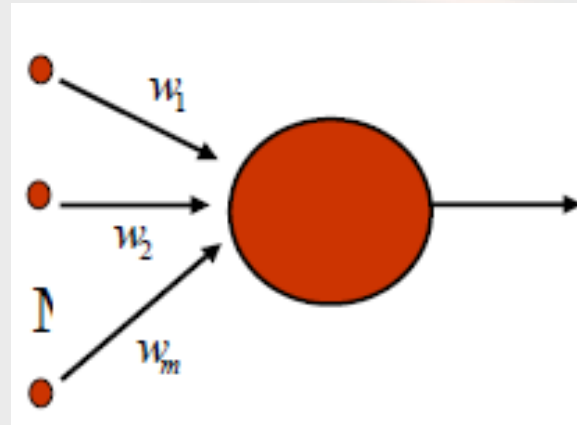
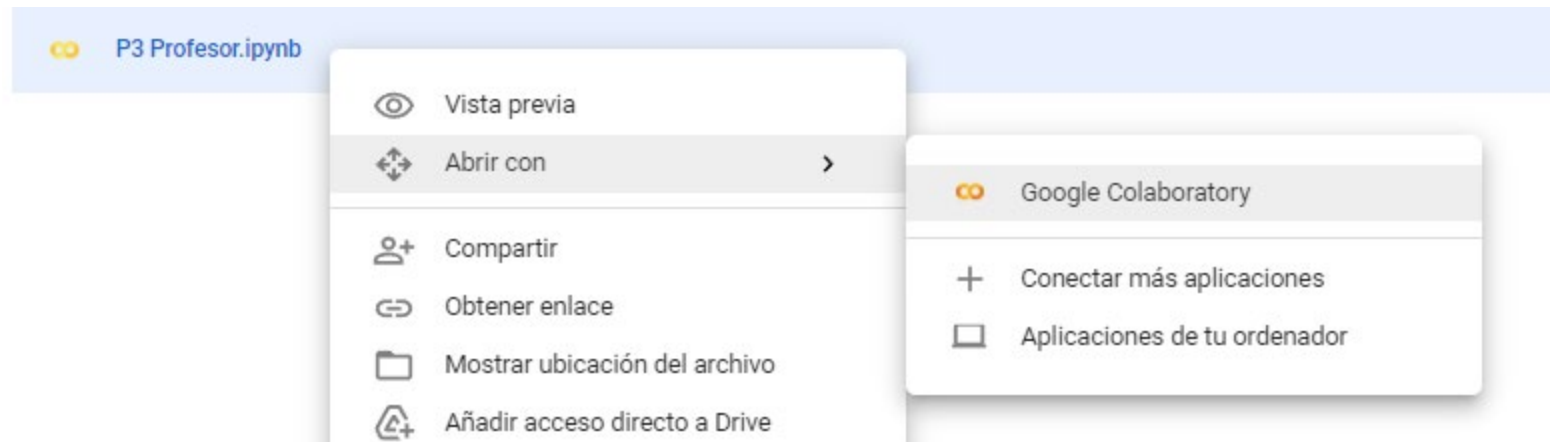


REDES NEURONALES



Grado en Ingeniería de Tecnologías y Servicios de
Telecomunicación



P3 Profesor.ipynb

- Vista previa
- Abrir con >
- Compartir
- Obtener enlace
- Mostrar ubicación del archivo
- Añadir acceso directo a Drive

- Google Colaboratory
- Conectar más aplicaciones
- Aplicaciones de tu ordenador



Google Workspace Marketplace



Buscar en apps



Colaboratory

Desinstalar

This allows Google Colaboratory to open and create files in Google Drive. It is automatically installed on first use; uninstalling this will not prevent access to Colaboratory.

Autor: [Colaboratory team](#)

Ficha actualizada: 17 de febrero de 2022

Compatible con:



★★★★★ 3.477 ↓ 10.000.000+





<https://www.kaggle.com/learn>

Explore Courses



Python

Learn the most important language for data science.



Intro to Machine Learning

Learn the core ideas in machine learning, and build your first models.



Pandas

Solve short hands-on challenges to perfect your data manipulation skills.



Intermediate Machine Learning

Handle missing values, non-numeric values, data leakage, and more.



Data Visualization

Make great data visualizations. A great way to see the power of coding!



Feature Engineering

Better features make better models. Discover how to get the most out of your data.

Lessons

Tutorial Exercise

1

Hello, Python

A quick introduction to Python syntax, variable assignment, and numbers



2

Functions and Getting Help

Calling functions and defining our own, and using Python's builtin documentation



3

Booleans and Conditionals

Using booleans for branching logic



4

Lists

Lists and the things you can do with them. Includes indexing, slicing and mutating



5

Loops and List Comprehensions

For and while loops, and a much-loved Python feature: list comprehensions



6

Strings and Dictionaries

Working with strings and dictionaries, two fundamental Python data types



7

Working with External Libraries

Imports, operator overloading, and survival tips for venturing into the world of external libraries



Operaciones básicas

Operator	Description	Example	Result
+	addition	2 + 3	5
-	subtraction	8 - 6	2
-	negative number	-4	-4
*	multiplication	5 * 2	10
/	division	6 / 3	2
**	raises a number to a power	10**2	100
_	returns last saved value	_ + 7	107

Trig function	Name	Description	Example	Result
math.pi	pi	mathematical constant π	math.pi	3.14
math.sin()	sine	sine of an angle in radians	math.sin(4)	9.025
math.cos()	cosine	cosine of an angle in radians	cos(3.1)	400
math.tan()	tangent	tangent of an angle in radians	tan(100)	2.0
math.asin()	arc sine	inverse sine, ouput in radians	math.sin(4)	9.025
math.acos()	arc cosine	inverse cosine, ouput in radians	log(3.1)	400
math.atan()	arc tangent	inverse tangent, ouput in radians	atan(100)	2.0
math.radians()	radians conversion	degrees to radians	math.radians(90)	1.57
math.degrees()	degree conversion	radians to degrees	math.degrees(2)	114.59

Math function	Name	Description	Example	Result
math.e	Euler's number	mathematical constant e	math.e	2.718
math.exp()	exponent	e raised to a power	math.exp(2.2)	9.025
math.log()	natural logarithm	log base e	math.log(3.1)	400
math.log10()	base 10 logarithm	log base 10	math.log10(100)	2.0
math.pow()	power	raises a number to a power	math.pow(2,3)	8.0
math.sqrt()	square root	square root of a number	math.sqrt(16)	4.0

Statistics function	Name	Description	Example	Result
mean()	mean	mean or average	mean([1,4,5,5])	3.75
median()	median	middle value	median([1,4,5,5])	4.5
mode()	mode	most often	mode([1,4,5,5])	5
stdev()	standard deviation	spread of data	stdev([1,4,5,5])	1.892
variance()	variance	variance of data	variance([1,4,5,5])	3.583

NumPy



Trabajar con Arrays



Más rápido que las
listas de Python



Links:

- [NumPy for Matlab users](#)
- [Numpy Cheat Sheet](#)

```
import numpy as np
```



Amplio uso en
Data Science

Links:

- [10 min intro](#)
- [Cheat Sheet](#)

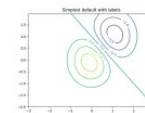
```
import pandas as pd
```

Visualización y gráficos

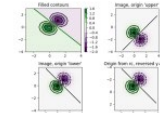
Links:

- Cheatsheets
- <https://matplotlib.org/cheatsheets/>

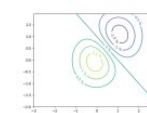
```
import matplotlib.pyplot as plt
```



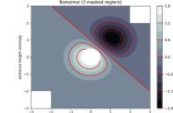
Contour Demo



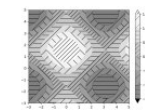
Contour Image



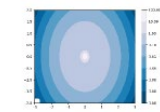
Contour Label Demo



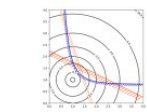
Contourf Demo



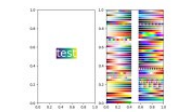
Contourf Hatching



Contourf and log color scale



Contouring the solution space of optimizations



BboxImage Demo

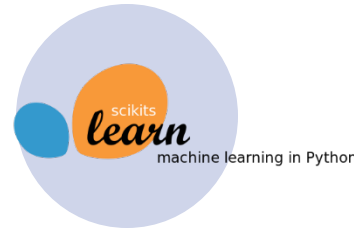
Redes Neuronales: Objetivos



APRENDER A
IMPLEMENTAR
UN
PERCEPTRÓN EN
PYTHON.



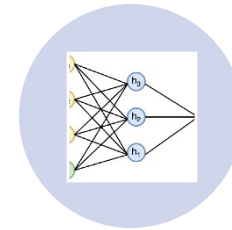
APRENDER A
IMPLEMENTAR
UN SISTEMA DE
REDES
NEURONALES
MULTICAPA EN
PYTHON.



APRENDER A
UTILIZAR LA
LIBRERÍA DE
PYTHON SCIKIT
LEARN.



CONSOLIDAR EL
APRENDIZAJE DE
LA PRÁCTICA
ANTERIOR PARA
VISUALIZAR
GRÁFICAS CON
MATPLOTLIB



ENTRENAR,
VALIDAR Y
COMPROBAR EL
CORRECTO
FUNCIONAMIENT
O DE UNA RED
NEURONAL.

- **¿CUÁNDO TENGO QUE ENTREGAR EL RESULTADO DE LA PRÁCTICA?** Para el desarrollo de la presente práctica el grupo deberá programar diversas funciones en Python que deberán ser subidas al poliformat de la asignatura antes de las **14h los días 25/26 de abril** (Importante: No se admitirán ningún archivo subido después de dicha hora). Cada alumno debe subir su propio código, aunque se haya hecho en parejas. Los nombres de los archivos a subir se indican durante el desarrollo de la práctica.

En esta práctica cada alumno deberá entregar un único archivo que contenga las siguientes funciones principales correspondiente a cada uno de los ejercicios de la práctica:

main_P4_SCBIO_DNI.ipynb

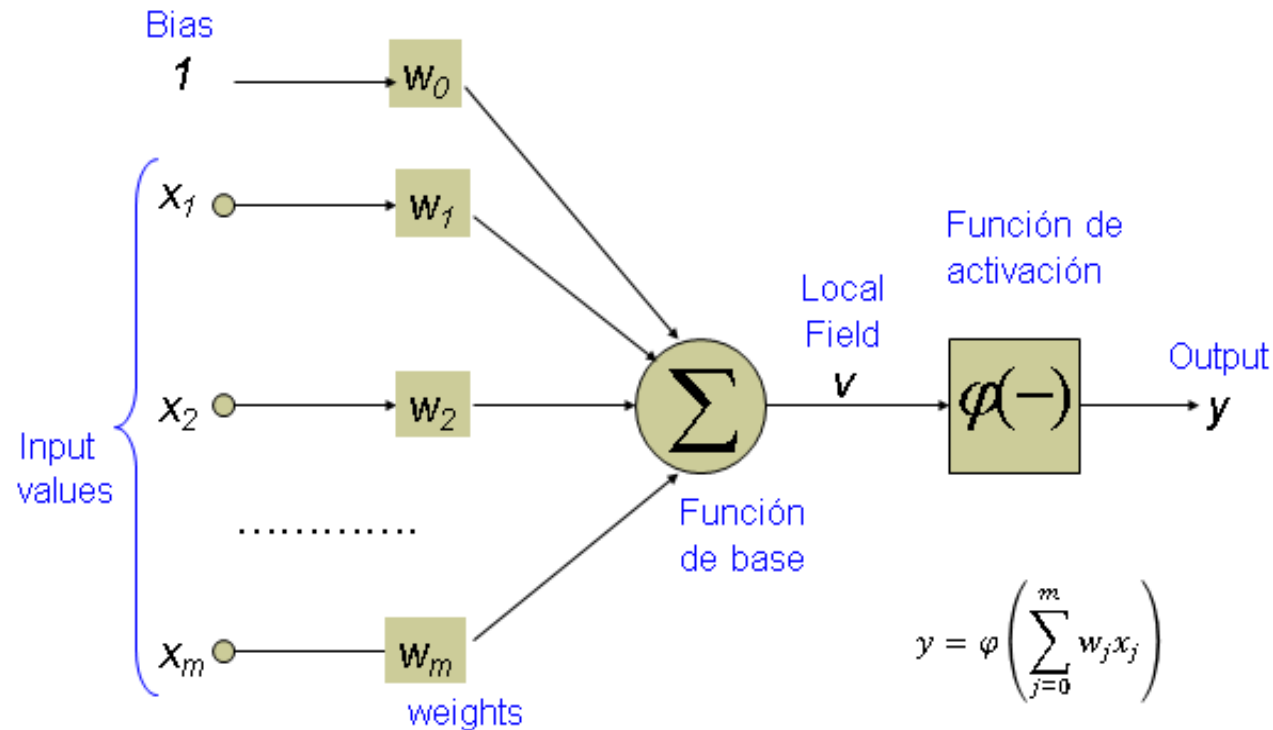
Subfunciones

1. myperceptron AND DNI
2. myperceptron XOR DNI
3. mynn XOR DNI
4. sklearn NN XOR()

La calificación se realizará ponderando la calidad del código y las respuestas en la entrevista. El apartado 1 tendrá una puntuación máxima de 3 puntos. El apartado 2 tendrá una puntuación de 1 punto y el apartado 3 tendrá una puntuación de 3 puntos. El apartado 4 valdrá 2 puntos. La originalidad en la solución, así como aquellos materiales extras como la estética sumarán hasta 1 punto extra.

1. Diseño de perceptrón simple

a. EVALUACIÓN CON LA FUNCIÓN AND



IN1	IN2	OUT
0	0	0
0	1	0
1	0	0
1	1	1

b. EVALUACIÓN CON LA FUNCIÓN XOR

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

Funciones a implementar

Función que inicialice el perceptrón con valores aleatorios para los pesos de las entradas y de las polarizaciones para un número dado de entradas:

initialize_perceptron(n_inputs)



Función que entrene el perceptrón partiendo de datos de entrenamiento. Utiliza una función de activación sigmoideal y backpropagation para el entrenamiento

train_perceptron(myperceptron,LR,input,output)



Función que evalúe el funcionamiento de la red para un conjunto de datos de validación:

useperceptron(myperceptron,input)



Función principal que incluirá las funciones previas y se encargará de llamar a dichas instrucciones para crear el perceptrón, entrenarlo y evaluar su funcionamiento. Esta función debe incluir un entorno gráfico.

myperceptron_AND_DNI

Función principal que incluirá las funciones previas y se encargará de llamar a dichas instrucciones para crear el perceptrón, entrenarlo y evaluar su funcionamiento. Esta función debe incluir un entorno gráfico.

myperceptron_AND_DNI

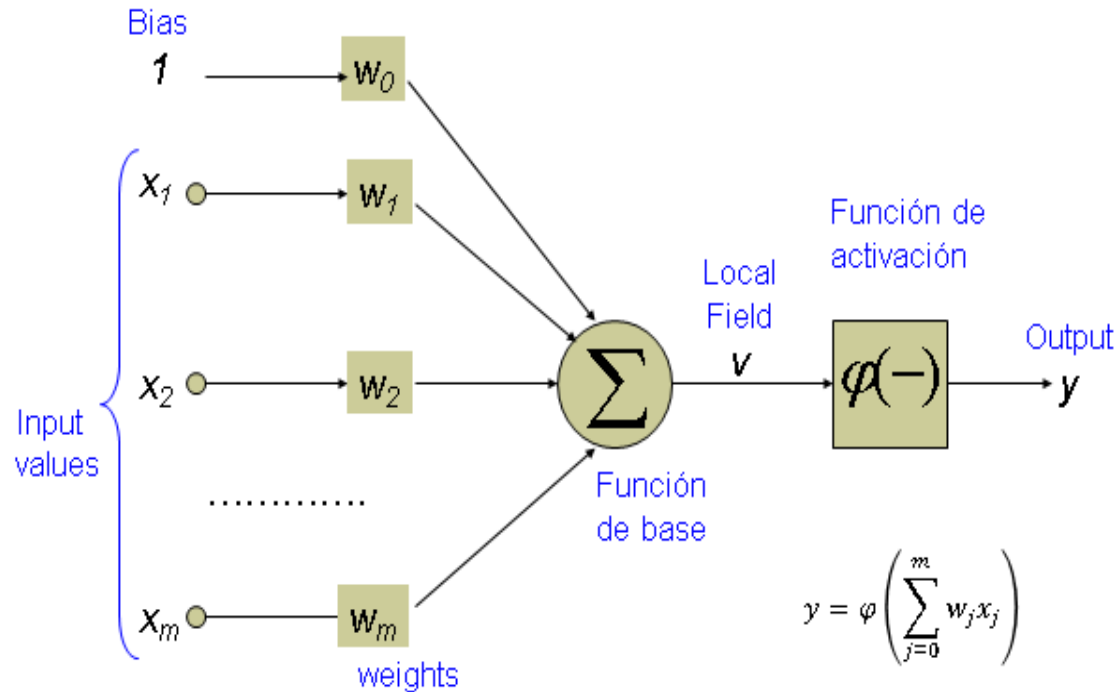
Función principal que realiza las funciones de

- ✓ 1) Creación de las variables para el banco de entrenamiento y banco de validación
- ✓ 2) Creación de la red neuronal (perceptrón)
- ✓ 3) Entrenamiento de la red neuronal con el set de valores del banco de entrenamiento
- ✓ 4) Validación de la red con el banco de validación.
- ✓ 5) Cálculo y representación del error cometido.

HAREIS DOS EJEMPLOS TRATANDO DE EMULAR UNA AND Y UNA XOR

REDES NEURONALES

PERCEPTRON - RED NEURONAL UNICELULAR



$$y = \varphi \left(\sum_{j=0}^m w_j x_j \right)$$

$$b = x_o \cdot w_o = 1 \cdot w_o$$

$$y = \varphi \left(\sum_{j=1}^m w_j x_j + b \right) = \varphi (X^T W + b)$$

$$y = \varphi \left(\sum_{j=0}^m w_j x_j \right)$$

Siendo:

w el peso,

j el número de peso de la neurona

η la constante de aprendizaje (e.g. 0.7)

δ el error cometido

$$\delta = d_k - Y_k$$

d_k = salida correcta

Y_k = salida calculada por la neurona


x_j la entrada **j**

BACKPROPAGATION

$$w_j(t + 1) = w_j(t) + \eta \cdot \delta(t) \cdot x_j(t)$$

- Programar un perceptrón desde 0

```
1 def initialize_perceptron(n_inputs):  
2     # Esta función crea e inicializa la estructura myperceptron.weights  
3     # donde se guardan los pesos del perceptron. Los valores de los pesos  
4     # iniciales deben ser números aleatorios entre -1 y 1.  
5     # n_inputs: numero de entradas al perceptron  
6     # OUTPUTS  
7     # bias: bias del perceptron (e.g. 1)  
8     # weights: pesos del perceptron  
9  
10    return out
```

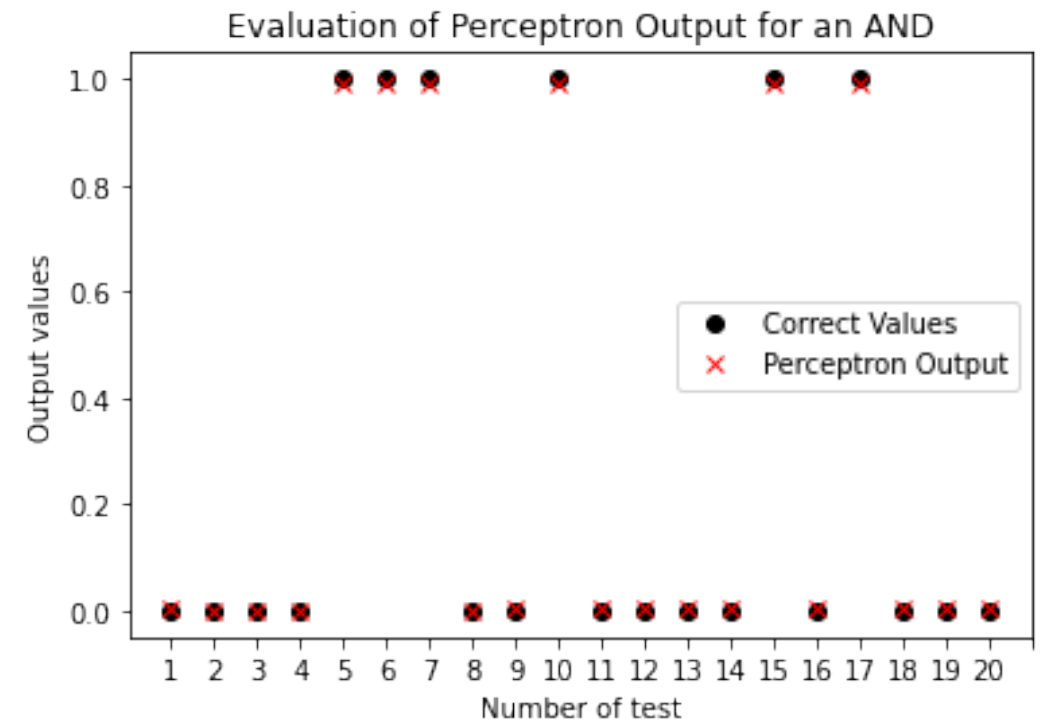
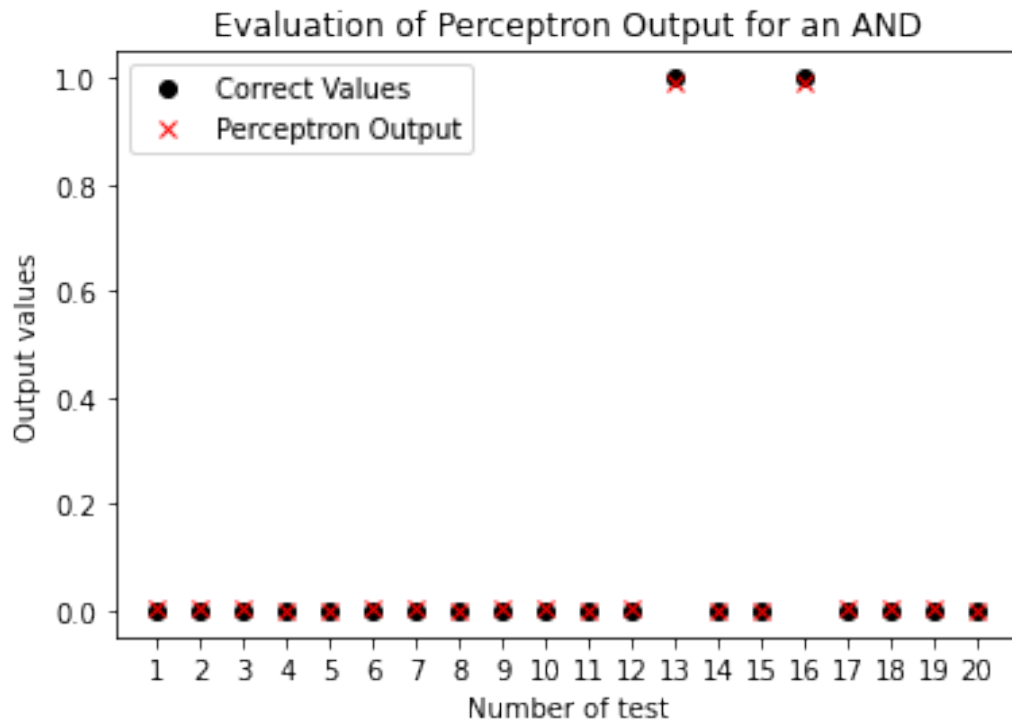


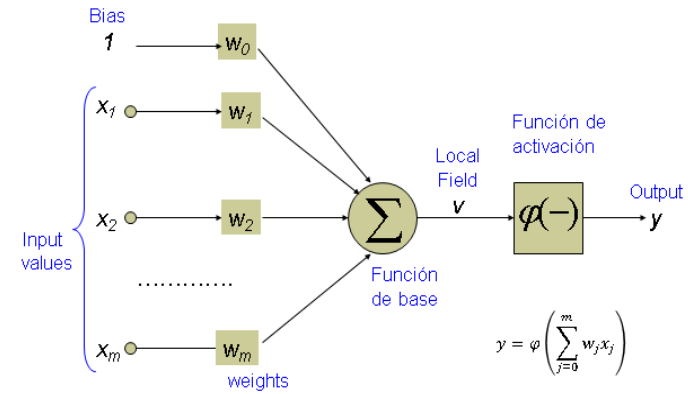


```
1 def train_perceptron(myperceptron, LR, input, output):
2     # Función que modifica los pesos del perceptrón para que vaya aprendiendo a partir
3     # de los vales de entrada que se le indican
4     # INPUTS
5     # myperceptron: estructura con el perceptron
6     # LR: tasa de aprendizaje
7     # input: matriz con valores de entrada de entrenamiento ([E1 E2])
8     # output: vector con valores de salida de entrenamiento ([SE])
9
10    # OUTPUTS
11    # myperceptron: perceptron ya entrenado
12    # bias: bias del perceptron
13    # weights: pesos del perceptron
14
15    # ESTE PERCEPTRÓN UTILIZA:
16    # Función de activación sigmoidal
```

```
1 def useperceptron(myperceptron, input):
2     # funcion que utiliza el perceptron para calcular las salidas a partir de
3     # las entradas de acuerdo con lo que haya aprendido el perceptron en la
4     # fase de entrenamiento
5
6     # INPUTS
7     # myperceptron: perceptron
8     # input: entrada que se le pasara al perceptron (datos test)
9     # OUTPUTS
10    # out: salida
```


Visualizar Resultados





a. EVALUACIÓN CON LA FUNCIÓN AND

IN1	IN2	OUT
0	0	0
0	1	0
1	0	0
1	1	1

b. EVALUACIÓN CON LA FUNCIÓN XOR

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

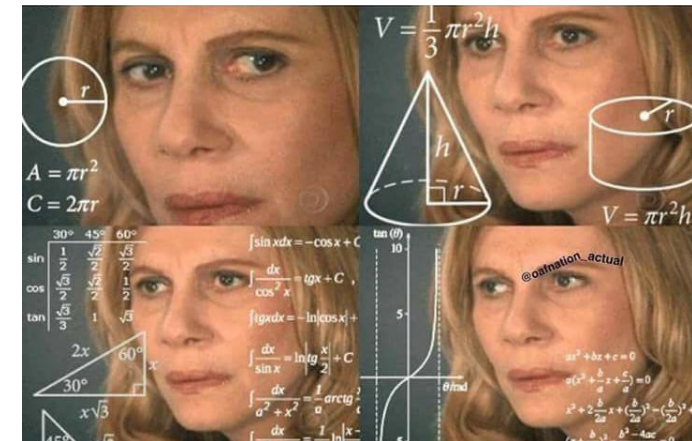
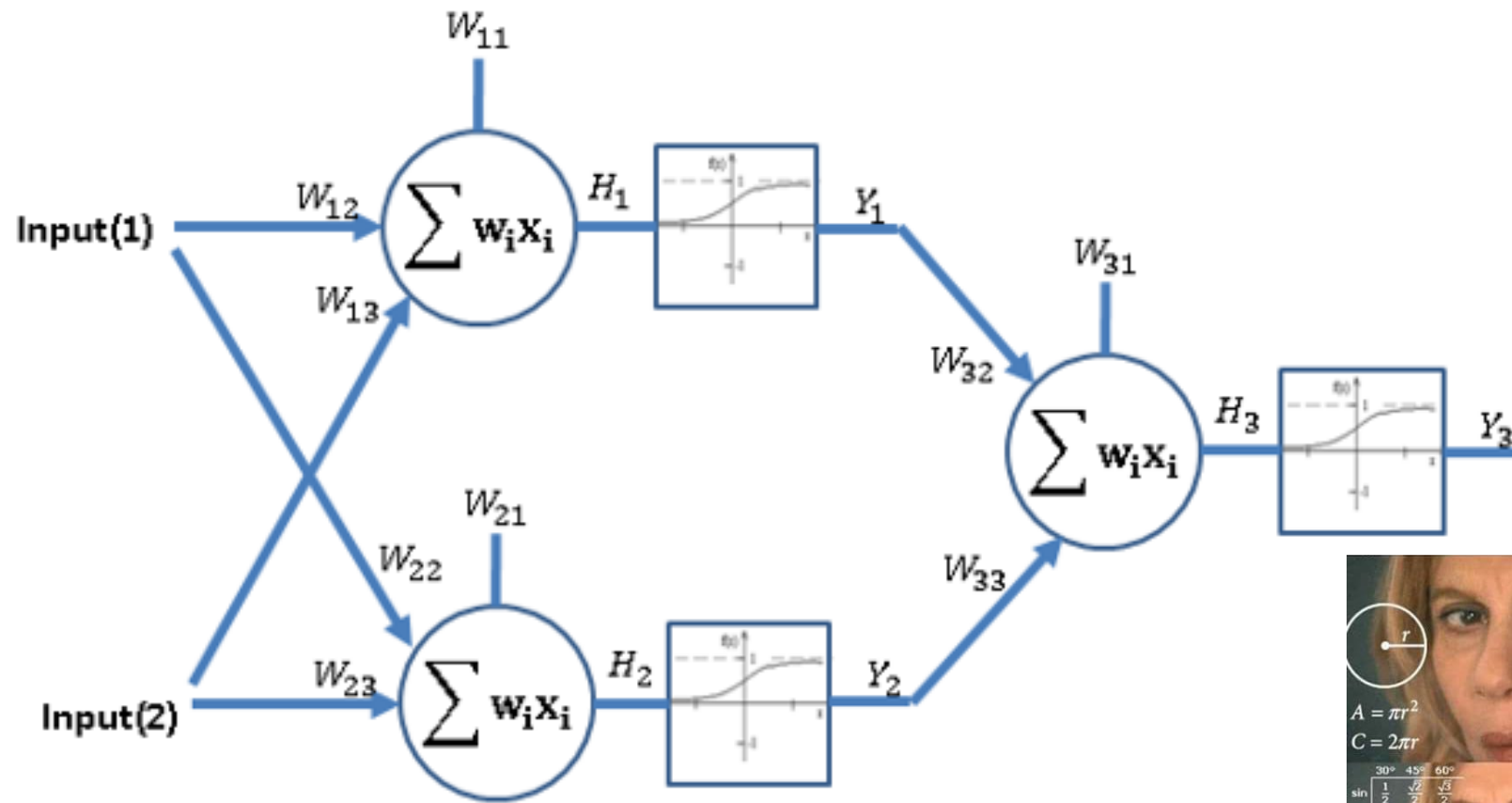
Perceptrón Multicapa

- 1) Creación de las variables para el banco de entrenamiento y banco de validación
- 2) Creación de la red neuronal de **DOS CAPAS**
- 3) Entrenamiento de la red neuronal con el set de valores del banco de entrenamiento
- 4) Validación de la red con el banco de validación.
- 5) Calculo y representación del error cometido.

Perceptrón Multicapa

```
43 def initialize_nn(n_inputs):
44     # INPUTS
45     # n_inputs: numero de entradas a la nn
46     # OUTPUTS
47     # bias: bias de la nn
48     # weights: matriz de pesos de la nn
49     bias = np.ones(n_inputs + 1) # tres bias
50     weights = np.random.uniform(low=-1, high=1, size=(n_inputs+1,n_inputs+1)) #inicializamos pesos
51
52     out = {"weights": weights, "bias": bias}
53     return out
54
```

RED NEURONAL MULTICAPA



RED NEURONAL MULTICAPA

BACKPROPAGATION

$$w_{kj}(t + 1) = w_{kj}(t) + \eta \cdot \delta_k(t) \cdot x_j(t)$$

Siendo k el número de neurona y j el número de peso de la neurona.

w el peso,

j el número de peso de la neurona

η la constante de aprendizaje (e.g. 0.7)

δ el error cometido

x_j la entrada j

δ_k : depende de si estamos en la neurona de la última capa o en una neurona oculta:

Neurona 3 (neurona de la capa final):

$$\delta_3(t) = Y_3(t) \cdot [1 - Y_3(t)] \cdot [OUT(t) - Y_3(t)]$$

Siendo OUT(t) el valor de salida esperado.

Neurona 1 y 2 (neuronas de la capa intermedia):

$$\delta_1(t) = Y_1(t) \cdot [1 - Y_1(t)] \cdot w_{32} \cdot \delta_3(t)$$

$$\delta_2(t) = Y_2(t) \cdot [1 - Y_2(t)] \cdot w_{33} \cdot \delta_3(t)$$

δ_k : depende de si estamos en la neurona de la última capa o en una neurona oculta:

Neurona 1 y 2 (neuronas de la capa oculta):

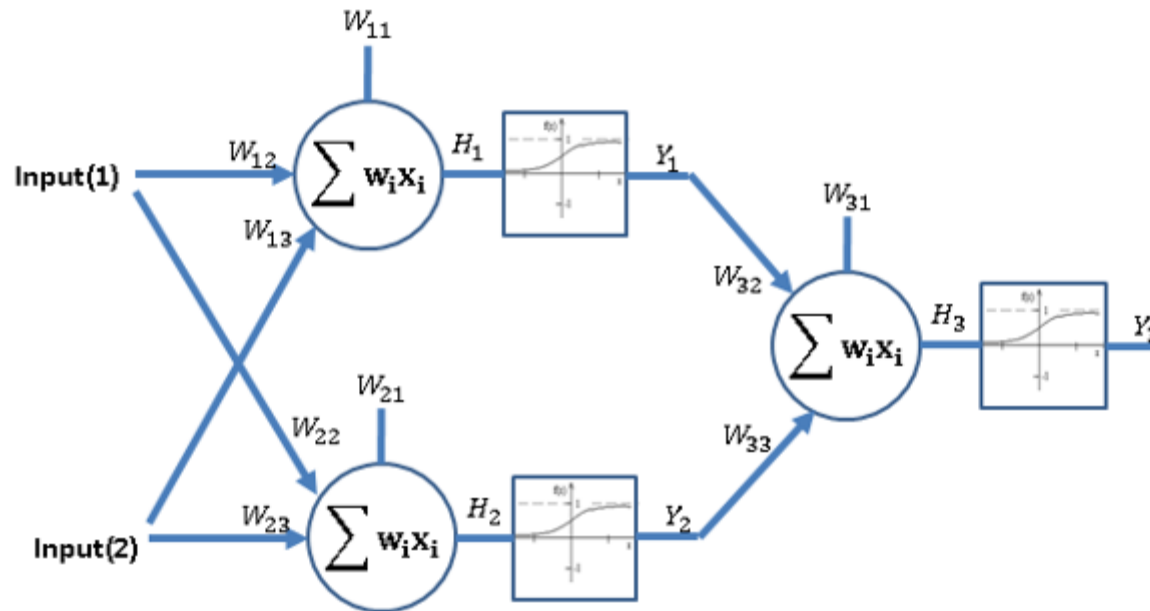
$$\delta_1(t) = Y_1(t) \cdot [1 - Y_1(t)] \cdot w_{32} \cdot \delta_3(t)$$

$$w_{11}(t + 1) = w_{11}(t) + \eta \cdot \delta_1(t) \cdot 1$$

$$w_{12}(t + 1) = w_{12}(t) + \eta \cdot \delta_1(t) \cdot Input_1$$

$$w_{13}(t + 1) = w_{13}(t) + \eta \cdot \delta_1(t) \cdot Input_2$$

Siendo OUT(t) el
valor de salida
esperado.



δ_k : depende de si estamos en la neurona de la última capa o en una neurona oculta:

Neurona 3 (neurona de la capa final):

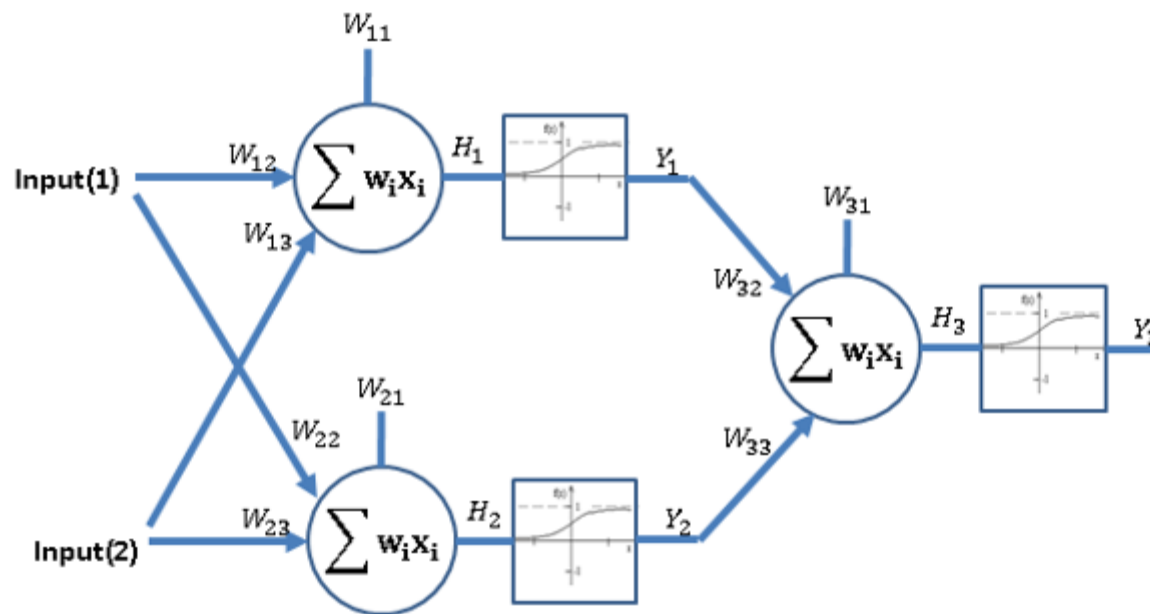
$$\delta_3(t) = Y_3(t) \cdot [1 - Y_3(t)] \cdot [OUT(t) - Y_3(t)]$$

$$w_{31}(t + 1) = w_{31}(t) + \eta \cdot \delta_3(t) \cdot 1$$

$$w_{32}(t + 1) = w_{32}(t) + \eta \cdot \delta_3(t) \cdot Y_1$$

$$w_{33}(t + 1) = w_{33}(t) + \eta \cdot \delta_3(t) \cdot Y_2$$

Siendo $OUT(t)$ el
valor de salida
esperado.





APRENDER
A UTILIZAR
LA LIBRERÍA
DE PYTHON
SCIKIT
LEARN.

sklearn.neural_network.MLPClassifier

```
class sklearn.neural_network.MLPClassifier(hidden_layer_sizes=(100,), activation='relu', *, solver='adam', alpha=0.0001,
batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None,
tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False,
validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000)
```

[\[source\]](#)

Multi-layer Perceptron classifier.

This model optimizes the log-loss function using LBFGS or stochastic gradient descent.

New in version 0.18.

Parameters: **hidden_layer_sizes** : tuple, length = $n_{\text{layers}} - 2$, default=(100,)
The ith element represents the number of neurons in the ith hidden layer.

activation : {'identity', 'logistic', 'tanh', 'relu'}, default='relu'

Activation function for the hidden layer.

- 'identity', no-op activation, useful to implement linear bottleneck, returns $f(x) = x$
- 'logistic', the logistic sigmoid function, returns $f(x) = 1 / (1 + \exp(-x))$.
- 'tanh', the hyperbolic tan function, returns $f(x) = \tanh(x)$.
- 'relu', the rectified linear unit function, returns $f(x) = \max(0, x)$

Methods

<code>fit(X, y)</code>	Fit the model to data matrix X and target(s) y.
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>partial_fit(X, y[, classes])</code>	Update the model with a single iteration over the given data.
<code>predict(X)</code>	Predict using the multi-layer perceptron classifier.
<code>predict_log_proba(X)</code>	Return the log of probability estimates.
<code>predict_proba(X)</code>	Probability estimates.
<code>score(X, y[, sample_weight])</code>	Return the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.



1 sklearn_perceptron_XOR()



[0 0 0 1 0 1 1 0 0 0 1 0 1 0 1 0 1 1]

