Facsímil 03

Arquitecturas y modelos

Capítulo 04

MLP, residual, LayerNorm, logits y sampling

686f6c61 · 2026

Capítulo 04: MLP, residual, LayerNorm, logits y sampling

Después de mirar, hay que pensar y decidir

En el capítulo anterior abrimos la atención por dentro: Q, K, V, máscara causal y softmax. Vimos cómo cada token mezcla información de posiciones permitidas. Pero un bloque Transformer no termina ahí.

Después de la atención queda trabajo: estabilizar números, dejar pasar información antigua, transformar cada posición con una red interna y, al final de muchas capas, convertir el último vector en una lista enorme de puntuaciones para tokens posibles.

Este capítulo va de ese tramo menos vistoso y muy importante. La atención decide **de dónde** tomar información. El MLP ayuda a transformar **qué significa** esa información. Las conexiones residuales permiten que las capas se apilen sin destruir lo anterior. LayerNorm mantiene los números en una zona manejable. Los logits y el sampling convierten todo eso en una palabra, un signo, un salto de línea o un fragmento de código.

Qué no es esta parte del Transformer

El MLP no es una memoria externa. No guarda documentos ni hechos como una base de datos. Es una transformación aprendida que se aplica a cada posición del tensor.

LayerNorm tampoco es una limpieza semántica del texto. No corrige ideas ni comprueba si algo es cierto. Normaliza números para que el entrenamiento y la inferencia sean más estables.

Y sampling no es una votación de verdad. Cuando el modelo elige el siguiente token, está usando una distribución de probabilidad lingüística. Que un token tenga mucha probabilidad significa que encaja con el contexto y con lo aprendido, no que la frase resultante sea necesariamente cierta.

Qué sí ocurre dentro de un bloque

Un bloque Transformer moderno suele seguir una idea parecida a esta:

```
entrada
  -> normalización
  -> atención
  -> suma residual
  -> normalización
  -> MLP
  -> suma residual
salida
```

El Transformer original ya combinaba atención, red feed-forward, normalización y conexiones residuales.[1] Muchas implementaciones actuales usan una variante llamada **pre-norm**: normalizan antes de la atención y antes del MLP.

Una forma compacta de escribirlo es:

$$a_l = x_l + \mathrm{Attention}(\mathrm{LN}(x_l))$$

$$x_{l+1} = a_l + \mathrm{MLP}(\mathrm{LN}(a_l))$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
x_l	Entrada del bloque l.	Tensor después del bloque anterior.
a_l	Estado intermedio después de atención y residual.	Lo que sale antes del MLP.
x_{l+1}	Salida del bloque.	Entrada del siguiente bloque.
LN	LayerNorm.	Normaliza cada vector de posición.
$\mathrm{Attention}$	Atención causal vista en el capítulo 03.	Q, K, V, máscara y softmax.
MLP	Red interna aplicada por posición.	Dos o tres proyecciones con una activación.

Cómo se lee esto sin saber matemáticas. La fórmula se lee de dentro hacia afuera, como una receta de tres pasos. Tomemos la primera línea, $a_l = x_l + \mathrm{Attention}(\mathrm{LN}(x_l))$:

1. $\mathrm{LN}(x_l)$: coge lo que llega al bloque, x_l, y ordena sus números a una escala estable. Es como nivelar el volumen antes de mezclar.
2. $\mathrm{Attention}(\ldots)$: sobre esa versión ordenada, deja que cada token mire a los demás y recoja contexto.

3. $x_l + \ldots$: suma ese resultado a lo que ya había, **sin borrarlo**. Ese total es a_l.

La segunda línea repite la misma jugada, pero con el MLP en lugar de la atención: normaliza a_l, lo pasa por la red interna y vuelve a sumar. El resultado, x_{l+1}, es lo que entra al siguiente bloque.

El símbolo que conviene no perder de vista es el $+$: el bloque **nunca reemplaza** lo que venía, siempre **añade** una corrección encima. Esa suma (la conexión residual) es una de las razones por las que se pueden apilar muchas capas sin que la información se pierda por el camino.

Pre-norm o post-norm: dónde va la normalización

Te habrás fijado en un detalle de las fórmulas: la LayerNorm está **dentro** del paréntesis, antes de la atención y antes del MLP. Eso es **pre-norm**, y no siempre fue así. El Transformer original normalizaba **después** de cada subcapa (*post-norm*): $x_{l+1} = \mathrm{LN}(x_l + \mathrm{Attention}(x_l))$. Funcionaba, pero al apilar muchas capas el entrenamiento se volvía inestable y exigía calentar el *learning rate* con cuidado. El pre-norm cambió eso: al normalizar la entrada de cada subcapa y dejar la vía residual **limpia** (sin normalizar), el gradiente fluye sin obstáculos desde la última capa hasta la primera. Por eso casi todos los LLMs profundos de hoy son pre-norm: es lo que hace viable entrenar decenas o cientos de capas sin que los números se descontrolen.

Un ejemplo entendible: autocompletar un mensaje

Imagina que escribes en una aplicación de correo:

> Nos vemos mañana a las

El modelo no piensa en castellano con una libreta al lado. Lo que tiene es una representación numérica del contexto y debe decidir el siguiente token. Aun así, podemos traducir las piezas del bloque a una historia bastante humana:

PIEZA	EN EL EJEMPLO	QUÉ APORTA
Atención	Mira «vemos», «mañana» y «a las».	Recupera que probablemente viene una hora.
Residual	Conserva la frase completa mientras añade señales nuevas.	No pierde que estamos hablando de una cita.
LayerNorm	Pone las señales numéricas en una escala comparable.	Evita que una dimensión grande domine solo por tamaño.
MLP	Transforma la representación de la última posición.	Refuerza patrones como «a las» → hora o número.
Logits	Asigna puntuaciones a tokens candidatos.	«10», «9», «reunión», «azul» reciben valores distintos.
Sampling	Elige el siguiente token según la distribución y la configuración.	Puede elegir «10» de forma conservadora o explorar alternativas.

Supongamos que, tras muchas capas, la cabeza de lenguaje produce estos logits:

TOKEN CANDIDATO	LOGIT	LECTURA INFORMAL
10	4,2	Muy compatible con «a las».
9	3,9	También muy compatible.
reunión	1,1	Tiene relación con el contexto, pero encaja peor justo después de «a las».
azul	-0,7	Gramatical y semánticamente flojo aquí.

Si usamos temperatura baja, el sistema tenderá a elegir entre «10» y «9». Si subimos la temperatura o abrimos mucho `top_p`, aumenta la posibilidad de tokens menos esperados. Esto no vuelve al modelo más profundo; cambia el margen de exploración.

Este ejemplo resume el capítulo entero: el bloque procesa la frase, los logits proponen continuaciones y el sampling decide cuánto riesgo aceptamos al elegir una.

Residual: dejar una vía abierta

Una conexión residual suma la entrada original con una transformación:

$$y = x + F(x)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
x	Representación que llega a una subcapa.	Vector de un token antes de la atención.
$F(x)$	Transformación aprendida.	Atención o MLP.
y	Resultado después de sumar entrada y transformación.	Vector actualizado.

La idea se popularizó con ResNet en visión por computador: una red profunda entrena mejor si cada bloque puede aprender una corrección sobre lo que ya venía circulando.[2]

En un LLM, la intuición práctica es esta:

SIN RESIDUAL	CON RESIDUAL
Cada subcapa tiene que reconstruir todo lo útil.	Cada subcapa puede añadir una corrección.
Es más fácil degradar información anterior.	La representación anterior sigue disponible.
Entrenar muchas capas se vuelve más frágil.	Apilar capas profundas es más viable.

Ejemplo pequeño:

$$x = [2,\ -1,\ 0{,}5]$$

$$F(x) = [0{,}1,\ 0{,}4,\ -0{,}2]$$

$$y = x + F(x) = [2{,}1,\ -0{,}6,\ 0{,}3]$$

La transformación ha modificado el vector, pero no lo ha sustituido por completo. Ese matiz importa mucho en redes profundas.

LayerNorm: poner los números en una escala manejable

LayerNorm normaliza cada vector de activaciones usando su propia media y desviación típica.[3] Para un vector x, calculamos:

$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i$$

$$\sigma = \frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2 + \epsilon$$

$$\mathrm{LN}(x)_i = \gamma_i \frac{x_i - \mu}{\sigma} + \beta_i$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
x_i	Componente i del vector.	$x_2 = -1$.
d	Dimensión del vector.	$d = 3$.
μ	Media del vector.	Media de $[2, -1, 0{,}5]$.
σ	Desviación típica con estabilidad numérica.	Incluye ϵ.
ϵ	Número pequeño para evitar división por cero.	10^{-5}.
γ_i	Escala aprendida.	Permite ajustar la salida normalizada.
β_i	Sesgo aprendido.	Permite desplazar la salida.

Si tomamos:

$$x = [2,\ -1,\ 0{,}5]$$

La media es:

$$\mu = \frac{2 - 1 + 0{,}5}{3} = 0{,}5$$

Las diferencias respecto a la media son:

$$[1{,}5,\ -1{,}5,\ 0]$$

La desviación típica, ignorando ϵ para leer el ejemplo, es:

$$\sigma = \frac{1{,}5^2 + (-1{,}5)^2 + 0^2}{3} \approx 1{,}225$$

Si $\gamma = [1, 1, 1]$ y $\beta = [0, 0, 0]$, la salida aproximada es:

$$\mathrm{LN}(x) \approx [1{,}225,\ -1{,}225,\ 0]$$

LayerNorm no cambia el orden de las ideas por sí sola. Cambia la escala de los números para que las siguientes operaciones trabajen en una zona más estable.

La versión moderna: RMSNorm

Muchos LLMs recientes (LLaMA, Mistral, Qwen y compañía) usan una variante más simple, **RMSNorm**.[4] La idea: la parte cara, y según se vio poco necesaria, de LayerNorm es restar la media. RMSNorm se la salta y normaliza solo por la magnitud del vector, su raíz cuadrática media:

$$\mathrm{RMSNorm}(x)_i = \gamma_i \frac{x_i}{\frac{1}{d}\sum_{j=1}^{d} x_j^2 + \epsilon}$$

No resta μ ni usa β: solo reescala. Hace casi lo mismo que LayerNorm, con menos operaciones y, por tanto, algo más rápido en entrenamiento e inferencia. Es un buen ejemplo de cómo los modelos modernos recortan piezas que costaban más de lo que aportaban.

MLP: transformar cada posición por dentro

Después de la atención, el bloque Transformer aplica una red feed-forward por posición. En muchos textos se llama MLP, aunque técnicamente es una red pequeña aplicada a cada token por separado.

Una versión clásica es:

$$\mathrm{MLP}(x) = W_2\phi(W_1x + b_1) + b_2$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
x	Vector de una posición.	Representación contextual de «banco».
W_1	Primera matriz aprendida.	Expande la dimensión interna.
b_1	Sesgo de la primera proyección.	Vector sumado antes de la activación.
ϕ	Función de activación.	GELU, ReLU o una variante con puerta.
W_2	Segunda matriz aprendida.	Devuelve la dimensión al tamaño del modelo.
b_2	Sesgo final.	Ajuste aprendido de salida.

Las redes profundas usan activaciones no lineales para poder representar funciones más ricas que una sola multiplicación de matrices.[5] En Transformers modernos, GELU se hizo muy común en modelos de lenguaje.[6] También hay variantes con puertas, como GLU y SwiGLU, que permiten modular qué información pasa.[7]

Dónde viven los parámetros: el MLP se ensancha

Hay un detalle de tamaño que sorprende: la matriz W_1 no mantiene la dimensión, la **expande**, normalmente a $4 \times d_{\text{model}}$, y W_2 la devuelve. Es decir, dentro de cada token el MLP trabaja en un espacio cuatro veces más ancho antes de volver. Esa expansión es cara: en un Transformer denso, **alrededor de dos tercios de los parámetros viven en los MLP**, no en la atención. Por eso, cuando en el capítulo 1 estimamos el tamaño de un modelo con $N \approx 12 \cdot L \cdot d^2$, buena parte de ese 12 es el MLP.

Por qué GELU, y no solo ReLU

La activación ϕ es la pieza que da la no-linealidad. La clásica es ReLU (deja pasar lo positivo y corta lo negativo a cero), pero los LLMs adoptaron **GELU**, una versión suave que no corta en seco sino que atenúa gradualmente los valores cercanos a cero; esa suavidad ayuda al gradiente. Y los modelos más recientes usan **SwiGLU**, una activación *con puerta*: en vez de aplicar una sola función, multiplica dos proyecciones, una de las cuales actúa como compuerta que decide cuánta señal deja pasar. A cambio de algo más de cómputo, suele mejorar la calidad, y por eso es la opción por defecto en muchos LLMs abiertos modernos.

La intuición:

PIEZA	QUÉ HACE
Atención	Mezcla información entre posiciones.
MLP	Transforma cada posición internamente.
Residual	Conserva una vía para lo que ya estaba.
LayerNorm	Mantiene los números en una escala razonable.

Si la atención es una mesa de mezclas entre tokens, el MLP es el taller interno de cada token. No decide a qué otros tokens mirar; trabaja con lo que ya recibió.

Del último vector a los logits

Tras repetir muchos bloques, el modelo tiene una representación final. Para predecir el siguiente token, toma el vector de la última posición y lo proyecta al tamaño del vocabulario:

$$z = hW_{\text{vocab}} + b$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
h	Vector final de la última posición.	Representación de todo el contexto disponible.
W_{vocab}	Matriz que lleva de dimensión interna a vocabulario.	De d_{model} a 50 000 tokens.
b	Sesgo de vocabulario.	Una puntuación adicional por token.
z	Vector de logits.	Una puntuación por token candidato.

Un logit no es una probabilidad. Puede ser negativo, positivo, grande o pequeño. Para convertir logits en una distribución usamos softmax, igual que vimos en el capítulo 01 y en el capítulo 03:

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{V} e^{z_j}}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
p_i	Probabilidad asignada al token i.	$p_{\text{Madrid}} = 0{,}256$.
z_i	Logit del token i.	$z_{\text{París}} = 4{,}0$.
V	Tamaño del vocabulario.	50 000 tokens candidatos.
$\sum_{j=1}^{V}$	Suma sobre todos los tokens.	Normaliza para que las probabilidades sumen 1.

Bishop presenta softmax como una transformación estándar de puntuaciones en probabilidades normalizadas.[8] En LLMs, esa transformación no juzga la verdad de una frase: convierte puntuaciones de continuación en una distribución de tokens.

Temperatura: abrir o cerrar el abanico

La temperatura modifica los logits antes de softmax:

$$p_i(T) = \frac{e^{z_i/T}}{\sum_{j=1}^{V} e^{z_j/T}}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
T	Temperatura.	$T = 0{,}5$, $T = 1$, $T = 2$.
z_i/T	Logit ajustado por temperatura.	Si $T < 1$, las diferencias crecen.
$p_i(T)$	Probabilidad después de aplicar temperatura.	Distribución más concentrada o más suave.

Supongamos cuatro tokens candidatos:

TOKEN	LOGIT
París	4,0
Madrid	3,0
Lyon	1,0
azul	0,0

Con $T = 1$:

TOKEN	PROBABILIDAD APROX.
París	0,696
Madrid	0,256
Lyon	0,035
azul	0,013

Con $T = 0{,}5$, las diferencias se agrandan:

TOKEN	PROBABILIDAD APROX.
París	0,879
Madrid	0,119
Lyon	0,002
azul	0,000

Con $T = 2$, las diferencias se suavizan:

TOKEN	PROBABILIDAD APROX.
París	0,509
Madrid	0,309
Lyon	0,114
azul	0,069

Temperatura baja no significa «más inteligente». Significa más conservadora en la elección del siguiente token. Temperatura alta no significa «mejor creatividad». Significa más dispersión en la distribución. En producto, esto se decide según la tarea: extracción estructurada, redacción, lluvia de ideas, código, resumen o conversación.

Top-k y top-p: limitar candidatos

Además de temperatura, se suelen aplicar reglas para limitar el conjunto de tokens candidatos.

Top-k conserva los k tokens con mayor probabilidad y pone el resto a cero:

$$C_k = \mathrm{TopK}(p, k)$$

$$p'_i = \begin{cases} \frac{p_i}{\sum_{j \in C_k} p_j} & \text{si } i \in C_k \\ 0 & \text{si } i \notin C_k \end{cases}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
C_k	Conjunto de tokens conservados.	Con $k = 2$: París y Madrid.
p'_i	Probabilidad renormalizada.	Probabilidad después de eliminar candidatos.
$i \notin C_k$	Token descartado por top-k.	Lyon y azul reciben 0.

Top-p, también llamado nucleus sampling, conserva el conjunto mínimo de tokens cuya probabilidad acumulada alcanza un umbral p. Holtzman y coautores popularizaron esta estrategia para reducir degeneraciones en generación abierta.[9]

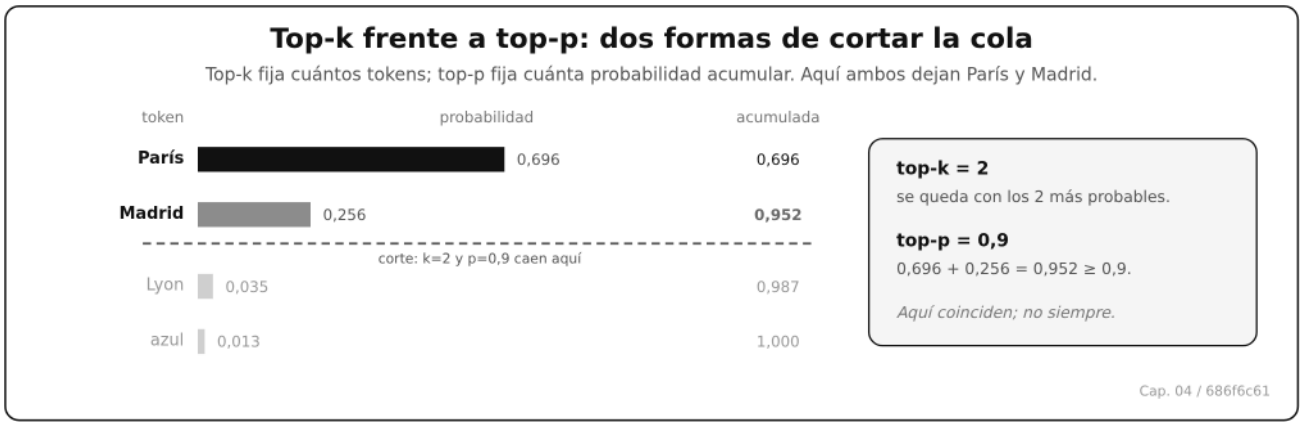
Si ordenamos de mayor a menor:

TOKEN	PROBABILIDAD	ACUMULADA
París	0,696	0,696
Madrid	0,256	0,952
Lyon	0,035	0,987
azul	0,013	1,000

Con $p = 0{,}9$, top-p conserva París y Madrid, porque con esos dos ya alcanza 0,952. Después renormaliza solo dentro de ese conjunto.

La diferencia importante:

REGLA	QUÉ FIJA	QUÉ CAMBIA
Top-k	Número de candidatos.	La masa de probabilidad conservada.
Top-p	Masa de probabilidad mínima.	El número de candidatos.



En una distribución muy concentrada, top-p puede conservar pocos tokens. En una distribución muy plana, puede conservar muchos. Esa elasticidad es la gracia.

El orden de los mandos, y qué viene después

En la práctica estos controles se aplican en cadena, y el orden importa: primero la **temperatura** reescala los logits, después **top-k** o **top-p** recortan la cola, y al final se muestrea de lo que queda. Calentar y luego recortar no da el mismo resultado que recortar y luego calentar; por eso conviene saber qué hace tu librería por debajo.

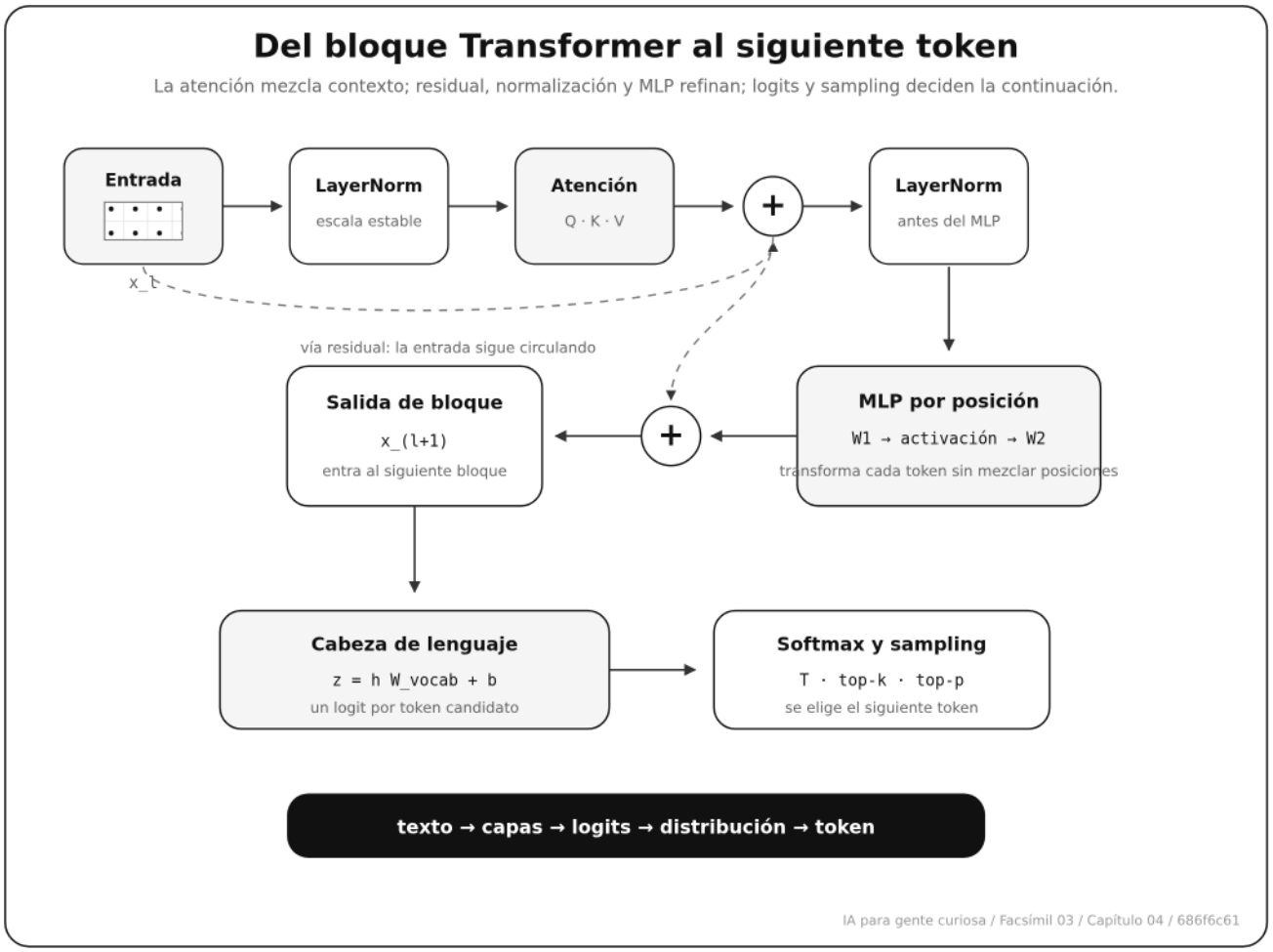
Y top-k y top-p no son el final de la historia. Estrategias más recientes como **min-p** ajustan el corte a la *forma* de la distribución: en vez de un número fijo o una masa fija, descartan los tokens cuya probabilidad cae por debajo de una fracción de la del token más probable. La idea de fondo es siempre la misma: decidir cuánta cola de la distribución dejamos viva antes de tirar el dado.

Modo ingeniero: LayerNorm y los mandos en código

Todas estas piezas son cortas en `numpy`. Aquí están LayerNorm, el softmax con temperatura y el recorte top-p:

```
import numpy as np

def layernorm(x, gamma, beta, eps=1e-5):
    mu = x.mean()
    sigma = np.sqrt(((x - mu) ** 2).mean() + eps)
    return gamma * (x - mu) / sigma + beta

def softmax_temp(logits, T=1.0):
    z = logits / T
    z = z - z.max()                          # estabilidad
    e = np.exp(z)
    return e / e.sum()

def top_p(probs, p=0.9):
    orden = np.argsort(probs)[::-1]          # de mayor a menor
    acum = np.cumsum(probs[orden])
    corte = np.searchsorted(acum, p) + 1     # cuántos tokens hasta alcanzar p
    keep = orden[:corte]
    out = np.zeros_like(probs)
    out[keep] = probs[keep]
    return out / out.sum()                   # renormaliza dentro del núcleo

logits = np.array([4.0, 3.0, 1.0, 0.0])      # París, Madrid, Lyon, azul
p = softmax_temp(logits, T=1.0)              # [0.696, 0.256, 0.035, 0.013]
print(top_p(p, 0.9).round(3))                # [0.731 0.269 0. 0.]
```

`layernorm` con `x=[2, -1, 0.5]` devuelve `[1.225, -1.225, 0.0]`, justo el ejemplo de antes; y `top_p` con `p=0.9` deja a París y Madrid y renormaliza entre ellos. Es exactamente lo que aísla el lab de este capítulo: cada mando por separado, para ver qué cambia.



Simuladores para verlo en pantalla

Dos recursos ayudan especialmente aquí:

RECURSO	QUÉ MIRAR
LLM Visualization	Recorre las capas y localiza el paso desde bloques internos a logits.[10]
Transformer Explainer	Cambia texto de entrada y observa cómo varía la distribución del siguiente token.[11]

Úsalos con una pregunta concreta: ¿dónde se convierte el vector final en logits?, ¿dónde aparece softmax?, ¿qué cambia si modifico el texto anterior?

En el día a día

Cuando quieres salida estable. Para extracción JSON, clasificación o tareas con formato rígido, normalmente quieres menos variación: temperatura baja, pocos candidatos y validación posterior. No porque el modelo «sepa más», sino porque reduces el abanico de continuaciones posibles.

Cuando quieres explorar. Para escritura creativa, alternativas de copy o lluvia de ideas, puedes permitir más variedad. Ahí temperatura y top-p se convierten en mandos de exploración. Conviene medir resultados, no elegir parámetros por superstición.

Cuando depuras una respuesta extraña. Si una salida cambia mucho entre ejecuciones, no siempre es un problema de prompt. Puede haber temperatura alta, top-p amplio o muestreo activado. Antes de cambiar de modelo, mira la configuración de generación.

Cuando eliges arquitectura. MLP, normalización y residuales parecen detalles internos, pero afectan a estabilidad, velocidad, memoria y calidad. En el capítulo 05 veremos cómo modelos modernos cambian estas piezas: MoE, activaciones con puerta y variaciones de arquitectura.

Por qué debería importarte

Porque esta parte une dos mundos: arquitectura interna y comportamiento visible. El lector ve texto, pero por debajo hay logits, temperatura, filtros de candidatos y muestreo. Entender esa cadena evita explicaciones vagas.

Si un modelo responde siempre igual, puede ser por una distribución muy concentrada o por una configuración conservadora. Si responde con demasiada dispersión, quizá la temperatura o top-p están abriendo demasiado el abanico. Si un modelo grande cuesta mucho, parte del coste está en repetir bloques con atención, MLP y proyecciones a vocabulario miles de veces por segundo.

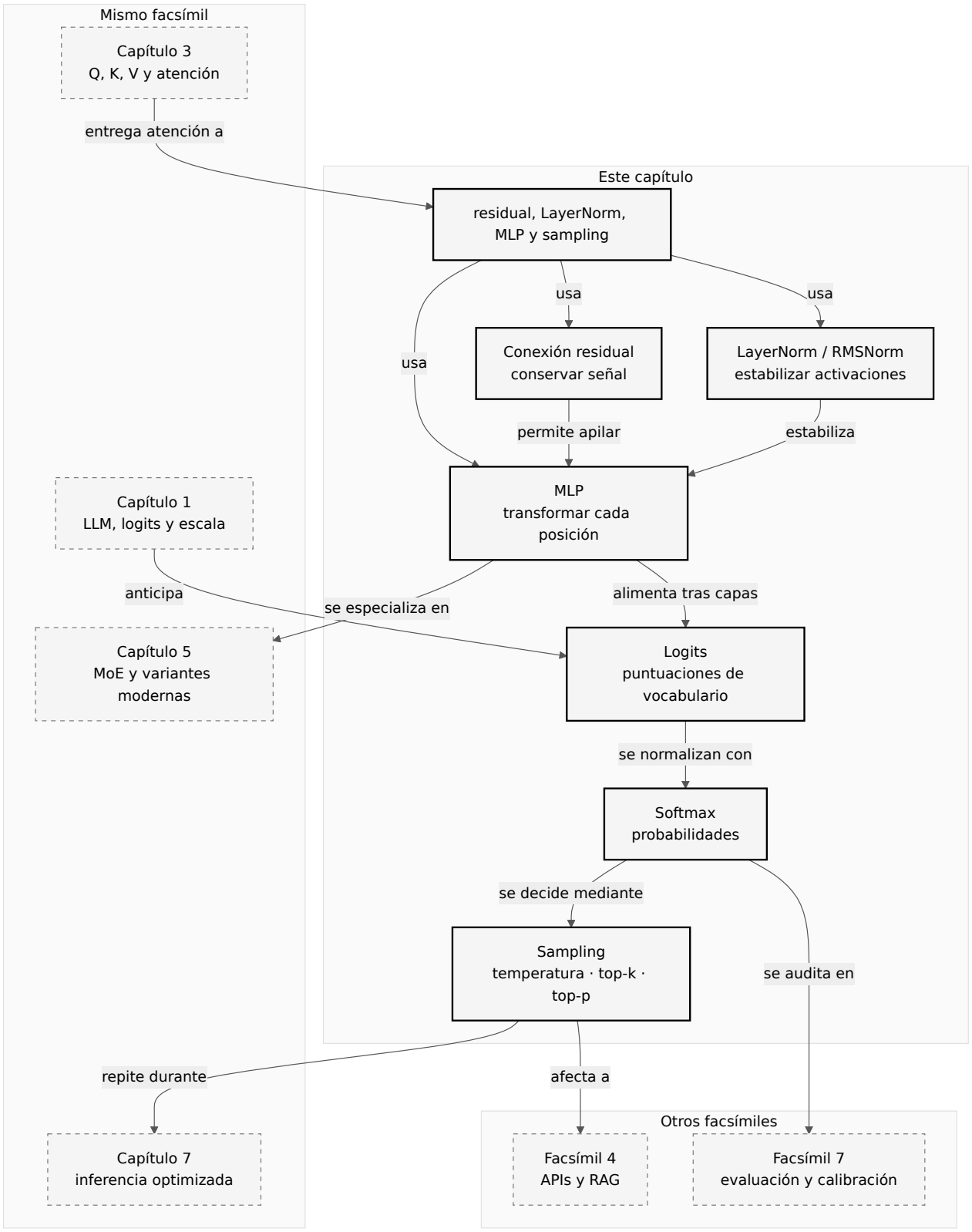
Dónde solía tropezar yo

ERROR	POR QUÉ ES UN ERROR	ANTÍDOTO
Pensar que LayerNorm entiende el texto	Normaliza números; no valida significado.	Pregunta siempre si una pieza trabaja con escala numérica o con decisión de contenido.
Confundir logit con probabilidad	Un logit es una puntuación cruda; puede ser negativa y no suma nada.	No hables de probabilidad hasta aplicar softmax.
Creer que temperatura baja equivale a verdad	Solo concentra la distribución en tokens más probables.	Para verdad factual, usa contexto, fuentes y validación.
Olvidar que el MLP no mezcla posiciones	La mezcla entre tokens la hizo la atención; el MLP transforma cada posición.	Repite: atención mezcla entre posiciones, MLP transforma dentro de una posición.
Usar top-k y top-p como si fueran lo mismo	Top-k fija número de candidatos; top-p fija masa acumulada.	Mira una tabla ordenada de probabilidades y calcula ambos a mano una vez.

Cómo encaja todo

Este mapa enseña el tramo que suele quedar invisible entre “atención” y “respuesta”. La atención entrega una representación contextual; residual, LayerNorm y MLP la estabilizan y transforman; logits, softmax y sampling la convierten en comportamiento visible.

Por eso el capítulo conecta arquitectura con producto: temperatura, `top_k` y `top_p` no son adornos de API, sino controles sobre una distribución que nace de todo el bloque Transformer.


Mismo facsímil
Capítulo 3
Q, K, V y atención
entrega atención a
Este capítulo
residual, LayerNorm,
MLP y sampling
usa
usa
usa
Conexión residual
conservar señal
LayerNorm / RMSNorm
estabilizar activaciones
permite apilar
estabiliza
MLP
transformar cada
posición
Capítulo 1
LLM, logits y escala
anticipa
se especializa en
alimenta tras capas
Capítulo 5
MoE y variantes
modernas
Logits
puntuaciones de
vocabulario
se normalizan con
Softmax
probabilidades
se decide mediante
Sampling
temperatura · top-k ·
top-p
se audita en
repite durante
afecta a
Otros facsímiles
Capítulo 7
inferencia optimizada
Facsímil 4
APIs y RAG
Facsímil 7
evaluación y calibración

Vocabulario aprendido

TÉRMINO	DEFINICIÓN
Conexión residual	Suma que conserva la entrada y añade una transformación aprendida.
LayerNorm	Normalización que usa media y desviación típica dentro de cada vector.
RMSNorm	Normalización que reescala por la raíz cuadrática media del vector, sin restar la media.
Pre-norm	Aplicar la normalización antes de cada subcapa, dejando limpia la vía residual.
MLP	Red interna aplicada por posición después de la atención.
Activación	Función no lineal que permite representar transformaciones más ricas.
SwiGLU	Activación con puerta usada en los MLP modernos; multiplica dos proyecciones para modular la señal.
Logit	Puntuación cruda de un token antes de softmax.
Temperatura	Parámetro que concentra o suaviza la distribución de salida.
Top-k	Muestreo que conserva solo los k candidatos principales.
Top-p	Muestreo que conserva candidatos hasta alcanzar una probabilidad acumulada.
Sampling	Proceso de elegir el siguiente token a partir de una distribución.

Antes de pasar página

- [ ] ¿Puedo explicar por qué una conexión residual suma en vez de reemplazar? (Si no, vuelve a «Residual: dejar una vía abierta».)
- [ ] ¿Sé calcular la media y desviación típica de una LayerNorm pequeña? (Si no, vuelve a «LayerNorm».)
- [ ] ¿Distingo pre-norm de post-norm y sé qué cambia RMSNorm frente a LayerNorm? (Si no, vuelve a «Pre-norm o post-norm» y «La versión moderna: RMSNorm».)
- [ ] ¿Entiendo por qué el MLP transforma cada posición pero no mezcla tokens? (Si no, vuelve a «MLP».)

- [ ] ¿Sé por qué el MLP se ensancha a $4d$ y por qué GELU o SwiGLU en vez de solo ReLU? (Si no, vuelve a «Dónde viven los parámetros» y «Por qué GELU».)
- [ ] ¿Puedo distinguir logit, probabilidad y token elegido? (Si no, vuelve a «Del último vector a los logits».)
- [ ] ¿Puedo explicar el ejemplo de «Nos vemos mañana a las» usando residual, LayerNorm, MLP, logits y sampling? (Si no, vuelve a «Un ejemplo entendible».)
- [ ] ¿Sé qué cambia cuando bajo o subo la temperatura? (Si no, vuelve a «Temperatura».)
- [ ] ¿Puedo calcular top-k y top-p con una tabla de cuatro tokens? (Si no, vuelve a «Top-k y top-p».)
- [ ] ¿Sé en qué orden se aplican temperatura, top-k y top-p, y qué hace min-p? (Si no, vuelve a «El orden de los mandos».)
- [ ] ¿Sé qué parámetro de muestreo tocar y cómo cambia la salida? (Si no, vuelve a «Temperatura» y «Top-k y top-p».)
- [ ] ¿Puedo conectar estos mandos con APIs, agentes y evaluación futura? (Si no, vuelve a «Cómo encaja todo».)

En resumen

IDEA FUERZA	DETALLE
Un bloque Transformer no es solo atención.	Residual, LayerNorm y MLP hacen que la información circule, se estabilice y se transforme.
Los logits son puntuaciones, no probabilidades.	Softmax convierte esas puntuaciones en una distribución sobre el vocabulario.
Sampling es una decisión de comportamiento.	Temperatura, top-k y top-p cambian cuánta variación permitimos al generar.
La arquitectura interna afecta al producto visible.	Estabilidad, coste, latencia y estilo de respuesta dependen de estas piezas.

Para saber más

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

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

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

Bycroft, B. (2023). *LLM Visualization*. https://bbycroft.net/llm

Cho, A., Kim, G. C., Karpekov, A., Helbling, A., Wang, Z. J., Lee, S., Hoover, B. y Chau, D. H. (2025). Transformer Explainer: interactive learning of text-generative models. En *Proceedings of the AAAI Conference on Artificial Intelligence*. https://ojs.aaai.org/index.php/AAAI/article/download/35347/37502

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

Hendrycks, D. y Gimpel, K. (2016). *Gaussian Error Linear Units (GELUs)*. https://arxiv.org/abs/1606.08415

Holtzman, A., Buys, J., Du, L., Forbes, M. y Choi, Y. (2020). The curious case of neural text degeneration. En *International Conference on Learning Representations*. https://openreview.net/forum?id=rygGQyrFvH

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. y Sutskever, I. (2019). *Language models are unsupervised multitask learners*. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

Shazeer, N. (2020). *GLU variants improve Transformer*. https://arxiv.org/abs/2002.05202

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

Notas

1. 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. El artículo introduce capas de atención multi-cabeza, redes feed-forward por posición, conexiones residuales y normalización. ↩

2. 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 ayudaron a entrenar redes mucho más profundas al permitir que la información y el gradiente circularan con menos degradación. ↩

3. Ba, J. L., Kiros, J. R. y Hinton, G. E. (2016). *Layer normalization*. https://arxiv.org/abs/1607.06450. LayerNorm normaliza las activaciones de una capa para cada ejemplo, lo que resulta útil en arquitecturas de secuencia y Transformer. ↩

4. Zhang, B. y Sennrich, R. (2019). Root Mean Square Layer Normalization. *Advances in Neural Information Processing Systems 32*. https://arxiv.org/abs/1910.07467. RMSNorm normaliza por la raíz cuadrática media del vector, sin restar la media, y resultó casi igual de eficaz que LayerNorm con menos cómputo. ↩

5. Goodfellow, I., Bengio, Y. y Courville, A. (2016). *Deep learning*. MIT Press. https://www.deeplearningbook.org. El libro explica cómo las capas lineales y activaciones no lineales permiten componer funciones complejas. ↩

6. Hendrycks, D. y Gimpel, K. (2016). *Gaussian Error Linear Units (GELUs)*. https://arxiv.org/abs/1606.08415. GELU propone una activación suave que se adoptó en varias arquitecturas de lenguaje. ↩

7. Shazeer, N. (2020). *GLU variants improve Transformer*. https://arxiv.org/abs/2002.05202. El trabajo estudia variantes con puertas para mejorar el bloque feed-forward de Transformers. ↩

8. Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer. Softmax aparece como una forma de convertir puntuaciones reales en probabilidades multinomiales. ↩

9. Holtzman, A., Buys, J., Du, L., Forbes, M. y Choi, Y. (2020). The curious case of neural text degeneration. En *International Conference on Learning Representations*. https://openreview.net/forum?id=rygGQyrFvH. El artículo analiza problemas de decodificación y propone nucleus sampling como alternativa a estrategias demasiado rígidas. ↩

10. Bycroft, B. (2023). *LLM Visualization*. https://bbycroft.net/llm. Visualización interactiva 3D de un Transformer estilo GPT. ↩

11. Cho, A., Kim, G. C., Karpekov, A., Helbling, A., Wang, Z. J., Lee, S., Hoover, B. y Chau, D. H. (2025). Transformer Explainer: interactive learning of text-generative models. En *Proceedings of the AAAI Conference on Artificial Intelligence*. https://ojs.aaai.org/index.php/AAAI/article/download/35347/37502. La herramienta permite visualizar componentes de un GPT-2 pequeño en navegador. ↩

Cuadernos para practicar

Escanea el código con el móvil para abrir el cuaderno en Google Colab y ejecutarlo sin instalar nada.

Cómo elige palabras un LLM: temperatura, top-k y top-p

https://colab.research.google.com/github/686f6c61/iaparagentecuriosa-notebooks/blob/main/Facs%C3%ADmil%2003/02-sampling-temperatura-topk-topp.ipynb