Facsímil 01

Los cimientos

Capítulo 12

Lo que deberías saber: recapitulación de fundamentos

686f6c61 · 2026

Capítulo 12: Lo que deberías saber: recapitulación de fundamentos

Entrando en el tema

Has recorrido once capítulos convertidos en una lectura lenta. Has construido una neurona, entrenado una red, propagado errores hacia atrás, tokenizado texto, explorado embeddings y contrastado paradigmas de aprendizaje.

Este capítulo no es un resumen. Es una **revisión activa**: cada sección recupera un concepto nuclear, lo reformula desde otro ángulo, lo conecta con el resto y te confronta con una pregunta. Si algo no te suena, el número de capítulo te dice exactamente dónde volver.

No es un examen. Es un espejo. Si te reconoces en estas páginas, puedes avanzar al facsímil 2.

---

1. Qué es y qué no es la IA

El concepto. La inteligencia artificial no es consciente, no «piensa» y no es infalible. Es un sistema de predicción estadística de tokens que opera sobre patrones aprendidos de cantidades masivas de datos.[1] La confusión entre elocuencia y corrección es el error más caro que puedes cometer al diseñar sistemas con IA.[2]

Para recordar. Cada vez que un LLM te responde con frases impecables, recuerda: está generando tokens a partir de patrones aprendidos, no deliberando como una persona que contrasta el mundo. La confianza con la que afirma algo no guarda relación directa con la veracidad de lo que afirma.

Ejemplo fresco. Imagina que le pides a un LLM que calcule `1573 × 2849`. Si le obligas a descomponer el cálculo paso a paso, aumentas las probabilidades de que cada parte sea coherente y verificable. Si le pides la respuesta directa, puede inventarse un número con total seguridad. No es que «sepa multiplicar» o «no sepa»: es que el contexto y el procedimiento que le das cambian la distribución de tokens que generará.

Vuelve al capítulo 1 si: no puedes explicar por qué la IA no es un oráculo ni un buscador.

---

2. Sistemas deterministas frente a sistemas probabilísticos

El concepto. `sumar(2, 3)` siempre devuelve 5. `preguntar("¿Qué es Rust?")` puede devolver tres respuestas distintas, todas correctas. En IA generativa, la variabilidad suele aparecer en el muestreo, en la configuración de inferencia y en el sistema que rodea al modelo.[3] Esto cambia cómo diseñas, cómo testeas y cómo despliegas.

Para recordar. No puedes hacer `assert respuesta == "Rust es un lenguaje..."`. Validarás propiedades: ¿contiene las palabras clave?, ¿respeta el formato?, ¿menciona los conceptos correctos? El cambio de mentalidad, de igualdad exacta a validación de propiedades, es el paso más importante al trabajar con IA.

Ejemplo fresco. Un sistema de atención al cliente usa un LLM para clasificar la urgencia de tickets. Dos ejecuciones con el mismo ticket pueden dar «urgente» y «normal». Si tu sistema despacha automáticamente basándose en esa clasificación, necesitas un umbral de confianza y, probablemente, revisión humana para los casos límite. La variabilidad no es un *bug*: es una característica del sistema que debes diseñar.

Vuelve al capítulo 2 si: no puedes explicar por qué `temperature = 0` no garantiza determinismo absoluto.

---

3. Los cinco principios

El concepto. Cinco ideas sostienen la IA moderna: aprendizaje supervisado (ejemplos etiquetados), no supervisado (encontrar estructura sin etiquetas), post-training (enseñar preferencias), atención (mirar todo a la vez) y *scaling laws* (relaciones empíricas entre datos, parámetros, cómputo y pérdida).[4]

Para recordar. No son compartimentos estancos. Un LLM se pre-entrena con aprendizaje auto-supervisado, se post-entrena con SFT (supervisado) y RLHF (refuerzo), y su arquitectura se basa en la atención. Las *scaling laws* ayudan a razonar sobre tendencias de escala bajo condiciones concretas; no sustituyen la evaluación de tu tarea, tu coste y tus datos.

Ejemplo fresco. Cuando OpenAI entrena un nuevo modelo, no es un solo paso. Primero, pre-entrenamiento masivo sobre texto de internet (auto-supervisado). Luego, SFT con conversaciones de alta calidad escritas por personas (supervisado). Finalmente, RLHF donde personas comparan respuestas (refuerzo). Tres paradigmas encadenados. La arquitectura Transformer con atención es el soporte que hace posibles los tres.

Vuelve al capítulo 3 si: no puedes nombrar los cinco principios y dar un ejemplo de cada uno.

---

4. La neurona artificial

El concepto. Una neurona artificial es una función matemática: $y = f(\sum_i w_i x_i + b)$. Entradas multiplicadas por pesos, sumadas, más un sesgo, pasadas por una función de activación.[5] Tres líneas de código. El átomo del *deep learning*.[6]

Para recordar. La inteligencia no está en una neurona. Está en los miles de millones de parámetros ajustados durante el entrenamiento a lo largo de capas y capas de neuronas interconectadas. Pero la unidad mínima es siempre la misma.

Ejemplo fresco. ¿Cuántas neuronas se activan cuando escribes «Buenos días» en ChatGPT? Cada token de entrada atraviesa todas las capas del modelo. En un modelo de 70B parámetros, una sola predicción involucra miles de millones de operaciones $y = f(\sum_i w_i x_i + b)$. Tu «Buenos días» desencadena una cascada de multiplicaciones y sumas que recorre toda la red en milisegundos.

Vuelve al capítulo 4 si: no puedes escribir la fórmula de memoria y calcular la salida para `x₁=0.5, x₂=0.3, w₁=0.2, w₂=-0.4, b=0.1` con activación ReLU.

---

5. Redes neuronales: capas y arquitectura

El concepto. Una red neuronal es un grafo de neuronas organizadas en capas.[7] Cada capa transforma los datos y los pasa a la siguiente. Las primeras capas detectan patrones simples; las profundas, conceptos abstractos.[8]

Para recordar. «*Deep*» significa «muchas capas». Más capas permiten más abstracción, pero también requieren más datos, más computación y técnicas para que el entrenamiento no se rompa: buena inicialización de los pesos, normalización (BatchNorm, LayerNorm) y conexiones residuales que dan un atajo al gradiente. La arquitectura determina lo que la red puede aprender.

Ejemplo fresco. Un MLP de 3 capas puede clasificar dígitos escritos a mano (MNIST). Un Transformer de 96 capas puede mantener una conversación de una hora. La neurona es la misma en ambos. Lo que cambia es la organización. La arquitectura es la decisión de diseño más importante después de elegir los datos.

Vuelve al capítulo 5 si: no puedes dibujar la estructura de una red con capa de entrada, dos ocultas y una de salida, y explicar qué aprende cada capa.

---

6. Retropropagación

El concepto. La retropropagación aplica la regla de la cadena para propagar el error desde la salida hacia atrás.[9] Para cada peso, calcula $\partial E/\partial w$: cuánto cambia el error si modifico este peso. La fórmula completa:

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial a} \cdot \sigma'(z) \cdot x$$

SÍMBOLO	SIGNIFICADO
$\partial E/\partial w$	Cuánto cambia el error si muevo un poco este peso
$\partial E/\partial a$	Cuánto cambia el error con la salida de la neurona
$\sigma'(z)$	Pendiente de la activación en el punto z
x	Entrada que llegó por esa conexión

En palabras: el error se propaga hacia atrás multiplicando tres factores encadenados, que es la regla de la cadena, y el producto indica en qué dirección y con qué fuerza corregir cada peso.

Para recordar. El gradiente dice «dirección y sensibilidad». El *learning rate* dice «tamaño del paso». Actualizamos con signo menos porque queremos **reducir** el error, no aumentarlo. La regla $w \leftarrow w - \eta \cdot \partial E/\partial w$ es la ecuación más importante del *deep learning*.

Ejemplo fresco. En el capítulo 6 calculaste que con `x=2, w=0.40, b=-0.10, y=1, η=0.1`, el gradiente `∂E/∂w = -0.148`. Tras la actualización, `w = 0.415`. La nueva predicción se acerca más a 1. Una iteración. Millones por delante. Esto, multiplicado por miles de millones de parámetros y repetido durante semanas en clústeres de GPUs, es como se entrena un LLM.

Vuelve al capítulo 6 si: no puedes calcular a mano el gradiente para el ejemplo anterior.

---

7. Funciones de pérdida y optimizadores

El concepto. La función de pérdida define qué significa «equivocarse». El optimizador decide cómo corregirlo.[10] *Cross-entropy* para clasificar, MSE para regresión. SGD es simple, Adam es adaptativo (mantiene dos medias móviles por parámetro, la del gradiente y la de su cuadrado, para adaptar el paso de cada uno), y AdamW, que además desacopla el *weight decay*, es el estándar para Transformers.[11]

Para recordar. La elección de la función de pérdida es la especificación formal de tu objetivo. Si usas MSE para clasificar, el modelo optimizará la distancia numérica entre probabilidades, no la probabilidad de la clase correcta. Aprenderá algo, pero no lo que quieres.

Ejemplo fresco. Tienes que clasificar 10 000 tickets de soporte en 5 categorías. Si usas MSE, el modelo tratará la diferencia entre «categoría 1» y «categoría 2» como numéricamente equivalente a la diferencia entre «categoría 1» y «categoría 5», lo cual no tiene sentido para categorías nominales. *Cross-entropy* con softmax respeta que las categorías son discretas y no ordenadas.

Vuelve al capítulo 7 si: no puedes explicar cuándo usar *cross-entropy* y cuándo MSE.

---

8. CNNs y RNNs

El concepto. Las CNNs detectan patrones locales en imágenes mediante filtros convolucionales.[12] Las RNNs procesan secuencias manteniendo un estado oculto. Las LSTMs añaden compuertas para recordar más tiempo.[13] El Transformer las superó con paralelismo y atención directa.[14]

Para recordar. Cada limitación que el Transformer resolvió era una limitación real de las RNNs. El estado oculto no bastaba para contexto largo. El procesamiento secuencial era demasiado lento. La atención directa y el paralelismo fueron las respuestas.

Ejemplo fresco. Procesar «El gato que perseguía al ratón que robó el queso que estaba en la cocina es negro» con una RNN: cuando llega a «es negro», el estado oculto ya ha sido sobrescrito decenas de veces y probablemente ha olvidado que el sujeto era «gato». Con un Transformer, la atención conecta directamente «negro» con «gato» sin importar cuántas palabras haya en medio.

Vuelve al capítulo 8 si: no puedes comparar CNN, RNN, LSTM y Transformer en una frase cada una.

---

9. Tokens y embeddings

El concepto. Un token es la unidad mínima de texto que procesa el modelo.[15] Un embedding es un vector denso de números que representa el significado de un token en un espacio de alta dimensionalidad.[16] Los embeddings son el pegamento entre el mundo del texto y el mundo de las matemáticas.

Para recordar. El español gasta más tokens que el inglés. «Machine learning is great» son 4 tokens; «El aprendizaje automático es genial» son 7-8. Todo se mide en tokens: contexto, precio, límites. Y la similitud coseno es la medida estándar para comparar embeddings.

Ejemplo fresco. Construyes un buscador semántico para documentación interna. Conviertes 5 000 documentos a embeddings (modelo `text-embedding-3-small`, 1536 dimensiones). Cuando un empleado busca «cómo solicitar vacaciones», el sistema compara el embedding de la consulta con los 5 000 embeddings de documentos y devuelve los más cercanos por similitud coseno. Encuentra la política de vacaciones aunque la consulta no comparta ni una palabra con el título del documento.

Vuelve al capítulo 9 si: no puedes explicar la diferencia entre token y embedding, y para qué sirve cada uno.

---

10. Entrenamiento frente a inferencia

El concepto. Entrenar es cambiar pesos con datos y una función de pérdida. Pre-entrenar un modelo fundacional puede requerir semanas, clústeres enormes y presupuestos altísimos; entrenar modelos pequeños o ajustar adaptadores puede ser mucho más accesible.[17] Inferir es usar pesos ya entrenados para responder. La cuantización reduce la precisión numérica de los pesos (FP16 → INT4) para que un modelo de 14 GB quepa en 3,5 GB.

Para recordar. Casi siempre harás inferencia, evaluación, RAG, integración o ajuste parcial antes que pre-entrenamiento de frontera. El *fine-tuning* está a medio camino. Y la cuantización es una herramienta para mover modelos a hardware más modesto, siempre que evalúes la pérdida de calidad en tu tarea. En inferencia, la KV cache acelera la generación token a token reutilizando lo ya calculado, y los adaptadores como LoRA permiten especializar un modelo grande entrenando solo una fracción diminuta de sus pesos.

Ejemplo fresco. Descargas un modelo de 7B parámetros en formato GGUF (cuantizado a 4 bits). Ocupa 3,5 GB. Lo ejecutas en tu portátil con llama.cpp, sin GPU, sin conexión a internet. La calidad es ligeramente inferior al modelo original en FP16, pero para resumir documentos internos, clasificar correos o generar primeras versiones, funciona perfectamente. Has pasado de depender de una API a tener el modelo en local.

Vuelve al capítulo 10 si: no puedes explicar la diferencia entre entrenamiento, inferencia, *fine-tuning* y cuantización.

---

11. Machine learning clásico

El concepto. Antes del *deep learning*, y todavía hoy para la mayoría de problemas, existe el ML clásico.[18] Clasificación, regresión, clustering, validación, matriz de confusión, *precision*, *recall*: este vocabulario no desaparece con los LLMs, se vuelve más importante.

Para recordar. La pregunta no es «¿qué modelo uso?» sino «¿qué salida necesito, qué señal tengo y cómo sé que generaliza?». Antes de aceptar un modelo, compáralo con una baseline, fija el umbral según el coste de cada error y revisa la calibración de sus probabilidades. Si necesitas explicar cada decisión, un árbol de decisión, que parte los datos por el corte que más reduce la impureza (Gini), se lee de la raíz a la hoja. Un *random forest* con 500 árboles puede ser más rápido, más barato y más interpretable que una red neuronal para tu problema concreto.[19]

Ejemplo fresco. Tienes 2 000 transacciones etiquetadas (147 operaciones irregulares, 1 853 legítimas). Un *random forest* te da 94 % de *recall* en la clase irregular (detectas 138 de 147) con 3 % de falsos positivos. Entrenar tarda 3 segundos. No necesitas GPU. Puedes explicar exactamente qué *features* usa el modelo para decidir. ¿Necesitas un LLM para esto?

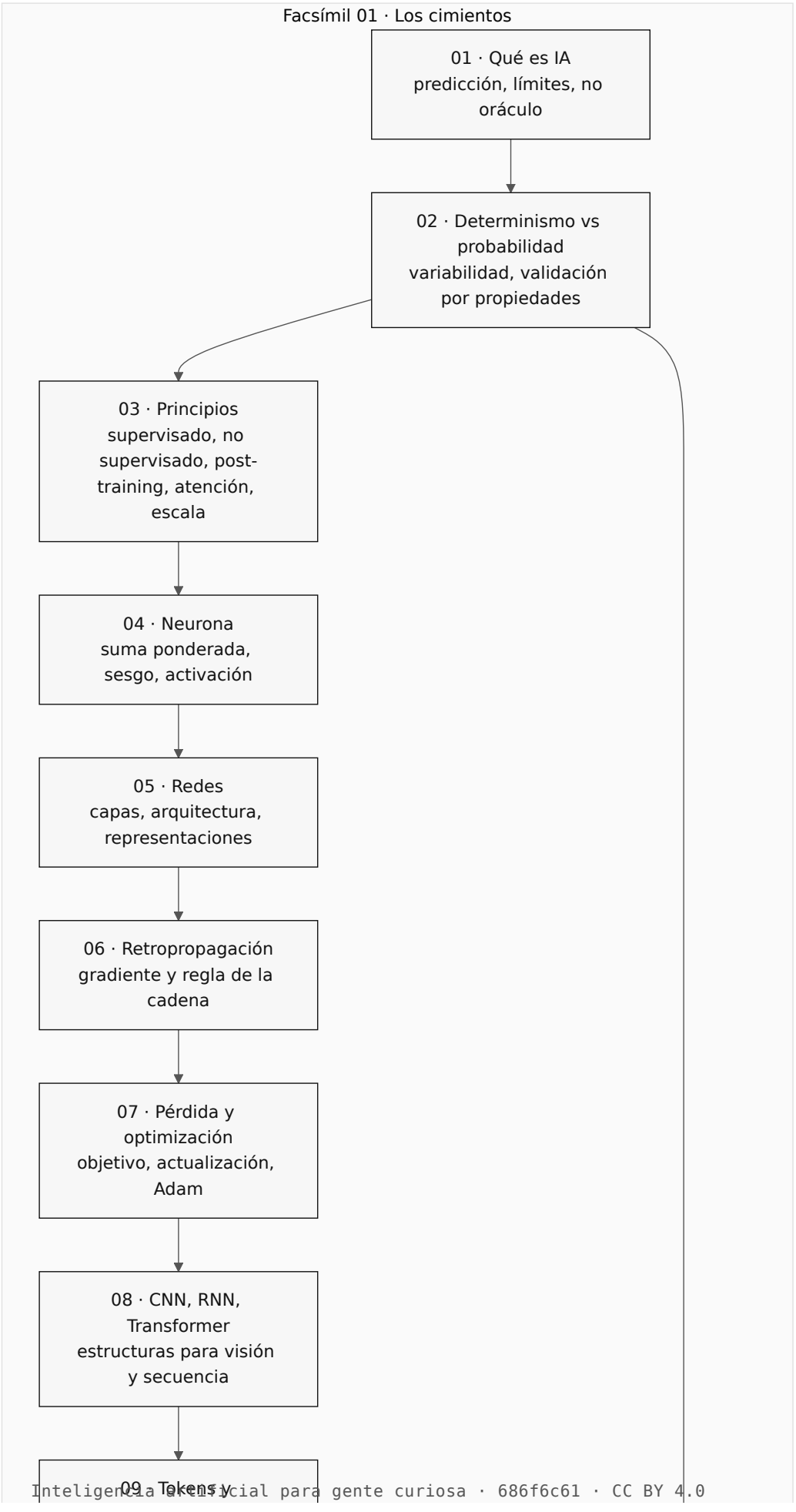
Vuelve al capítulo 11 si: no puedes definir *precision*, *recall* y explicar cuándo importa más uno que otro.

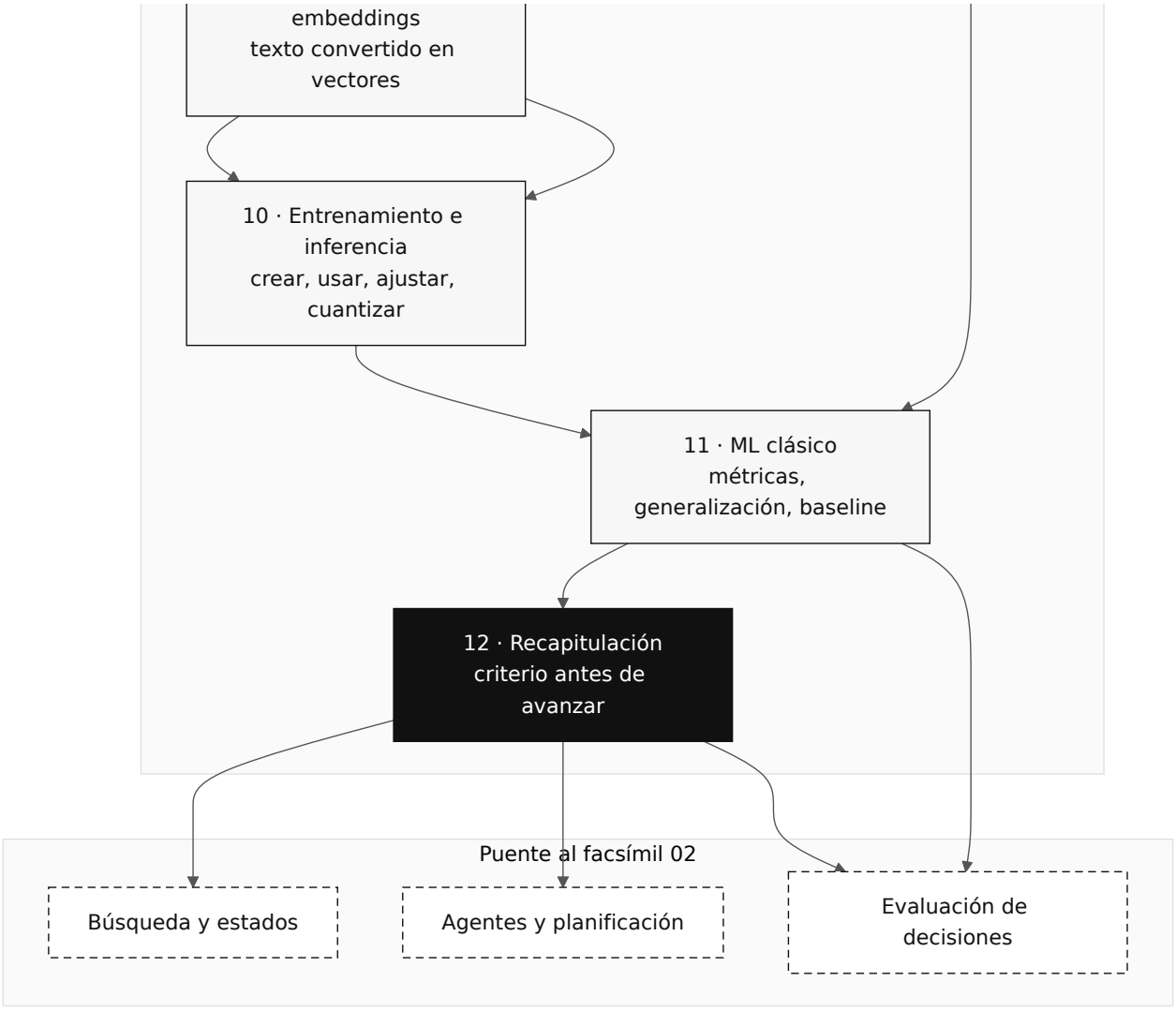
Cómo encaja todo

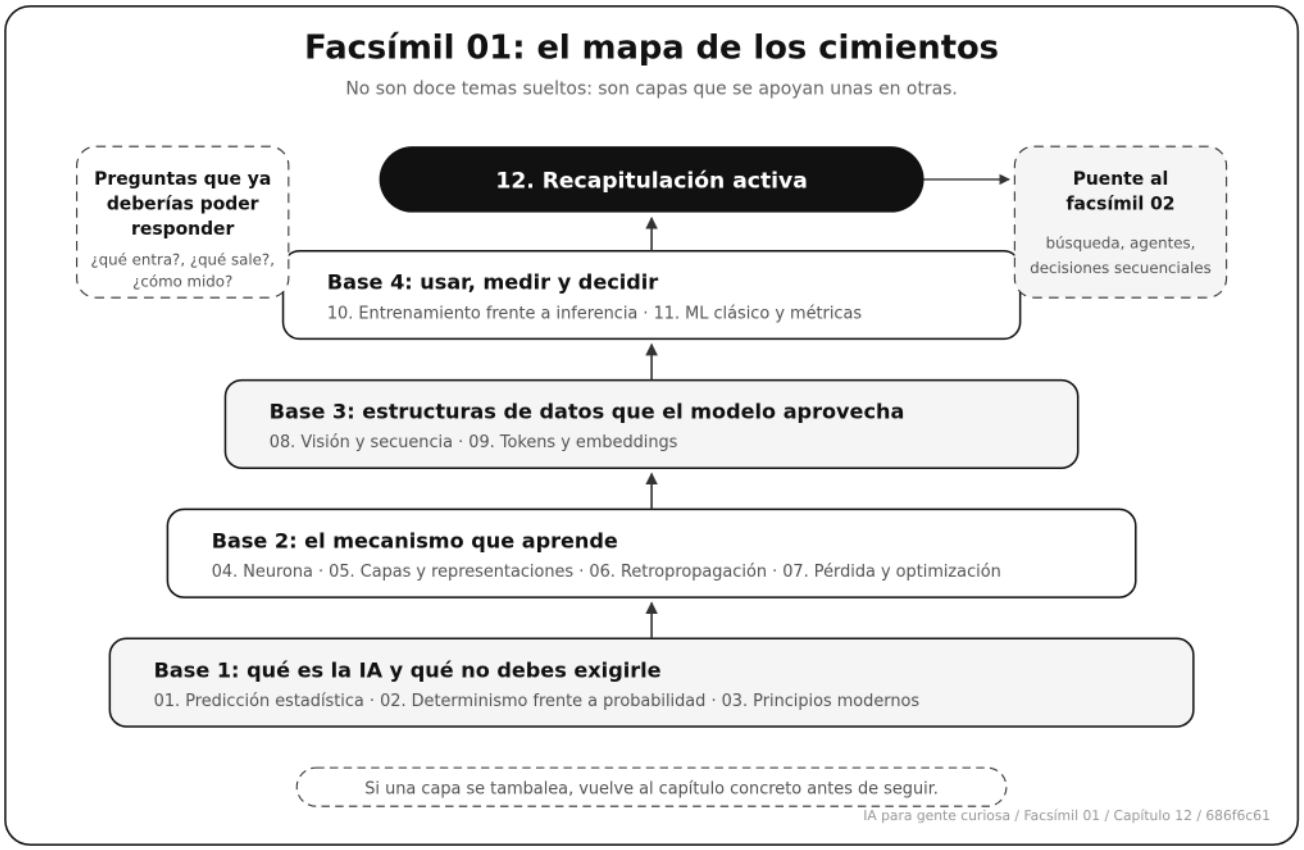
Este mapa no intenta repetir el facsímil entero. Léelo como una escalera: primero aclaramos qué puede y qué no puede hacer la IA, después construimos el mecanismo que aprende, luego convertimos texto e imágenes en representaciones matemáticas y, al final, aprendemos a usar, medir y comparar modelos.

La salida natural de este mapa es el facsímil 2. Allí la pregunta deja de ser «qué es un modelo» y pasa a ser «cómo busca, decide y planifica un sistema cuando tiene varias acciones posibles».

Facsímil 01 · Los cimientos




embeddings
texto convertido en
vectores
10 · Entrenamiento e
inferencia
crear, usar, ajustar,
cuantizar
11 · ML clásico
métricas,
generalización, baseline
12 · Recapitulación
criterio antes de
avanzar
Puente al facsímil 02
Búsqueda y estados
Agentes y planificación
Evaluación de
decisiones



Dónde solía tropezar yo

ERROR	POR QUÉ ES UN ERROR	ANTÍDOTO
Creer que recordar títulos es entender el facsímil	Puedes recitar “tokens, embeddings, backpropagation” y aun así no saber conectar las piezas.	Reexplica cada capítulo con un ejemplo propio y una fórmula mínima cuando exista.
Saltar al facsímil 2 con cimientos flojos	La búsqueda, los agentes y la planificación se apoyan en los conceptos de coste, error, representación y evaluación.	Si fallas una pregunta de la checklist, vuelve al capítulo concreto antes de seguir.
Confundir uso práctico con comprensión	Saber llamar a una API no implica entender inferencia, tokens, embeddings o evaluación.	Para cada herramienta que uses, escribe qué entrada transforma, qué salida produce y cómo medirías si funciona.
No medir	Sin métricas, cualquier demo parece buena hasta que falla en producción.	Antes de avanzar, identifica para cada sistema qué métrica usarías: pérdida, precisión, recall, coste o latencia.

Vocabulario aprendido

TÉRMINO	QUÉ DEBERÍAS PODER EXPLICAR YA
Inteligencia artificial	Sistema que transforma entradas en salidas útiles aprendiendo patrones, no una entidad consciente ni un oráculo.
Sistema probabilístico	Sistema cuya salida puede variar porque trabaja con distribuciones, muestreo e incertidumbre.
Neurona artificial	Función que combina entradas, pesos, sesgo y activación para producir una señal.
Red neuronal	Composición de muchas neuronas en capas que aprenden representaciones cada vez más útiles.
Gradiente	Dirección matemática que indica cómo cambiar un parámetro para modificar la pérdida.
Función de pérdida	Medida formal de error que define qué intenta mejorar el entrenamiento.
Token	Unidad mínima de texto que procesa un modelo de lenguaje.
Embedding	Vector numérico que representa texto, imagen u otra señal en un espacio matemático.
Inferencia	Uso de un modelo ya entrenado para producir una salida ante una entrada nueva.
Métrica	Medida que permite decidir si un modelo funciona para el objetivo real, no solo si parece convincente.

Antes de pasar página

Responde sin consultar los capítulos. Sé honesto con tu propio nivel: si fallas, el número te dice dónde volver.

- [ ] **1.** ¿Puedo explicar por qué un LLM no «piensa» ni «entiende», usando el concepto de predicción del siguiente token?
- [ ] **2.** ¿Sé por qué `temperature = 0` no garantiza que el modelo dé siempre la misma respuesta?
- [ ] **3.** ¿Puedo nombrar los cinco principios de la IA moderna y dar un ejemplo concreto de cada uno?
- [ ] **4.** ¿Puedo escribir la fórmula de una neurona artificial de memoria y calcular su salida para valores concretos?

- [ ] **5.** ¿Puedo explicar qué aprende cada capa de una red neuronal profunda?
- [ ] **6.** ¿Puedo calcular a mano el gradiente $\partial E/\partial w$ para una neurona con `x=2, w=0.4, b=-0.1, y=1` ?
- [ ] **7.** ¿Sé cuándo usar *cross-entropy* y cuándo MSE? ¿Y qué hace Adam que no hace SGD?
- [ ] **8.** ¿Puedo comparar CNN, RNN, LSTM y Transformer explicando la principal ventaja de cada una?
- [ ] **9.** ¿Entiendo la diferencia entre token y embedding? ¿Sé qué mide la similitud coseno?
- [ ] **10.** ¿Puedo explicar la diferencia entre entrenamiento, inferencia, *fine-tuning* y cuantización?
- [ ] **11.** ¿Sé qué mide la precisión y qué mide el *recall*? ¿Cuándo sacrificaría una por la otra?
- [ ] **12.** ¿Puedo explicar el *overfitting* y cómo detectarlo?

Diagnóstico: 10-12 respuestas sólidas: puedes avanzar al facsímil 2. 7-9: repasa los capítulos que fallaste. Menos de 7: vuelve a leer el facsímil con calma. No hay prisa. Los cimientos son eso: cimientos.

En resumen

IDEA FUERZA	DETALLE
La IA es predicción estadística, no conciencia.	Cada decisión de ingeniería depende de entender esto.
La neurona es el átomo, la red la molécula, la retropropagación el motor.	$y = f(\sum_i w_i x_i + b)$ → capas → $\partial E/\partial w$ → $w \leftarrow w - \eta \cdot \partial E/\partial w$. Todo se reduce a esto.
Los embeddings son el puente entre texto y matemáticas.	Sin ellos, las redes no podrían procesar lenguaje.
El ML clásico no ha muerto.	Para datos tabulares, pocos ejemplos o necesidad de explicabilidad, sigue siendo la mejor opción.
Si no sabes medir, no sabes si funciona.	*Precision*, *recall*, F1, validación, *overfitting*: el vocabulario para saber si tu modelo sirve.

Recursos para seguir: leer, construir y experimentar

Has recorrido los cimientos: la neurona, las redes, cómo aprenden, los tokens y los embeddings. Para que no se queden en teoría, aquí tienes recursos reales organizados por lo que quieras hacer.

Para experimentar sin escribir código. Casi todo lo de este facsículo se puede tocar en el navegador, y de hecho lo has hecho en las cajas «Pruébalo en 5 minutos» de cada capítulo: el TensorFlow Playground (`playground.tensorflow.org`) para ver una red aprender, ajustar capas y reventar el *learning rate*; CNN Explainer (`poloclub.github.io/cnn-explainer`) para mirar dentro de una red convolucional; el tokenizador de OpenAI (`platform.openai.com/tokenizer`) y el Embedding Projector (`projector.tensorflow.org`) para ver cómo el texto se vuelve números con significado; y Teachable Machine (`teachablemachine.withgoogle.com`) para entrenar tu propio clasificador con la webcam. Si quieres entenderlo en vídeo, la serie de 3Blue1Brown sobre redes neuronales es una de las mejores explicaciones visuales que existen.

Para construir. Cuando quieras pasar de jugar a programar, las librerías de referencia son PyTorch y TensorFlow/Keras para redes neuronales, y scikit-learn para el machine learning clásico (clasificadores, regresión, árboles). Para modelos ya entrenados que puedes usar o ajustar, Hugging Face (`huggingface.co`) es el punto de partida. No necesitas dominar las matemáticas para empezar: necesitas entender, y eso ya lo tienes.

Para leer. Cada capítulo cierra con su «Para saber más». Si tuvieras que quedarte con una base: entender bien la neurona y la retropropagación, porque todo lo demás (incluidos los grandes modelos de lenguaje de los siguientes facsículos) se construye encima de esas dos ideas. Los kits de este facsículo son el puente: son deterministas a propósito para que veas el mecanismo sin ruido, y te dejan a un paso de las librerías de verdad.

Para saber más

Bender, E. M., Gebru, T., McMillan-Major, A. y Shmitchell, S. (2021). On the dangers of stochastic parrots: can language models be too big? En *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency* (pp. 610-623). https://doi.org/10.1145/3442188.3445922

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5-32. https://doi.org/10.1023/A:1010933404324

Brown, T. B. y otros (2020). Language models are few-shot learners. En *Advances in Neural Information Processing Systems 33* (pp. 1877-1901). https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

Goodfellow, I., Bengio, Y. y Courville, A. (2016). *Deep learning*. MIT Press. https://www.deeplearningbook.org

Hastie, T., Tibshirani, R. y Friedman, J. (2009). *The elements of statistical learning* (2.ª ed.). Springer. https://web.stanford.edu/~hastie/ElemStatLearn/

Kaplan, J. y otros (2020). Scaling laws for neural language models. arXiv:2001.08361. https://doi.org/10.48550/arXiv.2001.08361

Kingma, D. P. y Ba, J. (2015). Adam: a method for stochastic optimization. En *International Conference on Learning Representations*. https://arxiv.org/abs/1412.6980

LeCun, Y., Bengio, Y. y Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436-444. https://doi.org/10.1038/nature14539

McCulloch, W. S. y Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4), 115-133. https://doi.org/10.1007/BF02478259

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 386-408. https://doi.org/10.1037/h0042519

Rumelhart, D. E., Hinton, G. E. y Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088), 533-536. https://doi.org/10.1038/323533a0

Russell, S. y Norvig, P. (2021). *Artificial intelligence: a modern approach* (4.ª ed.). Pearson.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. y Polosukhin, I. (2017). Attention is all you need. En *Advances in Neural Information Processing Systems 30* (pp. 5998-6008). https://papers.nips.cc/paper/7181-attention-is-all-you-need

Cuadernos para practicar

Has decidido, calculado y medido sobre los cuatro cimientos del facsímil; estos cuadernos te dejan *tocar* lo que has aprendido. Son *notebooks* que se abren en Google Colab —gratis, en el navegador— o te puedes descargar para correr en tu ordenador. Cada uno está explicado paso a paso, con salidas reales, y dice de qué capítulo sale. Empieza por el que más te pique.

Una red neuronal a mano con NumPy

Qué practicas: construir y entrenar una red neuronal desde cero, sin librerías de *deep learning*. **Dónde encaja:** capítulos 4 a 7 (la neurona, las capas, la retropropagación y los optimizadores). **Qué necesitas:** un navegador. Corre en CPU; ni GPU ni claves.

Partes de un caso cotidiano —un riego que debe activarse «cuando los dos sensores discrepan»— que esconde el problema XOR: el que enseñó a la IA, en los años setenta, que una sola neurona no basta. Primero lo intentas con una neurona y la ves *encogerse de hombros*: devuelve 0.5 para todo y acierta 2 de 4. Luego añades una capa oculta de cuatro neuronas, escribes la retropropagación *a mano* (sin cajas negras) y la red pasa a acertar 4 de 4 con total seguridad. Por el camino ves la curva de pérdida bajando época a época y la frontera de decisión *curvándose*, que es justo lo que una recta no podía hacer. Cierra con cuatro experimentos para que rompas cosas: quita la capa oculta, sube el ritmo de aprendizaje, cambia la semilla.

No es un «hola mundo»: es ver, con números reales, por qué la profundidad existe.

► Abrir en Google Colab **↓ Descargar el cuaderno (.ipynb)**

Machine learning clásico que decide de verdad

Qué practicas: clasificar texto con un modelo clásico (TF-IDF + regresión logística) y decidir según el coste de cada error. **Dónde encaja:** capítulo 11 (*machine learning* clásico) y capítulo 2 (determinista frente a probabilístico). **Qué necesitas:** un navegador. Corre en CPU; sin GPU ni claves.

Montas un clasificador para una bandeja de soporte: leer cada mensaje y decidir si es una *incidencia* (algo roto, hay que actuar) o una *consulta* (una duda que espera). Lo entrenas con mensajes reales en español, lo mides con una **matriz de confusión** —que no te dice solo cuánto acierta, sino *cómo* se equivoca— y descubres que una incidencia tratada como consulta es el error caro: un sistema roto que nadie atiende. Entonces mueves el **umbral** de decisión: bajas el listón para atrapar la incidencia que se escapaba (de 1 a 0) a cambio de molestar a alguna consulta de más (de 1 a 7). Cierras leyendo qué palabras delatan a cada clase, porque un modelo clásico —a diferencia de una red— se puede abrir y entender.

Es el 80% de los problemas de empresa resueltos sin un solo gigavatio: saber cuándo basta con lo clásico es criterio, no pereza.

► Abrir en Google Colab **↓ Descargar el cuaderno (.ipynb)**

Entrenamiento vs inferencia: el sobreajuste en directo

Qué practicas: distinguir memorizar de aprender, y por qué un error de entrenamiento bajo puede engañarte. **Dónde encaja:** capítulo 10 (entrenamiento frente a inferencia). **Qué necesitas:** un navegador. Corre en CPU; sin GPU ni claves.

Ajustas curvas de complejidad creciente a unos pocos datos con ruido y ves, en una sola imagen, el error más caro del *machine learning*. Un polinomio de grado 1 se queda corto; uno de grado 15 pasa por todos los puntos de entrenamiento —incluido el ruido— y entre punto y punto se dispara. Luego dibujas las dos curvas que lo cuentan todo: el error de entrenamiento, que solo baja, y el de prueba (datos no vistos), que baja y vuelve a subir formando una **U**. El error de entrenamiento es mínimo justo en el grado más complejo, pero el modelo que generaliza está en el grado 4. Si solo miraras lo primero, elegirías el peor modelo con total confianza.

No es teoría: es la diferencia exacta entre una demo que deslumbra y un sistema que aguanta en producción.

► **Abrir en Google Colab** ↓ **Descargar el cuaderno (.ipynb)**

Notas

1. Russell, S. y Norvig, P. (2021). *Artificial intelligence: a modern approach* (4.ª ed.). Pearson. ↩

2. Bender, E. M., Gebru, T., McMillan-Major, A. y Shmitchell, S. (2021). On the dangers of stochastic parrots: can language models be too big? En *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency* (pp. 610-623). https://doi.org/10.1145/3442188.3445922 ↩

3. Russell, S. y Norvig, P. (2021). *Artificial intelligence: a modern approach* (4.ª ed.). Pearson. Los capítulos 13-14 abordan la incertidumbre y el razonamiento probabilístico. ↩

4. Kaplan, J. y otros (2020). Scaling laws for neural language models. arXiv:2001.08361. https://doi.org/10.48550/arXiv.2001.08361 ↩

5. McCulloch, W. S. y Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4), 115-133. https://doi.org/10.1007/BF02478259 ↩

6. Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 386-408. https://doi.org/10.1037/h0042519 ↩

7. Goodfellow, I., Bengio, Y. y Courville, A. (2016). *Deep learning*. MIT Press. https://www.deeplearningbook.org ↩

8. LeCun, Y., Bengio, Y. y Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436-444. https://doi.org/10.1038/nature14539 ↩

9. Rumelhart, D. E., Hinton, G. E. y Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088), 533-536. https://doi.org/10.1038/323533a0 ↩

.0. Goodfellow, I., Bengio, Y. y Courville, A. (2016). *Deep learning*. MIT Press. https://www.deeplearningbook.org ↩

.1. Kingma, D. P. y Ba, J. (2015). Adam: a method for stochastic optimization. En *International Conference on Learning Representations*. https://arxiv.org/abs/1412.6980 ↩

.2. Krizhevsky, A., Sutskever, I. y Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. En *Advances in Neural Information Processing Systems 25* (pp. 1097-1105). https://papers.nips.cc/paper/4824 ↩

.3. Hochreiter, S. y Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735 ↩

.4. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. y Polosukhin, I. (2017). Attention is all you need. En *Advances in Neural Information Processing Systems 30* (pp. 5998-6008). https://papers.nips.cc/paper/7181-attention-is-all-you-need ↩

.5. Brown, T. B. y otros (2020). Language models are few-shot learners. En *Advances in Neural Information Processing Systems 33* (pp. 1877-1901). https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html ↩

.6. Mikolov, T., Chen, K., Corrado, G. y Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv:1301.3781. https://arxiv.org/abs/1301.3781 ↩

.7. Kaplan, J. y otros (2020). Scaling laws for neural language models. arXiv:2001.08361. https://doi.org/10.48550/arXiv.2001.08361 ↩

.8. Hastie, T., Tibshirani, R. y Friedman, J. (2009). *The elements of statistical learning* (2.ª ed.). Springer. https://web.stanford.edu/~hastie/ElemStatLearn/ ↩

.9. Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5-32. https://doi.org/10.1023/A:1010933404324 ↩