UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Sistemas Informáticos y Computación

Adaptación al dominio mediante selección de datos

Trabajo Fin de Máster

Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

AUTOR/A: Barroso Ordóñez, Martín

Tutor/a: Casacuberta Nolla, Francisco

Director/a Experimental: GARCIA MARTINEZ, MARIA MERCEDES

CURSO ACADÉMICO: 2021/2022

Resumen

Los transformers son una arquitectura neuronal que está arrasando dentro del área de la traducción automática pero uno de sus principales problemas es el gran número de muestras que necesita para su correcto funcionamiento. En este trabajo hemos afrontado el problema de la adaptación al dominio centrándonos en el reentrenamiento de modelos ya existentes para solventar la falta de muestras de entrenamiento y ahorrar el máximo tiempo de cómputo posible. Para realizar esta adaptación se han utilizado e implementado técnicas de selección de datos que utilizan modelos preentrenados para seleccionar que datos usar para nuestros reentrenamientos.

Palabras clave: Inteligencia artificial; Traducción automática; Procesamiento del lenguaje natural; Transformers; Modelos preentrenados; Adaptación al dominio; Modelos neuronales.

Resum

Els transformers són una arquitectura neuronal que està arrasant dins l'àrea de la traducció automàtica però un dels seus principals problemes és el gran nombre de mostres que necessita per al correcte funcionament. En aquest treball hem afrontat el problema de l'adaptació al domini centrant-nos en el reentrenament de models ja existents per resoldre la manca de mostres d'entrenament i estalviar-ne el màxim temps de còmput possible. Per realitzar aquesta adaptació s'han utilitzat i implemen- tat tècniques de selecció de dades que utilitzen models preentrenats per seleccionar quines dades utilitzar per als nostres reentrenaments.

Paraules clau: Intelůligència artificial; Traducció automàtica; Processament del llenguatge natural; Transformers; Models pre-entrenats; Adaptació del domini; Models neuronals.

Abstract

Transformers are a neuronal architecture that is taking machine translation by storm, but one of its main problems is the large number of samples needed for its correct operation. In this work, we have tackled the domain adaptation problem by focusing on the retraining of existing models to solve the lack of training samples and save as much computational time as possible. To perform this adaptation we have used and implemented data selection techniques that use pre-trained models to choose which data to use for our re-trainings.

Key words: Artificial intelligence; Machine translation; Natural language processing; Transformers; Pre-trained models; Domain adaptation; Neuronal models.

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

La traducción automática ha visto una gran mejora de sus resultados desde la llegada de los transformers[25], obteniéndose incluso sorprendentes mejoras en otros campos relacionados con el aprendizaje automático que nada tienen que ver con la traducción. Sin embargo, los transformers tienen un gran problema y es que son muy dependientes de los datos, necesitando una gran cantidad de estos para su correcto funcionamiento, lo cual es muy caro de obtener en el ámbito de la traducción automática. Esto ha hecho que se usen mucho los modelos preeentrenados, modelos entrenados en grandes computadores con enormes cantidades de datos para resolver una amplia variedad de problemas.

1.1 Motivación

Muchas empresas quieren que sus datos sean traducidos de forma rápida y barata por lo que recurren a motores de traducción. Estos modelos son genéricos y funcionan relativamente bien, pero si los datos que un cliente quiere traducir son muy específicos porque son de un dominio muy concreto (moda, legal, medicina o finanzas) podemos especializar nuestros motores a este tipo de datos de forma relativamente sencilla, aumentando así la calidad de la traducción, a esto se le conoce como adaptación al dominio.

Para hacer esto normalmente se utilizan datos etiquetados (frases en el idioma origen con sus correspondientes traducciones), también conocidos en el ámbito de la traducción como unidades de traducción, los cuales pueden no estar al alcance de todas las empresas o incluso que estas no tengan la cantidad de datos suficiente para que esta adaptación al dominio de los resultados que se esperan.

1.2 Objetivos

Queremos crear una serie de procesos que, a la hora de adaptar nuestros modelos a un corpus dado por un cliente, no solo utilice el corpus del cliente sino todos los datos de nuestros corpus más parecidos a los del cliente, mejorando así la adaptación de nuestro modelo a ese dominio. Esto permitirá también adaptar nuestro modelo al dominio del cliente incluso cuando este no tenga datos etiquetados con los que reentrenar el modelo, ya que solo necesitamos las frases en el idioma origen para realizar esta búsqueda sobre nuestros datos. Para estos experimentos se entrenarán modelos base de traducción, se reentrenarán, y se probarán los resultados con datos públicos, consiguiendo así que estos experimentos sean replicables.

1.3 Estructura

En el Capítulo 2 hablaremos sobre los modelos preentrenados en general para luego centrarnos en los modelos preentrenados más importantes.

Después, en el Capítulo 3 explicaremos en que se basan los transformers, como funcionan y cuales fueron sus precursores.

Tras esto, nos centraremos en el desarrollo del trabajo en el Capítulo 4, donde buscaremos datos de entrenamiento bilingües, en principio del español al inglés y viceversa, y los procesaremos para que puedan usarse para los entrenamientos, reentrenamientos y test. Tras esto entrenaremos los modelos base con los datos recopilados para posteriormente poder reentrenarlos. Para ello usaremos un transformer como arquitectura. Después, se crearán los procedimientos necesarios para, dado un conjunto de datos de reentrenamiento o a traducir (monolingües), obtener como resultado los datos que más puedan ayudarnos a la hora de reentrenar nuestros modelos. Para ello usaremos modelos preentrenados como el Universal Sentence Embeddings[5] y calcularemos la verosimilitud coseno entre las frases de entrada y nuestros datos.

Por último, se harán experimentos para ver si mejoran o no las traducciones al aplicar nuestros datos seleccionados, para ello se compararán los resultados obtenidos entre nuestro modelo base y este mismo reentrenado con los datos obtenidos por nuestra selección. Se mirará también cuantos datos seleccionar y cómo hacerlo para mejorar lo máximo posible los resultados. Todo esto se verá en el Capítulo 5.

CAPÍTULO 2
Estado del arte

Para el estado del arte nos enfocaremos principalmente en las opciones que vamos a utilizar para el desarrollo de nuestro trabajo, siendo estas las arquitecturas de transformers para la traducción, los modelos preentrenados y las distintas formas en las que se ha intentado afrontar el problema de la selección de datos.

2.1 Transformers para la traducción

Los transformers son ampliamente utilizados en la traducción automática y han sido el estado del arte prácticamente desde que se publicó su artículo[25]. La misma arquitectura mencionada en [25] con los mismo hyperparámetros que ahí se describen es la más empleada hoy en día para la traducción. Aun así, hay distintas variantes que merecen la pena destacar.

Partiendo de la arquitectura base cabría pensar que esta puede ser mejorada aumentando el número de pesos, cosa que se acaba probado con todos los modelos que surgen y con los que ya dominan el mercado. A esto se le conoce como BigTransformer y ya aparece en el mismo artículo que presentó el transformer dando mejores resultados que su versión base. Los problemas de este transformer son que ocupa más espacio, se tarda más en entrenarlo y que es más fácil que sobreentrene.

Otra de las cosas que podríamos querer probar es a hacerlo más profundo, aumentar el número de capas en lugar del número de pesos de cada capa. Esto es algo ampliamente probado con otros tipos de redes neuronales. Hay varios artículos donde se prueba esta opción con distintas profundidades y parecen que todos llegan a la conclusión de que aumentar el número de capas del codificador es mucho más beneficioso que aumentar las del decodificador. Así llegan a la idea de usar transformers de 12, 20, 30[26] e incluso 60[16] capas codificadoras manteniendo el número de capas decodificadoras (6).

Mencionar, por último, que también hay autores que han buscado mejorar el transformer en lo que a tiempo de cómputo se refiere. Un problema del transformer es que el tiempo en inferencia es cuadrático con respecto a la longitud de la cadena de entrada, lo que implica que a más larga sea la cadena de entrad más problemas tiene para procesarla. El Fastformer[27] ataca este problema haciendo que el coste pase de cuadrático a lineal. Esto por supuesto acaba haciendo que empeoren ligeramente los resultados.

2.2 Uso de datos externos en la traducción

La idea inicial de este trabajo era buscar una forma de sesgar un modelo base a unos datos específicos de un cliente, es decir, queremos buscar una forma de utilizar los datos del cliente para mejorar los resultados de nuestros modelos en esos datos. En la bibliografía consultada vimos que se habían probado distintas cosas al respecto.

En algunos artículos modificaban la arquitectura, añadiendo o quitando parámetros de los modelos en base a intuiciones y experimentos. En [9] combinaban los datos limpios que poseían más parecidos a los datos de test con la solución que devuelve el modelo en la decodificación y en [4] añadían nuevos parámetros en las capas finales que procesarían estos datos limpios similares.

En otras aproximaciones se enfocaban más en la selección de datos sin modificar el modelo. En [19] realizaban reentrenamientos en base a distintas selecciones previamente realizadas, entre las cuales se encuentra el uso de modelos preentrenados, y en [17] buscaban unidades de traducción que se parecieran a los datos de entrada, ya fuera a nivel de solapamientos de las palabras o de similitud semántica entre frases.

2.3 Modelos preentrenados

El procesamiento del lenguaje natural (PLN) es una rama de la IA que busca que las máquinas sean capaces de entender y producir el lenguaje humano. El PLN ha estado con nosotros durante décadas, pero ha experimentado un gran aumento de su popularidad desde la llegada de los modelos preentrenados, los cuales permiten implementaciones muy competentes con un esfuerzo muy reducido para los programadores.

Los modelos preentrenados para el PLN son modelos de aprendizaje profundo que han sido entrenados con grandes conjuntos de datos para unas tareas específicas de PLN. Cuando estos modelos son entrenados en grandes conjuntos de datos consiguen aprender representaciones universales de los lenguajes, lo cual puede ser muy beneficioso en tareas relacionadas con el PLN, ya que nos ahorra el tener que entrenar modelos desde 0. De esta forma pueden usarse este tipo de modelos para solucionar problemas de PLN sin necesidad de tener una gran cantidad de datos etiquetados, tener máquinas súper potentes que pueden entrenar modelos así de grandes rápidamente ni tiempo de cómputo, con los gastos que esto representa. El uso de estos modelos es cada vez más amplio ya que es más sencillo de implementar que un modelo entrenado desde 0, los resultados son mejores y requieren mucho menos entrenamiento. Debemos señalar también que muchos de estos modelos están disponibles de forma gratuita.

Hay una amplia gama de modelos transformers preentrenados para tareas de PLN tales como la clasificación de textos, respuesta a preguntas, traducción, similitud de oraciones, extracción de características de una frase... En nuestro caso queremos utilizar un modelo preentrenado para buscar oraciones que pertenezcan al domino de los datos del cliente, por lo que nos servirá con que el modelo pueda puntuar como de similares son dos oraciones que reciba como entrada.

En este apartado veremos diferentes modelos preentrenados enfocados en el ámbito del procesamiento del lenguaje natural que podrían ayudarnos a conseguir nuestro objetivo.

BERT[8] ha sido durante mucho tiempo el modelo del estado del arte para las tareas relacionadas con el PLN. Fue desarrollando por Jacob Devlin y sus compañeros en Google en 2018 y fue liberado como código abierto en marzo de 2019 en TensorFLow[1]. Este modelo del lenguaje de aprendizaje profundo basado en la parte codificadora del

transformer se caracteriza principalmente por, durante el entrenamiento de este, tener en cuenta el contexto total de la entrada.

Normalmente cuando de entrenaba un modelo como este lo que se hacía era, en entrenamiento e inferencia, tratar de adivinar la siguiente palabra de una frase incompleta dado el contexto (la parte que precede a la palabra a adivinar), este tipo de modelos son llamados autorregresivos. Sin embargo, con BERT, la propuesta fue enmascarar palabras de forma aleatoria de la entrada y luego tratar de acertarlas con la frase completa, esto es, con el contexto de la izquierda y la derecha de la palabra. Esto se entrenó para tareas como reconocimiento de entidades, segmentación de oraciones...

Este tipo de entrenamiento (basado en la enmascaración) tiene sus pros y sus contras, es cierto que las representaciones aprendidas son mejores, lo cual puede ayudar en numerosas tareas. Pero, tan bien perjudica a la tarea principal de los modelos del lenguaje, la generación de texto, ya que un modelo autorregresivo funcionará mejor a la hora de generar texto de la nada que uno acostumbrado a rellenar huecos dado un contexto por la derecha y la izquierda.

XML-R[6] es un modelo multilingüe entrenado por Facebook AI[1] basado en la enmascaración de palabras. Fue entrenado con texto monolingüe de 100 idiomas distintos y con ello superó en múltiples tareas a la versión multilingüe de BERT.

XLNet[29] es un modelo del equipo de Google AI Brain[2] que usa un nuevo método de aprendizaje no supervisado basado en permutaciones generalizadas. Con ello tratan de resolver los problemas de BERT, que aparte de no ser autorregresivo, también ignora las dependencias entre los distintos términos enmascarados dentro de una misma frase. Por ello han creado un modelo autorregresivo, que, a su vez, aprende durante el entrenamiento de los contextos completos de las frases. También utilizan el transformer-XL[7], lo que mejora el rendimiento para tareas del PLN que involucren largos contextos.

GPT-3[3] es un modelo del lenguaje auto regresivo enfocado en la producción de lenguaje que simule el lenguaje humano. Es el tercero de la línea de modelos del lenguaje GPT y ha sido creado por OpenAI[3]. Es uno de los modelos más grandes hasta la fecha y hay una gran cantidad de aplicaciones que están surgiendo a raíz del uso de este modelo. Es un modelo tan monstruosamente grande que todavía se está investigando como sacarle el máximo partido, ya que dependiendo de cómo se introduzca la entrada y de lo hábil que se sea haciéndolo se pueden obtener sorprendentes resultados. A esta técnica se le conoce como *prompting*.

USE[5] es otro modelo de Google. MUSE[28] es la versión posterior y multilingüe, que es en la que nos enfocaremos principalmente. Este modelo tiene un especial interés para nosotros puesto que las tareas principales en las que ha sido entrenado son: Búsqueda de información semántica, búsqueda de contestaciones relevantes dada una pregunta y búsqueda de traducciones más parecidas dada una oración en otro idioma. Esto es perfecto para nosotros ya que nuestro objetivo es encontrar en nuestras memorias de traducción las frases con información semántica más parecidas a las frases del cliente.

2.4 Propuesta

Combinaremos varias opciones actuales del estado del arte para la realización de nuestro trabajo.

[1] https://ai.facebook.com/
[2] https://research.google/teams/brain/
[3] https://openai.com/

Aunque existen muchas variantes del transformer y la mayoría son bastante fáciles de implementar (solo necesitamos cambiar el valor de algunos hyperparámetros), nosotros usaremos el transformer básico puesto que funciona bastante bien y nos es más sencillo de utilizar ya que únicamente necesitamos entrenar un modelo base con el que comparar nuestros resultados que, posteriormente, podamos reentrenar.

Para el uso de datos externos en la traducción hemos decidido centrar nuestro trabajo en torno a lo hecho en [19] y [17]. Nos enfocaremos en el cálculo de la similitud semántica entre los *sentence embeddings* de las frases para la selección de datos y usaremos las frases seleccionadas para el reentrenamiento de un modelo base.

Utilizaremos el MUSE[28] para calcular los *sentence embeddings* de todas nuestras unidades de traducción y de los datos del cliente.

Por último, nos apoyaremos en Faiss[11] para calcular de forma efectiva la similitud coseno entre todas las representaciones de nuestras unidades de traducción y las de los datos del cliente.

CAPÍTULO 3
Fundamentos teóricos

3.1 Precursores del transformer

Los transformers[25] son un tipo de red neuronal artificial, una arquitectura que ha ido ganando mucha popularidad en estos últimos años gracias a su increíble rendimiento. Esta arquitectura fue diseñada, en principio, para resolver problemas de traducción. Problemas en los que dada una secuencia de entrada había que devolver una secuencia de salida. Esto incluiría el reconocimiento de voz, la traducción de texto, síntesis de voz...

Por ello, para la traducción a nivel de frase, asumiendo una frase de entrada $\mathbf{x} = (x_1, \ldots, x_S)$ en el que S se corresponde con el número total de términos de entrada, y una frase de salida $\mathbf{y} = (y_1, \ldots, y_T)$, en el que T comprende el número total de términos de salida, la distribución condicional puede describirse como:

$$\hat{\mathbf{y}}_1^{\hat{T}} = \arg\max_{T, \mathbf{y}_1^T} \prod_{t=1}^{T} Pr_\theta(y_t | y_1^{t-1}, c(x_1^S)) \tag{3.1}$$

Donde y_t es el término actual a traducir, que depende de los términos anteriormente traducidos por el modelo y_1^{t-1}. La función c se aplica a la frase de entrada $\mathbf{x}_1^S$ y emplea los parámetros del modelo θ.

Para resolver este tipo de problemas se necesita un modelo que sea capaz de memorizar o tener en cuenta la entrada mientras genera la salida, ya que muchas veces, en el caso de una traducción, el contexto es muy importante, por ejemplo, para mantener la concordancia del género y del número.

Previamente se trataban de resolver este tipo de problemas con redes neuronales recurrentes, las cuales se crearon para tratar con información secuencial. Para la traducción, el mejor rendimiento que se consiguió con estas redes fue con la arquitectura codificadora decodificadora que puede verse en la Figura 3.1, en esta arquitectura el primer término de entrada generaba una representación intermedia, que se utilizaría para que, al procesar el siguiente término, se tuviera un contexto del anterior. Esto se haría con todos los términos de la entrada de izquierda a derecha hasta que terminara el procesamiento de la cadena. Esta sería solo la parte codificadora de la red, la cual nos da una representación intermedia que resume toda la entrada. Tomando esto como entrada, el decodificador empezará a generar la traducción que le servirá también de entrada a los siguientes pasos del decodificador.

Las redes recurrente codificadoras-decodificadoras se entrenan con el fin de maximizar la probabilidad logarítmica condicional dado un conjunto de pares de entrenamiento definidos como $D = \{(x_1, y_1), \ldots, (x_T, y_T)\}$ de la siguiente forma:

$$\arg\max \sum_{t=1}^{T} \log p\left(y_t | y_{<t}, \mathbf{x}\right) \tag{3.2}$$

Donde $\mathbf{x}$ es la frase de entrada, y_t es el término a traducir en el instante t y $y_{<t}$ es la traducción ya realizada que precede al término y_t. Estas redes neuronales tiene como entrada la frase en el lenguaje fuente u origen, y su salida es la frase que contiene la traducción en el lenguaje destino u objetivo.

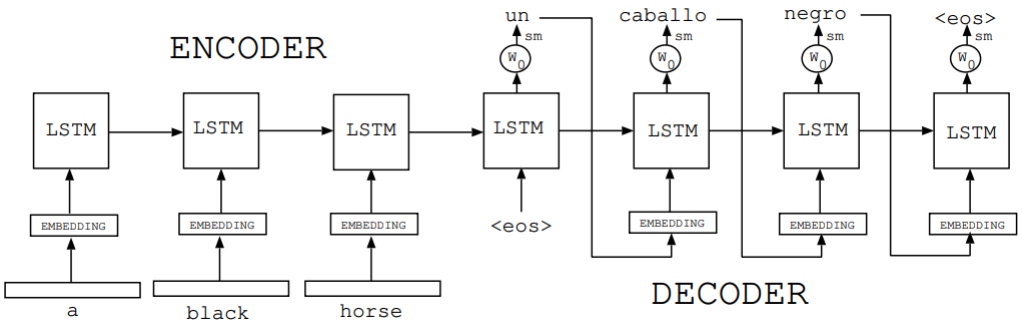

Figura 3.1: Red recurrente codificadora decodificadora.

El problema de este tipo de redes era que no lograban capturar las dependencias que se dan en secuencias moderadamente largas. Otro problema de este tipo de redes era que costaba mucho paralelizar los entrenamientos y la inferencia, ya que funcionaba como una cadena, donde solo se puede obtener el resultado del siguiente paso mirando todos los resultados anteriores, lo cual hacía que los entrenamientos fueran muy largos o que no se pudieran entrenar todo lo que se quería.

Así, surgieron los mecanismos de atención, los cuales fueron concebidos con la idea de ayudar a las redes recurrentes a que pudieran funcionar correctamente con secuencias más largas. Con estos mecanismos de atención, las redes podían centrarse en una combinación de las representaciones procesadas de cada término de la entrada además de en una representación que describía toda la entrada hasta ese momento.

Ahora, la red, mediante este mecanismo de atención, podía centrarse en ciertas partes de la entrada, en mayor o menor medida, a la hora de traducir cada término. En la Figura 3.2 puede verse el resultado del mecanismo de atención en la traducción de una frase del francés al inglés, donde los valores más brillantes son más altos e indican que hay una mayor atención en esa palabra y los más oscuros son más bajos e indican que hay una menor atención en esa palabra. Podemos ver como en este caso sigue una diagonal, ya que las palabras a traducir están correctamente alineadas salvo en algunos casos donde hay permutaciones al traducir algunos conjuntos de términos. También puede verse como, para algunos términos, hay varios valores con importancia. Por ejemplo, al traducir *en* por *in* se fija también ligeramente en *August*. Esto facilita enormemente el aprendizaje en estas redes.

Aun así, esto seguía sin resolver el problema de la paralelización. Para tratar de solventarlo se utilizaron redes convolucionales. Las redes convolucionales son el tipo de red más usado en los últimos tiempos en problemas que están relacionados con imágenes. Algunos ejemplos del uso de estas redes en problemas relacionados con secuencias fueron Wavenet[23] y Bytenet[13]. La razón por la que se optó por usar redes convolucionales es porque estas son fácilmente paralelizables y porque las distancias en estas redes entre el término de salida y cualquier término de entrada es del orden de log(N), tal y como

[1]https://towardsdatascience.com/evolution-of-natural-language-processing-8e4532211cfe

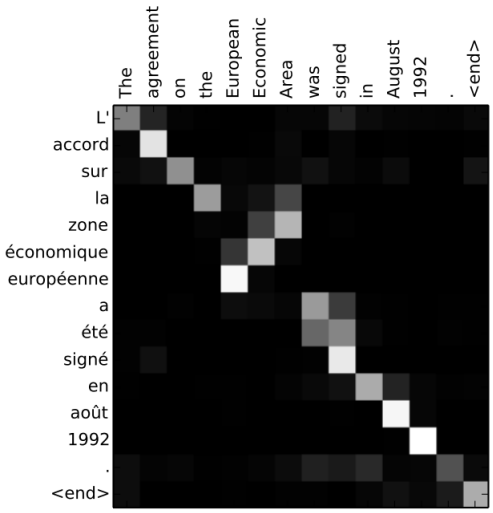

Figura 3.2: Valores de atención en la traducción de una frase traducida del inglés al francés.[1]

puede apreciarse en la Figura 3.3, donde N es el tamaño del árbol generado para conseguir ese término de salida dado esos términos de entrada. Esto mejora enormemente la distancia entre entrada y salida que había en las redes recurrentes.

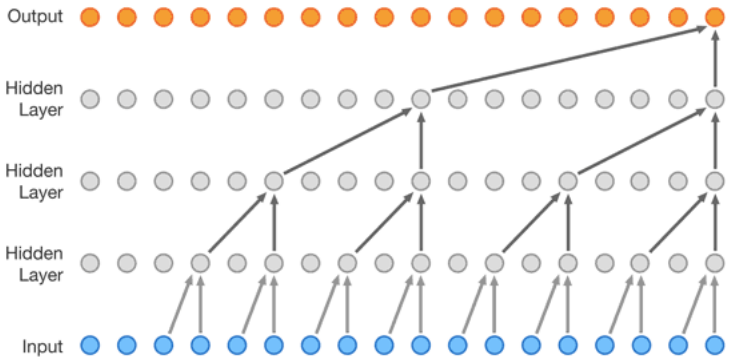

Figura 3.3: Distancia entre las palabras de la entrada y la salida en una red convolucional usada para la traducción a medida que la información va avanzando por la red.[2]

Estas redes convolucionales[18] no resolvían del todo bien el problema de las dependencias a la hora de traducir, por ello se creó el transformer, el cual usaba el mecanismo de atención que tan bien funcionaba en las redes recurrentes para sustituir el procesamiento en cadena que esta hacía.

3.2 Transformer

En principio, la arquitectura de un transformer puede recordar mucho a la de una red recurrente codificadora decodificadora tal y como puede verse en la Figura 3.4, ya que ambas siguen este esquema de codificar la entrada para luego decodificar la salida. En

[2]https://www.researchgate.net/publication/348542282_Google_Duplex_-_A_Big_Leap_in_the_Evolution_of_Artificial_Intelligence

este caso hay varios codificadores y decodificadores conectados en serie y la salida del último codificador está conectada a todos los decodificadores, en lugar de solo al primero.

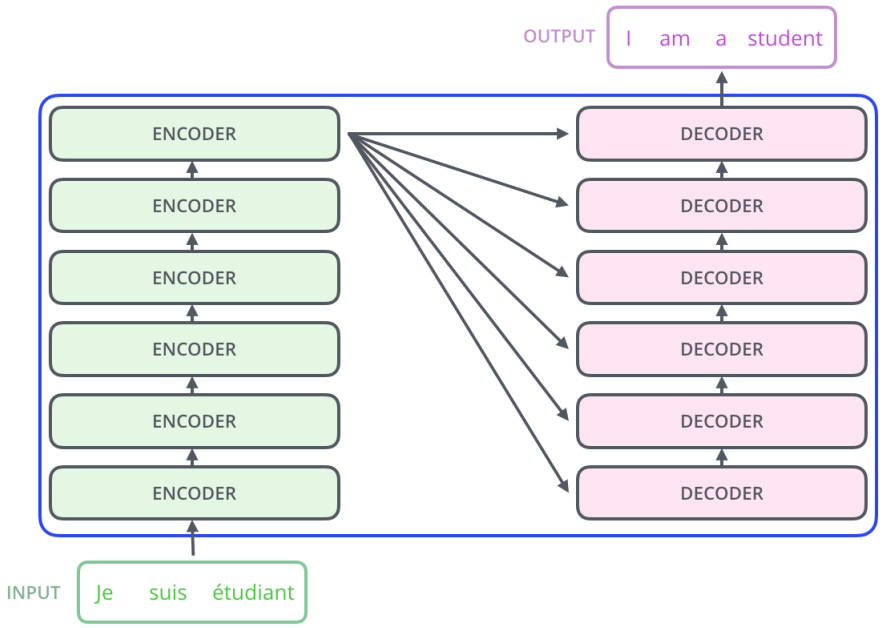

Figura 3.4: Arquitectura de un transformer encoder-decoder de 6 capas para la traducción.[3]

La novedad principal de los transformers y lo que hace que solvente los problemas mencionados anteriormente es la auto-atención. La auto-atención no es más que el mecanismo de atención que vimos anteriormente pero que, en lugar de mirar la entrada completa para adivinar un término de la salida, se aplica a la entrada para ver como cada término de esta se relaciona con el resto de términos de la misma. La auto-atención puede aplicarse en paralelo para cada término que tenga en su entrada, solventando así el principal problema de velocidad de las redes recurrentes.

La atención en los transformers se calcula de la siguiente forma. Se realiza un conjunto de consultas simultáneamente encapsuladas en una matriz Q. Por otra parte, las llaves y los valores se encapsulan en las matrices K y V de dimensiones d_k y d_v, respectivamente. La formulación de dicho cálculo es la siguiente:

$$Atencion\,(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.3}$$

La Figura 3.5 ilustra bastante bien la auto-atención, podemos ver como la puntuación de la auto-atención, representado por el color de las flechas, es mayor para los términos que se relacionan más con *kicked* (que es el término al que se le está aplicando la auto-atención en este ejemplo).

Este mecanismo aparece tanto en los codificadores como en los decodificadores. Decir que en los decodificadores se enmascaran los términos que siguen al término al que se le aplica la auto-atención porque en las traducciones no tendremos los términos siguientes hasta que los generemos. En la Figura 3.6 puede verse la arquitectura simplificada de los codificadores y decodificadores. La atención codificadora decodificadora del decodificador no es más que el mecanismo de atención de las redes recurrentes adaptado a esta nueva arquitectura.

[3]https://programmerclick.com/article/30221772811/
[4]https://www.cnblogs.com/wangxiaocvpr/p/10544185.html

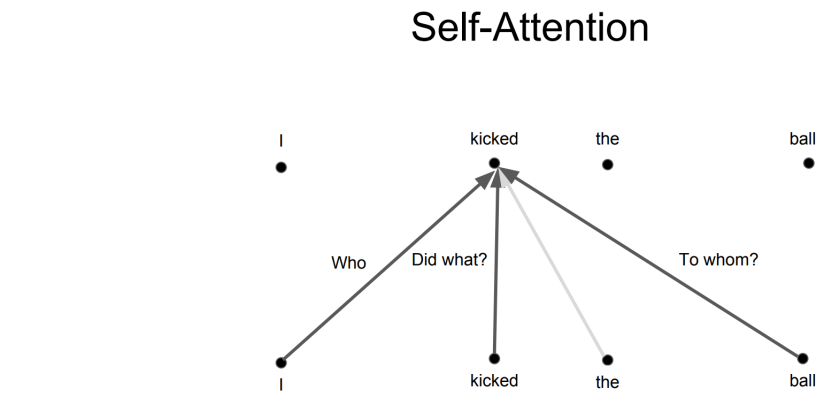

Figura 3.5: Auto-atención de la palabra *kicked* en una frase.[4]

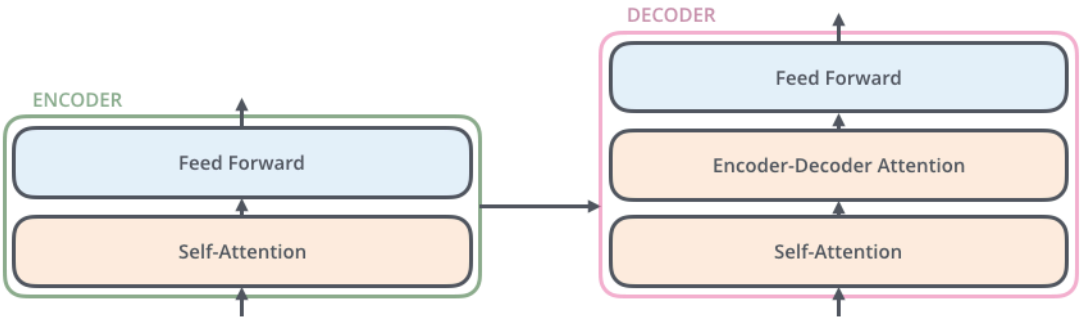

Figura 3.6: Estructura del codificador y decodificador de los transformers.[5]

Hay que decir también que los transformers normalmente usan lo que se conoce como atención multicabezal. Esto es que hay más de 1 dimensión por término en la atención, en concreto tantas dimensiones como números de cabezas. Esto suele dar mejores resultados ya que da mayor número de parámetros que entrenar a la red por lo que acaba teniendo más flexibilidad. La intuición que subyace al uso de varias cabezas es que cada una se especializará en una forma distinta de relacionar los términos de las frases.

$$Multicabezal\,(Q,K,V) = Concat\,(cabezal_1,\ldots,cabezal_h)\,W^o \tag{3.4}$$

donde

$$cabezal_i = Atencion(QW_i^Q, KW_i^K, VW_i^V) \tag{3.5}$$

Donde las proyecciones son matrices de parámetros $W_i^Q \in \mathbb{R}^{d_{modelo}*d_k}$, $W_i^K \in \mathbb{R}^{d_{modelo}*d_k}$, $W_i^V \in \mathbb{R}^{d_{modelo}*v}$, $W^o \in \mathbb{R}^{d_{modelo}*hd_v}$, los cuales son aprendidos durante la etapa de entrenamiento.

Otro factor a tener en cuenta es la posición de cada término en la entrada. Esta se codifica con una función antes de que entre en la red.

Aunque hasta el momento hayamos hablado de palabras, lo que realmente reciben como entrada y producen como salida los transformers y las otras arquitecturas de las que hemos estado hablando son valores numéricos. En un principio se utilizaba lo que se conoce como *One hot encoding*, un método muy sencillo que codifica cada palabra del vocabulario como un vector de ceros con un solo 1, donde el 1 indica de que palabra se trata. El problema de esta codificación es que necesitamos un vector de la talla del

[5] https://programmerclick.com/article/56951172757/

vocabulario para representar cada palabra y que la palabra *rey* se parece lo mismo a la palabra *reina* que la palabra *mesa*, puesto que la única diferencia es la posición del 1 en el vector.

El enfoque más utiliza actualmente es el uso de *word embeddings*, los *word embeddings* son representaciones de palabras mediante vectores densos. Se utilizan muchos menos valores para representar una palabra y, además, cada valor aporta información. Los *word embeddings* se consiguen gracias a modelos del lenguaje que aprenden la relación entre las palabras debido a que procesan una gran cantidad de texto monolingüe, obteniendo resultados tan sorprendentes como que el *word embedding* de *rey* sea muy parecido al de *reina* sumándole el de *hombre*. Los transformers también pueden aprender los *word embeddings* de las palabras durante el entrenamiento, no necesitando usar *word embeddings* externos.

Por último, queremos mencionar que actualmente se trabaja a nivel de subpalabras y no de palabras. Esto hace que nunca nos encontremos en inferencia palabras que no se vieron durante el entrenamiento, ya que normalmente uniendo subpalabras podemos formar todas las palabras existentes. También mejora los resultados ya que muchas veces hay partes de distintas palabras que coinciden y tienen un significado similar, por ello cuando se habla de la entrada y la salida de estos sistemas normalmente se habla de tokens, los cuales antes eran palabras y en los últimos tiempos suelen ser subpalabras. Estas subpalabras se suelen calcular con algoritmos como el BPE[22] y el SentencePiece[15], estos procesan los datos de entrenamiento para crear el conjunto de subpalabras que utilizará la red. Antes de que una palabra entre a la red será separada en subpalabras, la salida de la red serán subpalabras y estas se usaran para crear las palabras normales.

CAPÍTULO 4
Solución propuesta

En este capítulo presentaremos el diseño de la solución propuesta, tras esto hablaremos de la tecnología que se ha utilizado para su desarrollo y, finalmente, describiremos como se ha desarrollado.

4.1 Diseño de la solución

Es los siguientes apartados veremos los componentes principales de nuestra solución. Empezaremos con una visión general de estos componentes para después explicarlos en detalle.

4.1.1. Arquitectura del Sistema

Nuestra solución consistirá en pasar todos nuestros datos del idioma origen en una dirección (por ejemplo, los datos en español de la dirección español inglés), tanto los del cliente como los propios, por el modelo preentrenado seleccionado, en nuestro caso el MUSE. Esto nos dará una representación numérica de cada una de las frases.

Esta representación numérica se usará para hacer el cálculo de la similitud coseno entre las frases del cliente y las nuestras. Para ello, ya que contamos con una enorme cantidad de datos, usaremos algoritmos y herramientas especializadas en esta tarea. En nuestro caso usaremos Faiss[12], que nos devolverá los N valores más altos que le indiquemos de similitud coseno y a que oración les corresponde.

Una vez tengamos estas correspondencias podremos construir nuestro corpus de reentrenamiento como gustemos, pudiendo elegir un umbral mínimo de similitud coseno para incluir la oración, elegir las dos frases con mayor similitud coseno para cada consulta, etc.

El último paso será reentrenar nuestro modelo base con los datos seleccionados.

Una representación gráfica de este procedimiento puede verse en la Figura 4.1

4.1.2. Diseño Detallado

En esta sección hablaremos de la arquitectura explicada en la sección anterior de forma más detallada. Esta puede observarse en forma de diagrama en la Figura 4.2

La idea es buscar de entre nuestros datos los más parecidos a los del cliente, para ello usaremos MUSE, un modelo preentrenado entrenado, entre otras cosas, en *Question*

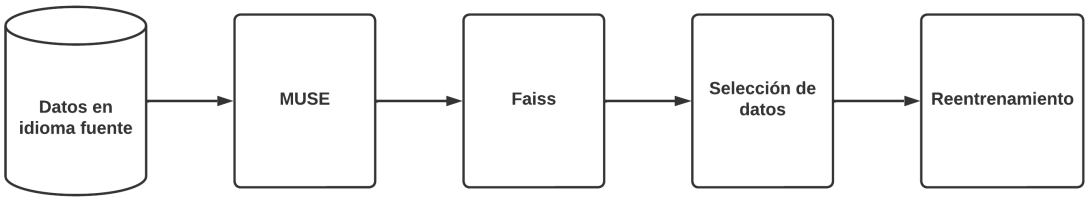

Figura 4.1: Arquitectura propuesta.

Answering y similitud semántica, por lo que la representación que nos dará debería ser útil para nuestra tarea.

MUSE recibe texto plano como entrada y nos da como resultado una representación numérica de 512 números decimales. Esta representación, conocida normalmente como *sentence embedding*, resume la información del texto que ha recibido como entrada y es la que nos servirá para saber que otras cadenas son similares a esta. Usaremos MUSE para la extracción de características de nuestros datos, ya que lo único que queremos es conseguir los *sentence embeddings* de estos.

Por lo tanto, extraeremos los *sentence embeddings* tanto del origen de nuestros datos bilingües como del origen de los datos del cliente (si es que nuestro cliente nos ha proporcionado datos bilingües), en este caso supondremos que están en español. Si el cliente nos proporcionara datos monolingües extraeríamos los *sentence embeddings* de la misma forma. Como nuestros datos no varían esto solo tendremos que repetirlo en el futuro con datos nuevos de clientes y no con los nuestros. En caso de que obtuviéramos nuevos datos que añadir a los nuestros solo tendríamos que sacar los *embeddings* de los nuevos datos y juntarlos con los antiguos.

Tras esto utilizaríamos Faiss para realizar la búsqueda de las frases más similares. Esta búsqueda se divide en 2 partes.

La primera es la creación de índices de búsqueda, cuando tenemos una gran cantidad de datos, como es nuestro caso (hasta 20 millones de unidades de traducción), normalmente se preprocesan estos datos para que posteriormente las búsquedas en estos sean más rápidas, a esta estructura donde se buscará se le llama índices de búsqueda. Para la creación de estos índices se necesitarán los *embeddings* de nuestros datos, ya que es lo que queremos comparar. Estos índices se crearán para la búsqueda de la similitud coseno y, al igual que en el paso anterior, no será necesario volver a hacer este procedimiento en el futuro puesto que nuestros datos serán siempre los mismos. En caso de tener que obtener nuevos datos esta vez sí que tendríamos que recalcular todos los índices de nuevo.

El segundo paso será realizar la búsqueda en estos índices, para ello necesitaremos cargar los índices calculados anteriormente, los *embeddings* de los datos del cliente y especificar el número N de valores que queremos encontrar para cada oración del cliente. Una vez tengamos todo esto Faiss realizará la búsqueda de forma muy rápida y en paralelo, ya sea utilizando las GPUs disponibles o los procesadores (según lo que le indiquemos). El resultado será dos matrices en formato Numpy, una con los N índices que corresponden a las frases de nuestros datos para cada oración del cliente y otra con las similitudes coseno entre nuestras frases y las del cliente. Si a la hora de realizar la búsqueda elegimos un N lo suficientemente alto no será necesario volver a realizar esta búsqueda con los datos de este cliente, ya que podremos variar el número de frases a escoger para cada oración del cliente simplemente cogiendo los primeros N' índices de la matriz resultado.

El siguiente paso será decidir qué datos seleccionar en base a la información que tenemos disponible. Podemos decidir un umbral mínimo de similitud coseno, un número

máximo o mínimo de frases para cada oración del cliente, una longitud mínima o máxima de las frases escogidas o incluso una combinación de todas estas características. Esto se hará con un sencillo *script* de Python. Para ello necesitaremos tener tanto las matrices de índices como las de similitud coseno calculadas anteriormente, nuestras unidades de traducción y los datos en el lenguaje origen (si es que queremos que la longitud de estas frases sea un factor determinante).

Por último, tras aplicar los criterios de selección deseados, obtendremos un conjunto de unidades de traducción, con estas unidades y los datos bilingües del cliente (si este nos ha proporcionado datos bilingües) reentrenaremos nuestro modelo base.

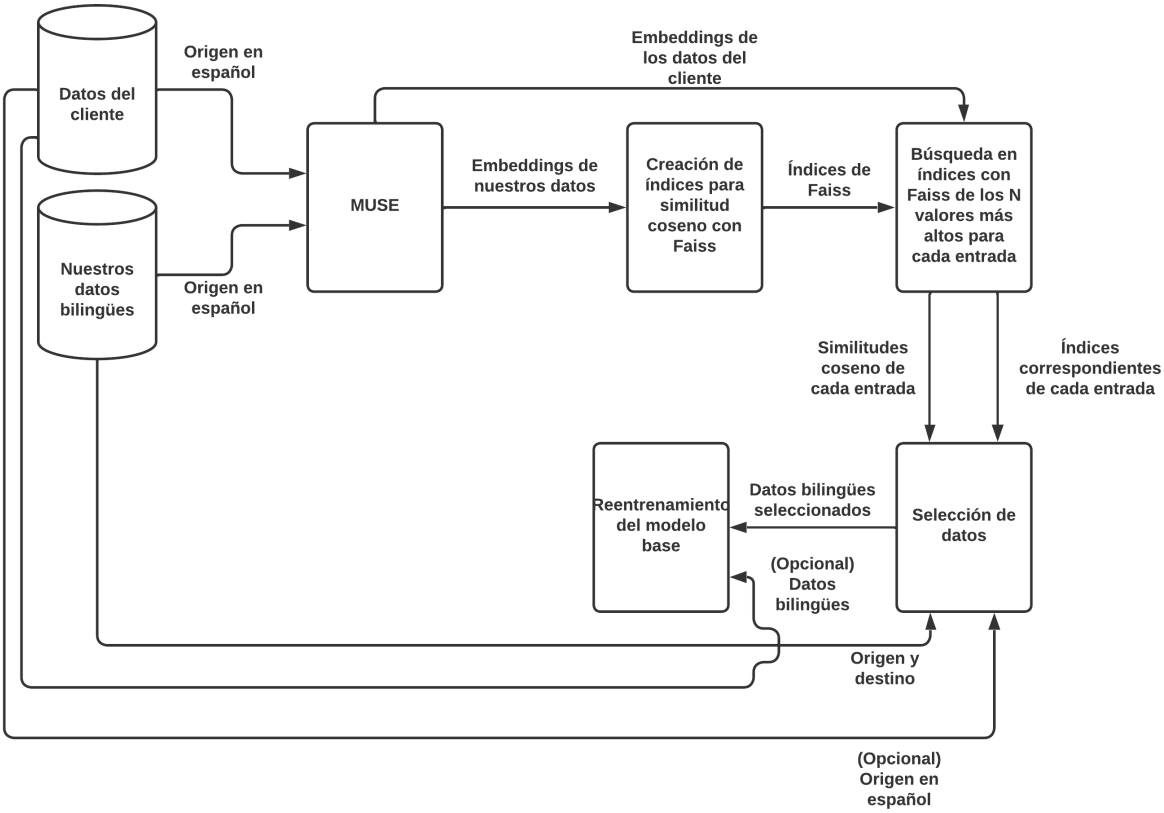

Figura 4.2: Arquitectura en detalle.

4.2 Tecnología utilizada

La totalidad del código ha sido escrito en Python3[24].

Para los entrenamientos de los modelos base y los reentrenamientos de estos se ha utilizado el software de nuestra empresa. Estos entrenamientos y reentrenamientos utilizan OpenNMT[14]. OpenNMT es un ecosistema de código abierto enfocado principalmente en la traducción automática y el aprendizaje de secuencias. Fue iniciado en diciembre de 2016 por el grupo de procesamiento del lenguaje neuronal de Harvard[1] y por SYSTRAN[2]. Es utilizado tanto para aplicaciones industriales como para proyectos de investigación. En nuestro caso usamos la arquitectura del transformer de OpenNMT y la mayoría de los parámetros que recomiendan. Tiene 2 versiones, una de Tensorflow[1] y otra de Pytorch[20], nosotros utilizaremos la segunda.

[1] https://nlp.seas.harvard.edu/
[2] https://www.systran.net/en/translate/

También utilizamos Tensorflow y Tensorflow-hub[3]. Tensorflow es una plataforma desarrollada por Google de código abierto que facilita la creación de modelos de aprendizaje automático. Se utiliza tanto en investigación como en proyectos industriales. Nosotros lo utilizaremos únicamente para hacer uso de Tensorflow-hub, un repositorio de modelos de aprendizaje automático entrenados y listos para utilizar. Gracias a esto podremos usar el MUSE de forma sencilla.

Faiss es otro de los recursos que usaremos. Se trata de una librería desarrollada principalmente por el grupo de desarrollo de IA de Meta[4]. Esta librería se utiliza principalmente para la búsqueda eficiente de similitudes entre vectores y para el *clustering* de vectores densos. Permite realizar búsqueda entre grandes cantidades de datos que no quepan en la memoria RAM y utilizar incluso las GPUs para estas búsquedas.

Por último, queríamos mencionar que también usaremos Numpy[10]. Numpy es una librería de Python que facilita la creación de vectores y matrices de gran tamaño y número de dimensiones, nos facilita la realización de operaciones entre ellos y guardar y cargar este tipo de datos. Los resultados tanto de Faiss como de MUSE vendrán dados en vectores/matrices de Numpy o en estructuras compatibles con este formato, lo cual hará que sea fácil de tratar con estos datos y guardarlos para poder cargarlos posteriormente.

4.3 Desarrollo de la solución propuesta

Puesto que queríamos aprovechar los datos del cliente para aumentar el rendimiento de nuestros modelos empezamos buscando bibliografía al respecto. Tras esto decidimos centrar nuestro trabajo en torno al reentrenamiento de modelos base y la selección de datos como hemos mencionado anteriormente.

Para buscar la similitud semántica entre frases decidimos hacer como en [17] y en [19], escoger un modelo preentrenado para procesar nuestras frases para luego calcular la similitud coseno de los *embeddings* resultantes. Así, empezamos a buscar distintos modelos preentrenados para ver cuál podría funcionar mejor para nuestra tarea. Finalmente acabamos optando por usar MUSE [28] debido a que es multilingüe y que ha sido entrenado en tareas que favorecen su desempeño en la búsqueda de similitud semántica, tal y como hemos comentado en secciones anteriores.

Una vez decidido el modelo a usar teníamos que ver el cómo utilizarlo. Investigando descubrimos que el modelo se encontraba en Tensorflow-hub, lo cual facilitaba muchísimo su uso. En pocas líneas de código podíamos procesar todas nuestras frases y conseguir sus respectivos *sentence-embeddings*, que es para lo único que necesitamos el MUSE.

Finalmente, partiendo de todo esto como base del trabajo, empezamos a buscar datos con los que entrenar un modelo base. Tras esto se hará la selección de datos sobre los datos de entrenamiento o sobre otro conjunto de datos que pueda ser similar a los del cliente y se reentrenará este modelo base con los datos de la selección realizada.

4.3.1. Búsqueda de datos y entrenamiento del modelo base

Para empezar a hacer experimentos necesitamos datos, hay que buscar los datos que usaremos para entrenar los modelos base, y en los cuales se realizará la búsqueda, y los datos que usaremos como datos proporcionados para el cliente. Los datos del cliente los teníamos claros, íbamos a usar los datos de Europeana, Europeana es un proyecto europeo en el que estamos trabajando que busca mejorar la accesibilidad de la herencia

[3] https://www.tensorflow.org/hub
[4] https://ai.facebook.com/

cultural europea traduciendo los documentos relacionados con esta temática desde todas las lenguas europeas oficiales al inglés. Empezaríamos a hacer experimentos con un modelo que tradujera del español al inglés, lo que nos haría tener 36.460 pares de frases de Europeana para simular los datos del cliente. Por último, teníamos que obtener los datos de entrenamiento, para ello decidimos usar los datos del WMT13[2] (último WMT donde se usó el inglés español). WMT es la principal competición de traducción automática y de la investigación relacionada con esta, suelen abrir al público una gran cantidad de datos para las distintas tareas que proponen cada año. Decidimos realizar nuestros experimentos para la traducción del español al inglés y viceversa, por lo que escogimos los datos de inglés español de Europarl v7[5], Common Crawl corpus[6], Tilde MODEL corpus[7] y ParaCrawl v9[8]. En la Tabla 4.1 se muestran cuantas frase contenía cada corpus.

	Europarl v7	Common Crawl	Tilde MODEL	ParaCrawl v9
Millones de frases	1.9	1.8	3.8	270
Millones de palabras	-	-	173	4374
Espacio en gigabytes	0.62	0.87	1	24

Tabla 4.1: Estadísticas de los corpus bilingües seleccionados para entrenar.

Estos datos no venían todo lo ordenados que nos hubiera gustado, los corpus venían en distintos formatos y algunos incluso incluían varios documentos en distintos formatos (bilingüe, tmx, tsv ...). Teníamos que pasar primero todos los ficheros a un formato común para poder utilizarlos. Decidimos pasarlo todo a tmx, un estándar XML para el intercambio de memorias de traducción. Una vez pasados todos los documentos a tmx podríamos empezar a limpiar nuestros datos y entrenar nuestros modelos.

Antes de entrenar nuestros modelos bases tenemos que elegir que arquitectura utilizar, que hiperparámetros y que preprocesos aplicar a nuestros datos de entrenamiento. Aplicaremos una serie de preprocesos a los datos que modificarán estos ligeramente para mejorar su calidad, una serie de validadores eliminarán unidades de traducción que cumplan una serie de condiciones y se aplicarán postprocesos que arreglaran ligeramente las traducciones generadas por el modelo. Todos estos procesos y los hyperparámtetros usados podrán encontrarse en los Apéndices A.1 y A.2 respectivamente.

Una vez que todo esté decidido, solo tendremos que entrenar nuestro modelo base con los datos que tenemos y guardarnos los datos después del proceso de limpieza para poder buscar en ellos posteriormente. Tuvimos un problema con los datos de ParaCrawl v9, eran demasiados datos y prolongaba demasiado el proceso de limpieza, a esto había que sumarle que no es un corpus demasiado limpio ya que se obtiene de extraer datos de internet de forma automática. Al final esto nos hizo acabar prescindiendo de este corpus.

A medida que fuimos haciendo experimentos fuimos ampliando también nuestros conjuntos de datos tanto del cliente como en los que realizar la búsqueda. Haciendo experimentos con datos de TAUS, EMEA, NTEU y WikiMatrix. En TAUS se dedican principalmente a vender datos, ya sea un dominio concreto o general. Debido a un proyecto de nuestra empresa relacionado con el Covid-19 disponemos de datos de dominio médico para hacer experimentos. Estos serán los únicos datos no libres que usaremos. EMEA es un corpus de datos hecho con los documentos de la Agencia de Medicina Europea, NTEU es un conjunto de datos de un proyecto europeo de creación de motores de traducción para las lenguas europeas y WikiMatrix es un corpus de datos paralelos de frases extraídos de Wikipedia por Facebook AI mediante técnicas de PLN.

[5]https://www.statmt.org/europarl/
[6]https://commoncrawl.org/
[7]https://opus.nlpl.eu/TildeMODEL.php
[8]https://opus.nlpl.eu/ParaCrawl.php

La idea principal es procesar todas las frases que se usen en el entrenamiento del modelo base para, posteriormente, poder comparar estas frases con las del cliente. Para saber que frases se van a usar en el entrenamiento de este modelo primero necesitamos entrenar este modelo, ya que muchas frases serán descartadas en los procesos de limpieza que se realizarán antes de los entrenamientos.

En el momento que ya teníamos un modelo que tradujera en una dirección, del español al inglés, y un conjunto de datos limpios en los que realizar la búsqueda, ya podíamos empezar a hacer las pruebas y desarrollar nuestro proceso de selección de datos.

4.3.2. Procesamiento de datos limpios y de datos del cliente

Una vez tenemos los datos limpios que compararemos con los datos del cliente, podemos empezar a procesar las frases fuente de ambos conjuntos de datos y obtener sus *embeddings*.

Lo primero será hacer funcionar el MUSE, para ello tendremos que realizar la instalación de Tensorflow y Tensorflow-hub con todas sus dependencias en el entorno que vayamos a utilizar, en nuestro caso un entorno de Anaconda. Esto fue bastante complicado, ya que había muchos problemas de dependencias difíciles de solucionar, el principal problema que tuvimos era que Tensorflow no conseguía localizar nuestras GPUs, las cuales son indispensables para que la extracción de los *embeddings* se haga a una velocidad admisible. Finalmente conseguimos arreglar nuestro entorno y que este reconociera nuestras GPUs correctamente.

A la hora de procesar tantos datos con el MUSE debemos de separar los datos en distintos *batches* para no quedarnos sin espacio en nuestras GPUs. Dependiendo del tamaño del *batch* y de los datos a procesar las ejecuciones fallaban o no, había datos con los que un tamaño de *batch* de 128 funcionaba bien y otros con los que teníamos que usar un tamaño máximo de *batch* de 32, creemos que esto se debe a que, por casualidad, se agrupaban muchas unidades de traducción muy largas en un mismo *batch*. Por ello decidimos procesar únicamente las unidades de traducción cuyas frases fuente tuvieran menos de 5000 caracteres, evitando así posibles colapsamientos de la memoria. También decidimos guardar los datos en local cada 10 *batches* en formato Numpy para no tener que repetir el proceso entero si ocurría cualquier problema durante la ejecución. Decidimos procesar un máximo de 6 millones de unidades de traducción ya que Numpy muchas veces era incapaz de mantener tantos datos en memoria y hacía que fallara la ejecución. Una vez controlados todos los posibles errores y problemas pudimos procesar todos nuestros datos y guardarlos en un archivo Numpy para posteriormente calcular las similitudes coseno.

4.3.3. Selección de datos similares

Para el cálculo de las similitudes primero tratamos de hacerlo nosotros mismos usando Numpy. Rápidamente nos dimos cuenta de que esta opción no era viable ya que sino tardaríamos meses en ejecutarlo una sola vez. Esto se reduciría bastante si paralelizáramos el código, pero aun así tardaría más de lo que podemos esperar. Tras investigar al respecto acabamos dando con Faiss[12], una librería que hace estos cálculos a una velocidad sorprendente, que paraleliza la carga entre todos los núcleos disponibles, puede ser lanzado en todas las GPUs disponibles si así se desea y es muy fácil de utilizar. Con esta librería podíamos crear índices de búsqueda a partir de los datos que quisiéramos que fueran invariantes, como son en nuestro caso los datos de entrenamiento. Una vez creados estos índices solo tendríamos que buscar que datos del cliente eran los más similares.

Nos pusimos manos a la obra y creamos estos índices para el cálculo de la similitud coseno con los *embeddings* de los datos que usamos para el entrenamiento, crear estos índices para más de 6 millones de datos solo llevó un par de minutos. Tras esto solo teníamos que realizar la búsqueda, para ello solo teníamos que decidir qué número de frases similares extraer de cada oración del cliente y Faiss se encargaría de devolvernos los índices y las similitudes coseno en una matriz de Numpy donde las columnas serían las frases más parecidas a cada frase de del cliente, que serían las filas. Para la búsqueda en los índices de 6 millones de frases de las 3 frases con mayor similitud coseno de cada una de las 36 mil frases del cliente solo tuvimos que esperar 5 minutos.

Al tener estas matrices podemos iterar sobre los datos de entrenamiento y los del cliente e ir escogiendo las frases que cumplan los requisitos que queramos. Podemos escoger las frases que se encuentren entre un umbral mínimo y máximo, las que se encuentren entre una longitud mínima y máxima, las N frases con mayor similitud coseno para cada oración del cliente, etc. Una vez seleccionadas las frases en torno a los criterios deseados solo tendremos que juntarlas con su correspondiente traducción y usar estos datos para reentrenar nuestro modelo base. Decir que antes de hacer más experimentos nos aseguramos de que todo lo hecho hasta el momento estuviera correctamente eligiendo un umbral muy alto, 0.9 (ya que el máximo de la similitud coseno es 1), y viendo las correspondientes selecciones. Las frases elegidas eran prácticamente idénticas como puede verse en el Apéndice A.3.1. Algunos ejemplos de frases no tan parecidas pueden apreciarse en el Apéndice A.3.2, donde utilizamos un umbral de 0.5.

Una vez tengamos una selección de las frases pasaremos a reentrenar el modelo base y a evaluar los resultados. El reentrenamiento consistirá en hacer 3 épocas sobre todos los datos seleccionados, es decir, entrenaremos el modelo de forma que pase 3 veces por todos los datos seleccionados. Una vez reentrenado el modelo sacaremos los resultados pasándole un test previamente separado. Este test está formado por un 20 % de las frases del conjunto de Europeana seleccionado de forma aleatoria. El 80 % restante de Europeana también ha sido guardado para futuros experimentos donde queramos reentrenar con datos reales del dominio. Más tarde, cuando usamos otros conjuntos de datos distintos como datos del cliente, seleccionamos 1000 frases del conjunto de datos para usarlas como test.

Todo el código escrito para la búsqueda de datos está separado por módulos. Hay 5 *scripts* principales, 2 encargados de la parte de la extracción de *embeddings*, donde el primero extrae los fragmentos de *embeddings* usando el MUSE y el segundo junta estos fragmentos para formar los archivos Numpy finales. Otros 2 *scripts* se encargan del uso de Faiss, el primero crea los índices mientras que el segundo busca en ellos las frases más similares a las del cliente. Y por último, el *script* que hace la selección de datos para el reentrenamiento usando los resultados anteriores y lo hiperparámetros que le pasemos. Todo el código está diseñado para lanzarse desde la misma carpeta e irá creando distintos directorios donde almacenará los resultados que luego usarán los siguientes *scripts* hasta llegar a la carpeta *fine-tuning*, donde estarán los datos bilingües finales. Creando un *script* que sirva de lanzador donde se establezcan las distintas opciones de los distintos *scripts* puede lanzarse todo con solo una orden por consola. Esto facilita enormemente el entendimiento del código desarrollado y la utilización de este para realizar los futuros experimentos.

CAPÍTULO 5
Experimentos

Este capítulo hablará de los distintos experimentos que realizamos para comprobar el funcionamiento de nuestro método de selección y se enseñarán los resultados obtenidos en función de métricas de traducción automática. Tras esto, hablaremos también de los resultados cualitativos observados.

5.1 Resultados experimentales

Se ha realizado 3 tipos de experimentos distintos. En el primer tipo de experimento se utilizaron datos de distintos corpus como datos del cliente y datos donde realizar la búsqueda. El segundo consistió en probar el caso idílico de que los datos del cliente pertenezcan a los datos donde se realiza la búsqueda. Por último, veremos algunos resultados cualitativos de nuestro método de selección de unidades de traducción.

Se han reentrenado tanto los modelos base en la dirección español inglés como en la dirección inglés español usando distintos umbrales de similitud coseno y distinto número de unidades de traducción máximo a devolver para cada oración del cliente. La métrica que usaremos para comparar el rendimiento de nuestros modelos será el BLEU, una de las métricas más usadas, sino es la más usada, en la traducción automática.

Se han usado umbrales de similitud coseno mínimos de 0.4, 0.5, 0.6 y 0.7. El umbral de 0.4 solo se ha usado en algunos experimentos ya que retrasaba mucho estos debido a la gran cantidad de datos que seleccionaba y a que los resultados con este umbral no solían ser buenos. No se han usado umbrales superiores a 0.7 ya que estos umbrales devuelven muy pocas frases similares.

De máximo número unidades de traducción por cada oración del cliente se han usado 1, 3, 10 y 25. De nuevo, el 25 no aparece en todos los experimentos por los mismos motivos que dejamos de usar el umbral 0.4. La idea que subyace de no devolver todas las unidades de traducción que superen el umbral es que no tengamos demasiadas unidades de traducción que traten de lo mismo para no sesgar demasiado el modelo en el reentrenamiento.

Aparte de mostrar todos los resultados en función de las características anteriormente explicadas también se han acompañado estos resultados de la cantidad de muestras recuperadas por nuestro método en cada ocasión.

Hemos usado las unidades de traducción obtenidas al realizar la búsqueda utilizando las frases del cliente para reentrenar los modelos base.

5.1.1. Con distintos conjuntos de datos

Primero hemos usado como datos del cliente los datos de Europeana para buscar en los datos que se usaron para entrenar los modelos base. Tras esto testeamos con el test de Europeana, este test es un 20 % de las frases de Europeana seleccionadas de manera aleatoria. Obteniendo los resultados de la Figura 5.1

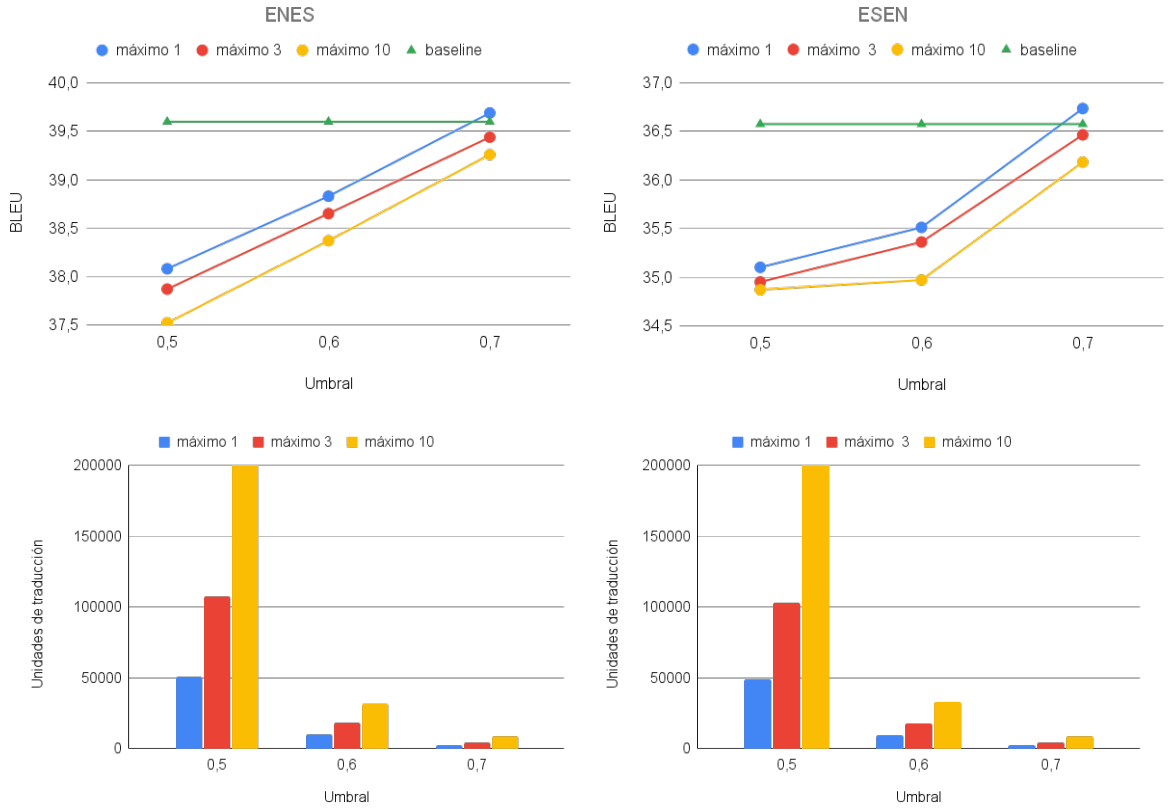

Figura 5.1: BLEU y muestras de entrenamientos de la selección en los datos de entrenamiento mediante los datos de Europeana en función del umbral y del máximo número de frases similares a buscar para cada frase.

Europeana consta de 36 mil unidades de traducción mientras que el conjunto de entrenamiento tiene 6 millones. Para la realización de la selección se usaron todos los datos de Europeana y no solo los de test.

Viendo los resultados uno podría pensar que mientras más alto es el umbral mejor funciona. Pero observando el número de muestras tan bajo que devuelve el umbral 0.7 y a la poca mejora que está aportando reentrenar con estos datos llegamos a la conclusión de que el modelo acaba siendo el mismo que antes del reentrenamiento. A la vista de la poca mejora, los malos resultados y la poca cantidad de unidades de traducción que devolvía nuestro método pensamos que tal vez no habría mucha relación entre los datos de entrenamiento y los de Europeana, por lo que decidimos probar otras cosas.

Intentamos utilizar esta vez un conjunto de datos distinto al de entrenamiento para realizar la selección. Para ello utilizamos los datos de NTEU[1] y WikiMatrix[21] como datos en los que realizar la búsqueda. Durante este experimento nos dimos cuenta de que el conjunto de datos de WikiMatrix tenía frases mal traducidas, esto puede verse en el Apéndice A.3.3. Los resultados de este experimento fueron muy parecidos a los

[1] https://www.nteu.org/

obtenidos en el experimento anterior, por lo que decidimos hacer pruebas con conjuntos de datos que pudieran ser más similares entre sí.

Para ello, hemos probado a coger dos conjuntos de datos que deberían ser similares. El que se usará de test será EMEA[2] y en el que se realizará la búsqueda son los datos de TAUS[3] del Covid-19 (700 mil unidades de traducción). Para hacer la búsqueda en los datos de TAUS, al igual que en el experimento anterior, no nos ceñiremos solo al test de EMEA, buscaremos los datos más similares teniendo en cuenta el corpus completo de EMEA (300 mil frases fuente). Esto dio lugar a los resultados que pueden verse en la Figura 5.2.

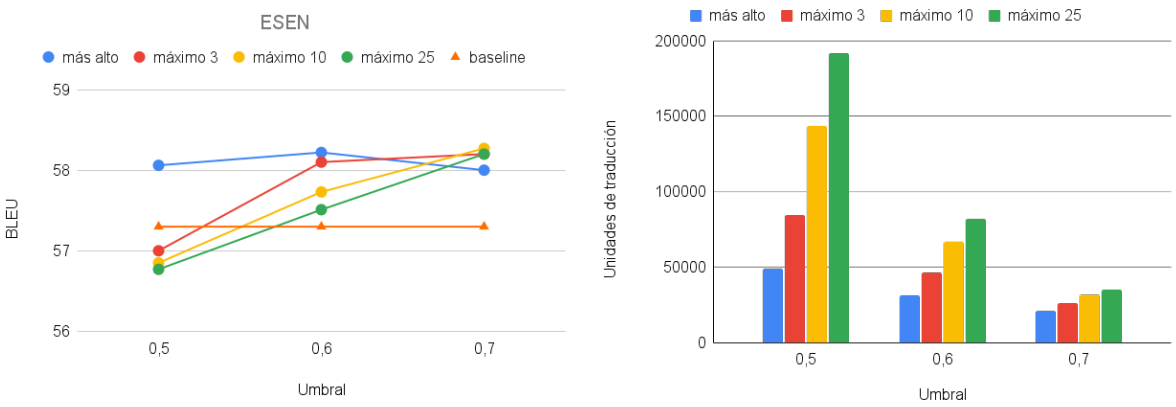

Figura 5.2: BLEU y muestras de entrenamientos de la selección en los datos de TAUS mediante los datos de EMEA en función del umbral y del máximo número de frases similares a buscar para cada frase.

Esta vez sí que obtenemos un número considerable de unidades de traducción para el reentranamiento de nuestro modelo base con todas las combinaciones de umbral y de máximo número de unidades de traducción distintas a devolver. Creemos que esto se debe a que estos corpus son más parecidos o pertenecen a dominios más similares. El hecho de que tengamos 300 mil frases por parte del cliente también es un punto a tener en cuenta.

Con este experimento conseguimos un aumento de cerca de 1 punto de BLEU. Podemos apreciar como los mejores valores del umbral son los más altos al igual que en el experimento anterior.

5.1.2. Con un solo conjunto de datos

Debido a los ligeros aumentos de BLEU en los experimentos anteriores decidimos hacer pruebas más sencillas para ver hasta donde podría llegar nuestro método de selección.

Para ello realizamos una búsqueda dentro de un mismo conjunto de datos, es decir, extrajimos 1000 frases de test del conjunto de datos de NTEU y realizamos la búsqueda en el mismo conjunto de datos de NTEU, el cual está compuesto por 20 millones de unidades de traducción. Antes de lanzar el experimento también nos aseguramos de eliminar las frases del test del conjunto donde se hará la búsqueda. Los resultados pueden verse en Figura 5.3.

[2]https://www.ema.europa.eu/en
[3]https://www.taus.net/

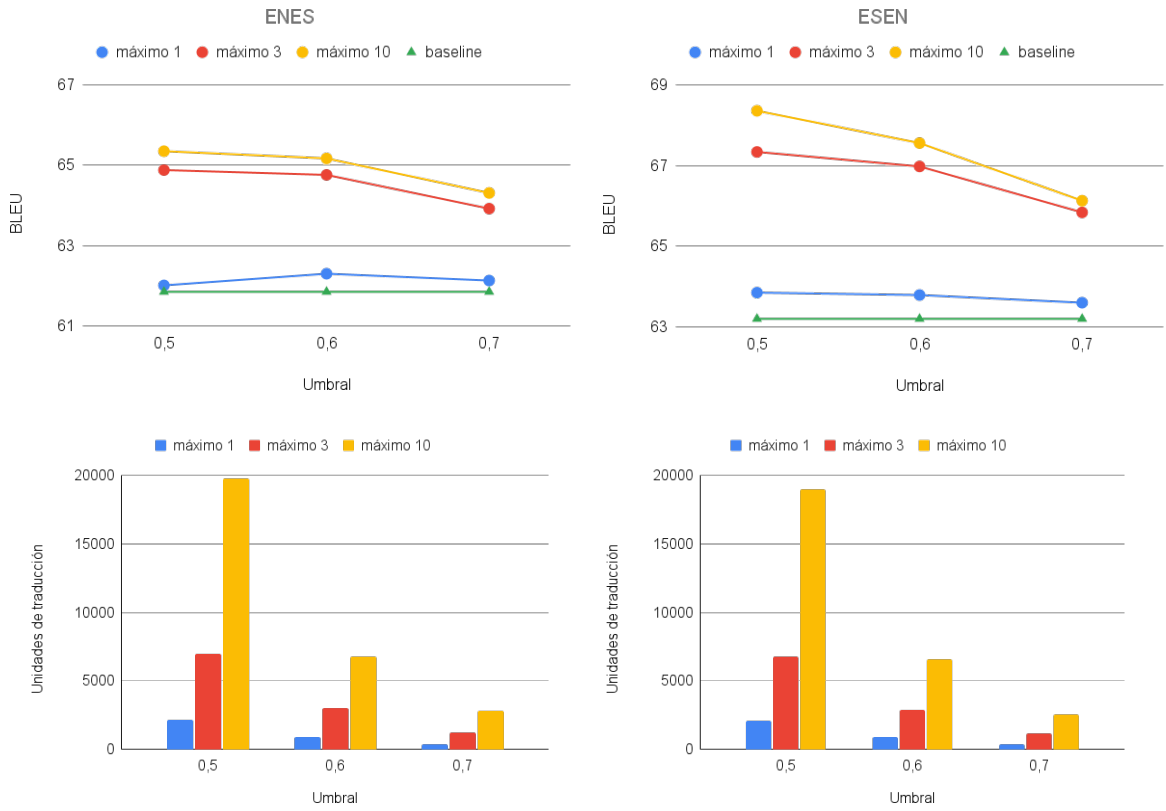

Figura 5.3: BLEU y muestras de entrenamientos de la selección en los datos de NTEU mediante los datos de test de NTEU en función del umbral y del máximo número de frases similares a buscar para cada frase.

Esta vez los resultados obtenidos son mucho mejores, obtenemos un incremento de hasta 3 de BLEU y 5 de BLEU dependiendo de la dirección. El problema es que aquí se estaría resolviendo un problema distinto al planteado. Aquí todos los datos son del dominio deseado, simplemente estamos buscando dentro del dominio los más parecidos al test o al dominio, todavía más específico, del test. Hemos hecho un experimento para ver qué pasaría si esta selección se hubiera hecho de forma aleatoria. Esto puede verse en la 5.4.

Puede observarse como para este problema, donde tenemos una gran cantidad de datos, nuestra selección podría ser una forma eficaz de reentrenar con una cantidad reducida de datos.

Esto no solo se da en NTEU. Si repetimos este experimento con los datos de EMEA (300 mil unidades de traducción) podemos observar unos resultados muy similares, esto puede verse en la Figura 5.5, donde conseguimos aumentos de hasta 2 y 5 de BLEU dependiendo de la dirección.

Podemos observar cómo los valores de umbral y número de unidades de traducción máximo a devolver que mejor funcionan para estos experimentos son los que devuelven el mayor número de unidades de traducción posibles.

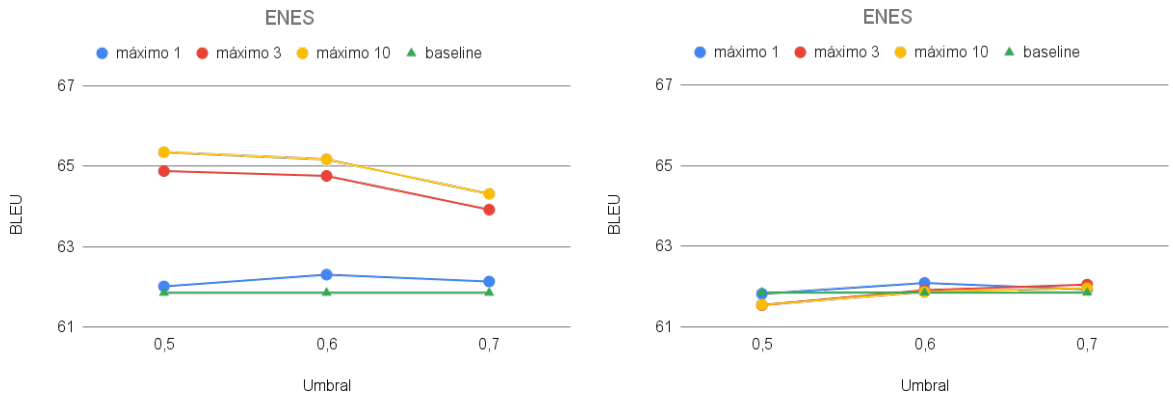

Figura 5.4: BLEU de la selección en los datos de NTEU mediante los datos de test de NTEU en función del umbral y del máximo número de frases similares a buscar para cada frase en comparación con una extracción aleatoria del mismo número de datos.

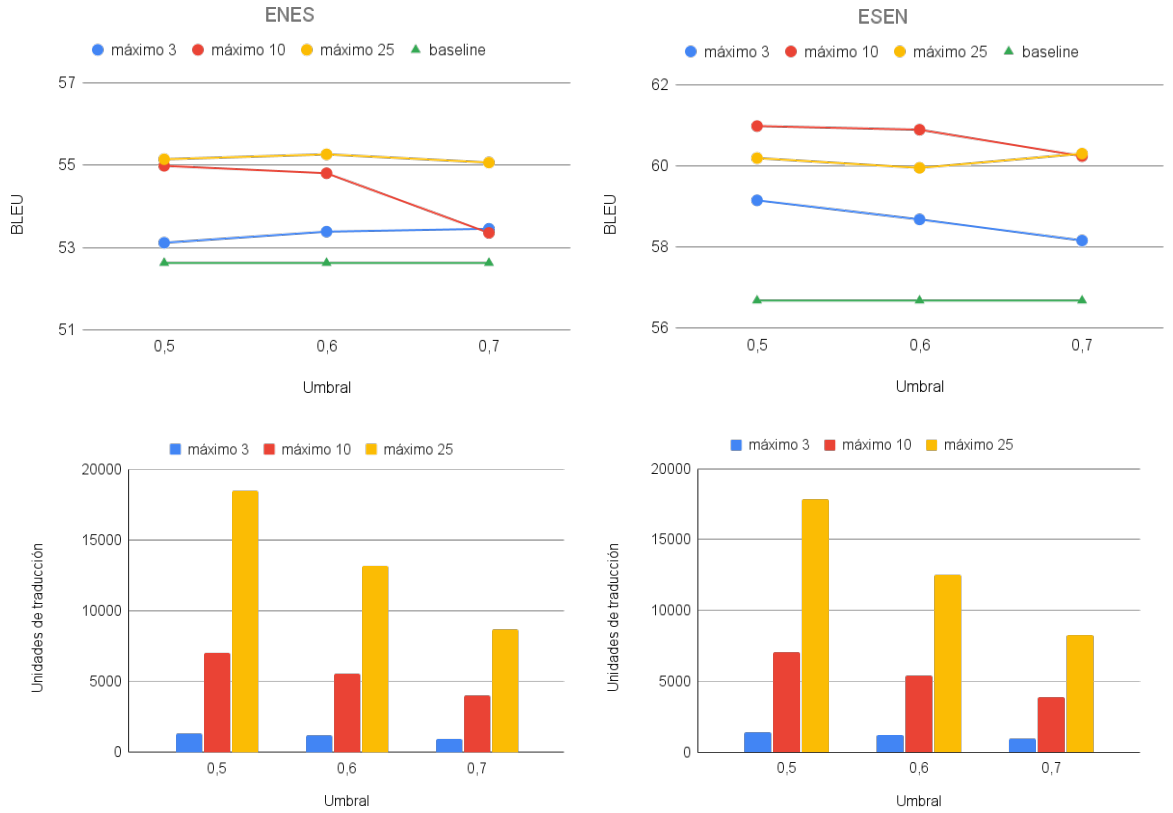

Figura 5.5: BLEU y muestras de entrenamientos de la selección en los datos de EMEA mediante los datos de test de EMEA en función del umbral y del máximo número de frases similares a buscar para cada frase.

5.1.3. Resultados cualitativos

Se ha probado a ver que unidades de traducción devuelve nuestro método en función del umbral escogido. Para ello hemos seleccionado 2 umbrales distinto, uno muy alto de 0.9 y otro más bajo de 0.5.

Para obtener estos resultados se han usado las frases del conjunto de entrenamiento más parecidas a las de Europeana al igual que se ha hecho en el experimento de la Figura 5.1 (pero con los umbrales anteriormente mencionados).

Hay que destacar que nuestro método solo compara las oraciones fuente de las unidades de traducción y que la primera unidad de traducción es siempre la del conjunto de datos del cliente, el resto son las encontradas por nuestro método. Algunos de los resultados obtenidos con un umbral de 0.9 son los siguientes.

Memorias más parecidas a:

Los riesgos asociados a su introducción están en constante crecimiento debido al aumento del comercio, del turismo, del transporte y de la globalización del mercado. ->The risks associated to its introduction are in constant growth due to the increase of trade, tourism, transport and market globalization.

Los riesgos asociados a su introducción están en constante aumento debido al incremento del comercio, del turismo, del transporte y de la globalización del mercado. ->The risks associated with their introduction are growing constantly, owing to the increase in trade, tourism, transport and market globalisation.

Memorias más parecidas a :

carcinoma de células escamosas. ->squamous cell carcinoma.

carcinoma de células escamosas ->squamous cell carcinoma

carcinoma de células escamosas ->squamous cell cancer

Memorias más parecidas a :

lavandera blanca ->White Wagtail

lavandera blanca enlutada ->pied wagtail

Todos los ejemplos con este umbral pueden verse en el Apéndice A.3.1

Cuando hacemos las pruebas con un umbral de 0.5 las frases son bastante menos parecidas, aunque aún pueden apreciarse algunos elementos comunes.

Memorias más parecidas a :

"Mensuales política, el arte, la ciencia , editado por Laura Di Nicola.>"Monthly politics, art, science, edited by Laura Di Nicola."

- política científica y tecnológica. ->- science and technology policy.

La política monetaria es ciencia y arte al mismo tiempo. ->Monetary policy is both a science and an art.

Memorias más parecidas a :

Vittorio Gassman prueba Macbeth/11 ->Vittorio Gassman test Macbeth/11

test de rachas de Wald-Wolfowitz ->Wald-Wolfowitz runs test

Memorias más parecidas a :
Las hembras puntillosos ->The females punctilious
punta de electrodo hembra ->electrode with a female taper
Las mujeres como cuidadoras ->Women as care-providers
Las mujeres como cuidadoras ->Woman as care-givers
telar de género de punto de recogida ->weft knitting machine

Todos los ejemplos con este umbral pueden verse en el Apéndice A.3.2

CAPÍTULO 6
Conclusiones

Los resultados obtenidos son ligeramente mejores que los que daban nuestros modelos bases. Hemos visto como al utilizar conjuntos de datos más similares entre ellos estos resultados eran mejores. La verdadera prueba de esto ha sido cuando hemos visto la gran mejora que podemos llegar a obtener si realizamos la búsqueda dentro del mismo dominio al que pertenecen los datos de test y comparar la cantidad de unidades de traducción devueltas en esta búsqueda cuando los datos son más similares y cuando no. Ha quedado claro que en caso de tener una inmensa cantidad de datos de un dominio concreto y no tener tiempo para entrenar un modelo desde 0, utilizar esta metodología para seleccionar un subconjunto de estos datos podría mejorar los resultados de un modelo base fácilmente y rápidamente.

6.1 Relación del trabajo desarrollado con los estudios cursados

A lo largo de este trabajo hemos trabajado, sobre todo, con transformers, modelos preentrenados y distintas librerías las cuales nos han ayudado a solventar los problemas que nos han ido surgiendo durante el desarrollo del trabajo.

Los transformers son el estado del arte en la traducción y vimos un poco sobre ellos en algunas asignaturas del máster. En Traducción Automática aprendimos ligeramente el como funcionan y los utilizamos un poco para tareas de traducción, mientras que, en Aplicaciones de la Lingüistica Computacional los utilizamos para la extracción de *sentence embeddings* para realizar una posterior clasificación de texto. Otro tema que también se vio ligeramente fueron los modelos preentrenados.

En este trabajo se ha profundizado mucho más en estas tecnologías y se ha aprendido a utilizarlas, se ha investigado cuales se utilizan y seleccionando cuál de las que se usan actualmente puede ser la que mejor nos puede ayudar para resolver de nuestro problema.

También, al estar trabajando en un problema real, la cantidad de datos es mucho mayor a la que se utilizaría en cualquier proyecto hecho en la universidad e incluso en muchos de los experimentos hechos por investigadores. Esto ha hecho que tengamos que buscar la forma de procesar los datos y realizar las búsquedas de la forma más eficiente posible, teniendo que hacer uso de las mejores prácticas para resolver algunos problemas, usar librerías externas especializadas.

6.2 Trabajos futuros

Por falta de tiempo no hemos podido hacer todas las pruebas que nos habría gustado hacer. Nos hubiera gustado probar nuestra selección de datos en otros pares de idiomas, aplicarlo en producción...

Nos hubiera gustado probar en más conjuntos de datos que fueran similares entre sí para ver si podíamos mejorar los resultados, ya que solo hemos podido hacer pruebas con dos conjuntos de datos relacionados con el dominio médico de los cuales uno no era público.

Otra opción que hubiera estado bien probar era combinar distintos conjuntos de datos y ver si usar todos ellos en conjunto como corpus donde realizar la búsqueda mejoraba los resultados o no. Esto no se llegó a realizar debido a la falta de conjuntos de datos de temática similar.

Por último, también podríamos haber realizado experimentos cruzando las frases fuente con las frases destino, es decir, en lugar de buscar las frases fuente de nuestro corpus que más se parecieran a las frases del cliente, buscar las frases destino de nuestro corpus más parecidas a las frases (fuente) del cliente, ya que el MUSE en teoría funciona muy bien a la hora de devolver *sentence embeddings* similares de una oración en diversos idiomas.

Una cosa de la que nos hemos dado cuenta durante la realización del trabajo es que esta forma de buscar frases similares podría adaptarse fácilmente para limpiar corpus de texto bilingües. Para ello solo tendríamos que comparar los *sentence embeddings* de las frases fuente y destino y ver si su similitud coseno es alta o baja. Esta idea surgió al ver el problema que tuvimos con WikiMatrix del Apéndice A.3.3.

APÉNDICE A
Apéndice

A.1 Limpieza de datos

Procedimientos usados para limpiar tanto los datos de entrenamiento como los datos en los que se realizará la búsqueda.

- Eliminación de espacios al principio y al final de la oración.
- Eliminación de frases vacías.
- Eliminación de frases con saltos de línea adicionales.
- Eliminación de unidades de traducción donde la oración fuente y la oración destino son iguales.
- Eliminación de unidades de traducción más largas que un máximo dado.
- Eliminación de unidades de traducción en las que la oración fuente sea 3 veces más larga o corta que la destino.
- Eliminación de unidades de traducción cuando esta solo se trata de un correo electrónico, solo contiene números, solo contiene URLs o solo contiene códigos de referencia.
- Eliminación de unidades de traducción en las que la oración fuente y destino no incluyan los mismos dígitos.
- Eliminación de unidades de traducción en las que la oración fuente y destino no aparezcan el mismo número de signos de puntuación.
- Eliminación de unidades de traducción en las que haya signos de puntuación de apertura sin cerrar.
- Normalizamos las tildes en la codificación Unicode.
- Aplicación del normalizador de Moses al fuente y destino en entrenamiento y solo al fuente durante la traducción.

A.2 Configuración del modelo base

Argumentos para el entrenamiento y decodificación de nuestro modelo base entrenado con OpenNMT-py.

Argumentos del entrenamiento:
-layers 6 -rnn_size 512 -word_vec_size 512 -transformer_ff 2048 -heads 8
-encoder_type transformer -decoder_type transformer -position_encoding
-train_steps 800000 -max_generator_batches 2 -dropout 0.1 -attention_dropout 0.1
-batch_size 4096 -batch_type tokens -normalization tokens -accum_count 2
-optim adam -adam_beta2 0.998 -decay_method noam -warmup_steps 8000
-learning_rate 2 -max_grad_norm 0 -param_init 0 -param_init_glorot
-label_smoothing 0.1 -valid_steps 10000 -save_checkpoint_steps 10000
-keep_checkpoint 10 -early_stopping 8

Argumentos de la decodificación:
-n_best: 5
-min_length: 0
-max_length: 300
-ratio: 0.0
-beam_size: 5

A.3 Ejemplos de frases similares obtenidas con nuestro método

Ejemplo de frases similares obtenidas con nuestro método y sus traducciones correspondientes para distintos umbrales. Hay que destacar que nuestro método solo compara las oraciones fuente de las unidades de traducción y que la primera unidad de traducción es siempre la del conjunto de datos del cliente, el resto son las encontradas por nuestro método.

A.3.1. Umbral de 0.9

Con un umbral de 0.9 para las frases del conjunto de entrenamiento más parecidas a las de Europeana conseguimos frases prácticamente idénticas.

Memorias más parecidas a:
Espacios naturales protegidos ->Protected natural areas
espacio natural protegido ->designated area
espacio natural protegido ->protected area
espacio natural protegido ->protected site

Memorias más parecidas a :

Los riesgos asociados a su introducción están en constante crecimiento debido al aumento del comercio, del turismo, del transporte y de la globalización del mercado. ->The risks associated to its introduction are in constant growth due to the increase of trade, tourism, transport and market globalization.

Los riesgos asociados a su introducción están en constante aumento debido al incremento del comercio, del turismo, del transporte y de la globalización del mercado. ->The risks associated with their introduction are growing constantly, owing to the increase in trade, tourism, transport and market globalisation.

Memorias más parecidas a :

carcinoma de células escamosas. ->squamous cell carcinoma.

carcinoma de células escamosas ->squamous cell carcinoma

carcinoma de células escamosas ->squamous cell cancer

Memorias más parecidas a :

¿Cómo se describe la seguridad? ->How is the security described?

¿Cómo se define la seguridad? ->How is safety defined?

Memorias más parecidas a :

caja de caudales ->safes (storage containers)

compartimento de caja de caudales ->safe deposit box

Memorias más parecidas a :

lavandera blanca ->White Wagtail

lavandera blanca enlutada ->pied wagtail

A.3.2. Umbral de 0.5

Con un umbral de 0.5 las frases del conjunto de entrenamiento más parecidas a las de Europeana ya son bastante menos parecidas, aunque aún pueden apreciarse algunos elementos comunes.

Memorias más parecidas a :

"Mensuales política, el arte, la ciencia , editado por Laura Di Nicola.>"Monthly politics, art, science, edited by Laura Di Nicola."

- política científica y tecnológica. ->- science and technology policy.

La política monetaria es ciencia y arte al mismo tiempo. ->Monetary policy is both a science and an art.

Memorias más parecidas a :

Vittorio Gassman prueba Macbeth/11 ->Vittorio Gassman test Macbeth/11

test de rachas de Wald-Wolfowitz ->Wald-Wolfowitz runs test

Memorias más parecidas a :

seminario ->seminar

seminario de información ->information workshop

Secretaría del seminario ->Seminar Secretariat

asistencia a las sesiones ->attendance at meetings

convocatorias a las sesiónes ->the convening of meetings

seminario para diplomáticos ->seminar for diplomats

seminario organizado para el personal ->seminar organized for all members of staff

Sesiones plenarias Sesiones plenarias ->The plenary sittings are the high point of parliament's work.

Participación en las sesiones ->Attendance of Members at sittings

Memorias más parecidas a :

Las hembras puntillosos ->The females punctilious

punta de electrodo hembra ->electrode with a female taper

Las mujeres como cuidadoras ->Women as care-providers

Las mujeres como cuidadoras ->Woman as care-givers

telar de género de punto de recogida ->weft knitting machine

A.3.3. Errores en el conjunto de datos seleccionados

Al seleccionar un umbral de 0.9 para las frases de WikiMatrix más parecidas a las de Europeana descubrimos que este conjunto de datos tenía muchas traducciones muy malas de la oración fuente Estados Unidos. Se puede apreciar como el problema está en las oraciones destino y no en las que recupera nuestro algoritmo, que son las oraciones fuente.

Estados Unidos ->United States

Estados Unidos de América ->et de l'Environnement

Estados Unidos de América ->Mr. Wolfgang Trautwein

Estados Unidos de América ->*Logistic Centre, Reconnaissance Battalion

Estados Unidos de América ->El Salvador

Estados Unidos de América ->France South Africa

Estados Unidos de América ->Documentation:

Estados Unidos de América ->Dominican Republic

Estados Unidos de América ->Bosnia and Herzegovina

Estados Unidos de América ->Monaco United States of America Holy See

Estados Unidos de América ->New Zealand

Estados Unidos de América ->New and dynamic sectors

Estados Unidos de América ->African Commission of Health and Human Rights Promoters

Estados Unidos de América ->United States

Estados Unidos de América ->Croatia Republic of Korea

Estados Unidos de América ->Chairman and Professor - Soc.

Estados Unidos de América ->Egypt Portugal

Estados Unidos de América ->Minister for the Environment and Heritage

Estados Unidos de América ->Index to this issue

Estados Unidos de América ->The Holy See was represented in its capacity as Observer State.

Estados Unidos de América ->Activities undertaken by UNCTAD in favour of Africa

Estados Unidos de América ->C. United States of America

Estados Unidos de América ->French Guiana

Estados Unidos de América ->Czech Republic

Estados Unidos de América ->Ethiopia Saudi Arabia

Estados Unidos de América ->Global WitnessPartnership Africa Canada

Estados Unidos de América ->J. United States of America

Estados Unidos de América ->Democratic Republic of the Congo Russian Federation

Estados Unidos de América ->The existence of diffuse network structures must rather be assumed.

Estados Unidos de América ->Democratic Peoples' Republic of Korea

Estados Unidos de América ->Uganda Government officials Ministry of Defence

Estados Unidos de América ->Equatorial Guinea

EUA Estados Unidos de América ->EAP Economically active population

Bibliografía

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Ondřej Bojar, Christian Buck, Chris Callison-Burch, Christian Federmann, Barry Haddow, Philipp Koehn, Christof Monz, Matt Post, Radu Soricut, and Lucia Specia. Findings of the 2013 Workshop on Statistical Machine Translation. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 1–44, Sofia, Bulgaria, August 2013.

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

[4] Qian Cao and Deyi Xiong. Encoding gated translation memory into neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3042–3047, Brussels, Belgium, October-November 2018.

[5] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder. *CoRR*, abs/1803.11175, 2018.

[6] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online, July 2020.

[7] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc Viet Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *ACL (1)*, pages 2978–2988, 2019.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[9] Jiatao Gu, Yong Wang, Kyunghyun Cho, and Victor O. K. Li. Search engine guided non-parametric neural machine translation. *ArXiv*, abs/1705.07267, 2017.

[10] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[11] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *CoRR*, abs/1702.08734, 2017.

[12] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[13] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aäron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *CoRR*, abs/1610.10099, 2016.

[14] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. Opennmt: Open-source toolkit for neural machine translation. In *Proc. ACL*, 2017.

[15] Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium, November 2018.

[16] Xiaodong Liu, Kevin Duh, Liyuan Liu, and Jianfeng Gao. Very deep transformers for neural machine translation. *CoRR*, abs/2008.07772, 2020.

[17] Yuan Liu and Yves Lepage. Covering a sentence in form and meaning with fewer retrieved sentences. In *Proceedings of the 35th Pacific Asia Conference on Language, Information and Computation*, pages 513–522, Shanghai, China, 11 2021.

[18] Fandong Meng, Zhengdong Lu, Mingxuan Wang, Hang Li, Wenbin Jiang, and Qun Liu. Encoding source language with convolutional neural network for machine translation. *CoRR*, abs/1503.01838, 2015.

[19] Tasnim Mohiuddin, Philipp Koehn, Vishrav Chaudhary, James Cross, Shruti Bhosale, and Shafiq R. Joty. Data selection curriculum for neural machine translation. *CoRR*, abs/2203.13867, 2022.

[20] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. 2019.

[21] Holger Schwenk, Vishrav Chaudhary, Shuo Sun, Hongyu Gong, and Francisco Guzmán. WikiMatrix: Mining 135M parallel sentences in 1620 language pairs from Wikipedia. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1351–1361, Online, April 2021.

[22] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *CoRR*, abs/1508.07909, 2015.

[23] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016.

[24] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[26] Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. Learning deep transformer models for machine translation. *CoRR*, abs/1906.01787, 2019.

[27] Chuhan Wu, Fangzhao Wu, Tao Qi, Yongfeng Huang, and Xing Xie. Fastformer: Additive attention can be all you need. *CoRR*, abs/2108.09084, 2021.

[28] Yinfei Yang, Daniel Cer, Amin Ahmad, Mandy Guo, Jax Law, Noah Constant, Gustavo Hernández Ábrego, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Multilingual universal sentence encoder for semantic retrieval. *CoRR*, abs/1907.04307, 2019.

[29] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237, 2019.