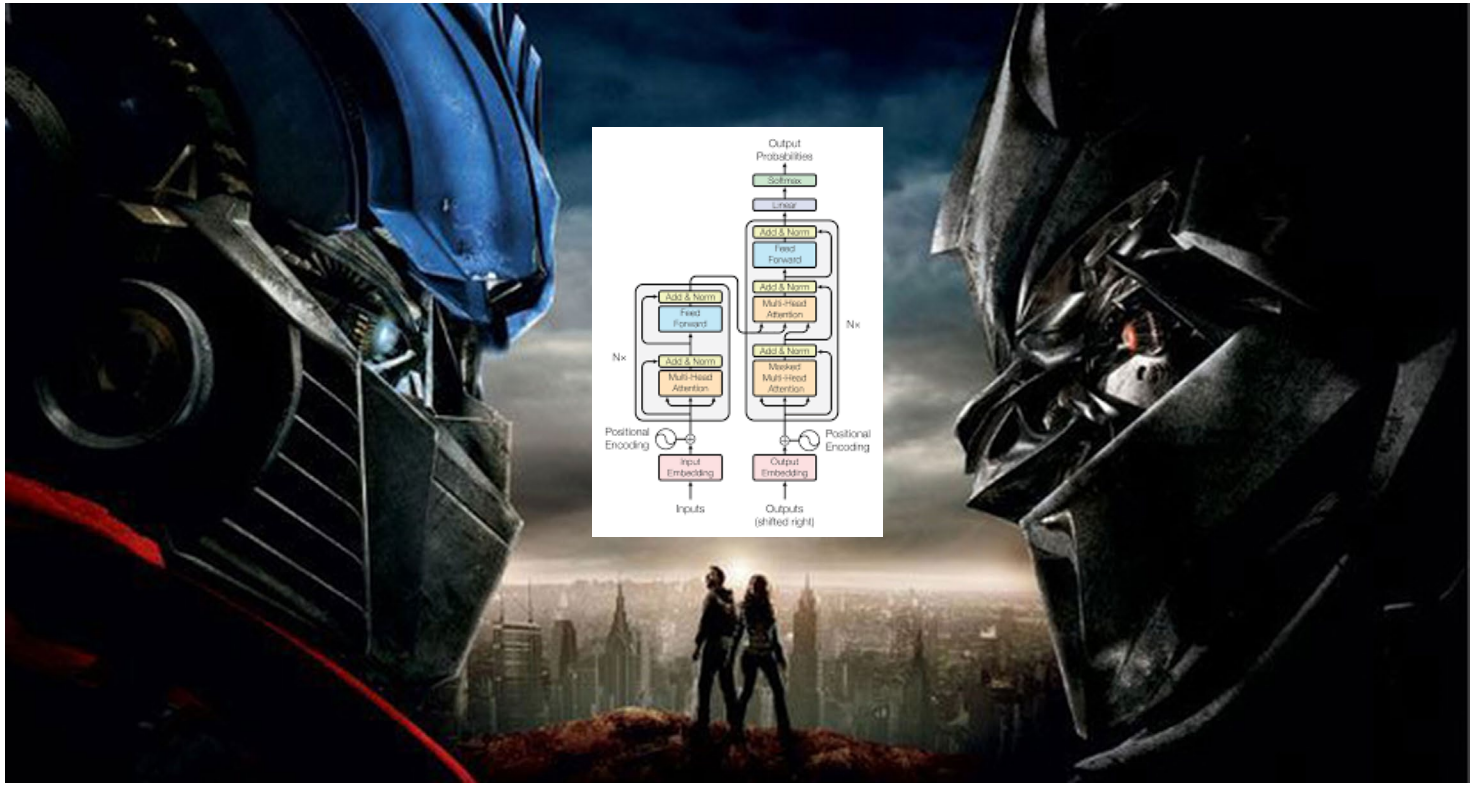


Enc-Dec / Attention / Transformers

Adaptado por: Jonnatan Arias Garcia

Creado por: Giuseppe Carenini

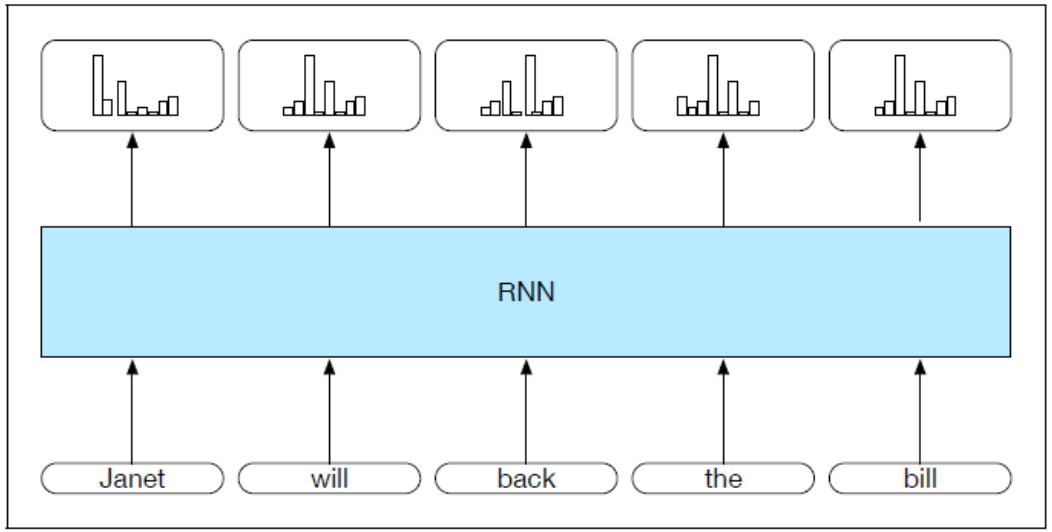
Slides Sources: Jurafsky & Martin 3rd Ed / blog https://jalammar.github.io/illustrated-transformer/

Indice

- Encoder- Decoder
- Attention
- Transformers

Encoder-Decoder

- **RNN:** la secuencia de entrada se transforma en secuencia de salida de forma uno a uno.



- **Objetivo:** Desarrollar una arquitectura capaz de generar secuencias de salida de longitud arbitraria y contextualmente apropiadas
- **Aplicaciones**:
  - Traducción automática
  - Resumen
  - Respuesta a preguntas,
  - Modelado de diálogos.

Red neuronal recurrente simple ilustrada como una red de retroalimentación

Cambio más significativo: nuevo juego de pesas, U

Conecte la capa oculta del periodo de tiempo anterior a la capa oculta actual.

Determine cómo la red debe hacer uso del contexto pasado para calcular la salida de la entrada actual.

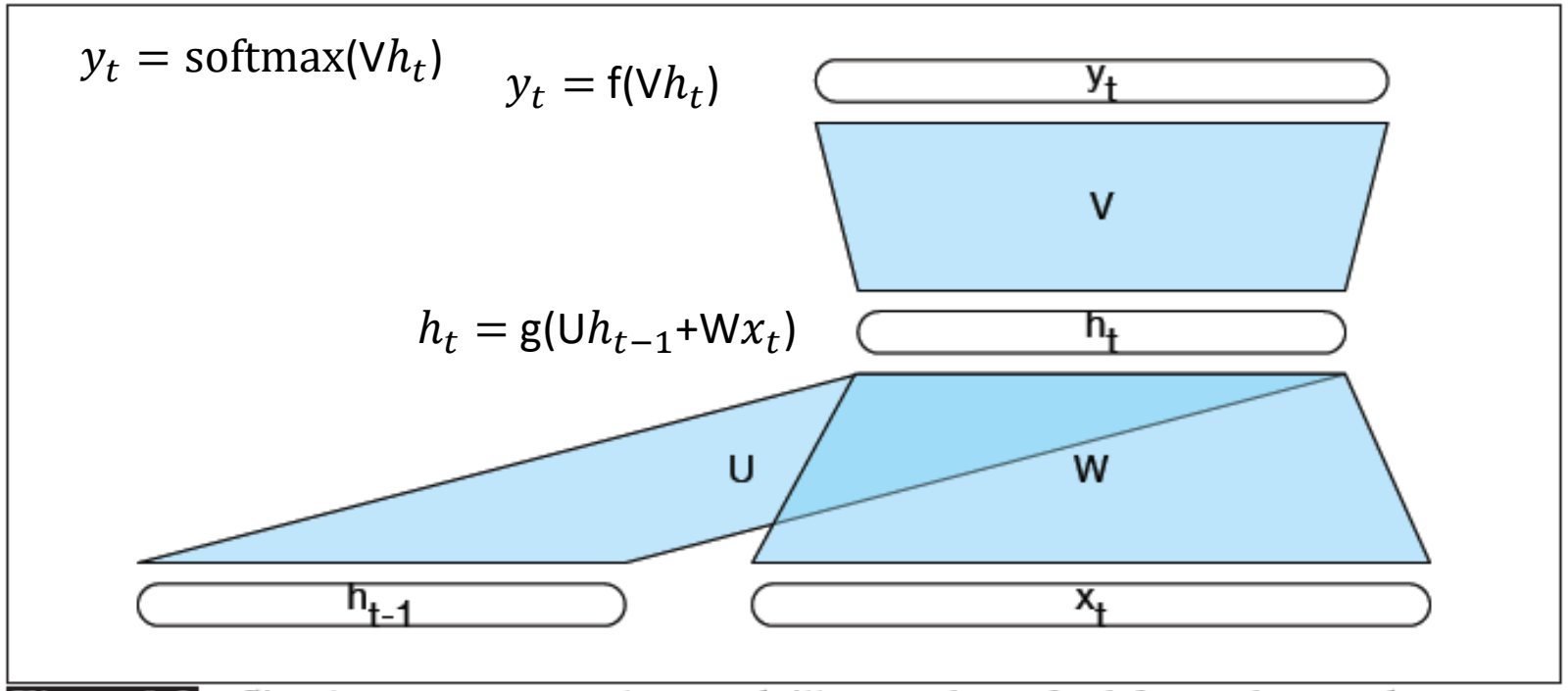
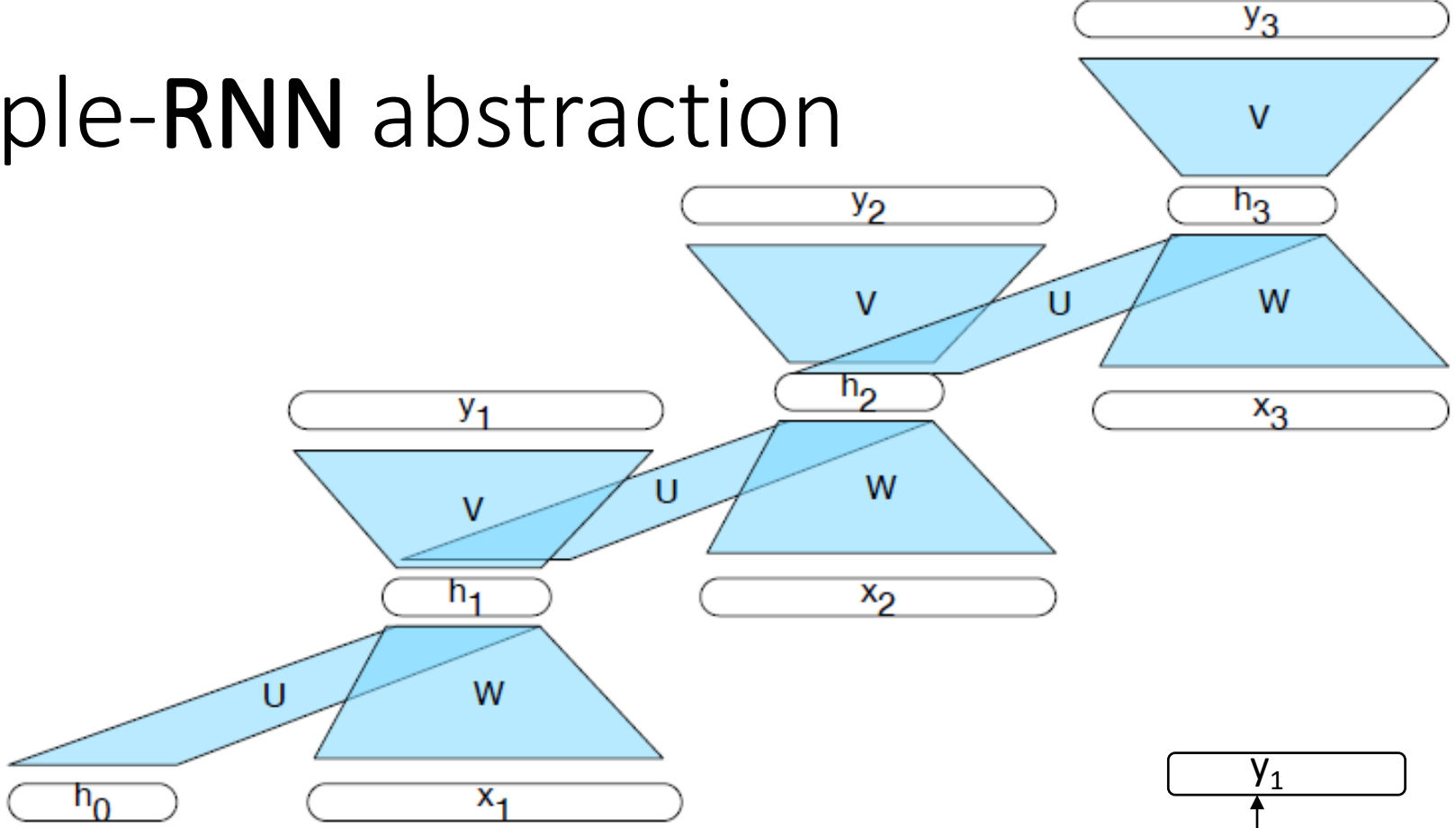
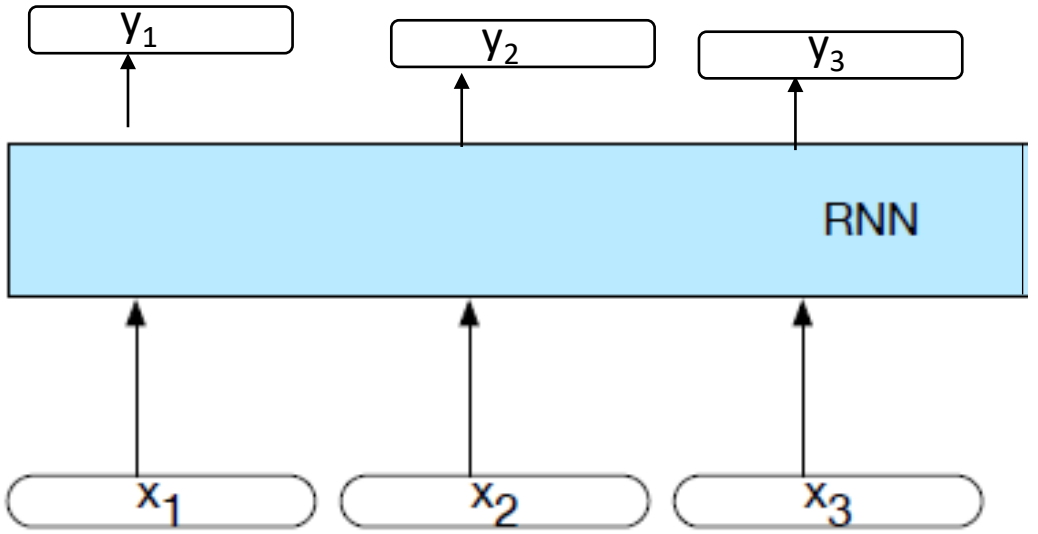


Figure 9.3 Simple recurrent neural network illustrated as a feed-forward network.

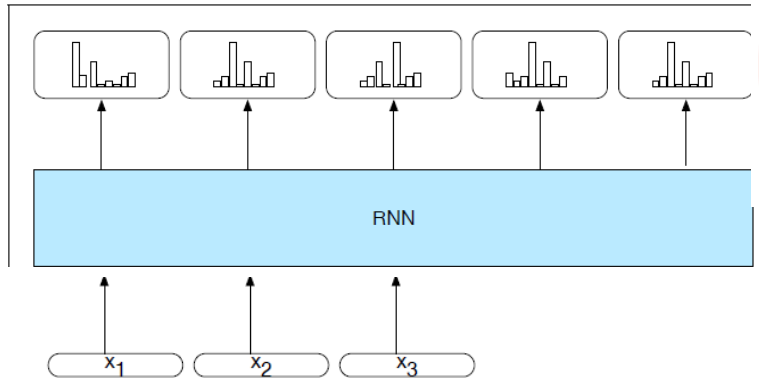
Simple-**RNN** abstraction

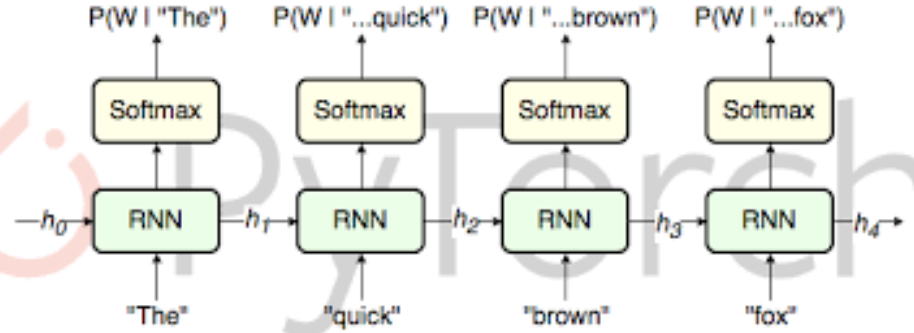




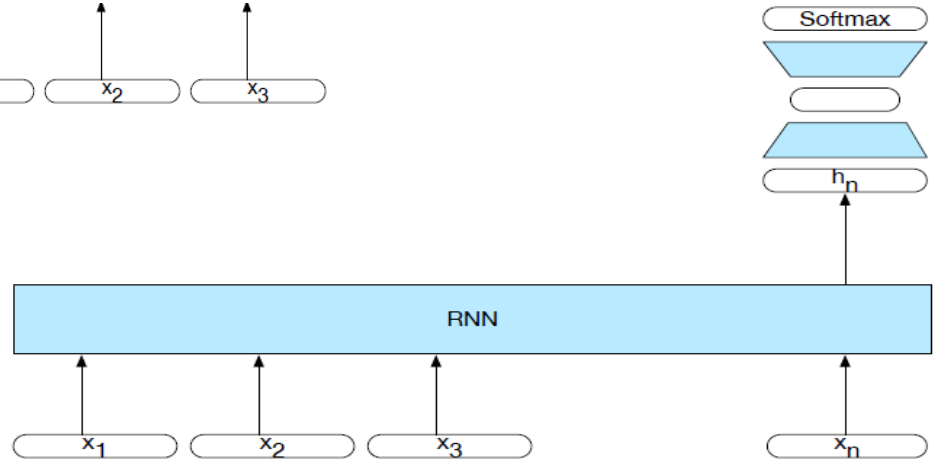
RNN Applications

- Modelado del lenguaj

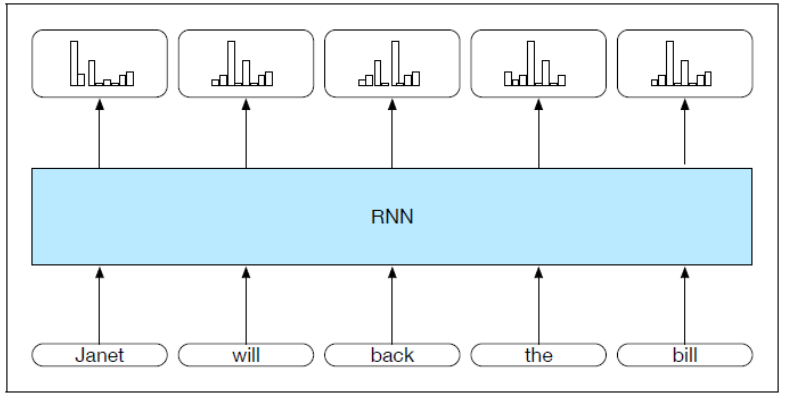




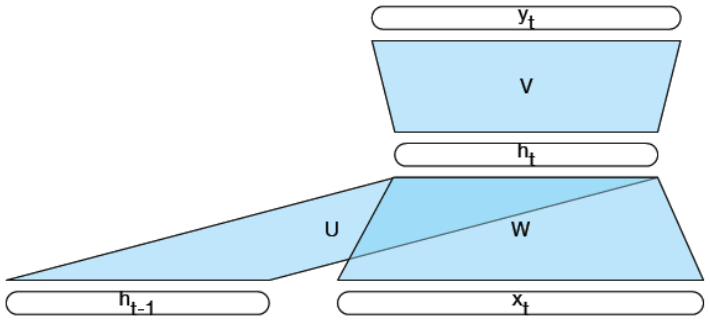
- Clasificación de secuencia (sentimiento, tema)



- Secuencia a secuencia

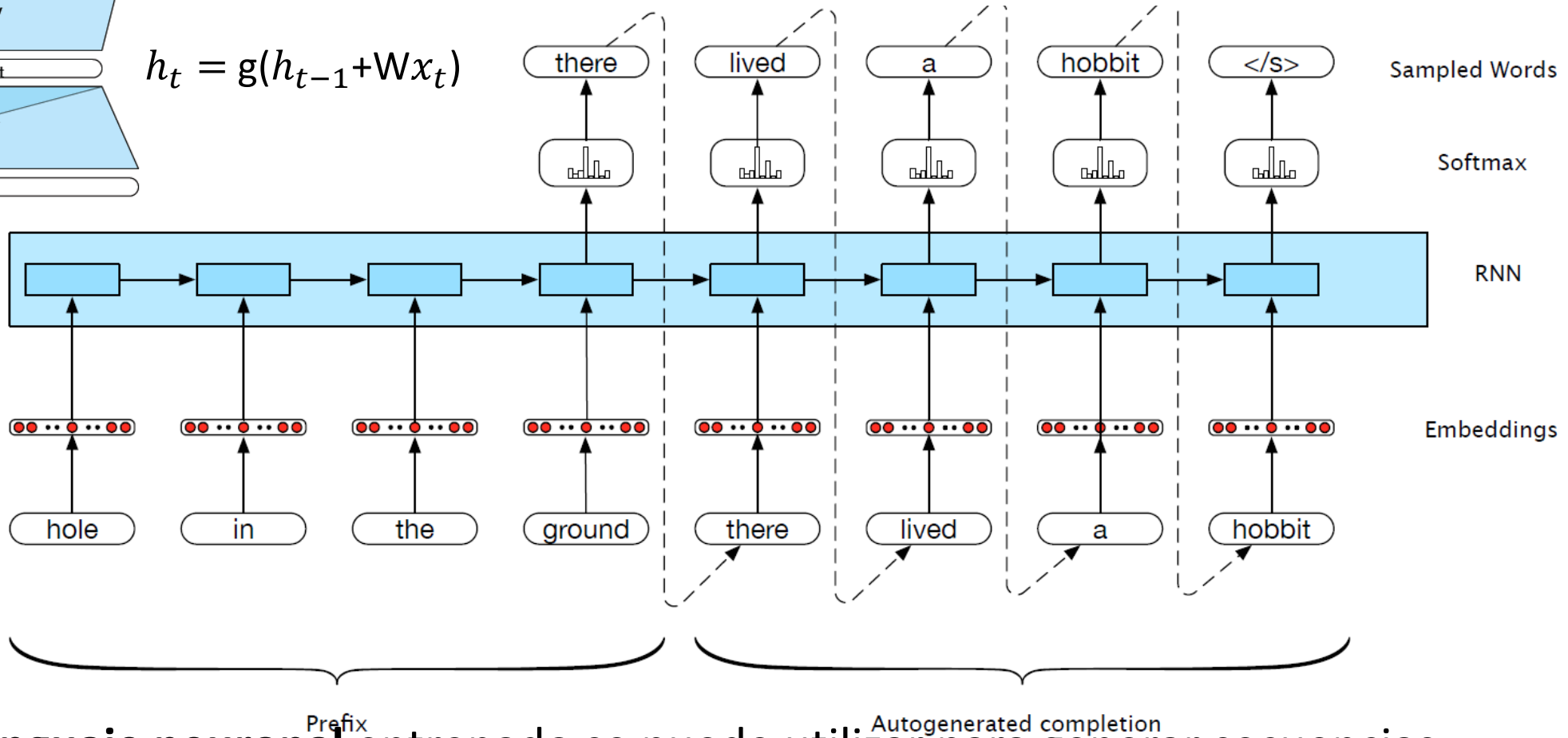


Sentence Completion using an RNN



$$y_t = \text{softmax}(Vh_t)$$

$$h_t = g(h_{t-1} + Wx_t)$$



- **El modelo de lenguaje neuronal** entrenado se puede utilizar para generar secuencias novedosas
- O para **completar** una secuencia determinada (hasta que se genere el token de fin de oración <\s>)

Extensión de la generación (autorregresiva) a la traducción automática

La palabra generada en cada paso de tiempo está condicionada por la palabra del paso anterior.

- Los datos de entrenamiento son texto paralelo. ejemplo:
- English / French

there lived a hobbit *vivait un hobbit*

……..

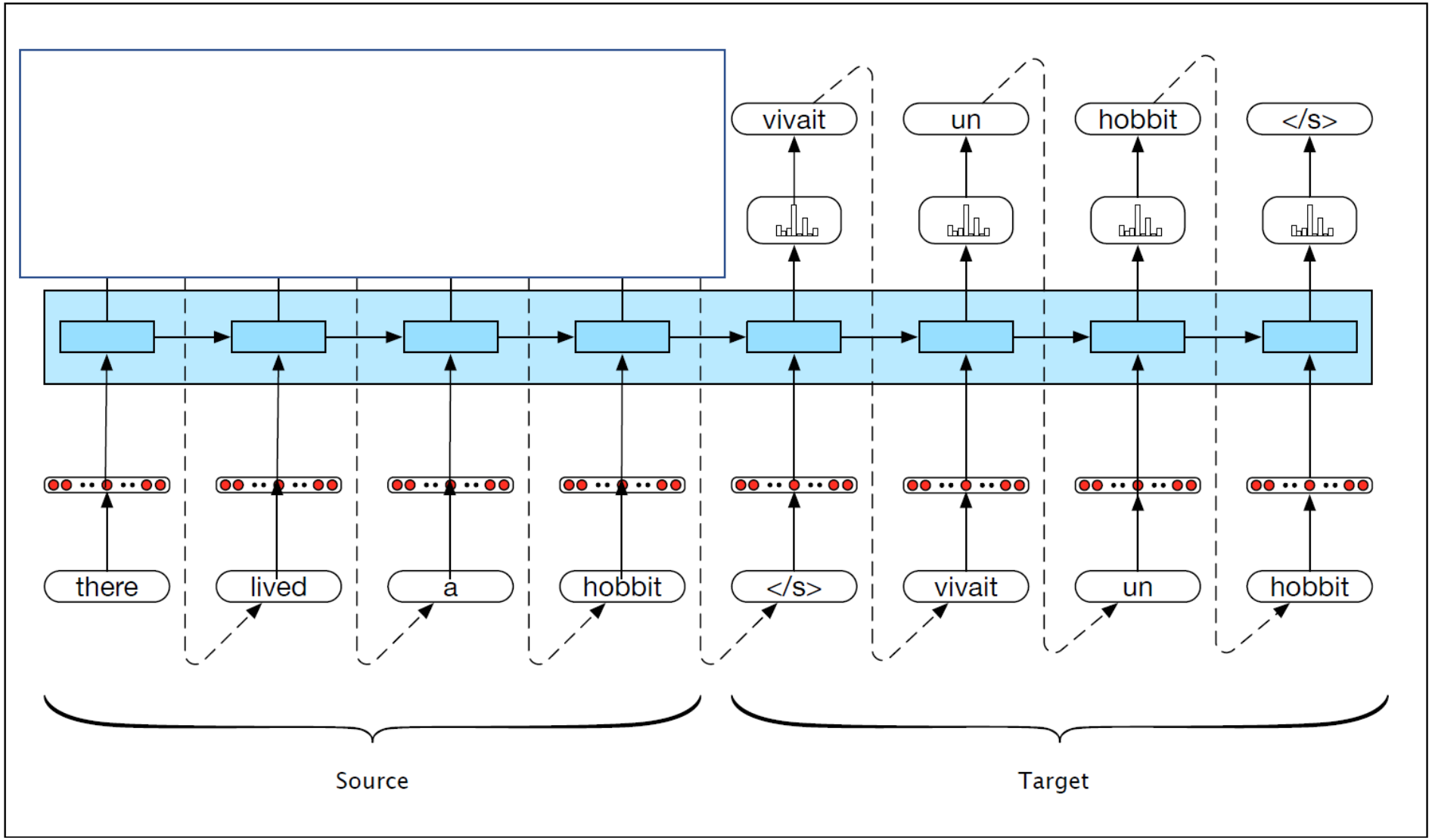
- Cree un modelo de lenguaje RNN en la concatenación de origen y destino

there lived a hobbit <\s> *vivait un hobbit* <\s>

……..

Extensión de la generación (autorregresiva) a la traducción automática

- ¡La traducción como completamiento de frases!



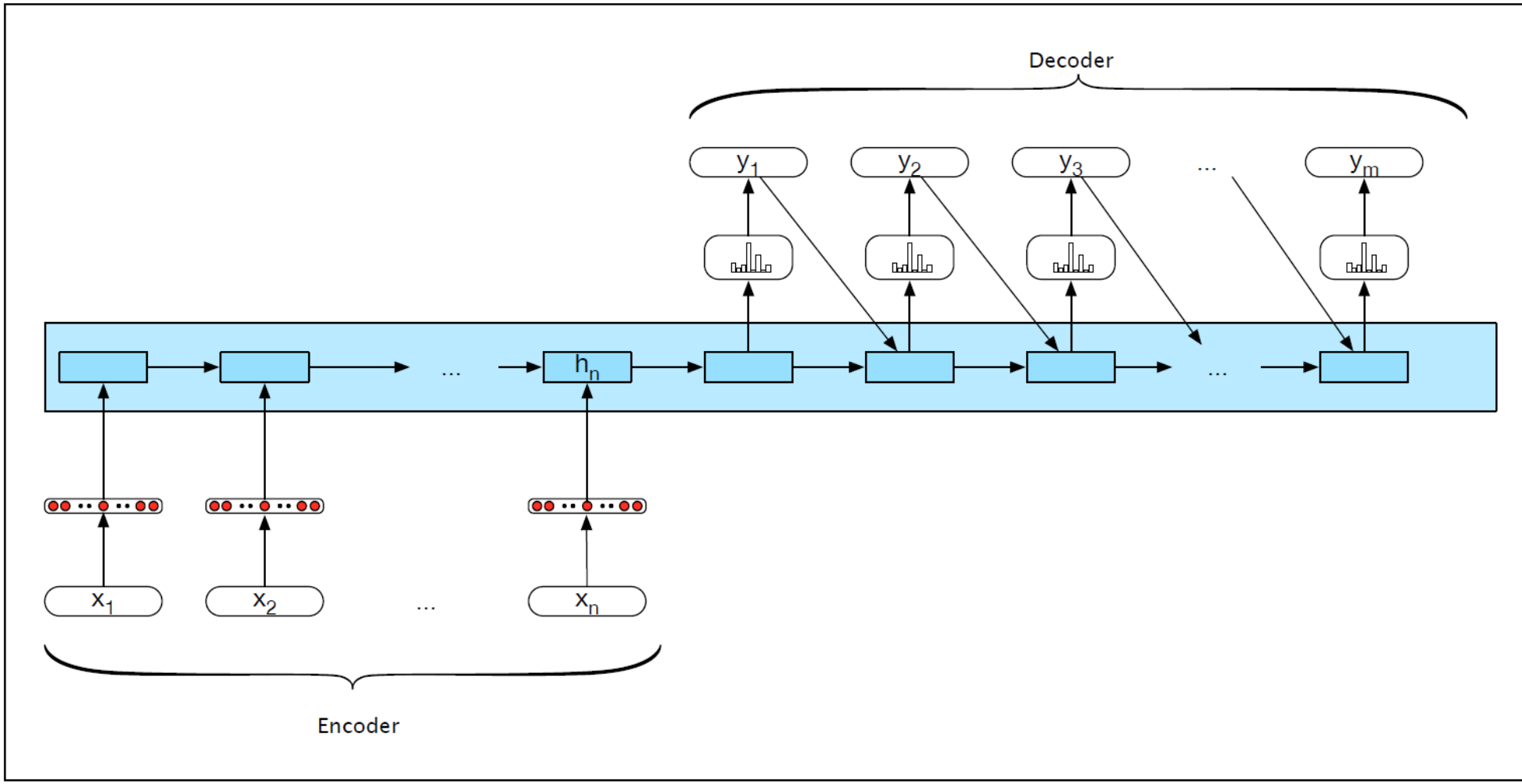
(simple) **E**ncoder **D**ecoder Networks

Limitación de las opciones de diseño

E y **D** se supone que tienen la misma estructura interna (en este caso, RNN)

El estado final de **E** es el único contexto disponible para **D**

este contexto solo está disponible para **D** como su estado oculto inicial.

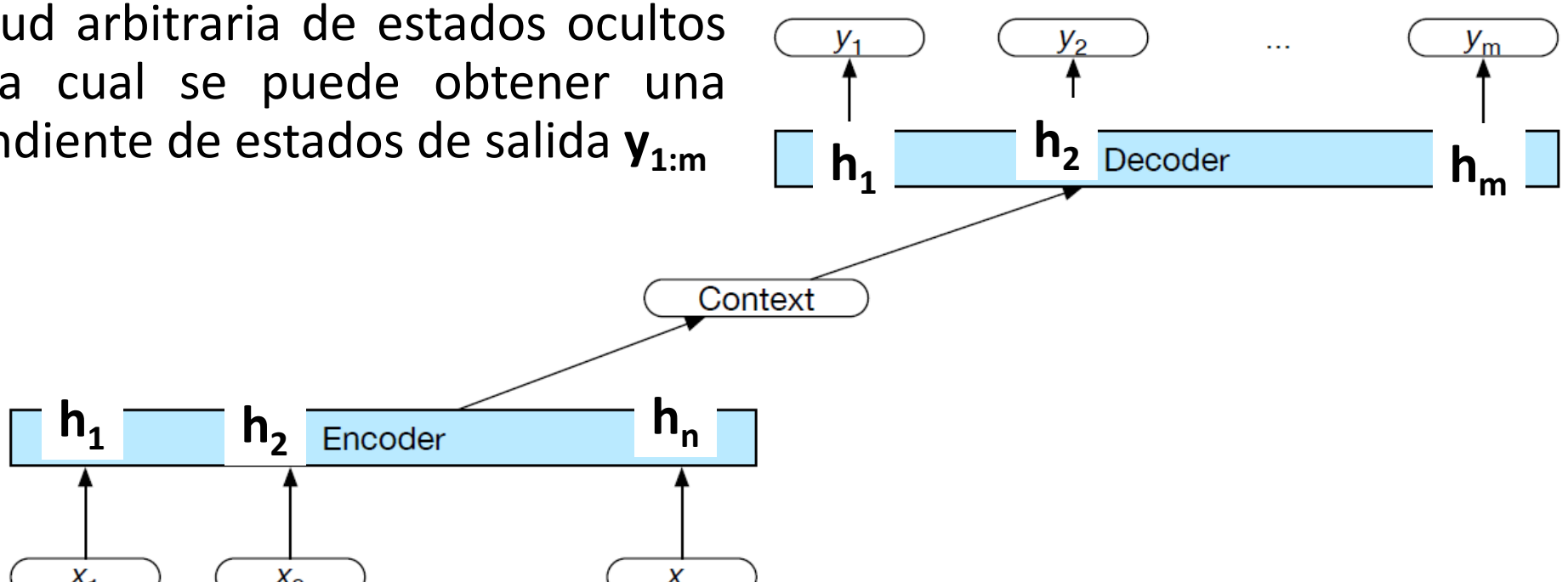


- El codificador genera una representación contextualizada de la entrada (último estado).
- El decodificador toma ese estado y genera autorregresivamente una secuencia de salidas

General Encoder Decoder Networks

Abstraerse de estas opciones:

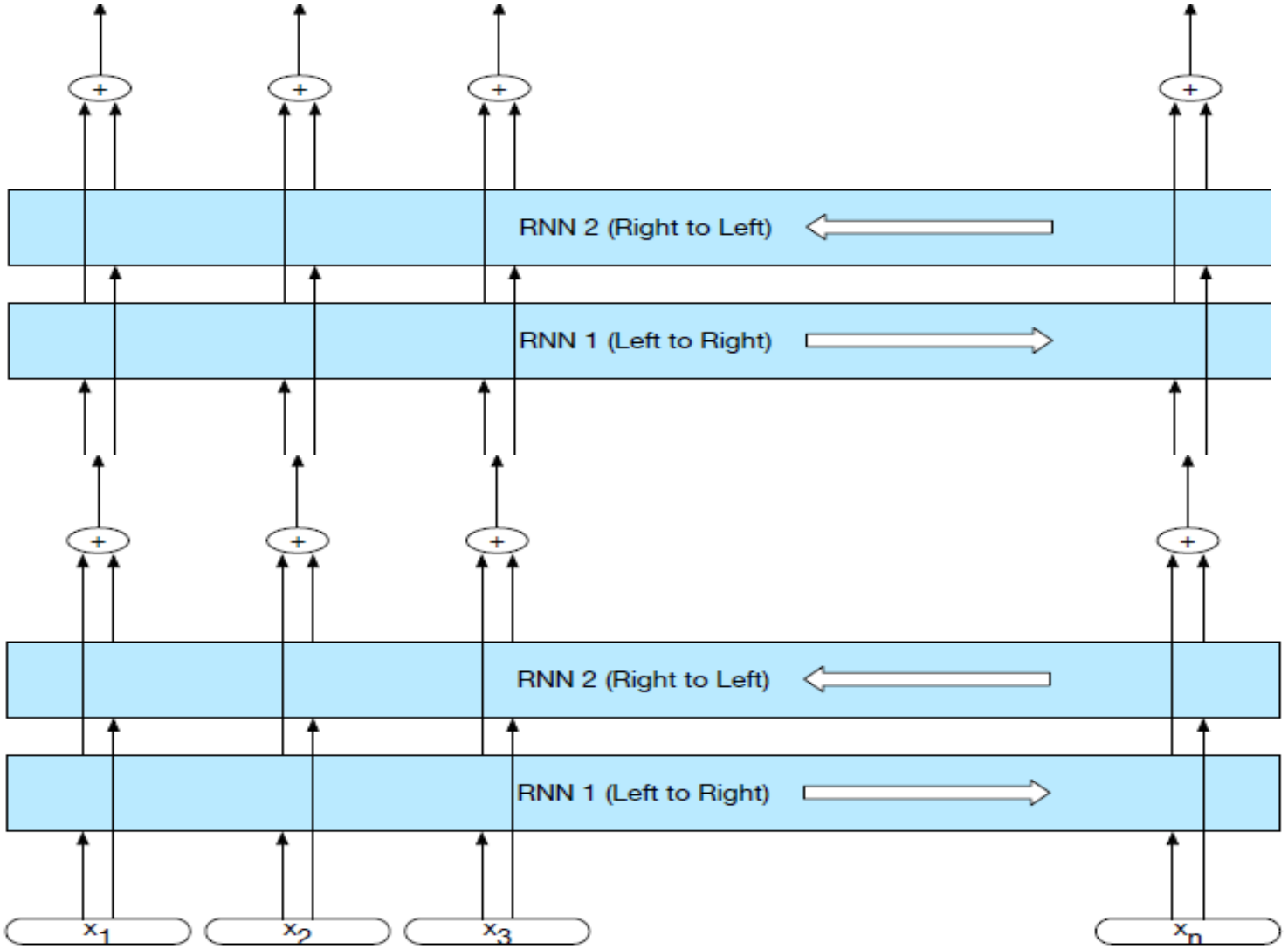
1. **Encoder**: acepta una secuencia de entrada, $\mathbf{x}_{1:n}$ y genera una secuencia correspondiente de representaciones contextualizadas,$\mathbf{h}_{1:n}$
2. **Context vector c**: función de $\mathbf{h}_{1:n}$ y transmite la esencia de la entrada al decodificador.
3. **Decoder**: Acepta **c** como entrada y genera una secuencia de longitud arbitraria de estados ocultos $\mathbf{h}_{1:m}$ a partir de la cual se puede obtener una secuencia correspondiente de estados de salida $\mathbf{y}_{1:m}$



Popular architectural choices: Encoder

Diseño de codificador ampliamente utilizado:**stacked Bi-LSTMs**

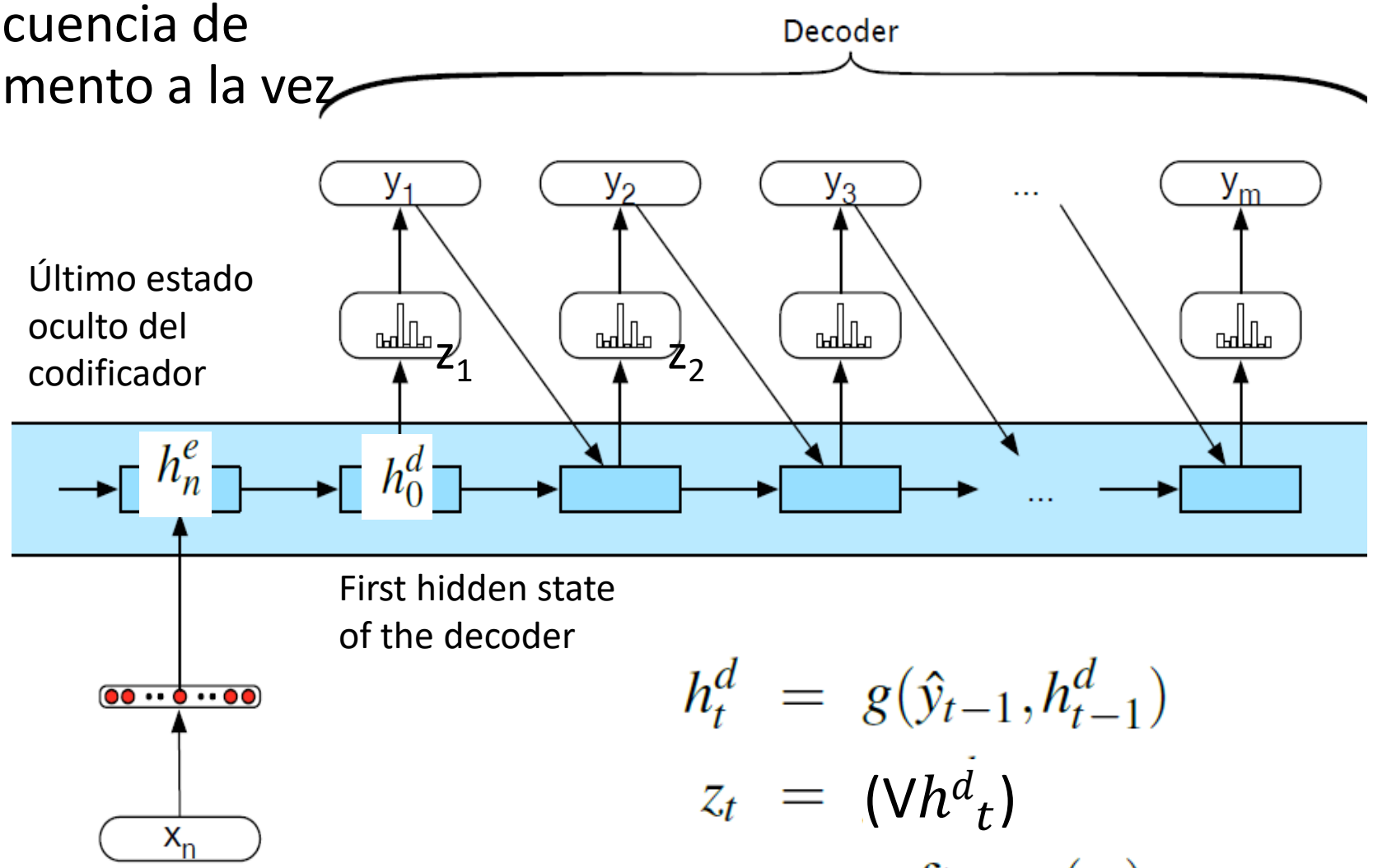
- Representaciones contextualizadas para cada paso de tiempo: **estados ocultos** de las capas superiores de los pasos hacia adelante y hacia atrás



Decoder Basic Design

- Producir una secuencia de salida de un elemento a la vez

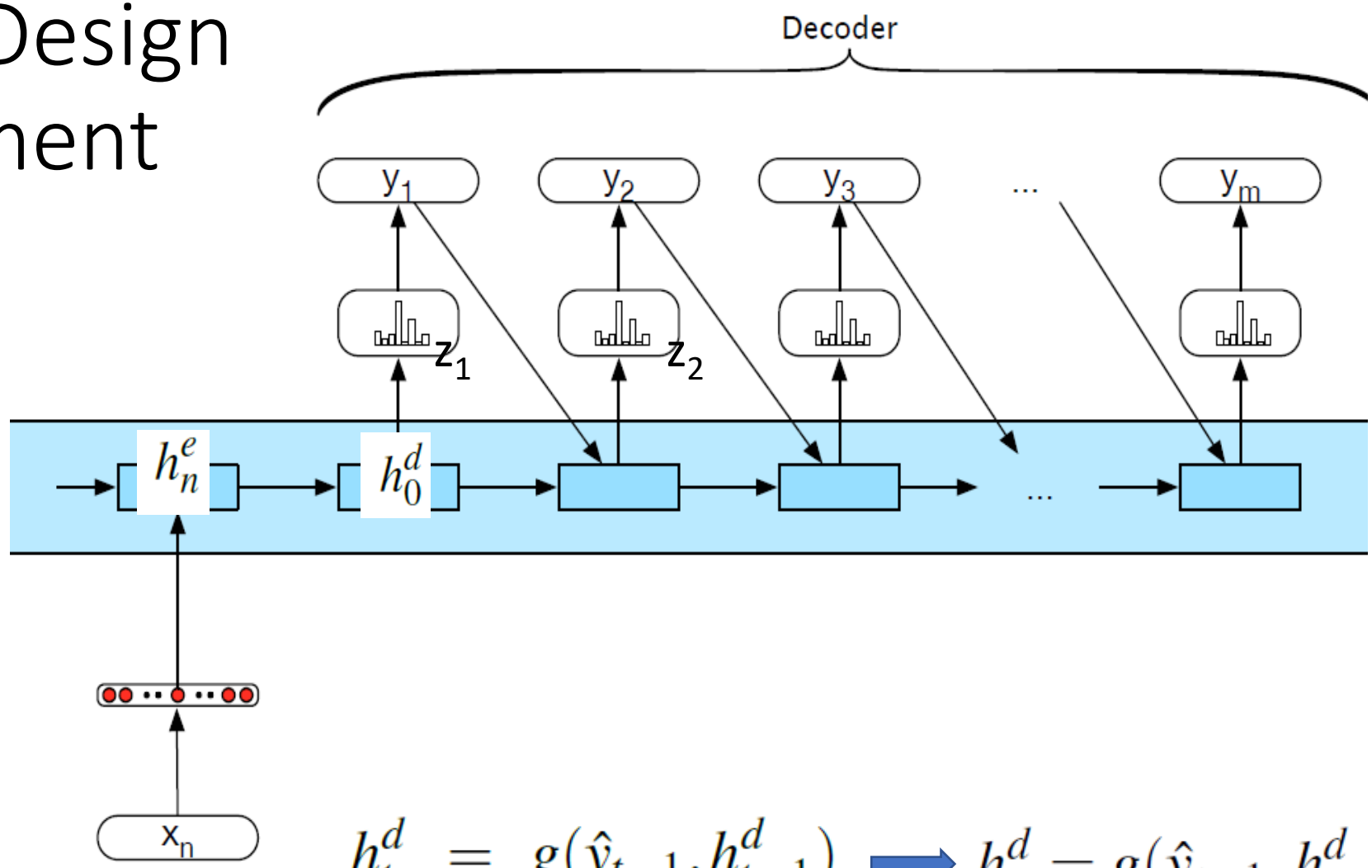
$$c = h_n^e$$
$$h_0^d = c$$



$$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d)$$
$$z_t = (Vh^d{}_t)$$
$$y_t = \text{softmax}(z_t)$$

Decoder Design Enhancement

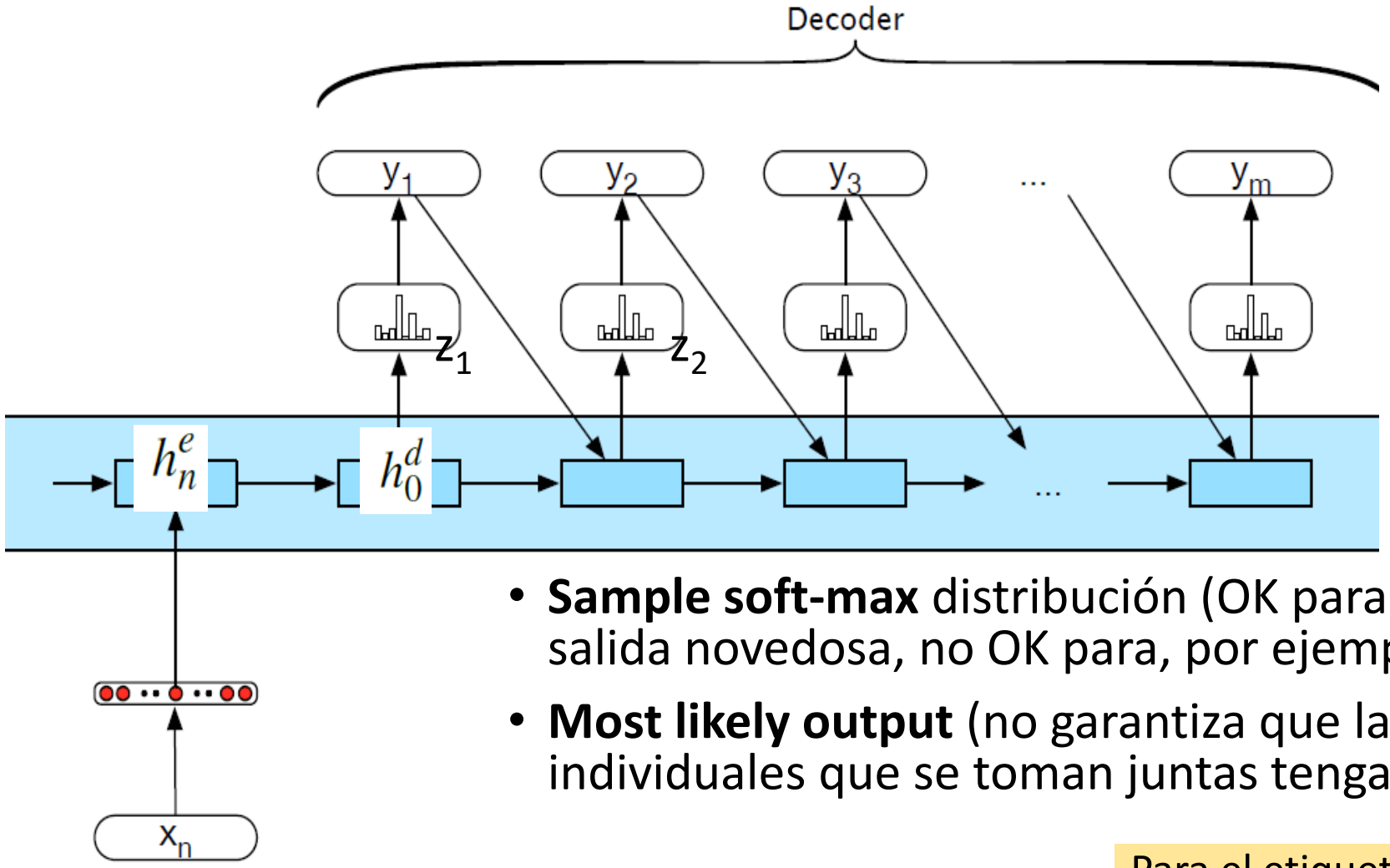
$$c = h_n^e$$
$$h_0^d = c$$



$$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d) \rightarrow h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d, c)$$
$$z_t = f(h_t^d)$$
$$y_t = \text{softmax}(z_t)$$

Contexto disponible en cada paso de la decodificación

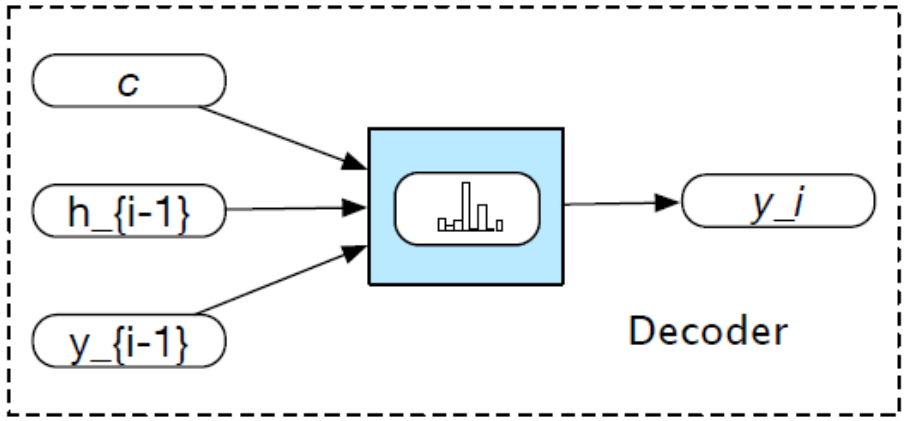
Decoder: How output *y* is chosen



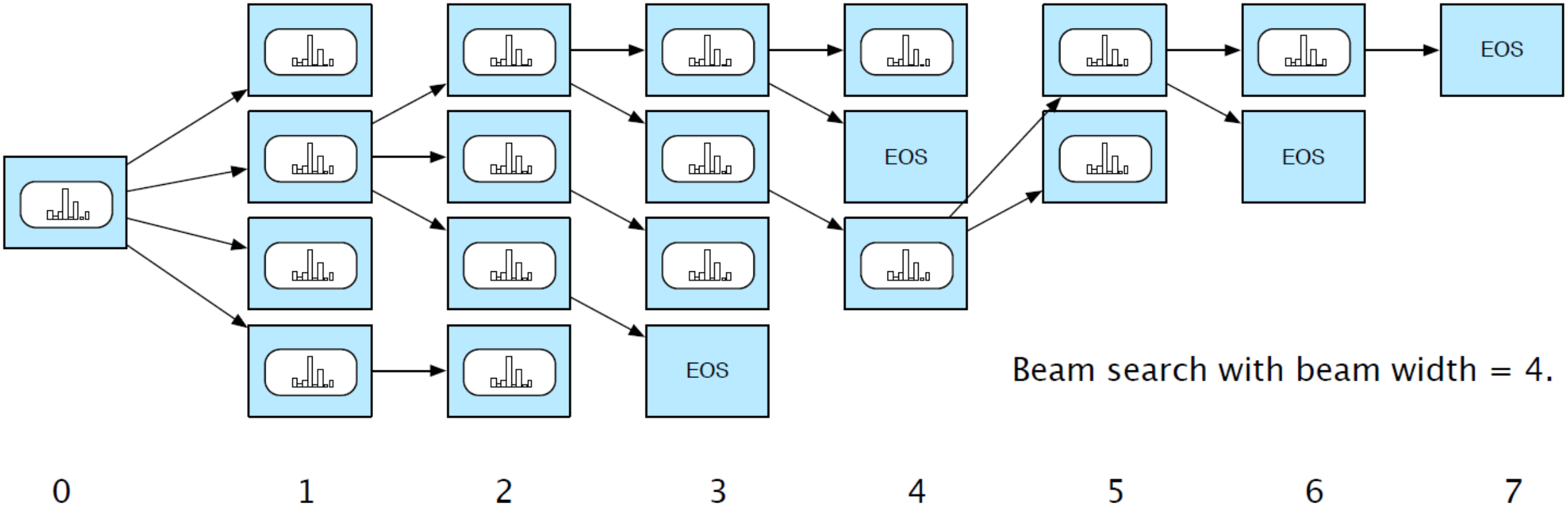
- **Sample soft-max** distribución (OK para generar una salida novedosa, no OK para, por ejemplo, MT o Summ)
- **Most likely output** (no garantiza que las decisiones individuales que se toman juntas tengan sentido)

Para el etiquetado de secuencias se utilizó

Viterbi – here not possible ☹



- 4 Lo más probable es que las "palabras" se hayan decodificado a partir del estado inicial
- Alimenta cada uno de ellos en el decodificador y guarda lo más probable 4 secuencias de dos palabras
- Introduzca la palabra más reciente en el decodificador y mantenga las 4 secuencias más probables de tres palabras .....
- Cuando se genera EOS. Detenga la secuencia y reduzca el haz en 1



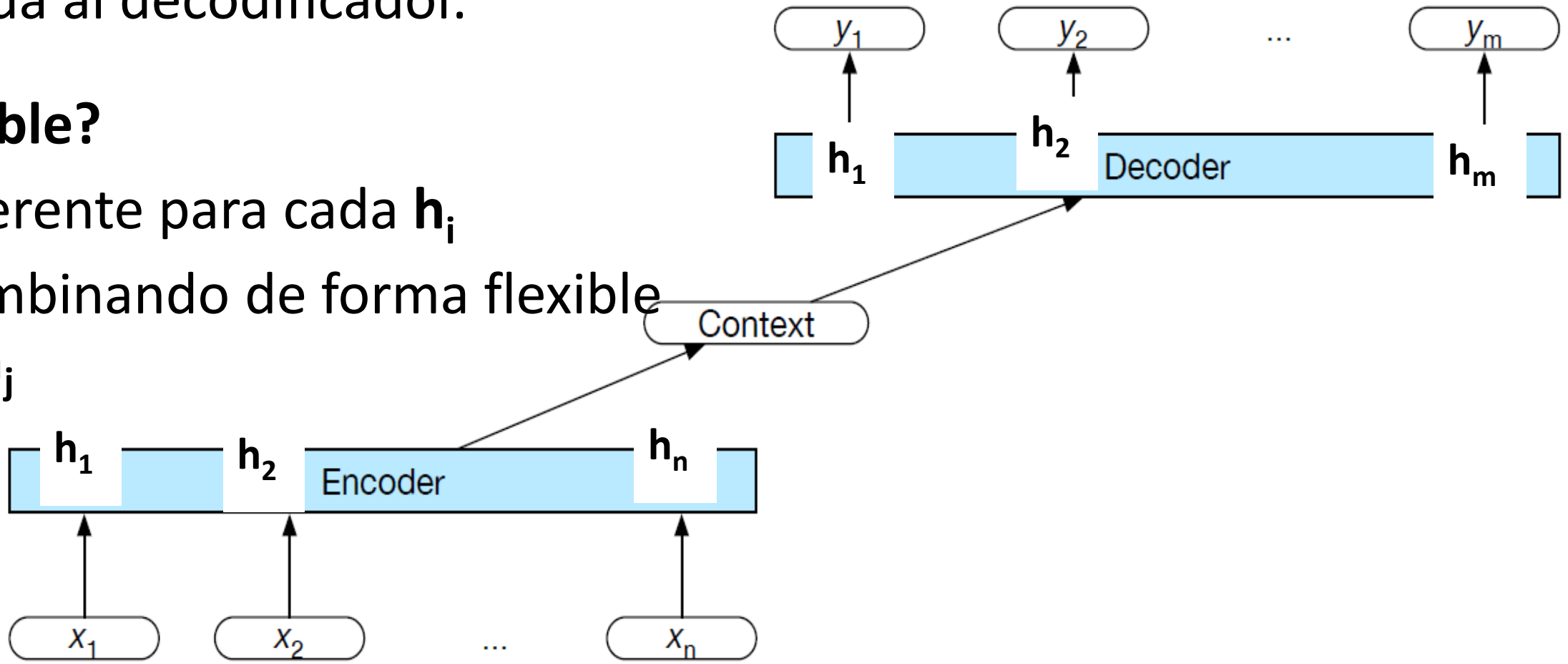
Attention

Contexto flexible: Attention

Context vector c: función de $\mathbf{h}_{1:n}$ y transmite la esencia de la entrada al decodificador.

Flexible?

- Diferente para cada $\mathbf{h}_i$
- Combinando de forma flexible el $\mathbf{h}_j$



Attention (1): Contexto derivado dinámicamente

- Reemplazar vector de contexto estático por dinámico $\mathbf{c_i}$
- derivados de los estados ocultos del codificador en cada punto i Durante la decodificación

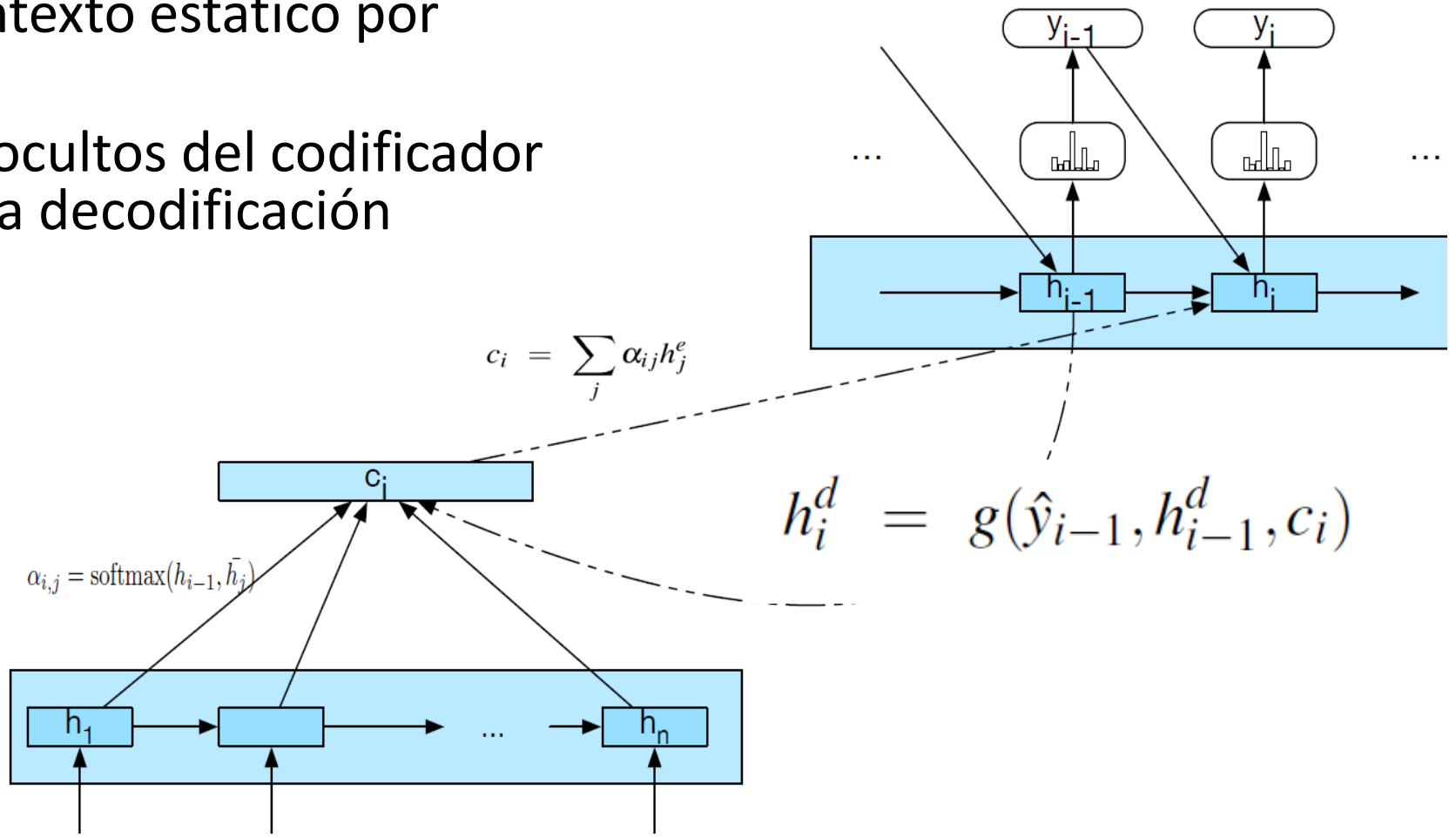
Ideas:

- debe ser una combinación lineal de esos estados

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

- debe depender de?

$$\alpha_{ij}$$



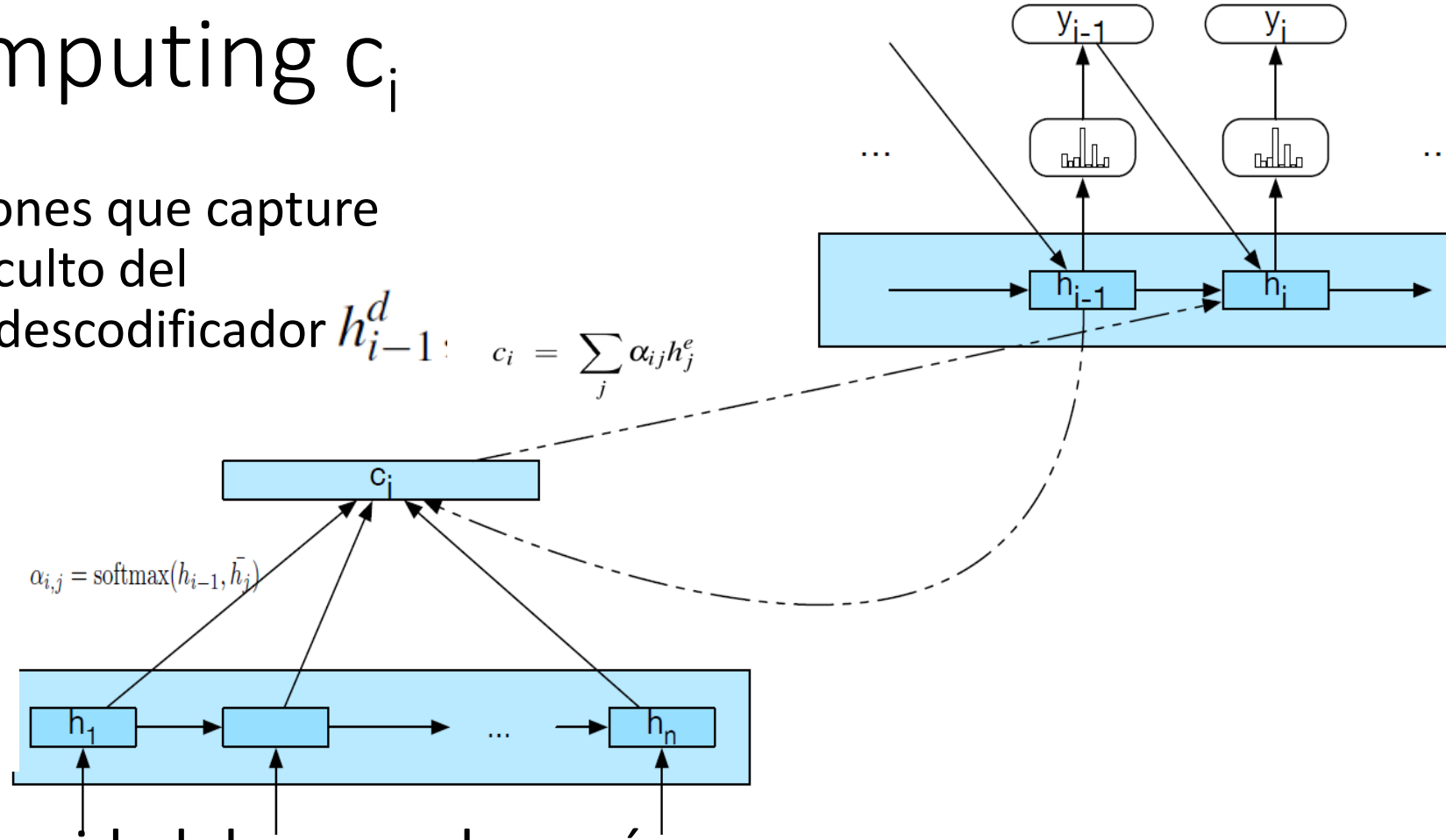
Attention (2): computing c_i

- Calcule un vector de puntuaciones que capture la relevancia de cada estado oculto del codificador para el estado del descodificador h_{i-1}^d :

$$score(h_{i-1}^d, h_j^e)$$

- Solo la similitud

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



- Proporcione a la red la capacidad de aprender qué aspectos de similitud entre los estados del decodificador y del codificador son importantes para la aplicación actual.

$$score(h_{i-1}^d, h_j^e) = h_{t-1}^d W_s h_j^e$$

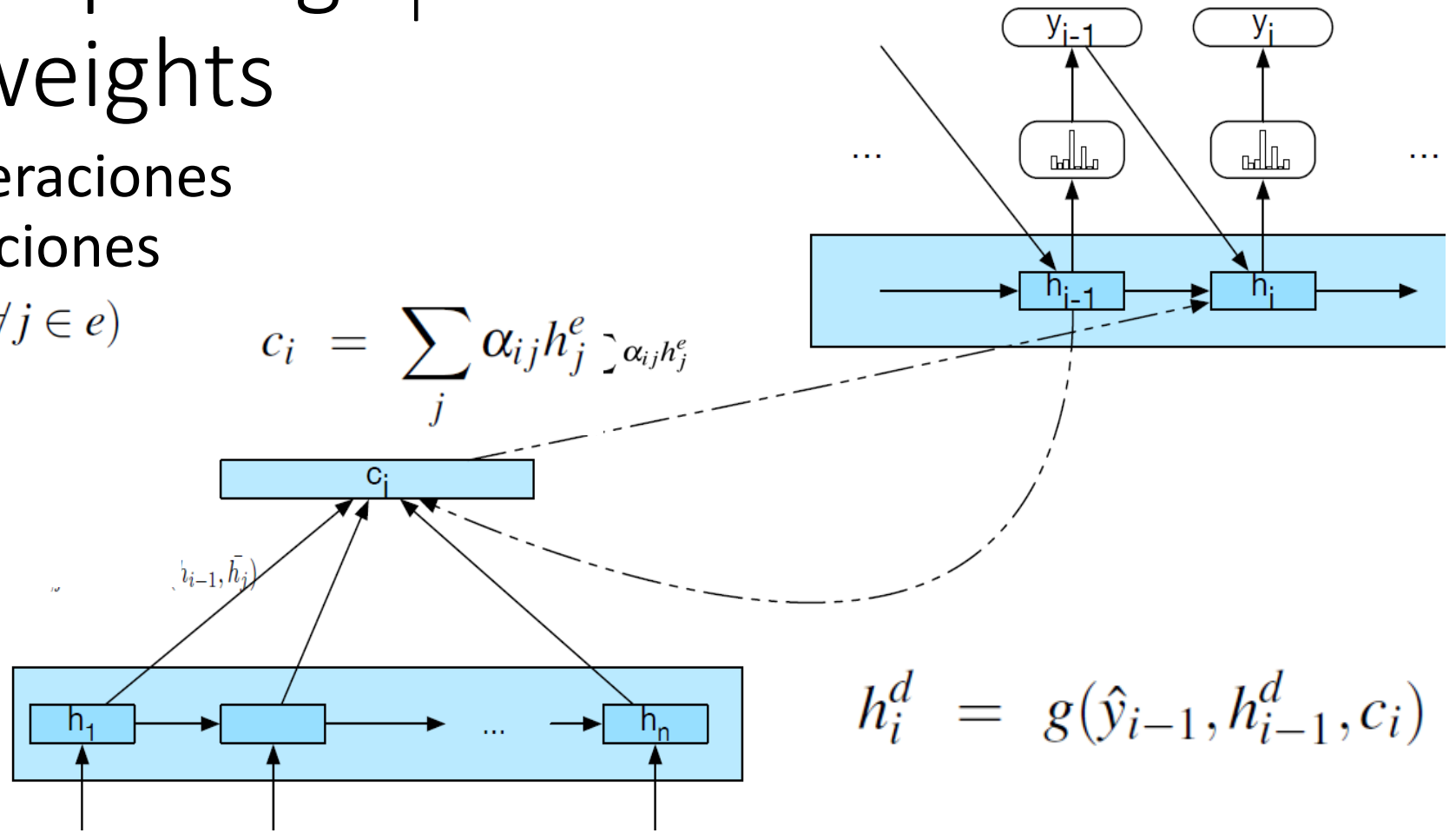
Attention (3): computing c_i From scores to weights

- Crear un vector de ponderaciones normalizando las puntuaciones

$$\alpha_{ij} = \text{softmax}(score(h^d_{i-1}, h^e_j) \ \forall j \in e)$$

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

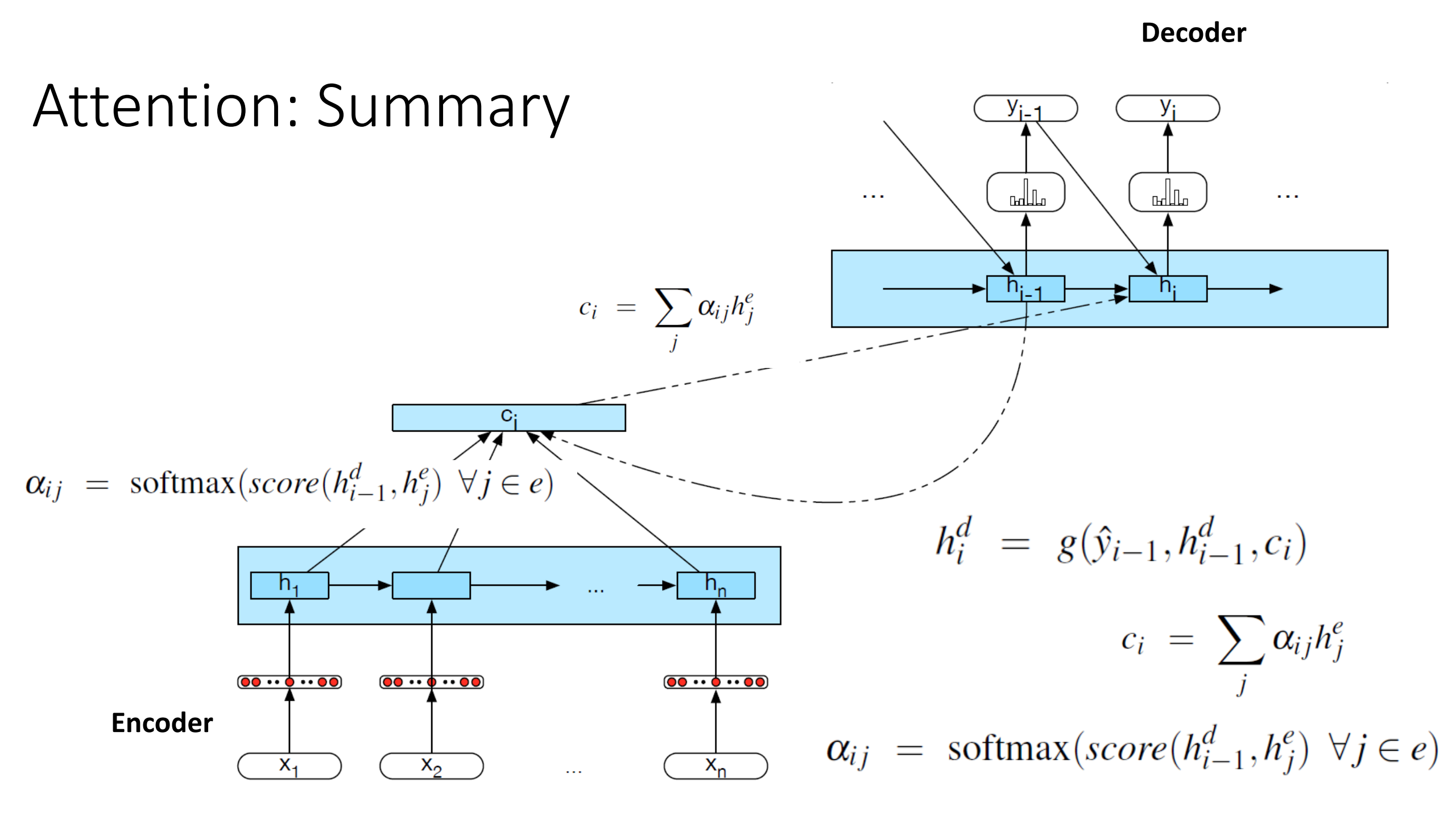
$$c_i = \sum_j \alpha_{ij} h^e_j$$



$$h^d_i = g(\hat{y}_{i-1}, h^d_{i-1}, c_i)$$

- **Objetivo alcanzado:** calcular un vector de contexto de longitud fija para el estado actual del decodificador tomando una media ponderada de todos los estados ocultos del codificador.

Attention: Summary



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

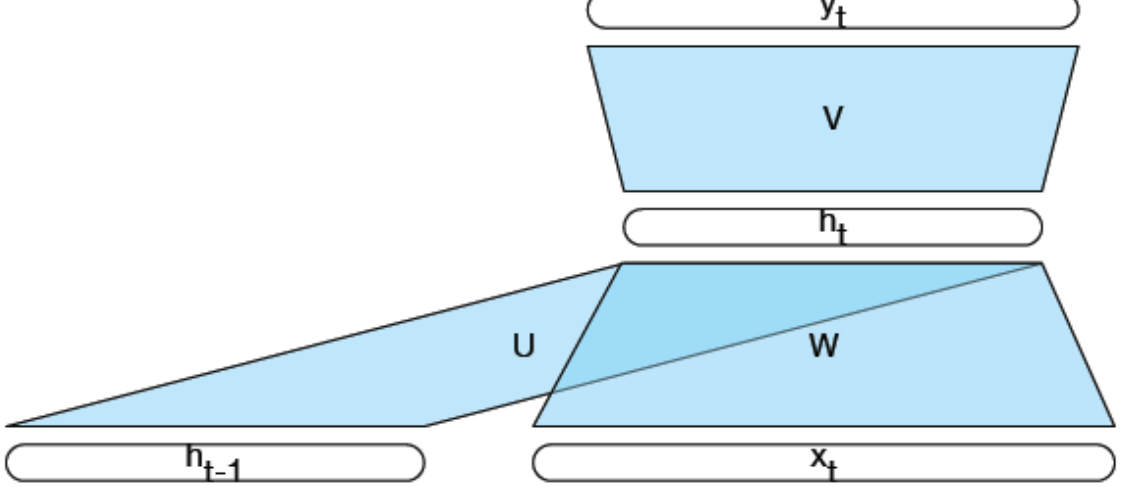
$$\alpha_{ij} = \text{softmax}(score(h_{i-1}^d, h_j^e) \ \forall j \in e)$$

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

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

$$\alpha_{ij} = \text{softmax}(score(h_{i-1}^d, h_j^e) \ \forall j \in e)$$

Explain Y. Goldberg different notation



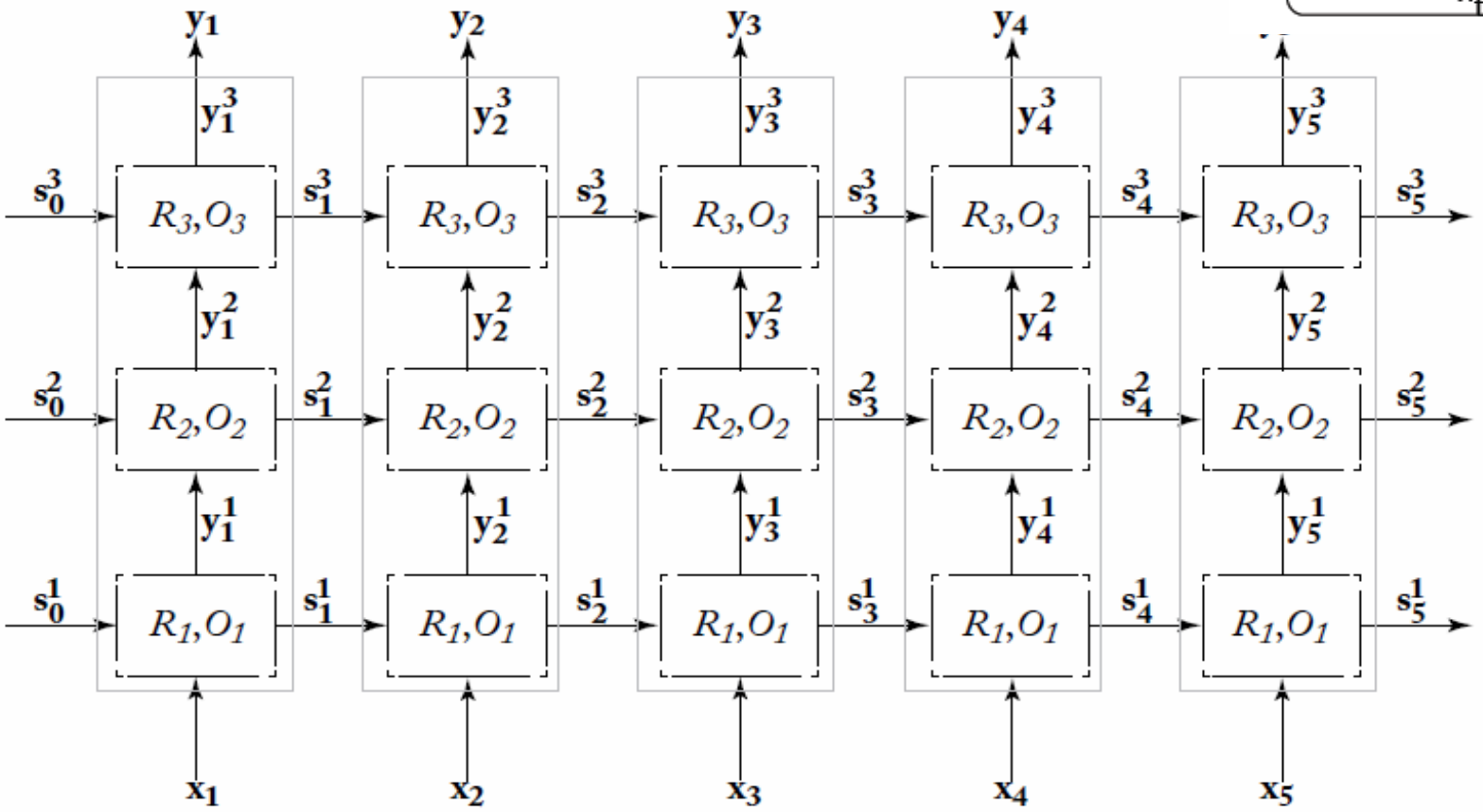


Figure 14.7: A three-layer ("deep") RNN architecture.

Intro to Encoder-Decoder and Attention (Goldberg's notation)

Decoder

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

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

$$\alpha_{ij} = \text{softmax}(score(h_{i-1}^d, h_j^e) \ \forall j \in e)$$

Encoder

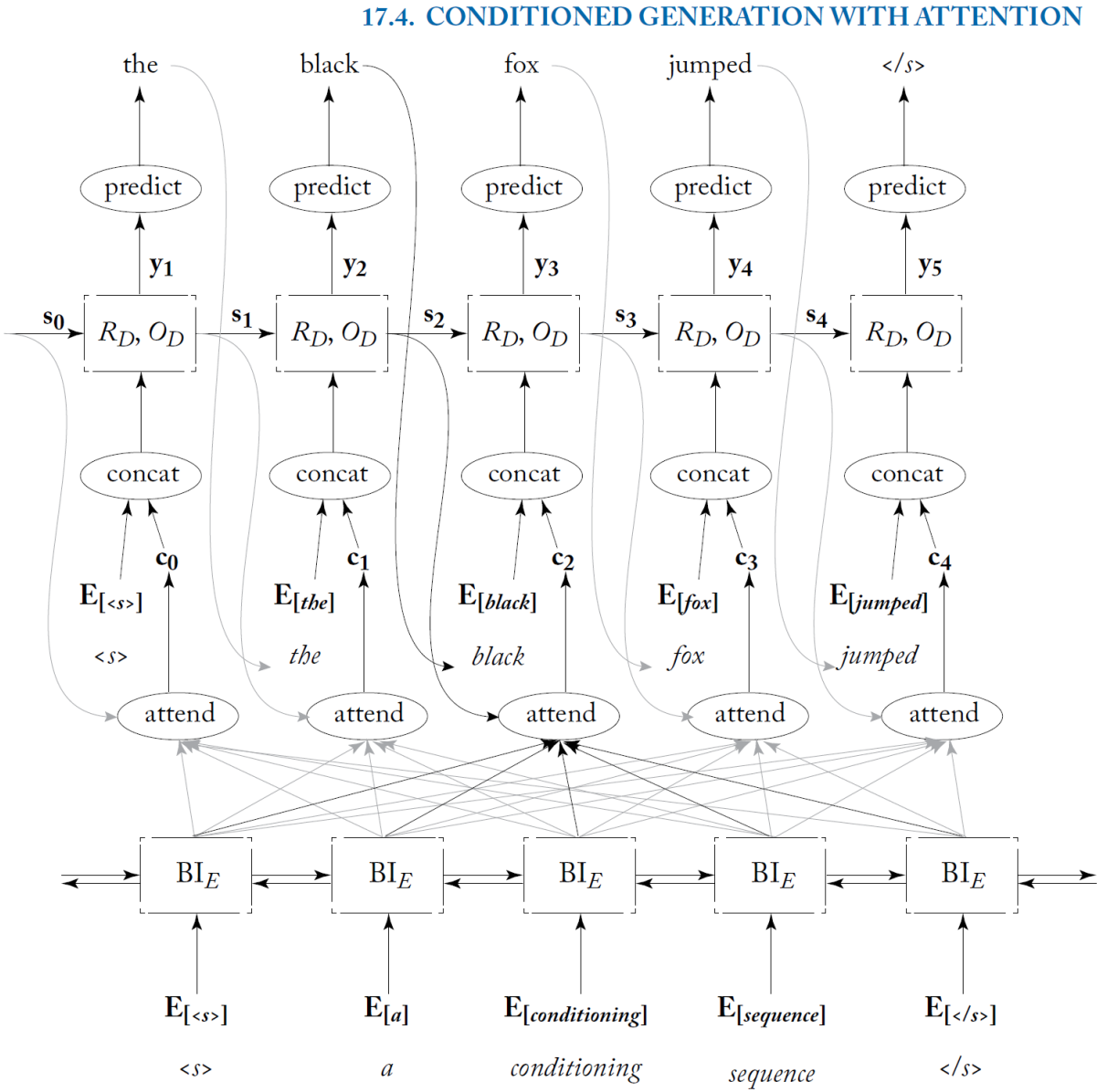


Figure 17.5: Sequence-to-sequence RNN generator with attention.

Trasnformer

Transformers (Attention is all you need 2017)

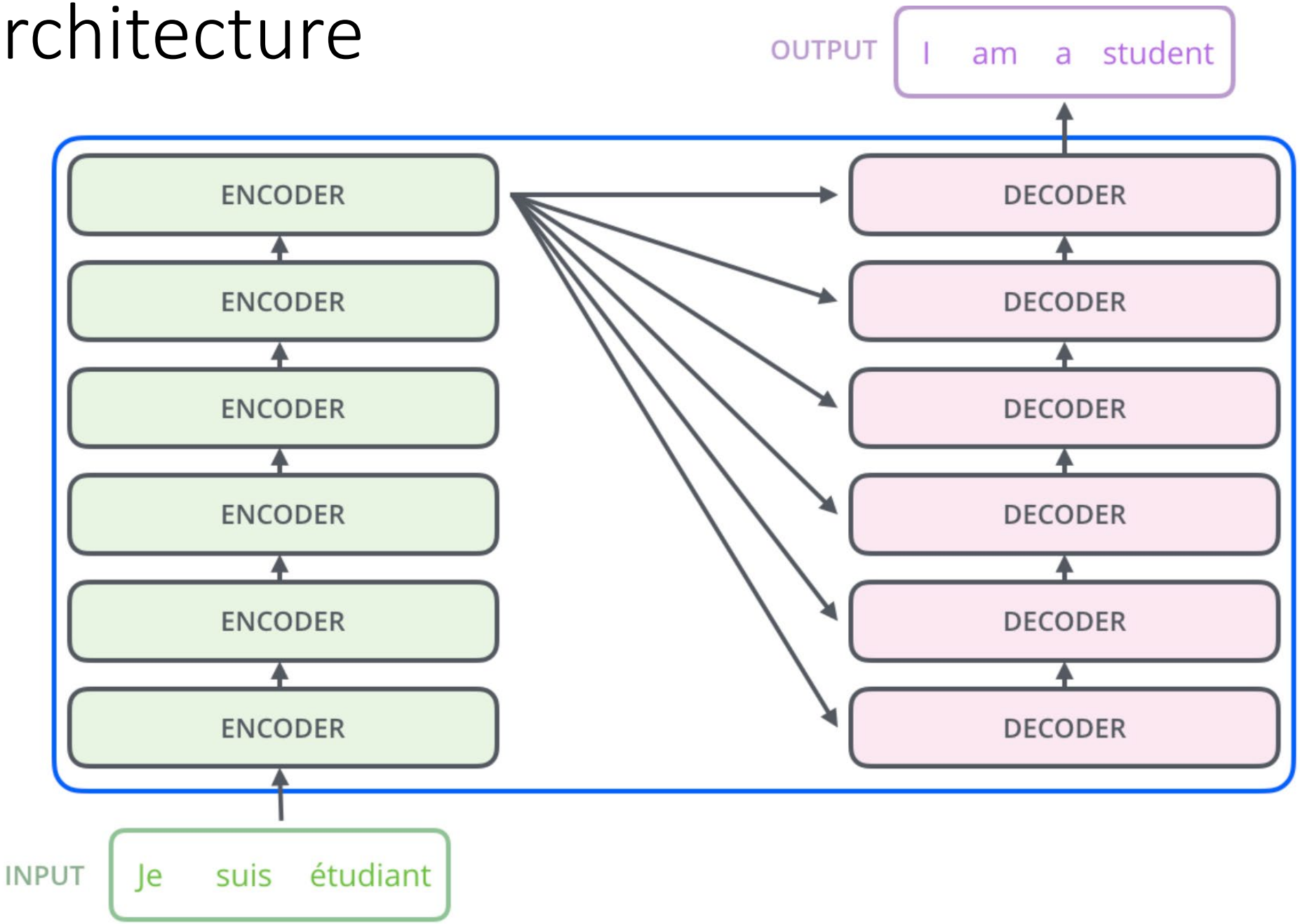
- **Solo una introducción:** Estos son dos recursos valiosos para conocer más detalles sobre la arquitectura y la implementación

- http://nlp.seas.harvard.edu/2018/04/03/attention.html

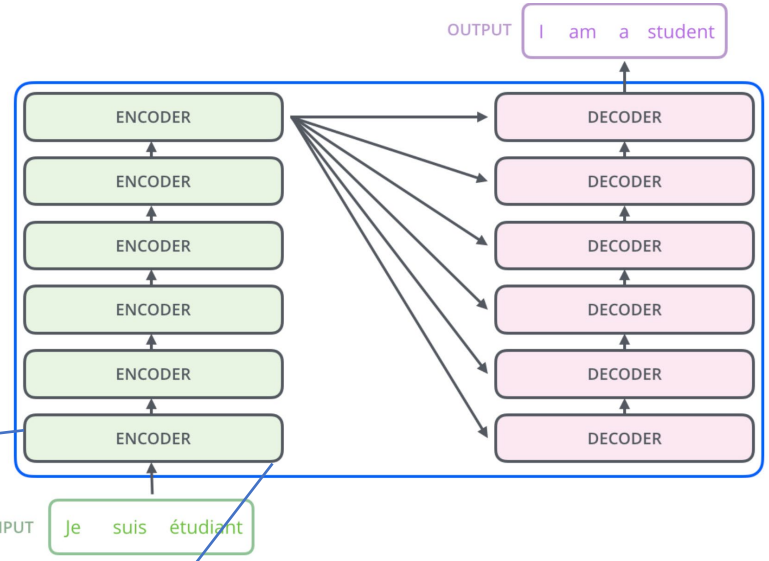
- https://jalammar.github.io/illustrated-transformer/ (slides come from this source)

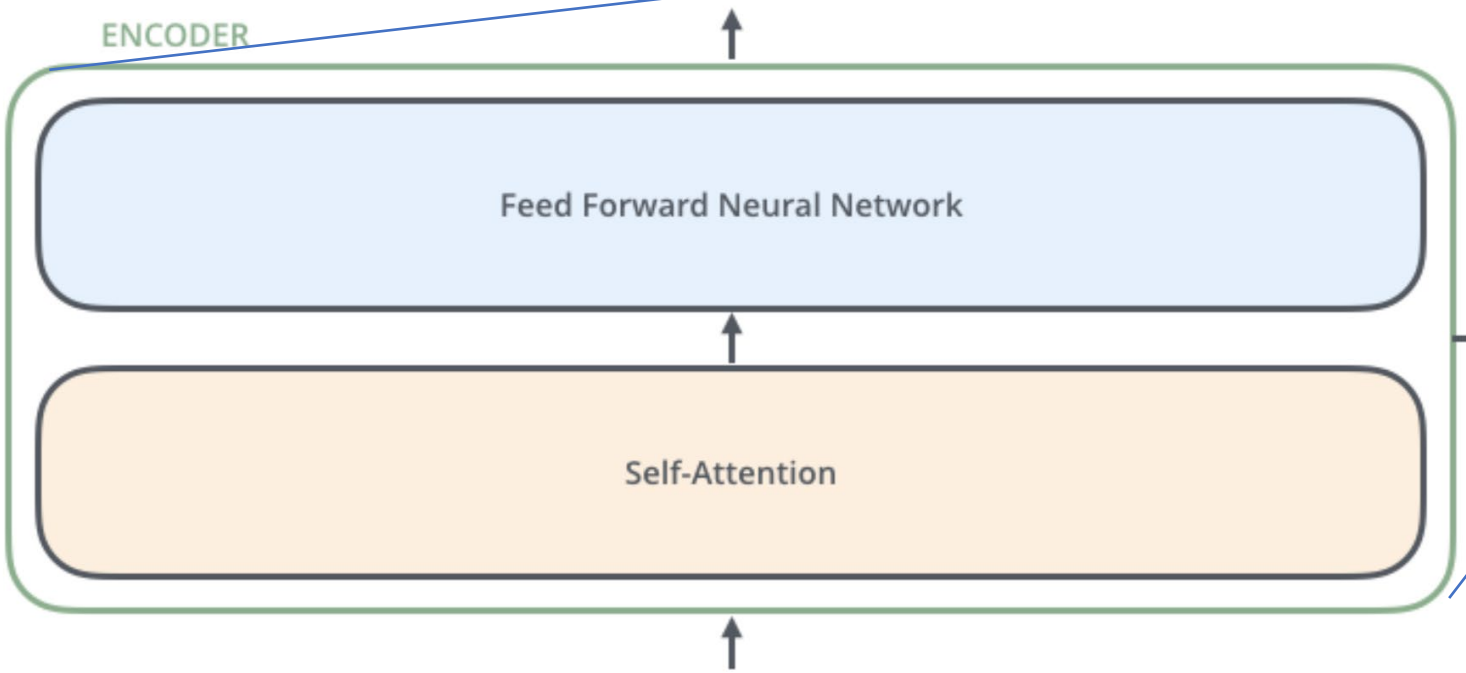
High-level architecture

- Solo mirará la parte del CODIFICADOR (s) en detalle



Los **codificadores** son todos **idénticos en estructura** (pero no comparten pesos).
Cada uno se divide en dos subcapas



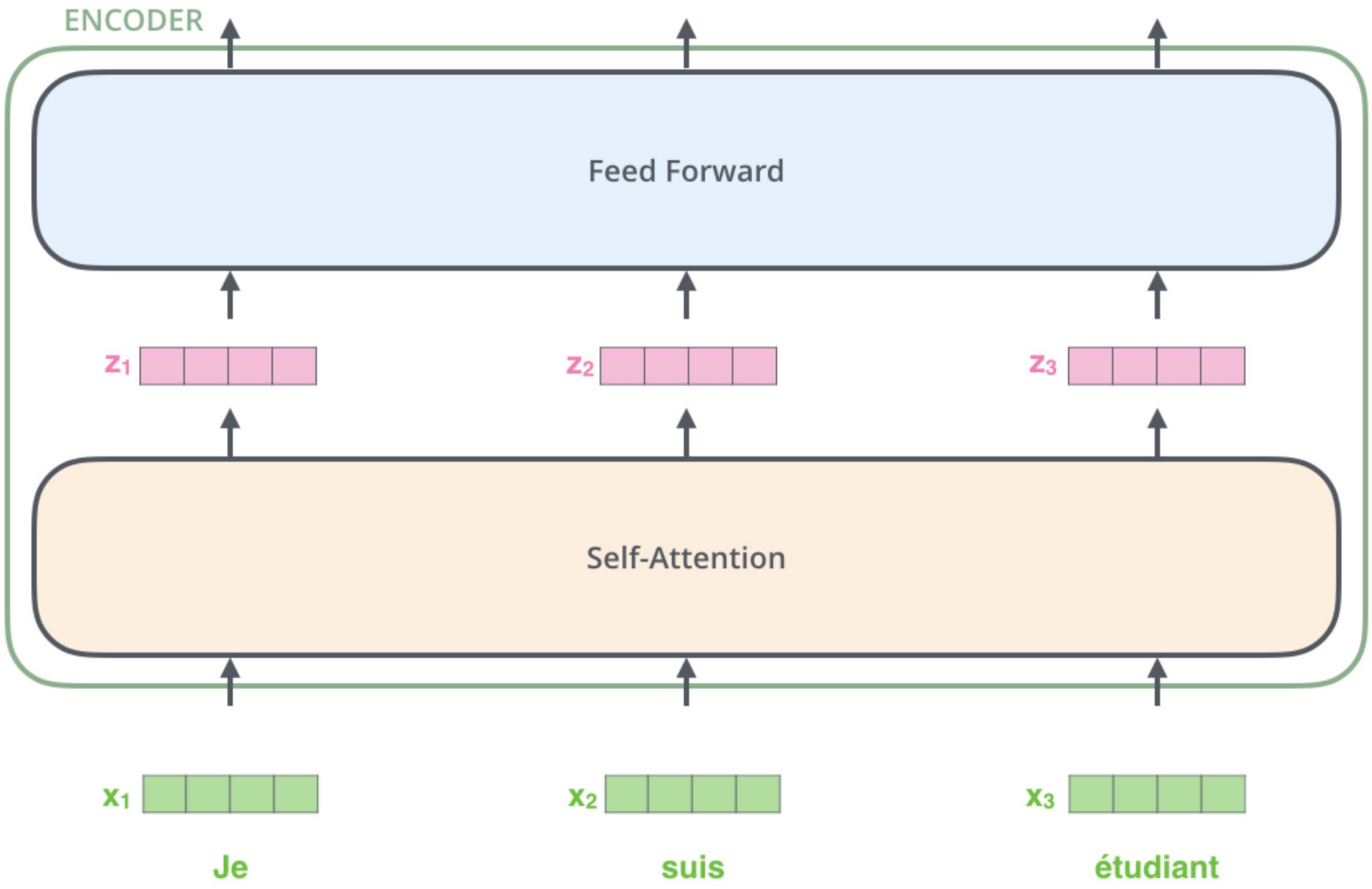


Los resultados de la autoatención se alimentan a una red neuronal de realimentación. Exactamente el mismo se aplica de forma independiente a cada posición.

Ayuda al codificador a examinar otras palabras de la frase de entrada a medida que codifica una palabra específica.

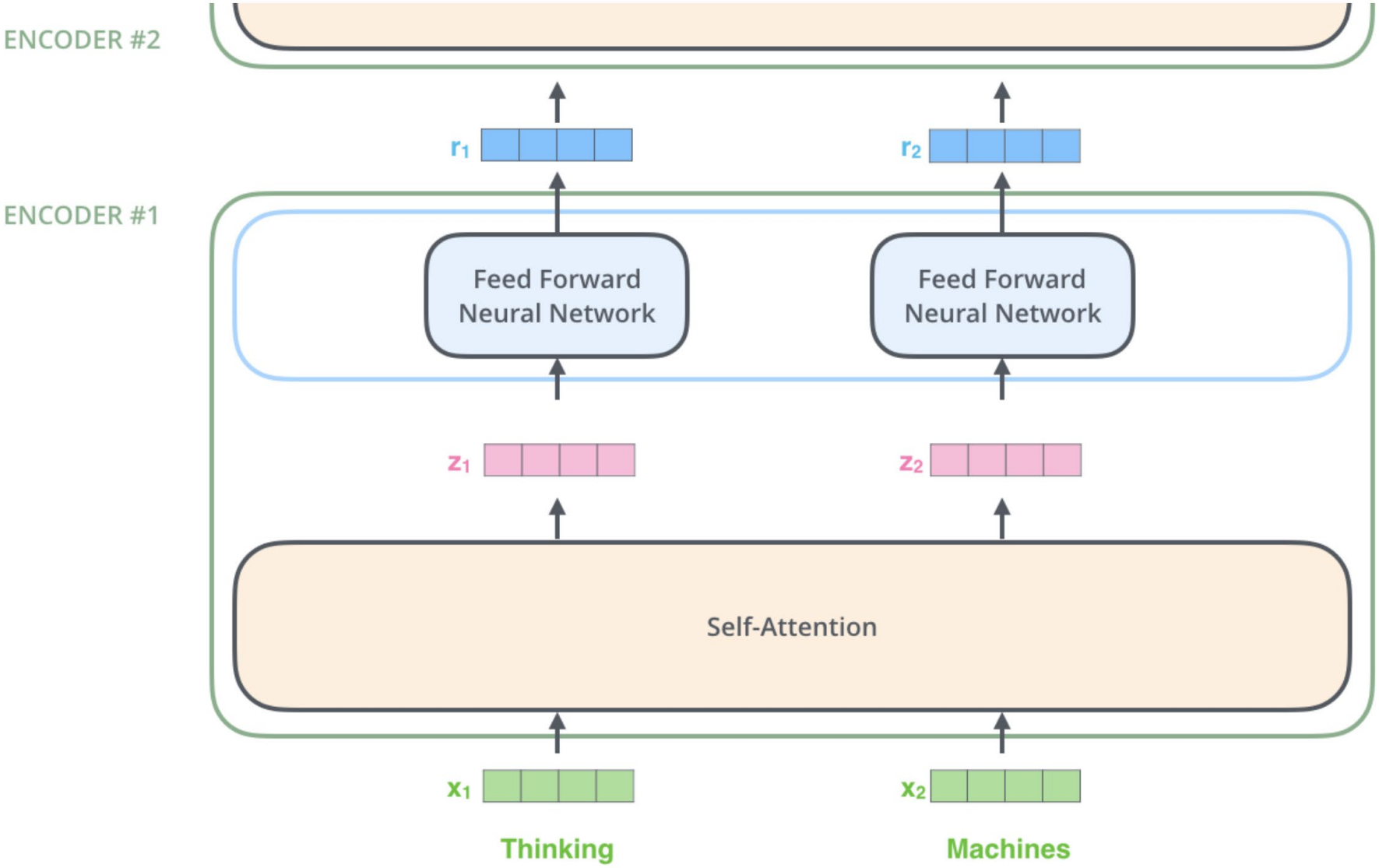
Propiedad clave de Transformer: La palabra en cada posición fluye a través de su propia ruta en el codificador.

- Hay dependencias entre estos caminos en la capa de autoatención.
- La capa Feed-forward layer no tiene esas dependencias=> ¡Se pueden ejecutar varias rutas en paralelo!



Visualmente más claro en dos palabras

- dependencias en la capa de autoatención.
- No hay dependencias en Feed-forward layer



Word embeddings

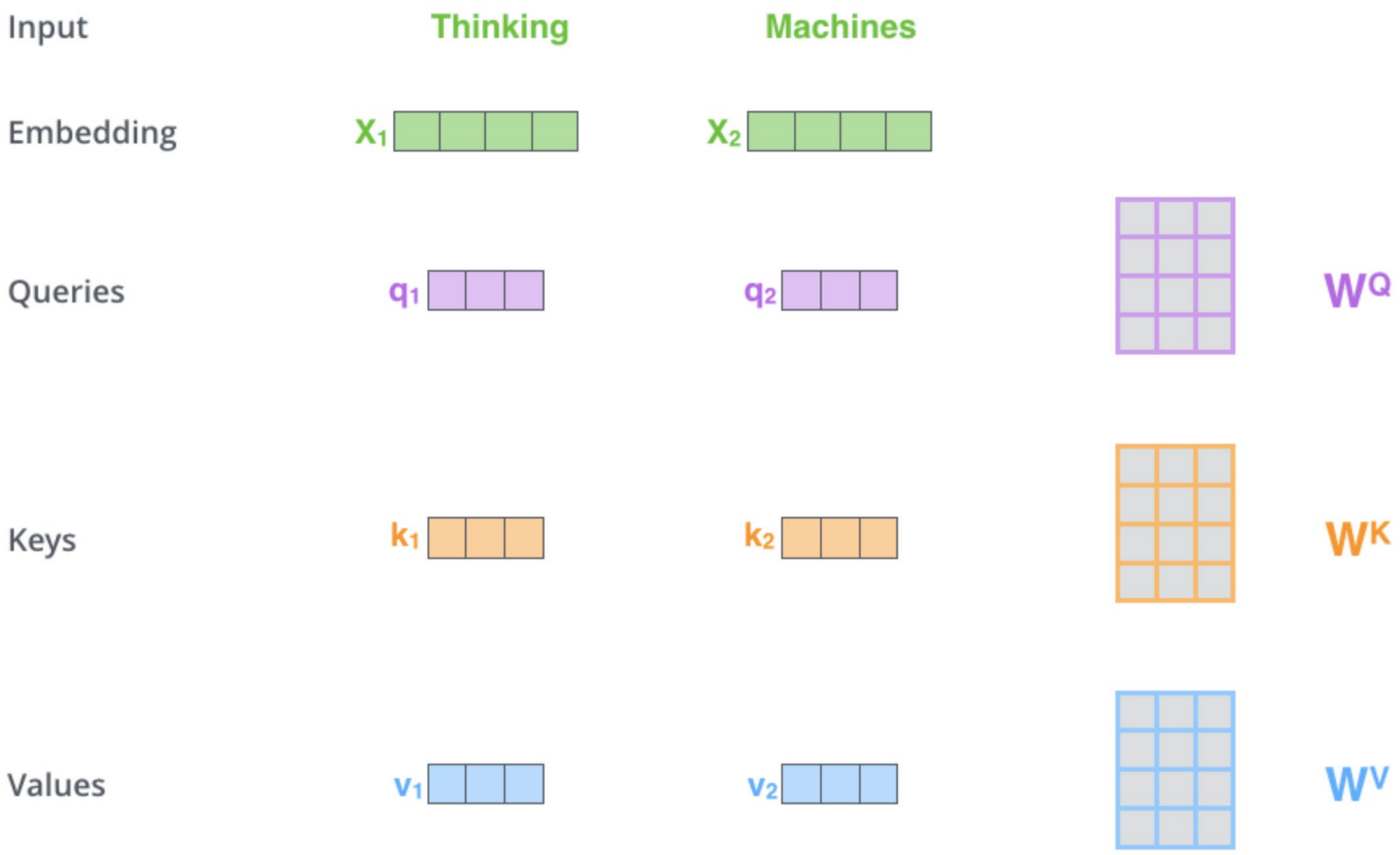
Self-Attention

Mientras se procesa **cada palabra**, permite mirar otras posiciones en la secuencia de entrada en busca de pistas para construir una mejor codificación para **esta palabra**.

Step1: Crea tres vectores de cada uno de los vectores de entrada del codificador:

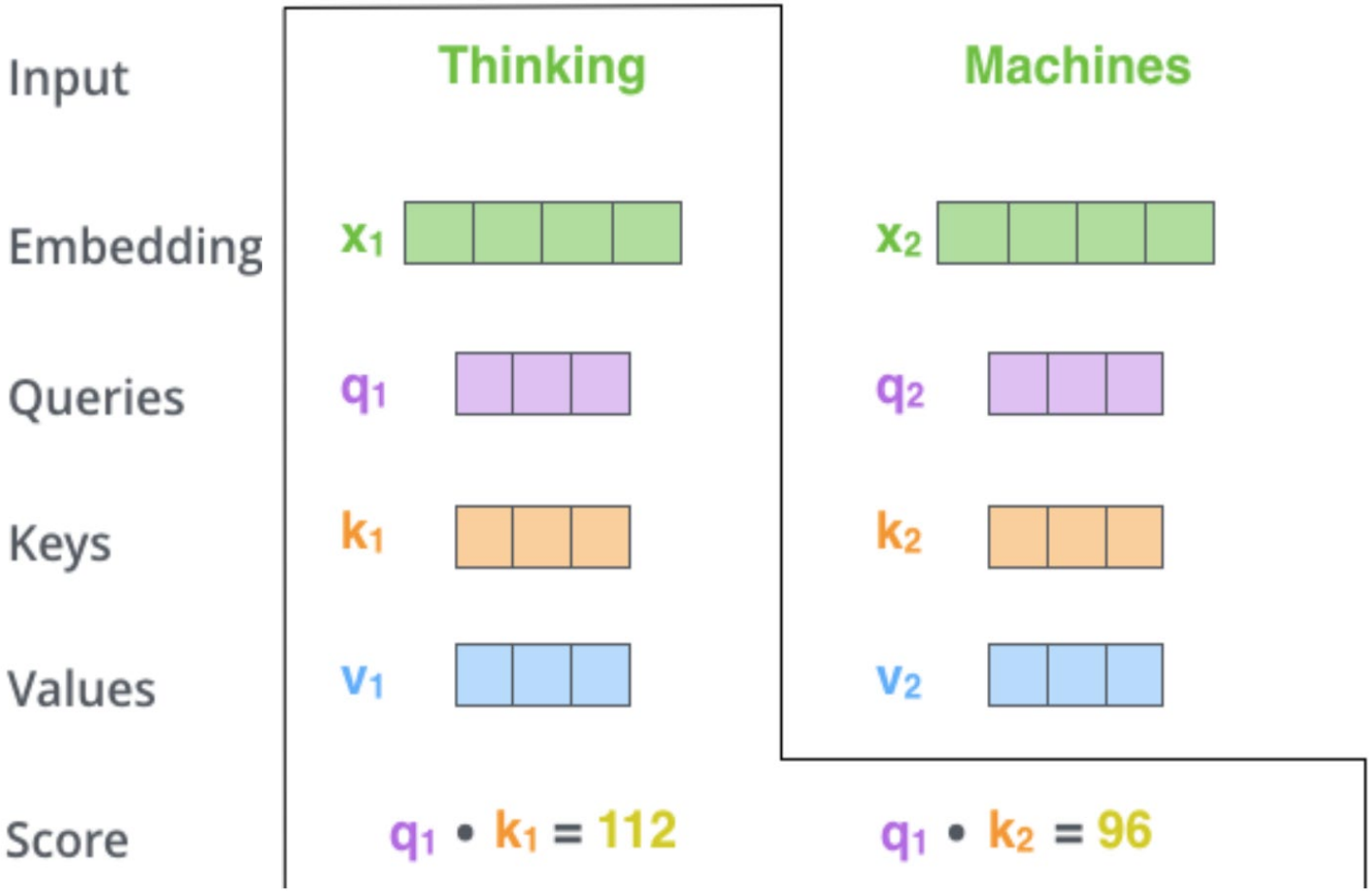
Query, a Key, Value (Dimensiones típicamente más pequeñas).

multiplicando el embedding por tres matrices que entrenamos



Self-Attention

Step 2: Calcular una puntuación (¡como hemos visto para la atención regular!) Cuánto enfoque colocar en otras partes de la oración de entrada a medida que codificamos una palabra en una posición determinada.

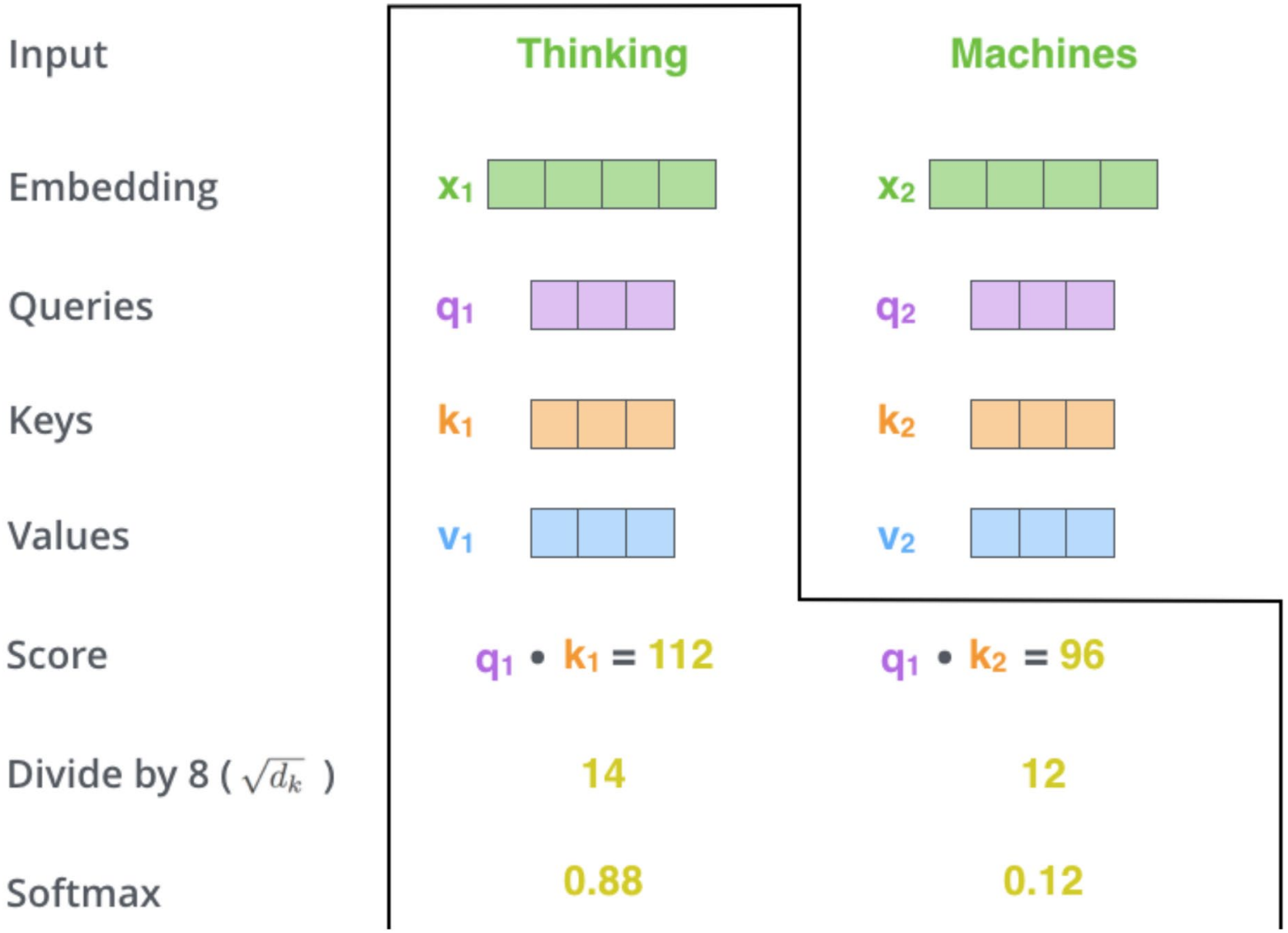


Tome el producto escalar del vector de consulta con el vector clave de la palabra respectiva que estamos puntuando.

Procesando la autoatención para la palabra "Pensamiento" en la posición #1, la primera puntuación sería el producto escalar de q1 y k1. La segunda puntuación sería el producto escalar de q1 y k2.

Self Attention

- **Step 3** Divida las puntuaciones por la raíz cuadrada de la dimensión de los vectores clave (gradientes más estables).
- **Step 4** Pase el resultado a través de una operación softmax. (todos positivos y suman 1)



Intuition: La puntuación softmax determina cuánto se expresará cada palabra en esta posición.

Self Attention

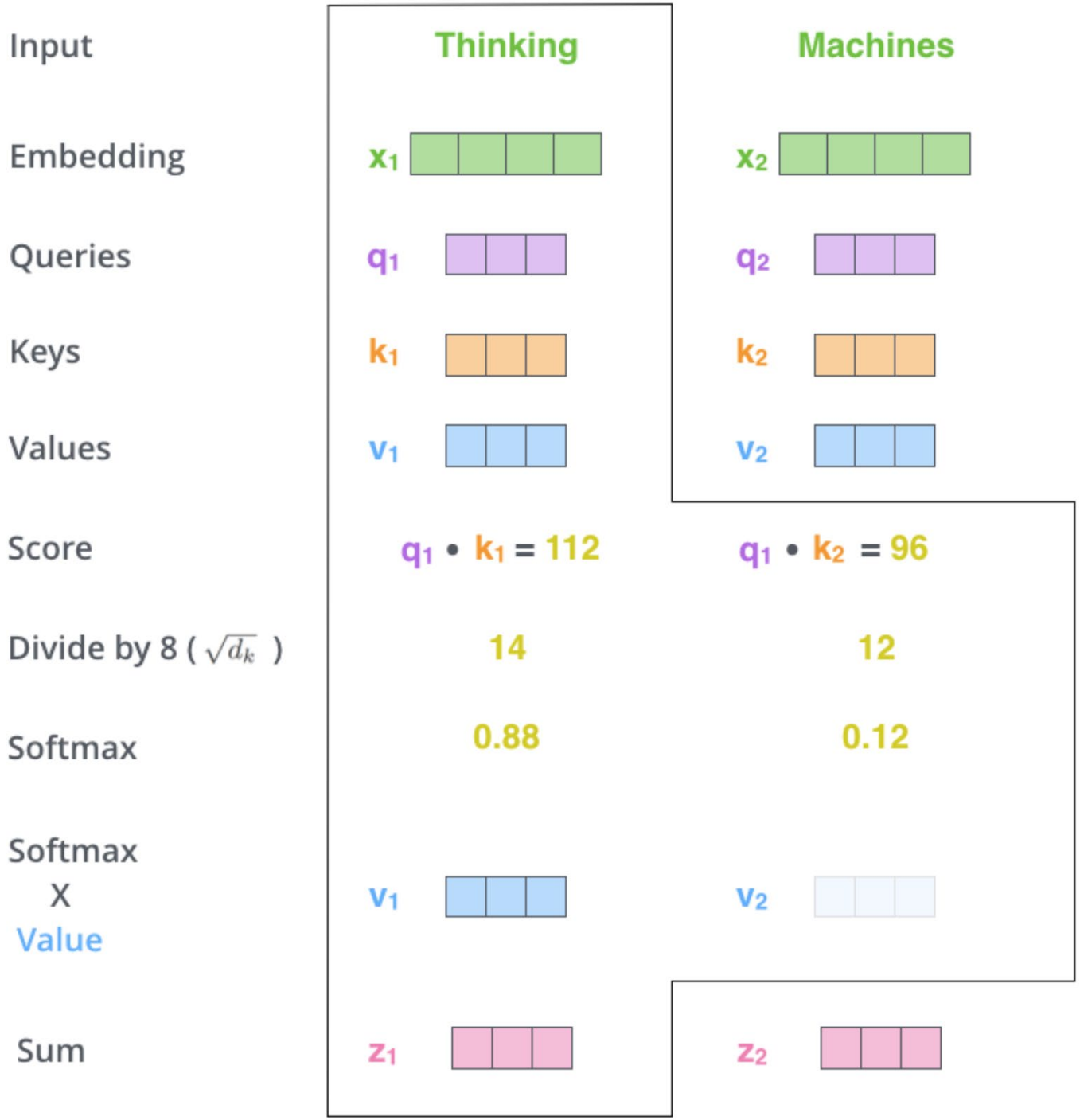
- **Step6** : Suma los vectores de valores ponderados. Esto produce la salida de la capa de autoatención en esta posición

Más detalles:

Lo que hemos visto para una palabra se hace para todas las palabras (usando matrices)

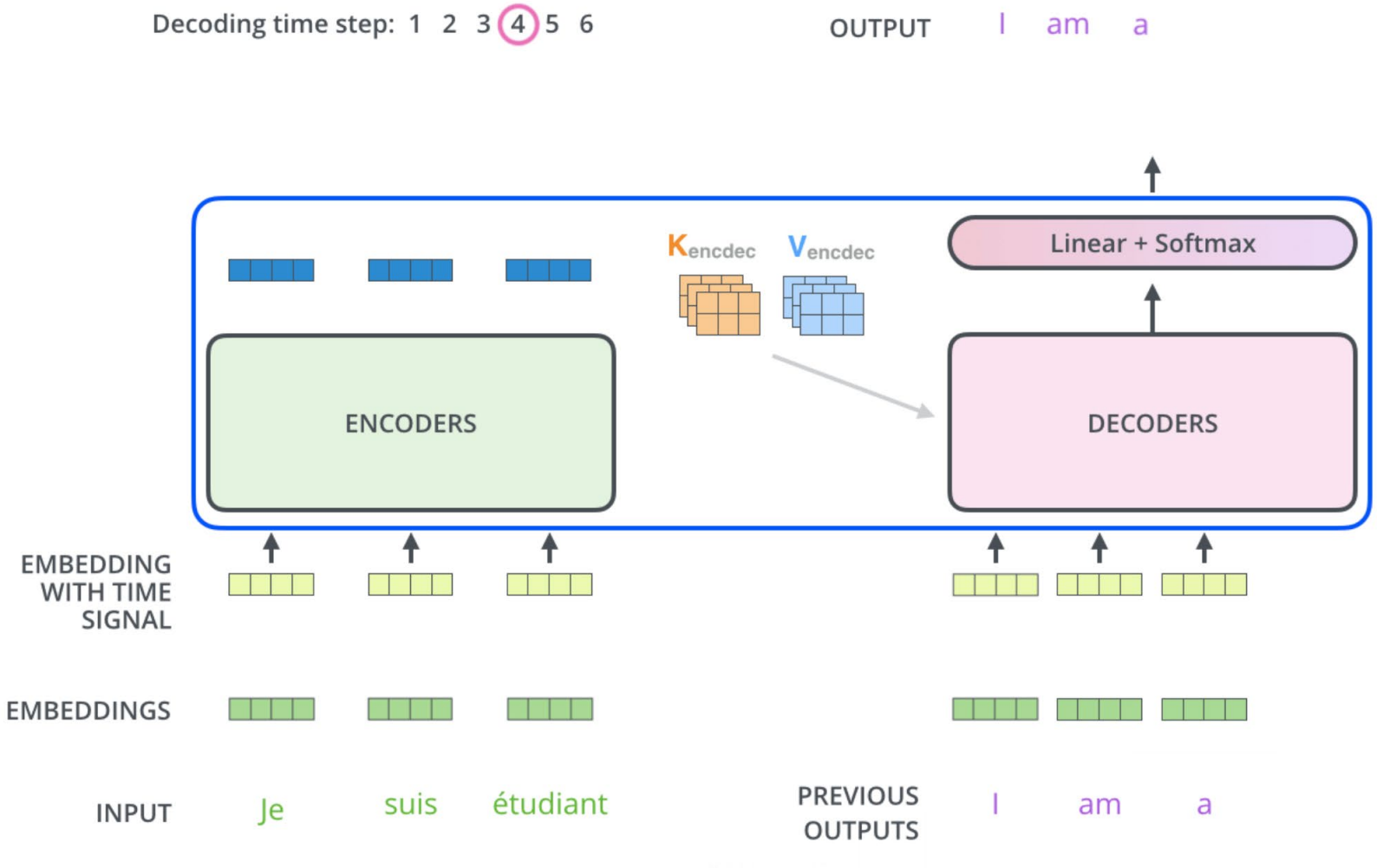
- Necesidad de codificar la posición de las palabras
- Y mejoró usando un mecanismo llamado atención "multicéfalo“

(algo así como múltiples filtros para CNN)



The Decoder Side

- Se basa en la mayoría de los conceptos del lado del codificador
- Animación enhttps://jalammar.github.io/illustrated-transformer/



Transformers

- WIKIPEDIA:
- Sin embargo, a diferencia de las RNN, los transformadores no requieren que la secuencia se procese en orden. Por lo tanto, si los datos en cuestión son lenguaje natural, el Transformer no necesita procesar el comienzo de una oración antes de procesar el final. Debido a esta característica, el transformador permite mucha más paralelización que las RNN durante el entrenamiento. [1]

El Transformer utiliza la atención de múltiples cabezales de tres maneras diferentes:

En Capaz "encoder-decoder attention", las consultas provienen de la capa decodificadora anterior, y las claves y valores de memoria provienen de la salida del codificador. Esto permite que cada posición en el decodificador para atender todas las posiciones de la secuencia de entrada. Esto imita los mecanismos típicos de atención de codificador-decodificador en modelos de secuencia a secuencia.

* La capa encoder contiene self-attention . **En una capa de autoatención, todas las claves, valores y consultas provienen del mismo lugar, en este caso, la salida de la capa anterior en el codificador.** Cada posición en el codificador puede atender a todas las posiciones en la capa anterior del codificador.

*De forma similar, self-attention layers en el decoder **Permite que cada posición en el decodificador atienda a todas las posiciones en el decodificador hasta esa posición inclusive.** Necesitamos evitar el flujo de información hacia la izquierda en el decodificador para preservar la propiedad autorregresiva. Implementamos esto dentro de la atención escalada-producto-punto enmascarando (estableciendo en infinito) todos los valores en la entrada del softmax que corresponden a conexiones ilegales.

En cada paso, el modelo es autorregresivo
[10], consumiendo los símbolos generados previamente como entrada adicional al generar el siguiente.
El modelo autorregresivo especifica que la variable de salida depende linealmente de sus propios valores anteriores y de un término estocástico (un término imperfectamente predecible);

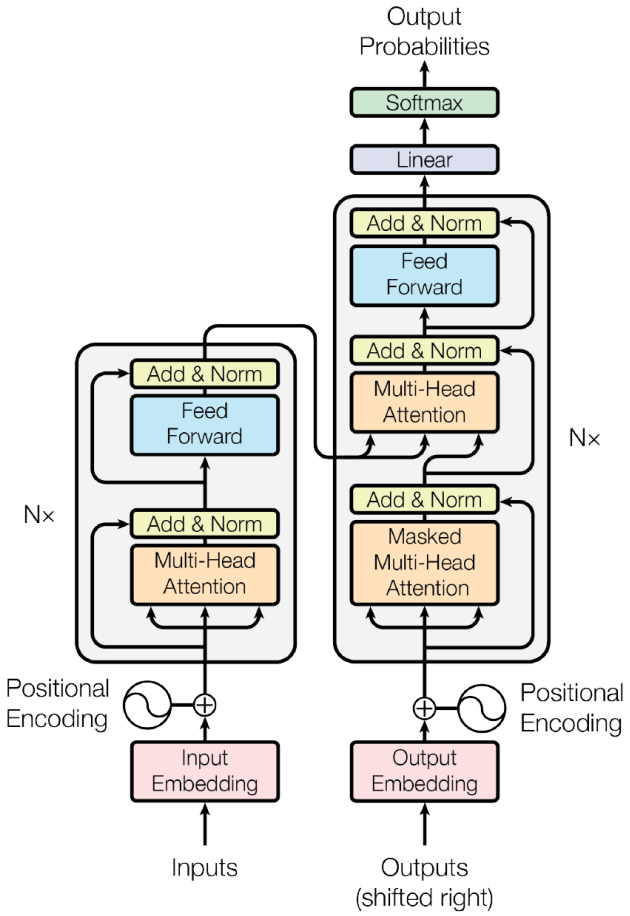


Figure 1: The Transformer - model architecture.

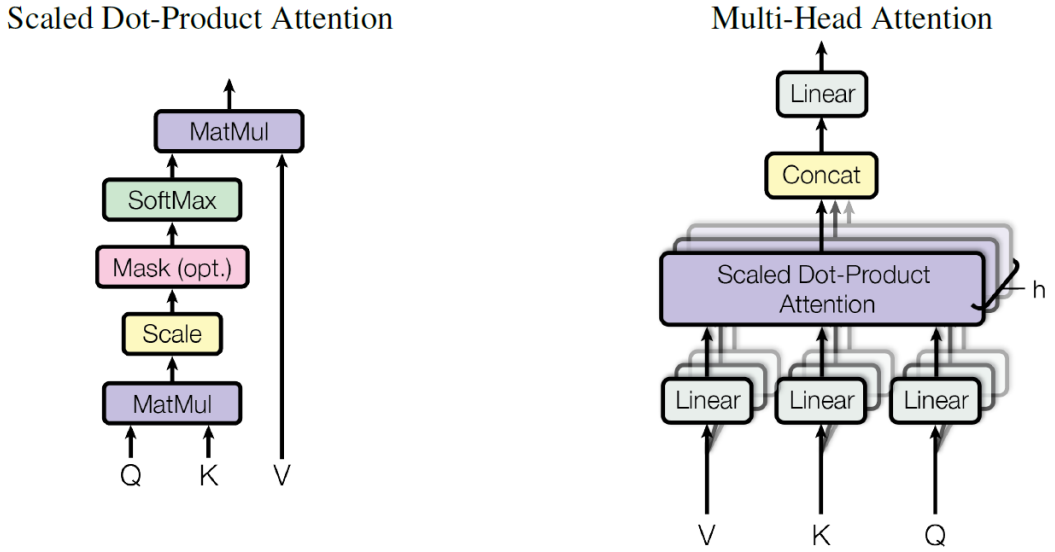


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Una función de atención se puede describir como la asignación de una consulta y un conjunto de pares clave-valor a una salida, donde la consulta, las claves, los valores y la salida son vectores. La salida se calcula como una suma ponderada de los valores, donde el peso asignado a cada valor se calcula mediante una función de compatibilidad de la consulta con la clave correspondiente.

3.3 Position-wise Feed-Forward Networks

In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2}$$

While the linear transformations are the same across different positions, they use different parameters from layer to layer. Another way of describing this is as two convolutions with kernel size 1. The dimensionality of input and output is $d_{\text{model}} = 512$, and the inner-layer has dimensionality $d_{ff} = 2048$.

- Positional Encodings

tokens in the sequence. To this end, we add "positional encodings" to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed. There are many choices of positional encodings, learned and fixed [9].

In this work, we use sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.

We also experimented with using learned positional embeddings [9] instead, and found that the two versions produced nearly identical results (see Table 3 row (E)). We chose the sinusoidal version because it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training.

En geometría, una transformación afín, mapa afín[o una afinidad es una función entre espacios afines que conserva puntos, líneas rectas y planos. Además, los conjuntos de rectas paralelas permanecen paralelos después de una transformación afín. Una transformación afín no conserva necesariamente los ángulos entre líneas o las distancias entre puntos, aunque sí conserva las proporciones de distancias entre puntos que se encuentran en una línea recta.