“Attention Is All You Need”: Comentario

Vamos a desgranar el paper: “Attention Is All You Need”. Es la base a partir del cual se desarrollan los transformers y la mayoría y más exitosos modelos de IA que se utilizan actualmente en la IA generativa y no generativa.

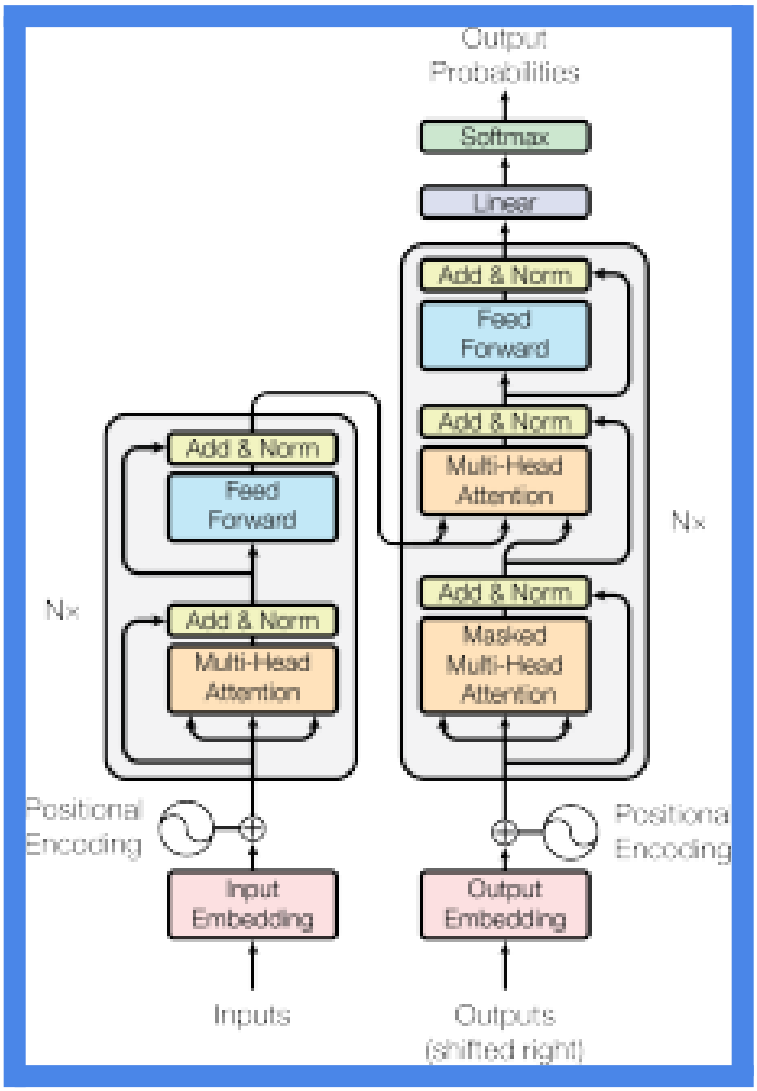


Diagrama del Modelo Transformer: Muestra la arquitectura general del Transformer, incluyendo las capas de codificador (encoder) y decodificador (decoder)

Antes de comenzar: ¿Embedding? Ejemplo de embedding de Word2vec: En el proceso de alimentar a un algoritmo de LLM (Large Language Model) debemos de procesar los datos correctamente para que se pueda entrenar con ellos.

Comencemos explicando cómo se realizaba el embedding previo al uso de transformers. Primeramente las palabras debían ser codificadas en método binario one-hot encodding. Una red neuronal (con aprendizaje no supervisado) se encargaba de reducir su dimensionalidad de tal forma que las palabras se encontraran organizadas y representadas en cluster cercanos según su parecido → representación eficiente de las palabras. La idea aquí es entrenar la matriz de peso de la capa oculta para encontrar representaciones eficientes para nuestras palabras. Esta matriz de ponderación suele denominarse matriz de incrustación. Esta representación además permite la manipulación aritmética de las palabras.

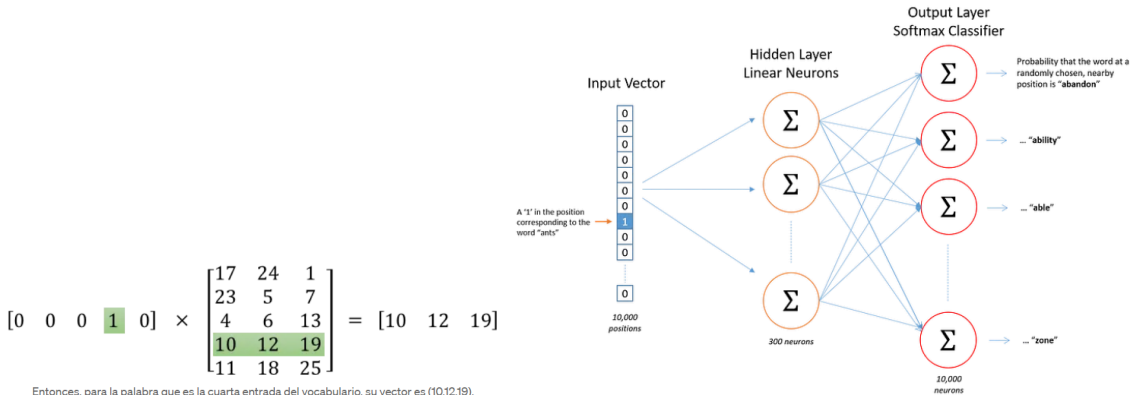


Diagrama del proceso de encodding y embedding

En el caso de los transformers como ChatGPT durante el entrenamiento se generan los embeddings contextuales. Durante el proceso de entrenamiento, el modelo aprende a generar embeddings a medida que se entrena para predecir la siguiente palabra en una secuencia de texto. Este proceso permite que el modelo capture complejas relaciones semánticas y contextuales de manera más efectiva que los métodos tradicionales como word2vec.

El transformer utilizado en procesamiento de lenguaje natural se compone principalmente de dos elementos: **El codificador y el decodificador.**

El codificador:

Pasos que sigue:

1. Cada palabra queda **codificada en vectores (tokenización y embedding (vectores de dimensiones fijas que representan características semánticas de los tokens y que posicionan tokens en espacios similares si poseen un contexto parecido)-incrustaciones;pasar de texto a vector-valor numérico).** Los embeddings se inician generalmente con pesos preentrenados y se ajustan durante el entrenamiento del modelo.

   Los vectores de embeddings tienen una dimensión fija, que suele ser un hiperparámetro del modelo (esta dimensión va a definir el número de “atributos”; Estos valores son inicialmente aleatorios y se ajustan mediante el aprendizaje profundo para capturar relaciones semánticas, de manera que palabras con semántica similar se encuentren en posición cercana en el espacio semántico multidimensional).

   Dimensiones comunes son 50, 100, 200, 300, o incluso 768 en modelos más complejos como BERT. Cuando proporcionas una oración a ChatGPT, el modelo toma los tokens de la oración y los convierte en vectores de embeddings utilizando una matriz de pesos de embeddings que fue aprendida durante el entrenamiento donde cada fila corresponde a un vector de embedding de un token del vocabulario. Si el tamaño de los embeddings es 512, por ejemplo, cada token se convierte en un vector de 512 dimensiones.

   ¿Influye el contexto de las palabras en el proceso de embedding del token?. Por supuesto. En esto consiste el proceso de autoatención que veremos más adelante

2. **Adición de vector de posición al embedding**: Esto se realiza para otorgar posición y orden a las palabras. Estos vectores siguen un patrón específico que el modelo aprende, lo que le ayuda a determinar la posición de cada palabra. Normalmente emplea método de codificación binaria empleando representación mediante ondas del vector posicional. En lugar de aprender estas codificaciones posicionales como parámetros adicionales, los autores del Transformer original (Vaswani et al., 2017) propusieron una forma determinística de generarlas usando funciones seno y coseno. Las funciones seno y coseno se utilizan para generar vectores de codificación posicional porque tienen propiedades matemáticas útiles para representar la posición de una manera continua y diferenciable.

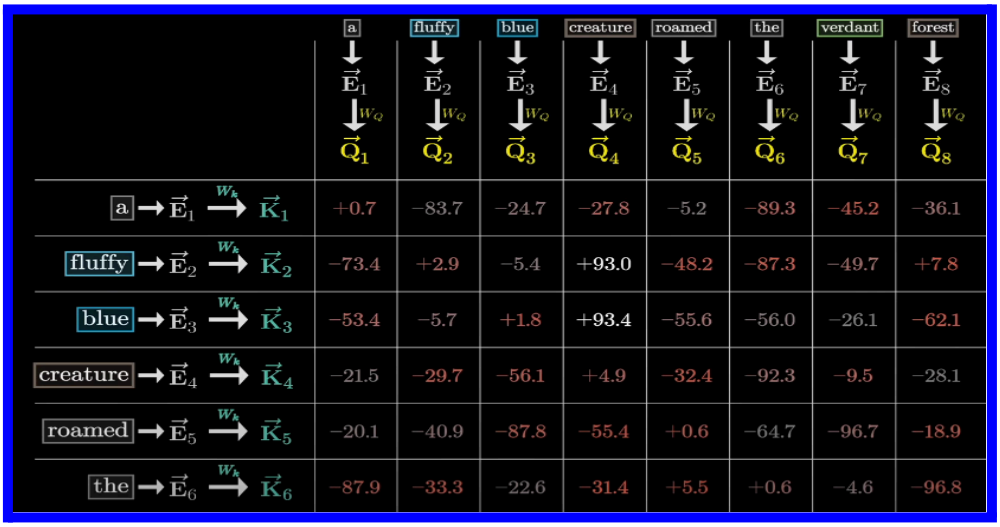
3. **Multi-Autoatención: Las palabras fluyen a la vez con los pasos que veremos a continuación. Genera un vector para cada palabra de tal forma que ha intentado ser el mejor según el resto de palabras proporcionadas.**

Mecanismo de atención.Explicación general: El proceso de self-attention comienza con la generación de tres vectores diferentes para cada token: la clave (key), la consulta (query) y el valor (value). La similitud entre la consulta de un token y las claves de todos los tokens se calcula para determinar los pesos de atención. Estos pesos indican cuánto debe "prestar atención" el token de interés a cada uno de los otros tokens en la secuencia.

Imaginando que entrenamos nuestro modelo con una frase, dicha frase se proporciona y se

procesa de varias formas. Se genera tanto un vector K para cada palabra que la posicione semánticamente en un espacio dimensional. Así como otro vector Q que pretende hallar lo que "busca" esa palabra y la posiciona igualmente en un plan multidimensional.

El producto escalar de VectorQ*Vector K de las palabras que conforman la oración nos proporciona cómo de similares son dichas palabras y la relación que tienen entre sí. Este vector se denomina *Vector Atención*. El conjunto de vectores de atención de todas las palabras se denomina matriz de atención. El producto escalar de mayor tamaño de la de la multiplicación nos indicará las palabras que se influyen más entre sí.



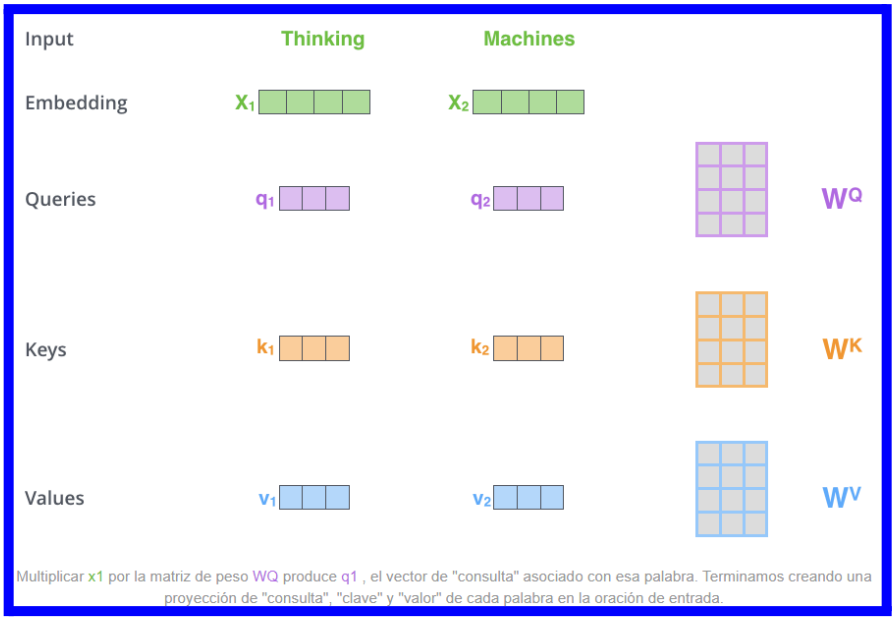
Ejemplo de diagrama de multiplicación de ambos vectores-mapa de atención

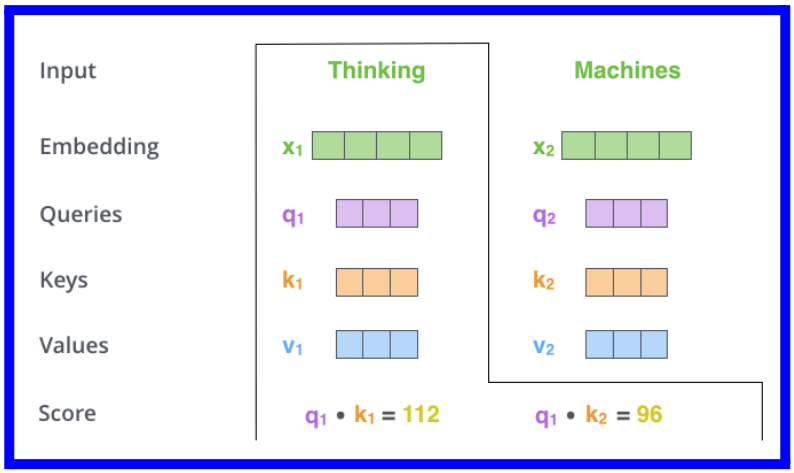
Posteriormente otra red neuronal se va a encargar de procesar el vector atención para la función específica que estamos buscando. De esta forma se obtiene lo que se denomina *vector valor.*

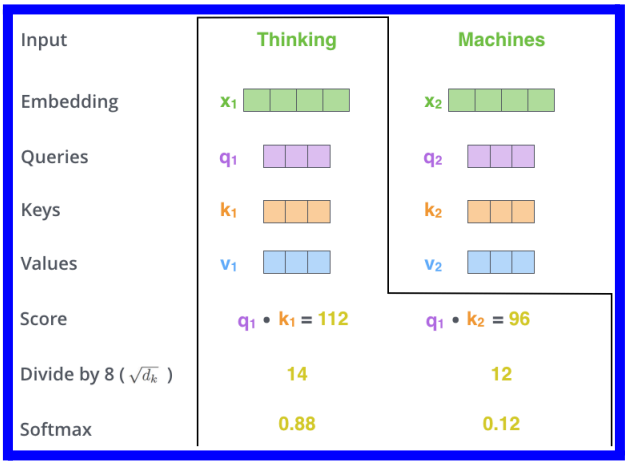
A continuación se obtiene un nuevo vector que es la multiplicación ponderada del vector de atención por el vector valor (vector V). Esto nos va indicar la influencia de cada palabra en la frase con respecto a las otras así como el "promedio de acción" para la tarea que se esté buscando. Este vector se conoce como vector output y vamos obtener tantos como palabras tenga nuestra frase.

En cada capa, el proceso de autoatención se repite, permitiendo que el modelo construya representaciones cada vez más complejas y contextualmente ricas de los tokens.

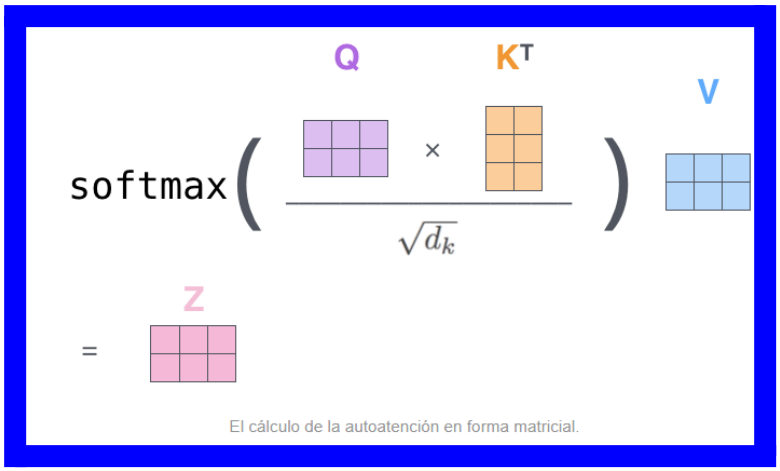
4. **Explicado de otra forma un poco más técnica,** cada token de la oración puede "atender" (prestar atención) a otros tokens de la oración. **Primer paso:** Se crean 3 vectores a partir de cado uno de los vectores de entrada → Q (consulta), K (Key) y V (valor) que se obtienen multiplicando el vector de entrada por las matrices Wq, WK y Wv, las cuales se van perfeccionando durante el entrenamiento (serían los pesos). **Segundo paso:** Calculamos puntuación: Consiste en aplicar el producto escalar del vector consulta por el vector clave de cada uno de las palabras. **Tercer paso:** División de la puntuación por la raíz cuadrada de la dimensión del vector clave. **Cuarto paso:** Pasamos el resultado por la función softmax para la normalización (positivas y con suma global a 1). **Quinto paso:** Multiplica el resultado anterior por el vector valor de dicha palabra (ahoga palabras irrelevantes). **Sexto paso:** Suma los vectores de los valores ponderados PRODUCIENDO ASÍ LA SALIDA DE LA CAPA DE AUTOATENCIÓN EN ESTA POSICIÓN (PRIMERA PALABRA).







Las tres imágenes anteriores representan los pasos del apartado 4



Fórmula matemática que resume los pasos del 2 al 6

5. **Concatenación: Proceso por el cual "juntamos" las matrices de cada cabeza de atención en una única matriz (compuesta por cada vector por cada palabra) para luego multiplicarla por una matriz de pesos adicionales (Wo).**

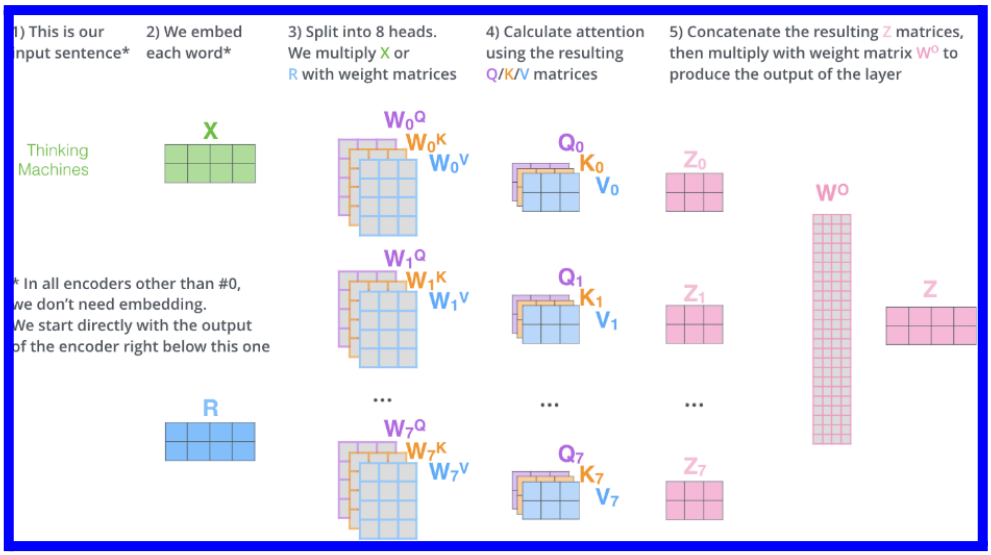


Imagen que representan el proceso de concatenación

- Paso adicional: Antes y después de la red neuronal feed forward se realiza un proceso de adicción (conexión residual) + normalización. **Esto se realiza porque mejora la estabilidad del entrenamiento y facilita la propagación del gradiente**: 1. Estabilidad mediante Normalización: Se usa la normalización, específicamente la normalización de Batch (Batch Normalization) o la normalización de Capa (Layer Normalization), para estabilizar y acelerar el proceso de entrenamiento. Esto se logra al mantener la media y la varianza de las activaciones de las capas dentro de un rango específico, lo que ayuda a mitigar problemas

como el desvanecimiento y la explosión de los gradientes. En los transformers, generalmente se utiliza la Normalización de Capa. 2. Propagación del gradiente: Las conexiones residuales, que son una forma de suma, ayudan a que los gradientes se propaguen más efectivamente a través de la red durante el proceso de retropropagación. Esto es crucial para entrenar redes profundas como los transformers, ya que facilita que la información fluya hacia atrás a través de las capas, reduciendo así la posibilidad de que los gradientes se desvanezcan o exploten.

Ejemplo: si la entrada a una capa es x *y la salida de la transformación (por ejemplo, atención o feed-forward) es F(x) entonces la salida final de la capa es y =x + F(x). La normalización sería:*

$$\text{LayerNorm}(x) = \frac{x-\mu}{\sigma} \cdot \gamma + \beta$$

donde μ y σ son la media y la desviación estándar de las activaciones x, y γ y β son parámetros aprendibles que escalan y desplazan la normalización.

¿Qué quiere decir que los gradientes se desvanezcan o exploten?*1* ***Desvanecimiento:*** Ocurre cuando los gradientes (las derivadas parciales de la función de pérdida con respecto a los parámetros de la red) se vuelven extremadamente pequeños a medida que se retropropagan a través de las capas de la red. Esto puede llevar a que los parámetros de las capas más profundas se actualicen muy poco, ralentizando significativamente el aprendizaje o incluso deteniéndolo por completo. 2. ***Explosión de gradientes:*** La explosión de gradientes es el fenómeno opuesto, donde los gradientes se vuelven muy grandes durante la retropropagación. Esto puede hacer que los parámetros de la red se actualicen con valores extremadamente grandes, llevando a oscilaciones erráticas en el valor de la función de pérdida, y potencialmente resultando en la inestabilidad del entrenamiento

Las conexiones residuales ayudan a mitigar ambos problemas proporcionando una ruta directa para el flujo del gradiente. Al añadir la entrada de una capa directamente a su salida (después de la transformación), se garantiza que, como mínimo, la identidad puede ser aprendida. Esto significa que incluso si la transformación aplicada (por ejemplo, una capa de atención o una capa feed-forward) produce gradientes pequeños o grandes, la ruta directa de la entrada permite que el gradiente sea pasado sin cambios significativos a través de la red. Esto facilita el aprendizaje de redes muy profundas.

Al normalizar las activaciones antes de pasarlas a la siguiente capa, se evita que las activaciones se vuelvan demasiado grandes o demasiado pequeñas, lo que a su vez previene la explosión o el desvanecimiento de los gradientes.

El decodificador:

Entra en acción una vez que el los codificadores se entrenaron.

Pasos que sigue:

Paso 1: La salida del codificador superior se transforma en un conjunto de vectores de atención K y V. Estos deben ser utilizados por cada decodificador en su capa de "atención ", para posteriormente aplicar la función final Lineal+ Softmax (que explicaré más adelante). Con lo que hay que quedarse en este primer paso es que antes de que el decodificador procese su entrada ya ha procesado los vectores de atención K y V del último codificador para que de esta manera se pueda centrar mejor en los lugares apropiados de la secuencia de entrada.

Resto de pasos: Los siguientes pasos repiten el proceso hasta que se alcanza un símbolo especial que indica que el descodificador del transformer ha completado su salida. La salida

de cada paso se alimenta al descodificador inferior en el siguiente paso temporal, y los descodificadores propagan sus resultados de decodificación, al igual que lo hicieron los codificadores. Y al igual que hicimos con las entradas del codificador, incrustamos y agregamos codificación posicional a esas entradas del descodificador para indicar la posición de cada palabra.

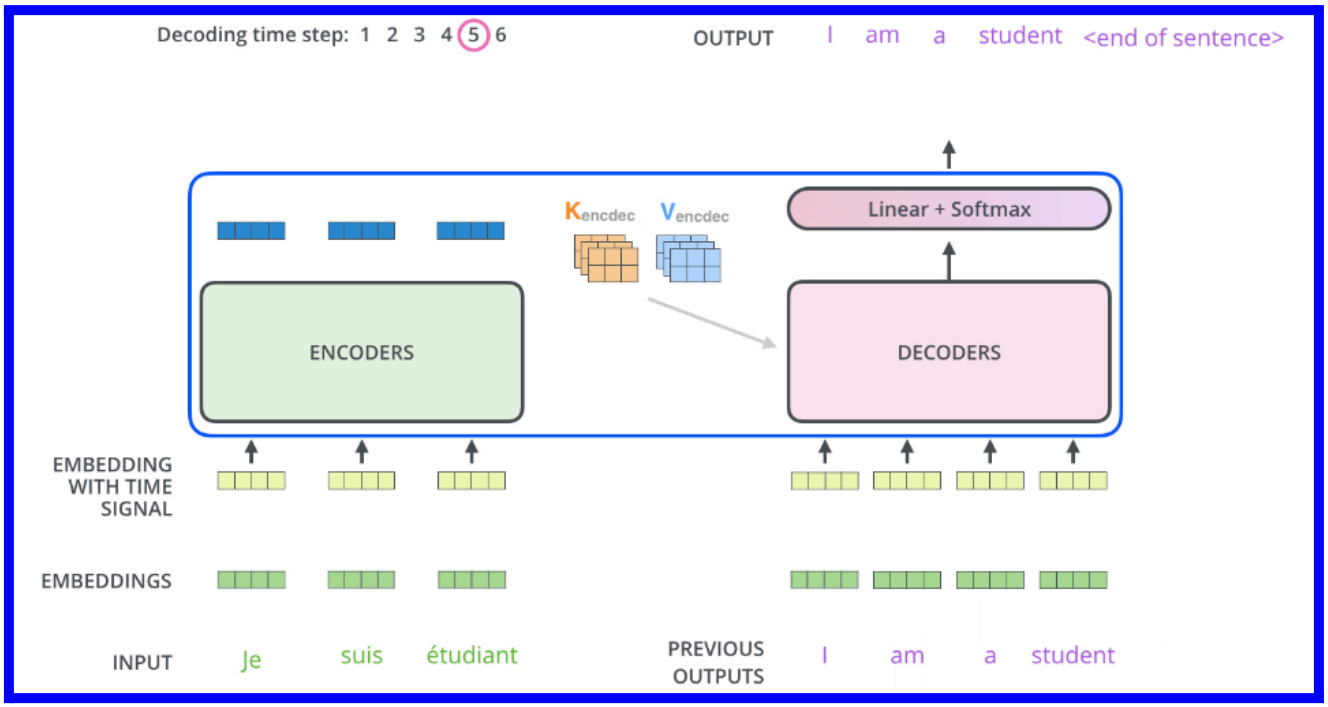
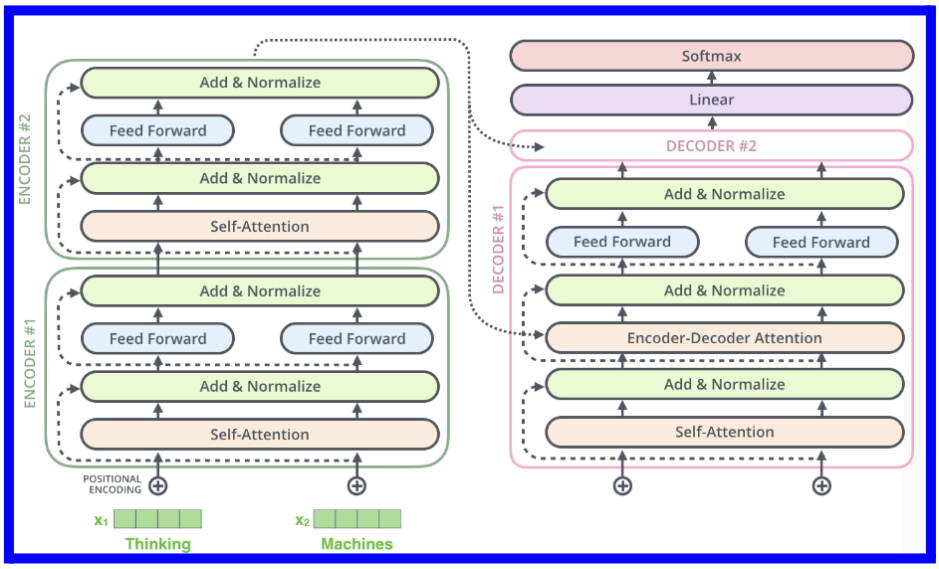


Diagrama que representa momento por el cual las matrices K y V aprendidas por el codificador permiten entrenar y enfocar el decodificador durante su entrenamiento

¿Qué es la capa final lineal y softmax?: Es una red neuronal que proyecta a un vector mucho más grande el vector final producido por el último decodificador. Esto es lo que se conoce como vector logit. Supongamos que nuestro modelo conoce 10.000 palabras únicas en inglés (el “vocabulario de salida” de nuestro modelo) que ha aprendido de su conjunto de datos de entrenamiento. Esto haría que los ***logits*** tuvieran un vector de 10.000 celdas de ancho, correspondiendo cada celda a la puntuación de una palabra única. Así es como interpretamos la salida del modelo seguida de la capa Lineal. Luego, la capa softmax convierte esas puntuaciones en probabilidades (todas positivas, todas suman 1,0). Se elige la celda con la mayor probabilidad y la palabra asociada a ella se produce como salida para este paso de tiempo



Ejemplo de transformer con 2 codificadores y 2 decodificadores

¿Cómo se produce el entrenamiento en un transformer?: La salida del descodificador se compara con la secuencia de salida esperada (ground truth) utilizando una función de pérdida, como la entropía cruzada, para calcular qué tan lejos están las predicciones de las respuestas correctas. Posteriormente: Retropropagación: Se calcula el gradiente de la función de pérdida con respecto a cada peso en la red utilizando el algoritmo de retropropagación. Actualización de Pesos: Los pesos del modelo se actualizan usando un optimizador (como Adam o SGD) que ajusta los pesos en la dirección que minimiza la pérdida. *¿Cuáles son los pesos que modifica?* Pesos de las Capas de Embedding: Estos pesos determinan cómo los tokens se representan como vectores en el espacio de embedding. Pesos de las Capas de Atención: Incluyen matrices de proyección para las consultas (queries), claves (keys) y valores (values) en los mecanismos de auto-atención y atención cruzada. Pesos de las Capas Feed-Forward: Estos son los pesos de las capas totalmente conectadas que siguen a los mecanismos de atención en cada bloque del codificador y del descodificador. Pesos de la Normalización de Capa: Parámetros de escala y desplazamiento aprendidos en las capas de normalización.

Efecto de los Pesos en Futuras Entradas

Los pesos aprendidos en un transformer afectan cómo se procesan las futuras entradas de las siguientes maneras:

- **Representación de Embeddings**: Los embeddings de entrada determinan cómo los tokens se representan en el espacio de características. Pesos bien ajustados permiten representaciones más precisas y relevantes para la tarea.
- **Atención**: Los pesos en los mecanismos de atención determinan qué partes de la entrada (o de la salida previamente generada) se consideran más importantes para la tarea actual. Esto permite al modelo enfocar su atención en los elementos más relevantes.
- **Transformaciones Feed-Forward**: Las capas feed-forward transforman las representaciones de los tokens de manera no lineal, permitiendo que el modelo capture relaciones complejas y patrones en los datos.
- **Normalización**: La normalización estabiliza las activaciones y facilita la propagación de gradientes, mejorando la eficiencia del entrenamiento y la calidad de las predicciones.

Modificación de la capa de embedding: Si *Wembed* es la matriz de pesos de embedding, su actualización se realiza como: $\mathbf{W}_{embed} \leftarrow \mathbf{W}_{embed} - \eta \frac{\partial L}{\partial \mathbf{W}_{embed}}$. Donde η es la tasa de aprendizaje y L es la función de pérdida

Modificación de la capa de atención: Se aplican a cada uno de los vectores valor, key… Ejemplo $\mathbf{W}^Q \leftarrow \mathbf{W}^Q - \eta \frac{\partial L}{\partial \mathbf{W}^Q}$

Modificación del peso de la capa Feed-Forward: Como ya conocemos su entrenamiento se desarrolla de forma similar a otras redes neuronales clásicas.

Vision Transformer (ViTs): *Paper: AN IMAGE IS WORTH 16X16 WORDS:TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE*

- Inconvenientes de las redes neuronales convolucionales: No responden bien a variaciones en la escala y rotaciones de las imágenes, IGNORA la relación existente entre las partes de la imagen, PÉRDIDA DE INFORMACIÓN en el proceso de convolución.

Los vision transformer permiten establecer las relaciones entre los diferentes componentes de una imagen: La estrategia habitual a seguir es la división de las imágenes en partes pequeñas denominadas “patches” (preferible a píxeles tanto por consumo como por resultados obtenidos). En problemas de clasificación de la imagen estos patches alimentan en orden al transformer para que posteriormente un multilayer perceptron se encargue del proceso de clasificación de la imagen. Estos patches deben de sufrir inicialmente un proceso de aplanamiento para luego reducir su dimensionalidad mediante embedding lineal; para mejorar rendimiento.

Aplanamiento del patch

Cada patch=cuadrante de división de la imagen normalmente se va considerar como una composición de tokens de entrada (el token más pequeño de dicho patch podría ser considerado de tamaño igual a un píxel). Los patches para ser introducidos en nuestro visual transformer

deben poseer también una CODIFICACIÓN POSICIONAL (proceso muy importante en cualquier transformer). Esto se adiciona tras el aplanamiento y embedding lineal de los vectores.

Por lo general y como diferencia con respecto a los transformers utilizados en IA generativa de procesamiento de lenguaje natural, los ViTs no poseen el componente de decodificador ya que no se necesita que se produzca una secuencia de outputs.

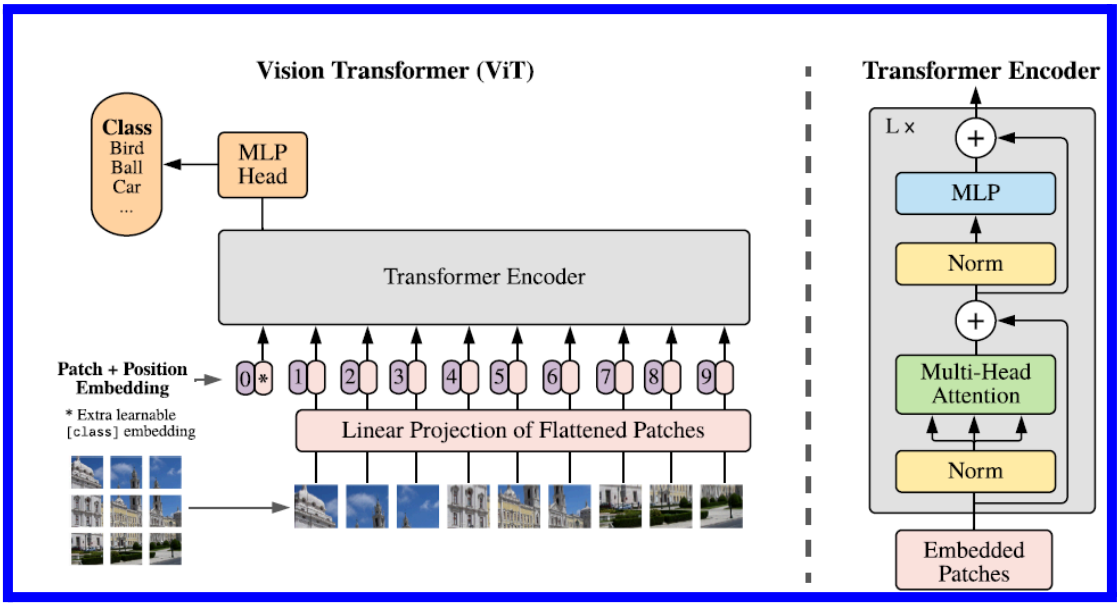


Diagrama de un ViTs. Dividimos una imagen en parches de tamaño fijo, incrustamos linealmente cada uno de ellos, añadimos incrustaciones de posición y alimentamos la secuencia resultante de vectores a un codificador Transformer estándar. Para realizar la clasificación, utilizamos el enfoque estándar de agregar un "token de clasificación" adicional y aprendible a la secuencia. La ilustración del codificador Transformer fue inspirada por Vaswani et al. (2017)

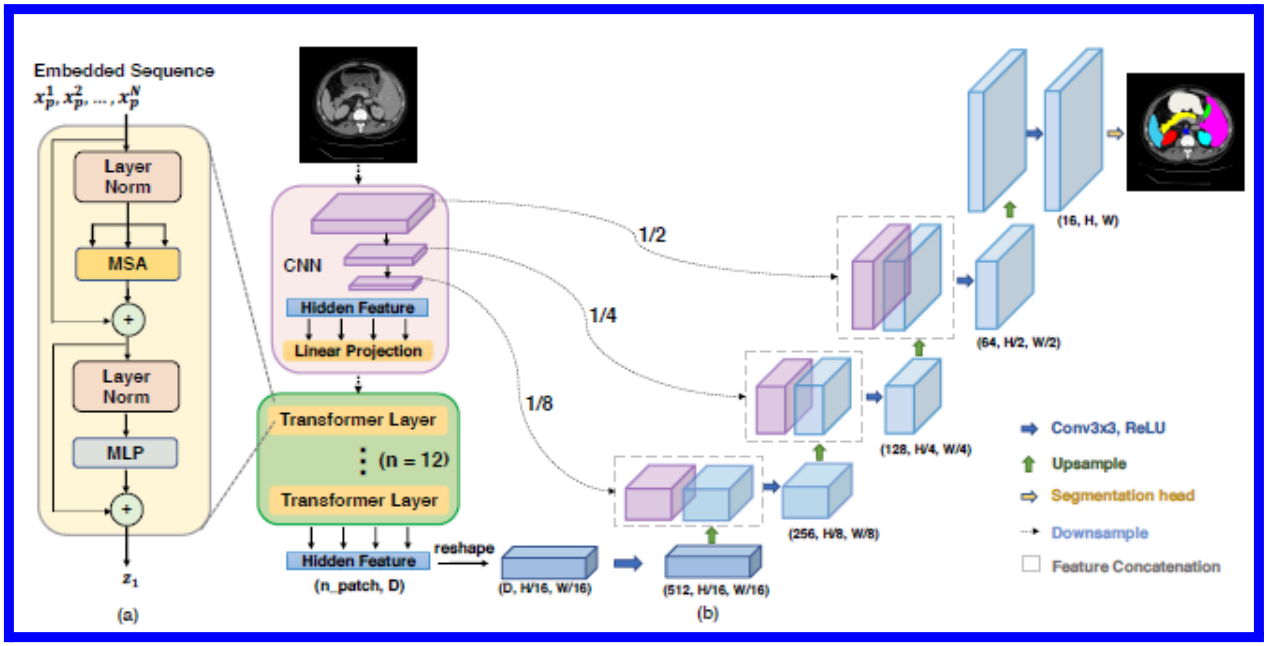
Existen también arquitecturas derivadas del visual transformer como es el El Swin Transformer (Shifted Window Transformer)(**paper:"*Swin* Transformer: Hierarchical Vision Transformer using Shifted Windows**"). La principal innovación del Swin Transformer es el uso de ventanas desplazadas para calcular la atención propia. En lugar de aplicar la atención a toda la imagen, se aplica dentro de pequeñas ventanas no superpuestas, lo que reduce significativamente la complejidad computacional. El modelo construye representaciones jerárquicas fusionando parches de imagen en capas más profundas concatenando características de grupos de parches vecinos y aplicando una capa lineal para reducir la dimensionalidad. Esto permite la creación de mapas de características de múltiples resoluciones, similares a las redes neuronales convolucionales (CNNs).

Existen modelos entrenados disponibles de visual transformers qué podemos utilizar mediante el proceso de transferencia de aprendizaje (coger un modelo ya pre entrenado y mediante un pequeño reentrenamiento con unas pocas iteraciones utilizarlo para una tarea para la cual no estaba diseñado).

¿Podemos compaginar las ventajas de ambos modelos?: Si, son las arquitecturas híbridas que combinan tanto visual transformer como redes neuronales convolucionales.

Las estrategias más habituales hoy en día son aplicar primero las redes neuronales convolucionales para extraer características de bajo nivel (mapas de características) de la imagen para posteriormente aplicar un vision transformer para establecer las dependencias entre las diferentes partes de la imagen. No obstante se están utilizando otros abordajes como son las redes neuronales ConvMixers que combina conceptos de convoluciones profundas con mecanismos de mezcla (mixing) de características. La parte de "mixer" en ConvMixers se refiere a cómo las características extraídas de diferentes partes de la imagen se mezclan y combinan. La arquitectura de ConvMixer consiste en una serie de bloques, cada uno compuesto por una convolución seguida de una operación de mezcla. Estos bloques se apilan en profundidad, permitiendo una extracción de características jerárquicas. ConvMixers están diseñadas para ser más simples y eficientes en términos de parámetros en comparación con otras arquitecturas avanzadas de visión por computadora, como los transformers vision.

Caso especial para el proceso de segmentación. Combinación de modelos. Redes basadas en arquitectura Trans-Unet: Se trata de una combinación de redes neuronales convolucionales junto con transformers CNN-Transformer. ***Paper: "TransUNet: Transformers Make Strong Encoders for Medical Image Segmentation"***
→ paper que cambia paradigma de segmentación de imágenes médicas.



Vista general de arquitectura de TransUnet