

Transformers



Dr. Manuel Castillo-Cara

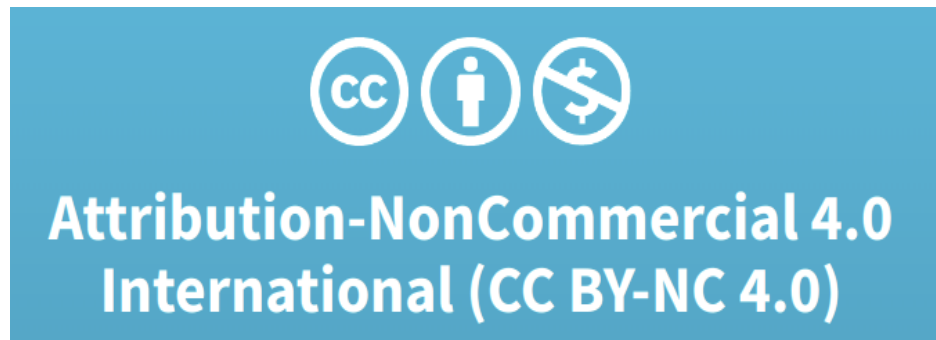
www.manuelcastillo.eu

**Departamento de Inteligencia Artificial
Escuela Técnica Superior de Ingeniería Informática
Universidad Nacional de Educación a Distancia (UNED)**

Preliminar



- Improving Deep Learning by Exploiting Synthetic Images © 2024 by Manuel Castillo-Cara is licensed under Attribution-NonCommercial 4.0 International



Índice



Background

Attention

Transformers

Vision Transformers

Inferencia en ViT

Disposiciones finales



Background

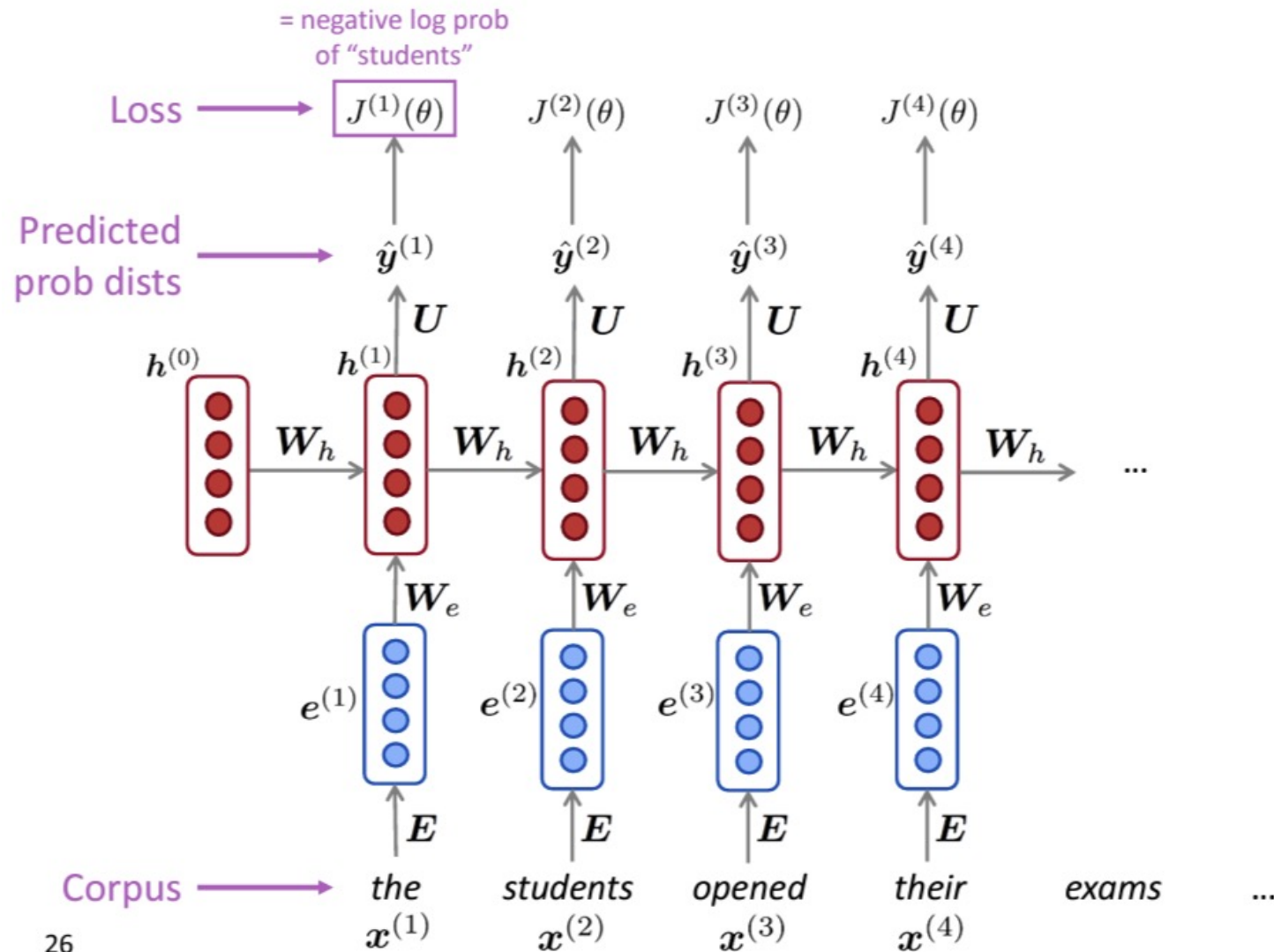


ETS de
Ingeniería
Informática

UNED

Redes Neuronales Recurrentes

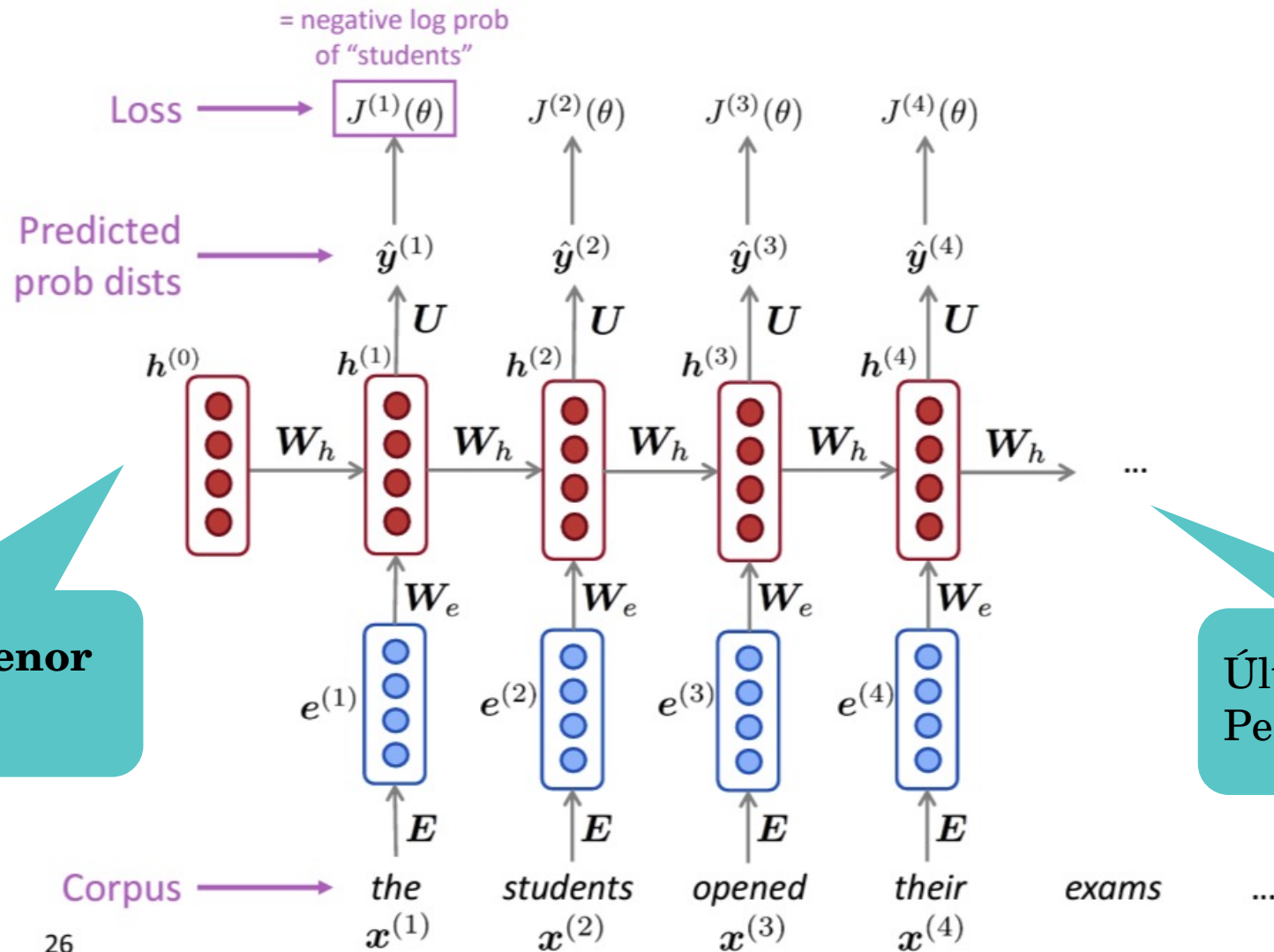
Problema





Redes Neuronales Recurrentes

Problema

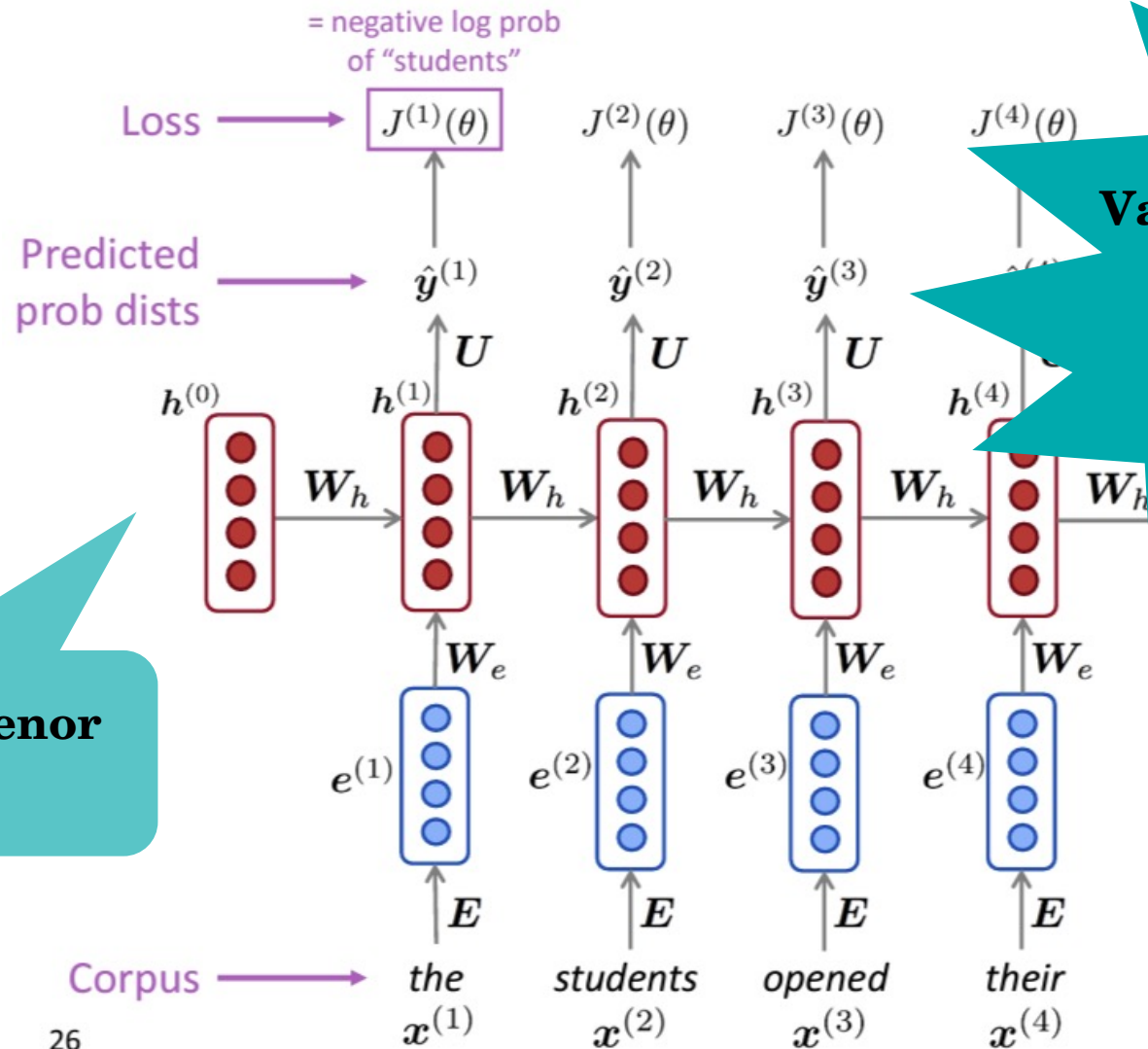


Primeras capas **menor**
Peso \rightarrow Olvida

Últimas capas **mayor**
Peso \rightarrow Recuerdo

Redes Neuronales Recurrentes

Problema



Va olvidando el contexto pasado

Primeras capas **menor**
Peso → Olvida

Últimas capas **mayor**
Peso → Recuerdo

Redes Neuronales Recurrentes

Problema



El jaguar dormía en la rama del árbol con su cola de soporte.

Redes Neuronales Recurrentes

Problema



El jaguar dormía en la rama del árbol con **su** cola de soporte.



Con las RRN, se olvida que
“su” hace referencia al jaguar

Redes Neuronales Recurrentes

Problema



El jaguar dormía en la rama del árbol con **su** cola de soporte.



**Attention resuelve
este problema**

Con las RRN, se olvida que
“su” hace referencia al jaguar



Desde las RNN a Word Embedding

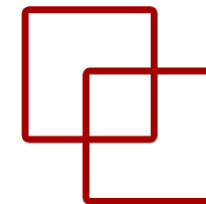


ETS de
Ingeniería
Informática

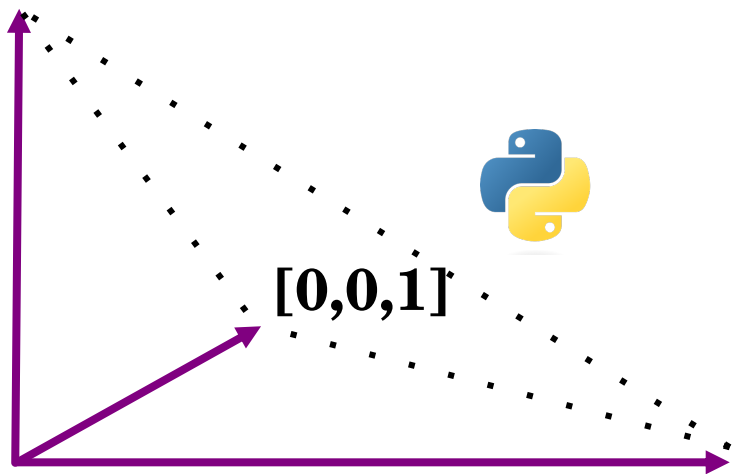
UNED

Encoding

One-Hot Encoding – Problema

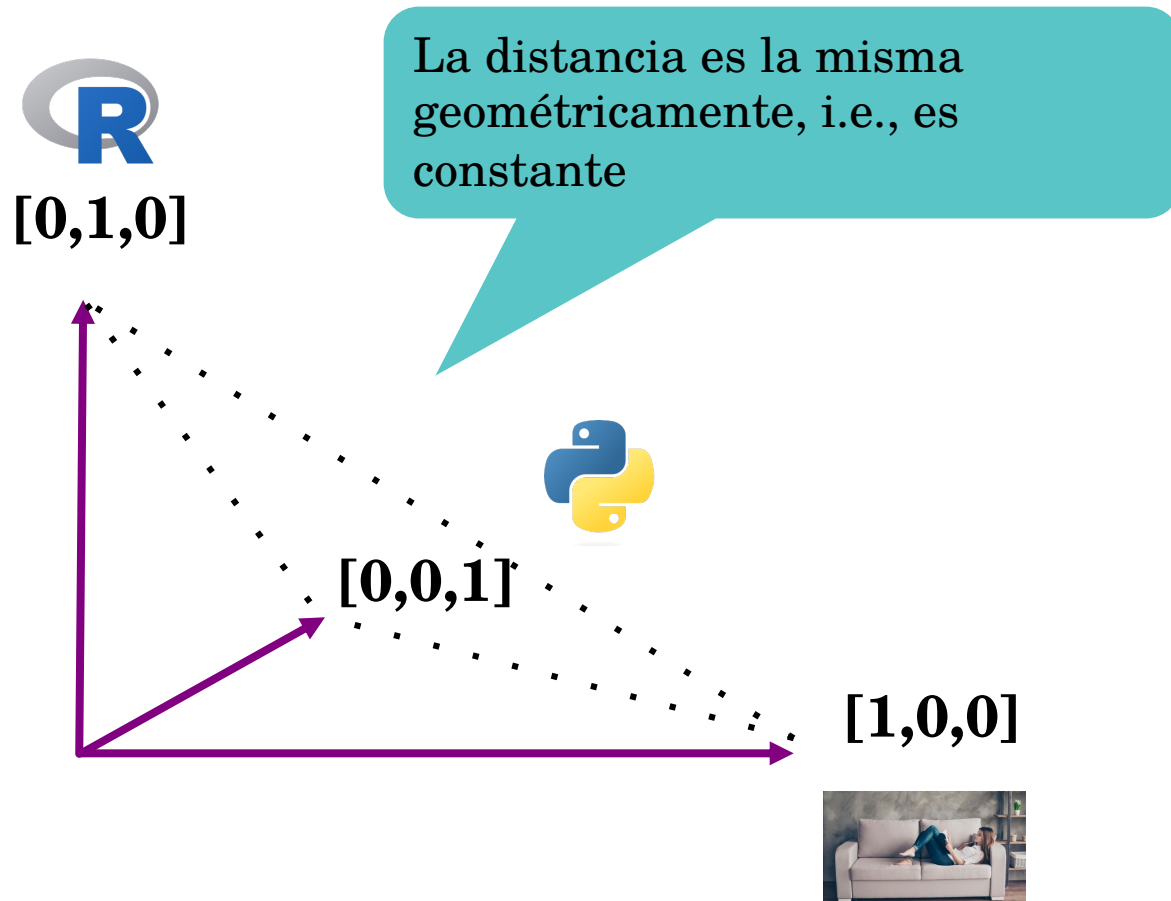
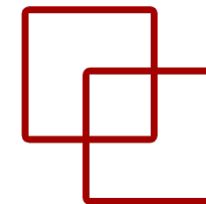


$[0,1,0]$



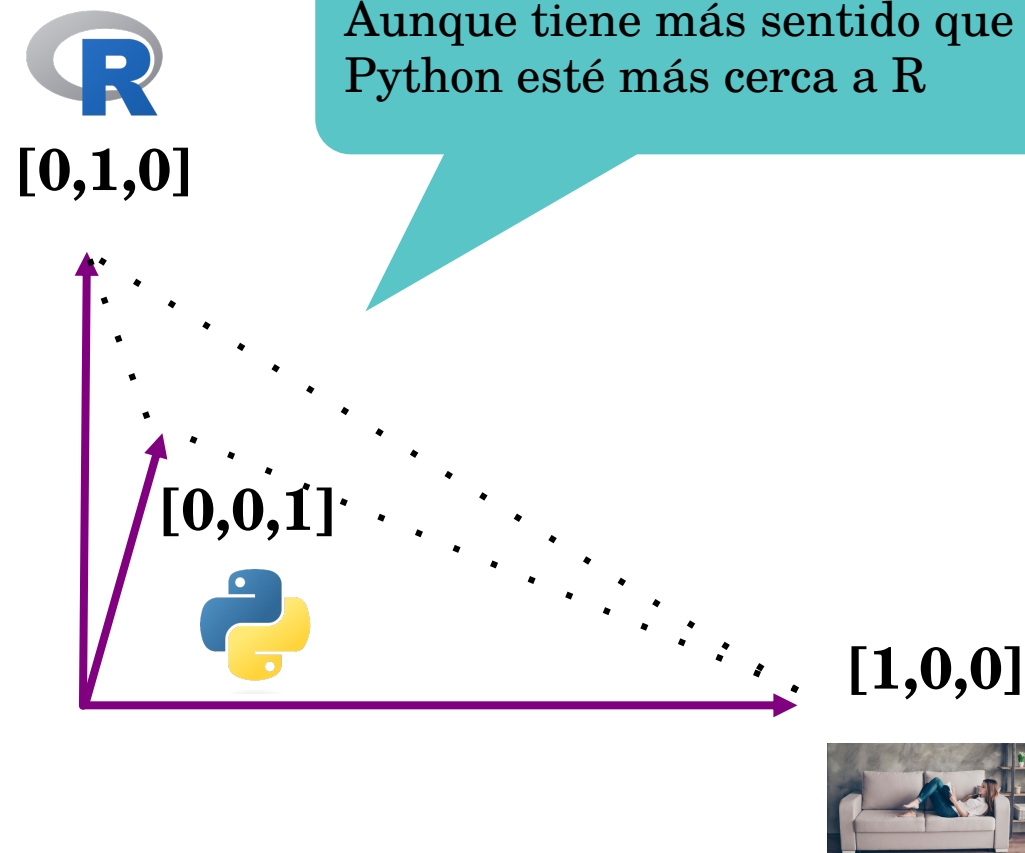
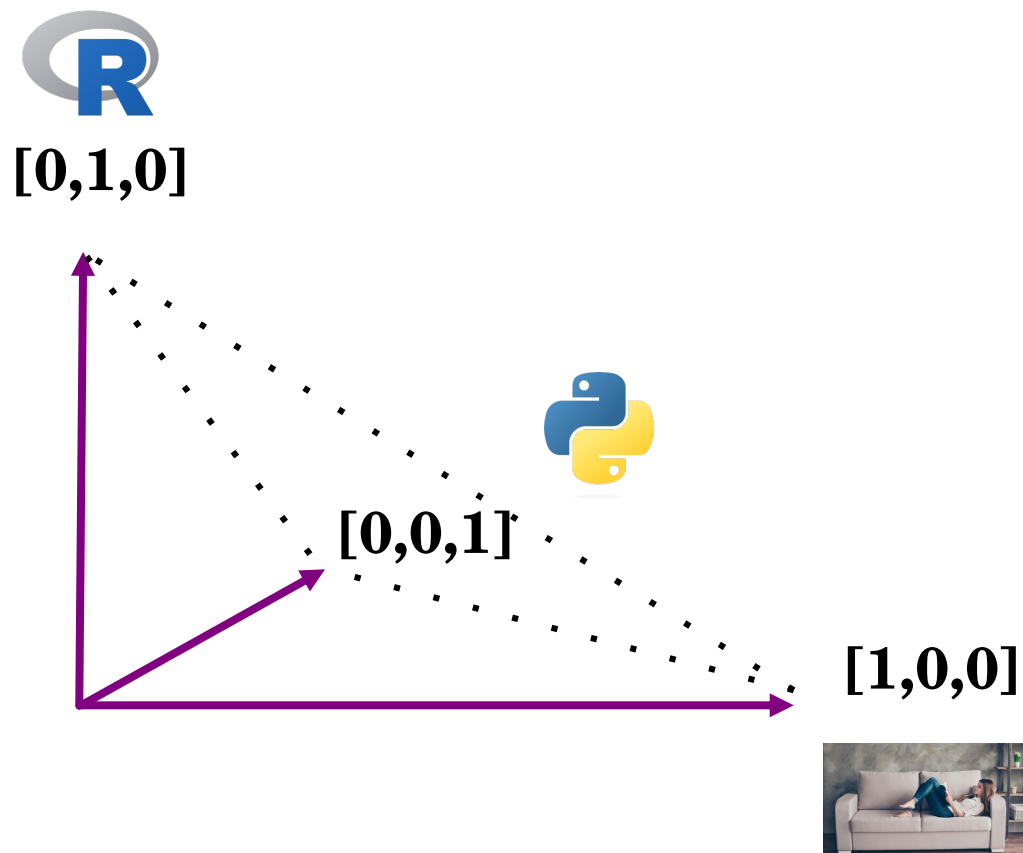
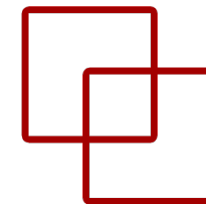
Encoding

One-Hot Encoding – Problema



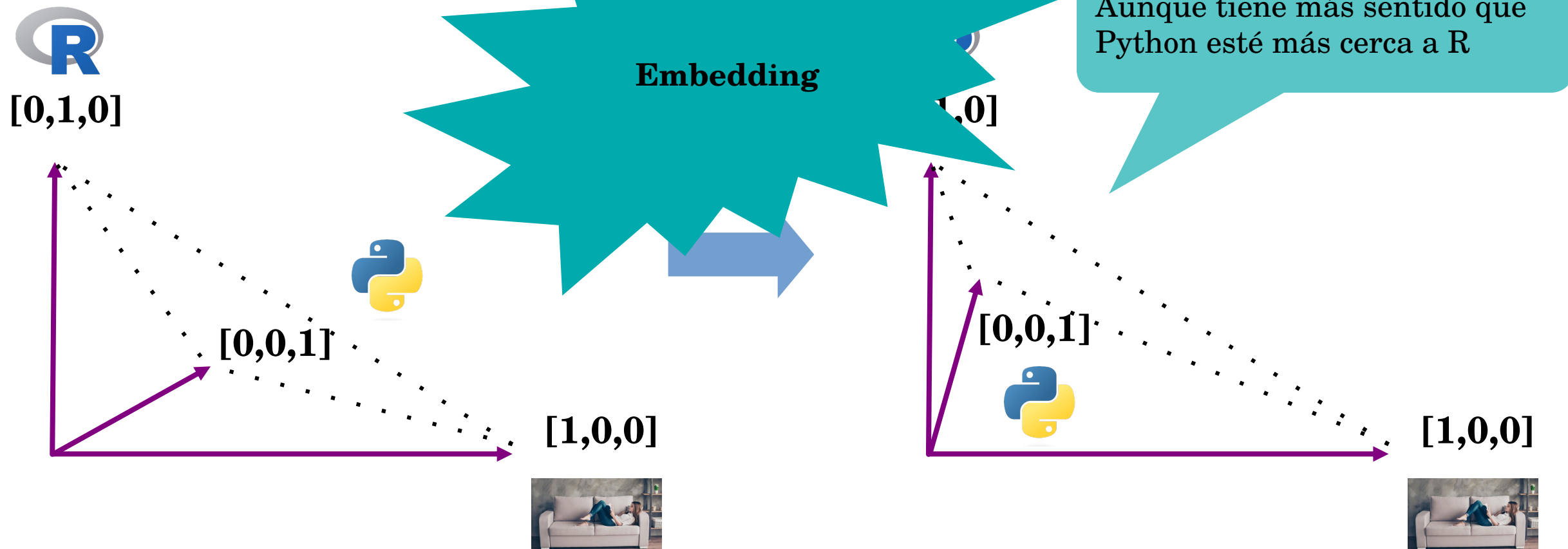
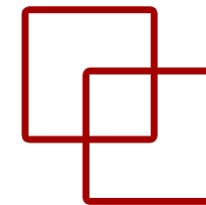
Encoding

One-Hot Encoding – Problema

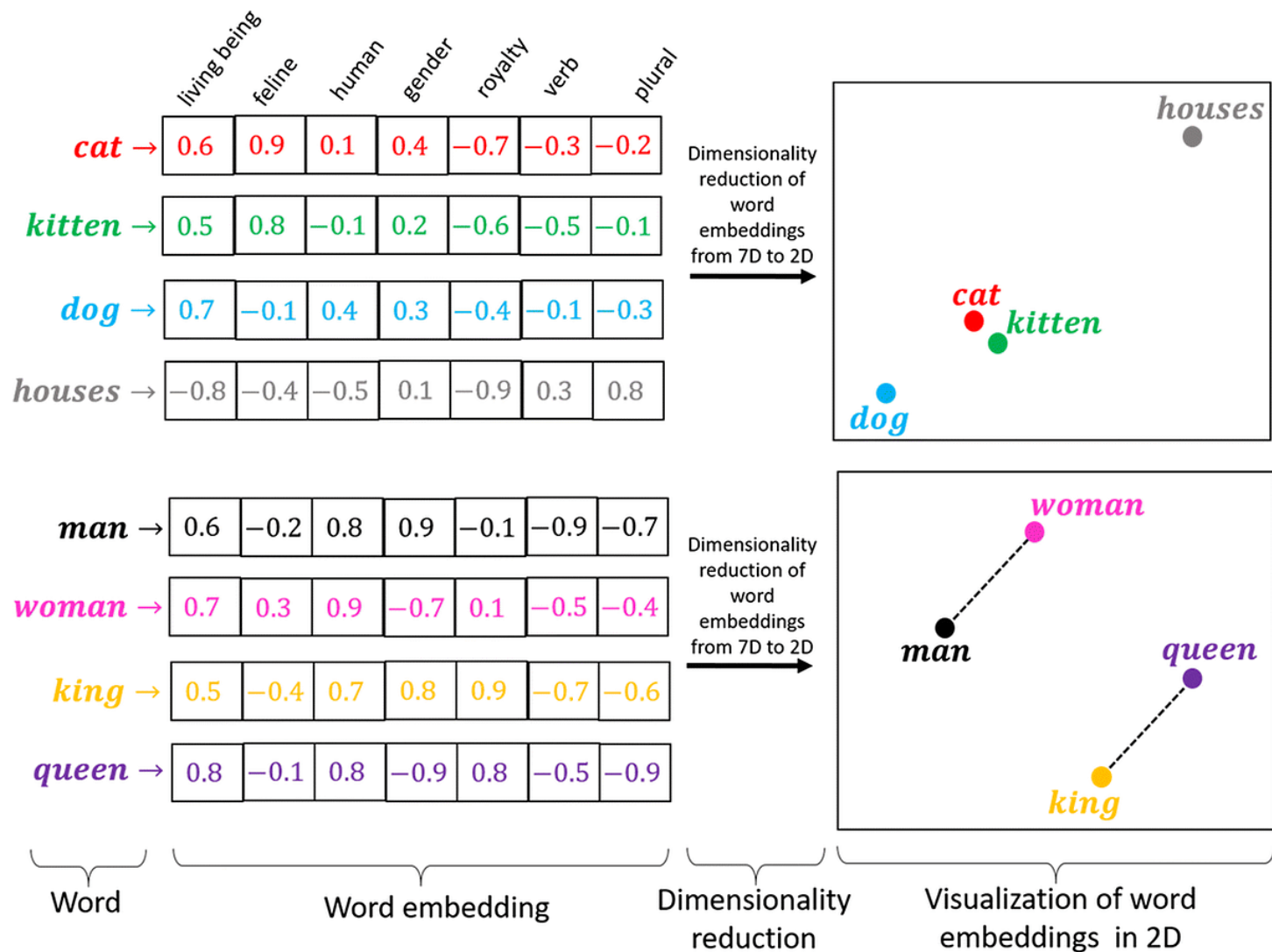
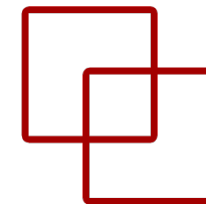


Encoding

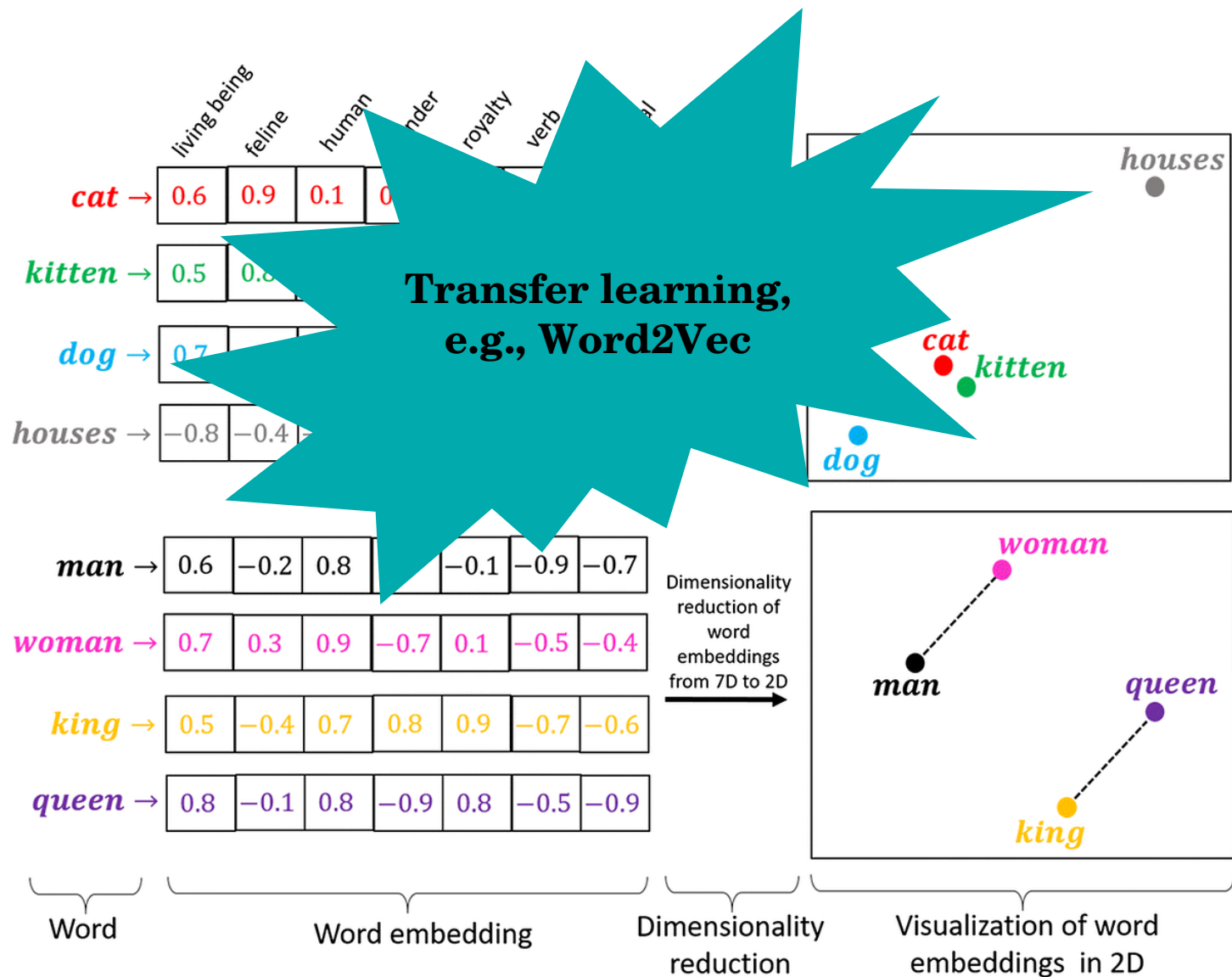
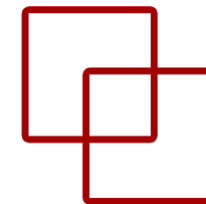
One-Hot Encoding – Problema



Word embedding



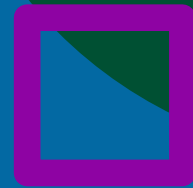
Word embedding





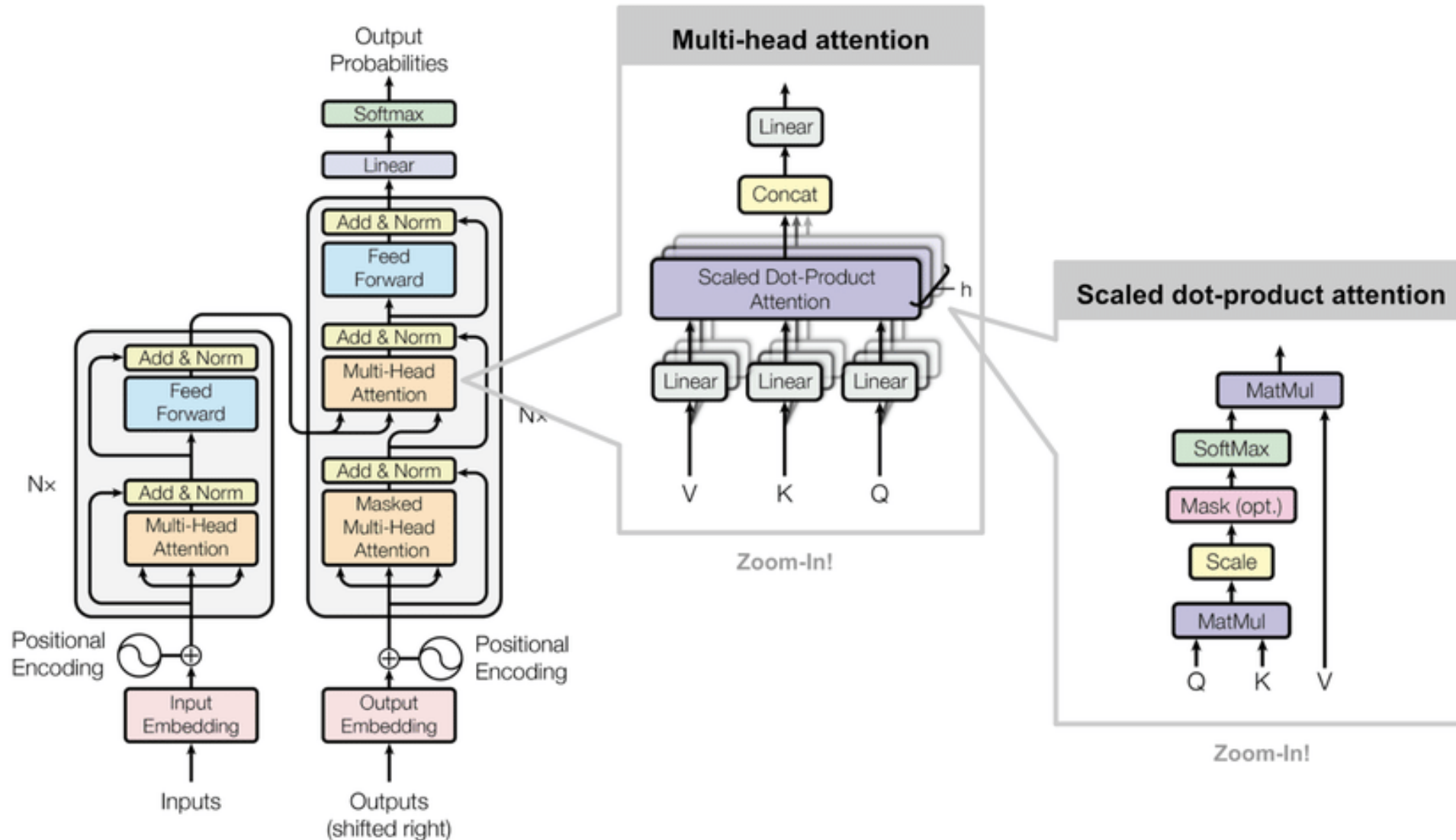
Attention

ETS de
Ingeniería
Informática



UNED

Attention Background



Attention

Background



El

jaguar

dormía

en

su

árbol

El

jaguar

dormía

en

su

árbol

Attention

Background



El

jaguar

dormía

en

su

árbol

...

El

jaguar

dormía

en

su

árbol



Attention Background

El

jaguar

dormía

en

su

árbol

...

El

jaguar

dormía

en

su

árbol

Busca la relación de todas
las palabras con todas
las otras palabras!



Vectores

Key - Query

El

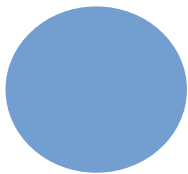
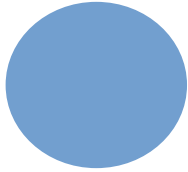
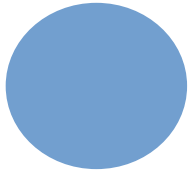
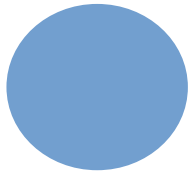
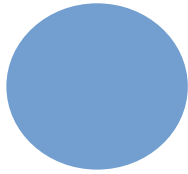
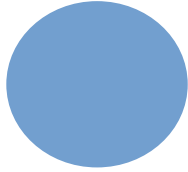
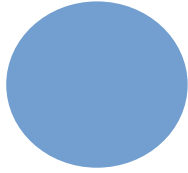
jaguar

dormía

en

su

árbol



El

jaguar

dormía

en

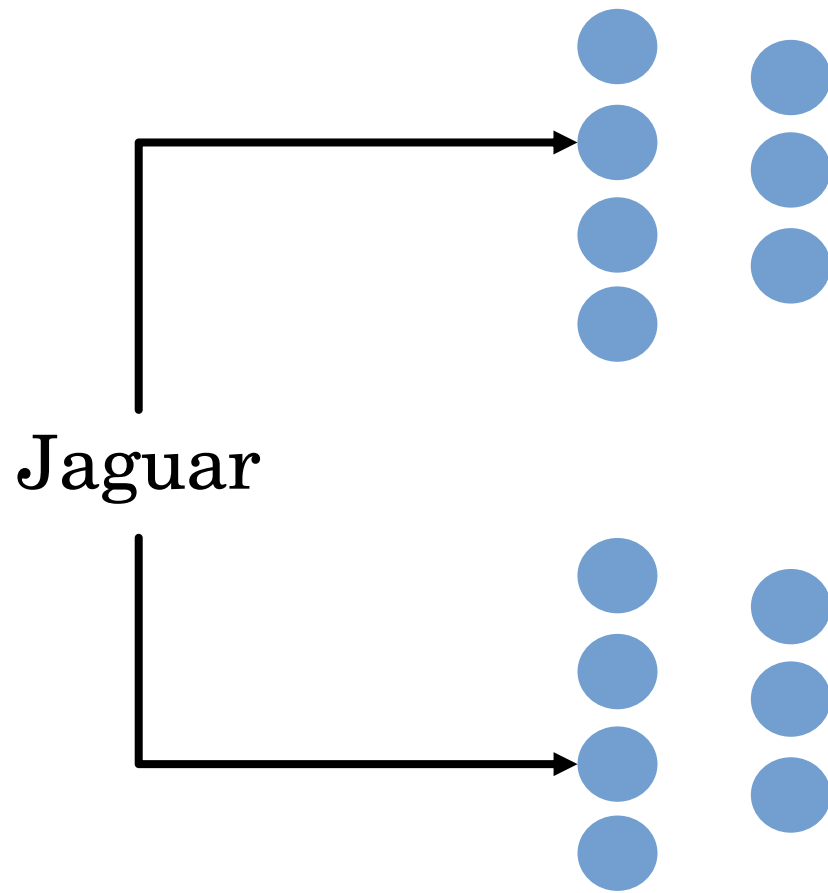
su

árbol

Y son redes neuronales
que aprenden a encontrar
estas relaciones

Vectores

Key - Query

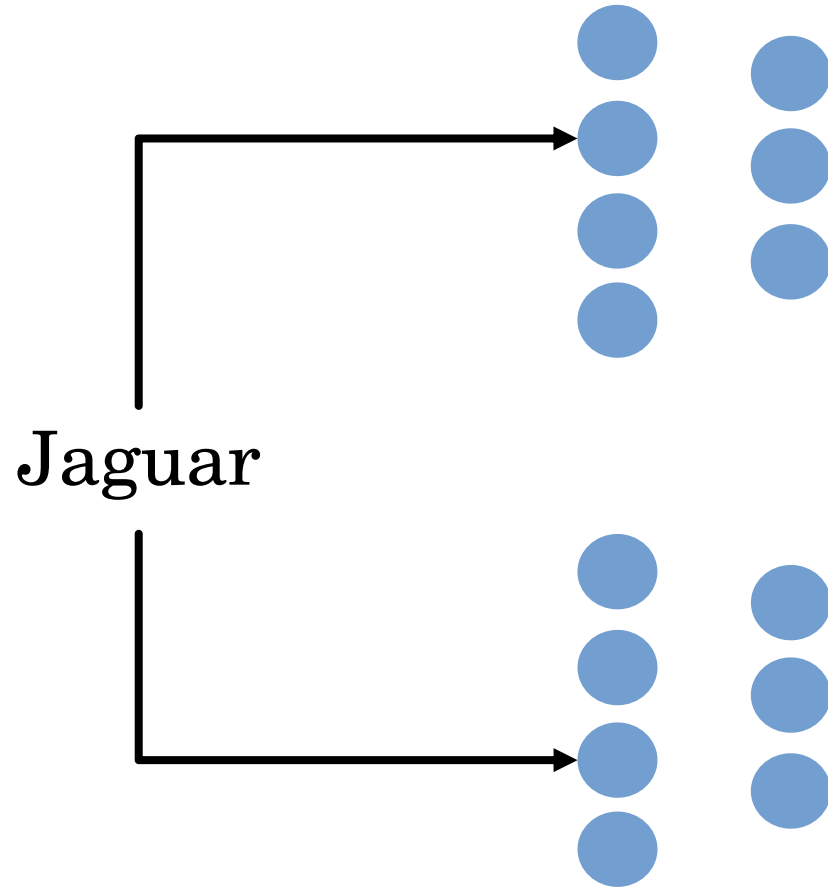


Dos NN diferentes para
generar dos vectores
distintos



Vectores

Key - Query



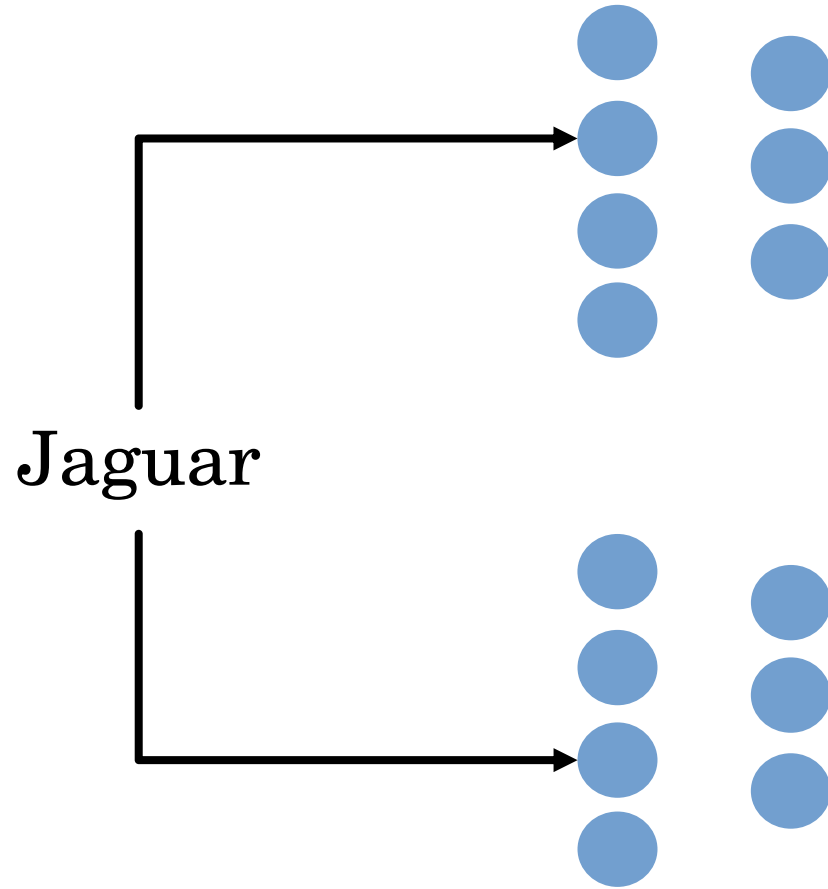
[0,74 -0,44 ...0,54]

Vector identificador (key):
Obtiene las propiedades
que describen dicha palabra
→ Significado



Vectores

Key - Query



[0,74 -0,44 ...0,54]

Vector identificador (key):
Obtiene las propiedades
que describen dicha palabra
→ Significado

[0,34 -0,14 ...0,34]

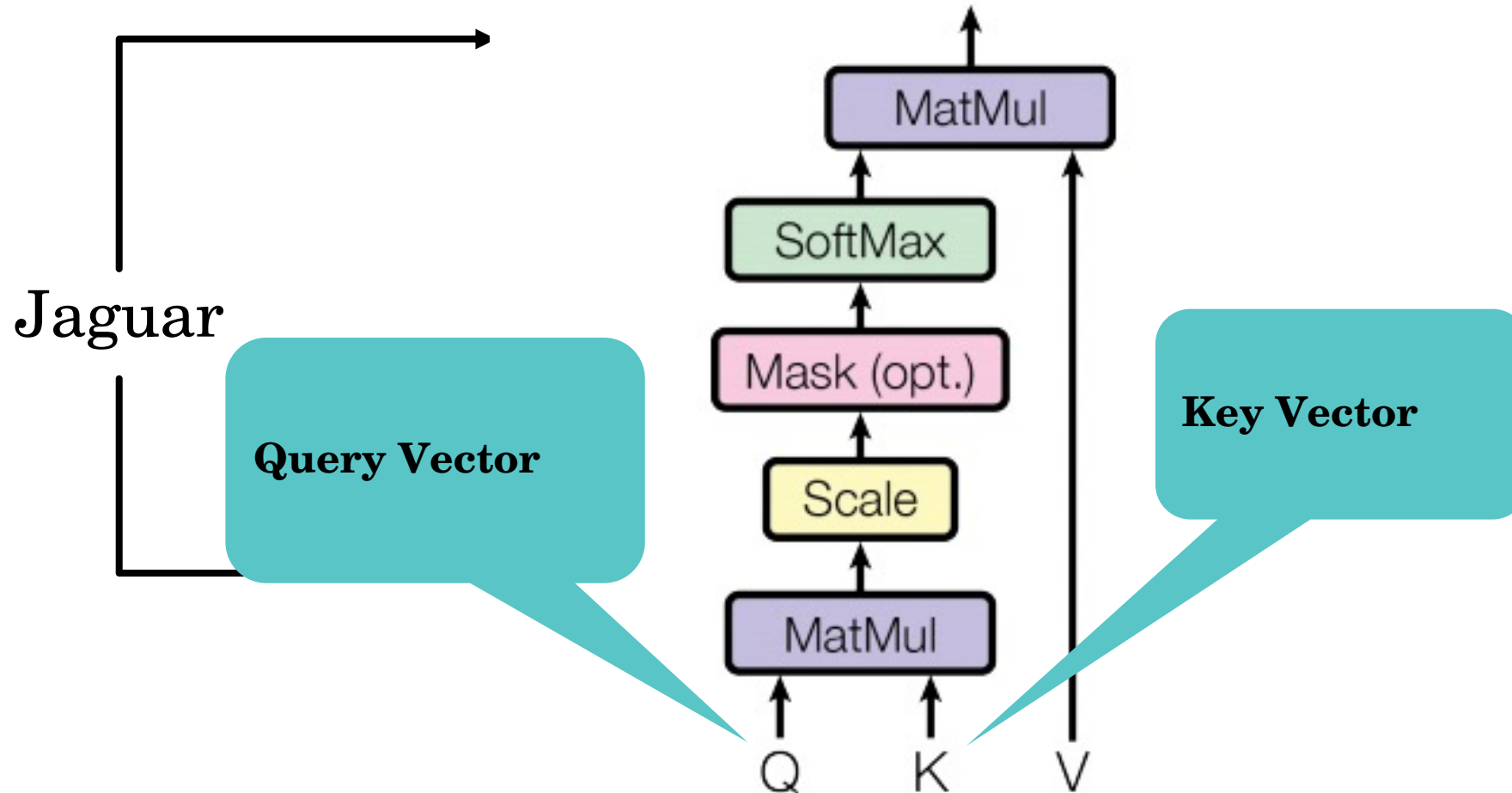
Vector búsqueda (query):
Obtiene las propiedades
de lo que busca la palabra
→ Contexto



Vectores

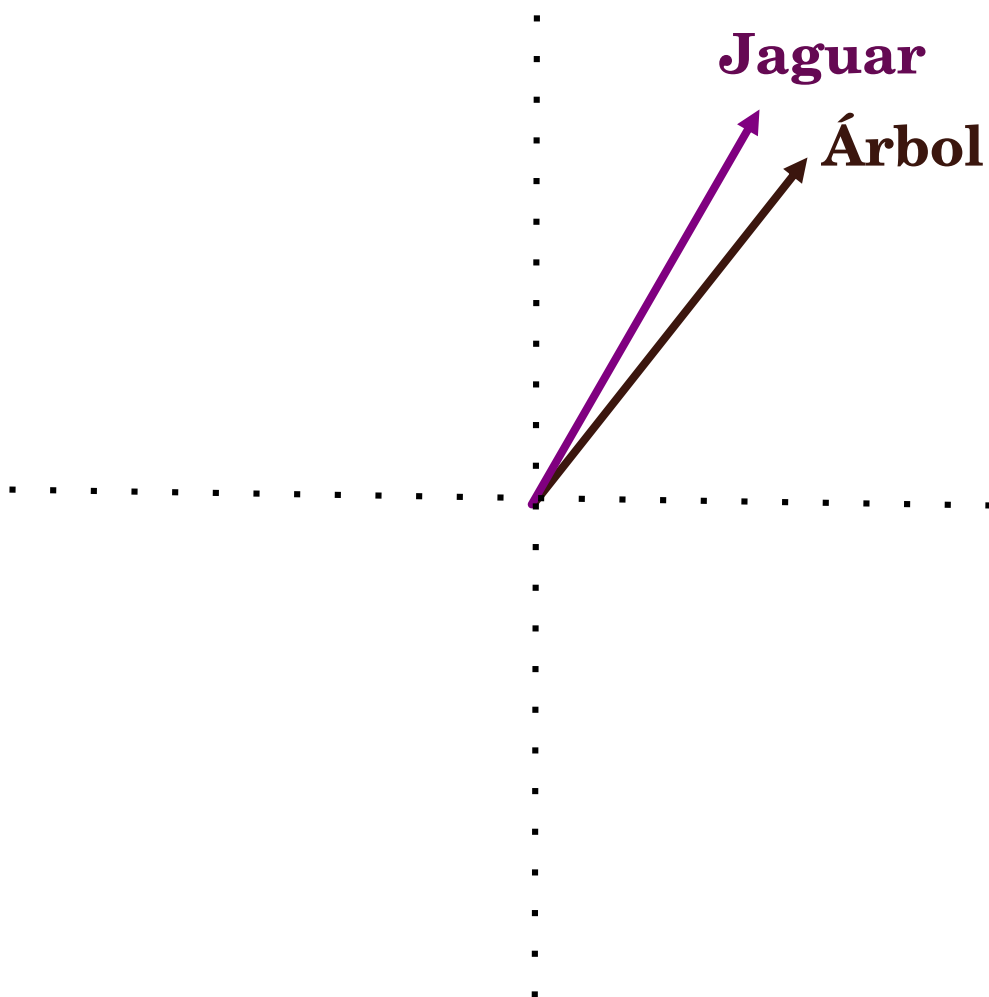
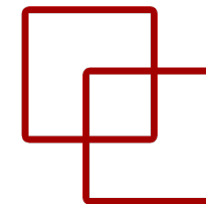
Key - Query

Scaled Dot-Product Attention



Attention

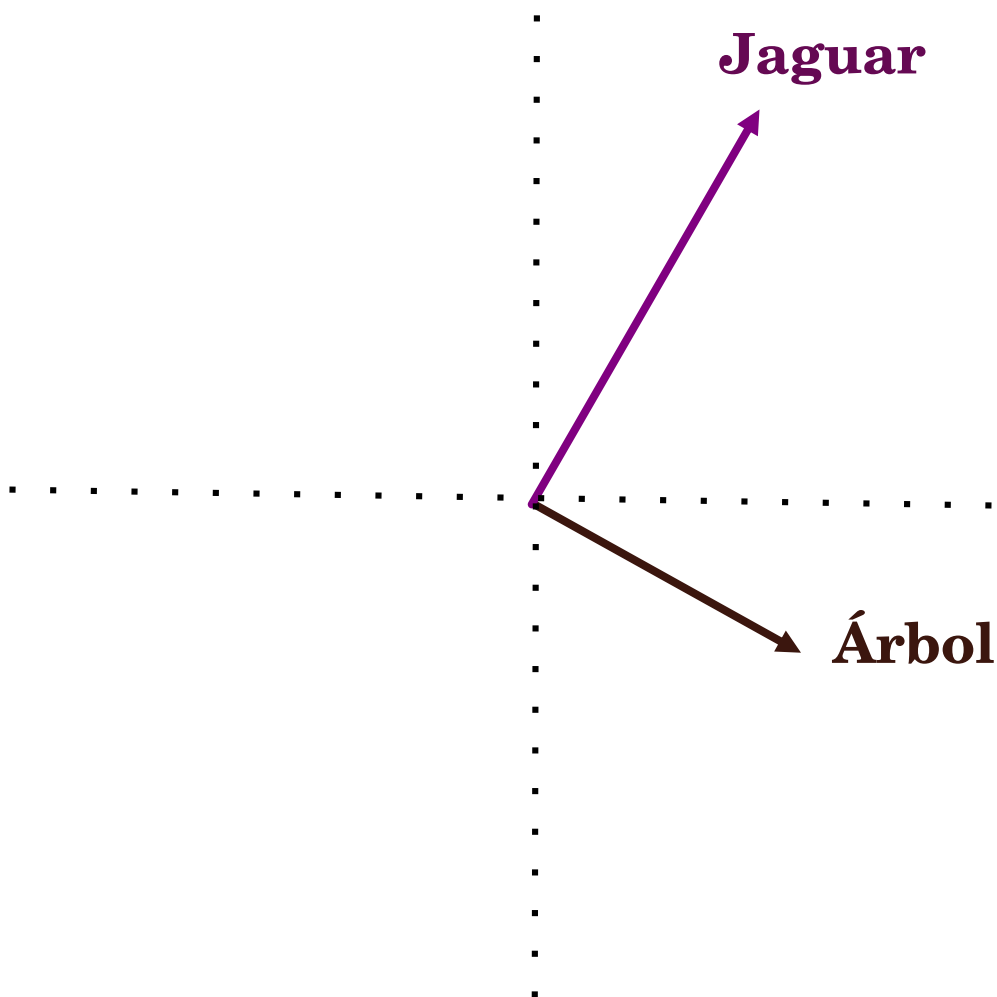
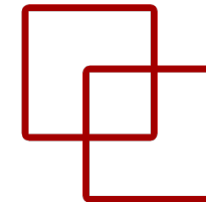
Producto escalar



Si los vectores están **próximos** entonces son palabras coincidentes → Están en su contexto

Attention

Producto escalar



Si los vectores están **alejados** entonces no son palabras coincidentes → No están en su contexto



Attention Background

El

jaguar

dormía

en

su

árbol

...

Vectores Key

El

jaguar

dormía

en

su

árbol

Vectores Query

Attention Background

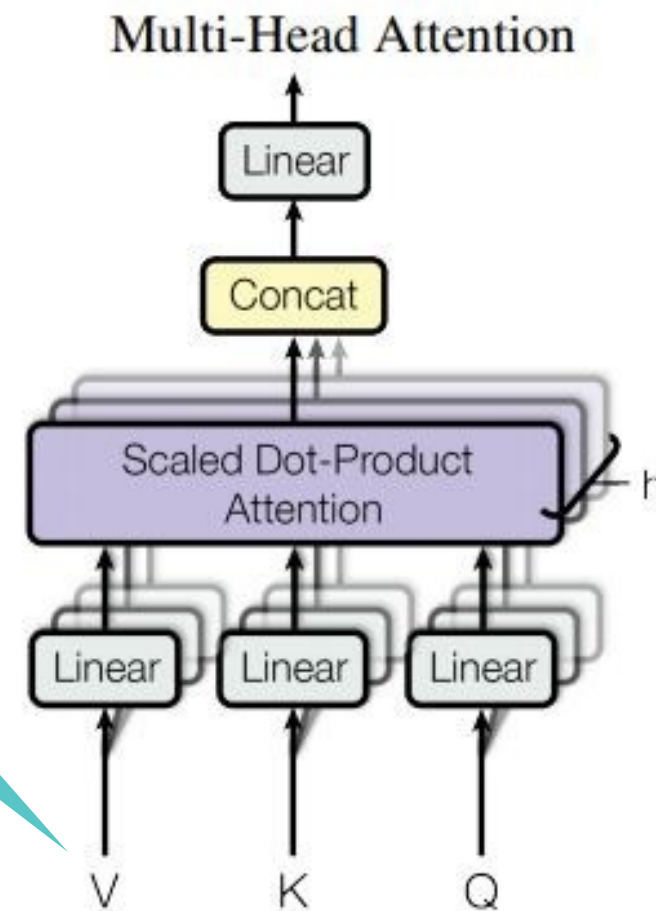


El
jaguar
dormía
en
su
árbol
...

Key Vector

El
jaguar
dormía
Value Vector?
su
árbol

Query Vector



Attention

Background



El

jaguar

dormía

en

su

árbol

El

jaguar

dormía

en

su

árbol

Attention Background



El
jaguar
dormía
en
su
árbol

El
jaguar
dormía
en
su
árbol

Puede calcular la
compatibilidad de cada
palabra identificador (*key*)
con todas las otras palabras
de búsqueda (*query*)



Attention Background

El

jaguar

dormía

en

su

árbol

El

jaguar

dormía

en

su

árbol

Puede calcular la compatibilidad de cada palabra identificador (*key*) con todas las otras palabras de búsqueda (*query*)

**Se llama vector
valor**



Attention Background

El

jaguar

dormía

en

su

árbol

El

jaguar

dormía

en

su

árbol

0,34

-0,44

0,19

-0,24

0,64

0,84

Calcula el producto escalar:
 $v(key) * v(query)$
→ el resultado contra mayor
sea, entonces mayor
compatibilidad entre esas
palabras



Attention Background

El

jaguar

dormía

en

su

árbol

El

jaguar

dormía

en

su

árbol

0,34

-0,44

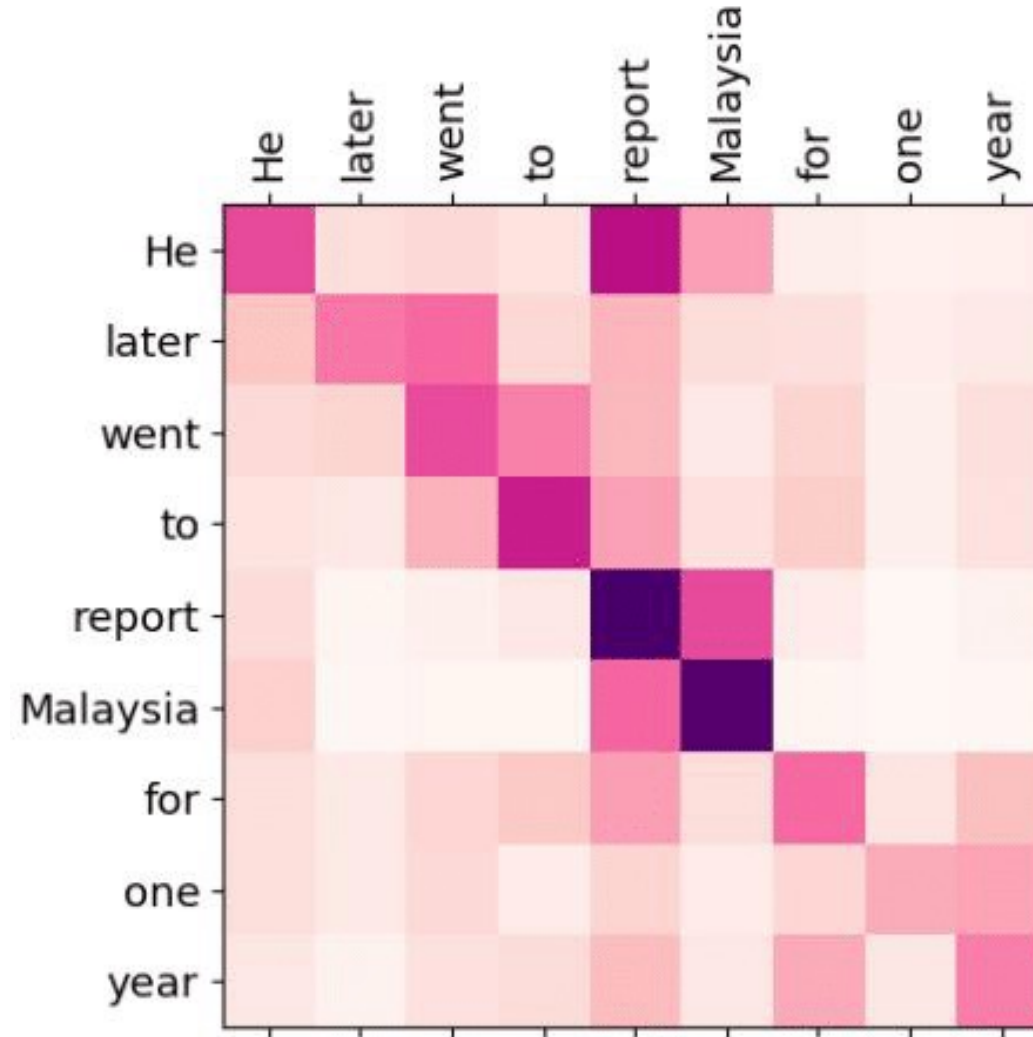
0,19

-0,24

0,64

0,84

Esto es **ATTENTION**





Vector Valor

Input



El **jaguar** dormía en su árbol

Attention



Vector Valor

Input



El **jaguar** dormía en su árbol

Attention

0,34

0,14

0,75

0,11

0,52

0,26



Vector Valor

Input



El **jaguar** dormía en su árbol

Attention

0,34

0,14

0,75

0,11

0,52

0,26

**Vectores
Valor**

0,34

-0,44

0,19

...

0,84



Vector Valor

Input



El **jaguar** dormía en su árbol

Attention

0,34

0,14

0,75

0,11

0,52

0,26

**Vectores
Valor**

0,34

0,31

-0,44

-0,54

0,19

0,49

...

...

0,84

0,34



Vector Valor

Input



El **jaguar** dormía en su árbol

Attention

0,34

0,14

0,75

0,11

0,52

0,26

**Vectores
Valor**

0,34

0,31

0,45

-0,44

-0,54

-0,14

0,19

0,49

0,26

...

...

...

...

0,84

0,34

0,43



Vector Valor

Input



El **jaguar** dormía en su árbol

Attention

0,34 0,14 0,75 0,11 0,52 0,26

**Vectores
Valor**

0,34		0,31				0,45
-0,44		-0,54				-0,14
0,19	+	0,49	+		+	0,26
...	
0,84		0,34				0,43



Vector output



Vector Valor

El

jaguar

dormía

en

su

árbol

El

jaguar

dormía

en

su

árbol

Vector output #1

Vector output #2

Vector output #3

Vector output #4

Vector output #4

Vector output #5



Vector Valor

El

jaguar

dormía

en

su

árbol

El

jaguar

dormía

en

su

árbol

Vector output #1

Vector output #2

Vector output #3

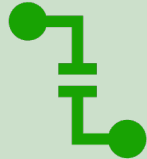
Vector output #4

Vector output #4

Vector output #5

Estos vectores valor
ahora ya si tienen
el contexto de todo
el texto

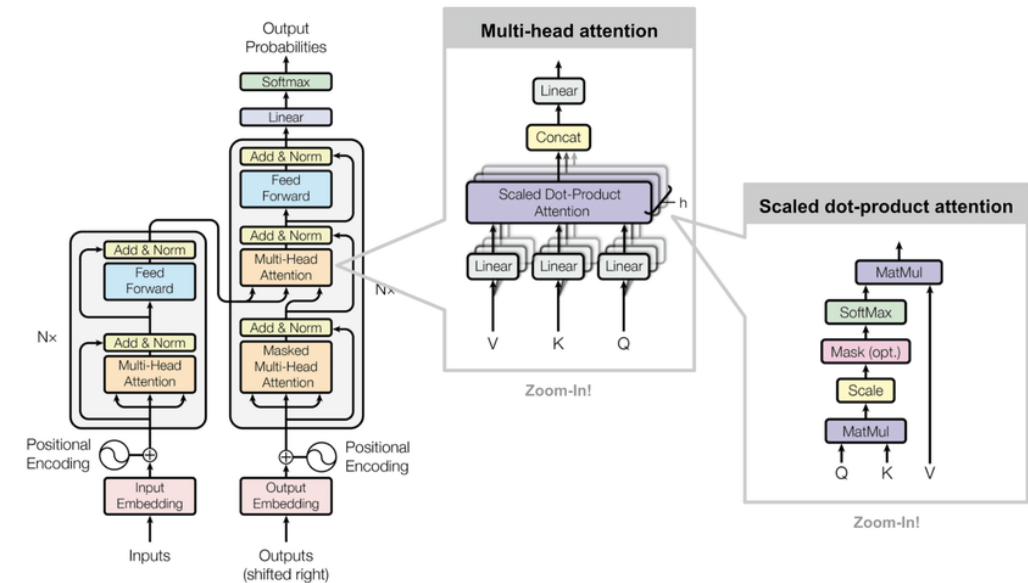
Encoder-Decoder with attention



El **mecanismo de atención** (ϕ) aprende un conjunto de pesos de atención que capturan la relación entre los **vectores codificados** (v) y el **estado oculto del decodificador** (h) para generar un **vector de contexto** (c) a través de una suma ponderada de todos los estados ocultos del codificador.



De este modo, el decodificador tendría **acceso a toda la secuencia de entrada**, centrándose específicamente en la información de entrada **más relevante** para generar la **salida**.





Transformers



ETS de
Ingeniería
Informática

UNED



Consideraciones previas



La arquitectura del Transformer prescinde de cualquier recurrencia y se basa únicamente en un mecanismo de autoatención (*self-attention*).



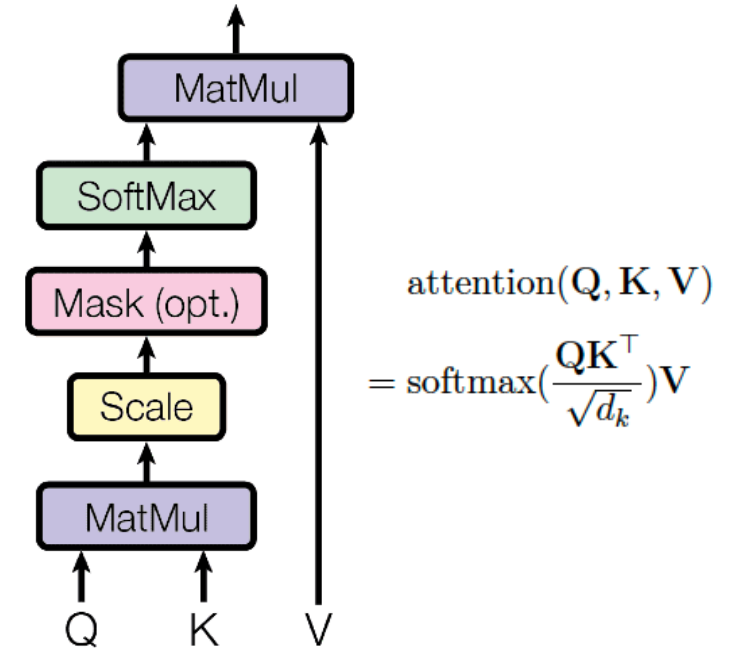
Self-attention se basa en el uso de **vectores Q, K y V**, que se generan multiplicando la representación del codificador de la misma secuencia de entrada con diferentes matrices de pesos.



El Transformer utiliza la atención de producto escalar, en la que cada consulta se compara con una **BBDD de claves** mediante una operación de producto escalar en el proceso de generación de los pesos de atención.



A continuación, estos **pesos se multiplican por los valores** para generar un vector de atención final.



Consideraciones previas



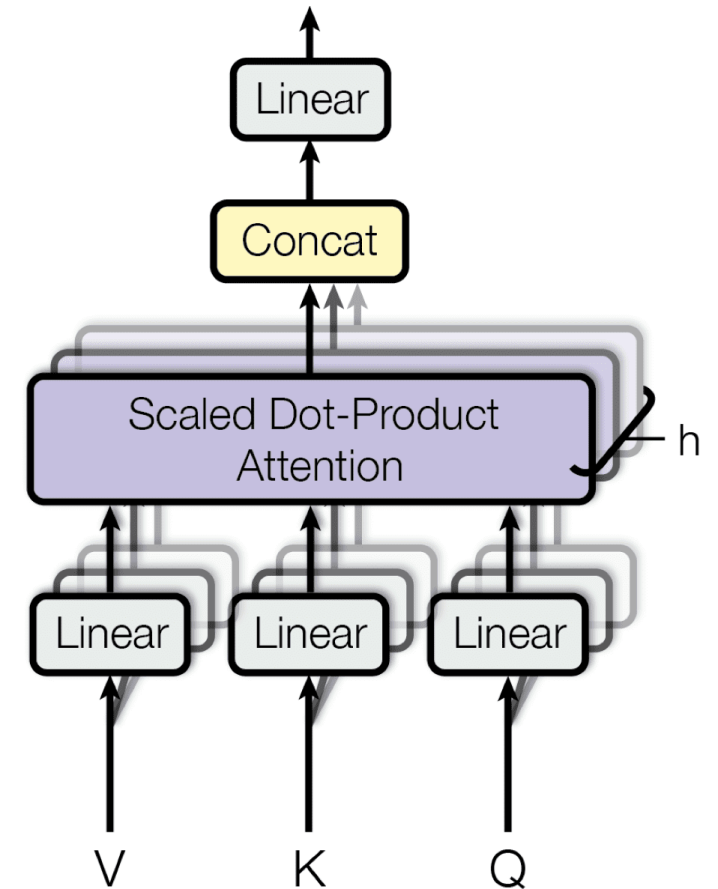
Además, se pueden **apilar** varias capas de atención en paralelo (*multi-head attention*).



Cada *head* trabaja en paralelo sobre **diferentes transformaciones** lineales de la misma entrada, y luego las **salidas** de las *heads* se **concatenan** para producir el resultado de atención final.



Dado que las múltiples cabezas pueden trabajar de forma **independiente y en paralelo**, un modelo *multi-head* puede hacer que cada *head* atienda a **diferentes** elementos de la secuencia (**proyecciones**).





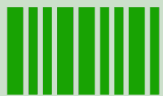
Arquitectura



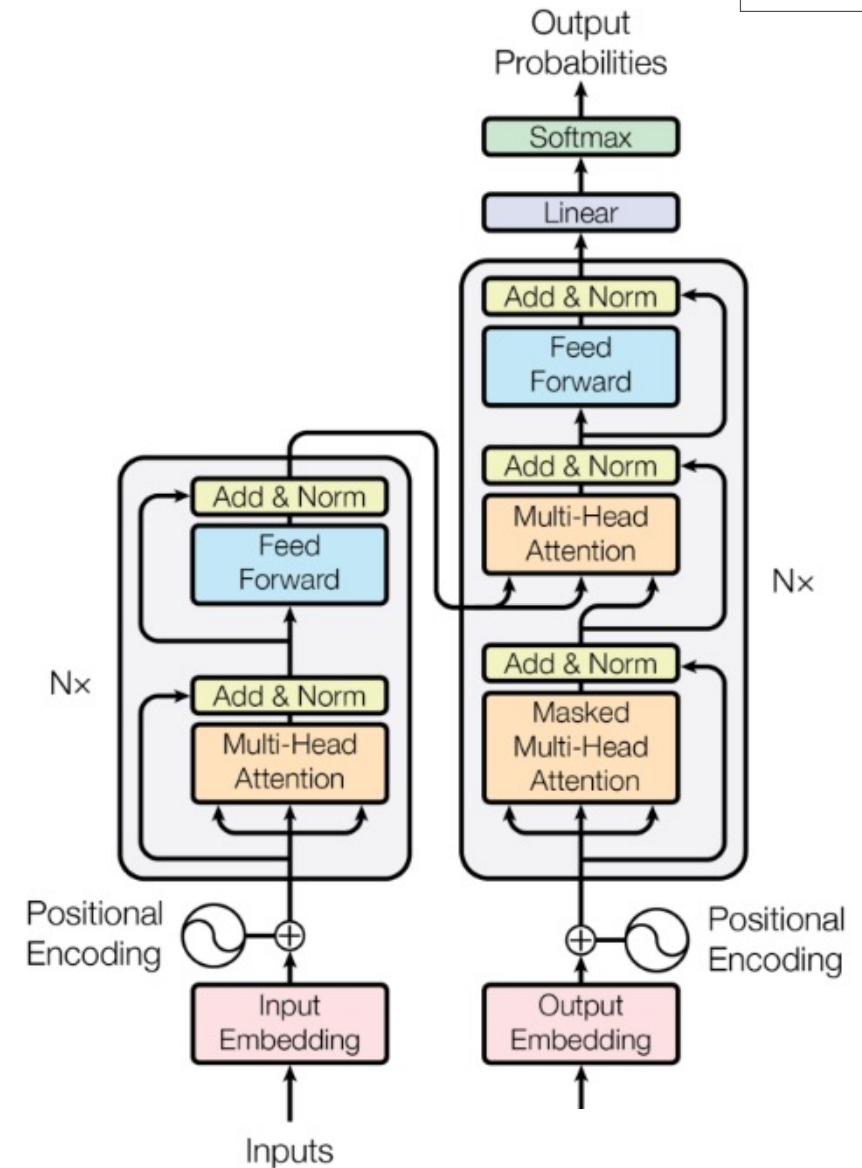
El Transformer sigue una estructura **codificador-decodificador** pero no depende de la recurrencia ni de las convoluciones para generar una salida.



La tarea del **codificador** (mitad izquierda) es **mapear una secuencia de entrada** a una secuencia de representaciones continuas, que luego se **introduce en un decodificador**.



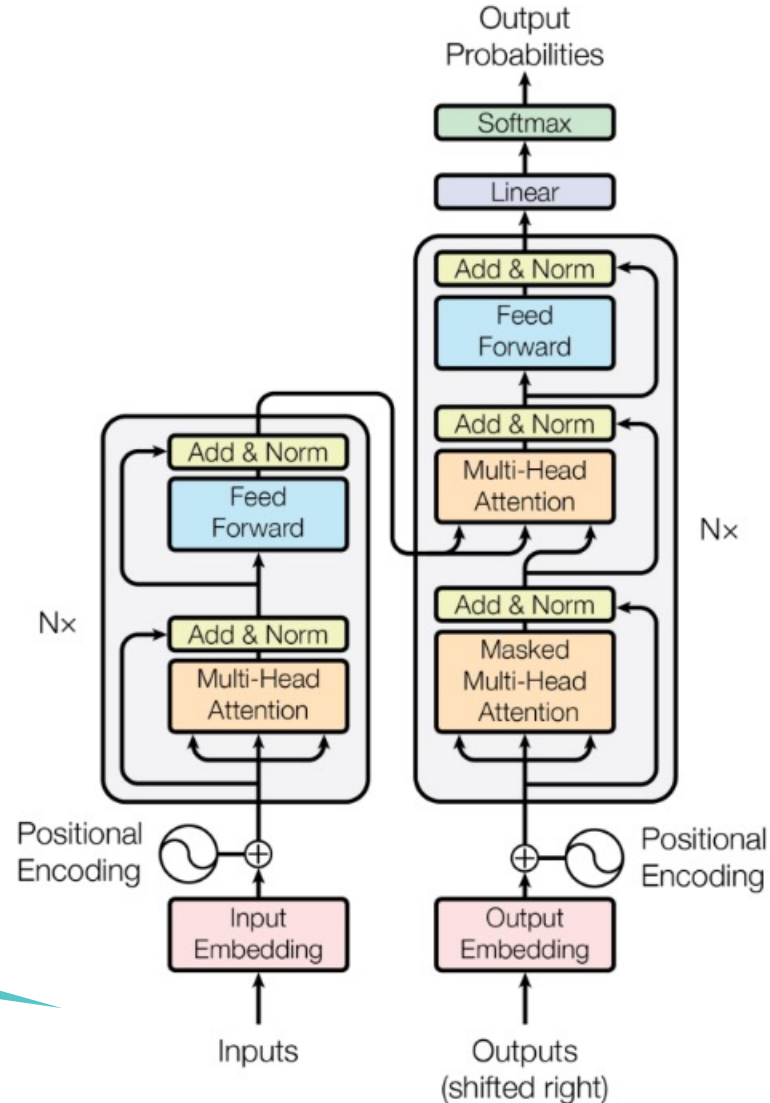
El **decodificador** (mitad derecha) recibe la salida del codificador junto con la salida del decodificador en el paso de tiempo anterior para **generar una secuencia de salida**.



Arquitectura Input



Tenemos una entrada embedding.
Los Transformers asumen todas
las palabras a la vez (al contrario
de las RNN que era palabra por
palabra)



Arquitectura

Input



TRANSFORMER

↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
El jaguar dormía en la rama del árbol con su cola de soporte.

Arquitectura

Input



TRANSFORMER

↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
El jaguar dormía en la rama del árbol con su cola de soporte.

Entran todas las palabras
a la vez

Arquitectura

Input



TRANSFORMER

↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
dormía jaguar su la soporte rama del cola árbol con en El de.

Entran todas las palabras
a la vez → ¡¡¡Sin importar el
Orden!!!!

Arquitectura

Input



TRANSFORMER

↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
dormía jaguar su la soporte rama del cola árbol con en El de.

Entran todas las palabras
a la vez → ¡¡¡Sin importar el
Orden!!!!

¿Y cómo tiene en cuenta
el contexto?



Posicionamiento

El **jaguar** dormía en su árbol



Posicionamiento

El jaguar dormía en su árbol			
0,34	0,31	0,45	
-0,44	-0,54	-0,14	
0,19	0,49	0,26	
...
0,84	0,34	0,43	

Vectores de palabras



Información posicional

El jaguar dormía en su árbol

0,34	1	0,31	2	0,45	3	
-0,44	1	-0,54	2	-0,14	3	
0,19	1	0,49	2	0,26	3	
...
0,84	1	0,34	2	0,43	3	

Vectores de palabras

Pero añadimos su **posición** en el texto



Información posicional

El		jaguar	su		árbol
0,34	1	0,31
-0,44	1
0,19	1	0,49
...
0,84	1	0,34	2	0,43	3

Problema: si tenemos texto muy grandes tendríamos posiciones muy elevadas

Vectores de palabras

Pero añadimos su **posición** en el texto



Información posicional

El jaguar dormía en su árbol					
0,34	1/6	0,31	2/6	0,45	3/6
-0,44	1/6	-0,54	2/6	-0,14	3/6
0,19	1/6	0,49	2/6	0,26	3/6
...
0,84	1/6	0,34	2/6	0,43	3/6

Solución: Normalizar el vector de posicionamiento por el número total de palabras



Información posicional

Vector posición:
 $4/6 = 0,66$

El jaguar dormía en su árbol



Información posicional

El jaguar dormía en su árbol

Vector posición:
 $4/6 = 0,66$

El jaguar dormía

Vector posición:
 $2/3 = 0,66$



Información posicional

Vector posición:
 $4/6 = 0,66$

El jaguar dormía en su árbol

El jaguar dormía

Vector posición:
 $2/3 = 0,66$

Entonces, ¿0,66 es la posición
2 o 4?



Codificación binaria

El

jaguar

dormía



3 3 3 ... 3 3 3



0 0 0 ... 0 1 1

en

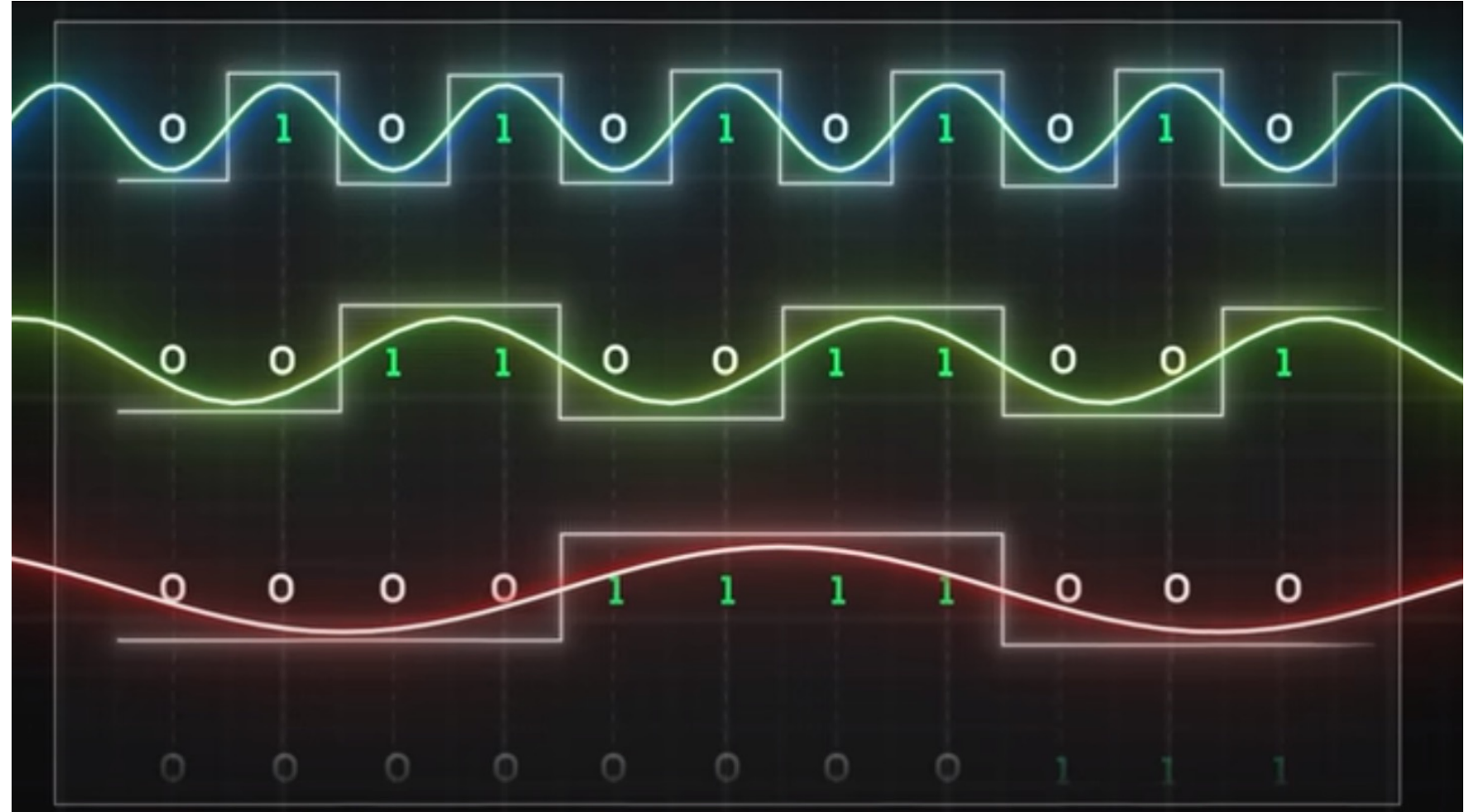
su

árbol

Codificación
binaria

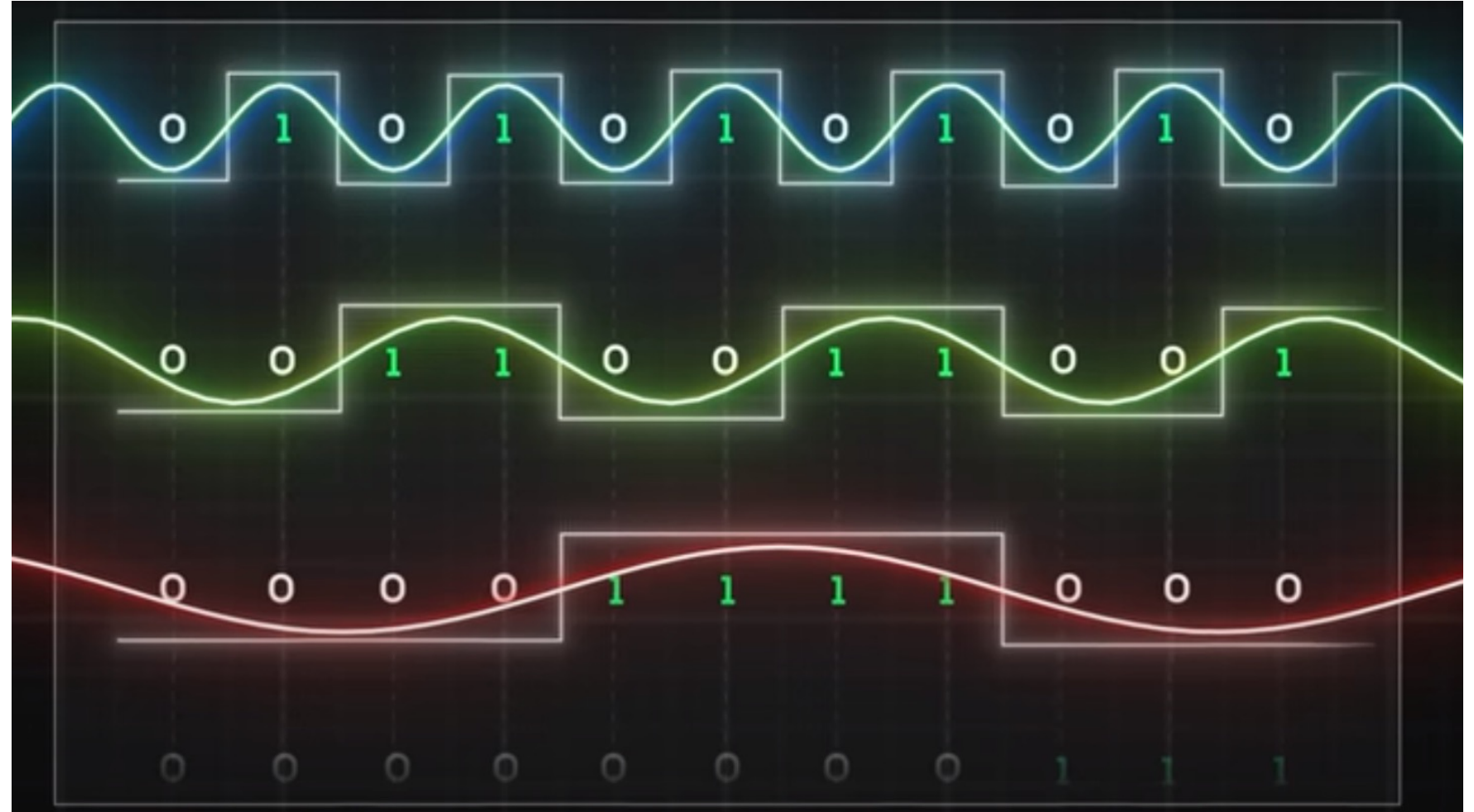
Codificación binaria

Se traduce a una función Senoidal



Codificación binaria

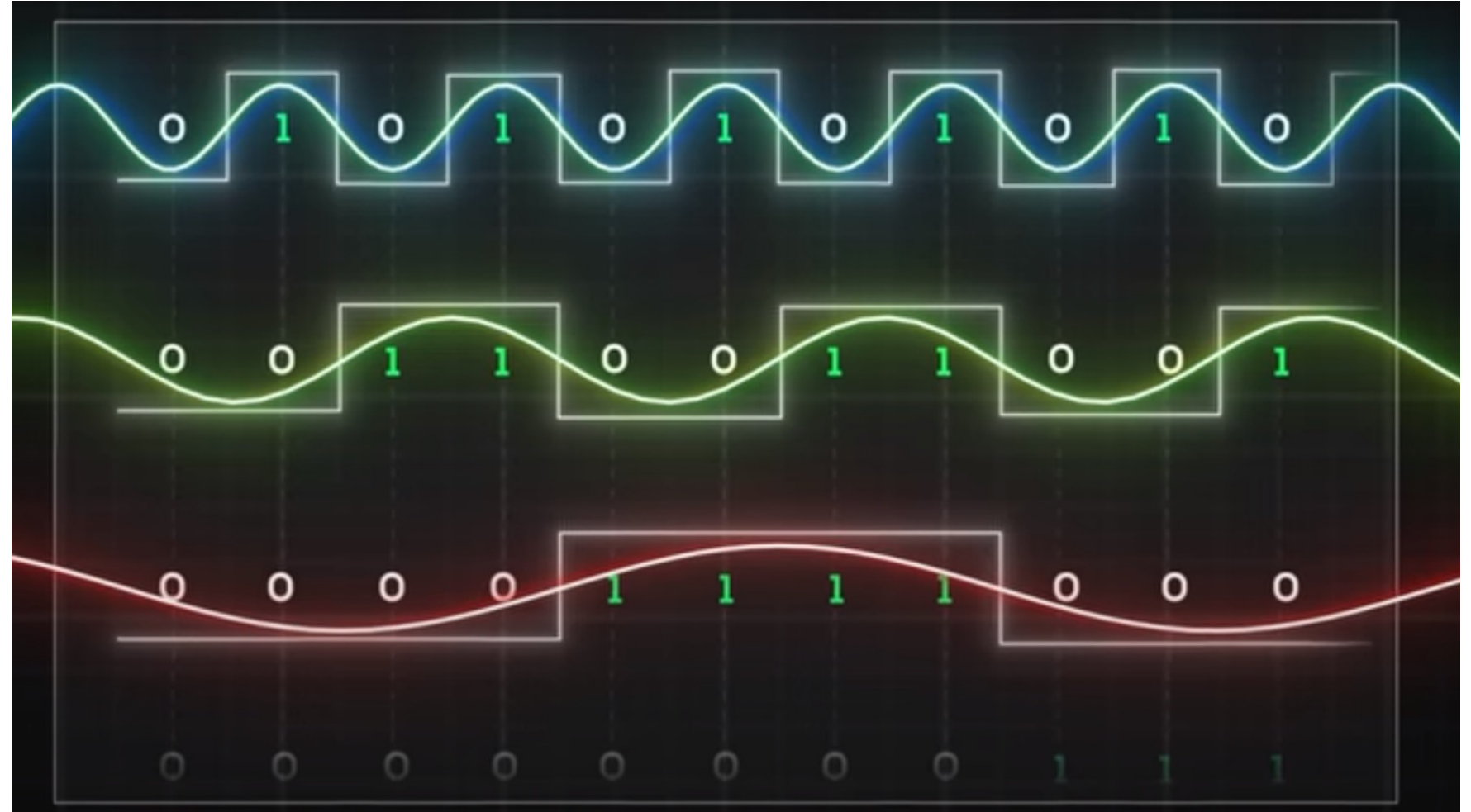
$\text{Sen}(1/1 * \text{pos})$





Codificación binaria

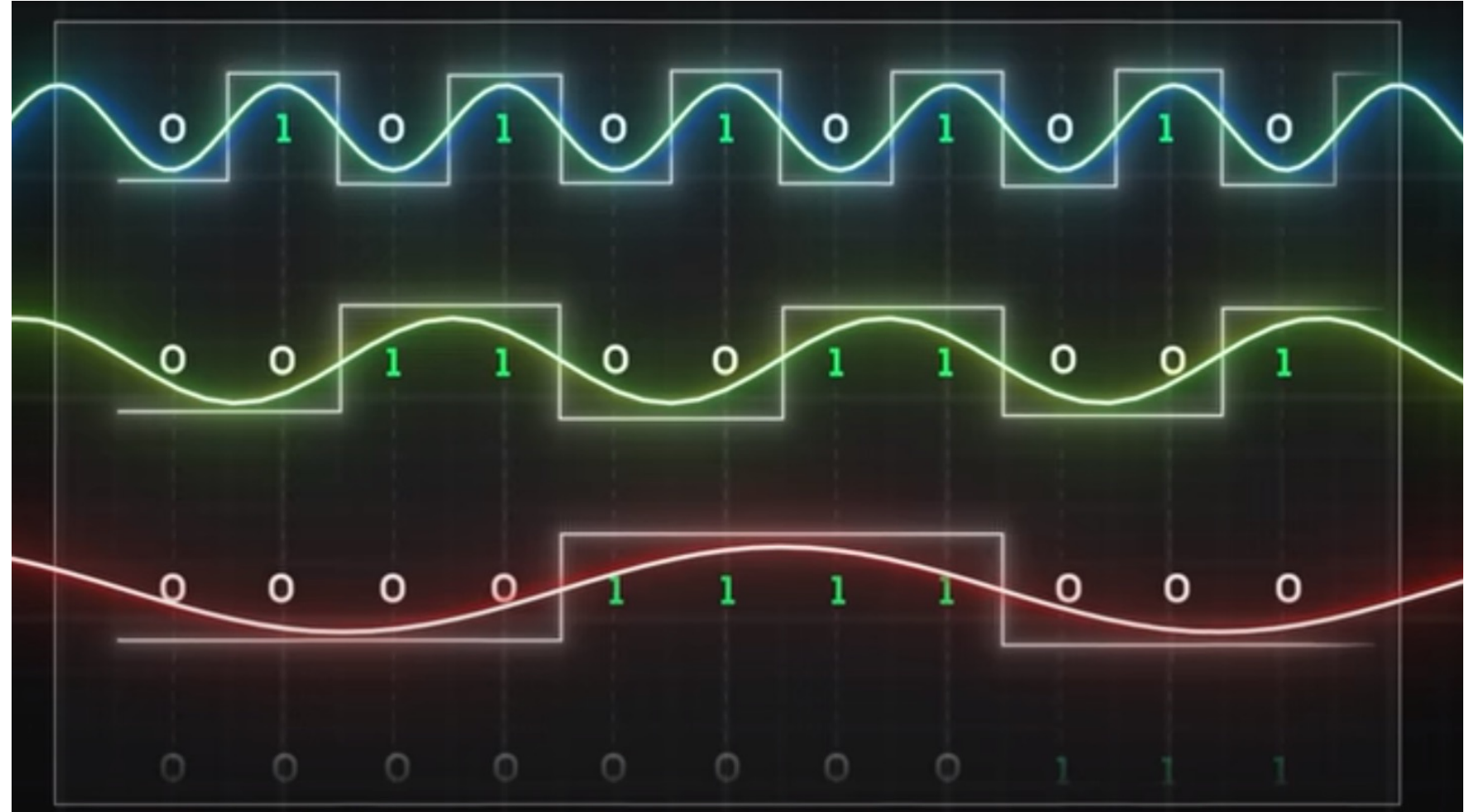
Sen($\frac{1}{2} * \text{pos}$)





Codificación binaria

Sen($\frac{1}{3} * \text{pos}$)

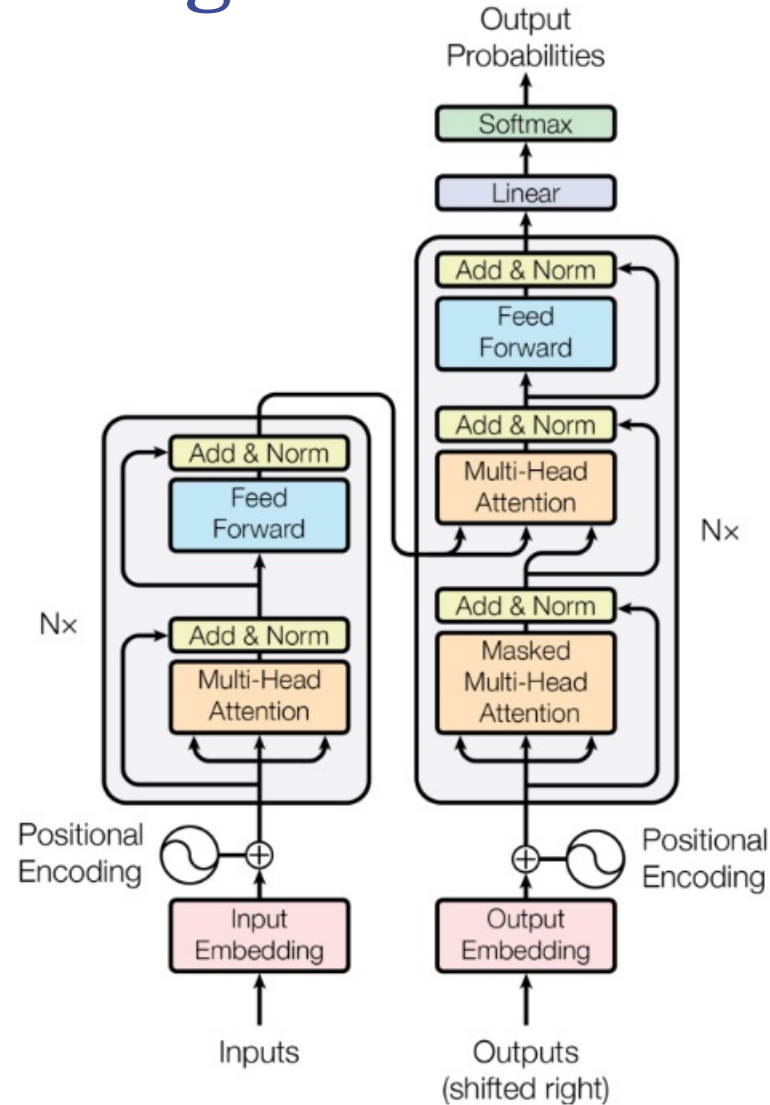




Arquitectura

Positional Encoding

Eso es!



Arquitectura

Input

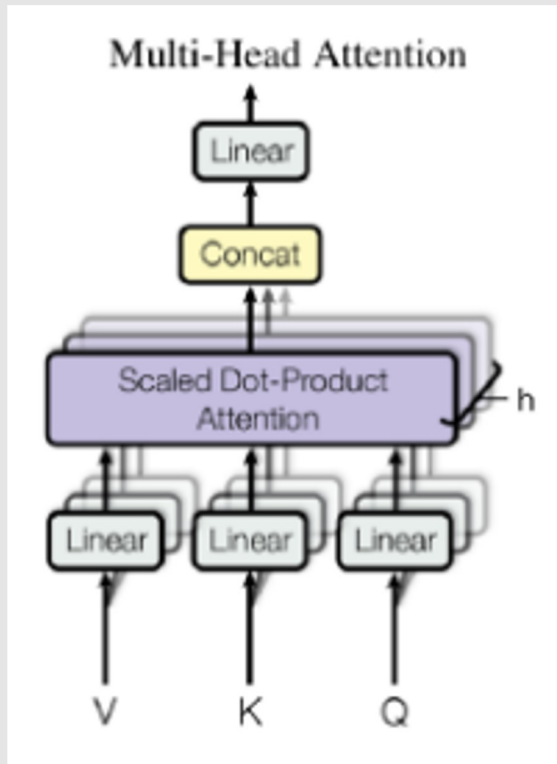


TRANSFORMER

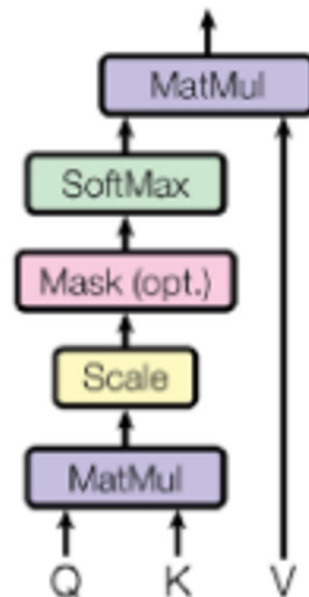
↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
dormía jaguar su la soporte rama del cola árbol con en El de.

Y al pasar todas las palabras
al mismo tiempo y sin importar
el orden → se puede **Paralelizar!!!**

Transformer – Encoder

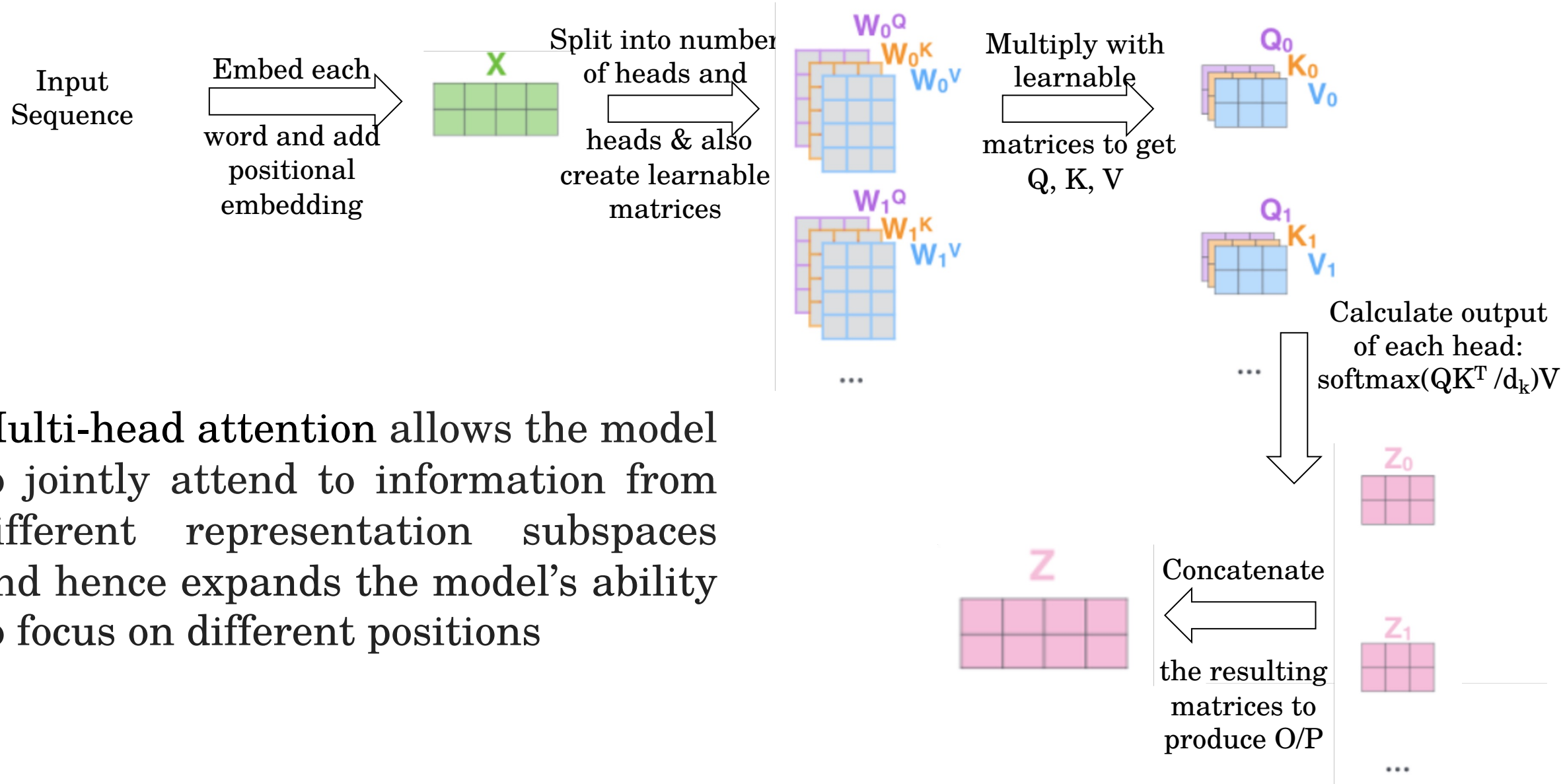


Scaled Dot-Product Attention



- Like LSTM, Transformer is an architecture for transforming one sequence into another one with the help of two parts (Encoder and Decoder), but it differs from the existing seq2seq models because it does not imply any Recurrent Networks (GRU, LSTM, etc.).
- 3 main parts : Position Embeddings + Multi Head Attention + Feed- forward Layers.
- Q (query) matrix: vector representation of one word in the sequence
- K (keys) matrix: vector representations of all the words in the sequence
- V (values) matrix: vector representations of all the words in the sequence. For the encoder, V consists of the same word sequence than Q.
- Attention weights = $\text{softmax}(QK^T / d_k)$
- These weights are defined by how each word of the sequence (represented by Q) is influenced by all the other words in the sequence (represented by K).
- Those weights are then applied to all the words in the sequence that are introduced in V.

Multi-Head Attention



Multi-head attention allows the model to jointly attend to information from different representation subspaces and hence expands the model's ability to focus on different positions



Vision Transformer (ViT)



ETS de
Ingeniería
Informática



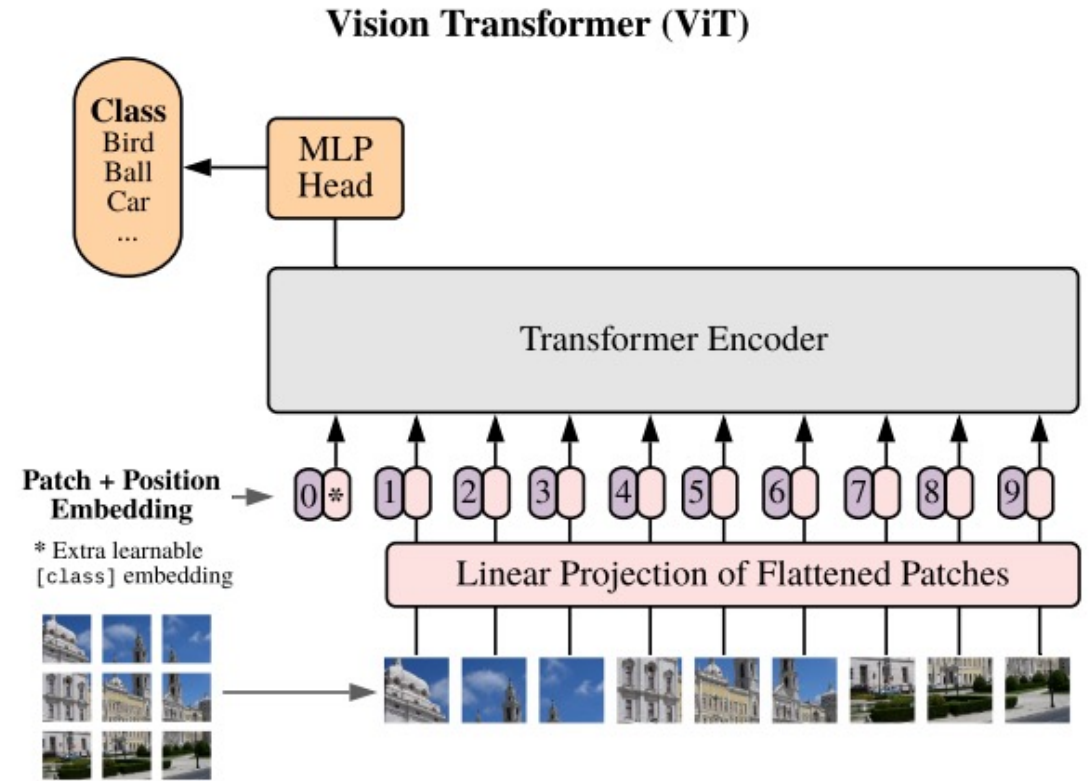
UNED

Arquitectura



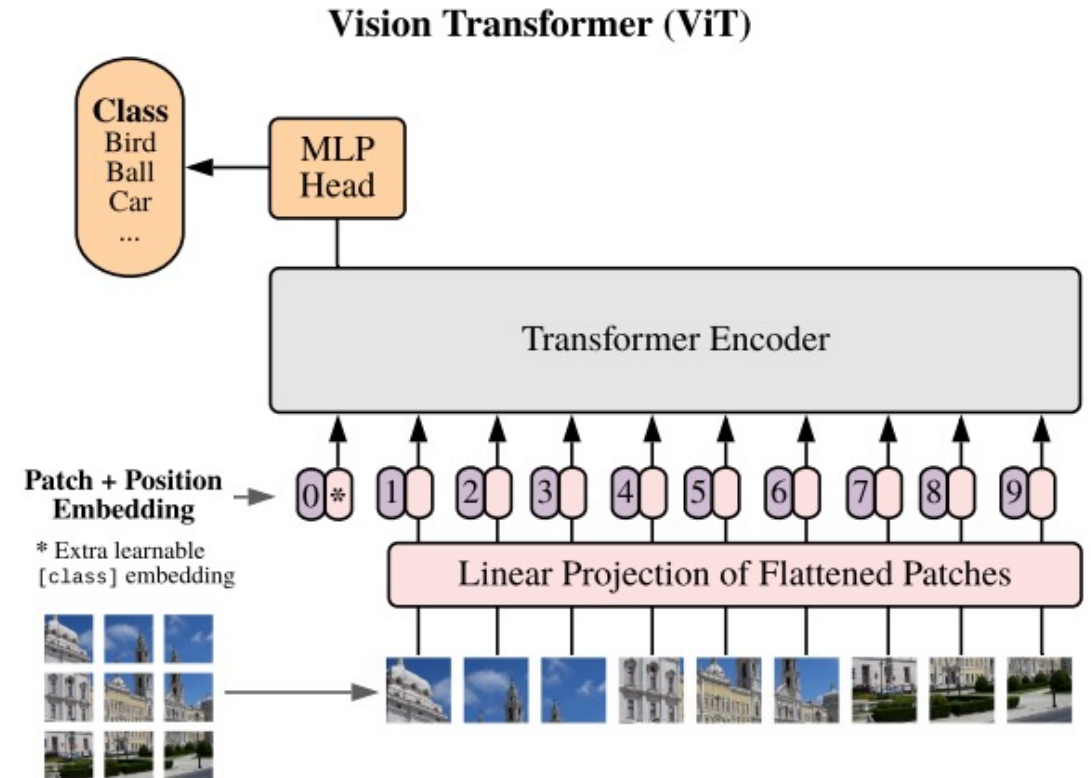
Disposiciones previas

- Los datos de entrada al Transformer se proporcionan en forma de **imágenes bidimensionales**.



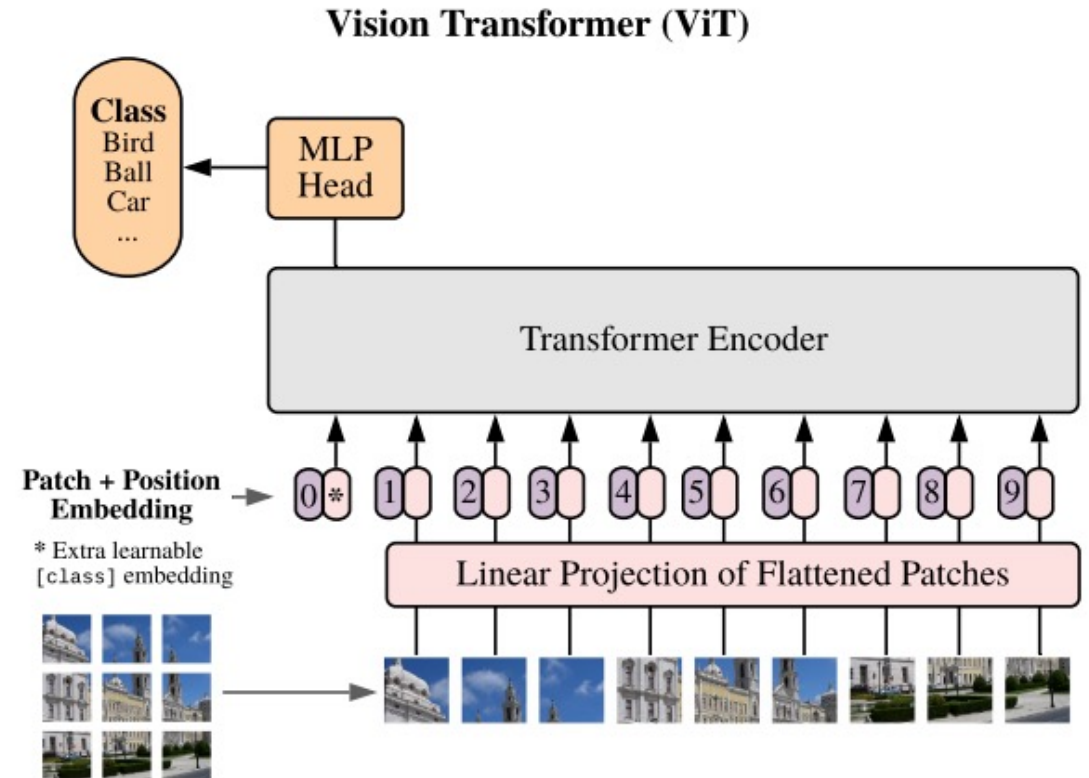
Disposiciones previas

- Los datos de entrada al Transformer se proporcionan en forma de **imágenes bidimensionales**.
- La imagen de entrada, de alto H , ancho W y canales C , se corta en **patches bidimensionales** más pequeños.



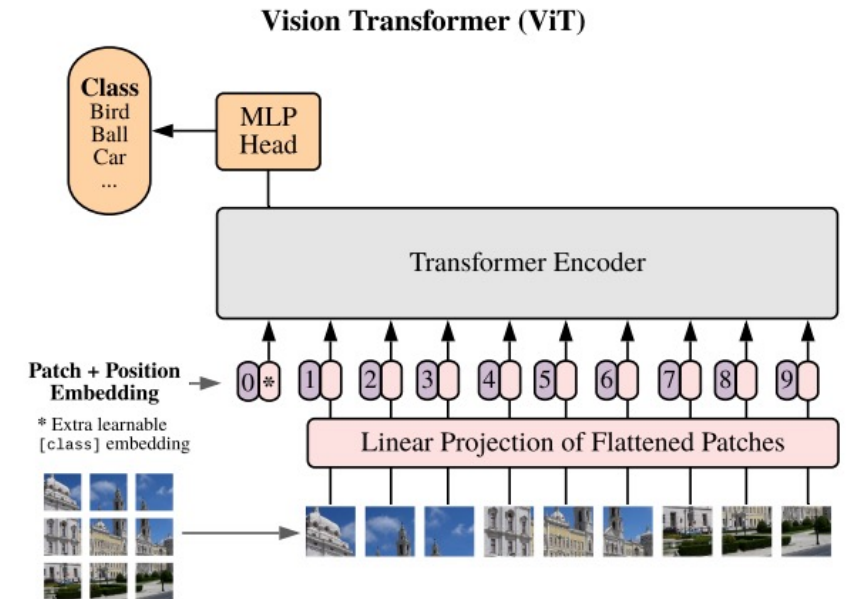
Disposiciones previas

- Los datos de entrada al Transformer se proporcionan en forma de **imágenes bidimensionales**.
- La imagen de entrada, de alto H , ancho W y canales C , se corta en **patches bidimensionales** más pequeños.
- Esto da como resultado un número $N=HW/P^2$ de **patches**, donde cada patch tiene una resolución de $P \times P$ píxeles.



Operaciones ViT

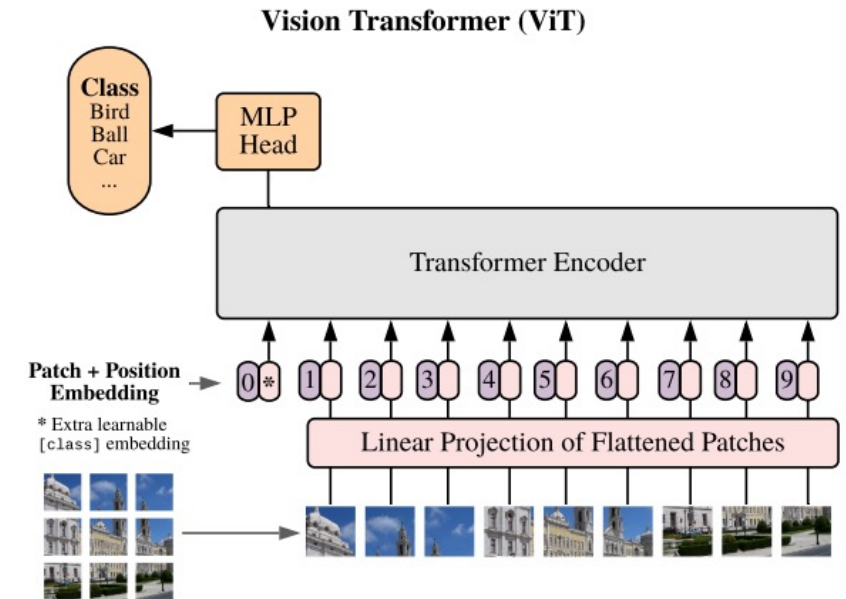
- Cada patch de imagen **se aplan**a en un vector \mathbf{x}_p^n de longitud $P \times C$ donde, $n=1, \dots, N$.
- Se genera una **secuencia de patches** de imagen embedding asignando los patches aplanados a dimensiones D , con una proyección lineal entrenable E .



Operaciones ViT

- Cada patch de imagen **se aplanan en un vector** \mathbf{x}_p^n de longitud $P \times C$ donde, $n=1, \dots, N$.
- Se genera una **secuencia de patches** de imagen embedding asignando los patches aplanados a dimensiones D , con una proyección lineal entrenable E .
- A la secuencia de imágenes embeddings se le añade una clase x_{class} . El valor de x_{class} representa el resultado de la clasificación y .
- Los patches embedding finalmente se aumentan con embedding posicionales unidimensionales E_{pos} , introduciendo así **información posicional** en la entrada, que se aprende durante el entrenamiento.
- La secuencia de vectores embeddings resultante:

$$\mathbf{z}_0 = [\mathbf{x}_{class}; \mathbf{x}_p^1 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{pos}$$

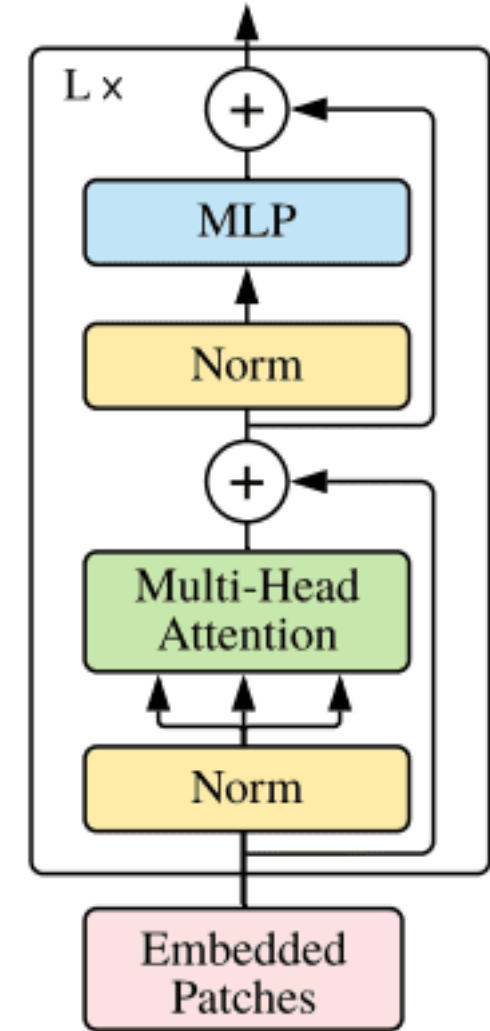




Operaciones ViT

- Para realizar la clasificación, alimentan z_0 en la entrada del codificador del Transformer, que consta de una pila de capas idénticas.

Transformer Encoder

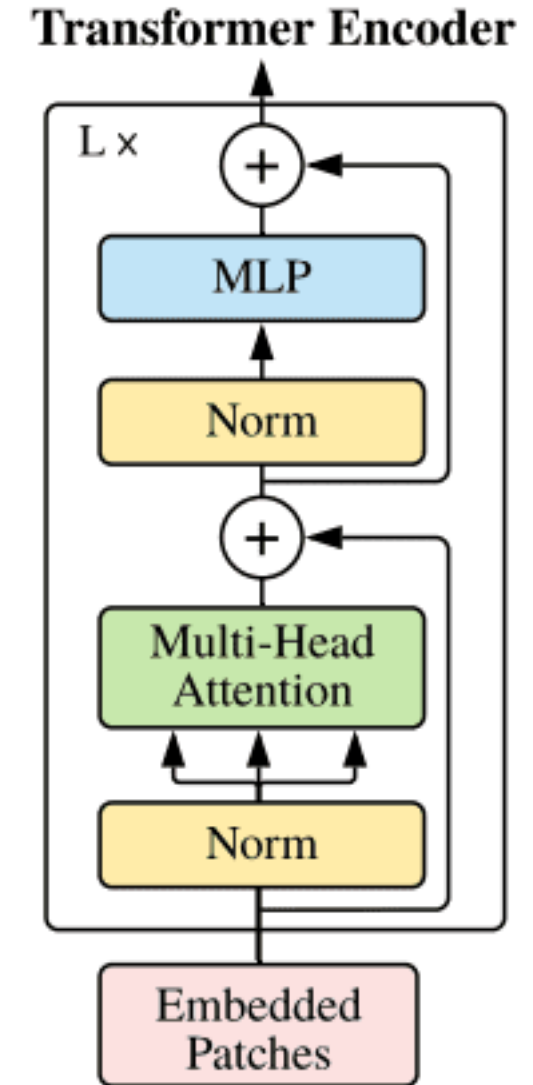


$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{\text{pos}}$$



Operaciones ViT

- Para realizar la clasificación, alimentan z_0 en la entrada del codificador del Transformer, que consta de una pila de capas idénticas.
- Luego, proceden a tomar el valor de x_{class} en la capa L de la salida del codificador y lo introducen en un MLP head de clasificación.



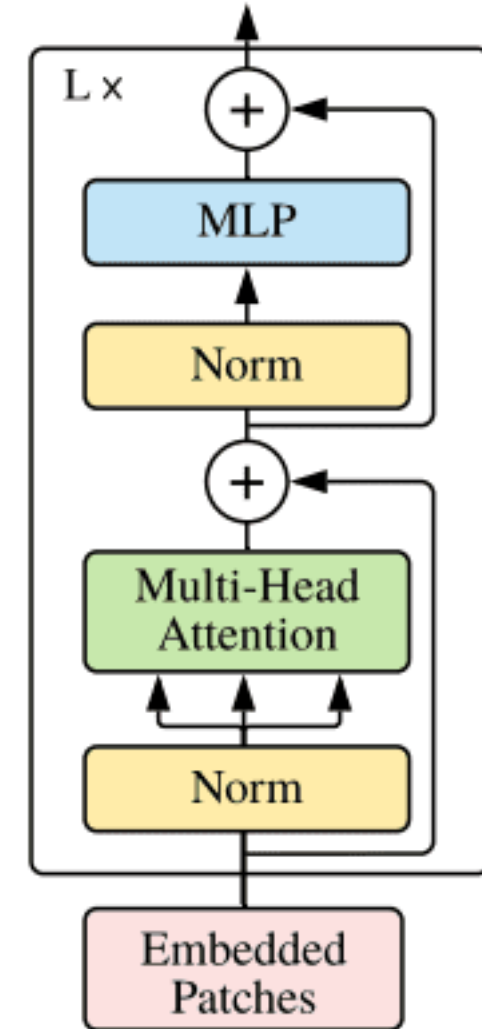
$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{\text{pos}}$$



Operaciones ViT

- Para realizar la clasificación, alimentan z_0 en la entrada del codificador del Transformer, que consta de una pila de capas idénticas.
- Luego, proceden a tomar el valor de x_{class} en la capa L de la salida del codificador y lo introducen en un MLP head de clasificación.
- El MLP de clasificación implementa la no linealidad de unidad lineal de error gaussiano (GELU).
- ViT emplea la parte Codificador de la arquitectura Transformer original.

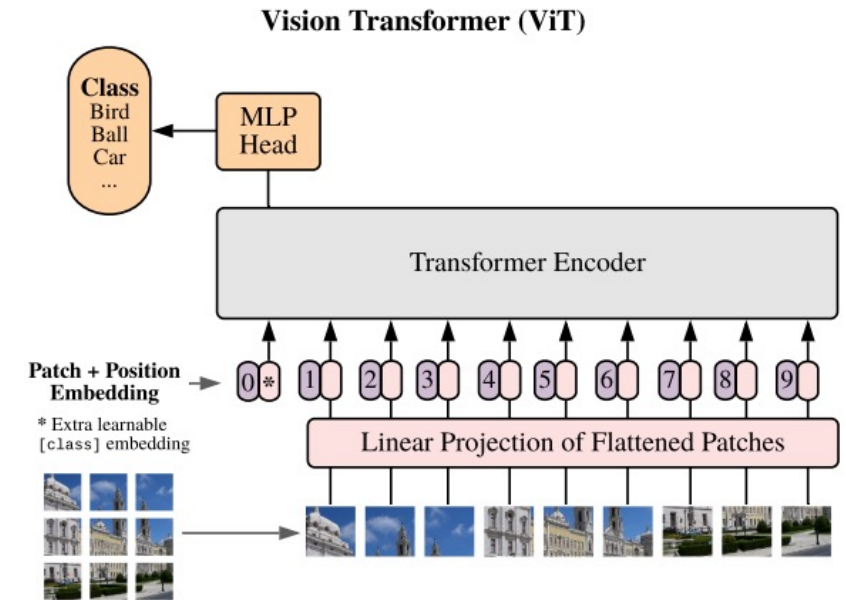
Transformer Encoder



$$\mathbf{z}_0 = [\mathbf{x}_{class}; \mathbf{x}_p^1 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{pos}$$

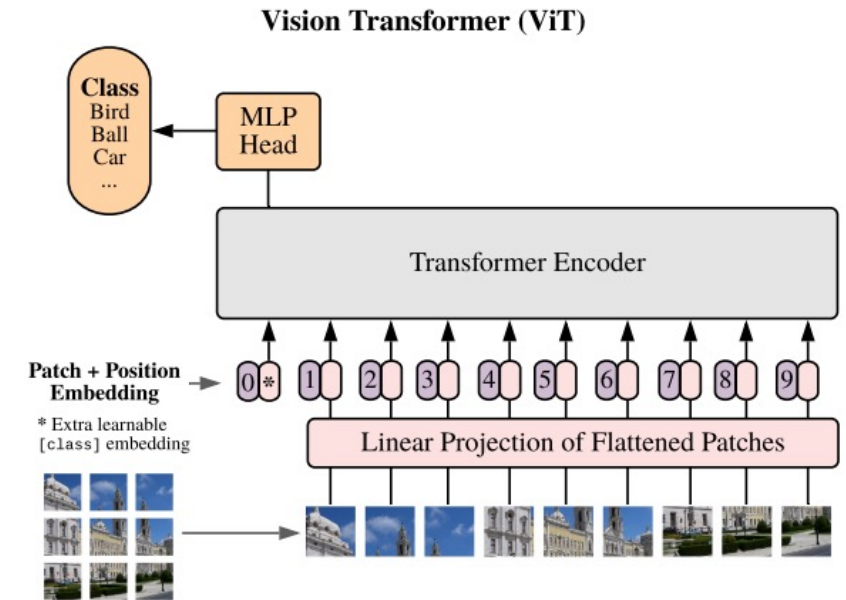
Mecanismo de funcionamiento

- La entrada al codificador es una secuencia de **patches** de imágenes embedding, que también se aumenta con información **posicional**.



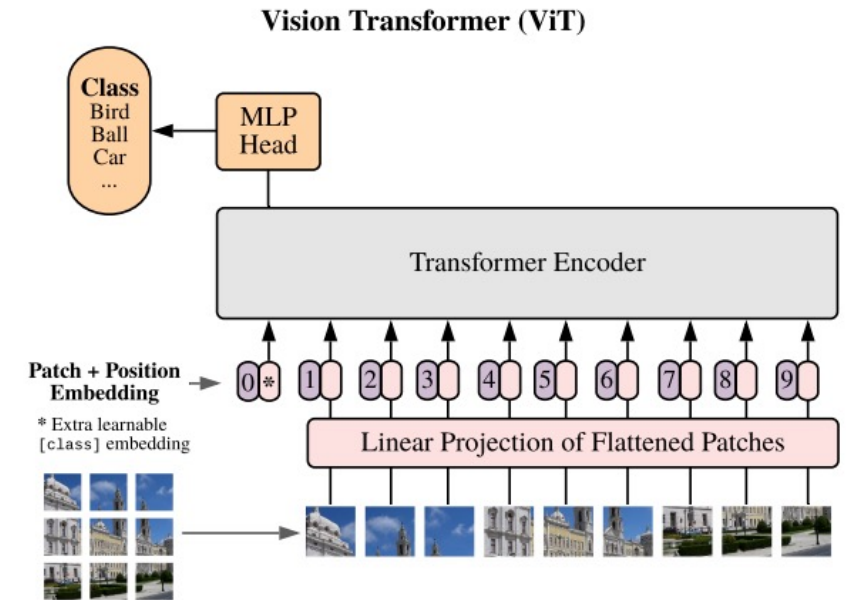
Mecanismo de funcionamiento

- La entrada al codificador es una secuencia de **patches** de imágenes embedding, que también se aumenta con información **posicional**.
- Un head de clasificación sigue a la salida del Codificador recibe el valor de la **clase incrustada** que se puede aprender para generar una salida de clasificación basada en su estado.



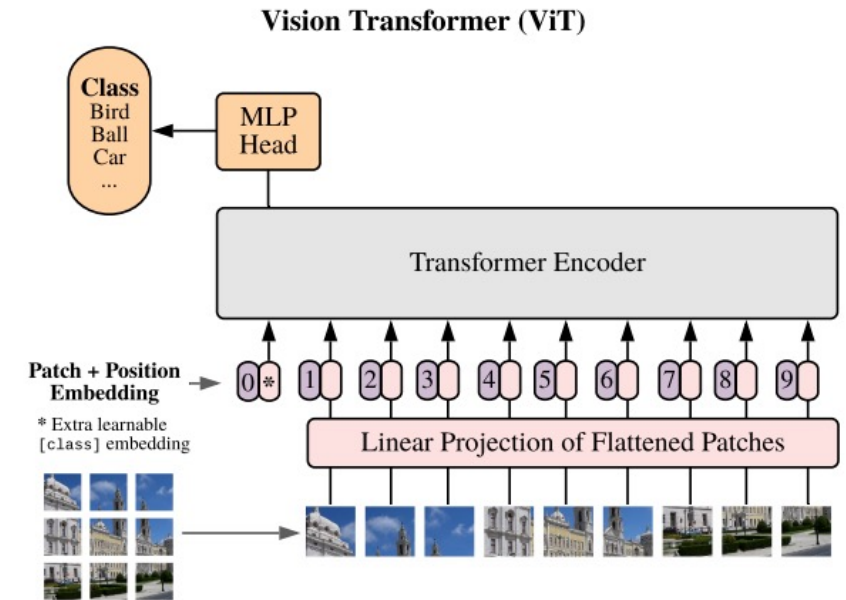
Mecanismo de funcionamiento

- La entrada al codificador es una secuencia de **patches** de imágenes embedding, que también se aumenta con información **posicional**.
- Un head de clasificación sigue a la salida del Codificador recibe el valor de la **clase incrustada** que se puede aprender para generar una salida de clasificación basada en su estado.
- En comparación con una capa convolucional, ViT no genera mapas de características separados para toda la imagen. En cambio, cada patch de la imagen se **convertirá en un embedding** en la que varias características se representan.



Mecanismo de funcionamiento

- La entrada al codificador es una secuencia de **patches** de imágenes embedding, que también se aumenta con información **posicional**.
- Un head de clasificación sigue a la salida del Codificador recibe el valor de la **clase incrustada** que se puede aprender para generar una salida de clasificación basada en su estado.
- En comparación con una capa convolucional, ViT no genera mapas de características separados para toda la imagen. En cambio, cada patch de la imagen se **convertirá en un embedding** en la que varias características se representan.
- Alternativamente, la imagen original puede introducirse en una **CNN** antes de pasarla al Codificador. La secuencia de patches de imagen se obtendría entonces a partir de los **mapas de características** de la **CNN**, mientras que el proceso subsiguiente de incrustación de los patches del mapa de características, **añadiendo un token de clase** y aumentando con información posicional sigue siendo el mismo.





Rendimiento ViT Vs ResNet

Exp. 1: Fine-tuning and testing on ImageNet



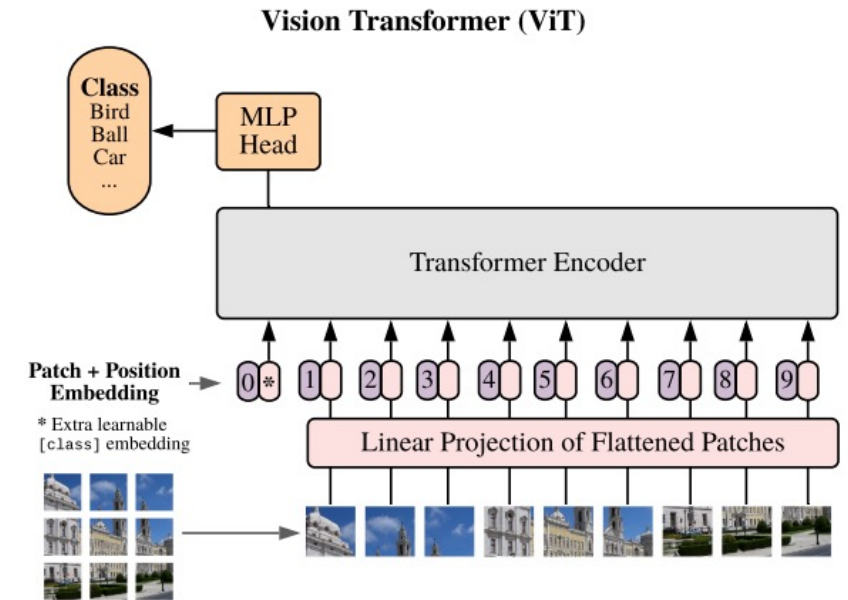
En el conjunto de datos más pequeño (ImageNet), los dos modelos ViT más grandes tuvieron un rendimiento inferior en comparación con su contraparte **más pequeña**. El **rendimiento** de todos los ViT se mantiene por debajo del de los ResNets.



Cuando se entrenan en un conjunto de datos **más grande** (ImageNet-21k), los tres modelos ViT se desempeñaron de **manera similar** entre sí, así como con las ResNets.



Cuando se entrenan en el conjunto de datos **más grande** (JFT-300M), el rendimiento de los modelos ViT más grandes **supera el rendimiento** de los ViT más pequeños y las ResNets.



$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{\text{pos}}$$



Rendimiento ViT Vs ResNet

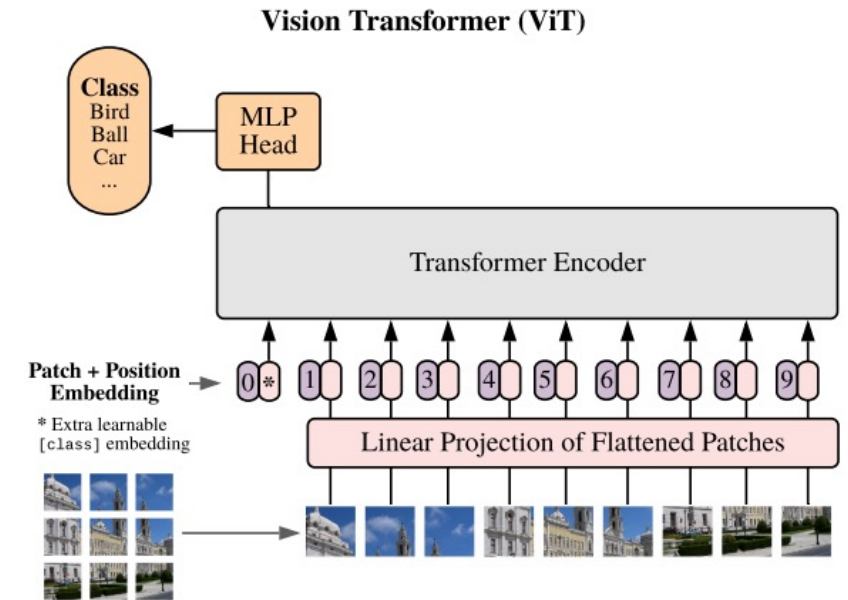
Exp. 2: Investigate the effect of the dataset size

Entrenamiento en subconjuntos aleatorios de diferentes tamaños del conjunto de datos JFT-300M y pruebas en ImageNet para investigar más a fondo el efecto del tamaño del conjunto de datos:

En subconjuntos **más pequeños** del conjunto de datos, los modelos ViT se ajustan más que ResNet y tienen un rendimiento considerablemente **inferior**.

En el subconjunto **más grande** del conjunto de datos, el rendimiento del modelo ViT más grande **supera el rendimiento** de los ResNets.

Este resultado refuerza que el sesgo inductivo convolucional es útil para conjuntos de datos más pequeños, pero para los más grandes, aprender los patrones relevantes directamente a partir de los datos es suficiente, e incluso beneficioso.

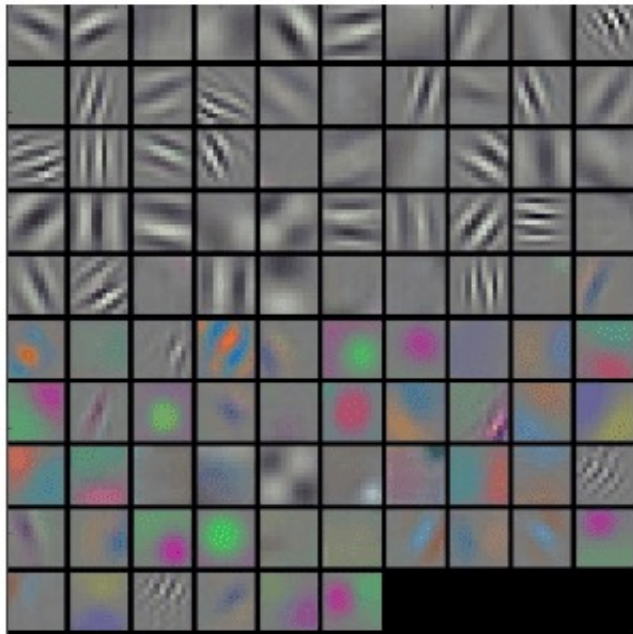


$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{\text{pos}}$$

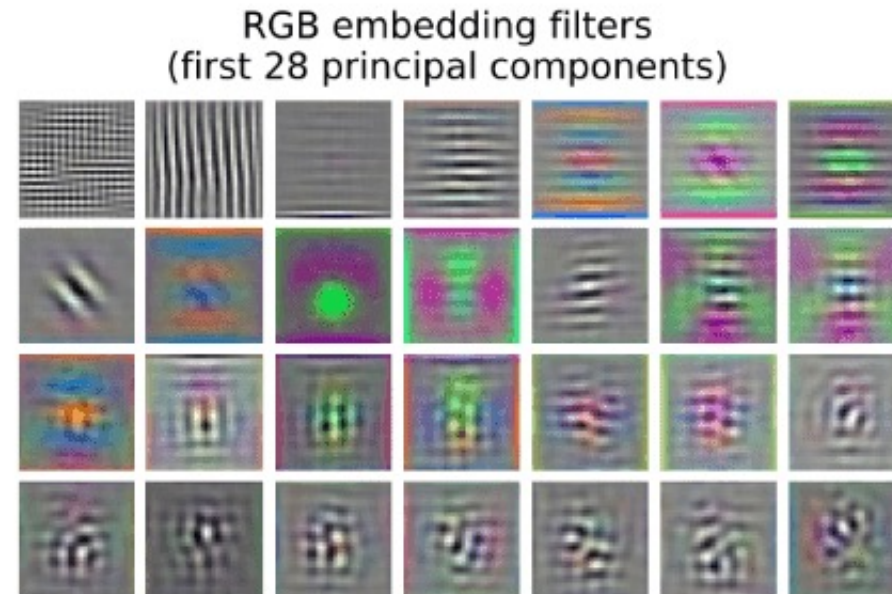
Representación Interna de Datos

- Los filtros de embedding aprendidos que se aplican inicialmente a los patches de imagen en la primera capa de ViT se **parecen a funciones básicas** que pueden extraer las características de bajo nivel dentro de cada patch

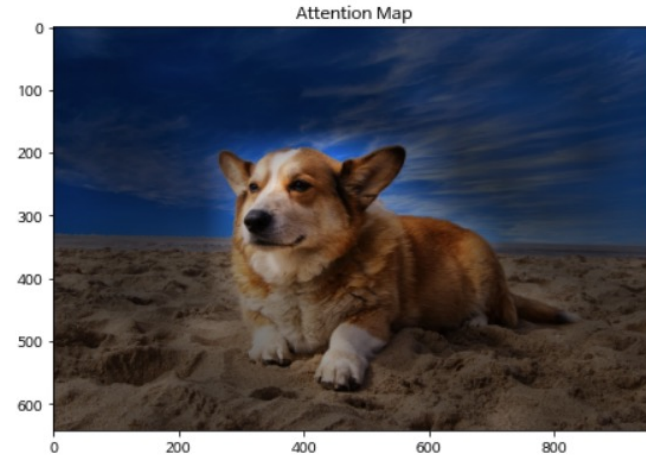
Alexnet 1st conv filters



ViT 1st linear embedding filters



Attention Map & LeGrad

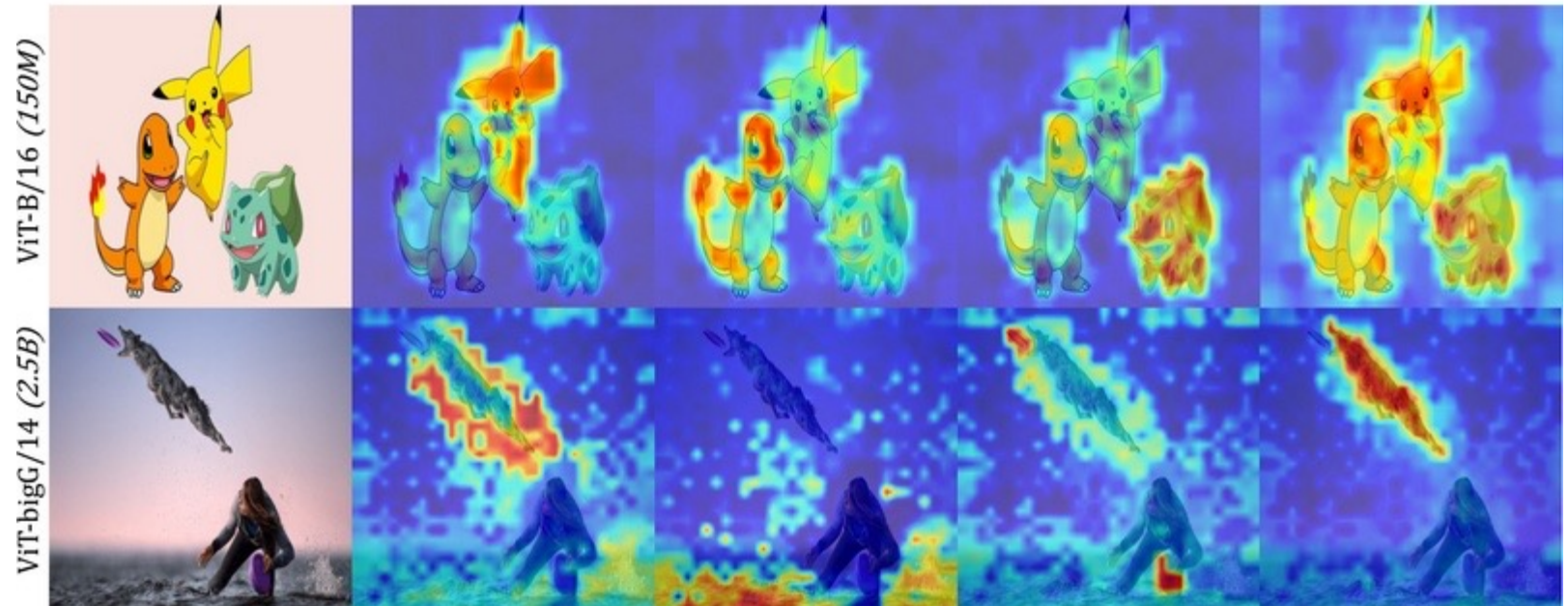


Pikachu

Charmander

Bulbasaur

Pokemons



Jump

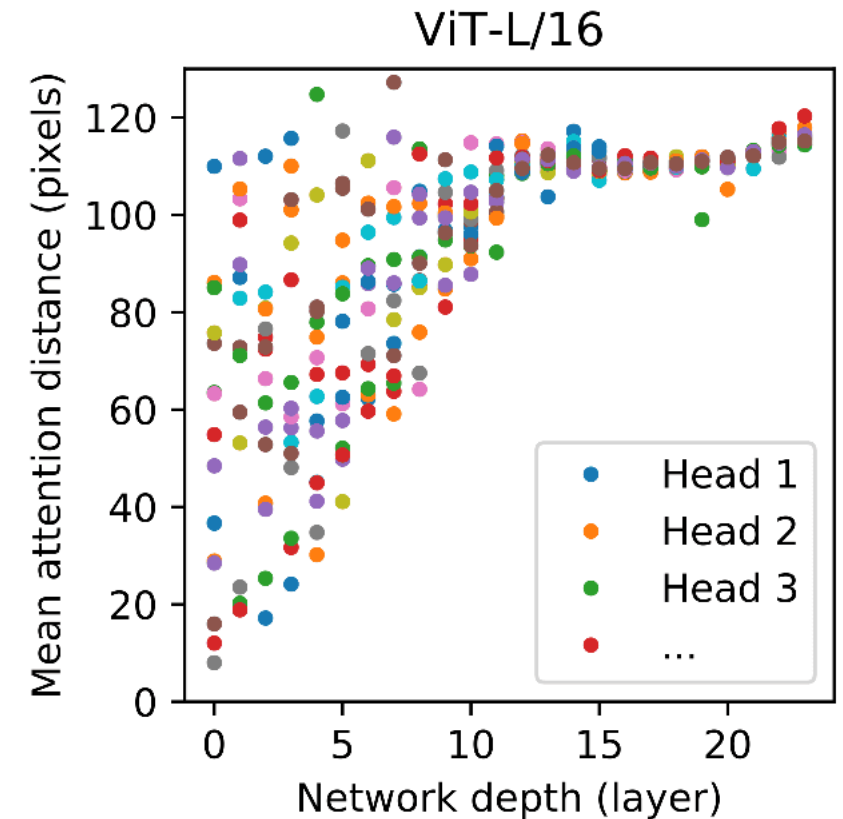
Water

Frisbee

Dog

Representación Interna de Datos

- Varias *multi-head attention* en las capas más bajas del modelo ya atienden la mayor parte de la información de la imagen (según sus pesos de atención), lo que demuestra la capacidad del mecanismo *self-attention* para integrar la información en **toda la imagen**.
- En la imagen se observa el tamaño del área de la imagen atendida por diferentes head *self-attention*
- Observamos que cada head se especializa en **diferentes patches**.





Inferencia



ETS de
Ingeniería
Informática

UNED

Inference from our Results

- El tamaño del patch en el ViT decide la longitud de la secuencia. **Un tamaño de patch menor conlleva un mayor intercambio de información durante el mecanismo de autoatención.**
 - Esto se comprueba por los mejores resultados que se obtienen utilizando un tamaño de parche menor, 4 sobre 8, en una imagen de 32x32.

Inference from our Results

- El tamaño del patch en el ViT decide la longitud de la secuencia. **Un tamaño de patch menor conlleva un mayor intercambio de información durante el mecanismo de autoatención.**
 - Esto se comprueba por los mejores resultados que se obtienen utilizando un tamaño de parche menor, 4 sobre 8, en una imagen de 32x32.
- Aumentar el número de capas de ViT debería conducir idealmente a mejores resultados, pero los **resultados del modelo de 8 capas son ligeramente mejores que los del modelo de 12 capas**, lo que puede atribuirse a los pequeños datasets utilizados para entrenar los modelos.
 - Los modelos de mayor complejidad requieren más datos para captar las características de la imagen.

Inference from our Results

- El tamaño del patch en el ViT decide la longitud de la secuencia. **Un tamaño de patch menor conlleva un mayor intercambio de información durante el mecanismo de autoatención.**
 - Esto se comprueba por los mejores resultados que se obtienen utilizando un tamaño de patch menor, 4 sobre 8, en una imagen de 32x32.
- Aumentar el número de capas de ViT debería conducir idealmente a mejores resultados, pero los **resultados del modelo de 8 capas son ligeramente mejores que los del modelo de 12 capas**, lo que puede atribuirse a los pequeños datasets utilizados para entrenar los modelos.
 - Los modelos de mayor complejidad requieren más datos para captar las características de la imagen.
- ViT preentrenado funciona mejor que métodos debido a que ha sido entrenado en **enormes conjuntos de datos** y, por tanto, ha aprendido mejor.



Disposiciones finales



ETS de
Ingeniería
Informática

UNED

Vision Transformers

Bibliografía



ViT Variants:

- [Swin Transformer: Hierarchical Vision Transformer using Shifted Windows](#)
- [Training data-efficient image transformers & distillation through attention](#)
- [Tokens-to-Token ViT: Training Vision Transformers from Scratch on ImageNet](#)
- [DeepViT: Towards Deeper Vision Transformer](#)
- [MViTv2: Improved Multiscale Vision Transformers for Classification and Detection](#)

Explicabilidad y otros:

- [LeGrad: An Explainability Method for Vision Transformers via Feature Formation Sensitivity](#)
- [How Attention works in Deep Learning: understanding the attention mechanism in sequence models](#)
- [How Transformers work in deep learning and NLP: an intuitive introduction](#)
- [How the Vision Transformer \(ViT\) works in 10 minutes: an image is worth 16x16 words](#)

¡Gracias!



Dr. Manuel Castillo-Cara

www.manuelcastillo.eu

**Departamento de Inteligencia Artificial
Escuela Técnica Superior de Ingeniería Informática
Universidad Nacional de Educación a Distancia (UNED)**