Maestría en Inteligencia Artificial Aplicada (MNA)

Transformers

All you need is Attention!

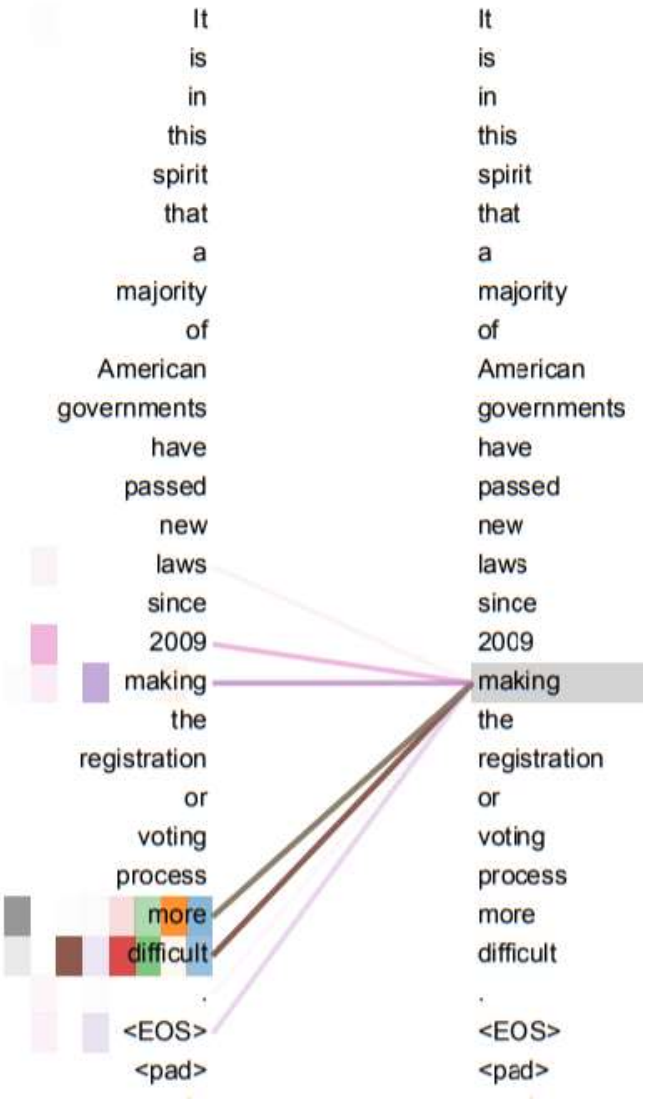
Procesamiento de Lenguaje Natural (NLP)

Luis Eduardo Falcón Morales

Mecanismo de Atención y modelos Transformer

Actualmente los modelos Transformer están revolucionando la manera en que se resuelven los problemas mediante inteligencia artificial.

El mecanismo llamado de Atención (Attention) ha resultado tener una gran impacto en la manera en que los problemas de procesamiento de lenguaje natural se pueden resolver. Más aún, dicho concepto se está aplicando a otras muchas áreas relacionadas con imágenes, audio y cualquier otro tipo de datos no estructurados.



Transformer

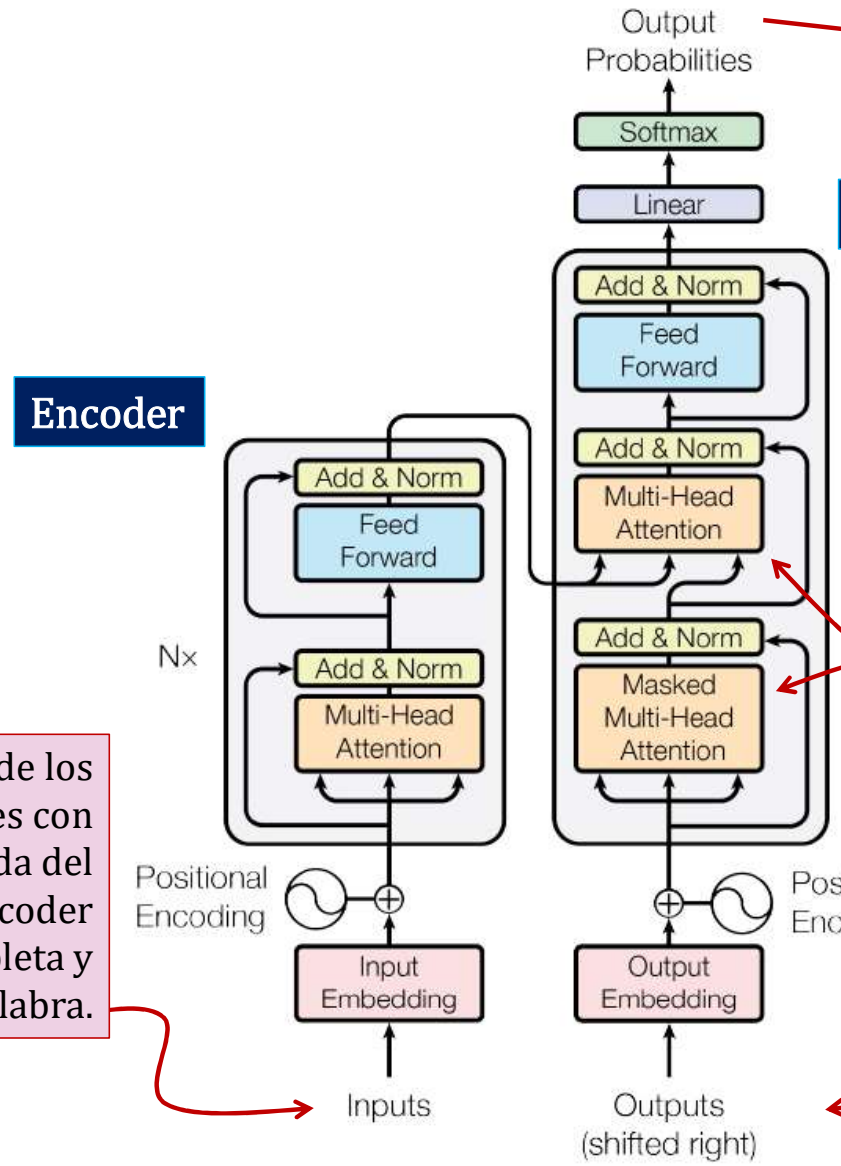
- Debido a que las RNN tienen el problema de que en enunciados muy largos la información de los primeros tokens se va diluyendo de un estado oculto al siguiente, se propone ahora eliminarlos y quedarse solamente con los módulos de Attention.

- Es decir, este modelo más simple, llamado Transformer, **ya no hará uso de bloques convolucionales o recurrentes.**

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

https://arxiv.org/abs/1706.03762



A diferencia de los modelos anteriores con RNNs, la entrada del enunciado al Encoder será una frase completa y no palabra por palabra.

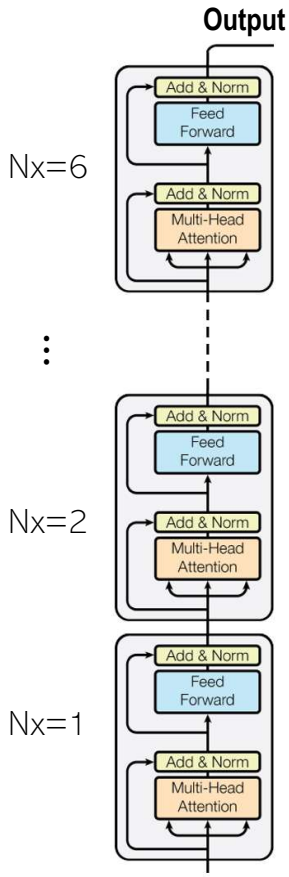
La salida (Output) del Decoder será nuevamente su propia entrada, hasta llegar al final del proceso <end>.

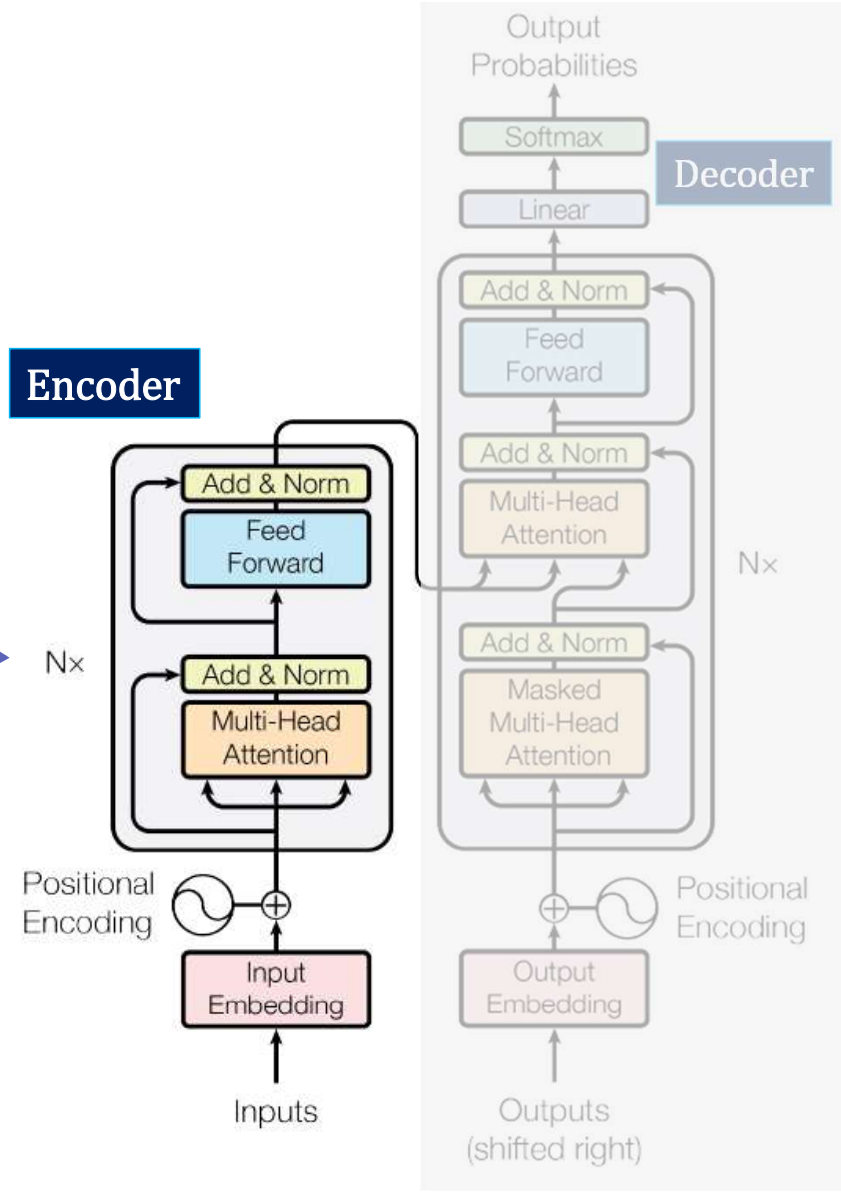
https://arxiv.org/abs/1706.03762

- El Encoder está formado básicamente por dos sub-capas (sub-layers), un Multi-Head Attention y un FeedForward, que se repiten Nx veces.

La salida de cada sub-capa en el Encoder es la entrada de la siguiente, hasta llegar a la última (Output), la cual será la entrada del Decoder.

En el artículo original se tomaron 6 sub-capas, pero este es un valor que puede modificarse.

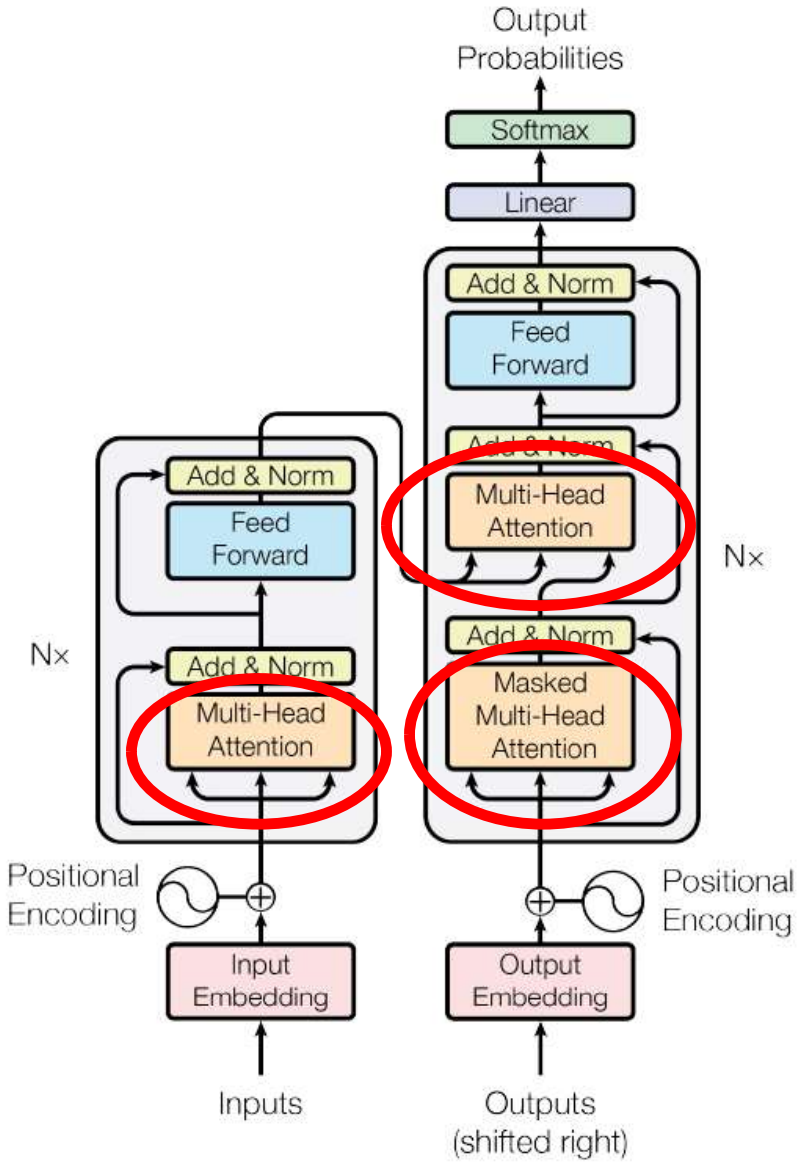




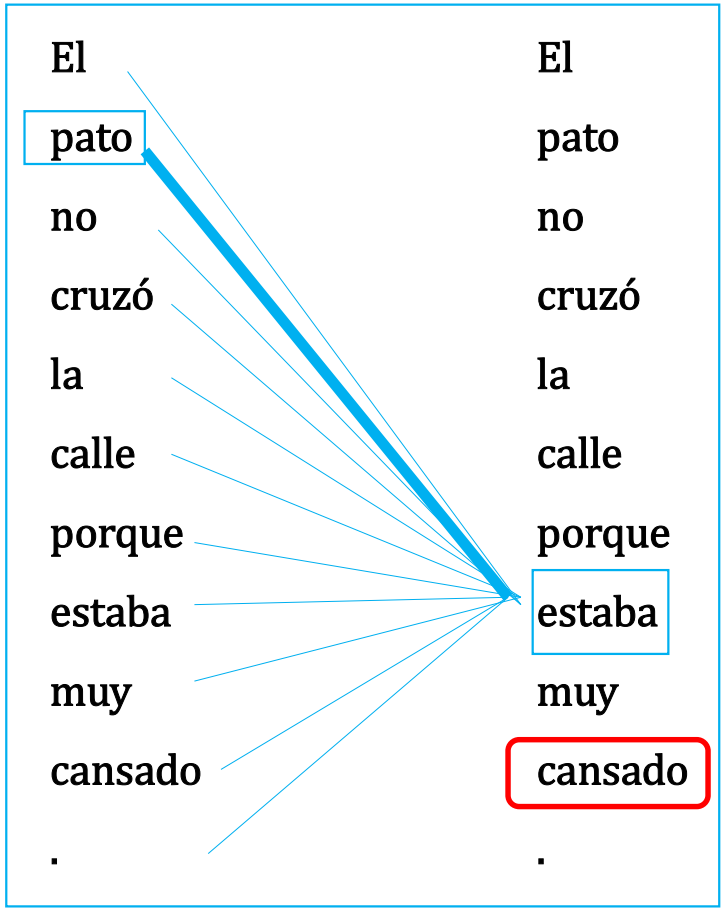
Multi-Head Attention : self-Attention

Los módulos “Multi-Head Attention” son la parte innovadora de los Transformers.

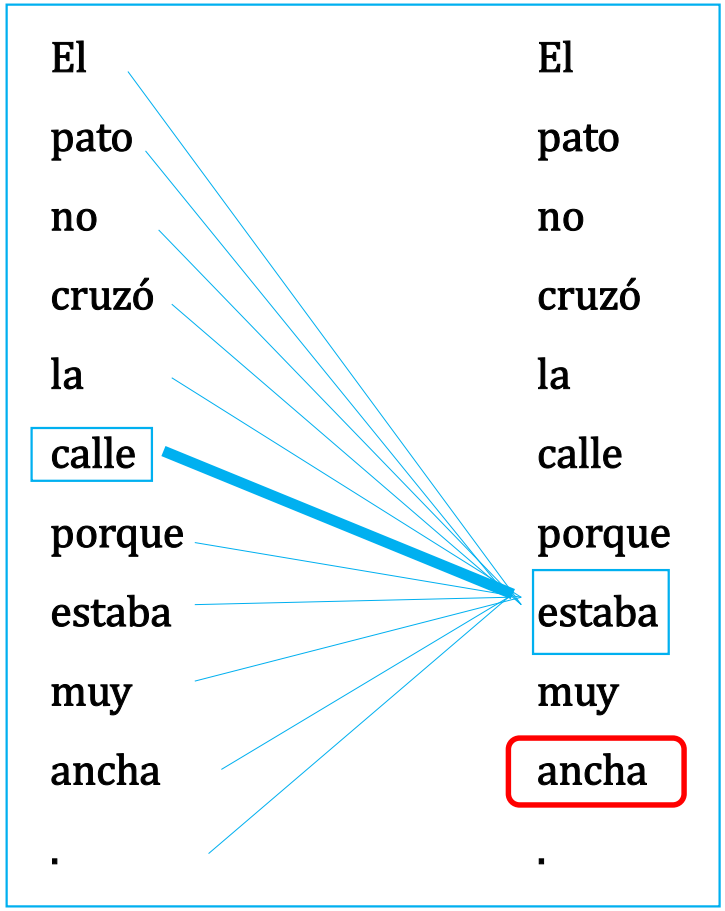
Veamos con mayor detalle cómo funcionan estos módulos y para ello necesitamos revisar primero el concepto llamado “Self-Attention” en el cual se basan.



Auto-Atención / Self-Attention



- En el bloque Encoder la sub-capa **Attention**, también llamado mecanismo **self-Attention**, ya que el objetivo es entender primero el enunciado de entrada, antes que intentar generar una respueta.
- Se busca la relación jerárquica de cada palabra (entidad) del enunciado (secuencia) con todas las demás.
- Por ejemplo, en estos dos ejemplos se muestra cómo el bloque self-Attention será capaz de identificar en cada caso, a quién se está refiriendo “estaba”: si al “pato” o a la “calle”.



El concepto de **Atención** está basado en tres matrices. Veamos cómo funcionan en el caso del Encoder:

Self-Attention : matrices Q, K, V

Q : query : consulta
K : key : clave
V : value : valor

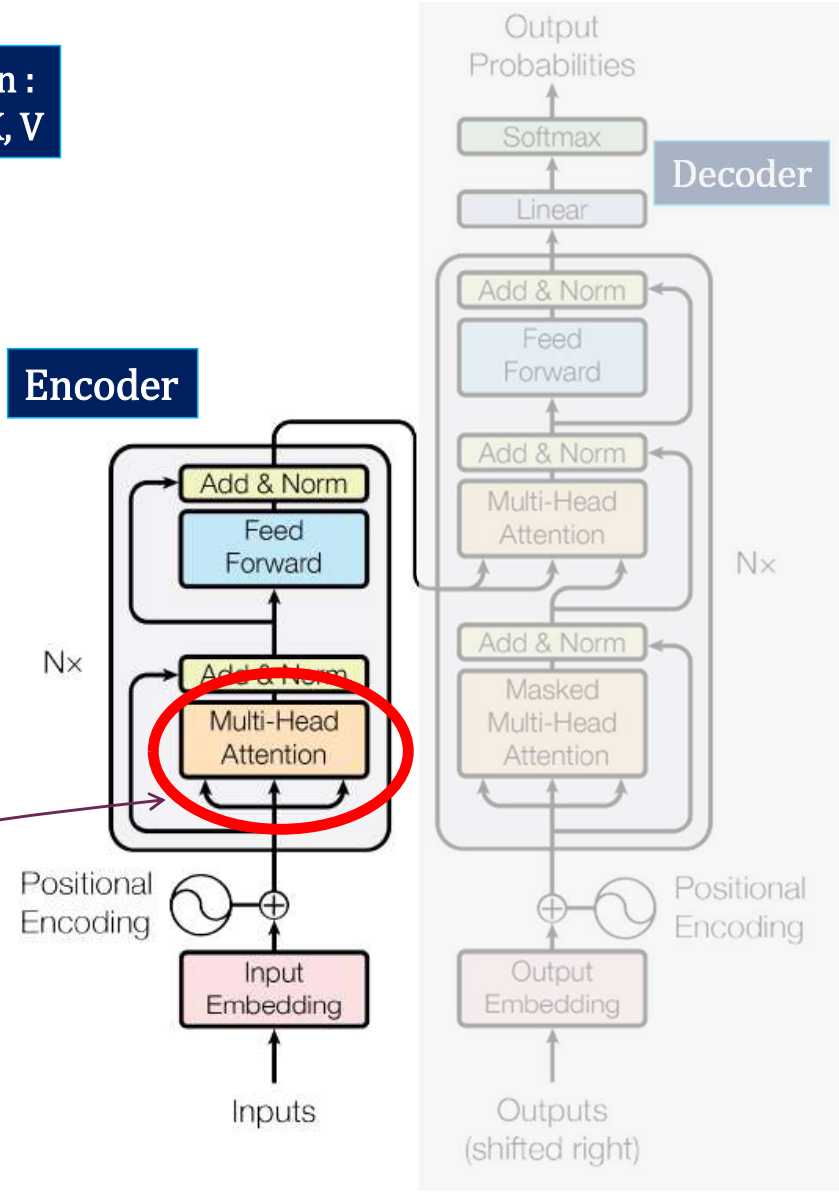
Cada capa Nx contiene estas tres matrices, las cuales **paralelizan** su aprendizaje (entrenamiento) a partir de estas otras tres matrices de pesos:

$$W^Q, W^K, W^V.$$

Las matrices Q, K y V del módulo Atención se generan con los siguientes productos, siendo X la matriz de vectores embebidos del texto de entrada:

$$XW^Q = Q$$
$$XW^K = K$$
$$XW^V = V$$

Las matrices W^Q, W^K y W^V se inicializan aleatoriamente y después se optimizan durante el entrenamiento.



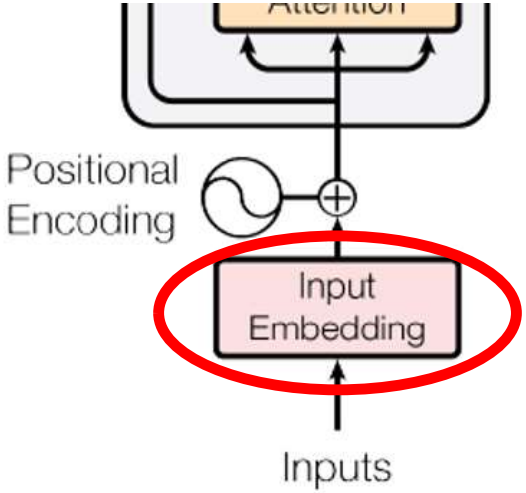
Por ejemplo, supongamos que la arquitectura transformer que se tiene es un modelo para realizar la traducción automática de frases en idioma español al idioma inglés y que la primera oración de entrada es “La vida es bella”.

Ahora, a cada una de estas palabras de entrada le asociaremos un vector embebido y formaremos con ellas la matriz X, donde el primer renglón es el vector embebido de “La”, el segundo renglón es el vector embebido de “vida” y así sucesivamente. En particular los autores del artículo original usaron vectores embebidos de dimensión 512, pero es un valor que puede modificarse a criterio del analista.

Entonces, en particular, la matriz X para la frase de entrada “La vida es bella” será de tamaño 4×512.

La cantidad de renglones de la matriz X estará asociada a la cantidad de palabras/tokens del enunciado de entrada.

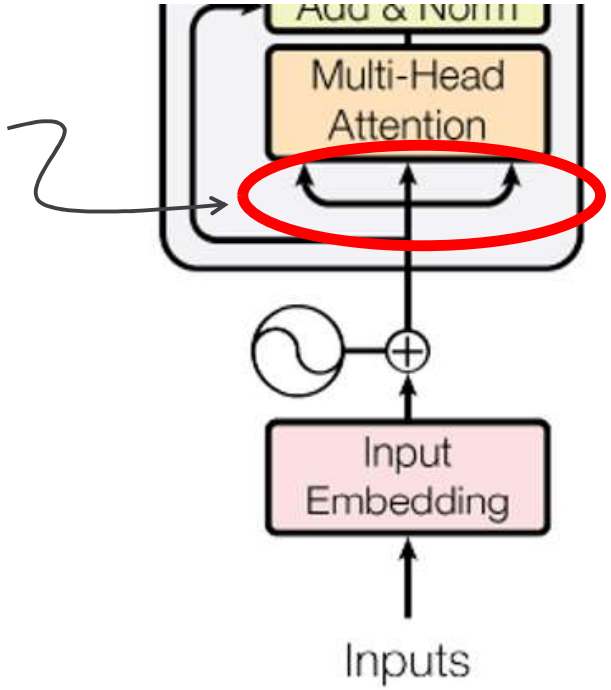
$X_{4\times512}$: matriz de entrada de vectores embebidos



Generamos ahora las matrices llamadas *query* Q, *key* K y *value* V, al multiplicar la matriz de vectores embebidos de entrada X por las matrices de pesos W^Q, W^K y W^V, respectivamente.

Es decir, para nuestro ejemplo:

$$X_{4\times512}W^{Q}_{512\times64} = Q_{4\times64} \quad \text{: Query matrix}$$
$$X_{4\times512}W^{K}_{512\times64} = K_{4\times64} \quad \text{: Key matrix}$$
$$X_{4\times512}W^{V}_{512\times64} = V_{4\times64} \quad \text{: Value matrix}$$



Los autores del artículo consideraron matrices de pesos W con 64 columnas, pero es algo que puede ajusarse a criterio del analista. La cantidad de renglones de estas matrices de pesos está determinada por la dimensión de los vectores embebidos elegida, 512 en este caso.

Las matrices W^Q, W^K y W^V se inicializan aleatoriamente y después se optimizan durante el entrenamiento.

Las matrices *query* Q, *key* K y *value* V son las que estarán extrayendo la información relevante en los Transformers, a través del concepto de Attention.

$X_{4\times512}$:

matriz de entrada de vectores embebidos

$$X_{4\times512}W^{Q}_{512\times64} = Q_{4\times64} \quad : \textit{Query} \text{ matrix}$$

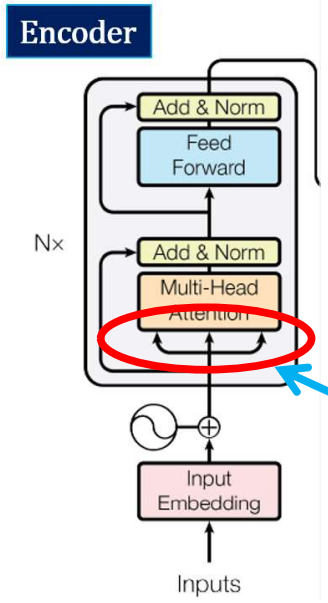
$$X_{4\times512}W^{K}_{512\times64} = K_{4\times64} \quad : \textit{Key} \text{ matrix}$$

$$X_{4\times512}W^{V}_{512\times64} = V_{4\times64} \quad : \textit{Value} \text{ matrix}$$

Observa que ahora, con este tratamiento matricial, la entrada de una frase será con todas las palabras/tokens a la vez y no de manera secuencial o palabra por palabra, como en las RNN.

De estas expresiones ya podemos inferir ahora lo siguiente de las matrices Q, K y V:

- El primer renglón de las matrices Q, K y V está asociado a los vectores *query*, *key* y *value* del token “La”.
- El segundo renglón de las matrices Q, K y V está asociado a los vectores *query*, *key* y *value* del token “vida”.
- El tercer renglón de las matrices Q, K y V está asociado a los vectores *query*, *key* y *value* del token “es”.
- El cuarto renglón de las matrices Q, K y V está asociado a los vectores *query*, *key* y *value* del token “bella”.



La vida es bella

$$X_{4\times512}W^{Q}_{512\times64} = Q_{4\times64}$$

$$X_{4\times512}W^{K}_{512\times64} = K_{4\times64}$$

$$X_{4\times512}W^{V}_{512\times64} = V_{4\times64}$$

query → La vida es bella

key → La vida es bella

value → La vida es bella

Con cada iteración en el proceso de entrenamiento mediante un aprendizaje supervisado, los pesos de las matrices W^Q, W^K y W^V se estarán aprendiendo y con ello se estarán generando las matrices Q, K y V, las cuales nos ayudarán a entender tanto las oraciones de entrada, como las que se vayan generando.

Así, **cada matriz Q, K y V estará procesando el mismo enunciado de entrada de forma diferente**, es decir, **con un objetivo/tarea diferente.**

All you need is ... Attention

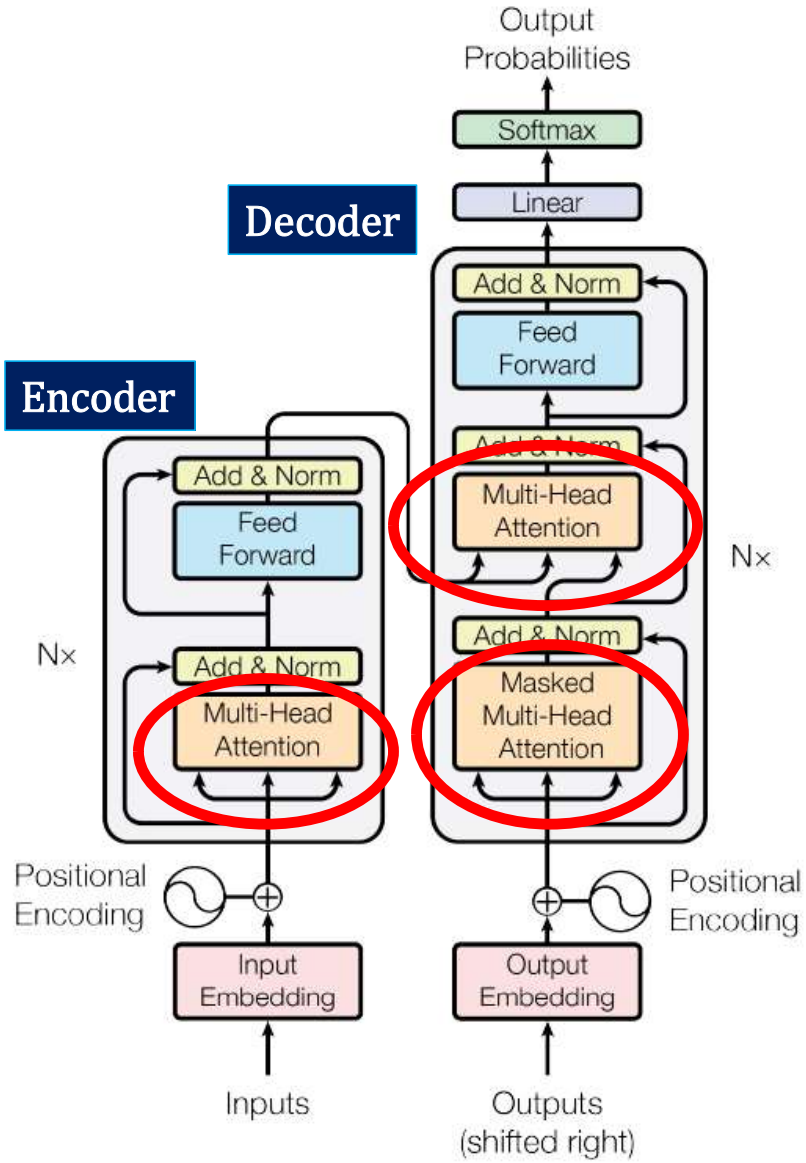
Veamos ahora cómo el concepto de **Atención** surge de estas tres matrices Q, K, V. Su definición es como sigue:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Observamos que este nuevo concepto llamado "**Attention**" está basado solamente en productos matriciales, replicado Nx veces, lo cual lo hará más rápido y más eficiente en cuanto al uso de memoria.

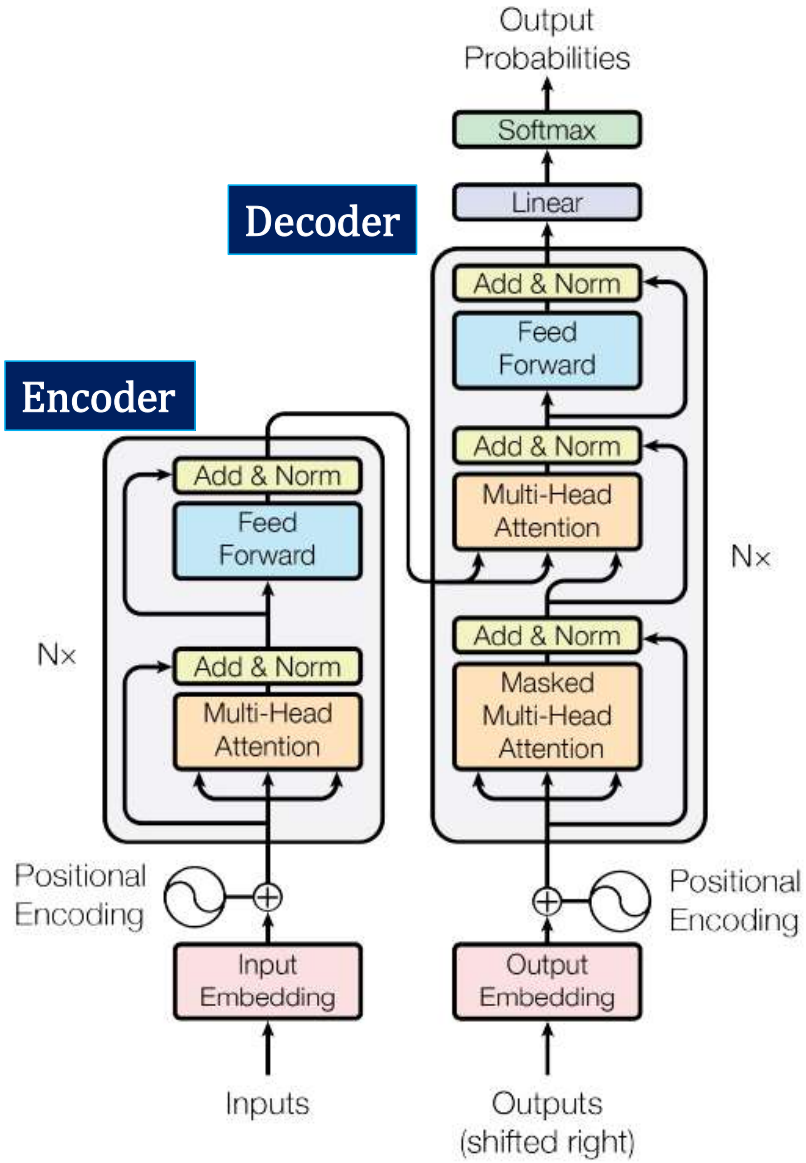
Estos módulos sustituyen a las capas recurrentes y convolucinales de los modelos RNN y CNN, respectivamente.

Además, el problema del desvanecimiento o explosión de los gradientes también se aminora, ya que los módulos de Attention involucran operaciones matriciales sin uso de derivadas parciales durante backpropagation y solamente los módulos "Feed-Forward" y "Linear" siguen usando capas neuronales.



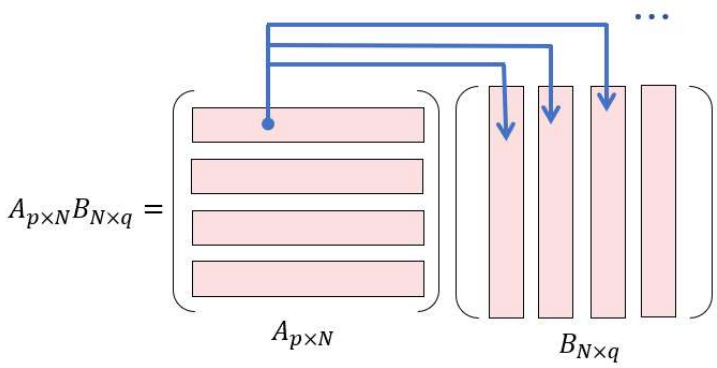
Arquitectura Transformer

- En el bloque Decoder hay dos sub-capas Attention: la primera procesa la secuencia de salida del propio bloque con un self-Attention y la segunda procesa la secuencia de salida del Encoder, con la salida del self-Attention previo.
- La primera sub-capa self-Attention del Decoder funciona de manera análoga al del Encoder, pero con el propio texto que se está generando.
- La segunda sub-capa Attention del Decoder busca establecer la relación jerárquica de las entidades de la secuencia de salida del Encoder, con las entidades de la secuencia de salida de su self-Attention previo.
- En el caso por ejemplo de la traducción de un texto de un idioma a otro, esta arquitectura permitirá ir más allá que la simple traducción “palabra por palabra”.



Productos Matriciales, Productos Interiores y Similaridad

Antes de seguir adelante recordemos que un producto matricial se puede ver como una serie de productos interiores entre vectores (de hecho, vectores renglón por vectores columna).

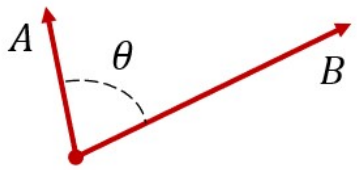


Con el primer renglón de la matriz A obtenemos el producto interior (o producto punto) con cada una de las columnas de la matriz B.

Y se repite lo mismo con cada uno de los renglones de A.

Así, al "producto matricial" de A con B también lo llamaremos simplemente el "producto interior" (o producto punto) entre A y B.

Y por otro lado recordemos también que el producto interior entre dos vectores se puede interpretar como una medida de **qué tan similares o relacionados están**, esto por la relación con el coseno del ángulo entre ellos:

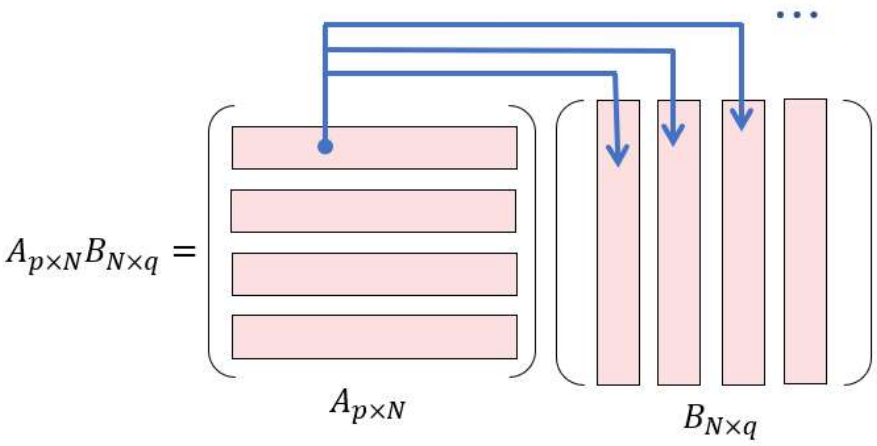


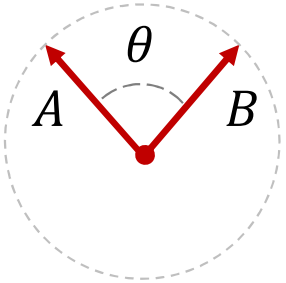
$$cos(\theta) = \frac{\langle A, B\rangle}{\|A\|\|B\|}$$

$$-1 \leq cos(\theta) \leq 1 \begin{cases} \approx +1 : \text{"similares"} \\ \approx 0 : \text{"sin relación"} \\ \approx -1 : \text{"opuestos"} \end{cases}$$

$\langle A, B\rangle$: producto interior entre los vectores
$\|A\|$: norma o magnitud del vector

En particular, si los vectores (renglones de A y columnas de B) son unitarios, entonces el producto interior entre los vectores (renglones y columnas) se puede interpretar como una medida de similaridad dada directamente por el coseno del ángulo entre los vectores:





vectores unitarios:

$$\|A\| = \|B\| = 1$$

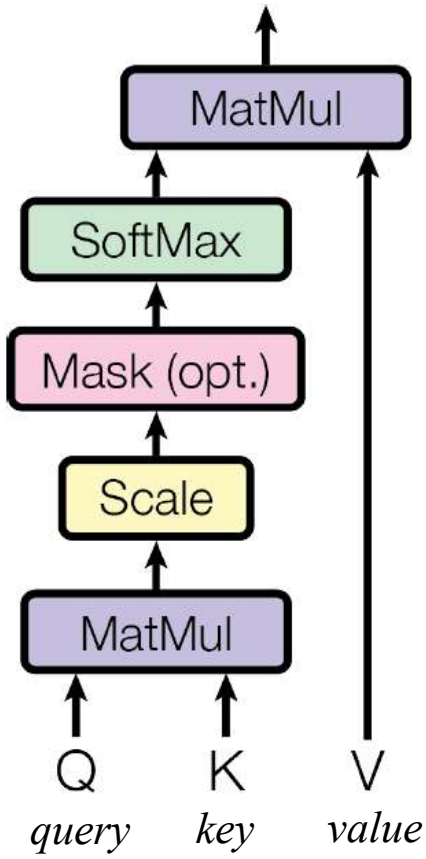
La expresión $cos(\theta) = \frac{\langle A,B\rangle}{\|A\|\|B\|}$ se reduce a:

$$\langle A, B\rangle = cos(\theta)$$

De esta manera podemos decir que **mientras más grande sea el producto interior entre cada par de vectores** (cada renglón de A con cada columna de B), **más similares o relacionados estarán dichos vectores.**

scaled dot-product Attention :
Attention como un producto interior escalado

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



Veamos paso a paso lo que nos está diciendo esta expresión de Attention.

En el artículo original hacen referencia a esta expresión como la **Atención del producto interior escalado** (scaled dot-product Attention).

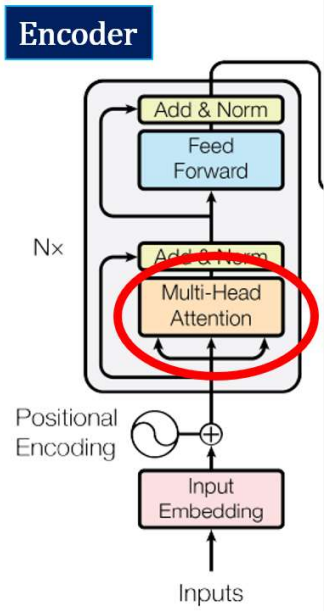
Nos apoyaremos en esta imagen de la derecha del artículo original para ir explicando cada operación y poder entender la manera en que funciona el concepto de Attention.

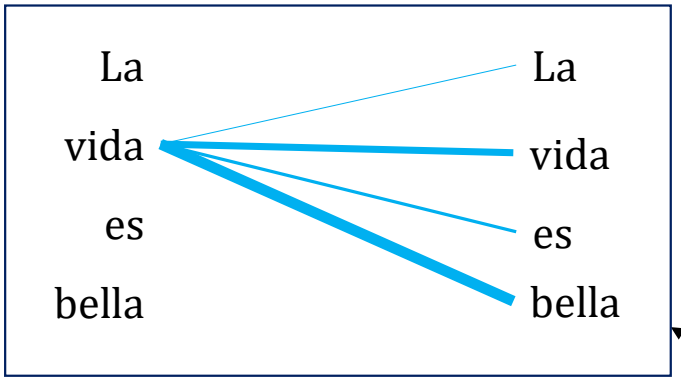
Seguiremos haciendo la analogía con el ejemplo de traducción de una frase en español como entrada al Transformer.

https://arxiv.org/abs/1706.03762

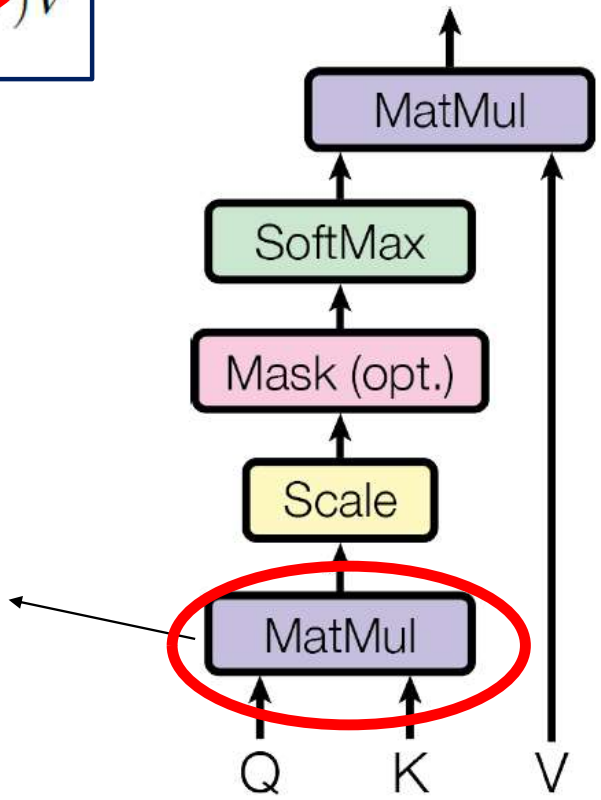
scaled dot-product Attention :
Attention como un producto interior escalado

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



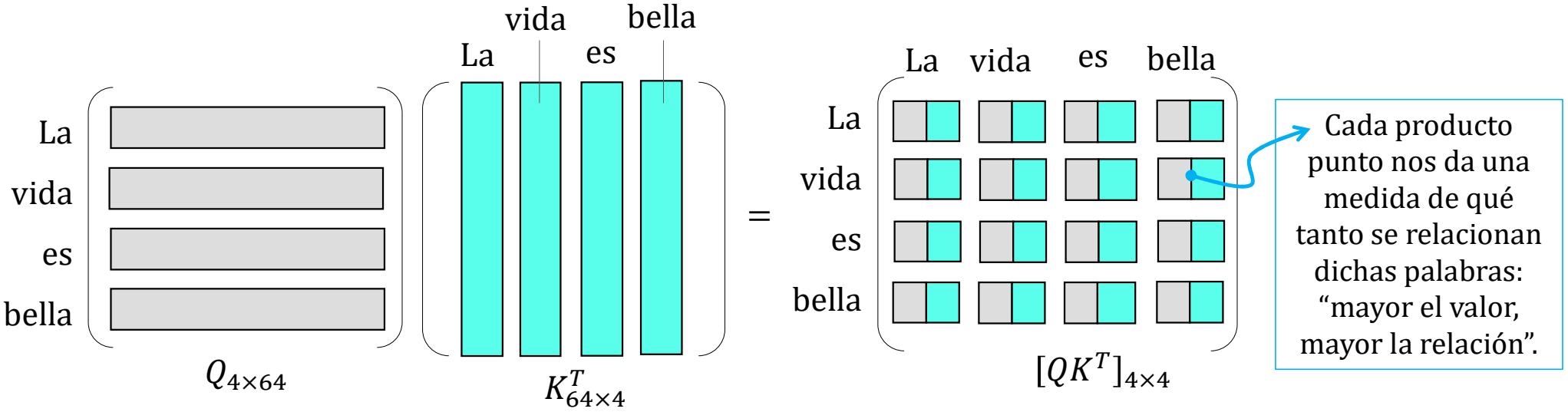


QK^T



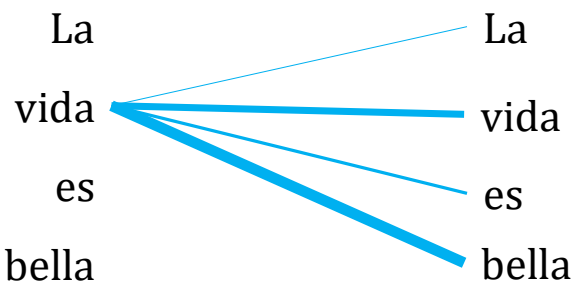
Con el producto punto QK^T en los Nx módulos del Attention del Encoder se está tratando de entender primero el texto de entrada, en lugar de querer directamente realizar una traducción literal palabra por palabra, la cual sabemos nos llevaría a un resultado muy pobre.

¿cómo funciona el producto QK^T?



Así, mediante el producto punto QK^T, el mecanismo de auto-Atención (self-Attention) está buscando la manera en que cada palabra de una oración se relaciona con todas las demás. Es decir, **es un tipo de correlación entre todas las palabras de la oración**.

Podemos decir que en esta etapa de auto-Atención dentro del Encoder se está tratando de entender primero el texto de entrada, antes de querer generar una salida, en este caso la traducción de dicha frase.



Aquí solamente se muestra "vida" contra todas, pero en realidad son todas contra todas.

scaled dot-product Attention :
La Attention como un producto interior escalado

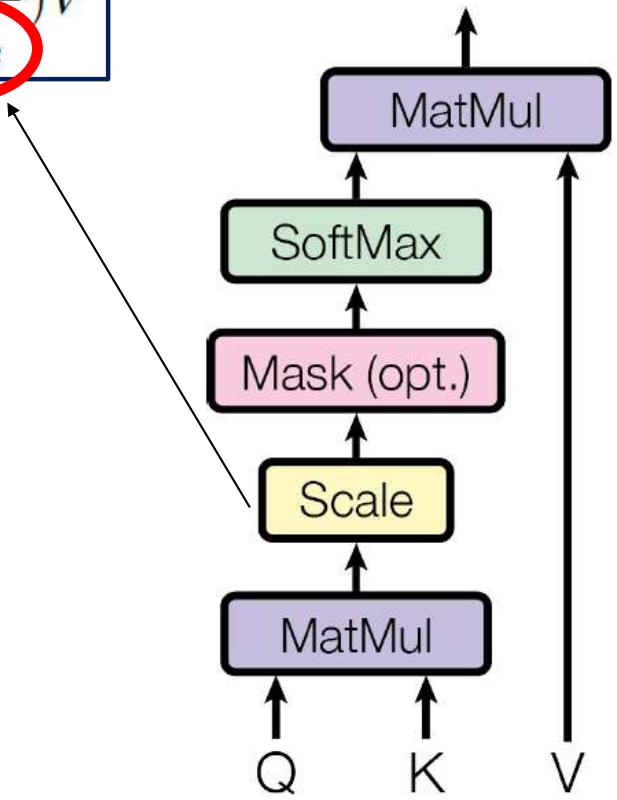
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Se escala el producto QK^T con respecto a la dimensionalidad de los vectores renglón de la matriz *key* K.

Esto nos ayuda a estabilizar la convergencia numérica durante el proceso de entrenamiento.

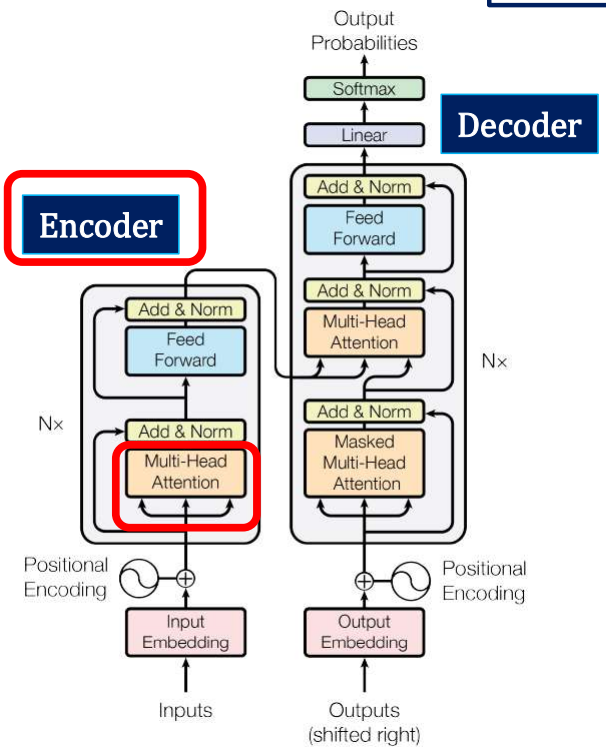
Además ayuda a que el gradiente de las capas FeedForward, Linear y Softmax no se desvanezca tan rápido en los valores grandes del producto QK^T.

En nuestro ejemplo de la traducción de la frase "La vida es bella", la matriz *key* es de tamaño $K_{4\times64}$, por lo que en este caso $\sqrt{d_k} = \sqrt{64} = 8$.

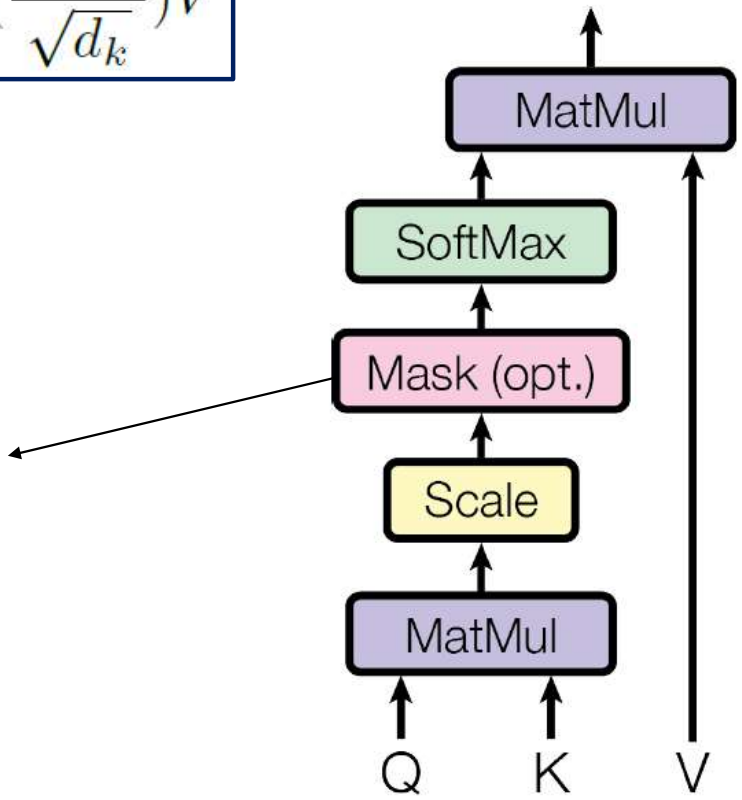


scaled dot-product Attention :
La Attention como un producto interior escalado

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

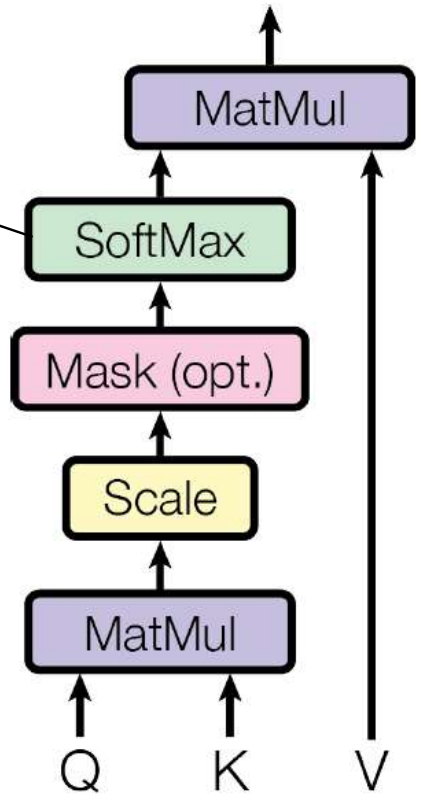


En el módulo Attention del Encoder no se requiere la capa “Mask”.



scaled dot-product Attention :
La Attention como un producto interior escalado

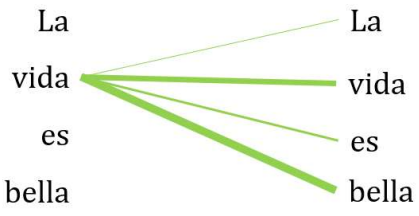
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

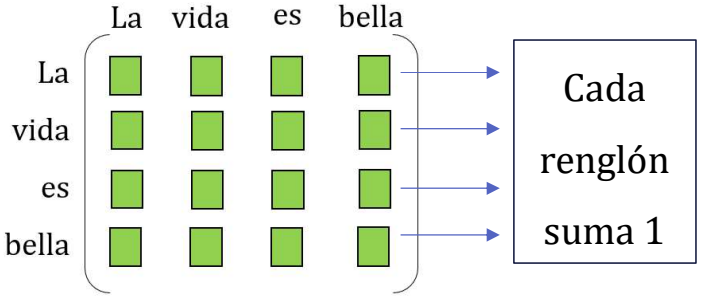


La función Softmax nos permite darle ahora un significado probabilístico a la relación entre las entidades (palabras) de ambas secuencias (enunciados).
Se está normalizando la llamada "matriz de puntuaciones" entre Q y K con la función softmax: nos dice "a qué le está poniendo atención cada palabra".

En el problema de traducción de español a inglés, estos pesos indican qué tan bien se está "alineando"/"entendiendo"/"relacionando" cada palabra, con todas las otras de la misma frase.

Relación normalizada:

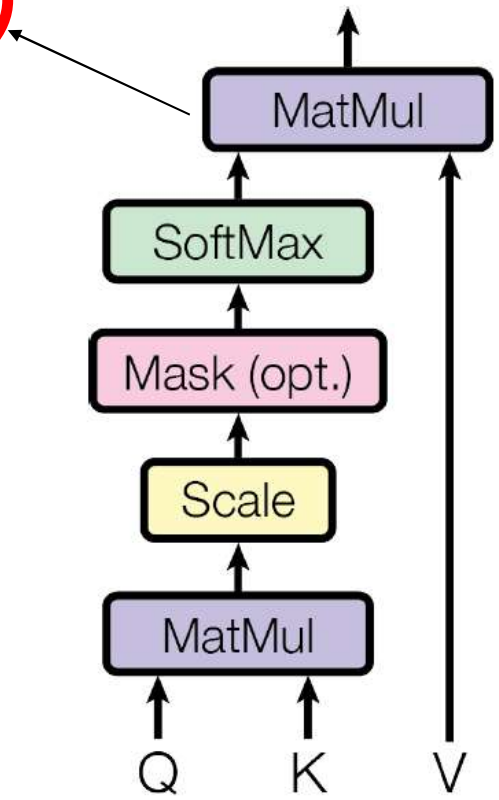




Aquí solamente se muestra "vida" contra todas, pero en realidad son todas contra todas.

scaled dot-product Attention :
La Attention como un producto interior escalado

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



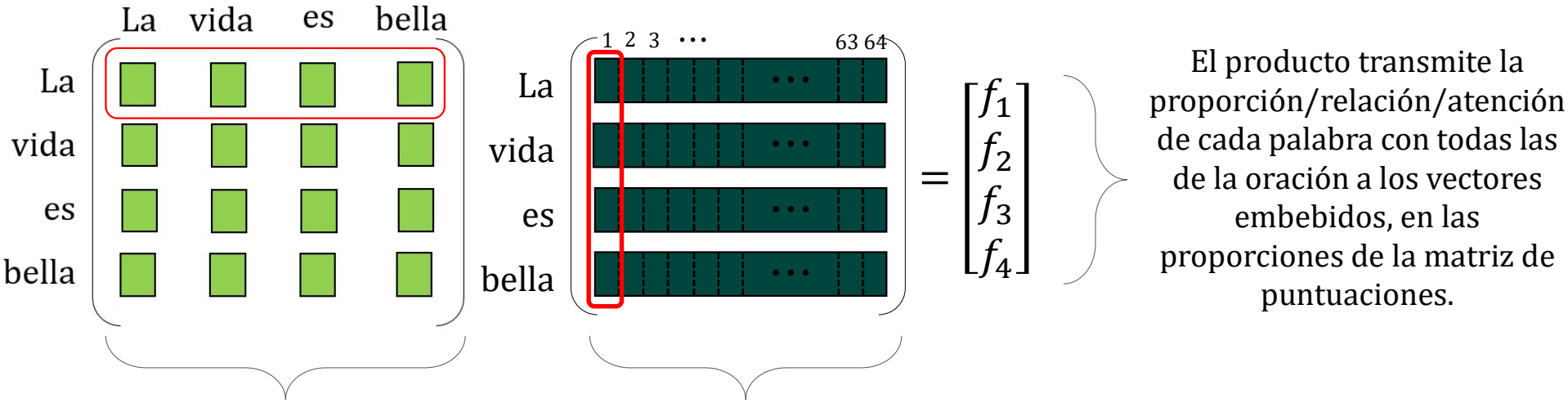
Finalmente se obtiene el producto punto de la matriz de puntuaciones con la matriz de valores V, para generar la llamada matriz de Atención (Attention matrix).

Para el ejemplo que estamos considerando con la oración “La vida es bella”, se tendría un producto matricial de la matriz de puntuaciones con la matriz *value*, con las siguientes dimensiones de cada uno:

$$F = \overbrace{\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)}^{4\times 4} \overbrace{V}^{4\times 64} = \begin{matrix} \text{La} \\ \text{vida} \\ \text{es} \\ \text{bella} \end{matrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix}_{4\times 64}$$

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

$$F = \overbrace{\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)}^{4\times 4} \overbrace{V}^{4\times 64} = \begin{matrix} \text{La} \\ \text{vida} \\ \text{es} \\ \text{bella} \end{matrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix}_{4\times 64}$$



El producto transmite la proporción/relación/atención de cada palabra con todas las de la oración a los vectores embebidos, en las proporciones de la matriz de puntuaciones.

La matriz de puntuaciones nos da el porcentaje de relación de cada palabra con todas las demás. Por ejemplo, el primer renglón nos da las probabilidades de relación de “La”, con cada palabra de la oración.

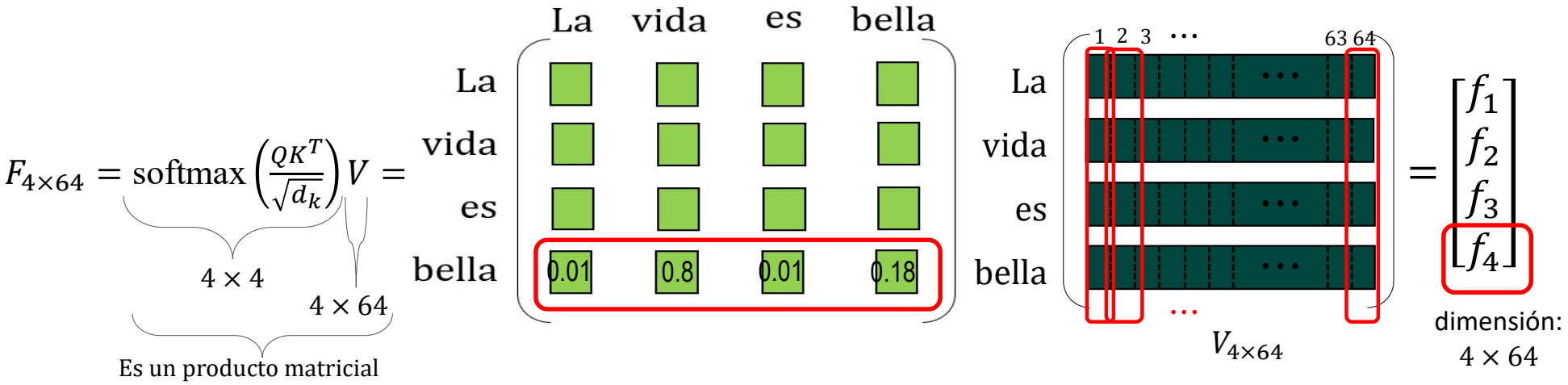
La matriz value K nos da la información embebida de cada palabra de la oración.

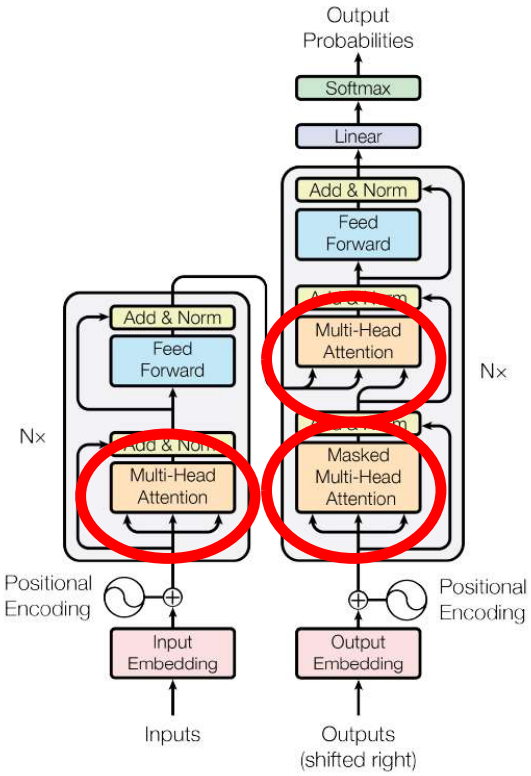
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Por ejemplo, supongamos que la matriz de puntuaciones tiene los valores indicados en la imagen de abajo para la palabra “bella”.

Entonces, dichas proporciones que tiene “bella” con todas las palabras de la oración (0.01, 0.80, 0.01 y 0.18, respectivamente), es transmitida a todos los vectores embebios para generar f_4. En particular, observa que “bella” tiene la mayor relación, o le presta mayor Atención, a la palabra “vida” con un 80%, indicando que el adjetivo de “bella” se refiere al sustantivo “vida” y que dicha información es muy relevante para entender bien el significado de la oración.

Así, f_4 contiene la información embebida que tenía la matriz *value* $V_{4\times64}$ para cada palabra de la oración, pero ahora en las proporciones dadas por la forma en que se relacionan (en que prestan Atención) con la palabra “bella”. El mismo razonamiento aplicará para las otras tres coordenadas de F.



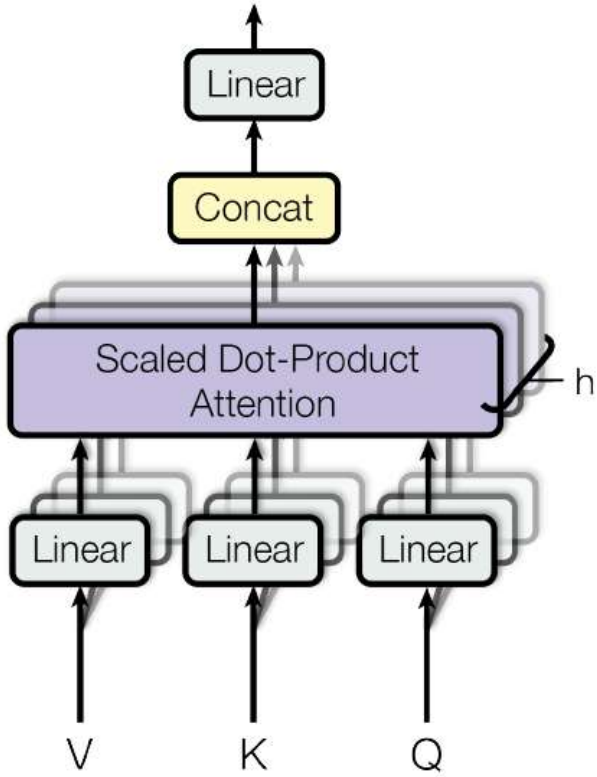


En realidad, para obtener la Attention del modelo Transformer, los autores proponen segmentar primero los enunciados en h=8 partes (heads) de dimensión 64 cada uno (las dimensiones de Q, K y V), procesar cada uno por separado y después concatenarlos de nuevo en dimensión 512. Formalmente son proyecciones lineales en h=8 subespacios de dimensión 512/8=64.

Esto permite encontrar más relaciones de similaridad entre "grupos de palabras" (proyecciones en subespacios) y evitar que la relación predominante opaque a las demás.

Como el proceso es paralelizable y los subespacios son de menor dimensión, aunque se tienen más subespacios que procesar, el costo computacional sigue siendo casi el mismo que procesar el espacio original de mayor dimensión, 512.

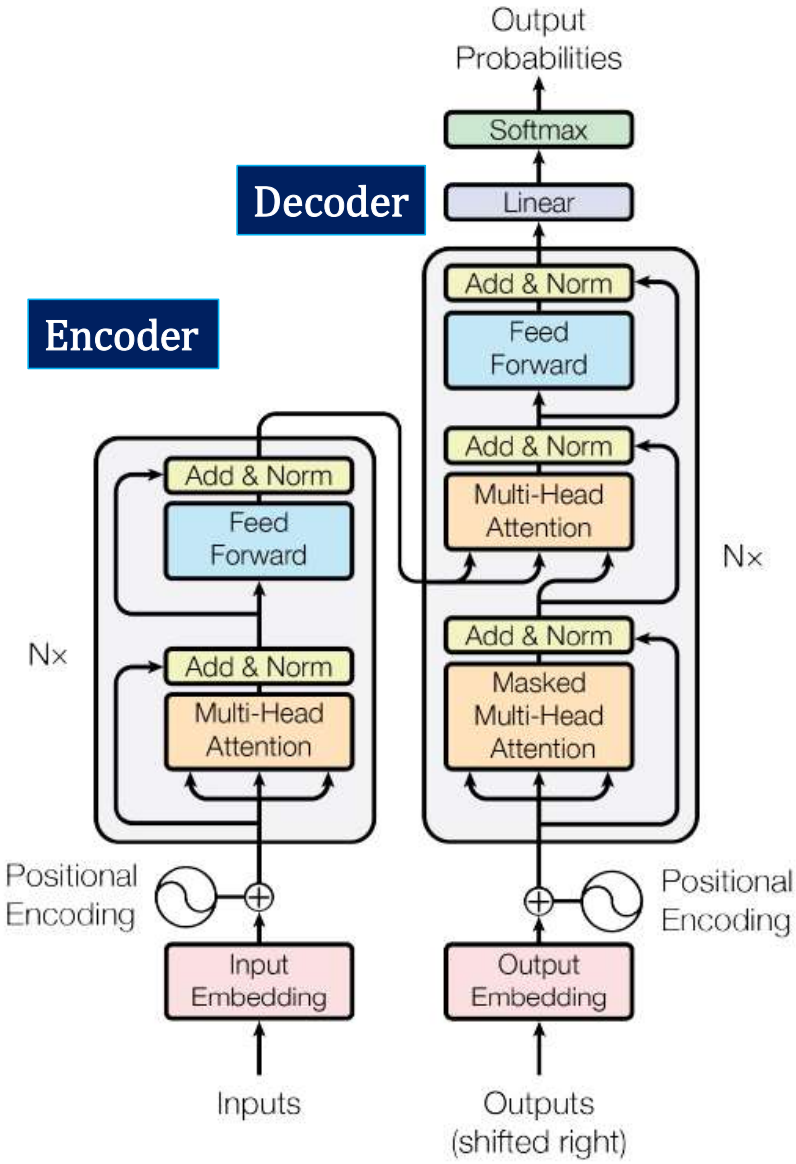
Multi-Head Attention



- Ya vimos que el Encoder está formado por dos sub-capas (sub-layers) --un Attention y un FeedForward-- que se repiten Nx=6 veces en el artículo original, pero podría variarse. Estas sub-capas tienen un comportamiento análogo en el Decoder.

- En cada sub-capa se aplican conexiones residuales (residual connections →) a la manera de la red ResNet CNN de Microsoft (2015): la salida de cada una de estas sub-capas se suma y normaliza con la información que traía previa a entrar a dicha sub-capa. Esto permite mantener la información relevante actualizada a lo largo de la red, es decir, a lo largo del análisis secuencial (tiempo). Por lo tanto, durante la etapa de entrenamiento y en particular durante el proceso de propagación hacia atrás (backpropagation), los pesos se mantendrán actualizados, ayudando a combatir el problema del desvanecimiento o explosión de los gradientes. Además, el proceso será bi-direccional, a diferencia del Decoder, que es uni-direccional.

- Para facilitar la conexión entre todas las sub-capas, las salidas de las capas Embedding, Attention y FeedForward serán todas de dimensión 512.



Feed-Forward layers & Residual Connections

Las **capas Feed-Forward** consisten de dos redes totalmente conectadas (Fully-Connected), donde la primera de ellas tiene como función de activación la ReLU:

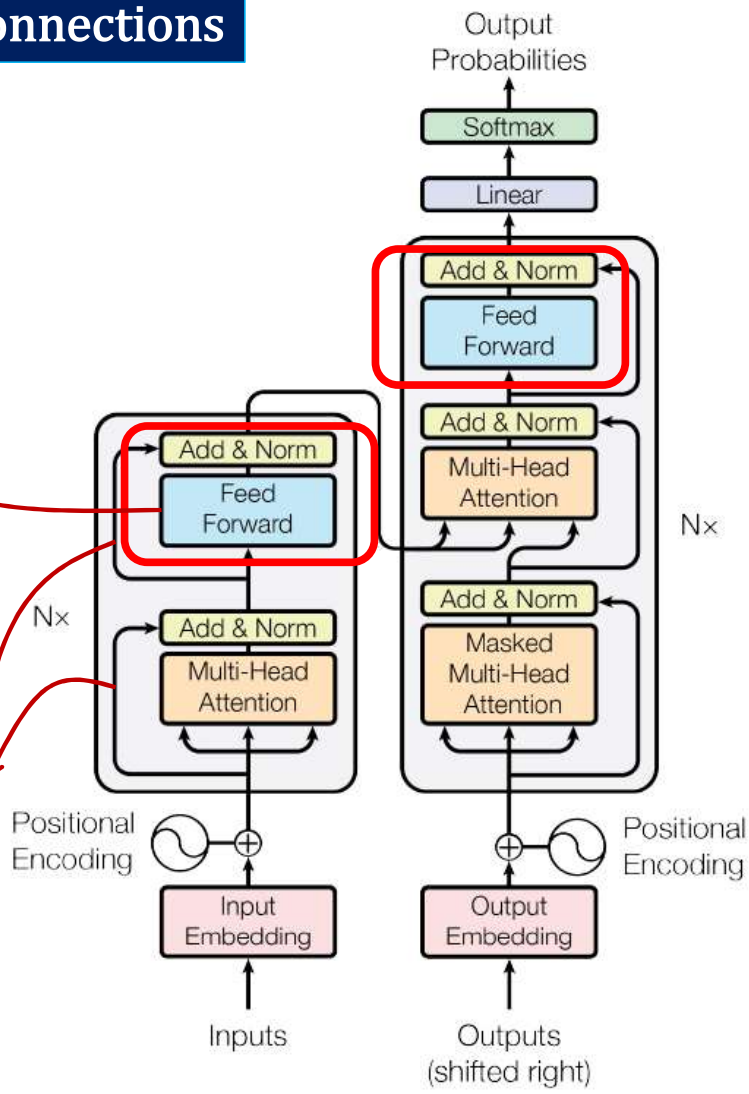
$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

Hay que considerar que la entrada a esta capa es elemento (palabra) por elemento (palabra) de manera independiente y no todo el enunciado completo.

Cada una de las Nx iteraciones en el Encoder y el Decoder tiene sus propias 2 capas Fully-Connected.

Las **capas residuales** después del FeedForward se aplican a la manera del modelo CNN ResNet, sumando y normalizando:

Add & Norm : $LayerNorm\big(x + sublayer(x)\big)$



Capa Dropout : Regularization

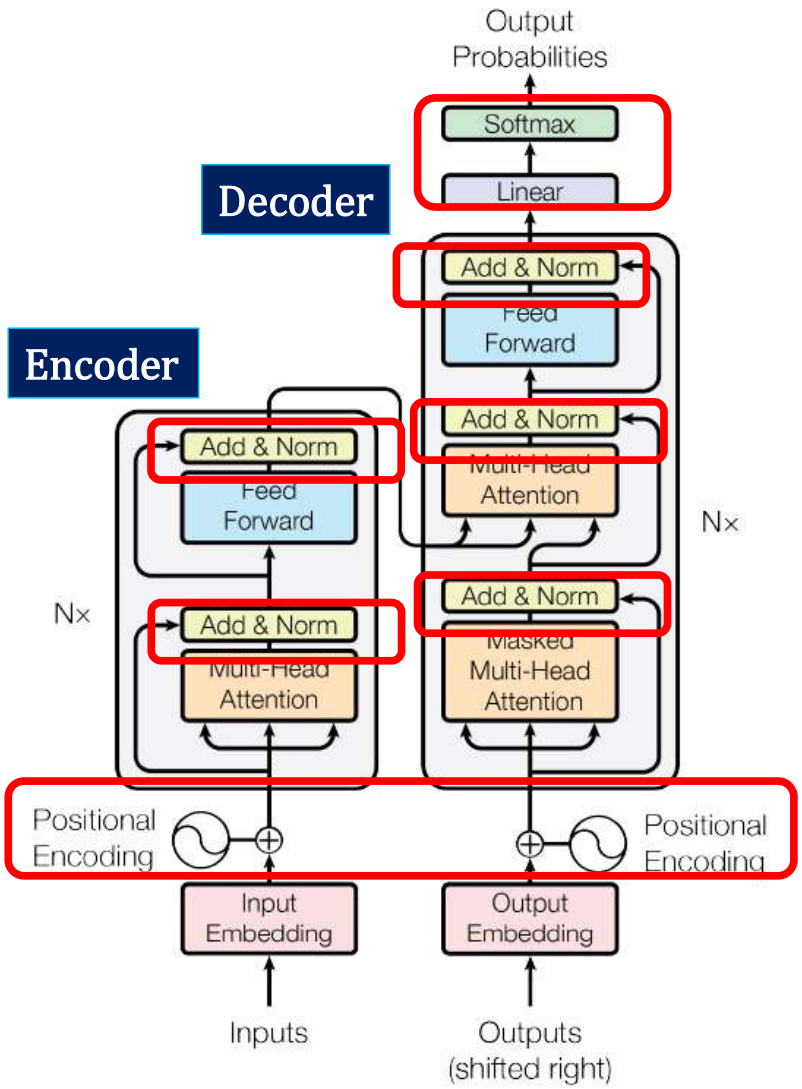
Para evitar el sobre-entrenamiento (overfitting) se aplican capas Dropout antes de cada capa Add & Norm y también después de cada uno de los procesos “Positional Encoding”. Recuerda que para evitar el filtrado de información (data leakage) el dropout solamente se aplica durante el entrenamiento y no en las etapas de validación o prueba.

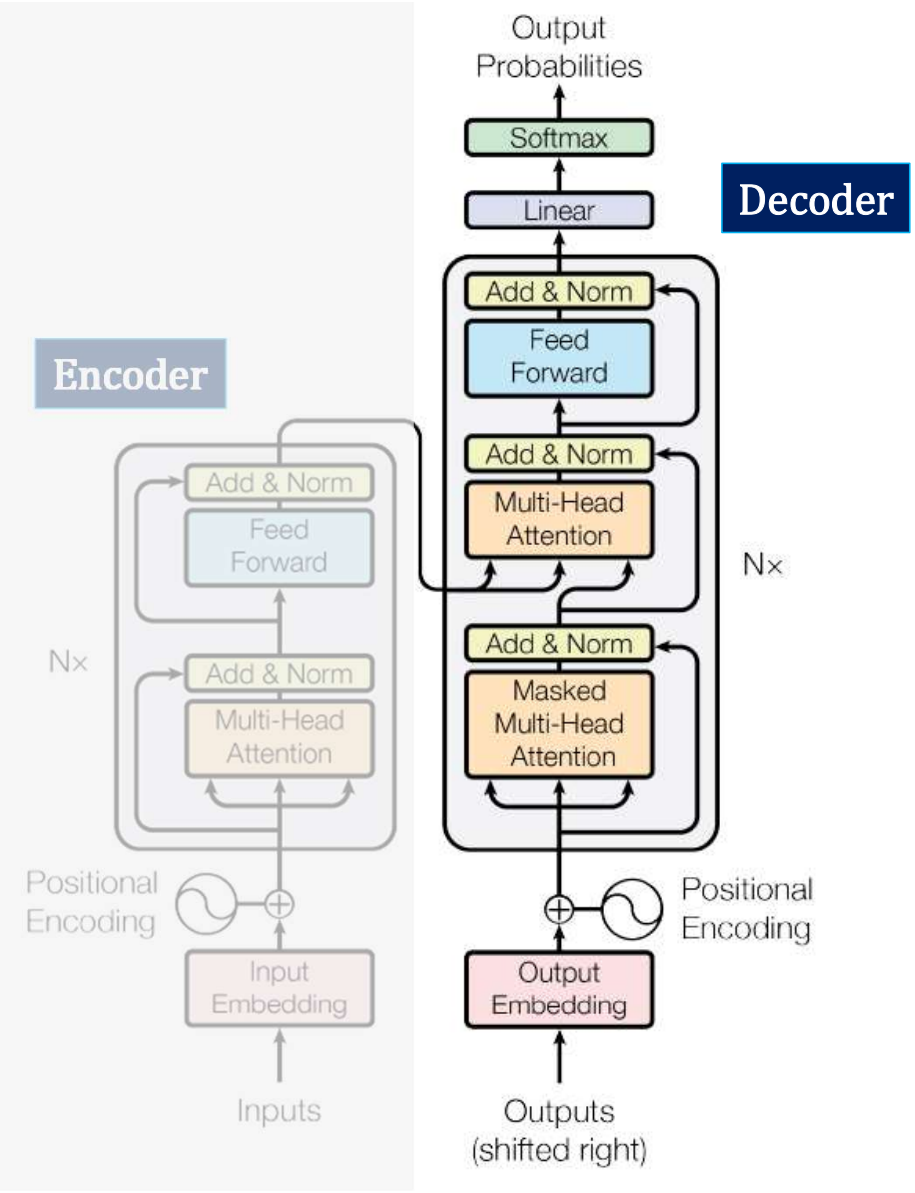
En particular se usó un valor de probabilidad de 0.1 en cada capa Dropout.

Capas de Salida del Decoder: Linear & Softmax

La capa “Linear” es una capa densa “Fully Connected” de dimensión el tamaño del vocabulario.

La capa Softmax igualmente es de dimensión el tamaño del vocabulario y nos dará la probabilidad de la palabra de salida del Transformer de mayor predicción.





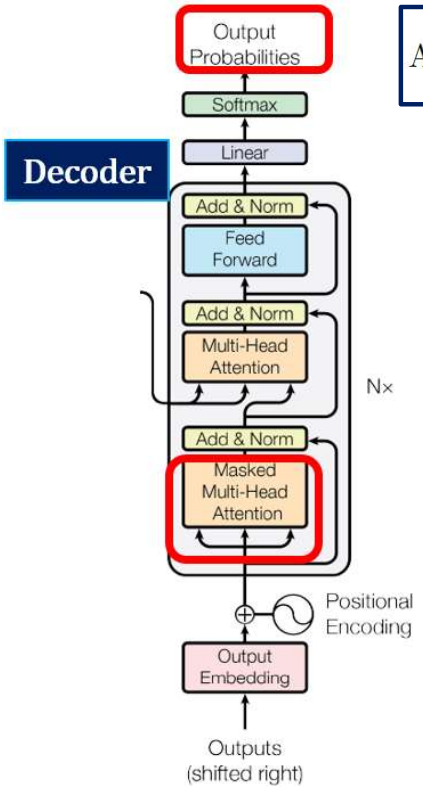
- De manera análoga al módulo Encoder, el módulo Decoder consiste de un bloque de tres sub-capas, dos de Attention y una de FeedForward que se repiten Nx=6 veces. En el artículo original se usa Nx=6, pero de hecho es un hiperparámetro que podría modificarse.

- A diferencia del Encoder, el Decoder tiene una sub-capa "enmascarada" de Attention (Masked Multi-Head Attention) adicional, que procesa la salida de probabilidades (Output Probabilities) del propio módulo, para después combinarla con la salida del Encoder y su segunda sub-capa Attention (Multi-Head Attention). Esta segunda sub-capa trata de compaginar la salida del Encoder con el texto que va generándose en el Decoder.

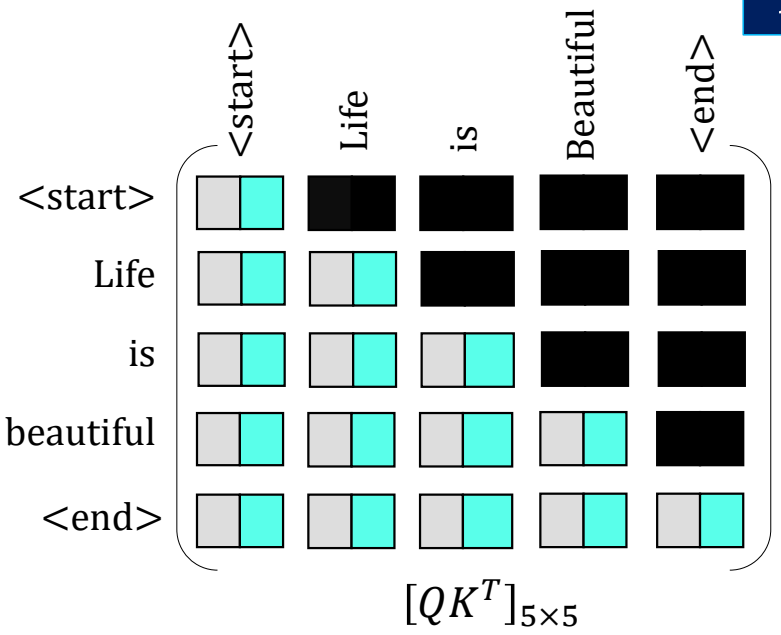
- En cada sub-capa se aplican nuevamente conexiones residuales (→), con suma y normalización a su salida.

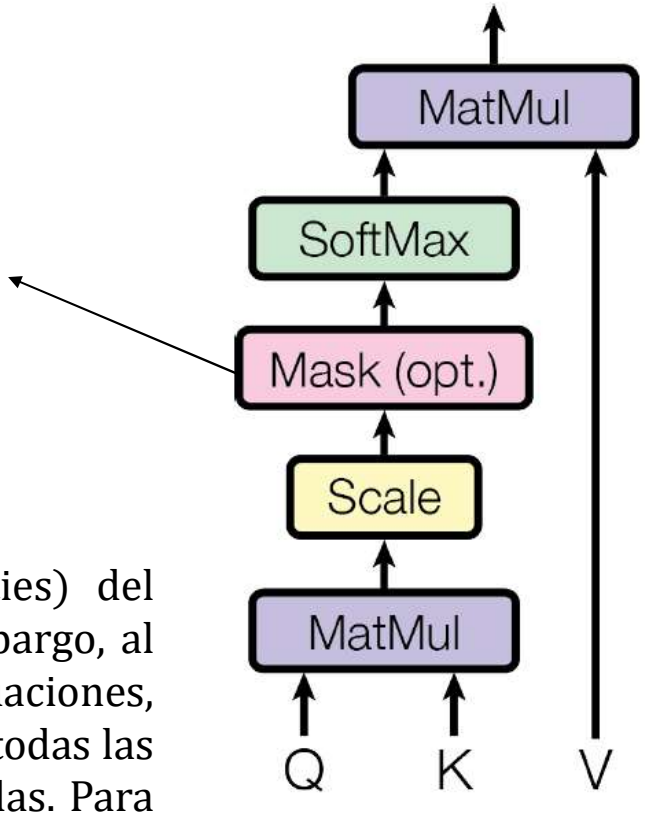
- La salida de cada sub-capa es de dimensión 512, incluyendo la capa embebida (Output Embedding).

Masked Multi-Head Attention : Enmascaramiento del Attention



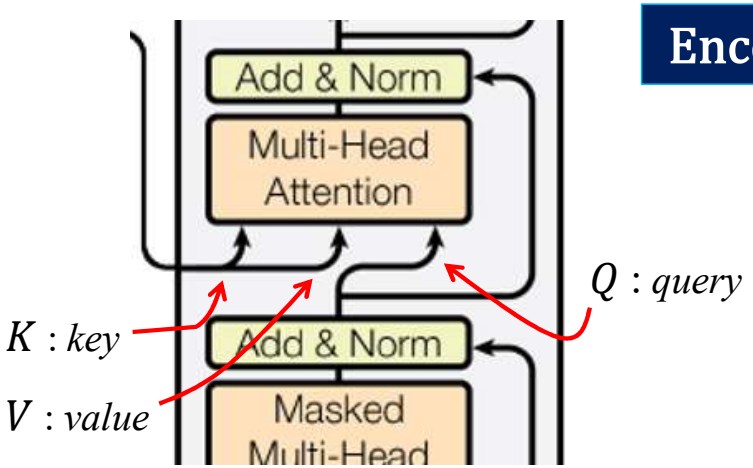
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



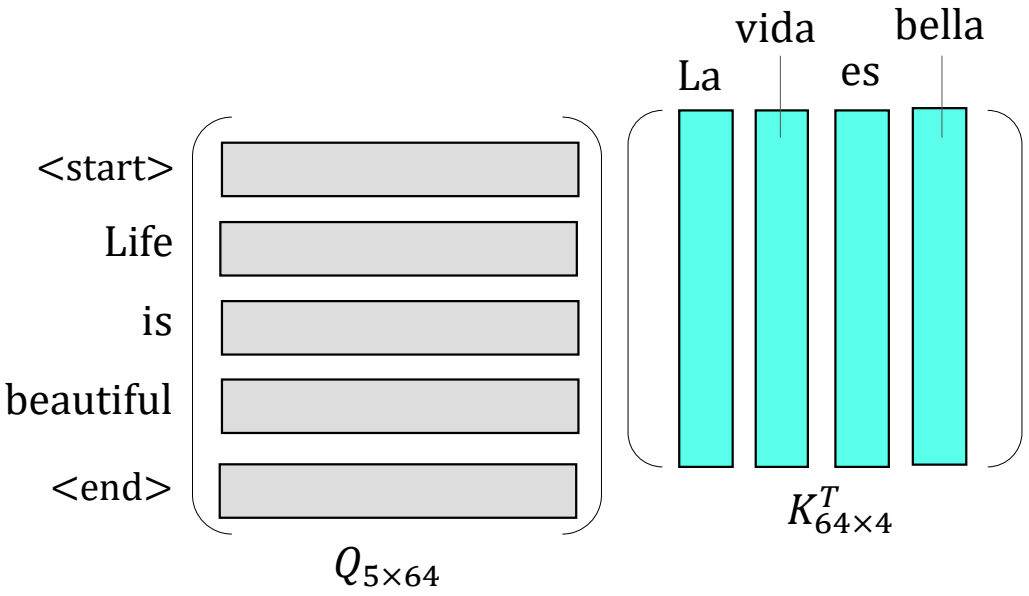


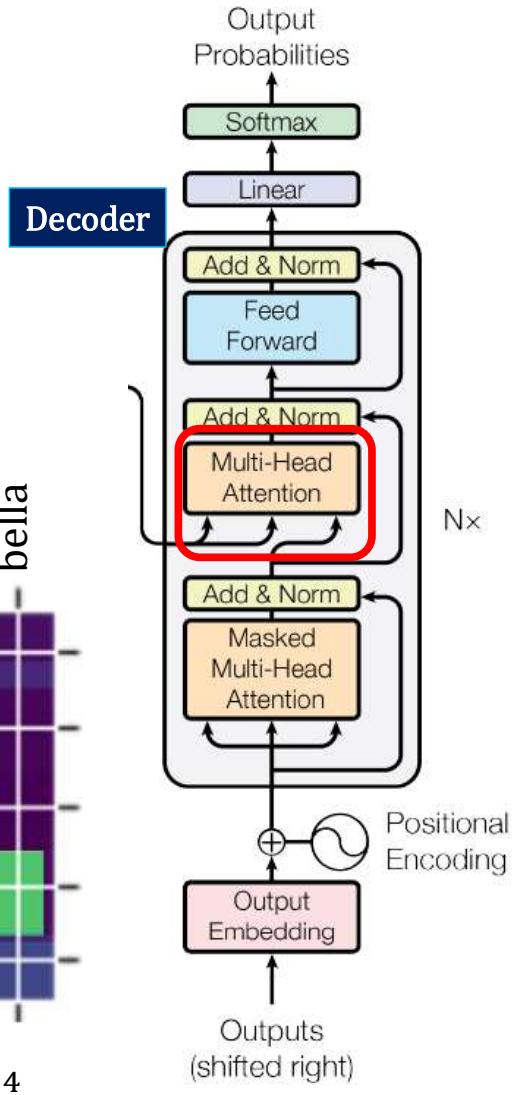
Sabemos que al generarse una predicción/salida (Output Probabilities) del Decoder, dicha salida será ahora la nueva entrada del Decoder. Sin embargo, al calcular la matriz QK^T y luego con la Softmax la matriz de puntuaciones, implícitamente estaríamos estableciendo relaciones entre una palabra y todas las demás, lo cual implica hacer uso de palabras antes de que sean generadas. Para evitar esto, se enmascaran los valores arriba de la diagonal principal de QK^T.

Encoder-Decoder Attention Layer



Esta sub-capa de Attention funciona de manera análoga a la del Encoder, salvo que el *query* Q es la salida del Masked-Attention previo y la *key* K y el *value* V vienen de la salida del Encoder.





Positional Encoding (PE)

pos : posición del token en el enunciado

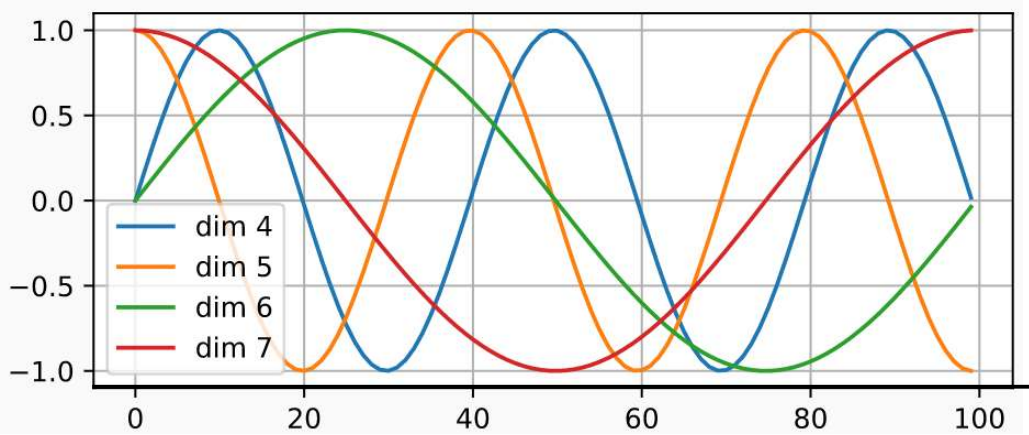
d_{emb} : dimensión del embedding

$i \in \{0,1, \ldots, \lfloor (d_{model} - 1)/2 \rfloor, \}$

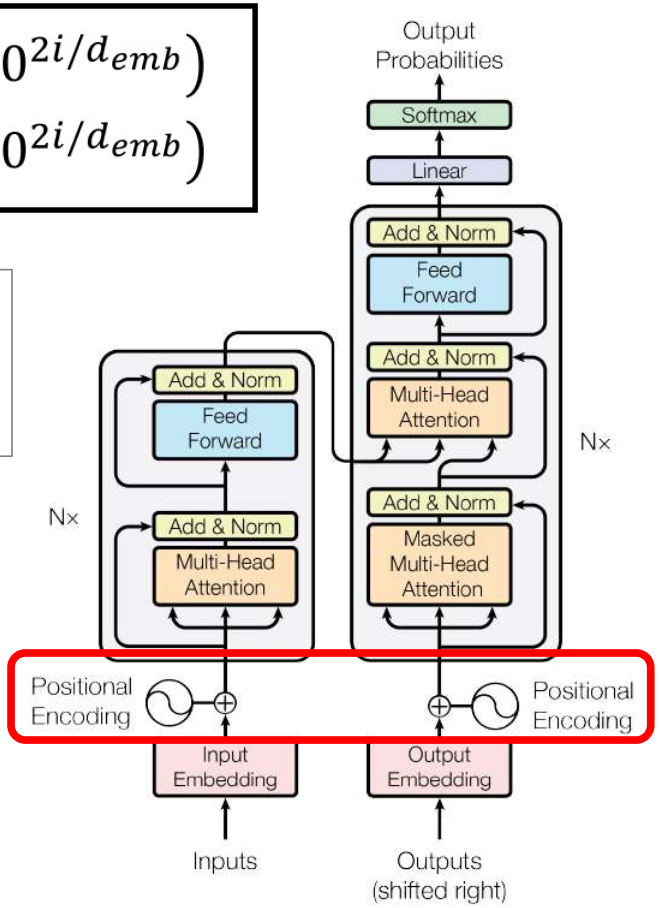
$dim(PE) = d_{emb}$

$$PE_{(pos=k,2i)} = sin\left(k/10000^{2i/d_{emb}}\right)$$

$$PE_{(pos=k,2i+1)} = cos\left(k/10000^{2i/d_{emb}}\right)$$



Se requerirán $d_{emb} = 512$ funciones sinusoidales.



Como ya no se tienen las capas convolucionales o recurrentes, se requiere introducir algún proceso que proporcione información sobre la posición relativa o absoluta de los tokens en un enunciado (secuencia).

Para ello se aplican estas funciones sinusoidales (en lugar de un modelo que aprenda dichas posiciones), ya que esta técnica nos permitirá extrapolar a enunciados de mayor longitud, que aquellos utilizados durante el proceso de entrenamiento.

En el artículo original se utiliza $d_{emb} = 512$

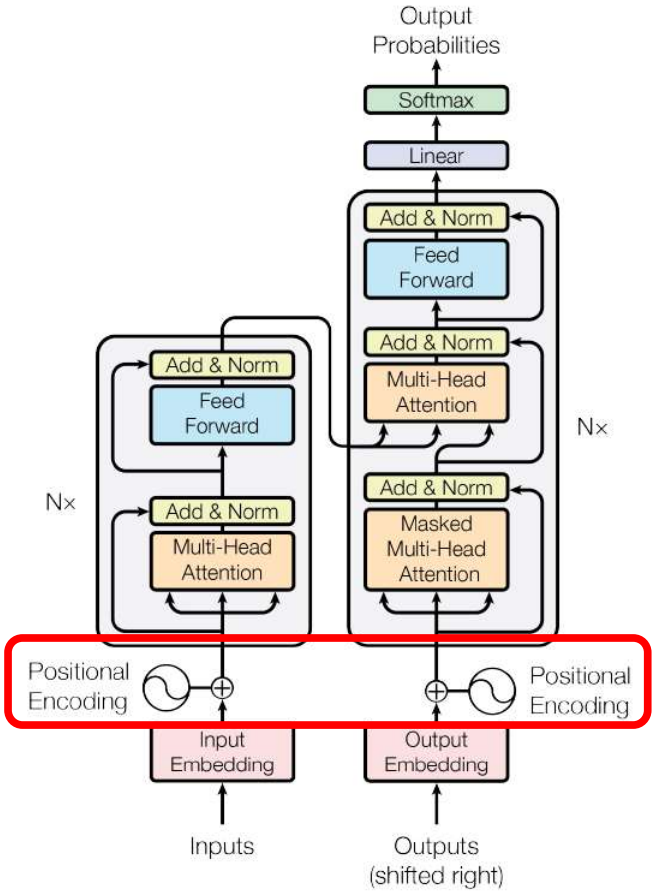
Positional Encoding (PE)

$$PE_{(pos=k,2i)} = sin\left(\frac{k}{10000^{2i/d_{emb}}}\right)$$

$$PE_{(pos=k,2i+1)} = cos\left(\frac{k}{10000^{2i/d_{emb}}}\right)$$

para cada k, se tiene $i = 0,1,2,\ldots,\lfloor(d_{emb} - 1)/2\rfloor$

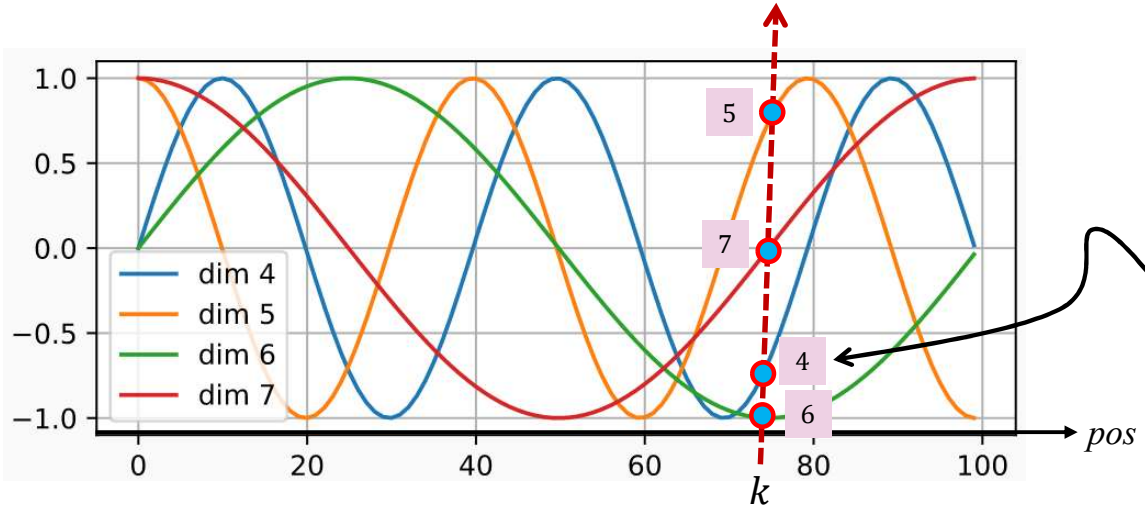
- La función seno genera $512/2 = 256$ coordenadas y la función coseno la otra mitad, para formar los vectores de codificación posicional PE de dimensión 512.
- Los vectores **Positional Encoding** y los **Input/Output Embeddings** son de la misma dimensión.
- Las longitudes de onda forman una progresión geométrica de 2π a $10000 \cdot 2\pi$
- Los autores plantearon la hipótesis de que con esta propuesta se puede conservar la información de la posición de los tokens.



En el artículo original se utiliza $d_{emb} = 512$

Positional Encoding (PE)

Vector de $d_{emb} = 512$ coordenadas asociado al token ("palabra") en la posición $pos{=}k$ del enunciado de entrada.



$$PE_{(pos=k,2i)} = sin\left(\frac{k}{10000^{2i/d_{emb}}}\right)$$

$$PE_{(pos=k,2i+1)} = cos\left(\frac{k}{10000^{2i/d_{emb}}}\right)$$

para cada k, se tiene $i = 0,1,2,\ldots,\lfloor(d_{emb} - 1)/2\rfloor$

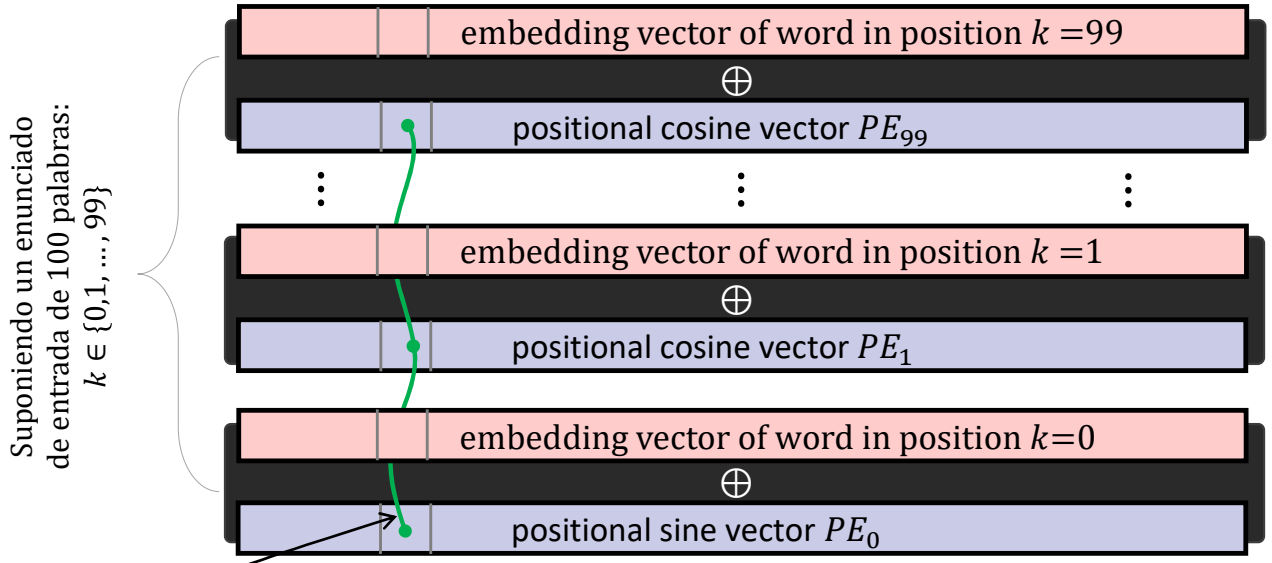
j-ésima coordenada del vector de $d_{emb} = 512$ coordenadas, asociada a la palabra en la $pos{=}k$.

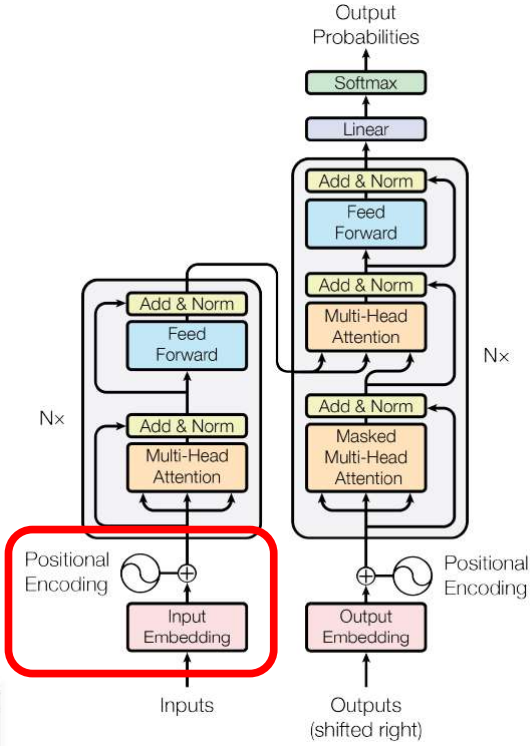
Supongamos que tenemos una frase de entrada con una longitud de 100 palabras (pero obviamente este valor es variable de acuerdo al tamaño de la frase de entrada).
A la primer palabra en la posición pos =0 le asociamos el vector PE(0, i) de dimensión $d_{emb} = 512$.
A la segunda palabra en la posición pos =1 le asociamos el vector PE(1, i) de dimensión $d_{emb} = 512$.
Y así sucesivamente,
A la palabra en la posición $pos{=}k$ le asociamos el vector PE (k, i) de dimensión $d_{emb} = 512$.
Dichos vectores se combinan con el vector de salida de la capa (Input/Output) Embedding.

Así, para una ventana de 100 palabras (tokens), estos 100 vectores PE de dimensión $d_{emb} = 512$ serán siempre los mismos y transmiten la información de la posición de cada palabra dentro de la frase.

En el bloque “Input Embedding” cada palabra (token) de un enunciado se transforma en un vector embebido y se asocia con un vector sinusoidal de codificación posicional:

Input Embedding

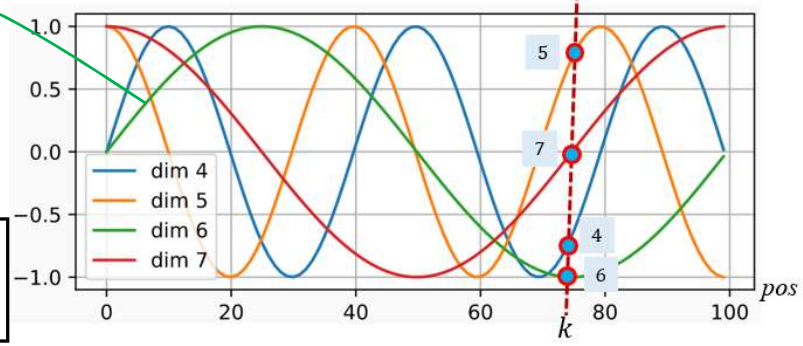




Dada una coordenada $i \in \{0,\dots,511\}$ fija, el valor posicional asociado a la coordenada correspondiente de la palabra en la posición k se obtienen con una de las funciones PE.

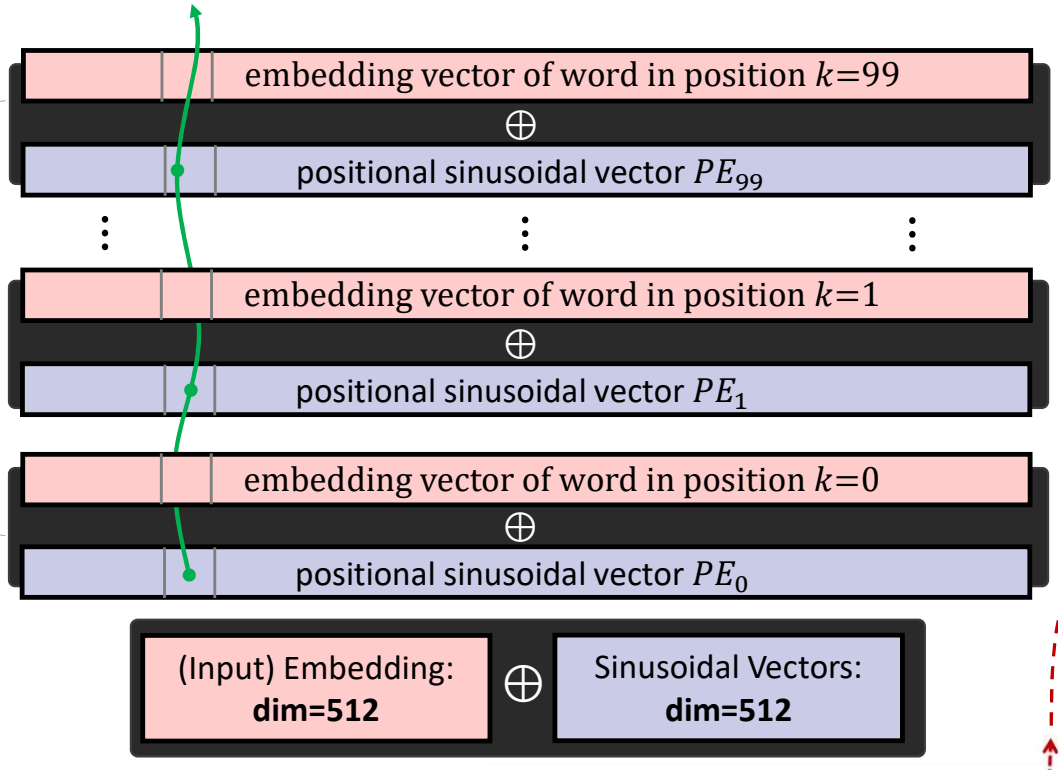
coordenadas pares: $PE_{(pos=k,2i)} = sin\left(k/10000^{2i/d_{emb}}\right)$

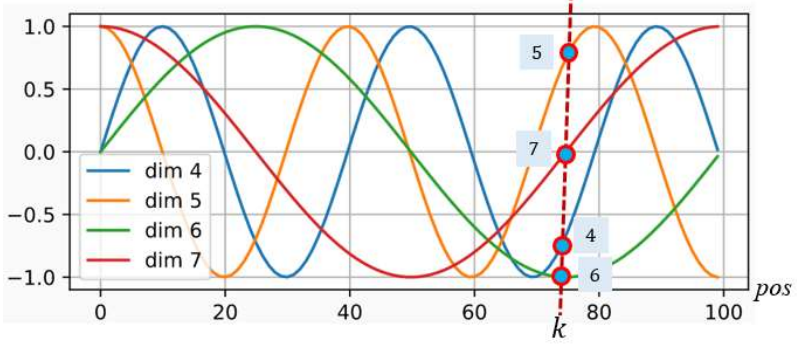
coordenadas impares: $PE_{(pos=k,2i+1)} = cos\left(k/10000^{2i/d_{emb}}\right)$



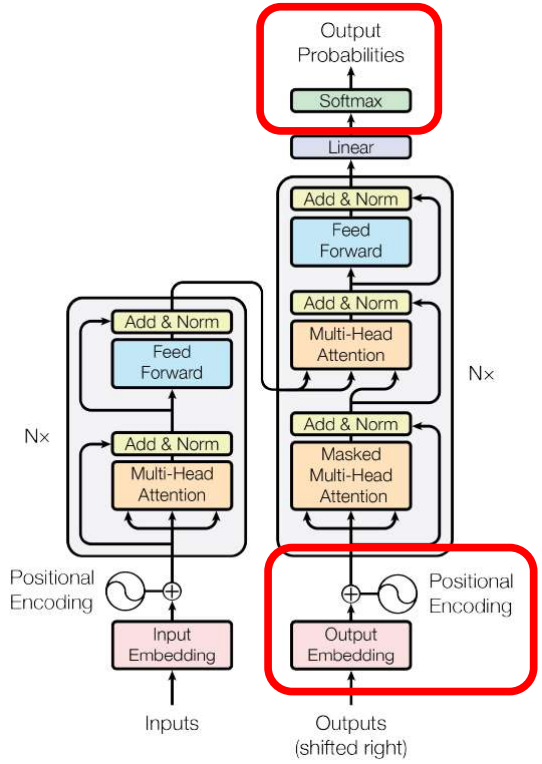
Predicción mediante Softmax de la posible palabra 101 siguiente.

Suponiendo que entró por el **Input Embedding** un enunciado de 100 palabras, entonces por la salida **Output Probabilities** se hace la predicción de la 101, y que se agregará a la entrada del **Output Embedding**.





Output Embedding



Comparación de la complejidad del modelo Attention
en relación a una capa convolucional o recurrente:

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

https://arxiv.org/abs/1706.03762