



# Redes Neuronales para Lenguaje Natural

2023

Grupo de Procesamiento de Lenguaje Natural  
Instituto de Computación

# Bibliografía

- Jurafsky & Martin, 3rd Ed. (draft) - Capítulos 9, 13 y 2
- Tutoriales sobre mecanismos atencionales de Stefania Cristina  
<https://machinelearningmastery.com/>
- Clase de Traducción Automática de IntroPLN
- Papers...



# RNN para tareas de PLN

# Tipos de problemas

1 palabra (o un par)  $\rightarrow$  1 categoría

*frío y caliente* son sinónimos?

antónimos?

**MLP**

n palabras  $\rightarrow$  1 categoría (k posibles)

este tweet tiene sentimiento positivo?

este tweet es un chiste?

**MLP, CNN, RNN**

n palabras  $\rightarrow$  0..k categorías

de qué temas habla este texto?

qué emociones presenta este tweet?

**MLP, CNN, RNN**

n palabras  $\rightarrow$  n categorías

POS-tagging, NER,

chunking, parsing

**CNN, RNN**

n palabras  $\rightarrow$  m palabras

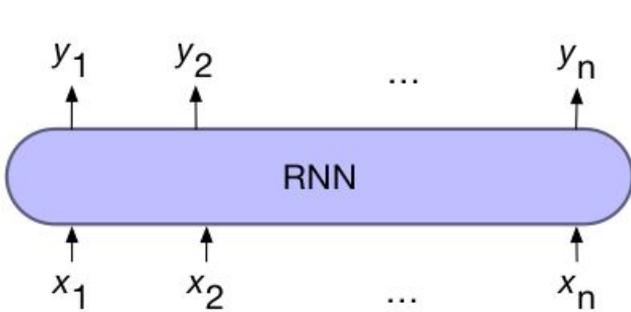
traducción automática

respuestas a preguntas

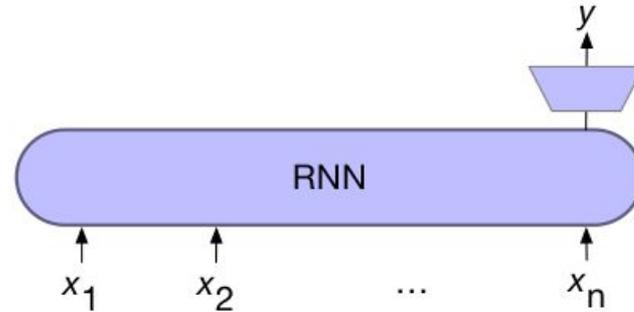
resúmenes automáticos

**Encoder-Decoder (RNN, Transformer)**

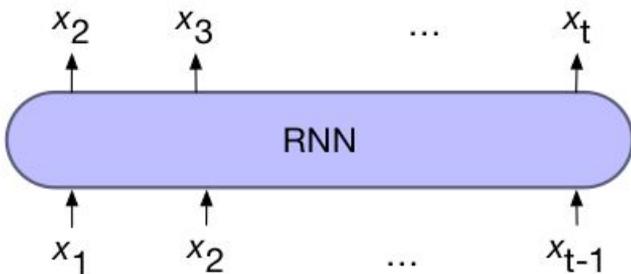
# Usos de las redes recurrentes



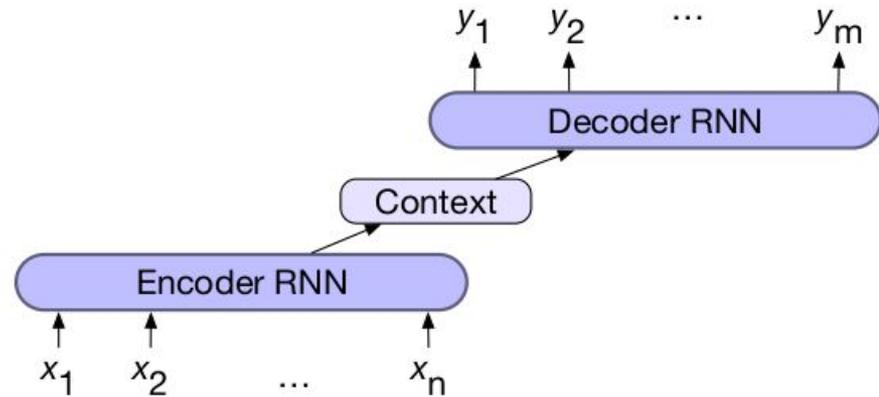
a) etiquetado secuencial



b) clasificación de secuencias

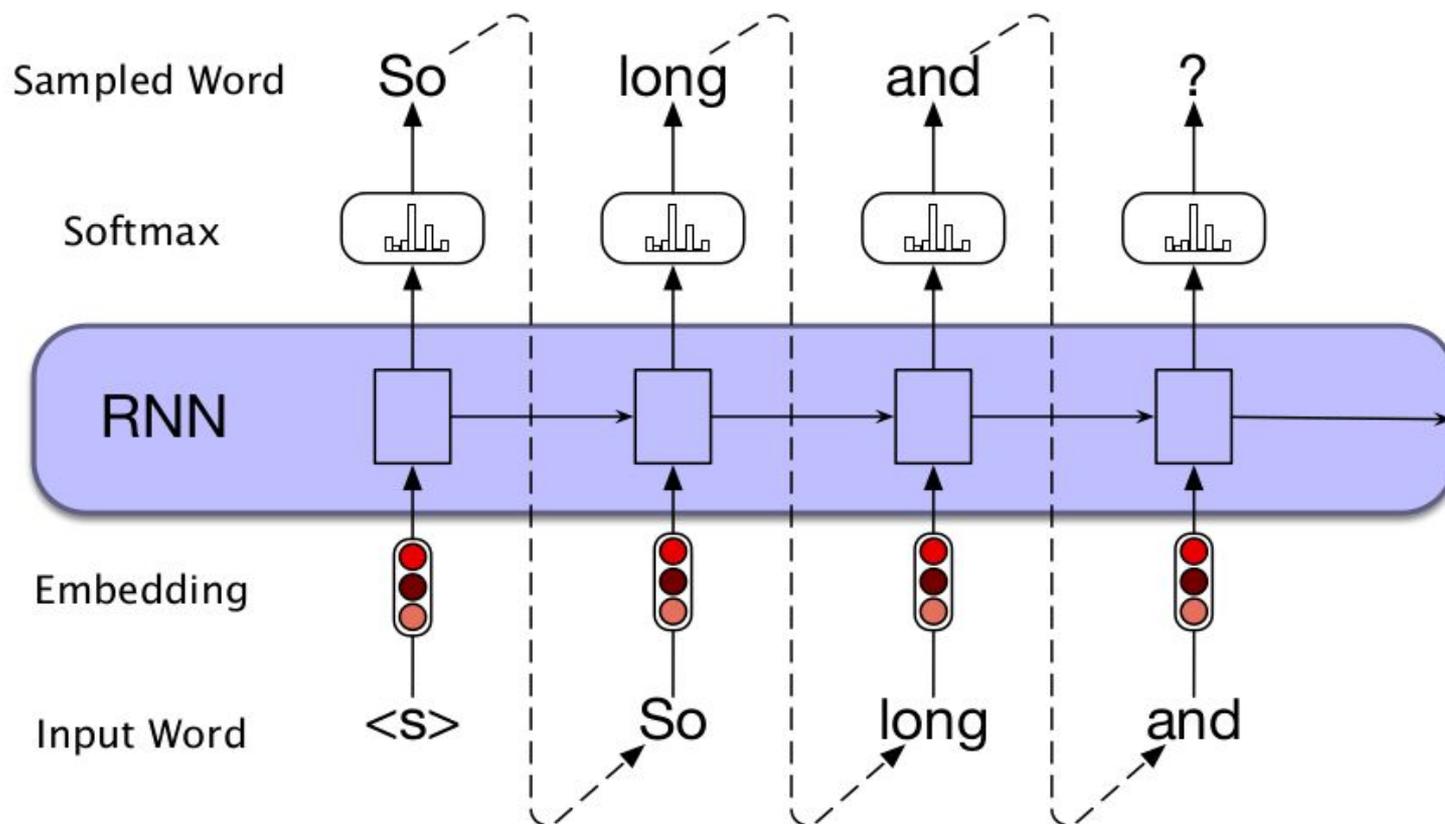


c) modelos de lenguaje



d) modelo encoder-decoder

# RNN para generación de secuencias



Modelo de generación *autorregresivo*, o generación por modelo de lenguaje causal

Sirve para resolver otras tareas, cambiando el estado inicial del generador



# Arquitectura Encoder-Decoder

# Arquitectura Encoder-Decoder

Toma una secuencia de entrada

Devuelve una secuencia de salida

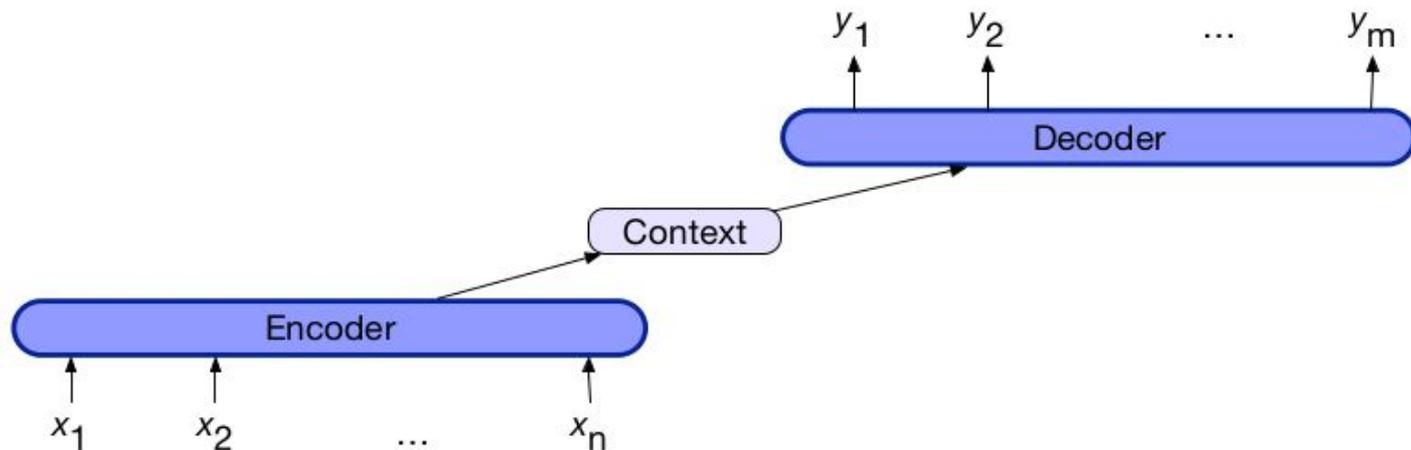
La cantidad de elementos de entrada y de salida pueden ser diferentes

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  $y_1^m$

# Modelo de lenguaje autorregresivo

Queremos generar la secuencia de salida  $y = y_1^m$

$$p(y) = p(y_1)p(y_2|y_1)p(y_3|y_1, y_2)\dots p(y_m|y_1, \dots, y_{m-1})$$

Los estados ocultos  $h_t$  de una red se calculan usando el estado oculto en el paso anterior, y el nuevo token en el tiempo  $t$

$$h_t = g(h_{t-1}, x_t)$$

La salida en el paso  $t$  se obtiene a partir del estado oculto  $h_t$

$$y_t = f(h_t)$$

# Generar salida a partir de una entrada

Una forma de modelarlo: consideremos que estamos haciendo un modelo autorregresivo sobre las secuencias origen y destino separados por un token especial <s>

$$x_1 x_2 \dots x_n \text{ <s> } y_1 y_2 \dots y_m$$

Sea  $x = x_1^n \text{ <s>}$ , entonces:

$$p(y|x) = p(y_1|x)p(y_2|y_1, x)p(y_3|y_1, y_2, x)\dots p(y_m|y_1, \dots, y_{m-1}, x)$$

Es el modelo autorregresivo con un generador condicionado al estado oculto de la codificación de  $x$  (secuencia de entrada)

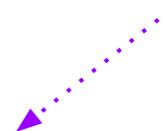
# Generar salida a partir de una entrada

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

En nuestra red usaremos la secuencia:

*the green witch arrived <s> llegó la bruja verde*

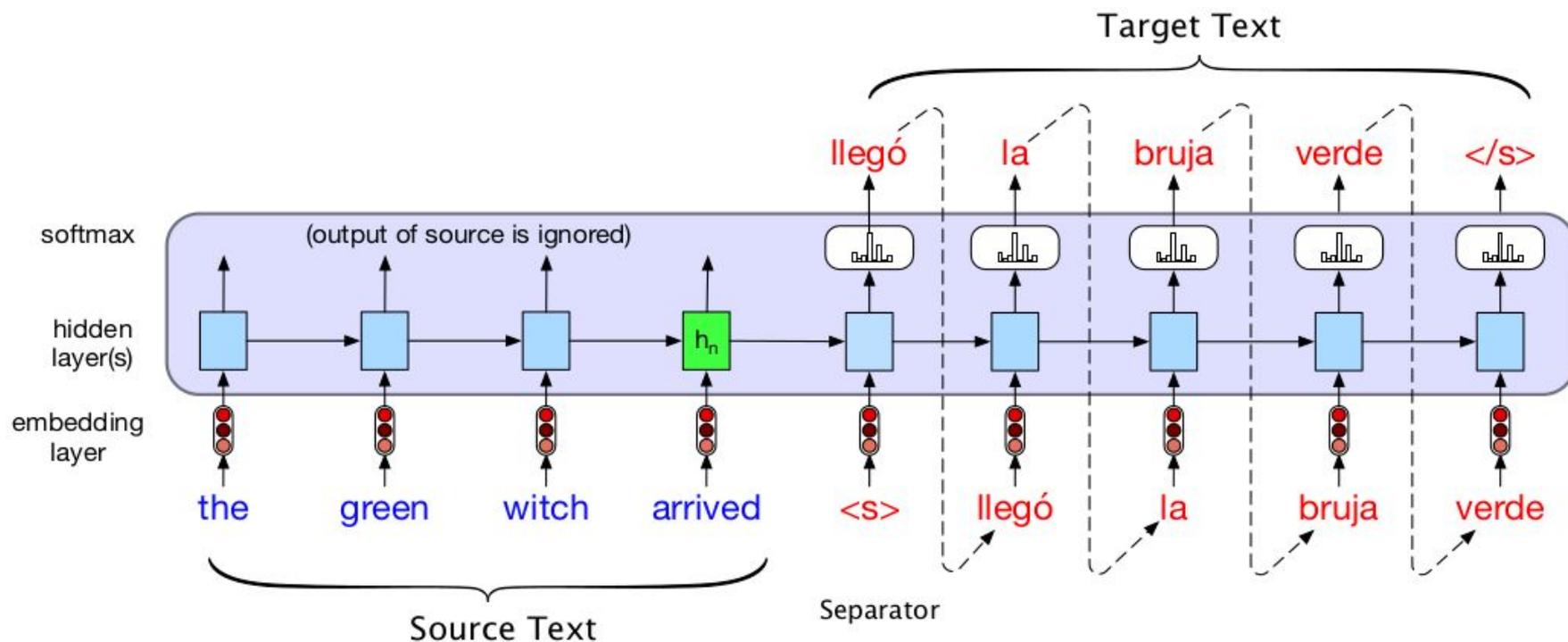
(salida esperada)



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

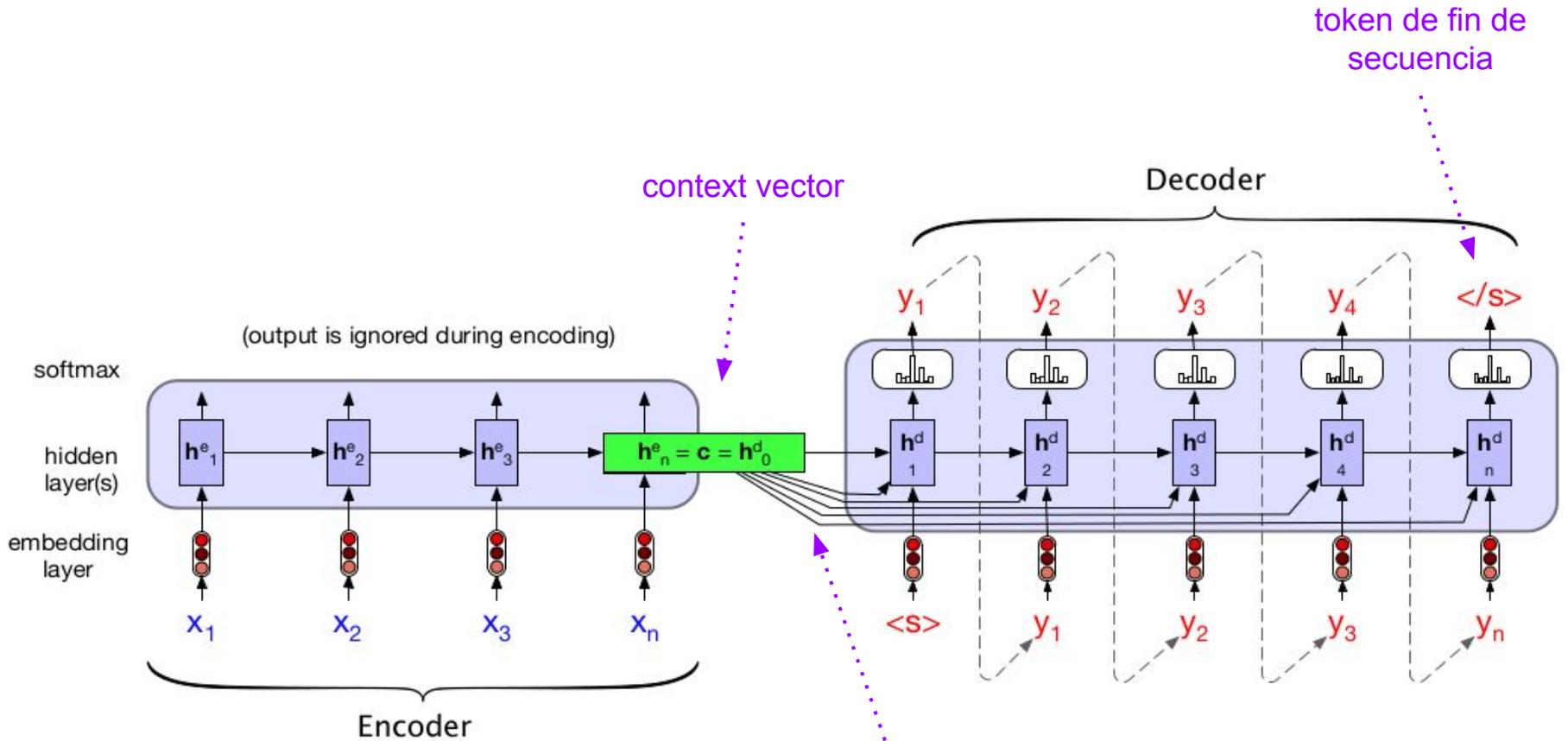
Formalizaremos la idea como composición de dos subredes

El encoder genera los estados ocultos  $h^e$ , el decoder los estados ocultos  $h^d$

El último estado oculto del encoder será el vector de contexto, que además es el vector inicial con que se alimenta el decoder

$$h_n^e = c = h_0^d$$

# 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

Codificación de la secuencia de entrada  $c = h_n^e$

Estado inicial del decodificador  $h_0^d = c$

Cada paso del decoder usa el estado oculto anterior, el token decodificado y el contexto  $h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d, c)$

El estado oculto se pasa por una nueva red feed forward y se obtiene el siguiente token  $z_t = f(h_t^d)$

con un softmax  $y_t = \text{softmax}(z_t)$

Cómo computar la siguiente salida:

$$\hat{y}_t = \operatorname{argmax}_{w \in V} P(w|x, y_1 \dots y_{t-1})$$

# Encoder-Decoder: entrenamiento

Se entrenan con pares de secuencias (entrada - salida)

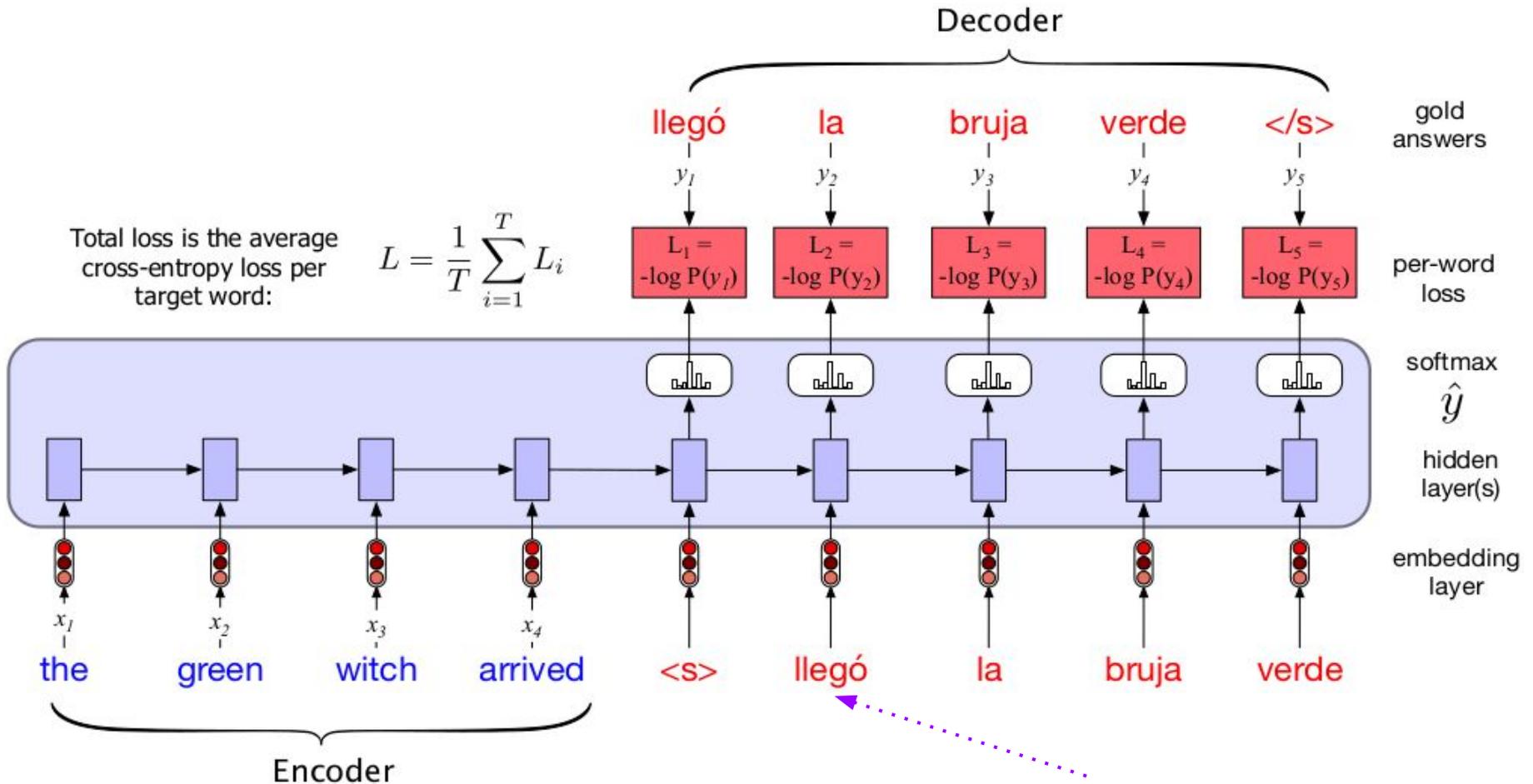
Por ejemplo, corpus paralelos en traducción automática: oraciones origen en inglés y oraciones destino en español

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

# Encoder-Decoder: entrenamiento



*teacher forcing*: independientemente de lo que se haya obtenido como la salida en el paso  $y_{i-1}$ , se usa el embedding de la palabra correcta para entrenamiento

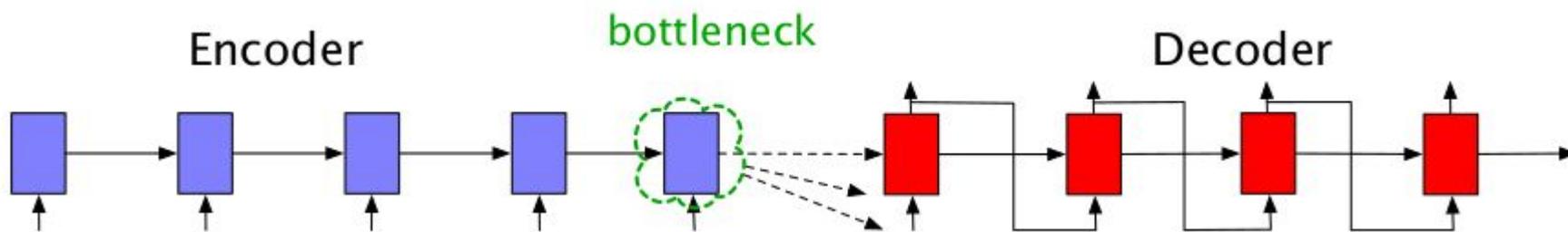


# Mecanismo Atencional

# Problema del vector de contexto

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

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?

$$c_i = f(h_1^e \dots h_n^e)$$

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  $h_1^e \dots h_n^e$

# 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  $h_j^e$  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  $(h_{i-1}^d, h_j^e)$  el esquema atencional que usaremos definirá cómo se calcula ese score

# Cálculo de scores

El score que le damos al par  $(h_{i-1}^d, h_j^e)$  representa qué tanta atención se le presta a  $h_j^e$  al generar la salida  $i$

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

$$\text{score}(h_{i-1}^d, h_j^e) = h_{i-1}^d \cdot h_j^e$$

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

# 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  $i$

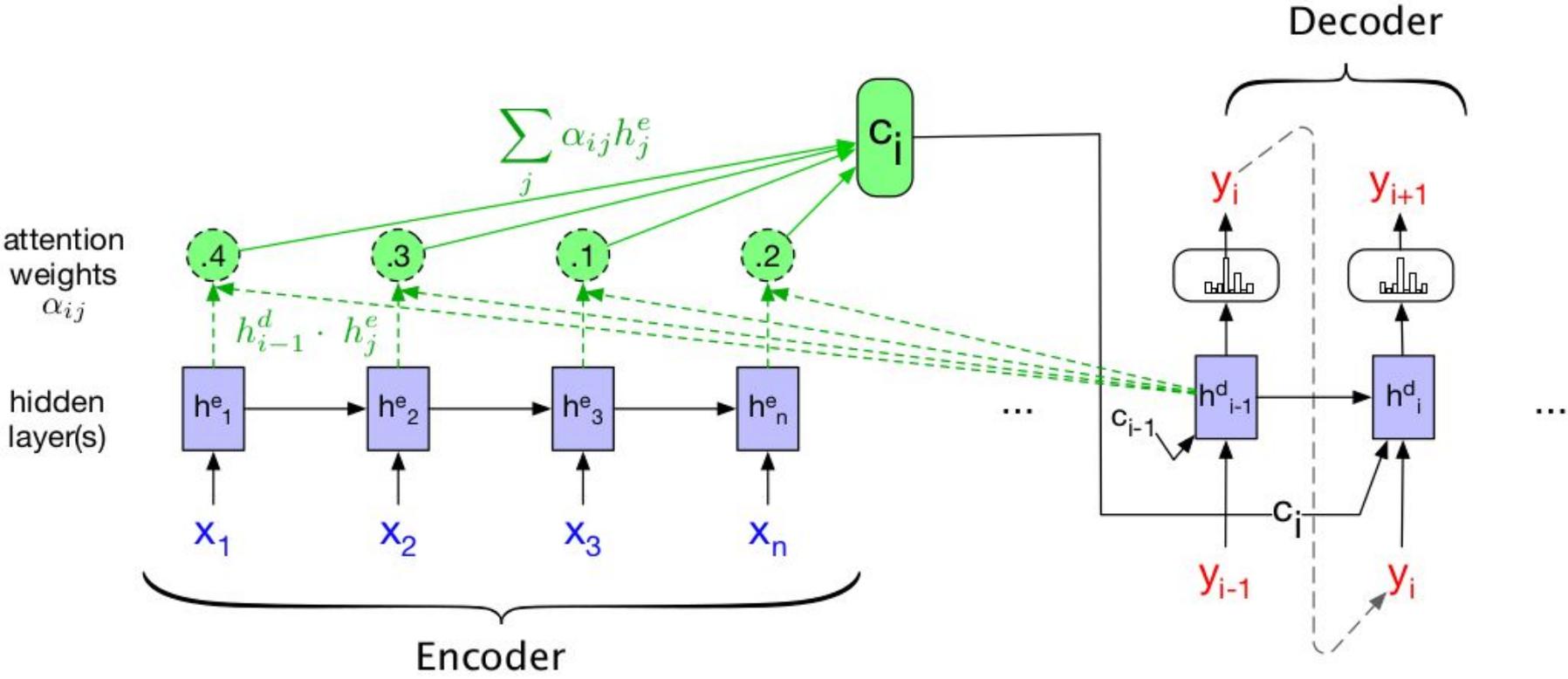
Softmax sobre todos los scores para normalizarlos a pesos

$$\begin{aligned}\alpha_{ij} &= \text{softmax}(\text{score}(h_{i-1}^d, h_j^e)) \\ &= \frac{\exp(\text{score}(h_{i-1}^d, h_j^e))}{\sum_k \exp(\text{score}(h_{i-1}^d, h_k^e))}\end{aligned}$$

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

$$c_i = \sum_j \alpha_{ij} h_j^e$$

# Mecanismo Atencional



# Otros scores

La *dot product attention* es simple, pero no permite ningún grado de expresividad extra porque no tiene parámetros

Se han desarrollado otros métodos de atención paramétricos que tienen mejores resultados

- Atención aditiva (Atención de Bahdanau)
- Atención multiplicativa (Atención de Luong)

# Atención aditiva

Modela la función de score como una red feedforward

Aparecen parámetros nuevos:

- Matriz  $W$  que opera sobre la concatenación de los estados ocultos
- Vector  $v$  de pesos para definir la salida final

$$\text{score}(h_{i-1}^d, h_j^e) = v^T \tanh(W[h_j^e; h_{i-1}^d])$$

Luego sigue igual: hacemos softmax sobre los scores para cada token para obtener pesos y calculamos el vector de contexto

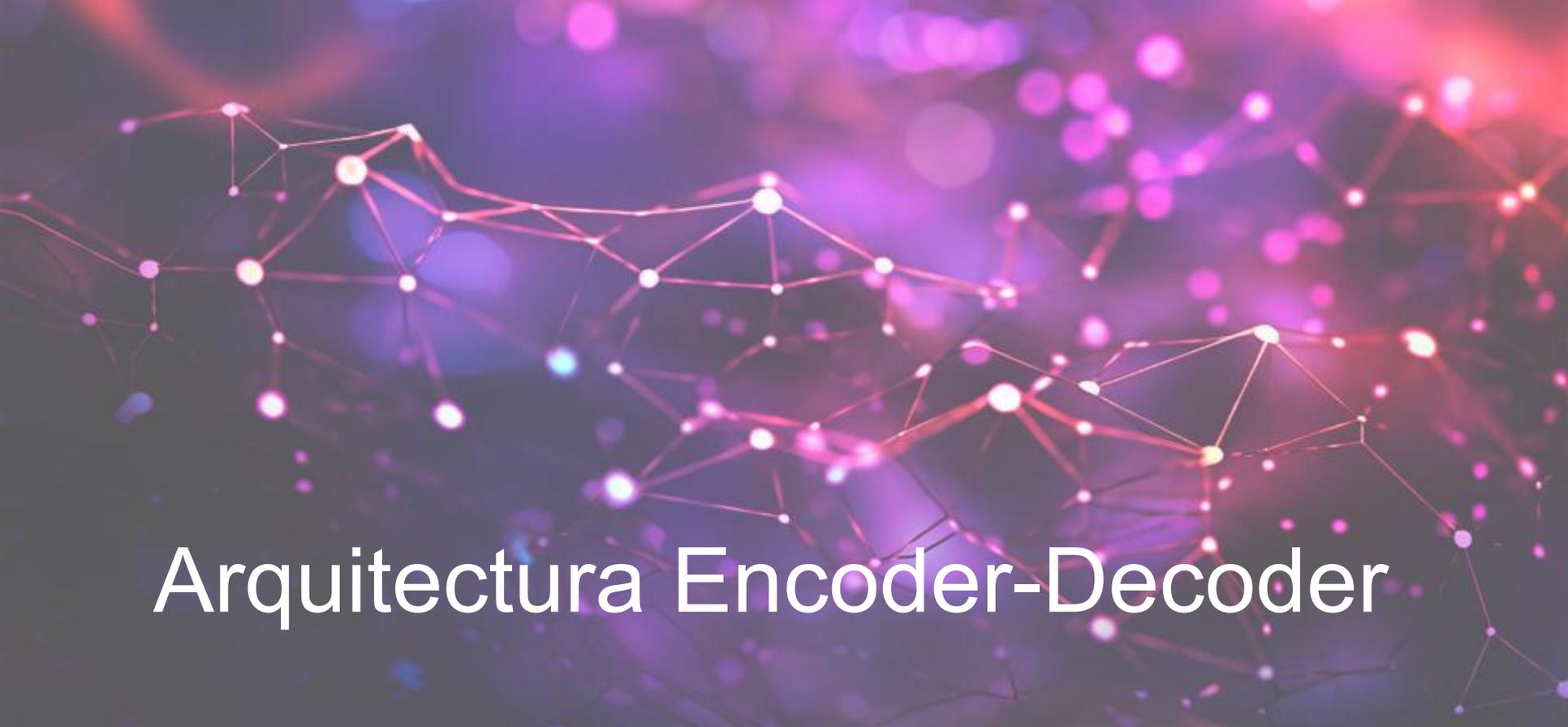
# Atención multiplicativa

En vez de modelar como red feedforward y usar una matriz y un vector de pesos, utiliza una única matriz con producto bilineal

$$\text{score}(h_{i-1}^d, h_j^e) = h_{i-1}^d W h_j^e$$

Es similar a la *dot product attention*, pero agrega la matriz de pesos parametrizables

Luego sigue igual: hacemos softmax sobre los scores para cada token para obtener pesos y calculamos el vector de contexto



# Arquitectura Encoder-Decoder