PRÁCTICA 4. Neural Networks (NN)

0. 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.
- Consolidad el aprendizaje de la práctica anterior para visualizar gráficas con matplotlib
- Entrenar, validar y comprobar el correcto funcionamiento de una red neuronal.

IMPORTANTE:

- **¿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** (IMP: No se admitirán ningún archivo subido después de dicha hora). Cada alumno debe subir una copia del código, aunque se haya hecho en parejas. Los nombres de los archivos a subir se indican durante el desarrollo de la práctica.

- **¿CÓMO VAN A EVALUARME ESTA PRÁCTICA?** La nota de la práctica será una ponderación entre la calidad e innovación del código entregado (Se evaluará muy positivamente la inclusión de cuantos más **comentarios posibles en él código** que expliquen y justifiquen la que hace cada línea de código) y el resultado de la **entrevista personalizada** que se realizará a los presencialmente.

- **¿QUÉ TENGO QUE ENTREGAR?** **Cada alumno debe subir una copia** del código en su poliformat independientemente de que 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.

PRÁCTICA 4. Neural Networks (NN)

1. PROGRAMACIÓN DE UN PERCEPTRÓN

a. EVALUACIÓN CON LA FUNCIÓN AND

Un perceptrón es la red neuronal más simple en el que no hay capas de neuronas ocultas, tal y como se muestra en el diagrama de la Figura 1.

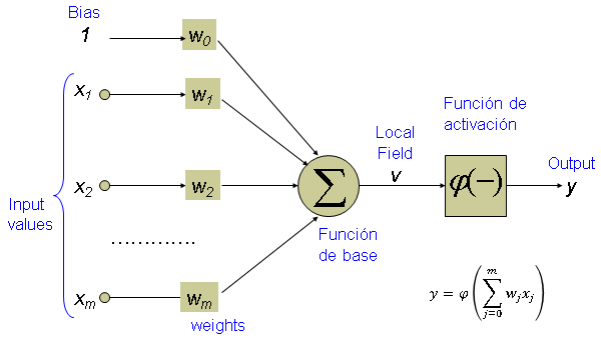


Figura 1. Esquema de un perceptrón.

En la presente sección de la práctica vamos a programar y evaluar una red neuronal capaz de memorizar funciones lógicas lineales. En concreto vamos a programar un perceptrón capaz de similar una puerta lógica AND. Es decir, este perceptrón tendrá 2 entradas que podrán valer 0 o 1 y una única salida:

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

Para la programación de esta red neuronal necesitarás implementar en Python las siguientes funciones:

1. 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)`

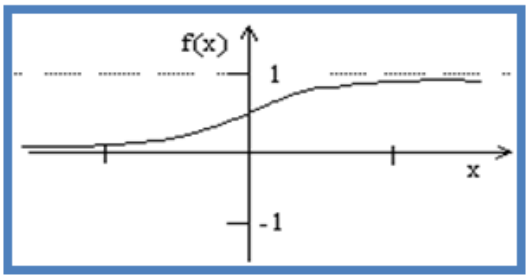
2. Función que entrene el perceptrón partiendo de datos de entrenamiento. Utiliza una función de activación sigmoidal y backpropagation para el entrenamiento
   `train_perceptron(myperceptron,LR,input,output)`

3. Función que evalúe el funcionamiento de la red para un conjunto de datos de validación:
   `useperceptron(myperceptron,input)`

4. 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`

Dicha instrucción principal servirá para crear, entrenar y comprobar el correcto funcionamiento de la red neuronal. En concreto deberás utilizar un banco de 10000 muestras para entrenamiento de la red y un banco de 20 muestras para la evaluación del resultado de la red ya entrenada.

En este ejemplo deberás programar el perceptrón utilizando como función de transferencia la función sigmoidal con un parámetro ***a*** igual a 1:



Sigmoid

$$\varphi(v) = \frac{1}{1+\exp(-av)}$$

Los valores iniciales de los pesos deberán ser aleatorios y el mecanismo de aprendizaje será el delta rule utilizando un parámetro de aprendizaje η igual a 0.7.

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

PRÁCTICA 4. Neural Networks (NN)

En el siguiente pseudo código se indica la estructura que deberá seguir el código:

```
def myperceptron_AND_DNI():
  # Función principal que realiza las funciones de:
  # 1) Creación de las variables para el banco de entrenamiento y b
anco de validación
  # 2) Creación de la red neuronal (perceptrón)
  # 3) Entrenamiento de la red neuronal con el set de valores del b
anco de entrenamiento
  # 4) Validación de la red con el banco de validación
  # 5) Cálculo y representación del error cometido

  # Creamos las entradas y salidas de entrenamiento para una funció
n AND

  # Inicializamos las variables E1, E2 y SE ideales para entrenamie
nto

  # INCLUYE TU CÓDIGO

  VT = 10000 #número de muestras de entrada
  E1 = np.random.randint(2, size=VT)
  E2 = np.random.randint(2, size=VT)
  SE = E1 & E2 # output función AND
  input = np.column_stack((E1, E2))


  # Creamos las entradas y salidas de validación E1V, E2V, SEV para
 test
  vtest = 20 #número de muestras de entrada


  # INCLUYE TU CÓDIGO

  # Inicializamos un perceptrón para 2 entradas
  myperceptron = initialize_perceptron(2)

  # Entrenamos el perceptron para un LR, por defecto 0.7
  LR = 0.7
  myperceptronT=train_perceptron(myperceptron,LR,input,SE)

  # Evaluamos el perceptrón

  S_est=useperceptron(myperceptronT,input_test)

  error = np.mean(abs(SEV-S_est))
```

```
  # Visualización del Resultado
  ### INCLUYE TU CÓDIGO para visualizar los datos de Test correctos
y la solución
  # dada por el perceptrón

  return None
```

Además tendrás que crear las siguientes funciones auxiliares:

```
def initialize_perceptron(n_inputs):
  # Esta función crea e inicializa la estructura myperceptron.weigh
ts
  # donde se guardan los pesos del perceptron. Los valores de los p
esos
  # iniciales deben ser números aleatorios entre -1 y 1.
  # n_inputs: numero de entradas al perceptron
  # OUTPUTS
  # bias: bias del perceptron
  # weights: pesos del perceptron
```

INCLUIR CÓDIGO

```
def sigmoid(x):
    # Funcion de activacion sigmoide
```

INCLUIR CÓDIGO

```
def train_perceptron(myperceptron, LR, input, output):
    # Función que modifica los pesos del perceptrón para que vaya a
prendiendo a partir
    # de los vales de entrada que se le indican
    # INPUTS
    # myperceptron: estructura con el perceptron
    # LR: tasa de aprendizaje
    # input: matriz con valores de entrada de entrenamiento ([E1 E2
])
    # output: vector con valores de salida de entrenamiento ([SE])

    # OUTPUTS
    # myperceptron: perceptron ya entrenado
    # bias: bias del perceptron
    # weights: pesos del perceptron

    # ESTE PERCEPTRÓN UTILIZA:
```

```
    # Función de activación  sigmoidal
```

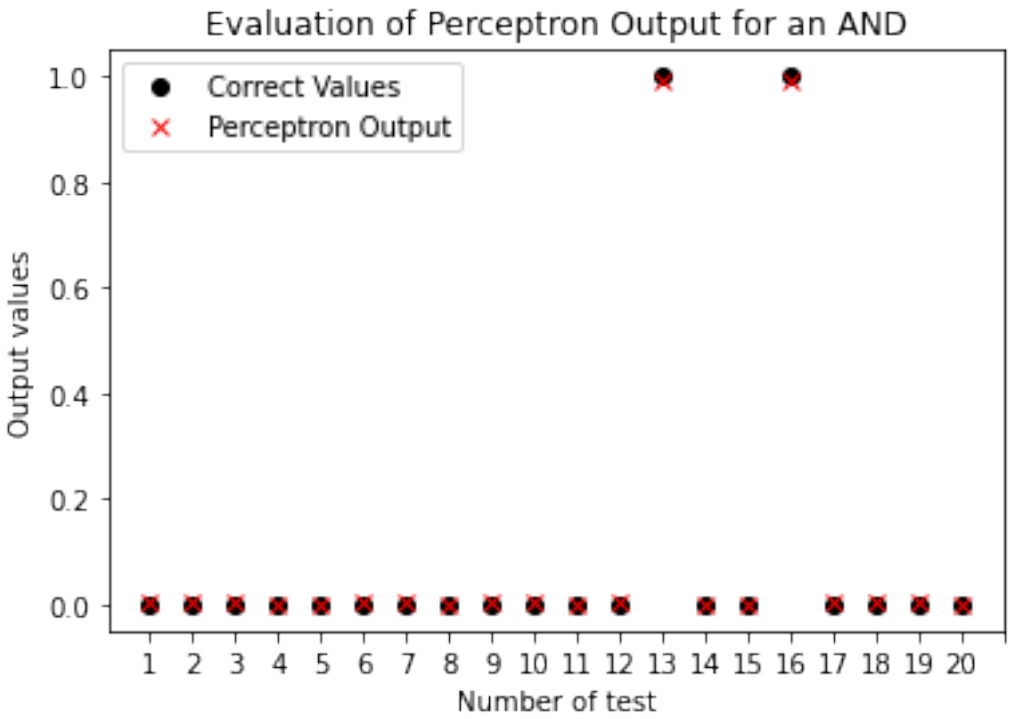
INCLUIR CÓDIGO

```
def useperceptron(myperceptron, input):
    # funcion que utiliza el perceptron para calcular las salidas a
partir de
    # las entradas de acuerdo con lo que haya aprendido el perceptr
on en la
    # fase de entrenamiento

    # INPUTS
    # myperceptron: perceptron
    # input: entrada que se le pasara al perceptron (datos test)
    # OUTPUTS
    # out: salida
```

INCLUIR CÓDIGO

La figura de salida debe ser similar a la siguiente figura, aunque con valores concretos de la serie distintos:



b. EVALUACIÓN CON LA FUNCIÓN XOR

Modifica las funciones anteriores que sean necesarias para que en vez de entrar la red mediante la instrucción and, la entrenes utilizando la instrucción XOR:

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

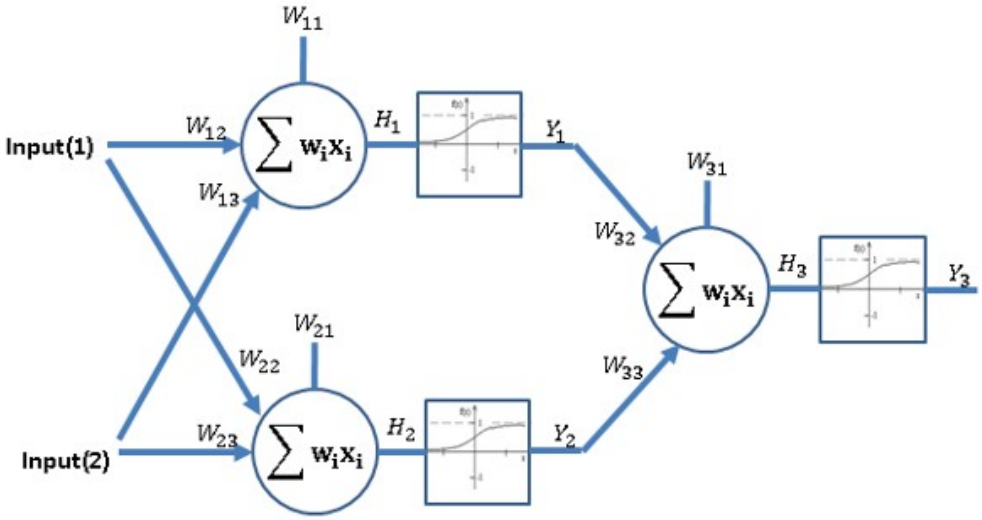
Está función deberá llamarse:

`myperceptron_XOR_DNI`

¿Cómo es la salida en este caso? ¿A qué se deben los errores cometidos?

2. RED NEURONAL MULTICAPA

Ahora vamos a programar una red con dos capas de neuronas para conseguir que efectivamente sea capaz de aprender cómo funciona una red XOR. Donde cada neurona de la primera capa tendrá solo una entrada. El esquema de las 3 neuronas que vamos a programar será el siguiente:



En este caso también vamos a utilizar como función de transferencia la función sigmoidal con un parámetro α igual a 1. Ahora el método de entrenamiento que utilizaremos será el BACKPROPAGATION con un coeficiente de aprendizaje η=0.7. Es decir la variación de los pesos en cada iteración del entrenamiento será igual a la siguiente función, donde delta se calculará de forma distinta si se trata de la neurona de salida o de una neurona de la capa oculta:

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

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

η: Constante aprendizaje (e.g. 0.7)

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):

$$\delta_1(t) = Y_1(t) \cdot [1 - Y_1(t)] \cdot w_{32} \cdot \delta_3(t)$$
$$\delta_2(t) = Y_2(t) \cdot [1 - Y_2(t)] \cdot w_{33} \cdot \delta_3(t)$$

PRÁCTICA 4. Neural Networks (NN)

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.

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

$$\delta_1(t) = Y_1(t) \cdot [1 - Y_1(t)] \cdot w_{32} \cdot \delta_3(t)$$
$$w_{11}(t+1) = w_{11}(t) + \eta \cdot \delta_1(t) \cdot 1$$
$$w_{12}(t+1) = w_{12}(t) + \eta \cdot \delta_1(t) \cdot Input_1$$
$$w_{13}(t+1) = w_{13}(t) + \eta \cdot \delta_1(t) \cdot Input_2$$

Deberás programar una estructura de funciones similar a la anterior pero ahora utilizando una red neuronal en vez de solo una neurona. El principal cambio radica en que antes solo teníamos en cuenta 3 pesos y ahora deberemos tener en cuenta 9 pesos, 3 por cada neurona. Para ello sustituiremos el vector de pesos por una matriz de pesos.

La fusión principal que debes crear se llamará: **mynn_XOR_DNI()**

```
def mynn_XOR_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 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) Cálculo y representación del error cometido.

    # Inicializamos las variables E1, E2 y SE ideales para entrenam
iento
    VT=10000                #numero de muestras de entrada
    NV=np.around(VT)           #aseguramos que VT sea un numero ent
ero
    E1 = np.random.randint(2, size=VT)
    E2 = np.random.randint(2, size=VT)
    SE= E1 ^ E2  #vector salida ideal para las entradas E1 y E
    input = np.column_stack((E1, E2))

    # Creamos entradas y salidas de validacion E1V, E2V, SEV para t
est
```

```
    vtest = 20 #número de muestras de entrada
    E1V = np.random.randint(2, size=vtest)
    E2V = np.random.randint(2, size=vtest)
    input_test = np.column_stack((E1V, E2V))
    SEV = E1V ^ E2V

    # Inicializamos un perceptron para 2 entradas
    mynn=initialize_nn(2)

    # Entrenamos el perceptron para un LR, por defecto 0.7
    LR=0.7
    mynnT=train_nn(mynn,LR,input,SE)

    # evaluamos el perceptron
    S_est=use_nn(mynnT,input_test)

    # Calculamos y representamos el error cometido
    error=abs(SEV-S_est)

    # Visualizamos los resultados

    return

def initialize_nn(n_inputs):
    # INPUTS
    # n_inputs: numero de entradas a la nn
    # OUTPUTS
    # bias: bias de la nn
    # weights: matriz de pesos de la nn
    bias = np.ones(n_inputs +1) # tres bias
    weights = np.random.uniform(low=-1, high=1, size=(n_inputs
+1,n_inputs+1))   #inicilizamos pesos

    out = {"weights": weights, "bias": bias}
    return out


def train_nn(mynn, LR, input_nn, output_nn):
    # Función que toma una red inicializada, un LR y unas entradas
    # y devuelve la red tras ser entrenada.
    # INPUTS:
    # mynn: red neuronal inicializada (bias y pesos)
    # LR: learning rate
    # input_nn: vectores de entrada para entrenar
    # output_nn: vector de salida esperada de los vectores de entra
da
```

```
    # OUTPUTS:
    # mynn: red entrenada
    return
```

```
def use_nn(mynn, input_nn):
    # Función que utiliza el perceptrón para calcular las salidas a
partir de
    # las entradas de acuerdo con lo que haya aprendido el perceptr
on en la
    # fase de entrenamiento
    # INPUTS:
    # mynn: red neuronal ya entrenada
    # input_nn: vectores de entrada que se le van a pasar a la red
(test)
    # OUTPUTS:
    # out: vector de salida con las soluciones estimadas
```

3. RED NEURONAL CON SCIKIT LEARN

Python cuenta con múltiples recursos y librerías que nos permiten implementar redes neuronales de manera sencilla y óptima.

Scikit-learn es una de las librerías más utilizadas en *machine learning*.
Podrás encontrar más información en el siguiente enlace:
https://scikit-learn.org/stable/modules/neural_networks_supervised.html

En este apartado tendrás que realizar una red neuronal para la función XOR utilizando las funciones propias de esta librería y visualizar los resultados como en los ejercicios anteriores.

La nueva función que implementaréis en este apartado se llamará:
sklearn_NN_XOR()

Las funciones principales a utilizar de esta librería son:

MLPClassifier

fit

predict