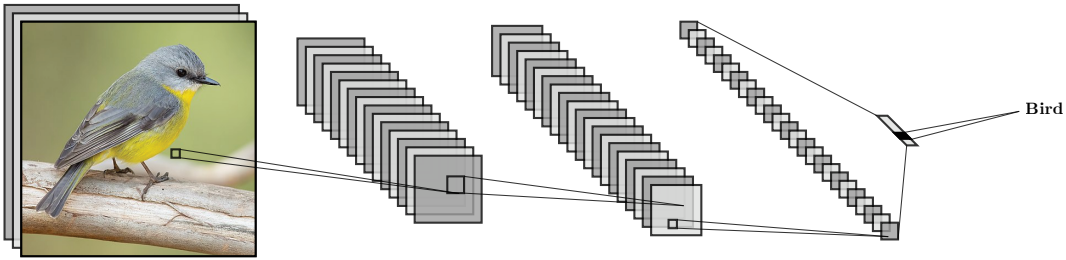
Attention mechanisms & Transformers

Motivación

Muchas tareas no necesitan de toda la entrada para predecir la salida.

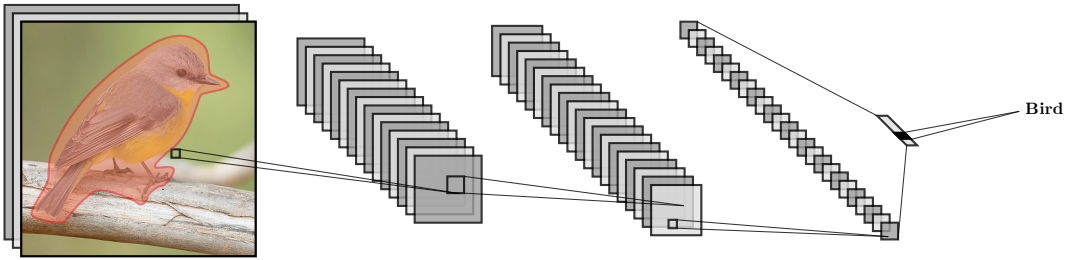
Ejemplo: Predecir la clase de una imagen.



Motivación

Muchas tareas no necesitan de toda la entrada para predecir la salida.

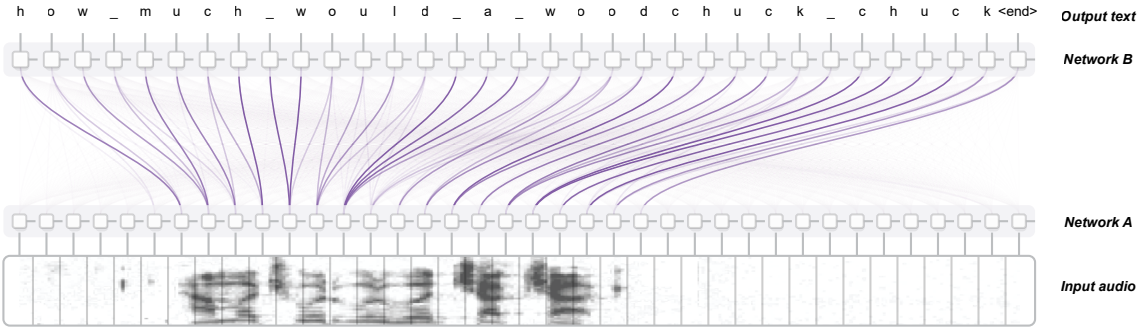
Ejemplo: Predecir la clase de una imagen.



Motivación

Muchas tareas no necesitan de toda la entrada para predecir la salida.

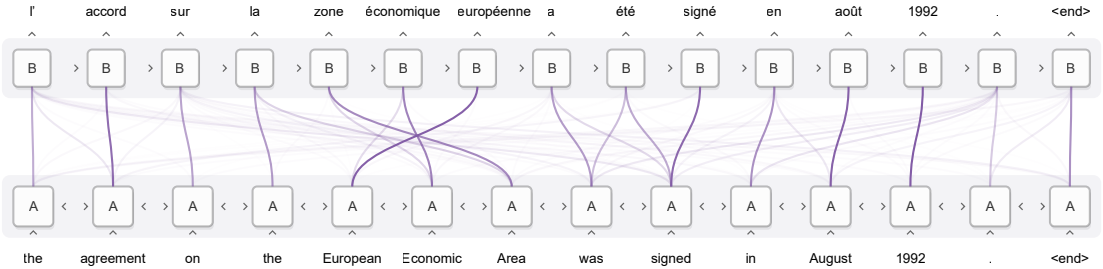
Ejemplo: Transformar audio en texto.



Motivación

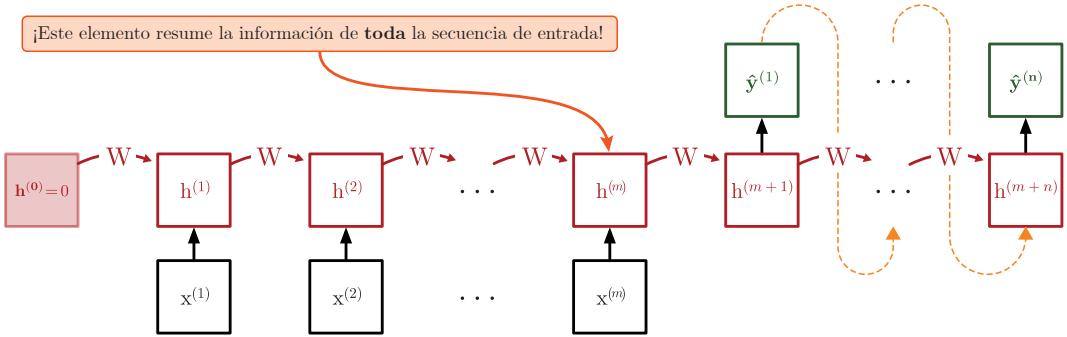
Muchas tareas no necesitan de toda la entrada para predecir la salida.

Ejemplo: Traducir entre idiomas.



Motivación

En tareas de Secuencia a Secuencia, las RNN condensan toda la información de la entrada en un único elemento. No es la mejor opción, sobre todo en largas secuencias.



En este contexto surgen los Transformers[1].

Esta nueva arquitectura:

- Mejora la eficiencia comptutacional de las RNN.
- Permite al modelo centrarse en partes concretas de la entrada para predecir la salida.
- Soluciona el problema de la memoria corto-placista de las RNN:
  - Permiten asociar palabras en una secuencia aunque estén muy separadas entre sí.

[1] Attention is all you need, Ashish Vaswani et al

Attention mechanisms

Attention mechanisms

Antes de comenzar a hablar de *Transformers*, es necesario entender el funcionamiento de su componente principal, los **attention mechanisms**.

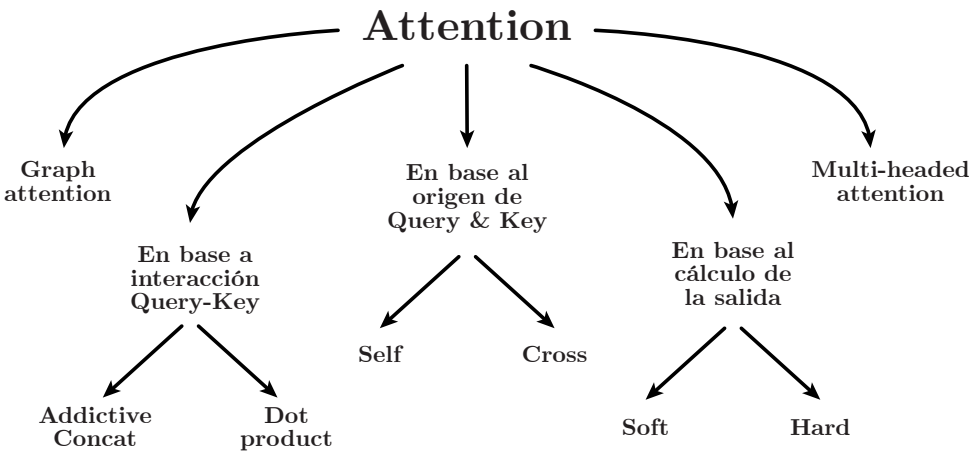
Definición

Los mecanismos de atención seleccionan que elementos de la(s) secuencia(s) de entrada son más importantes para predecir la secuencia salida.

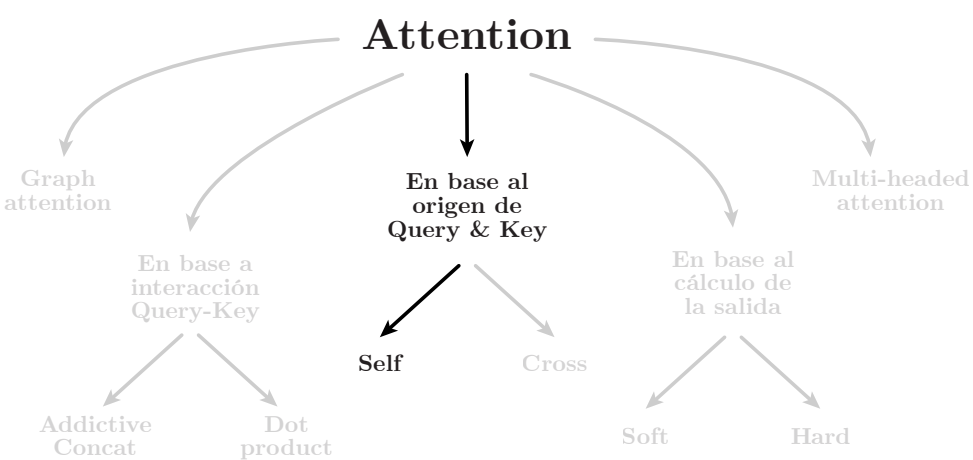
Detalles:

- La **entrada** de estos mecanismos espera **una o varias secuencias de datos**.
- Dentro de los *Transformers* se utilizan la llamada *Self-attention* pero, como verás a continuación, existen muchas otras variaciones.

Variaciones



Variaciones



Self-attention

Para comprender como funciona, imaginemos el siguiente escenario:

- **Secuencia de entrada:** $x_1, x_2, \ldots, x_t$
- **Secuencia de salida:** $y_1, y_2, \ldots, y_t$
- Todos los vectores tienen dimensión k.

Para producir cada vector y_i de la secuencia de salida, simplemente se obtiene la media ponderada de las entradas.

$$y_i = \sum_j w_{i,j} x_j$$

Donde la j recorre toda la secuencia y la suma de todos los $w_{.j}$ es igual a 1.

Self-attention

El peso $w_{i,j}$ **no es un parámetro**, como en una DNN, se deriva de una función sobre x_i y x_j.

La opción más sencilla para esta función es el **producto escalar**:

$$w_{ij}^{'} = \left\langle x_i{}^T, x_j \right\rangle$$

El peso representa la importancia de cada elemento de la entrada para el elemento actual.

- Nótese que x_i es el vector de entrada en la misma posición que el vector de salida actual.
- Para y_{i+1}, obtenemos una serie completamente nueva de productos escalares y una suma ponderada diferente.

Self-attention

El producto escalar anterior nos da valores entre $[-\inf, \inf]$.

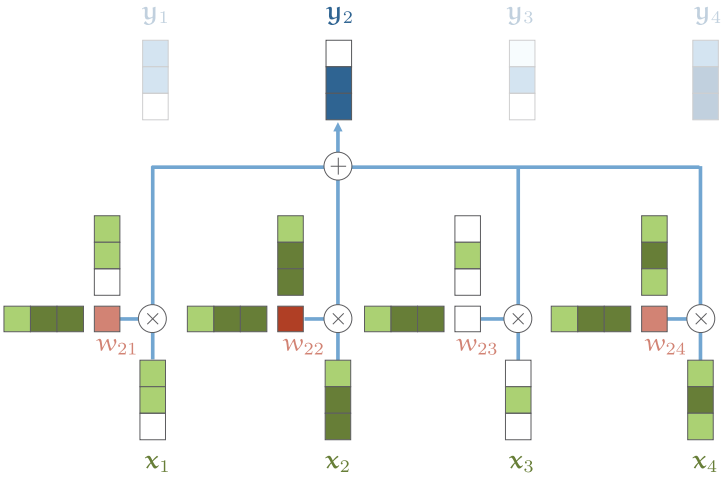
- Para obtener valores entre $[0, 1]$, aplicamos una *softmax*.
- De esta forma, para cada i, todos los j pesos sumarán 1.

Finalmente:

$$w_{ij} = \frac{\exp\ w_{ij}^{'}}{\sum_j \exp\ w_{ij}^{'}}$$

Self-attention

De forma gráfica (softmax omitida por simplicidad):



Cross-attention

Multihead-attention

Transformers