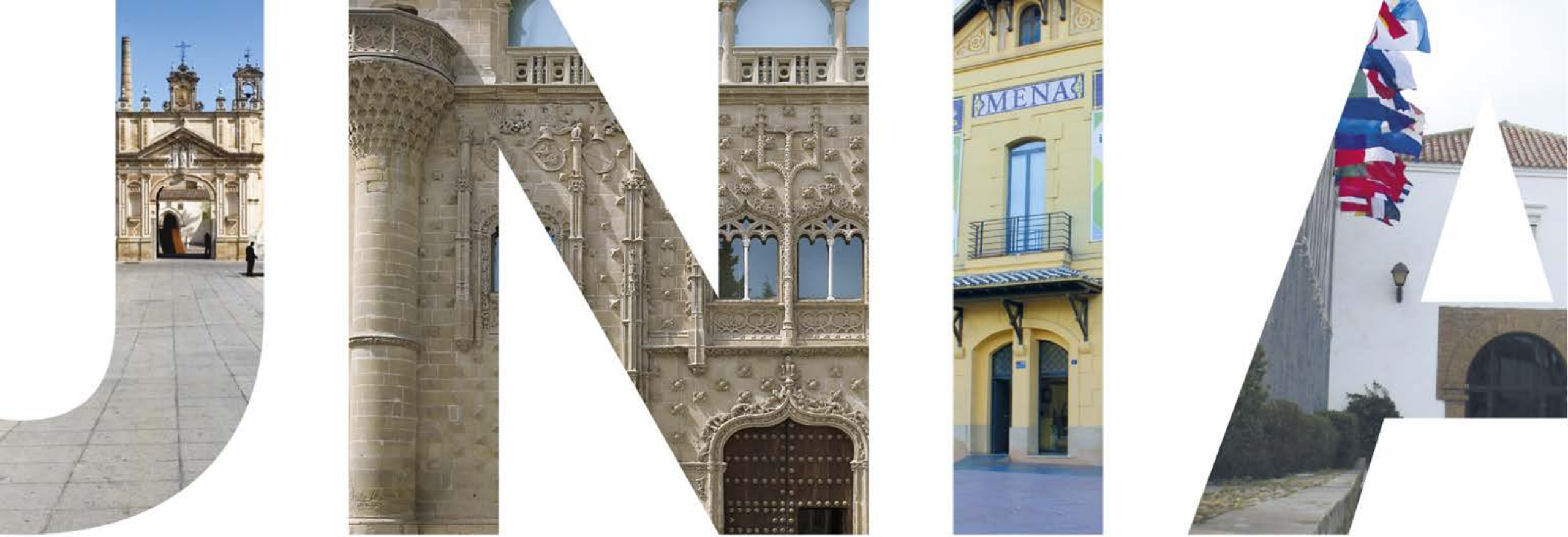


Tu universidad de postgrado

Your university for graduate studies

Tu universidad para una formación permanente

Your lifelong learning university

Tu universidad para una enseñanza innovadora

Your innovative education university

La universidad para tu futuro

The university for your future

unia Universidad Internacional de Andalucía

UNIVERSIDAD INTERNACIONAL DE ANDALUCÍA

BioSiP
Biomedical Signal Processing, Computational Intelligence and Communications Security

UNIVERSIDAD DE MÁLAGA

SiPBA
UNIVERSIDAD DE GRANADA

DaSCI
Instituto Andaluz Interuniversitario en Data Science and Computational Intelligence

UNIVERSIDAD DE GRANADA

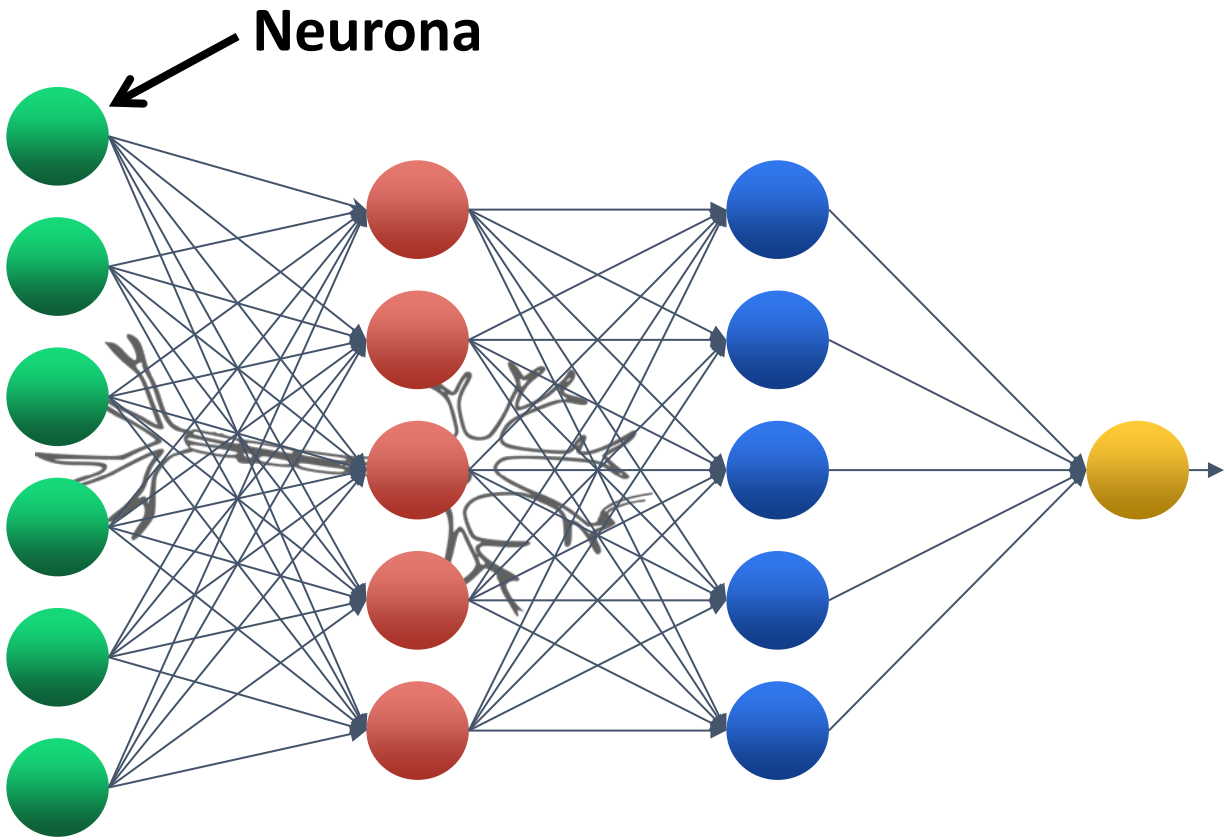
Introducción práctica a la Inteligencia Artificial y al Deep Learning

Introducción a las Redes neuronales

UNIVERSIDAD INTERNACIONAL DE ANDALUCÍA

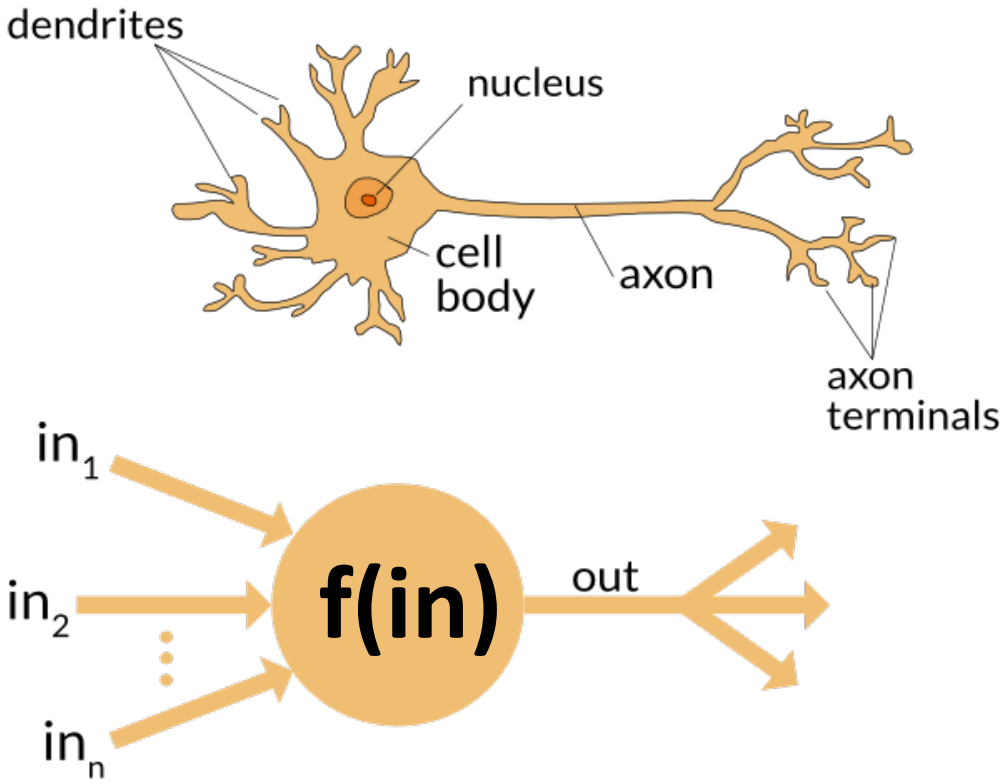
¿Qué es una red neuronal?

- Una red neuronal artificial (RNA) es un modelo computacional formado por la conexión de unidades que, individualmente, realizan un cálculo sencillo
- La teoría conexionista, que está detrás de las RNA, unidades simples interconectadas entre sí, pueden resolver problemas complejos



UNIVERSIDAD INTERNACIONAL DE ANDALUCÍA

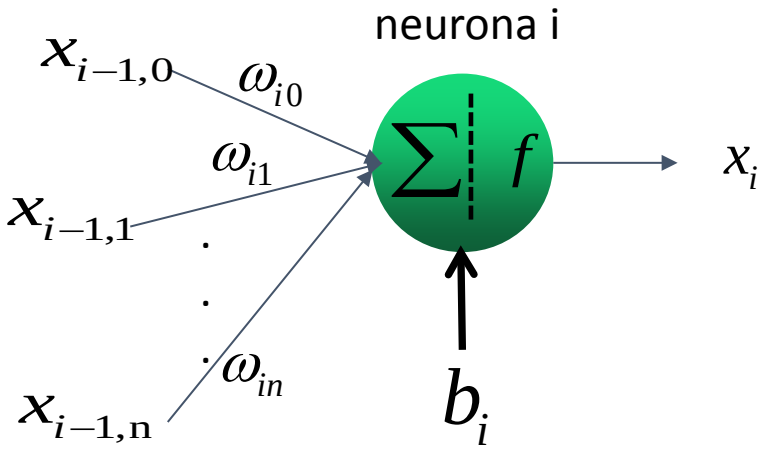
La neurona artificial como unidad básica de procesamiento en una red neuronal



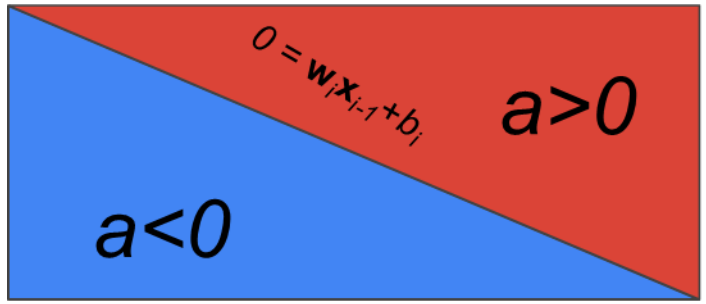
La neurona artificial como unidad básica de una red neuronal

$$x_i = \sum_{j=1}^{n} f\left(x_{i-1,j}\omega_{i,j} + b_i\right)$$

$$f(a) = \begin{cases} 1 & \text{if } a > 0 \\ -1 & \text{if } a < 0 \end{cases}$$

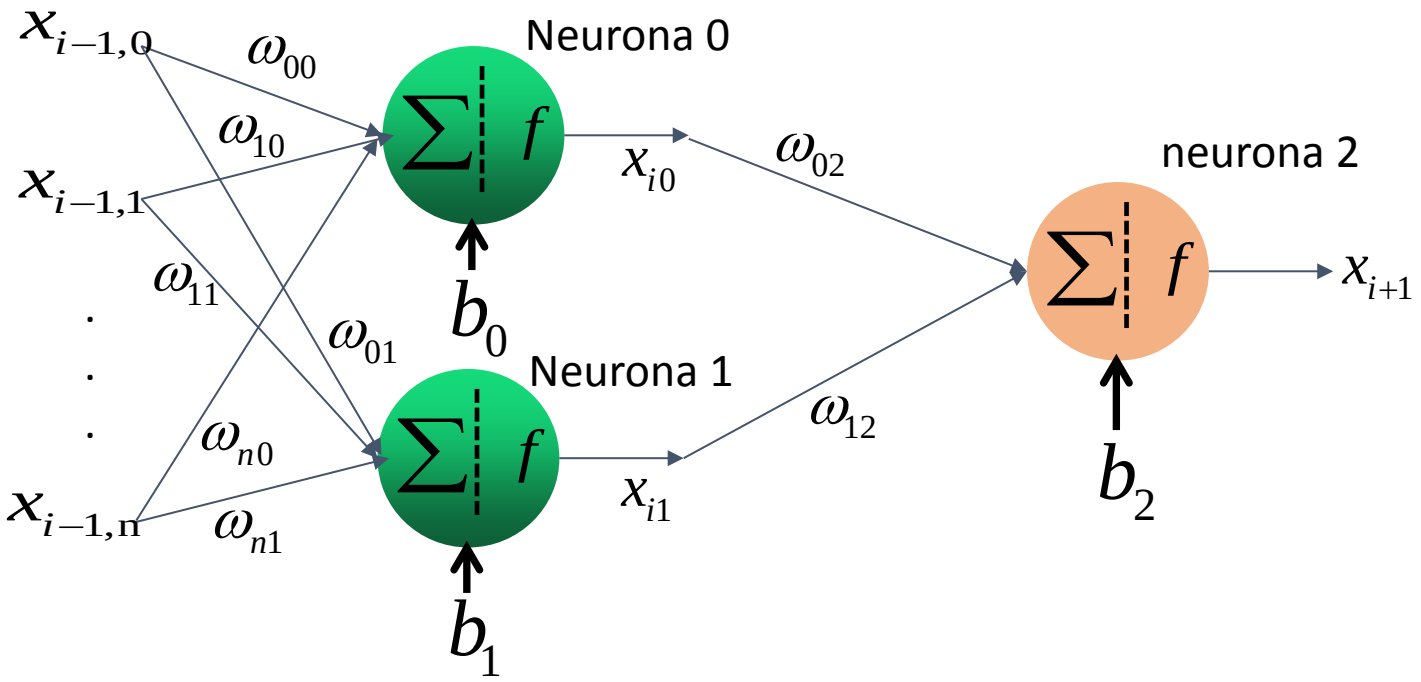


$$0 = \mathbf{w}_i \mathbf{x}_{i-1} + b_i$$

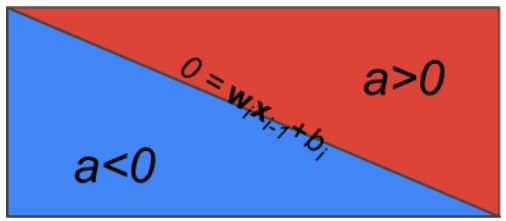


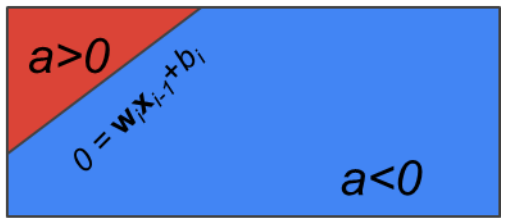
! Sólo separa clases linealmente separables (mediante una recta)

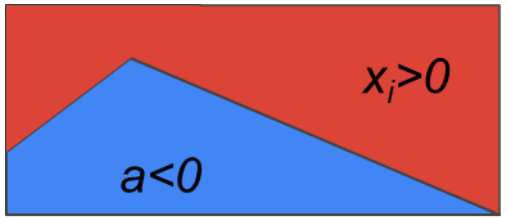
Red neuronal básica



La introducción de una capa de neuronas permite combinar varias rectas (planos) de decisión y así separar clases no linealmente separables







UNIVERSIDAD INTERNACIONAL DE ANDALUCÍA

Aprendizaje y capacidad de generalización

- El objetivo del aprendizaje no es que la red sea capaz de “reproducir” las salidas correspondientes a las entradas con las que ha sido entrenada.
- Se trata de que la red sea capaz de realizar predicciones correctas sobre entradas nuevas, que no han sido presentadas a la red durante el proceso de aprendizaje (que llamaremos **entrenamiento** de la red)
- **Distinguimos así entre**

Memorización → **Capacidad de un modelo para reproducir las salidas correspondientes a las entradas con las que ha sido generado**

Generalización → **Capacidad de un modelo para predecir correctamente la salida correspondiente a una entrada que no se ha utilizado para generar dicho modelo**

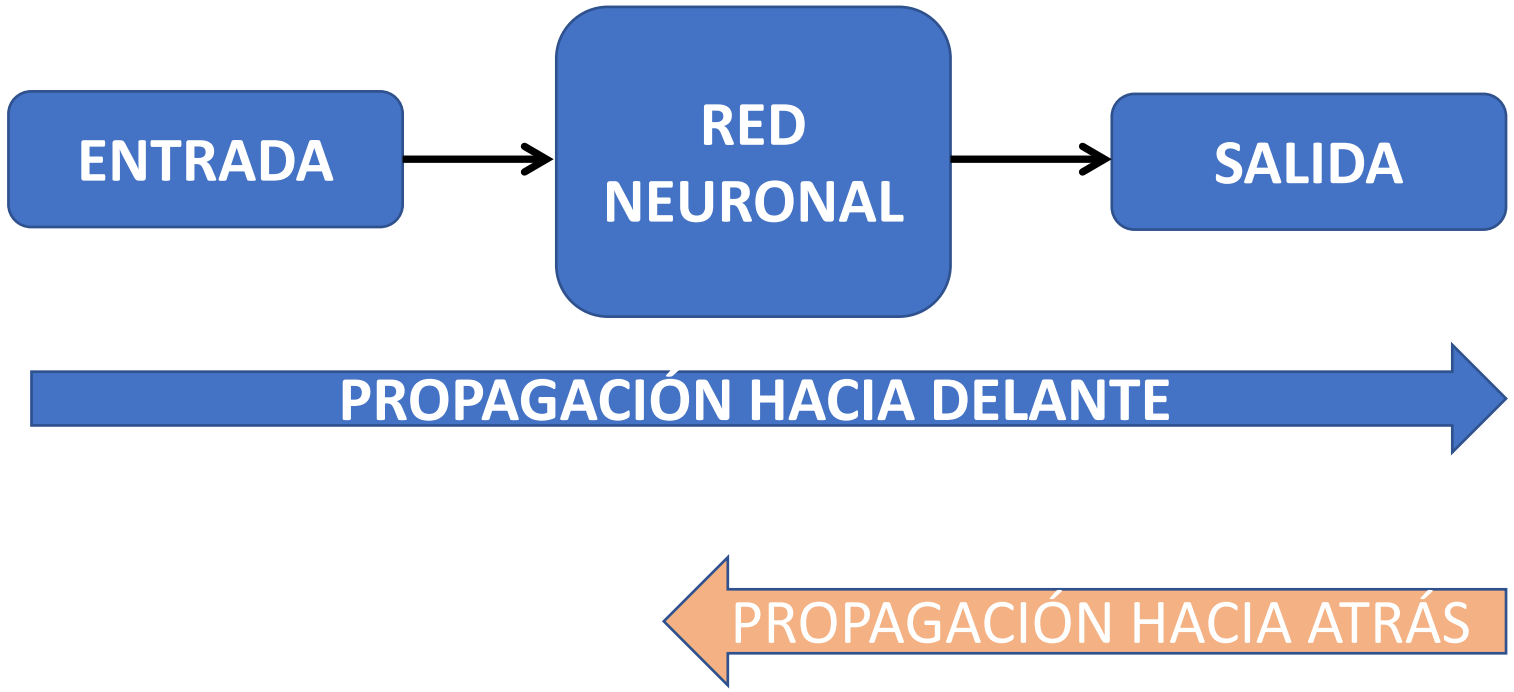
UNIVERSIDAD INTERNACIONAL DE ANDALUCÍA

Aprendizaje en una red neuronal

APRENDIZAJE → **Ajuste de los pesos ω**

Como después veremos, este ajuste de los pesos se realiza mediante un proceso de optimización de una función objetivo

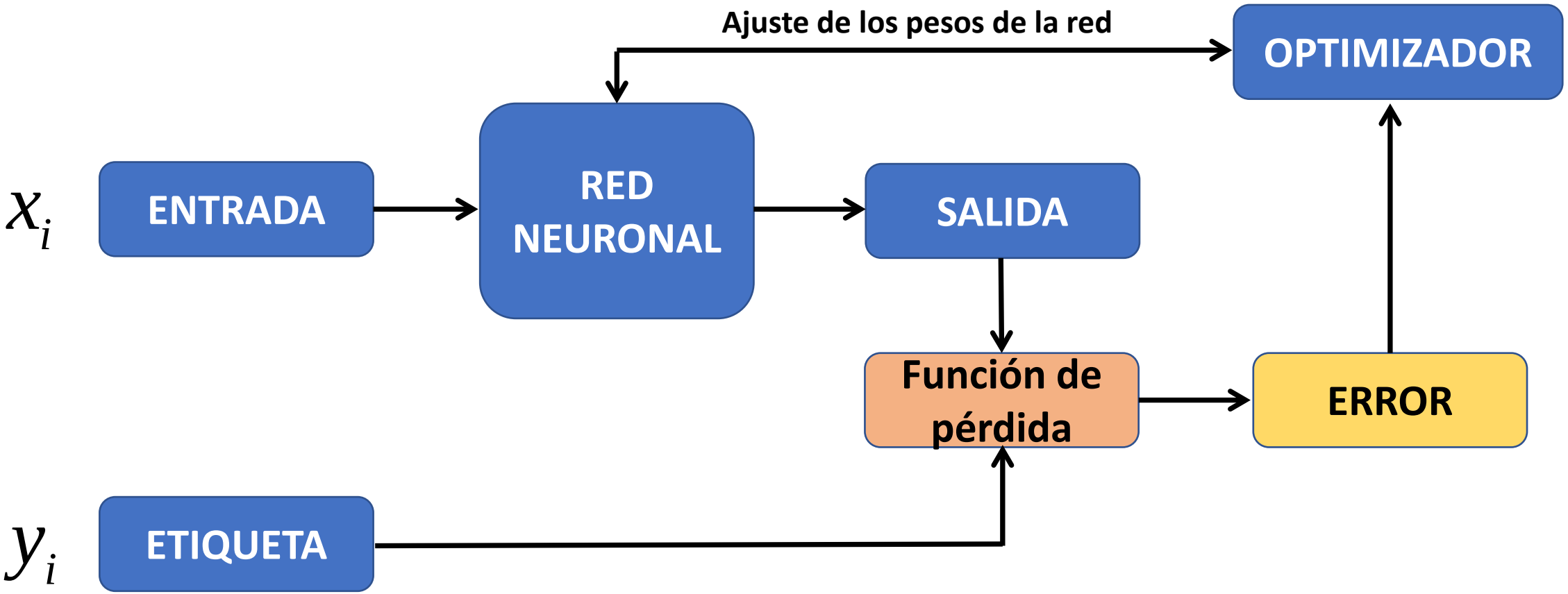
Funcionamiento de una red neuronal. Entrenamiento



Estimación del error

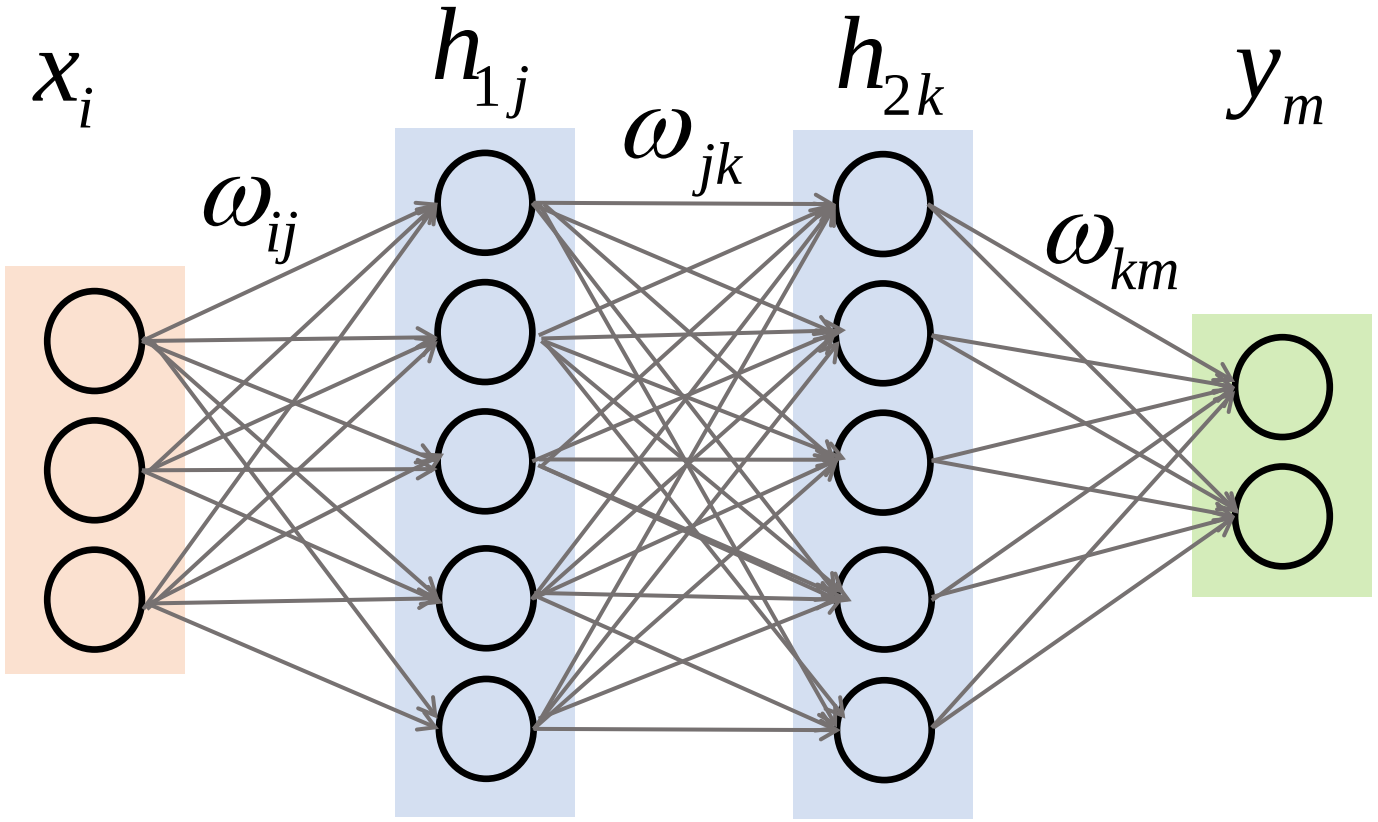
Ajuste de los pesos para minimizar el error

Entrenamiento de una red neuronal. Algoritmo *backpropagation*



UNIVERSIDAD INTERNACIONAL DE ANDALUCÍA

Perceptrón multicapa



ω_{jk} = peso de la neurona j de una capa a la neurona k de la capa siguiente

h_j = salida de las neuronas de la capa oculta j

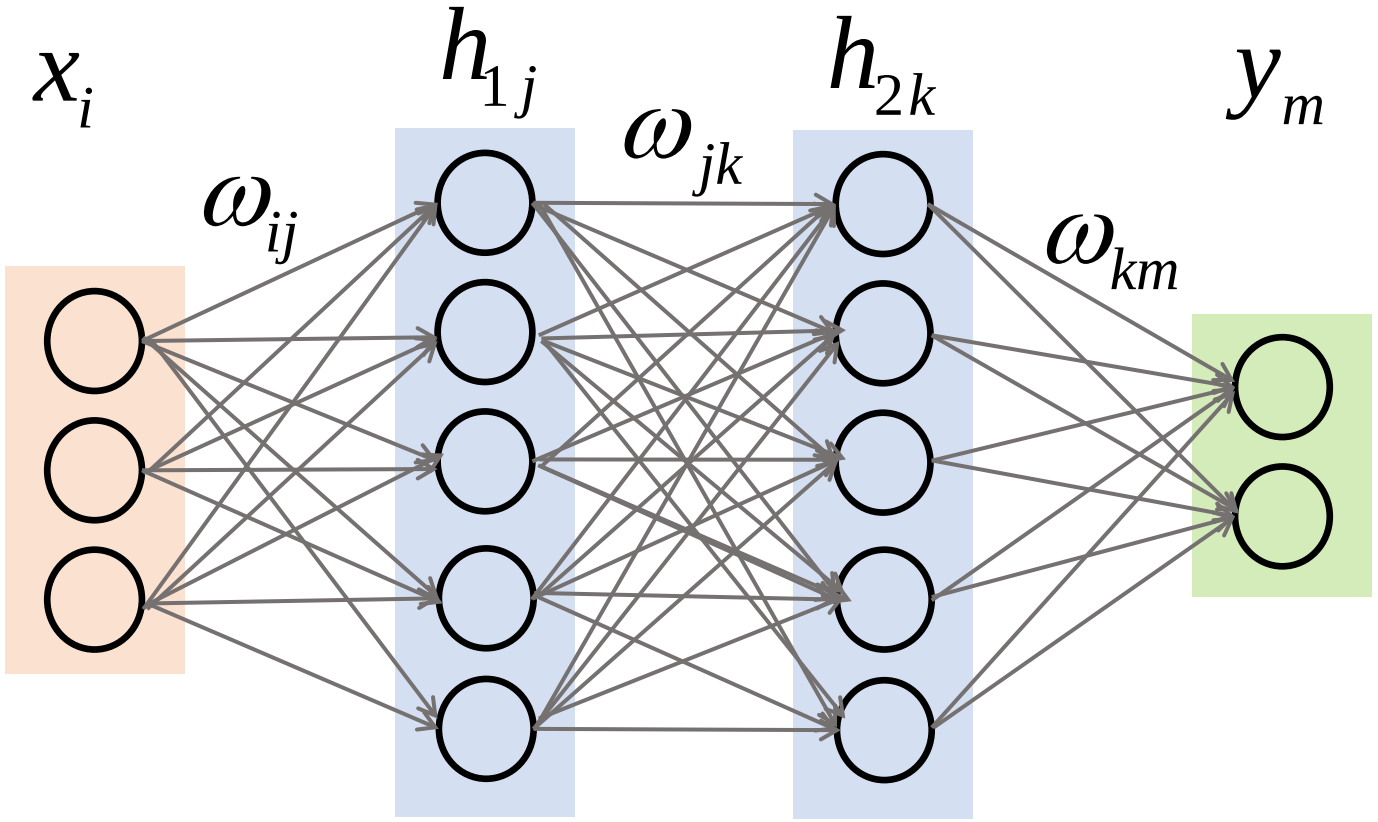
CAPA ENTRADA

CAPAS OCULTAS

CAPA SALIDA

UNIVERSIDAD INTERNACIONAL DE ANDALUCÍA

Perceptrón multicapa



La salida de cada capa se puede calcular:

$$y_m = f\left(\sum_{k=1}^{h} \omega_{jk} h_k\right) \qquad \mathrm{k} = 1, 2, \ldots, \mathrm{m}$$

Función de activación

CAPA ENTRADA

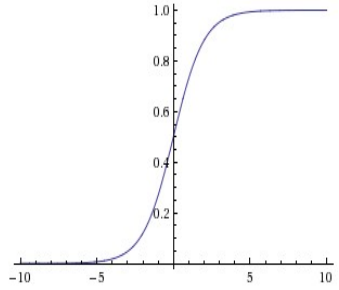
CAPAS OCULTAS

CAPA SALIDA

UNIVERSIDAD INTERNACIONAL DE ANDALUCÍA

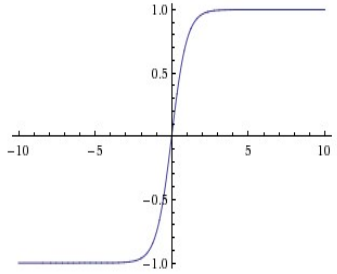
Funciones de Activación

Sigmoide



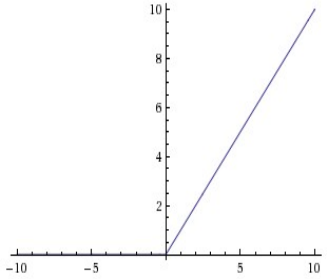
$$\frac{1}{1+e^{-x}}$$

tanh



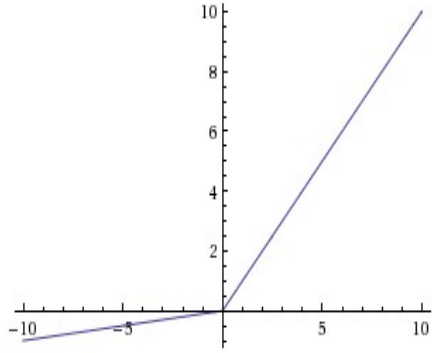
$$tanh(x)$$

ReLU



$$max(0,x)$$

Leaky ReLU



$$max(Cx,x)$$

Algoritmo *backpropagation. Cálculo del error*

- El error obtenido para una red con uno pesos ω

$$E(\omega) = \frac{1}{2}\left\| y - f(x,\omega) \right\|^2$$

Salida "deseada"

Salida que proporciona la red

Algoritmo *backpropagation. Actualización de los pesos*

- El error obtenido para una red con uno pesos ω

$$E(\omega) = \frac{1}{2}\left\| y - f(x,\omega) \right\|^2$$

- Con este error, los pesos se actualizan:

$$nuevo\ \omega = \alpha_0 * (anterior\ \omega) + \eta(-grad(E(\omega)))$$

momento

Tasa de aprendizaje

Gradiente del error con respecto a w

Algoritmo *backpropagation. Actualización de los pesos*

- Con la notación anterior, aplicando la regla de la cadena, se puede calcular la derivada del error

$$\frac{\partial E}{\partial \omega_{jk}} = -(d_k - y_k) f' h_j$$

- Si f es la identidad: f'=1
- Si f es la función sigmóide $f(\mathrm{x}) = \frac{1}{1+e^{-x}}$, $f' = f(1-f) = y_k(1-y_k)$

UNIVERSIDAD INTERNACIONAL DE ANDALUCÍA

Algoritmo *backpropagation. Resumen*

1. Con la entrada $\vec{x}$, calculamos $h_j = f\left(\sum_{i=1}^{n} \omega_{ij} x_i\right) \quad j = 1,\ldots,h1$

2. Propagamos hasta la salida $y_k = f\left(\sum_{i=1}^{h} \omega_{jk} h_j\right) \quad k = 1,\ldots,m$

3. Calculamos el error $\delta_k = (d_k - y_k) y_k (1 - y_k)$

FEED FORWARD

4. El incremento de los pesos (capa jk) $\Delta\omega_{jk} = \eta \delta_k h_j$

5. Actualizamos los pesos (capa jk) $\omega_{jk} = \omega_{jk} + \Delta\omega_{jk}$

6. El incremento de los pesos (ij) $\Delta\omega_{ij} = \eta h_j (1 - h_j) x_i \sum_{j=1}^{m} \delta_j \omega_{ij}$

7. Actualizamos los pesos (ij) $\omega_{ij} = \omega_{ij} + \Delta\omega_{ij}$

BACKPROP

UNIVERSIDAD INTERNACIONAL DE ANDALUCÍA

Algoritmo *backpropagation. Resumen*

1. Con la entrada $\vec{x}$, calculamos $h_j = f\left(\sum_{i=1}^{n} \omega_{ij} x_i\right) \quad j = 1,\ldots,h1$

2. Propagamos hasta la salida $y_k = f\left(\sum_{i=1}^{h} \omega_{jk} h_j\right) \quad k = 1,\ldots,m$

3. Calculamos el error $\delta_k = (d_k - y_k) y_k (1 - y_k)$

4. El incremento de los pesos (capa jk) $\Delta\omega_{jk} = \eta \delta_k h_j$

5. Actualizamos los pesos (capa jk) $\omega_{jk} = \omega_{jk} + \Delta\omega_{jk}$

6. El incremento de los pesos (ij) $\Delta\omega_{ij} = \eta h_j (1 - h_j) x_i \sum_{j=1}^{m} \delta_j \omega_{ij}$

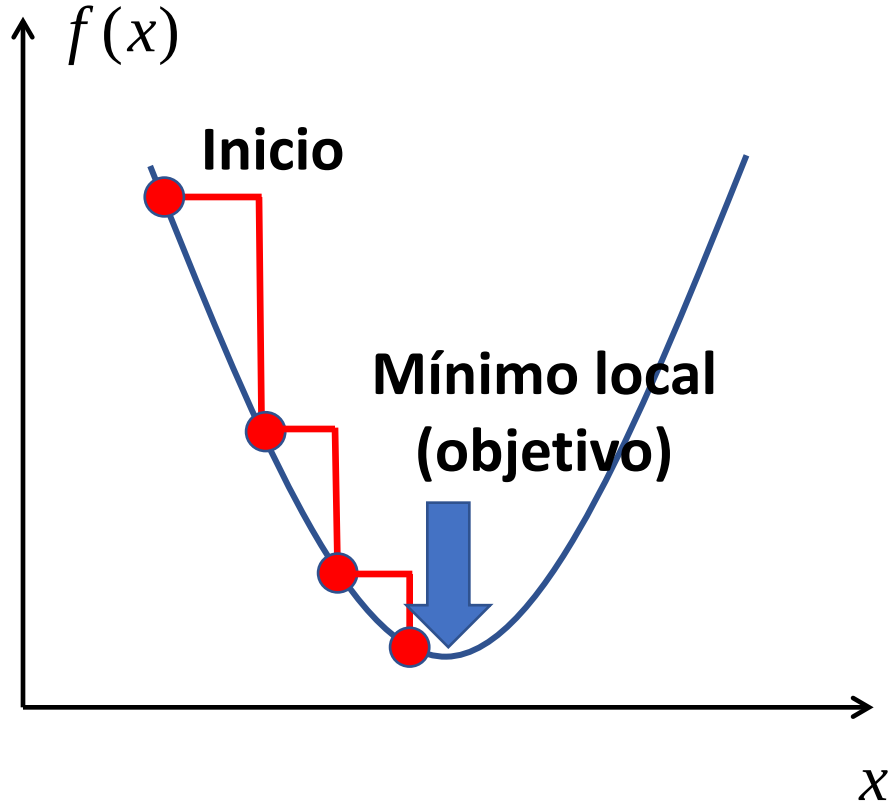
7. Actualizamos los pesos (ij) $\omega_{ij} = \omega_{ij} + \Delta\omega_{ij}$

PYTORCH HACE TODO ESTO CON SÓLO UNA ORDEN!

UNIVERSIDAD INTERNACIONAL DE ANDALUCÍA

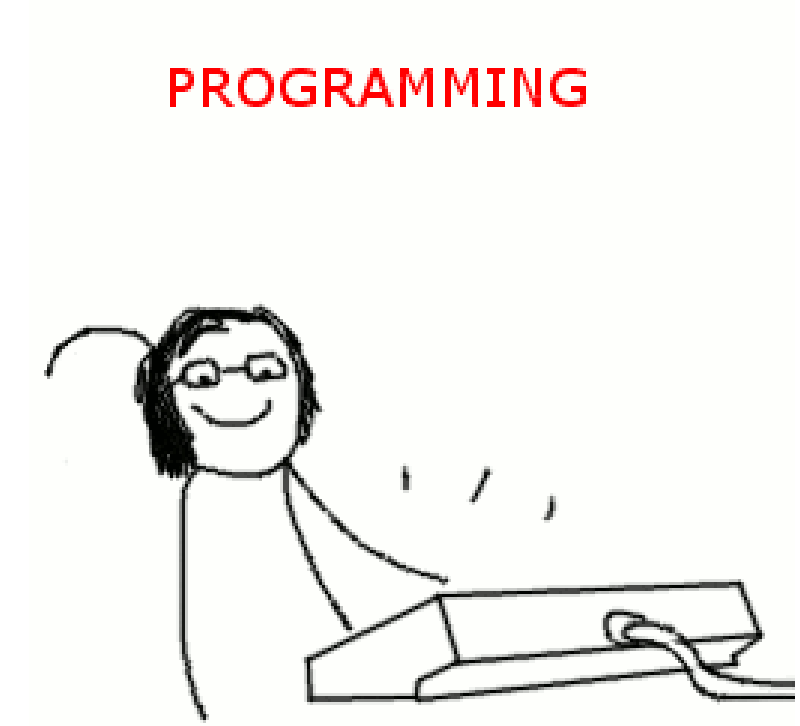
Algoritmo *backpropagation. Implementación*
Algoritmo de Descenso de Gradiente Estocástico (SGD)

- El algoritmo de gradiente descendente es un método de optimización para encontrar el mínimo local de una función diferenciable.
- El objetivo es determinar los parámetros que minimizan una función de coste
- Requiere que la función a optimizar sea convexa.
- Es el método de optimización más usado en la actualidad en Deep Learning.



UNIVERSIDAD
INTERNACIONAL
DE ANDALUCÍA

Práctica: Implementación de redes neuronales con PyTorch



UNIVERSIDAD
INTERNACIONAL
DE ANDALUCÍA

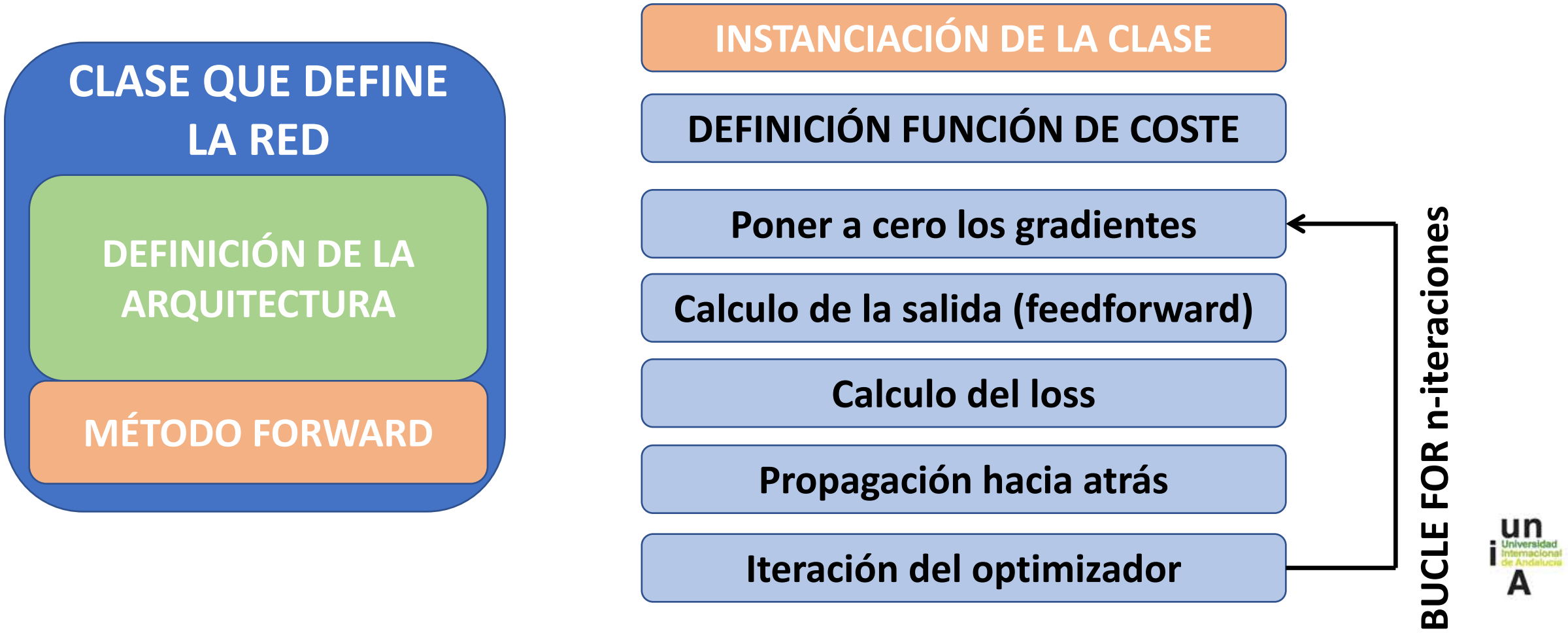
Estructura básica de una red neuronal en PyTorch

Lo primero que tenemos que hacer es importar pytorch y algunos módulos de la librería:

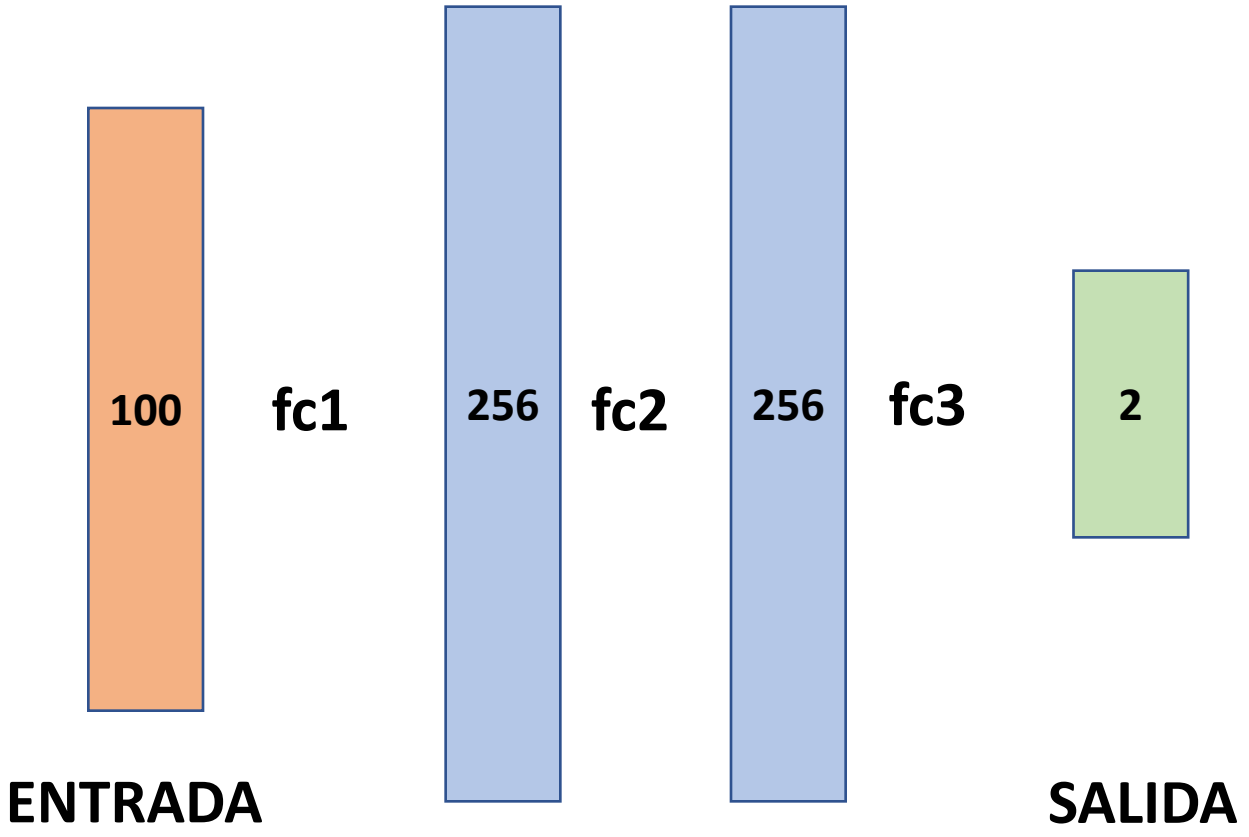
- **torch.nn**: La librería de redes neuronales que utilizaremos para crear nuestro modelo.
- **torch.autograd:** En concreto el módulo Variable de esta librería que se encarga de manejar las operaciones con los tensores y sus gradientes
- **torchvision.datasets**: El módulo que ayudará a cargar el conjunto de datos que vamos a utilizar y explicaremos más adelante.
- **torchvision.transforms**: Este módulo contiene una serie de funciones que nos ayudarán modificando el dataset.
- **torch.optim**: De aquí usaremos el optimizador para entrenar la red neuronal y modificar sus pesos.

UNIVERSIDAD INTERNACIONAL DE ANDALUCÍA

Estructura básica de una red neuronal en PyTorch

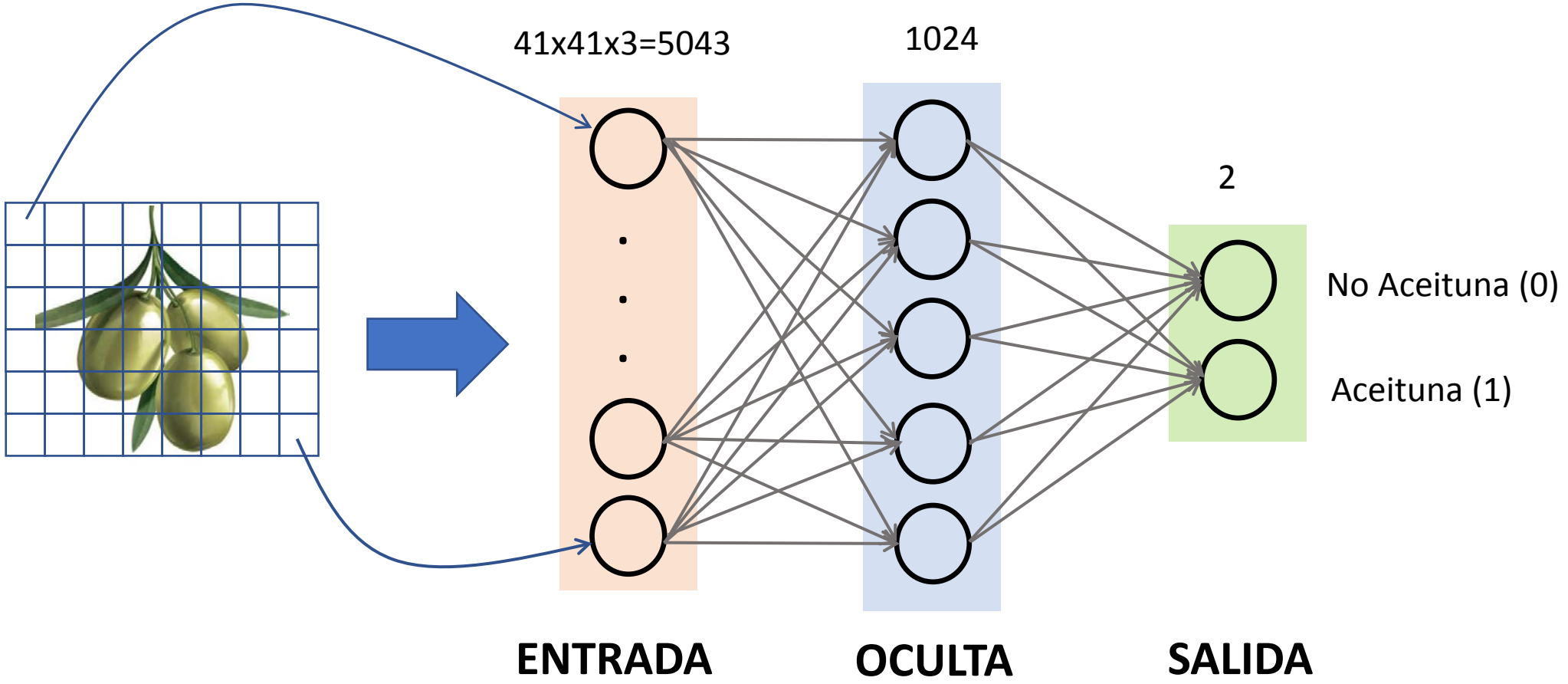


Diseño de un perceptrón multicapa (MLP)



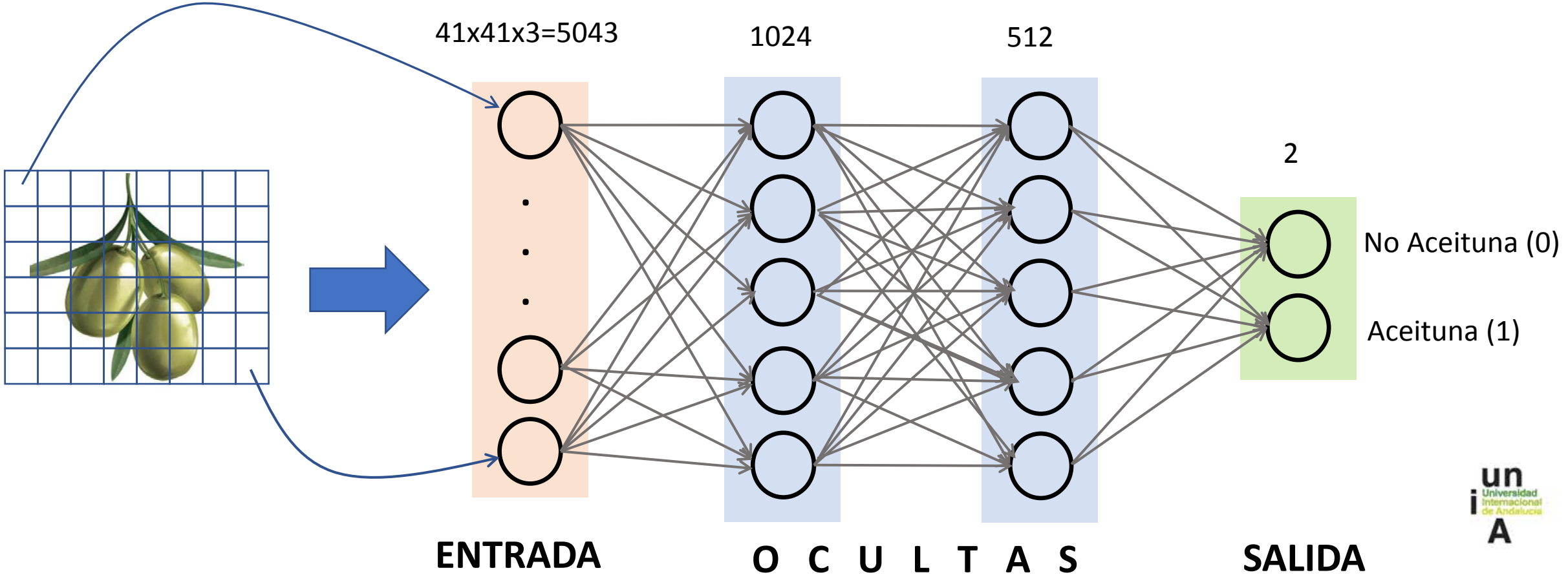
UNIVERSIDAD INTERNACIONAL DE ANDALUCÍA

Diseño de un perceptrón multicapa (MLP), 1 capa oculta
Aplicación a la clasificación de imágenes



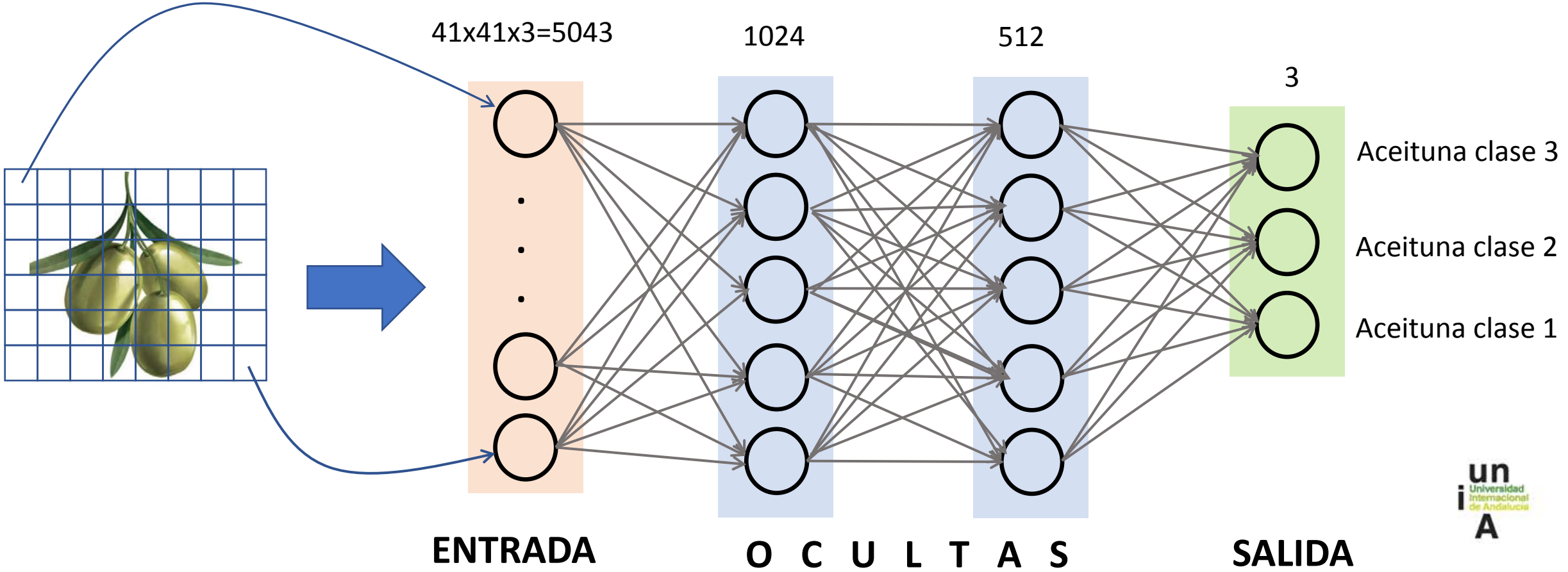
UNIVERSIDAD INTERNACIONAL DE ANDALUCÍA

Diseño de un perceptrón multicapa (MLP), 2 capas ocultas
Aplicación a la clasificación de imágenes



UNIVERSIDAD INTERNACIONAL DE ANDALUCÍA

Diseño de un perceptrón multicapa (MLP), 2 capas ocultas
Aplicación a la clasificación de imágenes. Clasificación multiclase



UNIVERSIDAD INTERNACIONAL DE ANDALUCÍA

¿ Para qué sirven las capas ocultas ?

Las capas ocultas proporcionan la capacidad discriminante de una red neuronal

- **Al incrementar el número de neuronas** de una capa, añadimos parámetros a esa capa, lo que permitirá un mejor ajuste a los datos de entrenamiento. Sin embargo, **reducimos el poder de generalización de la red → overfitting**
- **Al añadir capas**, incrementamos la complejidad dimensional que la red es capaz de aprender: **modificamos la forma del hiperplano** de separación.

Regla para calcular el óptimo número de capas y neuronas por capa

NO HAY → Ensayo y error

Carga de datos (aprendizaje supervisado)

- Para entrenar la red, y para comprobar posteriormente sus prestaciones, necesitamos disponer de los conjuntos de datos y sus correspondientes etiquetas (pares (x_i,y_i))

- En cada iteración, tendremos que cargar un array con los datos que se van a utilizar, por ejemplo, para entrenar la red o para la validación

- Es decir, tendremos un tensor X con los datos y un tensor Y con las etiquetas

- De esta forma, haremos: salida=modelo(X) → devuelve una salida por cada fila en X
- Y calculamos el error: error=loss(salida,Y) → devuelve un error para cada salida

UNIVERSIDAD INTERNACIONAL DE ANDALUCÍA

Carga de datos

¿ Qué ocurre si el tensor X es muy grande y no cabe en memoria (problema muy frecuente en Deep Learning!)?

- En este caso, tendremos que ir cargando los datos poco a poco → **entrenamiento *batch***
- Un *batch* es un subconjunto de datos que SÍ cabe en memoria
- La idea es calcular la salida y estimar los gradientes del error para cada *batch*

El tamaño del *batch* puede afectar a la estabilidad numérica del algoritmo de optimización (gradiente descendente).
La capacidad de generalización depende del tamaño del *batch*!. Es, por tanto, uno de los hiperparámetros más importantes a optimizar!

Carga de datos
Entrenamiento con algoritmos batch

- **En los extremos:**
  - Utilizando un tamaño de batch igual al del dataset (dataset completo), garantiza la convergencia al mínimo global de la función de coste. No obstante, la convergencia es muy lenta y puede requerir muchas iteraciones (recordemos que podemos tener redes con millones de pesos)
  - Tamaño de batch pequeños, pueden proporcionar buenos resultados (la red empieza a aprender sin heberle presentado todos los datos). No obstante, la convergencia al mínimo global no está garantizada
  - Utilizando un tamaño de batch = 1, el proceso de aprendizaje será muy lento

UNIVERSIDAD INTERNACIONAL DE ANDALUCÍA

Mejora de la capacidad de generalización. Regularización

La regularización consiste en aplicar una penalización a la función de coste durante el proceso de optimización para evitar el sobreajuste a los datos de entrenamiento

Si definimos la función de coste como $loss = error(\mathrm{y}, \hat{\mathrm{y}})$

Podemos añadir un término que incremente el loss:

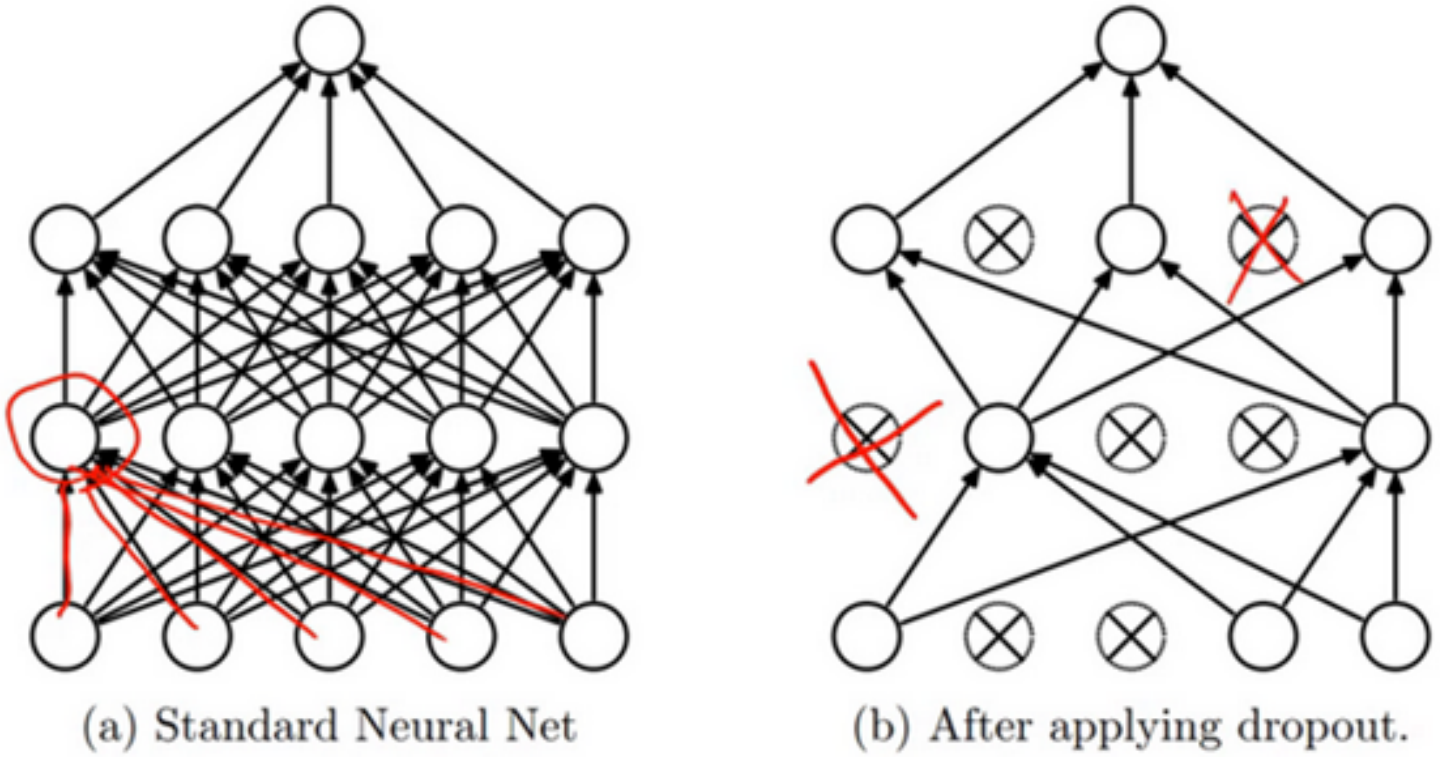
$$loss = error(\mathrm{y}, \hat{\mathrm{y}}) + \lambda\Psi \begin{cases} \lambda \rightarrow \text{ peso de la regularización} \\ \psi \rightarrow \text{ término de regularización} \end{cases}$$

UNIVERSIDAD INTERNACIONAL DE ANDALUCÍA

Mejora de la capacidad de generalización. Regularización

Tipos de regularización más comunes

- **Dropout**



(a) Standard Neural Net

(b) After applying dropout.

- Se desactivan aleatoriamente un porcentaje predeterminado de neuronas durante el entrenamiento
- Evita que las neuronas memoricen parte de la entrada

UNIVERSIDAD INTERNACIONAL DE ANDALUCÍA

Mejora de la capacidad de generalización. Regularización
Tipos de regularización más comunes

- **Early Stopping**

- Conseguir errores muy bajos durante el entrenamiento normalmente requiere de 1) un número alto de parámetros 2) realizar un número alto de iteraciones
- Ambas cosas hacen que la red sobreajuste los datos de entrenamiento

Early Stopping consiste en parar el proceso de entrenamiento cuando se haya alcanzado un determinado error, si dejar que se realicen más iteraciones que pueden llevar al sobreajuste.