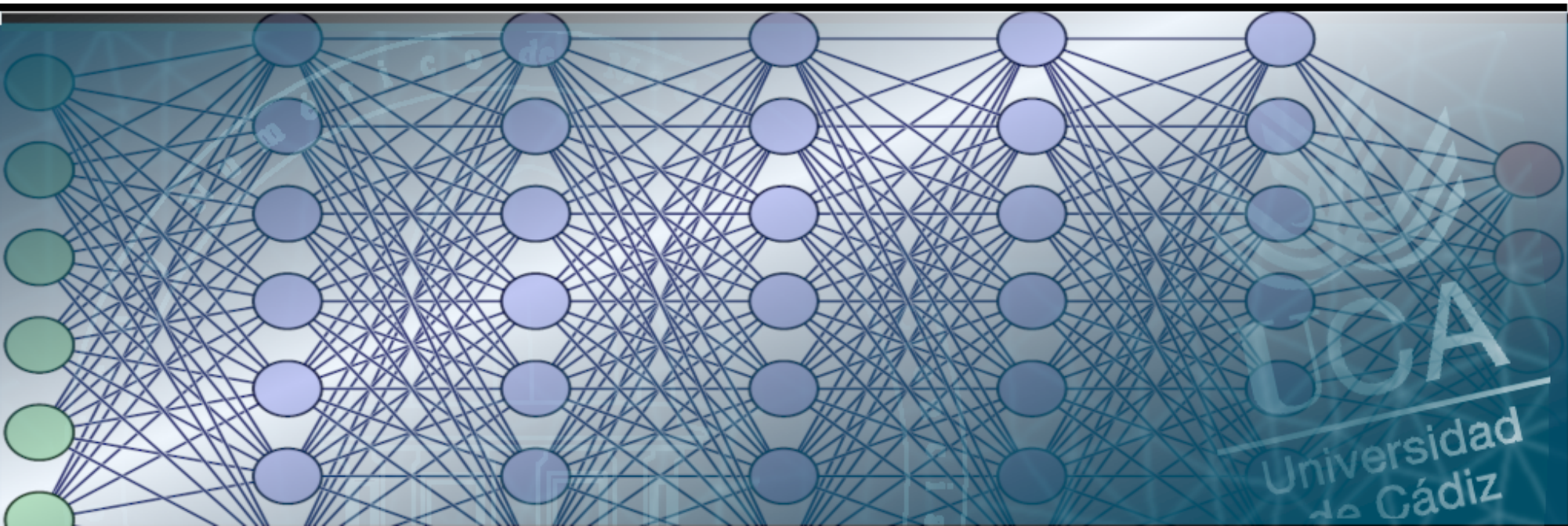
Neural Networks

Álex Pérez Fernández, Rafa Rodríguez Galván

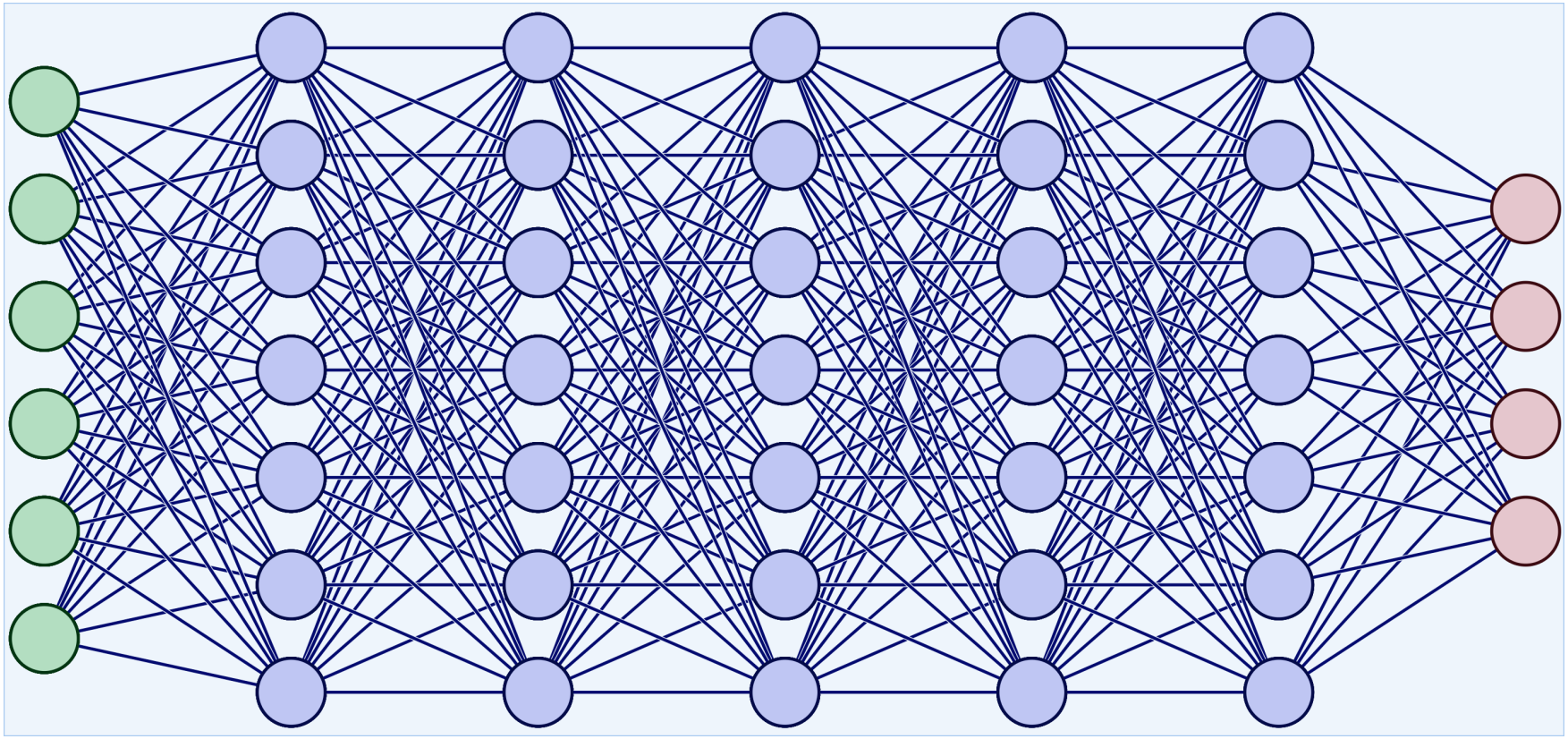
March 11, 2024



Section 1

Neural Networks

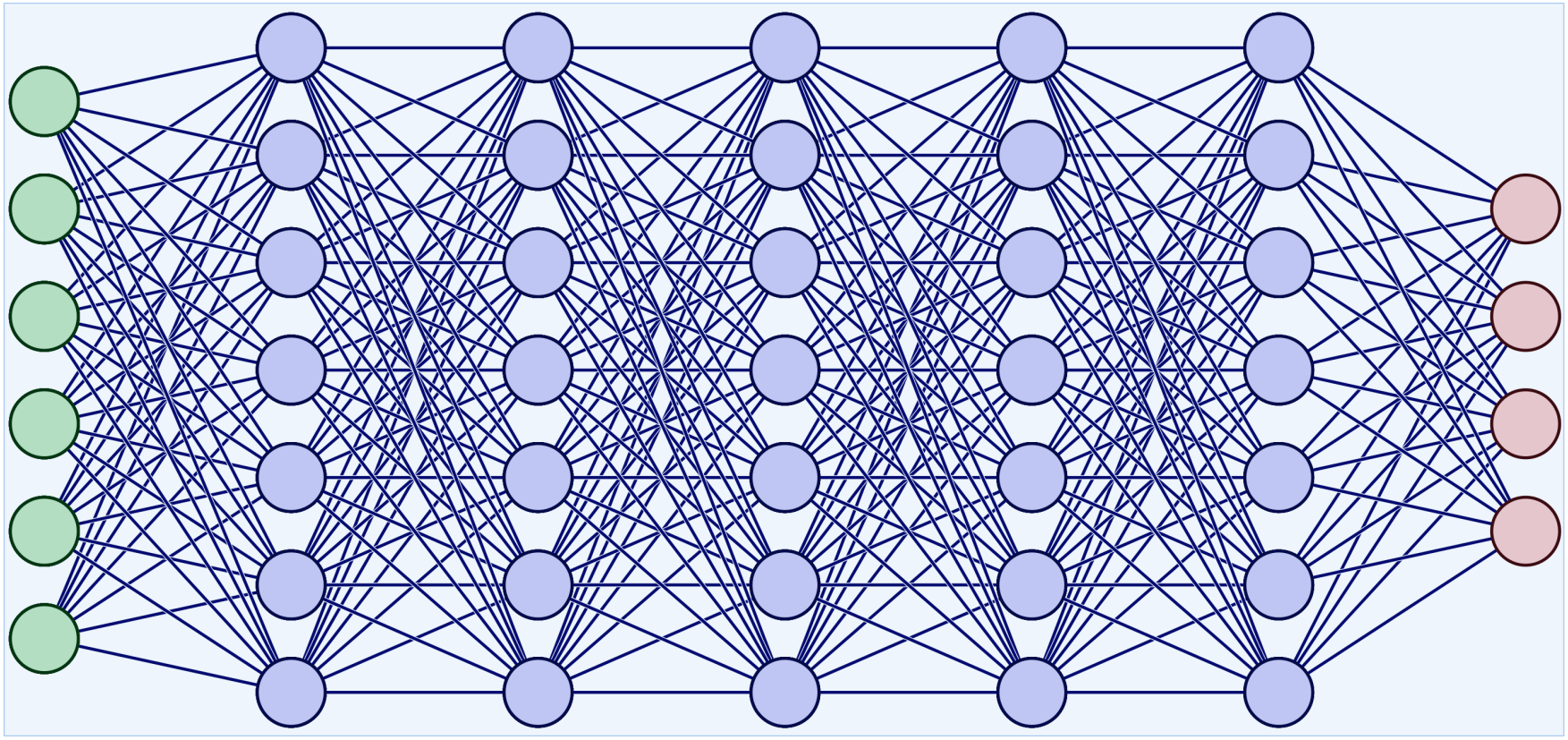
Neural Networks...



...are mathematical artifacts:

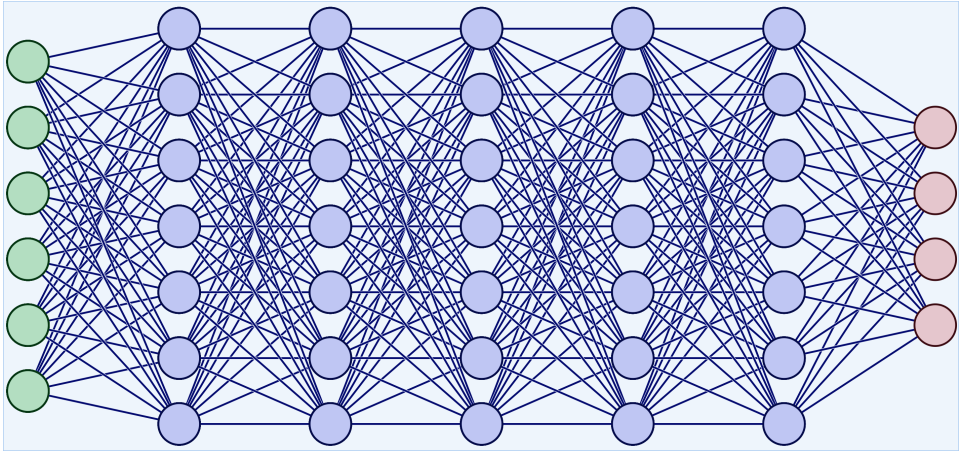
$$x \mapsto f_1(x) \mapsto f_2 \circ f_1(x) \mapsto \cdots \mapsto f_L \circ \cdots \circ f_2 \circ f_1(x) = y$$

Neural Networks...



...are mathematical artifacts:

$$x \mapsto f_1(x) \mapsto f_2 \circ f_1(x) \mapsto \cdots \mapsto f_L \circ \cdots \circ f_2 \circ f_1(x) = y$$



Definición:
una **Red Neuronal** (*RN* o *NN*) es una función $f_{NN} : \mathbb{R}^n \to \mathbb{R}^m$ del tipo:

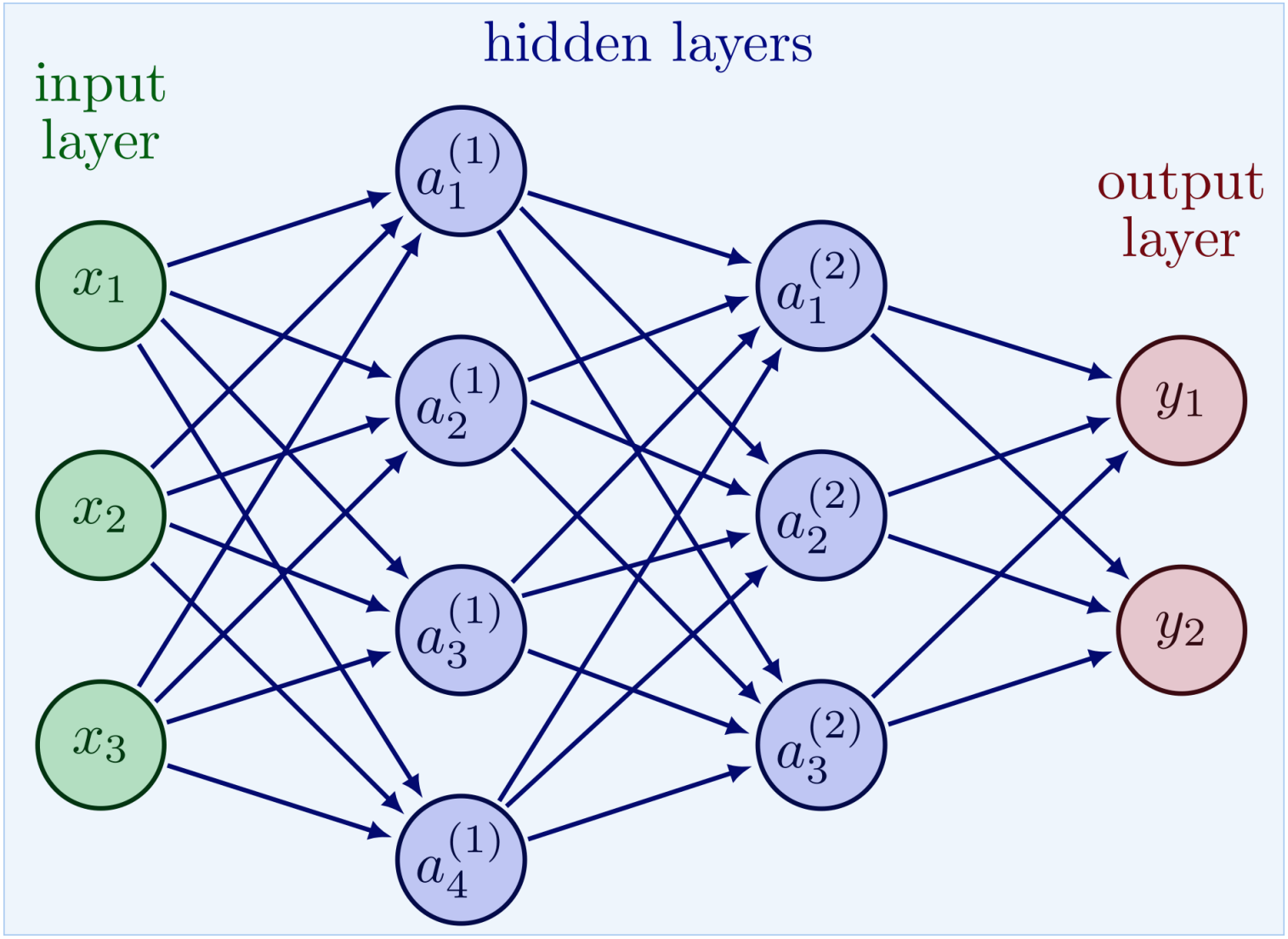
$$y = f_{NN}(x) = f_L \circ \cdots \circ f_2 \circ f_1(x).$$

Donde...

- Cada función f_i se llama una **capa** (entrada $\to$ oculta $\to$ salida)
- Cada capa f_i está compuesta por un nº variable de **neuronas**
- Cada neurona depende un conjunto de **parámetros**, que determinarán a la RN

$\star$ La RN de la figura se dice de tipo «*feed forward*» o prealimentada

Un ejemplo



$f_{NN} : \mathbb{R}^3 \to \mathbb{R}^2$con 2 capas ocultas de 4 y 3 neuronas

Neurona o perceptrón simple

Cada neurona j de una capa oculta f_i (o de salida y_i) es una función[1]:

$$\mathrm{x} \in \mathbb{R}^{N_i} \to a_j^{(i)}(\mathrm{x}) \in \mathbb{R},$$

composición de

- una función afín con parámetros $\mathrm{w} = (w_1, \ldots, w_{N_i})$ y b
- una función no lineal σ, llamada «función de activación»

$$a_j^{(i)}(\mathrm{x}) = \sigma(w_1 x_1 + w_2 x_2 + \cdots + w_{N_i} x_{N_i} + b) =$$
$$= \sigma\left(\sum_{k=1}^{N_i} w_k x_k + b\right) = \sigma(\mathrm{w} \cdot \mathrm{x} + b)$$

[1] Donde N_i es el número de neuronas de la capa $i - 1$

Con más propiedad...

Para aligerar la notación se omitieron los índices correspondientes a la capa, i, y a la neurona, j. Debería ser:

$$a_j^{(i)}(\mathrm{x}) = \sigma\Big(\sum_{k=1}^{N_i} w_{j,k}^{(i)} x_k + b_j^{(i)}\Big).$$

Así, si $\mathrm{W}^{(i)}$ denota a la matriz de valores $w_{j,k}^{(i)}$, y $\mathrm{b}^{(i)}$ es el vector $(b_j^{(i)})$, podemos escribir a tod la capa i como:

$$\mathrm{f}_i(\mathrm{x}) = \boldsymbol{\sigma}_i(\mathrm{W}^{(i)}\mathrm{x} + \mathrm{b}^{(i)})$$

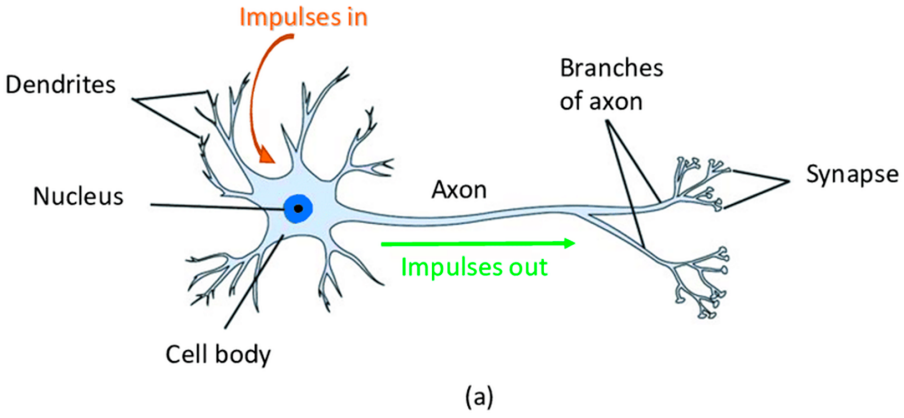
La RN está determinada por los parámetros $\mathrm{W}^{(i)}$, los desplazamientos $\mathrm{b}^{(i)}$ y las funciones de activación $\boldsymbol{\sigma}_i$

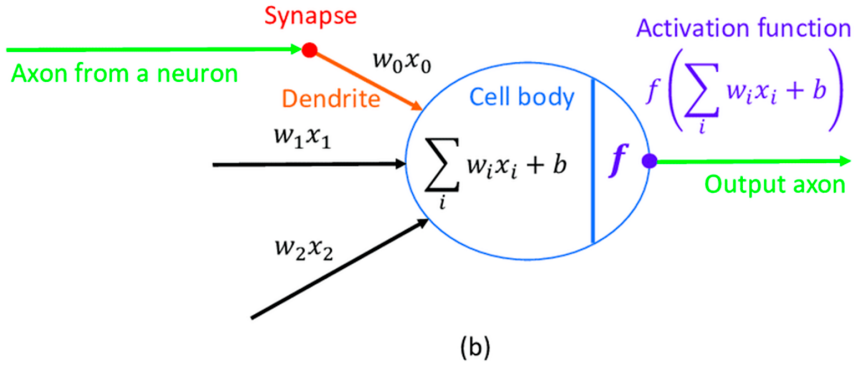
Section 2

Partes básicas de una red neuronal

Funciones de activación

Es una abstración que representa la **tasa potencial de acción.**





2

Dada una entrada se procesa **(parte lineal)** y se valora si la neurona dispara o no perceptron. Para comportamientos más complejos podemos devolver una probabilidad si es baja menos probable es que dispare y viceversa, de ahí proviene la función de activación sigmoidal.

[2] Ranking to Learn and Learning to Rank: On the Role of Ranking in Pattern Recognition Applications

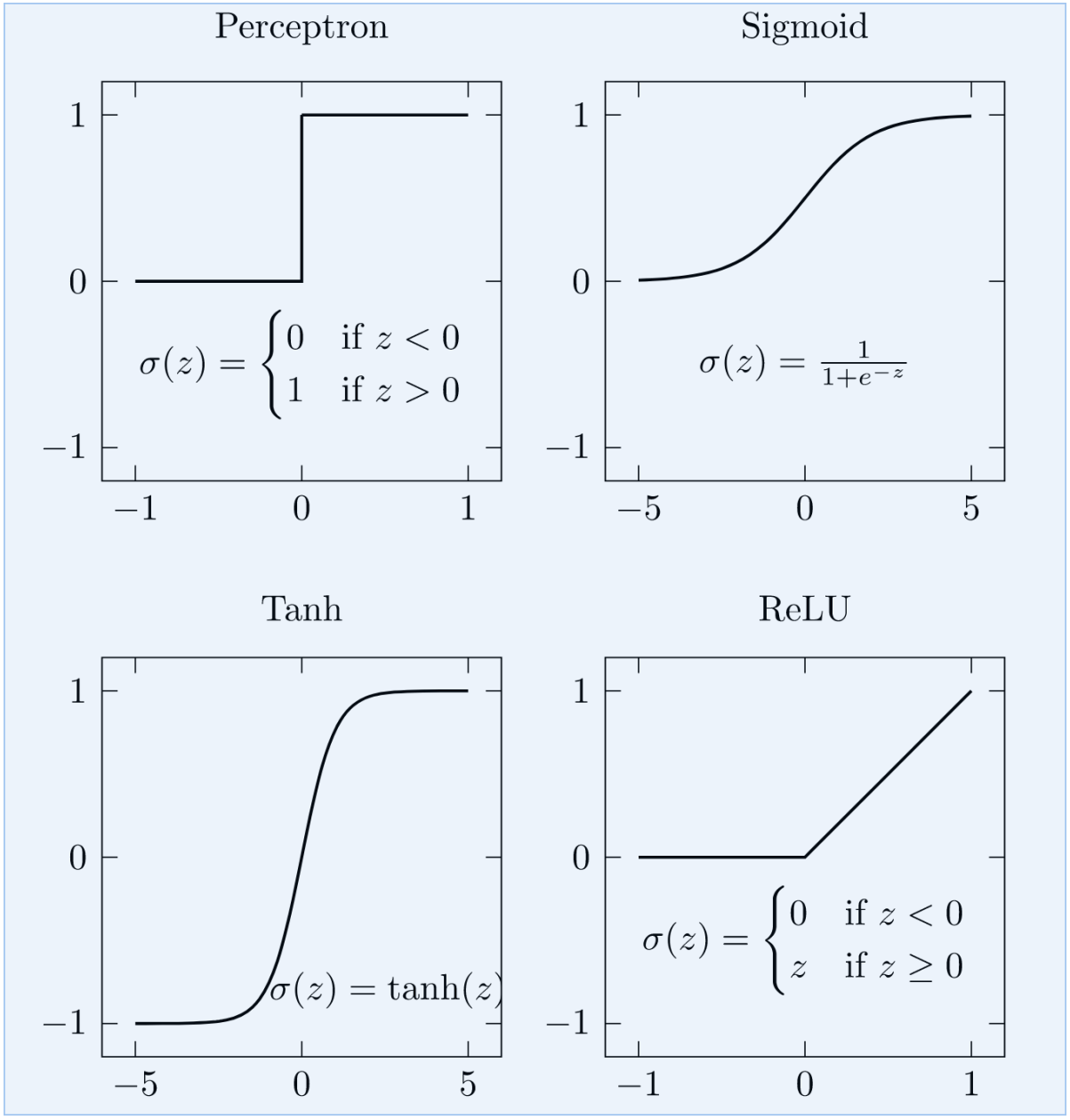
¿Por qué necesitamos funciones de activación?

Supongamos que no tenemos funciones de activación, luego la red neural es composición de aplicaciones lineales, esto implica que **la red es una función lineal.**

$$f_{NN}(x) = w_1 x_1 + w_2 x_2 + \cdots + w_{N_i} x_{N_i} + b$$

Por lo tanto, no importa cuantas capas tenga el modelo, puesto que estamos realizando una transformación lineal en las entradas. Esto es un problema, puesto que no podemos modelar problemas no lineales.

Representación de funciones de activación



Funciones de perdida

Una función de perdia compara la salida de la red neuronal con la salida esperada, y nos dice que tan **bien o mal lo esta haciendo la red neuronal.**

Cuando entrenamos, nuestro objetivo es minimazar la perdida entre la salida esperada y la salida de la red neuronal.

Ejemplos:

- Problema de regresión

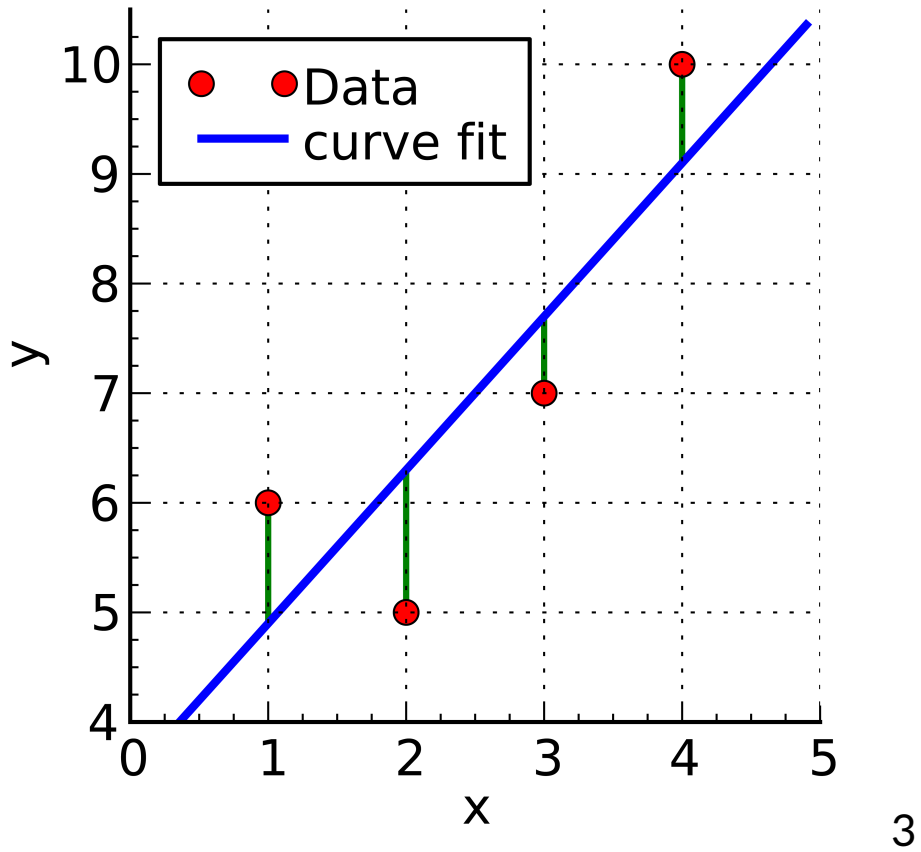
$$y_{pred} = \begin{pmatrix} 250 & 300 \\ 300 & 400 \end{pmatrix} \quad y_{real} = \begin{pmatrix} 100 & 150 \\ 400 & 200 \end{pmatrix}$$

- Problema de clasificación

$$y_{pred} = \begin{bmatrix} 0.12 \\ 0.48 \\ 0.4 \end{bmatrix} \quad y_{real} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

Podemos pensarlo como un residuo en estadística, que se encarga de medir la distancia entre los **valores actuales de y y los de la linea de regresión** (valores predecidos).

El objetivo es minimizar la distancia neta.



[3]

[3] Wikipedia - Linear regression

Ejemplos de funciones de perdida

Principalmente se agrupan en:

- **Perdida de regresión:** Dado un valor de entrada, el modelo predice el valor de salida.

  Ejemplos: **Mean Squared Error, Mean Absolute Error...**

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - y_{pred}^{(i)})^2$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y^{(i)} - y_{pred}^{(i)}|$$

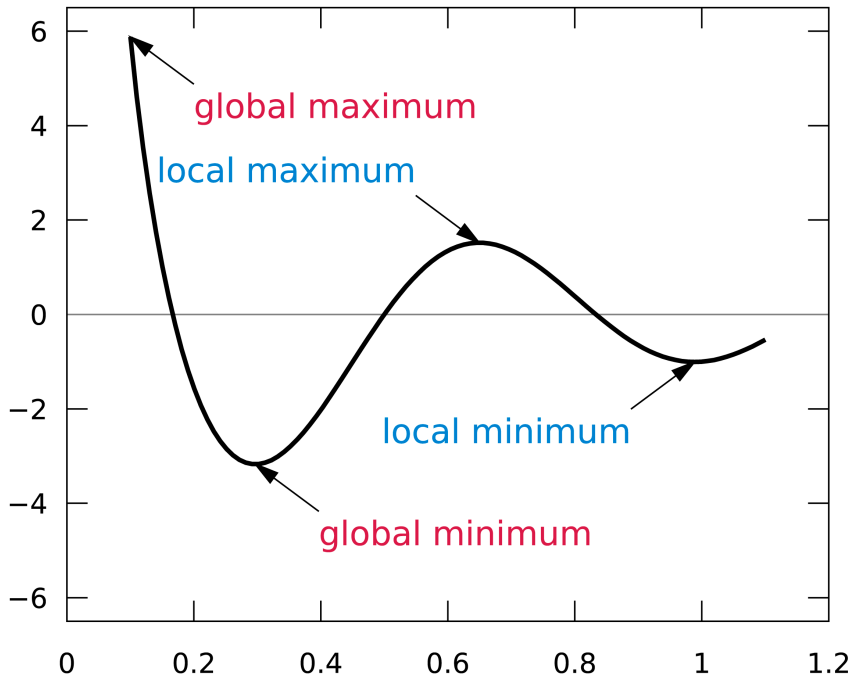
- **Perdida de clasificación:** Dado un valor de entrada, el modelo devuelve un vector de probabilidades de que la entrada pertencezca a una serie de categorias preestablecidas.

  Ejemplos: **Binary Cross Entropy, Categorical Cross Entropy...**

Optimizadores

Son algoritmos que se usan para **minimizar la función de perdida** o maximizar la función de ganancia. Ejemplos: Adam, RMSprop, SGD...

Cada vez que predecimos un valor mediante la red y comparamos con el valor real, **el optimizador ajusta los pesos de la red para minimizar la perdida.**



Queremos obtener un mínimo global en la función de perdida

Gradient Descent

El algoritmo de descenso de gradiente es un **algoritmo de optimización que se utiliza para minimizar una función objetivo.**
Para implementarlo:

$$a_n + 1 = a_n - \delta\lambda F(a_n)$$

La idea es que el gradiente muestra la dirección de descenso entonces al movernos en esa dirección vamos a llegar a un mínimo, el problema es que puede ser un mínimo o un máximo.

¿Cómo se calculan las derivadas?

En ausencia de funciones de activación, la red neural es una función lineal, por lo que el gradiente de la función de perdida con respecto a los pesos de la red es una matriz de valores constantes.
En presencia de funciones de activación, la expresión como composición de funciones no es tan sencilla y se utilizan técnicas numericos llamadas Backpropagation.

Hiperparámetros

Un hiperparámetro es un valor constante que se establece previo al entrenamiento. Ejemplos: Tasa de aprendizaje, número de capas, número de neuronas, funciones de activación, funciones de perdida, optimizadores.
Si un modelo no se comporta bien, entonces tenemos que modificar los hiperparámetros, ya sea para conseguir que el modulo aprenda más lento (tasa de aprendizaje), hacerlo más complejo (añadir más capas)…

Tasa de aprendizaje

La tasa de aprendizaje define el tamaño de los pasos correctivas para que el modelo ajuste los errores en cada observación. Una tasa de aprendizaje muy alta provoca que el tiempo de entrenamiento sea menor, pero puede que el modelo no sea tan precioso. Por otro lado, una tasa de aprendizaje menor provoca un tiempo mayor de procesamiento, pero tiene el potencial de tener más precisión.

Batch Size

El tamaño del lote es el número de ejemplos de entrenamiento que se proporcionan a la red antes de que el optimizador actualice los pesos. Un buen tamaño de lote es generalmente 32. Pueden probarse: 32, 64, 128, 256...

Numeros de epochs

Es el número de veces que se entrena la red con el conjunto de datos de entrenamiento.

Principalmente hay dos frameworks que se utilizan: TensorFlow es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, y desarrollado por Google para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos. PyTorch12 es una biblioteca de aprendizaje automático3 de código abierto basada en la biblioteca de Torch, utilizado para aplicaciones como visión artificial y procesamiento de lenguajes naturales, principalmente desarrollado por el Laboratorio de Investigación de Inteligencia Artificial4 de Facebook (FAIR).

Section 3

Ejemplos prácticos

title

La idea de esta sección es ir mostrando una serie de ejemplos prácticos, para que se vea como se implementa una red neuronal en la práctica. Para ello tenemos que responder: ¿Es una red neuronal adecuada para nuestro problema? ¿Cuál va a ser nuestros datos de entrenamiento? ¿Cuál es nuestra arquitectura? ¿Qué hiperparámetros escoger?

Conecta4 AI

Datos de entrenamiento Mediante el algoritmo Minimax, se genera una tabla de entrenamiento que recoge para una posición del tablero, la mejor jugada que se puede hacer.

Estado Tablero 1	Actua 2	Mejor acción 3	
$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ 1	Valor 2	Valor 3	Valor 4

Table: Tabla de ejemplo

arquitectura

Convolución

Son muy importantes cuando tenemos que procesar imágenes, puesto que ayudan a simplificar la imagen, para luego pasarlo por una capa densa y obtener un buen resultado.

Hiperparámetros

$\text{train}_s ize = 90000 val_s ize = 5000 test_s ize = 5000$
$\text{BATCH}_S IZE = 64 EPOCHS = 512 LR = 0.001$

Criptografia AI

Section 4

Ajuste de los parámetros

Aprendizaje supervisado

- En redes supervisadas, se dispone de datos de entrenamiento, formados por un conjunto de valores de entrada $\widehat{x}$, junto con los resultados asociados, $\widehat{y}$
- Es usual disponer además de datos de test, `xtest`, `ytest`

Función de coste y entrenamiento de la red neuronal

- El proceso de entrenamiento de la red neuronal consiste en determinar los parámetros (pesos, $w_{j,k}^{(i)}$ y desplazamientos, $b_j^{(i)}$) que minimizan un funcional, "función de coste", sobre los datos de entrenamiento:

$$\Theta^* = \text{argmin}\{J(\Theta; \widehat{x}, \widehat{y}), \quad \Theta = \left(w_{j,k}^{(i)}, b_j^{(i)}\right)\}$$

- La función de coste varía con cada tipo de red neuronal. Por ejemplo, en problemas de regresión se suelen usar mínimos cuadrados ("MSE: minimum mean square error"):

$$J(\Theta; \widehat{x}, \widehat{y}) = \frac{1}{N_{data}} \sum_{i=1}^{N_{data}} (\widehat{y}_i - f_{NN}(\widehat{x}_i))^2$$

Algoritmos de minimización

- Dificultades para la minimización: complejidad del funcional de coste, grandes valores de N_{data}
- Enormes requerimientos de cálculo para el entrenamiento, uso de grandes ordenadores, GPUs
- Se suelen utilizar algoritmos de tipo descenso de gradiente[4]

$$\Theta_{k+1} = \Theta_k - \ell_r \nabla_\Theta J(\Theta_k; \hat{x}, \hat{y}), \quad \ell_r \text{: "Learning Rate"}$$

- Necesidad de derivar de forma eficiente: diferenciación automática[5]
- Algoritmos de gradiente estocástico[6]: en cada paso, se calcula el gradiente pero sólo en un subconjunto aleatorio de datos

[4] https://en.wikipedia.org/wiki/Gradient_descent
[5] https://en.wikipedia.org/wiki/Automatic_differentiation
[6] https://en.wikipedia.org/wiki/Stochastic_gradient_descent