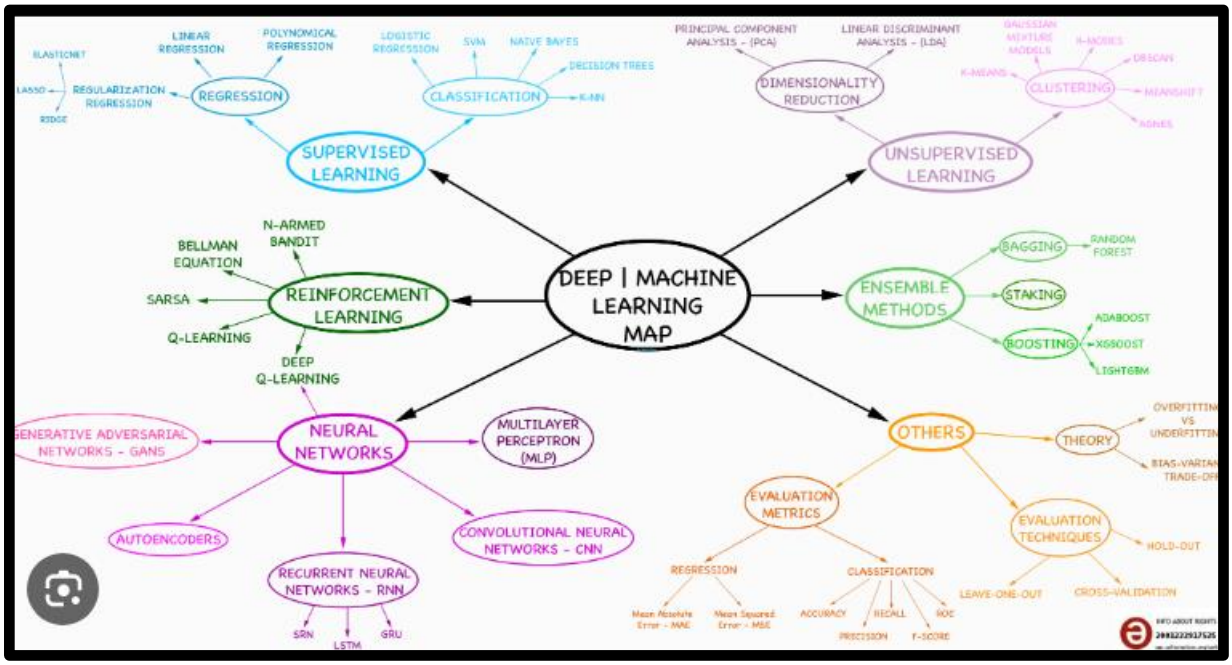
Conceptos generales IA

ÍNDICE

- El algoritmo……
    - Conceptos generales…….
    - Breve repaso matemático…….
    - Creación del algoritmo……
    - Aprendizaje no supervisado……
- Redes convolucionales…….
- Otras redes y aspectos…….
- Bibliografía

1. El algoritmo

1. Conceptos generales



Inteligencia Artificial (IA): Emula el comportamiento humano y sus decisiones.

Matching Learning (ML): Conjunto de diferentes modelos de algoritmos para convertir datos empíricos en modelos reutilizables y predictivos. Matching Learning (aprendizaje automático), es un subdominio de la IA que surge a partir de la creación de un modelo y su posterior ajuste automático para la generalización de su aplicabilidad a inputs de datos de una o diferente clase.

¿Cuándo utilizar ML?Desentrañar sistemas de muchas reglas, en sistemas que varían o fluctúan con frecuencia, en donde no somos capaces de determinar completamente el análisis de los datos (ejemplo clasificación de objetos en una imagen).

Deep learning (DL): Deep Learning (aprendizaje profundo). Modelo de redes neuronales artificiales de múltiples capas ocultas. De hecho a partir de 3 o más capas ocultas se considera deep learning. El total de capas va a determinar la profundidad del modelo.

Clasificación de los sistemas de ML:

En función de la manera que se entrenan:

- **Aprendizaje supervisado**
- **Aprendizaje no supervisado**
- **Aprendizaje semi-supervisado**
- **Aprendizaje reforzado**

En función de la manera en la que aprenden en el tiempo:

- **Aprendizaje online**
- **Aprendizaje batch**

En función de la forma en la que se realizan las predicciones

- **Aprendizaje basado en instancias**
- **Aprendizaje basado en modelos**

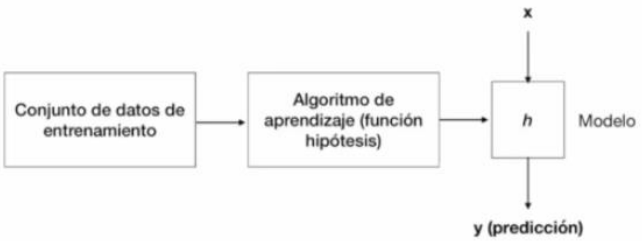
2. ¿Qué problemas pueden resolver los algoritmos?

De forma general los algoritmos de IA tradicionales pretenden estimar, clasificar o predecir. Para ello consideraremos los problemas a los que se enfrenten como problemas de regresión continua o discreta, de clasificación binaria o multiclase, de agrupamiento (clustering). Según el tipo de problema y su solución emplearemos un algoritmo u otro. Actualmente existen métodos de activación de conjunto de redes neuronales específicas para cada problema. Siguiendo estrategias tipo MoE (Mezcla de expertos) una inteligencia artificial puede resolver múltiple tipos de problemas según la activación de unas redes neuronales u otras (Ej: ChatGPT permite completar un poema o ayudar a la programación por la activación de redes neuronales específicas para cada área con inactivación de las no necesarias).

Antes de presentar como es la arquitectura básica de una red neuronal y el funcionamiento de una neurona veremos las estrategias más utilizadas actualmente en el diseño de las mismas, como son, el aprendizaje supervisado y el no supervisado.

Aprendizaje supervisado

- "*El aprendizaje supervisado es la tarea de aprendizaje automático que consiste en aprender una función que mapea una entrada a una salida basada en pares de entrada-salida de ejemplo*" [1]
- La función resultante es utilizada posteriormente para predecir valores a partir de ejemplos de datos no etiquetados



La diferencia entre el aprendizaje supervisado y el no supervisado es que el supervisado necesita de conjunto de datos de entrada y salida ETIQUETADOS. De una forma sencilla, consiste en suministrar a nuestro modelo los datos completamente explicados. (Ej: Modelo de ML supervisado capaz de discernir qué correos son spam o no entonces habría que decir cuáles de los correos son SPAM en el momento de entrenar el modelo).

Dentro del aprendizaje supervisado hay dos tipos de modelos para dos tipos de problemas:

- **REGRESIÓN (lineal, múltiples y general): Modelo capaz de PREDECIR VALORES CONTINUOS: en estos casos debemos, por lo general extraer varias características para que el modelo las analice.**

- **CLASIFICACIÓN: Predice VALORES DISCRETOS. Tipo: Regresión logística.**

Tipo de Regresión. La regresión lineal: Existe una relación lineal entre una y una/muchas variables. Para ello el modelo pretende encontrar las relaciones existentes entre múltiples relaciones dependientes con respecto a una variable independiente.

Tipo de regresión. Regresión múltiple lineal o multivariable: Se utiliza para modelar la relación entre una variable dependiente (o variable de interés) y dos o más variables independientes (o predictoras), asumiendo que existe una relación lineal entre ellas

Tipo de regresión. Regresión general: No nos limitamos a una relación lineal. Destacamos la regresión polinomial, la regresión no lineal y la regresión de múltiple salida. No vamos a profundizar en este tema por su complejidad.

Regresión lineal: Es supervisado, basado en modelos, corresponde con modelo lineal, predice valores continuos y realiza predicciones computando una suma ponderada de características de entrada y sumándole un *bias*.

Regresión logística: Supervisado, basado en modelos, modelo lineal generalizado, predice valores discretos y suma ponderada de las características de entrada más bias y aplicando una función logística

Podemos distinguir entre regresión logística binaria (dos clases) y regresión logística multiclase. Podemos emplear también modelos conocidos como árboles de decisión y clasificadores *bayesianos. De forma general la clasificación pretende discernir a qué clase, previamente descrita, pertenecen los datos mostrados de un vector de x características.*

De forma general el modelo de regresión lineal pretende encontrar la función a encontrar en la regresión lineal es h(x)=n + mx (n y m son los parámetros del modelo). La regresión lineal univariable sería aquella en la que solo tenemos una característica de entrada (Ejemplo: metros cuadrados de la casa para saber su precio). Si son muchas entonces es regresión lineal multivariable y su función sería: h(x)= n + m1x1 + m2x2….

En la regresión lineal univariable al análisis de correlación es similar al que se hace en estadística, que utiliza el coeficiente de correlación de Pearson y que mide cómo de fuerte es la asociación con la variable. Es muy útil para analizar la relación lineal entre dos

variables. En el caso que queramos conocer la relación de una variable con respecto a muchas podemos utilizar la correlación de Pearson pero mediante un análisis bivariado.

Aquí se muestra el coeficiente de correlación de Pearson que se utiliza en estadística:

$$r = \frac{\sum_{i=1}^{n}(X_i-\bar{X})(Y_i-\bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i-\bar{X})^2}\cdot\sqrt{\sum_{i=1}^{n}(Y_i-\bar{Y})^2}}$$

Donde:

- r es el coeficiente de correlación de Pearson.
- X_i y Y_i son los valores individuales de las dos variables que se están comparando.
- $\bar{X}$ y $\bar{Y}$ son los promedios de las variables X y Y, respectivamente.
- n es el número de pares de datos.

Este coeficiente puede tomar valores entre -1 y 1, donde:

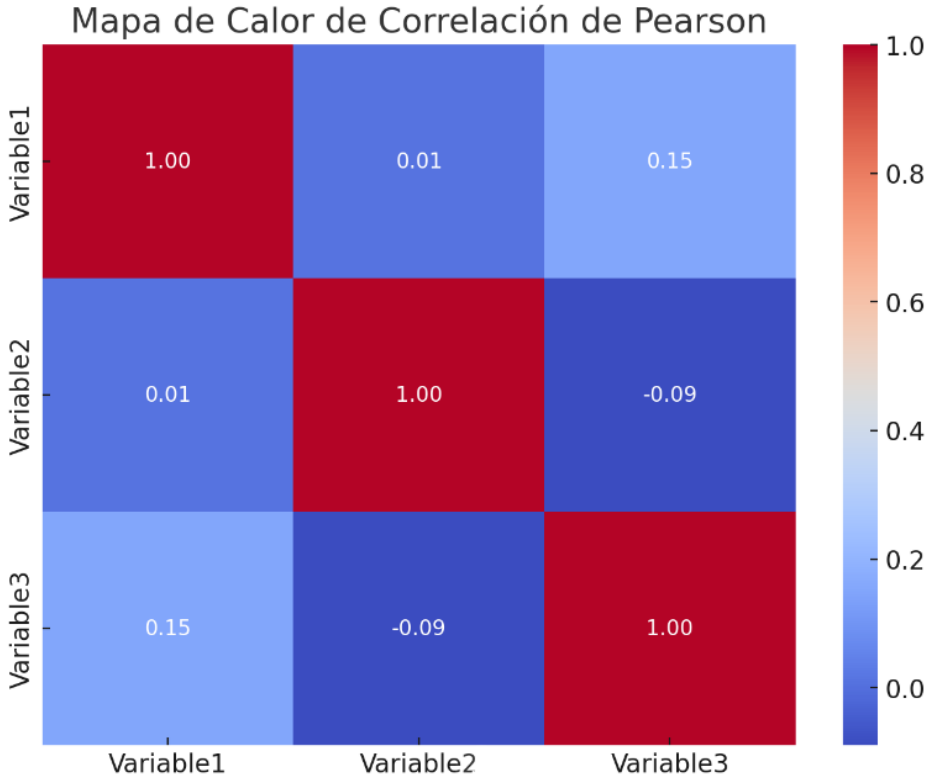
- Un valor de 1 indica una correlación positiva perfecta.
- Un valor de -1 indica una correlación negativa perfecta.
- Un valor de 0 indica que no hay correlación lineal entre las variables.

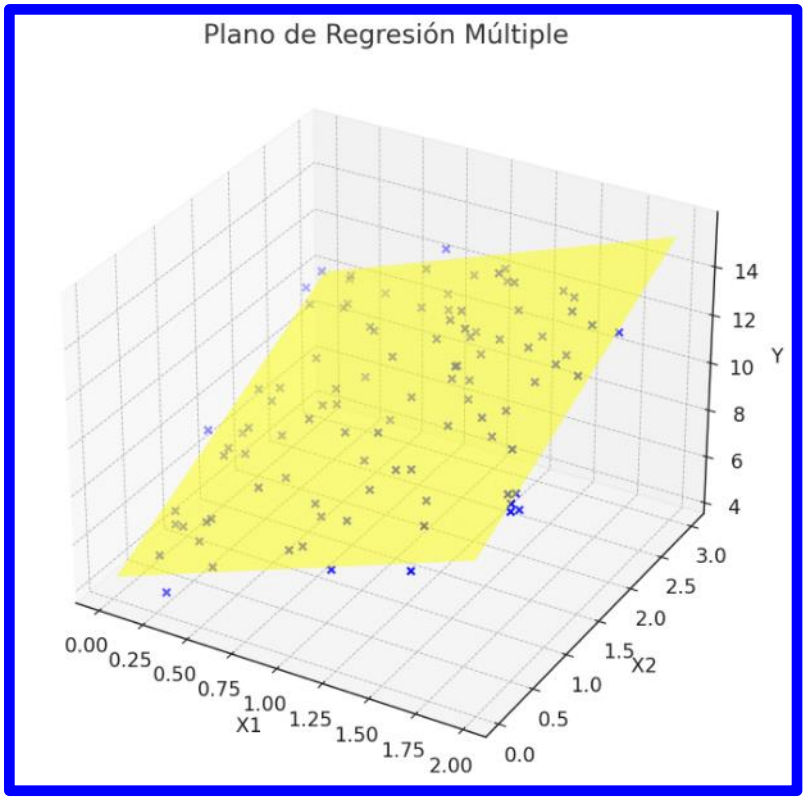
Tened en cuenta que no es un buen método para medir relaciones que no son lineales y para estudiar múltiples variables a la vez (regresión multivariable). ADEMÁS NO INDICA CAUSALIDAD.

En el momento que estemos diseñando nuestro algoritmo de ML puede ser interesante realizar una aproximación estadística mediante el coeficiente de correlación de Pearson para ver si un modelo tiene sentido o no. Examinamos por ejemplo si ciertas variables que tenemos presentan o no una correlación fuerte con otras variables y por tanto diseñar qué variables son más útiles de incluir en nuestro modelo. Para utilizar el coeficiente de correlación de Pearson podemos utilizar la librería de Numpy de Python.

Si nosotros realizamos un análisis de todos los coeficientes de correlación de Pearson de todas las variables mediante un análisis bivariado, entonces los podemos representar en forma de matrices

de correlación y de una forma más legible, mediante mapas de calor. Estos mapas de calor, como he comentado anteriormente, permiten sobre todo en etapas de exploración de los datos, mostrar aquellas variables que pueden ser interesantes de incluir en nuestro modelo y qué variables pueden tener más interés en general.





Regresión multivariable: Imagen que representa la relación de dos variables dependientes con respecto a nuestra variable independiente. El modelo pretende encontrar cual es es plano que explique las relaciones.

3. Breve repaso matemático

1. **La estructura de datos básica de una RN es: escalar-vector-tensor-conjunto.**

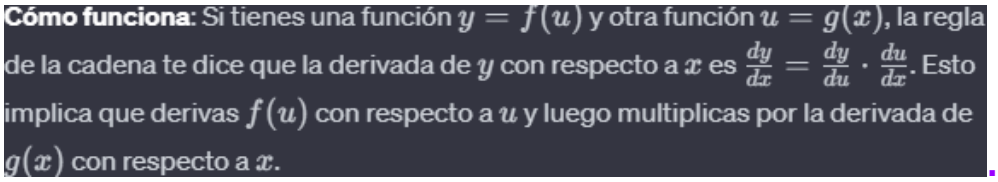
2. **Vector: Arreglo de números UNIDIMENSIONAL—> Representa varias características de un suceso y sus características se llaman atributos.**
   - **La norma de un vector es la longitud de ese vector y se considera que es el tamaño del vector. Permite calcular el error.**
   - **El producto punto=producto escalar o producto interno por el que se obtiene un escalar a partir de dos vectores. Es la proyección de un vector sobre otro, midiendo cuánto del primer vector se proyecta en la dirección del otro. El producto punto es conmutativo, es decir, A· B=B· A, y está relacionado con el coseno del ángulo θ entre los dos vectores.**

     $$\mathbf{A} \cdot \mathbf{B} = \|\mathbf{A}\|\|\mathbf{B}\| \cos(\theta)$$

     Donde $\|\mathbf{A}\|$ y $\|\mathbf{B}\|$ son las magnitudes de los vectores $\mathbf{A}$ y $\mathbf{B}$, respectivamente.
   - **Contracción de dos tensores: La contracción de dos vectores es simplemente su producto-punto.**
   - **Producto de Hadamard: Es la multiplicación de los elementos internos de dos matrices.**
3. **Propiedades de los vectores:**
   - ORDEN=EJE=DIMENSIÓN=GRADO DE LIBERTAD=NÚMERO DE EJES.
   - TAMAÑO=SIZE=Número de atributos
   - FORMA=SHAPE=TUPLA QUE INDICA NÚMERO DE ELEMENTOS POR EJE.

4. **Tensor: Objeto que aloja datos en dos o más ejes. EJ: una matriz es un tensor de 2 ejes. Transponer un tensor es intercambiar sus filas por columnas.**

5. **Algunos de los conceptos tratados a continuación serán expuestos en secciones más adelante. Derivada: Calcula la pendiente de una función en cada punto (para ello calculamos el límite cuando tiende a 0 entre dos puntos adyacentes). Nos va a permitir calcular mínimos locales para ajustar los pesos. El proceso de optimización es el proceso por el cual encontramos los máximos y mínimos de una función; en concreto los puntos críticos; no los mínimos locales. En este proceso utilizamos la regla de la cadena por la cual se calculan derivadas de funciones compuestas. Existen actualmente métodos para calcular la derivada de una función aplicando la regla de la cadena.**



. Las derivadas parciales (como cambia la función siendo el resto de variables constantes.) es fundamental para el cálculo del vector gradiente. Una vez que tenemos los puntos críticos de cada una de las funciones procedemos a realizar la 2 derivada y así comprobar si son máximos o mínimos (más o menos)

6. **Vector gradiente: Vector con las derivadas parciales. Sirve para encontrar la tasa de cambio máxima de una función. Se utiliza de forma automática para encontrar los valores cercanos al óptimo de la función de error.**

7. **La función de activación en las redes neuronales debe ser derivable, no lineal y continua para que se desarrolle el proceso de optimización.**

8. **Algoritmos primordiales: Algoritmo de propagación, de costo, de retropropagación y de gradiente descendente. 1- El de propagación:Va computando todas las neuronas y sus pesos hasta tener una estimación con respecto a cada una de las etiquetas de salida. La función de costo: En muchos casos se utiliza la función de error cuadrático medio. El algoritmo de retropropagación: Consiste en calcular un GRADIENTE de**

costo para cada uno de los pesos y *bias*. El algoritmo de gradiente descendente: Algoritmo que aplica de forma simultánea con la intervención del *learning rate* el cambio de cada uno de los pesos para disminuir el error de la estimación mediante la resta de cada uno de los pesos con su función de coste.

- **Gradiente explosivo y gradiente desvanecido: Se produce cuando los gradientes que son utilizados para actualizar el modelo se hacen o muy grandes o muy pequeños respectivamente. Esto provoca inestabilidades en el sistema. Existen técnicas como el abandono, la normalización por lotes y la regularización que mitigan estos problemas.**

Las técnicas más comunes para acondicionamiento del conjunto de datos son dos puntos estandarización: consiste en transformar el conjunto de datos a partir de la media y la desviación estándar muestrales. Esto se realiza para que el conjunto de datos adquiera una distribución normal. normalización: convierte los valores de las entradas a valores estandarizados para modificar su rango.

A la hora de trabajar con datos:

- **Transforma tus matrices (arrays) de numpy en dataframe (libr. pandas) para VISUALIZARLOS.**

- **Dividir el conjunto de datos en subconjunto de entrenamiento, subconjunto de pruebas y subconjunto de validación=hiperparámetro.**

Librería Numpy de python; algunos ejemplos de uso de arrays:

```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

a es un array:

- Con dos **axis**, el primero de longitud 2 y el segundo de longitud 4
- Con un **rank** igual a 2
- Con un **shape** igual (2, 4)
- Con un **size** igual a 8

```
np.ones((2, 3, 4))
```

```
array([[[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]],

       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]])
```

```
np.full((2, 3, 4), 8)
```

```
array([[[8, 8, 8, 8],
        [8, 8, 8, 8],
        [8, 8, 8, 8]],

       [[8, 8, 8, 8],
        [8, 8, 8, 8],
        [8, 8, 8, 8]]])
```

```
# Creación del array utilizando una función basada en rangos
# (mínimo, máximo, número elementos del array)
np.linspace(0, 6, 10)
```

```
array([0.        , 0.66666667, 1.33333333, 2.        , 2.66666667,
       3.33333333, 4.        , 4.66666667, 5.33333333, 6.        ])
```

```
np.random.rand(2, 3, 4)
```

```
array([[[0.54644301, 0.61653121, 0.12480113, 0.46933094],
        [0.72785198, 0.87236617, 0.14925486, 0.38542499],
        [0.26400079, 0.01936632, 0.40779719, 0.59169042]],

       [[0.96730686, 0.72646096, 0.96198198, 0.07893276],
        [0.7882073 , 0.65501645, 0.28074147, 0.33980307],
        [0.07038478, 0.01395099, 0.7811747 , 0.50715692]]])
```

Broadcasting

Si se aplican operaciones aritméticas sobre Arrays que no tienen la misma forma (shape) Numpy aplica un propiedad que se denomina Broadcasting.

Librería Pandas de Python:

Pandas es una librería que proporciona estructuras de datos y herramientas de análisis de datos de alto rendimiento y fáciles de usar.

- La estructura de datos principal es el DataFrame, que puede considerarse como una tabla 2D en memoria (como una hoja de cálculo, con nombres de columna y etiquetas de fila).
- Muchas funciones disponibles en Excel están disponibles mediante programación, como crear tablas dinámicas, calcular columnas basadas en otras columnas, trazar gráficos, etc.
- Proporciona un alto rendimiento para manipular (unir, dividir, modificar…) grandes conjuntos de datos

La librería Pandas, de manera genérica, contiene las siguientes estructuras de datos:

- **Series**: Array de una dimensión
- **DataFrame**: Se corresponde con una tabla de 2 dimensiones
- **Panel**: Similar a un diccionario de DataFrames

```python
# Creacion de un objeto Series
s = pd.Series([2, 4, 6, 8, 10])
s
```

```
0     2
1     4
2     6
3     8
4    10
dtype: int64
```

```python
# Creación de un objeto Series inicializándolo con un diccionario de Python
altura = {"Santiago": 187, "Pedro": 178, "Julia": 170, "Ana": 165}
s = pd.Series(altura)
s
```

```
Santiago    187
Pedro       178
Julia       170
Ana         165
dtype: int64
```

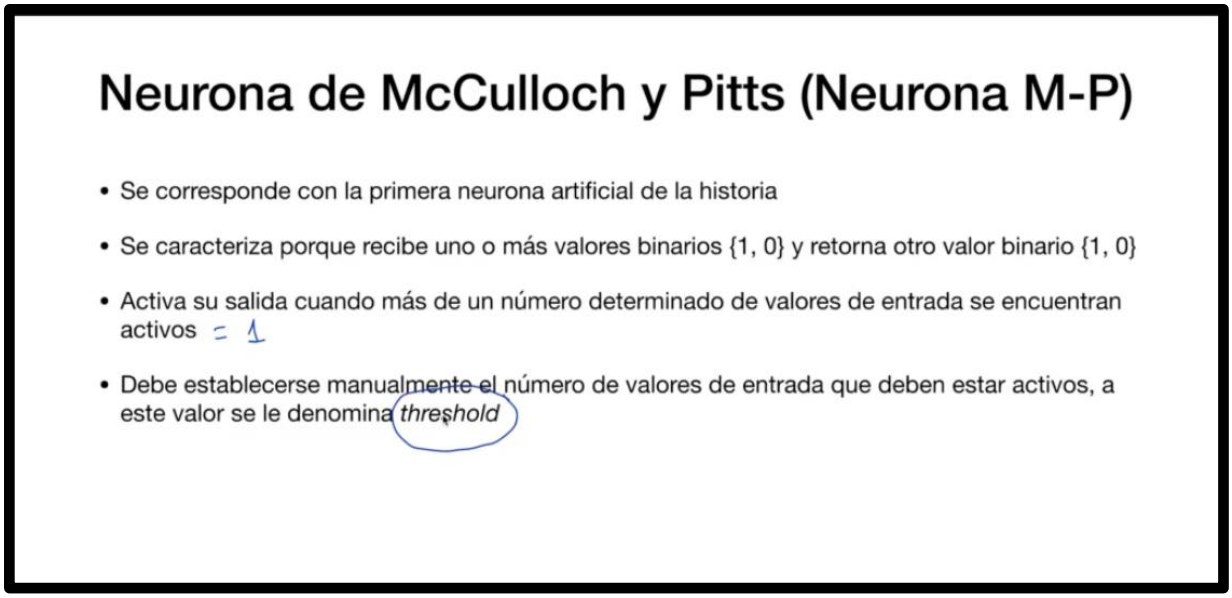
```python
# Creación de un objeto Series inicializándolo con algunos
# de los elementos de un diccionario de Python
altura = {"Santiago": 187, "Pedro": 178, "Julia": 170, "Ana": 165}
s = pd.Series(altura, index=["Pedro", "Julia"])
s
```

```
Pedro    178
Julia    170
dtype: int64
```

4. Creación del algoritmo

En cuanto al origen de la IA podemos describir un hito muy importante el desarrollo de lo que se conoció más adelante como la neurona de Neurona de McCulloch-Pitts que tuvo lugar en 1943.

1. **NEURONA MCCULLOCH Y PITTS: Era una neurona que permitía la clasificación binaria:**



A continuación se expone un ejemplo de implementación de neurona de McCulloch y Pitts haciendo uso del lenguaje de programación python:

1. Implementación de la MPNeuron

```
In [ ]: import numpy as np

        class MPNeuron:

            def __init__(self):
                self.threshold = None

            def model(self, x):
                # input: [1, 0, 1, 0] [x1, x2, .., xn]
                z = sum(x)
                return (z >= self.threshold)

            def predict(self, X):
                # input: [[1, 0, 1, 0], [1, 0, 1, 1]]
                Y = []
                for x in X:
                    result = self.model(x)
                    Y.append(result)
                return np.array(Y)
```

```
In [ ]: # Instanciamos la neurona
        mp_neuron = MPNeuron()
```

```
In [ ]: # Establecemos un threshold
        mp_neuron.threshold = 3
```

```
In [ ]: # Evaluamos diferentes casos de uso
        mp_neuron.predict([[1, 0, 0, 0], [1, 1, 1, 1], [1, 1, 1, 0]])
```

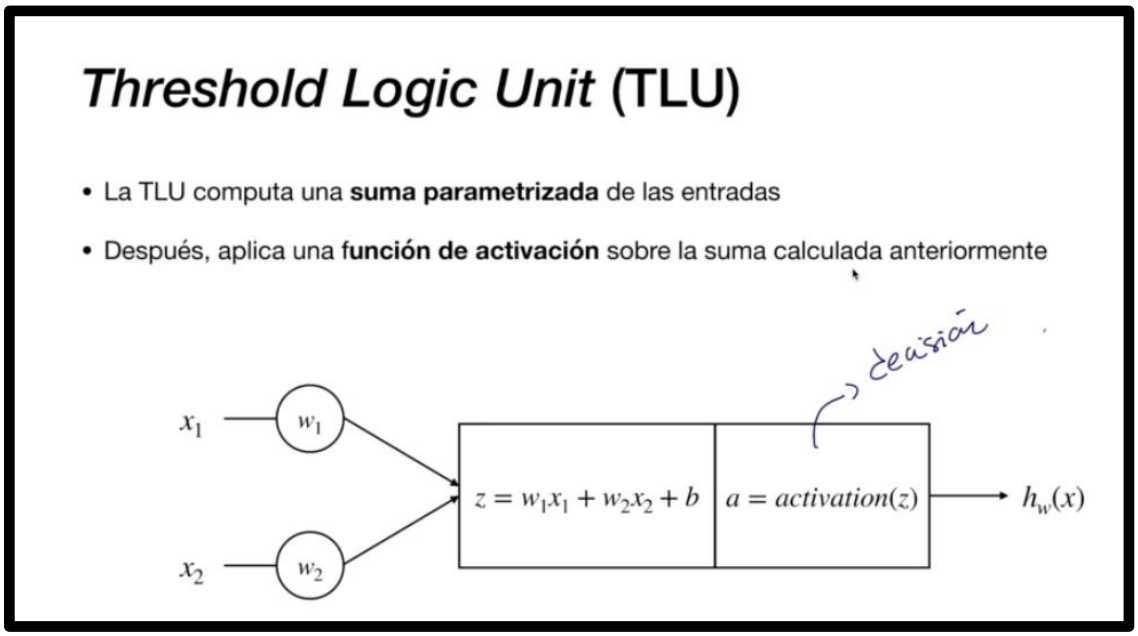
```
In [4]: # Evaluamos diferentes casos de uso
        mp_neuron.predict([[1, 0, 0, 0], [1, 1, 1, 1], [1, 1, 1, 0]])

Out[4]: array([False,  True,  True])
```

Aproximadamente una década más adelante tuvo lugar el desarrollo del perceptrón, basado en aprendizaje supervisado y útil para problemas lineales.

2. Perceptrón: Permite clasificación binaria y multiclase

El perceptrón se compone de varias TLUs (unidad de límite lineal): Recibe como entradas valores continuos y tiene funciones de activación más complejas; no únicamente una suma ponderada de las entradas. Además tener en cuenta que puede entrenarse con el learning rate. Fue desarrollado en 1958.



Importante limitación: Construye límites lineales. No proporciona resultados en forma de probabilidad sino basándose en un límite estático.

Ejemplo práctico del desarrollo de un perceptrón:

1. Estructura de un Perceptrón

Un perceptrón consiste en lo siguiente:

- **Entradas**: $x_1, x_2, \ldots, x_n$
- **Pesos**: $w_1, w_2, \ldots, w_n$
- **Umbral de activación** o **sesgo**: b
- **Función de activación**: típicamente una función escalón, que determina la salida en función de una suma ponderada de las entradas.

2. Función de Suma Ponderada

La función de suma ponderada calcula la suma de las entradas multiplicadas por sus respectivos pesos, añadiendo el sesgo al final. Esto se puede expresar como:

$$z = w_1x_1 + w_2x_2 + \ldots + w_nx_n + b$$

3. Función de Activación

La función de activación más común en un perceptrón clásico es la función escalón, que se define como:

$$\text{salida} = \begin{cases} 1 & \text{si } z \geq 0 \\ 0 & \text{si } z < 0 \end{cases}$$

4. Proceso de Aprendizaje

El aprendizaje en un perceptrón se realiza mediante la actualización de los pesos y el sesgo basado en los errores cometidos en las predicciones. El proceso de actualización se hace generalmente con la regla de aprendizaje del perceptrón, que se define como sigue:

- **Tasa de aprendizaje**: η (un parámetro que define cuánto cambian los pesos durante la actualización)

La regla de actualización para cada peso w_i es:

$$w_i \leftarrow w_i + \eta(y - \hat{y})x_i$$

donde:

- y es la etiqueta real de la muestra.
- $\hat{y}$ es la predicción hecha por el perceptrón.
- x_i es el valor de la entrada correspondiente al peso w_i.

El sesgo se actualiza de manera similar:

$$b \leftarrow b + \eta(y - \hat{y})$$

5. Iteración y Convergencia

El perceptrón itera sobre el conjunto de entrenamiento, ajustando los pesos y sesgos hasta que los errores se minimizan o hasta que se alcanza un número máximo de iteraciones. Teóricamente, si los datos son linealmente separables, el perceptrón convergerá a una solución.

5. Iteración y Convergencia

El perceptrón itera sobre el conjunto de entrenamiento, ajustando los pesos y sesgos hasta que los errores se minimizan o hasta que se alcanza un número máximo de iteraciones. Teóricamente, si los datos son linealmente separables, el perceptrón convergerá a una solución.

Ejemplo Numérico

Supongamos que tienes las siguientes entradas y pesos iniciales:

- Entradas: $x_1 = 1.5, x_2 = -2$
- Pesos iniciales: $w_1 = 0.5, w_2 = -0.5$
- Sesgo inicial: $b = 0$
- Tasa de aprendizaje: $\eta = 0.1$
- Etiqueta real: $y = 1$

Cálculo de la salida

$$z = (0.5)(1.5) + (-0.5)(-2) + 0 = 0.75 + 1 = 1.75$$

$$\hat{y} = 1 \text{ (ya que } z \geq 0)$$

3. Red neuronal profunda: Perceptrón multicapa.

Durante mucho tiempo una de las mayores limitaciones del perceptrón multicapa era que no existía una forma adecuada de entrenarlo. En 1986 se presenta un algoritmo que revoluciona de entrenar el perceptrón multicapa; el algoritmo de backpropagation. El cambio calve del algoritmo respecto a los anteriores fue el de reemplazar la función de activación Heaviside step function por la sigmoide. En la actualidad existen otras funciones de activación populares como tanh(z) y ReLu(z).

El perceptrón multicapa es el modelo más popular dentro del Deep Learning y se caracteriza por la composición de múltiples funciones.

- Todas las capas se encuentra totalmente conectadas (***fully connected***) con las siguientes capas
- Cuando una ANN tiene dos o más *hidden layers*, se denomina ***Deep Neural Network (DNN)***

El MLP (perceptrón multicapa) es una extensión del perceptrón simple y consta de múltiples capas de neuronas interconectadas, incluyendo una capa de entrada, una o más capas ocultas y una capa de salida. Nos permite obtener valores continuos.

Los grafos permiten mostrar y representar el funcionamiento de una red neuronal. Los nodos se corresponden con una operación o una variable y los arcos con valores de entrada/salida. En la gráfica de abajo vemos un ejemplo de grafo computacional donde el error se calcula para la última capa de neuronas.

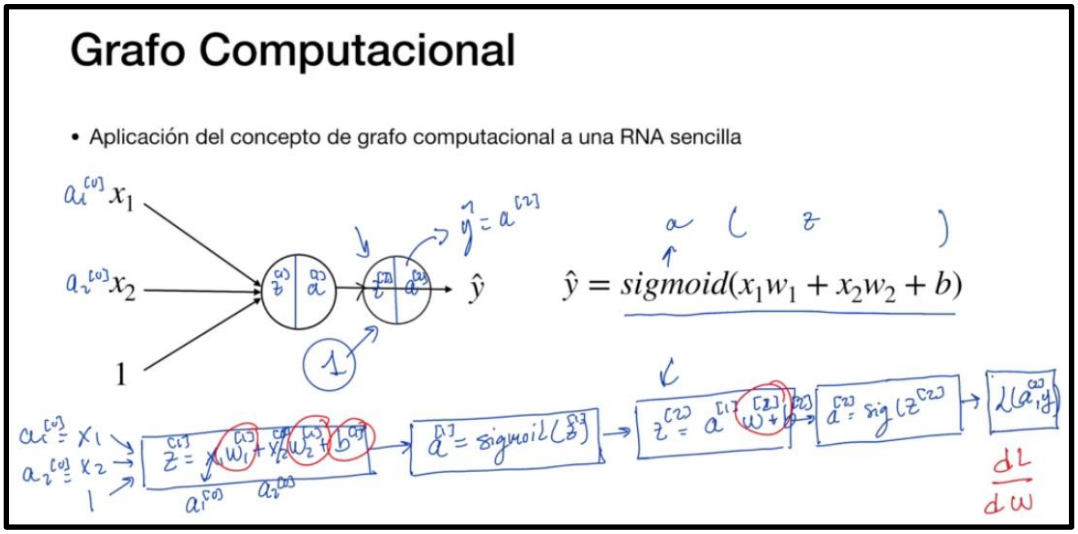


Imagen de perceptrón multicapa de aprendizaje supervisado. Cómo veremos más adelante, para poder aplicar la función de optimización en todos los pesos de las neuronas anteriores se realizará el proceso de: BACK PROPAGATION. En este grafo computacional se presenta el funcionamiento básico de un perceptrón multicapa con una capa oculta y una capa de salida conformada cada una por una neurona únicamente. En la capa oculta, por ejemplo, podemos ver cómo acceden a ellas varios parámetros (x1 y x2). Se ejecuta la función "z" sobre ellos. Posteriormente se ejecuta la función de activación "a" sobre el resultado de la función "z". El resultado de la función "a" será la entrada de la siguiente neurona, en este caso la neurona de la capa de salida. También se aprecia como tanto en la neurona de la capa oculta como en la neurona de salida se añade el factor "b" (bias) (sesgo, en español) y que aporta flexibilidad a la neurona. En este ejemplo, la función de activación es la función sigmoide.

Vamos a presentar como sería el funcionamiento de un perceptrón multicapa que sigue un modelo de aprendizaje supervisado. Este ejemplo sencillo nos va a servir para comprender la base general de cómo funcionan la práctica totalidad de las neuronas artificiales:

1. **Analizamos nuestro problema y la solución que queremos: Queremos que nuestro modelo clasifique en tres ejemplos de especies de flores. Para ello va a utilizar dos características/parámetros que nosotros pensamos que son relevantes como son: número de habitaciones (x1) y metros cuadrados (x2). Pero ¿Qué importancia tiene x1 y x2 a la hora de estimar el precio de una casa?¿Es más importante x1 o x2? Nuestro modelo va a intentar averiguar qué PESO tiene cada característica a la hora de saber a qué especie pertenece la flor que nosotros le presentemos.**

2. **Diseñamos el modelo y lo comprendemos. En nuestro caso va a ser un perceptrón multicapa de aprendizaje supervisado, como el expuesto en la imagen de más arriba. Todo modelo de este tipo debe constar de varias capas y cada capa con una o más neuronas. Cada neurona debemos de considerarla como una unidad de cálculo. En esta unidad de cálculo se van a desarrollar dos operaciones principales. La primera de estas operaciones consiste en aplicar la función principal de esa neurona a cada una de las características qué nosotros hemos considerado importantes (en nuestro caso número de habitaciones y número de metros cuadrados). La siguiente operación va a consistir en aplicar la función de activación (en el ejemplo de arriba era la función sigmoide) al resultado de la función principal. Inicialmente el algoritmo va a otorgar un PESO al azar a cada una de las dos características para luego ir ajustando el modelo y encontrar que auténtica importancia tiene cada una de las características.**

3. **Entrenamiento del modelo: Cálculo del error y su optimización, modificación de hiperparámetros: Como hemos comentado, inicialmente nuestro modelo va a introducir valores al azar en los parámetros recogidos. Ahora bien, esta importancia dada al azar por nuestro modelo hay que corregirla y ajustarla. Para ello, va a comparar cuánto de alejado ha estado su predicción del resultado final, para luego, a partir de su cálculo y posterior ajuste realizar una nueva estimación y volver a repetir el proceso. Como he comentado, primeramente calcula el error de su predicción con respecto al valor real del coste de la casa. Para ello va a utilizar la función de coste, que va a ser diferente para cada tipo de modelo. Una vez calculado el error se calcula la derivada parcial de cada característica sobre la función de coste para averiguar cuánto qué ponderación tenía cada característica sobre el error total y proceder a su corrección. Conforme se vaya produciendo el entrenamiento de nuestro algoritmo se irá desarrollando la función de tipo sigmoide que mejor ajuste el precio de la casa para las dos**

características presentadas (número de habitaciones y número de metros cuadrados).

4. **Evaluación de nuestro modelo.** Para ello aparte de los datos de entrenamiento y validación se emplea otro subconjunto que conforman los datos de prueba.

El proceso más importante en el desarrollo de nuestro modelo es el paso **3**. Para ello vamos a profundizar un poco más en la función de coste=error de nuestro modelo y el proceso de optimización=ajuste de las características para que puedan estimar mejor el precio real de la casa.

3.1 Primeramente comentaremos la función de coste. La función de coste se va a aplicar a todas las neuronas de todas las capas sumando el error de la diferencia entre la etiqueta (valor real de la casa) y el resultado proporcionado (estimación de nuestro modelo). Como hemos comentado anteriormente, la función de coste va a ser diferente según el problema inicial que nos planteemos.

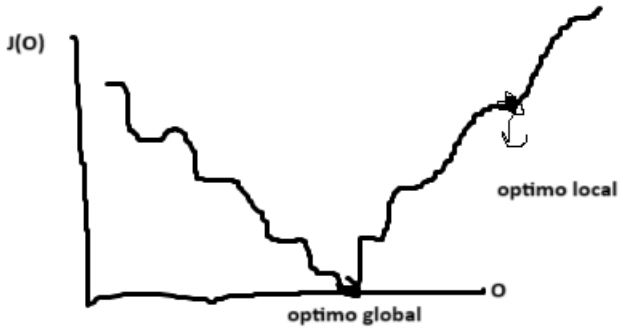
La función de error, como sabemos, va a ser diferente según el tipo de problema que estemos resolviendo y el tipo de función de activación empleada. Cuando estamos frente a problemas de clasificación y utilizamos la función softmax entonces emplearemos la siguiente función de error:

$H(y, \hat{y}) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$ Esta es la función que va a utilizar gradient descent para minimizar el error. En problemas de clasificación donde se utiliza la función softmax, la función de error más comúnmente empleada es la **entropía cruzada categórica**. La función softmax es ideal para problemas de clasificación multiclase, ya que convierte los logits (puntuaciones de clase sin procesar) de un modelo en probabilidades, asegurándose de que sumen uno y sean directamente interpretables como tales.

La entropía cruzada categórica mide la discrepancia entre las distribuciones de probabilidad reales y las predichas por el

modelo. Funciona muy bien en combinación con softmax porque penaliza las predicciones que están seguras pero incorrectas de manera muy significativa, lo que es deseable durante el entrenamiento de modelos de clasificación. La función de entropía cruzada aumenta a medida que la predicción se aleja de la etiqueta real.

La función de error en la regresión logística como hemos dicho no es la misma que en la regresión lineal, ya que solamente obtendríamos óptimos locales.



Las salidas producen valores que podemos interpretar como probabilidades por lo que empleamos la entropía cruzada. La entropía cruzada ("cross-entropy"), es una medida utilizada para evaluar la diferencia entre la distribución de probabilidad de las predicciones del modelo y la distribución de probabilidad real de las etiquetas de clase.

$$H(y, p) = - [y * \log(p) + (1 - y) * \log(1 - p)]$$

Donde:

- H(y, p) es la entropía cruzada.
- y es la etiqueta real de la clase (0 o 1).
- p es la probabilidad de predicción de la clase positiva por parte del modelo.

La función de costo-error en los modelos de regresión múltiple es:

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Donde:

- $J(\theta)$ es la función de costo.
- n es el número de observaciones en el conjunto de datos.
- $h_\theta(x^{(i)})$ es la predicción del modelo para la i-ésima observación. En el caso de la regresión lineal multivariable, esto se calcula como $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_k x_k$ donde θ_j son los parámetros del modelo y x_j son las variables predictoras.
- $y^{(i)}$ es el valor real de la i-ésima observación.
- La suma se realiza sobre todas las observaciones del conjunto de datos.

El coeficiente de determinación ajustado, comúnmente denotado como R^2 ajustado se utiliza más para medir el rendimiento de nuestro algoritmo de regresión lineal multivariable.
R^2 ajustado = 1 - [(1 - R^2) * (n - 1) / (n - p - 1)]

Dónde:
- R^2 es el coeficiente de determinación tradicional.
- n es el número de observaciones en el conjunto de datos.
- p es el número de predictores o variables independientes en el modelo; características.

El R^2 ajustado varía entre 0 y 1, y un valor más alto indica que el modelo es una mejor representación de los datos.

En el análisis del error del modelo de regresión vamos a utilizar principalmente el error global y el error local. El error global evalúa la media de las diferencias individuales entre las predicciones "Î" del modelo con respecto a los datos observados "I". De forma general, para medir este desempeño global utilizamos:

El Error Cuadrático Medio (ECM) es una medida comúnmente utilizada para evaluar la precisión de un modelo de regresión. La fórmula para calcular el ECM es:

$$\text{ECM} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Donde:

- n es el número total de observaciones.
- Y_i es el valor real de la i-ésima observación.
- $\hat{Y}_i$ es el valor predicho por el modelo para la i-ésima observación.

$$\text{RECM} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}$$

En forma de raíz cuadrada facilita su interpretación al estar en las mismas unidades que las medidas. Otras medidas son el error medio absoluto (EMA) y el error porcentual absoluto medio (MAPE), pero son menos usadas .

En cuanto al error local se utiliza más para evaluar el desempeño y solamente la debemos de aplicar con el conjunto de datos de prueba (*Sp; como se describe en la literatura*). Para ello se emplean las mediciones de Error absoluto

$$\text{EAM} = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i|$$

o el error medio relativo. Tener en cuenta que los errores medios locales no se encuentran integrados fácilmente en python.

De forma general podemos decir que la función de error para modelos de regresión lineal es:

Para modelos de regresión lineal, la función de error más comúnmente utilizada es el **error cuadrático medio (MSE, por sus siglas en inglés)**. Esta función mide la media de los cuadrados de las diferencias entre los valores observados reales y los valores predichos por el modelo. El MSE es muy popular porque es sencillo de comprender y de implementar, además de que penaliza más fuertemente los errores grandes, lo que puede ser deseable en muchos escenarios prácticos.

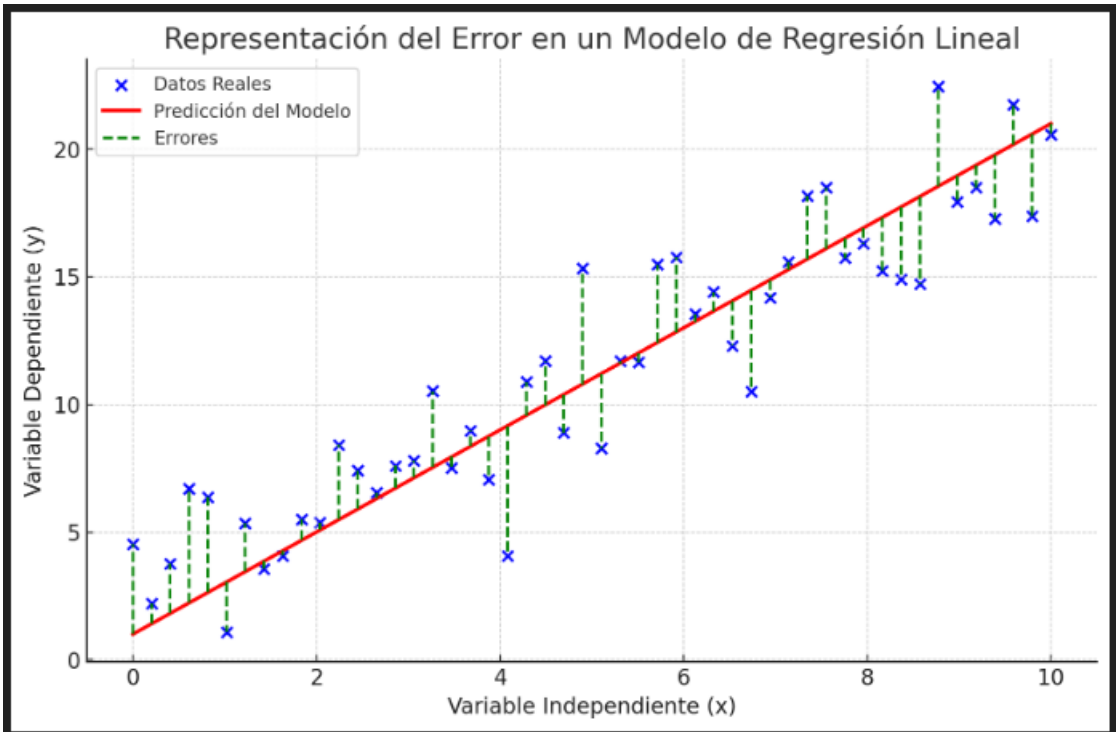
La representación matemática del error cuadrático medio es la siguiente:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Donde:

- n es el número total de muestras o puntos de datos.
- y_i es el valor real observado para la i-ésima muestra.
- $\hat{y}_i$ es el valor predicho por el modelo para la i-ésima muestra.

El objetivo al entrenar el modelo es minimizar esta función de error, lo que significa ajustar la línea de regresión de manera que la suma de los cuadrados de los errores (las diferencias entre los valores observados y los predichos) sea lo más pequeña posible.



En el gráfico, los puntos azules representan los datos reales y la línea roja muestra la predicción del modelo de regresión lineal. Las líneas verdes discontinuas indican los errores individuales para cada punto, que son las diferencias verticales entre los valores reales y los valores predichos por el modelo. Estos errores son los que se cuadran y suman para calcular el error cuadrático medio (MSE), el cual se minimiza durante el entrenamiento del modelo.

1. **Problemas de Regresión:**
   - En problemas de regresión, donde estás prediciendo valores numéricos, las funciones de error comunes incluyen el Error Cuadrático Medio (MSE) y el Error Absoluto Medio (MAE).
   - El MSE penaliza de manera más fuerte las predicciones erróneas, ya que eleva al cuadrado las diferencias entre las predicciones y los valores reales. Es más sensible a los errores grandes.
   - El MAE es menos sensible a los errores grandes, ya que toma el valor absoluto de las diferencias. Es más robusto si tienes valores atípicos en tus datos.
2. **Problemas de Clasificación:**
   - En problemas de clasificación, donde estás prediciendo etiquetas de clase, la Entropía Cruzada (CrossEntropy) es comúnmente utilizada.
   - La Entropía Cruzada mide la discrepancia entre la distribución de probabilidad predicha por el modelo y la distribución real de las clases. Es especialmente útil para problemas de clasificación binaria o multiclase.
   - Otras funciones de error, como la Pérdida de Bisagra (Hinge Loss) o la Pérdida Logarítmica (Log Loss), también son utilizadas en contextos específicos.
3. **Problemas de Generación de Texto:**
   - En problemas de generación de texto, donde estás generando secuencias de palabras, la Pérdida de Entropía Cruzada Secuencial (Sequence CrossEntropy Loss) es comúnmente utilizada.
   - Esta función de error tiene en cuenta la naturaleza secuencial de las predicciones y penaliza las predicciones incorrectas en cada paso de la secuencia.

La elección de la función de error depende del tipo de problema que estés abordando (regresión, clasificación, etc.). Aquí hay algunas funciones de error comunes:

1. **Error cuadrático medio (MSE - Mean Squared Error):**
   $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
   donde n es el número de ejemplos en el conjunto de datos, y_i son los valores reales, y $\hat{y}_i$ son las predicciones del modelo.
2. **Error absoluto medio (MAE - Mean Absolute Error):**
   $MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
3. **Entropía cruzada (para problemas de clasificación):**
   $CrossEntropy = -\frac{1}{n}\sum_{i=1}^{n}(y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i))$
   donde y_i es la etiqueta verdadera (0 o 1) y $\hat{y}_i$ es la probabilidad predicha por el modelo de que el ejemplo pertenezca a la clase positiva.

Esta imagen y la de arriba muestra la función de error más utilizada para cada tipo de problema. En nuestro caso estamos ante un ejemplo (cálculo de precio de una casa) de regresión lineal multivariable, por lo que emplearemos el error cuadrático medio.

Aquí os dejo el cálculo de la función de error más utilizada para un problema de clasificación:

$$\text{Función de coste: } J(W,B) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \ln(\hat{y}^{(i)}) + \left(1 - y^{(i)}\right) \ln\left(1 - \hat{y}^{(i)}\right)$$

3.2 *A partir de la función de error optimizamos el modelo:* En nuestro caso estamos intentando aplicar el proceso de optimización a la función del error cuadrático medio en un modelo de regresión lineal (función principal (y) del modelo es y= wx + b) donde "w" es el parámetro que quiere ajustar nuestro modelo, "x" la características proporcionadas y "b" el factor bias o de sesgo.

La fórmula del MSE es:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$
$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - (wx_i + b))^2$$

En esta imagen se muestra como primera función la función de error cuadrático medio. Abajo se sustituye la estimación por la función principal de nuestro modelo que la calcula.

1. Gradiente con respecto a w:
$$\frac{\partial}{\partial w}\text{MSE} = \frac{-2}{n}\sum_{i=1}^{n} x_i(y_i - (wx_i + b))$$
2. Gradiente con respecto a b:
$$\frac{\partial}{\partial b}\text{MSE} = \frac{-2}{n}\sum_{i=1}^{n}(y_i - (wx_i + b))$$

Se procede a la derivación parcial de la función de error para cada una de las características, representado como varía la función MSE para pequeños cambios de w y b.

$$w := w - \alpha \frac{\partial}{\partial w} \text{MSE}$$
$$b := b - \alpha \frac{\partial}{\partial b} \text{MSE}$$

En esta imagen se presenta la función de gradiente descendente por la que se ajusta el peso según la derivada parcial mostrada anteriormente. Alfa es el learning rate que va a definir cómo de rápido queremos que aprenda nuestro algoritmo.

Según esto tenemos dos pasos fundamentales: backpropagation y Gradient descent. Lo que se conoce como BACKPROPAGATION o proceso por el cual realizamos las derivadas parciales de la función de coste respecto a todos los pesos, en el que se propaga hacia atrás el error para ver cómo intervienen en él cada uno de los pesos. Posteriormente aplicamos la función de optimización Gradient descent para cada peso.

El proceso de BACKPROPAGATION se basa en el uso de dos conceptos matemáticos importantes; de la regla de la cadena y las derivadas parciales. La regla de la cadena es el proceso por el que podemos calcular la derivada de una función compuesta. En nuestro caso la función de error se trata de una función compuesta ya que se compone de las salidas de las funciones de activación de las neuronas de capas previas. La derivada parcial se utiliza para ver cómo cambia dicha derivada con respecto a cada uno de los parámetros de forma independiente.

Lo interesante de esto es que, una vez calculadas estas derivadas parciales, podemos usar la regla de la cadena para seguir recorriendo la red neuronal de derecha a izquierda, calculando el resto de derivadas parciales. Así, si quisiéramos calcular la derivada parcial de la función de coste parcial C_i con respecto a a_1 podríamos hacerlo aplicando la regla de la cadena:

$$\frac{\partial C_i}{\partial a_1} = \frac{\partial C_i}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial z_3} \frac{\partial z_3}{\partial a_1}$$

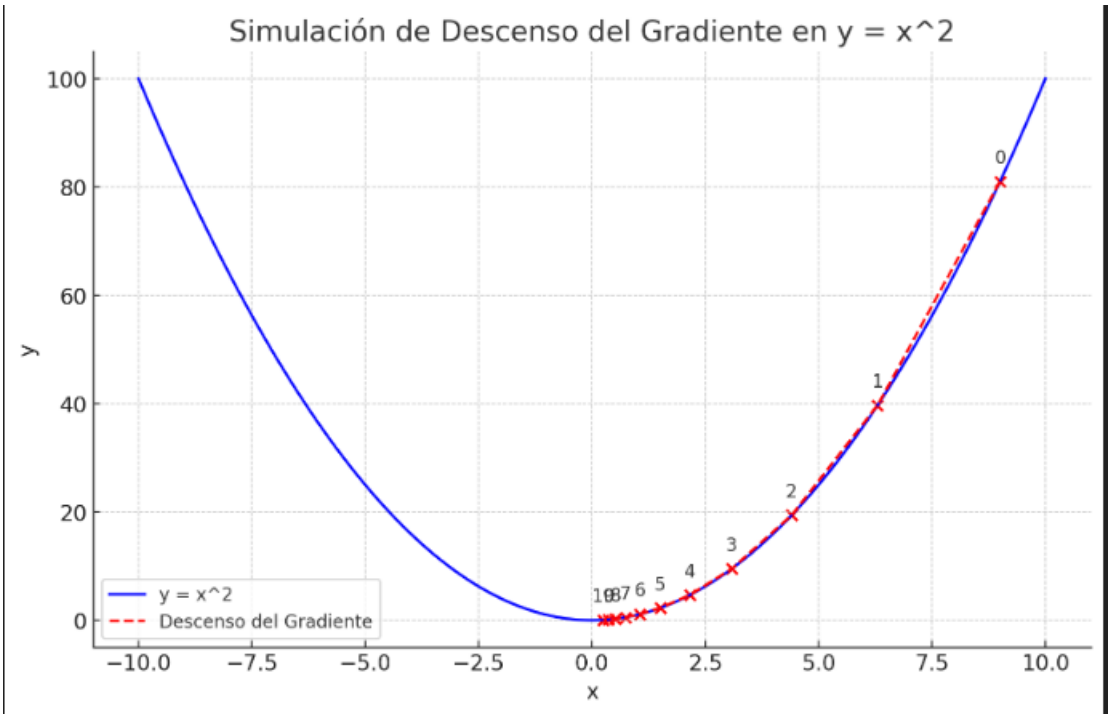
Si tenemos en cuenta la relación entre a_1 y z_3:

$$z_3 = a_1 w_{31} + a_2 w_{32} + b_3$$

...podemos calcular la derivada parcial de z_3 con respecto a a_1:

$$\frac{\partial z_3}{\partial a_1} = w_{31}$$

La función de optimización utilizada en la regresión lineal suele ser Gradient descent. Lo que pretende es encontrar su mínimo global de la función de error. Para ello vamos a ir calculando las derivadas y sus pendientes para alcanzar el mínimo punto como ya sabemos.



Aquí tienes un gráfico que representa el proceso de descenso del gradiente para la función $y = x^2$. He comenzado en el punto $x = 9$ y he aplicado iterativamente la actualización de descenso del gradiente utilizando una tasa de aprendizaje de 0.15. En cada paso, calculé el gradiente, lo multiplicé por la tasa de aprendizaje, y actualicé la posición de x para moverme hacia el mínimo de la función.

En el gráfico, los puntos rojos representan las posiciones de x en cada iteración, y la línea roja discontinua muestra el camino que sigue el descenso del gradiente. Las anotaciones numeran cada paso, mostrando cómo el valor de x se acerca al mínimo (0,0) donde la derivada es 0 y y alcanza su valor mínimo. [>-]

Gradient descendet: Básicamente resta al PESO anterior el AJUSTE del PESO actual para acercarnos al mínimo absoluto por el cual la derivada sea 0.

A continuación vamos a considerar otros aspectos importantes una vez entendido cómo es la arquitectura de un modelo de ML y como se produce el entrenamiento del mismo.

3.3 ¿ Es necesario que nuestro modelo tenga función de activación?

¿Por qué utilizar una función de activación y no utilizar el valor continuo obtenido de la función principal de nuestro modelo ? Lo principal es porque solamente podríamos resolver problemas lineales. La función de activación introduce a nuestro modelo la no linealidad, además de evitar el sobreajuste del modelo entre otras razones.

En los casos de clasificación binaria emplearemos la regresión logística y uso de funciones en forma de "S"; sigmoides. Como decimos, se utiliza la función sigmoide como activación EN LA PENÚLTIMA CAPA. Si quieres utilizar varias clasificaciones binarias a la vez entonces utilizas varias neuronas de salida.
En el caso de que nos encontremos ante un problema de clasificación binaria en la etapa final de nuestro modelo emplearemos la función sigmoide: $\sigma(x) = \frac{1}{1+e^{-x}}$ **siendo z la función de la regresión lineal (para poder así acotar la función sigmoide). Además utilizamos un threshold para transformar el valor continuo a discreto.**

Esta función mapea cualquier valor real a un valor entre 0 y 1, lo que la hace útil para tareas como la probabilidad de clasificación en modelos de aprendizaje automático.

En el caso de que nos encontremos ante un problema de clasificación multiclase emplearemos la función Softmax. La función Softmax toma las puntuaciones obtenidas anteriormente y las convierte en probabilidades de pertenecer a una clase u a otra.

**** Softmax toma un vector de valores (generalmente las puntuaciones de salida de la última capa de una red neuronal) y los normaliza en un rango de 0 a 1. Estos valores normalizados pueden interpretarse como probabilidades. Cada número representa la probabilidad de que la entrada pertenezca a una de las clases. Además Softmax amplifica las diferencias entre las puntuaciones. Si una de las puntuaciones es ligeramente más alta que las demás, tras aplicar Softmax, esta diferencia se acentúa, haciendo más evidente la elección de la clase predominante.**

****Diferenciación y Gradientes**: Softmax, además es diferenciable, lo que permite calcular gradientes durante el entrenamiento de la red neuronal. Esto es esencial para el proceso de retropropagación, donde los pesos de la red se ajustan en función del error.**

$$\text{Softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Donde:

- $\mathbf{z}$ es el vector de entrada (puntuaciones o logits) para el cual quieres calcular las probabilidades Softmax. Cada elemento z_i en $\mathbf{z}$ corresponde a la puntuación de una clase específica.
- e es la base del logaritmo natural.
- e^{z_i} es la función exponencial aplicada a cada puntuación z_i.
- K es el número total de clases.
- $\sum_{j=1}^{K} e^{z_j}$ es la suma de las funciones exponenciales de todas las puntuaciones, lo que sirve como un normalizador para que las probabilidades sumen 1.
- $\text{Softmax}(\mathbf{z})_i$ es la probabilidad de que la entrada pertenezca a la clase i después de aplicar la función Softmax al vector $\mathbf{z}$.

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$

En esta fórmula:

- e^{z_i} es la función exponencial de la salida z_i.
- $\sum_{j=1}^{C} e^{z_j}$ es la suma de las funciones exponenciales de todas las salidas.

La función softmax produce valores que suman 1, ya que cada valor se divide por la suma total de las funciones exponenciales. Esto significa que los resultados se pueden interpretar como probabilidades.

En términos prácticos, si tienes un vector de salidas $z = [z_1, z_2, ..., z_C]$, la salida de la capa softmax será otro vector $y = [\text{softmax}(z_1), \text{softmax}(z_2), ..., \text{softmax}(z_C)]$.

La función softmax es útil porque proporciona una salida adecuada para problemas de clasificación multiclase, donde se desea asignar probabilidades a cada clase. En el entrenamiento, se compara la salida de la softmax con las etiquetas reales usando una función de pérdida como la entropía cruzada para ajustar los pesos de la red.

La función softmax se utiliza comúnmente en la capa de salida de una red neuronal, especialmente en el contexto de clasificación multiclase. Cuando se aplica la función softmax a un conjunto de salidas, transforma estas salidas en una distribución de probabilidad, donde cada valor representa la probabilidad de pertenecer a una clase específica. Esto es útil en problemas de clasificación donde hay más de dos clases.

Supongamos que tienes un perceptrón multicapa (MLP) con una capa de salida que tiene C neuronas (donde C es el número de clases en tu problema de clasificación). La función softmax se aplica a las salidas $z_1, z_2, ..., z_C$ de estas neuronas. La salida de la función softmax para la clase i ($1 \leq i \leq C$) se calcula de la siguiente manera:

En nuestro ejemplo del cálculo del precio de la casa utilizamos la función sigmoide, pero al igual que las funciones de coste existen muchas otras dependiendo de las características de nuestro modelo. Vamos a presentar algunas funciones de activación:

Otras de estas funciones de activación son tanh(x): Una de sus limitaciones es que cuando el valor de z es muy grande entonces el gradient descent va muy lento corrigiendo errores, ya que la pendiente va a ser muy pequeña para corregir el error. Por esa razón "nace" otra función que es ReLU. Su desventaja es que cuando z es menor de 0 la pendiente es 0 y el valor siempre va a

ser 0. Para esto nace la función LeakyReLu. De todas formas las funciones x de las hidden layer rara vez obtienen valores negativos como input porque en la práctica utilizar la función ReLU o Leaky ReLU no es relevante. En problemas de clasificación binaria es recomendable utilizar ReLU para las Hidden layer y sigmoide para la output layer. En caso de que estemos ante una problema de clasificación no binaria utilizamos ReLU para las hidden layer y softmax para la output layer.

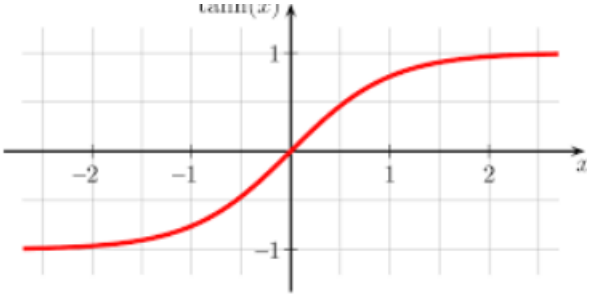
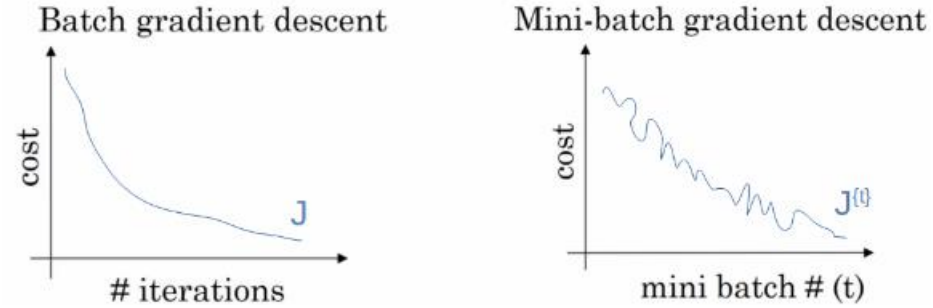


Imagen de ejemplo de la función tanh(z)

3.4 ¿ Podemos utilizar otra función de optimización que no se gradient descent?

Hay muchas funciones de optimización aparte de Gradient Descent:

Mini-Batch Gradient descent: El conjunto de datos se divide en varios subconjuntos y se aplica el gradient descent en cada uno de los bloques en vez de en todos a la vez.



Stochastic Gradient Descent: Se modifica el valor de los parámetros con cada uno de los pares de ejemplos de nuestro modelo en vez de que la actualización se realice en cada vuelta de entrenamiento.

Los algoritmos de optimización que emplean un solo ejemplar de datos a la vez se llaman métodos estocásticos. Son muy útiles para evitar caer en mínimos locales en funciones de costo irregulares. Por el contrario cuando se utilizan para funciones de coste regulares pueden presentar variaciones para los mínimos globales. Existen técnicas para acelerar la actualización del proceso de optimización con respecto al algoritmo de gradiente descendiente obtenido como son momentum=impulso.

Momentum gradient descent: hace una media de las optimizaciones de los últimos mini batch.

Otros algoritmos de optimización estocásticos a parte de momentum (ALGORITMO DE OPTIMIZACIÓN QUE EMPLEAN ÚNICAMENTE UN SOLO EJEMPLAR DE DATOS EN LA ACTUALIZACIÓN) son:

- **AdaGrad: Se produce una variación en la que la tasa de aprendizaje se ajusta individualmente para cada uno de los pesos de un modelo, a través de un escalamiento proporcional al inverso de la raíz cuadrada de la suma de los valores históricos de los gradientes (acmulación de gradientes). Este algoritmo adapta la tasa de aprendizaje a los parámetros, realizando actualizaciones más pequeñas para los parámetros asociados a las características que aparecen con frecuencia y actualizaciones más grandes para los parámetros asociados a las características poco frecuentes. Es muy adecuado para conjuntos de datos muy dispersos.**

- **Adadelta: Se basa en una tasa de aprendizaje adaptativa al igual que AdaGrad, pero es más robusto. Adapta las tasas de aprendizaje con base en una ventana móvil de actualizaciones del gradiente en vez de por acumulaciones de épocas previas. Además no es necesario establecer una tasa de aprendizaje global.**

- **RMSprop: acumula la suma de gradientes utilizando una ventana móvil al igual que AdaDelta con suavizado exponencial además se puede combinar con momentos de Nesterov.**

- **Adam, Nadam y AdaMax: Adam es un algoritmo de tasa de aprendizaje adaptativa. Esta adaptación se hace a través del uso de las normas de las medias móviles del histórico de los gradientes durante cada etapa de entrenamiento así como a través de momentos de primer y segundo orden de estos gradientes. De Adám se desprenden algunas otras variantes entre las que destacan dos: Nadam: esencialmente combinación de Adam con momentos de Nesterov. AdaMax: variante en la que se modifica la norma de los gradientes.**

3.5 Más allá de la función de error y de activación. Acerca de los hiperparámetros.

Algunos hiperparámetros son: Hidden layer, número de neuronas por hidden layer, función de activación, el learning rate, la función de optimización. Se dividen en hiperparámetros del modelo (su arquitectura) o del algoritmo (como por ejemplo como la tasa de aprendizaje).

KERAS TURNER: es una biblioteca que te ayuda a elegir el conjunto óptimo de hiperparámetros para nuestro modelo. Esto se conoce como *tuning de hiperparametros,* de tal forma que hace entrenamientos con varios hiperparametros. Lo que pasa es que no puede probarlo con todos, por lo que nosotros debemos ayudarlo a que haga pruebas entre varios.

Cuando se construye un modelo para realizar *hypertuning*, debe definirse el espacio de búsqueda de hiperparámetros además de la arquitectura del modelo. El modelo que estableces para realizar *hypertuning* se llama *hypermodel*.

Puedes definir un *hypermodel* a través de dos enfoques:

- Implementando una función que construye el modelo
- Creando una subclase la clase `HyperModel` de la Keras Tuner API

También puedes utilizar dos clases predefinidas de `HyperModel`, `HyperXception` y `HyperResNet` para aplicaciones de visión artificial.

En este tutorial, se implementa una función que construye un modelo de clasificación de imágenes. La función devuelve un modelo compilado y utiliza los hiperparámetros que se definan para realizar *hypertuning* del mismo.

```
[ ]: def model_builder(hp):

        # Definición del modelo
        model = keras.Sequential()
        model.add(keras.layers.Flatten(input_shape=(28, 28)))

        # Tunning del número de neuronas de la primera hidden layer
        # Seleccionamos el valor optimo entre 32 y 512
        hp_units = hp.Int('units', min_value = 32, max_value = 512, step = 32)
        model.add(keras.layers.Dense(units = hp_units, activation='relu'))

        # Tunning del ratio de aprendizaje para el algoritmo de optimizacion
        # Seleccionamos el valor optimo entre [0.01, 0.001, 0.0001]
        hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])

        model.compile(
            optimizer=keras.optimizers.Adam(learning_rate = hp_learning_rate),
            loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
            metrics=['accuracy'])

        return model
```

Una vez definida la función anterior, se instancia el Tuner para realizar el *hypertuning*. El *Keras Tuner* tiene cuatro *tuners* disponibles - `RandomSearch`, `Hyperband`, `BayesianOptimization` y `Sklearn`. En este tutorial, se utiliza `Hyperband` (https://arxiv.org/pdf/1603.06560.pdf)

Para instanciar el *tuner* de Hyperband, hay que especificar el *hypermodel*, el objetivo a optimizar y el número máximo de epochs a entrenar (max_epochs).

```
]: tuner = kt.Hyperband(
        model_builder,
        objective = 'val_accuracy',
        max_epochs = 10,
        factor = 3,
        directory = 'test_dir',
        project_name = 'hp_tuning'
    )
```

El algoritmo `Hyperband` utiliza la asignación adaptativa de recursos y la detención temprana para converger rápidamente en un modelo de alto rendimiento. El algoritmo entrena un gran número de modelos durante unos pocos epochs y lleva a la siguiente ronda sólo la mitad de los modelos de mayor rendimiento. Hyperband determina el número de modelos a entrenar mediante la fórmula $1 + log_{factor}(max\_epochs)$ y redondea el resultado al entero más cercano.

Implementación de la librería Keras Turner

Batch Gradient descent:Procesa el conjunto de datos completo en cada iteración. Función de optimización más lenta.Suele converger y alcanzar el mínimo de la función de error.

Mini-Batch Gradient descent: Procesa un subconjunto del conjunto de datos en cada iteración. Función de optimización más rápida que Butch Gradient Descent. Suele converger y alcanzar el mínimo de la función de error de manera menos directa.

Stochastic Gradient Descent: Procesa un ejemplo del conjunto de datos en cada iteración. Función de optimización más rápida de las tres. No suele converger y alcanzar el mínimo de la función de error, aunque alcanza un valor aceptable.

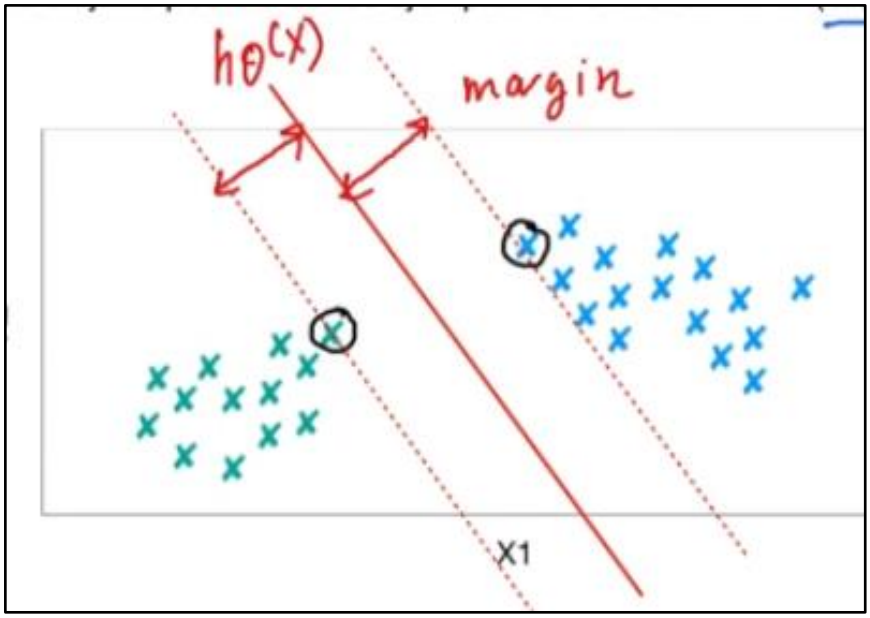
Hiperparámetros más importantes

Learnig rate: Es probablemente el parámetro más importante. La estrategia de selección del learnig rate más utilizada consiste en comenzar con un valor muy pequeño (10^-5) e incrementarlo hasta un valor grande (10).
Función de optimización: tal y como se ha mostrado en secciones anteriores existen varias funciones de optimización que pueden ayudar a mejorar el resultado del algoritmo.
Batch-size: un tamaño de batch elevado permite que componentes hardware como la GPU sean más eficientes procesando los ejemplos de entrenamiento. Sin embargo, en la práctica, entrenar con un batch-size elevado puede conducir a inestabilidades en el entrenamiento que generen un modelo que no sea capaz de generar bien. Como norma general utilizar baches de tamaños entre 2 y 32 suelen ser la selección de preferencia.
Función de activación: se han discutido en secciones anteriores las funciones de activación más populares y el efecto que produce cada una de ellas en el modelo en general, la función de activación relu suele ser la selección preferencia.

4. Otros tipos de modelos

A continuación se presentan algunos modelos de ML empleados para clasificación +/- regresión:

El límite de decisión intenta que sea lo más alejado de los ejemplos de entrenamiento (large margin classification). El límite de decisión lo hace a partir de los puntos de los diferentes grupos más cercanos entre sí, que se van a llamar support vectors.



En esta imagen se representa la función que construye nuestro modelo (línea roja) para separar ambos grupos. Lo hace añadiendo un margen de seguridad a ambos lados que va permitir darle verticalidad u horizontalidad a nuestro modelo para separar mejor ambos grupos. Rodeamos por un círculo negro se señalan los support vectors.

SVM: Hard Margin Classification: El método que se acaba de presentar obliga de manera estricta a que todos los ejemplos de entrenamiento de cada una de las clases se encuentren detrás de la línea de puntos.
Este método tiene dos problemas fundamentales: sólo funciona con conjuntos de datos linealmente separables. Es muy sensible a datos anómalos.

$$\textbf{SVM: } h_\theta(x) = \begin{cases} 1 & si\ \theta_0 + \theta_1 x \geq 0 \\ 0 & en\ caso\ contrario \end{cases}$$

Su función es :

SVM KERNELS: REGRESIÓN POLINÓMICA y Gaussian Kernel: Permite crear modelos no lineales.

5. Otras consideraciones de nuestro modelo de ML

Vamos a señalar brevemente lo que es el subajuste y el sobreajuste (overfitting) de nuestro modelo. El subajuste se produce cuando no es capaz de predecir o clasificar con unos niveles de precisión satisfactorios el conjunto de datos de validación y de prueba. El sobreajuste se produce porque nuestro modelo se ajusta demasiado bien al conjunto de datos de entrenamiento pero falla con los datos de prueba.

Razones por las que se presenta subajustes son: modelo con poca complejidad, modelo con número de capas ocultas insuficiente, alto grado de regularización en el modelo, escaso número de épocas, no hubo buena selección de hiper parámetros.

El sobreajuste es un problema muy habitual en el momento que nosotros probamos nuestro modelo. Existen diferentes técnicas para paliarlo. Cómo mejoramos el overfitting?Aumentamos el conjunto de datos, reducimos el número de capas y neuronas, regularizamos, hacemos dropout, data augmentation.
El data augmentation se basa en aumentar el conjunto de datos pero basándose en los que ya tenemos y realizando algunas modificaciones sobre los mismos datos (ejemplo; darle la vuelta a la misma imagen). El early stopping consiste en parar el entrenamiento sin llegar a realizar todo el entrenamiento. Esto es porque el overfitting suele darse en las últimas epochs.
Otra técnica, como hemos señalado, es la regularización. La regularización consiste en agregar una penalización en los diferentes parámetros del modelo para reducir su libertad. Esto hace que el que modelo se ajuste al ruido de los datos de entrenamiento y mejorará las capacidades de generalización del mismo. Para ello mantiene todas las características, pero reduce la magnitud de los parámetros W., funcionando muy bien cuando tenemos muchas características ligeramente útiles.

La técnica de regularización más comúnmente empleada en la práctica es el decaimiento del valor de los pesos. Fórmulas matemáticas:

L1: Penaliza los pesos en proporción a la suma de los valores absolutos de estos ayudando a ponderar los pesos de las características irrelevantes o apenas relevantes a valores cercanos a cero reduciendo el sobre ajuste. Tener en cuenta que en modelos sencillos su aplicación puede ser negativa.

$$J(\theta) = \text{Costo sin regularización}(\theta) + \frac{\lambda}{m} \sum_{j=1}^{n} |\theta_j|$$

- $J(\theta)$ es la función de costo total.
- El término $\text{Costo sin regularización}(\theta)$ representa la función de costo original del algoritmo sin regularización. Por ejemplo, en la regresión lineal, este término podría ser el costo del error cuadrático medio.
- θ son los parámetros del modelo.
- λ es el parámetro de regularización L2, también conocido como el término de regularización. Controla cuánto quieres penalizar los valores grandes de los parámetros.
- m es el número de ejemplos de entrenamiento.
- $\sum_{j=1}^{n} \theta_j^2$ es la suma de los cuadrados de los parámetros del modelo, excluyendo usualmente el término de intercepción o sesgo.

L2: penaliza los pesos en proporción a la suma de los cuadrados de los pesos. Dropout: Consiste en establecer aleatoriamente las salidas de las neuronas ocultas a un valor igual a cero con cierta frecuencia por una tasa de probabilidad. Esto provoca la eliminación de neuronas en las diferentes capas según una probabilidad que le demos. Además esta desaparición se va alternando según las actualizaciones del algoritmo en los sucesivos entrenamientos. Esto evita que las neuronas de las siguientes capas dependan mucho de ciertas neuronas que se han superespecializado y que estén desarrollando overfitting.

$$J(\theta) = \text{Costo sin regularización}(\theta) + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

La normalización de los datos según la varianza de una características o más de forma que los datos se agrupen más y va a permitir también reducir el sobreajuste.

1. **Estabilidad Numérica:**
   - La normalización ayuda a estabilizar los cálculos numéricos en algoritmos de optimización. Algunos algoritmos, como el descenso del gradiente, pueden converger más rápidamente en datos normalizados, ya que los valores de los parámetros no están en escalas muy diferentes.
2. **Independencia de la Escala:**
   - Al normalizar, los datos se escalan a un rango común, lo que elimina la dependencia de la escala original de las características. Esto es crucial en algoritmos basados en distancias, como el k-vecinos más cercanos (KNN) o el método de máquinas de soporte vectorial (SVM), donde la escala de las características puede afectar significativamente los resultados.
3. **Mejora de la Convergencia:**
   - En algoritmos iterativos, como el descenso del gradiente, normalizar los datos puede conducir a una convergencia más rápida y a un rendimiento más estable, ya que los pasos de actualización de los parámetros son más consistentes.
4. **Impacto Equitativo de las Características:**
   - La normalización garantiza que todas las características contribuyan de manera equitativa al modelo, independientemente de su magnitud original. Sin normalización, las características con valores más grandes pueden dominar el impacto en el modelo.

6. Keras y TensorFlow

Keras APIs

- **Sequential API**: Un modelo secuencial es apropiado para una simple pila de capas donde cada capa tiene exactamente un tensor de entrada y uno de salida
- La Sequential API **no** es apropiada cuando:
  - El modelo tiene múltiples entradas o múltiples salidas
  - Cualquiera de sus capas tiene múltiples entradas o múltiples salidas
  - Necesitas compartir las capas
  - Quieres una topología no lineal (por ejemplo, una conexión residual, un modelo de varias ramas)
- Para los casos en los que se requiera alguna de las características anteriores se utiliza la **Functional API** de Keras

Pasos en la implementación de una Red Neuronal Artificial en Keras

1. Define tu conjunto de datos de entrenamiento: vectores de entrada y de salida
2. Define la arquitectura de la Red Neuronal Artificial
3. Configura el proceso de aprendizaje mediante la seleccion de una funcion de error, una funcion de optimizacion y diferentes metricas para monitorizar el proceso
4. Entrena la RNA con tu conjunto de datos de entrenamiento mediante el uso del metodo ***fit()***

¿Para qué sirve Tensorflow?

- Su core es muy similar a Numpy con soporte para GPUs
- Soporta computación distribuida a lo largo de multiples servidores o sistemas
- Incluye un compilador que optimiza los cálculos computacionales para aumentar su velocidad y reducir su consumo de memoria
- Proporciona diferentes librerías de alto nivel que facilitan diferentes actividades a la hora de resolver un problema de Deep Learning. La más importante es tf.keras, pero existen otras como tf.image, tf.audio o tf.data.
- Actualmente soporta la ejecución de sus operaciones en CPUs, GPUs y TPUs.

Existen formas de guardar el progreso de un algoritmo de IA. El más utilizado es el H5. Permite guardar y cargar modelos en formato de archivo H5. Es altamente eficiente y compatible entre diferentes plataformas y permite también retomar el entrenamiento de modelos de manera rápida y sencilla.

5. Aprendizaje no supervisado

El aprendizaje automático no supervisado es la tarea de aprendizaje automático que consiste en inferir una función que describe la estructura de un conjunto de datos sin etiquetar (es decir, datos que no se han clasificado ni categorizado)

En aprendizaje no supervisado, nos devuelve el conjunto de datos dividido por grupos. Es entonces cuando un analista de datos se encarga de decir que representan cada uno de los grupos.

Algoritmo ML de Clustering;(algoritmo de tipo no supervisado):
Su objetivo es agrupar conjunto de datos sin etiquetar en subconjuntos o clusters. Se basan en el concepto de agrupación de datos por PROXIMIDAD evaluando la cercanía de los puntos. Existen varias técnicas para evaluar esta proximidad.

- **KMEANS: Se basa en la distancia euclídea entre los puntos. Permite escalar muy bien un conjunto de datos muy grande. Primeramente introduce dos datos aleatorios denominados clusters-centroides. A continuación lo que hace es calcular la distancia de cada uno de los datos con respecto a estos centroides dividiéndolos en clusters.**

  Posteriormente se mueven los centroides al punto medio de los datos que conforman su cluster repitiéndose este proceso hasta que es capaz de dividir en los grupos pertinentes.

Limitaciones del algoritmo KMEANS: Se debe incluir el número de clusters que genera el algoritmo. Si son datos etiquetados, elegir el número de clusters como un valor entre 1 y 3 veces el número de

etiquetas existentes. Hay que aplicar normalización al conjunto de datos. No debe utilizarse KMEANS con datos categóricos a los que se le aplica one hot en coding. Por el contrario, deben tratar de codificarse estas características como múltiple binary. Pierde eficiencia en conjuntos de datos con muchas dimensiones. Una práctica frecuente es reducir las dimensiones del conjunto de datos mediante el uso de PCA o SVD. Funciona mejor si los centroides iniciales se eligen aleatoriamente, esto provoca que los resultados puedan cambiar dependiendo de donde se inicialicen. Asume que los clusters son esféricos. No funciona correctamente en distribuciones de datos no esféricas.

- **DBSCAN; otra técnica de clustering: Interna resolver el problema de que los datos debían de seguir una distribución esférica. /////DBSCAN recibe dos parámetros: epsilon; define el radio alrededor de un ejemplo del conjunto de datos apartir del cual buscar ejemplos vecinos y minPoints; mínimo número de vecinos necesarios para formar un cluster.**

  Además cada ejemplo en el conjunto de datos es clasificado como: Core point (ejemplos que tienen al menos un número minPoints de ejemplos dentro de epsilon) Border Pints (no son Core points pero se encuentran en el radio de uno) Noisse points (No son Core points ni border points)

  Limitaciones: No funciona correctamente cuando los clusters del conjunto de datos tienen diferentes densidades. La selección de los parámetros Epsilon y minPoints es muy importante para el correcto funcionamiento del algoritmo. No se comporta adecuadamente en conjuntos de datos con muchas dimensiones puntos no se debe alterar las densidades del conjunto de datos al dividirlo en subconjuntos (entrenamiento, validación y pruebas)

Existen diferentes formas de evaluar los resultados de un método de ML de aprendizaje no supervisado como es el clustering. Si tuviéramos el conjunto de datos etiquetados y lo que hemos hecho nosotros es no darle la etiquetas las métricas utilizadas son: Homogeneidad: grado en el que los clusters contienen miembros de una clase determinada. Plenitud: el grado por el que todos los miembros de una clase determinada han sido asignados al mismo cluster. V- measure: métrica basada en la homogeneidad y en la plenitud que representa la precisión de las operaciones de agrupamiento. Pureza: para calcular la pureza, cada grupo se asigna a la clase que es más frecuente en el grupo, y luego la precisión de esta asignación se mide contando el número de documentos asignados correctamente y dividiendo por el número de elementos.

Para evaluar conjunto de datos no etiquetados entonces se utilizan dos métricas principalmente: Silhouette coefficient
Otro es el índice Calinski-Harabaz index

Pasemos a tratar otros tipos de modelos ML de aprendizaje no supervisado no basados en clustering

2. REDES NEURONALES CONVOLUCIONALES

La red neuronal convolucional (CNN) son el tipo de redes más utilizadas para algoritmos de IA en visión computerizada y tratamiento de imágenes. Son más complejas que las neuronas vistas anteriormente.

Las CNN a diferencia que las redes neuronales densas permiten reconocer características aisladas de la imagen como bordes, aristas… y son capaces de reconocer estructuras espaciales.

¿Convolución?: Operación matemática que genera una tercera función a partir de la interacción de dos anteriores. Para ello una de las funciones se desplazada de forma invertida una "x" distancia sobre la otra. A continuación se calcula el área bajo la curva del punto superpuesto (multiplicación de ambos puntos f(x) y g(x)) entre ambas funciones.

Formalmente, la convolución de dos funciones f y g se define como:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

Donde:

- $f * g$ denota la convolución de f y g.
- t es la variable de integración.
- τ es otra variable que se usa para el desplazamiento.
- La integral se toma sobre todo el dominio de las funciones, que a menudo es todo el conjunto de números reales.

La integral está expresada en términos infinitesimales pero en el apartado de las CNN se aplican en términos discretos —> convoluciones discretas.
Consiste, como hemos comentado, en el desplazamiento de un tensor sobre otro de forma invertida y proceder a la multiplicación de los elementos de ambos vectores. El desplazamiento "x" que hace un vector sobre otro es muy importante; se denomina en la jerga STRIDE y va a determinar el tamaño final del vector.

El tensor que se desplaza sobre el otro recibe el nombre de Filtro o Kernel; ESTE TIPO DE FILTRO ES EL QUE PERMITE CENTRARSE EN UNA CARACTERÍSTICA DE LA IMAGEN COMO EL CONTORNO, SOMBRA….. En Python existe la librería Scipy para hacer la convolución de dos tensores de 2 orden. El producto resultante de ambos vectores recibe el nombre de mapa de características.

Existen otras operaciones importantes en relación con las convoluciones como son el padding; consisten llenar de ceros alrededor del mapa de características de tal forma que tengamos un tensor final de tamaño similar al inicial; es decir previo a la convolución. Otras operaciones son el max pooling y el avarage pooling que consisten en resaltar el máximo o la media respectivamente en cada uno de los desplazamientos.

En el entrenamiento de las redes neuronales convolucionales estas se entrenan para aprender mapas de características, es decir elementos significativos distintivos de la imagen.

La anatomía de una CNN consiste en: 1 *Entrada*: característica del tensor que alimenta la red. 2 *Base*: Capas y número de neuronas y sus parámetros. Encargadas de extraer los mapas de características. 3. *Cabeza*. Capas destinadas a realizar una tarea en concreto. Pueden por ejemplo tener una función de activación tipo SoftMax para clasificación.
—----------

La visión computerizada incluye modelos de ML y cuyas secciones incluye: Clasificación de objetos- detección de objetos (detecta diferentes objetos y su localización en la imagen) y segmentación (divide imágenes en múltiples segmentos, segmentos y regiones según a la clase que pertenecen).
La segmentación la podemos dividir en dos apartados:

Segmentación semántica en la que se clasifican los píxeles de la imagen en etiquetas previamente descritas.

La segmentación por instancias es una extensión de la segmentación semántica ya que además de detectar diferencia a cada uno de los segmentos. Los modelos más empleados en segmentación son: FCN- SegNet-U-Net-Pyramid Scene Parsing Network, PSPNet - Mask R-CNN.

3. Otras redes y aspectos

Redes neuronales recurrentes:

Se utilizan cuando las características de los datos presentan relaciones de dependencia y orden entre ellas, es decir, los datos son secuenciales. Como decimos, los datos secuenciales contienen elementos ordenados en un dominio por lo que un par de datos tienen directa relación con los pares de datos anteriores. El timestep hace referencia al orden secuencial entre los datos. Las redes neuronales recurrentes se centran en el modelado del dominio secuencial de un conjunto de datos y su característica distintiva es que poseen la capacidad de enviar información a través de bucles y pasos temporales para captar la información almacenada en la secuencia de datos anterior al actual. Son por tanto capaces de recordar las entradas anteriores y utilizar esta información para procesar las entradas futuras. La unidad recurrente puede considerarse como una célula de memoria que guarda información de entradas anteriores. Estas redes sirven para pronosticar, procesado de lenguaje natural, análisis de texto y audio.

—----------

NiMARE:

NiMARE (Neuroimaging Meta- Analysis Research Environment) *S*e trata de una biblioteca-código abierto de python que ofrece un entorno de investigación de metanálisis de neuroimágenes gracias a implementar una variedad de algoritmos de metanálisis, incluyendo análisis de metadatos basados en coordenadas y en imágenes, anotación automatizada, decodificación funcional y modelado de coactivación meta-analítica.

Forma parte también de un creciente ecosistema de análisis de neuroimagen como es Neurosynth, NeuroVault, NeuroQuery y PyMARE.

NiMARE funciona con las versiones de Python 3.6 y superiores, y se puede instalar fácilmente con pip. Está diseñado para ser modular y orientado a objetos, con una interfaz que imita a las bibliotecas populares de Python, incluyendo scikit- learn (implementa una gran cantidad de algoritmos de aprendizaje automático) y nilearn.

Presenta varios módulos: nimare.meta, nimare.correct, nimare.annotate, nimare.decode y nimare.workflows.

Funciona a través de un servicio de APIs. Los datos que se van a manipular se encuentran dentro de la clase DATASET y hay dos formas de operar el dataset: 1) Uso de la clase Transformer: Realiza alguna transformación en el dataset.
2) Estimator aplican un algoritmo metaanalítico a un Dataset y devuelven un conjunto de imágenes estadísticas almacenadas en una clase contenedora MetaResult.

Se compone actualmente de 14 módulos:

- **nimare.dataset define la clase Dataset.**
- **nimare.meta incluye Estimadores para métodos de metaanálisis basados en coordenadas e imágenes.**
- **nimare.results define la clase MetaResult, que almacena mapas estadísticos producidos por metaanálisis.**
- **nimare.correct implementa clases Corrector para la corrección de comparaciones múltiples de errores familiares (FWE) y tasa de descubrimiento falso (FDR).**
- **nimare.annotate implementa una variedad de métodos de anotación automatizada, incluyendo la asignación latente de Dirichlet (LDA) y la asignación latente de Dirichlet de correspondencia generalizada (GCLDA).**
- **nimare.decode implementa una serie de algoritmos de decodificación y codificación funcional metaanalítica.**
- **nimare.io proporciona funciones para convertir estructuras de conjuntos de datos metaanalíticos alternativos, como archivos de texto de Sleuth o Conjuntos de Datos de Neurosynth, al formato de NiMARE.**

- nimare.transforms **implementa una variedad de transformaciones espaciales y de tipo de datos, incluyendo una función para generar nuevas imágenes en el Conjunto de Datos a partir de tipos de imágenes existentes.**
- nimare.extract **proporciona métodos para obtener Conjuntos de Datos y modelos a través de Internet.**
- nimare.generate **incluye funciones para generar datos para pruebas internas y validación.**
- nimare.base **define varias clases base utilizadas en todo el paquete.**
- nimare.stats y nimare.utils **son módulos para funciones estadísticas y utilitarias genéricas.**

¿Qué datos utiliza?

El metaanálisis basado en coordenadas (CBMA) es actualmente el método más popular para el metaanálisis de neuroimagen, dado que la mayoría de los artículos de fMRI actualmente informan sus hallazgos como picos de conglomerados estadísticamente significativos en un espacio estándar y no liberan mapas estadísticos sin umbral.
Estos picos indican dónde se encontraron resultados significativos en el cerebro y, por lo tanto, no reflejan una estimación del tamaño del efecto para cada prueba de hipótesis (es decir, cada voxel) como se esperaría en un metaanálisis típico. Como tal, no se pueden aplicar los métodos estándar para el metaanálisis basado en el tamaño del efecto. En las últimas dos décadas, se han desarrollado varios algoritmos para determinar si los picos convergen con el fin de identificar ubicaciones de activación consistente o específica asociada con una hipótesis dada.

Los métodos basados en kernel evalúan la convergencia de coordenadas en estudios mediante la convolución de focos con un kernel espacial para producir mapas de activación modelados específicos de cada estudio, luego combinan esos mapas de activación modelados en un mapa por muestra, que se compara

con una distribución nula para evaluar la significación estadística a nivel de vóxel. Los kernels de CBMA están disponibles como KernelTransformers en el módulo nimare.meta.kernel. Actualmente, hay tres kernels estándar disponibles: MKDAKernel, KDAKernel y ALEKernel. Cada clase se puede configurar con ciertos parámetros cuando se inicializa un nuevo objeto. Por ejemplo, MKDAKernel acepta un parámetro r, que determina el radio de las esferas que se crearán alrededor de cada coordenada de pico. El análisis de densidad de Kernel multinivel (MKDA) es un método basado en Kernel que convoluciona cada pico de cada estudio con una esfera binaria de un radio establecido. Estos mapas binarios específicos de pico se combinan luego en mapas específicos de estudio tomando el valor máximo para cada voxel

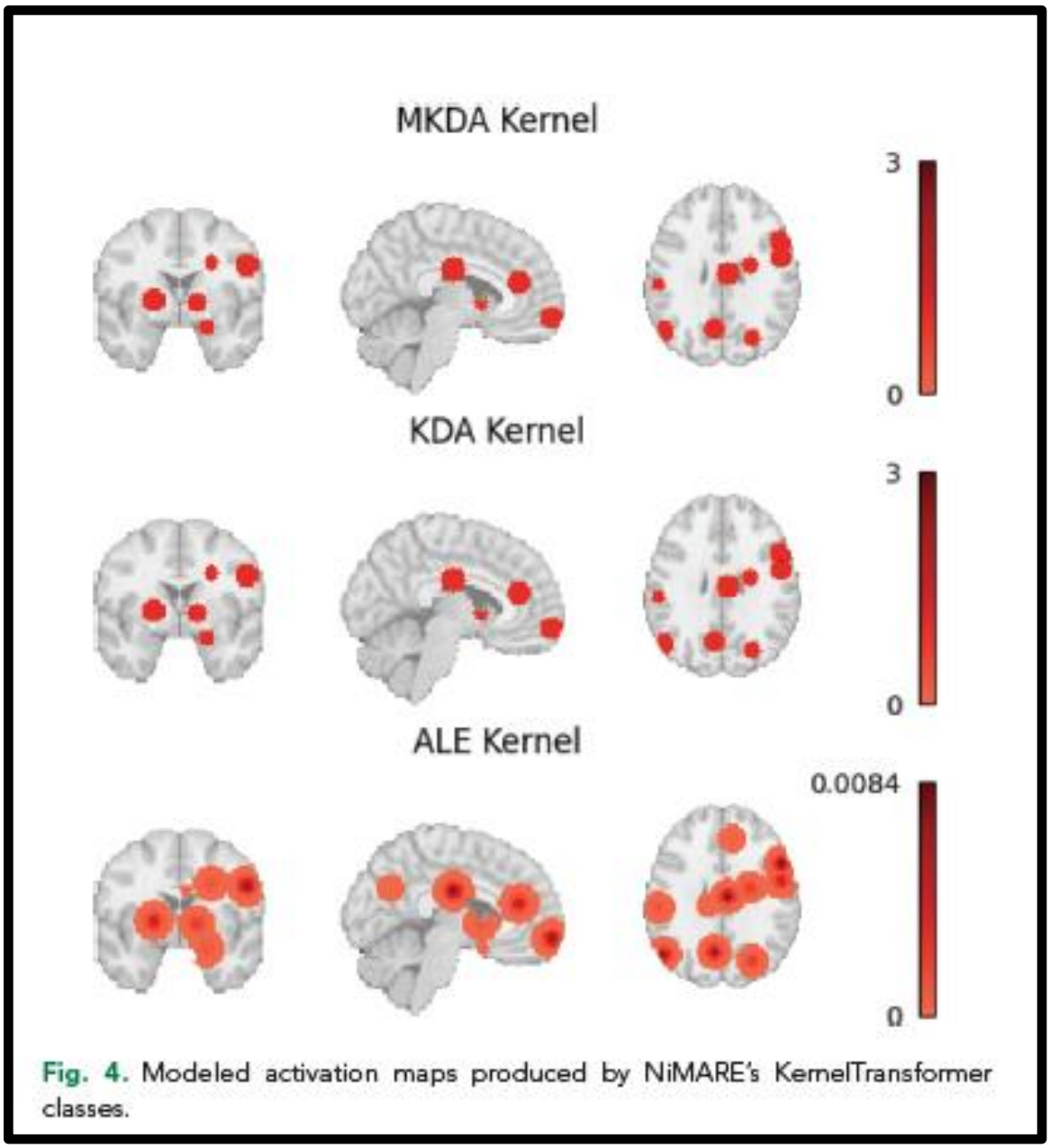


Fig. 4. Modeled activation maps produced by NiMARE's KernelTransformer classes.

La estimación de probabilidad de activación (ALE) evalúa la convergencia de picos en estudios mediante la generación de un mapa de activación modelado para cada estudio, en el que cada uno de los picos del experimento se convoluciona con una distribución gaussiana 3D determinada por el tamaño de muestra del experimento, y luego combinando estos mapas de activación modelados en estudios en un mapa ALE, que se compara con una distribución nula empírica para evaluar la significación estadística a nivel de voxel.

La estimación específica de probabilidad de coactivación (SCALE) es una extensión del algoritmo ALE desarrollado para análisis de modelado de coactivación metaanalítica (MACM). En lugar de comparar la convergencia de focos dentro de la muestra con una distribución nula derivada bajo la suposición de aleatoriedad espacial dentro del cerebro, SCALE evalúa si la convergencia en cada voxel es mayor que en la literatura general.

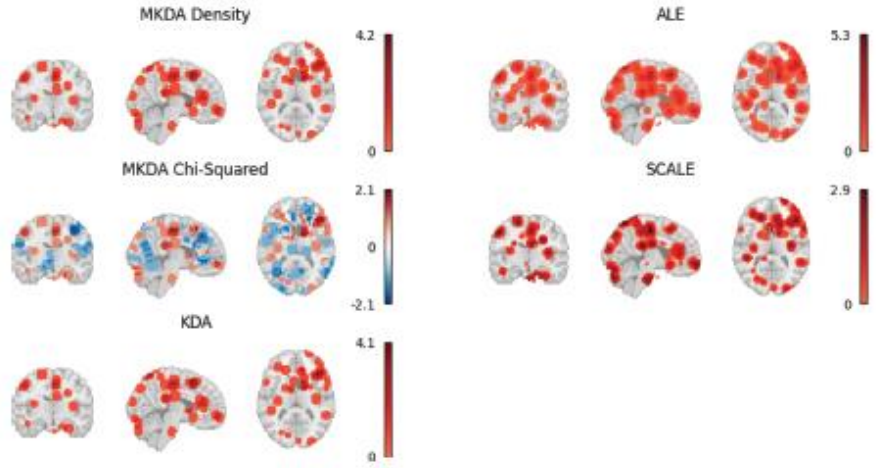


Fig. 5. Thresholded results from MKDA density, KDA, ALE, and SCALE meta-analyses.

Los métodos de metaanálisis basados en imágenes (IBMA) realizan un metaanálisis directamente en imágenes cerebrales (ya sea de todo el cerebro o parciales) en lugar de picos extraídos. En teoría, IBMA es superior a CBMA en prácticamente todos los aspectos, ya que la disponibilidad de estimaciones de parámetros y varianza a nivel de análisis en todos los voxels analizados permite a los investigadores utilizar el conjunto completo de técnicas estándar de metaanálisis, en lugar de recurrir a métodos basados en Kernel u otros métodos que requieren suposiciones espaciales adicionales. En principio, dadas un conjunto de mapas que no contienen valores faltantes (es decir, donde hay k pares válidos de estimaciones de parámetros y varianza en cada voxel), se puede realizar una versión a nivel de voxel de cualquier método de metaanálisis estándar o meta-regresión comúnmente utilizado en otros campos biomédicos o de ciencias sociales. En la práctica, la utilidad de los métodos de IBMA ha sido históricamente bastante limitada, ya que los mapas estadísticos no umbralizados no estaban disponibles para la gran mayoría de los estudios de

neuroimagen. Sin embargo, la introducción y rápida adopción de NeuroVault, una base de datos de imágenes estadísticas no umbralizadas, ha hecho que el metaanálisis basado en imágenes sea cada vez más viable.

Las bases de datos utilizadas combinan resultados de estudios de neuroimagen, ya sea representados como coordenadas de activaciones máximas o imágenes estadísticas sin umbral, con datos importantes sobre los estudios, como información sobre las muestras adquiridas, estímulos utilizados, análisis realizados y constructos mentales que se postulan como manipulados.

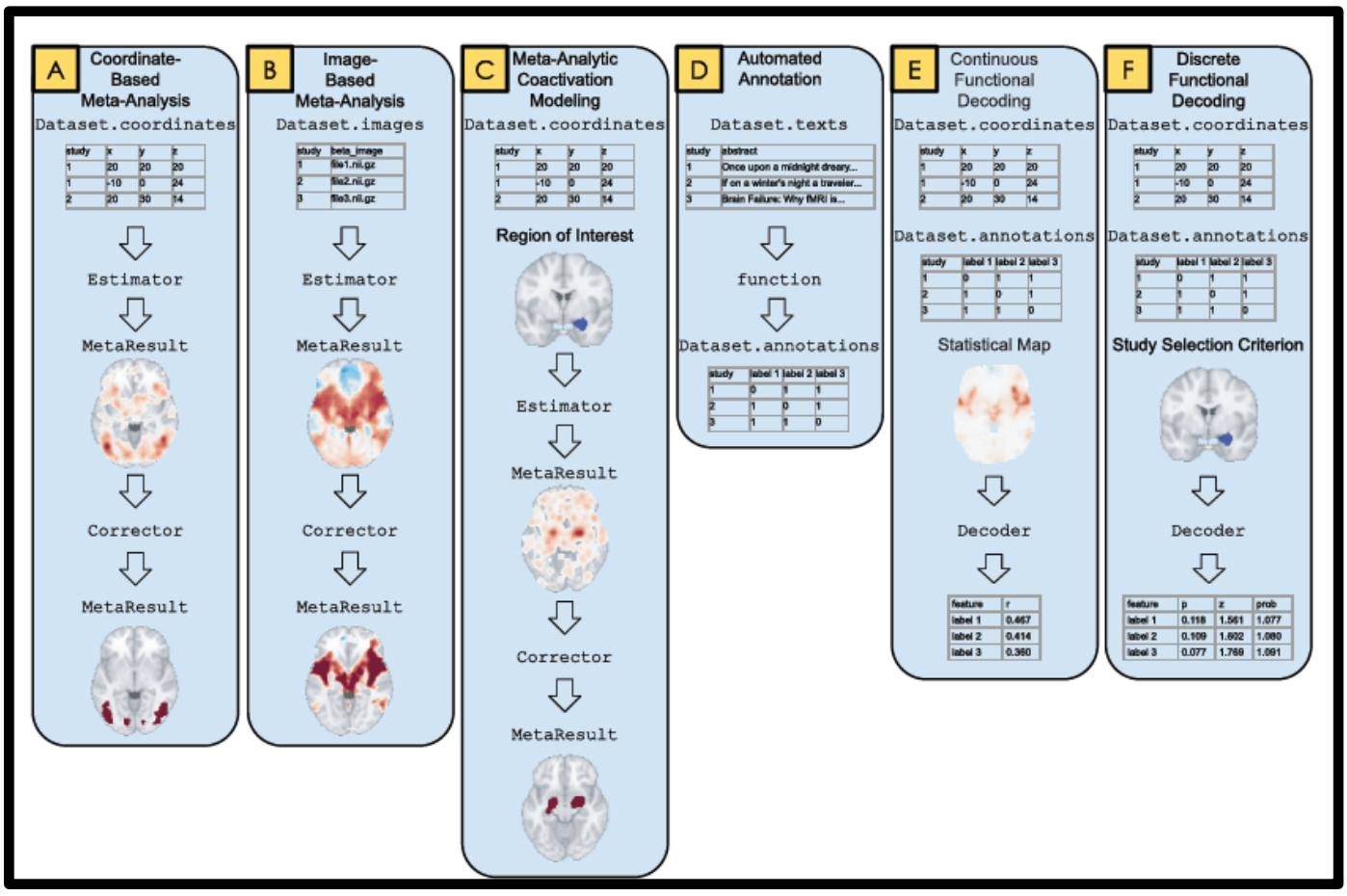
Las dos bases de datos metaanalíticas basadas en coordenadas más populares son BrainMap y Neurosynth, mientras que la base de datos basada en imágenes más popular es NeuroVault.

Importante comprender este tipo de datos: Las ontologías para la neurociencia cognitiva definen los estados mentales o procesos que se postulan como manipulados o medidos en experimentos, y también pueden incluir detalles de dichos experimentos. Algunas de estas ontologías están muy bien definidas, como las taxonomías generadas por expertos diseñadas específicamente para describir sólo ciertos aspectos de los experimentos y las relaciones entre elementos dentro de la taxonomía, mientras que otras están más vagamente definidas, en algunos casos simplemente construyendo un vocabulario basado en los términos comúnmente utilizados en artículos de neurociencia cognitiva.

Por ejemplo, BrainMap se basa en anotadores expertos para etiquetar las comparaciones individuales dentro de los estudios según su ontología interna desarrollada, la Taxonomía de BrainMap. Este enfoque es probablemente menos ruidoso que un método de anotación automática que utiliza el texto de los artículos o los resultados de imágenes para predecir el contenido.

Una base de datos anotada manualmente como BrainMap estará sesgada por las subáreas dentro de la literatura que están anotadas.

Neurosynth utiliza una combinación de web scraping y minería de texto para recolectar de forma automática estudios de neuroimagen de la literatura y anotarlos basándose en la frecuencia de términos dentro de los resúmenes de los artículos. Como resultado de su enfoque automatizado relativamente rudimentario, Neurosynth tiene sus propias limitaciones. En primer lugar, Neurosynth no puede delinear comparaciones individuales dentro de los estudios y, en consecuencia, utiliza el artículo completo como su unidad de medida, a diferencia de BrainMap



Representación gráfica de las herramientas y métodos implementados en NiMARE. Este diagrama describe seis de los casos de uso más comunes para NiMARE. (A) El Metaanálisis Basado en Coordenadas (CBMA) se realiza creando un Conjunto de Datos NiMARE con información de coordenadas almacenada en el atributo Dataset.coordinates, que luego se utiliza en un Estimador CBMA. Esto produce un objeto MetaResult con mapas estadísticos,

que luego se pueden usar en un objeto Corrector para la corrección de comparaciones múltiples. Una vez que se ha ajustado el Corrector, producirá una versión corregida del objeto MetaResult, que contiene mapas estadísticos actualizados. (B) El Metaanálisis Basado en Imágenes (IBMA) funciona de manera similar a CBMA, excepto que los Estimadores IBMA utilizan mapas estadísticos almacenados en el atributo Dataset.images. (C) El Modelado de Coactivación Metaanalítica (MACM) utiliza una región de interés para seleccionar estudios basados en coordenadas dentro de un Conjunto de Datos, después de lo cual se realiza el flujo de trabajo estándar de CBMA. (D) La Anotación Automatizada infiere etiquetas a partir de datos textuales (y a veces otros) asociados al Conjunto de Datos, tal como se almacena en el atributo Dataset.texts. Las funciones de anotación producen etiquetas que pueden integrarse en el Conjunto de Datos como el atributo Dataset.annotations. (E) La decodificación funcional de mapas estadísticos continuos funciona de manera similar a la decodificación discreta, en el sentido de que el Conjunto de Datos de entrada debe tener atributos de coordenadas y anotaciones. El Conjunto de Datos, junto con un mapa estadístico sin umbral para decodificar, se proporciona al objeto Decodificador, que luego produce medidas de similitud o asociatividad con cada etiqueta. (F) La decodificación funcional de entradas discretas aplica un criterio de selección a un Conjunto de Datos con atributos de coordenadas y anotaciones, utilizando un objeto Decodificador. El algoritmo de decodificación producirá medidas de similitud o asociatividad con cada etiqueta en las anotaciones.

4. Bibliografía

- Rodríguez-Sánchez, A. E. (2023). Redes neuronales artificiales: Principios y aplicaciones.

- Torres, J. (Año de publicación). Python Deep Learning: Introducción práctica con Keras y TensorFlow 2. Marcombo.

- Hernández, S. (2023). Machine Learning y Data Science: Curso Completo con Python. Udemy.

- Salo, T., Yarkoni, T., Nichols, T. E., Poline, J.-B., Bilgel, M., Bottenhorn, K. L., Eickhoff, S. B., Jarecka, D., Kent, J. D., Kimbler, A., Nielson, D. M., Oudyk, K. M., Peraza, J. A., Pérez, A., Reeders, P. C., Yanes, J. A., & Laird, A. R. (2022). NiMARE: Neuroimaging Meta-Analysis Research Environment. Florida International University, FL, USA.