Redes Neuronales para el **análisis** y la **generación** de texto

Grupo PLN
InCo- Fing- UdelaR

Arquitecturas Secuenciales

Motivación:

Juan no vio la película que me gustó

Juan vio la película que no me gustó

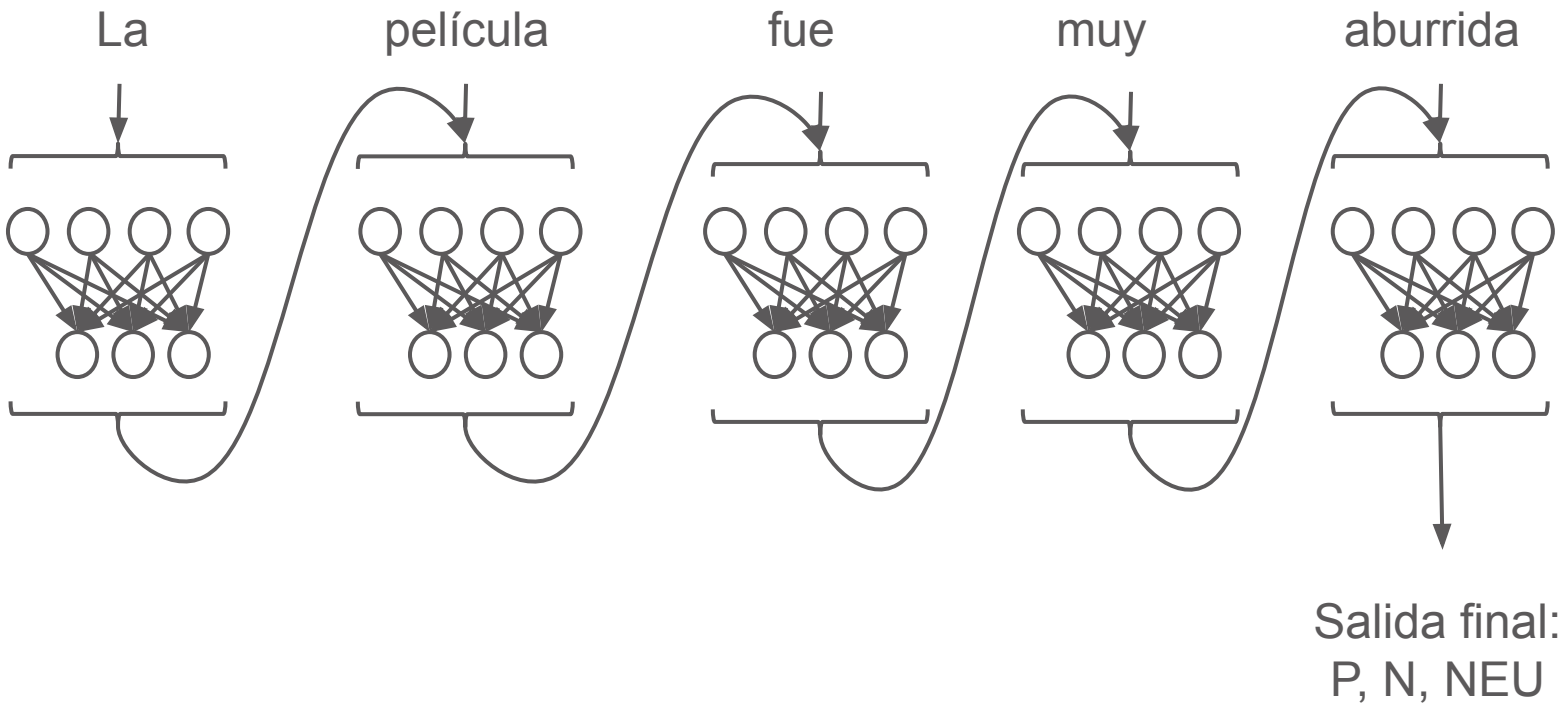
¿Cómo los representamos en BOW?

¿Y con embeddings?

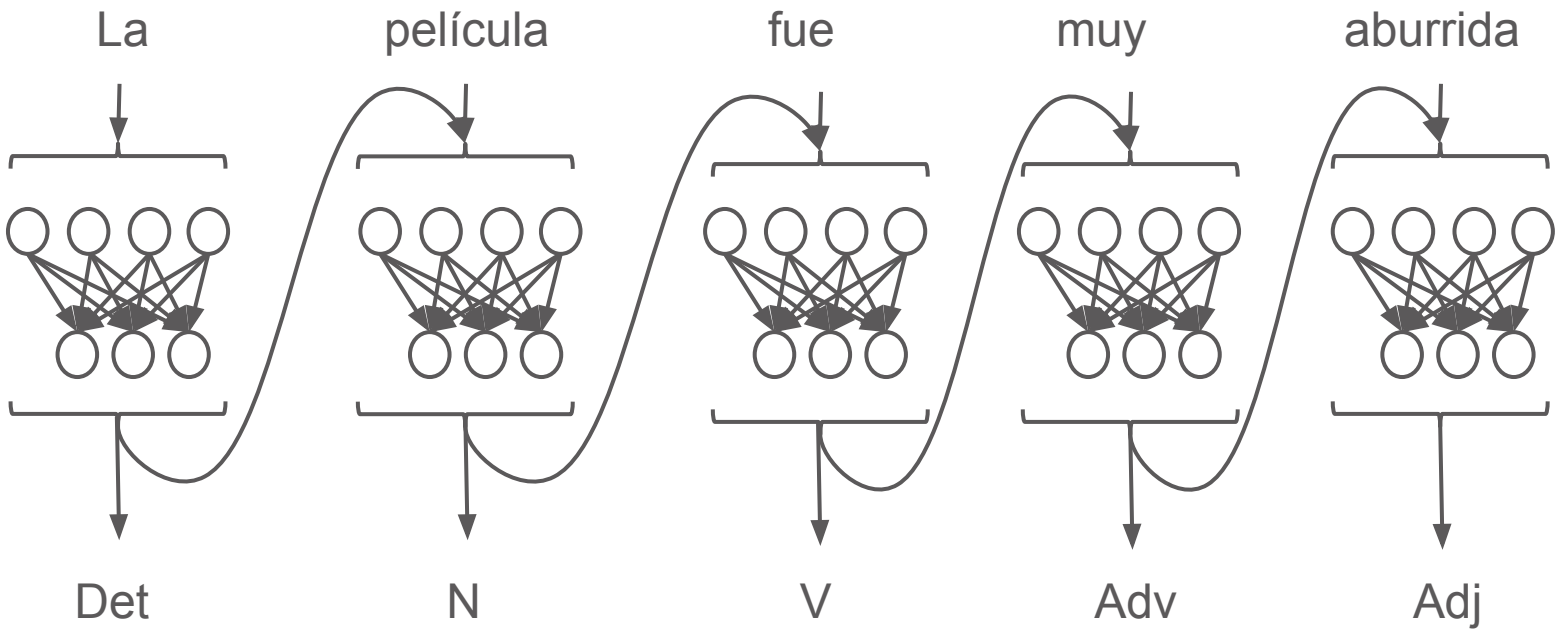
Arquitecturas Secuenciales

- El lenguaje está compuesto por palabras
- Secuencias de largo variable
- El orden de las palabras es muy relevante
- Pero los MLP tenían entrada de tamaño fijo…
- Sabemos representar una palabra (embeddings)… ¿y si pudiéramos ir presentándolas una a una en la red?

Red Recurrente



Red Recurrente - Salidas por palabra



Redes Recurrentes

- Recurrent Neural Network (RNN)
- Redes que contienen algún ciclo en sus conexiones
- El valor de alguna de las unidades está influido por valores anteriores de la entrada

Red Recurrente Simple

- La capa oculta con conexión recurrente actúa como una especie de memoria
- Teóricamente podríamos presentar una secuencia de cualquier largo y podría recordarla toda
- Información del inicio de la oración puede tener influencia al final
  - Veremos que en la práctica esto es muy difícil

Entrenamiento

- Para cada ejemplo de entrenamiento, se aplica palabra a palabra desarrollando la red
- Cada una de las salidas tiene su propio *loss*
- La pérdida se va acumulando
- Backpropagation “en el tiempo”
- Los valores de h_{ti} influyen en todos los t_j, $j > i$

Entrenamiento

- En este caso, solo me interesa la salida final
- Todavía voy mostrando la entrada palabra a palabra y desarrollando la red
- La pérdida se calcula solo al final
- Qué tanto influyen las primeras palabras de la secuencia en el cálculo final?
  - Desvanecimiento de gradiente *(vanishing gradient)*
  - Ocurre por la naturaleza multiplicativa de las derivadas

LSTM: Redes Neuronales Recurrentes mejoradas

- *Long Short-Term Memory* (Hochreiter y Schmidhuber, 1997)
- Intentan solucionar el problema de que la información más lejana es más difícil de acceder (recordar)
- Utiliza dos "estados ocultos"
  - uno de ellos mantiene la información "vieja"
  - el otro es el que se usa para calcular las salidas
- En cada paso, los dos estados y la nueva palabra de entrada se recombinan

Modelos de Lenguaje

Mira cómo los niños,

en un aire y tiempo de otro tiempo, **ríen**.

Cómo en su **inocencia**,

la Tierra es **inocente**

y es **inocente** el hombre.

Míralos cómo al *descubrir la muerte*

mueren, y ya definitivamente

ya sus ojos y dientes

comienzan a **crecer** junto a las horas

(Líber Falco)

Modelos de Lenguaje

Debido a las copiosas

Modelos de Lenguaje

Debido a las copiosas P=0.8 *lluvias* *de* *las* *últimas* *horas …*

P=0.09 *nevadas* *y avalanchas …*

P=0.0000001 *árbol*

- Predecir la probabilidad de una secuencia
- Predecir la siguiente palabra dado un prefijo

P(lluvias | debido a las copiosas)

$$P(w_{1:k}) = \prod_{i=1}^{k} P(w_i|w_{<i})$$

P(<s>debido a las copiosas lluvias</s>) = P(<s>) P(debido|<s>)
P(a | <s> debido) P(las | <s> debido a) … P(</s> | <s>debido a las copiosas lluvias)

Modelos de Lenguaje

P(<s>debido a las copiosas lluvias</s>) = P(<s>) P(debido|<s>)
P(a | <s> debido) P(las | <s> debido a) … P(</s> | <s>debido a las copiosas lluvias)

Históricamente se resuelve mediante conteo de N-gramas

$$P(w_i|w_{<i}) \approx P(w_i|w_{i-N+1:i-1})$$
$$= P(w_i|w_{i-N+1}w_{i-N+2}...w_{i-1})$$

el valor *N* de N-grama

El ejemplo anterior con tri-gramas:

P(<s>debido a las copiosas lluvias</s>) = P(<s>) P(debido|<s>)
P(a | <s> debido) P(las | debido a) P(copiosas | a las)
P(lluvias | las copiosas) P(</s> | copiosas lluvias)

Problemas?

Modelos de Lenguaje

- Hipótesis distribucional: algunos aspectos del significado podemos aprenderlos solamente leyendo.

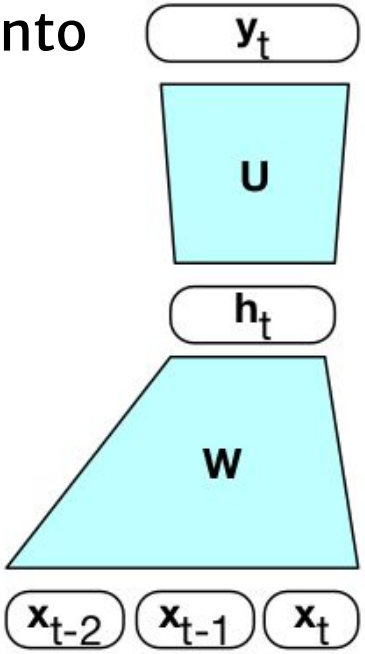
- Cuando leemos, vemos las palabras en conjunto, no de a una

- Los LLMs han mostrado que se puede aprender *mucho* simplemente intentando predecir la próxima palabra

Redes Neuronales como Modelo de Lenguaje

Podemos sustituir los N-gramas por una red feedforward

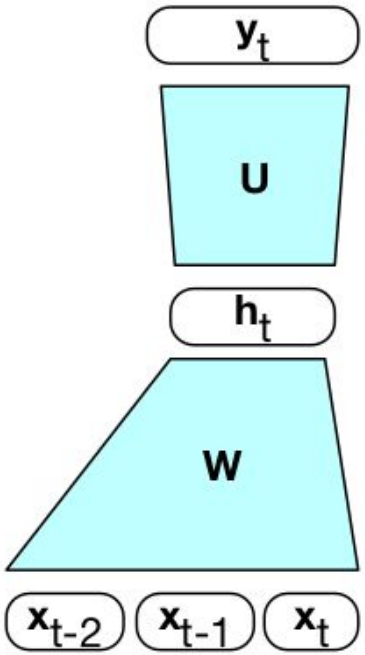
Mejora la generalización a ejemplos de N-gramas no vistos durante entrenamiento

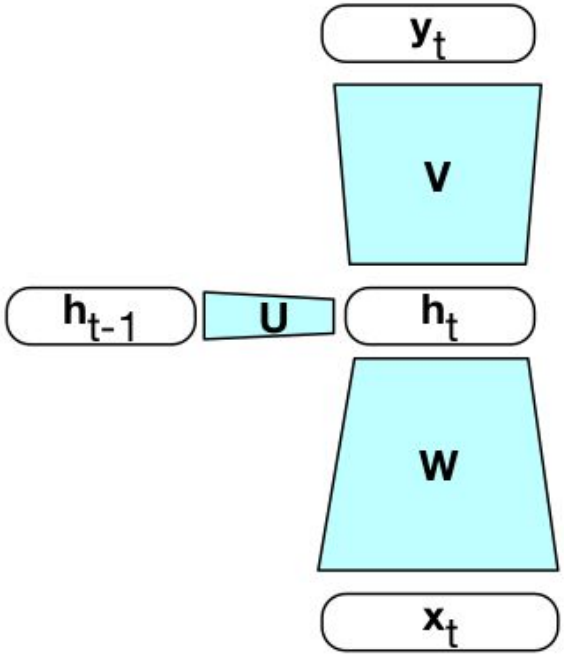


Pero sigue el problema de que las palabras del comienzo pierden relevancia

Redes Neuronales como Modelo de Lenguaje

Alternativa: Usar una RNN, de esta forma todo el contexto anterior se puede ir manteniendo en el estado oculto





Recurrent Neural Network based Language Model
Mikolov et al. (2010)

RNN como Modelo de Lenguaje

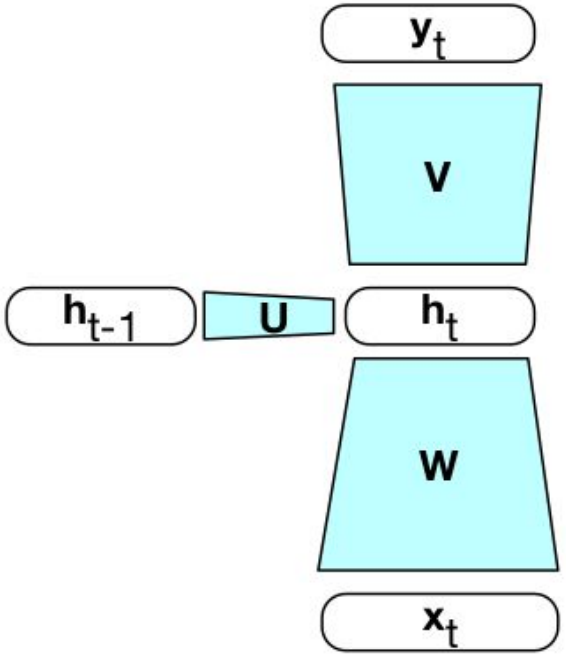
Tomamos la tabla de embeddings E

Matrices de pesos U, V y W

$$e_t = Ex_t$$
$$h_t = g(Uh_{t-1} + We_t)$$
$$y_t = \text{softmax}(Vh_t)$$

y_t es un vector del tamaño del vocabulario

La salida en el tiempo *t* es la distribución de probabilidad de la posible siguiente palabra *t+1*



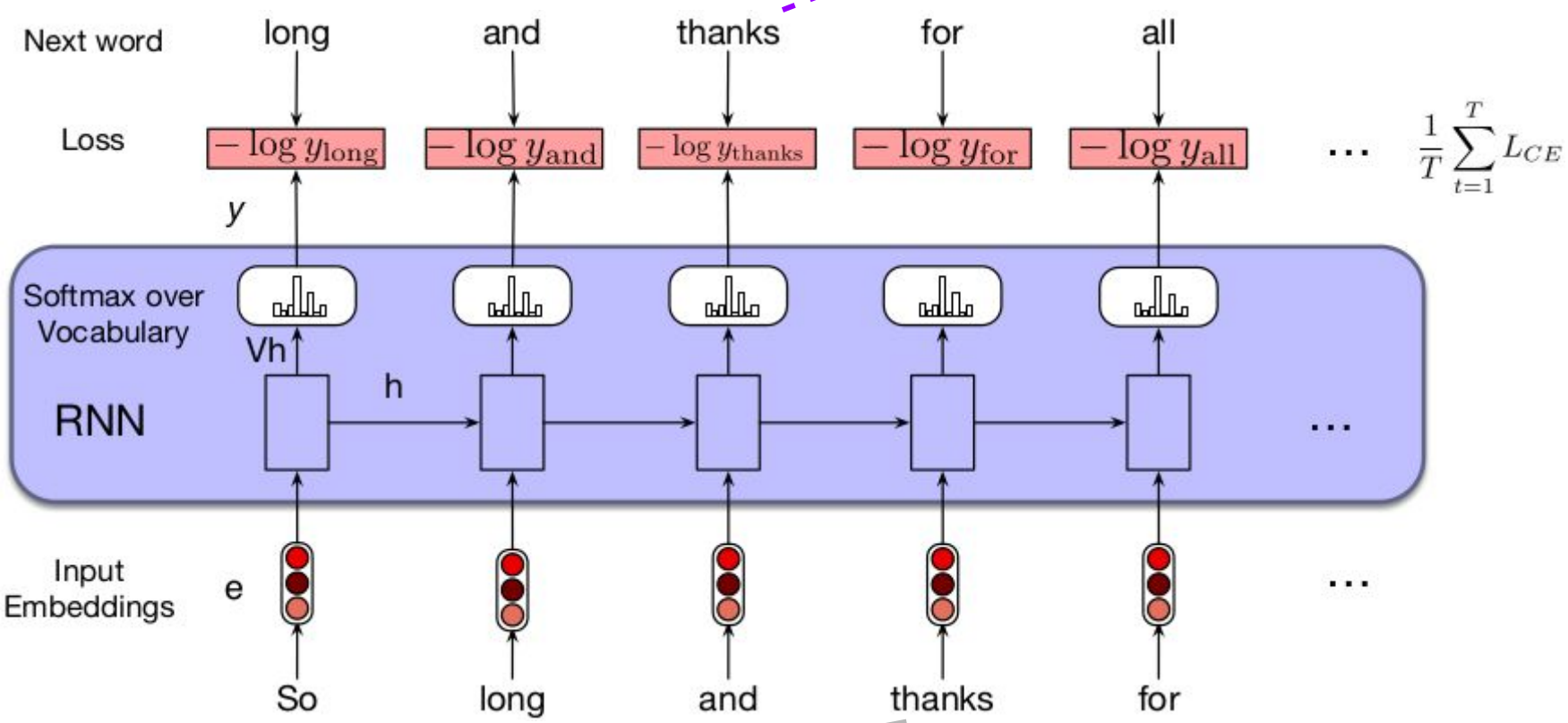
Entrenamiento

- Autosupervisado: usamos oraciones de un corpus conocido y no necesitamos etiquetas
- Se busca minimizar el error de predecir la siguiente palabra dadas todas las anteriores
- El cross-entropy loss en este caso considera que la salida esperada es un one-hot donde solo la siguiente palabra correcta vale 1

$$L_{CE} = -\log \hat{y}_t[w_{t+1}]$$

Entrenamiento

La palabra predicha podría ser la correcta, o no



Pero durante el entrenamiento siempre se presenta la correcta a continuación *(teacher forcing)*

Redes Neuronales como Modelo de Lenguaje

Otras denominaciones: modelo *autorregresivo* de lenguaje, modelo de lenguaje *causal*

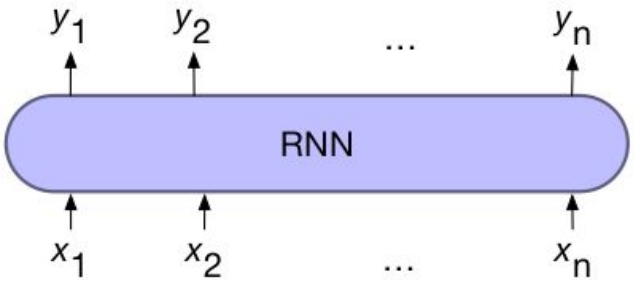
Nos permiten predecir la siguiente palabra

…e iterar y predecir toda una continuación de un *prompt*

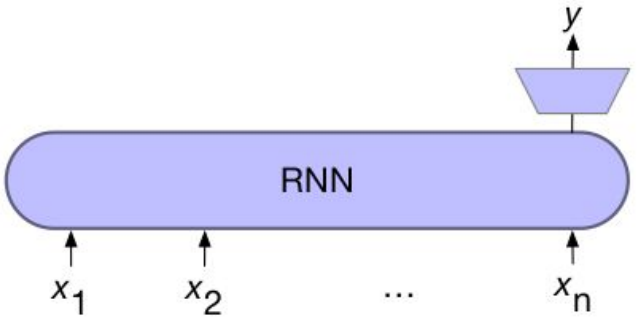
También calcular la probabilidad de toda una secuencia

¿Lo podemos usar para otros tipos de tareas?

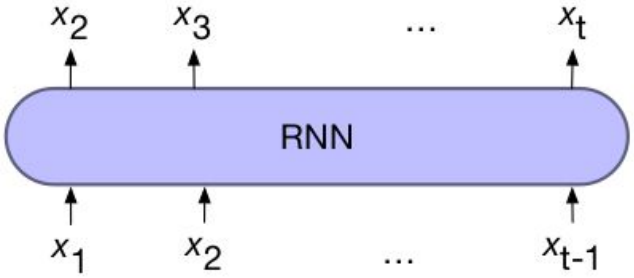
Usos de las redes recurrentes



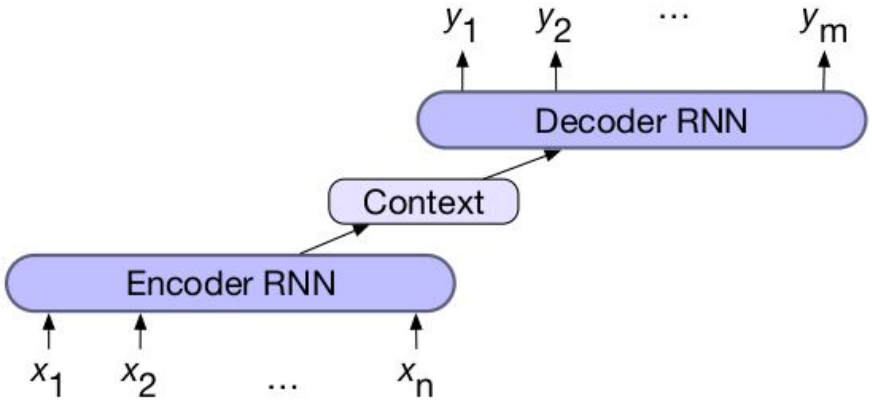
a) etiquetado secuencial



b) clasificación de secuencias



c) modelos de lenguaje



d) modelo encoder-decoder

Arquitectura Encoder-Decoder

Toma una secuencia de entrada

Devuelve una secuencia de salida

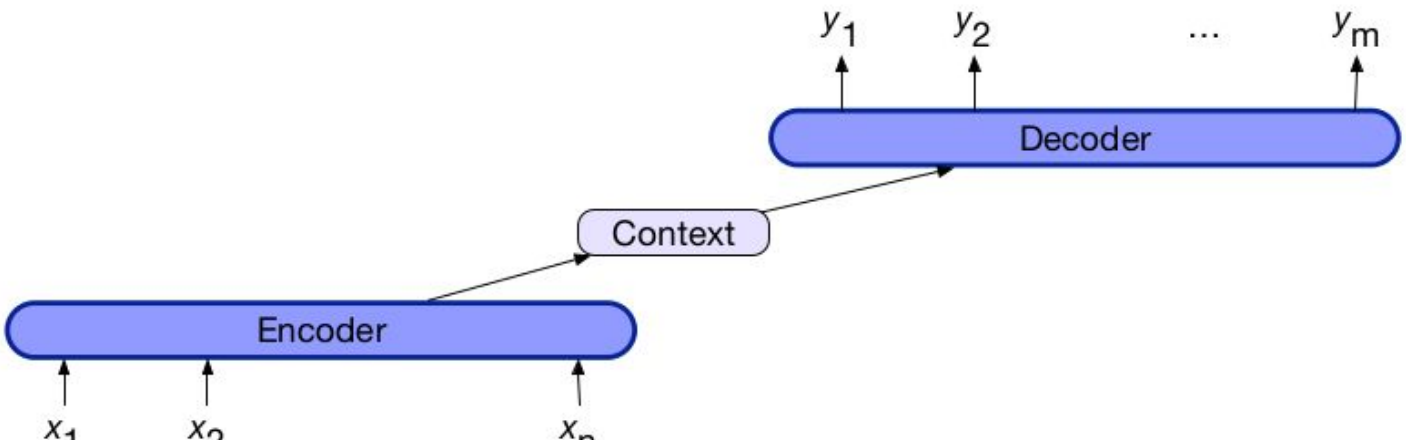
La cantidad de elementos de entrada y de salida pueden ser diferentes

Los tokens posibles también pueden ser diferentes de cada lado, por ejemplo en distinto idioma (traducción automática)

Arquitectura Encoder-Decoder

Una red compuesta por dos subredes:

- **Encoder**: red que codifica la entrada
- **Decoder**: red que decodifica (y construye) la salida
- **Context vector**: "embedding" de toda la secuencia de entrada



Arquitectura Encoder-Decoder

Secuencia de entrada x_1^n

- **Encoder:** toma x_1^n y genera una secuencia de representaciones contextualizadas h_1^n. Se puede hacer con LSTMs, convolucionales, transformers (redes secuenciales).

- **Context vector:** este vector c se construye a partir de las h_1^n y representa "toda la semántica" de la entrada

- **Decoder:** a partir de c, genera una tira de estados ocultos h_1^m con los que se construye las salidas de la red y_1^m

Generar salida a partir de una entrada

Supongamos que estamos construyendo un sistema que traduce del inglés al español

En nuestra red usaremos la secuencia:

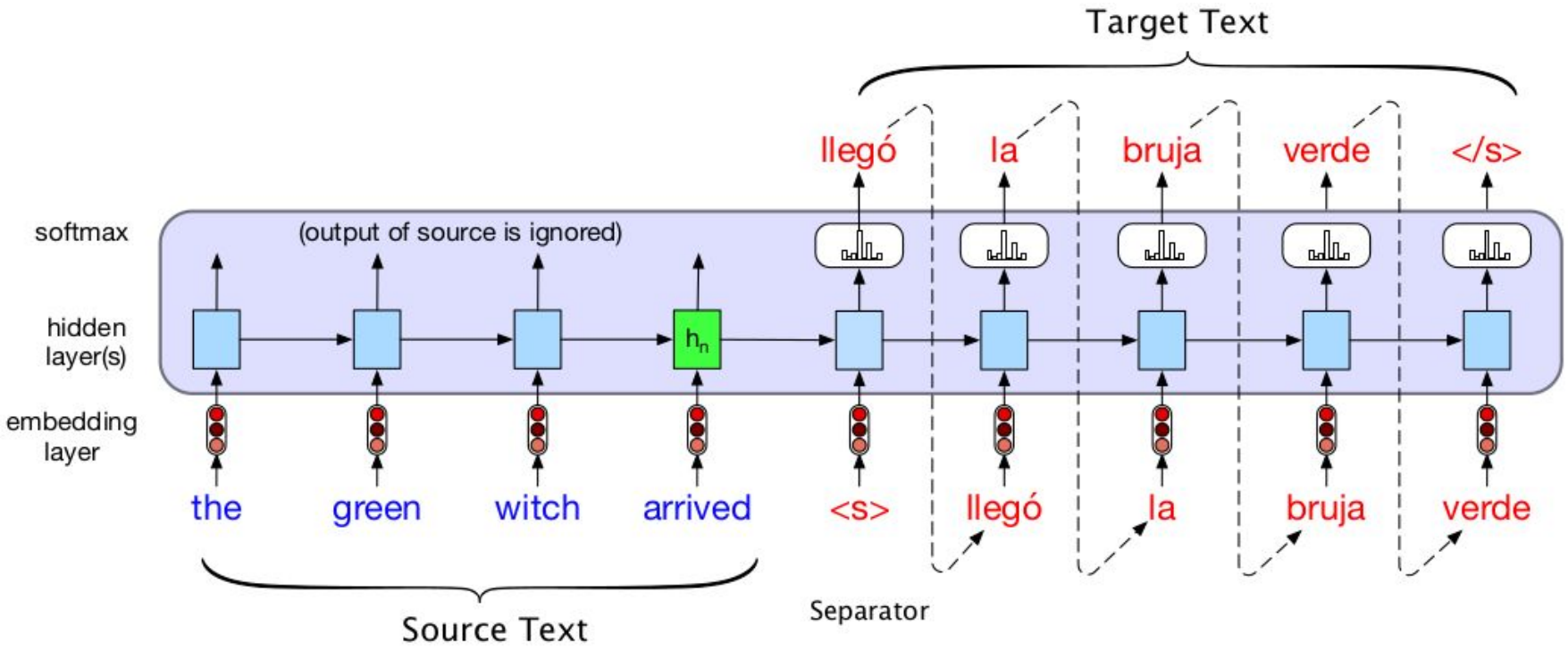
(salida esperada)

the green witch arrived <s> llegó la bruja verde

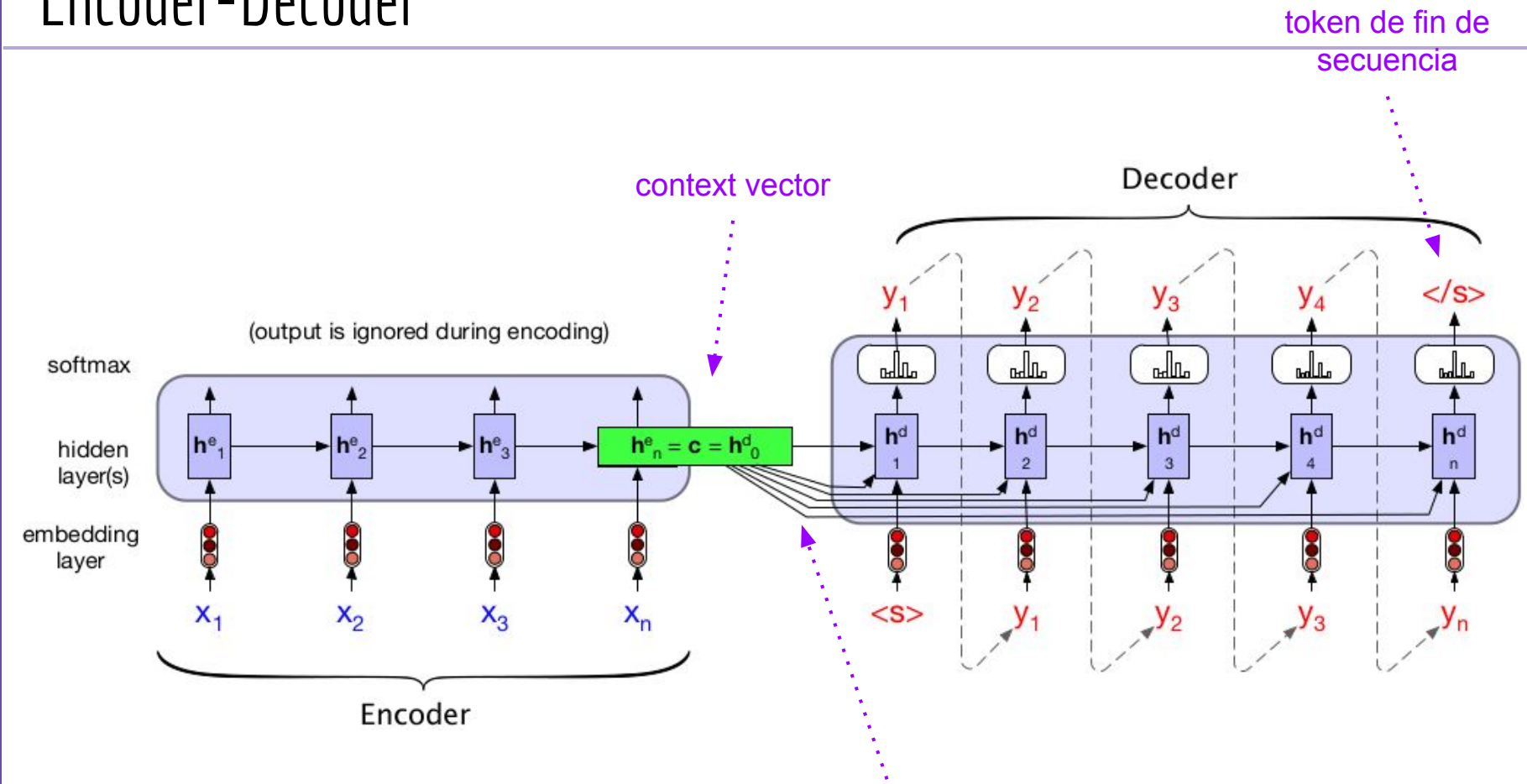
Alimentamos la red con los tokens en inglés hasta el token <s>, generando el estado oculto *(forward inference)*

Luego pasamos a usarlo como modelo de lenguaje generativo para obtener las palabras en español *(autoregressive generation)*

Generar salida a partir de una entrada



Encoder-Decoder



debido a que la influencia de c sobre la decodificación de la secuencia puede ir disminuyendo, se suele incluir en cada paso

Encoder-Decoder: entrenamiento

Se entrenan con pares de secuencias (entrada - salida)

Por ejemplo, corpus paralelos en traducción automática: oraciones origen en inglés y oraciones destino en español

La red *encoder* codifica la entrada, se ignoran las salidas intermedias y se mantiene solo el vector de contexto

A partir del vector de contexto se decodifica la salida con el *decoder* de forma autorregresiva

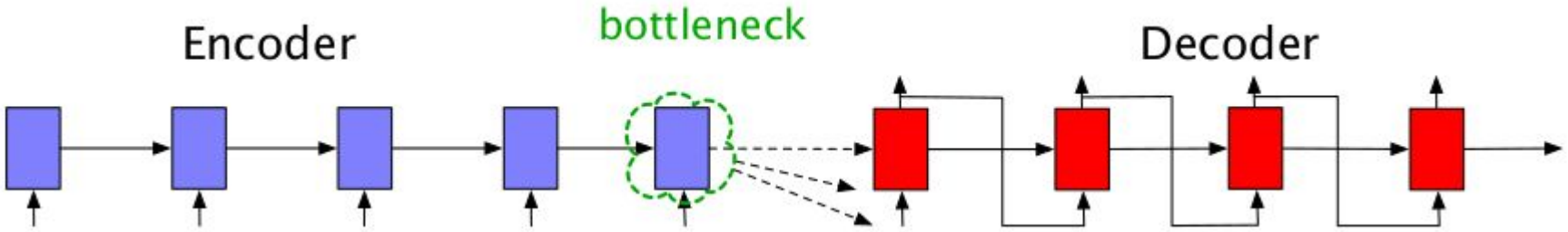
Se entrena *end-to-end*, minimizando el *loss* de acertar a las palabras correctas

Se utiliza *teacher forcing* al entrenar la decodificación de la salida

Problema del vector de contexto

El encoder genera un estado oculto para cada token de la secuencia $h^e_1 ... h^e_n$

Toda esta información se condensa en el *context vector*



Esto es un problema, porque es un vector de tamaño finito que tiene que poder representar la semántica de cualquier secuencia posible (cuello de botella)

Problema del vector de contexto

Solución: ¿qué tal si en cada paso del decoder se utilizara un vector de contexto distinto?

$$c_i = f(h_1^e ... h_n^e)$$

Cada vector de contexto es calculado en función de los estados ocultos del encoder para cada token de la entrada

La cantidad de tokens es variable, y la cantidad de estados ocultos también, lo que haremos es crear un vector de tamaño fijo usando pesos para combinar los distintos $h^e_1 ... h^e_n$

Mecanismo Atencional

Los estados ocultos del decoder ahora se calculan:

$$h_i^d = g(\hat{y}_{i-1}, h_{i-1}^d, c_i)$$

Los pesos que usamos para combinar los h^e_j y obtener c_i dan una idea de qué tan importante es cada token *j* al momento de decodificar el paso *i*

Pesos más grandes implican que el modelo le está "prestando más atención" a ese token en ese paso de decodificación

Le asignaremos un *score* a cada par (h^d_{i-1}, h^e_j), el esquema atencional que usaremos definirá cómo se calcula ese score

Cálculo de scores

El score que le damos al par (h^d_{i-1}, h^e_j) representa qué tanta atención se le presta a h^e_j al generar la salida *i*

Una primera idea es medir qué tan similares son el estado oculto anterior y el estado oculto del token *j*

$$\text{score}(h^d_{i-1}, h^e_j) = h^d_{i-1} \cdot h^e_j$$

Esta es la atención más simple de todas: atención de producto punto (*dot product attention*), y no introduce parámetros nuevos

Effective Approaches to Attention-based Neural Machine Translation
Luong, Pham, Manning, 2015

Cálculo del vector de contexto

Una vez que tenemos definida una función de score, pasamos definir cómo se calcula el vector de contexto para el paso *i*

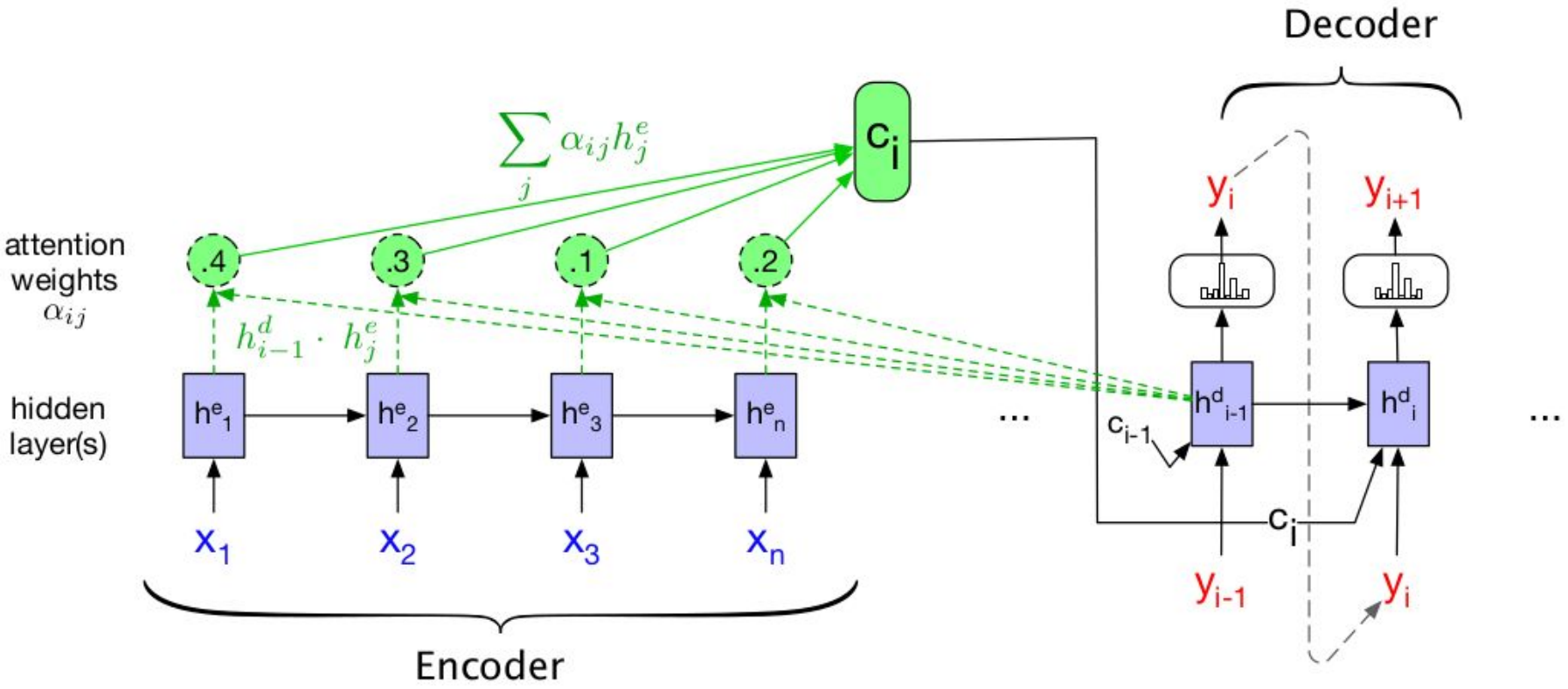
Softmax sobre todos los scores para normalizarlos a pesos

$$\alpha_{ij} = \text{softmax}(\text{score}(h^d_{i-1}, h^e_j))$$
$$= \frac{\exp(\text{score}(h^d_{i-1}, h^e_j))}{\sum_k \exp(\text{score}(h^d_{i-1}, h^e_k))}$$

Multiplicamos los pesos por los estados ocultos del encoder para generar el vector de contexto

$$c_i = \sum_j \alpha_{ij} h^e_j$$

Mecanismo Atencional



Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

1 Introduction

Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started

Transformer

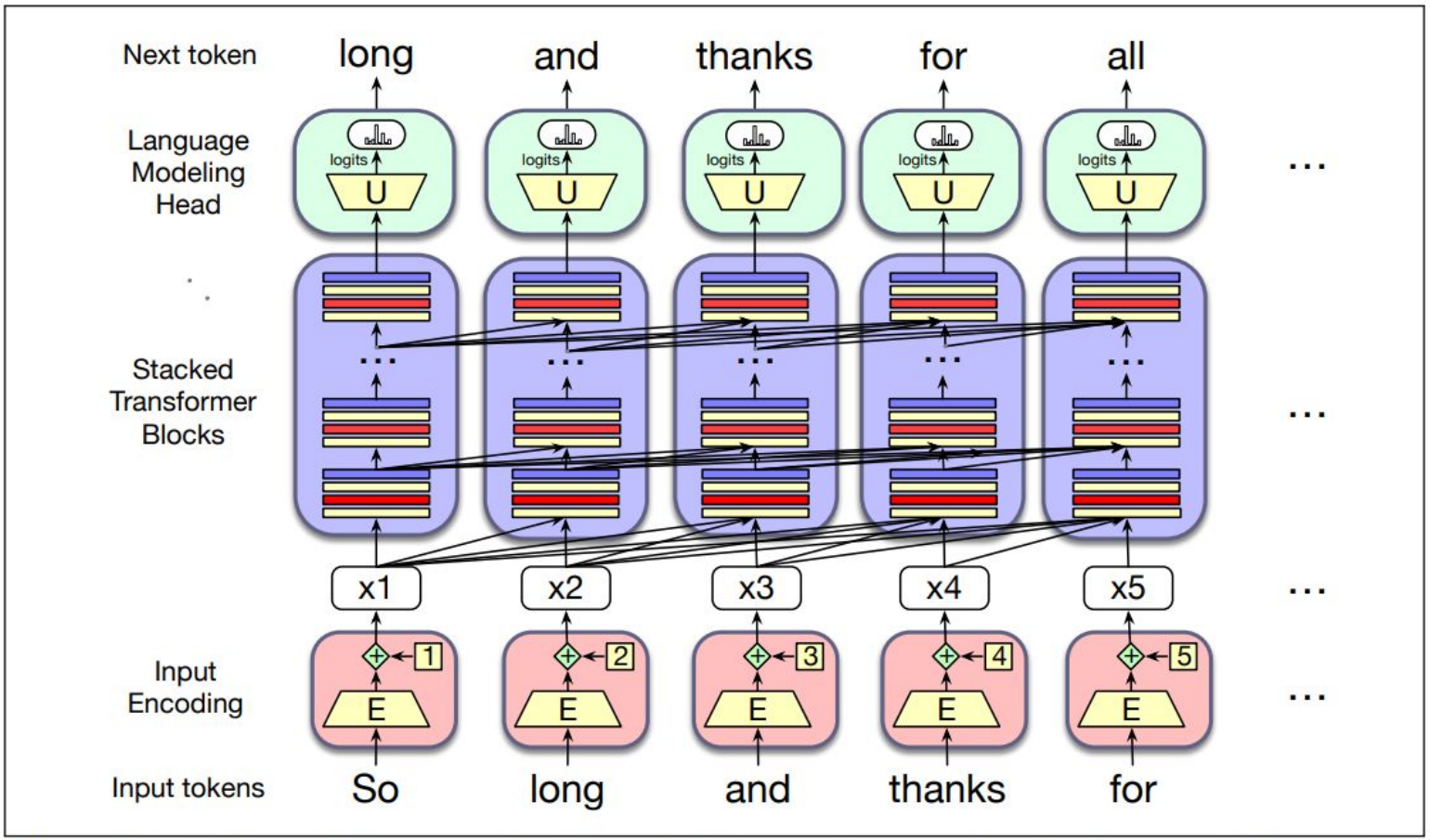


Figure 8.1 The architecture of a (left-to-right) transformer, showing how each input token get encoded, passed through a set of stacked transformer blocks, and then a language model head that predicts the next token.

Transformer

Las RNNs obligan a procesar una palabra después de otra en orden

El transformer liberará esta restricción procesando todo en paralelo

Introduce varios conceptos a analizar:

- Self-attention / Autoatención
- Conexiones residuales
- Embeddings posicionales
- Masked attention y Cross-attention

Motivación

Los transformers son capaces de crear embeddings contextuales (no son los únicos)

Ejemplo:

*The chicken didn't cross the road because **it** was too tired.*

*The chicken didn't cross the road because **it** was too wide.*

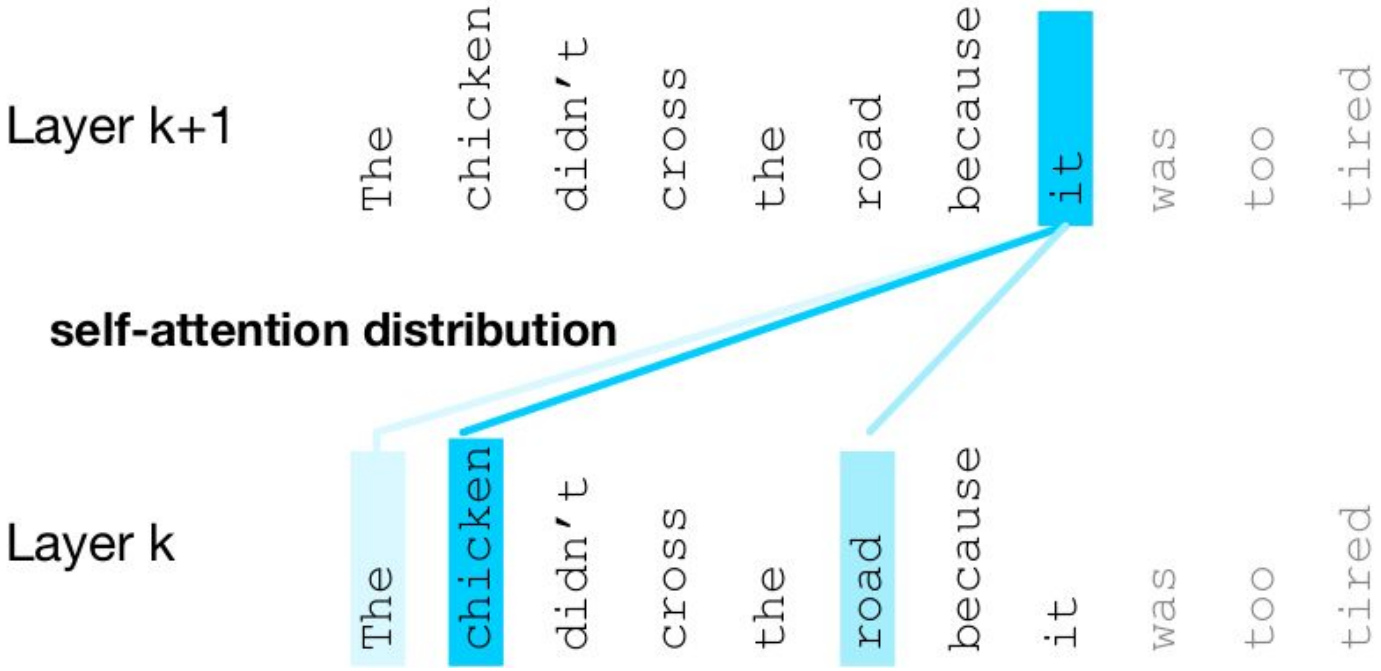
En español podría ser:

*El pollo no cruzó la calle porque **estaba** muy cansado*

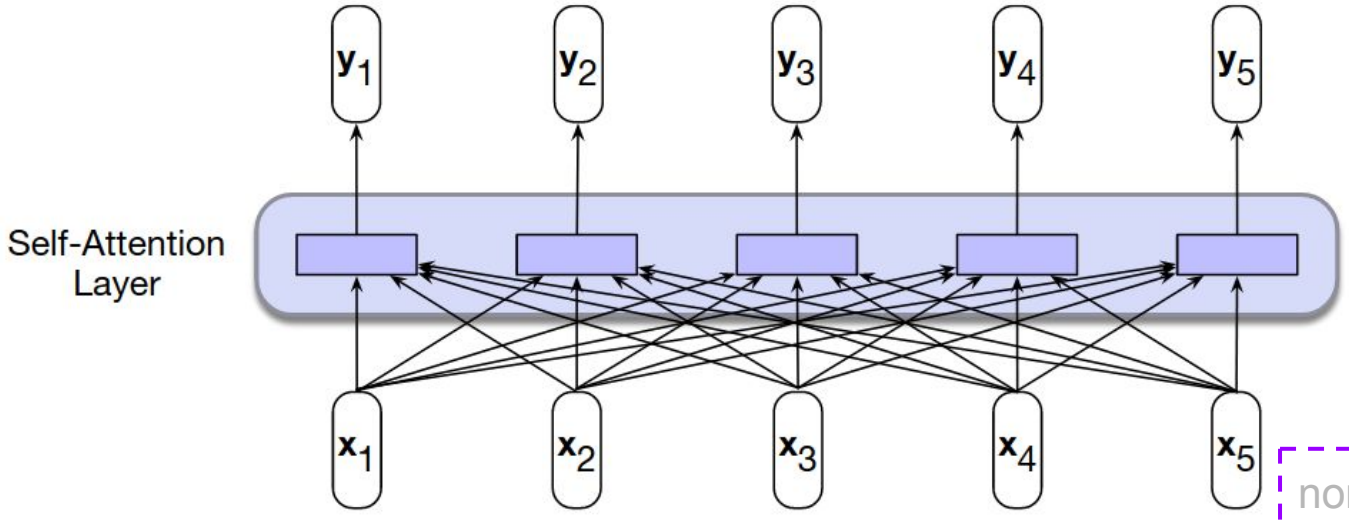
*El pollo no cruzó la calle porque **estaba** cortada*

Motivación

¿Qué esperamos que pase con el mecanismo de atención en estos casos?



Self-attention



normalización para disminuir la varianza

Tres proyecciones de cada $\mathbf{x}_i$ (**query**, **key** y **value**), con tres matrices

Los vectores $\mathbf{q}_i$ y $\mathbf{k}_j$ computan el **score** de cómo influye la entrada j para la entrada i

Luego softmax (en j) que ponderan los $\mathbf{v}_i$

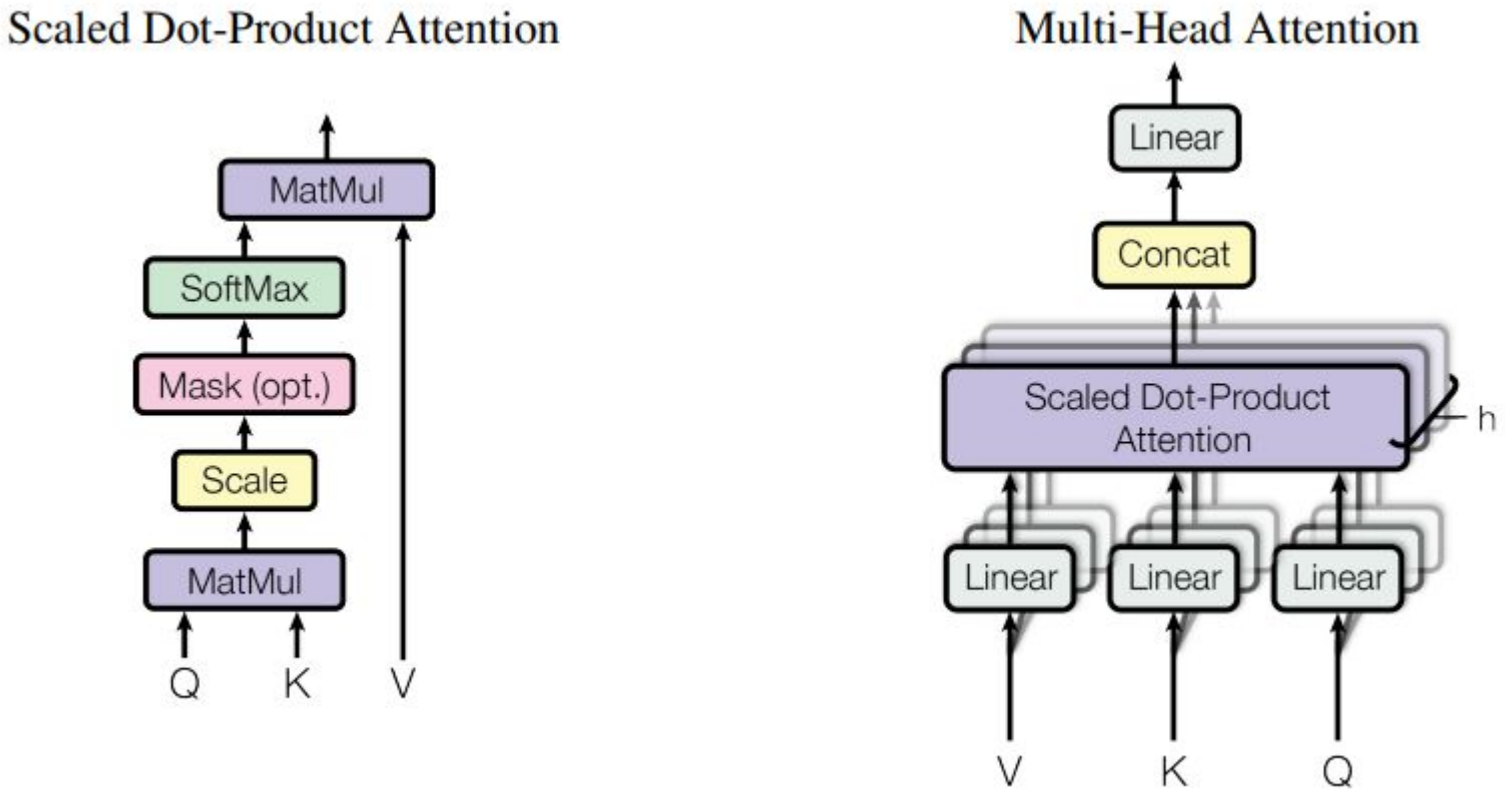
$$\mathbf{q}_i = \mathbf{x}_i\mathbf{W}^{\mathbf{Q}}; \quad \mathbf{k}_j = \mathbf{x}_j\mathbf{W}^{\mathbf{K}}; \quad \mathbf{v}_j = \mathbf{x}_j\mathbf{W}^{\mathbf{V}}$$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j))$$

$$\mathbf{y}_i = \sum \alpha_{ij}\mathbf{v}_j$$

Multi-head Attention



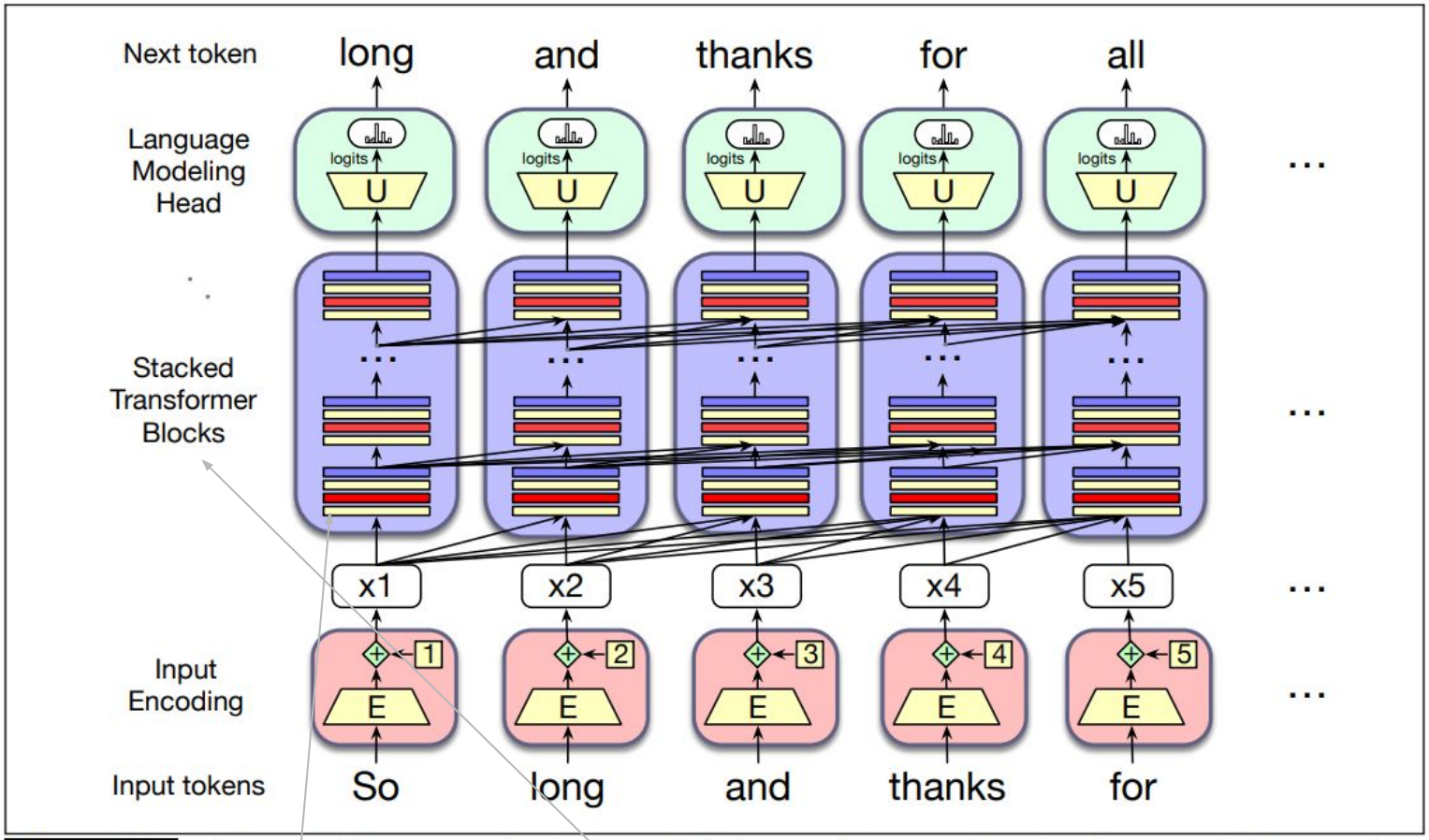


Figure 8.1 The architecture of a (left-to-right) transformer, showing how each input token get encoded, passed through a set of stacked transformer blocks, and then a language model head that predicts the next token.

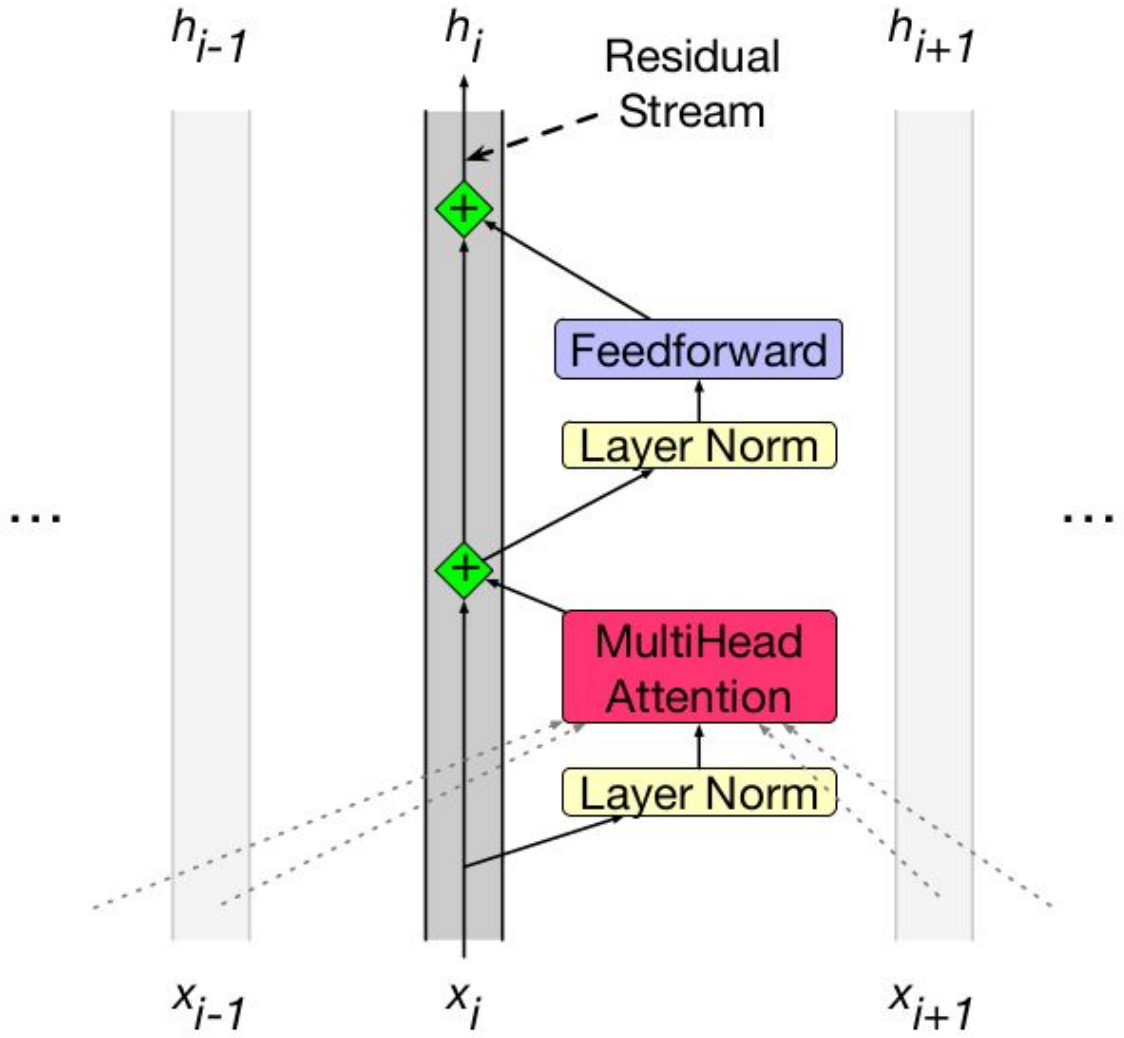
Mecanismo de self-attention

Bloque Transformer

Bloque Transformer

Faltan algunos elementos más del bloque transformer:

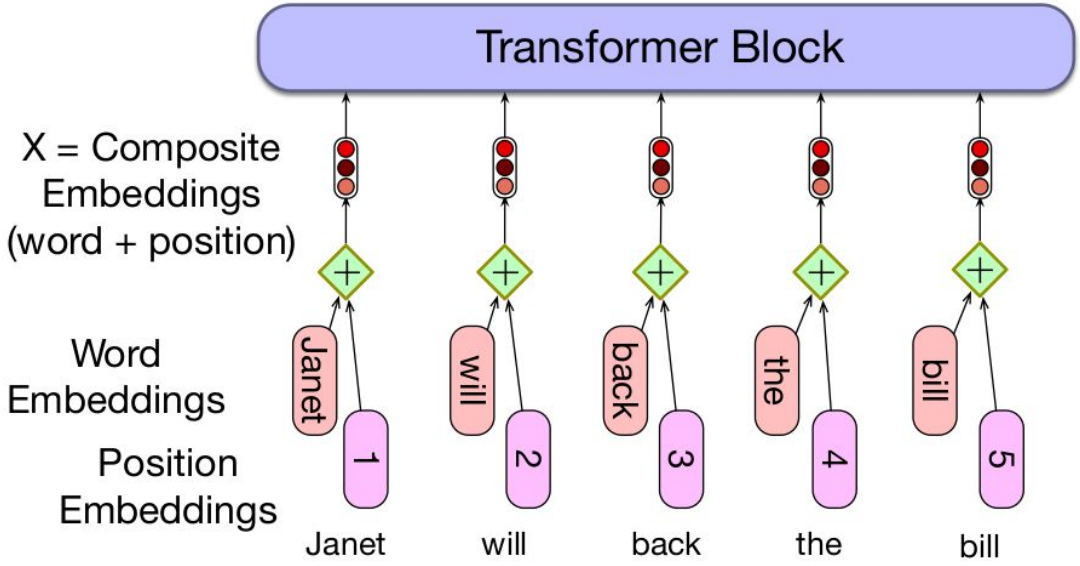
- Capa feedforward
- Conexiones residuales
- Capas de normalización



Codificación de la entrada

Hasta este punto no hay noción de orden en la entrada

Ni hay una recurrencia que "imponga" orden procesando de a uno

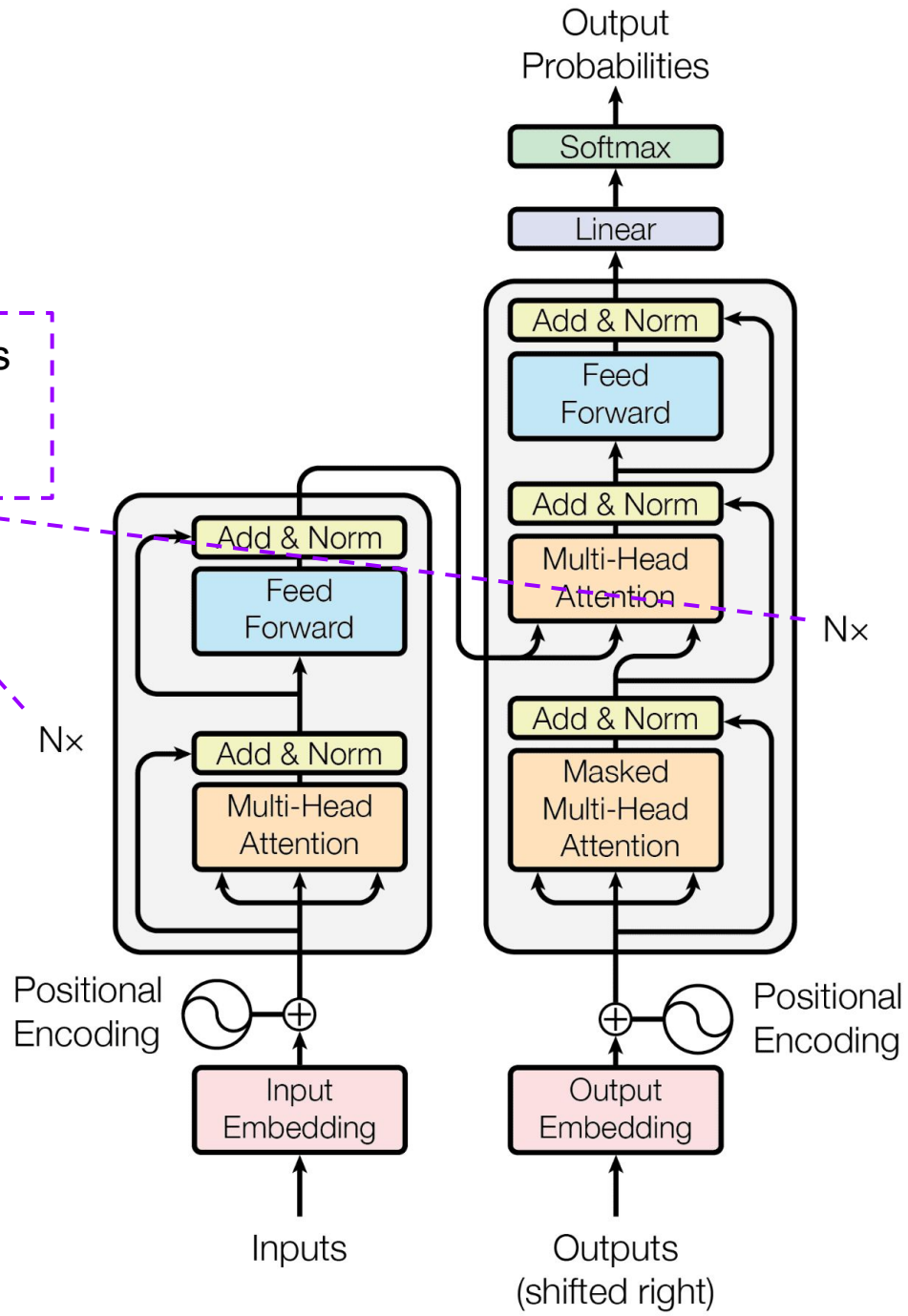


Solución para los Transformers:

"Aumentar" cada x_i para que incluya una representación de su posición

Embeddings posicionales (position embeddings)

Notar que son varios bloques transformer en stack



Formas de uso

El transformer original fue planteado como arquitectura encoder-decoder

Primer ejemplo de uso: traducción automática

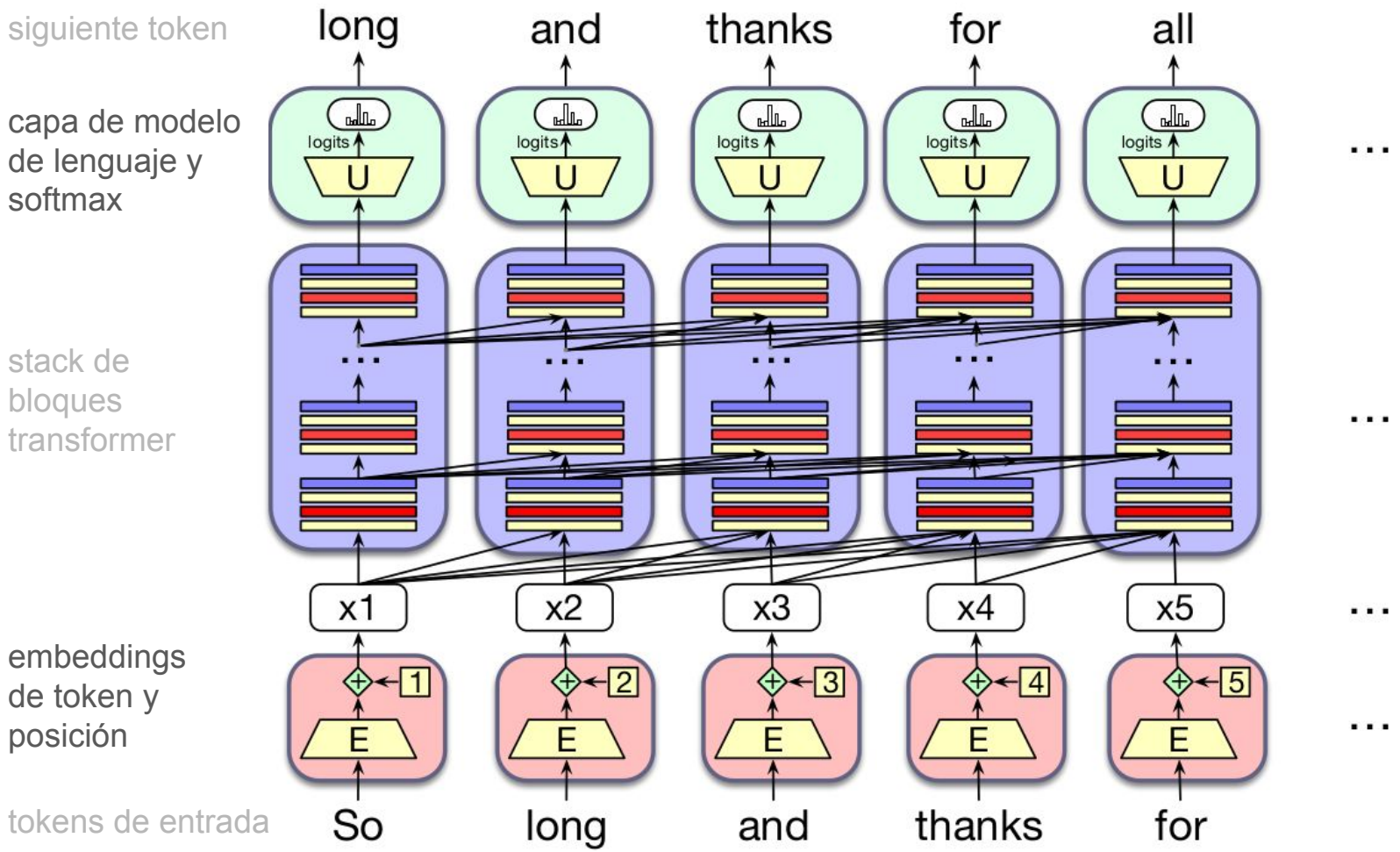
Pero luego aparecen otras variantes

Encoder-only (por ejemplo BERT)

Decoder-only (por ejemplo GPT)

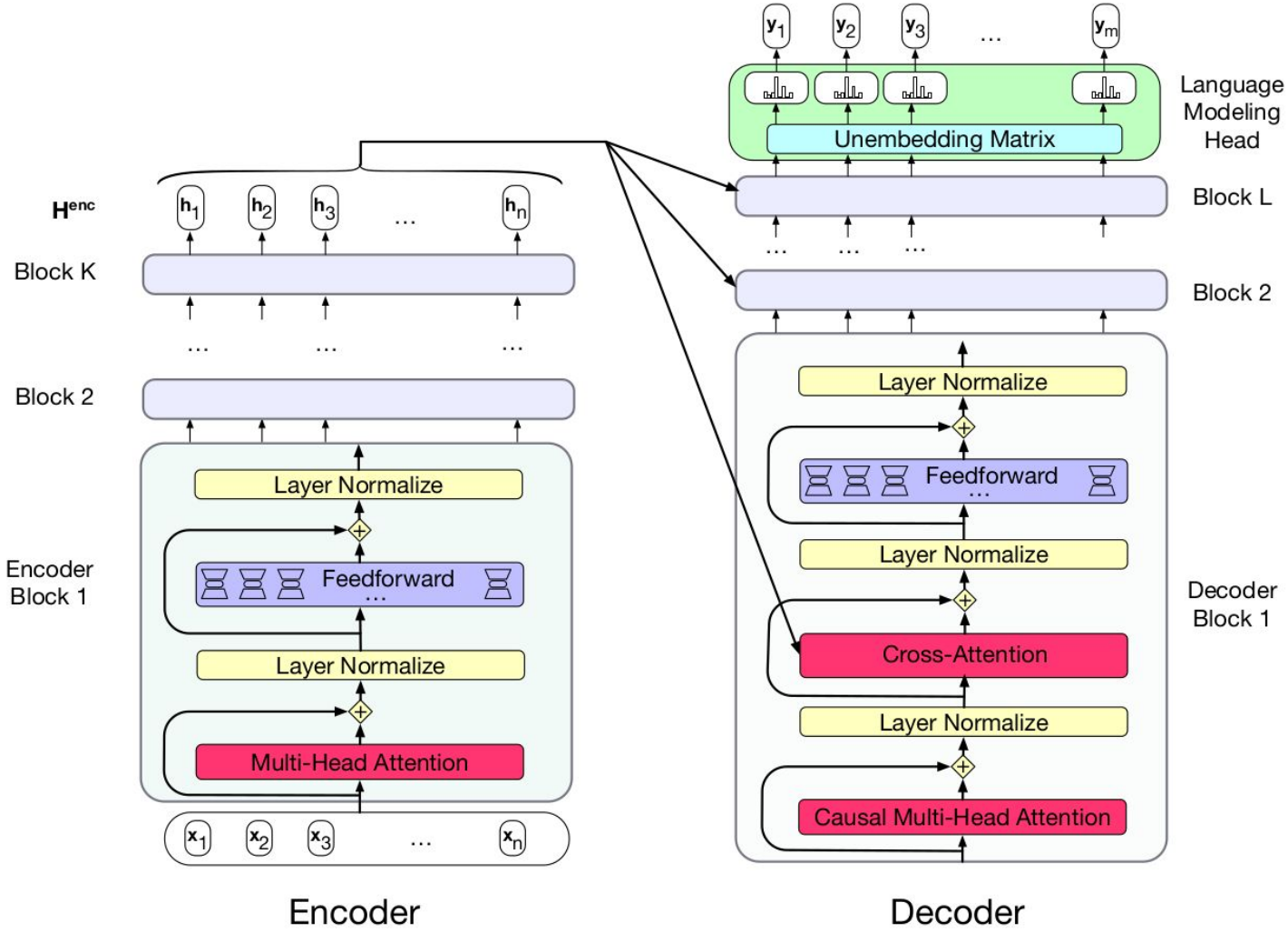
Transformer: Modelo de Lenguaje

Se usa el transformer en modo "causal"



Transformer: Traducción Automática

Se usa el modelo completo en modo encoder-decoder

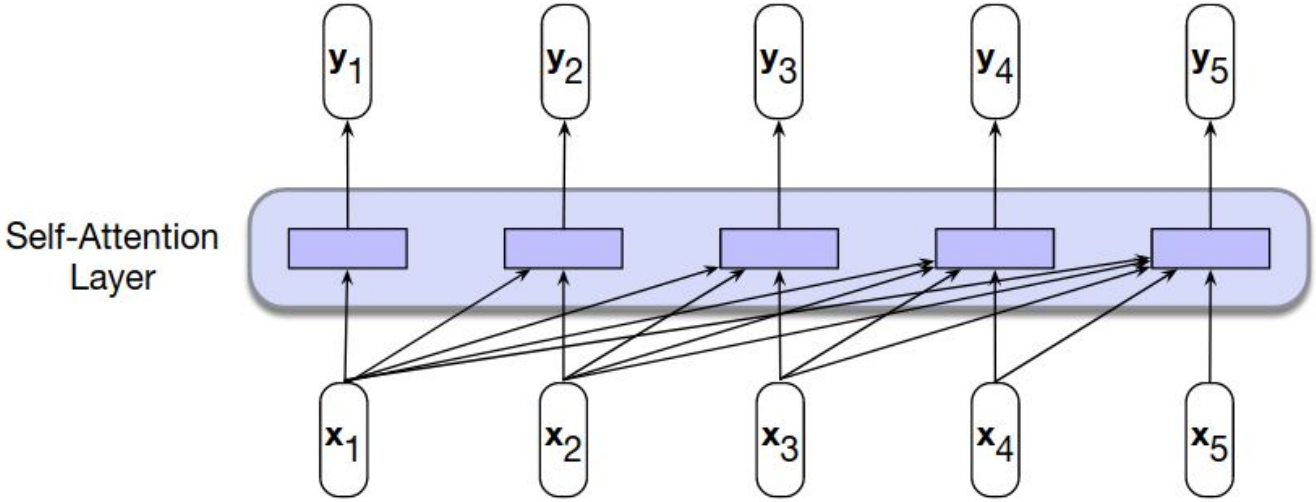


Transformer: Modelo de Lenguaje

En estos casos estamos usando el transformer en modo decoder

Va generando las palabras de a una

Por lo tanto, al procesar una palabra, no podemos tener acceso a las palabras siguientes, solo a las anteriores



Grandes Modelos de Lenguaje

- Hipótesis distribucional: algunos aspectos del significado podemos aprenderlos solamente leyendo.

- Cuando leemos, vemos las palabras en conjunto, no de a una

- Los LLMs han mostrado que se puede aprender *mucho* simplemente intentando predecir la próxima palabra

Definimos a los Grandes Modelos de Lenguaje como el resultado de entrenar modelos para predecir la siguiente palabra, sobre enormes cantidades de texto. Más técnicamente: una red neuronal que toma una entrada como contexto o prefijo, y devuelve una distribución de probabilidad sobre las posibles siguientes palabras.

Grandes Modelos de Lenguaje

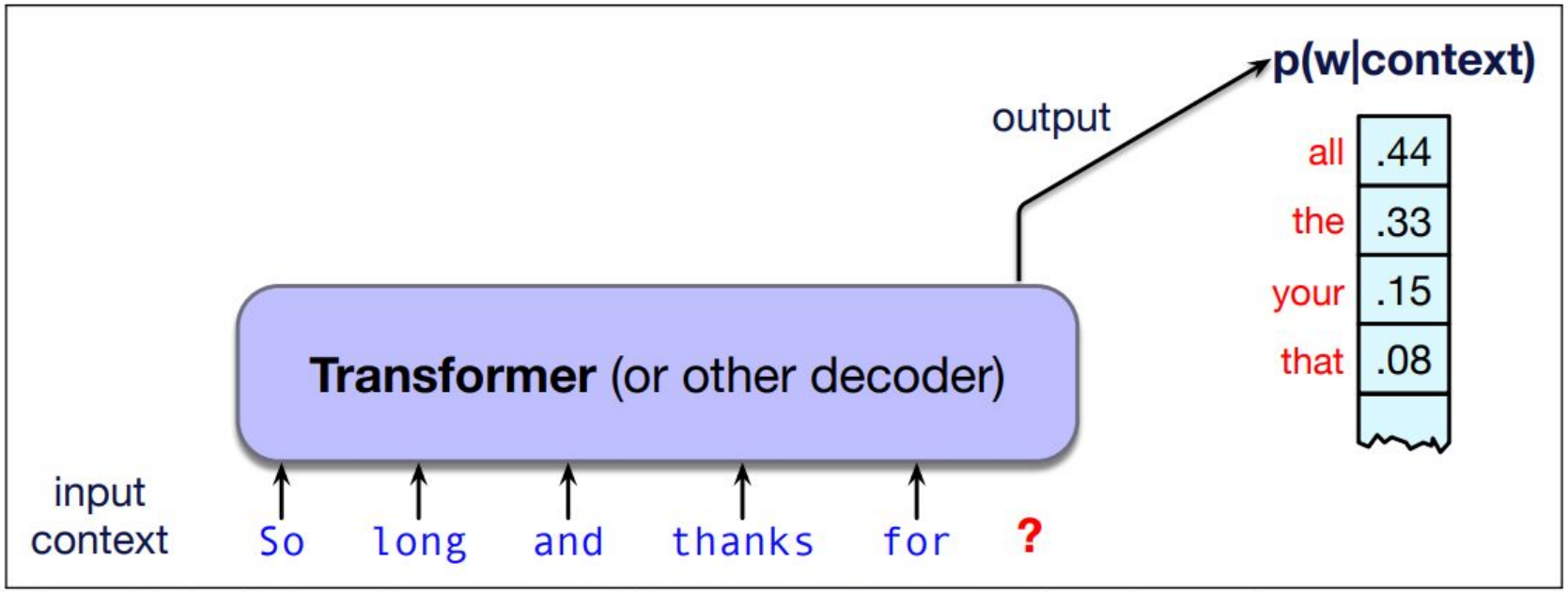


Figure 7.1 A large language model is a neural network that takes as input a context or prefix, and outputs a distribution over possible next words.

Nota: generalmente los LLMs usan Transformers, pero la clase de hoy será agnóstica al respecto

Grandes Modelos de Lenguaje

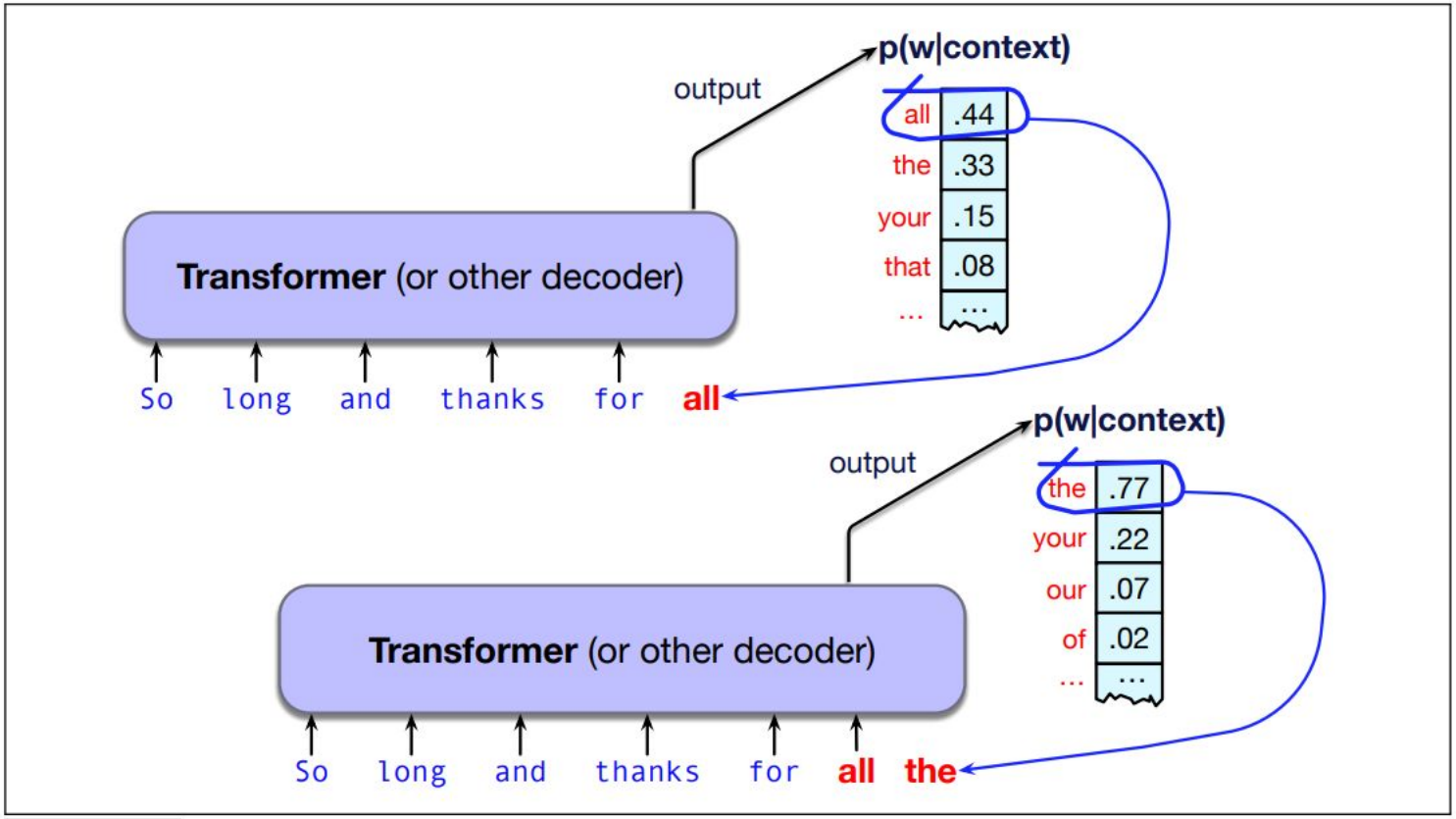


Figure 7.2 Turning a predictive model that gives a probability distribution over next words into a generative model by repeatedly sampling from the distribution. The result is a left-to-right (also called autoregressive) language models. As each token is generated, it gets added onto the context as a prefix for generating the next token.

Arquitecturas

- Encoder: recibe una secuencia de tokens y devuelve una representación para cada token. BERT, RoBERTA, Robertuito, ROUBerta. No son generativos. Lo veremos más adelante.

- Encoder-Decoder: ya lo vimos con RNNs. Toman una secuencia de tokens, los codifican, y a partir de esa representación generan una salida. Caso típico: Traducción Automática

- Decoder: toma una serie de tokens, y va generando iterativamente salidas a continuación. Es lo que llamamos en general LLMs: GPT, Claude, Llama, Mistral.

Arquitecturas

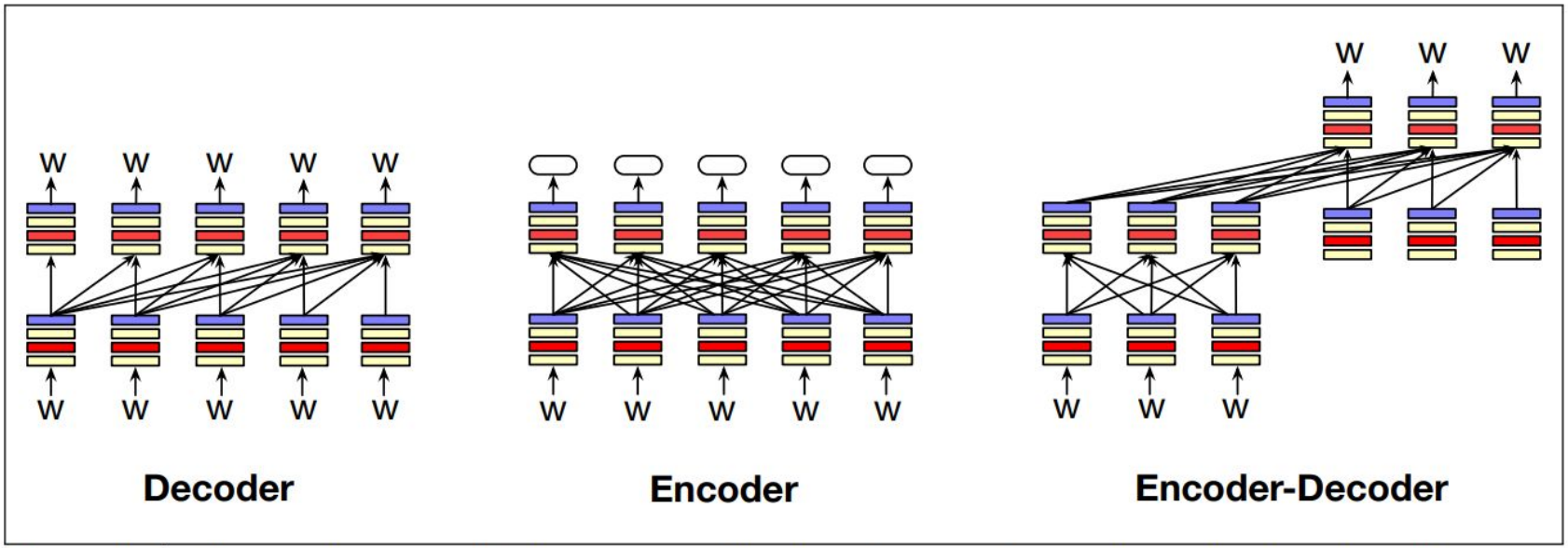


Figure 7.3 Three architectures for language models: decoders, encoders, and encoder-decoders. The arrows sketch out the information flow in the three architectures. Decoders take tokens as input and generate tokens as output. Encoders take tokens as input and produce an encoding (a vector representation of each token) as output. Encoder-decoders take tokens as input and generate a series of tokens as output.

Prompting e In Context Learning

- Podemos usar el contexto de un Decoder para "inducir" la respuesta correcta

- No cambiamos los pesos del modelo!

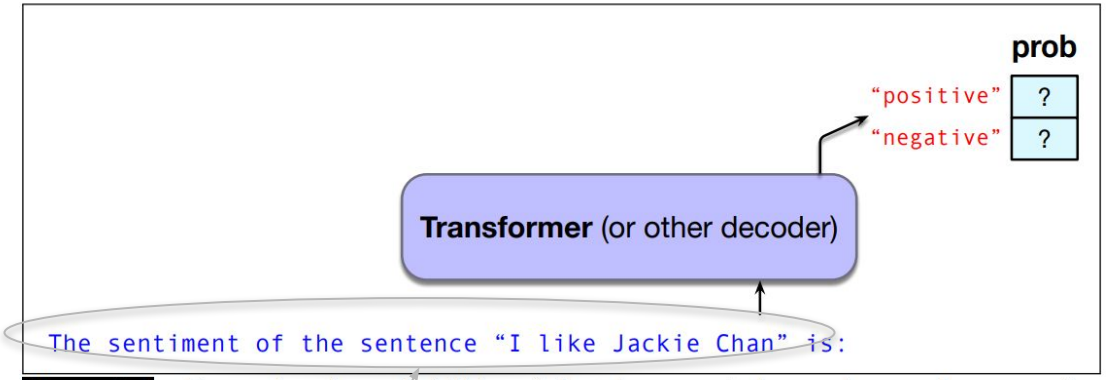


Figure 7.4 Computing the probabilities of the tokens `positive` and `negative` occurring after this prefix.

Prompt

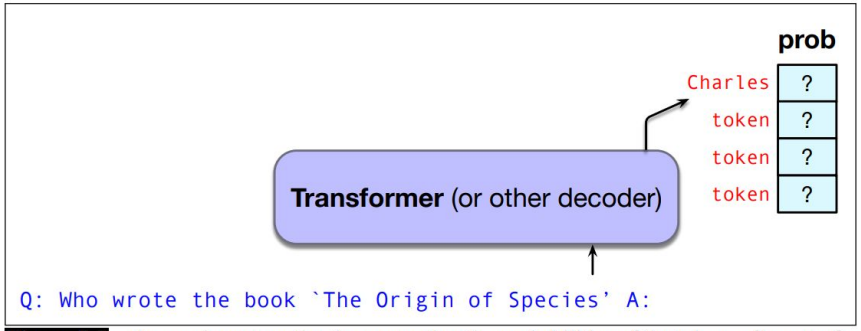


Figure 7.5 Answering a question by computing the probabilities of the tokens after a prefix stating the question; in this example the correct token `Charles` has the highest probability.

In Context Learning

The three settings we explore for in-context learning

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
1  Translate English to French:      ← task description
2  cheese =>                         ← prompt
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
1  Translate English to French:      ← task description
2  sea otter => loutre de mer        ← example
3  cheese =>                         ← prompt
```

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
1  Translate English to French:      ← task description
2  sea otter => loutre de mer        ← examples
3  peppermint => menthe poivrée      ←
4  plush girafe => girafe peluche    ←
5  cheese =>                         ← prompt
```

Traditional fine-tuning (not used for GPT-3)

Fine-tuning

The model is trained via repeated gradient updates using a large corpus of example tasks.

Language Models are Few-Shot Learners
Brown et al., 2020

In Context Learning

- Zero shot: no hay ejemplos (!)

- One shot: un ejemplo

- Few Shot: algunos ejemplos

- Many shot: unos cuantos ejemplos

Todos los ejemplos en el prompt

In Context Learning

They[Killer] had **killed**[Target] or captured about a quarter of the enemy's known leaders[Victim].

Figure 1: Annotation example of the Killing frame, extracted from FrameNet.

```
# GOAL:
Your goal is to detect which events, also
↪  called frames, are present in a text,
↪  by identifying which is the word that
↪  evokes the event or frame, called
↪  trigger.

Only the following events are of interest:
    - Killing
    - Robbery
    - Violence
    - Rape
    - Shoot_projectiles
    - Abusing
```

```
# EVENTS:
That word that evokes an event could be of
↪  any syntax type: a noun, a verb, an
↪  adjective, an adverb.

Below there are the definition of each
↪  frame and a list of triggers for each
↪  event of interest, but synonyms of
↪  those words might fit the same.

## Rape
- Definition: The words in this frame
↪  describe situations in which the
↪  Perpetrator has sexual intercourse
↪  with the Victim forcibly (or by
↪  threatening the use of force) and
↪  without his or her consent.
- Possible triggers: rape, rapist, raped
- Text example for this frame with trigger
↪  "raped": "Does the press make it more
↪  difficult for raped women to obtain
↪  justice?"
...
```

In-Context Learning for Frame-Semantic Parsing
Garat et al., 2025

In Context Learning

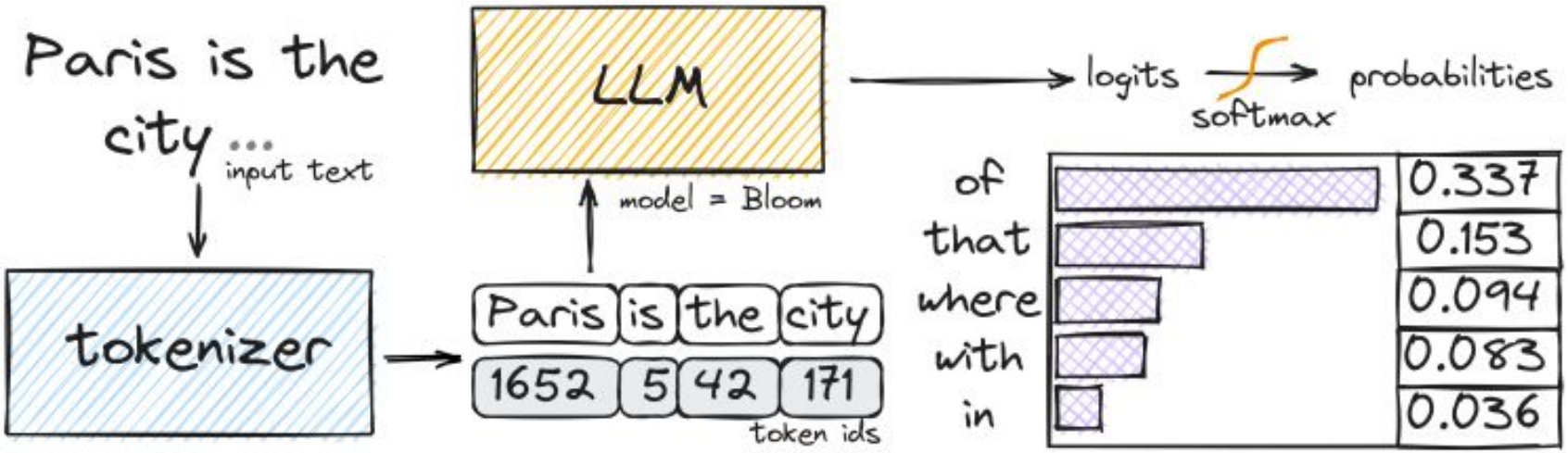
```
# GUIDELINES:
These are the guidelines for your task. These
↪  rules are strict and must be complied:

- You can only use the information provided in
↪  the input to generate your answer.
- Your answer must be one list of detected
↪  pairs (frame, trigger):
    -- If N number of events are found, you
    ↪  must answer with one list of N records.
    -- If no event is detected, the list must
    ↪  be empty.
- Do NOT reply with any explanation or any
↪  other text but one list of records.
- The output must be of the form [[frame 1,
↪  trigger 1],[frame 2, trigger 2]...[frame N,
↪  trigger N]]  formatted as a JSON string.
- Triggers must be substrings of the input.
```

In-Context Learning for Frame-Semantic Parsing
Garat et al., 2025

Generación y Muestreo

Tenemos una distribución sobre la siguiente palabra. ¿Cómo elegimos cuál generar?



How exactly LLM generates text?
Ivan Reznikov
(https://www.linkedin.com/pulse/how-exactly-llm-generates-text-ivan-reznikov)

Greedy Decoding

- La palabra más probable.

How exactly LLM generates text?
Ivan Reznikov
(https://www.linkedin.com/pulse/how-exactly-llm-generates-text-ivan-reznikov)

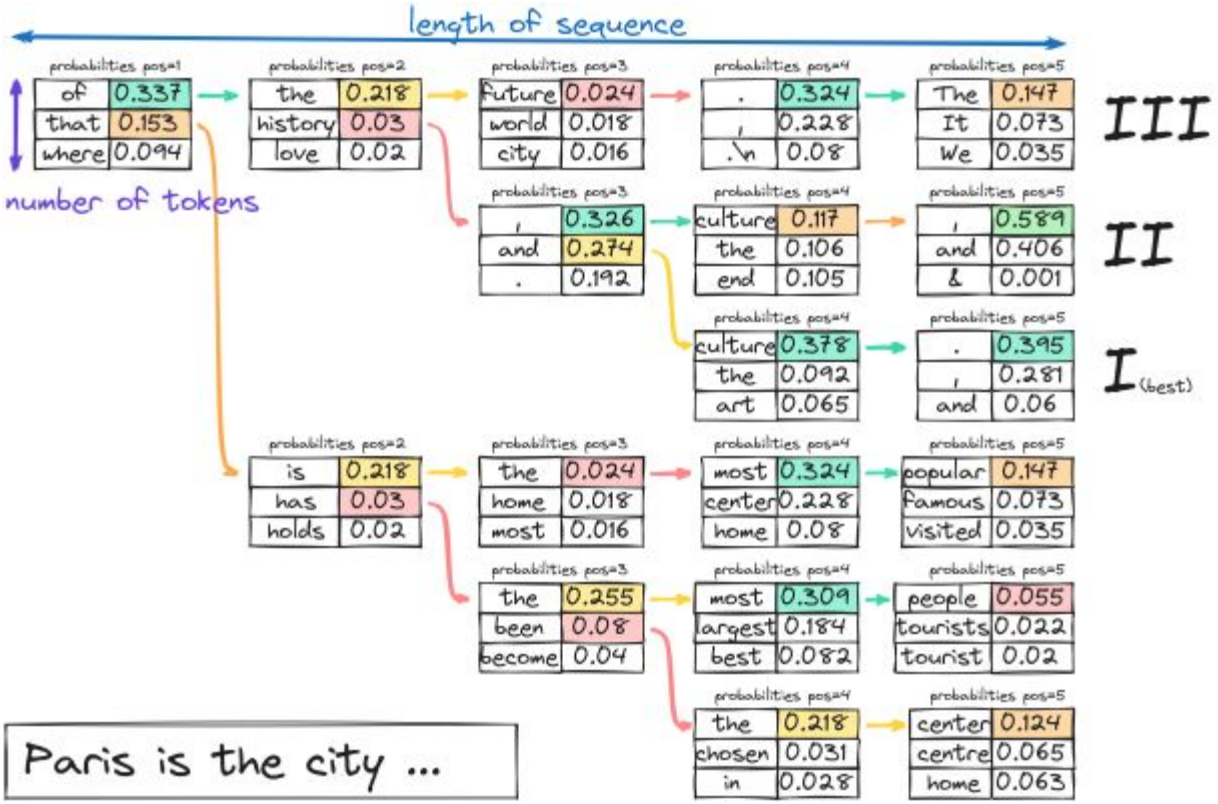
Greedy Decoding

- La palabra más probable.

- Problema: es determinista (casi). Aburrido. (Casi siempre) queremos diversidad en las salidas

How exactly LLM generates text?
Ivan Reznikov
(https://www.linkedin.com/pulse/how-exactly-llm-generates-text-ivan-reznikov)

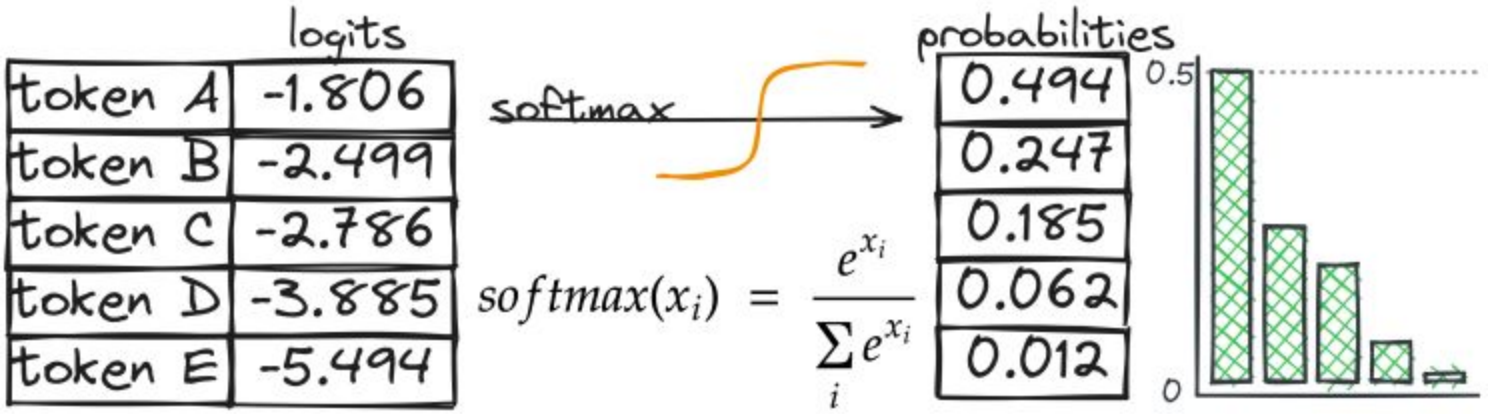
Beam Search



- Sigue siendo determinista, pero podemos mejorar las respuestas viendo más tokens "hacia adelante" (útil en Traducción, por ejemplo)

How exactly LLM generates text?
Ivan Reznikov
(https://www.linkedin.com/pulse/how-exactly-llm-generates-text-ivan-reznikov)

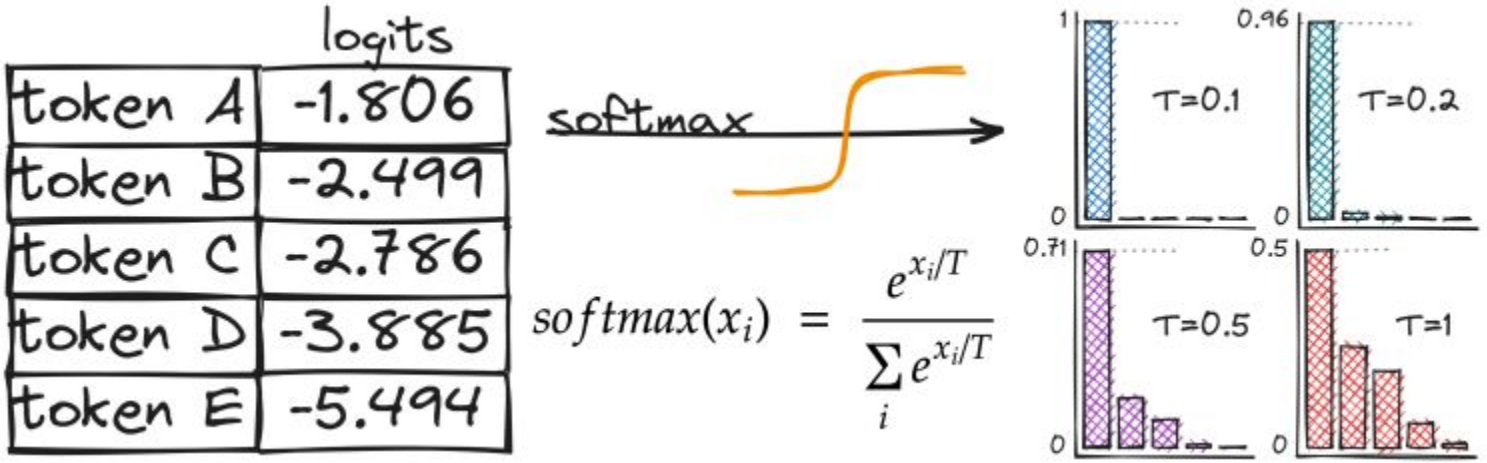
Random Sampling



- Muestreo mi distribución. Cada palabra aparece en forma proporcional a su probabilidad. Ojo: softmax "exagera" valores grandes o pequeños (igual que softmax!)

How exactly LLM generates text?
Ivan Reznikov
(https://www.linkedin.com/pulse/how-exactly-llm-generates-text-ivan-reznikov)

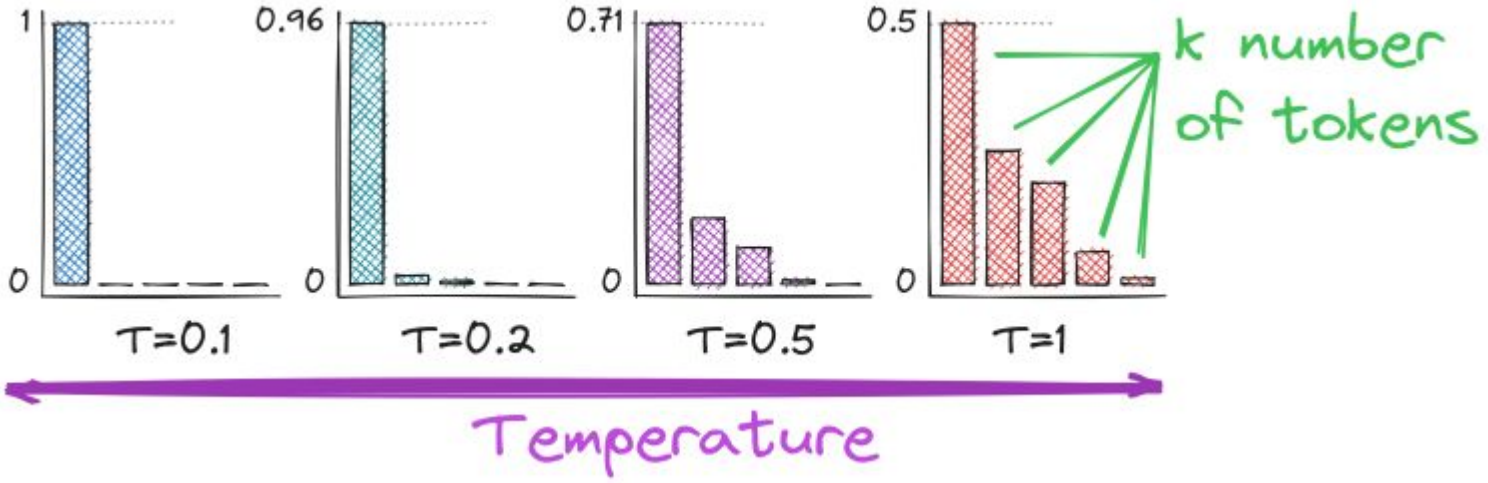
Random Sampling con Temperatura

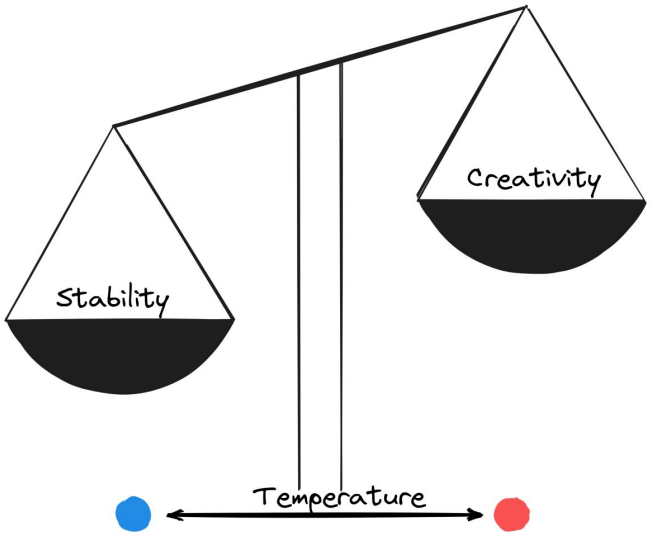


- Cambiamos la distribución para que aumente la probabilidad de tokens más comunes... o los más raros.
- Si T < 1, más cerca estamos de 0, más cerca de hacer greedy decoding.
- Si T>1 (high-temperature sampling), aumenta la diversidad de los resultados (los LLM siempre alucinan... ahora alucinan más)

How exactly LLM generates text?
Ivan Reznikov
(https://www.linkedin.com/pulse/how-exactly-llm-generates-text-ivan-reznikov)

Random Sampling con Temperatura





How exactly LLM generates text?
Ivan Reznikov
(https://www.linkedin.com/pulse/how-exactly-llm-generates-text-ivan-reznikov)

Entrenamiento de LLMs

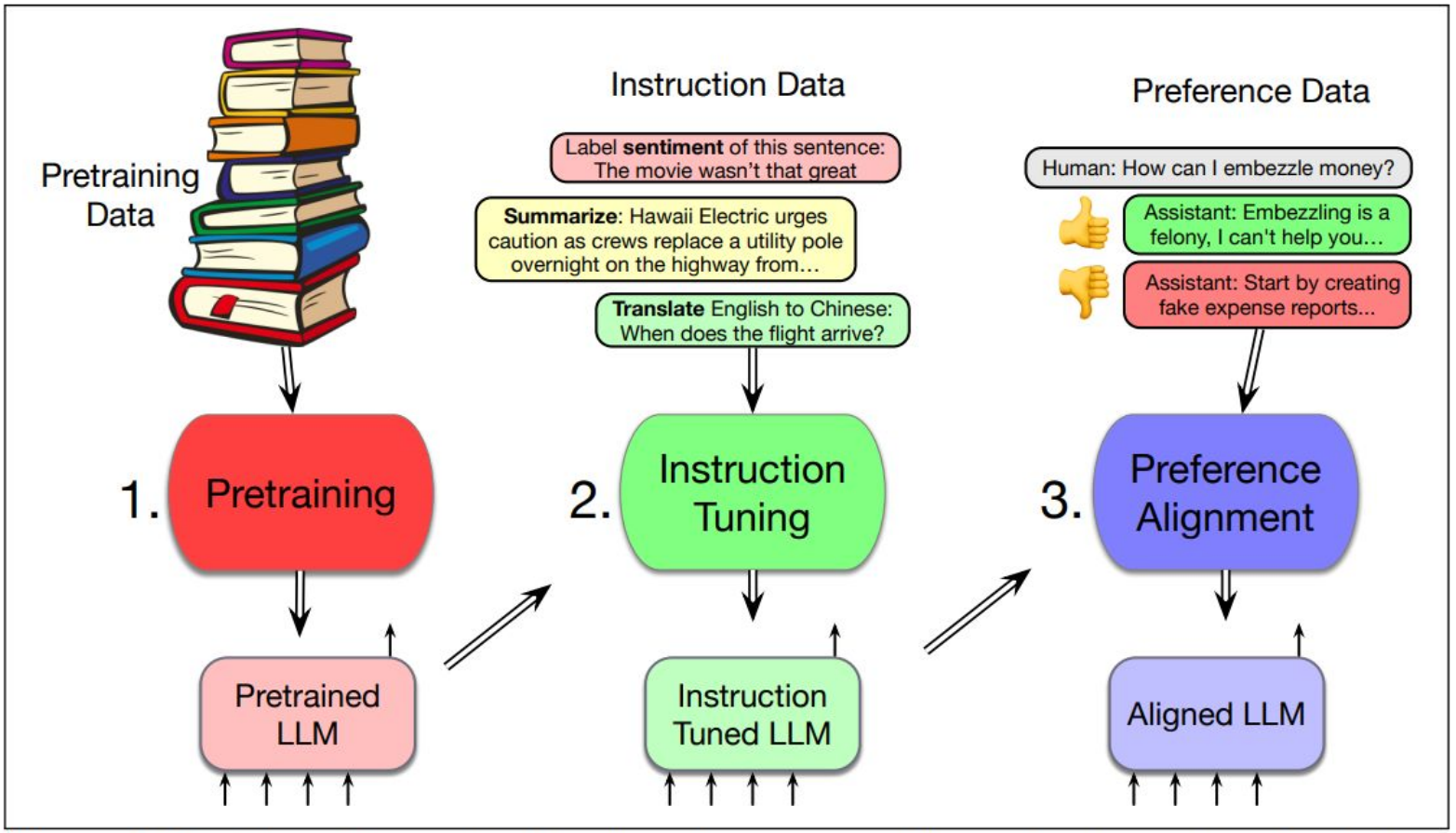


Figure 7.12 Three stages of training large language models: pretraining, instruction tuning, and preference alignment.

Pretraining

- Objetivo: predecir la siguiente palabra.

- Calcular la pérdida (Cross Entropy Loss) sobre textos de entrenamiento

- Autosupervisado!

$$L_{CE} = -\sum_{w \in V} \mathbf{y}_t[w] \log \hat{\mathbf{y}}_t[w]$$

Recordar:

- -log(p)) es la medida de la información de un evento (cuanto más raro el evento, más información tenemos)
- La entropía cruzada calcula la suma ponderada de la información de cada evento, según su frecuencia de aparición
- Es mínima cuando y y ŷ son iguales

Pretraining

- Objetivo: predecir la siguiente palabra.

- Calcular la pérdida (Cross Entropy Loss) sobre textos de entrenamiento

- Autosupervisado!

$$L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}]$$

$$L_{CE}(\text{batch of length T}) = \frac{1}{T}\sum_{t=1}^{T} -\log \hat{\mathbf{y}}_t[w_t]$$

Recordar:

- Nuestros ejemplos de entrenamiento son one-hot!
- En cada caso computamos de nuevo ŷ
- Usamos la palabra original, no la predicha (teacher forcing)

Pretraining

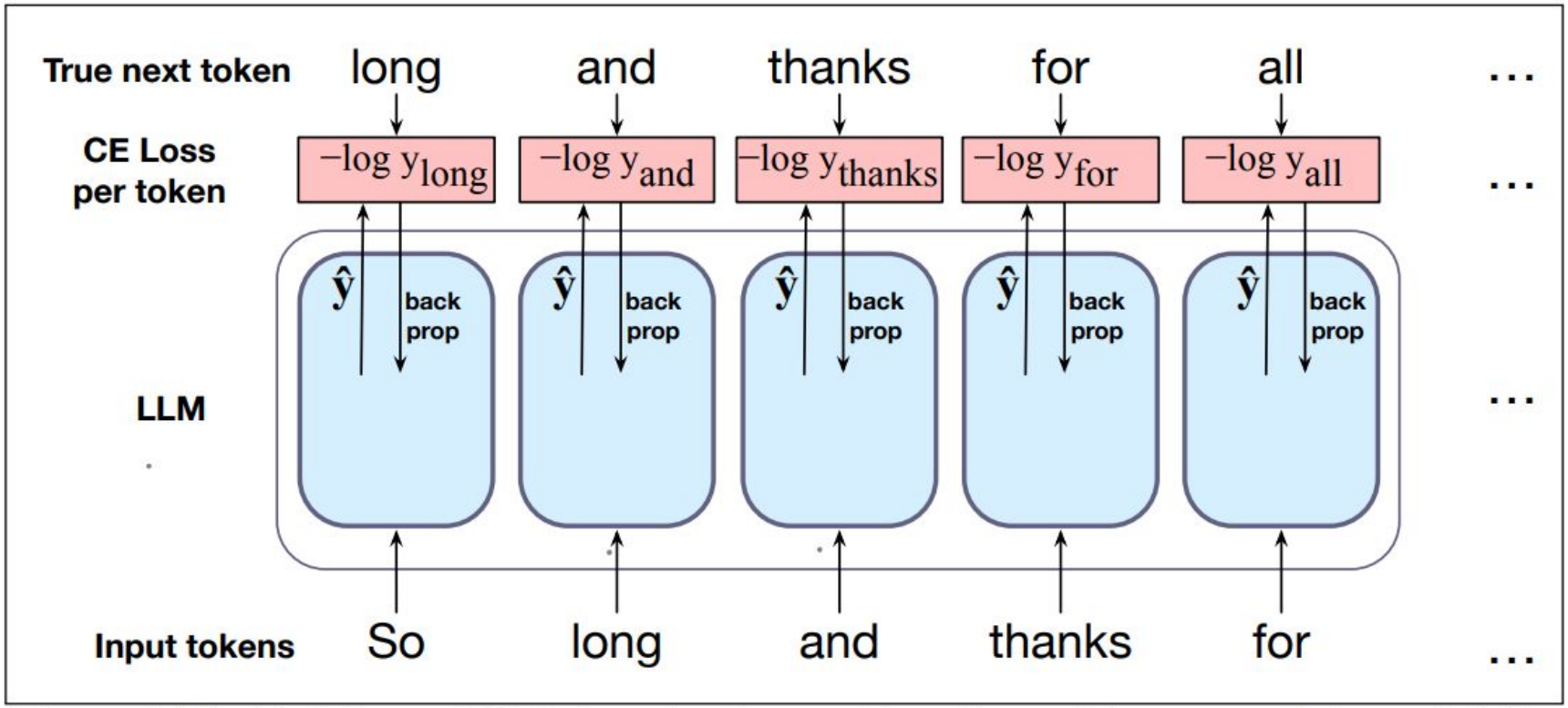
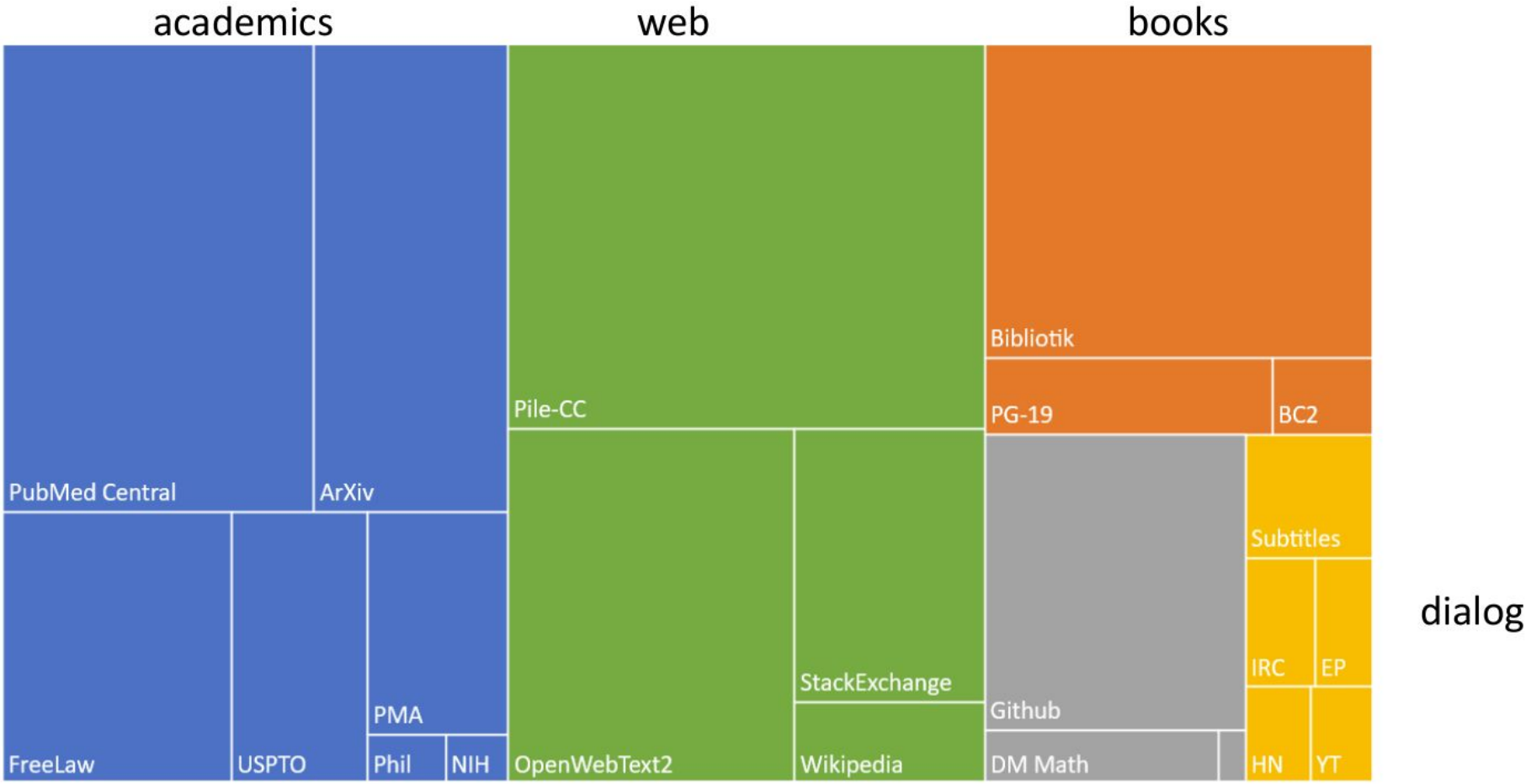


Figure 7.13 Training an LLM. At each token position, the model passes up $\hat{\mathbf{y}}$, its probability estimate for all possible next words. The negative log of the model's probability estimate for the correct token is used as the loss, which is then backpropagated through the model to train all the weights, including the embeddings. Losses are averaged over all the tokens in a batch.

Preentrenamiento: Corpora



The Pile

Preentrenamiento: Corpora

Problemas

- Copyright

- Consentimiento

- Privacidad

- Sesgo

Finetuning

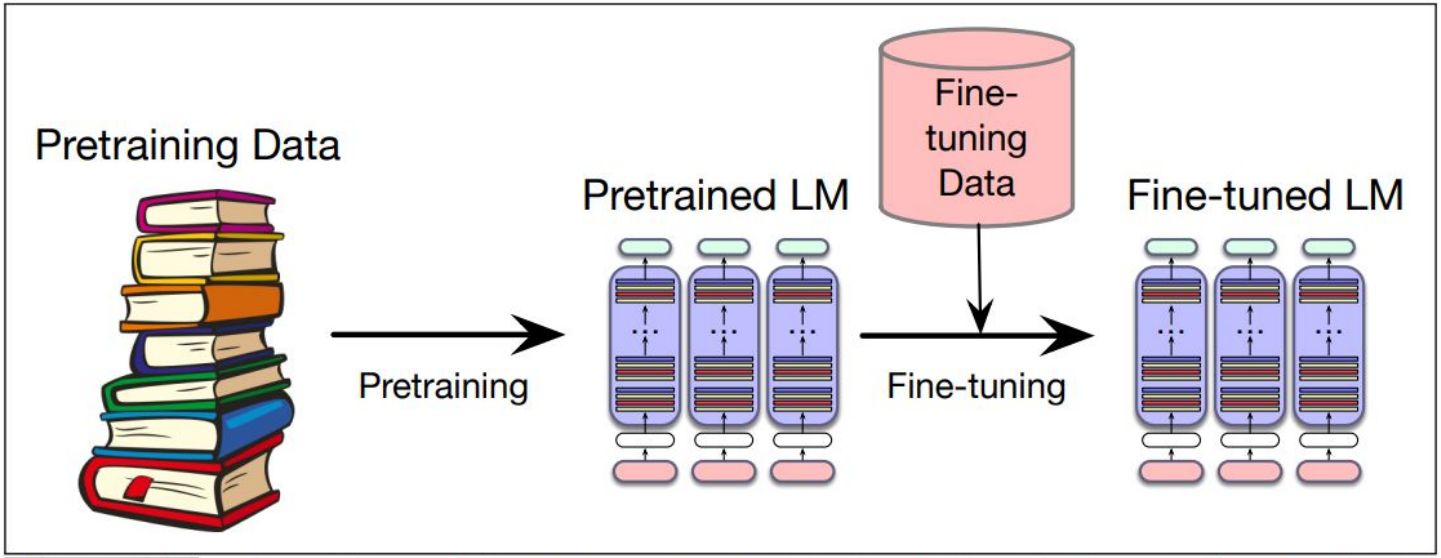


Figure 7.15 Pretraining and finetuning. A pre-trained model can be finetuned to a particular domain or dataset. There are many different ways to finetune, depending on exactly which parameters are updated from the finetuning data: all the parameters, some of the parameters, or only the parameters of specific extra circuitry, as we'll see in future chapters.