# REDES NEURONALES Y REDES CONVOLUCIONALES
## Introducción

Judith Sáinz-Pardo Díaz

Seguridad, privacidad y aspectos legales

**Este documento resume la idea BÁSICA sobre redes neuronales y convolucionales para poder realizar la práctica de Federated Learning, para aquellos alumnos que no cursan la asignatura Machine Learning I. Para la elaboración se han tomado los apuntes de dicha asignatura como referencia.**

## 1 Introducción. Redes Neuronales

El objetivo del concepto de perceptron o neurona artificial, es simular matemáticamente el comportamiento de una neurona biológica. Esto quiere decir, que sigue el mismo esquema: se recibe un estimulo/entrada, que es procesado (para ello se usa una función de activación), y produce una salida. En las redes neuronales, la salida de algunas neuronas, serán la entrada de otras. Es decir, las redes neuronales artificiales (ANN) se componen de múltiples capa ocultas. En función del número de capas ocultas se tratará de una red neuronal simple, o profunda.

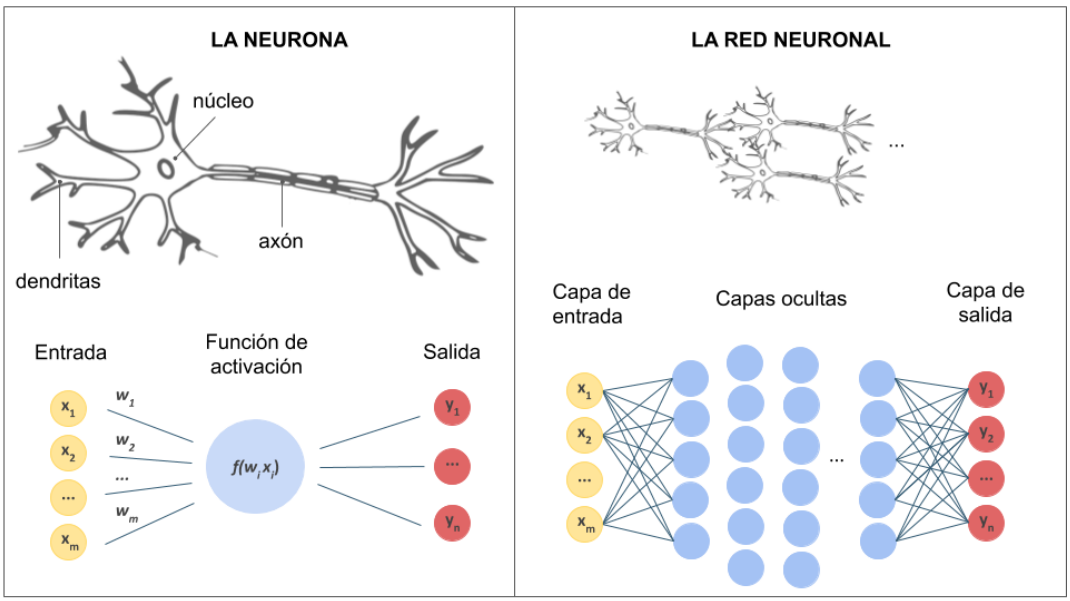


Figure 1: Esquema neurona y red neuronal artificial. Fuente: https://cebebelgica.es/es_ES/blog/10/que-es-una-red-neuronal-artificial.html

Por un lado tenemos las entradas (por ejemplo, en el caso de MNIST, las entradas serán los valores de los píxeles de las imágenes), y por otra parte, tenemos los pesos que se calculan asociados a cada entrada. Una vez tenemos estos pesos se suma el producto de cada entrada y su peso asociado (más un bias), y posteriormente debe aplicarse una función de activación para obtener la salida. Veamos la idea básica de como se procesa la información un perceptrón en la Figura 2:

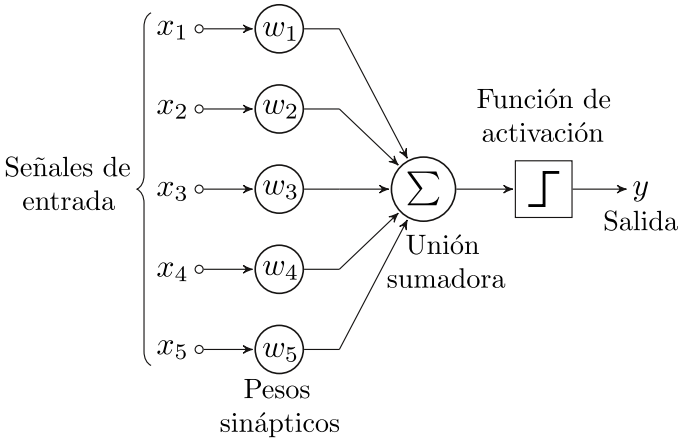


Figure 2: Esquema perceptron. Fuente: https://es.wikipedia.org/wiki/Perceptron

En cuanto a la función de activación, tenemos múltiples opciones, pero las más comunes son: función sigmoide, tangente hiperbólica, ReLU, y softmax, entre otras:

- **Sigmoide**: transforma los valores a una escala entre 0 y 1 (no incluidos).

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

- **Tangente Hiperbólica**: transforma los valores introducidos a una escala entre -1 y 1 (no incluidos).

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

- **ReLU**: transforma en 0 los valores negativos, y deja tal como estaban los positivos.

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

- **Softmax**: transforma las salidas a una representación en forma de probabilidades. Así, la suma de todas será 1. Por ejemplo, en el caso de MNIST, si tenemos la salida [0, 0, 0, 0.98, 0, 0.01, 0.01, 0, 0, 0], tenemos un 98% de probabilidad de que el número sea un 3, un 1% de que sea un 5, y un 1% de que sea un 6.

En https://keras.io/api/layers/core_layers/dense/ podemos ver un ejemplo sencillo de uso de capas densas. Las capas densas son aquellas en que cada neurona de una capa se conecta con todas las salidas de la capa anterior. En primer lugar, destacar que el modelo se crea como *secuencial*, y se van añadiendo capas. En la primera capa es necesario incluir la dimensión del input a introducir. También se especifica la función de activación de la capa. Por último, es necesario compilar el modelo, para lo cual se especifica la función de pérdida a utilizar (p.e. 'categorial_crossentropy'), el optimizador (p.e. 'rmsprop'), y una métrica, como puede ser la accuracy. Para más información sobre las distintas opciones de `model.compile()`, se puede consultar la documentación de `keras` y `tensorflow`. La Figura 4 muestra el ejemplo de un modelo con dos capas densas.

```
model = tensorflow.keras.models.Sequential()
model.add(tensorflow.keras.layers.Dense(64, activation = 'relu', input_shape=input_shape))
model.add(tensorflow.keras.layers.Dense(10, activation = 'softmax'))
model.compile(loss = 'categorical_crossentropy',
              optimizer = 'rmsprop',
              metrics = ['accuracy'])
model.summary()
```

Figure 3: Ejemplo: modelo con 2 capas densas.

## 2 Introducción. Redes Convolucionales

La principal diferencia entre una red densa y una red convolucional, es que mientras que las redes densas aprenden patrones globales (p.e. en el caso de imágenes, utilizan todos los píxeles de la imagen), las convolucionales aprenden patrones locales (en el caso de imágenes, se usan pequeñas ventanas o filtros). Por ello, las redes neuronales convolucionales (CNN) son muy apropiadas para procesar imágenes.

Una vez se aplican las capas convolucionales, en necesario usar una capa "Flatten" para pasar a la parte de clasificación (y entonces aplicar capas, por ejemplo, densas). A continuación se muestra un ejemplo:

```
model = tensorflow.keras.models.Sequential()
model.add(tensorflow.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=input_shape)
model.add((tensorflow.keras.layers.MaxPooling2D((2,2))))
model.add(tensorflow.keras.layers.Conv2D(64, (3,3), activation='relu'))
model.add(tensorflow.keras.layers.Flatten())
```

Figure 4: Ejemplo: modelo con 2 capas convolucionales, una MaxPolling, y una capa final Flatten.

En el ejemplo anterior, el input shape para el caso de imágenes en blanco y negro, de 28x28 píxeles, sería: (28, 28, 1). Si las imágenes son en color, sería (28, 28, 3). Destacar que la capa MaxPolling se utiliza para reducir las dimensiones (ancho y alto), por ejemplo, al aplicarlo en el ejemplo anterior, estaremos reduciendo a 14x14. Sobre las capas de convolución, en https://keras.io/api/layers/convolution_layers/convolution2d/ se encuentra la documentación donde se explica el significado de cada parámetro. Con la capa Flatten, estamos "aplanando" la entrada, es decir, pasamos de una entrada multidimensional a unidimensional. Así, podremos a continuación aplicar capas densas que nos permitan realizar la tarea de clasificación. El esquema de una red convolucional para la clasificación de imágenes, sigue la idea de la Figura 5.

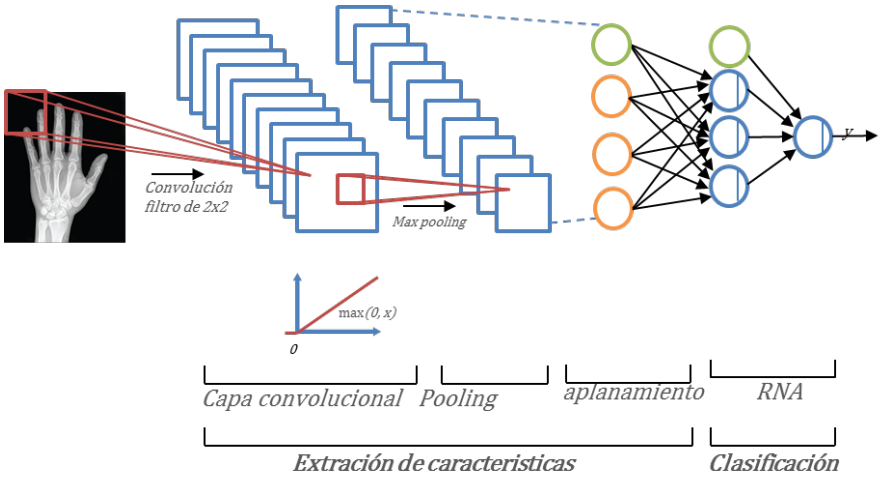


Figure 5: Esquema clasificación de imágenes. Fuente: https://www.redalyc.org/journal/1702/170262877013/html/.

Para la práctica de la asignatura, se va a usar el dataset MNIST (ver https://keras.io/api/datasets/mnist/). Aunque en dicha práctcia sea necesario crear un modelo de redes convolucionales, el objetivo es aplicar un proceso de Federated Learning, con lo que basta con crear un modelo sencillo. Para ello revisad la documentación de `keras` y/o `tensorflow`, y encontraréis también muchos ejemplos online. Recordar que al trabajar con imágenes, debemos introducirlas normalizadas. Por ejemplo, en imágenes en blanco y negro los píxeles tomarán valores entre 0 y 255, luego deberemos dividir entre 255 para que los valores estén entre 0 y 1.