Facsímil 01

Los cimientos

Capítulo 05

Redes neuronales: capas, arquitectura y flujo

686f6c61 · 2026

Capítulo 05: Redes neuronales: capas, arquitectura y flujo

Entrando en el tema

En el capítulo anterior construiste una neurona. Una. Tres entradas, tres pesos, un sesgo y una salida. Funciona. Pero una neurona sola no llega muy lejos: puede trazar una línea recta entre dos grupos de puntos y poco más.

El salto empieza cuando conectas neuronas entre sí. Miles, millones, miles de millones de ellas. Organizadas en capas, cada una transformando los datos un poco más, pasando el resultado a la siguiente. Como una cadena de montaje donde cada estación añade una capa de abstracción.

Eso es una red neuronal. Y este capítulo explica cómo se organiza.

De la neurona a la capa

Una capa es un conjunto de neuronas que reciben las mismas entradas pero tienen sus propios pesos y sesgos.[1] Si la capa anterior tiene 3 neuronas y esta capa tiene 5, hay 3 × 5 = 15 conexiones, cada una con su propio peso. Más 5 sesgos, uno por neurona.

Cada neurona de la capa hace exactamente lo que viste en el capítulo 4: multiplica entradas por pesos, suma, suma el sesgo, aplica activación. La diferencia es que ahora sus entradas no son los datos originales (como «horas de estudio» o «temperatura»), sino las salidas de las neuronas de la capa anterior.

Y así, capa tras capa, los datos se transforman. Lo que empezó como una fila de números crudos termina como una predicción.

En una red real no solemos escribir neurona por neurona. Escribimos capas con matrices. Si la capa anterior produce un vector de tamaño d_{l-1} y la capa actual tiene d_l neuronas, la capa se calcula así:

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}$$

$$a^{(l)} = f^{(l)}(z^{(l)})$$

SÍMBOLO	FORMA	QUÉ SIGNIFICA
$a^{(l-1)}$	d_{l-1}	Activaciones que llegan desde la capa anterior. En la primera capa, es el vector de entrada x.
$W^{(l)}$	$d_l \times d_{l-1}$	Matriz de pesos. Una fila por neurona de la capa actual y una columna por entrada recibida.
$b^{(l)}$	d_l	Vector de sesgos. Uno por neurona de la capa actual.
$z^{(l)}$	d_l	Puntuaciones antes de aplicar activación.
$a^{(l)}$	d_l	Salida de la capa después de la activación.

La regla de ingeniería es simple: **la salida de una capa debe tener la misma dimensión que la entrada esperada por la siguiente**. Si una capa entrega 64 valores, la siguiente necesita una matriz de pesos con 64 columnas. Este contrato parece una minucia, pero es una de las primeras cosas que se comprueban cuando una arquitectura no arranca.

También podemos contar parámetros antes de entrenar:

$$\text{parámetros de la capa } l = d_l \cdot d_{l-1} + d_l$$

El primer término son pesos. El segundo, sesgos. Para una red con capas $[4, 32, 16, 3]$, el conteo sería:

CONEXIÓN	CÁLCULO	PARÁMETROS
Entrada 4 → Oculta 32	$32 \cdot 4 + 32$	160
Oculta 32 → Oculta 16	$16 \cdot 32 + 16$	528
Oculta 16 → Salida 3	$3 \cdot 16 + 3$	51
Total	160 + 528 + 51	**739**

Este número no mide inteligencia. Mide coste y capacidad potencial. Si tienes 200 filas de datos y una red con 3 millones de parámetros, probablemente no estás haciendo ciencia: estás dando al modelo demasiadas formas de memorizar.

Para situar la escala: esta red de juguete tiene 739 parámetros. Un modelo de lenguaje de los que usas a diario tiene entre 7.000 y 70.000 millones. La operación de fondo es exactamente la misma (multiplicar por pesos, sumar el sesgo, activar), repetida miles de millones de veces

y organizada en bloques especializados. Lo que cambia no es la pieza, sino cuántas hay y cómo se conectan.

La arquitectura de una red

La estructura más básica se llama **perceptrón multicapa** (*MLP*, *multilayer perceptron*). Tiene tres tipos de capas:

```
Capa de entrada → Capa(s) oculta(s) → Capa de salida
```

Capa de entrada. No hace cálculos. Simplemente recibe los datos y los distribuye a la primera capa oculta. Si tu *dataset* tiene 10 columnas, tu capa de entrada tiene 10 neuronas. Una por *feature*.

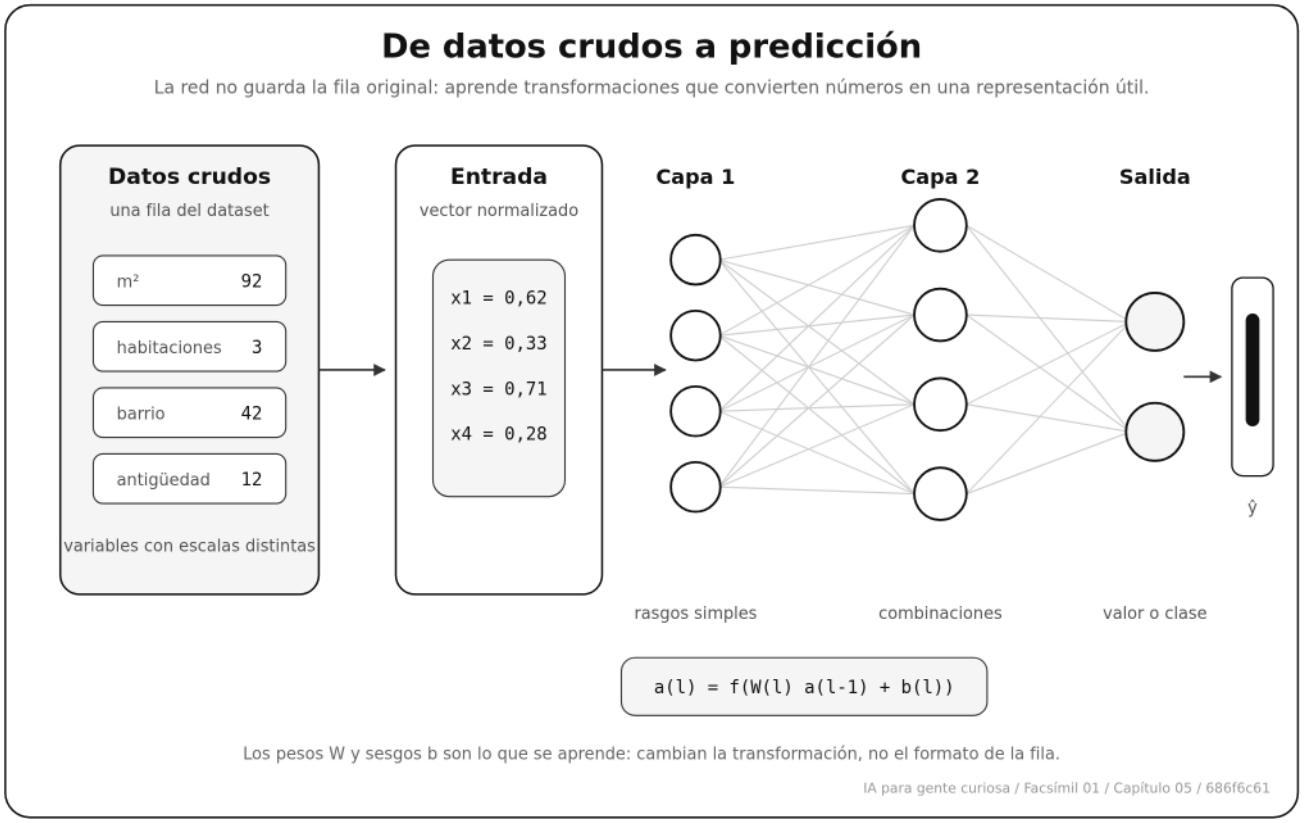
Capas ocultas. Aquí ocurre el aprendizaje. Una red puede tener una, diez o cien capas ocultas. Cada una transforma la representación de los datos en algo ligeramente más abstracto. Son «ocultas» porque no las ves desde fuera: solo ves lo que entra y lo que sale.

Capa de salida. Produce la predicción final. Si estás clasificando imágenes de dígitos (0-9), tu capa de salida tendrá 10 neuronas, una por clase, con softmax como activación. Si estás prediciendo el precio de una casa, tendrá 1 neurona, sin activación (o con activación lineal).

Para diseñarla con criterio, no empieces preguntando «¿cuántas capas pongo?». Empieza por el contrato del problema:

PREGUNTA DE DISEÑO	DECISIÓN TÉCNICA	EJEMPLO
¿Cuántas variables entran?	Tamaño de la capa de entrada.	24 columnas numéricas y categóricas codificadas → entrada de 24 dimensiones.
¿Qué debe salir?	Tamaño y activación de salida.	Binario → 1 salida + sigmoide. Multiclase excluyente → k salidas + softmax. Regresión → 1 salida lineal.
¿Cuánta no linealidad necesito?	Número de capas ocultas.	Datos tabulares sencillos: 1-3 capas. Imagen, audio o lenguaje: arquitecturas especializadas.
¿Cuánta capacidad puedo permitirme?	Ancho de cada capa y número total de parámetros.	64 neuronas pueden bastar para un clasificador interno; millones de parámetros exigen más datos y GPU.
¿Cómo voy a medir si funciona?	Métrica y conjunto de validación antes de entrenar.	Accuracy sola no basta si las clases están desbalanceadas; mira F1, recall, calibración o error absoluto según tarea.

Esta tabla evita una trampa común: diseñar una red como quien elige muebles. En ingeniería, arquitectura no es estética; es una hipótesis medible sobre datos, tarea, capacidad y coste.



Qué aprende cada capa

Una de las intuiciones más poderosas del *deep learning* es que las capas aprenden representaciones jerárquicas. No es una metáfora: es lo que se observa empíricamente al visualizar las activaciones de redes entrenadas.[2]

Capas tempranas. Detectan los patrones más elementales. En visión: bordes, colores, orientaciones. En texto: combinaciones de letras, prefijos y sufijos frecuentes. Son patrones que aparecen en cualquier imagen o texto, independientemente del contenido.

Capas intermedias. Combinan los patrones simples en conceptos más complejos. En visión: formas, texturas, partes de objetos (una rueda, un ojo, una esquina). En texto: frases hechas, relaciones sintácticas entre palabras.

Capas profundas. Abstracciones de alto nivel. En visión: objetos completos, rostros, escenas. En texto: significado semántico, intención del hablante, estructura argumental.

Esta jerarquía explica por qué el *transfer learning* funciona: las primeras capas de una red entrenada para reconocer imágenes sirven para casi cualquier tarea visual. Los bordes son bordes en todas partes. Son las capas profundas las que necesitan especializarse.[3]

Qué significa «deep» en *deep learning*

«*Deep*» significa «muchas capas». Una red con dos o tres capas es *shallow* (superficial). Una con decenas o cientos es *deep*. No hay un umbral oficial: la profundidad es una cuestión de grado.

Más capas permiten más abstracción, pero también traen problemas:

- **Más parámetros que entrenar**: más memoria, más computación, más datos necesarios.
- **Gradientes que se desvanecen**: en redes muy profundas, el gradiente puede hacerse tan pequeño en las primeras capas que dejan de aprender. Las funciones de activación como ReLU y arquitecturas como ResNet mitigan esto.[4]

La decisión de cuántas capas usar no es arbitraria: depende del problema, de los datos disponibles y del presupuesto computacional. Más capas no siempre es mejor. Pero cuando tienes datos masivos y tareas complejas (como procesar lenguaje natural), la profundidad marca la diferencia.

Lo que hace entrenable una red profunda

Apilar capas no basta. Si solo encadenas matrices y activaciones, una red muy profunda se entrena fatal: los gradientes se desvanecen o explotan, las activaciones se desestabilizan capa a capa y el modelo memoriza en vez de generalizar. Cuatro piezas, todas presentes dentro de un Transformer moderno, son las que arreglan esto.

Inicialización de pesos. Una red no arranca con los pesos a cero, sino con valores aleatorios pequeños y bien escalados. Si arrancan demasiado grandes, las activaciones explotan; si arrancan demasiado pequeños, se desvanecen ya en la primera pasada. Hay recetas para escalarlos según la activación: *He* para ReLU, *Xavier* (o *Glorot*) para tanh y sigmoide. No es un detalle menor: una mala inicialización puede impedir que una red profunda aprenda nada.

Normalización. Tras una capa, las activaciones pueden tomar cualquier escala, y eso desestabiliza el entrenamiento. La normalización las recentra y reescala para mantenerlas en un rango estable. Hay dos variantes habituales: *Batch Normalization* normaliza usando las estadísticas del lote, y *Layer Normalization* normaliza cada ejemplo por separado.[5] **LayerNorm es la que usan los Transformers**, y por tanto la que hay dentro de los LLMs.

Conexiones residuales. En lugar de pasar la salida de una capa tal cual, se le suma su propia entrada:

$$\text{salida} = x + f(x)$$

Es decir, la capa solo tiene que aprender la *diferencia* respecto a la entrada, no toda la transformación. Y, sobre todo, el `+ x` crea un atajo por el que el gradiente fluye directo hacia atrás sin desvanecerse. Esta idea simple es la que permitió entrenar redes de cientos de capas (*ResNet*) y está en cada bloque de cada Transformer.

Dropout y regularización. Son la respuesta directa al sobreajuste que aparece tantas veces en este capítulo. El *dropout* apaga al azar un porcentaje de neuronas en cada paso de entrenamiento, forzando a la red a no depender de ninguna en exclusiva y a aprender representaciones redundantes y robustas. El *weight decay* (regularización L2) penaliza los pesos grandes para que el modelo no se ajuste al ruido. Ninguna de las dos cambia la arquitectura: se activan al entrenar y se apagan al usar el modelo.

Cómo fluyen los datos

El viaje de los datos a través de una red se llama **propagación hacia delante** (*forward pass*). Es determinista: dados los mismos datos de entrada y los mismos pesos, la salida es siempre la misma.

En el *forward pass*, cada capa:

1. Recibe un vector de números de la capa anterior.
2. Multiplica cada entrada por su peso correspondiente.
3. Suma todo y añade el sesgo.
4. Aplica la función de activación.
5. Pasa el resultado a la capa siguiente.

Al final de la última capa, tienes una predicción. En el siguiente capítulo veremos cómo, a partir del error de esa predicción, la red ajusta todos sus pesos hacia atrás. Pero por ahora, quédate con esto: una red neuronal es simplemente una función matemática compuesta, donde la salida de cada capa es la entrada de la siguiente.

En código de producción suele aparecer una dimensión adicional: el **batch**. No procesas una sola fila, sino muchas a la vez. Si tienes 128 ejemplos y cada uno tiene 24 variables, la entrada puede tener forma 128×24. La capa no cambia su contrato interno: sigue esperando 24 columnas. Lo que cambia es que calcula 128 casos en paralelo.

OBJETO	FORMA TÍPICA	LECTURA
Entrada de un ejemplo	24	Una fila con 24 variables.
Batch de entrada	128×24	128 filas a la vez.
Pesos de la primera capa	64×24	64 neuronas, cada una mira las 24 variables.
Salida de la primera capa	128×64	128 filas, ahora representadas con 64 activaciones.
Salida final binaria	128×1	Una probabilidad por ejemplo.

Cuando una librería te devuelve un error de dimensiones, normalmente te está diciendo que una de estas piezas no encaja. Un ingeniero no lo resuelve probando números al azar: imprime formas, revisa contrato y comprueba dónde se rompió la cadena.

Presupuesto de una arquitectura

Antes de entrenar, puedes hacer una revisión fría de la red: formas, parámetros y memoria aproximada. No predice la calidad final, pero evita diseños absurdos.

Imagina un clasificador para priorizar tickets internos. Cada ticket ya está convertido a 48 variables numéricas: señales del usuario, categoría, antigüedad, canal, idioma y métricas históricas. La salida tiene 3 clases: baja, media y alta.

ARQUITECTURA	CAPAS	PARÁMETROS	LECTURA
MLP pequeño	$48 \to 32 \to 3$	1.667	Buen primer modelo. Barato, fácil de depurar y suficiente si las señales son fuertes.
MLP medio	$48 \to 128 \to 64 \to 3$	14.723	Más capacidad. Tiene sentido si hay bastantes ejemplos y relaciones no lineales.
MLP excesivo	$48 \to 1024 \to 1024 \to 3$	1.102.851	Puede funcionar, pero con pocos datos memorizará. También aumenta latencia, memoria y coste.

La comparación importante no es «qué red parece más profesional», sino qué red puedes justificar con datos. Si tienes 5.000 ejemplos limpios, empezar por un MLP pequeño o medio es razonable. Si tienes 200 ejemplos y clases desbalanceadas, el problema no se arregla con más capas: se arregla revisando datos, particiones, métrica y quizá usando un modelo más simple.

En el día a día

Cuando usas un LLM, no eliges cuántas capas tiene. El proveedor ya lo decidió por ti. Pero entender la arquitectura te permite tomar mejores decisiones:

Elegir el tamaño de modelo adecuado. Un modelo de 7B parámetros tiene menos capas (o capas más pequeñas) que uno de 70B. Más capas significan más capacidad de abstracción, pero también más latencia y más coste. Si tu tarea es clasificar el sentimiento de reseñas de producto, un modelo pequeño probablemente baste. Si necesitas análisis jurídico sobre documentos de cien páginas, necesitas profundidad, contexto y evaluación específica.

Hacer *fine-tuning* con criterio. Cuando ajustas un modelo pre-entrenado, la pregunta no es solo «qué capas entreno». En visión clásica sí era habitual congelar capas tempranas y entrenar capas finales. En LLMs modernos muchas veces se usan adaptadores, LoRA, QLoRA o entrenamiento parcial de módulos concretos, porque tocar todos los pesos es caro y puede degradar capacidades previas. La idea de fondo es la misma: decidir qué parte de la arquitectura permites modificar y cómo comprobarás que no empeora lo que ya funcionaba.

Depurar problemas de entrenamiento. Si tu *fine-tuning* no converge, entender la arquitectura te ayuda a diagnosticar: ¿tienes demasiadas capas para los datos que tienes? ¿El gradiente se está desvaneciendo en las capas tempranas? ¿La función de activación de la última capa es la adecuada para tu tarea?

Por qué debería importarte

La arquitectura de una red neuronal no es un detalle de implementación. Es la decisión de diseño más importante después de elegir los datos.

La diferencia entre un MLP de 3 capas y un Transformer de 96 capas no es solo de escala: es cualitativa. El MLP puede clasificar dígitos escritos a mano. El Transformer puede mantener una conversación larga, escribir código y trabajar con documentos extensos dentro de su ventana de contexto. La arquitectura determina qué patrones puede aprender y qué coste tendrá usarlos.

Y sin embargo, ambos se construyen con transformaciones paramétricas diferenciables: matrices, sesgos, no linealidades, normalizaciones y conexiones entre bloques. Lo que cambia es la organización. Por eso este capítulo importa: es el puente entre entender una operación elemental y entender por qué una arquitectura grande es ingeniería de formas, coste y flujo de información.

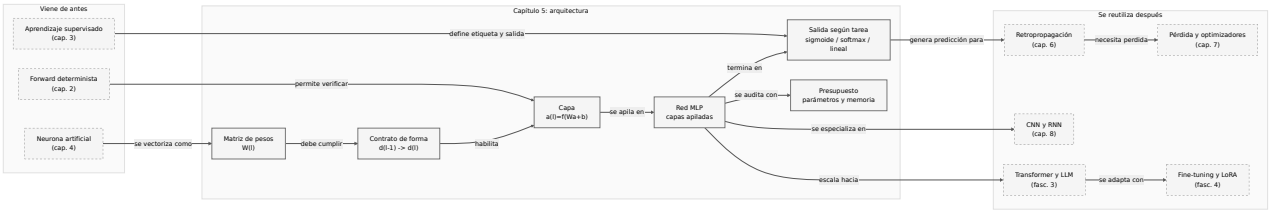
Dónde solía tropezar yo

ERROR	POR QUÉ ES UN ERROR	ANTÍDOTO
Añadir capas sin criterio	Más capas no siempre mejoran el rendimiento. Pueden causar *overfitting* si no tienes suficientes datos, o *underfitting* si el gradiente se desvanece.	Empieza con una arquitectura sencilla y añade complejidad solo cuando los datos y la tarea lo justifiquen. Mide el rendimiento en validación, no en entrenamiento.
Ignorar la función de activación de la capa de salida	Usar ReLU en la salida de un clasificador binario produce valores sin sentido. Usar softmax para regresión fuerza una distribución de probabilidad donde no aplica.	Elige la activación según la tarea: sigmoide para binario, softmax para multiclase, lineal para regresión, ninguna para valores sin restricción.
Creer que las capas profundas «entienden» conceptos humanos	Una capa profunda puede activarse fuertemente ante «gatos», pero no «sabe» qué es un gato. Reconoce un patrón estadístico, no un concepto.	No interpretes las activaciones como comprensión. Son correlaciones aprendidas. La interpretabilidad requiere técnicas específicas (facsímil 7).
Olvidar que el *forward pass* es determinista	Si obtienes resultados distintos con los mismos datos y pesos, el problema no está en la red. Está en el preprocesamiento, en el *batch* o en el muestreo posterior.	Aísla cada componente. El *forward pass* de una red con pesos fijos es determinista. Si hay variabilidad, búscala fuera.
No mirar las formas de los tensores	Muchos errores no son conceptuales: una capa entrega 128 valores y la siguiente espera 64, o la salida tiene 10 clases y la etiqueta viene como binaria.	Antes de entrenar, escribe la secuencia de dimensiones y cuenta parámetros. Si no puedes hacerlo en papel, tampoco lo depurarás bien en código.
Confundir más parámetros con más rigor	Una red grande puede parecer más seria, pero si el conjunto de datos es pequeño solo aumenta la capacidad de memorizar ruido.	Compara parámetros con ejemplos disponibles, mira validación y mantén una arquitectura base sencilla como control.

Cómo encaja todo

Este capítulo es el puente entre una neurona aislada y una red entrenable. Hereda del capítulo 4 la operación mínima $y = f(w^T x + b)$, pero la convierte en una arquitectura: matrices, capas, formas, activaciones y salida. Esa arquitectura no aprende todavía; solo calcula. Aprenderá en el capítulo 6, cuando la pérdida viaje hacia atrás y modifique pesos.

La idea que conviene guardar es esta: una red neuronal es una función compuesta con contratos de forma. Si respetas esos contratos, puedes apilar capas, medir coste y elegir salidas coherentes con la tarea. Si no los respetas, el modelo no falla por misterioso: falla porque la ingeniería básica no encaja.



Vocabulario aprendido

TÉRMINO	DEFINICIÓN
Red neuronal	Grafo dirigido de neuronas artificiales organizadas en capas que transforman datos secuencialmente.
Capa	Conjunto de neuronas que reciben las mismas entradas y producen salidas que alimentan la capa siguiente.
Capa oculta	Capa intermedia entre la entrada y la salida. Transforma los datos en representaciones progresivamente más abstractas.
MLP (*multilayer perceptron*)	Arquitectura donde cada neurona de una capa se conecta con todas las neuronas de la capa siguiente.
Deep learning	Entrenamiento de redes con muchas capas (decenas o cientos) que aprenden jerarquías de representaciones.
Forward pass	Recorrido de los datos desde la entrada hasta la salida. Es determinista: mismos pesos, misma salida.
Matriz de pesos	Conjunto de pesos de una capa escrito como matriz W. Su forma $d_l \times d_{l-1}$ define cuántas entradas recibe cada neurona y cuántas neuronas tiene la capa.
Ancho	Número de neuronas de una capa. Aumenta capacidad y coste.
Profundidad	Número de capas con parámetros. Permite composiciones más complejas, pero complica entrenamiento y depuración.
Contrato de forma	Requisito de que la salida de una capa tenga la dimensión que espera la siguiente. Es una validación básica antes de entrenar.
Presupuesto de parámetros	Conteo de pesos y sesgos de la arquitectura. Sirve para comparar capacidad, memoria aproximada y riesgo de sobreajuste.
Normalización (*LayerNorm*)	Recentrado y reescalado de las activaciones para mantenerlas en un rango estable. *LayerNorm* normaliza cada ejemplo por separado y es la variante que usan los Transformers.
Conexión residual	Atajo que suma la entrada a la salida de una capa ($x + f(x)$) para que el gradiente fluya hacia atrás sin desvanecerse.
Dropout	Apagado aleatorio de un porcentaje de neuronas en cada paso de entrenamiento para reducir el sobreajuste y forzar representaciones robustas.

Antes de pasar página

- [ ] ¿Puedo dibujar (mentalmente o en papel) la estructura de una red neuronal con capa de entrada, dos ocultas y una de salida? (Si no, vuelve al diagrama en «La arquitectura de una red».)
- [ ] ¿Puedo escribir la fórmula de una capa $a^{(l)} = f(W^{(l)}a^{(l-1)} + b^{(l)})$ y explicar la forma de W? (Si no, vuelve a «De la neurona a la capa».)
- [ ] ¿Entiendo qué aprende cada capa y por qué las capas tempranas son más genéricas? (Si no, vuelve a «Qué aprende cada capa».)
- [ ] ¿Sé qué significa «deep» en *deep learning* y qué problemas trae? (Si no, vuelve a «Qué significa deep».)
- [ ] ¿Puedo explicar el *forward pass*? (Si no, vuelve a «Cómo fluyen los datos».)
- [ ] ¿Sé defender una arquitectura calculando sus parámetros, formas y salida? (Si no, vuelve a «Presupuesto de una arquitectura».)

En resumen

IDEA FUERZA	DETALLE
Una red neuronal es una composición de capas.	Cada capa aplica una transformación $Wa + b$, una activación y entrega una nueva representación.
Las dimensiones son contrato, no decoración.	Si una capa produce 64 valores, la siguiente debe aceptar 64 entradas. Muchos errores reales empiezan ahí.
Los parámetros se pueden contar antes de entrenar.	$d_l \cdot d_{l-1} + d_l$ por capa. Ese conteo ayuda a razonar sobre memoria, datos necesarios y riesgo de sobreajuste.
Profundidad y ancho son decisiones de ingeniería.	Más capas o más neuronas pueden dar más capacidad, pero también más coste, más latencia y más superficie de fallo.
El capítulo 6 necesita este capítulo.	La retropropagación solo tiene sentido cuando sabes qué pesos, sesgos y capas debe actualizar.

Para saber más

Ba, J. L., Kiros, J. R. y Hinton, G. E. (2016). Layer normalization. arXiv:1607.06450. https://arxiv.org/abs/1607.06450

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

He, K., Zhang, X., Ren, S. y Sun, J. (2016). Deep residual learning for image recognition. En *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 770-778). https://doi.org/10.1109/CVPR.2016.90

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

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

Nielsen, M. (2015). *Neural networks and deep learning*. http://neuralnetworksanddeeplearning.com

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, L. y Polosukhin, I. (2017). Attention is all you need. En *Advances in Neural Information Processing Systems 30*. https://arxiv.org/abs/1706.03762

Notas

1. Goodfellow, I., Bengio, Y. y Courville, A. (2016). *Deep learning*. MIT Press. https://www.deeplearningbook.org. El capítulo 6 aborda en profundidad las arquitecturas de redes *feedforward*, desde el perceptrón simple hasta las redes profundas modernas. ↩

2. LeCun, Y., Bengio, Y. y Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436-444. https://doi.org/10.1038/nature14539. Los autores describen cómo las redes profundas aprenden representaciones jerárquicas: desde bordes y texturas en las primeras capas hasta partes de objetos y conceptos completos en las capas profundas. ↩

3. 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. AlexNet demostró que las características aprendidas por las capas tempranas de una CNN son transferibles entre tareas visuales, sentando las bases del *transfer learning* moderno. ↩

4. He, K., Zhang, X., Ren, S. y Sun, J. (2016). Deep residual learning for image recognition. En *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 770-778). https://doi.org/10.1109/CVPR.2016.90. Las conexiones residuales permiten que el gradiente fluya directamente a través de las capas, resolviendo el problema del desvanecimiento en redes de más de cien capas. ↩

5. Ba, J. L., Kiros, J. R. y Hinton, G. E. (2016). Layer normalization. arXiv:1607.06450. https://arxiv.org/abs/1607.06450. LayerNorm normaliza sobre las características de cada ejemplo y es la variante usada en los Transformers, donde el tamaño de lote y la longitud de secuencia varían. ↩

6. TensorFlow Playground. https://playground.tensorflow.org. Una herramienta educativa que visualiza en tiempo real el entrenamiento de redes neuronales sobre conjuntos de datos sintéticos. ↩