Introducción a la Inteligencia Artificial

Un recorrido para quienes quieren comprender cómo trabaja una “máquina que piensa”

Por **Javier Surasky**

Coordinador del Centro de Estudios en Inteligencia Artificial y relaciones Internacionales (CIARI)

Documentos de Trabajo

Documentos deTrabajo Nº 34 – Octubre 2025 ISSN2344-956X

Publicación de Actualización Continua, del Instituto de Relaciones Internacionales (IRI), Facultad de Ciencias Jurídicas y Sociales, Universidad Nacional de La Plata.
Calle 48, Nº 582, piso 5º. La Plata, Provincia de Buenos Aires.
iri@iri.edu.ar www.iri.edu.ar

Esta publicación fue diagramada en el Instituto de Relaciones Internacionales
por **DCV Juana Alvarez Eiras**

Tabla de contenido

Introducción general

Este trabajo está orientado a quienes buscan tener una primera aproximación a la Inteligencia Artificial (IA) que vaya más allá de su simple uso, pero sin pretensiones de convertirse en expertos en el tema y, especialmente, para quienes utilizan la IA en investigaciones habiéndose formado en campos académicos con escasa formación en matemáticas. No se trata de un trabajo académico lleno de citas y discusiones, sino de una aproximación más orientada a la difusión, pero sin perder seriedad.

Es un paso adelante para quienes alguna vez, o regularmente, trabajan con modelos de IA como ChatGPT, Gemini, Claude o Perplexity, o han creado imágenes y videos, o realizado presentaciones de Power Point, con un "asistente virtual" que utiliza IA.

Nos va a permitir entender qué es lo que ocurre cuando damos una instrucción, cuando escribimos un *prompt*, de manera algo más profunda. ¿Qué mecanismo se activan cuando pedimos algo a una IA? ¿Cómo se ha entrenado al sistema para que produzca ese resultado que vemos? ¿Por qué procesos de optimización ha pasado el modelo antes de llegar a nuestras pantallas? ¿Cuánto podemos entender de lo que ocurre dentro de la propia IA cuando se pone en movimiento para respondernos o hacer una tarea para alguien? ¿Qué tiene de particular la IA generativa? ¿Y el Tiny ML?

Iremos dando respuesta a esos interrogantes mirando la IA desde adentro, reduciendo a un mínimo las referencias a operaciones matemáticas y lógicas complejas.

En gran medida, estas páginas son resultado de haber realizado el curso sobre Introducción a la IA dictado por la Facultad de Informática de la Universidad nacional de La Plata entre agosto y septiembre de 2025, como parte de su oferta de formación de posgrado, por lo que es justo agradecer a cada uno de quienes lo dictaron: César Estrebou. Facundo Quiroga, Franco Ronchetti, Gastón Ríos, Pedro Dal Bianco y Santiago Ponte Ahón (por orden alfabético de sus nombres).

En un recorrido dividido en cuatro partes, comenzaremos por introducirnos en el mundo de la IA (parte 1), para luego adentrarnos en los procesos de entrenamiento y optimización (parte 2) y en la interpretabilidad (parte 3) de sus modelos.

Con esos elementos daremos nuestros dos pasos finales hacia la IA generativa (parte 4) y el Tiny ML (parte 5).

En este trabajo hemos reducido al mínimo las citas para simplificar la lectura, pero al final incorporamos una lista de bibliografía utilizada que sirve para ampliar los conceptos aquí trabajados.

Lograr que el mayor número de personas comprenda qué es la IA es indispensable para poder hacer de ella una herramienta positiva para la humanidad, comprender los desafíos que

implica construir una gobernanza en el campo y los efectos geopolíticos y en las relaciones internacionales que ya está produciendo, pero por sobre todo es darle a las personas información y conocimientos para que tomen sus decisiones sobre conocimiento y evidencia frente a las grandes preguntas que la IA nos plantea sobre nuestro futuro y sobre nuestra propia esencia como seres humanos.

Parte 1: Introducción a la IA

Definición de inteligencia

Existen muchas y diversas definiciones de inteligencia.

Jean Piaget, por ejemplo, conectó la idea de inteligencia con las operaciones lógicas humanas.

François Chollet la define como *la medida de la eficiencia en adquirir habilidades en un ámbito específico de **tareas** con respecto a conocimientos previos, experiencia y dificultad de generalización.*

El primer componente (adquirir habilidades) puede traducirse como "aprender", por lo que hablamos de una eficiencia en el aprendizaje de tareas

Creo el ARC (*Abstract Reasoning Challenge*) como medida de la inteligencia de la IA (una especie de IQ para la IA).

Para la IA necesitamos métricas verificables de inteligencia que incluyan inteligencias no humanas. Aquí entra el test de Turing.

En IA trabajamos con una métrica de **éxito en tareas** (o error, que es su función inversa)

El progreso que venimos viendo en materia de IA se sostiene sobre tres pilares que apoyan su capacidad de aumentar el éxito de sus tareas, el llamado "triángulo virtuoso" de la IA, cuyos vértices son el aumento de la capacidad de cómputo, la disponibilidad creciente de macrodatos y, por fin, la mejora de los algoritmos de operación.

Los tipos de problemas y la IA

Existen diferentes tipos de problemas. Algunos resultan de solución fácil para la IA, otros difíciles y algunos, hasta hoy, imposibles.

PROBLEMAS P: PROBLEMAS RESOLUBLES EN TIEMPO POLINOMIAL (RÁPIDOS).

Son problemas que pueden resolverse en tiempo polinomial por una computadora determinista (es decir, mediante la aplicación de un algoritmo que no deba trabajar con adivinanzas o azar).

Los problemas P son "fáciles" de resolver por algoritmos simples

Ejemplo: El algoritmo debe encontrar una palabra en una sopa de letras.

Problema P-Completo:

Siguen siendo problemas del tipo P, pero son más complejos, por lo que, como pauta general, no se puede acelerar su resolución utilizando procesadores trabajando en paralelo.

Ejemplo: si en el juego en que se lanza una bola desde arriba y va rebotando en clavijas hasta llegar a una celda abajo que le asigna un puntaje, si cada clavija se ubica en cada lanzamiento de la bola de forma arbitraria y queremos saber el resultado para múltiples lanzamientos estamos ante un problema P-completo

Problemas NP: Problemas cuya solución ocurre en tiempo exponencial, pero se puede verificar rápido.

Problemas que no necesariamente podemos resolver rápido porque requieren de tiempo exponencial, esto es que las opciones crecen en el tiempo, pero cuyas soluciones, si alguien nos las da, podemos verificar en tiempo polinomial.

Los problemas NP no son fáciles de resolver, pero dada la solución, esta es fácil de verificar.

Ejemplo: El Sudoku de 4 cuadros es más fácil de resolver que el de 8 cuadros, a más cuadros la complejidad (qué número va en cada cuadrado) crece, pero si tenemos la solución comprobar que no haya errores es rápido y fácil)

Problemas NP-completos: su complejidad crece a nivel exponencial.

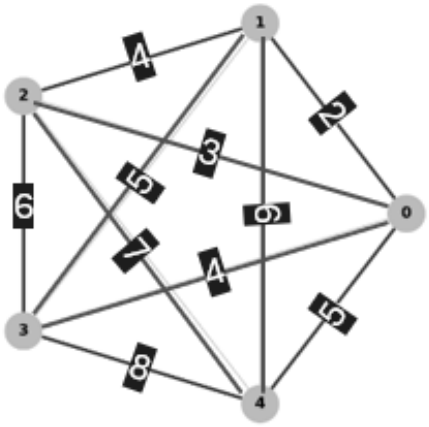
Son los problemas más difíciles de NP. Si encontráramos un algoritmo polinomial para resolver uno de ellos (lo que no ha pasado), podríamos resolver todos los NP en tiempo polinomial.

El problema del viajante (Travelling Salesman Problem, TSP)

Un viajante que tiene que visitar varias ciudades y tiene un mapa con las distancias entre ellas. Lo que busca es salir de una ciudad, visitar todas las demás una sola vez, y volver al inicio, por lo que se pregunta ¿Cuál es la ruta más corta posible que cumple esas condiciones?

Ejemplo para 5 ciudades

Ruta óptima (en rojo) = 25



S aumenta el número de ciudades, las rutas posibles crecen de forma exponencial y con ello las opciones a analizar, haciendo que encontrar la solución se mucho más complejo por cada ciudad que se agrega:

Crecimiento de recorridos posibles en TSP

3 ciudad = 1 recorrido

4 ciudades = 3 recorridos

5 ciudades = 12 recorridos

10 ciudades = 181.440 recorridos

20 ciudades = más de 60 cuatrillones de recorridos

EJEMPLOS DE PROBLEMAS P Y NP

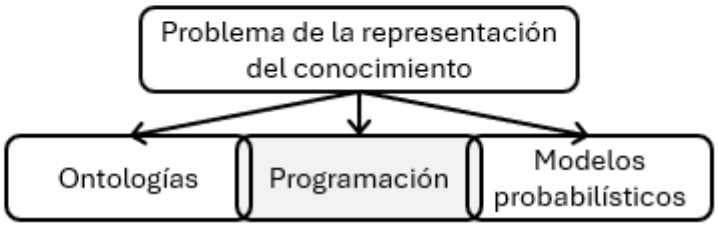
IA simbólica / IA no simbólica

Característica	**IA simbólica** **También llamada "clásica" o "GOFAI" (Good Old-Fashioned AI)**	**IA no-simbólica**
Clave de funcionamiento	"Le enseño reglas, sigue instrucciones".	"Le doy ejemplos, aprende patrones por sí misma".
Cómo funciona	Usa **reglas lógicas y símbolos** (sistemas expertos) definidos por humanos: "si pasa A, entonces B".	Aprende **patrones a partir de datos** (redes neuronales, *machine learning*): reconoce patrones sin reglas explícitas
Analogía	Un manual de instrucciones o libro de recetas	Un niño que aprende a partir de lo que percibe.
Ejemplo en juegos	Programa de ajedrez con reglas y heurísticas escritas por humanos (Turing).	AlphaZero: aprende a jugar ajedrez y Go jugando contra sí mismo (IBM).
Ejemplo en lenguaje	Traductor basado en gramáticas y diccionarios programados.	Google Translate: aprende traducción con millones de ejemplos de

		textos.
Fortalezas	Transparencia (explica sus decisiones).	Flexible (aprende de datos nuevos sin reglas predefinidas).
Debilidades	Poco adaptable; necesita muchas reglas manuales.	Caja negra: No se entiende cómo toma decisiones.

LA IA SIMBÓLICA

Se basaba en **sistemas expertos** en los que un experto del área establecía las reglas lógicas y símbolos que el programador "decodificaba" en el modelo.

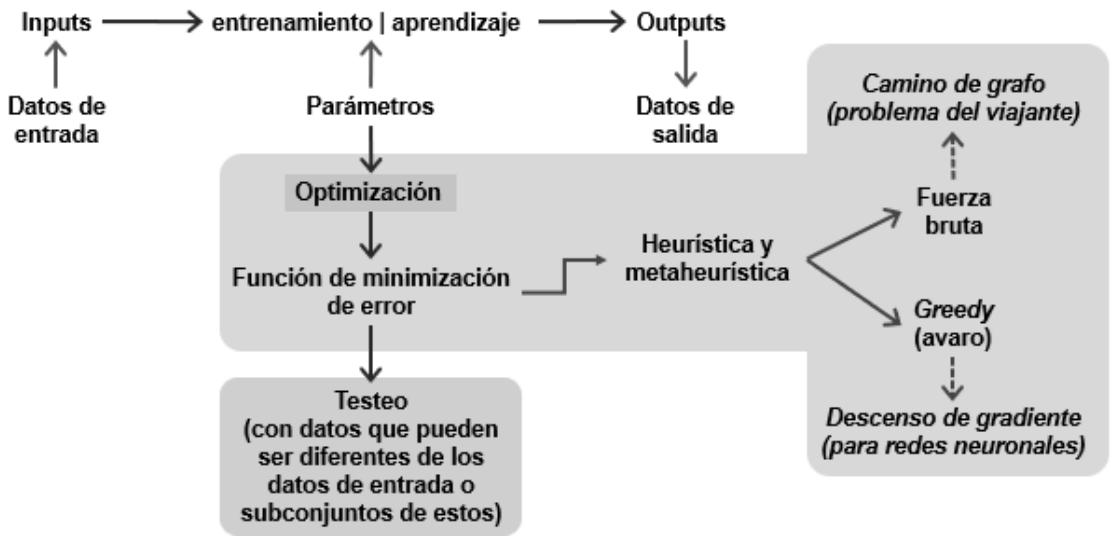


Los pasos que utilizamos hoy son

- **Planificación**: Determina la secuencia de acciones necesarias para alcanzar una meta.
- **Optimización**: Encuentra la mejor manera de ejecutar ese plan de manera eficiente y con el menor costo posible.
- **Toma de decisiones:** En cada paso del plan u optimización, selecciona la mejor acción, considerando los cambios en el entorno y la incertidumbre.

IA NO SIMBÓLICA

Es la Inteligencia Artificial basada en Datos, Machine Learning o Aprendizaje Automático.



Ver detalles y análisis del gráfico en el análisis de optimización

IA fuerte / IA débil

La **palabra clave** para distinguirlas es "**dominio**".

IA débil: trabaja en un solo dominio (ejemplo: juega al ajedrez, pero no puede jugar a las damas; puede detectar y monitorear un objeto en un video, pero no explicar lo que está ocurriendo en él)

IA fuerte: trabaja en múltiples dominios (ejemplo: puede generar y validar hipótesis en cualquier área de la ciencia. Hasta el día de hoy no existen). Es robusta, flexible, general y adaptable a diferentes dominios

Hay casos en que las IA débiles alcanzan niveles de perfección tan altos que dan la sensación de ser fuertes, pero no dejan de trabajar en un solo dominio (chat GPT solo responde a los inputs en lenguaje natural, pero no entiende lo que está diciendo)

LÍNEA DE TIEMPO DEL DESARROLLO DE CONCEPTOS

1950 – Alan Turing: idea de máquinas inteligentes
Publica *Computing Machinery and Intelligence*.
Propone el Test de Turing como forma de evaluar si una máquina "piensa".
Hablaba de la **posibilidad de desarrollar máquinas con inteligencia general**.

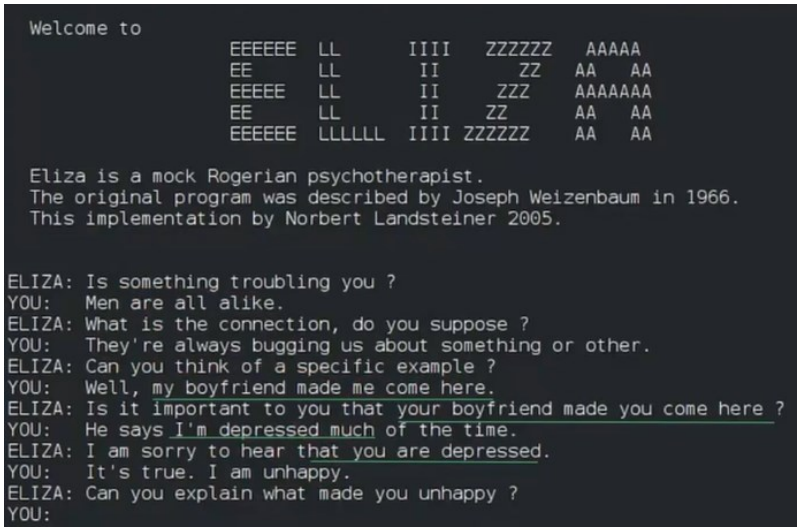
1956 – Dartmouth Conference
Nace el campo de la **IA**
Los primeros programas resuelven problemas matemáticos y juegos (ajedrez, damas).
Son IA débiles.

1965 – Herbert Simon y Allen Newell
Afirman que en 20 años habrá máquinas capaces de hacer todo lo que hace un humano.

Refuerzan la idea de una IA fuerte como meta final.

1966 – Chatbot Eliza

Primer chatbot. Simulas un psicólogo. Muy limitado (usaba IA simbólica), pero atractivo para las personas.



1970s – John Searle: se formaliza la distinción **IA débil vs. IA fuerte**.

Introduce explícitamente los términos en su **ensayo sobre la habitación china** (1980, pero ya circulaba en los 70).

IA débil = máquinas que simulan inteligencia en tareas específicas.

IA fuerte = máquinas con mente, comprensión y conciencia.

1980s – 1990s

Avances en sistemas expertos, visión artificial y reconocimiento de voz.

Todos los progresos siguen siendo IA débil (medicina, industria).

1997 – Deep Blue (IBM)

Derrota a Garry Kasparov en ajedrez.

Ejemplo de **IA débil que supera a la humana en su dominio específico**.

2010s – Deep Learning

Redes neuronales profundas impulsan la visión artificial, el procesamiento de lenguaje natural y el reconocimiento de voz.

Surgen masivamente nuevas aplicaciones de IA débil (traducción, conducción autónoma, diagnósticos médicos).

2014 – Redes generativas adversarias (Generative Adversarial Networks: GAN)

Una red aprende y otra le sirve de referencia calificando sus resultados, en procesos de reinicio iterativo a partir de los mejores resultados logrados en la vuelta anterior.

Fundamentales para la identificación de imágenes y los programas LLM.

2017 – AlphaZero (DeepMind)

Aprende ajedrez, shogi y Go desde cero, pero sigue siendo IA débil, porque solo puede actuar en ese dominio dado por los tres juegos para los que fue entrenada.

2020s – Modelos fundacionales (GPT, Perplexity, Gemini…)

Pueden escribir textos, programar, traducir, resumir y razonar de manera versátil.

Parecen IA fuertes, pero siguen siendo IA débiles: aunque muy avanzadas, siguen confinadas al dominio del lenguaje y los datos de entrenamiento.

2020s – Modelos multimodales

En 2021 se lanza ChatGPT (IA no simbólica basada en redes neuronales).

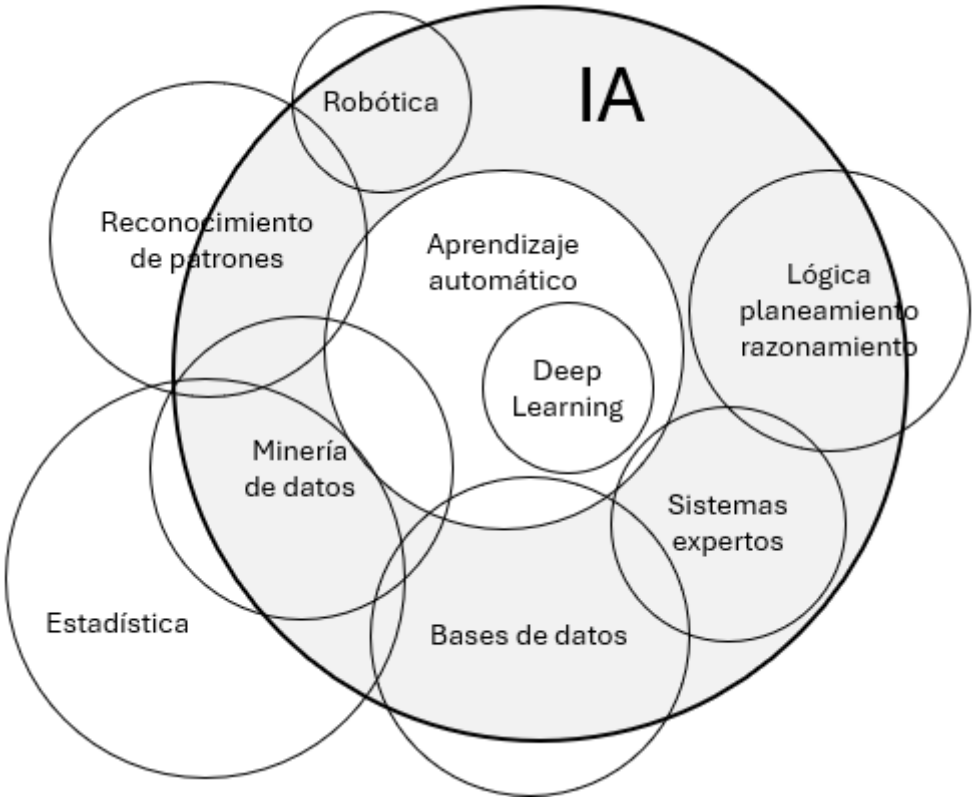
Hoy:

La **IA fuerte** continúa siendo un objetivo **hipotético**.

La "**superinteligencia**" plantea una IA fuerte que supere a la inteligencia humana en todos los dominios, incluso en creatividad y sensibilidad. Requiere entonces de una IA fuerte como condición previa, pero no suficiente. La "**singularidad**" es el momento en el tiempo en que la superinteligencia llegue a ser realidad y los humanos dejemos de ser las entidades más inteligentes del planeta.

Riesgo de "captura regulatoria" por parte de las grandes empresas, que quieren establecer "límites" o "reglas" a la IA de acuerdo con sus propias conveniencias y de **"tecnofeudalismo"** (Varufakis)

CAMPOS RELACIONADOS CON LA IA



Parte 2:
Entrenamiento y optimización de la IA

Introducción

Optimizar una IA es entrenarla para reducir su función de error basándonos en operaciones matemáticas. Para lograrlo, se deben ajustar sus parámetros tanto como sea posible o deseable.

Entrenamiento de IA

El aprendizaje se da por tres vías posibles

- Memoria: Recordar
- Aprendizaje inductivo: Aplica reglas (Sistemas Expertos; IA simbólica)
- Aprendizaje deductivo: Aprende de los datos (Machine Learning; IA no-simbólica)

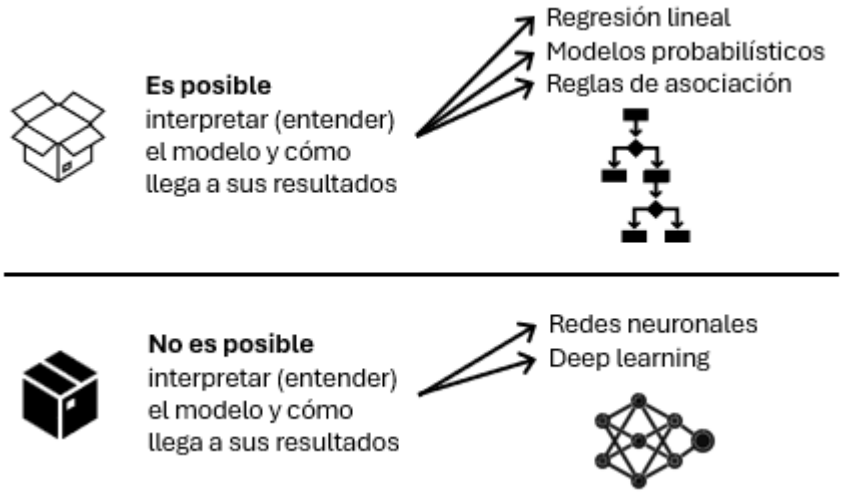
Existen cinco grandes tipos de entrenamiento en el aprendizaje automático que realizan las IA:

1. Supervisado, con el uso de datos etiquetados.
2. No supervisado, en que los datos no tienen etiquetas y es el propio modelo quien genera patrones para ordenar los datos.
3. Auto-supervisado: con datos sin etiquetas, la supervisión queda en manos de los propios datos con que ya cuenta el modelo.
4. Semi-supervisado, donde se utilizan unos pocos datos etiquetados y muchos sin etiquetar, tarea que se deja al modelo.
5. Por refuerzo: otorgando “premios” y “castigos” al modelo en forma de asignaciones de puntuación a los outputs obtenidos, según nos aproximen o no al resultado esperado.

Los problemas que enfrentamos mediante IA se agrupan en dos grandes grupos:

- Problemas de regresión: cuando queremos que nuestro modelo prediga un valor dentro de un rango continuo
- Problemas de clasificación: cuando queremos que nuestro modelo ordene los datos de entrada entre dos o más clases.

Los modelos se entrenan para reducir su función de error. El agente que guía el aprendizaje del modelo puede ser un humano, un modelo de *machine learning* o el propio modelo creando generalizaciones. Los resultados obtenidos pueden ser interpretables o no, en función de lo cual se los clasifica como de “caja blanca” o “caja negra”.



MODELO BÁSICO: KNN (K-NEAREST NEIBOUGHRS)

No es en sí un modelo de ML

Es un algoritmo supervisado que clasifica un dato nuevo según la mayoría de clases entre sus “k-vecinos” más cercanos en el espacio de características.

No requiere entrenamiento. Almacena datos y calcula distancias entre ellos durante la predicción de clasificación de nuevo dato,

Es simple y efectivo, pero puede volverse costoso con grandes volúmenes de datos.

KNN no optimiza parámetros del modelo (*Lazy Learning*).

Dataset			Dataset ordenado por distancia al nuevo			
Activación 1	Activación 2	Objetivo	Activación 1	Activación 2	Objetivo	Distribucion para K=5
3.09	2.83	⊙	3.09	2.83	⊙	0.19
2.79	2.01	⊙	3.37	3.68	#	0.78
0.14	2.58	⊙	2.79	2.01	⊙	1.01
3.43	1.89	⊙	2.46	4.04	₪	1.17
3.59	4.83	#	3.43	1.89	⊙	1.19
4.42	5.12	#	1.61	3.60	₪	1.52
6.67	2.45	#	1.43	3.55	₪	1.67
3.37	3.68	#	3.59	4.83	#	1.92
1.61	3.60	₪	3.74	5.20	₪	2.33
1.43	3.55	₪	4.42	5.12	#	2.55
2.46	4.04	₪	0.14	2.58	⊙	2.89
3.74	5.20	₪	6.67	2.45	#	3.71

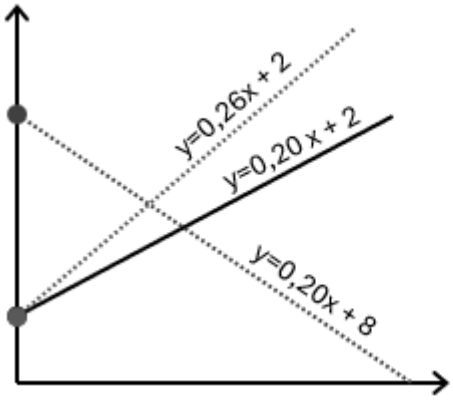
El número K lo asigna el usuario

⊙ es el grupo más repetido para K:5, por tanto ☯ (3-3) = ⊙

Regresión Lineal

No es en sí un modelo de ML.

Se aplica a problemas del tipo ¿Qué valor asume Y dependiendo del valor que asuma X? Si asumimos que la relación es lineal podemos crear un **modelo lineal**

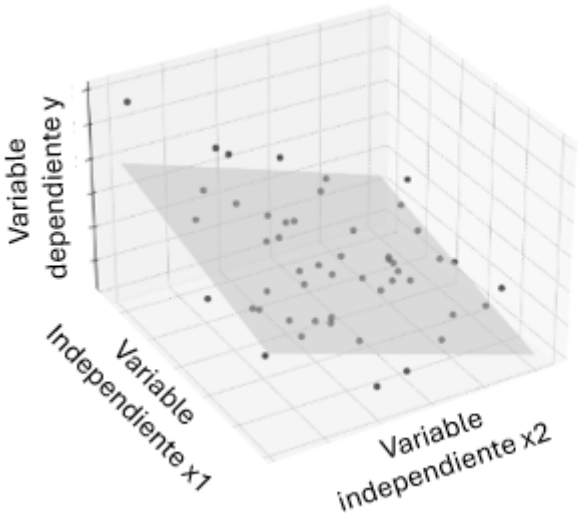


y = mx+b

m= pendiente de la recta
b= ordenada al origen

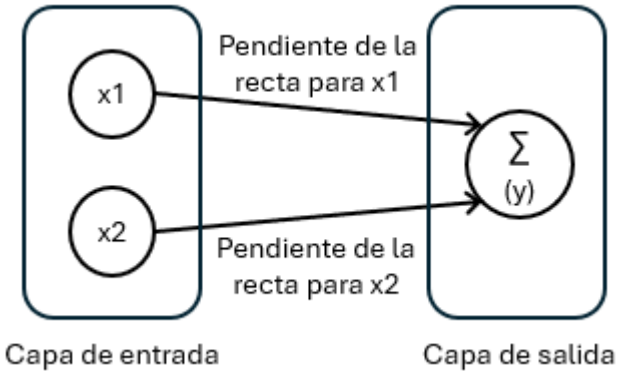
m=0,20; b=2
m=0,26; b=2
m=-0,20; b=8

La **regresión multivariada** aparece cuando tengo más de un valor de x. Nos muestra cómo múltiples variables explicativas (independientes) influyen sobre múltiples variables de resultado (dependientes), pudiendo predecir dos o más variables dependientes al mismo tiempo y permitiéndonos entender cómo conjuntos de factores (variables independientes) afectan simultáneamente a diversos resultados (variables dependientes).



Con dos variables independientes (x1 y x2) y una dependiente (y). El plano muestra el modelo ajustado que intenta explicar la relación entre las tres variables.

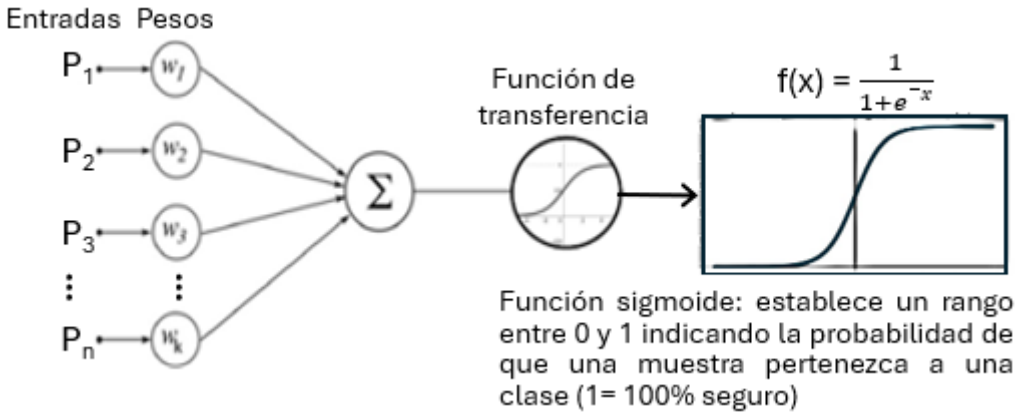
Explica cómo variables explicativas independientes influyen sobre más de una variable de resultado (dependiente). Puede predecir dos o más variables dependientes al mismo tiempo y nos permite entender cómo conjuntos de factores afectan simultáneamente a diversos resultados



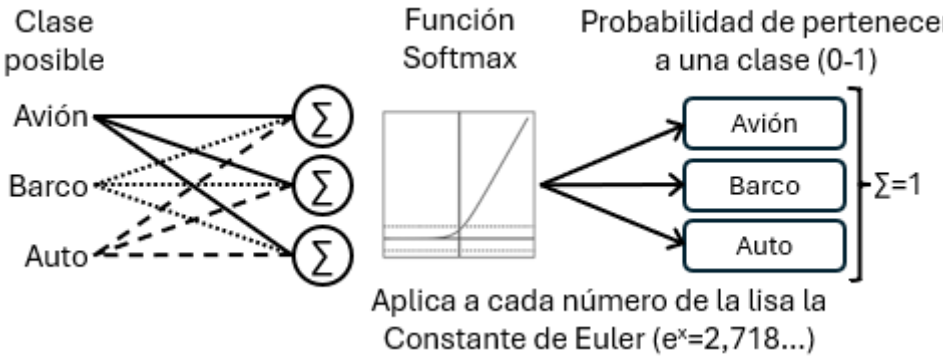
REGRESIÓN LOGÍSTICA BINARIA

Se utiliza en **problemas de clasificación entre dos clases.**

Para **transformar una regresión lineal en una logística** se debe aplicar un "función de transferencia" (sigmoide)



Pero cuando la clasificación incluye más de dos clases, deja de ser binaria y pasa a ser una **clasificación multiclase.** En estos casos **se aplica un regresor a cada clase, incluyendo una función Softmax**.



El proceso que se realiza aquí es el siguiente:

1. Se parte de las puntuaciones crudas asignadas a cada clase
2. Se estabilizan numéricamente las puntuaciones restando el máximo a todas ellas, lo que no cambia el resultado de softmax y evita *overflow* al exponenciar.
3. Exponenciamos aplicando la constante de Euler (e^x) a cada valor obtenido, de modo que todos asuman signo positivo.
4. Normalizamos: dividiendo e^x por la suma de todos los valores resultantes.

Categ.	**Valor $\Sigma_{(1,2,3)}$**	**Restamos el máximo valor**	**Exponenciamos al valor obtenido**	**∑ valor e^x**	**Normalizamos dividiendo e^x por la suma de los valores exponenciados**	
			e^x= 2,178…		**Probabilidad de (∈) a la categoría**	**Total**
Avión	2	2-2=**0**	e^0=**1**	**1,49**	**1**/1,49 = **0,67**	**∑= 1**
Barco	1	1-2=**-1**	e^{-1}=**0,36**		**0,367**/1,49=**0,24**	
Auto	0	0-2=**-2**	e^{-2}=**0.13**		**0,135**/1,49=**0,09**	

Redes Neuronales

Antes de entrar de lleno en las redes neuronales es necesario plantear el problema previo del posible gran número de dimensiones con que tendrán que operar. Para enfrentar esta enorme multidimensionalidad y tratar de escoger el mejor modelo para nuestro objetivo, comprender la dimensionalidad del problema a abordar es indispensable.

Para ello una de las técnicas más utilizadas es **t-SNE** (*t-distributed Stochastic Neighbor Embedding*) que ayuda a reducir la **dimensionalidad** de los datos contribuyendo a la posibilidad de **visualización** de estos.

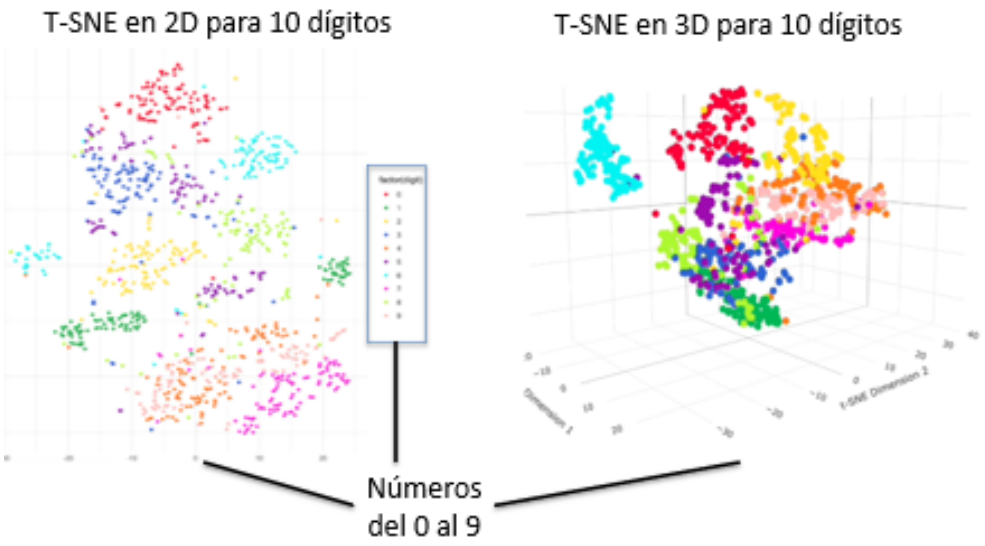
Así, **t-SNE** es un algoritmo no lineal de reducción de dimensiones, especialmente diseñado para representar datos complejos de alta dimensión en 2D o 3D. Su objetivo principal no es preservar las distancias exactas entre dimensiones, sino **conservar la estructura local manteniendo los puntos que son vecinos en alta dimensión** sigan situándose cerca en la representación de baja dimensión.

Supongamos que tenemos vectores de 300 dimensiones, **t-SNE** comienza por calcular la **probabilidad de vecindad** entre cada par de puntos. Luego, el algoritmo coloca los puntos en un espacio más pequeño donde vuelve a calcular probabilidades de vecindad y ajusta las posiciones en el espacio reducido para que las **probabilidades de cercanía en 2D** se parezcan lo más posible a las del espacio original (no es relevante, pero usa la divergencia de Kullback–Leibler (KL) como medida de error).

Con ello logra:

- **Una visualización clara de clústeres**: datos similares forman grupos compactos.
- **Separar entre clases**: útil para ver cómo un modelo organiza representaciones internas.
- **Explorar estructuras ocultas**: permite detectar patrones que no son visibles en el espacio original.

Como resultado, **t-SNE** convierte relaciones de cercanía en alta dimensión en representaciones visuales en 2D/3D, resaltando grupos y estructuras locales de los datos.



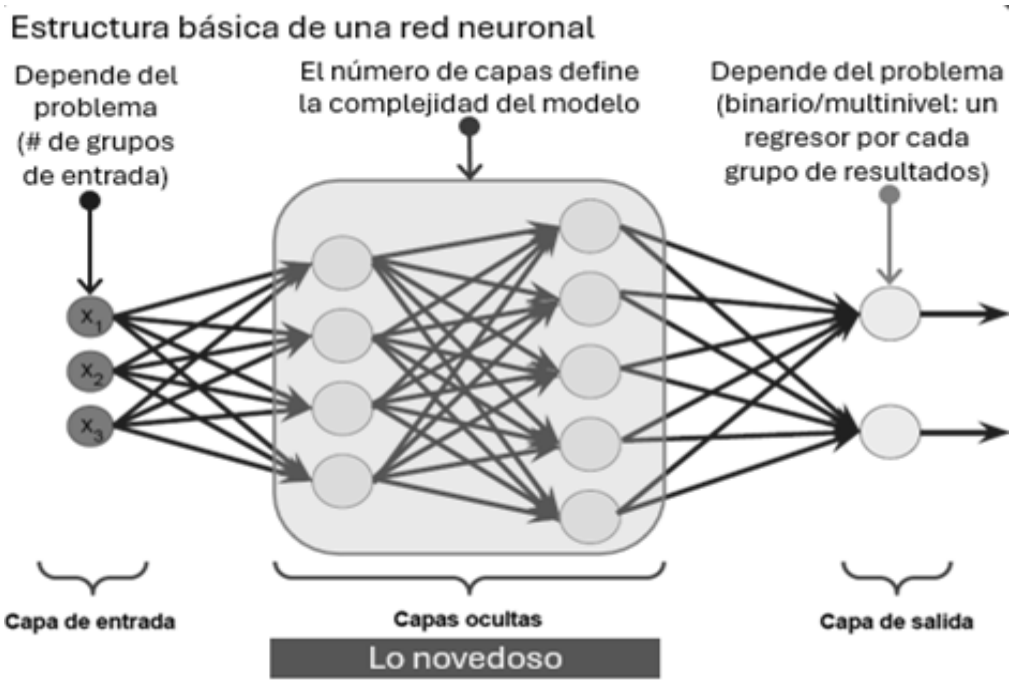
Fuente: https://www.appsilon.com/post/r-tsne

Ahora sí, vamos al tema de **las redes neuronales**.

Dado que hay problemas complejos que no pueden resolverse con funciones lineales, las redes neuronales combinan cualquier función con cualquier otra (regresiones lineales con funciones no lineales), lo que las hace capaces de aproximar cualquier función, por lo que entran en la categoría de los llamados "aproximadores universales".

Las capas están formadas por neuronas artificiales que realizan procesos, y cuyo nombre técnico es "**perceptrónes**".

Así, las redes neuronales son combinaciones de funciones matemáticas en capas (lo que da por resultado una nueva función surgida de la composición de las funciones de cada capa). Por tanto, el gran problema es el de poder "transformar" cualquier problema en una función matemática *(¿Podemos representar todo lo que hace un ser humano bajo la forma de función matemática?)***.** Logrado eso, se puede montar la red neuronal más conveniente para reducir la función de error en la salida.



LAS FUNCIONES MÁS USUALES

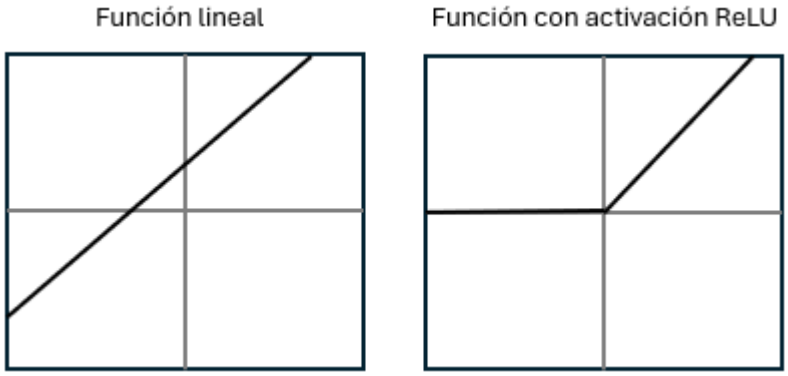
a) RELU

Uno de los tipos de funciones más aplicados en redes neuronales es el llamado "**ReLU**" *(Rectified Linear Unit)* que toma un número de entrada cualquiera y si su valor es positivo, lo deja igual, pero si es cero o negativo lo transforma en cero. Aplica la fórmula **f(x)=max(0,x)** en que

- **Para x<0: f(x)==0 (constante).**
- **Para x≥0: f(x)=x (lineal).**

Esto nos lleva a decir que ReLU es una función "lineal a partes" (*piecewise linear*): **el "quiebre" en el punto 0 es lo que hace que la red no sea lineal.** Ese cambio de pendiente permite

que la red combine múltiples cortes o regiones y modele patrones no lineales en los datos, lo que permite, por ejemplo, aproximar curvas.



Esta fórmula aporta **simplicidad de cálculo** y **velocidad,** ya que solo compara con cero, y **evita saturaciones,** al tiempo que **mantiene la no linealidad**, lo cual permite a la red aprender relaciones complejas y simplifica el **entrenamiento** al hacer más fácil la propagación del gradiente. El problema es que algunas neuronas pueden quedar "muertas" si siempre reciben valores negativos.

b) LA FUNCIÓN SIGMOIDE

También conocida como función *logística,* esta función de activación transforma cualquier número real en un valor entre **0 y 1**.

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

Su funcionamiento es simple: si la entrada que recibe es **positiva**, entonces el exponente **x tiende a infinito y el exponente -x tiene a -∞, por lo que e^{-x} tenderá a cero, dando como resultado que σ(x)→1**, pero si la entrada es **negativa -x tiende a ∞,** por lo que **e^{-x} tenderá a uno**, **dando como resultado que σ(x)→0.** Y si la **entrada es cero**, entonces

$$\sigma(0) = \frac{1}{1+1} = 0.5$$

En consecuencia, siempre devuelve un valor entre 0 y 1, lo que la hace ideal para problemas de clasificación binaria, y generará una gradiente suave.

c) LA FUNCIÓN TANH

Es una función hiperbólica que **transforma cualquier número real a un espacio entre +1 y** -1.

En cierto sentido es similar a la sigmoide ya que se calcula con la siguiente fórmula:

Esto implica que

Si x→ ∞ → tanh(x)→1

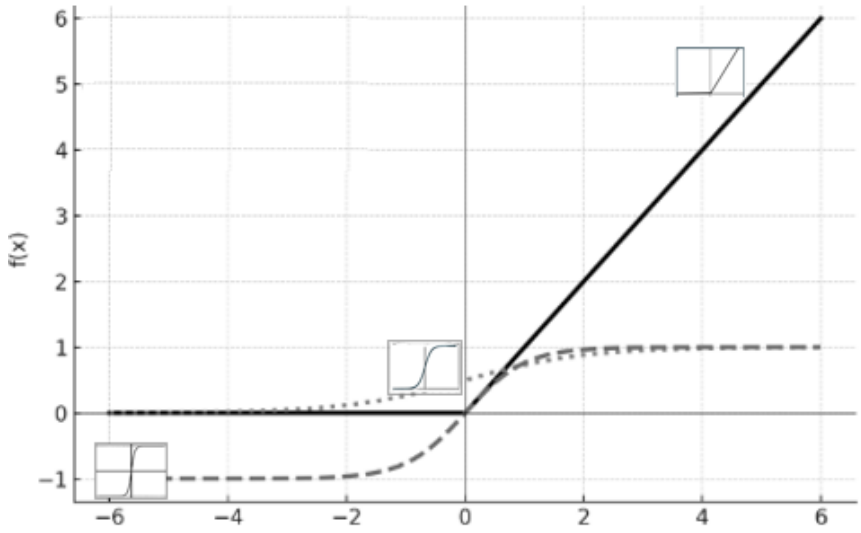
Si x→ -∞ → tanh (x)→−1

Si x→ 0 → tanh(x)→0

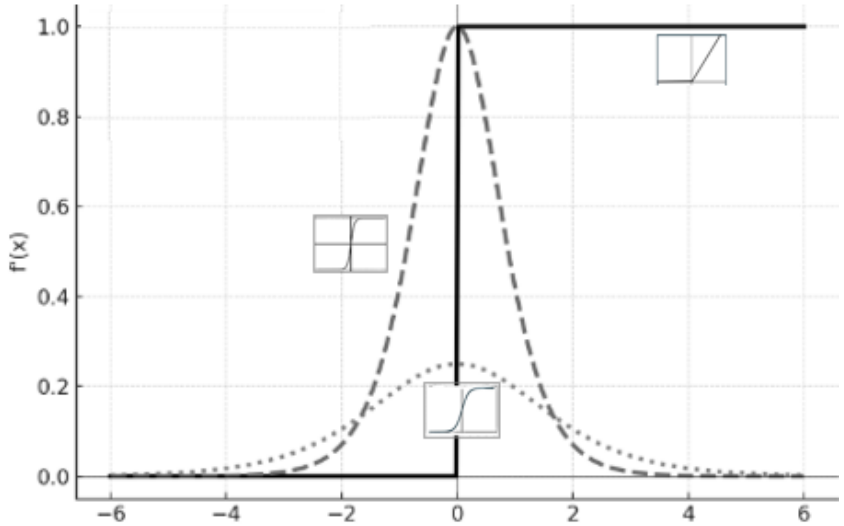
La función de cálculo es

$$\tanh(x) = \frac{e^{x} + e^{-x}}{e^{x} - e^{-x}}$$

Las funciones sigmoide y tanh confluyen hacia sus extremos y se diferencian hacia el centro



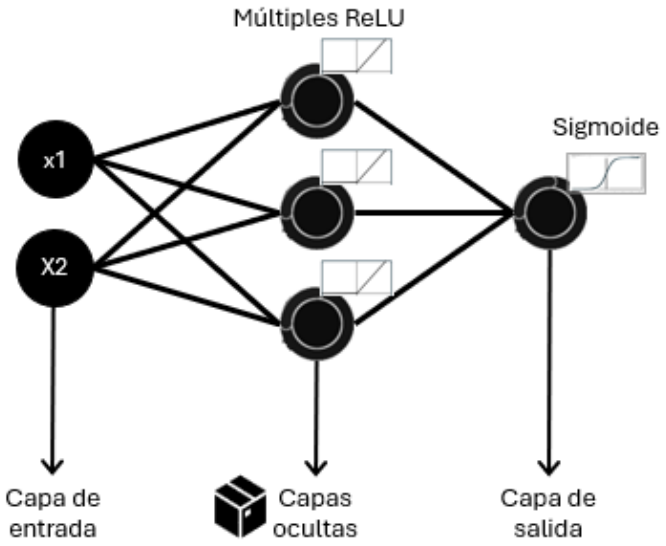
Y aquí podemos ver cómo cambian sus gradientes

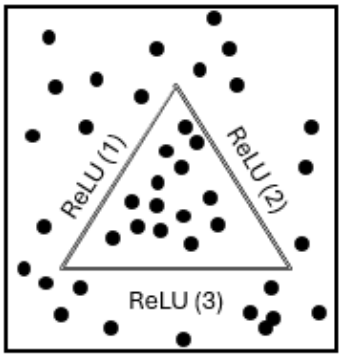


LAS CAPAS DE LAS REDES NEURONALES: COMBINACIONES DE FUNCIONES

Las diferentes capas de una red combinan funciones. Veamos algunos ejemplos:

FUNCIONAMIENTO DE UNA RED NEURONAL BINARIA CON FUNCIÓN SIGMOIDE DE SALIDA

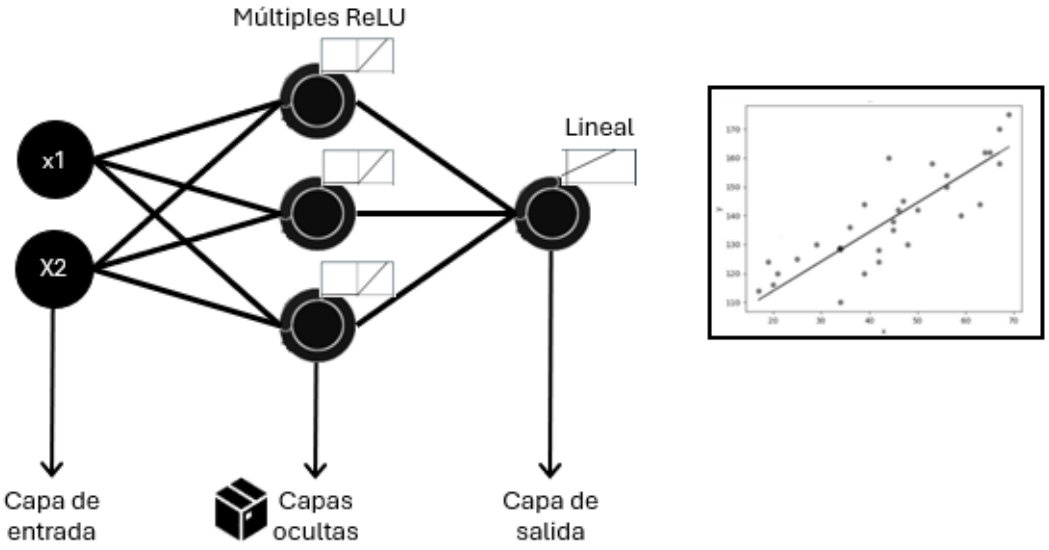




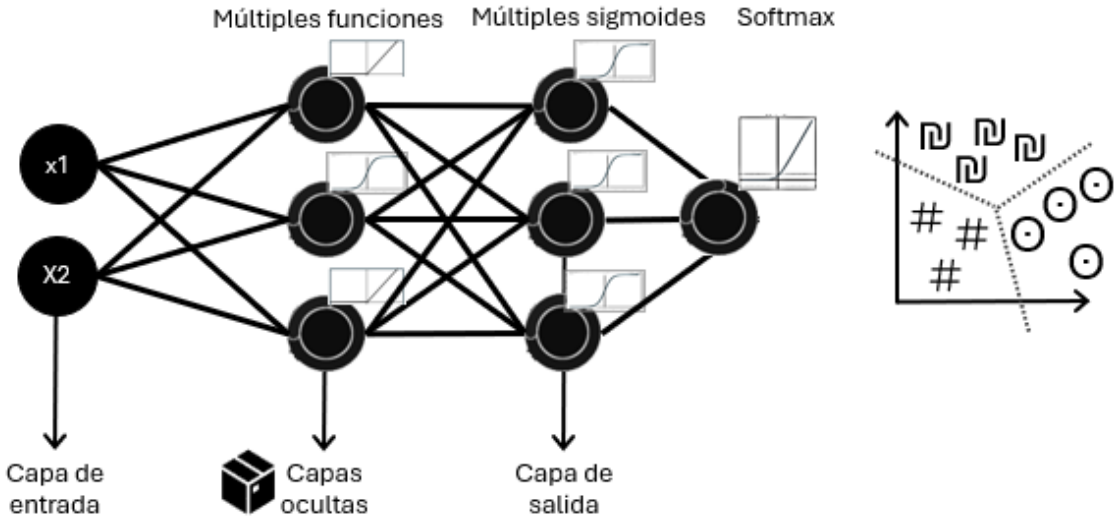
Entrada (x)	ReLU $f(x) = \max(0,x)$	Sigmoide $f(x) = \frac{1}{1+e^{-2}}$
-2	0	0,119
-1	0	0,269
0	0	0,500
1	1	0,371
2	2	0,881

Puedo usar redes neuronales para realizar regresiones, solo se trata de cambiar la función de salida a una función lineal.

FUNCIONAMIENTO DE UNA RED NEURONAL BINARIA CON FUNCIÓN LINEAL DE SALIDA



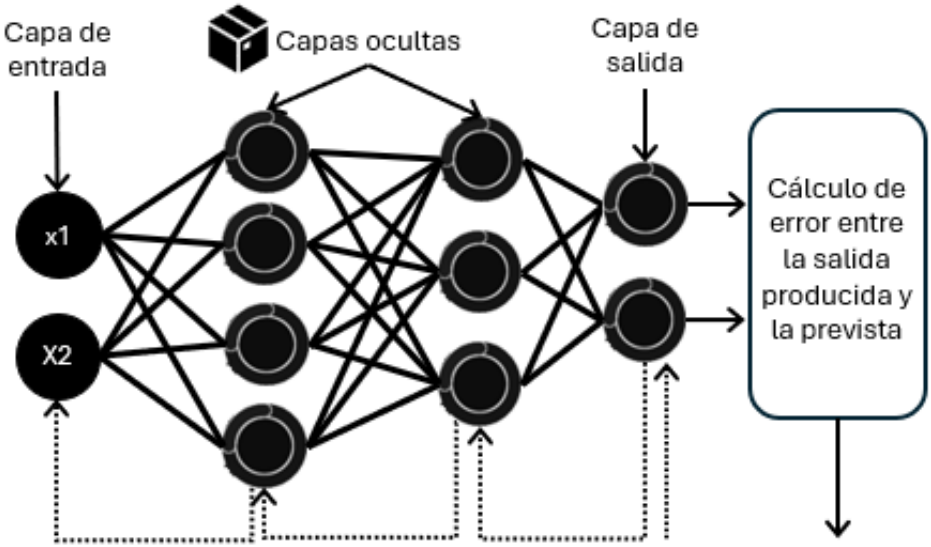
FUNCIONAMIENTO DE UNA RED NEURONAL MULTICLASE



Entrenamiento de una red neuronal

El entrenamiento consiste básicamente en un **ciclo de medición del cálculo de predicciones para poder medir el error entre la esperada y la obtenida**, calcular gradientes con *backpropagation* y ajustar los pesos de las variables mediante el descenso de gradiente obtenido. Cada uno de estos ciclos se denomina "época" y se repite tantas veces como sea necesario sobre los datos de entrenamiento hasta alcanzar un mínimo de error aceptable.

El algoritmo ***backpropagation*** funciona de la siguiente manera:



Cálculo de derivada parcial y actualización de parámetros "hacia atrás"

En palabras simples, detectado el error *backpropagation* inicia un camino hacia atrás para ajustar los espacios (variables) que lo han producido.

Evaluación de modelos

¿Cómo sé que un modelo es bueno? Entre dos modelos ¿Cuál es el mejor? Para saberlo necesito dos cosas:

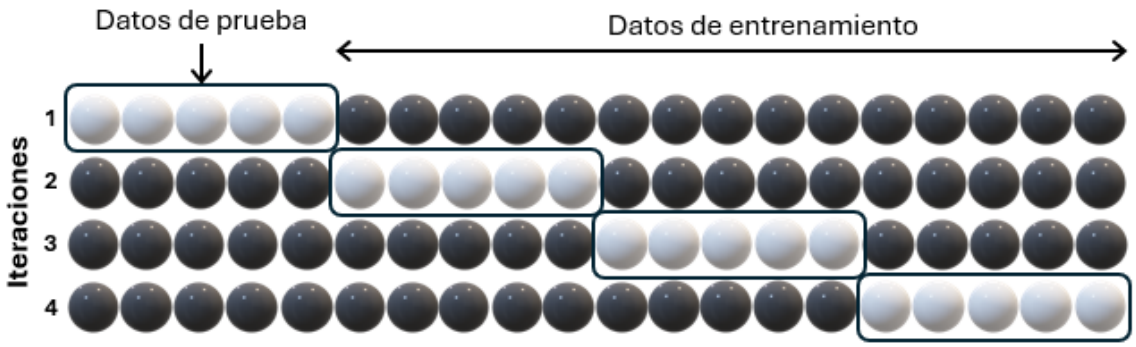
1. **Métricas** para comprobar el nivel de rendimiento del modelo.

   Ejemplos de métricas:

   - Para clasificación binaria, la medida más aplicada es la entropía cruzada binaria.
   - Para clasificación multiclase, la medida más aplicada es la entropía cruzada categórica
   - Para **regresión**: la medida más aplicada es el **error cuadrático medio**.
   - Cuando los datos están desbalanceados o tengo intereses específicos en ciertos espacios de resultados, es mejor aplicar medidas como **accuracy, f-measure, etc**.

2. **Conjuntos de datos diferentes a los de entrenamiento** para ver cómo funciona con ellos, llamados "datos de evaluación" (*testing data*)

   - Idealmente, utilizando nuevos datos completamente nuevos para el modelo.
   - Usando datos reservados para testeo de la base de datos original (riesgo: que se mantenga un sesgo que está en la base de datos de trabajo)
   - Por validación cruzada, iterando subconjuntos de los datos de prueba de entre los datos de entrenamiento y promediando las métricas.



Veamos las tres **medidas que mencionamos como las más usuales:**

La entropía cruzada binaria mide **cuánto se parece la probabilidad que predice el modelo a la realidad** utilizando la siguiente fórmula:

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

Siendo **y** la etiqueta real (clase 0 o clase 1, ya que se aplica a clasificaciones binarias) y **p** la probabilidad predicha para la clase 1. Cuanto más cercana a 1 sea la predicción, menor será el grado de error.

La **entropía categórica** mide lo mismo que la binaria, pero se aplica a la **categorización en varias categorías** utilizando la siguiente fórmula:

$$H = -\sum_{i=1}^{C} y_i \cdot \log(p_i)$$

Aquí C representa al número de clases; y_i es el indicador de la clase esperada (1 para ella, 0 para las demás) y p_i es la probabilidad predicha para la clase i.

Para tres clases (avión, barco, auto) siendo que la esperada es barco tenemos que y= [0, 1, 0] y las predicciones del modelo son p=[0.2, 0.7, 0.1], entonces $H=-(0 * \log 0.2 + 1 * \log 0.7 + 0 * \log 0.1) = -\log 0.7$. $H \approx 0.36$.

Si el resultado es un número bajo, como en este caso, se trata de una buena predicción.

Finalmente, el **error cuadrático medio (MSE) se aplica a regresiones** que buscan predecir valores numéricos continuos. Y mide la **diferencia promedio entre los valores reales y los predichos**, elevando las diferencias al cuadrado para evitar que los errores negativos se cancelen con los positivos.

En el **MSE el resultado es si**empre **≥ 0, y c**uanto más bajo sea, **mejor**. Sin embargo, no es posible definir un estándar de valores buenos, ya que depende de la escala de datos con que estemos trabajando: si tenemos 10 valores, un **MSE = 0.2** es excelente, pero si trabajamos con millones de valores, un **MSE = 2000** puede ser igualmente bueno.

La fórmula que se aplica para calcular el MSE es

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

donde y_i es el valor esperado; $\hat{y}_i$ es el valor predicho, y n es el número de datos.

Por ejemplo, tenemos que los valores esperados son [4,1] y las predicciones: [3,2]

n= 2, porque hay dos observaciones.

Calculo cada dato por separado: Primer dato: (4−3)=1; segundo dato: (1−2)=−1

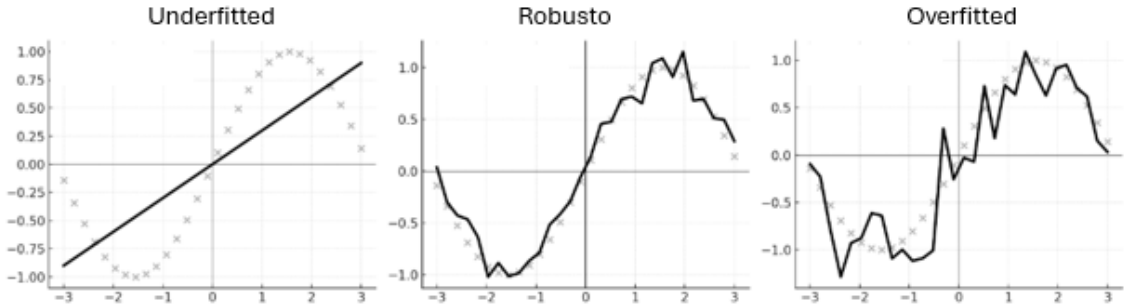
Elevo los números al cuadrado: Primer dato: $1^2 = 1$; segundo dato $(-1)^2 = 1$

Promedio los resultados: 1 + 1 2

-------- = --- = MSE: 1

2 2

El cálculo del error, por cualquier sistema, nos permite saber cómo se ajustan las probabilidades del modelo a lo esperado, y puede darnos tres resultados principales: el modelo está insuficientemente entrenado (*underfitted*), es robusto, o está sobreentrenado lo que atentará contra la posibilidad de clasificar correctamente nuevos datos (*overfitted*).



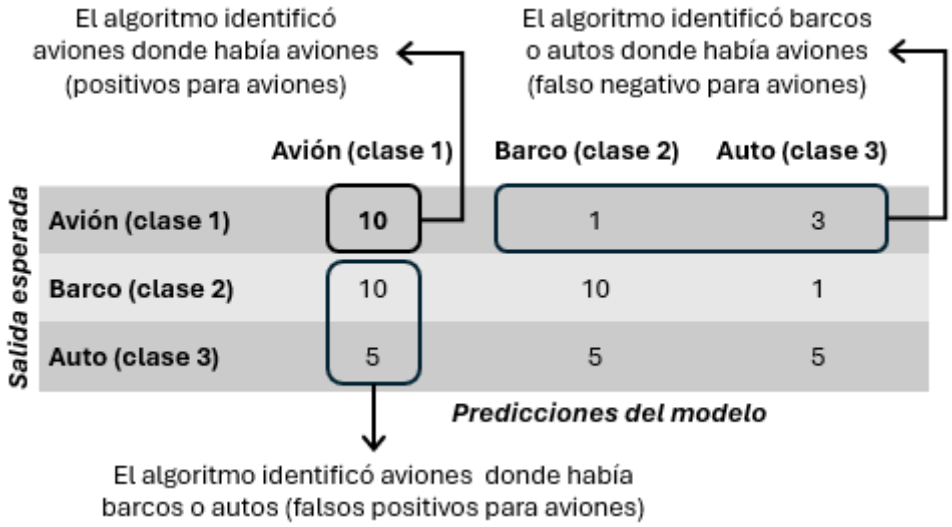
Sin embargo, el cálculo de error nos da el resultado general del modelo, pero no nos dice dónde están las fallas. Cuando se trata de modelos de clasificación, podemos resolverlo mediante la construcción de una **matriz de confusión**:

		Valor calculado por el sistema	
		Positivo	Negativo
Valor real	Positivo	**VP**= Verdadero positivo	**FN**: Falso negativo
	Negativo	**FP**=Falso positivo	**VN**= Verdadero negativo

Ejemplo para un **sistema bidireccional** que detecta quienes viajaron en avión a un encuentro internacional:

		Valor calculado por el sistema	
		Positivo	Negativo
Valor real	Positivo	**15**	20
	Negativo	13	**32**

Ejemplo para un **sistema multinivel** que detecta quienes viajaron en avión, barco o automóvil a un encuentro internacional:



De esta matriz obtenemos un conjunto de métricas, que pueden ser simples o combinarse entre sí.

MÉTRICAS SIMPLES

Accuracy (exactitud): **(VP+VN)/TOTAL**

- Informa qué porcentaje del total de casos clasificado por el sistema fue correctamente clasificado.
- Importante cuando lo que quiero es conocer la **exactitud de los resultados** que da el sistema, sean positivos o negativos.
- En el ejemplo (15+32) / 15+20+13+32 = 47/60 = 47/60 = 0.78

Precisión: **VP/(VP+FP)=VP/Marcados como positivos**

- Informa qué porcentaje de los valores que el sistema consideró positivos son verdaderamente positivos en la realidad
- Importante cuando lo que me interesa es el **nivel de acierto entre positivos**.
- En el ejemplo 15/(15+13) = 15/28 = 0.53

Sensibilidad (Recall): **VP/(VP+FN)**

- Informa qué porcentaje de los positivos encontró el sistema del total de los positivos reales.
- Es mejor cuando quiero estar seguro de **que no quede ningún positivo real excluido** (por ejemplo, en análisis de imágenes por IA para detectar enfermedades)
- En el ejemplo 15/(15+20) = 15/35 = 0.43

Especificidad: **VN/(VN+FP)**

- Informa qué porcentaje de los negativos encontró el sistema del total de los negativos reales.
- Es mejor si quiero evitar dar por positivo algo falso (por ejemplo, un embarazo)
- En el ejemplo 32/(13+32) = 32/45 = 0.71

MÉTRICAS COMBINADAS

Sirven para ver los resultados de forma integrada

F1-Score: **2 * ((precisión * recall)/(precisión + recall))**

- Muy utilizada en problemas en los que el conjunto de datos a analizar está desbalanceado.
- Combina la precisión con recall para obtener un valor más objetivo.
- En el ejemplo 2 * (0,53*0,43) / (0,53+0,43) = 2 * (0.22 / 0,96) = 0.51

Deep Learning y Machine Learning

Tanto el ***Machine Learning*** **(ML)** como el ***Deep Learning*** **(DL)** se apoyan en el descenso de gradiente como función de error que se busca minimizar, por lo que cualquier problema es, en última instancia, un problema de minimización.

El *machine learning* es un "área de estudio que da a las computadoras la habilidad de aprender sin ser explícitamente programadas" o, en otros términos, "estudia mecanismos que permiten a las computadoras aprender a clasificar o predecir en base a experiencias"."

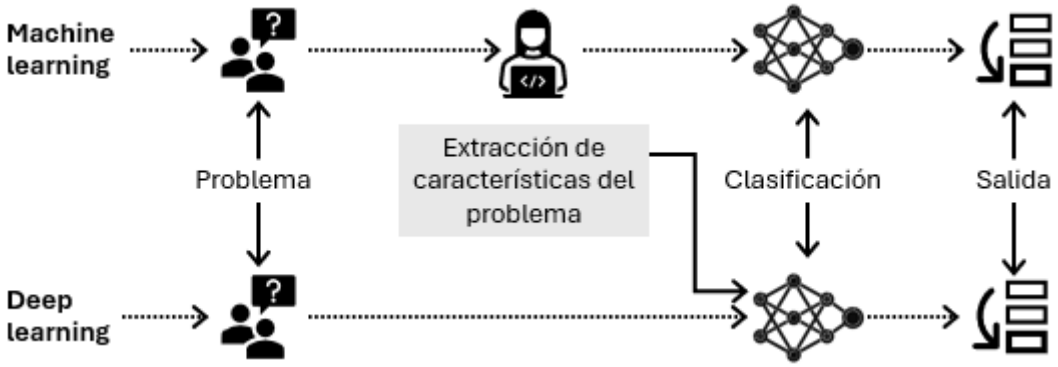
Para ello la metodología más utilizada es **CRISP-DM** (*Cross Industry Standard Process for Data Mining*), una secuencia iterativa de comprensión (del problema), preprocesamiento (de datos), modelado, evaluación y despliegue (del modelo).

Los datos de trabajo pueden estar **estructurados** o ser **multimodales** (fotos, videos, audios, etc)

Aplicando *machine learning* podemos resolver cinco grandes diferentes tipos de problemas, que requieren diferentes formas de entrenamiento

Tipo de problema	Sistema de entrenamiento	¿Hay clases o etiquetas predeterminadas?
Regresión	Supervisado	Existe una clase o etiqueta de datos previamente determinada
Clasificación		
Agrupamiento	No supervisado	No hay información de clase. Los datos se agrupan por similitud
Reducción de dimensionalidad		
Toma de decisiones	Por refuerzo	

El ***Deep Learning*** es "*machine learning* utilizando grandes modelos neuronales y cantidades de datos". Se lo utiliza especialmente para datos multimedia, no para datos tabulares



Hablamos aquí **de redes neuronales convolucionales**: una **función de convolución** es una operación matemática que **mezcla dos funciones** para producir una tercera que resulta de superponer ambas, moverlas y, en cada posición, calcular cuánto "coinciden".

En términos metafóricos, un ecualizador de audio funciona "convolucionando" la función "canción" con una función, por ejemplo, de "filtro de bajos". La convolución combina ambos: **la canción suena pasada por ese filtro**.

- En **procesamiento de imágenes**: difuminar una foto es convolucionar la imagen con un "filtro de desenfoque".
- En **IA, las redes convolucionales** detectan bordes, colores, esquinas o patrones en imágenes.

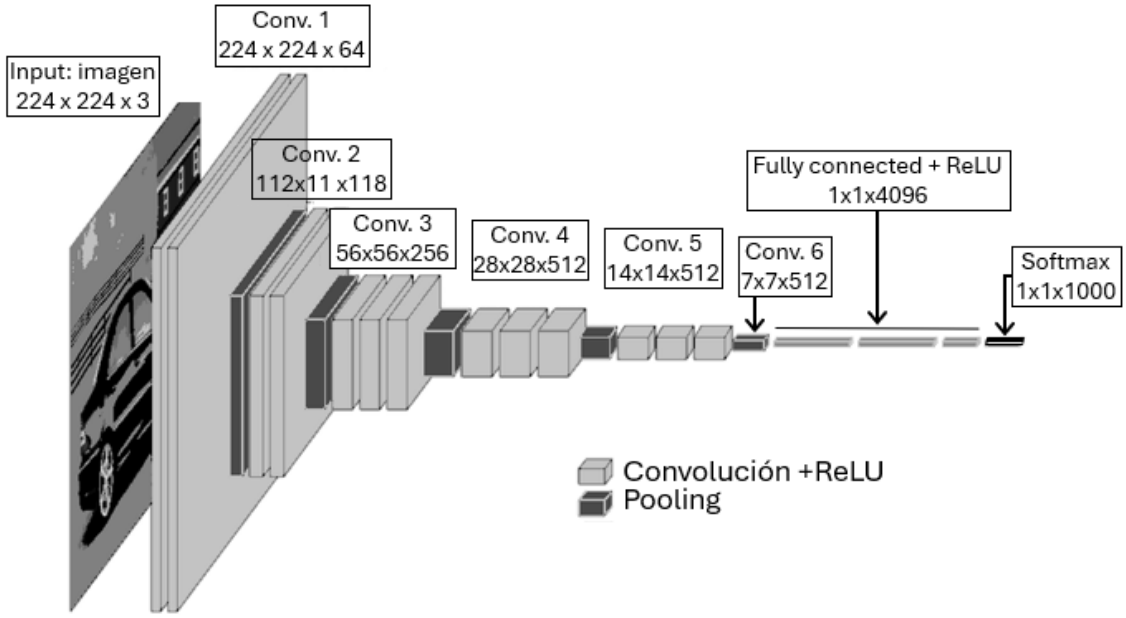
Las redes neuronales convolucionales (CNN) usan varias capas de convolución que van detectando desde patrones simples hasta los más complejos mediante la aplicación de filtros (kernel) que definen patrones de búsqueda de forma autónoma.

Luego, aplican capas de *pooling* para reducir gradualmente el tamaño de las representaciones, ganando en eficiencia

Finalmente, la clasificación o regresión se hace mediante múltiples capas *"fully connected"*, como las NN Standard, un conjunto de neuronas artificiales, cada una con sus pesos y sesgos propios, que transforman una entrada en una salida, multiplicando la primera por un peso y aplicándole una función de activación. Dado que cada neurona produce su propia salida, el resultado de cada capa es un vector que las contiene a todas.

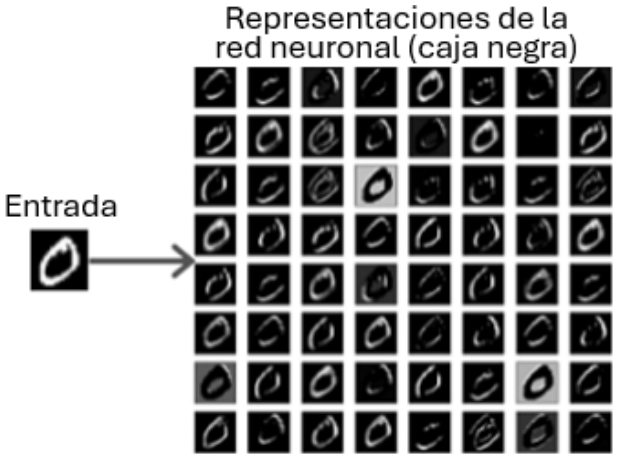
La **caja negra** aparece porque ya no puedo explicar lo que hizo el modelo en cada caso ni por qué lo hizo.

La estructura básica de una red de deep learning convolucional es la siguiente



Fuente: Classification of Palm Trees Diseases using Convolution Neural Network. Marwan Abo Zanona, Said Elaiwat, Shayma'a Younis et al.

A continuación, vemos un ejemplo real de una entrada que representa al dígito cero escrito a mano sobre el que se aplican 64 filtros convolucionales por una CNN. No es posible saber por qué fueron esos los resultados ni por qué la red eligió producirlos



Optimización e IA

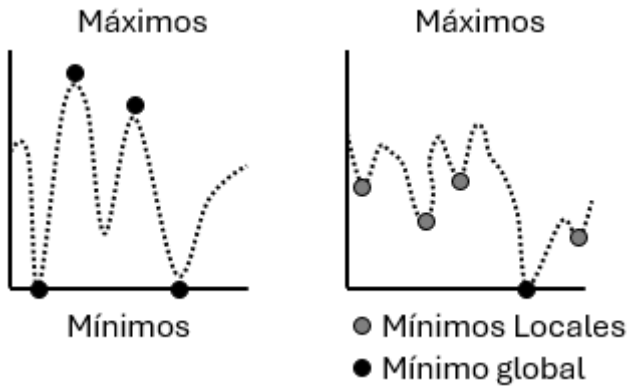
La IA **requiere** de la aplicación de procesos de optimización (por ejemplo: el algoritmo de descenso de gradientes en redes neuronales), pero **también es** optimización, como lo vimos con el problema del viajante.

Optimizar un modelo implica:

- Definir una función de error (E)
- En base a la función de error y los datos de entrada el modelo debe ser dotado de parámetros óptimos que minimicen la función de error en la salida.

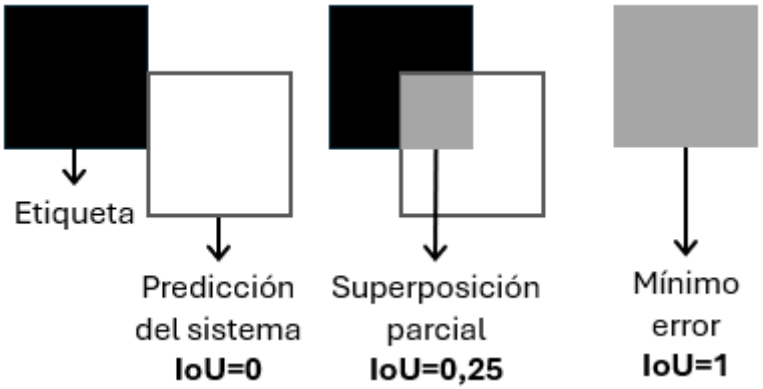
El aprendizaje que realiza la IA es, entonces, más certero cuanto más ajustados están los parámetros de la función de error. El aprendizaje nos lleva a la construcción de un modelo de IA con parámetros “óptimos” o “semióptimos”.

¿Qué significa **optimizar** una función de error? Buscar algún **mínimo de error moviendo los parámetros de esta** Se pueden distinguir mínimos locales (el mínimo para una región) del mínimo global



En la IA clásica se utilizaban algoritmos de grafos, en la moderna se utilizan técnicas más complejas:

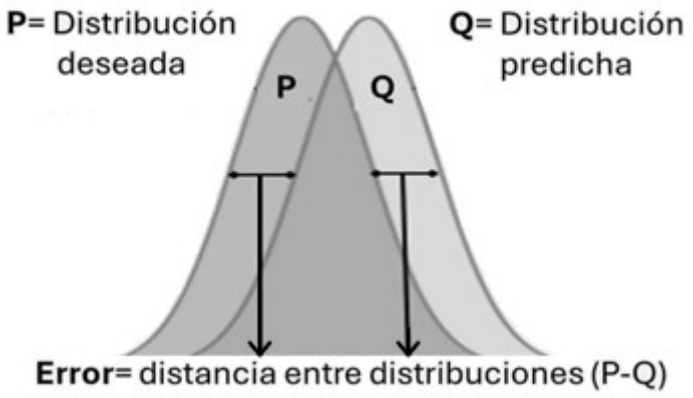
IDENTIFICACIÓN DE IMÁGENES (ALGORITMO IOU: INTERSECCIÓN SOBRE UNIÓN)



Clasificación de Objetos:

Aquí no se trata de que el modelo dibuje una "caja" alrededor del objeto (localización), sino también de que lo etiquete en la clase correcta (avión, barco, auto). Así, el error puede ser tanto la asignación de clase equivocada a pesar de una buena localización como la predicción de una clase que no corresponde a ningún objeto real (falso positivo).

Como ya sabemos, el error en la clasificación suele medirse a través del accuracy o la entropía cruzada (binaria o categórica).



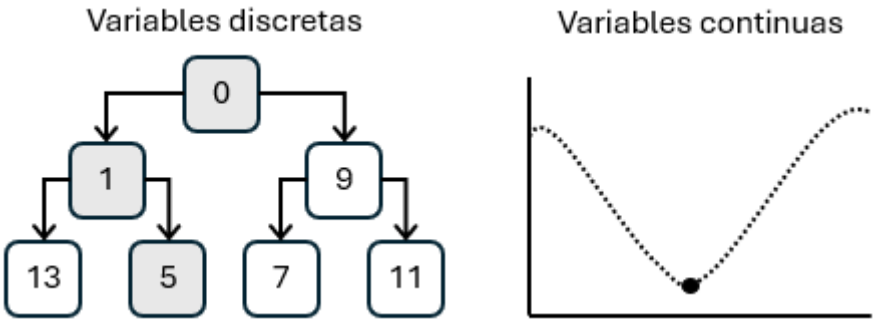
Tipos de problemas de optimización

1. SEGÚN LAS VARIABLES

Se usan para hablar de "espacios de valores" discontinuos o continuos:

1.a. ***De variables discretas*** *(¡más complejos de tratar!) – Espacio de soluciones discreto*

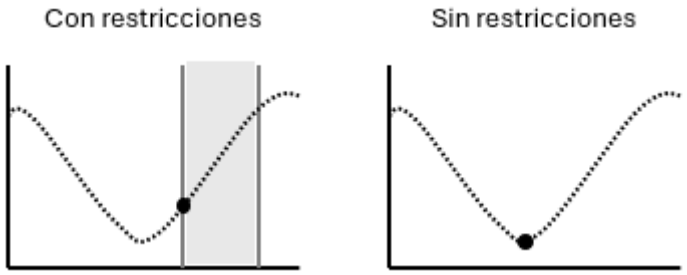
1.b. ***De variables continuas*** *(más simples de tratar) – Espacio de soluciones continuo*



2. Según las restricciones

2.a. ***Con restricciones****: el espacio de soluciones posibles está restringido (limitado)*

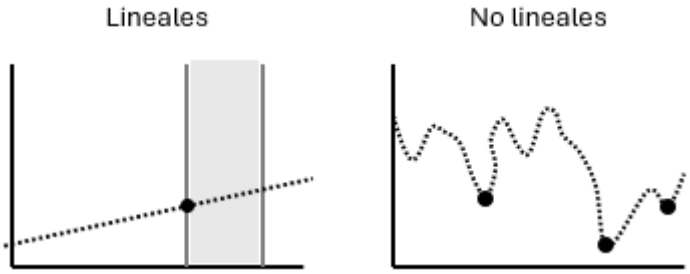
2.b. ***Sin restricciones****: el espacio de soluciones no tiene restricciones (ilimitado)*



3. Según la determinación del máximo y el mínimo

3.a. ***Lineales****: tienen un máximo y un mínimo en un espacio limitado de soluciones (son siempre acotados)*

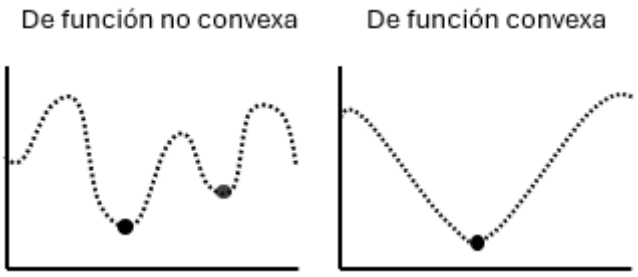
3.b. ***No lineales****: No tienen máximos ni mínimos determinados en un espacio ilimitado de soluciones posibles*



4. Según tengan uno o varios mínimos

4.a. Problemas de ***función convexa***

4.b. Problemas de ***función no convexa***



Algoritmos de solución de problemas

Pueden ser:

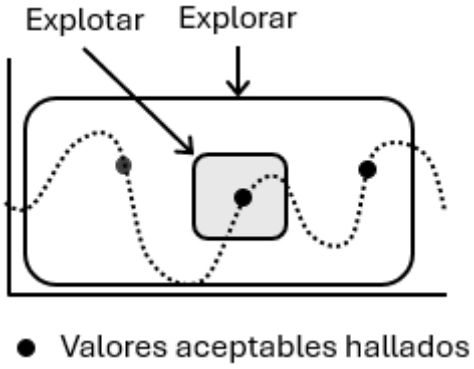
1. Generales: sirven para todo tipo de problemas

Tienen pocas asunciones, ofrecen pocas garantías y toman mayor tiempo de ejecución.

Ejemplos: Fuerza bruta (probar con todos los valores posibles y se queda con el mejor que encuentra, o sea el que más minimiza el error); Búsqueda aleatoria (evalúa valores al azar y se queda con el mejor que encuentra).

Hallada una función satisfactoria, pero nunca sé si es la más satisfactoria de todas las posibles. ¿Qué puedo hacer?

- Quedarme con ella.
- Explorar: continúo la búsqueda de óptimos sin detenerme en la zona del valor aceptable localizado
- Explotar: busco soluciones cerca de la que encontré para ver si mejoran



2. Especializados: sirven para ciertos problemas

No siempre existen. Cuando existen, no siempre son aplicables. Pueden ser exactos, pero resultan costosos.

Ejemplos: Camino mínimo en un grafo; descenso de gradiente.

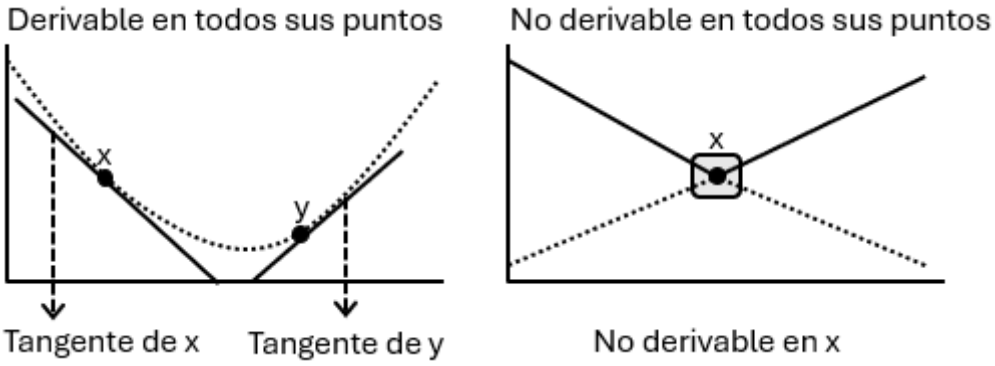
EL DESCENSO DE GRADIENTE

Es un algoritmo especializado que utiliza el **gradiente** (generalización de la derivada (se

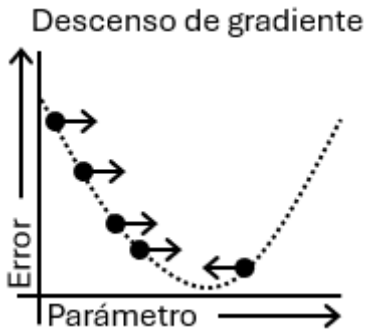
escribe con el símbolo **δ**) para funciones de varias variables como (x,y)=x2+y2 o la **derivada** (pendiente de una curva en un punto en funciones de una sola variable, como f(x)=x2 para guiar la optimización.

En términos simples, **la derivada es la pendiente** de una curva en un punto, y **el gradiente señala hacia dónde sube/baja más empinada** una superficie.

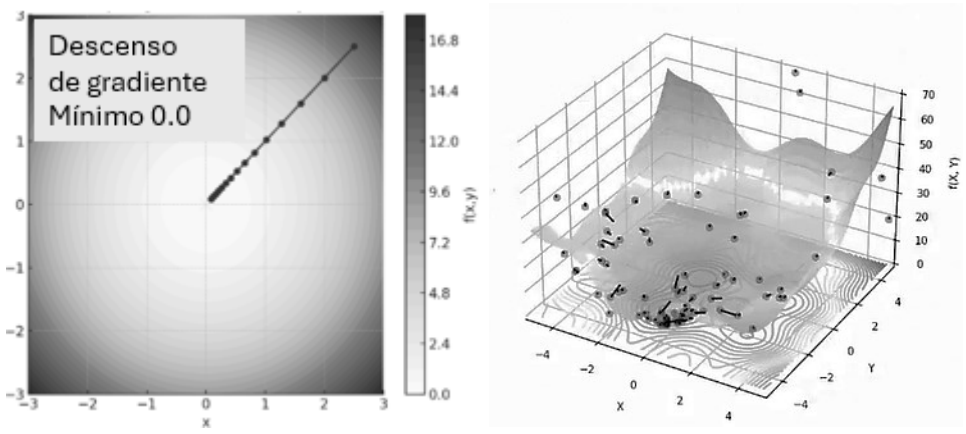
Aplicar el descenso de gradiente exige, por tanto, que la función sea derivable.



El algoritmo de descenso de gradiente actúa desplazando el error hacia abajo avanzando de forma iterativa y a una velocidad externamente determinada que es la "tasa de aprendizaje (denominado siempre como **α**) para evitar que dé grandes saltos de desplazamiento.



Descenso de gradiente (imaginar que el gráfico es un tazón visto desde arriba, se apoya una bolita en la parte superior y cae (desciende) hacia el centro (la parte más baja)



Hay que calcularlo para cada parámetro del modelo, y la función de error que se emplea está siempre asociada al problema a resolver por el modelo (ej: Problema de clasificación: Entropía cruzada; problema de regresión: Error Cuadrático Medio).

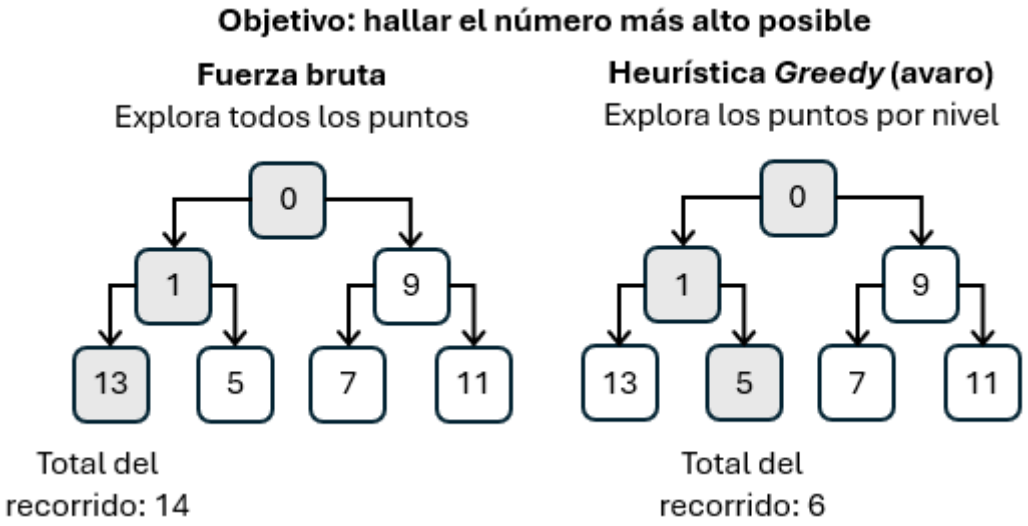
PRINCIPALES TIPOS DE HEURÍSTICAS DE OPTIMIZACIÓN

CUADRO COMPARATIVO ENTRE HEURÍSTICAS Y METAHEURÍSTICAS

	Heurística	**Metaheurística**
Especificidad	Problemas particulares	Problemas generales
Tipo de búsqueda	Local	Global
Velocidad	Alta	Baja

Las heurísticas de implementación son reglas empíricas que reducen el espacio de búsquedas y, por tanto, no garantizan soluciones óptimas.

La heurística que se escoge para cada problema está asociada a su naturaleza particular.



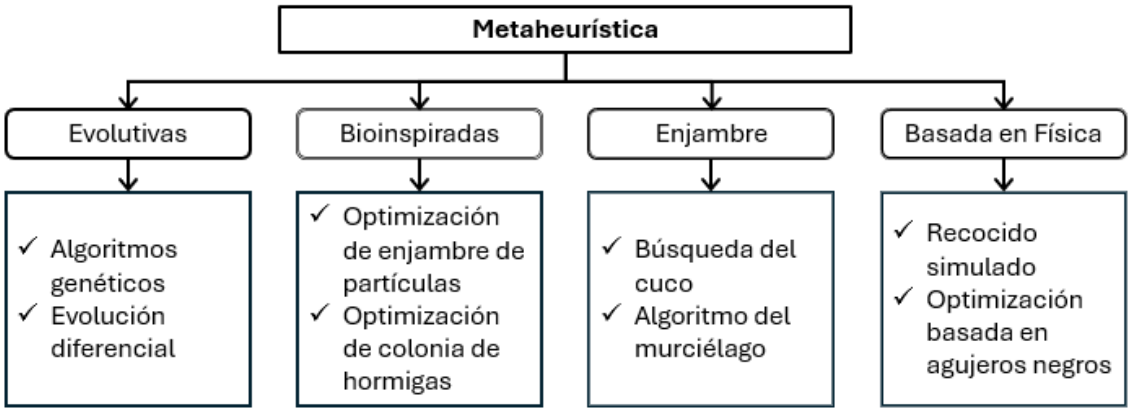
El descenso de gradiente es una heurística del tipo *greedy* ya que no vuelve sobre sus

pasos una vez que avanza en una dirección.

La **metaheurística** es una generalización de heurísticas que logra una búsqueda más global, para lo que proponen varias soluciones alternativas para un problema en lugar de brindar una solución única. Su campo de aplicación son los problemas generales.

La metaheurística establece conjuntos de reglas empíricas y reduce el espacio de búsqueda, sin llegar nunca a garantizar una solución óptima al problema general.

Hay diferentes tipos de metaheurísticas. Muchas de ellas se apoyan en soluciones basadas en la naturaleza y existen miles de algoritmos diferentes



Algoritmos genéticos: imitan la selección natural: tienes una "población" de posibles soluciones y, a través de cruces y mutaciones, se van generando nuevas soluciones cada vez mejores a los problemas.

Evolución diferencial: funciona de manera similar al anterior, pero focalizado en ajustar los parámetros mediante diferencias entre soluciones ya existentes, acelerando el proceso de búsqueda.

En redes neuronales permiten hallar los mejores valores de los pesos o para la optimización de parámetros como el número de capas o la tasa de aprendizaje.

Enjambre de partículas: imita cómo bandadas y cardúmenes se mueven en conjunto. Cada "parte" prueba una solución y comparte información con el grupo, haciendo que todos se muevan en conjunto hacia zonas de oportunidad.

Colonia de hormigas: simula cómo las hormigas encuentran caminos óptimos hacia la comida, dejando "feromonas" que guían a otras hormigas.

En redes neuronales ayudan a buscar configuraciones de parámetros que funcionan bien, evitando quedarse atrapados en malas soluciones.

Búsqueda del cuco: imita cómo los cucús ponen huevos en nidos de otros pájaros. Algunas soluciones (huevos) sustituyen a otras menos efectivas.

Algoritmo del murciélago: se inspira en el uso del sonar por los murciélagos, ajustando la búsqueda según lo que "escuchan" del entorno.

En redes neuronales se usan para explorar de forma creativa el espacio de soluciones por

fuera de lo ya conocido.

Recocido simulado: viene de la metalurgia, donde se calienta y enfría un metal para darle mejor estructura. Aquí se introducen "saltos aleatorios" controlados que permiten salir de soluciones malas para encontrar mejores opciones.

Agujeros negros: las soluciones se ven atraídas por un punto muy fuerte y van convergiendo hacia la mejor zona posible.

En redes neuronales permiten salir de mínimos locales y llegar a resultados más cercanos a óptimos.

Finalmente, es importante destacar que no existe un algoritmo de optimización que dé los mejores resultados para todos los tipos de problemas posibles: si promediamos la tasa de error de un algoritmo de optimización que ha enfrentado todas las formas de problemas existentes, ninguno será mejor que la búsqueda de un óptimo al azar.

La representación de texto

Casi todo lo que se ha hablado hasta ahora se refiere principalmente a la representación de imágenes. Vamos a ver ahora las particularidades de la representación de texto.

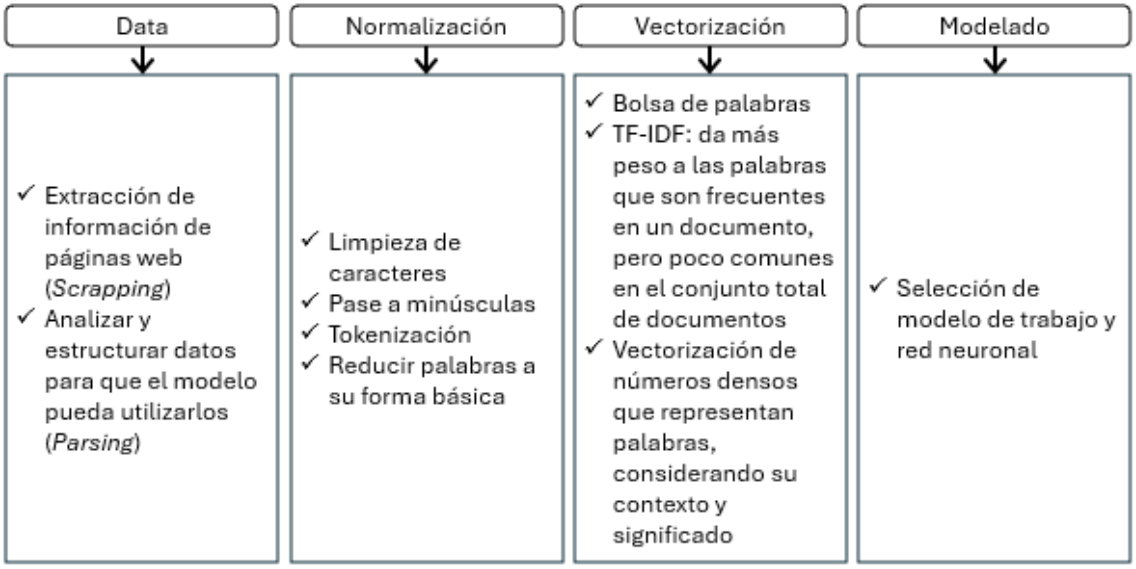
Comencemos por señalar que, a diferencia de lo que ocurre con las fotos, pero igual que sucede en los videos, en la representación de texto existe un problema de temporalidad. El orden de los datos de salida es clave.

En los videos, sigue la sucesión de imágenes

En el texto, sigue las normas sintácticas y gramaticales del idioma en que se produce el texto.

Trabajar con texto requiere convertirlo en un modelo matemático, por lo que sigue fundamentalmente cuatro pasos.

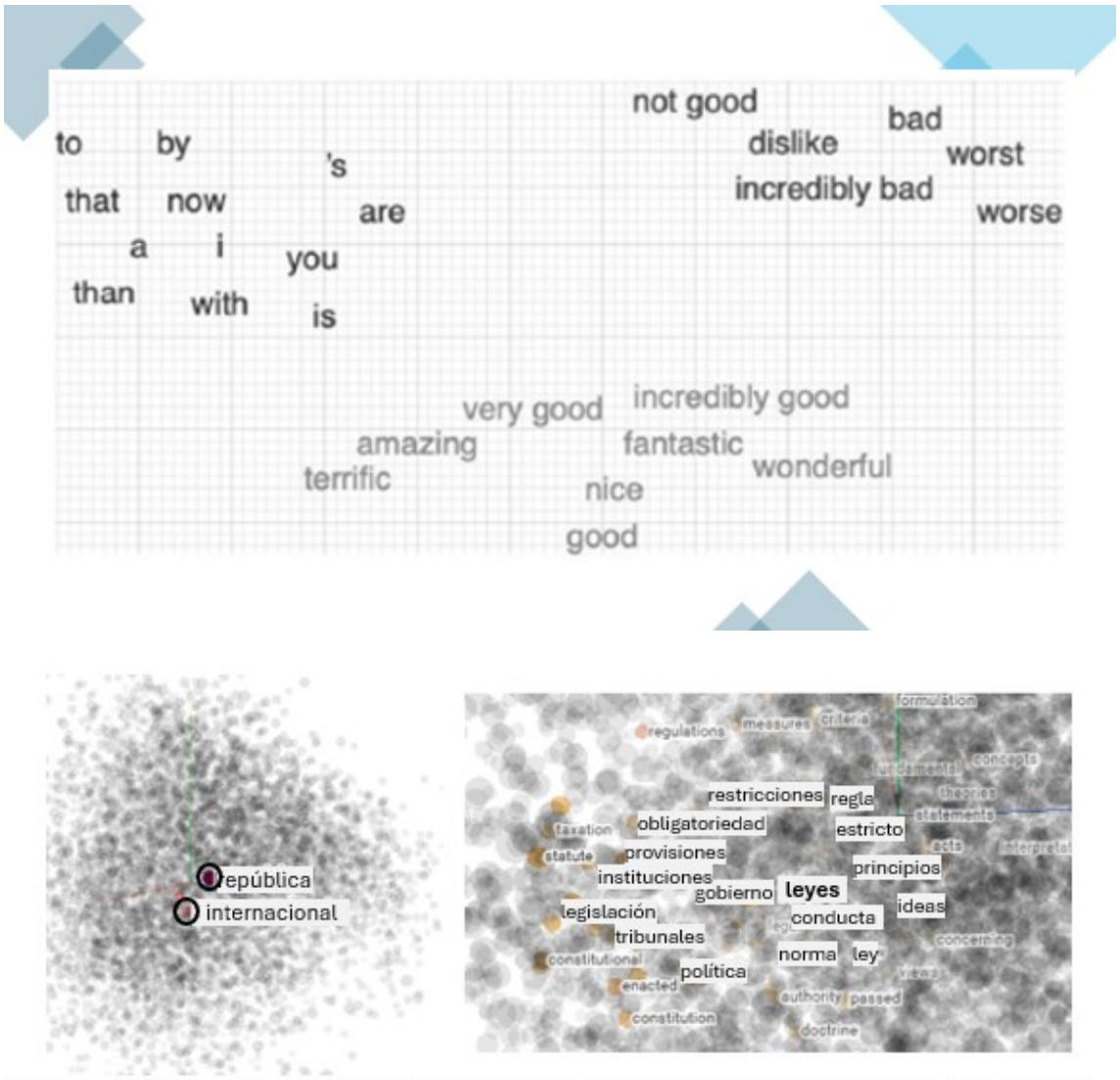
- Extracción de datos
- Normalización de datos (caracteres, mayúscula/minúscula, etc,), incluye la tokenización.
- Vectorialización: convertir texto en vectores.
- Modelado: selección del modelo a utilizar.



LA VECTORIZACIÓN

Convertir el ingreso en modelos matemáticos, por lo que el *embedding* puede aplicarse a texto, imagen, video, audio, etc. Y relacionarlos entre sí sin tener que saber qué "dicen".

El modelo que permitió dar el salto en el campo de los LLM es el "*word embedding*", una representación aprendida por el modelo del texto donde las palabras que tienen el mismo significado tienen una representación similar.



Representación gráfica reducida a 3D de una nube de embeddings donde se ve la proximidad entre las palabras "república" e "internacional" y parte de la "familia" de embeddings del "vecindario: ley"
https://projector.tensorflow.org/

No hay una medida estándar para fijar la medida de distancia entre palabras, Se suele utilizar la distancia del coseno.

La distancia del coseno

Comencemos por recordad que el coseno es una función matemática que, en términos simples, **mide qué tan "alineados" están dos ángulos o dos direcciones, en el ejemplo** "longitud de A" (‖A‖) y "longitud de B" (‖B‖)

Supongamos que tenemos dos vectores en 2 $A = (a_1, a_2), \quad B = (b_1, b_2)$ D donde A= (2,3), B=(4,1).

Busco el producto punto uniendo y multiplicando las X por un lado y las Y por el otro

$$A*B = (2*4)+(3*1) = 8+3 = \mathbf{11}$$

calculo la longitud de cada vector aplicando el teorema de Pitágoras

$\|A\|=\sqrt{2^2+3^2}=\sqrt{4+9}=\sqrt{13}\approx$**3.60** y $\|B\|\ \sqrt{4^2+4}=\sqrt{16+1}=\sqrt{17}=$ **4.12**

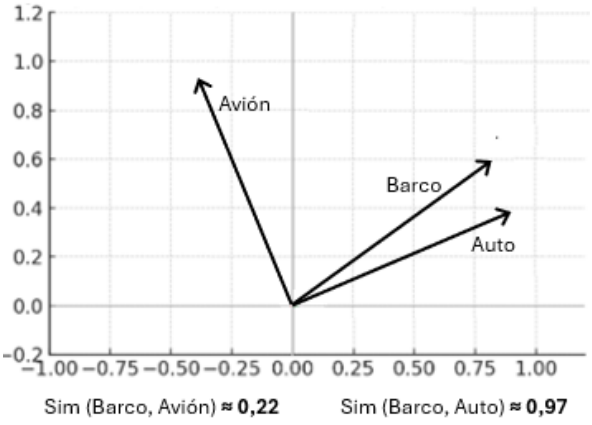
Finalmente, calculo el coseno del ángulo AB

$\cos\theta = (\|A\|*\|B\|) / (A*B) = 11/(3.6+4.12) = 11 / 14.83 =$ **0.74**

Utilizando el coseno se puede comparar la **orientación** de los vectores, no su tamaño. Para ello cada palabra se representa como un vector y se mide el **coseno del ángulo** entre los dos vectores, y la distancia se representa como $\text{distancia} = 1 - \cos(\theta)$

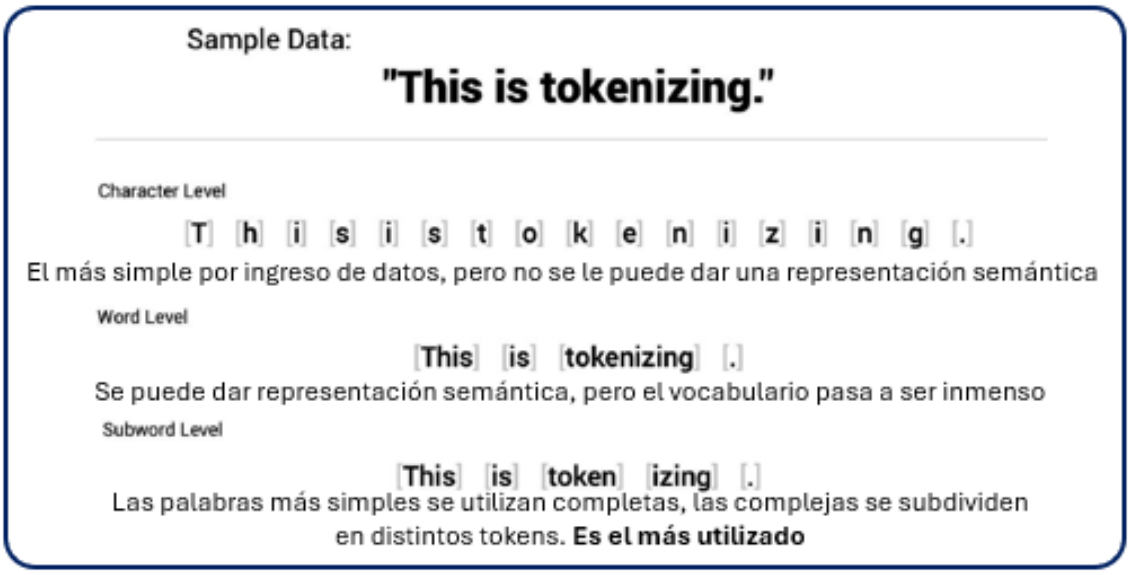
Si la distancia es 0, son idénticos en orientación; si es 1, son totalmente diferentes.

Ejemplo con las palabras "Avión", "Barco" y "Auto"



El espacio entre *embeddings* se denomina "espacio latente" y nos permite movernos entre dos vecindarios de vectores (en imágenes, por ejemplo para "envejecer mi cara" desplazándome del vecindario de "mi edad" hacia el vecindario de "personas mayores", en video, por ejemplo, para recomendar videos teniendo en cuanta qué películas has visto en comparación con las que han visto otros que vieron la misma que vos). Hoy existen bases de *embeddings* públicas.

LA TOKENIZACIÓN



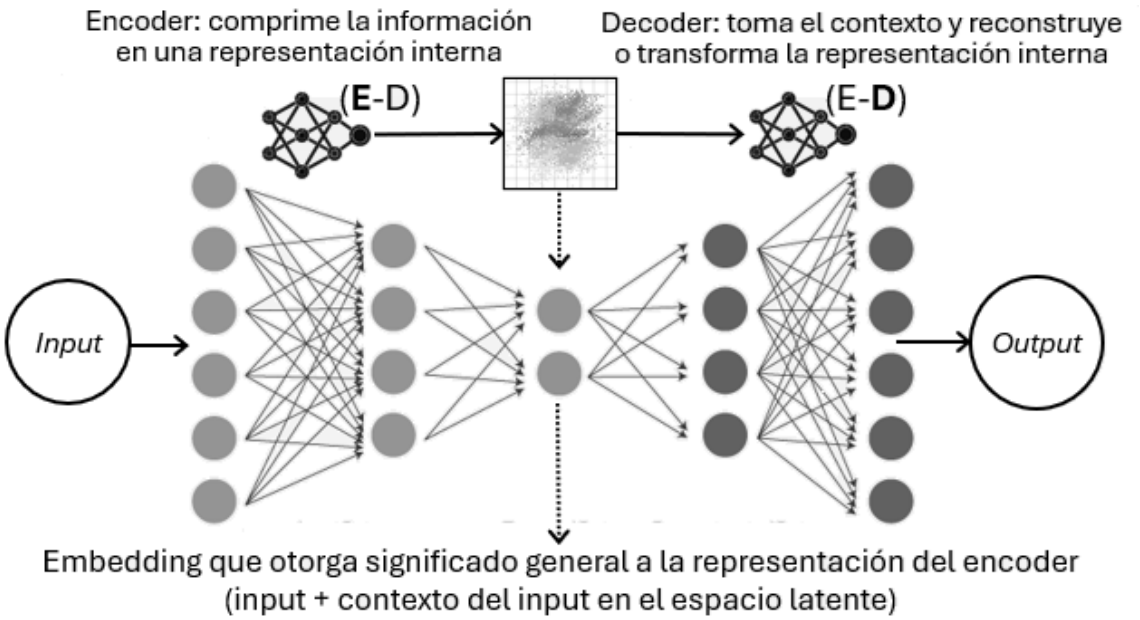
(Ver https://platform.openai.com/tokenizer para mostrar y contar tokens de ChatGPT)

Las representaciones temporales

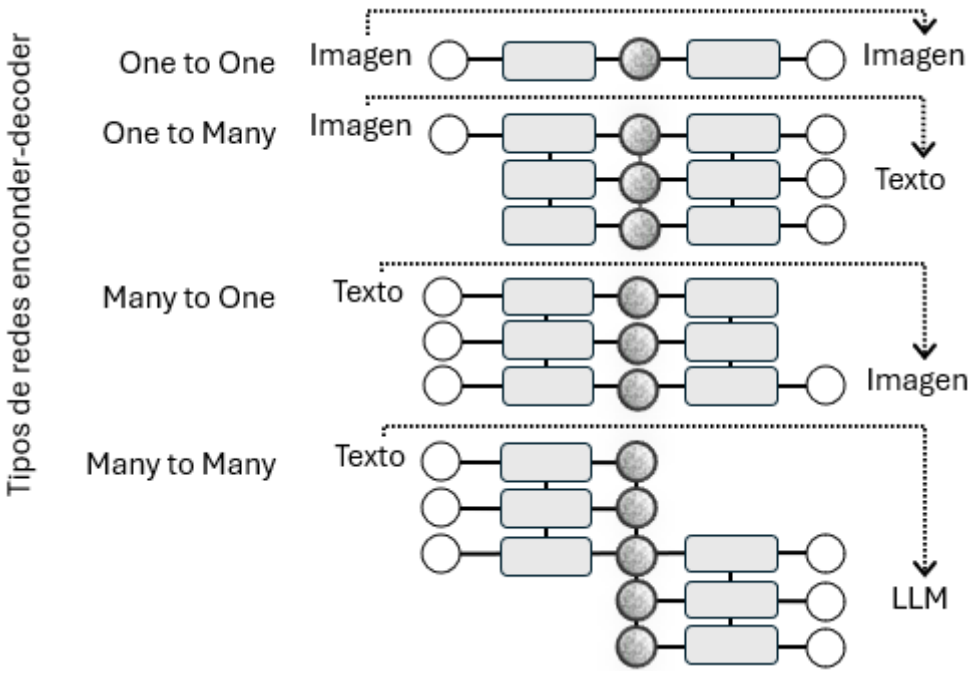
En videos y oraciones aparece implicado el elemento temporalidad (respecto de imágenes y palabras, respectivamente). Para trabajar sobre la temporalidad se utilizan las llamadas redes *Encoder-Decoder*

Una red Encoder-Decoder funciona básicamente en 5 etapas

- *Provisión de inputs*: entra la información original (ejemplo: una frase en inglés).
- *Encoder*: comprime esa información en una representación interna (el "significado").
- *Vector latente*: espacio intermedio donde queda la esencia compactada.
- *Decoder*: toma la representación y genera una salida (ejemplo: la frase traducida al español).
- *Provisión de output:* como resultado final.

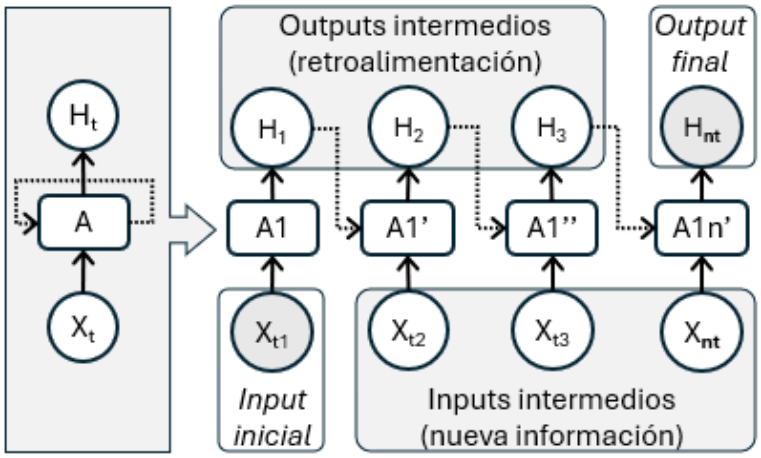


La configuración y capas de las redes encoder-decoder está directamente relacionada con el tipo de input y output que reciben y proveen:



LAS REDES NEURONALES RECURRENTES (RNN)

Son una función matemática recursiva: para predecir la primera salida toma la primera entrada; para predecir la segunda salida se combina el segundo y la salida del procesamiento de la primera (salida intermedia generada por la previsión anterior).



El problema que tenían las RNN es que les costaba captar las dependencias a largo plazo: dados los sucesivos pasos, cuando la cadena era muy larga, la función comenzaba a "ensuciarse" con los resultados de las operaciones previas de la propia red neuronal

Como respuesta surgen las redes LSTM (*Long-Short Term Memory*) que agrega una activación con una función sigmoidea (arroja valor entre 0 y 1) y permite a la red ir "limpiando" las partes que ya no son útiles de las entradas anteriores que tiendan a cero. En cada paso se agrega solo que es trascendente del nuevo *input* y se elimina lo que ya no es trascendente. Se reduce el peso de la entrada intermedia y se limpia la entrada nueva antes de integrarla.

LA ATENCIÓN (*ATTENTION*)

Una mejora posterior al surgimiento de las LSTM y que puede integrarse en ellas. Es la pieza clave de los modelos más nuevos de LLM: la **atención**, un mecanismo que permite que una red **se enfoque en las partes más importantes de la información de entrada cuando genera una salida**.

El nacimiento de la atención significó un salto adelante, porque vino a resolver un problema de "falta de memoria" que afectaba a las redes recurrentes, que al analizar frases largas comenzaban a "olvidar" información producida en los primeros pasos, información que, en ocasiones, era muy relevante en los pasos finales. Había un problema de "falta de memoria" como nos puede afectar a nosotros cuando no recordamos de qué iba una charla al inicio tras haber estado conversando con amigos durante horas.

Por ejemplo, tomemos la siguiente frase

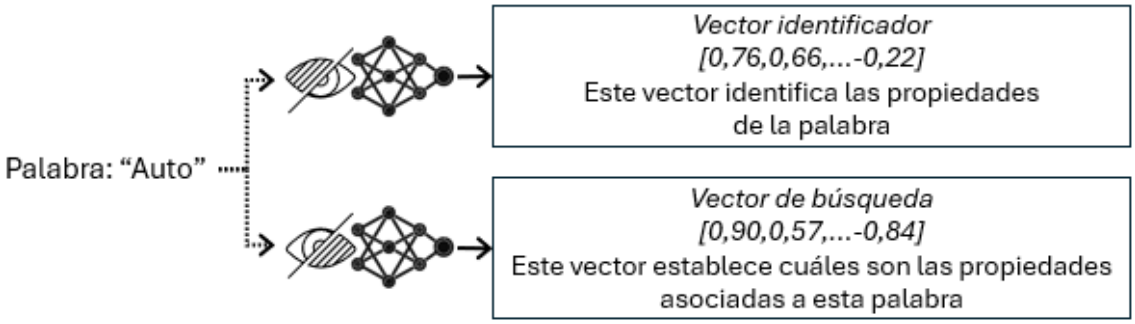
*El **auto** azul donde viajaban los cuatro amigos se había alejado ya varios kilómetros de la ciudad antes de llegar al parador, donde hicieron un descanso para comer y aflojarse, y fue entonces cuando llegaron a identificar que alguien había saboteado intencionalmente **su** motor.*

Se trata de 44 palabras y hay 42 entre auto y su (que se refiere al auto). Las RNN perdían esa relación dadas las múltiples entradas y salidas secuenciales entre una y otra palabra.

La atención vino a resolver este problema al introducir un análisis simultáneo de todas las palabras contra todas las palabras, y al hacerlo asigna pesos a cada una de ellas que le indican "estas son las relaciones entre palabras más importantes" (cuando se trata de imágenes, para encontrar un avión le presta más atención a las alas que al cielo alrededor).

Como resultado, la red entiende mejor las **dependencias largas,** como ocurre cuando se trata de conectar palabras que están lejos en una oración, pero además establece conjuntos de relaciones: no solo identifica que "su" refiere al auto, sino también encuentra que las relaciones entre "auto" y "motor" son próximas, al igual que la relación entre "auto" y "viajaban" y entre "auto" y "azul", pero no hay relaciones importantes entre "auto y "comer", aunque sí las hay entre "comer", "aflojarse" y "amigos".

Para hacerlo, lo que se hizo fue crear dos redes a fin de automatizar el proceso de búsqueda de relaciones al entrecruzar todos los términos en simultáneo



Para describir el proceso vamos a usar una frase más simple "el avión blanco vuela alto". Cada palabra de la frase se convierte en un vector (*embedding*) y el modelo genera tres versiones de cada vector:

- **Query (Q)**: se pregunta "¿Qué estoy buscando ahora?"
- **Key (K)**: se pregunta "¿Qué información ofrece esta palabra?"
- **Value (V)**: se pregunta ¿Qué información aporta esta palabra?"

Luego, compara cada **Query** con todos los **Keys** para calcular **similitudes, lo que le dice** "qué tanta atención debe dar a cada palabra" asignándole pesos, que se aplican sobre los **Values** para generar una combinación ponderada focalizada solo en la parte relevante del texto.

Ejemplo con "el avión vuela alto" (The plane flies high)

Cuando el modelo va a traducir *"plane"*:

En el *encoder*:

1. El ***query*** es el estado actual ("quiero la palabra avión en inglés").

2. Se compara con los *keys* de todas las palabras de entrada para focalizar la atención:

2.a. Key(el): baja similitud.

2.b. Key(avión): alta similitud.

2.c. Key(blanco): media similitud

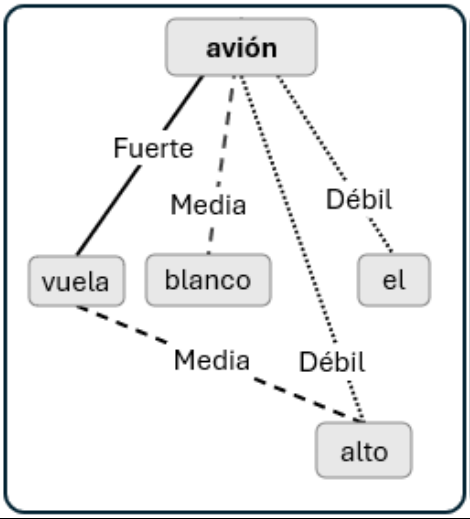
2.d. Key(vuela): media similitud.

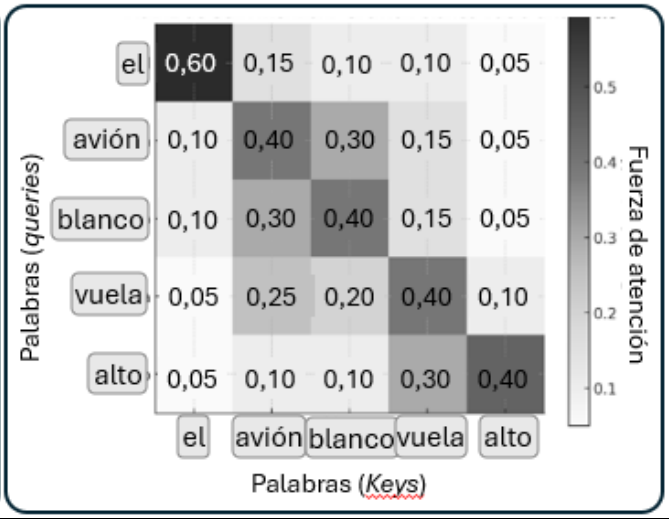
2.e. Key(alto): baja similitud.

Asigna en consecuencia pesos a cada palabra aplicando una función Softmax: [0.10, 0.40,

0.30, 0.15, 0.05]

El *decoder* los aplica a los *values*, que combinan toda la información en un nuevo vector que agrega como información qué palabra es la más importante de la frase y cómo las demás se relacionan con ella (relación media o débil) para darle al modelo el contexto exacto que necesita en cada paso que realiza.





LOS MODELOS TRANSFORMERS

Son un tipo de red neuronal creada en 2017 que se basa en el uso de **self-attention** para procesar datos en paralelo, en vez de hacerlo palabra por palabra.

Cómo funciona una red transformer

Input: toma una frase, ejemplo: *"El avión vuela alto"* y convierte a cada palabra en un vector (*embedding*).

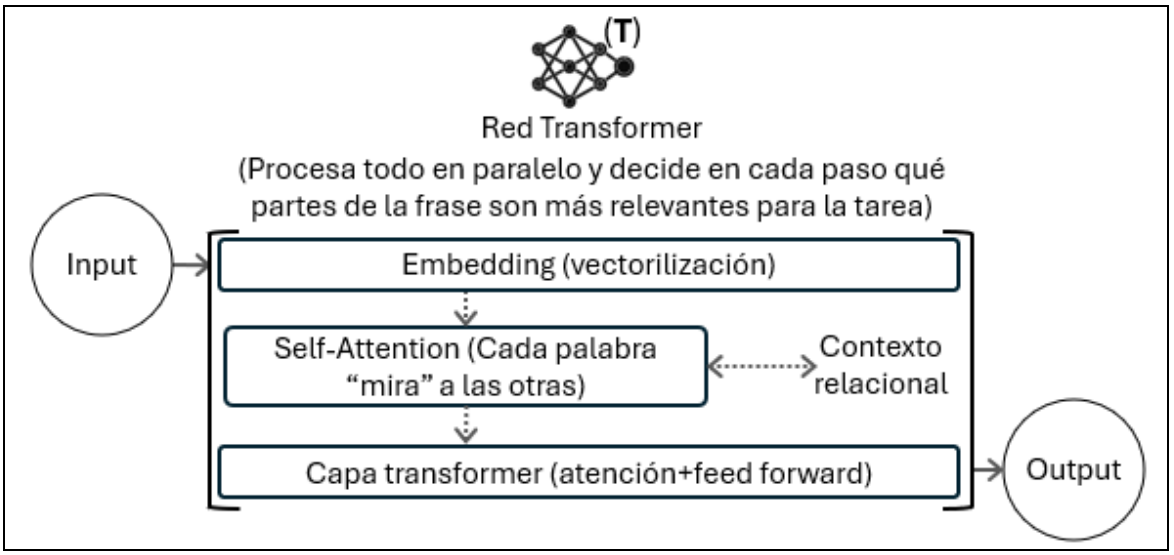
Self-Attention: cada palabra "mira" a todas las demás y decide cuánto le importan. *"avión"* se fija sobre todo en *"vuela"*, y un poco en *"alto"*.

Capas de Transformer: repite el mecanismo de atención en varias capas, refinando las representaciones.

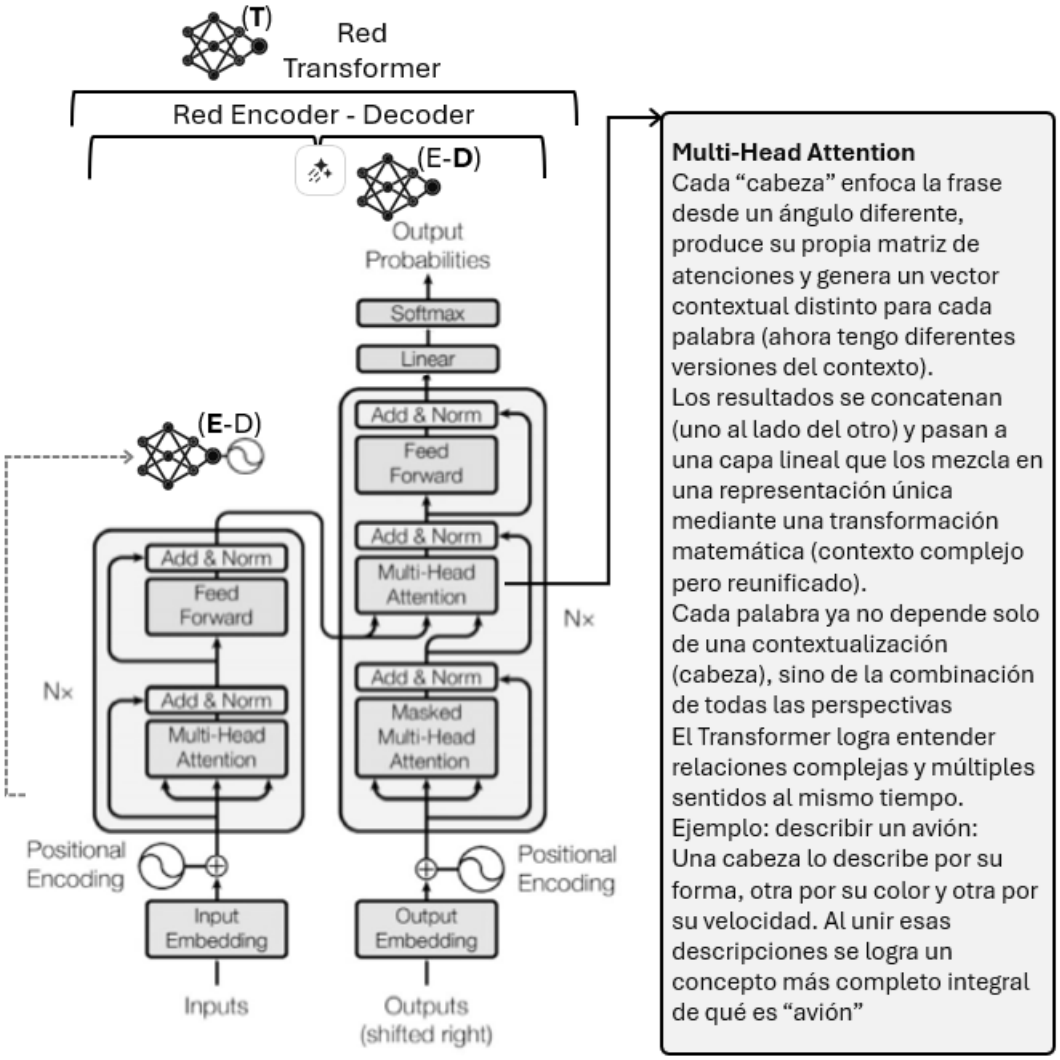
Se agregan también *capas de feed-forward* para procesar los resultados.

Output:

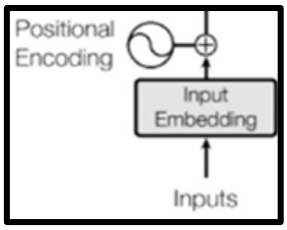
- si es un *encoder*: genera una representación de la frase (útil para clasificación, búsqueda, embeddings)
- si es un *decoder*: genera texto palabra por palabra (traducción, chat, resumen)
- si es un *encoder–decoder*: combina ambas (ej: traducción automática).



Los transformes pueden funcionar sobre redes *encoding-decoding* e incluir capas *multi-head attention*

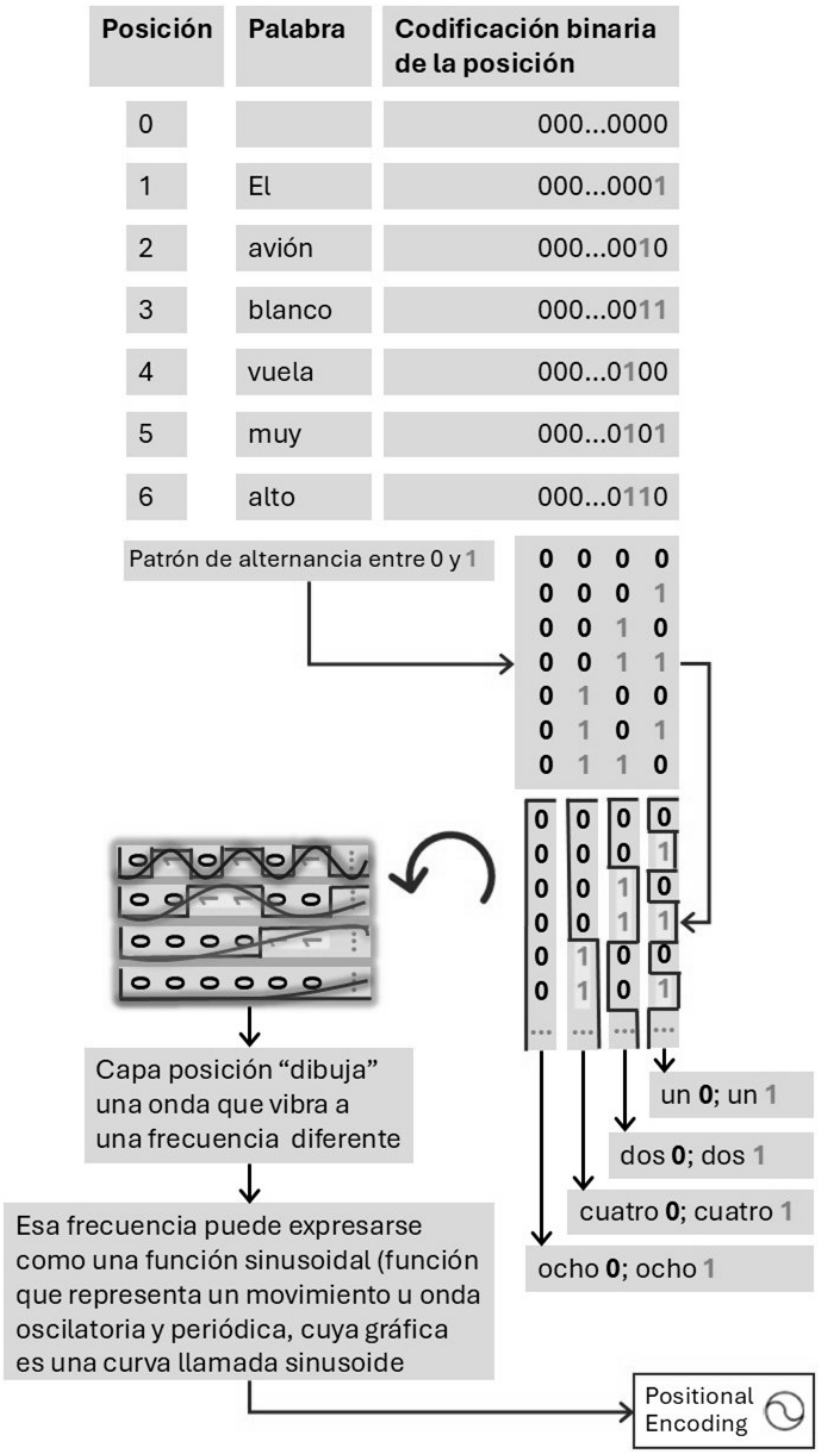


Fuente basado en *Attention is all you need*



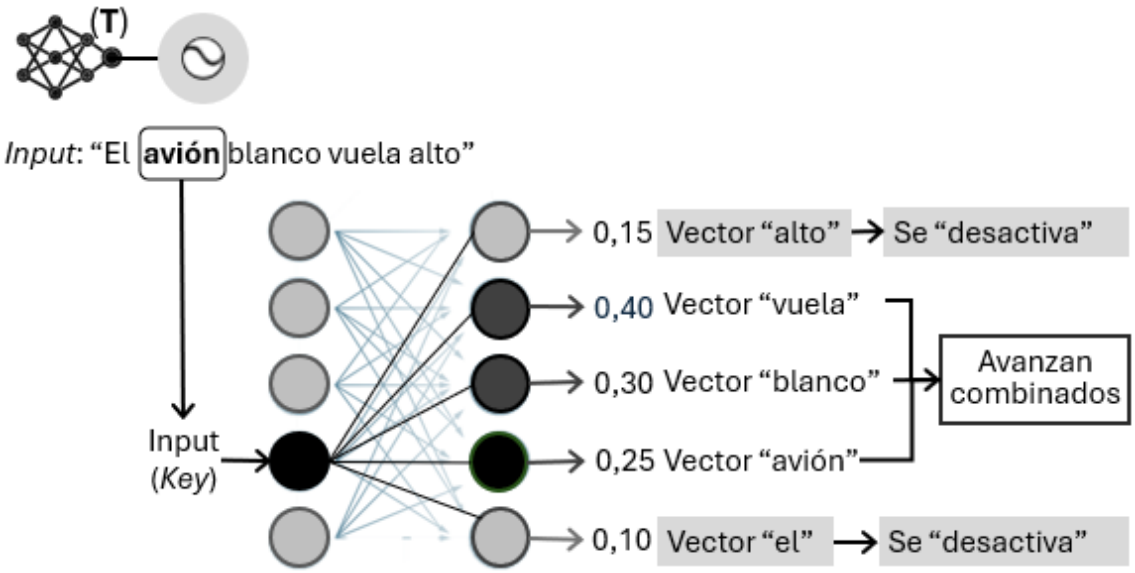
En estas redes, la clave de la mejora este en el procesamiento de los inputs de entrada, que ahora toman todas las palabras que forman la frase al mismo tiempo (paralelización en lugar de secuencialización), las transforman en vectores (*input embedding*) y les asigna su posición relativa en la palabra nuevamente (*positional encoding*)

La vectorización ya es tradicional, la paralelización genera el problema de que el modelo "pierde" el orden de las palabras al tratarlas todas al mismo tiempo. El *positional econding* resuelve eso asignando al vector de cada palabra una codificación, en binario, que marca su orden en la oración.



Con ello, el vector de cada palabra incluye la información de su posición en la frase integrada con los vectores de las palabras con las que tiene más relación, todo en una sola salida.

Podemos graficarlo así:



Los transformers son la nueva arquitectura de red que se encuentra detrás de todas las actuales redes LLMs.

Parte 3: Interpretabilidad de modelos de IA

Introducción

La **interpretabilidad** es un campo de investigación de la IA cuyo objetivo es desarrollar técnicas para comprender los estímulos por los cuales los modelos de caja negra generan determinadas salidas.

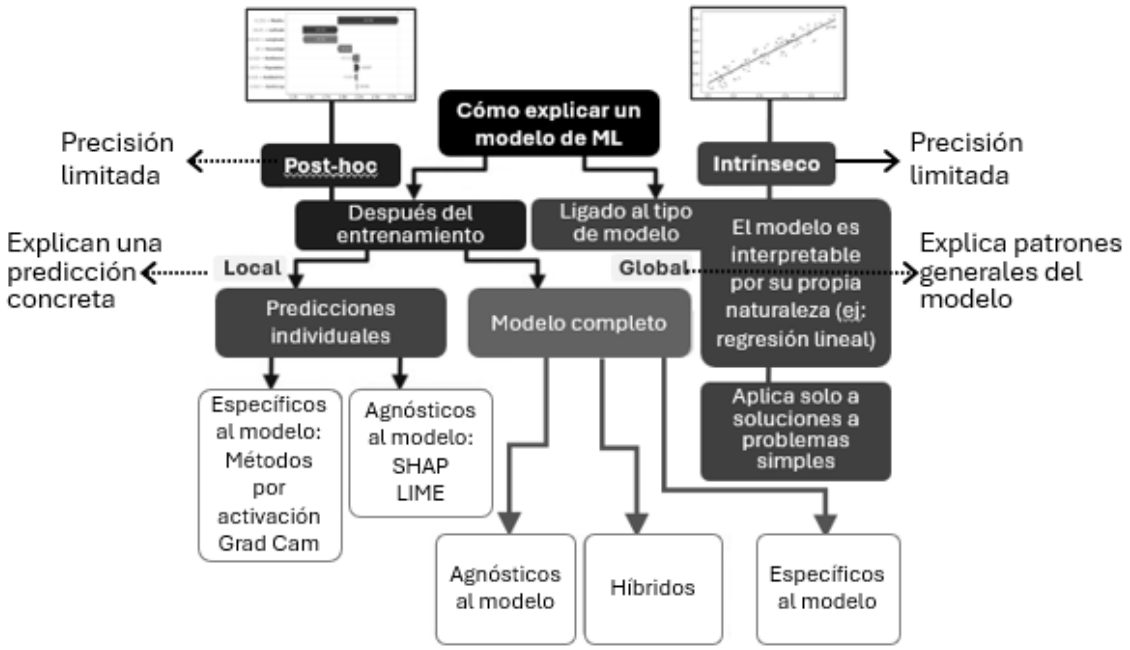
Lo que busca es entender lo que pasa internamente en el modelo, a diferencia de *explainability,* que busca explicar los resultados sin centrarse en el proceso que llevó a ellos.

La interpretabilidad permite realizar un diagnóstico sobre la forma en que las capas ocultas en los modelos contribuyen a las salidas, brindando una forma de entenderlos.

La interpretabilidad es un requisito ineludible cuando se trata de sistemas que toman decisiones de alto riesgo en los que se debe proveer de *feedback* a los usuarios finales respecto a cómo se generaron los resultados (ejemplo, cuando se utiliza una IA para decidir la extensión de penas de prisión).

Permite reconocer sesgos en los datos y en los modelos, y contribuye a una IA segura (*AI Safety*). Si no logramos "explicar" los modelos de IA es más difícil que las personas los adopten y los profesionales confíen en sus resultados, especialmente cuando se trata de adoptar decisiones que implican alto riesgo o sensibilidad.

Taxonomía de la Interpretabilidad



La regresión logística como ejercicio de interpretabilidad

En matemáticas y estadística, **regresión** significa un método para **predecir un resultado** a partir de datos. Por ejemplo: si tengo la edad de una persona, puedo intentar predecir cuánto mide. Eso se llama **regresión lineal**, porque el resultado puede ser cualquier número dentro de un rango

La **regresión logística** es una función matemática, también llamada **sigmoide** ya que genera una curva con forma de S, que transforma cualquier número en un resultado entre **0 y 1**. Se usa cuando queremos **predecir las posibilidades de un resultado para el cual solo hay dos opciones** (0/1, verdadero/falso).

La fórmula matemática de la regresión logística es:

$$p = \frac{1}{1 + e^{-z}}$$

- Si ***x*** es muy negativo, el resultado se acerca a 0.
- Si ***x*** es muy positivo, el resultado se acerca a 1.
- Si ***x***=0, el resultado es 0.5.

El modelo de regresión logística requiere que calculamos primero una **combinación lineal**:

$$z = b_0 + b_1 \cdot x$$

x = horas de estudio

b_0 = ordenada al origen

b_1 = coeficiente que indica cuánto influye cada variable en el resultado final (hora de estudio en aprobar o no aprobar un curso)

Entrenamos el modelo y obtuvimos:

$\mathbf{b_0}$=−4; $\mathbf{b_1}$=1

Entonces

$$z = -4 + 1 \cdot x$$

Ahora podemos aplicar la función logística

Si la variable ***x asumió un valor de 5 tenemos la siguiente regresión lineal***

*Z = -4+1*x = z= -4+1*5 =* ***z=-4+5 = z=1***

De esta se deriva la siguiente regresión polinómica

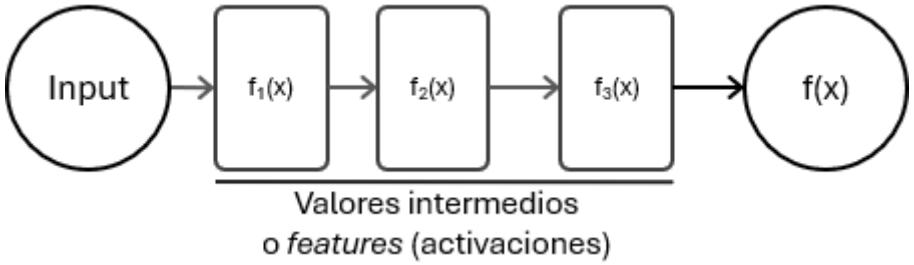
$$p = \frac{1}{1+e^{-z}} = p = \frac{1}{1+e^{-1}} \approx p = \frac{1}{1+0.367} \approx 0{,}73$$

Esto es aproximadamente 73% de probabilidad de que algo pase si la variable es 5

Si p≥0.5p decimos que la respuesta es 1; pero si p<0.5 decimos que la respuesta es cero. Como 0,73 es mayor que 0,5 entonces el resultado binomial es 1.

Cuando se aplica a un modelo de múltiples variantes, la regresión logarítmica considera que cambiar una característica no afecta a las demás, lo cual es una suposición generalmente inválida en conjuntos de datos reales.

ELEMENTO BÁSICO DE UNA RED NEURONAL PARA INTERPRETABILIDAD



Donde los *features* son valores intermedios del proceso que hace el modelo.

- δ f(x) / δ x indica cómo afecta la **entrada** a la **salida**
- δ f_i(x) / δ x indica cómo afecta la **entrada** al ***feature*** $_1$
- δ f(x) / δ f_1(x) indica cómo afecta el ***feature*** $_1$ a la **salida**
- δ f_1(x) / δ f_2(x) indica cómo afecta el ***feature*** $_1$ al ***feature*** $_2$

Enfoques para interpretar modelos

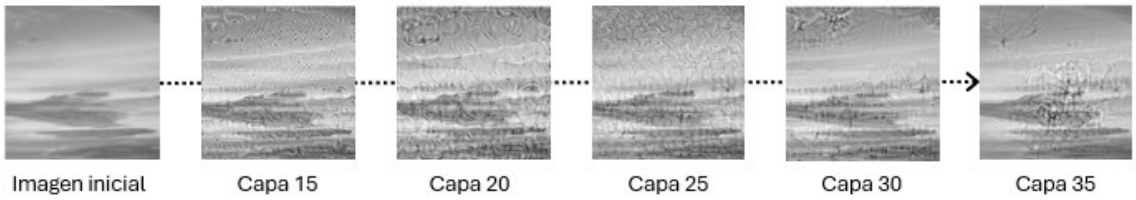
MÉTODOS GENERATIVOS

Permiten visualizar y comprender las características que aprenden los modelos.

d) DEEP DREAM

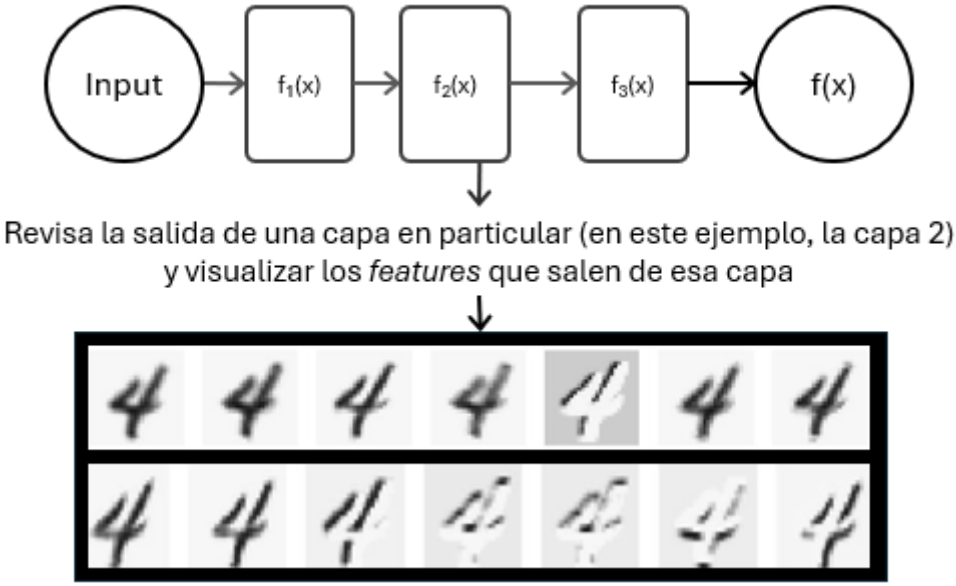
Deepdream es similar a los métodos utilizados para "engañar" a un clasificador de imágenes solo que en lugar de aplicar *backpropagation* a la imagen original para minimizar la pérdida que genera una etiqueta maliciosa, retropropagamos a la imagen original para maximizar una capa de activación en el centro de la red, produciendo un ascenso de gradiente intencional que maximiza la función (capa) interna de la red para saber qué le "gusta" recibir como insumo.

Ejemplo:



Fuente: *Deepdream and Mechanistic Interpretability*. https://swe-to-mle.pages.dev/posts/deepdream-and-mechanistic-interpretability/ - **Nota**: las imágenes y los cambios se observan mucho mejor en el enlace donde se pueden identificar los colores y tonos originales

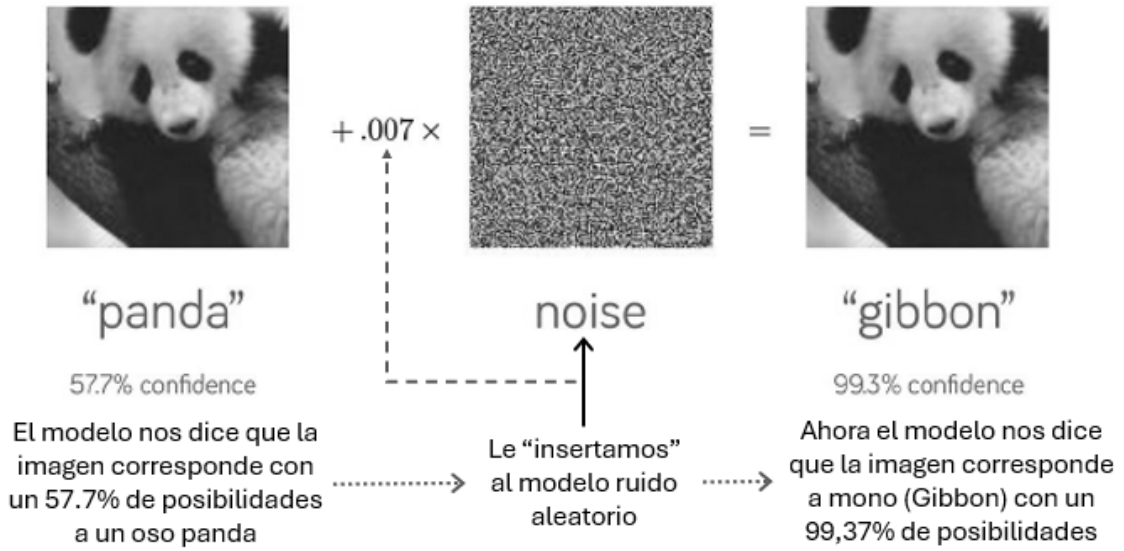
MÉTODOS POR ACTIVACIÓN



Como se observa en la imagen, algunos features dan prioridad a la textura, otros a las formas, otros al fondo, etc. Así vemos qué se activa en cada momento (filtro) de aprendizaje de la red neuronal.

MÉTODOS ADVERSARIOS

Permiten validar la robustez y vulnerabilidad del modelo ante posibles vulnerabilidades, perturbando la imagen de entrada intencionalmente para engañar al modelo y hacer que la clasifique como una clase errónea. Como resultado, obtenemos la tendencia general de preferencia del modelo.



Ejemplo: uso de 2 máquinas de imágenes médicas con diferentes calibraciones, si el sistema es vulnerable, analizará bien las imágenes de la máquina con la que se lo entrenó, pero no será posible aplicarlo a la otra imagen sin errores.

Pueden generar posibles problemas de sobre-entrenamiento.

Enfoques post-hoc

Se basan en derivadas (o gradientes) y perturbaciones a la entrada (ej. oclusiones)

Implican trabajar más a nivel de *feature atribution:* resaltan los píxeles que fueron relevantes para la clasificación de una imagen y explican predicciones (probabilidades) individuales, atribuyendo el grado de influencia de cada *feature* de entrada (píxel) en el cambio de la predicción de una clase (negativamente -lo que es raro- o positivamente -que es lo normal).

Si se aplican a modelos *black-box*, no requieren acceder a sus parámetros internos (pesos); pero sí cuando se aplican a modelos de caja blanca.

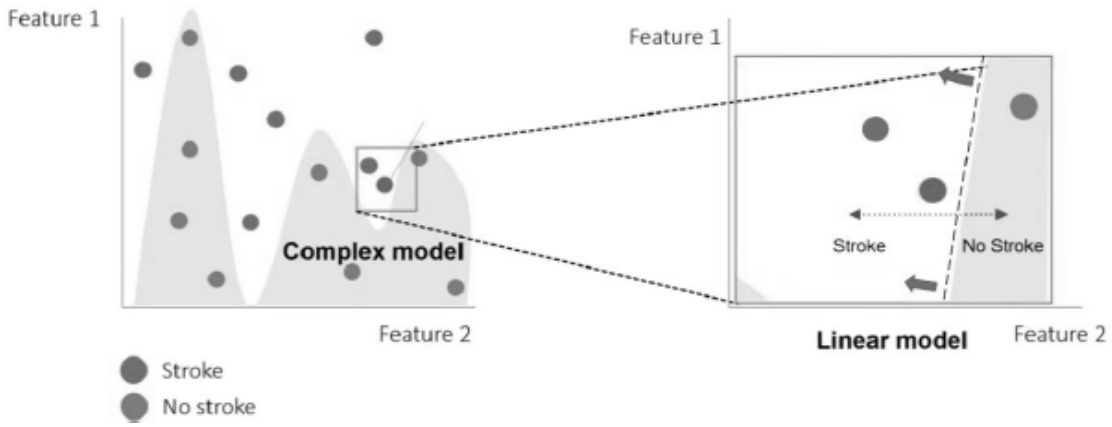
e) LIME (LOCAL INTERPRETABLE MODEL-AGNOSTIC EXPLANATION)

Se utilizan más para datos tabulares que para trabajar con imágenes

Utiliza la técnica de "sustitución" (*surrogate*) de modelos para obtener predicciones aproximadas localmente y explicables

Entrena un modelo sustituto "lineal y explicable" (ej. regresión lineal) para una zona local (vecindario) para una predicción individual y específica.

LIME genera un nuevo mini-dataset para el entrenamiento del modelo lineal



Puede explicar bastante bien el modelo de la "vecindad" trabajada, pero no lo que ocurre fuera de ella

f) SHAP (SHAPLEY ADDITIVE EXPLANATIONS)

Los valores de SHAP son consistentes (esto es, pueden agregarse para obtener una visión global del aporte de cada *feature* a la predicción realizada por el modelo.

Su origen se encuentra en la Teoría de Juegos Cooperativos, particularmente en los llamados "Valores de Shapley". Donde se considera la situación en que un grupo de personas (*coalition*) coopera en un juego para ganar un premio, y lo que buscamos es una solución justa para distribuirlo, que nos indique el promedio en que cada jugador ha contribuido al triunfo (el "*marginal value*" de cada participante de la coalición), teniendo en cuenta que la forma en que se combinen los miembros del grupo da resultados diferentes.

Cada jugador puede aportar más o menos al resultado total, pero ese aporte depende de con quién se junte. El valor de Shapley calcula **cuánto aporta cada jugador en promedio a todas las posibles coaliciones** que podrían formarse. Se hace probando todas las formas en que los jugadores podrían combinarse y midiendo cuánto "extra" aporta cada jugador al sumarse. Finalmente, el valor de Shapley de un jugador expresa el **promedio de su contribución marginal** a lo largo de todas las posibilidades.

Ejemplo de cálculo de valores de Sharpley: Tenemos tres jugadores: **A, B y C** (los *features* del modelo), un premio de 300 y los valores de cada jugador y coalición:

{A} = 20	{B} = 10	{C} = 40	
{A,B} = 110	{A,C} = 130	{B,C} = 120	{A,B,C} = 150

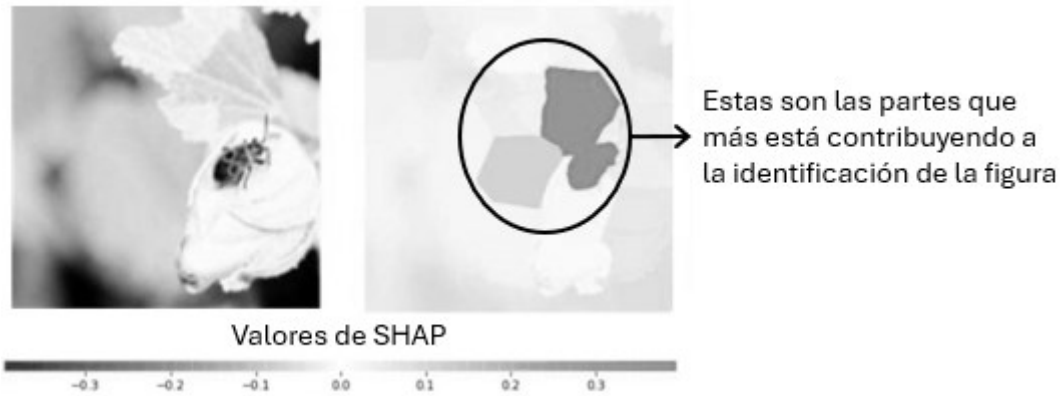
El "juego" es el modelo de caja negra en sí mismo y el "premio"·es la predicción que hace el modelo.

Cada jugador solo ya aporta algo, pero las combinaciones generan aún más valor. Dado

que son 3 jugadores, tenemos **6 órdenes** de entrada, para cada una de las cuales podemos calcular las contribuciones marginales de los jugadores.

A–B–C	A–C–B	B–A–C	B–C–A	C–A–B	C–B–A
{A}=20 Contrib. =20 {A,B}=110 Contrib. de B=90 {A,B,C}=150 Contrib. de C=40	{A}=20 Contrib. =20 {A,C}=130 Contrib. de C=110 {A,C,B}=150 Contrib. de B=20	{B}=10 Contrib. =10 {B,A}=110 Contrib. de A= 100 {B,A,C}=150 Contrib. de C=40	{B}=10 Contrib. =10 {B,C}=120 Contrib. de C=110 {B,C,A}=150 Contrib. de A=30	{C}=40 Contrib.=40 {C,A}=130 Contrib. de A=90 {C,A,B}=150 Contrib. de B=20	{C}=40 Contrib.=40 {C,B}=120 Contrib. de B=80 {C,B,A}=150 Contrib. de A=30
A=20 B=90 C=40	A=20 B=20 C=110	A=100 B=10 C=40	A=30 B=10 C=110	A=90 B=20 C=40	A=30 B=80 C=40
Sumamos los aportes de cada jugador y los dividimos por la cantidad de conmutaciones: **A:** (20 + 20 + 100 + 30 + 90 + 30) = 290 → 290 / 6 ≈ **48.3** **B:** (90 + 20 + 10 + 10 + 20 + 80) = 230 → 230 / 6 ≈ **38.3** **C:** (40 + 110 + 40 + 110 + 40 + 40) = 380 → 380 / 6 ≈ **63.3**				48.3+38.3+63.3≈ 150 (total combinado de los tres). Premio: 300/150=2. La contribución marginal de cada jugador *2 es la parte del premio que le toca	

Puede hacerse a nivel de imagen, permutando pixel por pixel, pero es muy costoso y hay millones de combinaciones. Es poco usado, pero cuando se lo utiliza no se lo hace a nivel de pixeles sino a nivel de gradientes.



FUENTE: EXPLAINABLE AI (XAI): ARE WE THERE YET? HTTPS://BLOGS.MATHWORKS.COM/DEEP-LEARNING/2023/05/08/EXPLAINABLE-AI-XAI-ARE-WE-THERE-YET/ - **NOTA**: LAS IMÁGENES Y LOS CAMBIOS SE OBSERVAN MUCHO MEJOR EN EL ENLACE DONDE SE PUEDEN IDENTIFICAR LOS COLORES Y TONOS ORIGINALES

Enfoques basados en gradientes

Saliency (Maps) o Vanilla Gradients

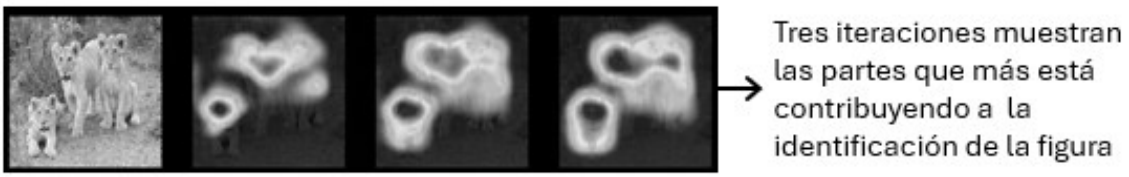
Método que genera un mapa de importancia (*saliency*) mediante una sola iteración de *backpropagation* a la red.

Se basa en el cálculo del gradiente del puntaje de una clase con respecto a la imagen de entrada: muestra cómo cambia la salida (completa o para una clase en particular) de un modelo al cambiar un pixel de la entrada.

Como si fuera un resaltador nos destaca qué partes de una entrada (imagen o texto) fueron más importantes para que la IA tomara su decisión.

Responde a la pregunta: "Si la IA clasificó esto como un avión, ¿qué píxeles de la foto fueron los que más influyeron?".

Para hacerlo se parte de la entrada que se da a la IA y se calcula cómo cambia la salida si modifico un poco cada una de sus partes. Aquellas en las que un cambio pequeño afecta mucho la salida se consideran más importantes y salen pintadas con colores brillantes (rojo = muy importante, azul = poco importante).



Fuente: What is Saliency Map? https://www.geeksforgeeks.org/machine-learning/what-is-saliency-map/ - **Nota**: las imágenes y los cambios se observan mucho mejor en el enlace donde se pueden identificar los colores y tonos originales

Este resultado nos ayuda a entender en qué se fijó la IA al hacer la clasificación, verificar que se esté enfocando en los elementos que queremos (el avión, no una nube en el fondo) y para detectar sesgos (nos muestra si un modelo se fija más en la piel de una persona que en un rasgo médico donde debería focalizarse).

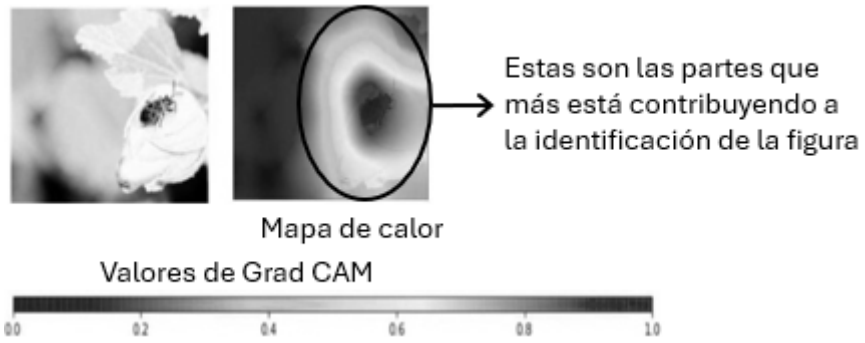
Grad-CAM (Gradient-weighted Class Activation Mapping)

Similar a *saliency*, pero el *backpropagation* no se hace sobre la imagen de entrada sino siguiendo los cambios a través de las capas convolucionales que se quiera insertar en el modelo y el lugar dónde se las ubique (puede ser hasta la última). Utiliza información interna de la red como gradientes y *features*.

Se basa en la hipótesis de que cada *feature* aprende patrones en los anteriores que le sirven de entrada. Si se analizan las regiones, los *feature maps* de las capas se pueden explicar en qué se fija el modelo para tomar sus decisiones. Para llegar a un resultado único, promediamos los mapas que salieron de cada capa analizada.

Por ejemplo, le damos al modelo la imagen de un insecto sobre una flor. La Grad-CAM

mira las últimas capas convolucionales (las que detectan partes y formas más abstractas) y calcula los gradientes que indican cuánto influye cada filtro en la predicción de la categoría. Con esos gradientes como insumo dibuja un mapa de calor que muestra qué partes fueron más importantes para la identificación del objeto.



FUENTE: EXPLAINABLE AI (XAI): ARE WE THERE YET? HTTPS://BLOGS.MATHWORKS.COM/DEEP-LEARNING/2023/05/08/EXPLAINABLE-AI-XAI-ARE-WE-THERE-YET/ - **NOTA**: LAS IMÁGENES Y LOS CAMBIOS SE OBSERVAN MUCHO MEJOR EN EL ENLACE DONDE SE PUEDEN IDENTIFICAR LOS COLORES Y TONOS ORIGINALES

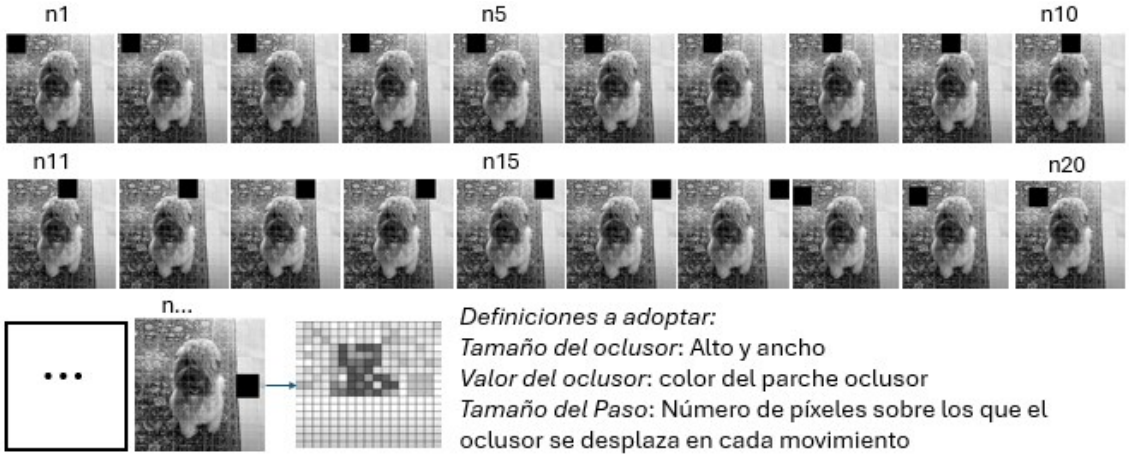
Enfoques basados en la perturbación

OCLUSIÓN

Genera máscaras que ocultan solamente un parche de la imagen por vez, con un tamaño fijo de ventana de oclusión y de salto.

Se toma la imagen de entrada, pero ahora se cubren pequeñas partes de esta con un parche (oclusor) y se vuelve a entrar la imagen con el oclusor en el modelo y se observa cuánto cambia la predicción.

Si la predicción baja mucho al tapar cierta zona, esa zona era importante para el modelo, y lo sabemos porque nos da como resultado un mapa de calor que muestra qué partes fueron más críticas. Es como jugar a adivinar una foto tapando pedacitos: si al tapar las alas ya no parece un avión, entonces las alas eran importantes.



FUENTE: *OCCLUSION-BASED SALIENCY MAPS | EXPLAINABLE AI FOR COMPUTER VISION* (VIDEO)

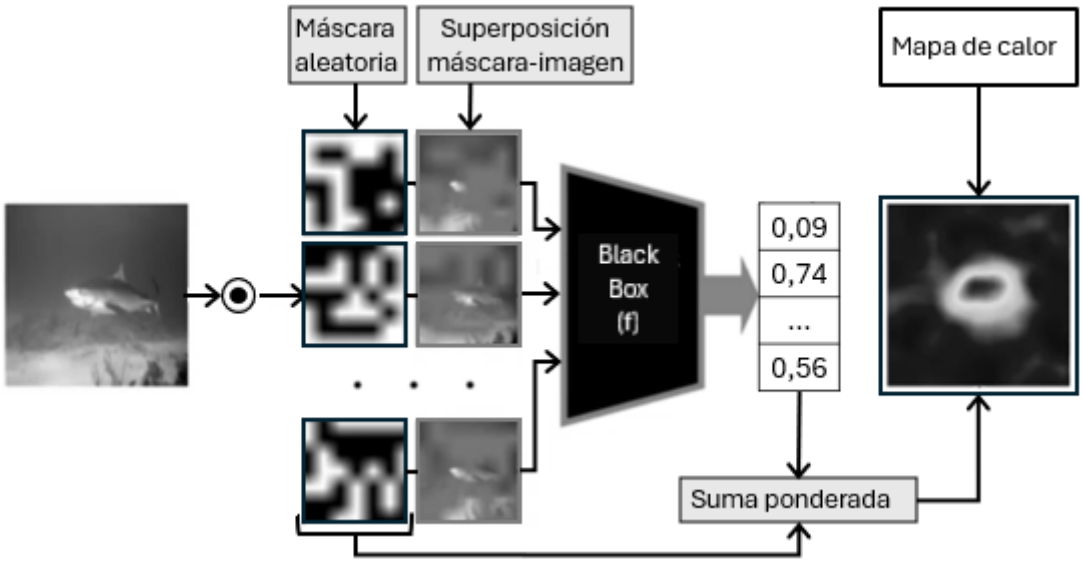
HTTPS://WWW.YOUTUBE.COM/WATCH?V=XXOQD3TCJDI - **NOTA**: LAS IMÁGENES Y LOS CAMBIOS SE OBSERVAN MUCHO MEJOR EN EL ENLACE DONDE SE PUEDEN IDENTIFICAR LOS COLORES Y TONOS ORIGINALES

RISE (RANDOMIZED INPUT SAMPLING FOR EXPLANATION OF BLACK-BOX MODELS)

Aplica juntos oclusión y SHAP

- Se pasa cada imagen enmascarada al modelo.
- Se mide cuánto afecta cada máscara al resultado (ej: "probabilidad de avión").
- Se combinan todas las máscaras y se obtiene un **mapa de importancia probabilístico** ponderando cada máscara y sumando las ponderaciones.

Es como hacer miles de versiones borrosas de la imagen de entrada y ver en cuáles de ellas el modelo sigue calificando correctamente.



FUENTE: *RISE: RANDOMIZED INPUT SAMPLING FOR EXPLANATION OF BLACK-BOX MODELS (AI PAPER SUMMARY)*(VIDEO) HTTPS://WWW.YOUTUBE.COM/WATCH?V=BQDK35G3LZY - **NOTA**: LAS IMÁGENES Y LOS CAMBIOS SE OBSERVAN MUCHO MEJOR EN EL ENLACE DONDE SE PUEDEN IDENTIFICAR LOS COLORES Y TONOS ORIGINALES

Este modelo tiene gran sensibilidad al tamaño de las máscaras y a la cantidad de máscaras que se apliquen a la imagen de entrada.

Enfoques Intrínsecos

Fuerzan a los modelos a que sean interpretables desde el momento de su diseño

CONCEPT WHITENING

Este método, si bien es intrínseco, **requiere modificar el modelo**, y por tanto no es un método post-hoc puro, sino que se encuentra en una zona intermedia entre los modelos intrínsecos y los post-hoc.

El *concept whitening* restringe/modela el espacio latente de un modelo obligándolo a que logre aprender a representar conceptos clave mediante el uso de vectores de concepto. En otras palabras, permite que las unidades internas de una red neuronal estén alineadas con conceptos humanos interpretables: en vez de que las neuronas intermedias aprendan características "opacas" (patrones que no sabemos describir), **el Concept Whitening las obliga a representar conceptos que sí entendemos** (como "ala", "ventana", "nube").

Este método de interpretabilidad toma una capa intermedia de la red, le aplica un proceso de *whitening* (decorrelación de activaciones), que reorganiza las neuronas y luego, asocia explícitamente cada neurona de la capa a un concepto que el usuario define. **Al entrenar esas neuronas no solo ayudan a la clasificación final, sino que también nos dicen qué concepto concreto activaron**.

Al aplicar CW a una capa inferior, tiende a capturar información de bajo nivel, como las características de color o textura de estos conceptos, y a medida que avanzan las capas, aumenta el nivel de la información que captura.

Así, ante la imagen de un tiburón en el mar, un modelo *black box* diría que hay un "95% de posibilidades de que sea un tiburón", pero no sabríamos *en qué se fijó*. En cambio, el CW genera una neurona a la que asocia con "primer aleta dorsal", otra con "aleta caudal", otra con "abertura nasal"… y como resultado nos dice que "Este es un tiburón porque detecté la primera aleta dorsal (alta activación), la aleta caudal (baja activación) y la abertura nasal (media activación)."

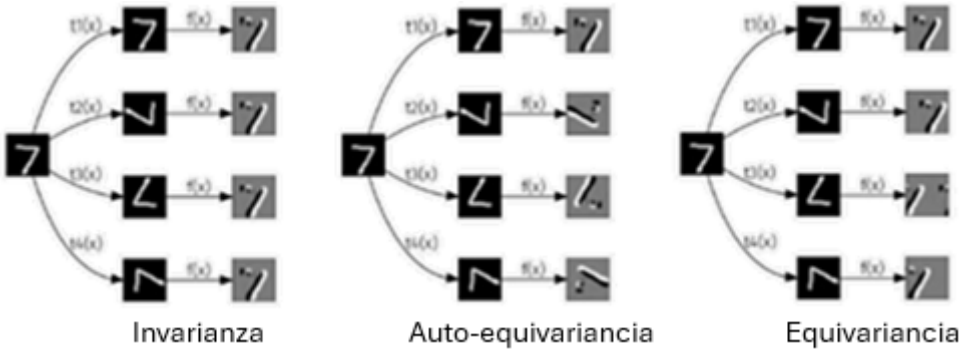
A diferencia de otros métodos que vimos, no se limita a elaborar mapas de calor, sino que "**traduce" la lógica interna de la red a conceptos comprensibles por humanos.**

Invarianza y equivarianza

Una red es:

- **Invariante** a un conjunto de transformaciones si su salida no cambia cuando se le aplica una transformación a su entrada.
- **Auto-equivariante** cuando la salida cambia siguiendo exactamente el sentido de la transformación introducida
- **Equivariante** si su salida cambia de forma predecible cuando introducimos una transformación.

Estas características hacen a la fiabilidad del resultado producido por el modelo, y la aplicación de una u otra opción depende del caso de trabajo (queremos invarianza para que un 6 no se confunda con un 9, pero podemos preferir la auto-equivarianza para la identificación de figuras que puedan estar rotadas y la equivariancia para la identificación de caras).



Arquitecturas invariantes

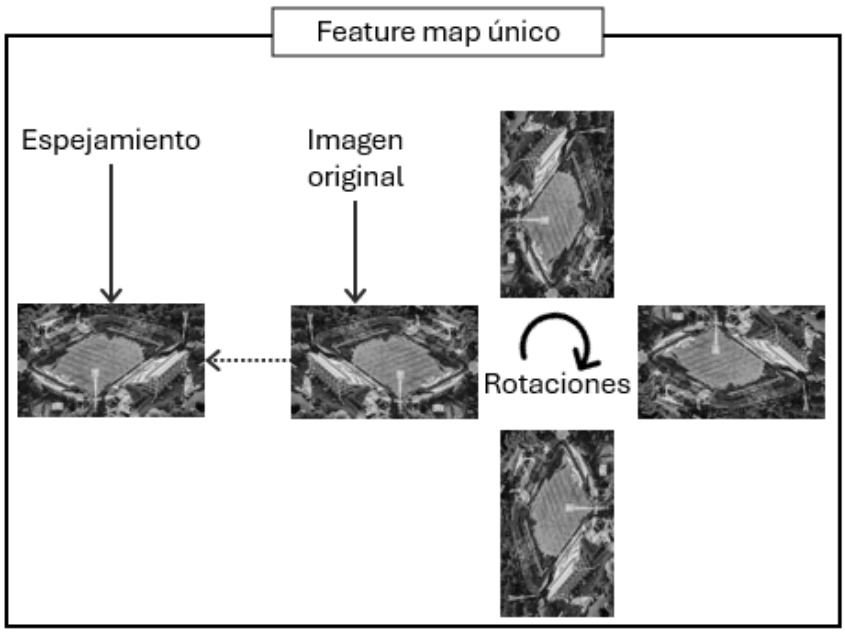
Comencemos por recordar que las **Redes Neuronales Convolucionales (CNNs)** son un tipo especial de red neuronal diseñada para procesar datos con estructura en forma de cuadrícula, como imágenes, mediante **convolución**, una operación matemática que permite que la red "vea" pequeñas regiones locales de la imagen (filtros o kernels), en lugar de mirar todos los píxeles a la vez.

Funcionan aplicando filtros (*Kernel*) que son "detectores" que recorren la imagen en busca de patrones como bordes verticales, esquinas, colores específicos, etc. Cuando un kernel reconoce en la imagen lo que está buscando se ilumina una zona en un mapa. A medida que avanzamos en la red, los kernels se combinan generando patrones más y más complejos: de buscar bordes pasan a buscar formas, luego partes y finalmente objetos completos, para terminar, dando una clasificación a la imagen dentro de un grupo (*"esto es un barco", "esto es un avión"*).

GROUP EQUIVARIANT CONVOLUTIONAL NETWORKS (GCNNS)

Las redes convolucionales funcionan muy bien para imágenes (u otros inputs) equivariantes, pero cuando el objeto está rotado, reflejado o tiene otros cambios, necesita aprender cada caso por separado, con muchos ejemplos. Las GCNNs vienen a resolver ese problema ampliando las CNNs para que no solo sean "equivariantes" a traslaciones, sino también a otras **transformaciones simétricas, como rotaciones en cualquier grado y reflexiones en espejo.**

Cuando la CNN aplica filtros de convolución sobre la imagen, la CGNN los amplifica a sus pares simétricos, logrando que los filtros reconozcan las transformaciones de manera tal que la red aprende que son la "misma cosa" vista desde otro ángulo. Como resultado, se puede entrenar la red con menos datos y mejorar su capacidad de generalización de patrones, aunque aparezcan en orientaciones distintas.

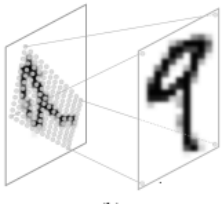


SPATIAL TRANSFORMERS NETWORK (STN)

Las **STN** son un módulo que puede insertarse en una CNN para darle la capacidad de **aprender a transformar la imagen por sí misma** antes de procesarla, de tal modo que hacen a las imágenes espacialmente invariantes

Actúa en tres pasos:

- Primero una **Localization Network:** una subred decide qué transformación aplicar (ej.: "rota 30° y haz zoom en esta parte").
- Luego un **Grid Generator:** crea una cuadrícula de coordenadas para aplicar la transformación indicada.
- Finalmente, un **Sampler:** reordena los píxeles de la imagen de acuerdo con la cuadrícula (rota, escala, recorta, etc.).



Como resultado, se obtiene una versión transformada de la imagen original, que la STN pasa a la CNN principal.

Se diferencia de la CGNN en que mientras esta incorpora matemáticamente simetrías en los filtros, **la STN transforma la imagen misma** para alinearla a un formato estándar.

Comentario final

Más allá de la explicabilidad, hay quienes sostienen que lo realmente importante es poder evaluar de forma rigurosa si, en la práctica, las decisiones del modelo resultan consistentes, justas, precisas y robustas en todos los escenarios imaginables.

En esa visión, lo que se considera primordial es validar el resultado por sobre el proceso, aceptando que partes del modelo puedan quedar sin explicación si lo que este hace es considerado correcto y fiable, ligado a una ética de resultados preestablecida.

Parte 4: Aplicaciones de Inteligencia de datos: Modelos generativos

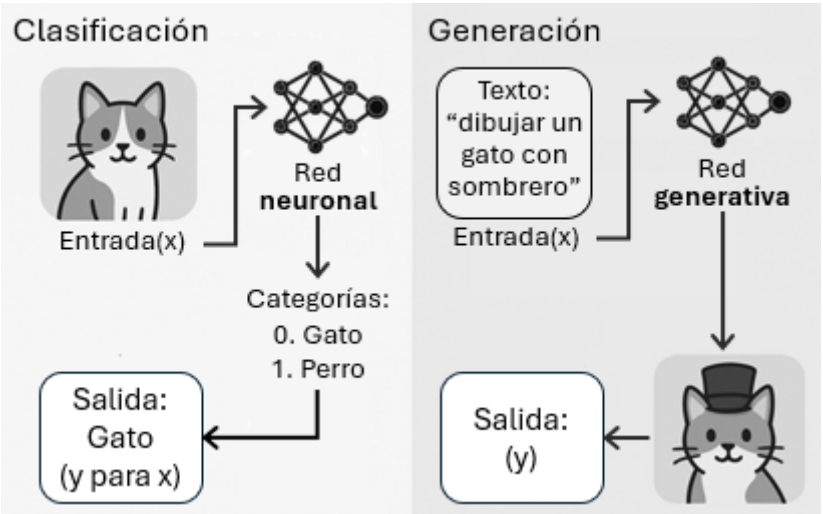
Introducción

Dentro de la IA hay dos grandes subgrupos de modelos

Modelos clasificadores: en los que tenemos una serie de datos (x) y de etiquetas (y), y el objetivo es medir la posibilidad de que cada dato se corresponda con una etiqueta específica, esto es que el objetivo se puede escribir como P(Y|X=x), para lo que se debe obtener una función que **mapee x en relación con y**

Modelos generativos: Nuevamente, (x) son datos e (y) etiquetas, pero ahora lo que buscamos es **obtener una función que genere x' de forma similar a la distribución real de los datos** (x) e (y).- Objetivo que escribimos como P(x) o P(x, y).

En términos simples, mientras un modelo clasificador selecciona la mejor categoría entre opciones ya dadas para ubicar un input, el modelo generativo crea datos nuevos a partir del input aplicando patrones aprendidos.



Los modelos generativos estadísticos

Dentro de los modelos generativos tenemos los **modelos generativos estadísticos**, que utilizan redes neuronales y responden a la descripción dada de manera directa: **aprenden las regularidades y patrones** de un conjunto de datos (texto, imágenes, sonidos, etc.) y, a partir de eso, pueden crear nuevos ejemplos que parecen provenir de esos datos.

Lo que hacen estos modelos generativos estadísticos es imitar la distribución estadística de los datos originales, es decir "observan" qué tan frecuentes son ciertas combinaciones de características en los datos para clasificarlo o crear nuevas salidas de acuerdo con ellos: un modelo entrenado con muchas fotos de gatos aprende que suelen tener orejas puntiagudas, ojos grandes y bigotes, y que esas características aparecen juntas con cierta probabilidad estadística conforme su distribución.

Datos de entrenamiento (X/Y):	Lo que el modelo aprende (Pdata)
Gato negro → frecuente	P(orejas puntiagudas) = 0.9
Gato blanco → menos frecuente	P(bigotes largos) = 0.8
	P(pelaje negro) = 0.6
	P(pelaje blanco) = 0.3

Cuando se le pide al modelo que "genere un gato", este combina esas probabilidades y produce un gato nuevo que respeta la estadística aprendida.

Los tres principales usos que se da a los modelos generativos estadísticos son:

Generación: *sampleamos [hacemos inferencias sobre el modelo para movernos dentro de la p(x)o los normalizamos]* para obtener nuevas imágenes de caras.

Estimación de probabilidad: p(x) debería ser alto para imágenes similares a caras y bajo en caso contrario (detección de anomalías).

Unsupervised representation learning: se extraen patrones en común de las imágenes (color de pelo, forma de la cara, sexo, etc).

De esta forma se crean nuevos datos (las salidas) que pueden ser reutilizados para entrenar modelos, aumentando la cantidad de datos disponibles, (ver más adelante "Synthetic data augmentation").

TIPOS DE MODELOS GENERATIVOS

Hay cuatro grandes tipos de modelos generativos, y pueden combinarse entre sí, y cada uno de ellos tiene sus variantes internas

- Autoregresivos
- Autoencoder
- GAN
- De difusión

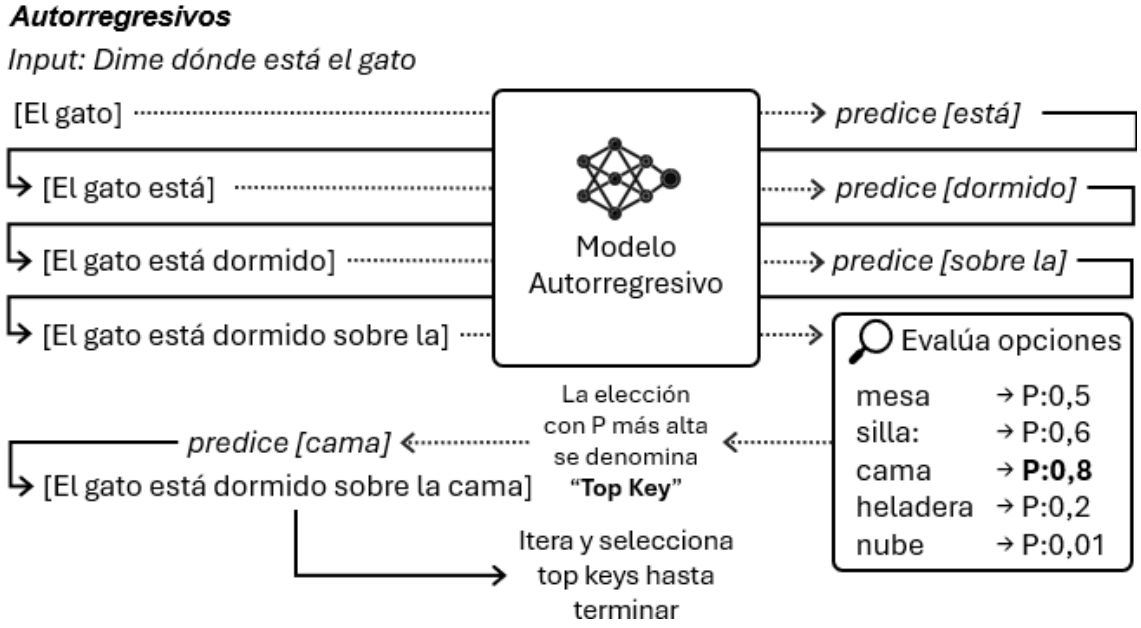
g) MODELOS AUTOREGRESIVOS

Se basan en la regla de la cadena dada por la probabilidad de intersección entre A y B = probabilidad de A + la probabilidad de B dado A

$$\mathbb{P}(A \cap B) = \mathbb{P}(B \mid A)\mathbb{P}(A)$$

Lo que hacen es predecir la siguiente parte de la información a crear (una palabra en un texto, un píxel en una imagen o una nota en una melodía) usando lo que ya ha generado antes como base.

Para elegir la palabra, siguiendo la regla de la cadena, calcula la probabilidad de que aparezca la siguiente palabra.



Cuando se trata de imágenes o de movimiento, se da la misma situación

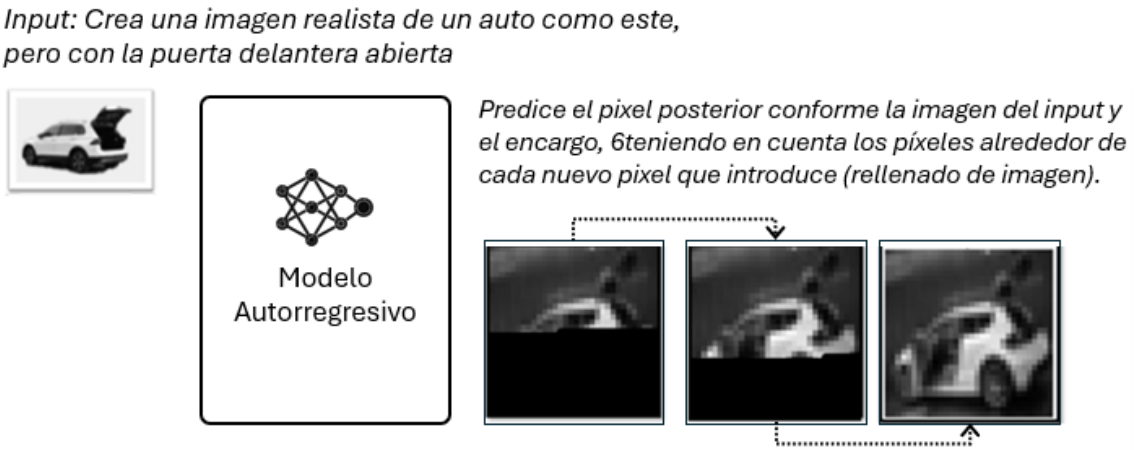
En la imagen se predice cuál será, con mayor P, el píxel siguiente al último que tenemos en nuestra cadena de píxeles.

Dado que la construcción de la imagen píxel a píxel es muy lenta, se suele trabajar con parches de píxeles (cada parche predice el contenido del parche siguiente) buscando equilibrio entre calidad y velocidad.

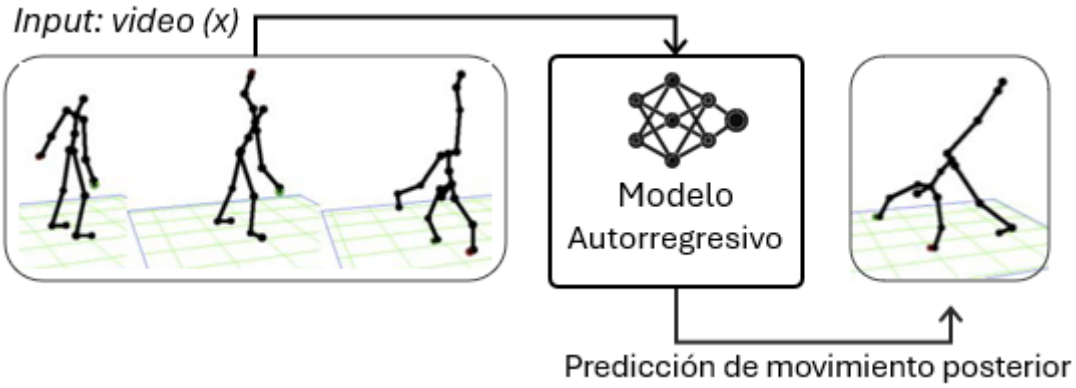
Cuando se trabaja utilizando modelos generativos basados en transformes, se agrega en la red una capa encargada de tokenizar el parche o imagen de ingreso que, convertida a token, permite que interactúe con otros formatos (imagen a texto, texto a video, etc.).

En la generación de movimiento se establen las P para los posibles movimientos posteriores de cuerdo a los datos de entrenamiento y el input recibido por el modelo.

Uso para la generación de imágenes



USO PARA LA GENERACIÓN DE VIDEO



Dado que cada salida está dada por la selección de una posibilidad, nosotros podemos elegir que tipo de salida queremos (la más probable, una intermedia…) o decidir que con cierta frecuencia se tome la opción con menor probabilidad. A esta elección la llamamos **temperatura**, un parámetro que controla qué tan "creativa" o "conservadora" es la red al elegir la siguiente palabra/imagen/sonido a insertar en la cadena.

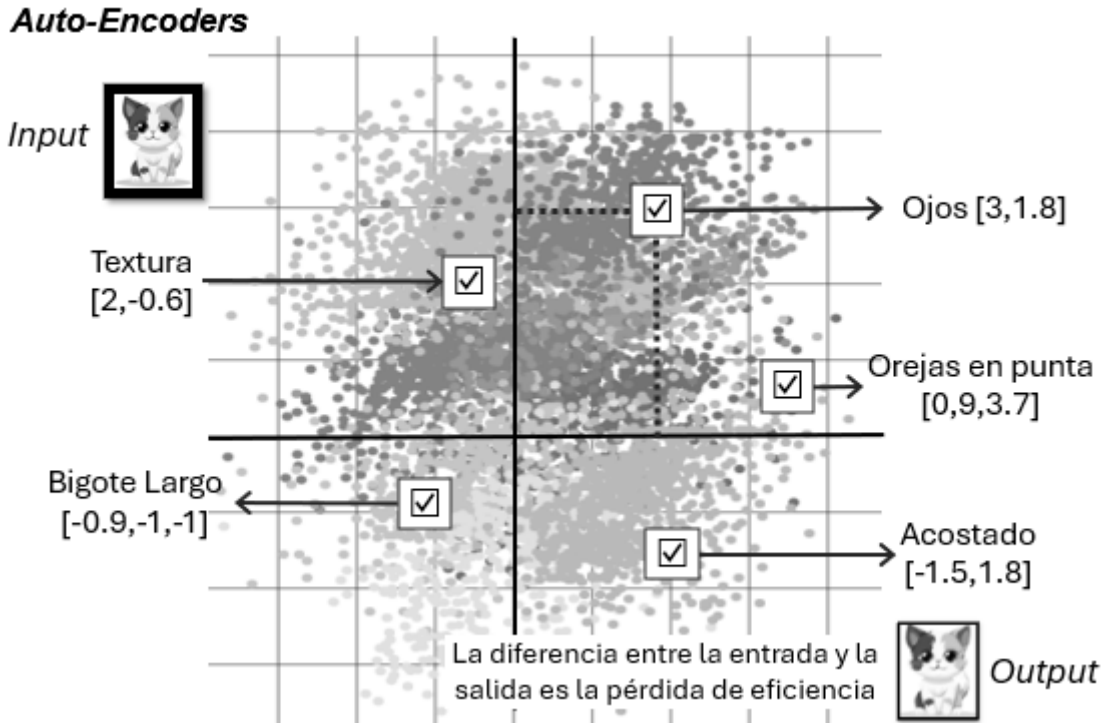
- Temperatura baja (≈ 0 – 0.3): El modelo se vuelve **muy determinista**: siempre elige la palabra más probable, dando como resultado respuestas más seguras, repetitivas y predecibles.
- Temperatura media (≈ 0.7 – 1.0): El modelo mantiene cierto **equilibrio**: mezcla palabras comunes con algunas menos probables, dando como resultado un texto coherente, pero con un abanico de variaciones.
- Temperatura alta (> 1.0): El modelo da más importancia a las palabras poco probables, lo que resulta en mayor **creatividad**, pero a costa de aumentar el riesgo de incoherencias.

h) VARIATIONAL AUTOENCODER (VAE)

Ya sabemos que las redes neuronales autoencoder están diseñadas para aprender una representación comprimida de los datos captando sus características más relevantes, y que funcionan en dos pasos: un encoder que toma los datos de entrada y los comprime en un espacio más pequeño y abstracto llamado espacio latente, y un decoder que toma esa representación comprimida y busca reconstruir los datos originales.

También pueden utilizarse como método no supervisado que permite obtener una representación de los datos de menor dimensionalidad.

Cuando se trata de un modelo generativo, el decoder no buscará reconstruir los datos originales, sino crear nuevos datos a partir del espacio latente creado por el encoder. Para entender ese proceso tenemos que comprender primero que, en el espacio latente, cada punto corresponde a una "versión posible" de los datos, y que, si tomamos un punto cualquiera del espacio latente y lo pasamos por el decoder, obtenemos una nueva muestra parecida a las originales, pero que no existía antes.



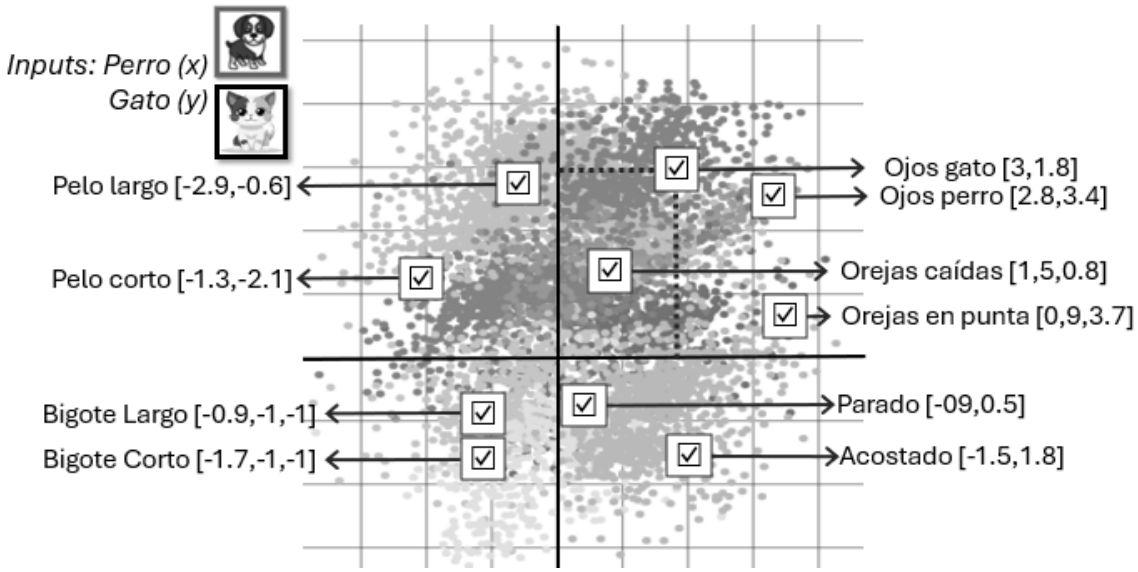
La clave está en el **espacio latente, que actúa como un "mapa de posiciones"** dentro del modelo, en el que cada punto representa una **versión posible** de los datos generados por el encoder (una imagen, un sonido, un texto) que ha sido transformada a un vector numérico que representa sus características principales.

En el espacio latente no hay un gato, pero sí un vector que representa sus características indicando en qué punto de espacio se encuentran: [0.7, -1.2, 3.4] podría significar orejas puntiagudas (0,7), bigotes largos (-1,2), ojos grandes (3.4). Ese vector es una especie de mapa de coordenadas de las características principales de un gato en el espacio latente.

El tamaño del espacio latente lo decide la **arquitectura del modelo** puede ir de dos a millones de dimensiones. Cuantas más dimensiones, el sistema se vuelve más potente y abstracto, ya que cada dimensión captura solo un aspecto del dato: una dimensión del gato representa la forma de sus orejas, otro el largo de sus bigotes, otro el tamaño de sus ojos, y otros representan sus colores posibles, su textura, sus poses, etc., o incluso dimensiones que las personas somos incapaces de capturar, pero el modelo sí identifica (black box).

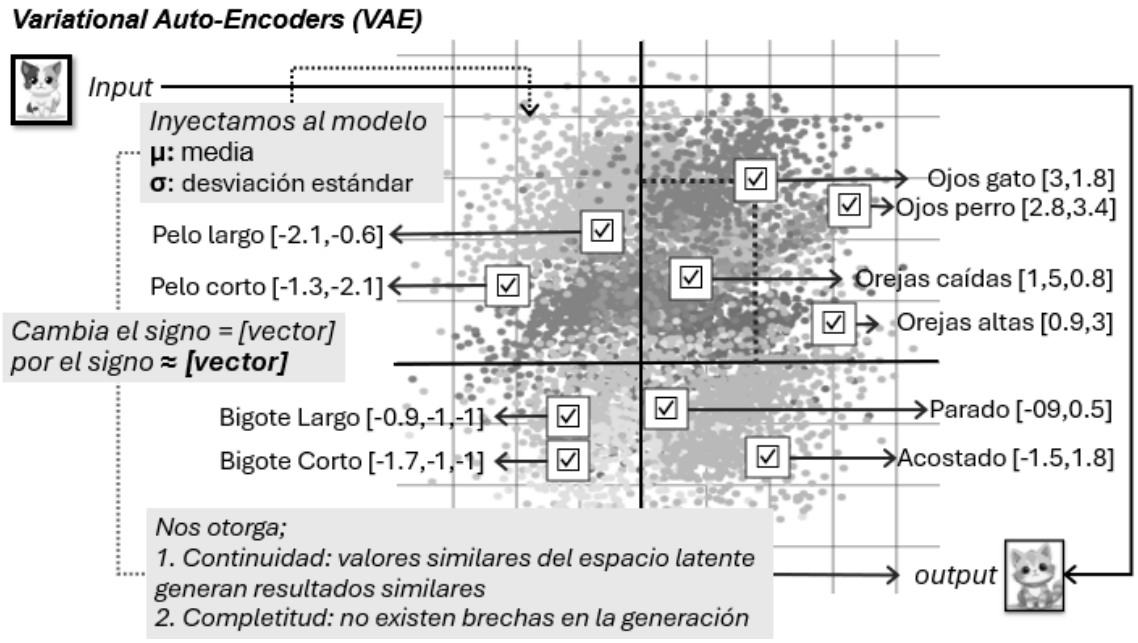
El mapa de posiciones de puntos que nos da el vector señala "cuánto" de cada característica hay en el dato (si es una imagen, cuánto de orejas puntiagudas, brillo, orientación, etc.; si es un texto, cuánto de formal/informal es, el tema que trata, la longitud de frase; si se trata de sonido, nos puede señalar tono, velocidad, timbre, etc.).

Por sus características, el espacio latente **no es fijo**, sino que cambia si modificamos la arquitectura del modelo, pero también si lo entrenamos con nuevas fuentes (x) y categorías (y): si antes solo había gatos y luego le agregamos fotos de perros y la categoría "perro", el espacio se reorganiza solo para incluir ambos tipos de animales.



Dicho esto, las redes encoder-decoder pueden usarse para reconstruir imágenes, pero también para crearlas, caso en el cual hablamos de **Variational Auto-Encoders (VAE),** que no son más que **una versión especial de los autoencoders diseñada no solo para comprimir y generar datos nuevos y variados modificando el manejo que hacen del espacio latente.**

El VAE ya no va a representar cada dato como un vector de punto fijo, sino como una **distribución de probabilidades.** "Gato" ya no es [1.3, -0.7, 2.5] sino que se transforma en **≈ valores alrededor de [1.3, -0.7, 2.5]** porque incluimos ahora un factor aleatorio (**estocástico**) en el espacio latente, llenando de puntos todo el espacio latente, permitiendo tanto reconstruir datos (como un autoencoder normal) como generar nuevas variaciones de los datos interpolando entre distribuciones.



Esto introduce un nuevo problema, el "*disentanglement*" (desentrelazamiento), pues ahora queremos que cada una de las variables del espacio latente no esté correlacionada o, en otros términos, que sea independiente de tal forma que pueda moverse sola, sin afectar a las demás, asegurando así que no haya "saltos" entre imágenes sino cambios parciales

Veamos, en un VAE bien entrenado, el espacio latente debería organizarse de forma clara y ordenada donde cada eje represente una variable (eje de forma de las orejas, eje de tamaño de pelo, eje de textura, eje de color, etc.), lo que hace que **cada dimensión** latente controle solo una característica del proceso de creación de manera independiente de las demás. Esto es lo que llamamos un espacio latente desentrelazado (*disentangled representation*).

Ahora bien, imaginemos un modelo VAE entrenado con imágenes de caras. Un eje es "sonrisa ↔ seriedad", otro es "ojos abiertos ↔ ojos cerrados" y un tercer eje marca "mejillas redondeadas ↔ mejillas rectas". Al aumentar el valor de la sonrisa automáticamente se mueven los valores de la **forma de los ojos y de las mejillas. Se trata de tres ejes** entrelazados (*entangled*) en el espacio latente, y no podemos controlar uno sin impactar en los otros.

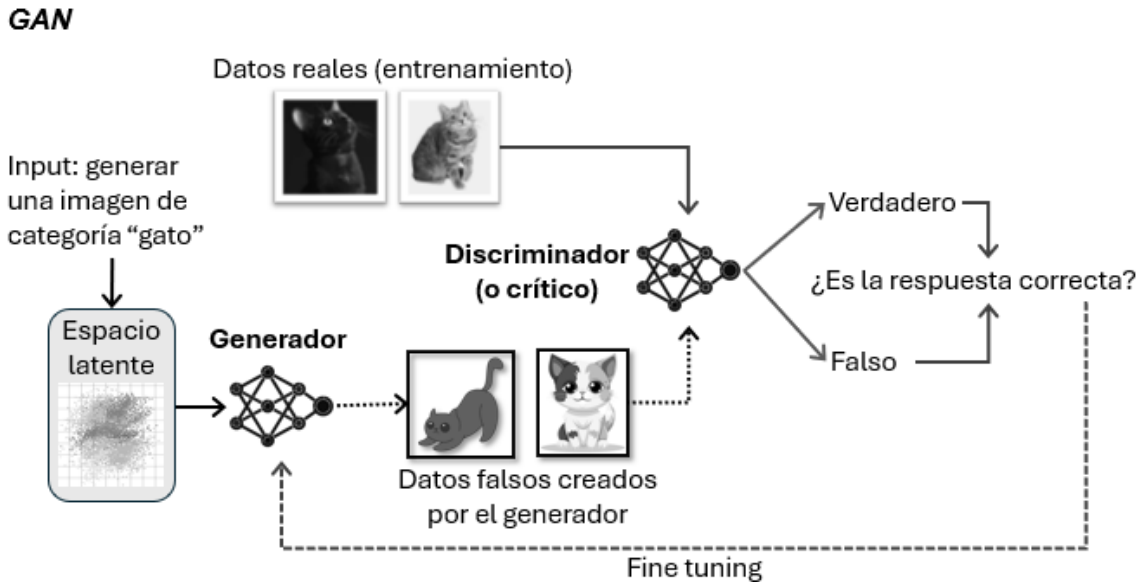
Esto puede percibirse como algo positivo porque, por ejemplo, da más naturalidad al cambio del rostro al sonreír, pero es un problema en otros espacios porque hace que no sea posible manipular atributos de manera precisa, afectando la interpretabilidad de los cambios, se vuelve un problema mayor en tareas de creatividad dirigida, como la mayor parte de las tareas científicas, donde es necesario que cada variable represente algo claramente y pueda controlarse de forma independiente.

Ya se han desarrollado modelos que son variantes de VAE pero con capacidad de desentrelazar mejor el espacio latente, entre ellas **β-VAE**: ajusta un parámetro β para forzar la independencia entre variables latentes, o **factorVAE**, que introduce penalizaciones adicionales para reducir la correlación entre dimensiones latentes.

En cuanto a sus usos, Spotify usa VAE para agrupar a usuarios según sus preferencias musicales, y el modelo también se usa para cambiar "variables" (como la textura) en imágenes, sin modificar el resto de sus atributos.

i) REDES GAN (GENERATIVE ADVERSARIAL NETWORKS)

Recordemos que las GAN (o redes adversariales) son un tipo de modelo de inteligencia artificial generativa inventado en 2014 por Ian Goodfellow que, pone a dos redes neuronales a competir entre sí: un generador inventa datos falsos (imágenes, sonidos, texto) y un discriminador debe detectar si los datos que le envía el generador son reales o falsos. El generador recibe la crítica del discriminador y mejora sobre ella, tratando de hacer sus engaños menos detectables. El bucle continúa y con ello el generador aprende a crear datos más y más realistas y el discriminador a afinar su capacidad de detección de datos falsos.



El uso de GAN genera algunos problemas. Como es un juego de suma cero entre generador y discriminador, si todo sale bien llega un momento en que el generador crea datos falsos tan realistas que el discriminador apenas puede diferenciarlos, pero a la vez el discriminador se vuelve lo suficientemente bueno para no dejarse engañar fácilmente (acierta en cerca del 50% de los casos por mero promedio estadístico). A partir de este punto, ninguno de los dos puede seguir mejorando de performance significativamente sin que el otro también lo haga: se alcanza un **punto de convergencia** (un "equilibrio de Nash").

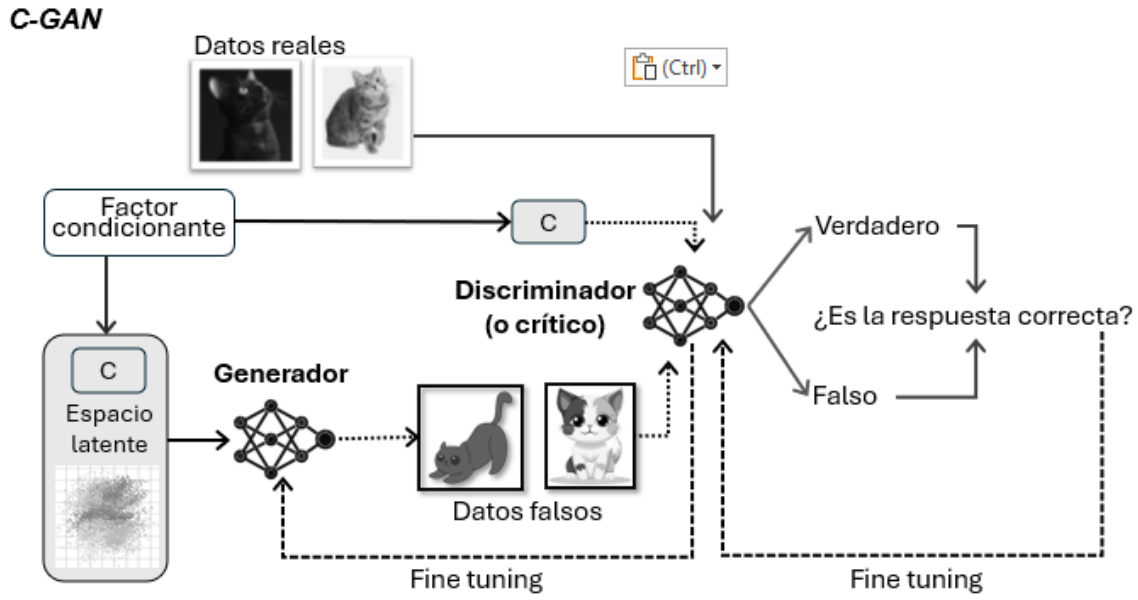
Por otro lado, puede ocurrir que el generador **encuentre una "fórmula fácil" para engañar** al discriminador con **solo un tipo de (o unas pocas) salidas, por lo que el generador se** "acomoda" a esa fórmula que le permite superar al discriminador y deja de producir diversidad de salidas. Esto se denomina **convergencia fugaz** *(model collapse)*.

Para evitar ambos tipos de convergencias hay opciones como las **Wasserstein GAN (WGAN)** o de creación de una métrica para distinguir la calidad de los resultados generados en cada época. El discriminador, que en las WGAN pasa a llamarse "crítico", indica cuán seguro está de que una imagen es real.

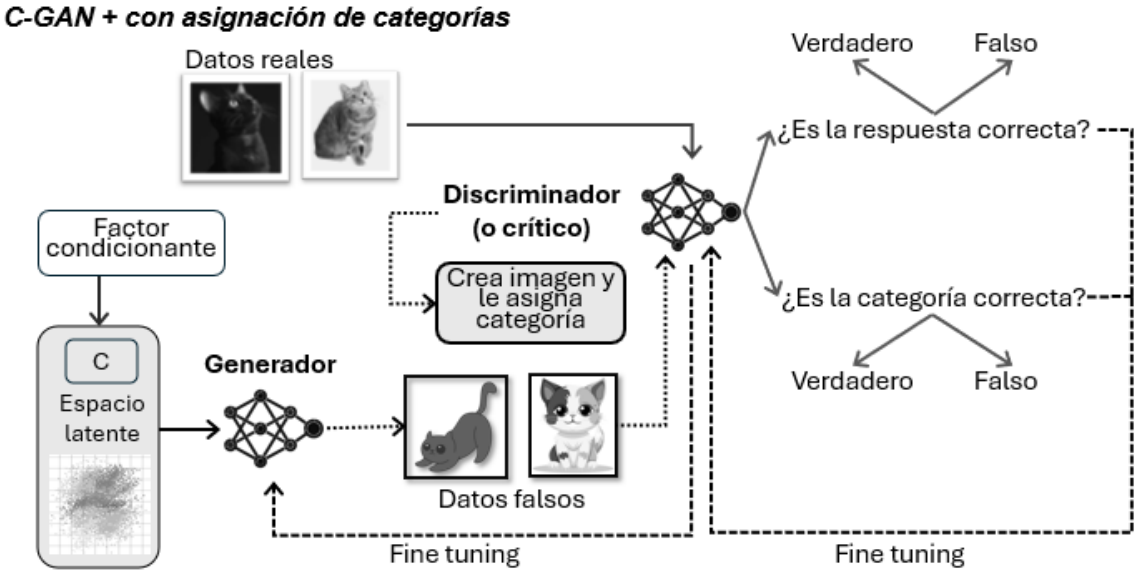
En el caso de las WGAN, pasamos de que el discriminador de una respuesta binaria (verdadero/falso) a calcular la distancia entre distribuciones.

En el segundo caso tenemos las llamadas ***Conditional Generative Adversarial Networks*** (**CGAN**) que incorporan al modelo un factor condicionante que permite direccionar la generación de sus outputs. Por ejemplo, la condición puede ser una categoría (y=perro) lo que fuerza al generador a crear imágenes de perros y al discriminador a analizar si las imágenes recibidas

se corresponden con perros



Como resultado, mientras las redes GAN generan outputs los más reales posibles, las CGAN generan outputs que sean lo más reales posibles **y que cumplan la instrucción dada en el factor condicionante**.



También es posible ver si el propio discriminador está haciendo una clasificación correcta de los inputs que recibe desde el generador, simplemente agregando una función de clasificación en la salida del primero

En otra línea hallamos las **Style GAN**, una variante avanzada de las GAN, creada por NVIDIA en 2018, cuyo objetivo es generar imágenes de altísima calidad y realismo.

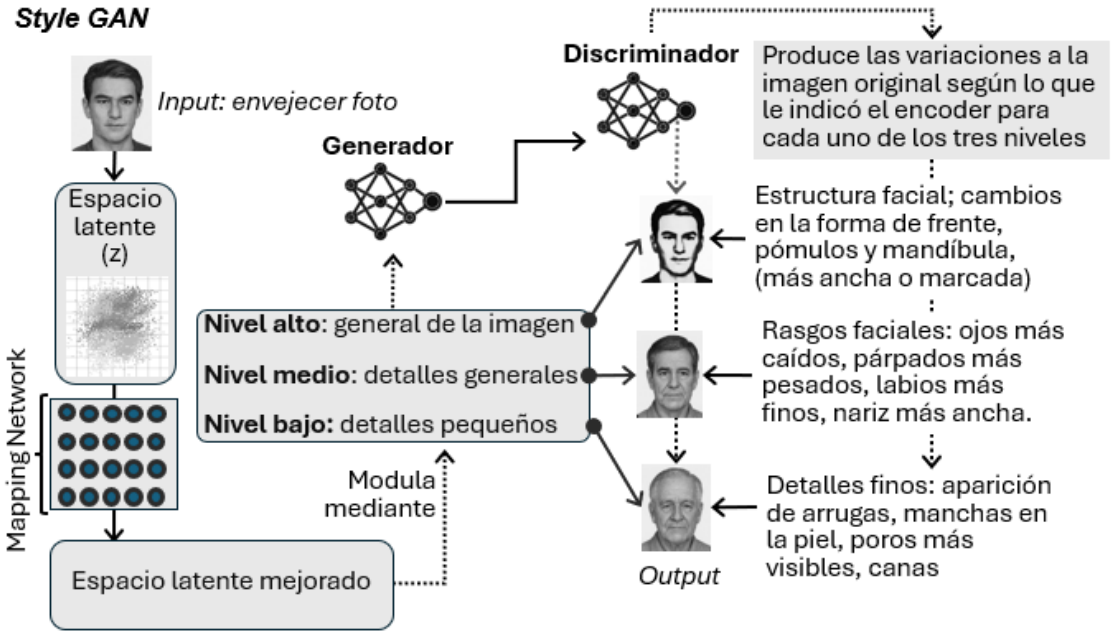
Aquí, a diferencia de una GAN tradicional, se incluyen en la red neuronal varias capas intermedias, que en su conjunto conforman el ***mapping network***.

La primera capa del mapping network recibe el vector de ruido [**z**] (la entrada aleatoria)

y, a través de sus capas, lo transforma en un vector nuevo [**w]** mejor ordenado. Este nuevo vector [w] no entra al generador, sino que **modula** distintos niveles de estilo mediante los cuales se **introducen ajustes en la imagen**:

- *Niveles altos*: controlan características generales como la forma de la cara, la posición de la cabeza y la estructura básica de la composición de la imagen.
- *Niveles medios*: controlan rasgos faciales específicos, como ojos, nariz, boca.
- *Niveles bajos***:** controlan los detalles más finos: textura de la piel, arrugas, reflejos en los ojos, cabello.

De esta forma se logra un control preciso sobre la apariencia de la imagen, inclusive cuando se trata de generar rostros sintéticos hiper-realistas e imágenes continuas y coherentes que pasan suavemente de una a otra.



j) DIFFUSION MODELS

Su objetivo es **aprender a transformar ruido en datos coherentes** (imágenes, audio, texto).

Al igual que ocurre con las redes adversariales, los modelos *diffusion* se entrenan en dos fases opuestas, dando origen a un proceso probabilístico de dos direcciones

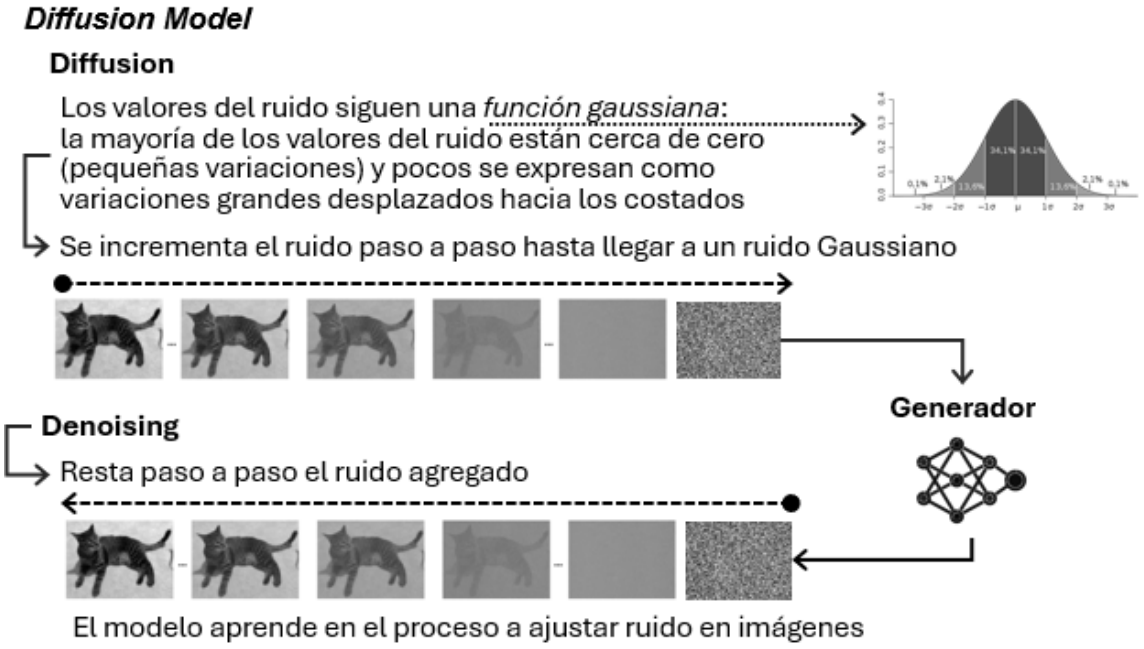
Difusión (como destrucción): a una imagen real se le agrega ruido poco a poco hasta que quede irreconocible, y en ese proceso el modelo aprende cada paso de esa destrucción.

Denoising (como reconstrucción): el modelo se entrena para hacer lo contrario: quitar el ruido paso a paso.

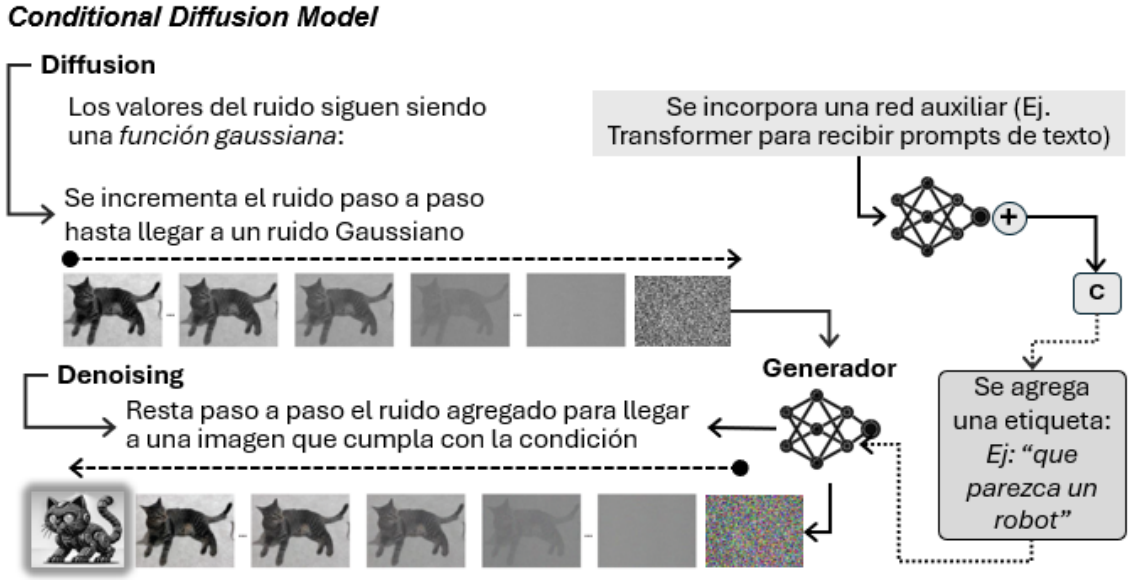
Como resultado final, el modelo es capaz de iniciar desde cualquier ruido y "limpiarlo" progresivamente hasta que aparezca una imagen nueva y coherente.

En su entrenamiento, estos modelos comienzan por "ver" miles de imágenes reales, a las que se les agrega ruido en distintas intensidades (desde un poquito hasta casi destruirlas). Así,

el modelo debe aprender a responder la siguiente pregunta; Si esta imagen tiene ruido nivel 450 de 1000, ¿cómo debería verse con un poquito menos de ruido?" Para alcanzar esa respuesta, el modelo no necesita memorizar imágenes, sino aprender la distribución de probabilidades de cómo deberían ser los datos en cada paso intermedio y las aplica mediante un **generador**.



Dentro de estos modelos tenemos los denominados "***Conditional Diffusion models***", que agregan una condición. Lo que marca la potencia de estos modelos, es que pueden ser entrenados agregando condiciones mediante instrucciones (dibuja en azul) o guías (respeta el estilo de Diego de Rivera al crear nuevas imágenes). Al insertar estas condiciones "guiamos" al modelo a través de las oportunidades existentes de creación de imágenes que encuentra, llevándolo al resultado de manera más directa (ya no quiero "cualquier" gato entre miles de opciones, quiero un gato que cumpla cierta condición)



Otra técnica usual en los modelos de difusión es el "***classifier free-guidance***", donde en

lugar de entrenar un clasificador, el mismo modelo de difusión aprende a trabajar con y sin condiciones (generar imágenes de gatos / generar una imagen de un gato hecho con origami).

Para ello, durante el entrenamiento, al modelo se le hace "olvidar" aleatoriamente la condición en un cierto porcentaje de pasos, lo que lleva a que aprenda a trabajar en ambos escenarios y, al momento de generar, se combinan los pesos de las salidas condicionada y la no condicionada aplicando una escala de guía (*guidance scale*) que controla con qué intensidad debe seguirse la condición.

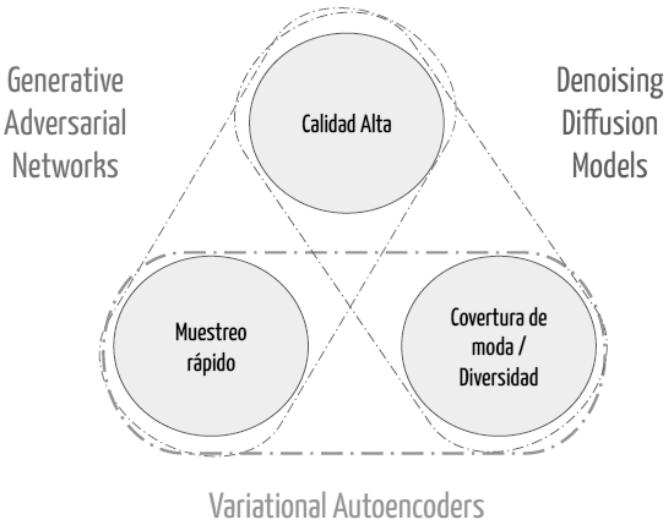
- *Guía alta:* el modelo fuerza demasiado la condición, lo que puede dar imágenes fieles al texto, pero rígidas o con artefactos
- *Guía media:* balancea la fidelidad al prompt y naturalidad de la imagen.
- *Guía baja:* la influencia del texto es débil, la imagen puede salirse de las instrucciones, pero es más creativa.

Ya no hay dos sistemas, uno generando ruido y el otro reduciéndolo, sino uno solo que puede trabajar en ambas direcciones, con y sin condiciones.

Su principal uso es la generación de imágenes, pero también se utiliza para generar video y audio de gran calidad.

k) VENTAJAS Y DESVENTAJAS DE CADA MODELO Y "TRILEMA DE LOS MODELOS GENERATIVOS"

Modelo	**Ventajas**	**Desventajas**
Auto-regresión	1. Son fáciles de entrenar y de obtener métricas de evaluación. 2. Se puede samplear fácilmente cada punto para generar nuevos datos. 3. Devuelven la probabilidad para cada punto.	1. Son lentos, requieren una generación píxel a píxel o resignar calidad utilizando parches.
Variational Autoencoder	1. Permite inferencias que luego pueden ser reutilizadas en otras tareas como compresión y agrupamiento de datos. 2. El *disentanglement* es fácil realizar. 3. Genera nuevos datos de forma rápida.	1. Los datos que genera tienen blur y son de baja calidad. 2. Requiere de un proceso de *fine tuning* del tamaño del espacio latente y de las pérdidas aceptables en variabilidad entre el input y el output obtenido a partir de este.
GAN	1. Buena calidad de imágenes. 2. Generación rápida de nuevos datos.	Baja diversidad en los resultados. No genera tan buenas imágenes como los modelos de difusión.

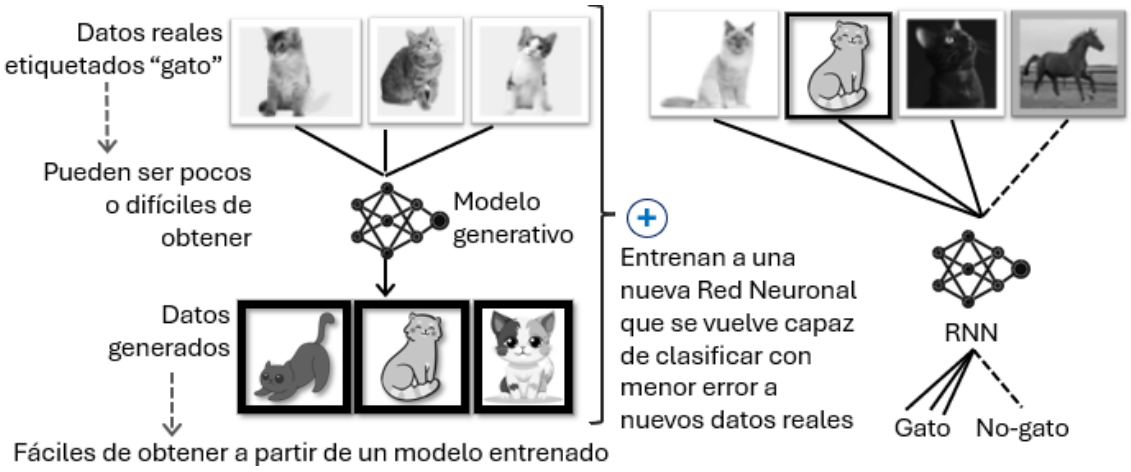
Diffusion	1. Estado del arte en calidad de imágenes. 2. Salidas diversas (con y sin etiquetas). 3. Es fácil generar variaciones a partir de una imagen (*style transfer*).	1. Los pasos requeridos para entrenar e inferir con el modelo conllevan un alto tiempo de cómputo (*trade-off* entre calidad y velocidad).
		

El trilema de los modelos generativos es una aplicación a la IA del dicho popular "*bueno, rápido o barato: elige dos de tres*".

Generación de datos sintéticos (*synthetic data augmentation*)

Todos los modelos que vimos pueden ayudar a crear datos sintéticos que dan acceso a más datos de entrenamiento, fáciles de obtener, que ya salen etiquetados y que son, en esencia, no personales (son imitación real de datos, pero no son datos que pertenecen a ninguna persona).

Los datos sintéticos pueden combinarse con datos reales



También se utiliza la opción de hacer un primer entrenamiento del modelo con datos sintéticos y luego utilizar datos reales para hacer el *fine tuning*.

Parte 5 TinyML

TinyML: Machine & Deep Learning para Pequeños Dispositivos

INTRODUCCIÓN

Es una rama de la inteligencia artificial que se enfoca en la implementación de modelos de aprendizaje automático en dispositivos de bajo consumo energético y con recursos limitados, como microcontroladores.

Encontramos tres escenarios principales de aplicación del Tiny ML:

- **Nube**: Modelos robustos, grandes y complejos. Requiere gran potencia de cómputo + Acceso a recursos prácticamente ilimitados, con desafíos en latencia y costos. Ej: Servicios IA a gran escala, análisis de big data, generación de texto, audio, voz, imagen, video
- **Borde**: Procesamiento más cercano al punto de origen de los datos que equilibra la potencia de cómputo con baja latencia y operación semi-autónoma. Ej: Sistemas de videovigilancia inteligente
- **Dispositivos**: Ejecución de modelos directamente en dispositivos de usuario final, priorizando privacidad, operación sin conexión y la eficiencia energética. Ej: wearables inteligentes, electrodomésticos con IA

Combina el *machine learning* con dispositivos diminutos y baratos (tipo Arduino, chips de bajo consumo) que normalmente funcionan con baterías y tienen escasa memoria y capacidad de cálculo. La idea es evitar enviar datos a la nube sino ejecutar modelos directamente en el dispositivo. Sus principales características son:

- Ultra-bajo consumo: puede funcionar con una pila durante meses o incluso años.
- Modelos pequeños: utiliza redes neuronales comprimidas y optimizadas para caber en memoria de unos pocos cientos de kilobytes.
- Autonomía: no necesita conexión a la nube para trabajar. El procesamiento tiene lugar en el propio dispositivo.

Esto es lo que ocurre, por ejemplo, en relojes inteligentes que reconocen gestos o patrones de actividad, sensores ambientales que detectan anomalías (fugas de gas), dispositivos médicos portátiles para monitoreo en tiempo real, etc.

FUENTE: WIDENING ACCESS TO APPLIED MACHINE LEARNING WITH TINYML. *HARVARD DATA SCIENCE REVIEW*. HTTPS://WWW.RESEARCHGATE.NET/PUBLICATION/358181339_WIDENING_ACCESS_TO_APPLIED_MACHINE_LEARNING_WITH_TINYML

El Tiny ML se relaciona con otros espacios de la informática, la IA y la mecánica:

Microcontrolador: chip que integra procesador, memoria y periféricos, brindando alta eficiencia energética y diseñado para realizar tareas con mínimos recursos de hardware (Ej.: Arduino, ESP32, Disp. ARM).

SoC (Sistema on Chip): similar a un microcontrolador, pero integra múltiples sistemas complejos en un chip como CPU +Video+ Audio (Ej: Smartphones, dispositivos móviles, Raspberry Pi).

Sistemas Embebidos: soluciones que realizan tareas específicas con precisión Integran hardware y software optimizados para la solución (Ej.: electrodomésticos, sistemas de control automotriz y equipos industriales).

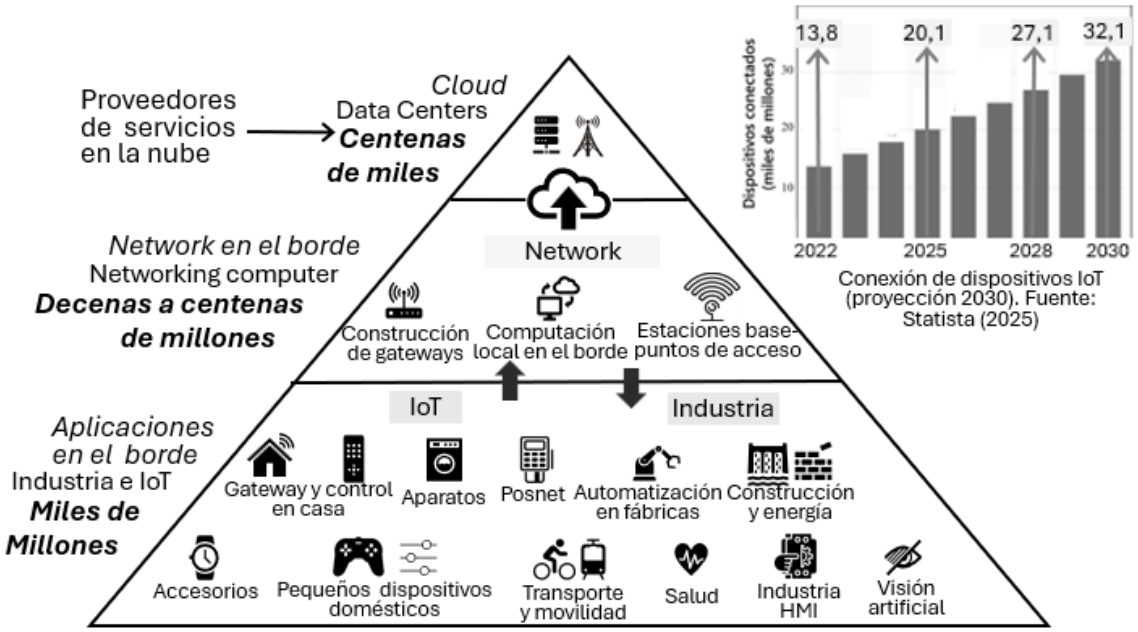
IoT: red global de todo tipo de dispositivos físicos interconectados que recopilan, intercambian y procesan datos a través de internet, facilitando la automatización de procesos y toma de decisiones en hogares, industria y salud.

Edge Computing: paradigma computacional que procesa datos en los límites de la red, cerca de su fuente de origen, utilizando la carga computacional próxima a dispositivos y sensores, reduciendo latencia y consumo de ancho de banda y garantizando eficiencia y privacidad.

En Tiny ML convergen diferentes técnicas

1. Algoritmos optimizados:
   1.1. Aprendizaje automático y redes neuronales compactas y eficientes
   1.2. Técnicas de compresión de modelos
   1.3. Algoritmos adaptados a recursos limitados
2. Herramientas de software:
   2.1. Frameworks especializados (TensorFlow Lite, Edge Impulse, EmbedIA)
   2.2. Bibliotecas de compresión y cuantización
   2.3. Entornos de desarrollo para microcontroladores
3. Hardware Especializado:

3.1. —Microcontroladores de bajo consumo
3.2. Procesadores optimizados para ML
3.3. Sistemas embebidos con capacidades de procesamiento inteligente



Tiny ML tiene tanto beneficios como desafíos por superar.

Entre los beneficios destacan:

- Optimización de Recursos

  - Reducción del ancho de banda al transmitir menos datos
  - Adopción económica por su bajo costo

- Seguridad y Privacidad

  - Operación en modo sigilo minimizando la huella digital
  - Sin problemas de ciberseguridad
  - Privacidad mejorada al mantener los datos en el dispositivo

- Independencia Tecnológica

  - Funcionamiento autónomo, sin conexión a internet
  - Elimina dependencia de servicios de la nube
  - Aplicaciones sin dependencia de conectividad

- Eficiencia Energética

  - Bajo consumo de energía
  - Latencia mejorada
  - Soluciones autónomas con batería

Entre los principales desafíos contamos:

- Complejidad de Desarrollo

  - Herramientas de desarrollo limitadas

- Restricciones de memoria y almacenamiento
- Conocimientos para optimizar modelos

- Desafíos de Implementación

- Problemas de sobrecalentamiento
- Gestión de energía más compleja
- Dificultad para actualizar modelos

- Limitaciones Algorítmicas

- Modelos con menos capacidad de generalización
- No todo algoritmo de ML es adaptable
- Algunos dominios requieren mayor procesamiento

- Limitaciones Computacionales

- Capacidad de procesamiento reducida
- Modelos simplificados con menor exactitud
- Equilibrio tamaño/rendimiento de modelos

El uso de microcontroladores es distintivo del Tiny ML, y permite ver rápidamente sus diferencias con los sistemas de cómputo basados en microprocesadores

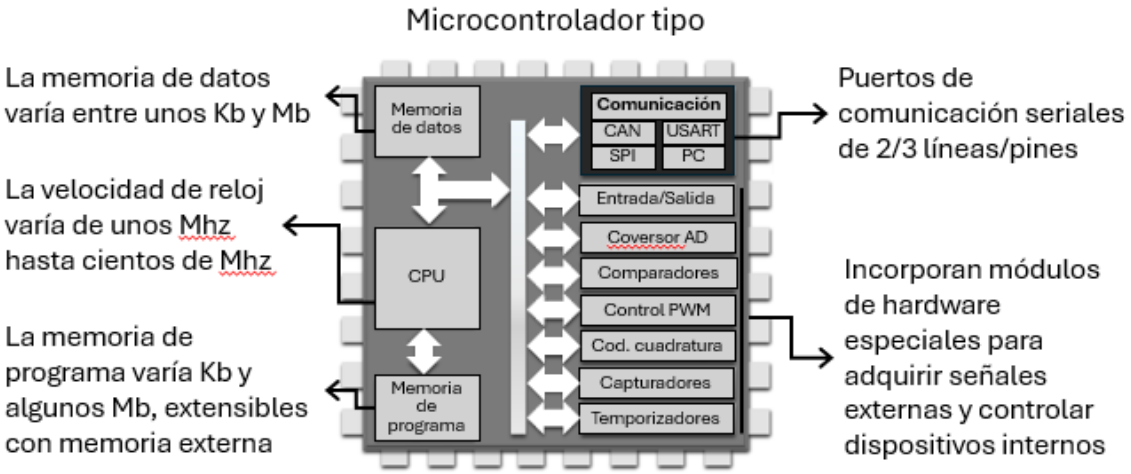
Característica	Micro**procesador (PC)**	Micro**controlador (Tiny ML)**
Diseño	Alto nivel de generalización	Alto nivel de especialización
Propósito	Procesamiento de información	Control de dispositivos electrónicos
Velocidad	Altas velocidades de reloj	Bajas/medias velocidades de reloj
Periféricos	Periféricos externos al chip	Periféricos integrados en el chip
Almacenamiento	Memoria de datos y programa externas	Memoria de datos y programa internas
Instrucciones	CISC, de tiempo variable	RISC, de tiempo constante
Consumo	Consumo de energía medio/alto	Consumo de energía bajo
Costo	Alto costo de adquisición	Bajo costo de adquisición

Este esquema es bien diferente al hardware y los chips que utilizan las computadoras en general y la IA en particular. En la siguiente página resumimos en una tabla los principales tipos de procesadores (resaltado en gris, los microcontroladores).

Antes de ver la tabla podemos resumir la función de cada tipo de chip de la siguiente manera:

- **CPU** = “cerebro generalista” de la computadora.

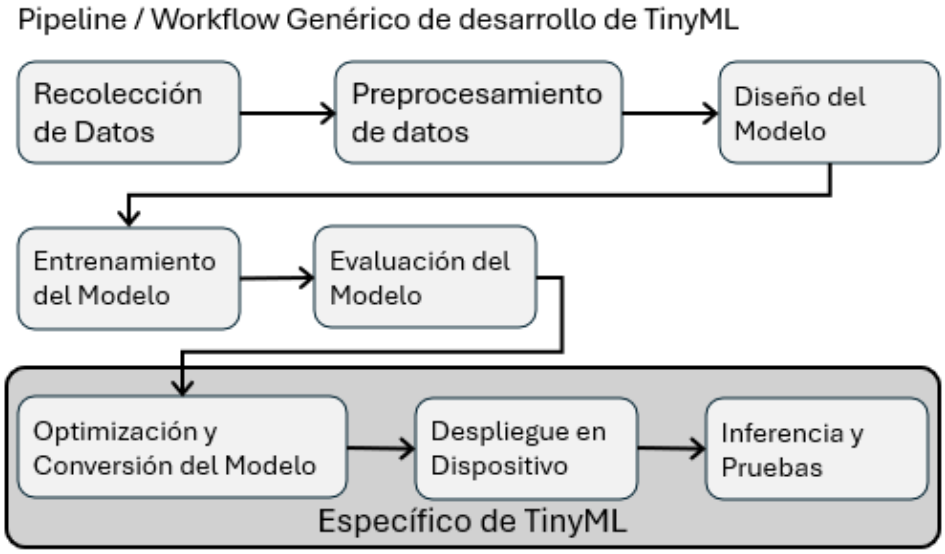
- **GPU** = “ejército de obreros” para tareas repetitivas en paralelo.
- **TPU** = “máquina industrial” diseñada solo para tensores de IA.
- **MCU** = “minicerebro barato” que controla dispositivos simples.
- **DSP** = “especialista” en audio, video y señales.
- **NPU** = “IA embebida” en móviles y dispositivos inteligentes.



Tipo	Nombre completo	Función principal	Características clave	Ejemplos de uso	Eficientes para redes neuronales
CPU	*Central Processing Unit* (Independiente)	Procesador general de un sistema.	Flexible: ejecuta cualquier tipo de programa. Pocos núcleos (2–32). Velocidad alta por núcleo. No especializado en paralelismo masivo.	Computadoras personales, servidores, laptops.	Limitada
GPU	*Graphics Processing Unit* (Requiere CPU)*t*	Procesamiento en paralelo de datos, originalmente gráficos.	Miles de núcleos pequeños. Excelente para operaciones repetitivas en paralelo (matrices, vectores). Muy usada en IA.	Gráficos 3D, videojuegos, entrenamiento de redes neuronales.	Moderada
TPU	*Tensor Processing Uni* (Requiere CPU)	Procesador especializado en tensores para IA.	Diseñado por Google. Optimizado para multiplicación de matrices. Alta eficiencia en redes neuronales profundas.	Entrenamiento e inferencia de IA en Google Cloud, modelos de lenguaje, visión artificial.	**Alta**
MCU	*Microcontroller Unit* (Independiente)	Procesador pequeño y autónomo para tareas simples.	Integra CPU + memoria + periféricos en un chip Bajo consumo y costo. Ideal para control en tiempo real.	Arduino, sensores IoT, electrodomésticos, automóviles.	Baja
DSP	*Digital Signal Processor* (Requiere CPU/MCU)	Procesador optimizado para señales digitales.	Maneja audio, video y telecomunicaciones. Instrucciones rápidas para multiplicar y acumular. Latencia muy baja.	Procesamiento de audio (micrófonos, audífonos), radio, compresión de video.	

NPU	*Neural Processing Unit* (Requiere CPU/MCU)	Procesador especializado en cálculos de redes neuronales.	Similar a TPU, pero más general. Integrado en móviles y SoCs. Acelera inferencia de IA (visión, voz, texto).		**Óptima**

Optimización de modelos Tiny ML



LA PODA (PRUNNING)

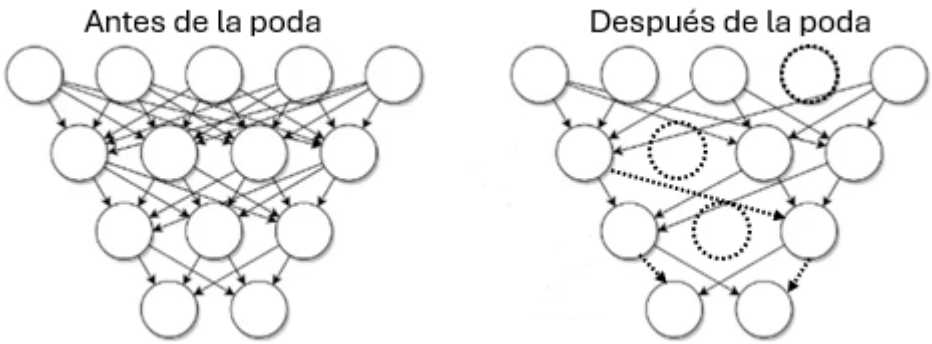
En el contexto de TinyML, la poda (*pruning*) es un proceso de simplificación donde se eliminan conexiones poco relevantes de la red neuronal para que el modelo sea más pequeño, rápido y eficiente, sin perder demasiada precisión.

La poda implica la eliminación de partes del modelo que casi no aportan al resultado, y puede aplicarse sobre dos elementos: los pesos (relaciones entre neuronas) y las conexiones.

- El proceso se inicia con el entrenamiento del modelo completo (con todas sus neuronas y conexiones).
- Luego, se mide la importancia de cada peso o neurona (por ejemplo, por la magnitud del valor).
- Hecha la medición, se eliminan los pesos y neuronas menos importantes (los más cercanos a cero).
- Finalmente, se vuelve a ajustar (reentrenar) el modelo completo para recuperar su precisión.

Hay dos clases de podas:

- **No estructurada**: que elimina pesos individuales dispersos. Es más precisa, pero más difícil de ejecutar en microcontroladores
- **Estructurada**: elimina neuronas, canales o capas completas. Menos precisa, pero mucho más fácil de ejecutar en microcontroladores.



La poda puede aplicarse en diferentes momentos:

- Durante el entrenamiento: se construye el modelo completo y, mientras se lo entrena, se van eliminando gradualmente conexiones o neuronas. De esta forma la red se va podando y ajustando simultáneamente.
- Después del entrenamiento: se entrena la red completa y luego se hace una poda para reducir su tamaño y, finalmente, un ajuste (fine-tuning) de la red ya podada.

LA CUANTIZACIÓN

A diferencia de la poda, que elimina conexiones innecesarias, la cuantización mantiene todas las conexiones, pero compacta el guardado de números. En términos simples, lo que hace es reducir la precisión numérica con la que el modelo guarda y calcula sus parámetros.

Es como pasar de usar una balanza ultrafina de laboratorio a una balanza de cocina casera: se pierde exactitud en las medidas, pero es más **barata, rápida y suficiente** para la tarea que se le requiere.

Lo que se hace es convertir valores de alta precisión (como números de punto flotante) en valores de menor precisión (como números enteros): los modelos normales usan números de **32 bits en punto flotante** (float32), mientras que un modelo cuantizado trabajará con **16, 8 (lo más usual) 4 o incluso 1 bit entrero** (lo último es lo más extraño, solo nos daría una opción binaria: sí/no). Es decir, no utiliza el punto flotante y considera todos los números como enteros, y de esta forma "comprime" la codificación que usa el sistema: un modelo cuantizado muestra reducción de consumo de energía, aceleración de la inferencia y requiere de menos requisitos de memoria para trabajar.

La cuantización se puede aplicar sobre pesos, valores de activación o gradientes.

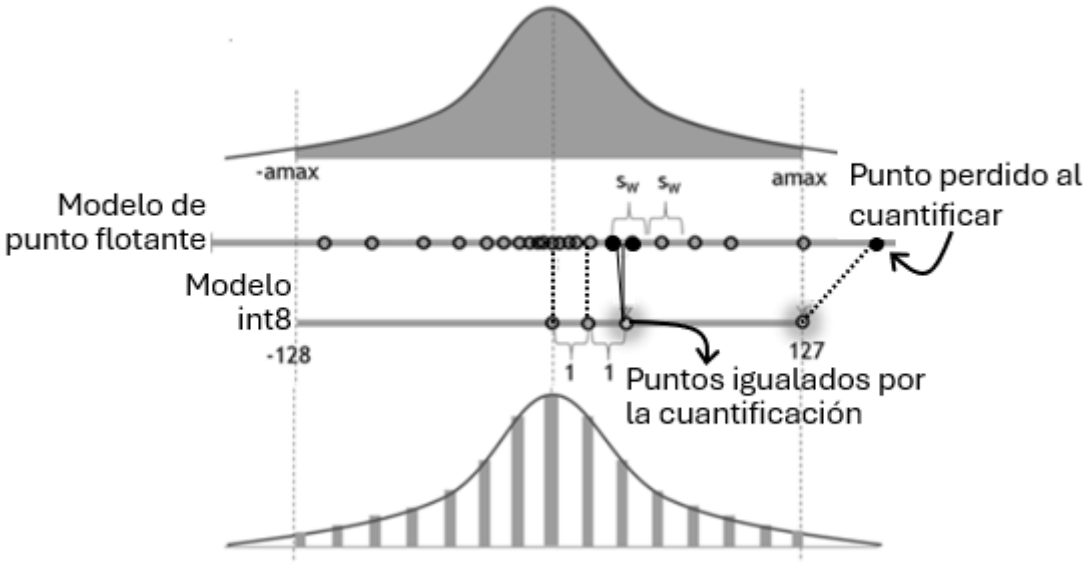
Funciona entrenando a un modelo normalmente en 32 bits y luego se lo transforma a un modelo de menor precisión (por ejemplo, int16 o int8). Se realiza entonces un ajuste de los rangos de valores para que el modelo siga siendo útil (proceso de normalización/reescalado). Opcionalmente, se reentrena el modelo cuantizado para recuperar precisión perdida y volver a su forma original

Este proceso es fundamental en TinyML pues genera menor tamaño en memoria, acelera las inferencias dado que los cálculos con enteros son mucho más ligeros y, en consecuencia, reduce el consumo de energía. El costo por pagar es la pérdida de precisión en los resultados del modelo.

Ejemplo de cuantificación:

Un peso entrenado (float32): 0.123456789 (32 bits)

Después de cuantización a int8: 0.12 (8 bits, menos exacto)



La cuantización puede realizarse

- Post-Entrenamiento: después de entrenar el modelo completo, resulta más simple y es fácil de implementar.
- Durante el Entrenamiento: el modelo es entrenado teniendo en cuenta la cuantización y con eso mejora su precisión en comparación con los modelos cuantizados postentrenamiento.

Las **ventajas** son la reducción del tamaño del modelo y la aceleración de la inferencia (***benchmarks***).

Los ***benchmarks*** son **pruebas estandarizadas** que se usan para medir y comparar el rendimiento de un sistema. En el caso de TinyML, los *benchmarks* son conjuntos de modelos, datos y métricas diseñados para evaluar qué tan bien funciona un modelo de IA en dispositivos muy limitados como microcontroladores, sensores, etc. y suelen medir el tamaño del modelo (¿cabe en la memoria del microcontrolador?), la latencia de inferencia (¿cuánto tarda en dar respuesta?), el consumo de energía (¿cuánta batería gasta?) y la precisión (¿qué tan bien clasifica o predice?).

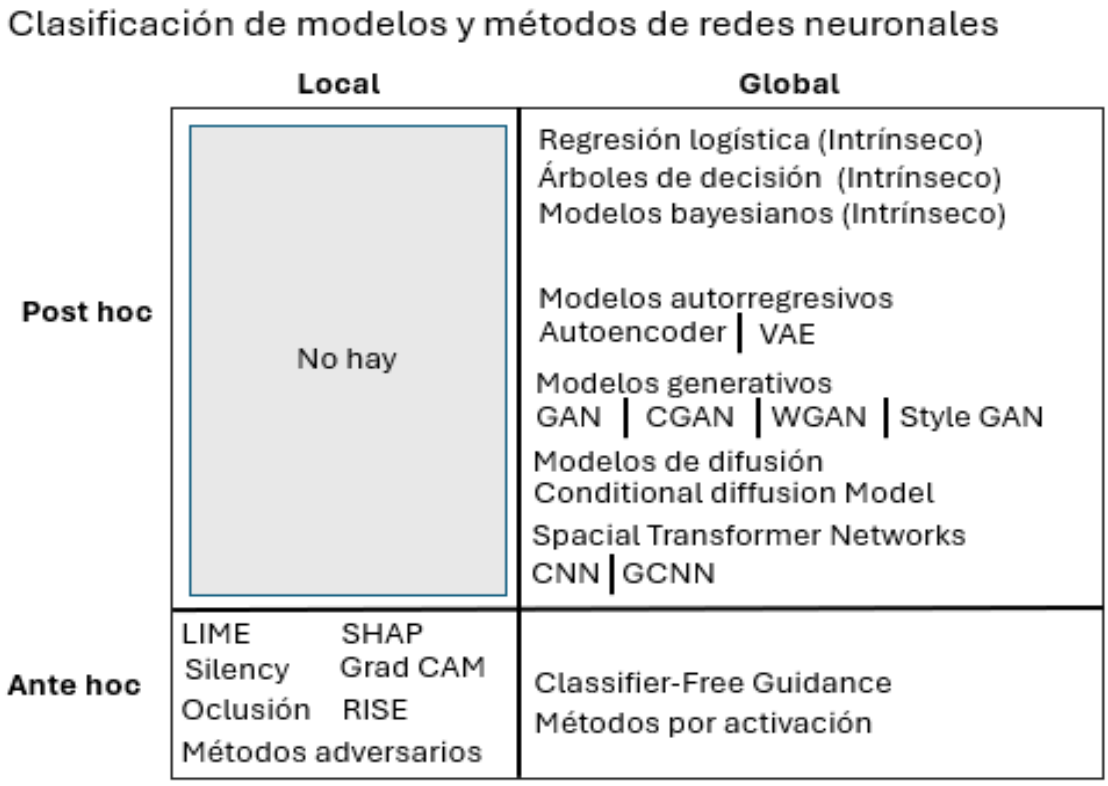
Los principales **desafíos** son el manejo de ***outliers***, la pérdida de precisión, el mantenimiento de la precisión en capas críticas y la dificultad de cuantización de operaciones complejas (exponenciales, raíces, etc.).

Un ***outlier*** (o valor atípico) es un **dato que se sale del patrón general** del resto de los datos. En TinyML, los *outliers* son especialmente importantes porque los modelos son pequeños y no tienen "tolerancia" para manejar datos raros, y tienen la capacidad de afectar mucho el rendimiento si no se tratan bien (desajustes t fallas).

Para manejarlos, en TinyML se busca detectarlos y eliminarlos antes de entrenar el modelo, se aplican técnicas de entrenamiento con métodos que no se vean tan afectados por valores extremos o una operación de cuantización con recorte (*clipping*) por la que el modelo ignora valores fuera de un rango razonable para no desperdiciar sus bits en números atípicos.

Anexo

Tabla comparativa de redes neuronales por modelos de interpretabilidad



Modelo / Método	**Clasificación**	**Principales usos**	**Caja**	**Enfoque**
Regresión logística *(Intrínseco)*	Ante hoc – Global	Clasificación simple, análisis predictivo en ciencias sociales, biomedicina	Caja blanca	Otros (modelo estadístico lineal)
Árboles de decisión *(Intrínseco)*	Ante hoc – Global	Clasificación y regresión, reglas claras de decisión	Caja blanca	Otros (estructura jerárquica)

Modelos bayesianos ***(Intrínseco)***	Ante hoc – Global	Inferencia probabilística, aprendizaje bajo incertidumbre	Caja blanca	Otros (probabilístico)
LIME	Post hoc – Local	Explicaciones individuales en modelos complejos	**Caja negra** (explica)	Perturbación
SHAP	Post hoc – Local y Global	Interpretación de la importancia de variables	**Caja negra**	Perturbación (valores de Shapley)
Saliency maps	Post hoc – Local	Identificación de características relevantes en imágenes	**Caja negra**	Gradientes
Grad-CAM	Post hoc – Local	Visualizaciones de activación en CNNs (mapas de calor)	**Caja negra**	Gradientes
Oclusión	Post hoc – Local	Interpretación eliminando/enmascarando entradas	**Caja negra**	Perturbación
RISE	Post hoc – Local	Relevancia mediante máscaras aleatorias	**Caja negra**	Perturbación
Concept Whitening **(Parcialmente Intrínseco)**	Ante hoc – Global	Interpretabilidad en CNNs con conceptos humanos	Caja blanca (parcial)	Otros (rotación de espacio latente)
CGNN (Causal Generative Neural Networks)	Ante hoc – Global	Modelado causal con generadores neuronales	**Caja negra**	Generativos
Spatial Transformer Networks	Ante hoc – Global	Invariancia espacial en imágenes (rotación, escala, traslación)	**Caja negra**	Otros (módulos de atención espacial)
Redes Neuronales Convolucionales (CNNs)	Ante hoc – Global (difíciles de interpretar)	Visión por computadora, imágenes, video	**Caja negra**	Otros (convolución, no explicativo per se)
Modelos autoregresivos (ej. GPT)	Ante hoc – Global	Generación secuencial (texto, audio, imágenes)	**Caja negra**	Generativos (autoregresivos)
Generative Adversarial Networks	Ante hoc – Global	Generación de imágenes, audio, datos sintéticos	**Caja negra**	Generativos + Adversarios

Modelos de difusión	Ante hoc – Global	Generación de imágenes de alta fidelidad	**Caja negra**	Generativos (difusión)
VAE (Variational Autoencoder)	Ante hoc – Global	Representaciones latentes, generación de datos	**Caja negra**	Generativos
Autoencoder (clásico)	Ante hoc – Global	Compresión, reducción de dimensionalidad	**Caja negra**	Otros (codificación / decodificación)
CGAN (Conditional GAN)	Ante hoc – Global	Generación condicionada a etiquetas (ej. imagen ↔ texto)	**Caja negra**	Generativos + Adversarios
WGAN (Wasserstein GAN)	Ante hoc – Global	Generación más estable de datos sintéticos	**Caja negra**	Generativos + Adversarios
StyleGAN	Ante hoc – Global	Síntesis realista de rostros y manipulación de atributos	**Caja negra**	Generativos + Adversarios
Conditional Diffusion Models	Ante hoc – Global	Generación condicionada en modelos de difusión	Caja negra	Generativos (difusión)
Classifier-Free Guidance	Post hoc – Global (usado en difusión)	Control de la generación sin clasificadores externos	Caja negra	Generativos (difusión, guidance)

Claves para leer la tabla

Ante hoc (caja blanca): modelos interpretables por construcción (

Ante hoc (caja negra): redes neuronales complejas que requieren técnicas de interpretabilidad.

Post hoc: técnicas de explicación aplicadas tras el entrenamiento.

Enfoques:

- *Perturbación*: alteran la entrada (LIME, SHAP, Oclusión, RISE, adversarios).
- *Gradientes: usan derivadas respecto a la entrada (Saliency, Grad-CAM).*
- *Generativos: crean datos o representaciones nuevas (GAN, VAE, difusión, autoregresivos).*
- *Adversarios: basados en juegos de dos redes o ataques (GAN, CGAN, WGAN, StyleGAN, adversarial methods).*
- *Otros: no encajan en lo anterior (árboles, regresión, bayesianos, CNNs básicas).*

El caso de la digitalización y procesamiento de placas espectrográficas mediante una aplicación de inteligencia de datos

La observación astronómica es el proceso de medir la luz (u otras señales) que llega desde un objeto celeste con instrumentos. La espectroscopia estelar es el análisis de las líneas de los espectros de las estrellas y es la herramienta más importante de la que dispone un astrónomo para investigar la naturaleza física de las estrellas.

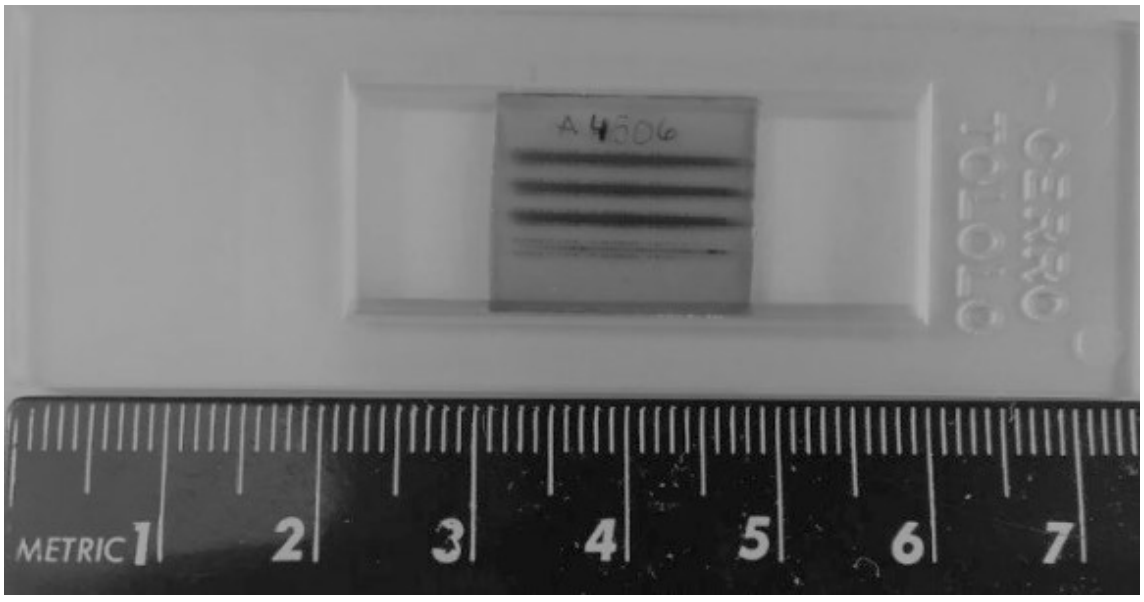
El responsable de la observación decide previamente qué medir, cuándo y cómo, y el resultado son datos científicos como imágenes, curvas de luz y *espectros*. La espectroscopia astronómica es una técnica de investigación que descompone la luz de un objeto celeste en sus longitudes de onda (colores) y mide cuánta energía llega en cada una. Ese "patrón" (el espectro) contiene huellas químicas y físicas del objeto estudiado.

El espectro capturado nos permite medir elementos como, entre otros, la composición química, la temperatura y la velocidad radial del objeto estudiado (corrimiento Doppler).

Para poder conocer la escala de registro de los colores, inmediatamente antes o después de realizar la observación se ilumina la placa donde se realiza el registro con una lámpara cuya composición química se conoce (por ejemplo: hierro, argón, neón), lo que generará una escala para calibrar los resultados que dé nuestra observación espectográfica. Conocer qué tipo de lámpara se ha utilizado para generar la escala es clave, ya que lámparas con elementos diferentes generan escalas diferentes.

La placa de captura es una placa fotográfica formada por una lámina de vidrio con emulsión fotosensible situada en el plano focal del telescopio, donde la luz forma una imagen que luego se revela químicamente. Ese método se usó desde 1920 hasta 1980. Hoy esa función la cumplen sensores digitales (CCD/CMOS), más sensibles y precisos

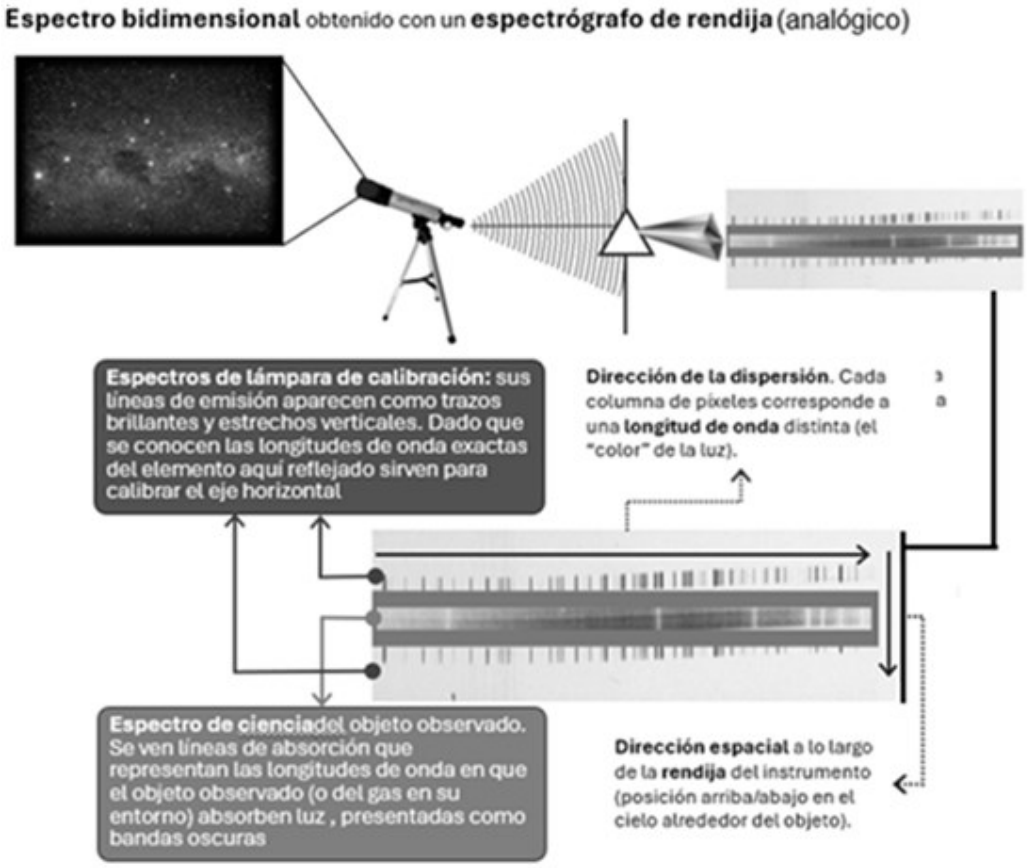
En el formato antiguo, el espectro de un objeto astronómico se registraba de manera bidimensional, mediante la dispersión de la luz del objeto medido en colores (longitudes de onda) que se registraban como una imagen de un segmento en blanco y negro sobre placas de vidrio.



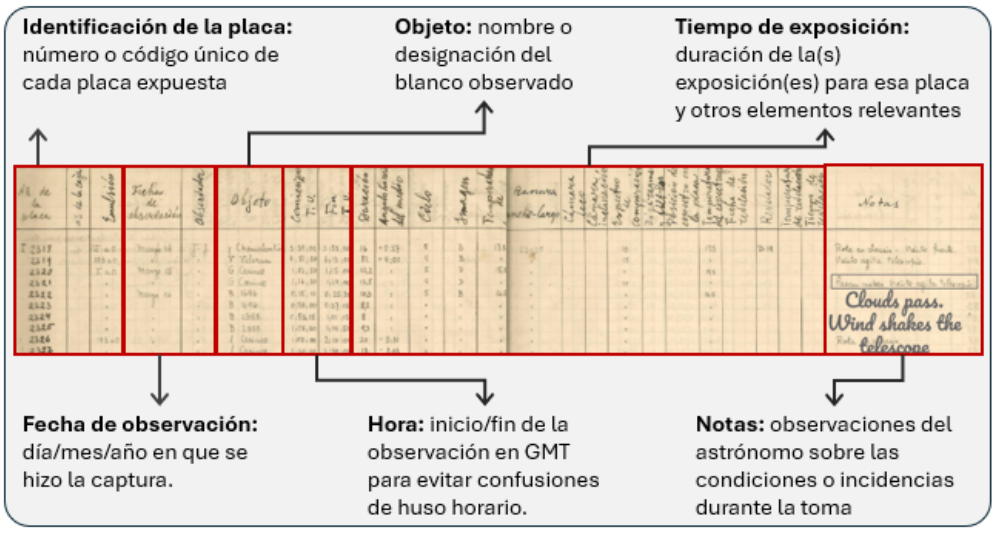
Vale la pena aclarar que, dadas las grandes distancias que nos separan de los cuerpos

celestes, para conocer sus cambios de composición o su movimiento es necesario en ocasiones contar con mediciones separadas por varias décadas, de allí que recuperar los registros espectográficos antiguos sea de enorme valor científico.

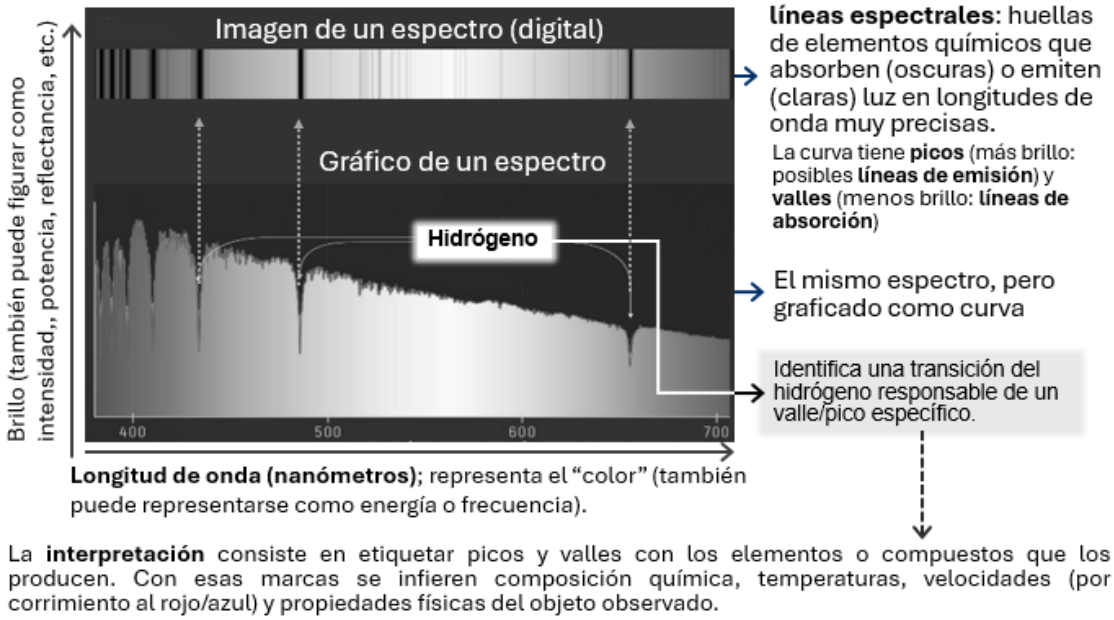
El siguiente cuadro muestra el proceso de registro de una observación mediante tecnologías analógicas.



Los datos registrados en estos espectros se acompañan de una notación de información complementaria como lugar en que se hizo el registro, día, hora y minuto y tiempo de exposición, que se incorporaban a un libro de registro de observaciones astronómicas (*logbook*)



Volviendo a la placa de registro del espectro, lo que se hace hoy es aplicar técnicas que permiten mejorar la visión y registrarla digitalmente para incluirlas en bases de datos: utilizando placas similares a la presentada, se interpreta cada línea vertical del espectro para obtener la intensidad promedio de cada píxel del espectro, lo que les permite armar un gráfico como el que mostramos a continuación:



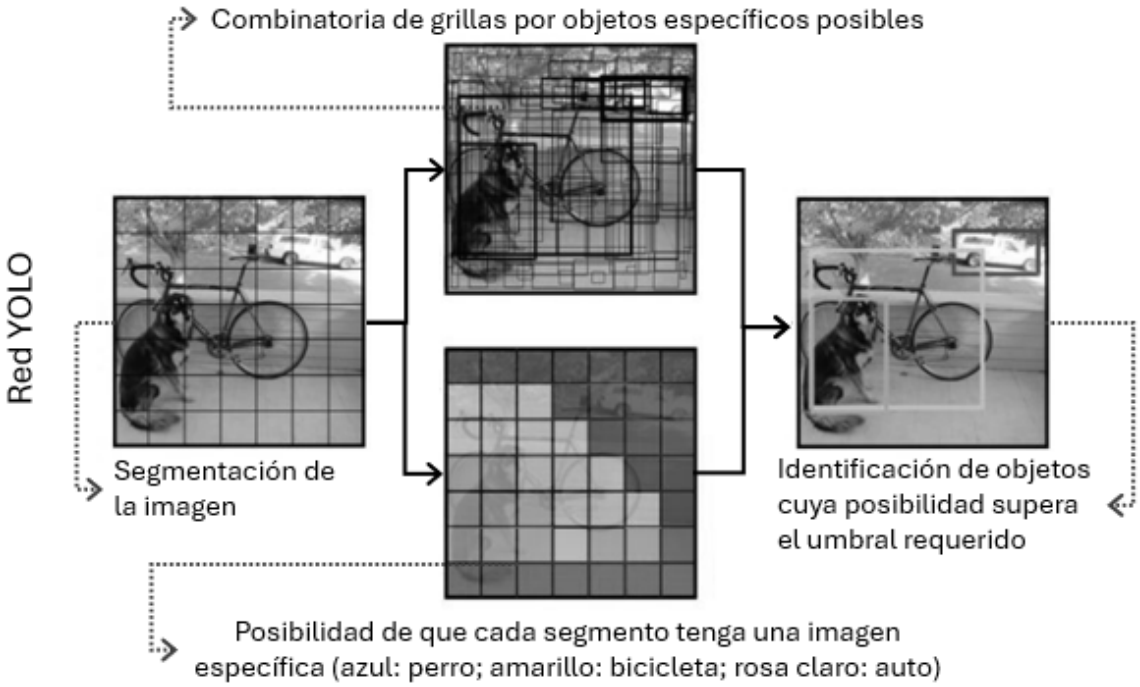
Nota: la imagen es a solo título ilustrativo, ya que su interpretación requiere verla en colores

Las líneas negras de la imagen superior (o los "pozos" en la inferior) se denominan "líneas espectrales" ya que muestran las ondas de luz que el objeto observado absorbe. Son claves en la observación. Ya que su posición nos permite conocer, por ejemplo, si el objeto se aproxima o se aleja de la Tierra y a qué velocidad

- Desplazamiento al rojo: Si la estrella se está alejando de la Tierra, las líneas espectrales en su registro se moverán hacia el extremo rojo del espectro (longitudes de onda más largas). La magnitud de este desplazamiento indica la velocidad a la que se aleja.
- Desplazamiento al azul: Si la estrella se está acercando a la Tierra, las líneas espectrales se moverán hacia el extremo azul del espectro (longitudes de onda más cortas). La magnitud del desplazamiento indica la velocidad a la que se acerca

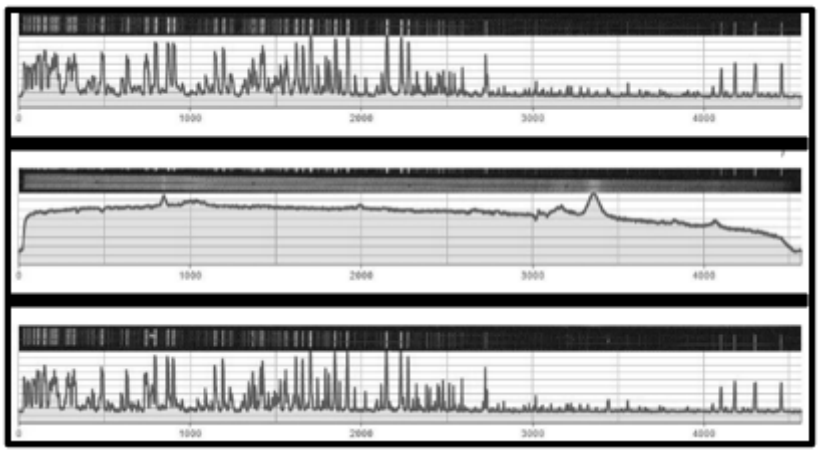
Hoy las redes neuronales permiten acelerar el proceso de análisis de los espectros a partir del procesamiento de imágenes y generar registros digitales de sus resultados, pero escoger una arquitectura para hacerlo apropiada fue un problema: se descartó el uso de RNN ya que actúan de manera secuencial, por lo que el procesamiento se hacía muy lento.

La utilización de redes convolucionales abrió más oportunidades, y en un trabajo realizado por la UNLP (Facultad de informática) se optó por una red convolucional YOLO (*You only look once*) donde se cuadricula la imagen y la red calcula la posibilidad de que un elemento específico se encuentre en una zona de la imagen. El usuario puede marcar el porcentaje de seguridad de que cada parte de la cuadrícula tenga una cierta imagen que se encuentre por encima de un umbral determinado (ejemplo: 70% de confianza).



Como cada cuadro de la grilla se procesa por separado, se puede hacer un procesamiento paralelo de todos ellos, acelerando el proceso de las RNN al evitar la secuencialidad.

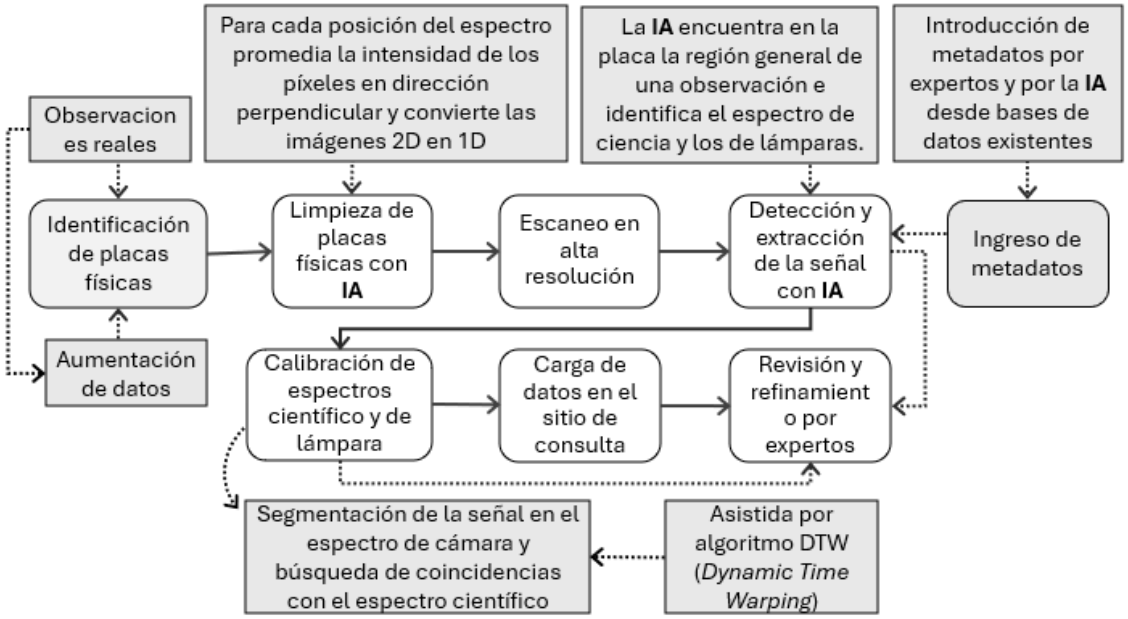
De este modo el registro espectográfico de una observación con soporte analógico pasa a ser un registro digital para cada una de las tres bandas que lo componen (espectros de lámpara y espectro de ciencia):



Al aplicar este esquema a placas espectográficas analógicas se creó una base de datos (dada la baja cantidad de espectros disponibles se hizo aumentación de datos para entrenarlo) dejando un 20% de los datos reales para hacer las pruebas de validación. A cada imagen se le incorporaron los metadatos técnicos de creación de la imagen y se le incorporan los datos de la propia observación (los que figuran en los libros de registro de observaciones astronómicas).

Una interfaz de usuario permite incorporar nuevas observaciones espectográficas analíticas con sus datos y trabajar con ellas, eligiendo además diferentes lámparas para hacer la medición sobre los espectros de lámparas, de tal forma que se pueda inferir los materiales que se registran en el espectro de ciencia.

RESUMEN DEL PROCESO



Bibliografía ampliatoria

Abu-zanona, A; Elaiwat, S; Younis, Innab; N. y Kamruzzaman, M. (2022). Classification of Palm Trees Diseases using Convolution Neural Network. *International Journal of Advanced Computer Science and Applications (IJACSA)*, (13)6. http://dx.doi.org/10.14569/IJACSA.2022.01306111

Bostrom, N. (2016). *Superinteligencia: Caminos, peligros, estrategias*. TEELL.

Goodfellow, I; Bengio, Y. y Courville, A. (2016) Deep Learning. (Adaptive Computation and Machine Learning series). *MIT Press.*

Marín Morales, R.L. (2008). *Inteligencia Artificial: Técnicas, Métodos y Aplicaciones*. McGraw-Hill Interamericana de España. 978-84-481-5618-3. 2008.

Marsland, S. (2015). *Machine Learning: An algorithmic perspective* (2nd Edition). CRC Press.

Molnar, C. (2020). *Interpretable machine learning*. Self-Published.

Marques, O. (2018, 8 de mayo) Explainable AI (XAI): Are we there yet? *MATLAB Help Center*. https://blogs.mathworks.com/deep-learning/2023/05/08/explainable-ai-xai-are-we-there-yet/

Reddi; v; Plancher, B; Kennedy, S; Moroney; L. Warden, P; Suzuki, L; Agarwal, A; Banbury, C; Banzi; M; Bennett⋄, M. Brown, M; Chitlangia, S; Ghosal, R; Grafman, S; Jaeger, R; Krishnan, S: Lam, M; Leiker, D; Mann, C; Mazumder, M; Pajak, D; Ramaprasad, D; Smith, E; Stewart, M. y Tingley, D. (2022). Widening Access to Applied Machine Learning With TinyML. *Harvard Data Science Review (HDSC)*, (1)4. DOI: 10.1162/99608f92.762d171a

Radečić, D. (2024, 8 de noviembre). R t-SNE: How to Visualize High-Dimensional Datasets in R. *Appsilon*. https://www.appsilon.com/post/r-tsne

Russell, S.y Norvig, P. (2004). *Inteligencia Artificial: Un Enfoque Moderno*. Pearson Educación. Madrid. 978-84-205-4003-0. 2004.

Sigman, M. y Bilinkis, S. (2023). *Artificial, la nueva inteligencia y el contorno de lo humano*. Debate. 978-98-779-5066-3.

Varoufakis, Y (2024) Tecnofeudalismo: el sigiloso sucesor del Capitalismo. Deusto. 978-84-234-3703-0.

Vaswani, A; Shazeer, N; Parmar, N; Uszkoreit, J. Jones, L; Gomez, A; Kaiser, L y Polosukhin, I (2017) Attention is all you need. *31st Conference on Neural Information Processing Systems (NIPS)*. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf