❑ DEEP LEARNING. REDES NEURONALES.

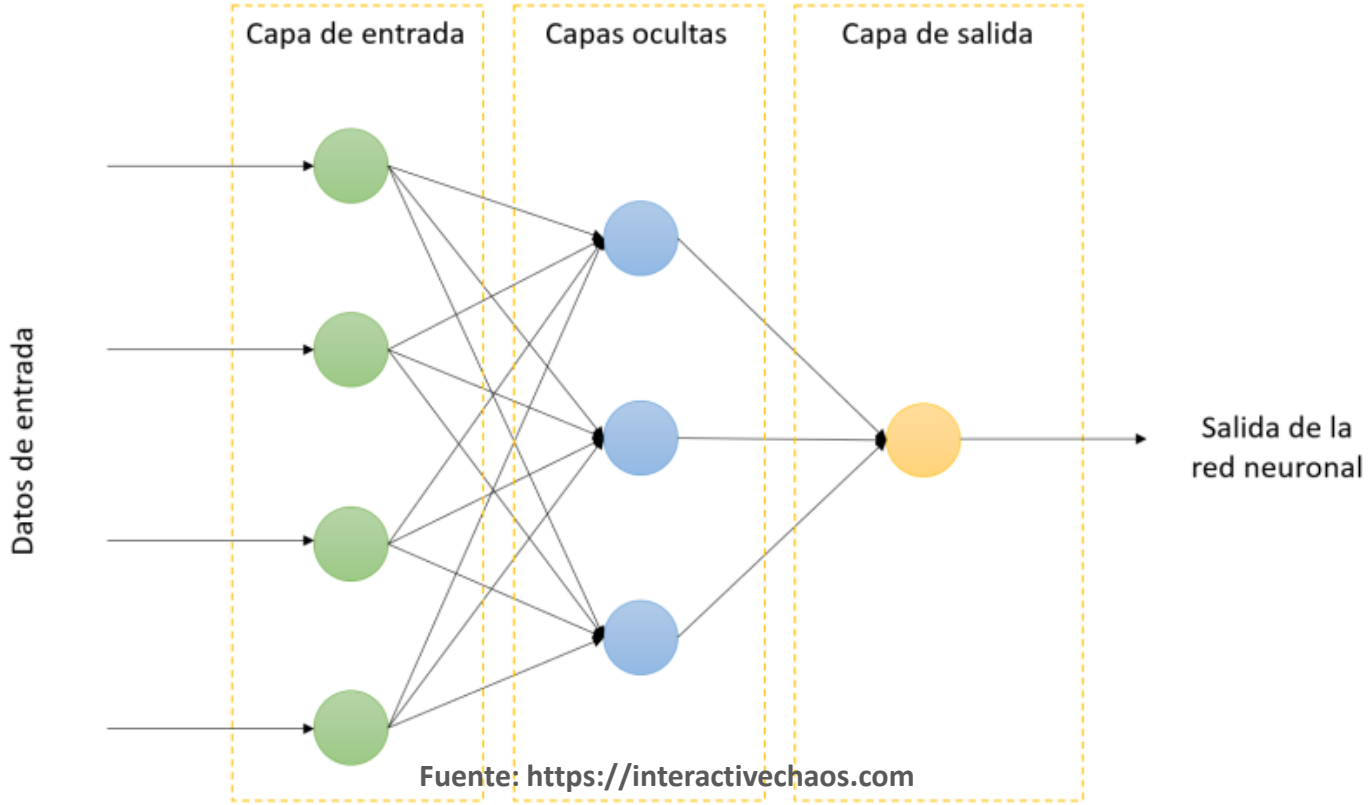
La información que llega a la red neuronal, va recorriéndola y se transforma en cada neurona artificial atravesada, hasta alcanzar el final de la red, donde se devuelve el valor resultante.

La idea de crear redes neuronales artificiales usando la salida de unas neuronas como entradas de otras se remonta a la década de 1960.

Estructura de una red neuronal:

- **Datos de entrada.**
- **Capa de entrada: se asignan los pesos iniciales a cada dato.**
- **Capas ocultas: formadas por neuronas. Las salidas de una capa son entradas de la siguiente capa.**
- **Capa de salida: capa final con una o más neuronas.**
- **Salida de la Red.**

❑ DEEP LEARNING. REDES NEURONALES.



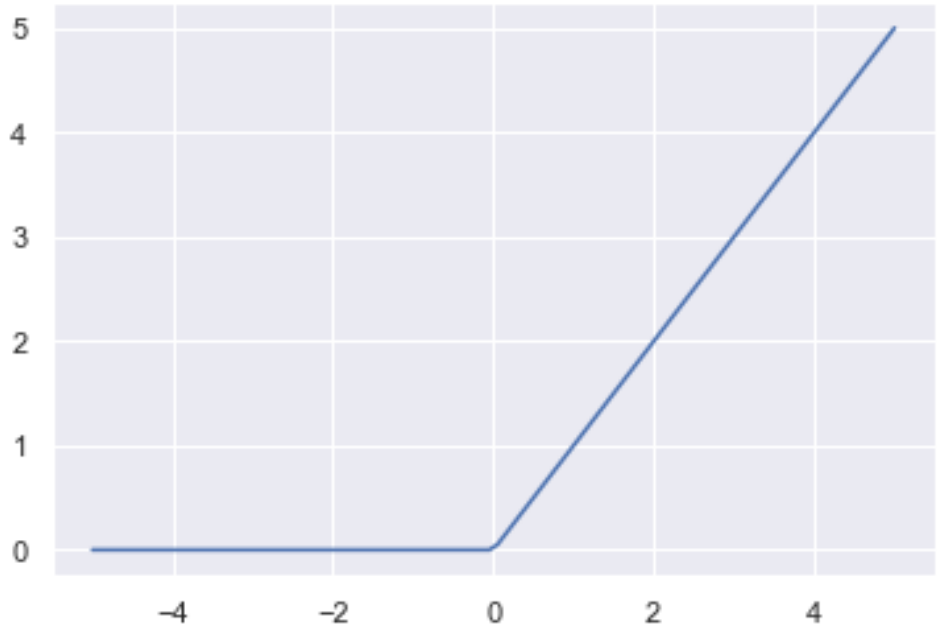
Fuente: https://interactivechaos.com

Las salidas de todas las neuronas de una capa sirven de entrada para todas las neuronas de la siguiente capa. En arquitecturas más complejas hay retroalimentaciones.

❑ DEEP LEARNING. REDES NEURONALES.

Funciones de activación: una de las más usadas es la Rectified Linear Unit (frecuentemente llamada ReLU).

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



❑ DEEP LEARNING. REDES NEURONALES.

Perceptrón multicapa:

➢ Esta arquitectura no está basada en perceptrones, simplemente se ha conservado el término "percentrón" en el nombre por motivos históricos.

➢ Esta arquitectura se corresponde con redes neuronales formadas por capas de neuronas. Los datos van desde la capa de entrada hasta la de salida, sin retroalimentaciones, y la salida de cada neurona sirve de entrada para todas las neuronas de la siguiente capa.

❑ REDES NEURONALES. Perceptrón multicapa.

Ejemplo: vamos a entrenar una red neuronal con una capa oculta de 2 neuronas.

- **Abrir un nuevo Notebook llamado 'redesNeuronales'.**
- **Clase MLPClassifier de Scikit-Learn:**
  - ✓ **El parámetro 'hidden_layer_sizes': sirve para indicar el número de neuronas artificiales en cada capa oculta.**
    - **(2,): una única capa oculta con 2 neuronas.**
    - **(5, 3): dos capas ocultas con 5 y 3 neuronas artificiales respectivamente.**

❑ REDES NEURONALES. Perceptrón multicapa.

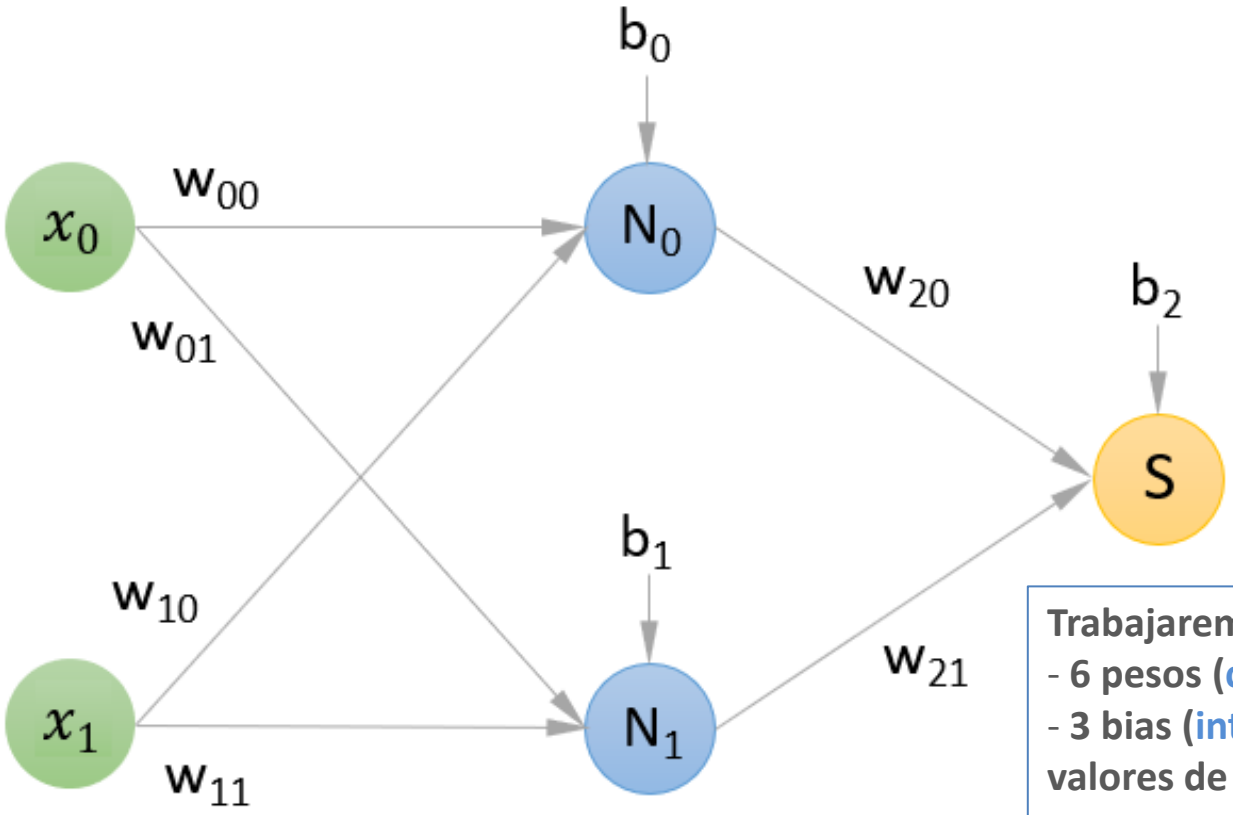
‘MLPClassifier’ de Scikit-Learn crea en la capa de salida tantas neuronas como clases existan, excepto si se trata de clasificación binaria.

Si sólo hay dos clases posibles, sólo se incluye una neurona en la capa de salida, cuya función de activación determina la clase a devolver como predicción.

Vamos a ver la estructura de la red neuronal de ejemplo con una capa oculta de 2 neuronas.

❑ REDES NEURONALES. Perceptrón multicapa.

Ejemplo: vamos a entrenar una red neuronal con una capa oculta de 2 neuronas.



Trabajaremos con:
- 6 pesos (coeficientes).
- 3 bias (interceptores) a añadir a los valores de entrada ponderados de las neuronas.

❑ REDES NEURONALES. Perceptrón multicapa.

Ejemplo: vamos a entrenar una red neuronal con una capa oculta de 2 neuronas.

```
import pandas as pd
```

Supongamos que tenemos el siguiente dataframe.

```
df = pd.DataFrame({
    "x0": [0.4, 0.2, 0.5, 0.6],
    "x1": [0.6, 0.1, 0.2, 0.7],
    "y": [1, 0, 0, 1]
})
df
```

❑ REDES NEURONALES. Perceptrón multicapa.

Ejemplo: vamos a entrenar una red neuronal con una capa oculta de 2 neuronas.

```
# Vamos a considerar categórica la variable objetivo 'y',
# para trabajar en un escenario de clasificación
y_train = df.pop("y")
df
```

	x0	x1
0	0.4	0.6
1	0.2	0.1
2	0.5	0.2
3	0.6	0.7

```
X_train = df
```

❑ REDES NEURONALES. Perceptrón multicapa.

Ejemplo: vamos a entrenar una red neuronal con una capa oculta de 2 neuronas.

```
# Importamos ahora la clase MLPClassifier e instanciamos el algoritmo
from sklearn.neural_network import MLPClassifier
```

Vamos a crear un modelo con una tasa de aprendizaje de 0.1, especificando como función de activación de las capas ocultas la función Rectified Linear Unit (ReLU).

```
model = MLPClassifier(
    (2, ),
    random_state = 0,
    learning_rate_init = 0.1,
    activation = "relu"
)
```

❑ REDES NEURONALES. Perceptrón multicapa.

Ejemplo: vamos a entrenar una red neuronal con una capa oculta de 2 neuronas.

```
# Entrenamos el modelo
model.fit(X_train, y_train)
```

Vamos a ver ahora coeficientes e interceptores

```
model.intercepts_
# nos devuelve un array por cada capa, con los 'bias'
# [b0,b1]
# [b2]
```

❑ REDES NEURONALES. Perceptrón multicapa.

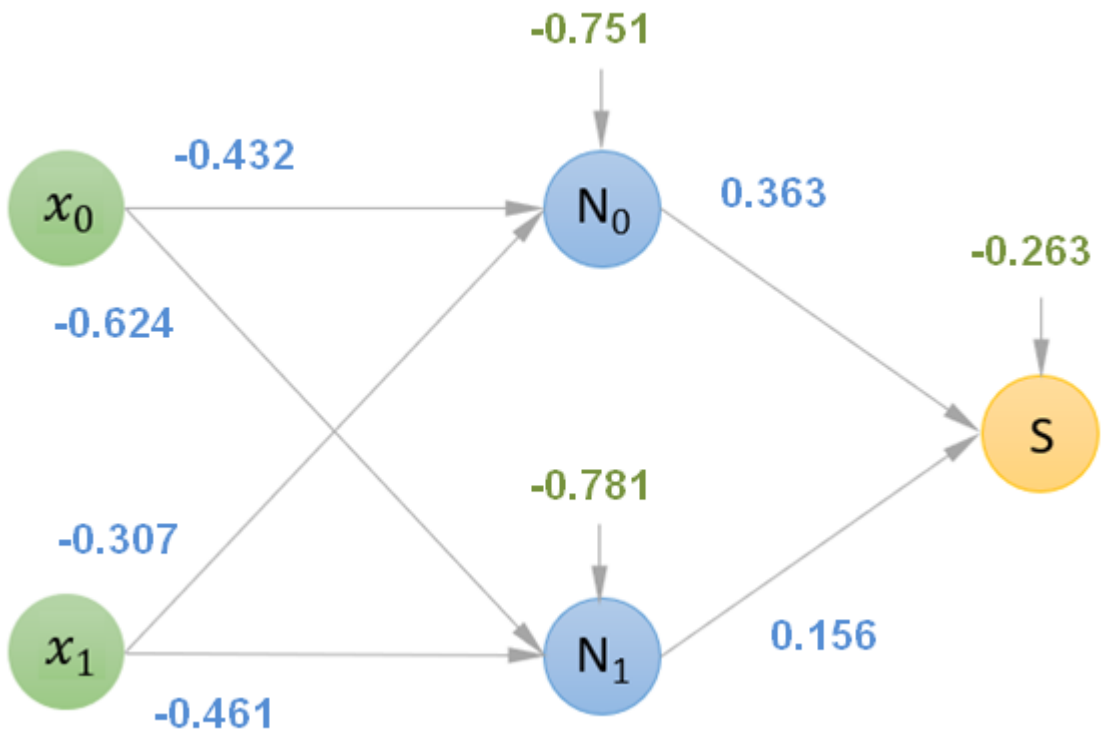
Ejemplo: vamos a entrenar una red neuronal con una capa oculta de 2 neuronas.

```
model.coefs_
# nos devuelve un array por cada capa, con los pesos
# [w00,w01],[w10,w11]
# [w20,w21]
```

```
[array([[-0.4320317 , -0.6241389 ],
        [-0.30745678, -0.46121447]]),
 array([[0.36319675],
        [0.15671958]])]
```

❑ REDES NEURONALES. Perceptrón multicapa.

Ejemplo: vamos a entrenar una red neuronal con una capa oculta de 2 neuronas.



❑ REDES NEURONALES. Perceptrón multicapa.

Ejercicio:

- ✓ Cargar el dataset Iris de ‘seaborn’.
- ✓ Coger las variables predictoras "sepal_length“ y "sepal_width“.
- ✓ Coger ‘species’ codificado como variable objetivo.
- ✓ Escalar las variables de entrada con ‘StandardScaler’.
- ✓ Crear los datasets de entrenamiento y validación.
- ✓ Instanciar el modelo con MLPClassifier (max_iter = 1000).
- ✓ Comprobar el porcentaje de aciertos en el dataset de validación.
- ✓ Visualizar las fronteras de decisión.
- ✓ Repetir el proceso hasta obtener el nuevo score, usando las 4 variables predictoras del dataset Iris.

❑ REDES NEURONALES. Perceptrón multicapa.

Ejercicio: solución.

```
import seaborn as sb
iris = sb.load_dataset("Iris")
iris["label"] = iris.species.astype("category").cat.codes
```

```
X = iris[["sepal_length", "sepal_width"]]
y = iris.label
```

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

❑ REDES NEURONALES. Perceptrón multicapa.

Ejercicio: solución.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y,
                                                    test_size = 0.2)
```

```
from sklearn.neural_network import MLPClassifier
```

```
model = MLPClassifier(max_iter = 1000)
model.fit(X_train, y_train)
```

```
model.score(X_test, y_test)
```

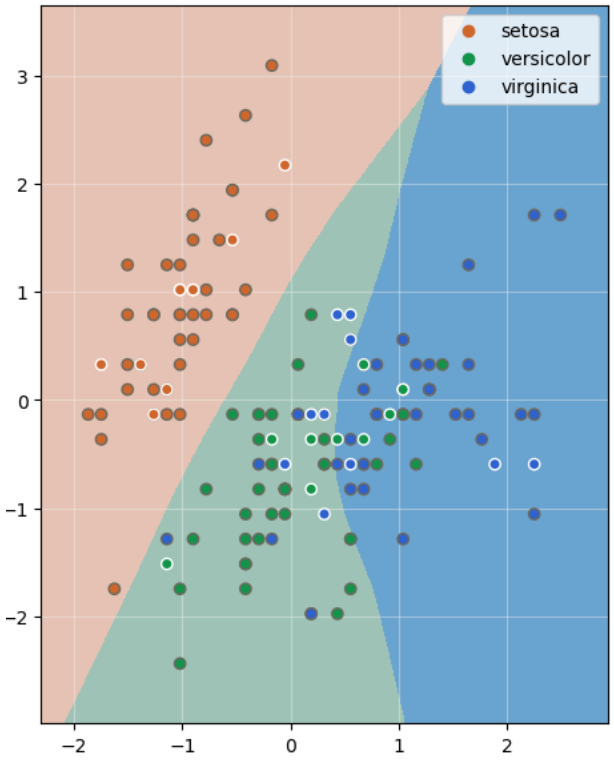
```
0.5666666666666667
```

❑ REDES NEURONALES. Perceptrón multicapa.

```
import fronteras as fr
```

```
fr.mostrar_fronteras(model, X_train, X_test, y_train, y_test, iris.species.unique())
```

Solución del Ejercicio.



❑ REDES NEURONALES. Perceptrón multicapa.

Ejercicio: solución.

```
X = iris.drop(["species", "label"], axis = 1)
y = iris.label
X1 = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X1, y, stratify = y,
                                                    test_size = 0.2)
model = MLPClassifier(max_iter = 1000)
model.fit(X_train, y_train)
model.score(X_test, y_test)
```

```
0.9333333333333333
```