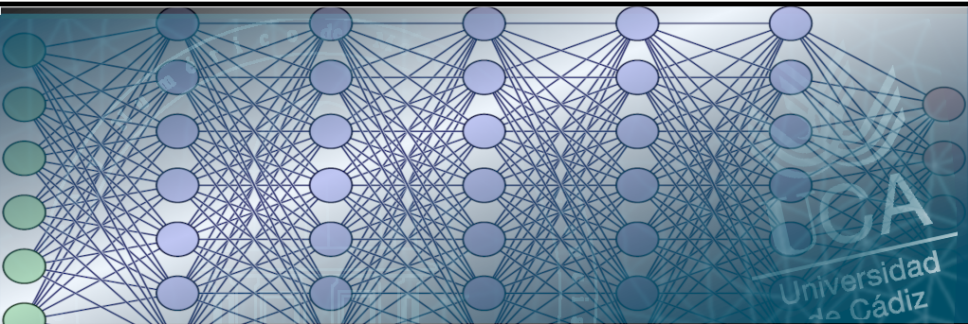


Neural Networks

Álex Pérez Fernández, Rafa Rodríguez Galván

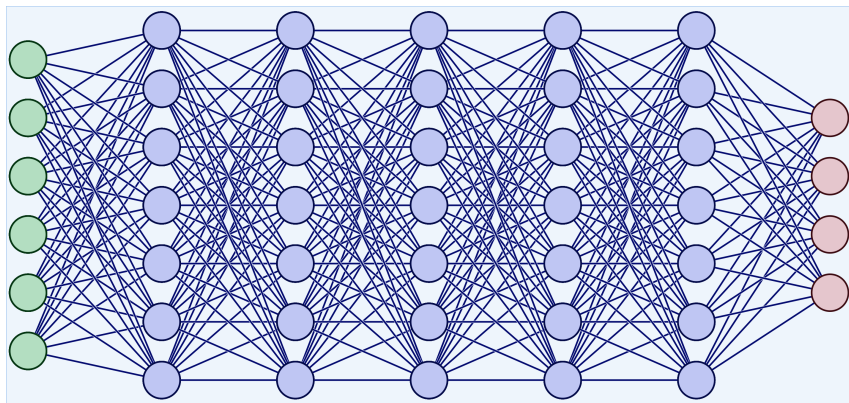
March 11, 2024



Section 1

Neural Networks

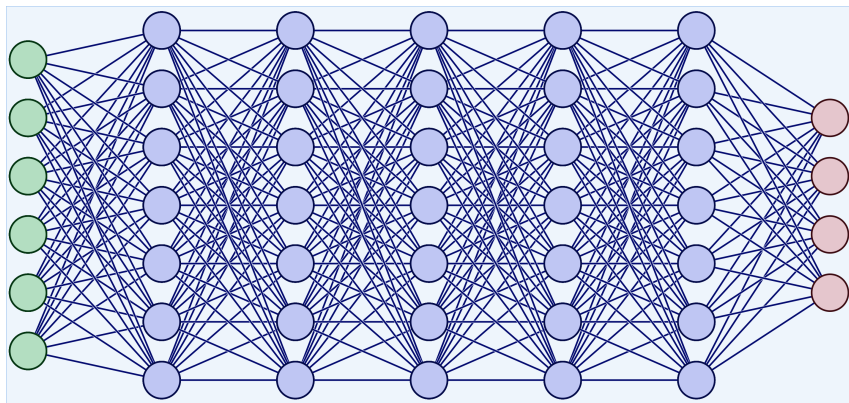
Neural Networks...



...are mathematical artifacts:

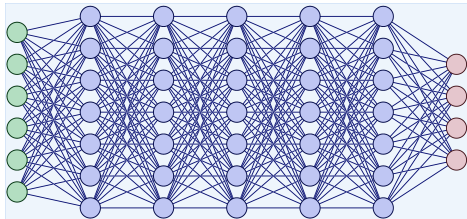
$$x \mapsto f_1(x) \mapsto f_2 \circ f_1(x) \mapsto \dots \mapsto f_L \circ \dots \circ f_2 \circ f_1(x) = y$$

Neural Networks...



...are mathematical artifacts:

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



Definición:

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

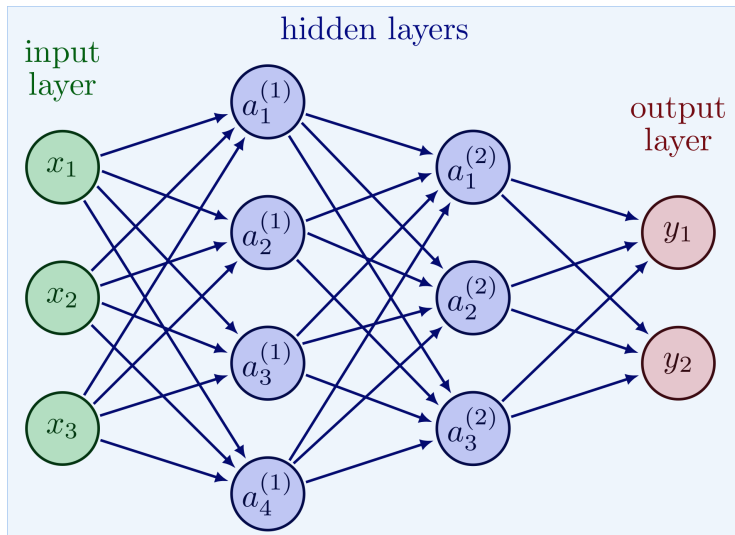
$$y = f_{NN}(x) = f_L \circ \dots \circ f_2 \circ f_1(x).$$

Donde...

- Cada función f_i se llama una **capa** (**entrada** \rightarrow **oculta** \rightarrow **salida**)
- Cada capa f_i está compuesta por un n° variable de **neuronas**
- Cada neurona depende un conjunto de **parámetros**, que determinarán a la RN

★ La RN de la figura se dice de tipo «*feed forward*» o prealimentada

Un ejemplo



$f_{NN} : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ 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 función¹:

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

composición de

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

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

¹Donde 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:

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

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

$$f_i(x) = \sigma_i(W^{(i)}x + b^{(i)})$$

La RN está determinada por los parámetros $W^{(i)}$, los desplazamientos $b^{(i)}$ y las funciones de activación σ_i

Funciones de activación

Es una abstracción que representa la tasa potencial de acción. Si recibimos cierta entrada debemos valorar, si vamos a realizar una acción o no, es decir 0 si no se toma la acción y viceversa. Puede ocurrir también que queramos una probabilidad entre 0 y 1, cuanto más cercana sea a 0 más probable es no tomar acción, de ahí una función de activación sigmoideal.

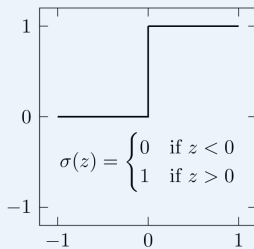
¿Por qué necesitamos funciones de activación?

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.

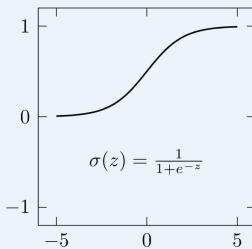
Por lo tanto, no importa cuantas capas tenga el modelo, puesto que estamos realizando una transformación lineal en las entradas. Esto es un problema, puesto que no podemos modelar problemas no lineales.

Representación de funciones de activación

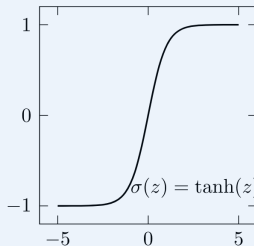
Perceptron



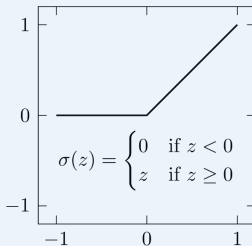
Sigmoid



Tanh



ReLU



Funciones de perdida

Una función de perdida 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.

Podemos pensarlo como un residuo en estadística, que se encarga de medir la distancia entre los valores actuales de y y los de la línea de regresión (valores predichos). El objetivo es minimizar la distancia neta.

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...

Perdida de clasificación: Dado un valor de entrada, el modelo devuelve un vector de probabilidades de que la entrada pertenezca a una serie de categorías preestablecidas. Ejemplos: Binary Cross Entropy, Categorical Cross Entropy...

Optimizadores

Los optimizadores son algoritmos que se usan para minimizar la función de pérdida o maximizar la función de ganancia. Cada vez que predecimos un valor mediante la red y lo comparamos con el valor real, el optimizador ajusta los pesos de la red para minimizar la pérdida. Ejemplos: Adam, RMSprop, SGD...

SGD

El algoritmo de descenso de gradiente estocástico (SGD) es un algoritmo de optimización que se utiliza para minimizar una función objetivo. Para implementarlo: $a_{n+1} = a_n - \delta F(a_n)$

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 pérdida 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 numericas 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 pérdida, optimizadores...

Si un modelo no se comporta bien, entonces tenemos que modificar los hiperparámetros, ya sea para conseguir que el módulo 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 correctivos 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 preciso. 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.

En resumen

Mientras que $\text{epoch} < n^{\circ} \text{ epoch}$ Tomamos una porción de datos (BATCH_SIZE) y la pasamos por la red, calculamos la pérdida (función de pérdida) y actualizamos los pesos.
Cada vez que empleamos la función de pérdida lo usual es mostrar por consola la pérdida, para ver si el modelo está aprendiendo o no. Una pérdida alta implica que al modelo todavía le queda mucho por aprender y no es preciso. (underfitting) Una pérdida extremadamente baja puede llevar a casos de overfitting.

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 2

Ejemplos prácticos

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: ¿Cómo

Conecta4 AI

¿Cómo podemos entrenar una red para que juegue a un juego de mesa?

Section 3

Ajuste de los parámetros

Aprendizaje supervisado

- En redes supervisadas, se dispone de **datos de entrenamiento**, formados por un conjunto de valores de entrada \hat{x} , junto con los resultados asociados, \hat{y}
- Es usual disponer además de **datos de test**, x_{test} , y_{test}

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:

$$\Theta^* = \operatorname{argmin}\{J(\Theta; \hat{x}, \hat{y}), \quad \Theta = (w_{j,k}^{(i)}, b_j^{(i)})\}$$

- 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"):

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

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**²

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

- Necesidad de derivar de forma eficiente: **diferenciación automática**³
- Algoritmos de **gradiente estocástico**⁴: en cada paso, se calcula el gradiente pero sólo en un subconjunto aleatorio de datos

²https://en.wikipedia.org/wiki/Gradient_descent

³https://en.wikipedia.org/wiki/Automatic_differentiation

⁴https://en.wikipedia.org/wiki/Stochastic_gradient_descent