ATENCIÓN ES TODO LO QUE NECESITAS

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

ABSTRACT

Los modelos de transducción de secuencias dominantes se basan en redes neuronales complejas, recurrentes o convolucionales, que incluyen un codificador y un decodificador. Los modelos de mejor rendimiento también conectan el codificador y el decodificador mediante un mecanismo de atención. Proponemos una nueva arquitectura de red simple, el Transformer, basada únicamente en mecanismos de atención, prescindiendo por completo de la recurrencia y las convoluciones. Los experimentos en dos tareas de traducción automática muestran que estos modelos son superiores en calidad, al mismo tiempo que son más paralelizables y requieren mucho menos tiempo para entrenarlos. Nuestro modelo logra 28,4 BLEU en la tarea de traducción del inglés al alemán del WMT 2014, mejorando los mejores resultados existentes, incluidos los conjuntos, en más de 2 BLEU. En la tarea de traducción del inglés al francés de WMT 2014, nuestro modelo establece una nueva puntuación BLEU de última generación de un solo modelo de 41,8 después de un entrenamiento durante 3,5 días en ocho GPU, una pequeña fracción de los costos de entrenamiento de los mejores. modelos de la literatura. Mostramos que Transformer se generaliza bien a otras tareas al aplicarlo con éxito al análisis de distritos electorales en inglés con datos de entrenamiento grandes y limitados.

*Igual contribución. El orden de listado es aleatorio. Jakob propuso reemplazar los RNN con autoatención y comenzó el esfuerzo de evaluar esta idea. Ashish, con Illia, diseñó e implementó los primeros modelos de Transformer y ha participado de manera crucial en todos los aspectos de este trabajo. Noam propuso la atención de producto escalado, la atención de múltiples cabezas y la representación de posición sin parámetros y se convirtió en la otra persona involucrada en casi todos los detalles. Niki diseñó, implementó, ajustó y evaluó innumerables variantes de modelos en nuestro código base original y tensor2tensor. Llion también experimentó con nuevas variantes de modelos, fue responsable de nuestra base de código inicial y de inferencias y visualizaciones eficientes. Lukasz y Aidan pasaron innumerables días diseñando varias partes e implementando tensor2tensor, reemplazando nuestro código base anterior, mejorando enormemente los resultados y acelerando enormemente nuestra investigación.

†Trabajo realizado mientras estaba en Google Brain.

‡Trabajo realizado mientras estaba en Google Research.

1 Introducción

Las redes neuronales recurrentes, la memoria a largo plazo [13] y las redes neuronales recurrentes cerradas [7] en particular, se han establecido firmemente como enfoques de última generación en el modelado de secuencias y problemas de transducción, como el modelado del lenguaje y la traducción automática [35, 2, 5]. Desde entonces, numerosos esfuerzos han seguido ampliando los límites de los modelos de lenguaje recurrentes y las arquitecturas de codificador-decodificador [38, 24, 15]. Los modelos recurrentes suelen factorizar el cálculo según las posiciones de los símbolos de las secuencias de entrada y salida. Al alinear las posiciones con los pasos en el tiempo de cálculo, generan una secuencia de estados ocultos h_t, en función del estado oculto anterior h_{t-1} y la entrada para la posición t. Esta naturaleza inherentemente secuencial impide la paralelización dentro de los ejemplos de entrenamiento, lo que se vuelve crítico en secuencias de mayor longitud, ya que las limitaciones de memoria limitan el procesamiento por lotes entre ejemplos. Trabajos recientes han logrado mejoras significativas en la eficiencia computacional a través de trucos de factorización [21] y cálculo condicional [32], al tiempo que mejoran el rendimiento del modelo en el caso de este último. Sin embargo, persiste la limitación fundamental del cálculo secuencial. Los mecanismos de atención se han convertido en una parte integral del modelado de secuencias convincentes y los modelos de transducción en diversas tareas, permitiendo modelar dependencias sin tener en cuenta su distancia en las secuencias de entrada o salida [2, 19]. Sin embargo, en todos los casos, excepto en unos pocos [27], dichos mecanismos de atención se utilizan junto con una red recurrente. En este trabajo proponemos el Transformer, una arquitectura modelo que evita la recurrencia y, en cambio, se basa completamente en un mecanismo de atención para dibujar dependencias globales entre entrada y salida. El Transformer permite una paralelización significativamente mayor y puede alcanzar un nuevo estado del arte en calidad de traducción después de haber sido entrenado durante tan solo doce horas en ocho GPU P100.

2 Background

El objetivo de reducir el cálculo secuencial también forma la base de Extended Neural GPU [16], ByteNet [18] y ConvS2S [9], todos los cuales utilizan redes neuronales convolucionales como bloque de construcción básico, calculando representaciones ocultas en paralelo para todas las entradas y salidas. posiciones de salida. En estos modelos, el número de operaciones necesarias para relacionar señales de dos posiciones de entrada o salida arbitrarias crece en la distancia entre posiciones, linealmente para ConvS2S y logarítmicamente para ByteNet. Esto hace que sea más difícil aprender las dependencias entre posiciones distantes [12]. En el Transformador, esto se reduce a un número constante de operaciones, aunque a costa de una resolución efectiva reducida debido al promedio de posiciones ponderadas por atención, un efecto que contrarrestamos con Atención de múltiples cabezales como se describe en la sección 3.2.

La autoatención, a veces llamada intraatención, es un mecanismo de atención que relaciona diferentes posiciones de una sola secuencia para calcular una representación de la secuencia. La atención personal se ha utilizado con éxito en una variedad de tareas que incluyen comprensión lectora, resúmenes abstractivos, vinculación textual y aprendizaje de representaciones de oraciones independientes de la tarea [4, 27, 28, 22].

Las redes de memoria de un extremo a otro se basan en un mecanismo de atención recurrente en lugar de una recurrencia alineada con secuencias y se ha demostrado que funcionan bien en tareas de modelado de lenguaje y respuesta a preguntas en lenguaje simple [34].

Sin embargo, hasta donde sabemos, Transformer es el primer modelo de transducción que se basa completamente en la autoatención para calcular representaciones de su entrada y salida sin utilizar RNN alineados en secuencia o convolución. En las siguientes secciones, describiremos el Transformer, motivaremos la atención personal y discutiremos sus ventajas sobre modelos como [17, 18] y [9].

3 Arquitectura modelo

La mayoría de los modelos competitivos de transducción de secuencias neuronales tienen una estructura codificador-decodificador [5, 2, 35]. Aquí, el codificador asigna una secuencia de entrada de representaciones de símbolos $(x_1, ..., x_n)$ a una secuencia de representaciones continuas $z = (z_1, ..., z_n)$. Dado z, el decodificador genera una secuencia de salida $(y_1, ..., y_m)$ de símbolos, un elemento a la vez. En cada paso, el modelo es autorregresivo [10], consumiendo los símbolos generados previamente como entrada adicional al generar el siguiente.

El Transformer sigue esta arquitectura general utilizando autoatención apilada y capas puntuales y completamente conectadas tanto para el codificador como para el decodificador, como se muestra en las mitades izquierda y derecha de la Figura 1, respectivamente.

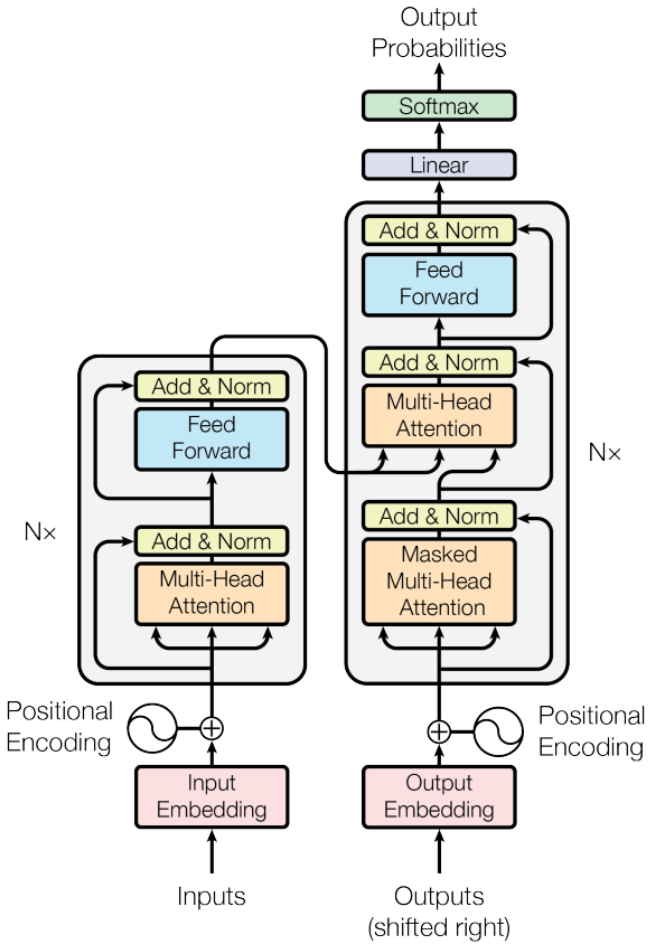

Figure 1: El Transformador - modelo arquitectura.

3.1 Pilas de codificadores y decodificadores

Codificador: El codificador está compuesto por una pila de $N = 6$ capas idénticas. Cada capa tiene dos subcapas. El primero es un mecanismo de autoatención de múltiples cabezales y el segundo es una red de alimentación directa simple, posicional y completamente conectada. Empleamos una conexión residual [11] alrededor de cada una de las dos subcapas, seguida de una normalización de capa [1]. Es decir, la salida de cada subcapa es $LayerNorm(x + Sublayer(x))$, donde $Sublayer(x)$ es la función implementada por la propia subcapa. Para facilitar estas conexiones residuales, todas las subcapas del modelo, así como las capas de incrustación, producen resultados de dimensión $d_{model} = 512$.

Decodificador: El decodificador también se compone de una pila de $N = 6$ capas idénticas. Además de las dos subcapas en cada capa del codificador, el decodificador inserta una tercera subcapa, que realiza atención de múltiples cabezales sobre la salida de la pila del codificador. De manera similar al codificador, empleamos conexiones residuales alrededor de cada una de las subcapas, seguidas de la normalización de capas. También modificamos la subcapa de autoatención en la pila de decodificadores para evitar que las posiciones atiendan a posiciones posteriores. Este enmascaramiento, combinado con el hecho de que las incorporaciones de salida están compensadas en una posición, garantiza que las predicciones para la posición i puedan depender sólo de las salidas conocidas en posiciones menores que i.

3.2 Atención

Una función de atención se puede describir como el mapeo de una consulta y un conjunto de pares clave-valor a una salida, donde la consulta, las claves, los valores y la salida son todos vectores. La salida se calcula como una suma

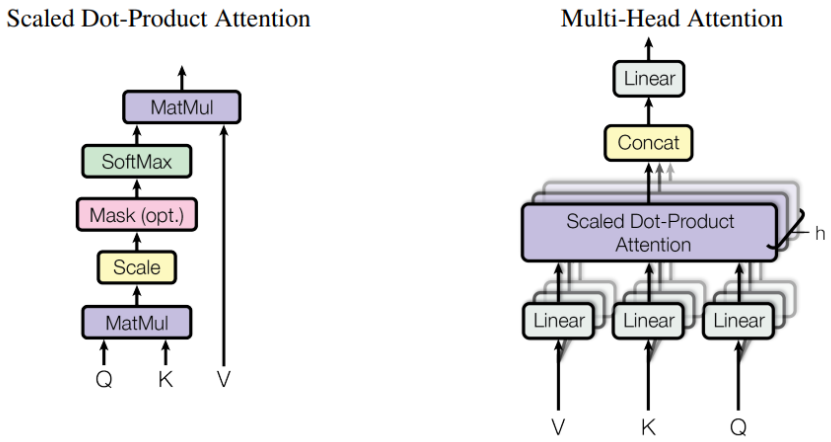

Figure 2: (izquierda) Atención de producto escalado. (derecha) La atención multicabezal consta de varias capas de atención que se ejecutan en paralelo.

ponderada de los valores, donde el peso asignado a cada valor se calcula mediante una función de compatibilidad de la consulta con la clave correspondiente.

3.2.1 Atención del producto escalado

Llamamos nuestra atención particular "Atención de producto escalado" (figura 2). La entrada consta de consultas y claves de dimensión d_k y valores de dimensión d_v. Calculamos los productos escalares de la consulta con todas las claves, dividimos cada una por $\sqrt{d_k}$ y aplicamos una función softmax para obtener los pesos de los valores. En la práctica, calculamos la función de atención en un conjunto de consultas simultáneamente, empaquetadas en una matriz Q. Las claves y los valores también se empaquetan en matrices K y V. Calculamos la matriz de resultados como:

$$Atencion(Q, K, V) = sofmax(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

Las dos funciones de atención más utilizadas son la atención aditiva [2] y la atención de producto escalar (multiplicativa). La atención del producto escalar es idéntica a nuestro algoritmo, excepto por el factor de escala de $\frac{1}{\sqrt{d_k}}$. La atención aditiva calcula la función de compatibilidad utilizando una red de avance con una única capa oculta. Si bien los dos son similares en complejidad teórica, la atención del producto punto es mucho más rápida y eficiente en el espacio en la práctica, ya que se puede implementar utilizando un código de multiplicación de matrices altamente optimizado.

Mientras que para valores pequeños de d_k los dos mecanismos funcionan de manera similar, la atención aditiva supera la atención del producto escalar sin escalar para valores más grandes de d_k [3]. Sospechamos que para valores grandes de d_k, los productos escalares crecen en magnitud, empujando la función softmax a regiones donde tiene gradientes extremadamente pequeños[4]. Para contrarrestar este efecto, escalamos los productos escalares en $\frac{1}{\sqrt{d_k}}$.

3.2.2 Atención de múltiples cabezales

En lugar de realizar una única función de atención con claves, valores y consultas $d_{model} - dimensional$, encontramos beneficioso proyectar linealmente las consultas, claves y valores h veces con diferentes proyecciones lineales aprendidas a dimensiones d_k, d_k y d_v, respectivamente. En cada una de estas versiones proyectadas de consultas, claves y valores, realizamos la función de atención en paralelo, generando valores de salida $d_v - dimensional$. Estos se concatenan y una vez más se proyectan, dando como resultado los valores finales, como se muestra en la Figura 2.

La atención de múltiples cabezales permite que el modelo atienda de manera conjunta información de diferentes sub espacios de representación en diferentes posiciones. Con una sola cabeza de atención, el promedio inhibe esto.

[4]Para ilustrar por qué los productos escalares aumentan de tamaño, supongamos que los componentes de q y k son variables aleatorias independientes con media 0 y varianza 1. Entonces su producto escalar, $q \cdot k = \sum_{i=1}^{dk} q_i k_i$, tiene media 0 y varianza dk.

$$MultiHead(Q, K, V) = Concat(head1, ..., head_h)W^O$$
$$where\ head_i = Attention(QW_i^Q, KW_i^K, KW_i^V)$$

Donde las proyecciones son matrices de parámetros $W_i^Q \epsilon \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \epsilon \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \epsilon \mathbb{R}^{d_{model} \times d_v}$ y $W^O \epsilon \mathbb{R}^{hd_v \times d_{model}}$.

En este trabajo empleamos $h = 8$ capas de atención paralelas, o cabezas. Para cada uno de estos usamos $d_k = d_v = d_model/h = 64$. Debido a la dimensión reducida de cada cabeza, el costo computacional total es similar al de la atención de una sola cabeza con dimensionalidad completa.

3.2.3 Aplicaciones de la Atención en nuestro Modelo

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

- En las capas de "atención codificador-decodificador", las consultas provienen de la capa decodificadora anterior y las claves y valores de memoria provienen de la salida del codificador. Esto permite que cada posición en el decodificador atienda todas las posiciones en la secuencia de entrada. Esto imita los mecanismos típicos de atención codificador-decodificador en modelos secuencia a secuencia como [38, 2, 9].
- El codificador contiene capas de autoatención. En una capa de autoatención, todas las claves, valores y consultas provienen del mismo lugar, en este caso, la salida de la capa anterior en el codificador. Cada posición en el codificador puede atender a todas las posiciones en la capa anterior del codificador.
- De manera similar, las capas de autoatención en el decodificador permiten que cada posición en el decodificador atienda todas las posiciones en el decodificador hasta esa posición inclusive. Necesitamos evitar el flujo de información hacia la izquierda en el decodificador para preservar la propiedad autorregresiva. Implementamos esto dentro de la atención del producto escalado enmascarando (estableciendo en ∞) todos los valores en la entrada del softmax que corresponden a conexiones ilegales. Ver Figura 2.

3.3 Redes de retroalimentación por posición

Además de las subcapas de atención, cada una de las capas de nuestro codificador y decodificador contiene una red de alimentación directa completamente conectada, que se aplica a cada posición de forma separada e idéntica. Consiste en dos transformaciones lineales con una activación ReLU en el medio.

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

Si bien las transformaciones lineales son las mismas en diferentes posiciones, utilizan diferentes parámetros de una capa a otra. Otra forma de describir esto es como dos convoluciones con tamaño de núcleo 1. La dimensionalidad de entrada y salida es $d_{model} = 512$, y la capa interna tiene dimensionalidad $d_{ff} = 2048$.

3.4 Incrustaciones y Softmax

De manera similar a otros modelos de transducción de secuencias, utilizamos incrustaciones aprendidas para convertir los tokens de entrada y los tokens de salida en vectores de dimensión dmodel. También utilizamos la transformación lineal aprendida habitual y la función softmax para convertir la salida del decodificador en probabilidades predichas del siguiente token. En nuestro modelo, compartimos la misma matriz de peso entre las dos capas de incrustación y la transformación lineal previa a softmax, similar a [30]. En las capas de incrustación, multiplicamos esos pesos por $\sqrt{d_{model}}$.

Abandono residual Aplicamos abandono [33] a la salida de cada subcapa, antes de agregarlo a la entrada de la subcapa y normalizarlo. Además, aplicamos abandono a las sumas de las incrustaciones y las codificaciones posicionales en las pilas de codificador y decodificador. Para el modelo base, utilizamos una tasa de $P_{drop} = 0,1$.

3.5 Codificación posicional

Dado que nuestro modelo no contiene recurrencia ni convolución, para que el modelo utilice el orden de la secuencia, debemos inyectar alguna información sobre la posición relativa o absoluta de los tokens en la secuencia. Con este fin, agregamos "codificaciones posicionales" a las incrustaciones de entrada en la parte inferior de las pilas de codificadores

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	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Table 1: Longitudes de ruta máximas, complejidad por capa y número mínimo de operaciones secuenciales para diferentes tipos de capas. n es la longitud de la secuencia, d es la dimensión de representación, k es el tamaño del núcleo de las convoluciones y r el tamaño de la vecindad en autoatención restringida.

y decodificadores. Las codificaciones posicionales tienen la misma dimensión d_{model} que las incrustaciones, de modo que las dos se pueden sumar. Hay muchas opciones de codificaciones posicionales, aprendidas y fijas [9].

En este trabajo utilizamos funciones seno y coseno de diferentes frecuencias:

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

donde pos es la posición y i es la dimensión. Es decir, cada dimensión de la codificación posicional corresponde a una sinusoide. Las longitudes de onda forman una progresión geométrica desde 2π hasta 100002π. Elegimos esta función porque planteamos la hipótesis de que permitiría que el modelo aprendiera fácilmente a atender por posiciones relativas, ya que para cualquier desplazamiento fijo k, P_{Epos+k} se puede representar como una función lineal de PE_{pos}.

También experimentamos con el uso de incrustaciones posicionales aprendidas [9] y descubrimos que las dos versiones produjeron resultados casi idénticos (consulte la fila (E) de la Tabla 3). Elegimos la versión sinusoidal porque puede permitir que el modelo extrapola longitudes de secuencia más largas que las encontradas durante el entrenamiento.

4 Por qué la autoatención

En esta sección comparamos varios aspectos de las capas de autoatención con las capas recurrentes y convolucionales comúnmente utilizadas para mapear una secuencia de representaciones de símbolos de longitud variable $(x1, ..., xn)$ a otra secuencia de igual longitud $(z1, ..., zn)$, con x_i, z_i $\epsilon\mathbb{R}^d$, como una capa oculta en un codificador o decodificador de transducción de secuencia típico. Para motivar nuestro uso de la autoatención consideramos tres desiderata.

Uno es la complejidad computacional total por capa. Otra es la cantidad de cálculo que se puede paralelizar, medida por el número mínimo de operaciones secuenciales requeridas.

El tercero es la longitud del camino entre dependencias de largo alcance en la red. Aprender dependencias de largo alcance es un desafío clave en muchas tareas de transducción de secuencias. Un factor clave que afecta la capacidad de aprender tales dependencias es la longitud de los caminos que deben recorrer las señales hacia adelante y hacia atrás en la red. Cuanto más cortos sean estos caminos entre cualquier combinación de posiciones en las secuencias de entrada y salida, más fácil será aprender dependencias de largo alcance [12]. Por lo tanto, también comparamos la longitud máxima de la ruta entre dos posiciones de entrada y salida cualesquiera en redes compuestas por diferentes tipos de capas.

Como se indica en la Tabla 1, una capa de autoatención conecta todas las posiciones con un número constante de operaciones ejecutadas secuencialmente, mientras que una capa recurrente requiere $O(n)$ operaciones secuenciales. En términos de complejidad computacional, las capas de autoatención son más rápidas que las capas recurrentes cuando la longitud de la secuencia n es menor que la dimensionalidad de representación d, que es el caso más frecuente con las representaciones de oraciones utilizadas por los modelos más modernos en traducciones automáticas, como representaciones de fragmentos de palabras [38] y pares de bytes [31]. Para mejorar el rendimiento computacional para tareas que involucran secuencias muy largas, la autoatención podría restringirse a considerar solo una vecindad de tamaño r en la secuencia de entrada centrada alrededor de la posición de salida respectiva. Esto aumentaría la longitud máxima del camino a $O(n/r)$. Planeamos investigar este enfoque más a fondo en trabajos futuros.

Una única capa convolucional con un ancho de núcleo $k < n$ no conecta todos los pares de posiciones de entrada y salida. Hacerlo requiere una pila de capas convolucionales $O(n/k)$ en el caso de núcleos contiguos, u $O(logk(n))$ en el caso de convoluciones dilatadas [18], aumentando la longitud de los caminos más largos entre dos posiciones cualesquiera. en la red. Las capas convolucionales son generalmente más caras que las capas recurrentes, por un factor de k. Las

convoluciones separables [6], sin embargo, disminuyen considerablemente la complejidad, a $O(k \cdot n \cdot d + n \cdot d^2)$. Sin embargo, incluso con $k = n$, la complejidad de una convolución separable es igual a la combinación de una capa de autoatención y una capa de retroalimentación puntual, el enfoque que adoptamos en nuestro modelo.

Como beneficio adicional, la autoatención podría generar modelos más interpretables. Inspeccionamos las distribuciones de atención de nuestros modelos y presentamos y analizamos ejemplos en el apéndice. Las cabezas de atención individuales no sólo aprenden claramente a realizar diferentes tareas, sino que muchas parecen exhibir un comportamiento relacionado con la estructura sintáctica y semántica de las oraciones.

5 Entrenamiento

Esta sección describe el régimen de entrenamiento para nuestros modelos.

5.1 Datos de entrenamiento y procesamiento por lotes

Nos entrenamos con el conjunto de datos estándar inglés-alemán WMT 2014 que consta de aproximadamente 4,5 millones de pares de oraciones. Las oraciones se codificaron utilizando codificación de pares de bytes [3], que tiene un vocabulario fuente-destino compartido de aproximadamente 37000 tokens. Para inglés-francés, utilizamos el conjunto de datos inglés-francés WMT 2014, significativamente más grande, que consta de 36 millones de oraciones y tokens divididos en un vocabulario de 32000 palabras [38]. Los pares de oraciones se agruparon por longitud de secuencia aproximada. Cada lote de entrenamiento contenía un conjunto de pares de oraciones que contenían aproximadamente 25 000 tokens de origen y 25 000 tokens de destino.

5.2 Hardware y programación

Entrenamos nuestros modelos en una máquina con 8 GPU NVIDIA P100. Para nuestros modelos base que utilizan los hiperparámetros descritos a lo largo del artículo, cada paso de entrenamiento tomó aproximadamente 0,4 segundos. Entrenamos los modelos base para un total de 100.000 pasos o 12 horas. Para nuestros modelos grandes, descritos en la última línea de la tabla 3), el tiempo de paso fue de 1,0 segundos. Los grandes modelos fueron entrenados para 300.000 pasos (3,5 días).

5.3 Optimizador

Usamos el optimizador Adam [20] con $\beta_1 = 0.9, \beta_2 = 0.98 y \epsilon = 10^{-9}$. Variamos la tasa de aprendizaje a lo largo de la formación, según la fórmula:

$$lrate = d_{model}^{-0.5} \cdot min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}) \tag{3}$$

Esto corresponde a aumentar la tasa de aprendizaje linealmente para los primeros pasos de entrenamiento de $warmup_steps$ y disminuirla posteriormente proporcionalmente a la raíz cuadrada inversa del número de paso. Usamos $warmup_steps = 4000$.

5.4 Regularización

Empleamos tres tipos de regularización durante la formación:

Suavizado de etiquetas Durante el entrenamiento, empleamos un suavizado de etiquetas de valor $\epsilon_{ls} = 0,1$ [36]. Esto duele la perplejidad, ya que el modelo aprende a ser más inseguro, pero mejora la precisión y la puntuación BLEU.

Abandono residual Aplicamos abandono [33] a la salida de cada subcapa, antes de agregarlo a la entrada de la subcapa y normalizarlo. Además, aplicamos abandono a las sumas de las incrustaciones y las codificaciones posicionales en las pilas de codificador y decodificador. Para el modelo base, utilizamos una tasa de $P_{drop} = 0.1$.

Suavizado de etiquetas Durante el entrenamiento, empleamos un suavizado de etiquetas de valor $\epsilon_{ls} = 0.1$ [36]. Esto duele la perplejidad, ya que el modelo aprende a ser más inseguro, pero mejora la precisión y la puntuación BLEU.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.92		$1.4 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
MoE [32]	26.03	40.56	$8.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$1.2 \cdot 10^{21}$
GNMT + RL Ensemble [38]	26.30	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.26	41.29	$1.8 \cdot 10^{20}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$3.3 \cdot 10^{18}$	
Transformer (big)	28.4	41.8	$2.3 \cdot 10^{19}$	

Table 2: El Transformer logra mejores puntajes BLEU que los modelos anteriores de última generación en las pruebas Newstest2014 de inglés a alemán e inglés a francés a una fracción del costo de capacitación.

6 Resultados

6.1 Traducción automática

En la tarea de traducción del inglés al alemán del WMT 2014, el modelo de transformador grande (Transformador (grande) en la Tabla 2) supera a los mejores modelos reportados anteriormente (incluidos los conjuntos) en más de 2.0 BLEU, estableciendo un nuevo estado de la situación. Puntuación artística BLEU de 28,4. La configuración de este modelo se enumera en la línea inferior de la Tabla 3. El entrenamiento tomó 3.5 días en 8 GPU P100. Incluso nuestro modelo base supera todos los modelos y conjuntos publicados anteriormente, a una fracción del costo de capacitación de cualquiera de los modelos competitivos.

En la tarea de traducción del inglés al francés del WMT 2014, nuestro modelo grande logra una puntuación BLEU de 41.0, superando a todos los modelos individuales publicados anteriormente, a menos de 1/4 del costo de capacitación del estado del arte anterior. El modelo Transformer (grande) entrenado para inglés a francés utilizó una tasa de deserción $Pdrop = 0.1$, en lugar de 0.3.

Para los modelos base, utilizamos un modelo único obtenido promediando los últimos 5 puntos de control, que se escribieron en intervalos de 10 minutos. Para los modelos grandes, promediamos los últimos 20 puntos de control. Utilizamos búsqueda de haz con un tamaño de haz de 4 y una penalización de longitud $\infty = 0,6$ [38]. Estos hiperparámetros se eligieron después de experimentar en el conjunto de desarrollo. Establecemos la longitud máxima de salida durante la inferencia en la longitud de entrada + 50, pero terminamos temprano cuando es posible [38].

La Tabla 2 resume nuestros resultados y compara nuestra calidad de traducción y costos de capacitación con otras arquitecturas modelo de la literatura. Estimamos la cantidad de operaciones de punto flotante utilizadas para entrenar un modelo multiplicando el tiempo de entrenamiento, la cantidad de GPU utilizadas y una estimación de la capacidad sostenida de punto flotante de precisión simple de cada GPU5[5].

6.2 Variaciones del modelo

Para evaluar la importancia de los diferentes componentes del Transformer, variamos nuestro modelo base de diferentes maneras, midiendo el cambio en el rendimiento en la traducción del inglés al alemán en el conjunto de desarrollo, newstest2013. Utilizamos la búsqueda de haz como se describe en la sección anterior, pero no promediamos el punto de control. Presentamos estos resultados en la Tabla 3.

En las filas de la Tabla 3 (A), variamos el número de cabezas de atención y las dimensiones clave y de valor de atención, manteniendo constante la cantidad de cálculo, como se describe en la Sección 3.2.2. Si bien la atención con un solo cabezal es 0.9 BLEU peor que la mejor configuración, la calidad también disminuye con demasiados cabezales.

En las filas de la Tabla 3 (B), observamos que reducir el tamaño de la clave de atención d_k perjudica la calidad del modelo. Esto sugiere que determinar la compatibilidad no es fácil y que una función de compatibilidad más sofisticada que el producto escalar puede ser beneficiosa. Además, observamos en las filas (C) y (D) que, como se esperaba, los modelos más grandes son mejores y que el abandono es muy útil para evitar el sobreajuste. En la fila (E) reemplazamos

[5]Utilizamos valores de 2,8, 3,7, 6,0 y 9,5 TFLOPS para K80, K40, M40 y P100, respectivamente.

nuestra codificación posicional sinusoidal con incrustaciones posicionales aprendidas [9] y observamos resultados casi idénticos al modelo base.

	N	d_{model}	d_{ff}	**h**	d_k	d_v	P_{drop}	ϵ_{ls}	**train steps**	**PPL (dev)**	**BLEU (dev)**	**params** $\times 10^6$
Base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	
	4									5.19	25.3	
	8									4.88	25.5	
		256			32	32				5.75	24.5	
		1024			128	128				4.66	26.0	
			1024							5.12	25.4	
			4096							4.75	26.2	
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	4.33	26.4	213

Table 3: Variaciones sobre la arquitectura Transformer. Los valores no listados son idénticos a los del modelo base. Todas las métricas están en el conjunto de desarrollo de traducción del inglés al alemán, newstest2013. Las perplejidades enumeradas son por pieza de palabra, según nuestra codificación de pares de bytes, y no deben compararse con perplejidades por palabra.

6.3 Análisis de circunscripciones inglesas

Para evaluar si el Transformer puede generalizarse a otras tareas, realizamos experimentos en el análisis de distritos electorales en inglés. Esta tarea presenta desafíos específicos: el resultado está sujeto a fuertes restricciones estructurales y es significativamente más largo que el insumo. Además, los modelos RNN secuencia a secuencia no han podido lograr resultados de última generación en regímenes de datos pequeños [37].

Entrenamos un transformador de 4 capas con $d_{model} = 1024$ en la parte del Wall Street Journal (WSJ) del Penn Treebank [25], alrededor de 40.000 oraciones de entrenamiento. También lo entrenamos en un entorno semisupervisado, utilizando corpus más grandes de alta confianza y BerkleyParser con aproximadamente 17 millones de oraciones [37]. Usamos un vocabulario de 16K tokens para la configuración de WSJ únicamente y un vocabulario de 32K tokens para la configuración semisupervisada.

Realizamos solo una pequeña cantidad de experimentos para seleccionar la deserción, tanto la atención como la residual (sección 5.4), las tasas de aprendizaje y el tamaño del haz en el conjunto de desarrollo de la Sección 22; todos los demás parámetros se mantuvieron sin cambios con respecto al modelo base de traducción del inglés al alemán. Durante la inferencia, aumentamos la longitud máxima de salida a la longitud de entrada + 300. Utilizamos un tamaño de haz de 21 y $\infty = 0.3$ tanto para WSJ únicamente como para la configuración semisupervisada.

Nuestros resultados en la Tabla 4 muestran que a pesar de la falta de ajuste específico de la tarea, nuestro modelo funciona sorprendentemente bien, arrojando mejores resultados que todos los modelos informados anteriormente, con la excepción de la gramática de redes neuronales recurrentes [8].

A diferencia de los modelos secuencia a secuencia RNN [37], Transformer supera a BerkeleyParser [29] incluso cuando se entrena solo en el conjunto de entrenamiento WSJ de 40K oraciones.

Parser	Training	WSJ 23 F1
Vinyals & Kaiser el al. (2014) [37]	WSJ only, discriminative	88.3
Petrov et al. (2006) [29]	WSJ only, discriminative	90.4
Zhu et al. (2013) [40]	WSJ only, discriminative	90.4
Dyer et al. (2016) [8]	WSJ only, discriminative	91.7
Transformer (4 layers)	WSJ only, discriminative	91.3
Zhu et al. (2013) [40]	semi-supervised	91.3
Huang & Harper (2009) [14]	semi-supervised	91.3
McClosky et al. (2006) [26]	semi-supervised	92.1
Vinyals & Kaiser el al. (2014) [37]	semi-supervised	92.1
Transformer (4 layers)	semi-supervised	92.7
Luong et al. (2015) [23]	multi-task	93.0
Dyer et al. (2016) [8]	generative	93.3

Table 4: The Transformer se generaliza bien al análisis de distritos electorales en inglés (los resultados se encuentran en la Sección 23 del WSJ).

7 Conclusión

En este trabajo, presentamos Transformer, el primer modelo de transducción de secuencia basado completamente en la atención, reemplazando las capas recurrentes más comúnmente utilizadas en arquitecturas de codificador-decodificador con autoatención de múltiples cabezas.

Para tareas de traducción, Transformer se puede entrenar significativamente más rápido que las arquitecturas basadas en capas recurrentes o convolucionales. Tanto en las tareas de traducción de inglés a alemán de WMT 2014 como de inglés a francés de WMT 2014, logramos un nuevo estado del arte. En la primera tarea, nuestro mejor modelo supera incluso a todos los conjuntos reportados anteriormente.

Estamos entusiasmados con el futuro de los modelos basados en la atención y planeamos aplicarlos a otras tareas. Planeamos extender Transformer a problemas que involucran modalidades de entrada y salida distintas al texto e investigar mecanismos locales de atención restringida para manejar eficientemente grandes entradas y salidas, como imágenes, audio y video. Hacer que la generación sea menos secuencial es otro de nuestros objetivos de investigación.

El código que utilizamos para entrenar y evaluar nuestros modelos está disponible en `https://github.com/tensorflow/tensor2tensor`. Agradecimientos Agradecemos a Nal Kalchbrenner y Stephan Gouws por sus fructíferos comentarios, correcciones e inspiración.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[3] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. Massive exploration of neural machine translation architectures. In Martha Palmer, Rebecca Hwa, and Sebastian Riedel, editors, *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1442–1451, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[4] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. In Jian Su, Kevin Duh, and Xavier Carreras, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 551–561, Austin, Texas, November 2016. Association for Computational Linguistics.

[5] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

[6] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

[7] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

[8] Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. Recurrent neural network grammars. In Kevin Knight, Ani Nenkova, and Owen Rambow, editors, *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209, San Diego, California, June 2016. Association for Computational Linguistics.

[9] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning, 2017.

[10] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[12] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer and J. F. Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[14] Zhongqiang Huang and Mary Harper. Self-training PCFG grammars with latent annotations across languages. In Philipp Koehn and Rada Mihalcea, editors, *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 832–841, Singapore, August 2009. Association for Computational Linguistics.

[15] Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *CoRR*, abs/1602.02410, 2016.

[16] Ł ukasz Kaiser and Samy Bengio. Can active memory replace attention? In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[17] Lukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. *arXiv: Learning*, 2015.

[18] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aäron van den Oord, Alexander Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. 2016.

[19] Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. Structured attention networks. In *International Conference on Learning Representations*, 2017.

[20] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[21] Oleksii Kuchaiev and Boris Ginsburg. Factorization tricks for lstm networks, 2018.

[22] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A STRUCTURED SELF-ATTENTIVE SENTENCE EMBEDDING. In *International Conference on Learning Representations*, 2017.

[23] Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*, 2015.

[24] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In Lluís Màrquez, Chris Callison-Burch, and Jian Su, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[25] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[26] David McClosky, Eugene Charniak, and Mark Johnson. Effective self-training for parsing. In Robert C. Moore, Jeff Bilmes, Jennifer Chu-Carroll, and Mark Sanderson, editors, *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 152–159, New York City, USA, June 2006. Association for Computational Linguistics.

[27] Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. In Jian Su, Kevin Duh, and Xavier Carreras, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2249–2255, Austin, Texas, November 2016. Association for Computational Linguistics.

[28] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization, 2017.

[29] Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In Nicoletta Calzolari, Claire Cardie, and Pierre Isabelle, editors, *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440, Sydney, Australia, July 2006. Association for Computational Linguistics.

[30] Ofir Press and Lior Wolf. Using the output embedding to improve language models. In Mirella Lapata, Phil Blunsom, and Alexander Koller, editors, *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163, Valencia, Spain, April 2017. Association for Computational Linguistics.

[31] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In Katrin Erk and Noah A. Smith, editors, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics.

[32] Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

[33] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[34] Sainbayar Sukhbaatar, arthur szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[35] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[36] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015.

[37] Oriol Vinyals, Ł ukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. Grammar as a foreign language. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[38] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016.

[39] Jie Zhou, Ying Cao, Xuguang Wang, Peng Li, and Wei Xu. Deep recurrent models with fast-forward connections for neural machine translation. *Transactions of the Association for Computational Linguistics*, 4:371–383, 2016.

[40] Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. Fast and accurate shift-reduce constituent parsing. In Hinrich Schuetze, Pascale Fung, and Massimo Poesio, editors, *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 434–443, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

Visualizaciones de atención

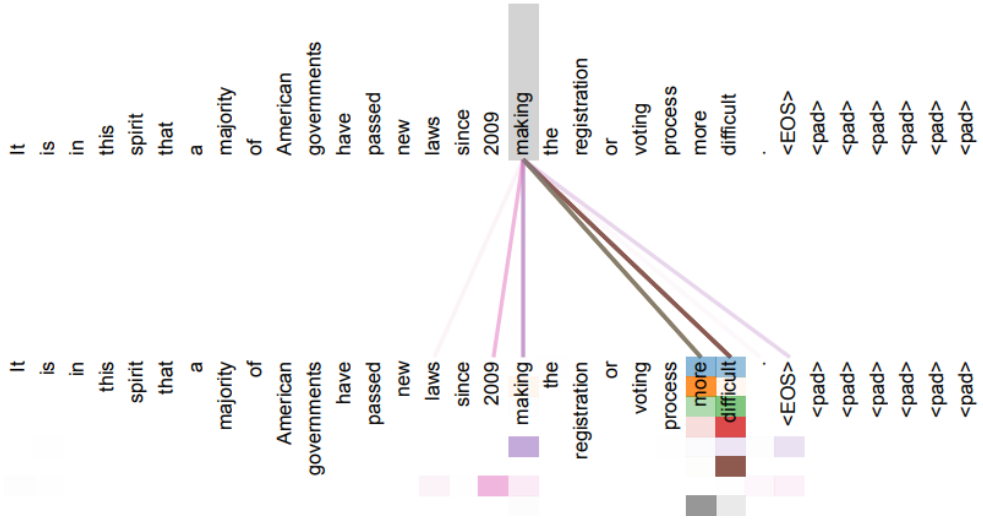

Figure 3: Un ejemplo del mecanismo de atención que sigue dependencias a larga distancia en la autoatención del codificador en la capa 5 de 6. Muchas de las cabezas de atención atienden a una dependencia distante del verbo "making (hacer)", completando la frase "making...more difficult". Las atenciones aquí se muestran sólo para la palabra "making (hacer)". Diferentes colores representan diferentes cabezas. Mejor visto en color.

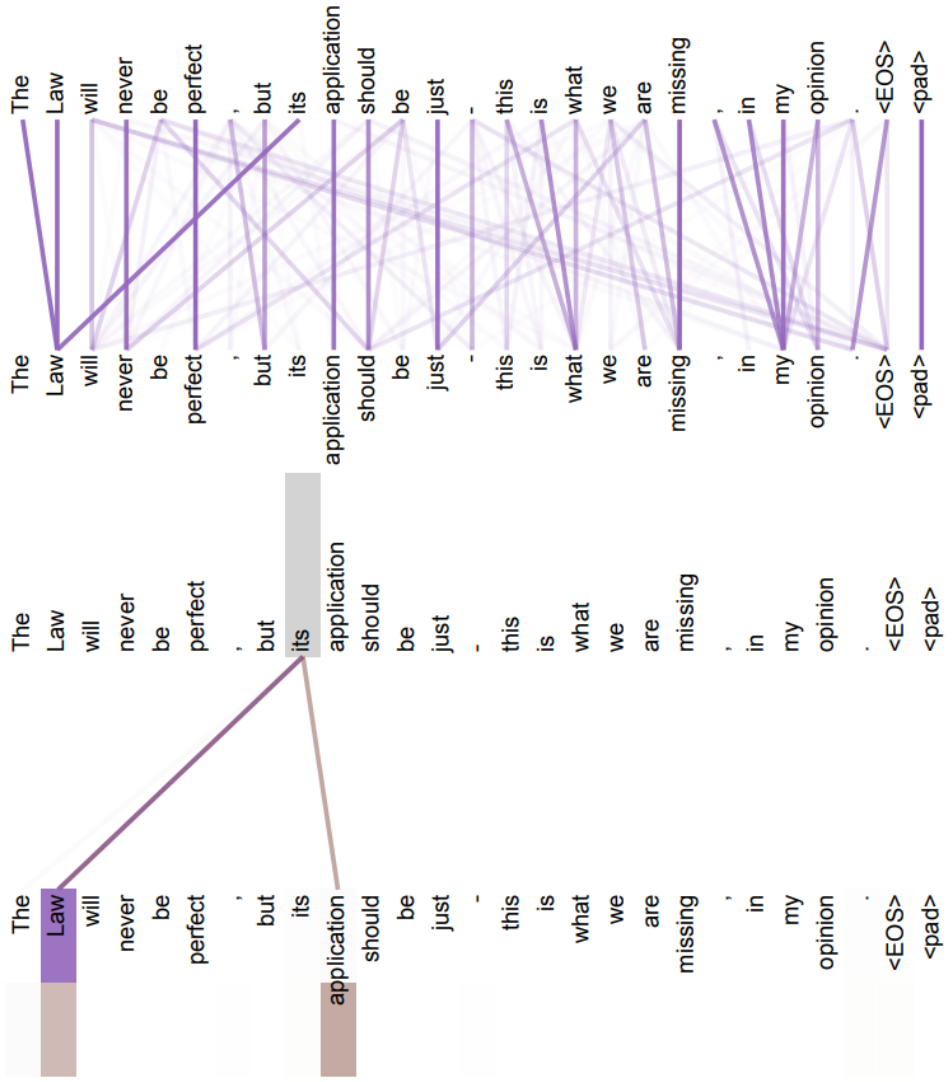

Figure 4: Dos cabezas de atención, también en la capa 5 de 6, aparentemente involucradas en la resolución de la anáfora. Arriba: atenciones completas para los encabezados 5. Abajo: atenciones aisladas solo de la palabra "its" para los encabezados de atención 5 y 6. Tenga en cuenta que las atenciones son muy nítidas para esta palabra.

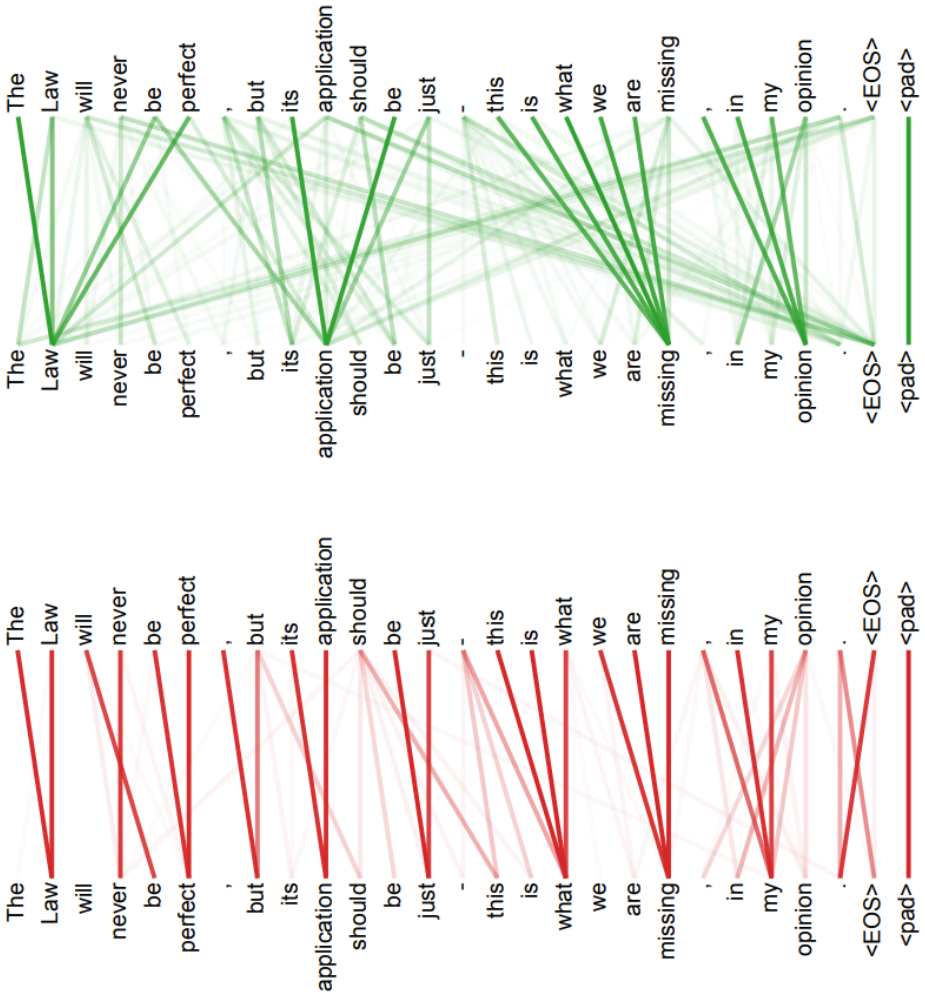

Figure 5: Muchas de las cabezas de atención exhiben un comportamiento que parece relacionado con la estructura de la oración. Arriba damos dos ejemplos de este tipo, de dos cabezales diferentes de la autoatención del codificador en la capa 5 de 6. Los cabezales claramente aprendieron a realizar diferentes tareas.