Multiformer: A head-configurable Transformer for Direct Speech Translation

Trabajo Fin de Grado realizado en la

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya

por

Gerard Sant Muniesa

En cumplimiento parcial de los requisitos para el Grado en

Ingeniería de Tecnologías y Servicios de Telecomunicación

Supervisores:

Gerard I. Gállego

Marta R. Costa-Jussà

Barcelona, enero 2022

Abstract

Los modelos basados en el *Transformer* se han ido estableciendo como líderes en varios campos de *Natural Lenguage Processing*, como es el caso de *Speech Translation*. Sin embargo, dado que éste posee una complejidad cuadrática respecto a la longitud de la secuencia de entrada, su aplicación directa a tareas de audio no es trivial. Por ende, es común reducir la longitud de las secuencias de audio mediante capas convolucionales previas al *Encoder*. Este trabajo añade otro enfoque y se cuestiona la utilización de la *Multi-Head Attention*, originalmente planteada para texto, para la extracción de información de la voz, concretamente en *Speech-to-Text Translation*.

En este estudio, presentamos el *Multiformer*, un modelo basado en el *Transformer* que permite aplicar diferentes mecanismos de atención en cada *head*. Al introducir diversidad de atención, se reduce la redundancia de la información extraída por los distintos *heads*. Nuestros resultados muestran que el *Multiformer* con diversidad de atención tanto a nivel de *head*, como a nivel de capa, supera al modelo de referencia por ≈ 0.4 puntos de BLEU.

Agradecimientos

Quiero agradecer a Gerard I. Gállego y Marta R. Costa-Jussà por descubrirme el grupo de *Machine Translation*, donde he despertado mi vocación por el mundo de la investigación. Agradecer también su total apoyo y guía en el desarrollo de este proyecto. Además, quiero hacer mención especial a Belén Alastruey, quien, desinteresadamente, siempre ha ofrecido su ayuda, haciendo honor al significado de la amistad.

Índice

Índice de Figuras

Índice de Tablas

Nomenclatura

NLP	*Natural Lenguage Processing*
ST	*Speech Translation*
S2T	*Speech-to-Text Translation*
MT	*Machine Translation*
SMT	*Statistical Machine Translation*
ASR	*Automatic Speech Recognition*
End2End	*End to End*
Seq2Seq	*Sequence to Sequence*
RNN	*Recurrent Neural Network*
FFNN	*Feedforward Neural Network*
LSTM	*Long Short Time Memory*
HMM	*Hiden Markov Model*
WER	*Word Error Rate*
BLEU	*Bilingual Evaluation Understudy score*

1 Introducción

El ser humano logró diferenciarse del resto de los animales debido a su capacidad de utilizar herramientas e ingenio para suplir sus carencias físicas. Como muy bien acertó Lev Vigotski con la afirmación "*El lenguaje es la herramienta de las herramientas*" [Vygotsky and Cole, 1978], la comunicación fue el detonador de nuestra evolución. El habla nos proporcionó la capacidad de transmitir nuestros conocimientos y experiencias con nuestros iguales. Por ende, no es precipitado situar la comunicación como el eje principal del progreso. Sin embargo, una correcta comunicación requiere que emisor y receptor entiendan la lengua empleada. Por este motivo, el procesamiento del habla y la traducción han sido siempre objeto de investigación. En el campo de la Inteligencia Artificial, la tarea encargada de unir ambos conceptos es *Speech Translation*, en particular *Speech-to-Text Translation*.

El planteamiento convencional para *Speech-to-Text Translation* consiste en la concatenación de dos bloques independientes: (i) Un modelo encargado de realizar *Automatic Speech Recognition*, transcripción del audio, y (ii) otro modelo de *Machine Translation* para la traducción de dicha transcripción al idioma deseado. Sin embargo, este método en cascada [Ney, 1999] ignora cierta información del audio, dado que traduce mediante la transcripción, y es vulnerable a la propagación de errores, ya que un error en el bloque ASR provoca automáticamente una mala traducción [Sperber and Paulik, 2020, Bentivogli et al., 2021]. Por consiguiente, en los últimos años las alternativas *End-to-End* basadas en una estructura *Encoder-Decoder* y mecanismos de atención han incrementado su popularidad [Anastasopoulos et al., 2016, Duong et al., 2016, Bérard et al., 2016, Weiss et al., 2017, Bérard et al., 2018]. Éstas son capaces de traducir el audio sin la necesidad explícita de una transcripción, por lo que evitan los problemas del planteamiento en cascada y permiten la optimización unificada de los parámetros en entrenamiento.

En el año 2017, la aparición del *Transformer* [Vaswani et al., 2017] revolucionó el campo de traducción de texto, logrando que modelos basados en esta arquitectura obtuvieran las mejores métricas. Actualmente, se utilizan modelos basados en el *Transformer* para procesar todo tipo de datos, como es el caso de imágenes [Parmar et al., 2018] o, en este estudio, la voz [Vila et al., 2018, Di Gangi et al., 2019]. En el preprocesamiento del audio, se emplea una ventana con un *Hop size* de $10ms$, por lo que, en S2T, se tiene un *token* cada $10ms$ de audio en la entrada del modelo, lo que provoca que estas secuencias sean considerablemente más largas que las de texto y, por consiguiente, requiere la implementación de estrategias de *down sampling* previas a la entrada del modelo [Wang et al., 2020a]. Además, debido que en el preprocesamiento de la voz se utiliza una ventana de

25ms, provocando *overlaping* en muestreo, junto a que se requieren varios *tokens* para la representación de un único fonema, existe mucha redundancia entre los *tokens* de audio. Por lo tanto, la naturaleza de las secuencias de audio difiere enormemente respecto a las texto, cuyos *tokens* están formados por un conjunto de caracteres. Por ende, la calidad de las traducciones S2T obtenidas con el *Transformer* es inferior que en MT. En esta línea, algunos estudios proponen extraer secuencias más ricas en información mediante módulos de compresión preentrenados [Salesky et al., 2019, Zhang et al., 2020, Na et al., 2019, Gaido et al., 2021, Papi et al., 2021]. Si bien éstos consiguen buenos resultados, nosotros proponemos un cambio de enfoque, cuestionando el uso de la *Multi-Head Attention*, originalmente planteada para texto, para la extracción de información de las secuencias de voz.

El objetivo de este estudio es contribuir a la investigación de *Speech-to-Text Translation* presentando el *Multiformer*. Basado en el *Transformer*, nuestro modelo permite un alto nivel de personalización del *Encoder*, encargado de extraer información de la secuencia de entrada. Mediante esta configuración, se pretende adaptar la extracción de características a las características de los datos de voz. Para dicha adaptación, el *Multiformer* ofrece la oportunidad de emplear paralelamente distintos mecanismos de atención en un mismo *Encoder-layer*, gracias a la creación de la *Multi-Head Multi-Attention*. Este nuevo módulo permite escoger y configurar el mecanismo de atención de cada uno de sus *heads*. En particular, la *Multi-Head Multi-Attention* dispone de (i) la *Sliding window*, implementada en el *Longformer* [Beltagy et al., 2020] y encargada de la extracción de relaciones locales, (ii) la *ConvAttention*, presente en el *Speechformer* [Papi et al., 2021] y responsable de la obtención de relaciones entre *tokens* más informativos, (iii) la *Fast-Attention*, propuesta en el *Performer* [Choromanski et al., 2020b] para una extracción global eficiente, y (iv) la *Full-Attention*, presente en el *Transformer* para realizar una búsqueda global exhaustiva. Gracias a la utilización de distintos mecanismos de atención en cada *head*, estimulamos la creación de relaciones más variadas y, por consiguiente, disminuimos la redundancia de la información extraída [Voita et al., 2019, Bian et al., 2021]. Con el fin de permitir una adaptación más concreta y forzar una mayor diversificación de la atención del modelo, el *Multiformer* habilita la posibilidad de configurar cada una de las capas del *Encoder*. Mediante esta diversidad forzada y la utilización de mecanismos de atención más adecuados para audio, pensamos que el modelo es capaz de crear relaciones más variadas, complejas y apropiadas para la voz.

2 Background

En esta sección se expondrán los conceptos básicos para entender la motivación de este estudio. Además, se dará contexto al estado actual de la materia, explicando la evolución de las tecnologías de *Natural Language Processing* (NLP) protagonistas en la motivación y creación de *Speech Translation* (ST).

2.1 Antecedentes de *Speech2Text Translation*

En este apartado se presentarán los dos antecedentes naturales claves para entender la historia y la evolución de ST.

2.1.1 *Automatic Speech Recognition*

Automatic Speech Recognition, que responde a las siglas ASR, es la tecnología que permite a los ordenadores reconocer y convertir el habla humana en texto lográndose, de este modo, una tarea similar a la transcripción. Este tipo de tareas son de gran complejidad, ya que se debe tener en cuenta los problemas asociados de trabajar con el habla oral. Entre ellos podemos destacar (i) los relacionados con la calidad de la señal, como la presencia de ruido en las señales acústicas o el eco existente en estas señales, y (ii) los relacionados con la propia lengua, como los discursos desorganizados, las palabras homófonas y los distintos acentos entre los hablantes de una misma lengua.

Este ha sido un campo de gran interés y avance en los últimos 60 años. A finales de la década de 1970, el campo del ASR experimentó su primer gran cambio de paradigma [Baker, 1975], abandonando los métodos simples de reconocimiento de patrones, basados en plantillas y medidas de distancia espectral; y adoptando un método estadístico para el procesamiento de voz, basado en el *Hidden Markov Model* (HMM) [Márkov, 1906].

Siendo $\boldsymbol{Y}_{1:T} = \boldsymbol{y}_1, ..., \boldsymbol{y}_T$ una secuencia de vectores acústicos de longitud fija, estos modelos estadísticos intentan encontrar la secuencia de palabras y/o fonemas $\boldsymbol{\omega}_{1:L} = \omega_1, ..., \omega_L$ más probable de haber sido generada por $\boldsymbol{Y}$:

$$\hat{\boldsymbol{\omega}} = arg \max_{\omega} \{P(\boldsymbol{\omega}|\boldsymbol{Y})\} \tag{1}$$

Como $\{P(\boldsymbol{\omega}|\boldsymbol{Y})\}$ es muy difícil de modelar directamente, mediante la regla de Bayes[1] obtenemos un problema equivalente:

[1]El Teorema de Bayes [Bayes, 1763] es una teoría de probabilidad que relaciona la probabilidad condicional de un evento aleatorio A dado B con la distribución de probabilidad condicional del evento B dado A y la distribución de probabilidad marginal de A.

$$\hat{\boldsymbol{\omega}} = arg\max_{\omega} \{p(\boldsymbol{Y}|\boldsymbol{\omega})P(\boldsymbol{\omega})\} \tag{2}$$

Donde la probabilidad $p(\boldsymbol{Y}|\boldsymbol{\omega})$ es el *modelo acústico*, que indica la probabilidad de que $\boldsymbol{Y}$ sea el espectrograma que representa las palabras y/o fonemas $\boldsymbol{\omega}_{1:L}$, y $P(\boldsymbol{\omega})$ es el *modelo de lenguaje*, que determina la probabilidad de que el conjunto de palabras generado sea gramaticalmente correcto.

Los modelos estadísticos permitían la detección de pequeños patrones con el objetivo de modelar frases cortas, palabras o fonemas. Aún pudiendo reconocer ciertos patrones, sus resultados eran de calidad insuficiente, pero sentaron precedente en los progresos de las siguientes cuatro décadas, donde aparecieron modelos como *n-gram* [Katz, 1987], uniendo el antiguo HMM con la teoría de la Información de Shannon [Shannon, 1948].

En la década de los años 2000s, los primeros modelos neuronales substituyeron el modelo de lenguaje y el modelo acústico por *Long short-term memory* (LSTM) [Hochreiter and Schmidhuber, 1997], basada en Redes Neuronales Recurrentes (RNN) [Rumelhart et al., 1986]. Este tipo de redes son útiles en tareas donde existe una relación recurrente entre los elementos de entrada y salida del sistema. En 2014, se produjo el mayor avance en ASR con la aparición del sistema propuesto por Graves and Jaitly [2014] basado en "Clasificación temporal conexionista" (CTC) [Graves et al., 2006] y en LSTM. Esta arquitectura logró los mejores resultados hasta el momento, pero proseguía con la dificultad de mantener el contexto de largas secuencias. Esto era debido a las suposiciones de independencia condicional, que ya aparecían en los modelos de HMM.

En estos últimos años, han aparecido modelos de ASR basados en *Attention* (§2.3.2), como "*Listen, Attend and Spell*" (LAS) [Chan et al., 2016], cuyo nombre hace referencia a su funcionamiento literal: "*Listen*", escuchar la señal acústica; "*Attend*", prestar atención a las diferentes partes de la señal; y "*Spell*" deletrear el texto resultante. Si bien estas nuevas arquitecturas buscan combatir la falta de contexto lejano mediante mecanismos de atención, sus resultados no superan los basados en CTC-RNN.

2.1.2 *Machine Translation*

Machine Translation (MT) es el proceso de conversión de texto de un idioma a otro mediante un software de traducción automática. El objetivo es traducir automáticamente textos, expresiones y modismos complejos de un idioma a otro. Lejos de ser una tarea sencilla, su ejecución resulta complicada debido a las diferencias existentes en la sintaxis, la semántica, y la gramática de los diferentes idiomas. Ya sea un traductor humano o

automático, el texto debe dividirse en elementos básicos para extraer completamente el mensaje y restaurarlo con precisión en el idioma de destino. Por ello, es fundamental que un traductor automático abarque la totalidad de los matices de un idioma, incluidos los subdialectos regionales.

Desde una visión más analítica, la dificultad de esta tarea viene dada principalmente por los siguientes motivos:

- *Disambiguation*: La dificultad de encontrar una traducción adecuada cuando una palabra o secuencia puede tener distintos significados.
- Secuencias largas: La capacidad de mantener el contexto de una frase para predecir su traducción es limitada, ya que en secuencias de larga extensión pueden aparecer problemas de falta de memoria, necesaria para guardar toda la información. Por otro lado, si no se considera la totalidad del texto o de la frase a traducir, es posible perder el contexto.
- Tamaño de *Corpus*: Es uno de los problemas más comunes en *Deep learning*. En tareas de MT puede suponer una falta de recursos para poder entrenar adecuadamente modelos de traducción de lenguas minoritarias.
- Alineamiento de palabras y/o frases 1: La construcción semántica del idioma destino puede variar completamente respecto a la de la lengua origen.

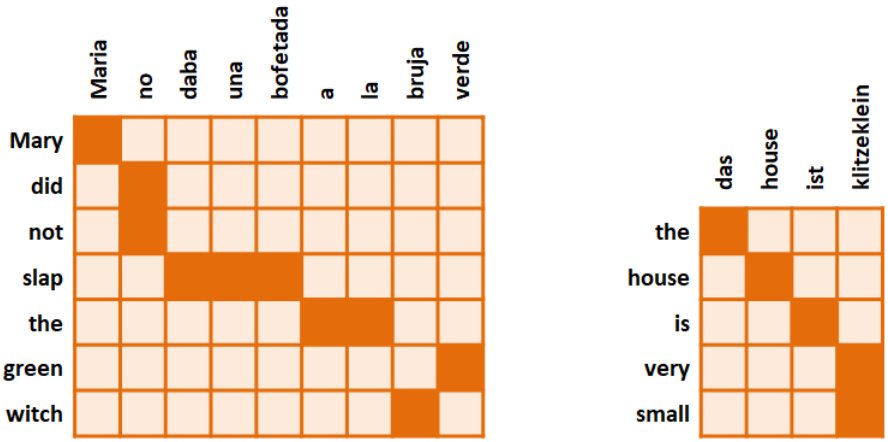


Figura 1: Alineamiento entre palabras: Izquierda Inglés-Castellano, Derecha Inglés-Alemán

- Palabras y discursos no comunes.
- *Domain mismtach* [Shen et al., 2021]: En diferentes dominios las palabras pueden tener traducciones distintas, o los significados pueden estar expresados en estilos

diferentes. Consecuentemente, en traducciones con pocos recursos de entrenamiento, puede conducir a cambios de estilo, registro o forma.

El interés en el campo de MT se remonta al siglo IX, cuando el criptógrafo Árabe Al-Kindi desarrolló las primeras técnicas de traducción sistemática, sentando los cimientos sobre métodos estadísticos. No obstante, no fue hasta los años 1950s cuando aparecieron los primeros investigadores especializados en este campo, logrando crear sistemas de procesamiento de lenguaje asistido por computadoras. A principios de los años 1980s, los ordenadores empezaron a ser económicamente más accesibles, creciendo el interés en modelos estadísticos para la traducción automática (SMT en adelante).

Los SMT estaban basados en la teoría de la información [Shannon, 1948] e inspirados en ideas de criptografía [Shannon, 1949]. Si se define $\boldsymbol{e}$ como la cadena de la lengua origen (e.g, Inglés) y $\boldsymbol{f}$ como la cadena en el idioma de objetivo (e.g, Catalán), el documento será traducido siguiendo la distribución de probabilidad $\boldsymbol{p(e|f)}$. Entonces, encontrar la mejor traducción $\hat{\boldsymbol{e}}$ significa elegir la de mayor probabilidad:

$$\hat{\boldsymbol{e}} = arg\max_{e \in e^*} \{p(\boldsymbol{e}|\boldsymbol{f})\} \xrightarrow{R.Bayes} arg\max_{e \in e^*} \{p(\boldsymbol{f}|\boldsymbol{e})p(\boldsymbol{e})\} \tag{3}$$

Como se define en la ecuación 3 que si se desea efectuar una implementación rigurosa del modelo, es necesario realizar una búsqueda exhaustiva a través de todas las cadenas e^* en la lengua nativa. Este hecho evidencia la existencia de un compromiso, también presente en ASR, entre la calidad y el tiempo requerido para la traducción.

Los modelos de traducción estadística estuvieron inicialmente basados en palabras [Vogel et al., 1996, Och and Ney, 2003], es decir, la traducción se realizaba palabra por palabra. Sin embargo, en la década de los años 2000, se lograron grandes avances con propuestas de modelos que trabajaban a nivel de frase [Koehn et al., 2003, Chiang, 2005].

En 2014, aparecieron los primeros estudios científicos que proponían la utilización de redes neuronales en MT [Sutskever et al., 2014, Cho et al., 2014], ganando éstas gran popularidad entre la comunidad científica por sus buenos resultados[Bahdanau et al., 2015], que superaron considerablemente a los de los antiguos SMT.

2.2 *Speech-to-text Translation*

Speech-to-text Translation (S2T) representa la "unión" de las tareas de ASR y MT (§2.1.1 y §2.1.2). Al lidiar con secuencias de audio[2], es decir el habla oral, aparecen dificultades añadidas a las ya propias de MT, como la degradación de la calidad del mensaje. Si bien en la traducción de texto a texto se goza mayoritariamente de un *input* conciso, claro y limpio, en ST, este *input* posee mucha más redundancia, ruido, construcciones incoherentes, reformulaciones de frases, etc. complicando considerablemente la obtención de traducciones precisas.

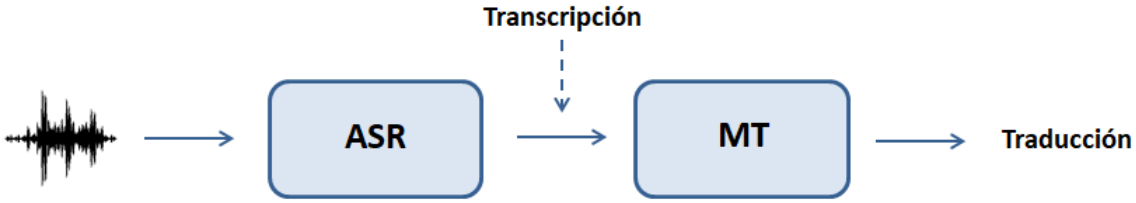


Figura 2: Esquema de un modelo de S2T en Cascada.

En la breve historia de las arquitecturas orientadas a ST, sus progresos han hecho de éste campo científico un sector muy prometedor. A finales de la década de los 90s, Ney [1999] presentó el primer modelo para ST basado en un esquema en cascada (figura 2). Estos modelos consisten en el posicionamiento en cadena de dos bloques claramente diferenciados, el bloque de ASR, transcribiendo la señal de audio a texto, y el de MT que traduce el texto de la salida del primer bloque.

Al ser estas arquitecturas la concatenación de dos modelos cuyas tareas son plenamente independientes entre sí presentan una gran sensibilidad a la propagación de errores y a la pérdida de información, ya que un solo error en el bloque ASR provoca automáticamente una mala traducción por parte del otro bloque [Sperber and Paulik, 2020, Bentivogli et al., 2021]. Además, dependiendo del modelo usado en MT, estos errores pueden significar mucho más que la simple mala traducción de una palabra, ya que pueden desencadenar una pérdida total del contexto de la frase.

Después de muchos años con la predominancia del modelo en cascada, se presentó el primer modelo *End-to-End* (o directo) para ST [Bérard et al., 2016, Weiss et al., 2017] (figura 3) como una prometedora alternativa a los sistemas clásicos, proponiendo una arquitectura *Encoder-Decoder* basada en un sistema *sequence-to-sequence* ASR [Chan

[2]Las secuencias de audio generalmente están formadas por *tokens* que contienen características del audio extraídas del espectrograma de mel [Stevens et al., 1937].

et al., 2016]. Si bien estos modelos solucionan la propagación de errores de los modelos en cascada, sufren de la escasez de *corpus* especifico disponible que dificulta drásticamente su capacidad de entrenamiento. Esto no ocurre con los modelos en cascada, que permiten entrenar cada bloque por separado, rompiendo con la necesidad de utilizar un *Corpus* exclusivo de S2T y, por ende, la cantidad de datos disponibles para su entrenamiento es mayor. Consecuentemente, se deduce la existencia de un compromiso entre la propagación de errores del planteamiento en cascada y la falta de grandes conjuntos de datos para entrenar los sistemas de ST directos.

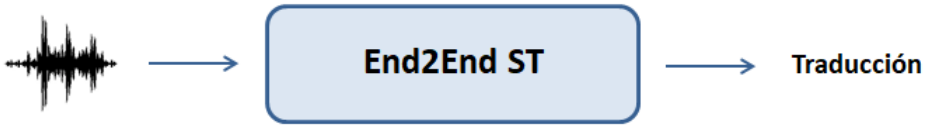


Figura 3: Esquema de un modelo *End-to-End*.

Los modelos *End-to-End* continúan siendo objeto de investigación, ya que existe la intuición de que estos sistemas pueden mejorar los resultados debido a dos razones principales: (i) eliminan las propagaciones de errores y (ii), al tener acceso directo al audio de entrada, evitan pérdidas de información propias de la transcripción (e.g. La entonación, el énfasis, etc.). No obstante, incluye complicaciones añadidas, como la escasez de *corpus*, mencionada anteriormente, y la dificultad de alineación entre los *inputs* y los *outputs*, debida a una gran diferencia en sus longitudes y características.

Actualmente, entre la comunidad científica no existe un consenso claro a la hora de afirmar que planteamiento es el más adecuado. Esta división de opiniones surge a raíz de las puntuaciones de BLEU (§2.4.2) alcanzadas por ambos planteamientos en las últimas ediciones de la IWLST.[3] Si bien entre las ediciones de 2018 [Jan et al., 2018] y 2019 [Niehues et al., 2019] el enfoque *End2End* se situó de 7,4 a 1,6 puntos por debajo del planteamiento en cascada, en 2020 [Ansari et al., 2020] superó por primera vez a los modelos en cascada, por 0.24 puntos de BLEU de diferencia. Fruto de los resultados tan similares de ambos enfoques, Bentivogli et al. [2021] proponen que no existe una diferencia plausible o, por lo menos, relevante. Su estudio se ve reforzado por los resultados obtenidos en la pasada edición de la IWLST [Anastasopoulos et al., 2021], donde, aún no alcanzando los resultados de 2020, nuevamente los modelos en cascada se interponen por 1,2 puntos frente los *End2End*.

[3]La *International Conference on Spoken Language Translation* (IWLST) organiza anualmente una campaña dedicada a evaluar los sistemas de *Speech-to-text Translation*.

2.3 *Sequence-to-Sequence*

Tal y como se ha comentado en apartados anteriores, para lograr una buena traducción, ya sea en MT o ST, se requiere la capacidad de mantener el contexto del discurso. Es decir, existe la necesidad de que los algoritmos de traducción puedan tener acceso a la información de toda, o gran parte de la oración para poder discernir la palabra o representación más adecuada. Consecuentemente, en este tipo de tareas no es adecuado traducir palabra por palabra, dado que considerar únicamente la propia palabra para traducir da lugar a traducciones literales y de calidad insuficiente (figura 4).

Figura 4: Ejemplo de traducción palabra por palabra de Castellano a Inglés.

Fruto de la motivación de considerar el contexto, para mejorar estas traducciones literales, nacieron los modelos *Seq2Seq*, permitiendo trabajar a nivel de secuencia (palabras, frases, segmentos de audio, etc.). A gran escala, estos modelos se basan en arquitecturas "*encoder-decoder*" (figura 5), donde ambos bloques desempeñan funciones diferenciadas. El *encoder* se encarga de procesar y almacenar la información de la secuencia de entrada. Por otro lado, el *decoder* utiliza esa información para generar la secuencia de salida.

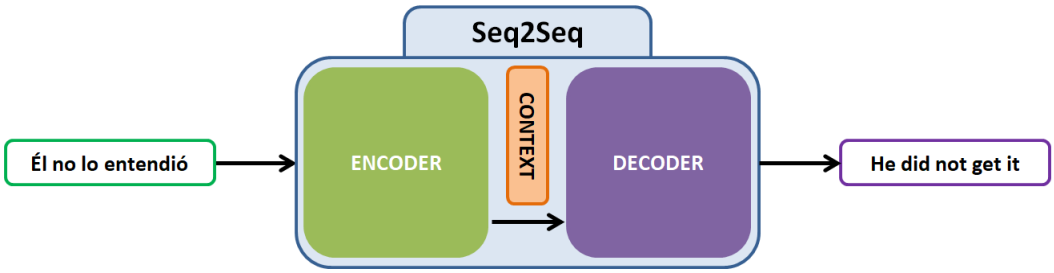


Figura 5: Esquema general del funcionamiento de un modelo *Seq2Seq*.

Con el surgimiento de los modelos *Seq2Seq* se revolucionó el campo de NMT, apareciendo los primeros intentos de aplicación de redes neuronales en este terreno [Sutskever et al., 2014, Cho et al., 2014, Kalchbrenner et al., 2017]. Estos modelos implementaban el *encoder* y el *decoder* mediante Redes Neuronales Recurrentes.

2.3.1 "*Recurrent Neural Network*"

Las RNN son un tipo de red neuronal que mediante el uso de una "memoria" interna, conocida como *hidden state*, son capaces de procesar secuencias de entrada de longitud variable. Las RNN pueden manejar series temporales mediante el *hidden state* recurrente cuya activación en cada instante depende de la del instante anterior. El planteamiento de la RNN esta basado en la utilización de una celda, encargada de actualizar el *hidden state* y de calcular el vector *embedding*.[4] El funcionamiento de la celda es el siguiente (figura 6):

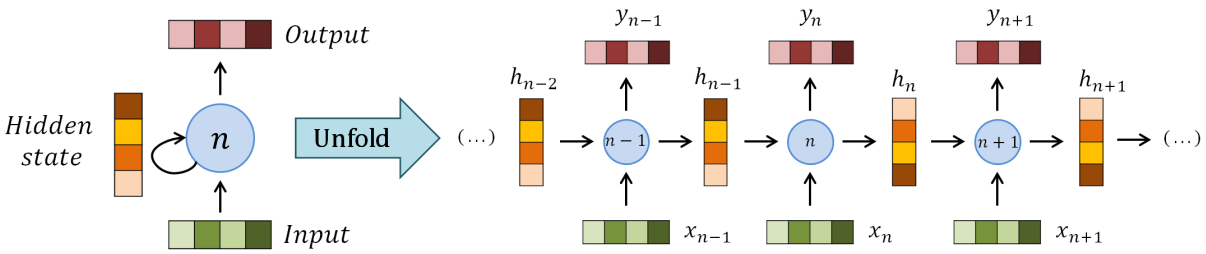


Figura 6: Esquema del funcionamiento de una celda de la RNN.

1. En cada instante de tiempo se introducen dos vectores a la celda: (i) el vector *embedding* y (ii), el vector *hidden state* del instante anterior $(n-1)$.
2. Mediante el procesamiento de estos dos vectores, la celda devuelve el vector de salida y actualiza el vector *hidden state*. Este último vector almacena información de los instantes de tiempo transcurridos y se utilizará para realizar el mismo procedimiento en la siguiente iteración.
3. Se repite este procedimiento hasta haber procesado todos los *tokens* de la secuencia, dando lugar a la recurrencia de este modelo.

Para crear un esquema *Seq2Seq* se emplean las celdas RNN para construir el sistema "*encoder-decoder*" representado en la figura 7. En la misma, se muestra cómo en cada iteración el *encoder* comprime toda la información en el vector de contexto (h_{out}). Seguidamente, el *decoder* utiliza este vector para efectuar las predicciones de manera autoregresiva. Es decir, cada vez que se predice un *token*, éste se introduce de nuevo en el *decoder* para realizar la siguiente predicción. El proceso no finaliza hasta que se predice el *token* <*EOS*> (*End Of Sentence*). Este proceso varia ligeramente en entrenamiento, donde se utiliza la técnica de *Teacher forcing*. Ésta consiste en introducir siempre el *token*

[4]Para poder procesar secuencias de texto, se utiliza técnicas de procesamiento del lenguaje para representar palabras o frases como vectores de números reales. A estos vectores resultantes se les conoce como *embeddings*.

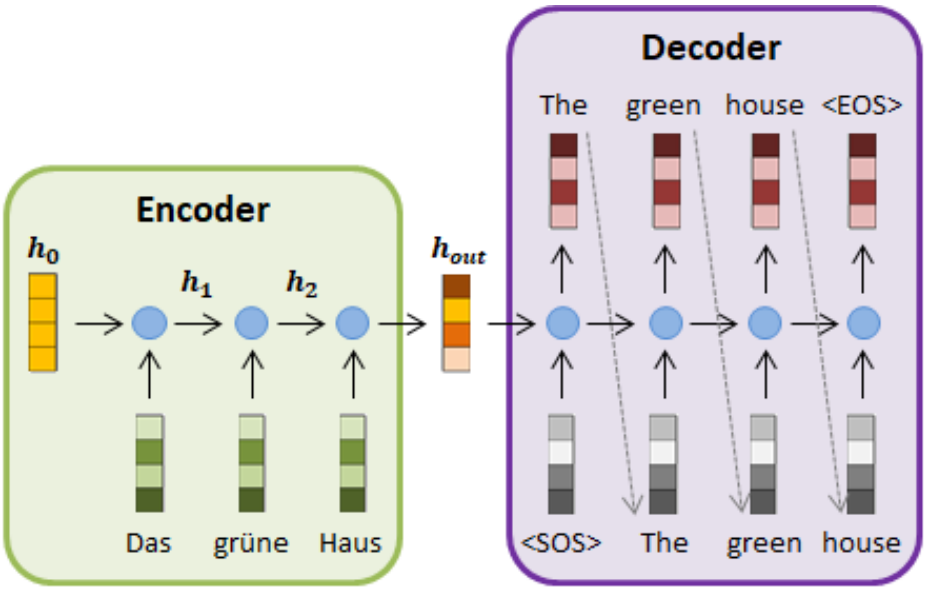


Figura 7: Esquema "*encoder-decoder*" implementado con RNN.

correcto al *decoder*, aún cuando éste ha fallado con su predicción. Gracias a este método se evita la propagación de errores a lo largo de una secuencia.

Esta dinámica de funcionamiento permite a las RNN obtener traducciones de secuencias manteniendo el contexto de la frase. Sin embargo, toda la información extraída por el *encoder* se almacena en un único vector, provocando que las relaciones a larga distancia se pierdan. Esta problemática afecta de igual modo a las LSTM[5] que, pese a que permiten mantener dependencias a largo plazo, persisten en la limitación de emplear un mismo vector h_{out} para almacenarlas.

2.3.2 *Attention*

Tal y como se ha explicado en el apartado 2.3.1, el hecho de disponer de un único vector de contexto supone una gran limitación a la hora de trabajar con secuencias largas. Para acabar con dicha restricción se propuso un nuevo mecanismo conocido como *Attention*[6] [Bahdanau et al., 2015, Luong et al., 2015], cuya implementación mejoró la calidad de los sistemas de MT basados en redes neuronales, logrando superar los resultados de los clásicos algoritmos de SMT.

[5]La red *Long short-term memory* (LSTM) es un tipo de RNN capaz de discernir y almacenar las partes más importantes e informativas de una secuencia. Es decir, en vez de siempre "olvidar" el inicio de las secuencias, su diseño permite preservar la información más relevante y eliminar la menos prioritaria.

[6]El mecanismo de *Attention* fue originalmente planteado para tareas de *Computer Vision* [Larochelle and Hinton, 2010]. Éste se basaba en la idea de focalizar la atención en distintas partes de una imagen con la finalidad de acumular información útil para su clasificación.

El mecanismo de *Attention* permite a los modelos atender a las partes más relevantes de las secuencias de entrada. A gran escala, gracias a este planteamiento, el *decoder* puede acceder a todos los vectores *hidden state* del *encoder* y, por consiguiente, aumenta la capacidad de mantener el contexto en secuencias más largas. El procedimiento del mecanismo de *Attention* descrito en Bahdanau et al. [2015] es el siguiente (figura 8):

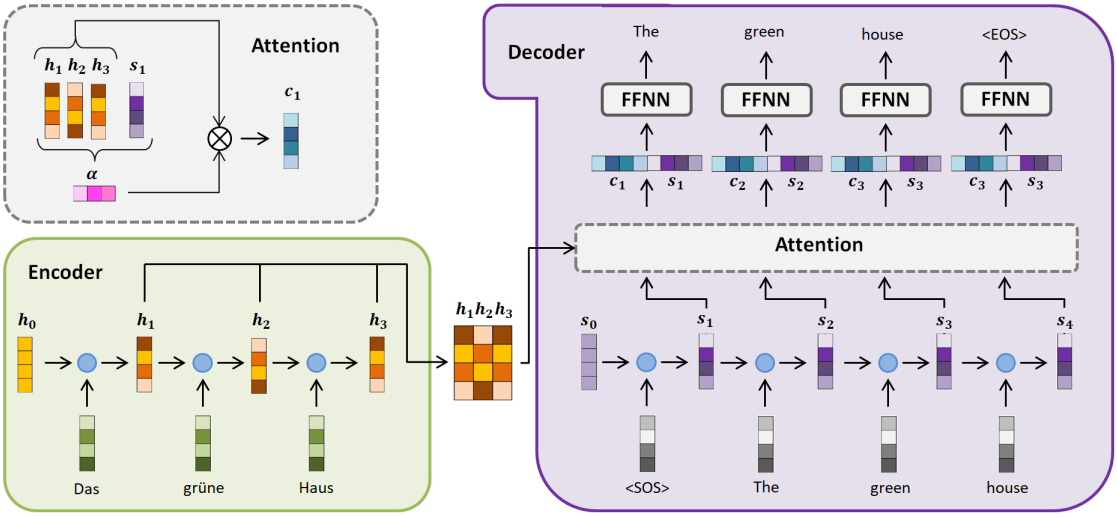


Figura 8: Esquema "*encoder-decoder*" implementado con RNN utilizando el mecanismo de Atención.

1. A diferencia de las RNN tradicionales, la etapa codificadora va almacenando los *hidden state* de cada instante de tiempo. Una vez el *Encoder* a recorrido toda la sequencia d'entrada, entrega los *hidden state* de cada instante de tiempo al *Decoder*.

2. Sea s_n el n-ésimo *hidden state* del *decoder*, y $h_1, ..., h_M$ los *hidden state* del *encoder*. Mediante una función f_{att}[7] se calculan los *alignment scores* entre el actual *hidden state* del *decoder* y cada *hidden state* del *encoder*. Los *alignment scores* (a_{nm}) para s_n son:

$$a_{nm} = f_{att}(s_n, h_m) \quad \text{con} \quad m \in [1, M] \tag{4}$$

3. Se aplica la función *softmax*[8] a los *alignment scores* para obtener los pesos (α_{nm}):

$$\alpha_{nm} = softmax(a_{nm}) \tag{5}$$

[7]Bahdanau et al. [2015] proponen la utilización de un perceptrón multicapa como f_{att}. De este modo, se logra parametrizar f_{att} y, por ende, puede ser entrenada conjuntamente con el modelo.

[8]La función *softmax* toma un vector $\boldsymbol{x} \in R^k$ y devuelve otro vector $\boldsymbol{y} \in R^k$ de valores reales dentro del rango [0,1] cuya suma es igual 1. Comúnmente es empleada para conversión de valores a distribuciones de probabilidad.

4. Se calcula el vector de contexto multiplicando los *hidden states* del *encoder* (h_m) por los pesos correspondientes:

$$c_n = \sum_{m=1}^{M} \alpha_{nm} \cdot h_m \tag{6}$$

5. Se concatena el vector de atención resultante (c_n) con el vector *hidden state* del *decoder* (s_n) y se introducen en una "*feed-forward neural network*" (entrenada conjuntamente con el modelo) que actúa como un clasificador para dar las puntuaciones de probabilidad de la palabra a predecir.

6. Se repiten el mismo procedimiento (del paso 2 al 5) para las siguientes iteraciones.

En la figura 9 se muestra un ejemplo de los pesos de la matriz de atención.

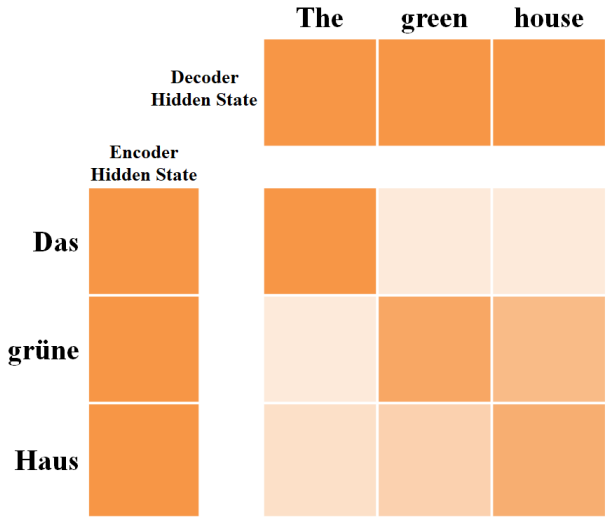


Figura 9: Matriz de alineación de palabras. En ella se utiliza una escala de intensidad de color para representar el porcentaje de atención, que destina el *decoder* a cada uno de los *hidden state* del *encoder*.

Los modelos basados en RNN con *Attention* han sido muy utilizados en tareas *Seq2Seq*. Sin embargo, su naturaleza secuencial reduce la posibilidad de paralelización, perjudicando considerablemente los tiempos de entrenamiento e inferencia.

2.3.3 *Transformer*

Como sus antecesores, el *Transformer* [Vaswani et al., 2017] es un modelo *Seq2Seq* que adopta una arquitectura *"encoder-decoder"*. El *encoder* consiste en un número N de capas de codificación que procesan el *encoder input* de manera iterativa, una tras otra. Análogamente, aunque con algunas variaciones (detalladas en el apartado *decoder*), el *decoder* consiste en una pila de M capas decodificadoras que además de procesar el *decoder input* lo relacionan con la información entregada por el *Encoder*. El potencial del *Transformer* reside en la implementación del mecanismo *Self-Attention* en cada una de las capas de *encoder* y *decoder*. Contrariamente a los modelos RNN, el *Transformer* no necesita procesar los *tokens* de entrada de manera recurrente, ya que el propio mecanismo de *Self-Attention* es capaz de procesar todas las posiciones en la secuencia de entrada a la vez, permitiendo la paralelización en entrenamiento.

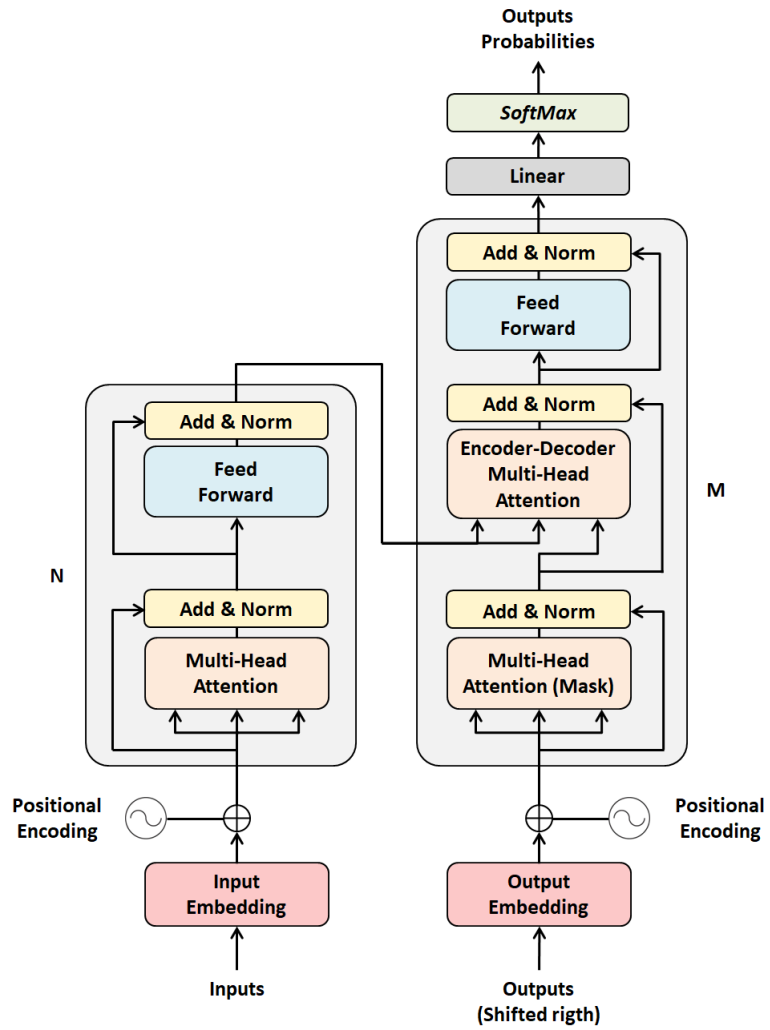


Figura 10: Estructura del *Transformer*.

Self-Attention

El mecanismo de *Self-Attention* (Figura 11) aprende a extraer relaciones entre tokens de una misma secuencia. Para ello, calcula un promedio ponderado de la relación de un *token* con cada uno de los otros *token* de la misma secuencia. Formalmente, el procedimiento es el siguiente:

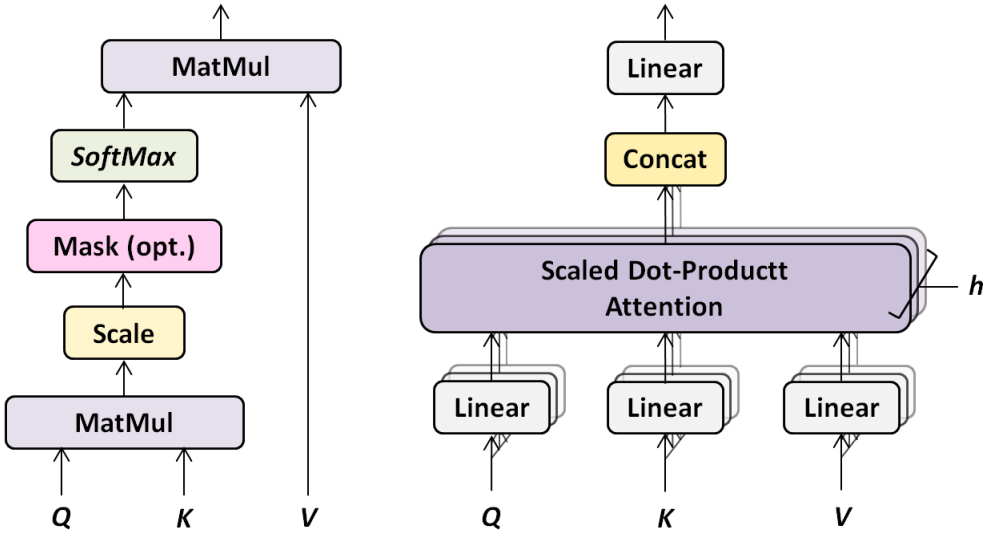


Figura 11: (Izquierda) Esquema de *Scaled Dot-Product Self-Attention*. (Derecha) Esquema de *Multi-Head Attention*.

Dada una secuencia de entrada con un número n de *tokens* de dimensión d_{model} [9], se define la matriz de entrada como $X = (x_1, ..., x_n) \in \mathbb{R}^{n \times d_{model}}$ donde $x_i \in \mathbb{R}^{d_{model}}$ son los vectores fila de *embeddings*.

1. Se proyecta la matriz de entrada X mediante otras tres matrices $W_Q \in \mathbb{R}^{d_{model} \times d_q}$, $W_K \in \mathbb{R}^{d_{model} \times d_k}$, y $W_V \in \mathbb{R}^{d_{model} \times d_v}$ que se entrenan conjuntamente con el modelo. De estas proyecciones se extraen las representaciones Q (*query*), K (*key*), y V (*value*) con $d_k = d_q$. Así, las expresiones para calcular Q, K, y V son

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V.$$

2. Se calcula la matriz de atención A como

$$A = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) \in \mathbb{R}^{n \times n} \tag{7}$$

[9] d_{model} es la dimensión de los vectores de *embeddings* $\boldsymbol{x}_i$.

Cada posición A_{ij} de esta matriz posee un valor entre 0 y 1 llamado peso, que puede ser interpretado como el grado de relación que existe entre los *tokens* i y j. A mayor peso, mayor relación.

3. Finalmente, se obtiene la matriz de contexto Z:

$$Z = A \cdot V \tag{8}$$

con $Z = (z_1, ..., z_n) \in \mathbb{R}^{n \times d_v}$ donde $z_i \in \mathbb{R}^{d_v}$ es el vector fila de contexto asociado al vector x_i. Juntando las expresiones 7 y 8 se define la expresión general:

$$Z = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V \in \mathbb{R}^{n \times d_v} \tag{9}$$

En el *paper* se propone explotar más este mecanismo mediante la paralelización de éste mediante capas independientes, conocidas como *heads* de atención. Esto mejora el rendimiento de la capa de atención ya que permite que el modelo se focalice en distintos tipos de relaciones dentro de un misma secuencia. Al inicializar las matrices de pesos W_Q, W_K, y W_V de manera aleatoria, cada *head* proyecta la matriz de *embeddings* a sub-espacios distintos. Por consiguiente, en cada uno de ellos se aprenderán relaciones y patrones diferentes.[10]

Si se considera el uso de la *Multi-Head Attention* se definen las siguientes expresiones generales:

$$Q^i = XW_Q^i \quad K^i = XW_K^i \quad V^i = XW_V^i \quad \text{con } i \in [1, ..., h]$$

$$Z^i = softmax\left(\frac{Q^i(K^i)^T}{\sqrt{d_k}}\right) \cdot V^i \in \mathbb{R}^{n \times d_v} \tag{10}$$

Finalmente se concatenan todas las matrices Z^i, formando $Z_{concat} = (Z_1, ..., Z_h) \in \mathbb{R}^{n \times h \cdot d_v}$, y se proyectan con la matriz $W_0 \in \mathbb{R}^{h \cdot d_v \times d_{model}}$.

$$Z = (Z_1, ..., Z_h) \cdot W_0 \tag{11}$$

Donde $Z = (z_1, ..., z_n) \in \mathbb{R}^{n \times d_{model}}$ es la matriz de los vectores de contexto.

Cabe señalar que las dimensiones de la matriz de entrada X y de la matriz de salida Z

[10]Por ejemplo, es posible que en un *head* se aprenda la conexión entre los sustantivos y los verbos, mientras que en otro se consiga discernir entre el objeto y el sujeto de una oración.

coinciden. Es decir, el uso de uno o varios *heads* de *Self-Attention* mantiene intactas las dimensiones de la secuencia ($d_v = {}^{d_{model}}/_{h}$). De este hecho se deduce que la implementación con un solo *head* implica $d_v = d_{model}$.

Para comprender el origen de este estudio, es imprescindible entender que, fruto del producto entre *queries* y *keys* (ecuación 7), la aplicación de la *Self-Attention* provoca que el modelo adquiera una complejidad de cómputo de $O(d \cdot n^2)$ En otras palabras, el hecho de que los n *tokens* deban atender a n *tokens* implica el cálculo de $d \cdot n^2$ multiplicaciones, escalando de manera cuadrática respecto a la longitud de secuencia. Esta complejidad cuadrática conlleva una gran limitación de eficiencia cuando se trabaja con secuencias largas.

Feed-forward Layers

Un componente importantes dentro del *Transformer* son las capas de *Feed-Forward Neural Network*. Éstas se componen de dos transformaciones lineales conectadas a través de una función de activación ReLU[11]. Léase, la expresión es la siguiente:

$$Z = F_2(ReLU(F_1(X))) \tag{12}$$

donde F_1, F_2 son funciones de la forma $W_x + b$, y X es la entrada del módulo.

Encoder

El *Encoder* está formado por N capas idénticas. Cada una de ellas, a su vez, se compone de dos sub-capas distintas: (i) El mecanismo de *Multi-Head Attention* y, (ii) una *Feed-Forward Neural Network*. Además, se utilizan unas conexiones residuales que conectan la entrada de cada sub-capa a una capa de normalización. Así, se puede expresar la salida de cada sub-capa como $Out = LayerNorm(x + F(x))$, donde $F(x)$ es la función realizada por la propia sub-capa.

Teniendo en cuenta que en cada capa *Encoder* se añaden las relaciones halladas por la *Multi-Head Attention* a los *tokens* de entrada, a la salida de la etapa de codificación se obtiene una matriz con los *tokens* contextualizados. Por este motivo, a gran escala se puede entender la función del codificador como un contextualizador de los *tokens* de entrada.

[11] La función *Rectified Linear Unit* (ReLU) es una función de activación muy utilizada en el campo del *Deep Learning*.

Decoder

La etapa de decodificación se constituye por M capas iguales. Sin embargo, además de las dos sub-capas presentes en cada *encoder*, el *decoder* introduce una tercera sub-capa que realiza un mecanismo de *Multi-Head Attention* usando la salida de la etapa *encoder*. Concretamente, la *Encoder-Decoder Multi-Head Attention* utiliza el *encoder output* para el cálculo de *keys* y *values*, y la salida de la primera sub-capa del *decoder* para calcular las *queries*. Esto permite al *decoder* disponer de la información extraída por el *encoder* al predecir la secuencia resultante.

Se puede interpretar la función del decodificador como un "generador"[12] de texto condicionado. En otras palabras, al introducir un *token* <*SOS*>, el *decoder* generar texto influido por la información extraída por el codificador.

En la fase de entrenamiento el *decoder*, *concretamente su self-Attention*, no debe tener acceso a posiciones que aún no se han predicho. De lo contrario, el *decoder* se entrenaría con una información de la que no se dispone en la práctica. Para evitar el uso de estas relaciones inválidas, se aplica una máscara diagonal justo antes de realizar la $softmax$ de la ecuación 7 (figura 11).

Componentes *Linear* y *softmax*

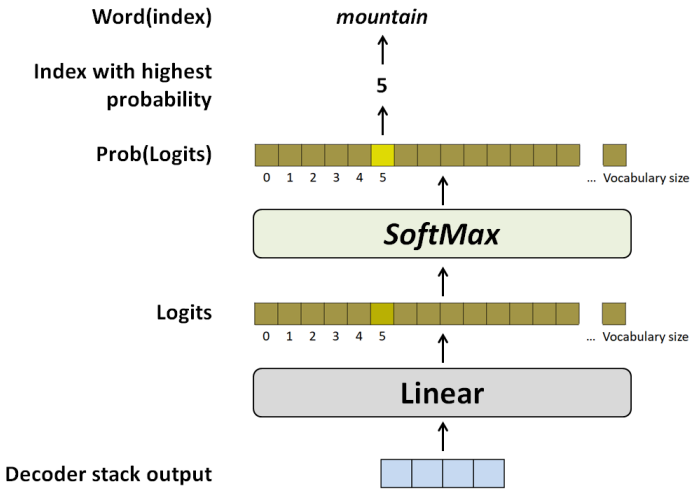


Figura 12: Esquema del procedimiento realizado en los bloques *Linear* y *Softmax*.

Estas dos últimas capas son las encargadas de transformar el *output* de la salida de las capas del *Decoder* en una distribución de probabilidad. La capa *linear* está compuesta por

[12]El *decoder* no genera el texto *per se*, sino que calcula las probabilidades con las que se debe imprimir una secuencia u otra.

una red neuronal que proyecta el vector de la salida del *decoder* contra el vector de *logits*. Cada posición de este vector corresponde a una palabra conocida en el idioma de destino, cuantas más palabras conozca nuestro modelo, mayor será la longitud de este vector. La salida de la capa *linear* es un vector cuyas posiciones representan la "puntuación" de cada palabra del vocabulario. Mediante la aplicación de la función $softmax$ se convierten esas puntuaciones en probabilidades entre 0 y 1.

Finalmente, es habitual utilizar la técnica *greedy decoding*. Consiste en elegir el *token* cuyo índice sea el de mayor probabilidad (figura 12). Sin embargo, existen otras opciones para elegir la mejor traducción, entre ellas el algoritmo de *beam search*. Éste mantiene varias hipótesis de salida para una misma traducción hasta que, al finalizar, elije la de mayor probabilidad. Por ejemplo, si en la primera predicción el algoritmo otorga probabilidades parecidas a las palabras "yo" y "él", continuará ambas predicciones por separado sin descartar ninguna directamente.

Positional encoding

Por su propia naturaleza estructural, el *Transformer* no contiene ningún tipo de recurrencia. Con el fin de permitir al modelo hacer uso del orden de la secuencia, se introducen dos codificadores de posición antes de la entrada del *encoder* y del *decoder*. Los *Positional encodings* añaden información a los *tokens* acerca de su posición relativa y absoluta dentro de la secuencia. Vaswani et al. [2017] eligen las funciones seno y coseno para realizar la codificación:

$$PE_{(pos,2i)} = sin(pos/10000^{\frac{2i}{d}}) \tag{13}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{\frac{2i}{d}}) \tag{14}$$

donde *pos* es la posición y i es la dimensión. Es decir, cada dimensión del *position encoding* corresponde a una sinusoide, cuya longitud de onda forma una progresión geométrica entre 2π y $10000 \cdot 2\pi$.

2.4 Métricas de evaluación

En este apartado se exponen las dos métricas más comúnmente empleadas para evaluar el funcionamiento de los modelos de ASR, NMT y ST.

2.4.1 *Word Error Rate*

La WER, del inglés *Word Error Rate*, es una de las métricas más utilizadas para evaluar modelos de reconocimiento del habla. A menor WER mejor será el funcionamiento del modelo examinado. Formalmente, se define la WER entre la frase generada por el sistema y la de referencia como:

$$WER = \frac{Substituciones + Borrados + Inserciones}{N} \tag{15}$$

donde

- *Substituciones*: Número de veces donde el modelo ha predicho una palabra incorrecta.
- *Borrados*: Número de Ocasiones donde el modelo no ha predicho nada cuando debería haber predicho una palabra.
- *Inserciones*: Número de veces donde el modelo ha predicho una palabra cuando debería no haber predicho nada.
- N: Número de palabras de la frase de referencia.

2.4.2 *Bilingual Evaluation Understudy score*

Bilingual Evaluation Understudy score [Papineni et al., 2002], que responde a las siglas de BLEU, es una medida para evaluar la calidad de las traducciones realizadas por sistemas de traducción automática. Se entiende como calidad el nivel de semejanza y/o proximidad que existe entre la predicción y la frase de referencia, generalmente originada por un humano. Una mayor puntuación indica un mayor grado de similitud con la traducción de referencia y, por tanto, una mejor traducción. Además, es habitual establecer más de una frase de referencia para una misma traducción, proporcionando más robustez a la medida frente a la multitud de estilos de expresión.

La BLEU calcula la precisión en *n-grams*[13] entre la traducción del sistema y la de referencia. La precisión viene dada como:

$$P = \frac{n\text{-}grams\ comunes}{n\text{-}grams\ predicción} \tag{16}$$

[13]Los *n-grams* son una subsecuencia de n elementos de una secuencia mayor.

Sin embargo, como la expresión anterior proporciona mejores resultados a predicciones cortas, se introduce una penalización por brevedad. Ésta se aplica a traducciones cuya longitud (l_{pred}) es menor que la de la frase de referencia (l_{ref}). Formalmente, se define la penalización como:

$$PB = \begin{cases} 1 & si \quad l_{pred} > l_{ref} \\ e^{1-\frac{l_{ref}}{l_{pred}}} & resto \end{cases} \tag{17}$$

Finalmente, la BLEU se define como la media:

$$BLEU = PB \cdot exp\left(\sum_{n=1}^{N} \frac{1}{N} log(P_n)\right) \tag{18}$$

3 State of the art

En esta sección se presentará la situación actual en el campo de *Speech-to-Text Translation*, detallando cuál es el modelo de referencia utilizado. Además, se expondrán algunas arquitecturas (*Efficient Transformers*) que presentan alternativas al mecanismo de *Self-Attention*.

3.1 *Speech-to-Text Transformer*

El surgimiento del *Transformer* [Vaswani et al., 2017] revolucionó el campo del NMT. Además, sus buenos resultados incentivaron nuevos planteamientos, que tuvieron gran impacto en los modelos *Seq2Seq* y en todo el campo del NLP. En el año 2018, se propuso por primera vez el uso del *Transformer* como modelo *End2End* para ST [Vila et al., 2018], convirtiéndose en el actual modelo de referencia. Sin embargo, trabajar con audio supone lidiar con secuencias de entrada de un orden de magnitud mayor que con texto. Por consiguiente, el *Vanilla Transformer*[14], que posee una complejidad computacional de $O(n^2)$ respecto a la longitud de secuencia, resulta extremadamente ineficiente para procesar secuencias largas como los datos de *Speech*.

Con el fin de solucionar este inconveniente, algunos modelos abordan el problema colapsando vectores adyacentes de una manera fija. Es decir, comprimen la secuencia de vectores d'entrada (normalmente en un factor de 4) usando capas convolucionales. Este es el caso de Di Gangi et al. [2019], que propusieron una adaptación del *Transformer* para ST con varias capas previas al *Encoder*. En concreto, su modelo implementa un conjunto de capas convolucionales 2D, para disminuir la longitud de la secuencia, y varias capas de *Self-Attention* 2D, permitiendo modelar las dependencias de largo alcance. Además, también introduce una penalización por distancia, sesgando las capas de atención al contexto local. En la misma linea, pero únicamente con capas convolucionales 1D, Wang et al. [2020a] propusieron una adaptación del *Transformer* para *Speech* que se consolidó como el actual modelo de referencia en ST (*S2T Transformer* en adelante).

Por un lado, estos planteamientos de reducción de la longitud de secuencia consiguen disminuir la redundancia de la entrada. Sin embargo, ignoran cómo está distribuida la información lingüística y fonética de las señales de audio, al otorgar el mismo peso a todas sus características. En consecuencia, las características relevantes se penalizan y

[14]El adjetivo *Vanilla* es utilizado con frecuencia para indicar que no se ha añadido ningún complemento al modelo de base. Léase, *Vanilla Transformer* es el modelo originalmente propuesto por Vaswani et al. [2017].

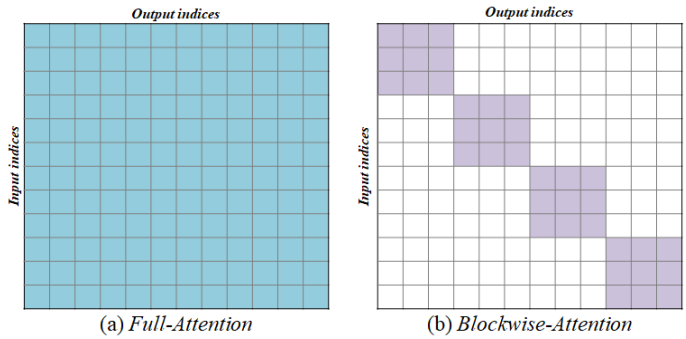
se consideran igualmente importantes que las irrelevantes, lo que podría resultar en una pérdida de información.

3.2 *Efficient Transformers*

El *Transformer* [Vaswani et al., 2017] se ha establecido como el modelo de referencia en muchas áreas de *Deep Learning*. Sin embargo, como se ha mencionado anteriormente, debido al mecanismo de *Self-Attention*, el modelo presenta una complejidad cuadrática respecto a la longitud de la secuencia de entrada. Esto implica una ineficiencia, muchas veces limitante, frente al procesamiento de secuencias largas, como es el caso de las secuencias de audio.

Tal y como se ha visto en el apartado 3.1, los modelos más utilizados en la actualidad conservan la estructura del *Vanilla Transformer* y se focalizan en la reducción de las dimensiones de la secuencia de entrada. Sin embargo, existen otros planteamientos centrados en modificar el diseño base del *Transformer* para reducir su complejidad cuadrática. Estos modelos, conocidos como *Efficient Transformers*, se pueden diferenciar entre las siguientes categorías:

- **Patrones fijos**: Fueron las primeras modificaciones del mecanismo de *Self-Attention*. Consisten en limitar la *Full-Attention* (figura 13a) mediante la utilización de patrones fijos.
  - **Patrones de intervalo**. Consiste en la utilización de patrones que atienden únicamente a intervalos fijos. Este tipo de patrones se pueden encontrar en modelos como el "*Sparse Transformer*" [Child et al., 2019] y el *Longformer* [Beltagy et al., 2020], con la implementación de la *sliding window* (figura 16a).



(a) *Full-Attention* (b) *Blockwise-Attention*

Figura 13: Patrones de la matriz de *Self-Attention*.

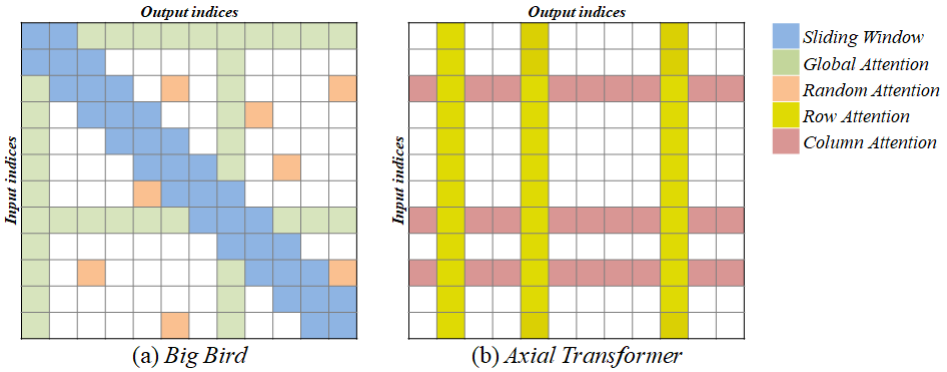
  - **Patrones de bloque**. Focalizan la atención entre los elementos más próximos mediante la aplicación de patrones fijos más reducidos. Por consiguiente, sólo se permite la generación de relaciones locales. Dos ejemplos de esta implementación son el *Blockwise* [Qiu et al., 2020] (figura 13b) y la *Local-Attention* implementada en el *Image Transformer* [Parmar et al., 2018]. Estos patrones permiten reducir la complejidad del modelo de $O(n^2)$ a $O(b^2)$, donde b es el tamaño del bloque y se cumple que $b << n$. Este tipo de patrones han sido la base para la construcción de patrones más complejos.

- **Patrones de Memoria**: Otro método destacado es aprovechar un patrón de memoria lateral, que puede acceder a varios tokens a la vez. Una forma común es la de la memoria global, o *Global Attention* (figura 14), que permite a algunos *tokens* acceder a todos los *tokens* de la secuencia. Los *tokens* globales actúan como una memoria que recopila información de la secuencia de entrada. Este tipo de patrones se encuentran en modelos como el ETC [Ainslie et al., 2020] o el *Longformer*. Gracias al uso de un número reducido de *tokens* de "memoria", el modelo es capaz de realizar algunas relaciones generales.

- **Combinación de patrones**: La idea principal es la combinación de distintos patrones para permitir al modelo obtener relaciones más variadas. Por ejemplo, el *Big Bird* [Zaheer et al., 2020] (figura 14a) combina la *sliding window*, la *Global-Attention* (figura 14) y una *Random-Attention*[15]. Del mismo modo, el "*Axial Transformer*" [Ho et al., 2019] (figura 14b) utiliza distintos patrones en forma de fila orientados en los distintos ejes. En resumen, la superposición de patrones disminuye la complejidad del modelo, de la misma manera que lo hacen los patrones fijos. La diferencia es que la combinación de varios patrones aumenta la capacidad de extraer relaciones más globales.

- **Clasificación de *tokens***: Estos modelos son capaces de clasificar los *tokens* en grupos. Dicha clasificación es posible mediante una función de similitud que es entrenada de manera *End2End* conjuntamente con el modelo. Al tener los *tokens* ordenados y/o clasificados en distintos grupos, estos modelos proponen calcular únicamente el producto entre *queries* y *keys* similares. Siguiendo esta línea encontramos (1) el *Reformer* [Kitaev et al., 2020] que propone utilizar una función de *hash* como función de similitud, y (2), el "*Routing Transformer*" [Roy et al., 2021], que agrupa los *tokens* mediante el método *k-means*[16].

[15] La *Random-Attention* permite al modelo atender a posiciones aleatorias de la matriz de atención.
[16] El algoritmo de agrupación *K-means* clasifica n *tokens* en k grupos. Esta clasificación se realiza minimizando la suma de distancias entre cada *token* y el valor medio de cada grupo.



(a) *Big Bird* (b) *Axial Transformer*

Figura 14: Superposición de patrones fijos en la matriz de *Self-Attention*.

- **Métodos de aproximación *Low-Rank***: Algunos modelos mejoran la eficiencia mediante la aplicación de aproximaciones *low-rank* de la matriz de *Self-Attention*. Estos métodos asumen que la matriz $n \times n$ de *Self-Attention* sigue una estructura *low-rank*. El *Linformer* [Wang et al., 2020b] es un claro ejemplo de este planteamiento, ya que éste proyecta la longitud de las *keys* y los *values* a una representación dimensionalmente menor ($n \rightarrow k$). Sean E^i, $F^i \in \mathbb{R}^{n \times k}$ dos matrices de proyección lineal, el cómputo de la *Low-rank Self-Attention* es:

$$Q^i = XW_Q^i \quad K^i = XW_K^i \quad V^i = XW_V^i \quad \text{con } i \in [1, ..., h]$$

$$Z^i = \overbrace{softmax\left(\frac{Q^i(E^iK^i)}{\sqrt{d_k}}\right)}^{\overline{P}:n \times k} \cdot \overbrace{F^iV^i}^{k \times d_v} \tag{19}$$

  Gracias a la aproximación *Low-Rank* ($\overline{P}$) de la matriz de *Self-Attention*, este método logra una complejidad lineal del modelo. Recientemente se ha propuesto el *Speechformer* [Papi et al., 2021], que adapta los conceptos introducidos por el *Linformer* para la creación de un modelo pensado para ST.

- ***Kernels***: Algunos *Efficient Transformers* proponen afrontar el problema de la complejidad de la *Self-Attention* mediante la Kernalización. El origen del planteamiento nace de la propia definición de la *Self-Attention*.

$$Z = \overbrace{softmax\left(\frac{\overbrace{Q}^{n \times d}\overbrace{(K)^T}^{d \times n}}{\sqrt{d_k}}\right)}^{A:n \times n} \cdot \overbrace{V}^{n \times d_v} \tag{20}$$

Al no poder descomponerse la función *softmax*, no es posible realizar primero el cómputo entre K^T y V, por lo que se requiere calcular antes la matriz de atención (A). La idea principal reside en utilizar *kernels* [Katharopoulos et al., 2020, Choromanski et al., 2020a] para reescribir matemáticamente la expresión de la *Self-Attention* y, de este modo, evitar calcular explícitamente la matriz $n \times n$ (figura 15). Mediante el uso de *kernels*, se formula una expresión general de la matriz $A \in \mathbb{R}^{n \times n}$:

$$A(i,j) = \kappa(\boldsymbol{q}_i, \boldsymbol{k}_j^T) \qquad \text{con} \quad \kappa(x,y) = \mathbb{E}_w[\phi(x) \cdot \phi(y)] \tag{21}$$

siendo:

- $\boldsymbol{q}_i / \boldsymbol{k}_j$ los vectores *i-ésimo/j-ésimo* de *queries/keys* de Q/K.
- $\kappa : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_+$ la función *kernel*.
- $\phi : \mathbb{R}^d \to \mathbb{R}_+^r$ $(r > 0)$ una función cualquiera utilizada como aproximación. Recibe el nombre de *Random feature map*.

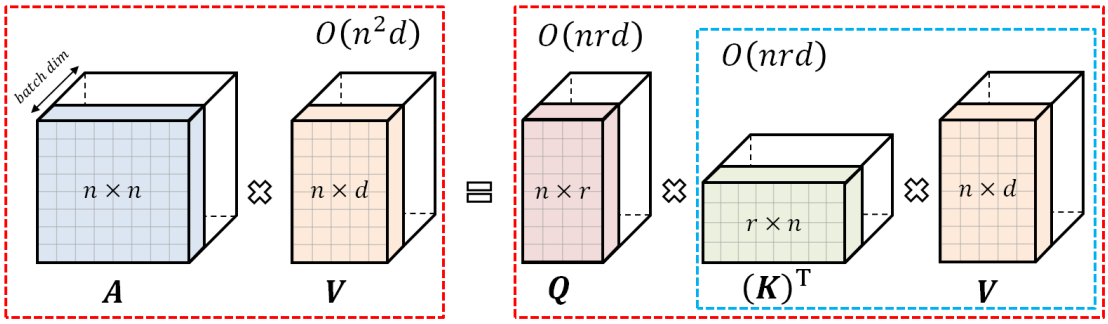


Figura 15: *Self-Attention* mediante *kernel*.

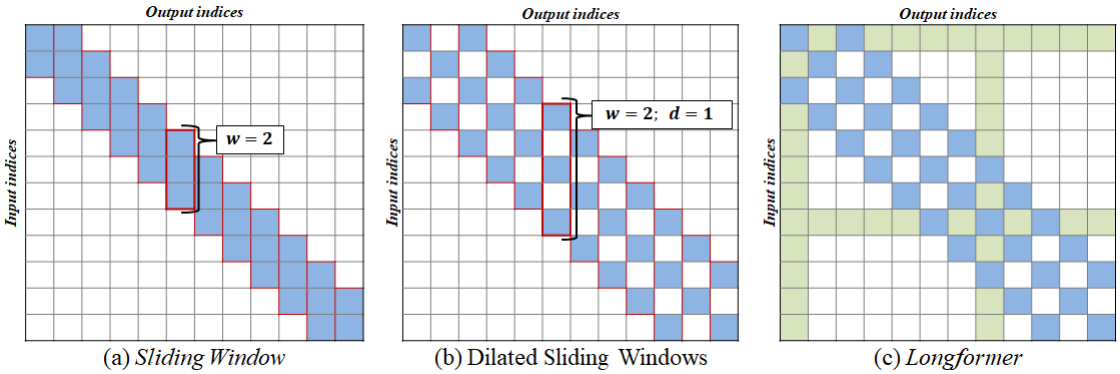
Los modelos que implementan este planteamiento difieren en la elección de la función ϕ, ya que de ésta depende la calidad de la aproximación. Algunos ejemplos son el *Performer* [Choromanski et al., 2020b] y el R_{FA} [Peng et al., 2021], proponiendo dos configuraciones distintas para la función ϕ.

3.2.1 Modelos en profundidad

En este apartado se profundizará en los modelos de *Efficient Transformers* más relevantes para el desarrollo de este estudio.

Longformer

El *Longformer* [Beltagy et al., 2020] forma parte del grupo de *Efficient Transformers* basados en la combinación de patrones de atención. Mediante la aplicación de distintos patrones, el modelo ahorra el cómputo de *Attention* entre algunos *tokens*, consiguiendo reducir la complejidad de cuadrática a lineal ($O(n)$). En concreto, el *Longformer* implementa los siguientes patrones:



(a) *Sliding Window* (b) Dilated Sliding Windows (c) *Longformer*

Figura 16: Patrones de atención propuestos por el *Longformer*.

- ***Sliding Window* (Figura 16a):** Componente principal del modelo, cuya función es mantener el contexto local. Consiste en una ventana de longitud fija centrada en cada *token*. El planteamiento de la *Sliding window* es sencillo: Dada una ventana de longitud w cada *token* puede atender a los $\frac{1}{2}w$ *tokens* de su alrededor. Por consiguiente, la complejidad de este patrón es $O(n \times w)$, escalando de manera lineal respecto a la longitud de la entrada n. Al emplear varias capas de este patrón, se consigue un efecto similar a los de las CNNs[17], donde las últimas capas tienen de una gran parte de la secuencia. Por ejemplo, en un *Transformer* con l capas, el tamaño del campo receptivo[18] en la última capa es $w \times l$.

- ***Dilated Sliding Window* (Figura 16b):** Con el objetivo de incrementar el campo receptivo sin aumentar la complejidad. El *paper* propone esta variante de la *Sliding Window*. Consiste en "dilatar" la ventana dejando d espacios entre posiciones atendidas. De este modo, el tamaño del campo receptivo en la capa final es $l \times w \times d$.

[17]Las Redes Neuronales convolucionales (CNNs) [Fukushima et al., 1983] son un tipo de NN que consisten en múltiples capas de filtros convolucionales de una o más dimensiones.

[18]El término "Campo receptivo" es una analogía al campo receptivo de una neurona. Éste se refiere a la región del espacio (posiciones de la matriz de atención), en la cual la presencia de un estímulo repercute al cómputo final.

- ***Symmetric Global Attention:*** Permite a algunos *tokens* preseleccionados atender al resto de tokens de la secuencia, y, del mismo modo, todos los *tokens* pueden atender a éstos. Gracias a la implementación de la *Global Attention*, se puede sesgar la atención del modelo a conveniencia y, por lo tanto, adaptar el modelo a posibles tareas de NLP que requieran de *tokens* especiales.

En la línea de estudio, Alastruey et al. [2021] proponen una arquitectura inspirada en el *Longformer* para *Direct*-ST y ASR. En concreto, substituyen el módulo de atención del *Vanilla Encoder-Transformer* por la *Sliding Window* del *Longformer* y prescinden de realizar *downsampling* a la secuencia de entrada. Además, añaden una capa convolucional 1D a la salida del *encoder* para facilitar el alineamiento[19] en la *Encoder-Decoder Attention*.

Speechformer

El *Speechformer* [Papi et al., 2021] propone una variación del *Vanilla Transformer* inspirada en el *Linformer* [Wang et al., 2020b] y enfocada a tareas de *"Speech Translation"*. Su arquitectura está formada por los siguientes componentes (Figura 17):

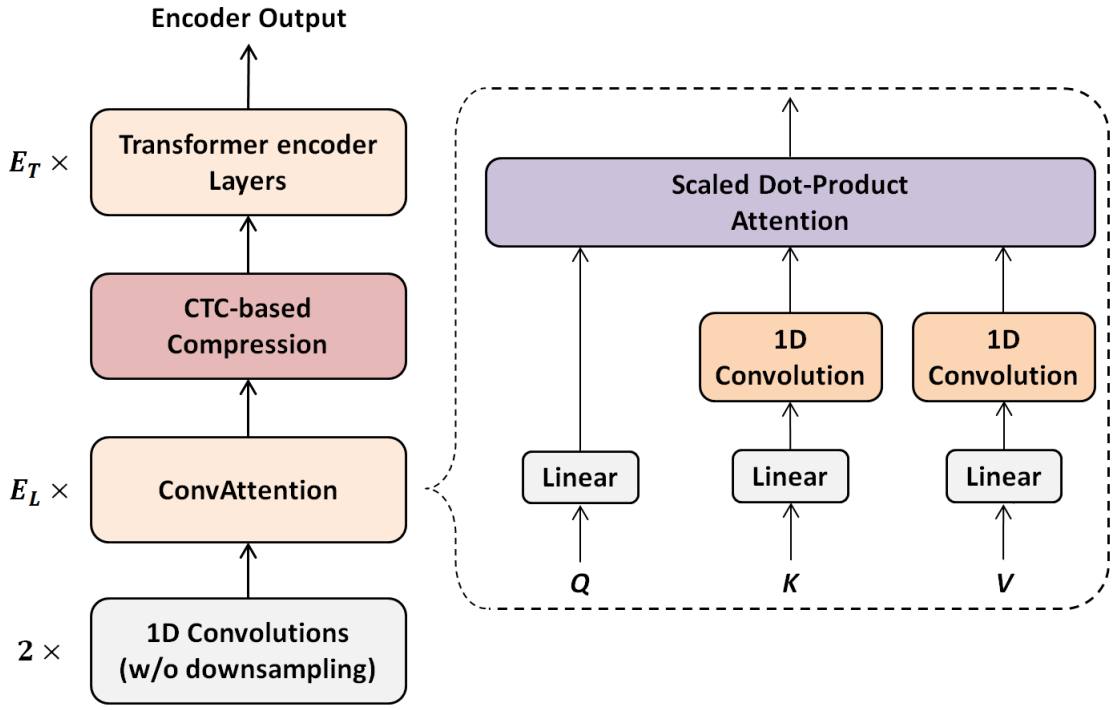


Figura 17: Arquitectura del *Speechformer* y del módulo de *ConvAttention*.

[19]En ST, como en todas las tareas de traducción, la entrada y la salida se alinean para realizar el cómputo de la *Encoder-Decoder Self-Attention*. Sin embargo, en ST, mientras el *output* es texto, el *input* es un espectrograma, generándose una discrepancia en sus longitudes.

- ***1D Convolutions*:** El *Speechformer* mantiene las convoluciones del actual modelo de referencia, sin embargo, éstas no producen ningún tipo de *down-sampling*. Mediante la implementación de las capas convolucionales, el modelo es capaz de aprender mejores representaciones del *input*.
- ***ConvAttention*:** Este módulo permite procesar los datos directamente del espectrograma de mel sin aplicar *down-sampling*. La *ConvAttención* resuelve los problemas[20] que presenta el *Linformer* frente a secuencias de longitud variable, como acostumbran a ser las del audio. Para ello, sustituye las proyecciones lineales de la atención del *Linformer* por una capa convolucional unidimensional. Por consiguiente, se consigue que las longitudes de las secuencias introducidas en la *"Scaled Dot-product Attention"* dependan del factor de compresión (χ) de la capa convolucional. Léase, si se define n como la longitud de K y V, la longitud de la salida de la convolución es $\frac{n}{\chi}$. Gracias a dicha reducción, la complejidad de la *ConvAttention* resta como $O((\frac{n}{\chi})^2)$. Además, dado que la capa convolucional no aplica a Q, las dimensiones a la salida de la *ConvAttention* permanecen iguales a las de la entrada.
- ***CTC Compression* [Gaido et al., 2021]:** Con el fin de reducir la redundancia y las partes poco informativas de las secuencias, el *Speechformer* introduce un mecanismo de compresión preentrenado con la *CTC Loss* [Graves et al., 2006]. Este tipo de compresión es muy útil en tareas donde realizar alineamiento entre secuencias resulta muy complejo. La *CTC Compression* genera una predicción en cada instante de tiempo (n_i) y luego colapsa las predicciones consecutivas que son iguales (Figura 18). Mediante esta compresión, la secuencia de audio es reducida a una dimensión similar a la de su secuencia de texto equivalente, sin que se produzca una pérdida relevante de información. Por consiguiente, la secuencia de salida del módulo de *CTC Compression* puede ser procesado por la *Self-Attention* del *Vanilla Transformer Encoder*.

[20] El *Linformer* proyecta linealmente las *keys* y los *values* dando lugar a proyecciones de longitud fija. Al trabajar con secuencias de longitud variable, como las de ST, se pueden dar dos problemas: (1) Pérdida masiva de la información de una secuencia y, (2), dado que el tamaño de la matriz de proyección es $n \times k$, los *inputs* con una longitud de secuencia $n' < n$ provocan que los gradientes tengan diferentes dimensiones, dando lugar a errores en el entrenamiento.

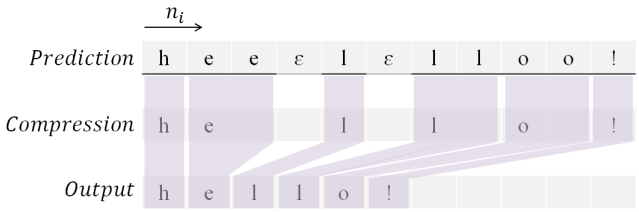


Figura 18: Ejemplo del procedimiento de *CTC compression*.

- ***Vanilla Transformer Encoder*:** Mientras las E_L capas de *ConvAttention* se encargan de discernir el contenido lingüístico del Audio para mejor el rendimiento de la *CTC Compression*, el *Speechformer* también implementa E_T capas de codificación del *Vanilla Transformer*. Éstas son las responsables de extraer las representaciones que son enviadas al *Decoder*.

Performer

El *Performer* [Choromanski et al., 2020b] propone realizar una aproximación matemática de la función *Softmax* mediante *Kernels*. En consecuencia, es posible efectuar primero el cómputo entre *keys* y *values*, ahorrando, de este modo, el cálculo explícito de la matriz $n \times n$ de *Self-Attention* (ecuación 7). Para realizar una buena aproximación de la función *softmax*, el *Performer* propone modelar el *softmax-kernel*[21] mediante la función ϕ:

$$\phi(\boldsymbol{x}) = \frac{h(\boldsymbol{x})}{\sqrt{m}} \left(f_1(\omega_1^\top \boldsymbol{x}), ..., f_1(\omega_m^\top \boldsymbol{x}), ..., f_l(\omega_1^\top \boldsymbol{x}), ..., f_l(\omega_m^\top \boldsymbol{x})\right) \tag{22}$$

con:

- $f_1, ..., f_l : \mathbb{R} \rightarrow \mathbb{R}$ son funciones de modelaje. A mayor número de estas funciones, más concreta podrá ser la aproximación.
- h es una función determinista de la entrada.
- ω_i $(i \in [1, m])$ son vectores deterministas independientes e idénticamente distribuidos en $\mathcal{D} \in \mathcal{P}(\mathbb{R})$.

La expresión propuesta para ϕ intenta ser lo más general posible, permitiendo la implementación de *kernels* ya existentes. Por ejemplo, al escoger $h(\boldsymbol{x}) = 1$, $l = 1$, $f_1 = sgn$ y $\mathcal{D} = \mathcal{N}(0, \boldsymbol{I}_d)$ se obtiene el estimador conocido como PNG-*kernel* [Choromanski et al., 2018].

[21] El *softmax-kernel se define como* $SM(\boldsymbol{x}, \boldsymbol{y}) \overset{\text{def}}{=} exp\left(\boldsymbol{x}^\top \boldsymbol{y}\right)$.

La mayoría de aproximaciones del *softmax-kernel* se basan en el estimador K_{gauss}[22] [Rahimi and Recht, 2008]. Sin embargo, Choromanski et al. [2020b] demuestran que al modelar ϕ, mediante las funciones *seno* y *coseno*, se pueden obtener valores de dimensión negativos que provocan un comportamiento inestable. Ésto aún es más notable en valores de *kernel* próximos a 0 (muy comunes en la matriz $\boldsymbol{A}$ debido a *tokens* con poca relevancia). El *Performer* evita el uso de funciones trigonométricas y propone dos configuraciones distintas de ϕ para aproximar el *Softmax-kernel*:

- $\widehat{\boldsymbol{SM}}_{\boldsymbol{m}}^{+}$**:** Escogiendo $h(\boldsymbol{x}) = exp\left(-\frac{||\boldsymbol{x}||^2}{2}\right)$, $l = 1$, $f_1 = exp$ y $\mathcal{D} = \mathcal{N}(0, \boldsymbol{I}_d)$.
- $\widehat{\boldsymbol{SM}}_{\boldsymbol{m}}^{\boldsymbol{hyp+}}$**:** Con $h(\boldsymbol{x}) = \frac{1}{\sqrt{2}} exp\left(-\frac{||\boldsymbol{x}||^2}{2}\right)$, $l = 2$, $f_1(u) = exp(u)$, $f_2(u) = exp(-u)$ y $\mathcal{D} = \mathcal{N}(0, \boldsymbol{I}_d)$.

Mientras que en los estimadores basados en las funciones $sen()$ y $cos()$ el MSE [23] tiende a infinito para valores de la matriz de atención próximos a 0, las nuevas aproximaciones presentan un comportamiento más estable para todas las regiones. Por consiguiente, para $SM(\boldsymbol{x}, \boldsymbol{y}) \to 0$ consiguen $MSE(\widehat{SM}_m^{+}), MSE(\widehat{SM}_m^{hyp+}) \to 0$.

[22]Nótese, que para configurar el *kernel* Gaussiano K_{gauss} se debe elegir $h(\boldsymbol{x}) = 1$, $l = 2$, $f_1 = sin$, $f_2 = sin$ y $\mathcal{D} = \mathcal{N}(0, \boldsymbol{I}_d)$.

[23]El error cuadrático medio (MSE) de un estimador mide el promedio de los errores al cuadrado. Para un estimador insesgado, como es el caso del propuesto en la ecuación 22, el MSE representa su varianza.

4 Metodología

Tras repasar en los apartados 2 y 3 los conocimientos y las tecnologías de vanguardia fundamentales para el seguimiento de este estudio, a continuación se expondrán los detalles de nuestra propuesta. En este trabajo presentamos el *Multiformer* (Figura 20) que, basado en el *Transformer* [Vaswani et al., 2017], introduce dos nuevas ideas que permiten personalizar la extracción de relaciones realizada en el *Encoder*. Para ello, introducimos un nuevo módulo llamado *Multi-Head Multi-Attention*, que permite la elección de distintos mecanismos de atención en cada *head*. Además, el *Multiformer* habilita la posibilidad de diseñar cada una de las capas del *Encoder* de manera independiente.

En la línea de este estudio, introducimos el *Multiformer* como una alternativa viable al modelo base de *Speech-to-Text Translation*. Para la creación, evaluación y el análisis del *Multiformer* se han seguido los siguientes pasos:

- **Planteamiento del *Multiformer*:** Se fundamenta el nacimiento del *Multiformer* y su gran capacidad de configuración. Se explica la creación de la *Multi-Head Multi-Attention* y, de entre los mecanismos de atención mencionados en el apartado 3.2, se mencionan los que ésta incorpora.
- **Entrenamiento y evaluación:** Se explica el contenido teórico d'etras de algunas decisiones realizadas en la fase de experimentación.
- **Análisis de la matriz W_o:** Explicación teórica del análisis de W_o realizado para el estudio de la contribución de cada *head*.

4.1 Planteamiento del *Multiformer*

La utilización de una ventana de $25ms$ con un *Hop size* de $10ms$ en el preprocesamiento de la señal del audio, provoca que las secuencias acústicas sean considerablemente más largas que las de texto. El modelo de referencia en S2T, propuesto en Wang et al. [2020a], es una adaptación a la voz del *Vanilla Transformer* [Vaswani et al., 2017], que fué inicialmente planteado para el procesamiento de texto. A grandes rasgos se limitan a comprimir la secuencia de entrada en un factor de 4, obteniendo *tokens* que representan unos 55ms de audio. Por lo que resta, el modelo de vanguardia es muy similar a su equivalente en texto. Esto significa que éste no lidia con otras diferencias entre audio y texto, como la presencia de redundancia entre los *tokens* de audio debida a que se necesitan varios de éstos para la representación de un único fonema.

Nosotros proponemos explorar modificaciones en el módulo de *Self-Attention* para un

mejor funcionamiento de éste en datos de voz y, por consiguiente, introducimos la *Multi-Head Multi-Attention* (Figura 19), un módulo que, mediante la utilización de distintos tipos de atención a nivel de head, busca adaptar la creación de relaciones contextuales a las características de la secuencia de entrada. Si bien este concepto es muy amplio, en este estudio se focaliza en la implementación de *heads* orientados a la naturaleza de la voz.

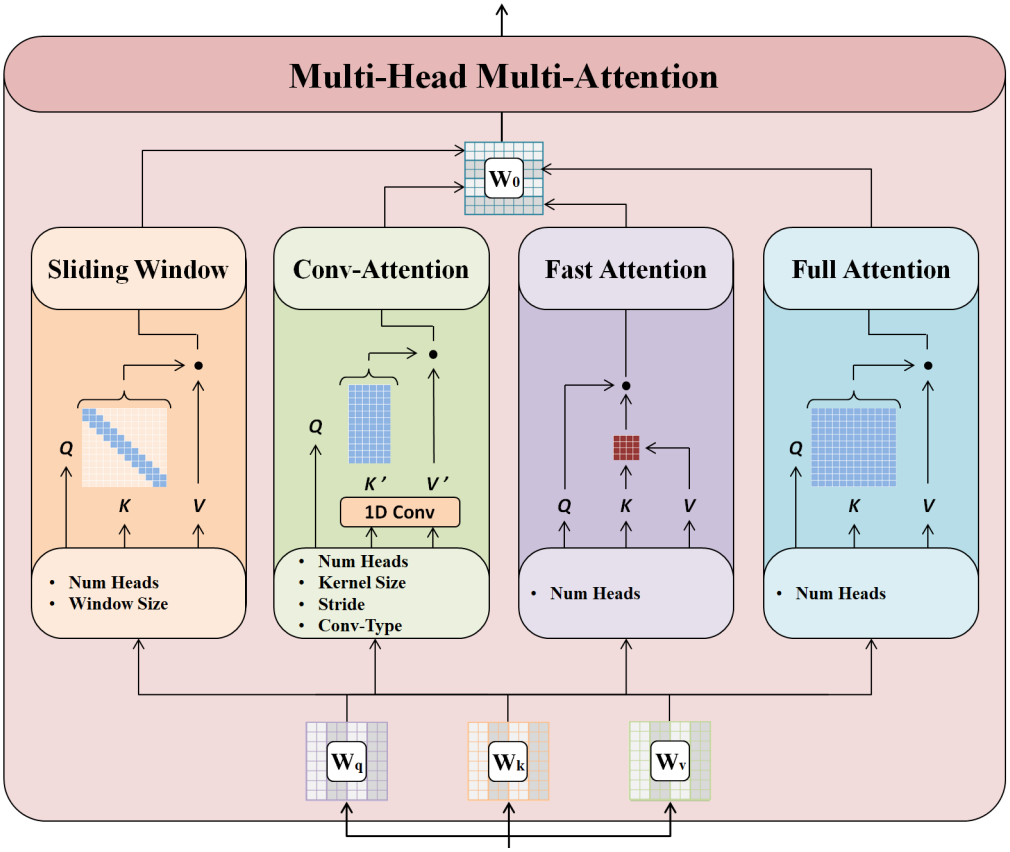


Figura 19: Estructura de la *Multi-Head Multi-Attention*.

Para combatir la redundancia de las secuencias de entrada elegimos utilizar la *Conv-Attention* propuesta en el *Speechformer* [Papi et al., 2021]. Al comprimir la longitud de secuencia de *keys* y *values* se consigue comprimir *tokens* repetitivos y, por consiguiente, evitar el cómputo de relaciones innecesarias.

En la comunicación oral es común el uso de estructuras simples, repeticiones, reformulaciones y contextualizaciones constantes para facilitar al receptor la comprensión del mensaje, por lo que el contexto local goza de mucha relevancia. Con el objetivo de favorecer la atención a un contexto local implementamos la *Sliding window*, utilizada en modelos como el *Longformer* [Beltagy et al., 2020] y *Big Bird* [Zaheer et al., 2020]. Al limitar el patrón de atención a *tokens* próximos a la diagonal de la matriz de *Self-Attention*, la *Slid-*

ing Windows fuerza al modelo a establecer relaciones locales. Además, al "deslizarse" ésta dentro de la matriz de atención, la *Sliding-Windows* adopta un comportamiento similar a la de una capa convolucional, cuyos pesos son distintos en cada desplazamiento. Por este motivo, la *Sliding-Windows* también realiza un procesamiento extra de la secuencia de entrada.

Si bien la *Conv-Attention* proporciona contexto global, éste es entre tokens comprimidos. Para permitir al modelo la creación de relaciones lejanas entre *tokens* más específicos, el *Multiformer* dispone de la posibilidad de utilizar *heads* de *Full-Attention* propios del *Vanilla Transformer*. Por ende, con una configuración donde todos los *heads* sean de *Full-Attention*, nuestro nuevo módulo se comporta como la *Multi-head Attention* del S2T *Transformer*. Además, hemos habilitado la posibilidad de substituir la *Full-Attention* por la *Fast-Attention* presente en el *Performer* [Choromanski et al., 2020b]. Mediante la aproximación mediante *kernels* de la función $softmax()$, la *Fast-Attention* disminuye la complejidad. Por ende, se entiende la *Fast-Attention* como una alternativa eficiente a la *Full-Attention*.

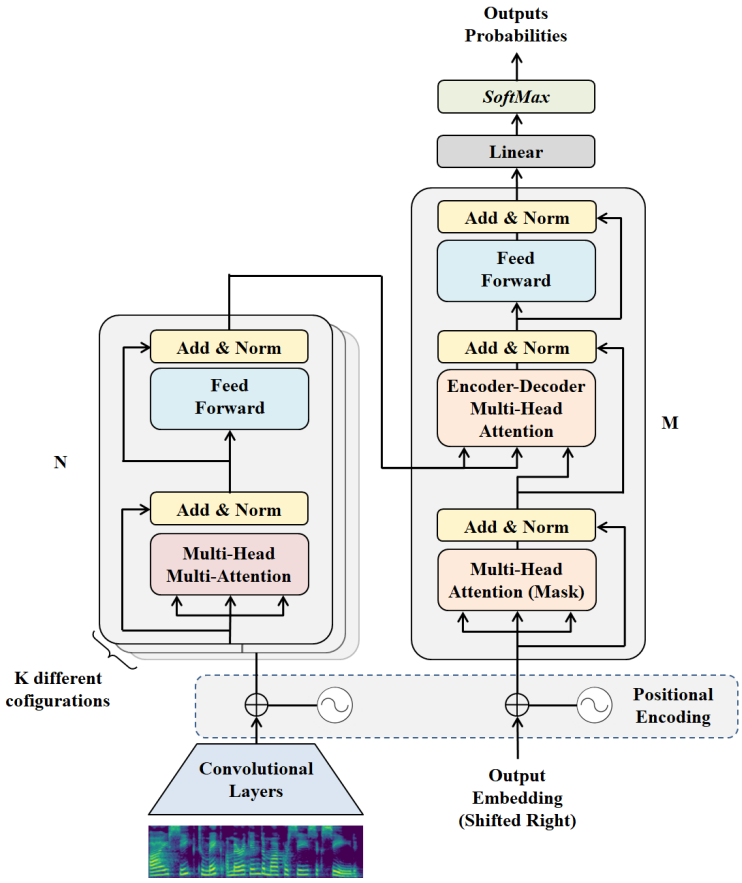


Figura 20: Estructura del *Multiformer*.

Finalmente, si bien la *Multi-Head Multi-Attention* ya admite un alto nivel de diversidad de atención que estimula la extracción de distintos tipos de relaciones, el *Multiformer* goza de la posibilidad de variar la configuración de cada capa del *Encoder*. De este modo, nos permite estudiar la construcción de arquitecturas más complejas que empleen distintas configuraciones de atención entre capas. Por ejemplo, se pueden crear arquitecturas cuyas primeras capas se encarguen de procesar el *input*, las intermedias busquen relaciones más locales y las últimas extraigan un contexto más general.

4.2 Entrenamiento y evaluación

Proponemos entrenar arquitecturas del *Multiformer* con distintas configuraciones. Entre ellas, (i) arquitecturas que únicamente realicen distintos tipos de atención a nivel de *heads*, (ii) configuraciones cuyos mecanismos de atención difieran exclusivamente a nivel de *layer*, y (iii) arquitecturas con variedad de atención en ambos niveles cuyo objetivo sea adaptarse a datos de voz. Además, proponemos realizar entrenamientos desactivando el *downsampling* de las capas convolucionales (Apendice B).

Para ello, primero entrenamos las arquitecturas para la tarea de ASR y, seguidamente, reutilizamos los *Encoders* preentrenados en ASR para los entrenamientos de S2T [Bérard et al., 2018]. Se sigue este procedimiento debido a que en S2T el *Encoder* debe aprender a realizar dos operaciones diferenciadas: (i) Modelado acústico y (ii) modelado semántico. Por lo que el preentrenamiento del modelo es efectivo debido a que la cantidad de datos de ASR es considerablemente mayor que para S2T. Mediante el preentrenamiento en ASR el *Encoder* aprende a modelar el audio antes de realizar el entrenamiento en S2T, y, por consiguiente, el algoritmo puede aprovechar el corpus de S2T para aprender a realizar una buena traducción. Nótese, que el idioma de la secuencia de voz procesada por el *Encoder* es el mismo tanto en ASR como en S2T. Por otro lado, el *Decoder* se entrena desde cero debido a que el alineamiento aprendido en ASR es completamente distinto al de S2T. Además, el idioma de los datos del *Decoder* difiere para ambas tareas, por lo que el modelado de lenguaje aprendido no es adecuado.

Una vez las distintas arquitecturas del *Multiformer* estén entrenadas, se pondrán a prueba con datos de test. A partir de los resultados obtenidos, calcularemos la calidad de éstas en ASR mediante la WER. Para evaluar la calidad de las traducciones de S2T se calculará la BLEU.

4.3 Análisis de la matriz W_o

Debido a que el *Multiformer* permite un gran numero de configuraciones, se analizará las contribución de cada *head* en cada capa del *Encoder* de aquella arquitectura con mejores resultados. El análisis nos permitirá realizar una segunda iteración de experimentos con arquitecturas que corrijan las carencias observadas, substituyendo los *heads* de poca relevancia.

Se puede expresar el computo realizado en la *Multi-Head Multi-Attention* mediante la generalización de las ecuaciones 10 y 11. Por consiguiente, se define la salida del módulo (Z) como:

$$Z = \overbrace{(Z_1, ..., Z_h)}^{Concatenación} \cdot W_o \quad con \quad Z_i = F_i\left(Q_i, K_i, V_i\right) \quad y \quad i \in [0, h] \tag{23}$$

donde $Z_i \in \mathbb{R}^{n \times d_{head}}$ es la la salida de cada *head* obtenida mediante una función F_i que procesa *queries, keys* y *values* dependiendo del mecanismo de atención seleccionado. Por otra parte, $W_o \in \mathbb{R}^{h \cdot d_{head} \times d_{model}}$ es una matriz de pesos entrenada con el modelo, por lo que sus parámetros contienen información acerca de la relevancia de cada *head*. Además, debido a que las salidas de los *heads* se encuentran concatenadas, se pueden diferenciar las regiones de la matriz W_o correspondientes a cada *head*.

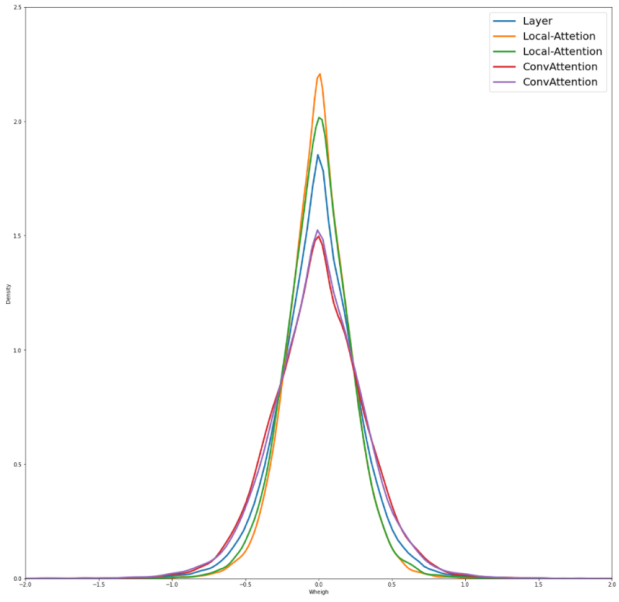


Figura 21: Distribución de pesos de la primera capa del s2t_multiformer_s_h_lc. En ella se puede observar como los pesos correspondientes a los *heads* de *Local-Attention* se encuentran distribuidos considerablemente más cerca del 0 que los de *ConvAttention*.

Por este motivo, en cada capa del *Encoder* se realizará el análisis de la contribución de cada *head* mediante la representación de los pesos de las distintas regiones de W_o. Por ende, si observamos que la distribución de los pesos correspondientes a un determinado *head* son mayoritariamente cercanos a 0, significará que su contribución en la arquitectura es baja. En la figura 21 se observa un ejemplo.

5 Experimentos y Resultados

En este apartado se explicarán las decisiones clave y previas a la fase de experimentación y se expondrán los distintos experimentos realizados, así como los parámetros y resultados de cada uno. Para la evaluación de la calidad de nuestros resultados se ha establecido el *S2T Transformer* [Wang et al., 2020a], explicado en el apartado 3.1, como modelo de referencia. Concretamente, nuestros resultados se compararán con los obtenidos por la arquitectura *s2t_transformer_s*.

5.1 Especificaciones de Implementación

La implementación del modelo se ha realizado mediante la librería *Fairseq* [Ott et al., 2019]. Ésta dispone de un amplio catálogo de herramientas escritas en *PyTorch*[24] para el modelaje de secuencias. Además, *Fairseq* permite a investigadores y desarrolladores de *Software* entrenar modelos de generación de texto para tareas de NLP.

Con el fin de lograr una comparación fiel entre los resultados de nuestro modelo y los obtenidos por Wang et al. [2020a], hemos realizado la implementación del *Multiformer* lo más parecida al modelo de referencia. Por ende, gran parte de la implementación de nuestro modelo se basa en el *S2T Transformer*, modificando y añadiendo exclusivamente las partes necesarias. Por ende, dado que en este estudio no se propone ninguna modificación para la etapa decodificadora, el *Multiformer* utiliza el *Transformer Decoder* disponible en *Fairseq*.

Encoder. Reemplazamos la *Multi-Head Attention* por la implementación de la *Multi-Head Multi-Attention* explicada en el apartado 5.1.1. Además, hemos modificado la construcción de las distintas capas del *Encoder* para poder realizar configuraciones de atención diferentes entre ellas.

Capas convolucionales. El *Multiformer* base mantiene el *down sampling* realizado por estas capas. Sin embargo, dado que la *Multi-Head Multi-Attention* implementada en nuestro modelo permite configuraciones de atención más eficientes, hemos habilitado la opción de desactivar dicha reducción. En consecuencia, en el apéndice B, también se entrenan arquitecturas que prescinden del *down sampling*.

[24] *PyTorch* es una librería de código abierto basada en *Python* y muy utilizada en *Deep Learning*. Esto es debido a que esta realiza los cálculos mediante el uso de tensores, muy útiles para la representación de los datos utilizados.

5.1.1 Implementación de la *Multi-Head Multi-Attention*

Tal y como se ha mencionado previamente (§4.1), la *Multi-Head Multi-Attention* dispone de la capacidad de realizar distintos tipos de atención entre *heads*. Para su funcionamiento se requiere la implementación en *Fairseq* de los siguientes módulos:

- ***Sliding window***: Este módulo se encuentra implementado en un repositorio basado en *Fairseq*[25]. Sin embargo, éste no incorpora la opción de la variante *Dilated Sliding Window*. Además, la implementación difiere ligeramente respecto a su planteamiento teórico. Esto es consecuencia de que la iteración de la ventana a través de los *tokens* resulta ineficiente. Por consiguiente, la ventana itera respecto a bloques de *tokens*, tomando a un comportamiento similar al de la *Local-Attention*. Con todo, el módulo disponible es igualmente capaz de desempeñar la función prevista. Su implementación en el *Multiformer* permite elegir el tamaño de la ventana deseado mediante la configuración del modelo.

- ***Conv-Attention***: Módulo íntegramente implementado por nosotros mediante su descripción teórica en Papi et al. [2021]. Con la configuración del *Multiformer* es posible ajustar el grado de compresión mediante el *Kernel size* y el *stride*[26]. Además, también se puede elegir entre tres distintos tipos de convolución: Estándar, *Depthwise*[27] o *Separable*[28].

- ***Fast-Attention***: Módulo ya implementado en el repositorio basado en *Fairseq* del *Performer* [Choromanski et al., 2020b]. Únicamente se ha requerido de una adaptación apropiada dentro de la *Multi-head Multi-Attention*.

- ***Full-Attention***: Hemos realizado una nueva implementación de la *Full-Attention*. Esto es debido a que el módulo ya existente en *Fairseq* está integrado dentro de la *Multi-head Attention*. Por ende, éste es incompatible con la intercalación de distintos *heads* de la *Multi-Head Multi-Attention*.

[25] https://github.com/lucidrains/local-attention

[26] El tamaño del *kernel* de una convolución determina el número de *tokens* que influyen en la creación del *token* resultante. Por otro lado, el *stride* indica el número de posiciones que se desplaza el *kernel* en cada salto. Mediante el ajuste del *stride* es posible controlar el factor de compresión de la secuencia resultante.

[27] La Convolución *depthwise* es un tipo de convolución que aplica un único filtro convolucional para todos los canales de entrada.

[28] La Convolución *separable* esta formada por una *depthwise convolution* seguida de una *pointwise convolution*. A diferencia de la *depthwise convolution*, este tipo de convolución es capaz de operar también en la dimensión de profundidad.

5.2 Definición del Entorno de Experimentación

En esta sección se indicarán las condiciones de experimentación, como el conjunto de datos empleado, el entorno de ejecución y los parámetros de entrenamiento.

Conjunto de Datos. Todo modelo de aprendizaje automático requiere ser entrenado y evaluado sobre un conjunto de datos. En nuestro caso, hemos decidido emplear el banco de datos "*Multilingual ST Corpus*" (MuST-C) [Cattoni et al., 2021]. La elección de este conjunto de datos se debe a sus cualidades, como: (i) una gran cobertura y diversidad lingüística (del inglés a 14 lenguas de familias diferentes), (ii) su tamaño (cerca de 237 horas de grabaciones transcritas para cada uno de los idiomas, lo que corresponde a 430 muestras por lengua), (iii) una amplia variedad de temas y oradores, y (iv) una alta calidad de sus datos. Para nuestros experimentos, se utilizará el subconjunto de datos Inglés-Alemán.

Entorno de Ejecución. Para alojar el entrenamiento de las distintas arquitecturas del *Multiformer* se utilizará Calcula, un servidor de la UPC con nodos dotados con tarjetas gráficas NVIDIA GeForce RTX 2080 Ti. Cada entrenamiento requerirá únicamente de una GPU.

Parámetros de entrenamiento en ASR. Para garantizar una comparación fiable, se han utilizado, en la medida de lo posible, los parámetros establecidos por Wang et al. [2020a]. En particular, en los preentrenamientos en ASR se han utilizado 4 CPU y 4 *workers* para la carga de los datos. El número máximo de *tokens* por *batch* se ha fijado en 20.000, por lo que se ha compensado la frecuencia de actualización de los parámetros del sistema en 1 actualización cada 16 *batches*. De este modo, se consigue que los parámetros se actualicen tras procesar 320.000 *tokens*. Se ha empleado el optimizador Adam y un *learning rate* de $1 \cdot 10^{-3}$ con el *inverse square root scheduler*. También se ha utilizado *warm-up* en las primeras 10.000 actualizaciones y un *clip norm* de 10 para evitar la explosión de los gradientes. Por ultimo, se ha empleado la *label smoothed Cross-entropy* con factor de suavizado 0.1 como función de perdida.

Parámetros de entrenamiento en S2T. En *Speech-to-text Translation* se asignan los mismos valores para prácticamente todos los parámetros de ASR, en excepción del *learning rate* que toma el valor de $2 \cdot 10^{-3}$.

Los modelos (*S2T Transformer* y *Multiformer*) serán entrenados en sus arquitecturas "s" hasta completar 50.000 actualizaciones. Estas constan de 12 capas de codificación y 6 de

decodificación, y sus módulos de atención utilizan 4 *heads* tanto en el *Encoder* como en el *Decoder*. Además, la dimensión de *embbedings* es de 256, y 2048 en las capas FFNNs. La dimensión de la salida del *decoder* es de 256, coincidiendo con la dimensión de los *embbedings* del decodificador. Las arquitecturas de tipo "s" establecen una probabilidad de *dropout* del 10% para los *Attention weights* y para las activaciones de las FFNNs, basadas en la función ReLU.

Por otra parte, el *Multiformer* incorpora nuevos parámetros específicos relativos a la construcción de su configuración que serán definidos para cada experimento. Además, nuestras arquitecturas tipo "s" admiten la modificación del *stride* y del tamaño de *kernel* de las capas convolucionales. Este cambio permite mantener el *downsampling*, con los valores del modelo de referencia ($stride = 2$ y $kernel\_size = 5$), y, a su vez, crear arquitecturas sin dicha reducción ($stride = 1$).

5.3 Descripción de los Experimentos

Además de comparar nuestro sistema con el *Speech-to-Text Transformer* de *Fairseq*, también queremos estudiar la influencia de las variables de nuestro modelo. Léase, queremos comprobar si la adición de diversidad de mecanismos de atención favorece a la creación de soluciones más complejas, y por ende, se obtienen mejores resultados. Por este motivo, se entrenan arquitecturas sin diversidad de atención, otras con diversidad únicamente a nivel de capa o *head* y arquitecturas con diversidad en ambos niveles. Las arquitecturas entrenadas en esta la fase de experimentación se encuentran descritas en la tabla 1.

Tal y como se ha comentado en el capítulo 4, primero se realiza un entrenamiento de cada arquitectura para ASR y seguidamente se utiliza el *Encoder* preentrenado para el entrenamiento en ST [Bérard et al., 2018].

Con el fin de realizar una futura comparación entre modelos con y sin diversidad de mecanismos de atención, empezamos la fase de experimentación con arquitecturas basadas en un único mecanismo de atención [1-5]. Entre ellas, se ha decidido oportuno el entrenamiento de dos arquitecturas de *ConvAttention* [4 y 5] para discernir la configuración más adecuada, resultando ser ésta $kernel\_size = 5$ y $stride = 2$. En las arquitecturas con *Local-Attention* hemos utilizado 64 como tamaño de ventana ($\approx 3.5s$ de audio) dado que Alastruey et al. [2021], quienes utilizaron el *Longformer* [Beltagy et al., 2020] para ST, observaron que éste obtiene los mejores resultados.

ID	Nombre			
1	s2t_multiformer_s_full	$12 \times (\ 4\times\ Full\ )$		
2	s2t_multiformer_s_fast	$12 \times (\ 4\times\ Fast\ )$		
3	s2t_multiformer_s_local	$12 \times (\ 4\times\ Local\,(64)\ )$		
4	s2t_multiformer_s_conv_5_2	$12 \times (\ 4\times\ Conv\,(5,2)\ )$		
5	s2t_multiformer_s_conv_7_3	$12 \times (\ 4\times\ Conv\,(7,3)\ )$		
6	s2t_multiformer_s_h_lc	$12 \times \begin{pmatrix} 2\times\ Local\,(64) \\ 2\times\ Conv\,(5,2) \end{pmatrix}$		
7	s2t_multiformer_s_l_lc	$6 \times (\ 4\times\ Local\,(64)\ )$	$6 \times (\ 4\times\ Conv\,(5,2)\ )$	
8	s2t_multiformer_s_l_cl	$6 \times (\ 4\times\ Conv\,(5,2)\ )$	$6 \times (\ 4\times\ Local\,(64)\ )$	
9	s2t_multiformer_s	$2 \times (\ 4\times\ Conv\,(5,2)\ )$	$6 \times \begin{pmatrix} 2\times\ Local\,(64) \\ 2\times\ Conv\,(5,2) \end{pmatrix}$	$4 \times \begin{pmatrix} 2\times\ Full \\ 2\times\ Conv\,(7,3) \end{pmatrix}$
10	s2t_multiformer_s_v2	$6 \times \begin{pmatrix} 1\times\ Local\,(64) \\ 3\times\ Conv\,(5,2) \end{pmatrix}$	$6 \times \begin{pmatrix} 2\times\ Local\,(64) \\ 2\times\ Conv\,(5,2) \end{pmatrix}$	
11	s2t_multiformer_s_v3	$6 \times \begin{pmatrix} 1\times\ Local\,(64) \\ 3\times\ Conv\,(5,2) \end{pmatrix}$	$6 \times \begin{pmatrix} 3\times\ Local\,(64) \\ 1\times\ Conv\,(5,2) \end{pmatrix}$	
12	s2t_multiformer_s_v4	$6 \times \begin{pmatrix} 1\times\ Local\,(64) \\ 3\times\ Conv\,(5,2) \end{pmatrix}$	$3 \times \begin{pmatrix} 2\times\ Local\,(64) \\ 2\times\ Conv\,(5,2) \end{pmatrix}$	$3 \times \begin{pmatrix} 3\times\ Local\,(64) \\ 1\times\ Conv\,(5,2) \end{pmatrix}$
13	s2t_multiformer_s_v5	$4 \times (\ 4\times\ Conv\,(5,2)\ )$	$3 \times \begin{pmatrix} 1\times\ Local\,(64) \\ 3\times\ Conv\,(5,2) \end{pmatrix}$	$5 \times \begin{pmatrix} 2\times\ Local\,(64) \\ 2\times\ Conv\,(5,2) \end{pmatrix}$

Table 1: Arquitecturas entrenadas del *Multiformer*. La notación para cada configuración es la siguiente: $N_{layers} \times (\ N_{heads}\times\ Attention\,(parameters)\ )$

Las arquitecturas 6, 7 y 8, además de intentar superar el modelo de referencia, se han entrenado con el fin de observar qué tipo de diversidad de atención (a nivel de *head* o a nivel de *layer*) es más enriquecedora para el modelo. Por ende, el s2t_multiformer_s_h_lc realiza todas las capas del *Encoder* con 2 *heads* de *Local-Attention* y de *ConvAttention*,[29] mientras que s2t_multiformer_s_l_lc y s2t_multiformer_s_l_cl constan de 6 capas de *ConvAttention* y 6 más de *Local-Attention*. Estos dos mecanismos de atención han sido elegidos en la comparación debido a que la *Local-Attention* favorece la búsqueda de contexto local, y la *ConvAttention* la extracción de contexto mediante *tokens* más informativos[30].

El s2t_multiformer_s es una arquitectura exclusivamente planteada para mejorar los resultados del modelo de referencia, dado que su construcción esta motivada en conseguir un correcto procesado del audio. Por consiguiente, el s2t_multiformer_s consta de dos primeras

[29]Si no se especifican los hyperparámetros de la *ConvAttention*, se asume que son $kernel\_size = 5$ y $stride = 2$.

[30]Al introducir *keys* y *values* a la capa convolucional con $kernel\_size = 5$ y $stride = 2$, éstos se comprimen en un factor de 2, por lo que se realiza un mecanismo de atención con *tokens* que contienen $\approx 275ms$ de audio.

capas de codificación basadas íntegramente en *ConvAttention* para realizar un procesado extra al de las capas convoluciones previas. A continuación, incorpora 6 capas con 2 *heads* de *Local-Attention* y *ConvAttention*, impulsando la extracción de contexto local y, a su vez, manteniendo la capacidad de crear relaciones globales. Finalmente, tras haber forzado la obtención de contexto más local en las capas intermedias, el s2t_multiformer_s incorpora 4 capas con 2 *heads* de *Full-Attention*, para favorecer la creación detallada de un contexto global, y 2 más de *ConvAttention* con $kernel\_size = 7$ y $stride = 3$, para un contexto general entre *tokens* con más contenido secuencial.

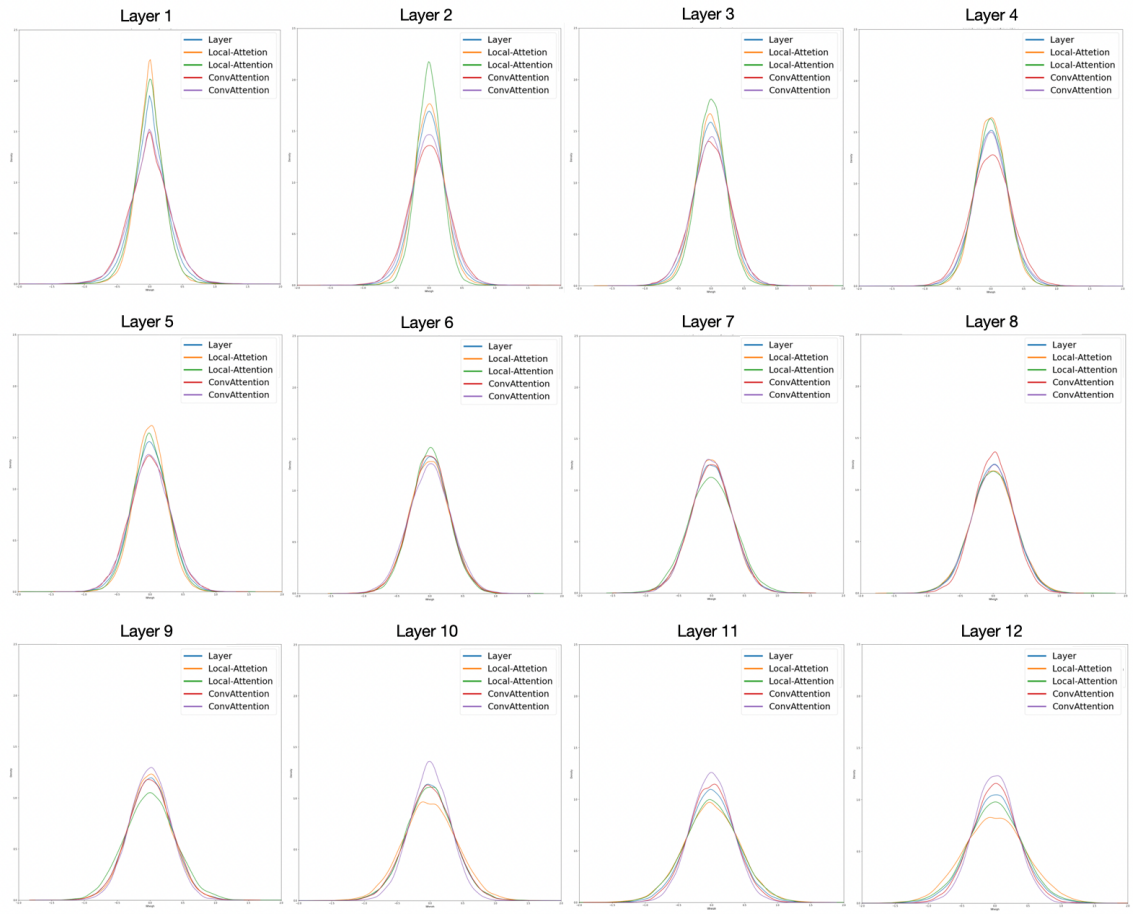


Figura 22: Distribuciones de los pesos de W_0 en cada capa del s2t_multiformer_s_h_lc. En ellas se puede observar como en las primeras capas los pesos correspondientes a los *heads* de *Local-Attention* se encuentran distribuidos considerablemente más cerca del 0 que los de *ConvAttention*. Esta tendencia se invierte de manera progresiva a lo largo de las capas, terminando la *Local-Attention* con distribuciones menos concentradas en 0.

Tras una primera fase de experimentación, motivados por alcanzar los mejores resultados posibles, se han analizado las distribuciones de los pesos de las matrices W_0 de cada capa del s2t_multiformer_s_h_lc, dado que éste alcanza la mejor puntuación de BLEU. Mediante

la representación de dichas distribuciones, podemos observar el grado de contribución de cada *head* en cada capa (figura 22). Por ende, este estudio permite la construcción de arquitecturas que corrijan aquellos *heads* cuyos pesos estén mayoritariamente distribuidos en 0, lo que significa que el modelo no les esta asignando gran relevancia. Debido a que existe una relación de causalidad entre capas,[31] éste análisis ha motivado la creación de 4 arquitecturas con distinto grado de corrección: (i) el s2t_multiformer_s_v2, que en las primeras 6 capas substituye uno de los *heads* de *Local-Attention* por *ConvAttention*, (ii) el s2t_multiformer_s_v3, que además de hacer la misma corrección anterior, también intercambia un *head* de *ConvAttention* por otro de *Local-Attention* en las 6 capas restantes, (iii) el s2t_multiformer_s_v4, realizando una corrección más suave y progresiva que las arquitecturas anteriores, y (iv) el s2t_multiformer_s_v5, con modificaciones más estrictas.

	Configuración	**BLEU(↑)**
s2t_transformer_s	-	22.25
s2t_multiformer_s_full	-	22.20
s2t_multiformer_s_fast	-	20.28
s2t_multiformer_s_local	-	22.00
s2t_multiformer_s_conv_5_2	-	22.36
s2t_multiformer_s_conv_7_3	-	21.91
s2t_multiformer_s_l_lc	*LC*	21.55
s2t_multiformer_s_l_cl	*LC*	22.36
s2t_multiformer_s_h_lc	*HC*	22.52
s2t_multiformer_s	$HC + LC$	22.42
s2t_multiformer_s_v2 †	$\boldsymbol{HC + LC}$	**22.63**
s2t_multiformer_s_v4 †	$HC + LC$	22.28
s2t_multiformer_s_v3 †	$HC + LC$	22.41
s2t_multiformer_s_v5 †	$\boldsymbol{HC + LC}$	**22.60**

Table 2: Resultados en ST de los experimentos realizados. En la columna de "Configuración" se contemplan los casos: "**LC**" (Arquitecturas que presentan distintas configuraciones a nivel de capa), "**HC**" (Arquitecturas que presentan distintas configuraciones a nivel de *head*) y "**-**" (Arquitecturas sin configuración).
†: Experimento de la segunda iteración.

Todas las arquitecturas han sido entrenadas con el conjunto de datos de *MuST-C* durante 90 *epochs* ($\approx$ 92 horas para cada experimento) y utilizando el mejor *chekpoint* para el

[31] La distribución de pesos de una capa depende de las distribuciones de pesos de las capas anteriores.

prentrenamiento en ASR del *Encoder*. En la tabla 2 se muestran los resultados en BLEU obtenidos por cada arquitectura.

5.4 Análisis de Resultados

Tras obtener los resultados de las arquitecturas sin diversidad (tabla 2) se observa que: (i) El s2t_transformer_s y el s2t_multiformer_s_full, al ser teóricamente idénticos, sus resultados también son muy similares[32] (22.25 y 22.20 puntos de BLEU respectivamente). (ii) El s2t_multiformer_s_fast obtiene los peores resultados (20.28 puntos de BLEU). Creemos que esto se debe a que en presencia de muestras cuyas secuencias sean de longitud menor que la dimension de *embeddings*, el desempeño de la *Fast-Attention* empeora respecto a la *Full-Attention*. (iii) Tal y como ya observaron Alastruey et al. [2021], aunque el uso de varias capas de codificación permite que los tokens de la salida del *Encoder* gocen de contexto global, con el uso exclusivo de *Local-Attention* éste podría no ser suficiente. (vi) El s2t_multiformer_s_conv_5_2, mediante la simple substitución de la *Full-Attention* por *ConvAttention* ($kernel\_size = 5$ y $stride = 2$), consigue 22.36 puntos de BLEU, situándose por encima del modelo de referencia. Creemos que esta mejora puede atribuirse a que, al hacer el computo de los *queries* y unas *keys* comprimidas, la *ConvAttention* otorga cierta noción grupal entre *tokens* cercanos, lo que puede aportar información estructural del audio. Nótese que los resultados del s2t_multiformer_s_conv_7_3 (21.91 puntos de BLEU) sugieren que un exceso de compresión de *keys* y *values*, puede ser desfavorable. Además se verifica que el uso de capas convolucionales previas al *Encoder* como única modificación para la adaptación del *Transformer* [Vaswani et al., 2017] a audio es insuficiente. Si bien estas capas reducen ≈ 4 veces la longitud de la secuencia de audio (pasando de una proporción 200:1 a 50:1 entre número de *tokens* de audio respecto a *tokens* de texto), esta sigue siendo considerablemente mayor que las de texto, por lo que la utilización inmediata de la *Self-Attention*, planteada para traducción de texto, no es óptima.

También se comprueba que, en general, la construcción de modelos con diversidad limitan la existencia de *heads* redundantes, mejorando la capacidad d'extracción y por ende, superando los resultados de las arquitecturas monótonas P(≈ 22.34 y ≈ 21.83 puntos de BLEU de media respectivamente). Además, de los experimentos 6, 7 y 8 se observa como el impacto de la diversidad de atención a nivel de *heads* (HC) es mayor que a nivel de capa (la mejor arquitecturas en HC se sitúa 0.16 puntos de BLEU por encima de la mejor en LC). Creemos que esto se debe a que, mientras que la configuración HC mantiene el grado de diversidad a lo largo de las capas, en las configuraciones puramente LC la consecución

[32]Dado que en el entrenamiento se realiza una inicialización aleatoria de los parámetros de las arquitecturas, diferencias de ± 0.05 puntos de BLEU no son consideradas como relevantes.

de varias capas con el mismo mecanismo de atención las hace vulnerables a la generación de redundancia y al abandono de *heads*.

Por otro lado, mientras que en un inicio creíamos que en las primeras capas se debía dar más importancia al contexto local y en las ultimas priorizar el global, los resultados del s2t_multiformer_s_l_lc y del s2t_multiformer_s_l_cl sugieren lo contrario, obteniendo 21.55 y 22.36 puntos de BLEU respectivamente. Nótese que estos resultados concuerdan con el análisis realizado a s2t_multiformer_s_h_lc, donde se aprecia como la matriz W_o asigna una mayor concentración de parámetros cercanos a 0 para los *heads* de *Local-Attention* que para los de *ConvAttention*.

Por último, el s2t_multiformer_s_v2 y el s2t_multiformer_s_v5, arquitecturas construidas a partir del análisis de las matrices W_0 del s2t_multiformer_s_h_lc y con diversidad de atención multinivel, consiguen los mejores resultados de este estudio, con 22.63 y 22.60 puntos de BLEU respectivamente. Resultados que reafirman que la diversidad de atención reduce la redundancia entre los distintos *heads*.

6 Conclusión y Desarrollo Futuro

En este trabajo presentamos el *Multiformer*, un modelo basado en el *Transformer* [Vaswani et al., 2017] que propone realizar un mecanismo de atención más acorde a los datos de voz. Para ello, el modelo incorpora la *Multi-Head Multi-Attention* que ofrece una plena configuración de la misma mediante la elección del tipo de atención de cada *head*, de entre 4 mecanismos disponibles. En consecuencia, el módulo admite la utilización de distintos mecanismos de atención entre *heads*, lo que estimula la extracción de información más variada y, por ende, limita la presencia de *heads* con poca relevancia [Voita et al., 2019, Bian et al., 2021, Zhang et al., 2021]. Además, el *Multiformer* también habilita la creación del *Encoder* a nivel de capa, permitiendo la construcción de arquitecturas más complejas y con mayor diversidad de atención.

Todas las arquitecturas del *Multiformer* con diversidad de atención multinivel superan los resultados obtenidos por el modelo de referencia en ST [Wang et al., 2020a]. Concretamente, el *s2t_multiformer_s_v2* y el *s2t_multiformer_s_v5* se sitúan cerca de 0.4 puntos de BLEU por encima del modelo de referencia. Esto sugiere que la diversidad resulta apropiada para combatir las extracciones redundantes durante el *Encoder*.

Además, hemos observado cómo la utilización de la *Self-Attention* del *s2t_transformer_s* no resulta óptima para datos de audio, dado que la arquitectura del *Multiformer* basada íntegramente en *ConvAttention* supera ligeramente la puntuación de BLEU del modelo base. Por ende, creemos que sería apropiado la búsqueda de nuevos planteamientos que, a diferencia de la *Self-Attention*, no estén basados íntegramente en el procesamiento de texto.

Por otro lado, contrariamente a la hipótesis inicial de realizar una extracción de contexto local en las primeras capas y global en las últimas, los resultados sugieren que es preciso mantener un equilibrio y permitir la extracción de relaciones de corta y larga distancia durante todo el *Encoder*.

A la vista de los resultados, creemos que seria interesante realizar un análisis más profundo de la contribución de cada *head* considerando en éste también la matriz W_v, debido a que ésta también posee información acerca de la relevancia de cada *head*. Un análisis más preciso de la contribución de cada *head* permitiría la construcción de mejores arquitecturas. Además, creemos que resultaría interesante realizar la implementación de la *Dilated Sliding Windows* dentro de la *Multi-Head Multi-Attention*, debido a que, a parte de dotar al modelo de más diversidad, su patrón de atención parece adecuado para combatir la redundancia entre los *tokens* de audio.

Finalmente, si bien se han superado los resultados del modelo de referencia, el impacto de los cambios en el *Encoder* ha sido bajo, obteniendo mejoras pequeñas. En consecuencia, creemos que es necesario profundizar en la *Encoder-Decoder Attention*, bajo la intuición de que la falta de un alineamiento apropiado entre la salida del *Encoder* y los datos del *Decoder* dificulta una correcta transmisión de información entre *Encoder* y *Decoder*.

Referencias

Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. Etc: Encoding long and structured inputs in transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 268–284, 2020.

Belen Alastruey, Gerard I. Gállego, and Marta R. Costa-jussà. Efficient transformer for direct speech translation, 2021.

Antonios Anastasopoulos, David Chiang, and Long Duong. An unsupervised probability model for speech-to-translation alignment of low-resource languages, 2016.

Antonios Anastasopoulos, Ondřej Bojar, Jacob Bremerman, Roldano Cattoni, Maha Elbayad, Marcello Federico, Xutai Ma, Satoshi Nakamura, Matteo Negri, Jan Niehues, et al. Findings of the iwslt 2021 evaluation campaign. In *Proceedings of the 18th International Conference on Spoken Language Translation (IWSLT 2021)*, pages 1–29, 2021.

Ebrahim Ansari, Amittai Axelrod, Nguyen Bach, Ondřej Bojar, Roldano Cattoni, Fahim Dalvi, Nadir Durrani, Marcello Federico, Christian Federmann, Jiatao Gu, Fei Huang, Kevin Knight, Xutai Ma, Ajay Nagesh, Matteo Negri, Jan Niehues, Juan Pino, Elizabeth Salesky, Xing Shi, Sebastian Stüker, Marco Turchi, Alexander Waibel, and Changhan Wang. FINDINGS OF THE IWSLT 2020 EVALUATION CAMPAIGN. In *Proceedings of the 17th International Conference on Spoken Language Translation*, pages 1–34, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.iwslt-1.1. URL `https://aclanthology.org/2020.iwslt-1.1`.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1409.0473`.

J. Baker. The dragon system–an overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23(1):24–29, 1975. doi: 10.1109/TASSP.1975.1162650.

T. Bayes. An essay towards solving a problem in the doctrine of chances. *Phil. Trans. of the Royal Soc. of London*, 53:370–418, 1763.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020.

Luisa Bentivogli, Mauro Cettolo, Marco Gaido, Alina Karakanta, Alberto Martinelli, Matteo Negri, and Marco Turchi. Cascade versus direct speech translation: Do the differences still make a difference? *arXiv preprint arXiv:2106.01045*, 2021.

Alexandre Bérard, Olivier Pietquin, Laurent Besacier, and Christophe Servan. Listen and translate: A proof of concept for end-to-end speech-to-text translation. In *NIPS Workshop on end-to-end learning for speech and audio processing*, 2016.

Alexandre Bérard, Laurent Besacier, Ali Can Kocabiyikoglu, and Olivier Pietquin. End-to-end automatic speech translation of audiobooks. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6224–6228. IEEE, 2018.

Yuchen Bian, Jiaji Huang, Xingyu Cai, Jiahong Yuan, and Kenneth Church. On attention redundancy: A comprehensive study. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 930–945, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.72. URL `https://aclanthology.org/2021.naacl-main.72`.

Roldano Cattoni, Mattia Antonino Di Gangi, Luisa Bentivogli, Matteo Negri, and Marco Turchi. Must-c: A multilingual corpus for end-to-end speech translation. *Computer Speech & Language*, 66:101155, 2021.

William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4960–4964. IEEE, 2016.

David Chiang. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd annual meeting of the association for computational linguistics (acl'05)*, pages 263–270, 2005.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers, 2019.

Kyunghyun Cho, B van Merrienboer, Caglar Gulcehre, F Bougares, H Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014.

Krzysztof Choromanski, Mark Rowland, and Adrian Weller. The unreasonable effectiveness of structured random orthogonal embeddings, 2018.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, David Belanger, Lucy Colwell, and Adrian Weller. Masked language modeling for proteins via linearly scalable long-context transformers, 2020a.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. In *International Conference on Learning Representations*, 2020b.

Mattia A Di Gangi, Matteo Negri, and Marco Turchi. Adapting transformer to end-to-end spoken language translation. In *INTERSPEECH 2019*, pages 1133–1137. International Speech Communication Association (ISCA), 2019.

Long Duong, Antonios Anastasopoulos, David Chiang, Steven Bird, and Trevor Cohn. An attentional model for speech translation without transcription. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 949–959, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1109. URL `https://aclanthology.org/N16-1109`.

Kunihiko Fukushima, Sei Miyake, and Takayuki Ito. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):826–834, 1983. doi: 10.1109/TSMC.1983.6313076.

Marco Gaido, Mauro Cettolo, Matteo Negri, and Marco Turchi. Ctc-based compression for direct speech translation. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 690–696, 2021.

Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on International Conference on Machine Learning-Volume 32*, pages II–1764, 2014.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, page 369–376, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933832. doi: 10.1145/1143844.1143891. URL https://doi.org/10.1145/1143844.1143891.

Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers, 2019.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Niehues Jan, Roldano Cattoni, Stüker Sebastian, Mauro Cettolo, Marco Turchi, and Marcello Federico. The iwslt 2018 evaluation campaign. In *16th International Workshop on Spoken Language Translation*, 2018.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time, 2017.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and Francois Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *ICML 2020: 37th International Conference on Machine Learning*, volume 1, pages 5156–5165, 2020.

S. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3):400–401, 1987. doi: 10.1109/TASSP.1987.1165125.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer, 2020.

Philipp Koehn, Franz J Och, and Daniel Marcu. Statistical phrase-based translation. Technical report, UNIVERSITY OF SOUTHERN CALIFORNIA MARINA DEL REY INFORMATION SCIENCES INST, 2003.

Hugo Larochelle and Geoffrey E Hinton. Learning to combine foveal glimpses with a third-order boltzmann machine. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010. URL https://proceedings.neurips.cc/paper/2010/file/677e09724f0e2df9b6c000b75b5da10d-Paper.pdf.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, 2015.

A.A. Márkov. Rasprostranenie zakona bol'shih chisel na velichiny, zavisyaschie drug ot druga. *Izvestiya Fiziko-matematicheskogo obschestva pri Kazanskom universitete*, 2-ya seriya(15):135–156, 1906.

Rui Na, Junfeng Hou, Wu Guo, Yan Song, and Lirong Dai. Learning adaptive downsampling encoding for online end-to-end speech recognition. In *2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 850–854, 2019. doi: 10.1109/APSIPAASC47483.2019.9023043.

H. Ney. Speech translation: coupling of recognition and translation. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258)*, volume 1, pages 517–520 vol.1, 1999. doi: 10.1109/ICASSP.1999.758176.

Jan Niehues, Roldano Cattoni, Sebastian Stüker, Matteo Negri, Marco Turchi, T Ha, Elizabeth Salesky, Ramon Sanabria, Loic Barrault, Lucia Specia, et al. The iwslt 2019 evaluation campaign. In *16th International Workshop on Spoken Language Translation*, 2019.

Franz Josef Och and Hermann Ney. A systematic comparison of various statistical alignment models. *Computational linguistics*, 29(1):19–51, 2003.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

Sara Papi, Marco Gaido, Matteo Negri, and Marco Turchi. Speechformer: Reducing information loss in direct speech translation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1698–1706, 2021.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International Conference on Machine Learning*, pages 4055–4064. PMLR, 2018.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A. Smith, and Lingpeng Kong. Random feature attention, 2021.

Jiezhong Qiu, Hao Ma, Omer Levy, Wen-tau Yih, Sinong Wang, and Jie Tang. Blockwise self-attention for long document understanding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 2555–2565, 2020.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2008. URL `https://proceedings.neurips.cc/paper/2007/file/013a006f03dbc5392effeb8f18fda755-Paper.pdf`.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Elizabeth Salesky, Matthias Sperber, and Alan W Black. Exploring phoneme-level speech representations for end-to-end speech translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1835–1841, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1179. URL `https://aclanthology.org/P19-1179`.

Claude E Shannon. Communication theory of secrecy systems. *The Bell system technical journal*, 28(4):656–715, 1949.

Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

Jiajun Shen, Peng-Jen Chen, Matthew Le, Junxian He, Jiatao Gu, Myle Ott, Michael Auli, and Marc'Aurelio Ranzato. The source-target domain mismatch problem in machine translation. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1519–1533, 2021.

Matthias Sperber and Matthias Paulik. Speech translation and the end-to-end promise: Taking stock of where we are. *arXiv preprint arXiv:2004.06358*, 2020.

S. S. Stevens, John E. Volkmann, and Edwin B. Newman. A scale for the measurement of the psychological magnitude pitch. *Journal of the Acoustical Society of America*, 8: 185–190, 1937.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Laura Cross Vila, Carlos Escolano, José A. R. Fonollosa, and Marta Ruiz Costa-jussà. End-to-end speech translation with the transformer. In *IberSPEECH*, 2018.

Stephan Vogel, Hermann Ney, and Christoph Tillmann. Hmm-based word alignment in statistical translation. In *COLING 1996 Volume 2: The 16th International Conference on Computational Linguistics*, 1996.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned, 2019.

Lev Semenovich Vygotsky and Michael Cole. *Mind in society: Development of higher psychological processes*. Harvard university press, 1978.

Changhan Wang, Yun Tang, Xutai Ma, Anne Wu, Dmytro Okhonko, and Juan Pino. Fairseq S2T: Fast speech-to-text modeling with fairseq. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 33–39, Suzhou, China, dec 2020a. Association for Computational Linguistics. URL `https://aclanthology.org/2020.aacl-demo.6`.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity, 2020b.

Ron J. Weiss, Jan Chorowski, Navdeep Jaitly, Yonghui Wu, and Zhifeng Chen. Sequence-to-sequence models can directly translate foreign speech, 2017.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences, 2020.

Biao Zhang, Ivan Titov, Barry Haddow, and Rico Sennrich. Adaptive feature selection for end-to-end speech translation, 2020.

Tianfu Zhang, Heyan Huang, Chong Feng, and Longbing Cao. Enlivening redundant heads in multi-head self-attention for machine translation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3238–3248, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.260. URL https://aclanthology.org/2021.emnlp-main.260.

Anexos

A Resultados en ASR

Tal y como se ha comentado en la fase de experimentación, previamente a los entrenamientos en *Speech-to-Text Translation* se han preentrenado las arquitecturas para *Automatic Speech Recognition* [Bérard et al., 2018]. En la tabla 3 se muestran los resultados en ASR obtenidos por las distintas arquitecturas en los *checkpoints* utilizados para el posterior entrenamiento en S2T.

	Configuración	**WER(↓)**
s2t_transformer_s	-	13.53
s2t_multiformer_s_full	-	13.47
s2t_multiformer_s_fast	-	16.16
s2t_multiformer_s_local	-	14.00
s2t_multiformer_s_conv_5_2	-	13.58
s2t_multiformer_s_conv_7_3	-	13.86
s2t_multiformer_s_l_lc	*LC*	13.92
s2t_multiformer_s_l_cl	***LC***	**13.40**
s2t_multiformer_s_h_lc	***HC***	**13.45**
s2t_multiformer_s	$HC + LC$	13.60
s2t_multiformer_s_v2 †	$HC + LC$	13.74
s2t_multiformer_s_v3 †	$HC + LC$	13.46
s2t_multiformer_s_v4 †	$HC + LC$	13.50
s2t_multiformer_s_v5 †	$HC + LC$	13.54

Table 3: Resultados en ASR de los experimentos realizados. En la columna de "Configuración" se contemplan los casos: "**LC**" (Arquitecturas que presentan distintas configuraciones a nivel de capa), "**HC**" (Arquitecturas que presentan distintas configuraciones a nivel de head) y "**-**" (Arquitecturas sin configuración).
†: Experimento de la segunda iteración.

En general, en ASR las arquitecturas del *Multiformer* entrenadas logran resultados semejantes a los obtenidos por el modelo de referencia. Si bien esta tarea no ha sido el objeto de estudio, sería interesante realizar un futuro análisis de interpretabilidad para comprender el porqué no se aprecian las mismas mejoras que en S2T.

B Experimentación sin *Downsampling*

En la linea de estudio abierta por Alastruey et al. [2021], quienes investigaron la utilización de *Efficient Transformers* (§3.2) como alternativa al *dowsampling*, y dado que el *Multiformer* implementa mecanismos de atención más eficientes que el modelo base (§4.1), se ha creído oportuno el entrenamiento de algunas arquitecturas sin dicha reducción. En este caso, al no realizar compresión mediante capas convolucionales, el entrenamiento del modelo puede beneficiarse debido a una nula pérdida de información de la secuencia de entrada.

B.1 Detalles de experimentación

Especificaciones de implementación. Con el fin de realizar una comparación fiel con el modelo de referencia, definido en Wang et al. [2020a], los experimentos sin *downsampling* mantienen las capas convolucionales previas al *Encoder*, aunque estas no reducen la longitud de la secuencia. Esta decisión permite guardar más paridad en el número de parámetros de entrenamiento entre las arquitecturas entrenadas y el modelo de referencia y mantener el preprocesado del espectrograma.

Parámetros de entrenamiento. Se utilizan los mismos parámetros de entrenamiento empleados para los entrenamientos con *downsampling* (§5.2), en excepción del número máximo de *tokens* por *batch*, que se ha fijado en 5.000, y la frecuencia de actualización de los parámetros del sistema, que se ha compensado en 64 *batches*. De este modo, se consigue mantener una frecuencia real de una actualización de parámetros cada 320.000 *tokens* procesados. Nótese, que aunque la arquitectura del modelo de referencia sigue siendo la misma, modificar los parámetros de entrenamiento obliga a nuevo entrenamiento de esta con los parámetros actuales. De lo contrario, no se podría realiza una comparación fidedigna.

Conjunto de datos. Tal y como se ha procedido en los experimentos con *downsampling* (§5.2), las arquitecturas se han entrenado en el subconjunto de datos Inglés-Alemán incluido en MuST-C [Cattoni et al., 2021].

Arquitecturas del *Multiformer* entrenadas. Para los experimentos sin *downsampling* se han entrenado las arquitecturas de la primera iteración que han mostrado un mejor desempeño (Tabla 4). Cabe remarcar, que debido a la gran cantidad de tiempo requerido para el entrenamiento de arquitecturas sin reducción, no ha sido posible incluir experimentos con arquitecturas de la segunda iteración.

ID	Nombre			
1	s2t_multiformer_s_h_lc	$12 \times \begin{pmatrix} 2\times\ Local\,(64) \\ 2\times\ Conv\,(5,2) \end{pmatrix}$		
2	s2t_multiformer_s	$2 \times \left(4\times\ Conv\,(5,2) \right)$	$6 \times \begin{pmatrix} 2\times\ Local\,(64) \\ 2\times\ Conv\,(5,2) \end{pmatrix}$	$4 \times \begin{pmatrix} 2\times\ Full \\ 2\times\ Conv\,(7,3) \end{pmatrix}$

Table 4: Arquitecturas del *Multiformer* entrenadas sin *downsampling*. La notación para cada configuración es la siguiente: $N_{layers} \times \left(N_{heads}\times\ Attention\,(parameters) \right)$

Todas las arquitecturas han sido entrenadas durante 90 *epochs* en cada modalidad (ASR y S2T), lo que supone $\approx$ 410 horas para cada experimento, y se ha utilizando el *Encoder* preentrenado en su mejor *chekpoint* de ASR.

B.2 Resultados

En la tabla 5 se muestran los resultados obtenidos por cada arquitectura en ASR y S2T. En ella se puede observar como ninguna arquitectura del *Multiformer* supera los resultados obtenidos por el modelo de referencia en WER y BLEU. Estos resultados sugieren que el uso de capas convolucionales con *downsampling* es apropiado, debido a que realiza una primera purga de redundancia.

	Configuración	WER(↓)	BLEU(↑)
s2t_transformer_s	-	13.76	22.31
s2t_multiformer_s_h_lc	HC	14.00	21.78
s2t_multiformer_s	$HC + LC$	13.83	22.15

Table 5: Resultados ASR y S2T de los experimentos realizados sin *downsampling*. En la columna de "Configuración" se contemplan los casos: "**LC**" (Arquitecturas que presentan distintas configuraciones a nivel de capa), "**HC**" (Arquitecturas que presentan distintas configuraciones a nivel de head) y "**-**" (Arquitecturas sin configuración).

Por otro lado, también creemos que la gran cantidad de errores del tipo "*Out of Memory*" producidos durante el entrenamiento de las arquitecturas del *Multiformer* ha influido en estos resultados.[33] Al no reducir éstas la longitud de las secuencias de entrada, la GPU no es capaz de almacenar los tensores durante el entrenamiento, provocando dicho error y, por ende, entrenándose con una menor cantidad de datos.

[33]Este tipo de error ocurre cuando la GPU no dispone de espacio libre suficiente para almacenar y realizar las operaciones del entrenamiento de un *batch*, lo que provoca que ese *batch* sea descartado y se proceda al entrenamiento mediante el siguiente *batch*. En consecuencia, el error "*Out of Memory*" causa la no utilización de todos los datos disponibles para el entrenamiento.

C Historial de revisiones y registro de aprobación

Revisión	Fecha	Propósito
0	14/10/2021 – 23/01/2022	Redactado del contenido del documento.
1	27/11/2021	Revisión del documento.
2	21/12/2021	Revisión del documento.
3	14/01/2022	Revisión del documento.
4	22/01/2022	Revisión del documento.
5	23/01/2022	Revisión del documento.

Nombre	e-mail
Gerard Sant Muniesa	gerard.muniesa@estudiantat.upc.edu
Gerard Ion Gállego Olsina	gerard.ion.gallego@upc.edu
Marta R. Costa-jussà	marta.ruiz@upc.edu

Escrito por:		Revisado y aprobado por:	
Fecha	23/01/2022	Fecha	23/01/2022
Nombre	Gerard Sant Muniesa	Nombre	Gerard Ion Gállego Olsina
Posición	Autor del Proyecto	Posición	Supervisor del Proyecto