eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Computación

Trabajo de Fin de Grado

Adaptación y evaluación de modelos de consulta inversa de diccionarios monolingües

Autor

Ander Gil Moro

2021

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Computación

Trabajo de Fin de Grado

Adaptación y evaluación de modelos de consulta inversa de diccionarios monolingües

Autor

Ander Gil Moro

Director

Germán Rigau Claramunt

Resumen

Los diccionarios tradicionales son una obra de consulta en las que se recoge y se define o traduce, generalmente en orden alfabético, un conjunto de palabras de una o de más lenguas o de una materia determinada. Los diccionarios inversos, en cambio, resuelven esa tarea de forma inversa: dada la definición o descripción de un concepto, se pretende encontrar la palabra que mejor expresa el concepto definido.

En este Trabajo de Fin de Grado, se revisan la bibliografía y los diferentes modelos y sistemas disponibles de diccionarios inversos, y se adaptan algunos de estos sistemas propuestos a distintos diccionarios en inglés, o bien se toman como referencia para elaborar modelos propios, y se evalúa su rendimiento en un conjunto de datos comparable con el estado del arte.

Índice general

Índice de figuras

Índice de tablas

1. CAPÍTULO

Introducción

Este proyecto se enmarca en el campo de la Inteligencia Artificial, más concretamente en el área del Procesamiento del Lenguaje Natural, el cual constituye un área central de investigación dentro de este campo. Las tecnologías del lenguaje (TL) son ingredientes muy importantes (y a veces invisibles) de muchas aplicaciones diversas: motores de búsqueda, correctores de ortografía, traductores automáticos, sistemas de recomendaciones, asistentes virtuales, sintetizadores de voz, y muchas otras. Este trabajo también está relacionado con la lingüística, ya que se trabajará bastante con diccionarios y con el significado de las palabras. Principalmente, se pretende investigar los **diccionarios inversos** y crear algún diccionario de este tipo.

La tarea del diccionario inverso [Hill et al., 2015, Bilac et al., 2004] consiste en dada la definición o descripción de un concepto, encontrar la palabra que mejor exprese ese concepto definido. En la imagen 1.1 se muestra un ejemplo de esta tarea: al introducir la definición "*a road where cars go very quickly without stopping*", el diccionario inverso devuelve el concepto que se define (*expressway*) junto con palabras semánticamente parecidas (*freeway, motorway, autobahm...*).

1.1. Motivación

Los diccionarios inversos pueden ser herramientas muy útiles. En primer lugar, pueden resolver eficientemente el problema de "tener una palabra en la punta de la lengua"[Brown and Mcneill, 1966], el cual le sucede a menudo a personas que escriben mucho en su

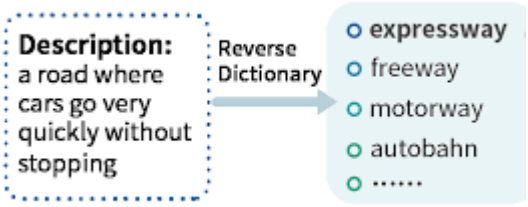

Figura 1.1: Ejemplo de tarea de diccionario inverso

día a día, tal y como investigadores, escritores o estudiantes. Además, también tienen un gran impacto práctico para las personas que están en el proceso de aprender un idioma y conocen una cantidad limitada de palabras. Por ejemplo, un profesor podría pedirles a sus alumnos que describiesen una palabra, y si el diccionario inverso lograse encontrar la palabra original a partir de la descripción del estudiante, significaría que el estudiante definió la palabra correctamente. Esta utilidad podría ser útil para preparar métodos automáticos de aprendizaje sin la necesidad de tener un profesor supervisando el trabajo.

La motivación de disponer diccionarios inversos se extiende hasta la medicina. Dentro de este campo, se cree que estos diccionarios pueden llegar a ser extremadamente útiles para ayudar a los pacientes que padecen anomia. La anomia es un desorden neuropsicológico caracterizado por la dificultad para recordar los nombres de las cosas e impide que estos pacientes puedan llamar a las cosas por su nombre [Hadar et al., 1987]. El uso de los diccionarios inversos ayudaría a estos pacientes en la selección de las palabras.

Por último, y ya que este TFG se enmarca en el Procesamiento del Lenguaje Natural, los diccionarios inversos también son útiles dentro de este campo. En términos de NLP, los diccionarios inversos pueden usarse para evaluar la calidad de la representación de oraciones [Hill et al., 2015]. También son beneficiosos para las tareas que involucran el mapeo de texto a entidad, incluyendo las tareas de contestación de preguntas y recuperación de información [Kartsaklis et al., 2018].

1.2. Objetivos del proyecto

El objetivo principal de este proyecto ha sido revisar, entender y adaptar los sistemas propuestos disponibles de diccionarios inversos en inglés.

Entre los objetivos más generales, se encuentran los siguientes:

- Revisar la bibliografía existente referente a los diccionarios inversos en el marco

del Procesamiento del Lenguaje Natural.

- Estudiar y entender las tecnologías detrás de los prototipos existentes de diccionarios inversos.
- Adaptar algunos de estos sistemas a distintos diccionarios monolingües.
- Evaluar y comparar el rendimiento de los sistemas.

A partir de las tecnologías existentes, se plantea construir tres prototipos de diccionarios inversos:

1. Diccionario inverso a partir de los modelos de lenguaje *Transformer*s ofrecidos por *HuggingFace* [1]. Concretamente se usan los modelos de Mask-Filling. (4.1)
2. Diccionario inverso usando SentenceTransformers [2], utilizando el principio de medir la distancia mínima entre dos definiciones tras haberlas convertido en vectores. (4.2)
3. Diccionario inverso creado a partir de la API *datamuse* [3], un motor de búsqueda de palabras para desarrolladores. (4.3)

Conociendo estos sistemas que se pretenden desarrollar, estos son los objetivos más concretos del proyecto:

- Leer y entender el funcionamiento de los modelos de lenguaje BERT (Bidirectional Encoder Representations from Transformers) [Devlin et al., 2019], RoBERTa (A Robustly Optimized BERT Pretraining Approach) [Liu et al., 2019], BART [Lewis et al., 2019]... ofrecidos por *HuggingFace*.
- Probando estos modelos de lenguaje, elegir el que mejor se adapte a las necesidades del problema y crear un diccionario inverso básico a partir del Mask-Filling.
- Entrenar los modelos de lenguaje *(Fine-tuning)* para mejorar los resultados.
- Leer y entender el funcionamiento detrás de los SentenceTransformers.

[1] https://huggingface.co/transformers/index.html
[2] https://www.sbert.net/
[3] https://www.datamuse.com/api/

- Crear un segundo diccionario inverso basado en el uso de SentenceTransformers.
- Desarrollar un diccionario inverso plurilingüe que, dada una definición en un idioma, encuentre la palabra del concepto definido en otro idioma distinto. [Gollins and Sanderson, 2001]
- Recoger los datos proporcionados por una API y adaptarlos para crear mi propio prototipo de diccionario inverso
- Evaluar los resultados obtenidos por los tres modelos obtenidos y compararlos con el conjunto de datos evaluable creado por [Hill et al., 2015]

1.3. Organización de la memoria

La memoria está constituida por cinco capítulos, además del resumen:

1. La introducción. Se trata de este mismo capítulo, donde se introduce el problema, se explora la motivación de resolver dicho problema, y se discuten los objetivos a alcanzar en el proyecto.
2. Las bases teóricas y el Estado del Arte. En este capítulo se estudian los fundamentos teóricos de los sistemas empleados para la realización de los prototipos que resuelven la tarea de los diccionarios inversos. También se investiga en profundidad el estado del arte en el que se encuadra el problema: se revisan los diferentes sistemas que existen para la realización de la tarea que nos ocupa y se estudian algunos de los resultados.
3. Gestión del proyecto. En este capítulo se trabaja todo lo referente a la gestión del proyecto: la planificación, la gestión de riesgos, las desviaciones respecto a la planificación inicial, etcétera.
4. Desarrollo del proyecto. Este capítulo está formado por dos secciones principales. En la primera, se presentan los diferentes prototipos desarrollados que cumplen con la tarea de los diccionarios inversos. En la segunda, se describe la experimentación requerida para su optimización, se evalúa su rendimiento en un conjunto de datos comparable y se compara con los resultados obtenidos por los sistemas del estado del arte.

5. Conclusiones. En este último capítulo, se discute el impacto que ha tenido el desarrollo de este TFG en el aprendizaje del alumno y a ver si finalmente se han alcanzado los objetivos propuestos. También se plantea el trabajo a realizar en el futuro.

2. CAPÍTULO

Bases teóricas y Estado del Arte

En este capítulo, se verán las bases teóricas más relevantes para poder comprender mejor el desarrollo del TFG y cómo funcionan los sistemas de diccionarios inversos del estado del arte. También se repasará el funcionamiento de estos sistemas, los cuales consiguen resultados excelentes en nuestra tarea.

El Procesamiento del Lenguaje Natural (PLN) es un área de la Inteligencia Artificial que permite el tratamiento computacional del lenguaje humano, y que se enfoca en la comprensión y en el manejo del significado en documentos escritos. Se encarga de la creación o desarrollo de formalismos y recursos sobre el funcionamiento del lenguaje. Un ejemplo de estos formalismos serían los *word embeddings*, que se verán a continuación (2.1). Esta transformación de texto a información estructurada es la base de muchos sistemas automáticos de análisis lingüístico, como pueden ser los diccionarios inversos.

Los sistemas de PLN se basan en conjuntos de texto extensos (corpus) representativos, y se utilizan técnicas de *machine learning* para automatizar las tareas de análisis de texto. BERT es un modelo de última generación que es capaz de realizar este tipo de tareas, el cual se estudia en este mismo capítulo (Ver apartado 2.2).

2.1. *Word embeddings*

El *deep learning* o aprendizaje profundo para el procesamiento del lenguaje natural tiene muchas aplicaciones que han demostrado ser muy eficientes. Puede usarse para muchas

tareas, como el análisis de sentimientos [Liu et al., 2010] o el reconocimiento de entidades [Mikheev et al., 1999]. Sin embargo, antes de implementar dichas tareas, se deben preprocesar los datos. Las redes neuronales toman números como entrada, y en el campo del lenguaje natural se trabaja con texto. Por eso, es necesario convertir las palabras en un formato numérico. El método tradicional era el *one hot encoding* [1], pero hoy en día se usan los llamados vectores de palabras. Estos son vectores de grandes dimensiones que representan palabras o incluso frases enteras.

Los vectores de palabras son una manera eficiente de representar texto. A diferencia del método *one-hot*, los vectores tienen la capacidad de guardar información semántica de la palabra. Antes de ver en profundidad el método para lograr esto, es necesario entender el principio de que palabras de significado similar ocurren juntas más frecuentemente que palabras que no lo son. Por ejemplo, la palabra "perro" ocurrirá en la vecindad de la palabra "gato" con más frecuencia que en la vecindad de la palabra "castillo". Esto sucede porque hay una similitud semántica entre ambas palabras, y este es el factor clave que se ha aprovechado para construir los vectores de palabras.

Hay varios algoritmos que resuelven esta tarea de convertir palabras en vectores: GloVe [2], FastText [3] y Word2Vec[4], por ejemplo. A continuación se verá cómo funciona el algoritmo desarrollado por Google, Word2Vec [Mikolov et al., 2013a, Mikolov et al., 2013b].

Lo primero de todo es acumular datos. Se necesita un conjunto de datos para saber qué palabras ocurren cerca de cierta palabra. Para esto, se usa algo llamado ventana de contexto *context window*. Se necesita fijar el parámetro conocido como tamaño de la ventana. Al iterar por todas las palabras, este tamaño de ventana marcará cuantas palabras se deben observar alrededor de la palabra actual. Si el tamaño de la ventana es de dos, se considerarán las dos palabras anteriores y las dos posteriores de la palabra actual. En la figura 2.1 se ilustra un ejemplo de una ventana de tamaño dos.

Tras pasar la ventana de contexto por todas las palabras, se forma un *dataset* de formato (Palabra objetivo, palabra contextual). Para el mismo ejemplo de la figura 2.1, se crearía el siguiente conjunto de datos: *(Deep, Learning), (Deep, is), (Learning, Deep), (Learning, is), (Learning, very), (is, Deep), (is, Learning), (is, very), (is, hard), (very, learning), (very, is), (very, hard), (very, and), (hard, is), (hard, very), (hard, and), (hard, fun), (and, very), (and, hard), (and, fun), (fun, hard), (fun, and)*. Estos serán los datos de entrenamiento para

[1] https://en.wikipedia.org/wiki/One-hot
[2] https://nlp.stanford.edu/projects/glove/
[3] https://fasttext.cc/
[4] https://code.google.com/archive/p/word2vec/

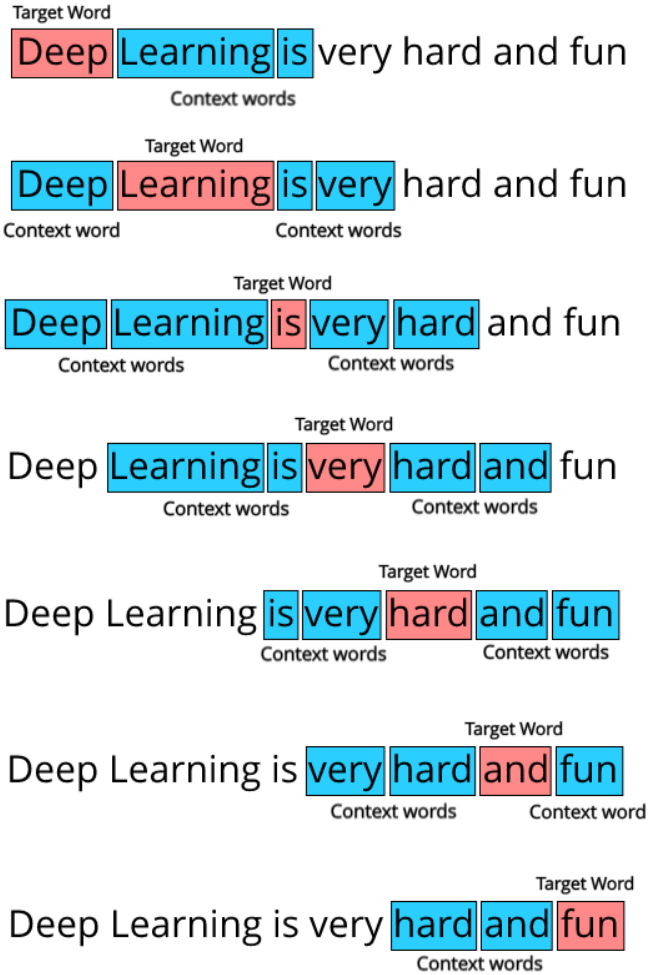

Figura 2.1: Ejemplo de ventana de contexto de tamaño dos

el algoritmo. Dentro del algoritmo Word2Vec, se distinguen dos arquitecturas diferentes: la arquitectura *Skip Gram* y la arquitectura *CBOW*.

2.1.1. Arquitectura Skip Gram

En la arquitectura Skip Gram, se intenta predecir cada palabra contextual dada una palabra objetivo. Para esta tarea de predicción, se usa una red neuronal. La entrada de la red neuronal será la palabra objetivo en su formato *one-hot* y la salida será la versión *one-hot* de la palabra contextual. Por ello, el tamaño de las capas de salida y de entrada será el tamaño del vocabulario, V. Esta red neuronal tiene una capa oculta en el medio. El tamaño de esta capa determinará el tamaño de los vectores de palabras que queremos obtener. Esta arquitectura se ilustra en la figura 2.2

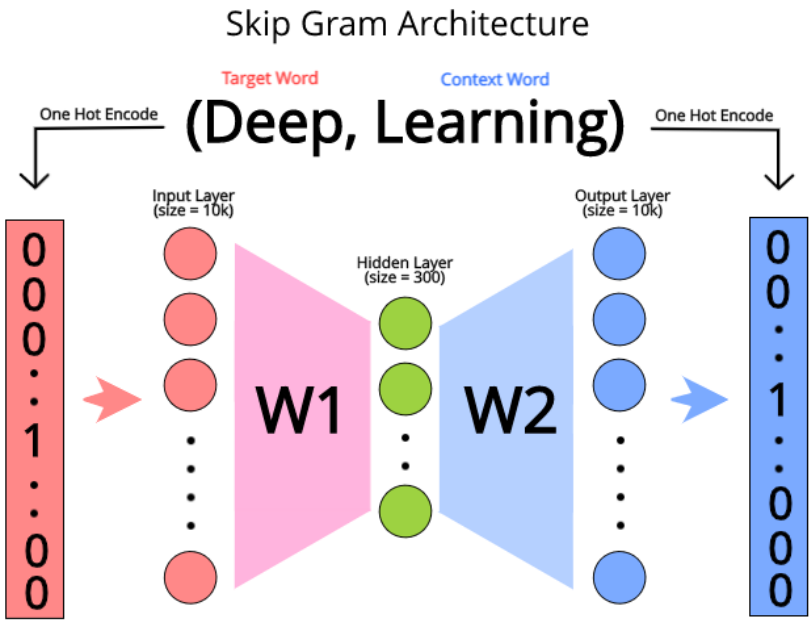

Figura 2.2: Arquitectura Skip Gram, con un tamaño de vocabulario de 10.000 y vectores de palabras de tamaño 300.

Como la red se compone de tan solo tres capas, solo habrá dos matrices de pesos: W1 y W2. En el ejemplo de 2.2, W1 y W2 tendrán una dimensión de 10000*300. Estas dos matrices de pesos serán determinantes a la hora de calculas los vectores de palabras.

Para el conjunto de datos de (palabra objetivo, palabra contextual), se pasará cada par a la red neuronal y se entrenará para adivinar palabras contextuales dada una palabra objetivo. Tras el entrenamiento, la red devolverá un vector de salida que representará las palabras que tienen una alta probabilidad de aparecer cerca de la palabra de entrada.

2.1.2. Arquitectura CBOW

En esta arquitectura, la diferencia está en que se intenta predecir la palabra objetivo dadas las palabras contextuales. Es decir, se invierte el modelo skip gram, tal y como se muestra en la figura 2.3

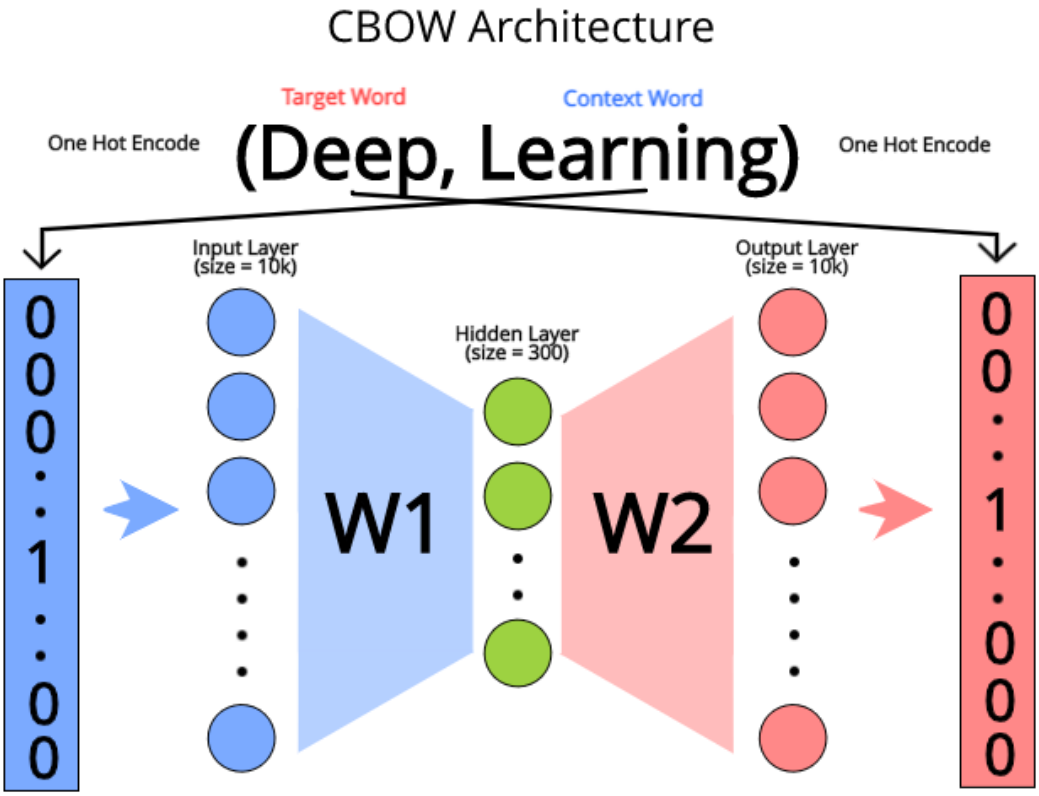

Figura 2.3: Arquitectura CBOW, con un tamaño de vocabulario de 10.000 y vectores de palabras de tamaño 300.

En este caso, la entrada será la representación de un grupo de palabras contextuales, y la salida será la palabra objetivo más adecuada dentro de la vecindad de esas palabras. Por ejemplo, dada la oración “Deep _____ is very hard”, donde “Deep”, “is”, “very” Y “hard” son las palabras contextuales, la red debería devolver la palabra "Learning". Este principio es la base tras la tarea de *Mask-Filling*, técnica que se utiliza en el sistema 4.1 para construir un diccionario inverso.

Esta tarea cometida por las redes neuronales sirve para capturar la información semántica de las palabras y aprender qué palabras son semánticamente similares a otras. Esto sucede por el fenómeno explicado anteriormente donde las palabras similares ocurren juntas frecuentemente. La red neuronal captura mucha información semántica, y esta se extrae de la matriz de pesos W1. En la primera capa de la red, se multiplican los valores de la capa de entrada con la primera matriz de pesos. Como la entrada es un vector *one-hot*, al realizar la multiplicación una fila de la matriz retendrá sus valores mientras que las demás se volverán nulas. Por lo tanto, la fila no nula capturará toda la información referente a

la palabra de entrada y serán esos valores lo que se usarán para construir el vector de la palabra. Cada fila de la matriz de pesos será el *embedding* de una palabra distinta, como se muestra en la figura 2.4

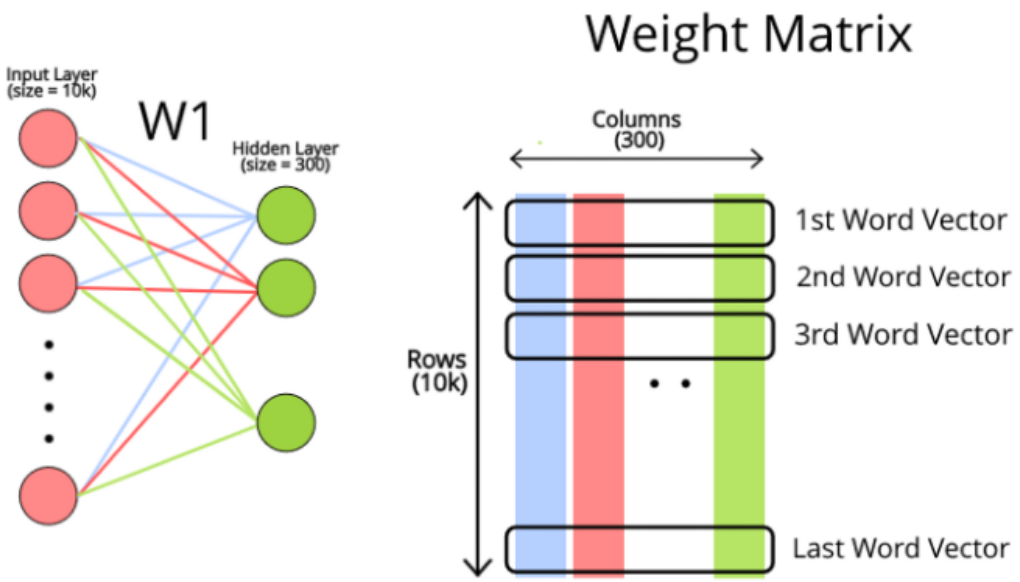

Figura 2.4: Formación de los vectores de palabras mediante la matriz de pesos W1.

2.2. BERT

El modelo de lenguaje BERT [5] (Bidirectional Encoder Representations from Transformers) [Devlin et al., 2019] ha sido uno de los cimientos principales en el desarrollo de este proyecto.

A continuación, se repasarán los fundamentos principales que nos ayudarán a entender cómo funciona en conjunto este modelo.

2.2.1. Encoder-Decoder y Sequence to Sequence

Para mapear una secuencia de entrada a una secuencia de salida, muchas veces se aplica la transformación secuencia a secuencia (*sequence-to-sequence*, o *seq2seq*) usando un modelo de codificador-decodificador (*encoder-decoder*). Un ejemplo del *seq2seq* es la

[5] https://huggingface.co/transformers/model_doc/bert.html

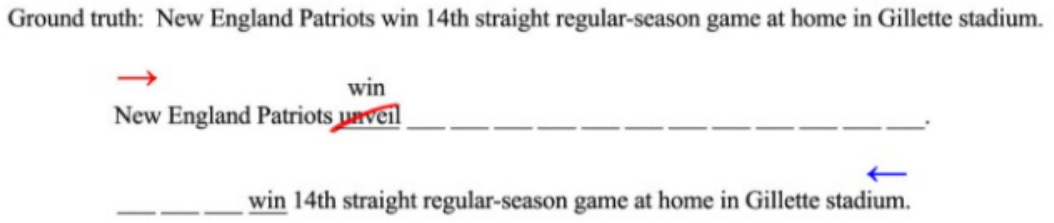

Figura 2.5: Las redes neuronales recurrentes (RNN) tienen el inconveniente de funcionar en un único sentido.

traducción de una frase de un lenguaje a otro. Durante años, en el modelo *seq2seq* se han usado redes neuronales recurrentes (RNN [6]), con más frecuencia LSTM [7] o GRU [8] para evitar el problema del desvanecimiento del gradiente (Que las redes no puedan avanzar con su entrenamiento porque las actualizaciones son excesivamente pequeñas). Pero estos tipos de redes neuronales sufren algunos inconvenientes:

- Aprender el contexto de las palabras se vuelve cada vez más complicado con RNN según aumenta la distancia.
- RNN es direccional. En el ejemplo de la figura 2.5, la RNN habría tenido más probabilidades de acertar la palabra "win" correctamente si fuese en sentido inverso, de atrás hacia delante.

Para resolver estos problemas, se podría diseñar un modelo que incluyese dos RNN, una hacia delante y otra hacia atrás, construyendo una RNN bidireccional. Pero al final, se podría argumentar que para poder entender el contexto de una palabra lo mejor posible, lo mejor sería revisar todas las palabras del párrafo al mismo tiempo. Es decir, deberíamos aplicar capas totalmente conectadas directamente a cada palabra, pero esto es inviable. Por lo que el problema se puede solucionar de otra forma: la atención.

2.2.2. La atención

Cuando se crea un contexto para nuestra consulta, no se debería dar el mismo peso a toda la información que se tiene. Se necesita prestar atención a las cosas más importantes.

[6] https://en.wikipedia.org/wiki/Recurrent_neural_network
[7] https://en.wikipedia.org/wiki/Long_short-term_memory
[8] https://en.wikipedia.org/wiki/Gated_recurrent_unit

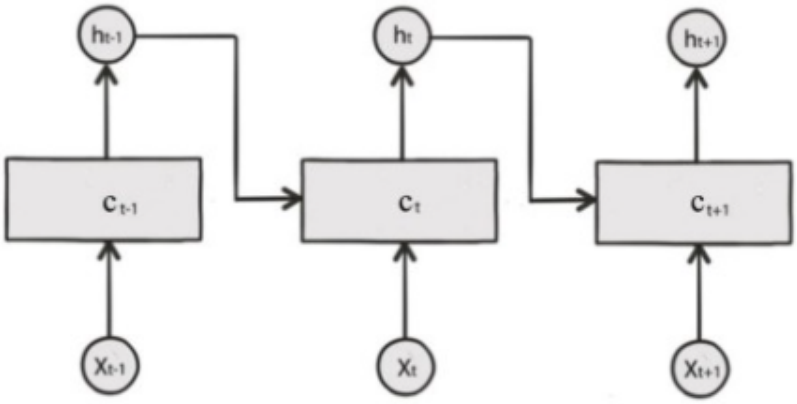

Figura 2.6: Las predicciones de las RNN se basan en la entrada y en el estado anterior.

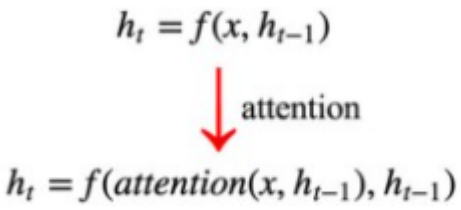

Figura 2.7: En un sistema basado en la atención, la entrada se sustituye por la atención.

Se tiene que crear un contexto de lo que nos interesa, basándose en nuestra consulta. En una RNN, se realizan predicciones basadas en una entrada xt y en el estado oculto anterior $h(t-1)$, como se ilustra en la figura 2.6. Sin embargo, en un sistema basado en la atención, la entrada x será sustituida por la atención, como se aprecia en la figura 2.7

Se puede conceptualizar que el proceso de la atención mantiene la información que es importante actualmente. Por ejemplo, para cada característica de entrada x_i, se puede entrenar una capa totalmente conectada para puntuar la importancia de cada característica i bajo el contexto del estado oculto anterior h. Después, se normaliza la puntuación usando una función *Softmax* para formar los pesos de atención α. Finalmente, la atención Z sustituye la entrada.

2.2.3. Codificador de Transformer

Para entender el modelo del lenguaje BERT, es necesario explicar cómo funcionan los codificadores de los Transformers [9]. Los Transformers son complejas arquitecturas a la altura del estado del arte en el procesamiento del lenguaje. En este apartado, se explicará solamente su parte codificadora, ya que es la que se usa en BERT.

Para la explicación del codificador de Transformers, se usará la siguiente oración de 13 palabras: *New England Patriots win 14th straight regular-season game at home in Gillette stadium.* En el paso de la codificación, el Transformer usa los *word embeddings* (ver subapartado 2.1) para convertir esas 13 palabras en 13 vectroes de 512 dimensiones. Después son transferidas a un codificador basado en la atención para generar una representción de cada palabra dependiente de su contexto. Cada *word embedding* tendrá un vector de salida h_i. En la figura 2.8, h_i es un vector de 512 dimensiones que codifica x_i según su contexto.

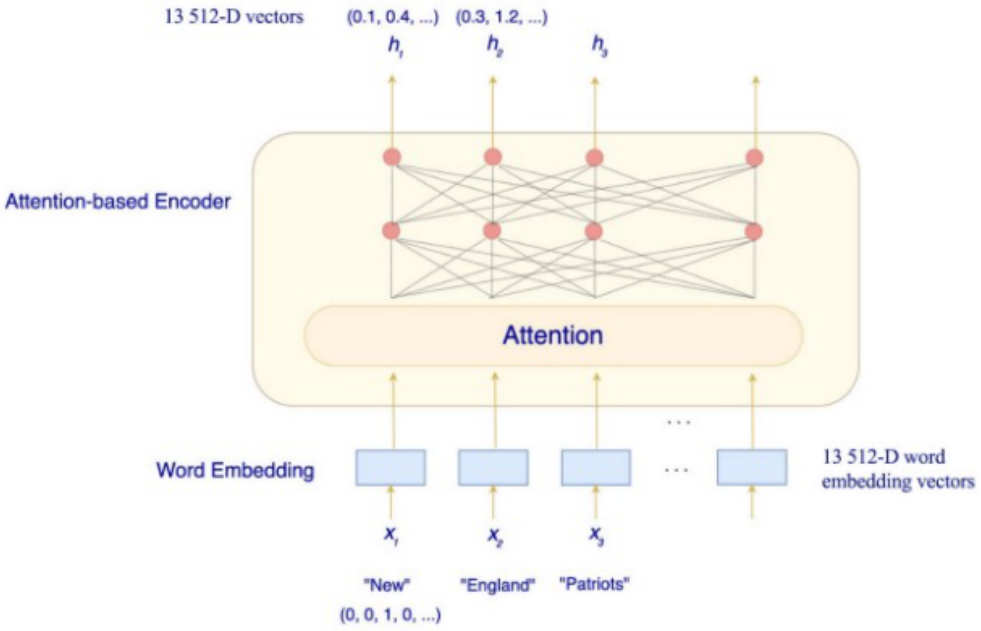

Figura 2.8: Codificador que codifica *wiord embeddings* en otros vectores según su contexto, basándose en la atención.

Hagamos zoom en este codificador basado en la atención. En realidad, el codificador apila seis codificadores. La salida de cada codificador se redirige a la entrada del siguiente.

[9] https://huggingface.co/transformers/

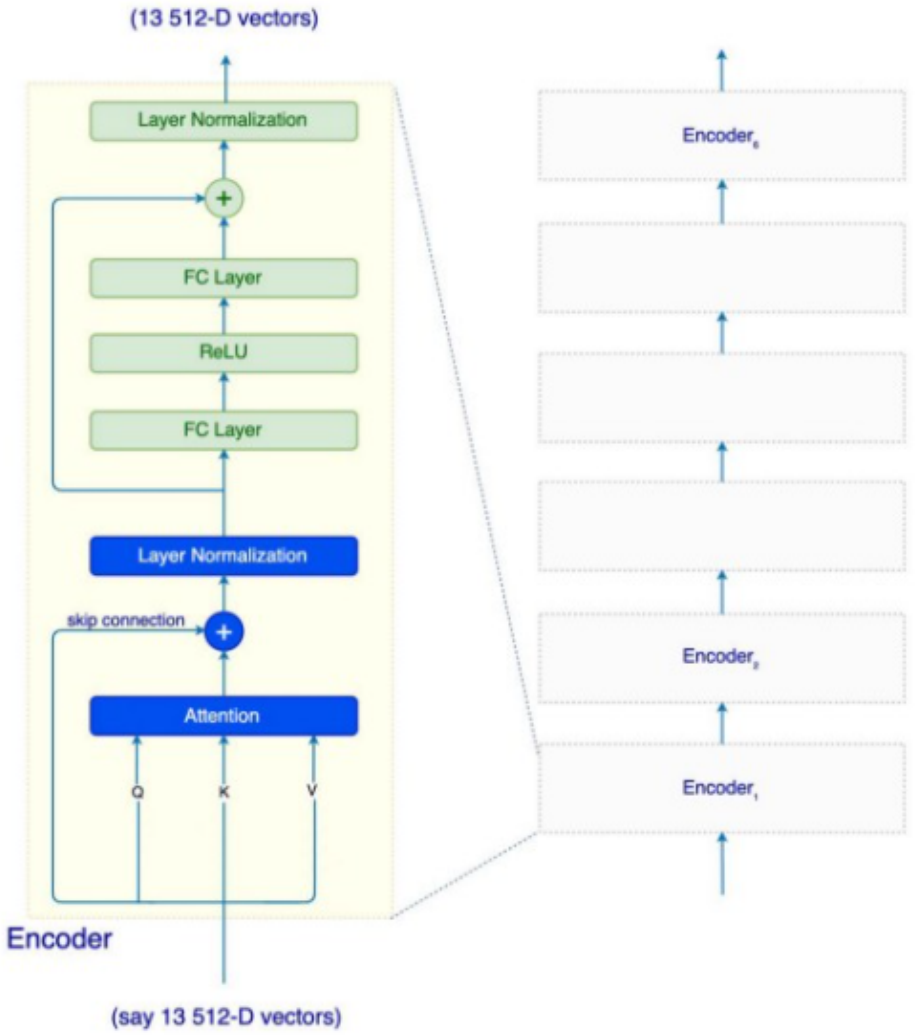

Figura 2.9: En el codificador del Transformer, se apilan seis codificadores.

Para el primer codificador, la entrada será los *word embedding* iniciales. Esta pila de codificadores se puede apreciar en la figura 2.9

En cada codificador, primero se aplica la atención. En el ejemplo, se tienen 13 palabras y por lo tanto 13 consultas. Las 13 atenciones se computan al mismo mediante productos escalares. Sin embargo, existe la atención múltiple (*multi-head*), la cual genera h atenciones por cada consulta. Conceptualmente, se agrupan h atenciones juntas. En el Transformer, se usan 8 atenciones por consulta, para tener 8 conjuntos diferentes de proyecciones. Esto nos da 8 perspectivas diferentes, y eventualmente aumentará la precisión del sistema. En la figura 2.10, se puede apreciar el codificador usando la atención múltiple.

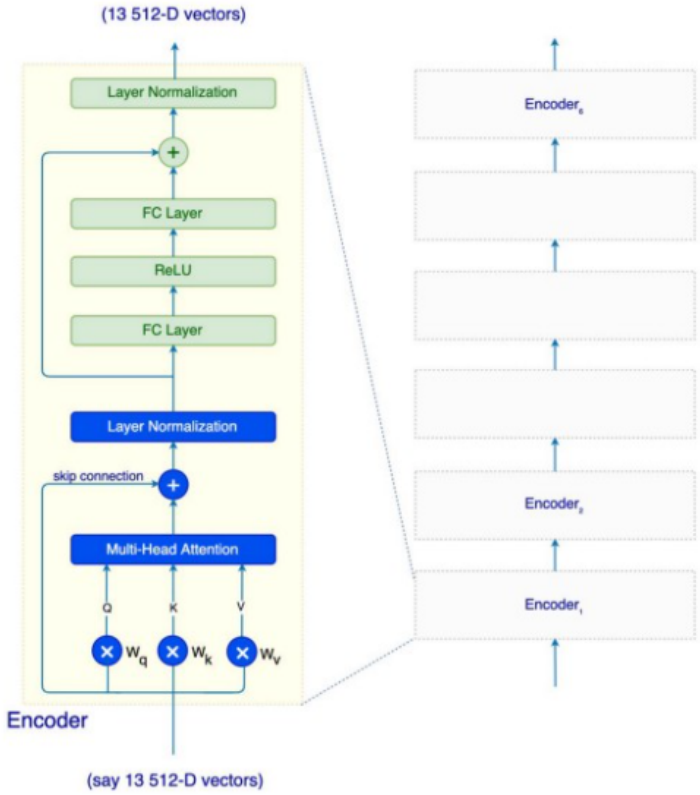

Figura 2.10: Codificador del Transformer usando atención múltiple

En la figura 2.10, también se muestra que el Transformer aplica la *skip connection* (bloques residuales) a la salida de la atención múltiple, seguido de una capa de normalización. Ambas técnicas permiten que el entrenamiento sea más fácil y estable. En la capa de normalización ,se usan valores de la misma capa para realizar la normalización. Tampoco es necesario profundizar en estas dos técnicas.

Después, a los resultados de la atención se aplica una capa totalmente conectada, una función de activación ReLU [10] y otra capa totalmente conectada.

El Transformer codifica todas las palabras de una frase al mismo tiempo. Se espera que finalmente el modelo extraerá y utilizará la posición de cada palabra y que ordenará la información. Si esto falla, dos frases con significado totalmente diferente podría codificarse de forma similar. Una solución a este problema es aportar información sobre la posición de una palabra en su mismo *embedding*, tal y como se muestra en la figura 2.11

[10] https://es.wikipedia.org/wiki/Rectificador_(redes_neuronales)

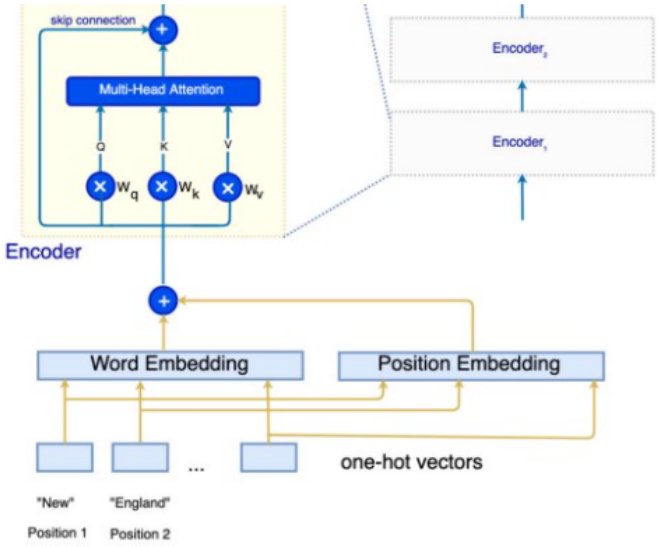

Figura 2.11: Para ayudar en la correcta codificación de una frase, se provee información sobre la posición de cada palabra.

2.2.4. BERT

Ahora que conocemos las bases teóricas necesarias para entender cómo funciona BERT, veamos más en profundidad el propio modelo, que como se ha mencionado anteriormente, utiliza el codificador del Transformer para crear la representación del vector.

Representaciones de la entrada y la salida

El modelo necesita una o dos secuencias para resolver diferentes tareas del procesamiento del lenguaje. Todas las entradas empezarán con un token especial "[CLS]". Si la entrada está compuesta por dos secuencias, se separarán por un token "[SEP]". Si la entrada tiene T tokens, incluidos los token especiales, también habrá T salidas. Esto se puede apreciar en la figura 2.12. Diferentes partes de la salida serán usadas para hacer predicciones en diferentes tareas. La primera salida es C, y es la única salida que se utiliza para cualquier tarea de clasificación. Para cualquier otra tarea que sea de no clasificación y que sea de solamente una secuencia, se usan las salidas restantes.

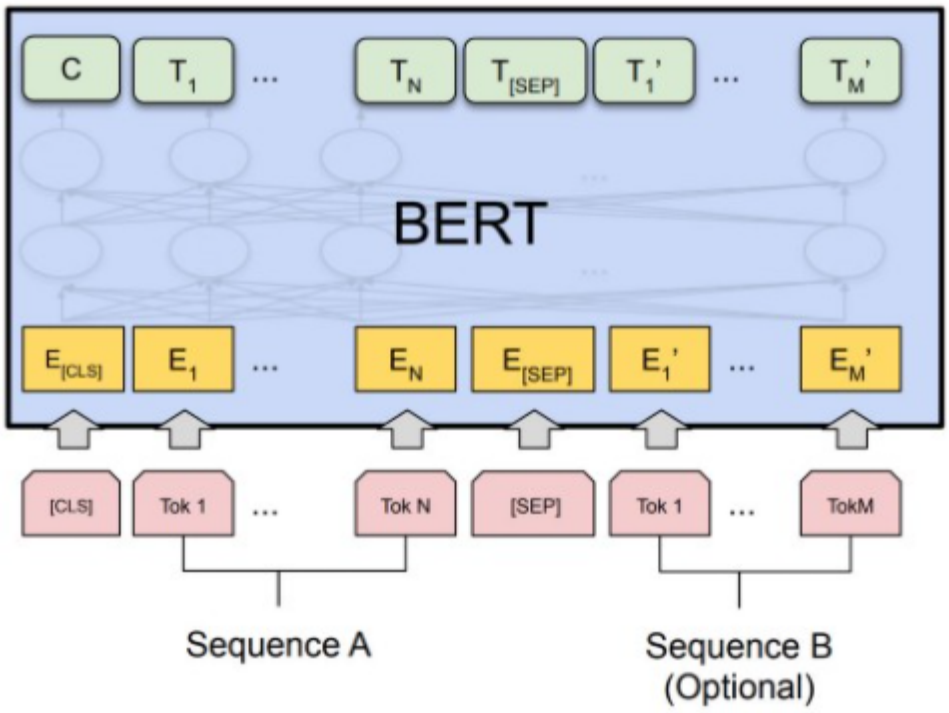

Figura 2.12: Entradas y salidas del modelo BERT

En BERT, las palabras se separan en piezas. Por ejemplo, la palabra “helping” se descompondría en “help” y ´´ing”. El *embedding* de entrada se compone por el *embedding* de la pieza de la palabra, el *embedding* de segmento y el *embedding* de posición.

El preentrenamiento

BERT se preentrena con dos tareas: *Masked Language Model (Masked LM)* y *Next Sentence Prediction (NSP)*

En la tarea de *Masked LM*, el modelo tiene que reconstruir oraciones donde algunas palabras están enmascaradas. Se enmascaran el 15 % de las palabras. Algunas de estas serán reemplazadas por el token "[MASK]", otras serán reemplazadas por un token aleatorio, y otras mantendrán el token original.

En la segunda tarea, *NSP*, el objetivo es crear una salida C que codifique la relación entre las secuencias de entrada A y B. En el 50 % de las ocasiones en el entrenamiento, BERT usa dos secuencias consecutivas. En esos casos, se espera que el modelo lo prediga con un “IsNext”. En el resto de casos, las secuencias escogidas serán aleatorias y el modelo tiene que predecir “NotNext”.

Finetuneando BERT

Una vez el modelo es preentrenado, podemos *fine-tunearlo* para que resuelva otras tareas. Se pueden añadir clasificadores para refinar los parámetros del modelo. BERT se asemeja más a una estrategia de entrenamiento que a una arquitectura.

2.3. Estado del arte

La tarea de los diccionarios inversos ha sido ampliamente investigada en muchos estudios. En este capítulo, se revisarán varios sistemas publicados en la actualidad que cumplen eficientemente la tarea.

El primer intento que se conoce de crear un diccionario inverso fue en 2004. [Bilac et al., 2004] propuso la técnica de recuperación de información para resolver el problema propuesto. Primero, creó una base de datos con varios diccionarios. Cuando llegaba una consulta, se buscaba en la base de datos la definición más cercana y se devolvía la palabra correspondiente. Para medir la distancia entre las definiciones, es posible utilizar diferentes métricas, tales como la distancia euclídea [11], la distancia Chebyshov [12], los espacios Lp [13], o la geometría del taxista [14] [Johnson et al., 2017]. Más tarde, [Shaw et al., 2011] mejoró este mismo sistema con la inclusión de la base de datos léxica en inglés Wordnet [Miller, 1995]. Es este mismo planteamiento el que se utiliza en el sistema propuesto 4.2, en el capítulo 4.

Desde entonces, han ido proponiéndose muchos sistemas de diccionarios inversos, los cuales llegan a obtener resultados francamente buenos. A continuación se presentan algunos.

2.3.1. WantWords

El primer sistema que veremos en profundidad se trata del diccionario inverso *open-source* WantWords [15] [16] [Qi et al., 2020, Zhang et al., 2020].

[11] https://es.wikipedia.org/wiki/Distancia_euclidiana
[12] https://es.wikipedia.org/wiki/Distancia_de_Chebyshov
[13] https://es.wikipedia.org/wiki/Espacios_Lp
[14] https://es.wikipedia.org/wiki/Geometría_del_taxista
[15] https://wantwords.thunlp.org/
[16] https://github.com/thunlp/WantWords

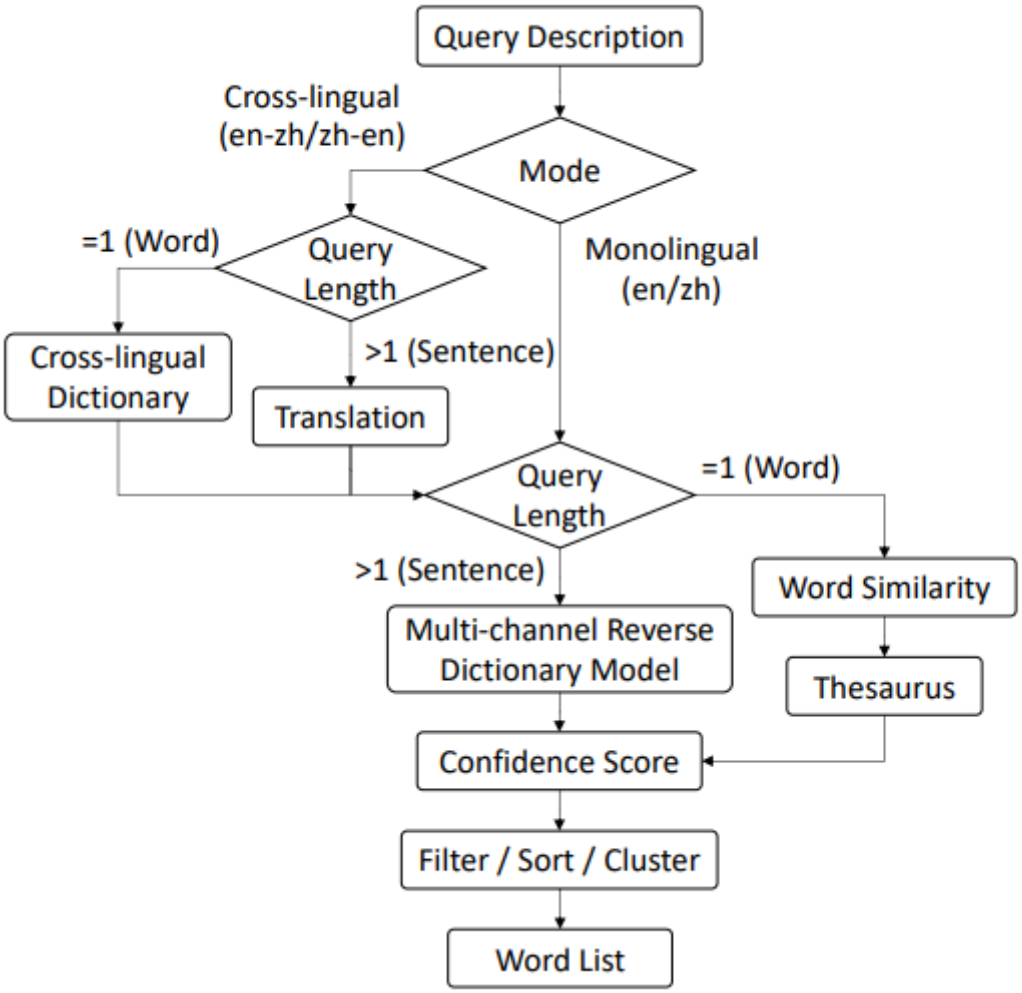

Figura 2.13: Flujo de trabajo de WantWords

Flujo de trabajo general

WantWords es un diccionario inverso versátil. Por una parte, sirve como diccionario inverso monolingüe en inglés y en chino, y por otra, también sirve como diccionario inverso translingual. Es decir, devuelve la palabra concepto en un idioma, siendo la definición de entrada de un idioma diferente. El flujo de trabajo de WantWords se muestra en la figura 2.13.

Tal y como se ve en la figura y como se ha mencionado anteriormente, hay dos modos de diccionario inverso: el monolingüe y el translingual.

En el modo monolingüe, si la descripción de la consulta es más larga que de una sola pala-

bra, se pasará directamente al modelo de diccionario inverso multicanal (Apartado 2.3.1), el cual calcula una puntuación de confianza para cada palabra candidata del vocabulario. Si la descripción de la consulta es de una sola palabra, dicha puntuación de confianza se basa principalmente en la similitud coseno [17] entre los *embeddings* (Explicados en el apartado de teoría 2.1) de la palabra consultada y la palabra candidata.

En el modo translingual, donde existen el idioma fuente y el idioma objetivo, si la descripción de la consulta es más larga que de una sola palabra, primero se traduce al idioma objetivo y después es procesada en el modo monolingüe. En cambio, si la descripción es de una sola palabra, se consultan diccionarios translinguales para las definiciones de la palabra consultada y después esas definiciones se pasan al modelo de diccionario inverso multicanal para calcular las puntuaciones de confianza.

Tras obtener todas las puntuaciones de confianza, todas las palabras candidatas del vocabulario se ordenan descendentemente y listadas como salida del sistema. Las palabras que están incluidas en la consulta son excluidas ya que es improbable que sean la palabra objetivo.

Modelo de diccionario inverso multicanal (MRDM)

El modelo de diccionario inverso multicanal (MRDM) es el núcleo del sistema WantWords. En el modelo, se emplea el modelo de lenguaje BERT [Devlin et al., 2019] como codificador de oraciones. La figura 2.14 ilustra el modelo.

Para cualquier consulta, MRDM calcula una puntuación de confianza para cada palabra. Esta puntuación está compuesta por cinco partes:

1. la primera parte es la puntuación de la palabra. Para su obtención, la descripción de entrada se codifica a un vector por BERT. Luego, el vector es mapeado a un espacio de *embeddings* por un perceptrón de una capa (*single-layer perceptron*), y finalmente, la puntuación de la palabra es el producto escalar entre el vector mapeado y el *embedding* de la palabra candidata.

2. La segunda parte es la puntuación de la oración (*Part-of-speech*, PoS), la cual está basada en la predicción para la PoS de la palabra objetivo. MRDM primero calcula una predicción de la puntuación para cada PoS tag, y finalmente, la puntuación PoS

[17] https://es.wikipedia.org/wiki/Similitud_coseno

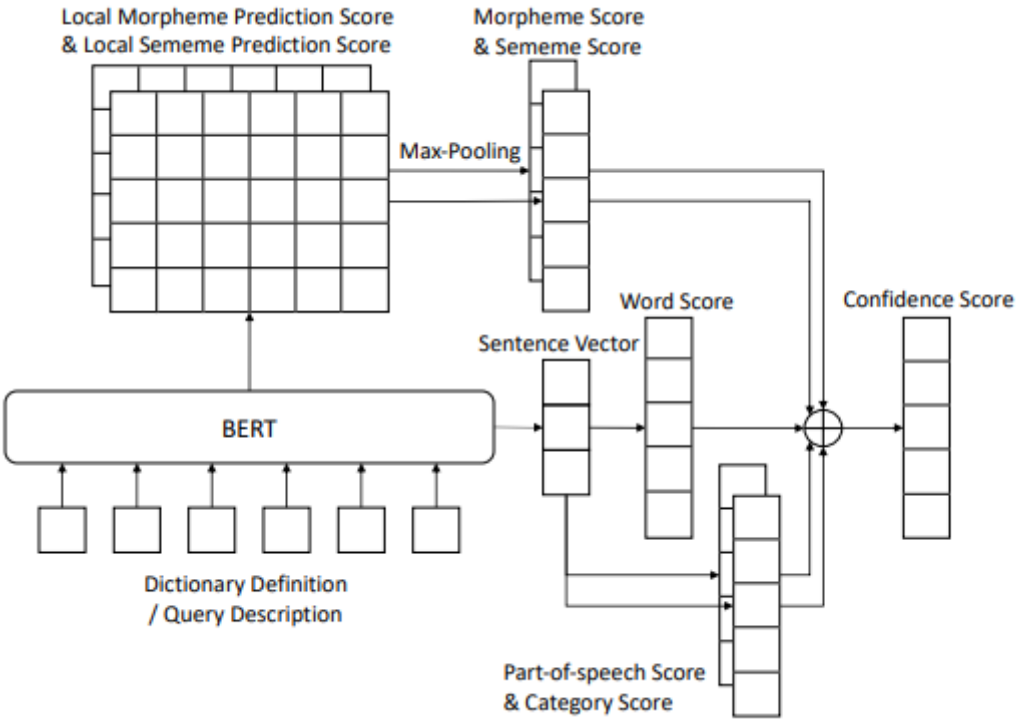

Figura 2.14: Versión revisada del modelo de diccionario inverso multicanal

de la palabra candidata es la suma de todas las puntuaciones predichas de cada PoS tag.

3. La tercera parte es la puntuación de categoría. Esta puntuación está relacionada con la categoría de la palabra objetivo y se obtiene de manera similar a la puntuación PoS.

4. La cuarta parte es la puntuación del morfema, que debe capturar los morfemas de la palabra objetivo. Para cada token de la descripción, se obtiene una puntuación local del morfema mediante un perceptrón de una capa. Después, para cada morfema se obtiene una puntuación de predicción tras hacer un *max-pooling* [18], y finalmente, la puntuación del morfema de la palabra candidata es la suma de las puntuaciones predichas de todos sus morfemas.

5. La quinta y última parte es la puntuación del semema, que está basada en la predicción de los sememas de la palabra objetivo y es calculada de manera similar a la puntuación del morfema.

[18] https://computersciencewiki.org/index.php/Max-pooling_/_Pooling

Tanto para los diccionarios inversos monolingües en inglés como para en chino, se usan los modelos oficiales preentrenados BERT. Para el *fine-tuning* (entrenamiento) en inglés, se usa el conjunto de datos de definiciones de diccionarios creado por [Hill et al., 2015], el cual contiene 100.000 palabras y 900.000 definiciones, extraídas de cinco diccionarios. Para el *fine-tuning* en chino, se usa un *dataset* creado a partir del conjunto de datos creado por [Zhang et al., 2020]. Contiene 137.174 palabras y 270.549 definiciones, extraídas de varios diccionarios. MRDM requiere algunos otros recursos, y en [Qi et al., 2020] se explica que se sigue el marco explicado en [Zhang et al., 2020]. Específicamente, para el inglés, usan Morfessor [Virpioja et al., 2013] para segmentar las palabras en morfemas, usan WordNet [Miller, 1995] para obtener PoS y la información respecto a la categoría, y OpenHowNet [19] [Qi et al., 2019] para obtener la información de los sememas. Para el chino, se usan los caracteres chinos como morfemas y se utilizan los PoS tags en el *Modern Chinese Dictionary*. Por último, usan HIT-IR Tongyici Cilin [20] para la información de la categoría de la palabra y OpenHowNet para la información de los sememas.

Evaluación

Para la evaluación del sistema en inglés, se usa el conjunto de datos estandarizado creado por [Hill et al., 2015]. Este conjunto de datos se usa para evaluar la mayoría de sistemas de diccionarios inversos. El *dataset* se detalla en la sección 4.4. En este caso, solo se ha evaluado el sistema en las dos últimas tareas: en las definiciones no vistas durante el entrenamiento y en las descripciones de definciones Por otro lado, para la evaluación en chino, se usa el conjunto de datos creado por [Zhang et al., 2020]. Está formado por tres grupos:

1. Conjunto de definiciones: contiene 2.000 pares de palabras y definiciones de diccionarios que se seleccionan aleatoriamente y no se encuentran en el conjunto de datos de entrenamiento.

2. Conjunto de descripciones: se compone de 200 descripciones de palabras dadas por hablantes nativos chinos.

3. Conjunto de preguntas: colecciona 272 preguntas y respuestas de exámenes chinos reales que consisten en escribir la palabra correcta dada su descripción de Internet.

[19] https://github.com/thunlp/OpenHowNet
[20] https://github.com/yaleimeng/Final_word_Similarity/tree/master/cilin

Modelo	Inglés-Chino		Chino-Inglés	
MRDM	40	.12/.31/.63	8	.20/.52/.76
BERT	**16**	.14/.40/.75	7	.21/.54/.76
WantWords	19	.14/.38/.76	8	.22/.53/**.78**

Tabla 2.1: Resultados de las evaluaciones de diccionarios inversos translingüales. En cada celda, la mediana y la precisión@1/10/100

Para evaluar el rendimiento del diccionario translingual, construyen dos conjuntos de prueba basadas en dos conjuntos de descripciones monolingües. Traducen manualmente las palabras del conjunto de descripciones en inglés al chino para obtener el conjunto de descripciones ingles-chino, que se compone de 200 pares de descripciones en inglés y palabras chinas. De manera similar, construyen el conjunto de descripciones chino-inglés.

Los resultados de las evaluaciones se pueden observar en la tabla 2.2, que recoge los resultados de todos los sistemas revisados en este capítulo, y el que se usará más adelante para comparar los sistemas propuestos en este TFG.

Teniendo como baseline el sistema OneLook, se puede observar cómo WantWords consigue mejor resultado que el famoso sistema online y consigue un rendimiento de última generación en los diccionarios monolingües. En la figura 2.1 se observan los resultados de los diccionarios inversos translinguales. En este caso, si se compara con los sistemas de la baseline, los sistemas básicos de MRDM y BERT, los tres prototipos alcanzan unos resultados similares.

2.3.2. BERT for Monolingual and Cross-Lingual Reverse Dictionary

El segundo sistema que revisaremos se trata de BERT for Monolingual and Cross-Lingual Reverse Dicionaty [Yan et al., 2020]. Como su título indica, en este sistema también usan el modelo del lenguaje BERT para resolver el problema. Sin embargo, su arquitectura difiere del primer prototipo estudiado.

En este apartado, el problema del diccionario inverso se referirá como el problema de encontrar la palabra clave w dada su definición $d = [w_1, w2_2, ..., w_n]$

BERT

En el apartado 2.2 de las bases teóricas, se explica más detalladamente cómo funciona este modelo ofrecido por HuggingFace y el funcionamiento de los Transformers. Con-

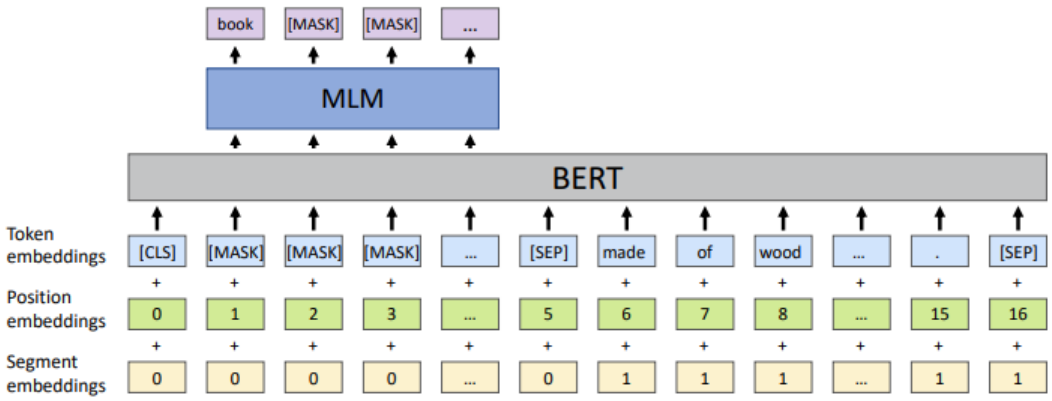

Figura 2.15: Estructura del modelo para el diccionario inverso monolingüe y translingual. "[MASK].[en] la entrada es el marcador de posición donde BERT necesita hacer la predicción. Los marcadores de posición se concatenan con la definición de la palabra antes de mandarla a BERT. Se necesita un procesamiento posterior para convertir la predicción para "[MASK].[en] un *ranking* de palabras.

cretamente, BERT contiene varias capas codificadoras de Transformer. BERT se puede formular mediante las ecuaciones 2.1 y 2.2.

$$\hat{h}^l = LN(h^{l-1} + MHAtt(h^{l-1})) \tag{2.1}$$

$$\hat{h}^l = LN(\hat{h}^l + FFN(\hat{h}^l)) \tag{2.2}$$

h^0 es la entrada BERT. Para cada *token*, la entrada es la suma del *token embedding*, *position embedding*, y *segment embedding*. LN es la capa de normalización. MHAtt es la atención multi-cabeza. FFN contiene tres capas. La primera es una capa linear de proyección, la segunda una capa de activación y la última otra capa lineal de proyección. l es la profundidad de las capas. La cantidad total de capas de BERT es 12 o 24.

Para pre-entrenar BERT, se usaron 2 tareas, explicadas en el subapartado 2.2.4. Un dato importante es que, en vez de usarse directamente las palabras, BERT usa las subpalabras BPE [Sennrich et al., 2016] por lo que una palabra puede que se divida en varios *tokens*.

BERT para el diccionario inverso monolingüe

La estructura del modelo se puede ver en la figura 2.15. La secuencia de entrada x debe tener la forma "[CLS] + [MASK] $*k$ + [SEP] + [secuencia de subpalabras de la definición d] + [SEP]". Se quiere que BERT recupere la palabra clave w de los k "[MASK]"*tokens*, basándose en la definición d. Primero, se utiliza BERT para predecir las máscaras. Se puede formular como la ecuación 2.3

$$S_{subword} = MLM(H_k^L) \tag{2.3}$$

En la ecuación, H_k^L son los estados de los k *tokens* enmascarados en la última capa, MLM es el modelo del lenguaje de máscaras pre-entrenado, y $S_{subword}$ es la distribución de puntuaciones de las k posiciones. Aunque se puede predecir la palabra directamente con BERT, se sufrirán dos problemas de esta manera: el primero es que no se puede obtener ninguna ventaja de las subpalabras habituales de las palabras, tales como los prefijos y los sufijos. El segundo problema es que predecir palabras es inconsistente con las tareas del pre-entrenamiento.

Después de obtener las $S_{subword}$, se necesita convertirlas a puntuaciones de palabra. Sin embargo, hay demasiadas combinaciones posibles de subpalabras, por lo que representar palabras cruzando subpalabras se vuelve computacionalmente intratable. Para esta tarea, el número posible de palabras objetivo posible es fijo ya que la cantidad de palabras únicas en un diccionario es única. Por ello, en vez de combinar secuencias de palabras, podemos fijarnos solamente en la secuencia de subpalabras que pueda formar una palabra válida.

Específicamente, para un idioma, primero se listan todas las palabras válidas y se encuentra la secuencia de subpalabras de cada palabra. Para una palabra w con la secuencia de subpalabras $[b_1, ..., b_k]$, su puntuación es calculada con la ecuación 2.4.

$$S_{word} = \sum_{i=1}^{k} S^i_{subword}[b_i] \tag{2.4}$$

En la ecuación, S_{word} es la puntuación de la palabra w, $S^i_{subword}$ es la distribución de puntuación de la subpalabra en la posición i y $S^i_{subword}[b_i]$ es la agrupación del elemento b_i con el elemento en $S^i_{subword}$. Sin embargo, no todas las palabras se pueden descomponer en k subpalabras. Si una palabra tiene menos subpalabras que k, se rellena con "[MASK]". Por otro lado, el sistema no puede manejar palabras con más de k subpalabras. Con este

método, cada palabra obtiene una puntuación. Por eso, se puede usar la pérdida de entropía para *fine-tunear* el modelo, mediante la ecuación 2.5, donde N es el número total de muestras y w es la palabra clave. A la hora de realizar el ranking, todas las palabras se ordenan según su puntuación.

$$L_w = -\sum_{i=1}^{N} w^{(i)} log\_softmax(S_{word}^{(i)}) \tag{2.5}$$

BERT para el diccionario inverso translingual

Para el diccionario inverso translingual, se usa la misma arquitectura, la de la figura 2.3.2. La única diferencia entre las dos versiones es el modelo pre-entrenado usado. En este caso, se usa mBERT en vez de BERT. mBERT tiene la misma estructura que BERT, pero fue entrenada en 104 idiomas. Por lo tanto, su *token embedding* contiene subpalabras en distintos idiomas.

BERT para el diccionario inverso translingual no alineado

La arquitectura usada para esta versión se puede ver en la figura 2.16. Comparándola con BERT, se añade un *embedding* de idioma extra en el inferior, que tiene el mismo tamaño que los otros *embeddings*. Exceptuando el *embedding* inicializado aleatoriamente que se añade, el modelo es inicializado con el modelo mBERT pre-entrenado.

En esta versión, en vez de usar MLM para obtener $S_{subword}$, se usa la ecuación 2.6, donde Emb_{token} son los *embeddings* de los *token* de las subpalabras. Esta fórmula permitirá un rendimiento mejor que usando MLM, presuntamente porque los datos de entrenamiento solo son de un único idioma, por lo que le resultará muy difícil al modelo predecir palabras en otro lenguaje. Sin embargo, si se usa Emb_{token}, el modelo puede utilizar la similitud entre las subpalabras para realizar predicciones razonables.

$$S_{subword} = H_k^L Emb_{token}^T \tag{2.6}$$

Tras obtener $S_{subword}$, se utiliza la ecuación 2.4 para obtener la puntuación de cada palabra. Cada idioma tiene diferentes listas de palabras, y la pérdida se calcula mediante la ecuación 2.7, donde M es la cantidad de idiomas, N_k es la cantidad de muestras del idioma j, $w_j^{(i)}$ es la palabra clave en el idioma j, y $S_{word_j}^{(i)}$ es la distribución de la puntuación

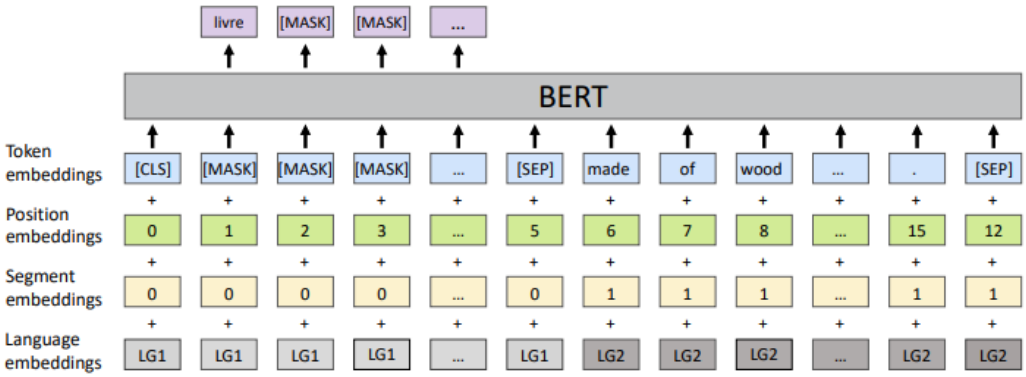

Figura 2.16: Estructura del modelo para el diccionario inverso translingual no alineado. Se añade un *embedding* de idioma inicializado aleatoriamente para distinguir los idiomas. Como solo se tienen datos de entrenamiento monolingües, "LG1τ "LG2"toman el mismo valor en la fase de entrenamiento, pero diferentes en la fase de evaluación.

para las palabras en el lenguaje j. Cuando se obtiene el ranking para un idioma, solo se calculan las puntuaciones de palabra de ese mismo idioma.

$$L_w = -\sum_{j=1}^{M}\sum_{i=1}^{N_j} w_j^{(i)} log\_softmax(S_{word_j}^{(i)}) \tag{2.7}$$

Evaluación

Para la evaluación del prototipo, de nuevo se usa el conjunto de datos estandarizado creado por [Hill et al., 2015] detallado en el apartado 4.4, como en el sistema anterior.

Los resultados monolingües en inglés se pueden ver en la tabla 2.2. OneLook es el diccionario inverso más usado comercialmente. "SuperSense", RDWECI", "MS-LSTMτ "Mul-Channel"son de [Pilehvar, 2019, Morinaga and Yamaguchi, 2018, Kartsaklis et al., 2018, Zhang et al., 2020]. Como se puede observar, en el conjunto de datos de descripción, RoBERTa alcanza unos resultados de última generación. En el caso de los conjuntos de definiciones vistas y no vistas, BERT y RoBERTa mejoran el rendimiento de "Mul-Channel". Aunque MS-LSTM consiga una puntuación remarcablente alta en el conjunto de datos de definiciones vistas, falla al reconocer definiciones que no han sido vistas con anterioridad y las descripciones humanas. En cambio, BERT y RoBERTa consiguen un aumento de rendimiento considerable en los conjuntos de definiciones no vistas y descripciones respecto a los demás sistemas.

Model	Seen			Unseen			Description		
OneLook	0	.66/.94/.95	200	-	-	-	5.5	.33/.54/.76	332
WantWords	-	-	-	19	.10/.38/.72	-	2	.36/.75/.92	-
RDWECI	121	.06/.20/.44	420	170	.05/.19/.43	420	16	.14/.41/.74	306
SuperSense	378	.03/.15/.36	462	465	.02/.11/.31	454	115	.03/.15/.47	396
MS-LSTM	**0**	**.92/.98./.99**	65	276	.03/.14/.37	426	1000	.01/.04/.18	404
Mul-Channel	16	.20/.44/.71	310	54	.09/.29/.58	358	2	.32/.64/.88	203
BERT	**0**	.57/.86/.92	240	**18**	**.20/.46/.64**	418	1	.36/.77/.94	94
RoBERTa	**0**	.57/.84/.92	228	37	.10/.36/.60	405	**1**	**.43/.85/.96**	46

Tabla 2.2: Resultados en el diccionario inverso en inglés. En cada celda, los valores corresponden a "Mediana", "Precisión@1/10/100z "Varianza". Los resultados son de [Zhang et al., 2020] y [Yan et al., 2020]

2.3.3. Scalable Database-Driven Reverse Dictionary

El último sistema que revisaremos se trata del diccionario inverso escalable impulsado por conjuntos de datos (Scalable Database-Driven Reverse Dictionary) [Shaw et al., 2013]

Estrategia de solución propuesta

Se define como “término” a cualquier palabra válida en el diccionario inglés. Una frase, P, se define como una secuencia de uno o más términos: $P =< t_1, t_2, ..., t_i, ..., t_n >$. Por conveniencia, el hecho de que el término t forme parte de la frase P se denota como $t \in P$. Un diccionario D es el conjunto de mapeos $P \rightarrow P$. Por claridad, se distinguirán dos tipos de frases, las frases de palabra (w) y las frases de significado (S), donde W se refiere a una palabra o secuencia de palabras indexadas en un diccionario, y S se refiere a la definición de W en el diccionario. Por lo tanto, bajo esta clasificación, un diccionario se redefine como un conjunto de mapeos $W \rightarrow S$.

En alto nivel, la estrategia de la solución consiste en dos pasos secuenciales. Cuando se recibe la consulta del usuario, primero se buscan palabras candidatas tomando como fuente un diccionario convencional, donde las definiciones de esas palabras candidatas tienen cierta similitud con la entrada del usuario. Después, se ordenan las palabras candidatas según su calidad de emparejamiento. El primer paso mencionado, consta de dos subpasos: construir los conjuntos de mapeo inverso (RMS) y consultar dicho RMS.

La arquitectura del diccionario inverso se puede ver en la figura 2.17.

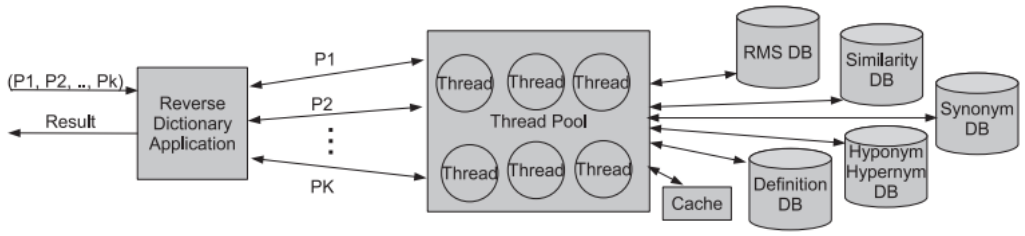

Figura 2.17: Arquitectura del diccionario inverso.

Construir el RMS

Intuitivamente, construir el RMS de un término t, $R(t)$, es cuestión de encontrar todas las Ws cuya definición contenga el término t. Sin embargo, dado el tamaño de un diccionario común, esto es inviable. Por ello, precrean esos Rs para cada término relevante en el diccionario. Esta es una tarea que se realiza una sola vez y de manera *offline*, así que el coste de crear este corpus no interfiere con el rendimiento de ejecución. Para un diccionario de entrada D, se crean R mapeos para todos los términos en su forma más genérica que aparecen en las frases de significado (S) en D. Para cada término genérico $\hat{t}$ que aparece en S, se añade la frase de palabra correspondiente, W, a $R(\hat{t})$ y se incrementa el número de apariciones $N(\hat{t})$

Consultar el RMS

Cuando se recibe una frase como entrada, se consultan las R ya presentes en la base de datos. Para entender este paso, se explica mediante un ejemplo: en la entrada U "un edificio alto", primero se extraen los términos principales: "alto" y "edificio". Luego, se consulta en la base de datos $R(alto)$ y $R(edificio)$, para encontrar aquellas palabras en cuyas definiciones aparezcan "alto" y "edificio" simultáneamente. Cada una de esas palabras se convierte en un palabra candidata. Si no se alcanza el mínimo de palabras candidatas predefinido, se amplía el alcance para incluir sinónimos, hipónimos, e hiperónimos de los términos. Cuando se tienen las palabras candidatas necesarias, se ordenan en base a su similitud con U y se devuelven los primeros βWs, donde β es un parámetro tuneable.

Para ordenar las palabras candidatas, se basan en la similitud semántica de las definiciones $S_1, ..., S_x$ de W comparado con U. Para realizar este orden, se necesita asignar una medida de similitud entre cada par (S, U). En esta tarea, se ha de tener en cuenta que el orden de la secuencia de palabras es importante, y que algunas palabras contribuyen más

al significado de la frase que otras. Por lo tanto, se necesitan medidas para la similitud entre dos términos y para la importancia de un término. Se pueden generar similitudes con peso para cada par de términos (a,b) donde $a \in S$ y $b \in U$, de forma que si a y b son conceptualmente similares, y las dos tienen gran importancia en la frase en la que aparecen, entonces el par (S,U) sería medido como más similar que el par (S',U'), donde U y U' contienen el mismo conjunto de términos, pero a es más importante en el significado de S que en el de S'.

Primero se considera la similitud entre términos. Se computa la similitud $p(a,b)$ basándose en [Wu, 1995], para dos términos a y b según su ubicación en la jerarquía de WordNet, donde se organizan las palabras del lenguaje inglés desde las raíces hasta los nodos hoja más específicos. Intuitivamente, dos términos tendrán poca similitud si su ancestro común menor es la raíz, y será mayor cuanto más profundo se encuentre este ancestro en la jerarquía. Siendo $A(a,b)$ dicho ancestro y $E(t)$ la profundidad del término t en la jerarquía, la similitud $p(a,b)$ se calcula mediante la fórmula 2.8.

$$p(a,b) = \frac{2 \times E(A(a,b))}{(E(a)+E(b))} \tag{2.8}$$

Después se considera la importancia del término t en una frase P, $\lambda(t,P)$. Para generar la importancia de cada término, se usa un analizador sintáctico. Por ejemplo OpenNLP [21], que devuelve la estructura gramatical de una frase en forma de árbol sintáctico. En este árbol, las palabras que contribuyen más a la frase aparecen más arriba, y las menos significativas más abajo. Conociendo esto, para un árbol sintáctico de la frase P, $Z(P)$, d_t se usará para denotar la profundidad del término t en el árbol, donde el árbol tiene una profundidad total de $d(Z(P))$. La importancia del término t en la frase P se calcula finalmente mediante la fórmula 2.9

$$\lambda(t,P) = \frac{(d(Z(P)) - d_t)}{d(Z(P))} \tag{2.9}$$

Finalmente, las similitudes con peso para cada par de términos donde $a \in S$ y $b \in U$ se computa mediante la fórmula 2.10

$$\mu(a,S,b,U) = \lambda(a,S) \times \lambda(b,U) \times p(a,b) \tag{2.10}$$

A partir de esta matriz de similitudes, se usa el algoritmo descrito en [Dao and Simpson,

[21] https://opennlp.apache.org/

2005] para obtener la similitud entre la frase de significado S y la frase dada por el usuario U, y finalmente se pueden devolver los primeros β mejores emparejamientos.

3. CAPÍTULO

Gestión del proyecto

Este capítulo sirve como documentación de cómo ha sido gestionado el proyecto. Para el desarrollo de este apartado, se han seguido las directrices y recomendaciones del libro PMBoK (Project Management Body of Knowledge)

3.1. Alcance del proyecto

Dentro del alcance del proyecto, se encuentran varios objetivos principales: revisar y estudiar la teoría básica necesaria para entender y adaptar los sistemas propuestos disponibles de diccionarios inversos en inglés, y, a partir de las tecnologías existentes, construir tres prototipos de diccionarios inversos.

3.2. Planificación del proyecto

3.2.1. Diagrama EDT

El diagrama EDT correspondiente a este Trabajo de Fin de Grado se puede ver en la figura 3.1.

- El Paquete de Trabajo de Planificación (P) agrupará las tareas de planificación ini-

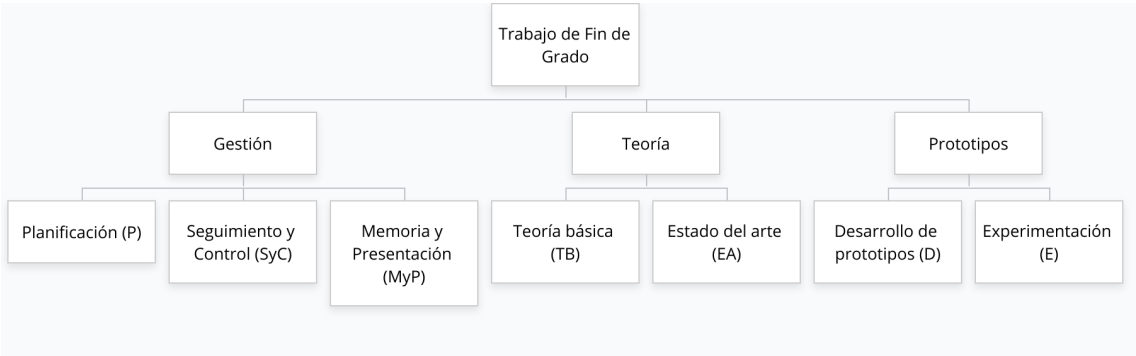

Figura 3.1: Diagrama EDT del TFG

cial, así como aquellas que en su caso fuera necesario realizar para mantener una planificación adecuada.

- El Paquete de Trabajo Seguimiento y Control (SyC) agrupará las tareas necesarias para garantizar el adecuado desarrollo del proyecto y, en particular, el seguimiento de dedicaciones de las tareas del proyecto y cumplimiento de plazos.
- El Paquete de Trabajo Memoria y Presentación (MyP) agrupará las tareas referentes a la elaboración de la memoria del proyecto y de la presentación que se utilizará para su defensa.
- El Paquete de Trabajo de Teoría básica (TB) agrupará las tareas relacionadas con la búsqueda y estudio de información útil para el desarrollo del proyecto y para entender mejor el funcionamiento de los diccionarios inversos existentes.
- El Paquete de Trabajo de Estado del arte (EA) agrupará las tareas del estudio del estado del arte del problema trabajado.
- El Paquete de Trabajo Desarrollo de prototipos (D) agrupará todas las tareas relacionadas con el desarrollo y adaptación de los prototipos propios de diccionarios inversos.
- El Paquete de Trabajo Experimentación (E) agrupará las tareas pertinentes a la experimentación de los prototipos propuestos con objetivo de mejorar su rendimiento.

3.2.2. Entregables e hitos en el desarrollo del proyecto

Los entregables son productos medibles y verificables que se elaboran para completar un proyecto o una aparte del mismo. En este caso, los entregables serán los siguientes:

Hitos	Fecha
Finalización de los Paquetes de Trabajo de Teoría básica (TB) y Estado del arte (EA)	1/05/2021
Finalización de los prototipos	01/08/2021
Entrega de la memoria, el póster y el código	05/09/2021

Tabla 3.1: Hitos del proyecto y sus fechas límite

Paquete de Trabajo /Meses	Febrero	Marzo	Abril	Mayo	Junio	Julio	Agosto	Septiembre
Planificación	✓	✓						
Seguimiento y control		✓	✓	✓	✓	✓	✓	
Memoria y presentación		✓			✓	✓	✓	✓
Teoría básica		✓	✓					
Estado del arte			✓	✓				
Desarrollo de prototipos				✓	✓	✓	✓	
Experimentación					✓	✓	✓	

Figura 3.2: Diagramma de Gantt del periodo de realización de los paquetes de trabajo.

1. Esta propia memoria.
2. Un póster.
3. Los scripts utilizados a lo largo del proyecto, accesibles a través del repositorio de GitHub https://github.com/AnderGil/TFG_Diccionarios_Inversos.
4. Predicciones realizadas por los modelos en las pruebas estandarizadas, también accesibles a través del mismo repositorio.

Los hitos del proyecto y sus respectivas fechas límite se ven en la tabla 3.1

3.3. Periodos de realización de las tareas

3.3.1. Periodo de desarrollo de los paquetes de trabajo

Los periodos de realización de los diferentes paquetes de trabajo se reflejan en el diagrama Gantt 3.2.

3.3.2. Estimación de dedicación de las tareas y desviación

La estimación total de dedicación de este Trabajo de Fin de Grado era de 240 horas, repartidas de la siguiente forma entre los diferentes paquetes de trabajo:

- Planificación: 10 horas
- Seguimiento y control: 10 horas
- Teoría básica: 30 horas
- Estado del arte: 30 horas
- Desarrollo de prototipos: 60 horas
- Experimentación: 50 horas
- Memoria y presentación: 50 horas

Finalmente, ha habido alguna desviación respecto a la planificación inicial: en el desarrollo de los prototipos se han invertido aproximadamente 10 horas menos de las esperadas mientras que en la experimentación se han necesitado 10 horas más, por lo que tampoco ha supuesto ningún mayor inconveniente.

3.4. Análisis de riesgos

3.4.1. Detección de riesgos

En el análisis de riesgos, se han detectado algunos riesgos bastante habituales en este tipo de proyectos:

- Riesgo 1: Al trabajar con Google Colaboratory, el tiempo de ejecución está limitado a 12 horas. Para este tipo de pruebas, muchas veces es necesario una ejecución más larga.
- Riesgo 2: Teniendo estrecha relación con el riesgo 1, es posible que para realizar toda la experimentación no se disponga de suficiente capacidad computacional, derivando esto en la imposibilidad de probar todos los prototipos adecuadamente.

- Riesgo 3: Debido a la pandemia del Covid 19 que sufrimos actualmente, cabe la posibilidad de caer enfermo durante un periodo de tiempo razonablemente prolongado, retrasando así todas las tareas. Debido a esta pandemia, también cabe la posibilidad de que algunas herramientas públicas de la Facultad de Ingeniería Informática o de alguna otra entidad sean inutilizables.

- Riesgo 4: otro riesgo a afrontar ha sido el tiempo. Al haber cursado el primer cuatrimestre en otra universidad con el programa Sicue, el desarrollo del TFG comenzó más tarde de lo habitual, en febrero.

3.4.2. Prevención y mitigación de riesgos

Lo ideal es realizar un plan para prevenir todos los riesgos detectados. Sin embargo, también es importante saber cómo mitigarlos en caso de que finalmente ocurran a pesar de los esfuerzos.

- Riesgo 1 y 2: optimizar el código lo máximo posible para reducir los tiempos de ejecución. En caso de que sigan siendo excesivamente largos, se podría usar algún ordenador de la Facultad de alta capacidad computacional para poder realizar las ejecuciones. También se podría guardar el estado de la ejecución para poder terminarla en tantas sesiones de 12 horas como hagan falta.

- Riesgo 3 y 4: Para prevenir el riesgo 3 lo fundamental es seguir las medidas de seguridad recomendadas por las autoridades sanitarias. Para mitigar este riesgo y el riesgo 4, lo ideal es llevar trabajo adelantado por si surge cualquier contratiempo.

4. CAPÍTULO

Desarrollo del proyecto

En este capítulo, se explicarán los prototipos propios diseñados, ya sean creados desde cero o a partir de sistemas existentes. Se verán los conjuntos de datos y herramientas que han sido utilizadas para el correcto funcionamiento de los sistemas, y finalmente, se revisará su eficacia y se realizará un análisis de los resultados obtenidos, comparándolos con los resultados de otros sistemas del estado del arte actual.

Los tres prototipos han sido desarrollados en el entorno de trabajo de Google Colaboratory, dentro de la siguiente carpeta de Google Drive: https://drive.google.com/drive/folders/11vYB17x57Y38ChM7NjshT2dwoDY2OvMi?usp=sharing.

Los Scripts usados a lo largo del proyecto y las predicciones realizadas por los prototipos creados están accesibles a través del repositorio de GitHub: https://github.com/AnderGil/TFG_Diccionarios_Inversos.

4.1. Primer prototipo: diccionario inverso mediante Mask-Filling

Para este primer prototipo, se han usado los modelos ofrecidos por HuggingFace. El modelo principal usado ha sido el modelo BERT (Ver el apartado 2.2 de las bases teóricas) y modelos basados en este, como RoBERTa. En un comienzo también se hicieron pruebas con el modelo BART para conseguir un diccionario inverso translingual, pero los resultados eran muy malos ya que este modelo se centra más en la creación de texto y no en el *mask-filling*, por lo que se decidió concentrarse en el diccionario inverso monolingüe.

Para empezar, se probó la eficacia del *mask-filling* para la predicción de las palabras clave de las definiciones y ver si el sistema iba a ser viable o no. Para ello, se realizaron unas pequeñas pruebas, con las siguientes líneas de código:

```
1  nlpBartBase = pipeline("fill-mask", model="facebook/bart-base")
2  nlpRobertaBase = pipeline("fill-mask", model="roberta-base")
3
4  print("Bart:", nlpBartBase(f"The definition of {nlpBartBase.tokenizer.mask_token} is: a wheeled
     vehicle that has two wheels and is moved by foot pedals."))
5  print("Roberta:", nlpRobertaBase(f"The definition of {nlpRobertaBase.tokenizer.mask_token} is:
    a wheeled vehicle that has two wheels and is moved by foot pedals."))
```

Como se puede ver, se están probando los modelos BART y RoBERTa. En este caso, BART predijo la palabra "a" y RoBERTa la palabra "bycicle". Aquí se aprecia el mal rendimiento del modelo BART que se ha comentado antes. Sin embargo, RoBERTa acertó la palabra, lo que es muy positivo.

Después, se probaron varios modelos base para ver cual de todos conseguía los mejores resultados en un pequeño conjuntos de datos (Ver apartado 4.4) con un *script* evaluador, para decidir a qué modelo de todos aplicar el *fine-tuning*, ya que es un proceso costoso y que no se puede aplicar a todos los modelos. En este *script* evaluador se utiliza el *AutoModelForMaskedLM* de cada modelo para cumplir la tarea de *mask-filling*, y devuelve cuantas definiciones acierta cada modelo en precisión p@1, p@2, p@3, p@4 y p@5. Los resultados se pueden ver más en profundidad en el apartado 4.5.1, pero el modelo que obtuvo la mejor eficacia fue el de RoBERTa-Large, un modelo basado en BERT. Por ello, es a este al modelo al que se le aplicó el *fine-tuning* con el corpus de WordNet entero (exceptuando las definiciones del conjunto de definiciones no vistas durante el entrenamiento). Para cerciorarme de que los demás modelos seguían obteniendo peores resultados aún después de aplicar un *fine-tuning*, se realizó un pequeño *fine-tuning* a los modelos BART y RoBERTa-Base con *datasets* de 100, 1.000 y 10.000 definiciones, y los resultados seguían siendo peores que los obtenidos con RoBERTa-Large.

Predecir definiciones mediante *mask-filling* tiene una gran desventaja: en la tarea de *mask-filling*, los modelos rellenan la máscara con una sola palabra, y las palabras clave de muchas definiciones están formadas por varias palabras. Ese conjunto de palabras no se puede predecir de esta forma. Por eso se intentó usar el modelo creador de texto BART pero no hubo éxito, así que finalmente se ha trabajado con definiciones de una sola palabra. De todas formas, en el conjunto de datos usado para la evaluación (4.4) propuesto por [Hill et al., 2015] también se trabaja con definiciones de una sola palabra.

4.2. Segundo prototipo: diccionario inverso mediante Sentence-Transformers

La idea fundamental de este segundo prototipo se basa en la misma idea que el sistema propuesto en [Shaw et al., 2013], explicado anteriormente en el apartado 2.3.3: se elegirá la palabra clave que tenga la mayor similitud semántica con las definiciones almacenadas en una estructura de datos.

Para ello, el primer paso es obtener un *dataset* con muchas definiciones de varios diccionarios. (Ver apartado 4.4). Para poder medir la similitud semántica entre palabras, lo que hace este prototipo es representar las definiciones mediante *embeddings* (Ver apartado teórico 2.4). Esta transformación de letras a vectores se hace mediante la herramienta de Sentence-Transformers [1] [Reimers and Gurevych, 2019, Reimers and Gurevych, 2020]. Esta herramienta nos permite codificar numéricamente frases según su significado semántico. Por lo tanto, dos codificaciones de frases que sean parecidas tendrán menos distancia entre ellas que dos codificaciones de frases totalmente diferentes.

Primero, se carga un modelo de los que se encuentran disponibles (Pueden verse en `https://www.sbert.net/docs/pretrained_models.html`). Después, se va leyendo el JSON entero donde están almacenadas las palabras clave y las definiciones y se van guardando en sus respectivas estructuras de datos, y se codifica el vector que contiene todas las definiciones a *embeddings*. Así ya tendremos todas las definiciones representadas con números junto con sus respectivas palabras clave.

El siguiente paso es calcular cual es la definición semánticamente más similar a la que nos pregunte el usuario. Cuando el usuario ingresa una definición, esta se convierte en un *embedding* de la misma forma. Ahora, la tarea trata de encontrar cual es el *embedding* con la menor distancia a este nuevo *embedding* de todos los que hay almacenados. Para ello, habría que comparar todos los vectores uno a uno, por lo que resultaría demasiado costoso. Para paliar con este problema, se hace uso de la herramienta Faiss [Johnson et al., 2017]. Esta herramienta nos permite comparar una gran cantidad de vectores en un tiempo razonable. Faiss dispone de varias métricas para medir la distancia entre los vectores. En este caso, ya que los vectores no están normalizados, se ha usado la métrica de distancia euclídea cuadrada. Tras comparar la representación de la definición dada por el usuario con todas las representaciones almacenadas de las definiciones, se devuelve una lista con las 100 representaciones de menor distancia. Si la palabra clave asociada con alguna de

[1] `https://www.sbert.net/`

estas representaciones concuerda con la que tenía en mente el usuario, significa que el sistema ha predicho la palabra correctamente.

Además de esta lista con las 100 definiciones más próximas, también se devuelve la distancia a la definición con mayor similitud semántica. Si la distancia es 0, significa que la definición del usuario y la almacenada eran exactamente iguales. De esta forma también, en caso de que el prototipo no acierte la palabra clave en los 100 primeros intentos, si esta distancia es pequeña, por lo menos podemos saber que ha devuelto una palabra clave parecida a la original.

4.3. Tercer prototipo: diccionario inverso usando la API datamuse

En una primera instancia, se pretendía construir este último prototipo mediante el traductor automático OpenNMT[2]. La idea era entrenar el modelo para "traducir" las definiciones a sus respectivas palabras clave. Sin embargo, el uso de la herramienta ha dado muchos problemas en el entorno de trabajo, ya que constantemente había errores con los módulos y paquetes requeridos en las diferentes versiones de la herramienta. Por lo tanto, se cambió de idea y finalmente el tercer y último prototipo está construído a partir de la API datamuse[3], un motor de búsqueda de palabras para desarrolladores. El sistema de OneLook[4] también esta creado a partir de esta API.

Para nuestra tarea de crear un diccionario inverso, de entre todas las funciones que ofrece la API se ha usado la de **ml** o *means like*, que se traduce como "significa como". Esta función devuelve como resultado las palabras con el significado más relacionado a un valor de *string*, ya sea una sola palabra o una secuencia de palabras. El resultado que da la API siempre es un JSON con una lista de 100 palabras, tal y como se puede ver en el siguiente ejemplo: si se dirige a la dirección URL `https://api.datamuse.com/words?ml=placewherepeoplelive`, ahí se podrá ver el JSON con el resultado. En este caso, la primera palabra que aparece para la definición de "Lugar donde vive gente" es la palabra clave "Ciudad".

Para acceder a los datos de la API desde nuestro entorno de trabajo Google Colaboratory, se ha usado la biblioteca *Requests*. Usando el comando

[2] https://opennmt.net/
[3] https://www.datamuse.com/api/
[4] https://www.onelook.com/reverse-dictionary.shtml

```
requests.get(url).json()
```

de esta biblioteca obtenemos el JSON de la dirección URL correspondiente. Teniendo esto en cuenta, se ha codificado un *script* que dado un JSON con todas las definiciones que se quieren predecir, se realiza una llamada por cada definición y se obtienen todos los resultados.

4.4. Datasets

En el desarrollo de este TFG, se han utilizado numerosos conjuntos de datos.

El *dataset* usado más importante es el conjunto de datos estandarizado propuesto por Hill en [Hill et al., 2015]. Es un conjunto de datos que se usa para evaluar los diccionarios inversos y poder comparar la eficacia de los mismos en un mismo entorno. Para esta evaluación de los diccionarios inversos, se usan definiciones extraídas del diccionario electrónico *Wordnet* [Miller, 1995], entre las que se distinguen tres tipos:

1. Un conjunto de definiciones vistas. Es decir, se seleccionaron 500 palabras al azar entre las que se usan a la hora de entrenar el modelo, y después se seleccionó aleatoriamente una definición de cada una de las palabras (Ya que cada palabra puede tener varias definiciones distintas). Este conjunto está formado por 500 definiciones.

2. Un conjunto de definiciones no vistas. Las definiciones son elegidas de la misma forma que en el conjunto de definiciones anterior, con la diferencia de que en este caso, las palabras elegidas no han sido vistas en el proceso de entrenamiento. Este conjunto también está formado por 500 definiciones.

3. Finalmente, el último conjunto está formado por 200 descripciones de conceptos que tampoco aparecen en los datos de entrenamiento. Para ello, se seleccionaron 200 adjetivos, sustantivos o verbos de entre los más frecuentes en el *British National Corpus* [Leech et al., 1994] y se les pidió a 10 hablantes ingleses nativos que describiesen esas palabras en una única frase. Para asegurarse de que las definiciones eran suficientemente precisas, se les pidió a otros dos hablantes que adivinasen a qué palabras se referían las descripciones, y si en tres intentos no lo conseguían, se reescribían las descripciones.

A la hora de utilizar este conjunto de datos para realizar la evaluación de un diccionario inverso, para cuantificar la calidad de los resultados se reportan tres estadísticas: la mediana del rango de la respuesta correcta (cuanto más baja mejor), la cantidad de casos en las que la palabra correcta aparece en el top 1/10/100 del ranking (Precision@1/10/100, cuanto más alta mejor) y la varianza del rango de la respuesta correcta (cuanto más baja mejor).

Para el primer prototipo creado (4.1) mediante el uso de Mask-Filling, también se ha usado principalmente el diccionario de WordNet [Miller, 1995] para la optimización del sistema. Este corpus se ha procesado mediante un script, creando así varios ficheros con las definiciones en la forma *The definition of* ***keyword*** *is:* ***definition***.

1. Fichero con 100 definiciones, para evaluar todos los modelos básicos ofrecidos por Hugging Face y así poder elegir el que mejor resultados obtenga y aplicarle el *fine-tuning*.
2. Ficheros de entrenamiento y evaluación con 1.000 y 10.000 definiciones para aplicarles a los modelos BART y RoBERTa-Base un pequeño fine-tuning, por si de esta manera aumentase su rendimiento considerablemente.
3. Fichero de entrenamiento con el conjunto entero de definiciones para *fine-tunear* el modelo básico que ha obtenido los mejores resultados.
4. Fichero de evaluación con el conjunto entero de definiciones para observar la mejoría de los resultados obtenidos por el modelo *fine-tuneado*.

Además del diccionario WordNet, también se han usado otros diccionarios diferentes para ver si este *fine-tuning* también mejora los resultados obtenidos en otros conjuntos de datos diferentes:

- Fichero de evaluación con 1000 definiciones extraídas del diccionario ldoce[5].
- Fichero de evaluación con 1000 definiciones extraídas del diccionario def.

Para el segundo prototipo (4.2), se ha usado el conjunto de datos que propuso [Hill et al., 2015] para el entrenamiento de su modelo. Este *dataset* está compuesto por definiciones extraídas de cinco diccionarios electrónicos: *Wordnet, The American Heritage Dictionary,*

[5] https://www.ldoceonline.com/es-LA/

The Collaborative International Dictionary of English, Wiktionary y *Webster's*. Estos diccionarios son de libro acceso a través de la API WordNik[6].

4.5. Experimentación y resultados

A continuación, se detallarán los diferentes experimentos realizados con cada prototipo para conseguir los mejores resultados posibles. En la tabla final 4.4 se reúnen los resultados finales de todos los prototipos propuestos junto con los de algunos sistemas del estado del arte actual.

Por cada prueba, se detalla la precisión@1, precisión@10 y precisión@100. La mediana y la varianza de cada prueba no se ha detallado ya que estas dependen únicamente de la precisión obtenida, por lo que solo se han plasmado en la tabla final de los resultados, donde se encuentran las versiones finales de los resultados de los prototipos.

4.5.1. Diccionario inverso mediante Mask Filling

Los resultados con los datos de prueba de 100 definiciones para los modelos base se pueden ver en la tabla 4.1. Como se puede ver y ya se ha mencionado anteriormente, el modelo que ha conseguido los mejores resultados ha sido el de RoBERTa Large: ha predicho 16 definiciones en precisión p@1. Por lo tanto, a partir de aquí se trabajará con este modelo. En estas pruebas, se aprecia que el modelo BART no funciona tan mal como se pensaba. Sin embargo, esto es porque en este caso se ha utilizado la funcionalidad de *mask-filling* del modelo y no la de creación de texto, que es la que haría falta para predecir palabras clave formadas por varias palabras.

El siguiente paso ha sido elegir los hiperparámetros para *fine-tunear* de manera óptima el modelo con el diccionario completo de WordNet. Para ello, se han realizado varias pruebas con el mismo corpus de prueba de 100 definiciones. La idea principal es probar diferentes *learning rate* para diferentes cantidades de *epoch*. Así, al *fine-tunear* el modelo con el diccionario completo, podremos elegir el *learning rate* adecuado dependiendo del tiempo que queremos que dure el *fine-tuning* (Dependiendo de los *epoch*). El *batch size* y el *gradient acummulation steps* han quedado fijados en 8 y 4, respectivamente. Los *learning rate* óptimos se pueden ver en la tabla 4.2, donde se expone el rendimiento en precisión p@1 y p@5.

[6] https://developer.wordnik.com/

Modelo	p@1	p@2	p@3	No acertadas
Roberta-base	11	4	6	76
Roberta-large	16	7	8	67
xlm-roberta-base	3	0	2	92
xlm-roberta-large	1	1	0	97
Bart-base	1	2	0	95
bart-large	1	1	0	89
albert-base-v2	3	1	2	94
albert-large-v2	5	3	1	90
distilbert-base-uncased	4	2	0	93
distilbert-base-cased	2	1	3	91
deberta-xlarge-mnli	0	0	0	100
deberta-base	0	0	0	100
flaubert-base-uncased	0	0	0	100
flaubert-base-cased	0	0	0	100
camembert-base	0	0	0	100

Tabla 4.1: Resultados de diferentes modelos en prueba de 100 definiciones

Epoch	lr (best top-1)	lr (best top-5)
3	5,00E-05	1,00E-04
5	1,00E-04	1,00E-04
10	1,00E-04	1,00E-04
15	2,50E-04	1,00E-04
20	1,00E-04	7,50E-05
30	1,00E-04	7,50E-05
50	7,50E-05	7,50E-05
100	5,00E-05	5,00E-05

Tabla 4.2: El mejor *learning rate* para diferentes *epoch*

Finalmente, es hora de *fine-tunear* el modelo con el diccionario completo (Excluyendo las definiciones "unseen" del *dataset*). En mi caso, teniendo en cuenta mi velocidad de cómputo y el límite de sesión establecido de Google Colaboratory de 12 horas, he hecho el *fine-tuning* con 10 *epoch*. Si miramos la tabla 4.2, sabemos que para esa cantidad de *epochs* $1,00E-04$ es el mejor *learning rate*. El *fine-tuning* se realiza mediante el método que ofrece el propio HuggingFace:

```
!python3 /content/drive/MyDrive/TFG/transformers/examples/language-modeling/run_mlm.py \
 --model_name_or_path="roberta-large" \
 --train_file=/content/drive/MyDrive/TFG/Finetunning/cleaned_data/wordnet_dictionary.txt \
 --do_train \
 --output_dir=finetuned_models/roberta-large/10epochs \
 --per_device_train_batch_size=8 \
```

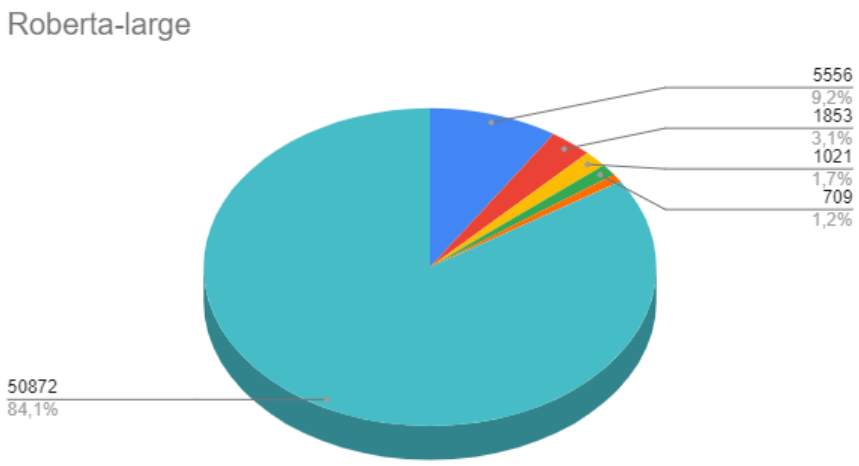

Figura 4.1: Resultados con modelo base RoBERTa Large

```
    --gradient_accumulation_steps=4 \
    --learning_rate=1e-4 \
    --num_train_epochs=10 \
    --save_steps=300000 \
    --overwrite_output_dir \
    --overwrite_cache \
    --line_by_line \
    --fp16
```

En la imagen 4.1 se ven los resultados obtenidos con el modelo base para las definiciones del corpus entero del diccionario WordNet, y en la imagen 4.2 los resultados obtenidos con el modelo tras haberlo *fine-tuneado*. En los gráficos se ve claramente que el proceso de *fine-tuning* ha mejorado significativamente las predicciones del modelo: el modelo base no consigue malos resultados (de 60.505 definiciones acierta 5.556, un 9,18 %, en el primer intento y 9.633 en los primeros cinco), y tras haber aplicado el *fine-tuning*, se han acertado 11.410 definiciones (18,86 %) en precisión p@1. Es decir, se han duplicado los aciertos.

También se ha probado este mismo modelo *fine-tuneado* para predecir definiciones de otros diccionarios que no sean WordNet. Para ello, se han usado 1000 definiciones escogidas aleatoriamente de los diccionarios ldoce y def, tal y como se ha explicado en el anterior apartado 4.4. Los resultados obtenidos se ven en la tabla 4.3. En este caso, al intentar predecir palabras clave que no se han visto durante el entrenamiento (de diccionarios que no sean de WordNet), no han mejorado los resultados, sino que han empeorado. Esto puede deberse a varias razones, pero la más probable es que haya habido *overfitting*

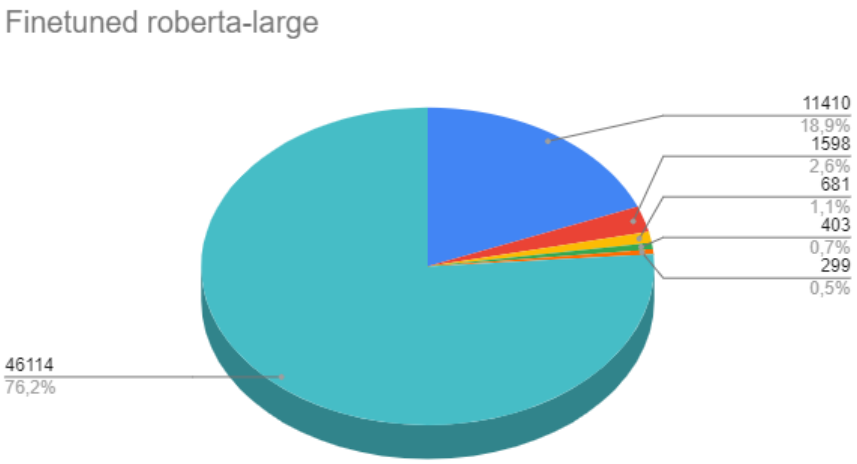

Figura 4.2: Resultados con modelo *finetuneado* RoBERTa Large

en el proceso de entrenamiento. Este problema se podría solventar ajustando los *epoch* y el *learning rate*, pero en esta manera de resolver la tarea sigue siendo muy complicado aprender nuevas definiciones de diccionarios que no se han visto nunca.

		LDOCE					
	Finetuned with	epochs	learning rate	p@1	p@2	p@3	Sin acertar
Roberta-large	Base	Base	Base	110	40	19	805
Roberta-large	WordNet	10	1,00E-04	26	15	5	940
		DEF					
	Finetuned with	epochs	learning rate	p@1	p@2	p@3	Sin acertar
Roberta-large	Base	Base	Base	265	53	17	642
Roberta-large	WordNet	10	1,00E-04	116	35	15	812

Tabla 4.3: Resultados en otros diccionarios con el modelo *finetuneado* con WordNet.

Finalmente, estos son los resultados obtenidos en el *dataset* de evaluación propuesto por [Hill et al., 2015], que recordemos que se tratan de definiciones del mismo diccionario WordNet. Se han conseguido las siguientes precisiones@1/10/100:

- En la prueba con las definiciones vistas (*Seen*): 0.06/0.15/0.22
- En la prueba con las definiciones no vistas (*Unseen*): 0.04/0.14/0.23
- En la prueba con las descripciones de palabras (*Description)*: 0.26/0.65/0.82

Como se puede observar, los resultados obtenidos en las primeras dos pruebas son bastante malos y no están a la altura de los prototipos del Estado del Arte ni de los otros dos

prototipos propuestos en esta memoria. Sin embargo, en la tercera prueba, en la de las descripciones de palabras, se han obtenido muy buenos resultados. De hecho, se acercan o incluso superan a algunos sistemas del Estado del Arte.

4.5.2. Diccionario inverso mediante Sentence-Transformers

Este prototipo no requiere de mucha experimentación para conseguir la mayor eficacia posible. Solamente se han probado los diferentes modelos preentrenados ofrecidos por Sentence-Transformers para realizar la conversión a *embeddings* de las definiciones y ver con cual se consiguen mejores resultados.

El primer modelo que se ha probado es *paraphrase-distilroberta-base-v1*, que es el que se usa de ejemplo en la documentación. Con este modelo, se han conseguido las siguientes precisiones@1/10/100:

- En la prueba con las definiciones vistas (*Seen*): 0.69/0.96/0.97
- En la prueba con las definiciones no vistas (*Unseen*): 0.01/0.01/0.01
- En la prueba con las descripciones de palabras (*Description)*: 0.18/0.46/0.76

El segundo modelo que se ha puesto a prueba es el modelo *paraphrase-mpnet-base-v2*. Este es el modelo con el que, según la documentación, se ha conseguido la mejor media de resultados en diferentes tareas. En el caso que nos ocupa, se han conseguido las siguientes precisiones@1/10/100:

- En la prueba con las definiciones vistas: 0.67/0.97/0.98
- En la prueba con las definiciones no vistas: 0.01/0.01/0.01
- En la prueba con las descripciones de palabras: 0.23/0.52/0.85

El tercer y último modelo probado es *paraphrase-MiniLM-L6-v2*. En la documentación se explica que este modelo es de alta rapidez, pero que mantiene una calidad alta en los resultados. Se han conseguido las siguientes precisiones@1/10/100:

- En la prueba con las definiciones vistas: 0.70/0.96/0.98

- En la prueba con las definiciones no vistas: 0.01/0.01/0.01
- En la prueba con las descripciones de palabras: 0.22/0.46/0.75

Lo primero que llama la atención es la baja eficacia del prototipo con las definiciones no vistas, independientemente del modelo usado para la conversión de los *embeddings*. Esto se debe a que el prototipo siempre predecirá una de las palabras guardadas en la estructura de datos donde se almacenan todas las definiciones. Si la palabra clave no está almacenada en esta estructura, nunca se devolverá esta palabra exacta. Sin embargo, el prototipo sí que devuelve una palabra clave de gran similitud con la que se está buscando.

En las otras dos pruebas, se consiguen resultados muy buenos, a la altura de otros sistemas del estado del arte. Los mejores resultados se han conseguido con el segundo modelo probado, *paraphrase-mpnet-base-v2*, llegando a conseguir un acierto de hasta el 98 % en precisión@100 en la prueba de definiciones vistas, y un acierto del 85 % en la prueba de las descripciones escritas por personas.

4.5.3. Diccionario inverso usando la API datamuse

En el caso de este prototipo dependemos de la API datamuse por lo que no se ha requerido ningún tipo de experimentación para mejorar los resultados.

Tal y como se ha mencionado antes, el sistema OneLook también funciona a partir de esta API, por lo que los resultados que se han obtenido son iguales a este sistema, o muy parecidos. Se han conseguido las siguientes precisiones/1/10/100:

- En la prueba con las definiciones vistas: 0.69/0.94/0.95. En [Zhang et al., 2020], se expone que los resultados obtenidos en el sistema OneLook son de 0.66/0.94/0.95. Esta ligera mejoría se debe de deber a algún cambio en la API que haya producido algún cambio en alguna predicción.
- En la prueba con las definiciones no vistas se obtiene un resultado de 0.64/0.94/0.94, pero no es un resultado válido, ya que en la API sí que se ha trabajado con este conjunto de palabras que en realidad no se tendrían que haber visto.
- En la prueba con las descripciones de palabras: 0.34/0.54/0.76

4.5.4. Comparación de los resultados con el estado del arte

En la tabla 4.4 se resumen todos los resultados obtenidos en los tres prototipos junto a los resultados de los sistemas del estado del arte. Entre los resultados obtenidos, se destacan los del prototipo de Sentence-Transformers en el *dataset* de definiciones vistas y los del prototipo de Mask-Filling en el *dataset* de las descripciones de definiciones. En ambos casos, se han superado los resultados obtenidos en casi todos los sistemas del estado del arte. En el caso de Sentence-Transformer, la eficacia de este modelo solo es superado por el sistema de MS-LSTM propuesto por [Kartsaklis et al., 2018], y supera a todos los demás, incluyendo diccionarios inversos muy famosos como OneLook. En el caso de Mask-Filling, es superado por los modelos de BERT y RoBERTa propuestos en [Yan et al., 2020] y por WantWords. Sin embargo, obtiene mejores resultados que OneLook (En las precisiones p@10 y p@100) y se asemejan mucho a los obtenidos en [Zhang et al., 2020].

Donde no se han conseguido buenos resultados ha sido en el *dataset* de definiciones no vistas. El modelo que mejores resultados ha conseguido ha sido el de Mask-Filling, pero no han superado a ningún sistema de la tabla. Los otros dos prototipos, los que se basan en Sentence-Transformers y la API datamuse, no sirven para esta prueba por los problemas mencionados anteriormente.

Model	Seen			Unseen			Description		
OneLook	0	.66/.94/.95	200	-	-	-	5.5	.33/.54/.76	332
WantWords	-	-	-	19	.10/.38/.72	-	2	.36/.75/.92	-
RDWECI	121	.06/.20/.44	420	170	.05/.19/.43	420	16	.14/.41/.74	306
SuperSense	378	.03/.15/.36	462	465	.02/.11/.31	454	115	.03/.15/.47	396
MS-LSTM	**0**	**.92/.98./.99**	65	276	.03/.14/.37	426	1000	.01/.04/.18	404
Mul-Channel	16	.20/.44/.71	310	54	.09/.29/.58	358	2	.32/.64/.88	203
BERT	**0**	.57/.86/.92	240	18	**.20/.46/.64**	418	1	.36/.77/.94	94
RoBERTa	**0**	.57/.84/.92	228	37	.10/.36/.60	405	**1**	**.43/.85/.96**	46
Mask Filling	3	.06/.15/.22	7061	**10**	.04/.14/.24	10382	3	.26/.65/.82	6417
Sentence-Transformers	**0**	.67/.97/.98	202	-	.01/.01/.01	-	7	.23/.52/.85	4951
API datamuse	0	.69/.94/.95	38	-	-	-	5.5	.34/.54/.76	332

Tabla 4.4: Resultados finales de los tres prototipos junto a los del estado del arte. En cada celda, los valores corresponden a "Mediana", "Precisión@1/10/100" y "Varianza". Los resultados que no son de los sistemas propios han sido extraídos de [Qi et al., 2020], [Zhang et al., 2020] y [Yan et al., 2020]

5. CAPÍTULO

Conclusiones

En este capítulo se concluye el proyecto, repasando los objetivos alcanzados y reflexionando sobre el desarrollo académico y personal que ha supuesto.

5.1. Sobre los objetivos alcanzados

A lo largo del desarrollo del proyecto, se han ido cumpliendo los objetivos descritos en el apartado 1.2. Se ha adquirido conocimiento bastante avanzado en el campo del Procesamiento del Lenguaje Natural dentro de la Inteligencia Artificial. Teniendo una fuerte conexión con este campo, se han investigado en profundidad los Transformers y se ha trabajado con ellos. También se ha conseguido entrenar uno de los prototipos creados mediante *fine-tuning*, y se han evaluado los tres sistemas en un entorno estandarizado con el mismo conjunto de datos.

Entre todos los objetivos establecidos, ha habido uno que no se ha podido cumplir: el de crear un diccionario inverso translingual. No se ha podido construir un diccionario inverso que predijese una palabra de un idioma diferente debido a el gran aumento de dificultad que suponía esta tarea.

5.2. Sobre el desarrollo académico y personal

Tras reflexionar sobre lo que ha supuesto este proyecto, diría que el principal hito ha sido el de realizar un proyecto tan avanzado por mi cuenta con la ayuda de un tutor, con todo lo que eso supone: he adquirido mucha experiencia sobre cómo gestionar mi tiempo y a medir el alcance adecuado de cada tarea. A llevar a cabo reuniones fructíferas con el tutor para que me guiase en mi trabajo, pero después teniendo que abordarlo por mi cuenta.

Ha sido satisfactorio haber sido capaz de desarrollar modelos que compitan con los mejores sistemas actuales, y más tratándose de un campo tan avanzado como es el Procesamiento del Lenguaje Natural.

El desarrollo de este Trabajo de Fin de Grado me ha motivado a seguir formándome en el campo de la Inteligencia Artificial, y a pesar de finalmente optar por el máster de Ingeniería Computacional y Sistemas Inteligentes, el tema me ha parecido muy interesante y me ha suscitado ganas de aprender más sobre él, ya no solo sobre el tema de diccionarios inversos, sino sobre el amplio abanico del Procesamiento del Lenguaje Natural. Los conceptos descritos en los apartados de teoría del comienzo de esta memoria son solo la base.

5.3. Trabajo futuro

En cuanto a posible trabajo futuro se refiere, hay varias funcionalidades que mejorarían considerablemente los diccionarios inversos:

- Principalmente, la primera funcionalidad que se me viene a la cabeza es el objetivo que se ha quedado sin cumplir: el de crear un diccionario inverso translingual, el cual realice predicciones en un idioma diferente al de la definición dada.
- Otro objetivo interesante sería el de crear un diccionario inverso multilingüe. Es decir, que sirva para varios idiomas, por ejemplo el euskera. Para ello, se requeriría explorar otros modelos diferentes de HuggingFace que soporte esos idiomas, ya que también existen versiones multilingües de BERT y RoBERTa.
- Otro punto que ha sido un poco decepcionante ha sido los resultados obtenidos en el *dataset* de definiciones no vistas. Como trabajo futuro, desarrollar un prototipo que obtenga buenos resultados en esa tarea sería la prioridad.

Además de añadir funcionalidades a estos diccionarios inversos o crear otros desde cero mediante otros sistemas, este proyecto también me ha dado ideas de otros trabajos diferentes dentro del Procesamiento del Lenguaje Natural. En la planificación inicial del proyecto, cuando todavía no tenía claro cómo iba a ser el proyecto exactamente, se barajaron otras proyectos diferentes y muy interesantes:

- Procesar el diccionario online de Oxford, Lexico[1]. Extraer todas las definiciones, traducirlas y proyectar su significado usando SimAlign[2] o Awesome[3], por ejemplo.
- Procesar términos médicos de Wikipedia para realizar clasificaciones de trastornos, realizar diagnósticos o asignar tratamientos.
- Comparar algoritmos neuronales para la desambiguación de significado de palabras (*Word Sense Disambiguation*, WSD).
- Traducción automática de las glosas de WordNet o de otros corpus.
- Traducción automática de diccionarios en castellano o euskera al inglés.

[1] https://www.lexico.com/
[2] https://github.com/cisnlp/simalign
[3] https://github.com/neulab/awesome-align

Bibliografía

[Bilac et al., 2004] Bilac, S., Watanabe, W., Hashimoto, T., Tokunaga, T., and Tanaka, H. (2004). Dictionary search based on the target word description.

[Brown and Mcneill, 1966] Brown, R. and Mcneill, D. (1966). The "tip of the tongue" phenomenon. *Journal of Verbal Learning and Verbal Behavior*, 5:325–337.

[Dao and Simpson, 2005] Dao, T. N. and Simpson, T. (2005). Measuring similarity between sentences. *The Code Project*.

[Devlin et al., 2019] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding.

[Gollins and Sanderson, 2001] Gollins, T. and Sanderson, M. (2001). Improving cross language retrieval with triangulated translation. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '01, page 90–95, New York, NY, USA. Association for Computing Machinery.

[Hadar et al., 1987] Hadar, U., Jones, C., and Mate-Kole, C. (1987). The disconnection in anomic aphasia between semantic and phonological lexicons. *Cortex*, 23(3):505–517.

[Hill et al., 2015] Hill, F., Cho, K., Korhonen, A., and Bengio, Y. (2015). Learning to understand phrases by embedding the dictionary. *Transactions of the Association for Computational Linguistics*, 4.

[Johnson et al., 2017] Johnson, J., Douze, M., and Jégou, H. (2017). Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*.

[Kartsaklis et al., 2018] Kartsaklis, D., Pilehvar, M. T., and Collier, N. (2018). Mapping text to knowledge graph entities using multi-sense lstms.

[Leech et al., 1994] Leech, G., Garside, R., and Bryant, M. (1994). Claws4: The tagging of the british national corpus. In *Proceedings of the 15th Conference on Computational Linguistics - Volume 1*, COLING '94, page 622–628, USA. Association for Computational Linguistics.

[Lewis et al., 2019] Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. (2019). BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461.

[Liu et al., 2010] Liu, B. et al. (2010). Sentiment analysis and subjectivity. *Handbook of natural language processing*, 2(2010):627–666.

[Liu et al., 2019] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach.

[Mikheev et al., 1999] Mikheev, A., Moens, M., and Grover, C. (1999). Named entity recognition without gazetteers. In *Ninth Conference of the European Chapter of the Association for Computational Linguistics*.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space.

[Mikolov et al., 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality.

[Miller, 1995] Miller, G. A. (1995). Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41.

[Morinaga and Yamaguchi, 2018] Morinaga, Y. and Yamaguchi, K. (2018). *Improvement of Reverse Dictionary by Tuning Word Vectors and Category Inference: 24th International Conference, ICIST 2018, Vilnius, Lithuania, October 4–6, 2018, Proceedings*, pages 533–545.

[Pilehvar, 2019] Pilehvar, M. T. (2019). On the importance of distinguishing word meaning representations: A case study on reverse dictionary mapping. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2151–2156, Minneapolis, Minnesota. Association for Computational Linguistics.

[Qi et al., 2019] Qi, F., Yang, C., Liu, Z., Dong, Q., Sun, M., and Dong, Z. (2019). Openhownet: An open sememe-based lexical knowledge base.

[Qi et al., 2020] Qi, F., Zhang, L., Yang, Y., Liu, Z., and Sun, M. (2020). Wantwords: An open-source online reverse dictionary system. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–181.

[Reimers and Gurevych, 2019] Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

[Reimers and Gurevych, 2020] Reimers, N. and Gurevych, I. (2020). Making monolingual sentence embeddings multilingual using knowledge distillation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

[Sennrich et al., 2016] Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

[Shaw et al., 2011] Shaw, R., Datta, A., Vander Meer, D., and Dutta, K. (2011). Building a scalable database-driven reverse dictionary. *Knowledge and Data Engineering, IEEE Transactions on*, 25:1–1.

[Shaw et al., 2013] Shaw, R., Datta, A., VanderMeer, D., and Dutta, K. (2013). Building a scalable database-driven reverse dictionary. *IEEE Transactions on Knowledge and Data Engineering*, 25(3):528–540.

[Virpioja et al., 2013] Virpioja, S., Smit, P., Grönroos, S.-A., and Kurimo, M. (2013). Morfessor 2.0: Python implementation and extensions for morfessor baseline. Workingpaper. VK: coin.

[Wu, 1995] Wu, Z. (1995). Verb semantics and lexical selection.

[Yan et al., 2020] Yan, H., Li, X., and Qiu, X. (2020). Bert for monolingual and cross-lingual reverse dictionary.

[Zhang et al., 2020] Zhang, L., Qi, F., Liu, Z., Wang, Y., Liu, Q., and Sun, M. (2020). Multi-channel reverse dictionary model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 312–319.