# Ecuación del perceptrón

$$y = \sigma(wx + b)$$

Perceptrón simple

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

Perceptrón multicapa

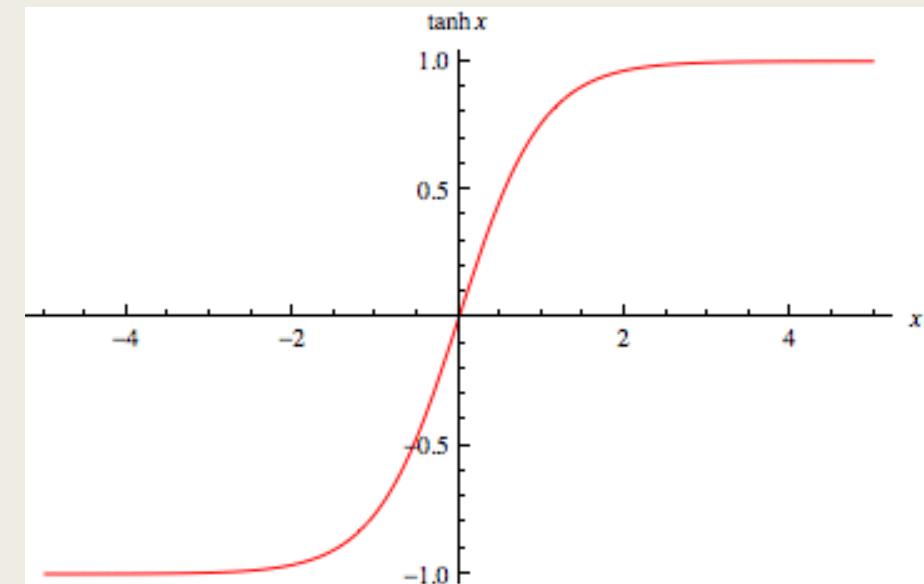
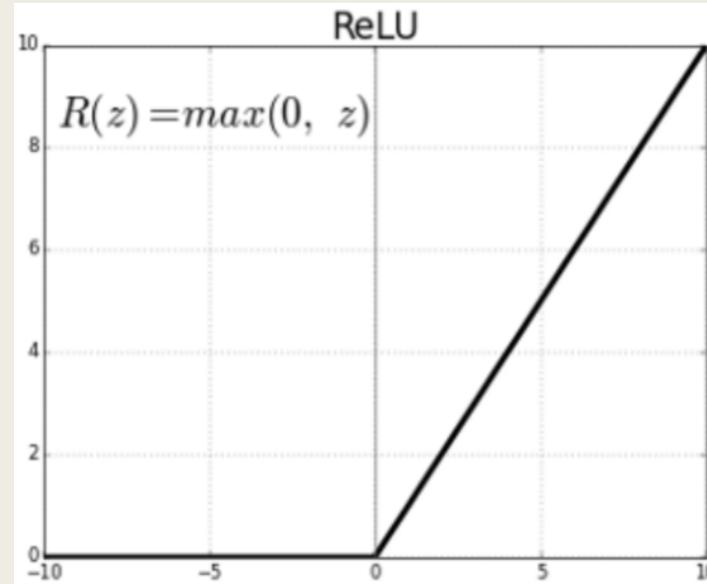
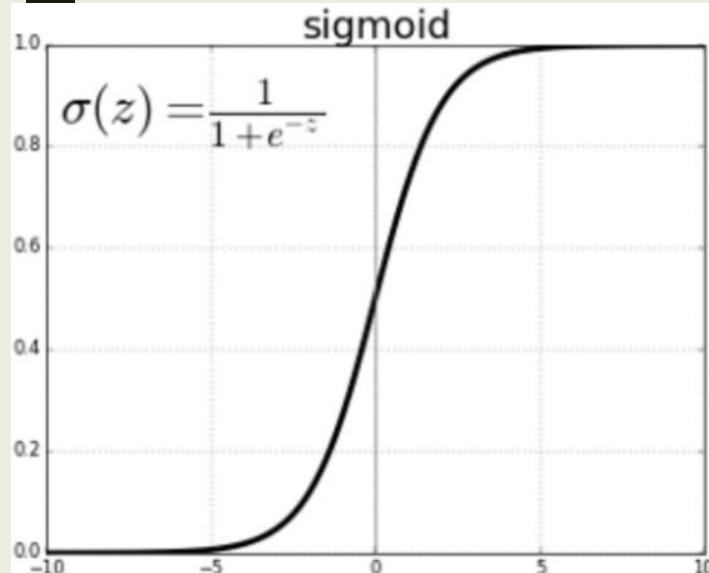
# Conceptos básicos de ML

Término	Descripción
Perceptrón	Unidad básica de una red neuronal
Entrada	Valor que usado para entrenar la red
Salida	Resultado de la red
Función de activación	Función matemática que “restringe” los resultados a un intervalo
Pesos	Valores aleatorios que multiplican a las entradas
Bias (Sesgo)	Valor aleatorio para ayudar en el aprendizaje
Época	El ciclo donde la red ha usado todas las entradas para entrenar
Set de entrenamiento	Conjunto de datos usados exclusivamente para entrenar la red
Set de validación	Conjunto de datos reservados para validar los resultados del entrenamiento
Batch	El tamaño de las muestras que tomamos para entrenar antes de actualizar los pesos
Capa	Estructura donde se agrupan los perceptrones u otras operaciones
Capa convolucional	Capa específica donde se realiza la convolución

# Conceptos básicos de ML

Concepto	Descripción
Error	Diferencia entre el resultado real y el resultado predicho
Exactitud (accuracy)	El número de “aciertos” que la red pudo clasificar de todo el conjunto de datos
Precisión	La “calidad” de la red neuronal al momento de hacer clasificaciones.
Recall (Exhaustividad)	La cantidad de aciertos que la red neuronal es capaz de identificar ¿Qué porcentaje de aciertos somos capaces de identificar?
Perdida	El valor que se trata de minimizar durante el entrenamiento
Función de pérdida	La función objetivo que tratamos de lograr, su valor de final es el valor de pérdida.
Optimizador	Método usado para disminuir el error y mejorar el desempeño de la red
Métrica	Parámetros que nos permiten evaluar el desempeño de nuestros modelos

# Funciones de activación



- Valores de cero a uno en el eje Y.
- Lenta convergencia.
- No centrada en cero.
- Buena para clasificación binaria.

- Se activa solo con valores positivos.
- No está acotada.
- Buen desempeño en CNN's.
- Puede “matar” neuronas.

- Valores entre -1 y 1 en el eje Y.
- Lenta convergencia.
- Centrada en cero.
- Buena para trabajar con multclases o RNN's.

# Metodología



# Los problemas a resolver...

- El objetivo del machine learning es resolver problemas de manera óptima y automatizada.
- Suponiendo que lográsemos abstraer todos los problemas existentes a dos grupos, tendríamos organizados dichos grupos de la forma →



# El problema de la clasificación

- Siempre se trabajan con variables discretas, estas serán nuestras salidas.
- A las salidas se les conoce como ‘etiquetas’ o ‘clases’.
- Buscamos separar linealmente las etiquetas.
- Lo que buscamos es que la red sea precisa y exacta al momento de hacer la clasificación

```
31     def __init__(self, path=None, debug=False):
32         self.file = None
33         self.fingerprints = set()
34         self.logduplicates = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, "fingerprints.txt"), "a+")
39             self.file.seek(0)
40             self.fingerprints.update(self._read_file())
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool("GENERAL.DEBUG")
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)
```

# Accuracy, precision & recall

- Estas métricas se usan para evaluar a los modelos de machine learning.
- Nos dice si la red hizo bien su trabajo o no.
- Se usan en los problemas de clasificación.
- Para calcularlos usaremos la matriz de confusión.

# La matriz de confusión

		Lo que el clasificador predice	
		negativo	positivo
Lo que tenía que decir	negativo	VERDADERO NEGATIVO	FALSO POSITIVO
	positivo	FALSO NEGATIVO	VERDADERO POSITIVO

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

# Un ejemplo

Supongamos que queremos entrenar una red neuronal para que clasifique si un email es spam o no.

Nuestras dos posibles salidas serán: spam o no spam.

Recolectamos 100 correos en total, la distribución es la siguiente:

70 correos que no eran spam se clasificaron como no spam.

10 correos se etiquetaron como spam pero no eran no spam.

15 correos que si eran spam se clasificaron como no spam.

Por último, solo 5 correos se etiquetaron como spam y eran spam.

		Predicción	
		Matriz de confusión	
		No spam (0)	Spam (1)
Realidad	No spam (0)	70	10
	Spam (1)	15	5

# Calculando las métricas

$$Precision = \frac{TP}{TP+FP} = \frac{5}{5+10} = 0.33$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{5 + 70}{5 + 70 + 10 + 15} = 0.75$$

$$Recall = \frac{TP}{TP + FN} = \frac{5}{5 + 15} = 0.25$$

## Clasificación binaria vs Clasificación multiclase

- Cuando hay sólo dos clases se conoce como un problema de clasificación binaria, es decir, solo dos posibles resultados.
- Cuando hay más de dos clases, el problema se convierte en una clasificación multiclase.
- Puede haber  $n$  cantidad de clases.
- Aunque se puede usar las métricas ‘clásicas’ es mejor trabajar con probabilidades al momento de evaluar a la red.

```
31     def __init__(self, path=None, debug=False):
32         self.file = None
33         self.fingerprints = set()
34         self.logduplicates = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, "fingerprint.log"), "a+")
39             self.file.seek(0)
40             self.fingerprints.update(self._read_file())
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool("GENERAL.DEBUG")
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)
```

## Práctica 8.1: Clasificación de flores

```
31     def __init__(self, path=None, debug=False):
32         self.file = None
33         self.fingerprints = set()
34         self.logduplicates = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, "fingerprints.txt"), "a+")
39             self.file.seek(0)
40             self.fingerprints.update(self._read_file())
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool("GENERAL.DEBUG")
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)
```

## Práctica 8.2: Clasificación de vinos

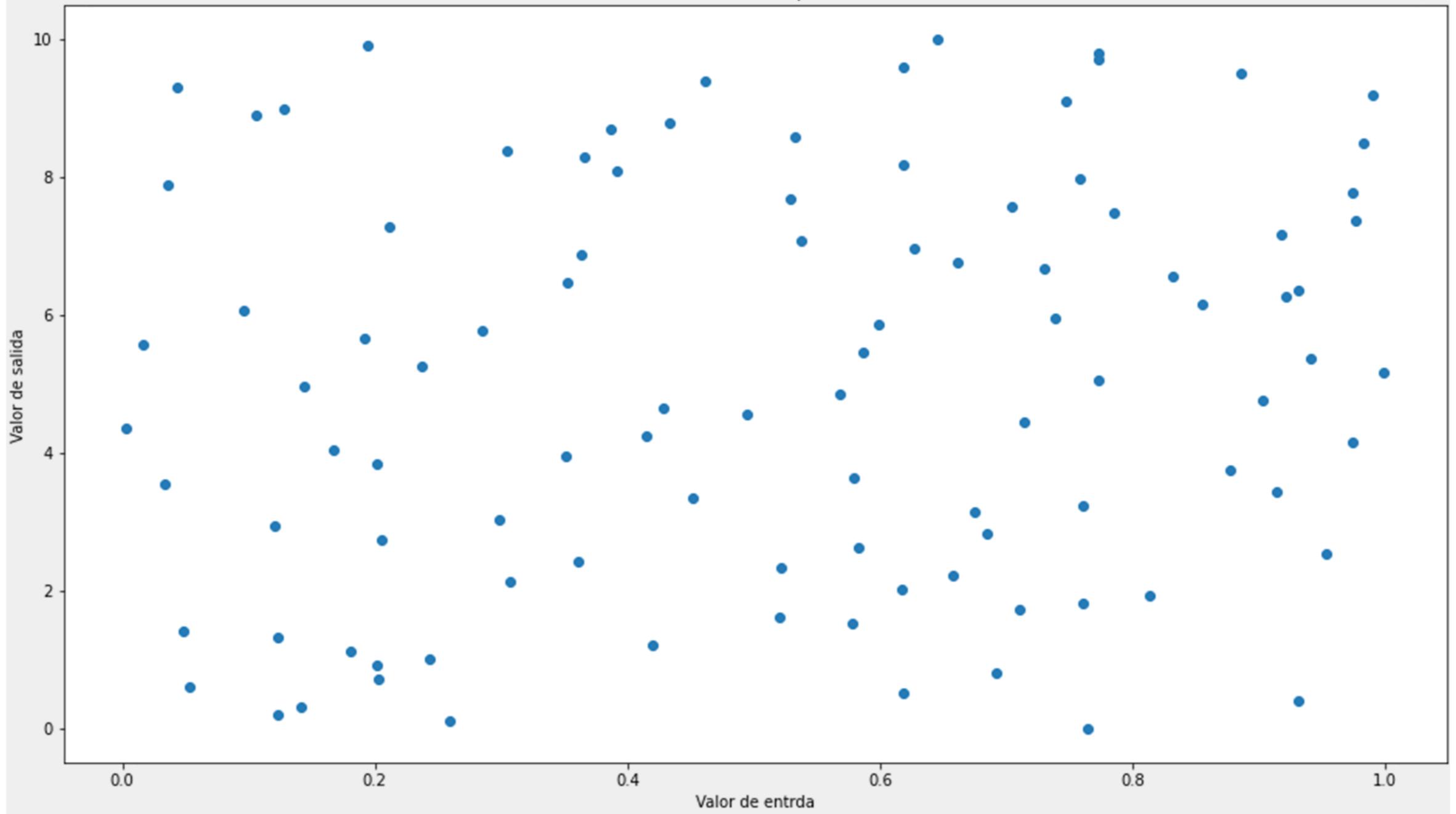
```
31     def __init__(self, path=None, debug=False):
32         self.file = None
33         self.fingerprints = set()
34         self.logduplicates = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, "fingerprint.log"), "a+")
39             self.file.seek(0)
40             self.fingerprints.update(self._read_file())
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool("GENERAL.DEBUG")
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)
```

## Práctica 8.3: Clasificación de dígitos

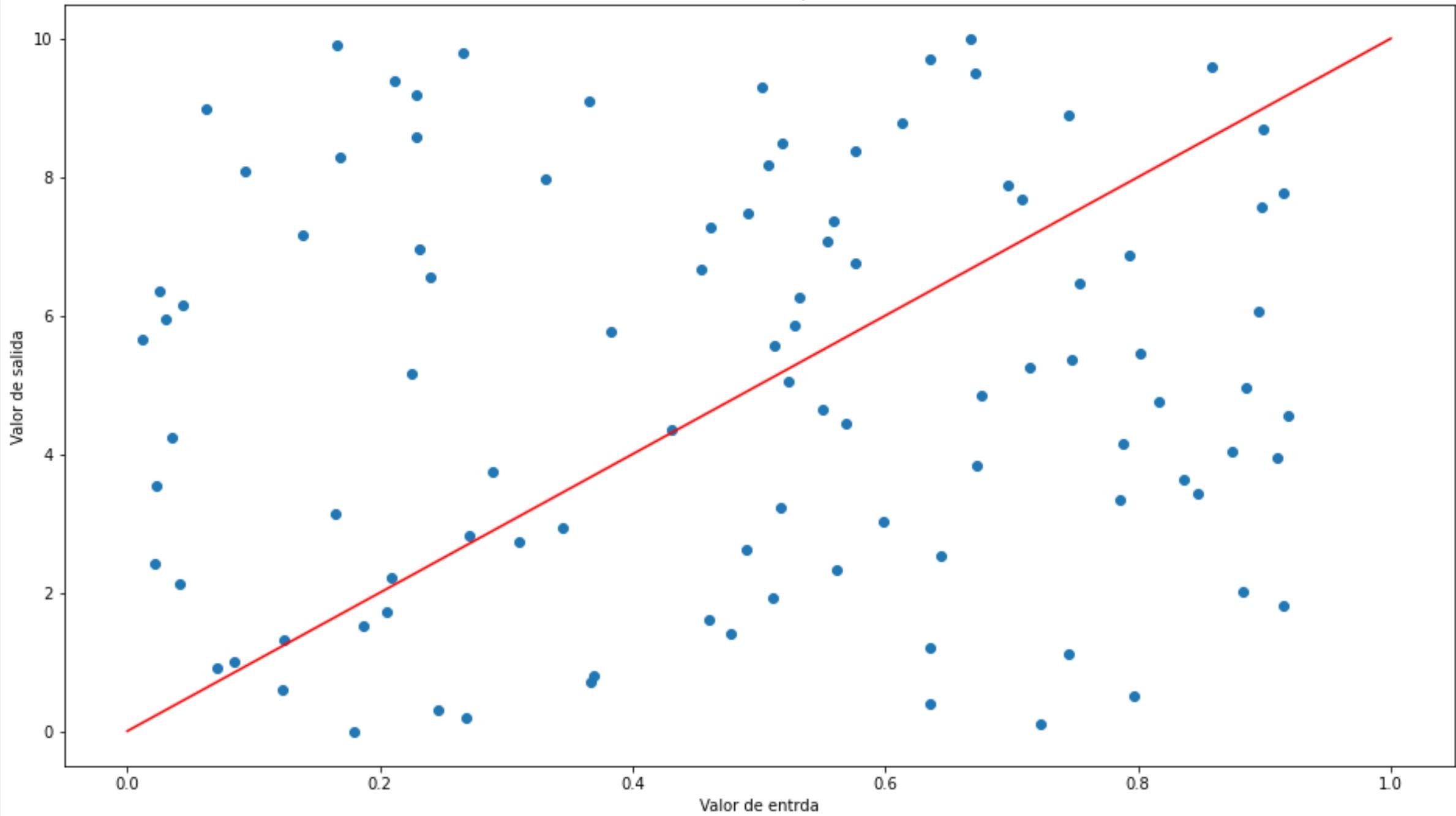
# Predicción aka Regresión

- También se le conoce como regresión.
- Usaremos variables continuas en lugar de variables discretas.
- Se pueden definir dos grupos: regresión lineal y regresión polinomial.
- Las métricas usadas son el error y el valor de pérdida.

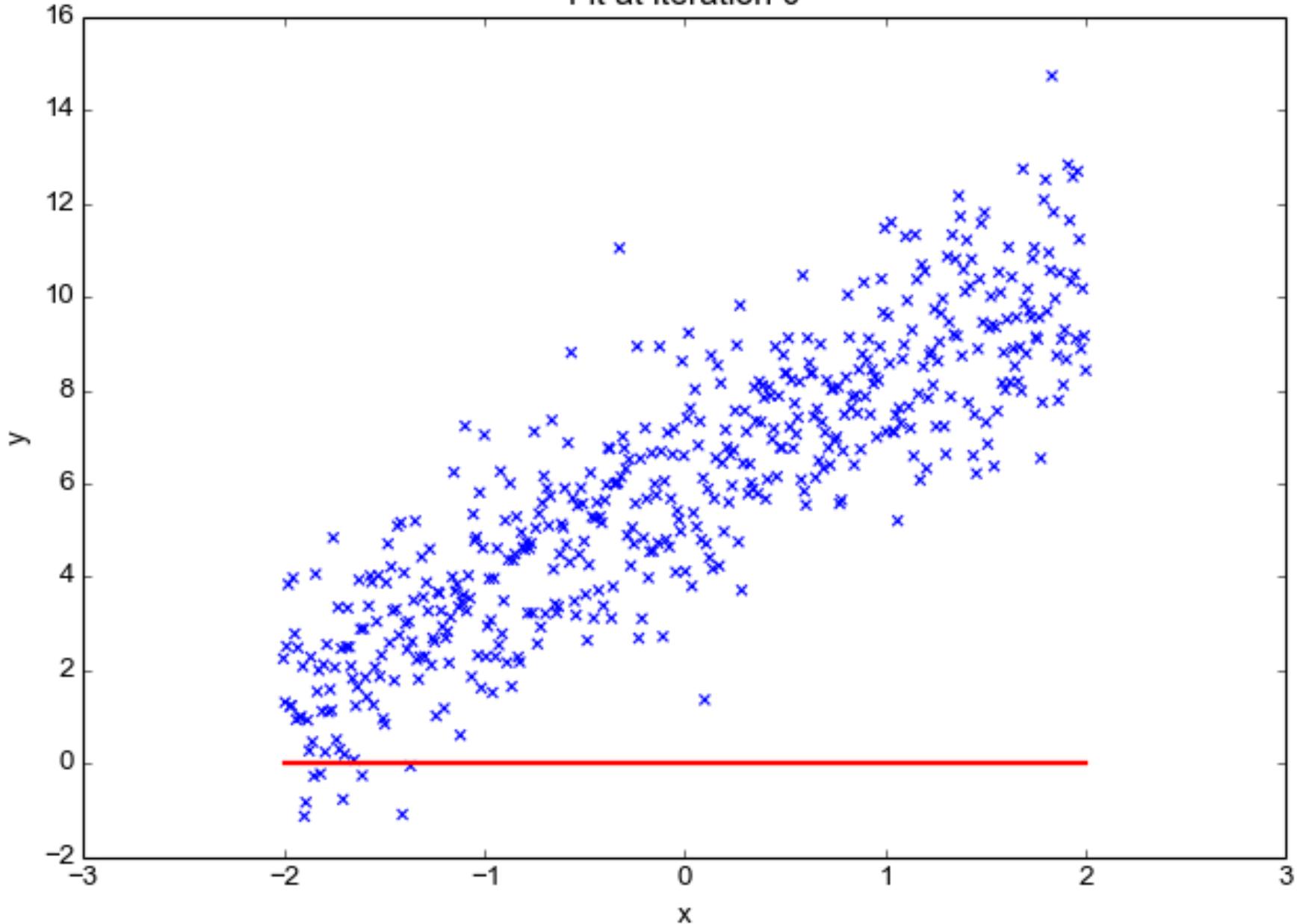
Distribución de puntos



Distribución de puntos



Fit at iteration 0



```
31     def __init__(self, path=None, debug=False):
32         self.file = None
33         self.fingerprints = set()
34         self.logduplicates = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, "fingerprint.log"), "a+")
39             self.file.seek(0)
40             self.fingerprints.update(self._read_file())
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool("GENERAL.DEBUG")
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)
```

## Práctica 8.4: Predicción de precios de casas

```
31     def __init__(self, path=None, debug=False):
32         self.file = None
33         self.fingerprints = set()
34         self.logduplicates = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, "fingerprints.txt"), "a+")
39             self.file.seek(0)
40             self.fingerprints.update(self._read_file())
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool("GENERAL.DEBUG")
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)
```

## Práctica 8.5: Predicción del consumo de gasolina

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 \quad [1.0]$$

$$\frac{d}{d\theta_0} J(\theta_0, \theta_1) = \frac{d}{d\theta_0} \left( \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 \right) \quad [1.1]$$

$$= \frac{1}{m} \sum_{i=1}^m \frac{d}{d\theta_0} (h_\theta(x^{(i)}) - y^{(i)})^2 \quad [1.2]$$

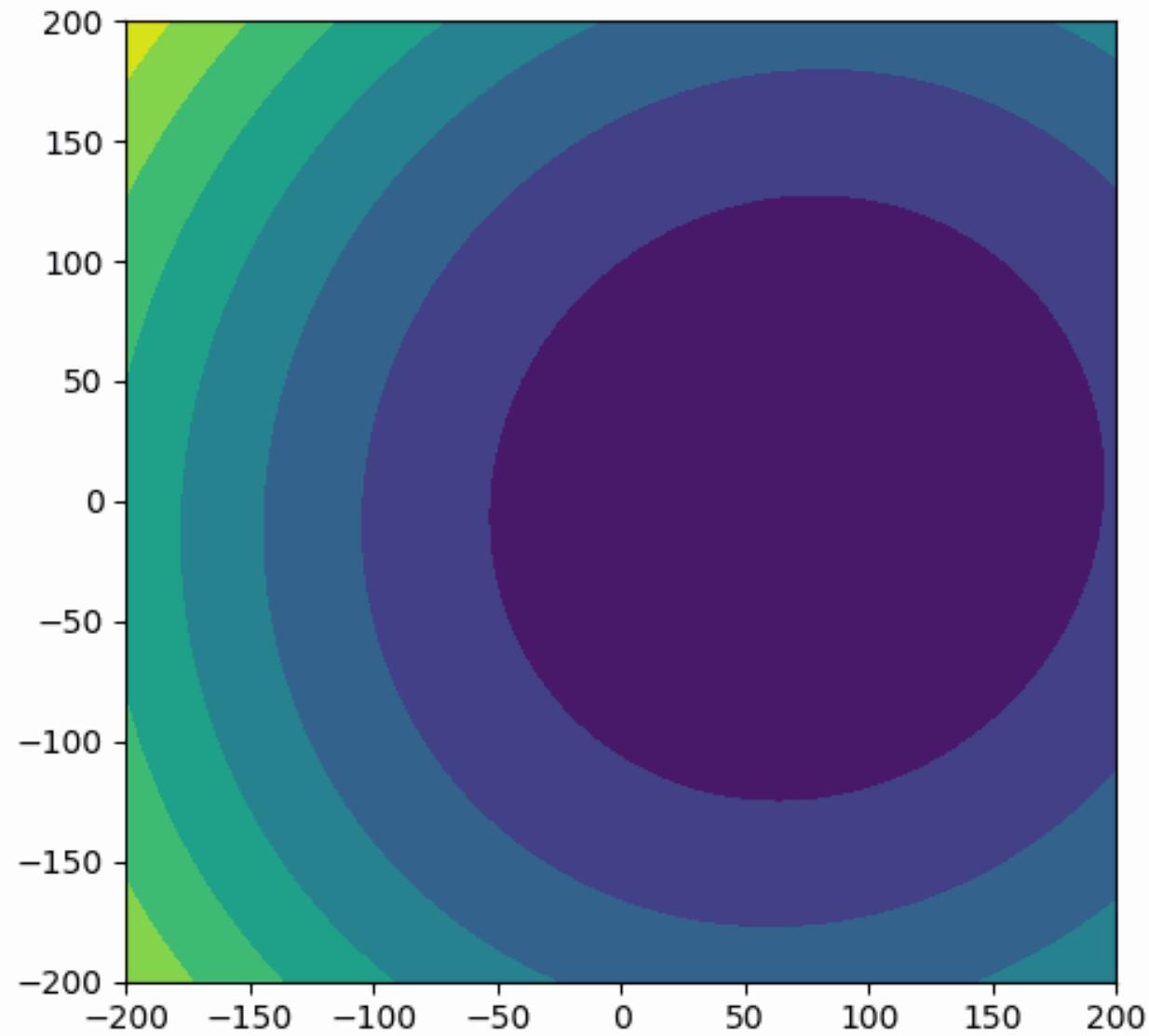
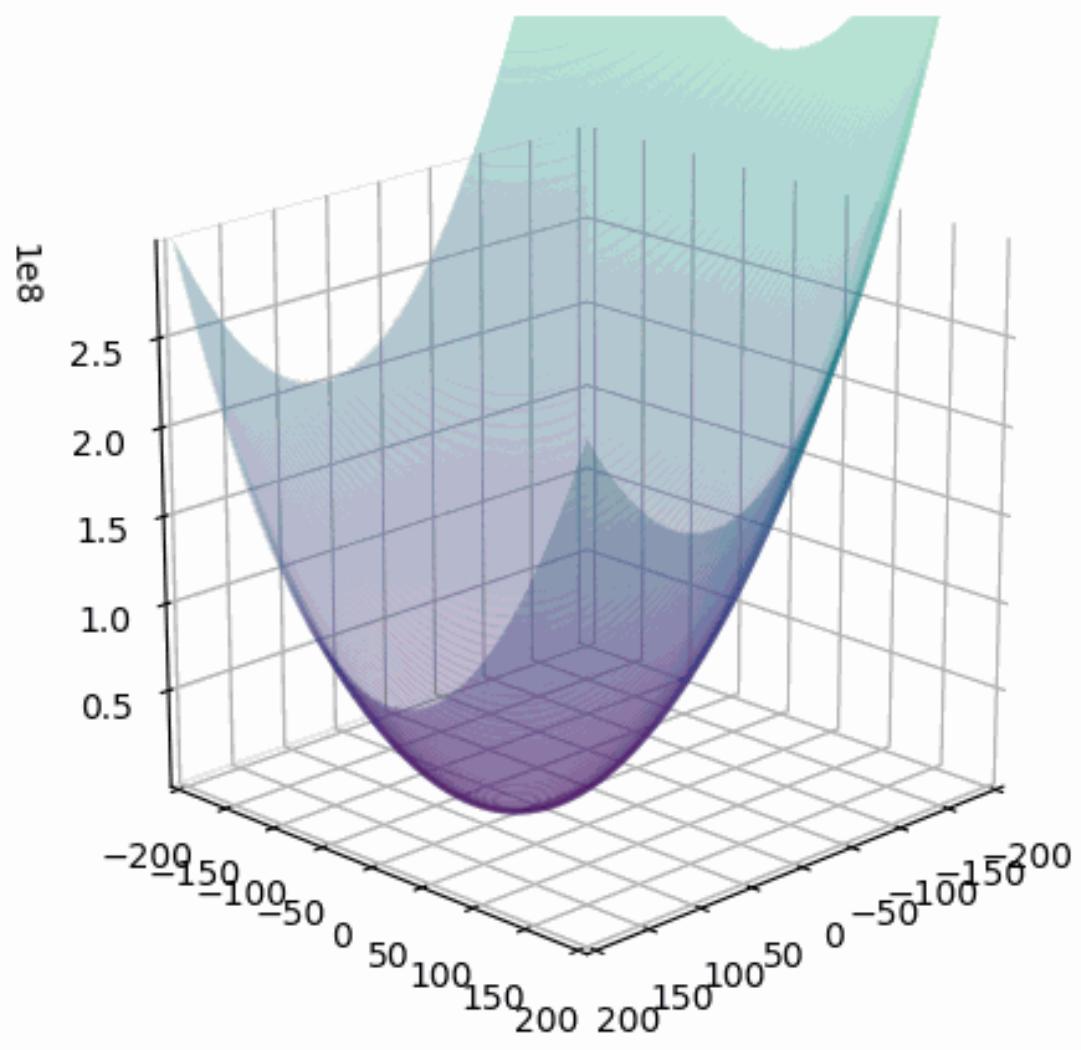
$$= \frac{1}{m} \sum_{i=1}^m 2(h_\theta(x^{(i)}) - y^{(i)}) \frac{d}{d\theta_0} (h_\theta(x^{(i)}) - y^{(i)}) \quad [1.3]$$

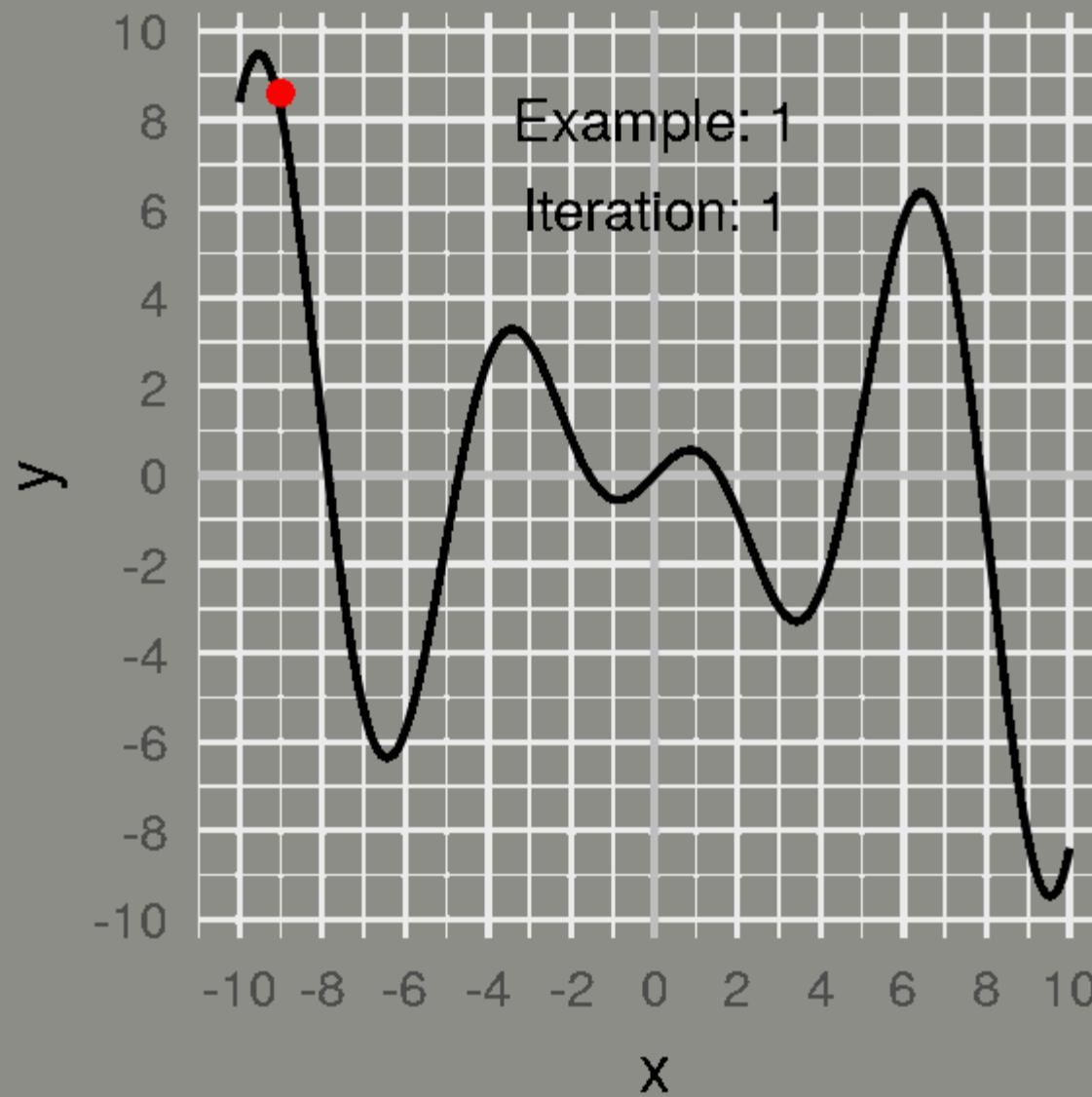
$$= \frac{2}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \quad [1.4]$$

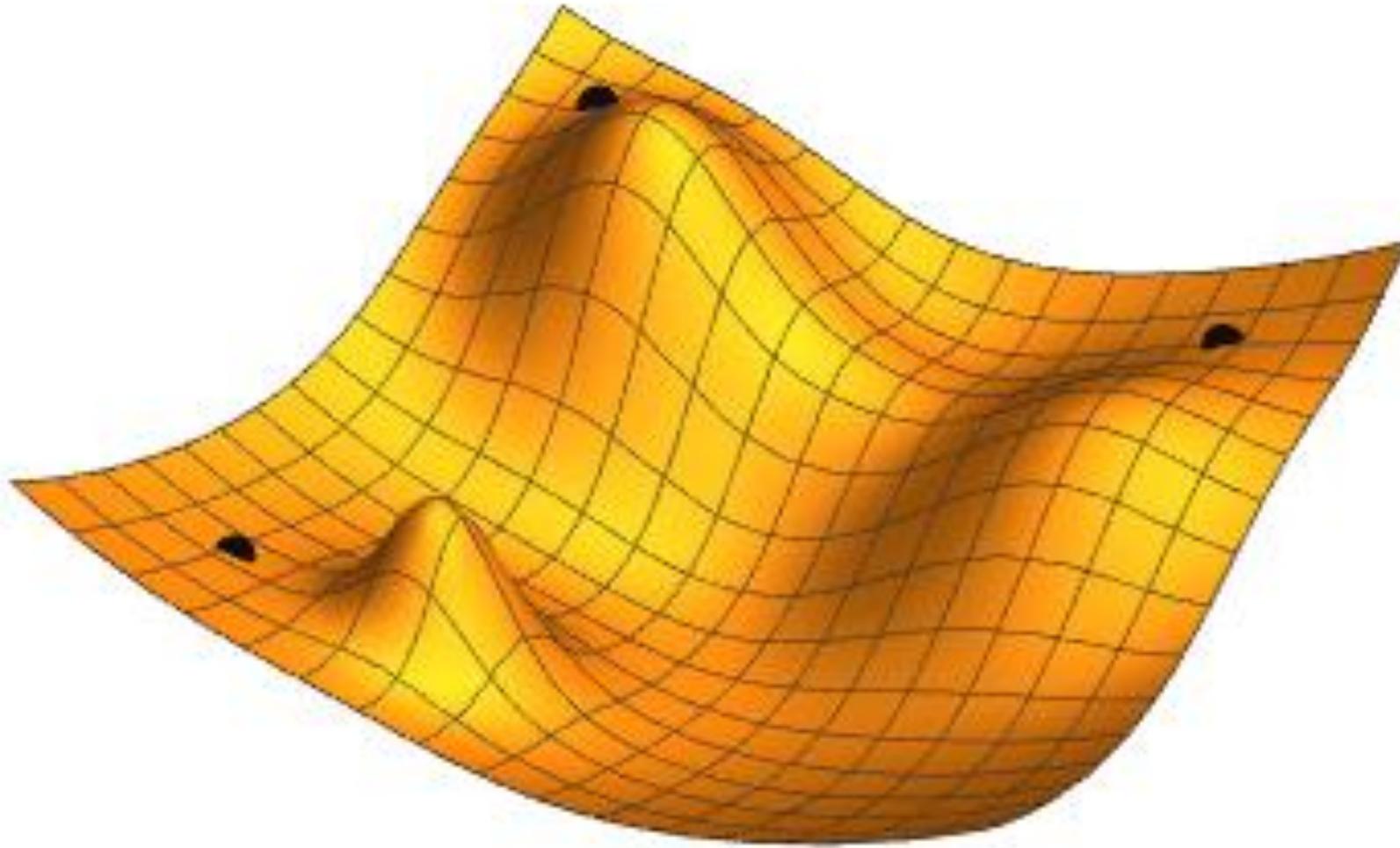
# DESCENSO POR GRADIENTE

En lugar de abrumarlos con ecuaciones y cálculos numéricos vamos a plantear un ejercicio mental.

SUPONGAMOS QUE SUBIMOS A LA CIMA  
DE UNA MONTAÑA, AL LLEGAR NOTAMOS  
QUE UN DENSA NIEBLA NOS IMPIDE VER  
EL CAMINO. ¿CÓMO PODEMOS BAJAR DE  
LA MONTAÑA?



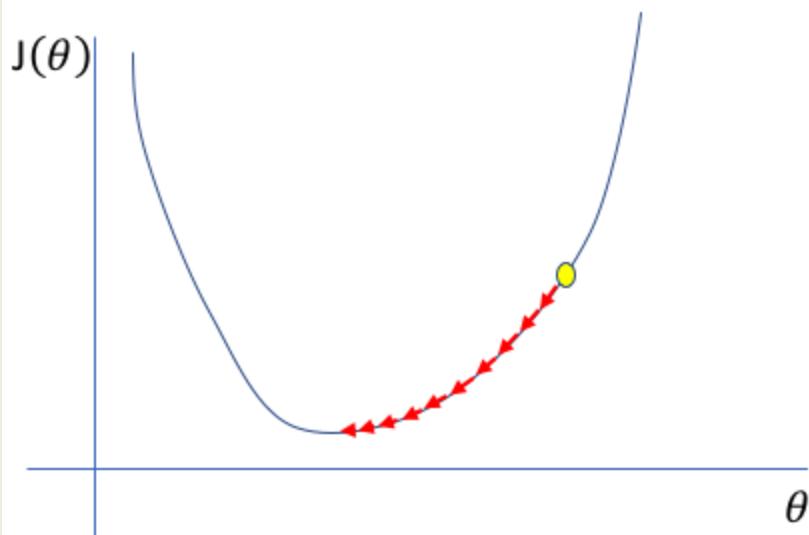




# Ahora sí ¿qué es el descenso por gradiente?

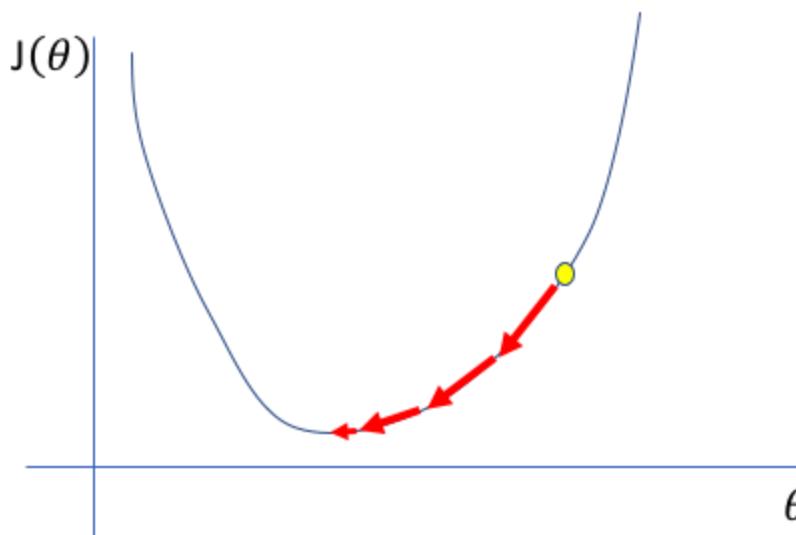
- Es una técnica de optimización que permite mejorar el aprendizaje de una red neuronal.
- El objetivo es encontrar el mínimo global de una función, es decir, la posición donde el gradiente sea el menor o igual a 0.
- Al la acción de descender hasta el mínimo se llama convergencia.
- El learning rate es el valor que determina qué tan agresivo o conservador será nuestro descenso.
- Un valor alto puede permitir una convergencia rápida pero en aras de mejorar el resultado, la posición podría empeorar.
- Una valor bajo puede ocasionar que nunca se realice la convergencia.

**Too low**



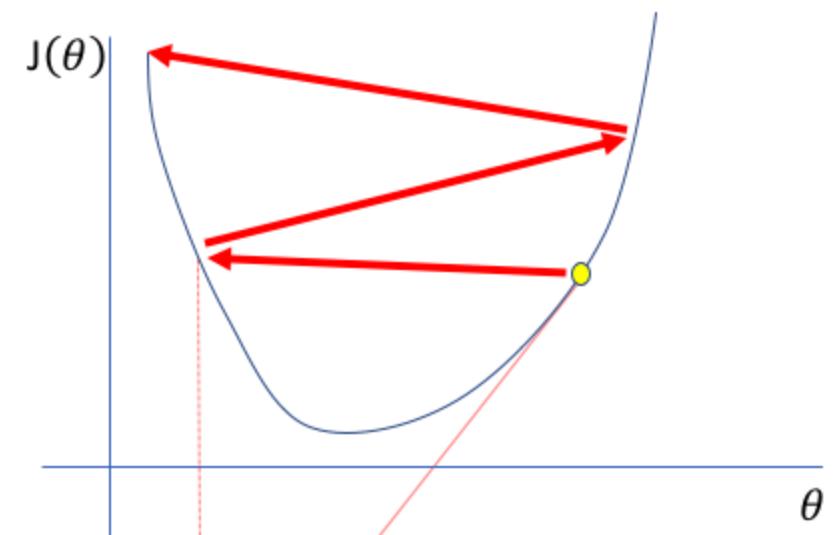
A small learning rate requires many updates before reaching the minimum point

**Just right**



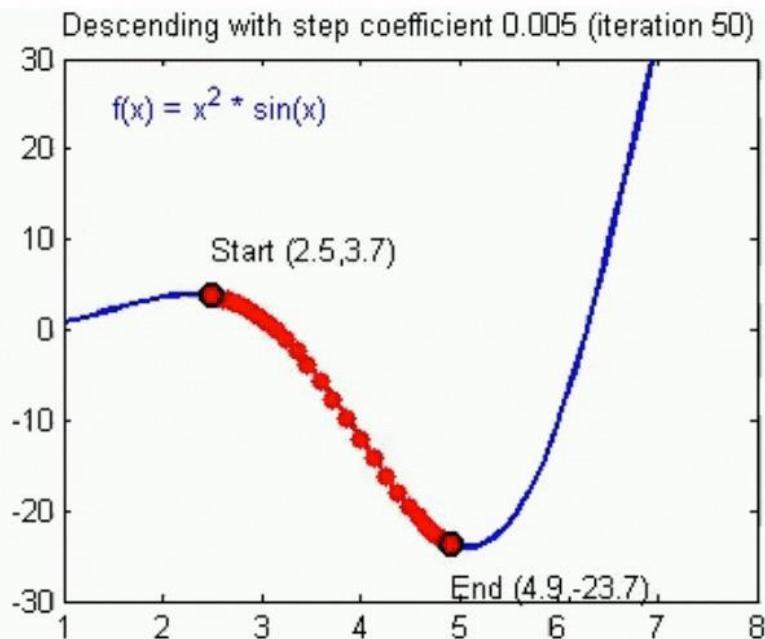
The optimal learning rate swiftly reaches the minimum point

**Too high**

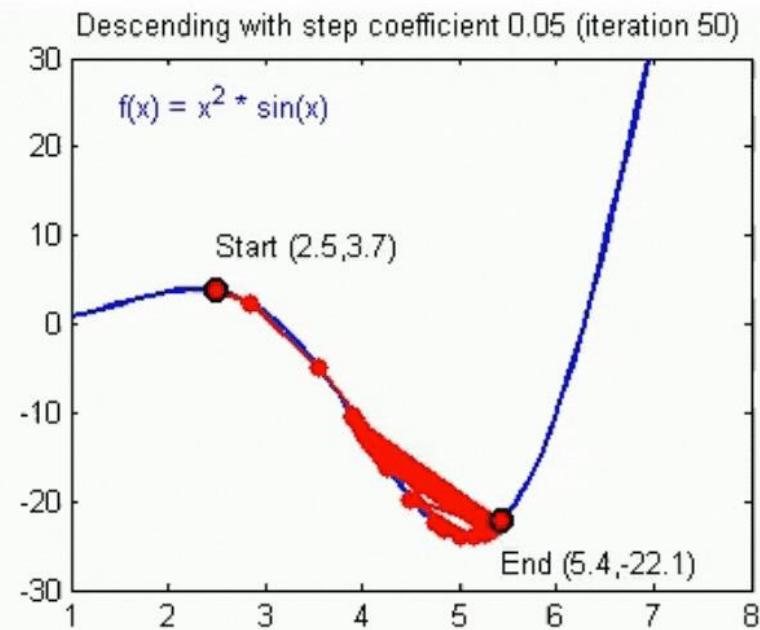


Too large of a learning rate causes drastic updates which lead to divergent behaviors

## Convergence



## Divergence



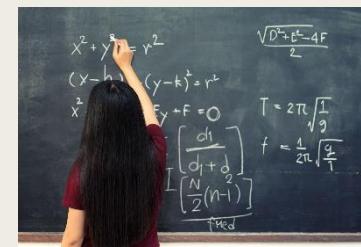
# BACKPROPAGATION



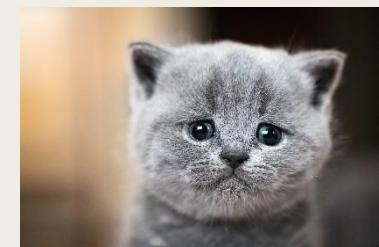
Despertar



Desayuno



Examen  
importante



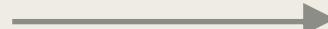
Resultado no  
esperado

*EL ALGORITMO DE BACKPROPAGATION NOS  
PERMITE CONOCER EL LUGAR EXACTO  
DONDE SE OCASIONÓ EL ERROR. DE ESTA  
FORMA SE PUEDE OPTIMIZAR DE MANERA  
RÁPIDA Y SIMPLE, SIN LA NECESIDAD DE  
USAR LA FUERZA BRUTA*

# Algunos problemas del machine learning

- Overfitting y Underfitting
- Falta de datos
- Tiempo de entrenamiento
- CPU vs TPU vs GPU

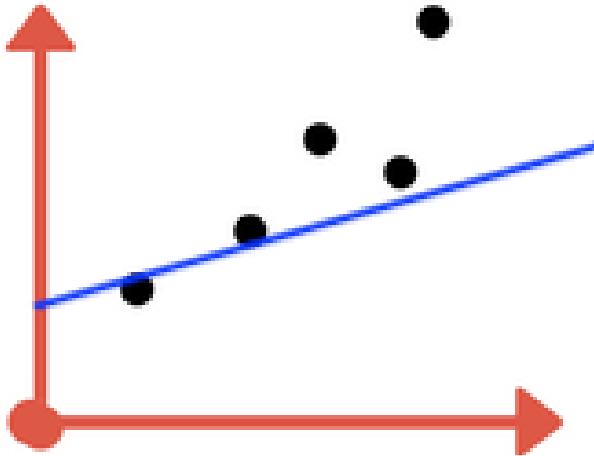
# El problema del overfitting



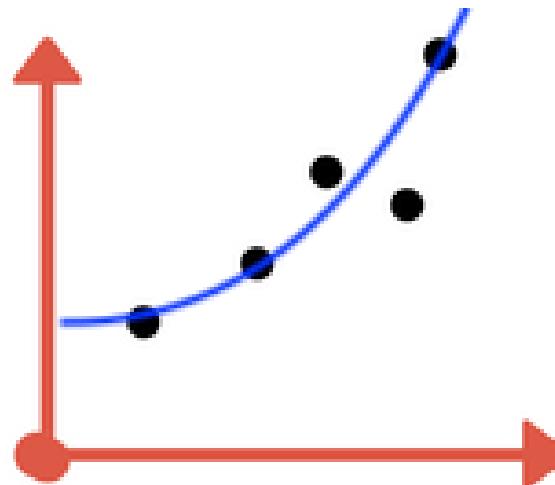
Underfitting



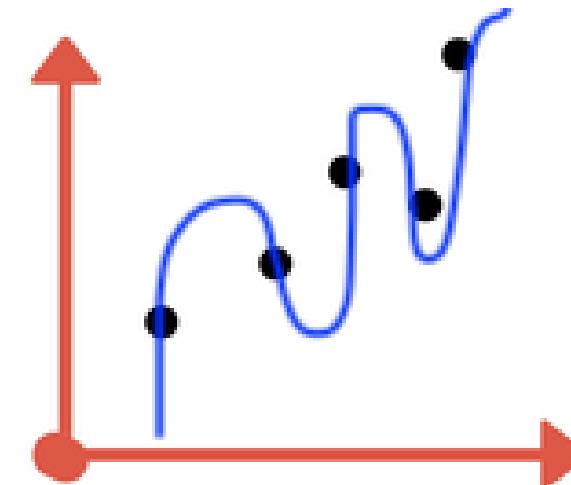
Overfitting



underfitting



correcto

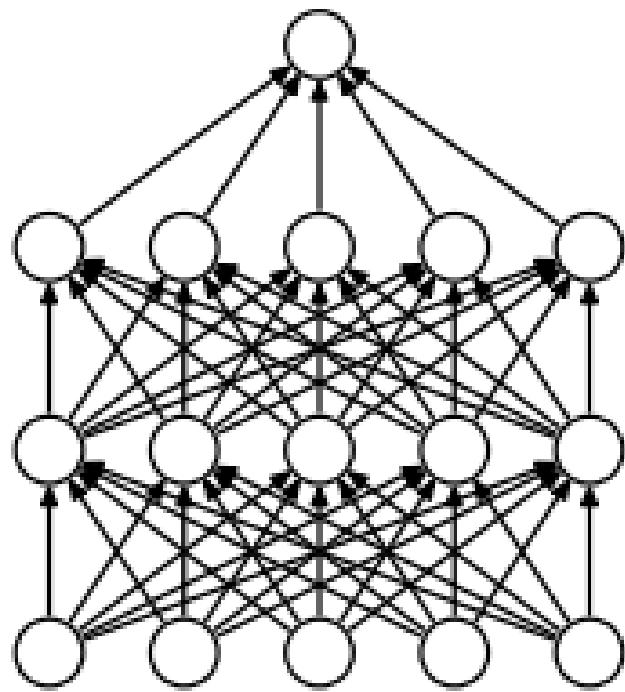


overfitting

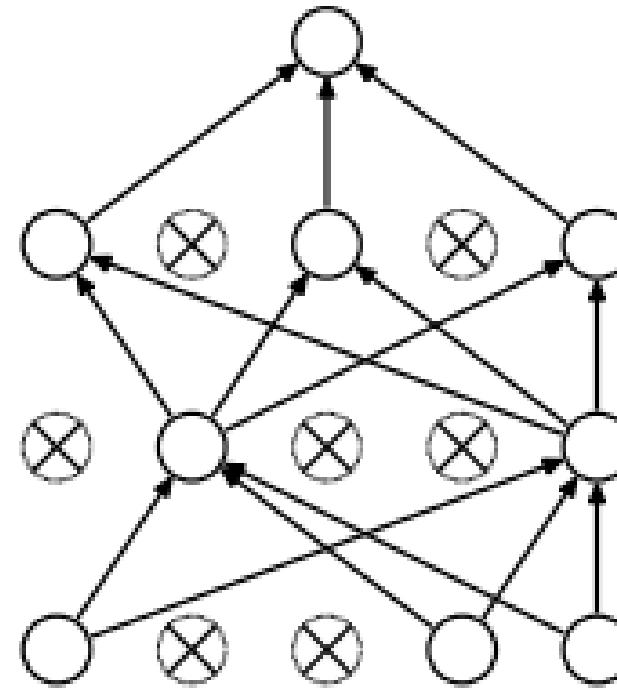
	<b>Underfitting</b>	<b>Just right</b>	<b>Overfitting</b>
<b>Symptoms</b>	<ul style="list-style-type: none"> <li>• High training error</li> <li>• Training error close to test error</li> <li>• High bias</li> </ul>	<ul style="list-style-type: none"> <li>• Training error slightly lower than test error</li> </ul>	<ul style="list-style-type: none"> <li>• Very low training error</li> <li>• Training error much lower than test error</li> <li>• High variance</li> </ul>
<b>Regression illustration</b>			
<b>Classification illustration</b>			
<b>Deep learning illustration</b>			
<b>Possible remedies</b>	<ul style="list-style-type: none"> <li>• Complexify model</li> <li>• Add more features</li> <li>• Train longer</li> </ul>		<ul style="list-style-type: none"> <li>• Perform regularization</li> <li>• Get more data</li> </ul>

# Resolviendo el *overfitting*

- Si creen necesitar más datos, consíganlos.
- Entrenar por más tiempo sin excederse.
- Cambiar la arquitectura de la red neuronal.
- Variar *learning rates* en los optimizadores.
- Probar distintas funciones de activación, funciones de pérdida.
- Aplicar técnicas de reducción de dimensiones (PCA, Boruta, entre otros).
- Usar el *dropout*.



(a) Standard Neural Net

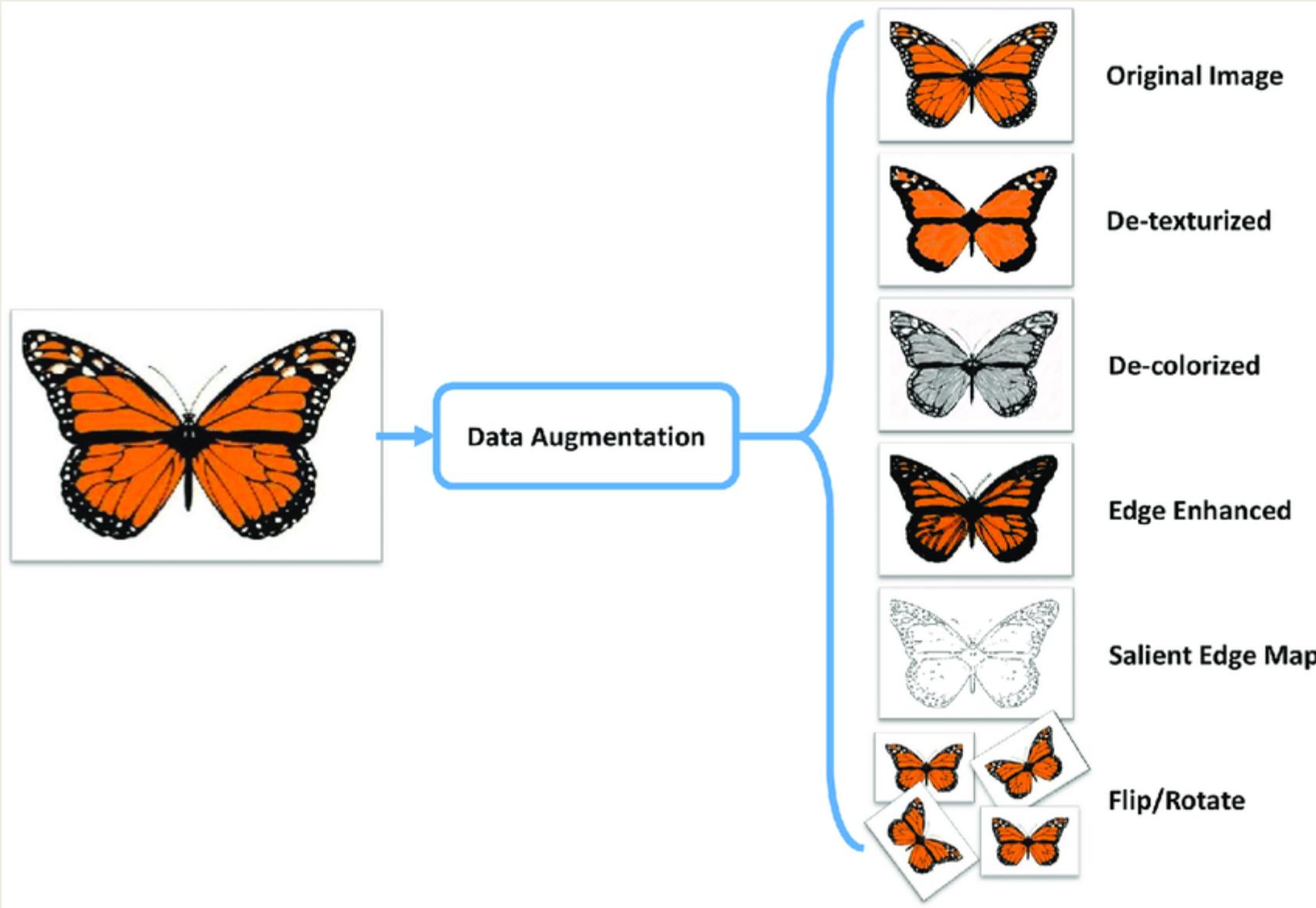


(b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left:** A standard neural net with 2 hidden layers. **Right:** An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

# Resolviendo la falta de datos

- En ocasiones podemos encontrarnos con pocos datos disponibles para entrenar.
- Estas situaciones se pueden resolver de varias formas:
  - *Consiguiendo más datos.*
  - *Generar nuevos datos de manera sintética.*
  - *Realizar aumentación de datos (data augmentation).*
- La aumentación de datos es “manipular” los datos para que parezcan distintos.
- Esta práctica sirve para enfrentar a la red a distintas situaciones que pudiera encontrarse al momento de implementarse.

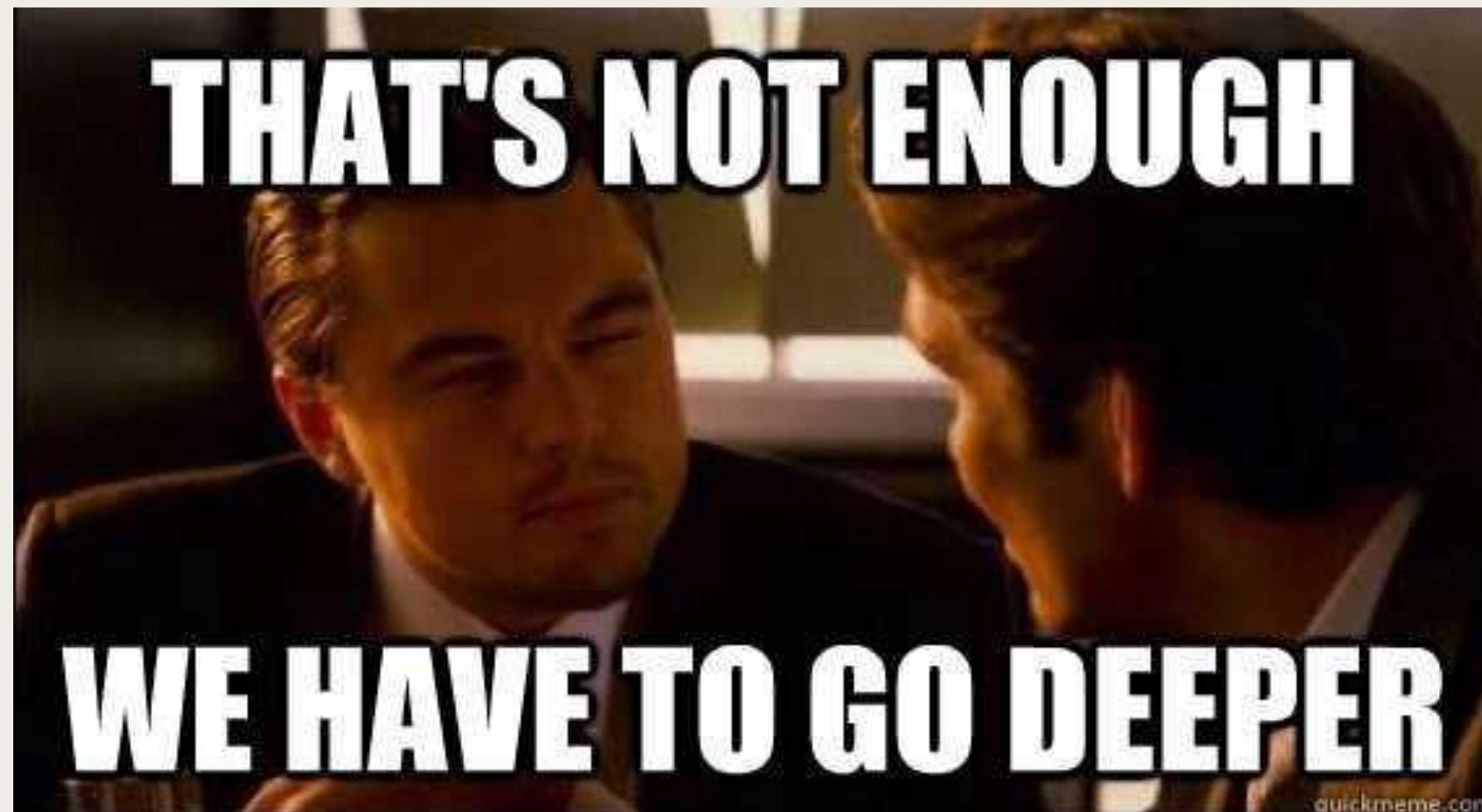


Data augmentation increases accuracy of your model – but how?, Medium, 2019.

```
31     def __init__(self, path=None, debug=False):
32         self.file = None
33         self.fingerprints = set()
34         self.logduplicates = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, "fingerprints.txt"), "a+")
39             self.file.seek(0)
40             self.fingerprints.update(self._read_file())
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool("GENERAL.DEBUG")
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)
```

## Proyecto final: Detección de fraudes bancarios

CNN's



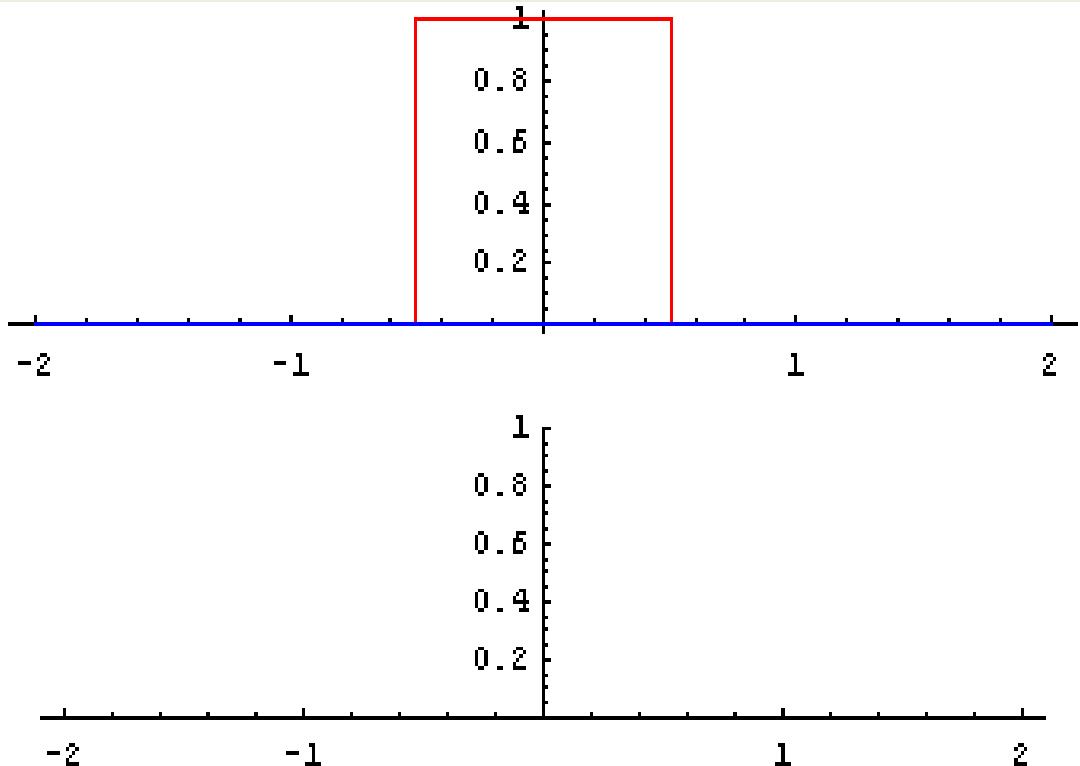
quickmeme.com

# Convolutional Neural Networks

- Una CNN es una red neuronal que tiene capas convolucionales dentro de su arquitectura.
- Permite reducir las dimensiones de los datos sin perder información que podría ser útil.
- Ideales para procesar imágenes.
- Son usadas en los autos autónomos, detección de objetos, reconocimiento de rostros, aplicaciones médicas, entre otras.

# La convolución

- Estas son las operaciones que dan vida a una CNN.
- La convolución es un operador matemático que toma dos funciones  $f(x)$ ,  $g(x)$  y las transforma en una nueva función  $(f * g)(t)$ .
- En las redes, usaremos a la imagen como nuestra función original y un *kernel* como la función de convolución, de forma que tendremos una nueva imagen con la cual trabajar.
- El *kernel* se refiere a una matriz de 3x3 generalmente, cuyos valores transforman a la imagen original.



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

Image

4		

Convolved  
Feature

# Max Pooling

- Se usa para “acumular” características importantes después de una convolución.
- Reduce las dimensiones aun más que la convolución.
- El max pooling se hace para prevenir (de cierta forma) el overfitting.
- Reduce el costo computacional al momento de entrenar.
- Al diferencia de la convolución, usamos *strides* para definir la cantidad de espacios a recorrer y un *kernel* para indicar el tamaño de la ventana de donde extraeremos los datos.

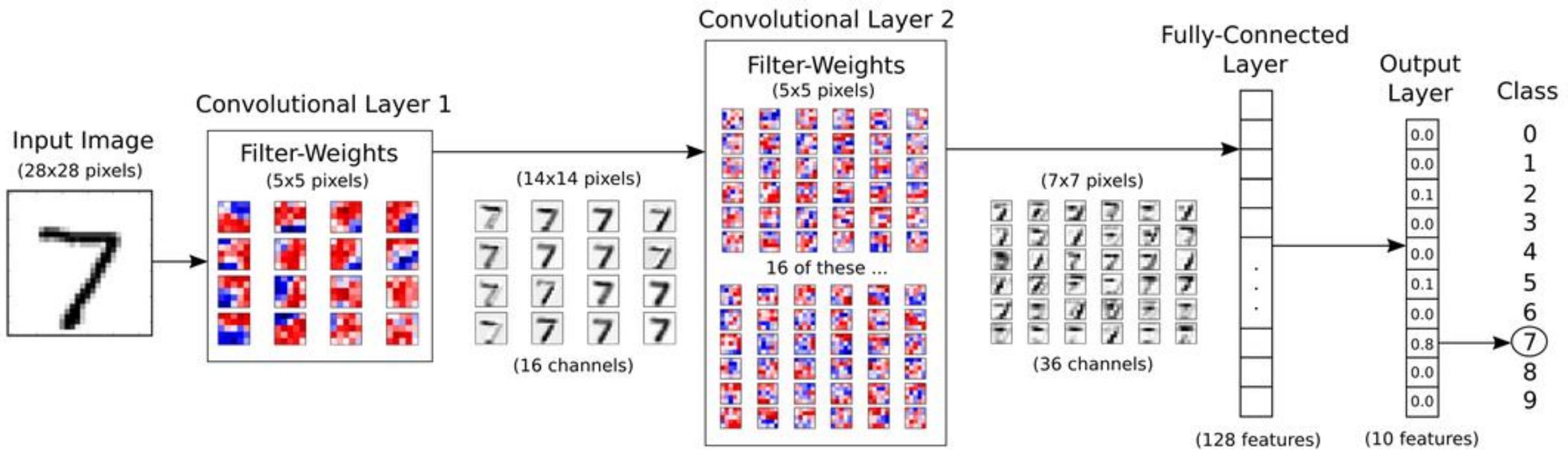
Input

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

maxpool

Output

8	6
9	9



# **USAR CAPAS CONVOLUCIONALES Y DE MAX POOLING**



```
31     def __init__(self, path=None, debug=False):
32         self.file = None
33         self.fingerprints = set()
34         self.logduplicates = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, "fingerprint.log"), "a+")
39             self.file.seek(0)
40             self.fingerprints.update(self._read_file())
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool("GENERAL.DEBUG")
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)
```

## Practica 9: Redes neuronales vs redes convolucionales.

YA PARA TERMINAR...

**THE GREATEST TEACHER,**

**FAILURE IS.**

# Bibliografía

- From zero to hero, José Portilla, Udemy, 2019.
- Learning to program: the fundamentals, University of Toronto, Coursera, 2016.
- Python Tutorial, w3schools.com
- Python, python.org