Desambiguación de acrónimos en literatura médica española

María Elena García García

Trabajo Fin de Máster en Sistemas Inteligentes De Diagnóstico, Planif. Y Control

Máster en Universitario en Inteligencia Artificial Avanzada

Dirigido por Prof. Dra. Raquel Martínez Unanue y Prof. Dra. Soto Montalvo Herranz

Escuela Técnica Superior de Ingeniería Informática. Departamento de Lenguajes y Sistemas Informáticos

Universidad Nacional de Educación a Distancia (UNED)

Madrid, España

Septiembre 2021

A mis tutoras que me han ayudado a crear este proyecto. A mis padres que siempre me inspiran, a Tete que me apoya siempre de forma incondicional, a mis chicas que se implican en cada experiencia como si fuese suya y a Pablo y Trias que han estado ahí en cada momento que lo he necesitado.

Resumen

La literatura biomédica esta repleta de abreviaciones y acrónimos, los cuales en muchas ocasiones son ambiguos. En las tareas de Procesamiento del Lenguaje Natural en los que este tipo de textos están involucrados, supone un gran problema por parte del sistema, para poder identificar y comprender tanto el documento como este tipo de palabras. En la última década se han desarrollado muchas investigaciones para poder desambiguar los acrónimos en literatura médica según el contexto del documento. Sin embargo, el reto siempre ha estado en el coste computacional que supone entrenar un modelo con textos de un ámbito concreto. Recientemente ha habido avances en este tema gracias a modelos lingüísticos basados en mecanismos de atención llamados *Transformers*, especialmente aquellos preentrenados ya con grandes corpus, como BERT (*Bidirectional Encoder Representations from Transformers*). Estos novedosos modelos han sido usados en los últimos tres años para la desambiguación de acrónimos en literatura médica, especialmente inglesa. En este trabajo se propone adaptarlos para poder realizarlo en literatura médica española.

Palabras clave: Acrónimo, Desambiguación, Biomedicina, Transformers, mecanismos de atención, BERT, BETO

Abstract

Biomedical literature is full of abbreviations and acronyms, which are often ambiguous. In Natural Language Processing tasks in which these types of texts are involved, it is a big problem for the system to identify and understand both the document and these types of words. In the last decade, a lot of research has been done to disambiguate acronyms in medical literature according to the context of the document. However, the challenge has always been the computational cost of training a model with texts from a specific field. Recently there have been advances in this subject thanks to linguistic models based on attention mechanisms called *Transformers*, especially those already pre-trained with large corpus, such as BERT (*Bidirectional Encoder Representations from Transformers*). These models have been used in the last three years for the disambiguation of acronyms in medical literature, especially in English. In this document we propose to adapt them to use in Spanish medical literature.

Keywords: Acronym, Disambiguation, Biomedical, Transformers, attention mechanisms, BERT, BETO

Índice

Lista de Tablas

Lista de Figuras

1. Introducción

1.1 Descripción del problema

La literatura médica está repleta de abreviaciones. Acortar palabras es una técnica muy utilizada para ahorrar tiempo al personal sanitario. Podemos agrupar estas abreviaciones en tres grandes grupos: abreviaturas, acrónimos y símbolos. Las abreviaturas suponen truncar las palabras eliminando algunas de las letras que contienen, por ejemplo, *Serv. coron.* para referirse a *Servicio coronario*. También pueden generarse por contracción, eliminando letras intermedias y conservando las correspondientes a la parte inicial y final de la palabra (por ejemplo *rdte.* para referirse a *residente*).

Los símbolos son abreviaciones que siguen unas normas y están estipuladas por organismos oficiales. Un ejemplo claro son las abreviaturas de medidas, como *L* por *litro* o los símbolos químicos de la tabla periódica (*Na* representa al *sodio*).

Las siglas son palabras formadas por cada letra inicial de una expresión, y los acrónimos son un subconjunto de estas, ya que no necesariamente tienen que contener únicamente las primeras letras de cada palabra.

El uso de estas abreviaciones lleva en primer lugar a un problema para el personal sanitario, lo cual puede llevar a errores médicos. En (Aleixandre-Benavent et al., 2015) se exponen varios problemas médicos derivados del recorte del lenguaje. Afirman que según el *Institute for Safe Medication Practice* (ISMP) (2005), pueden atribuirse más de 7.000 muertes al año debidas a errores en la administración de los medicamentos a los pacientes, y el uso de abreviaciones contribuye significativamente a esta estadística.

Uno de los problemas del uso de abreviaturas o acrónimos, no es solo su entendimiento, sino su correcta interpretación. Este tipo de palabras pueden tener diversos significados, por ejemplo *PCR* puede referirse a *Parada Cardiorespiratoria*, *Proteína C Reactiva* o *polymerase chain reaction*, la famosa prueba para la detección de coronavirus. Otros ejemplos pueden ser *HTA* que puede referirse a *hipertensión arterial* o *histerectomía total abdominal* o *ACO*: *anticoagulantes orales* o *anticonceptivos orales*. (Liu et al., 2001) en un estudio hecho sobre UMLS (*Unified Medical Language System*), un repositorio de vocabulario biomédico desarrollado por la Biblioteca Nacional de Medicina de los EEUU, encontraron que el 33.1% de las abreviaturas eran ambiguas, donde la media de posibles significados para aquellas con menos de 7 caracteres, era de 2.28.

1.2 Motivación

La Inteligencia Artificial (IA) cada vez está presente en más aspectos de nuestra vida cotidiana: clasificación de correos electrónicos, identificación de objetos en imágenes, recomendaciones basadas en nuestros gustos, etc. Aunque todas las áreas en las que la IA es aplicada tienen bases comunes, dependiendo del tipo de problema a resolver nos encontraremos en un campo concreto de la misma: visión artificial, sistemas de recomendación, aprendizaje automático, etc.

El campo de la Inteligencia Artificial encargado de analizar y trabajar con el lenguaje (ya sea escrito o por voz) es el Procesamiento del Lenguaje Natural (PLN o NLP por sus siglas en inglés *Natural Language Processing*). Las diferentes aplicaciones del PLN son muy diversas: traducción automática, sistemas Q&A (*Question-Answering*), análisis de sentimiento, *chatbots*, etc. Lo que distingue el procesamiento del lenguaje de otras aplicaciones de procesado de datos es el uso del conocimiento del lenguaje. Por ejemplo, si pedimos a un sistema que cuente la cantidad de palabras en un texto, primero deberá entender qué es una palabra. Por ello, estos sistemas deben de tener conocimientos del lenguaje: fonética y fonología, morfología, sintaxis, semántica, pragmática y discurso (Jurafsky and Martin, 2009).

Entre otras, una de las dificultades de los sistemas de procesamiento del lenguaje es la ambigüedad del mismo. Podemos distinguir diferentes tipos de ambigüedades:

- Léxica: de significado. La palabra banco puede tener varias acepciones, banco de peces, banco para sentarse o banco donde sacar dinero.

- Sintáctica. En la frase "María observó a Jaime con unos prismáticos" podría ser María quien tuviese los prismáticos y con ellos viese a Jaime, o podría ser Jaime quien los tuviese.
- De referencia. En la frase "Ella le dijo que lo comprara después" podrían surgir las siguientes preguntas: ¿A quién se lo dijo? ¿Que comprase el qué? ¿Después de qué?

En el caso a resolver en este trabajo nos centraremos en la ambigüedad léxica, es decir, aquella que presentan las palabras polisémicas, acrónimos concretamente en este trabajo.

Los textos biomédicos son susceptibles de tener mayor cantidad de acrónimos que otros textos de cualquier otra índole. Esto es debido a que además de acortar procedimientos por optimización de tiempo por parte del personal médico, como ya comentábamos, incluyen gran cantidad de nombres de fármacos y químicos. En ocasiones, la descripción del acrónimo se incluye dentro del propio texto, pero en otras ocasiones no. Podemos comprender lo peligroso que resulta que existan elementos ambiguos en un texto médico, que puede contener la medicación pautada a un paciente, las pruebas a realizar o la enfermedad que padece. Además, estas ambigüedades no sólo pueden suponer un problema para alguien sin conocimientos técnicos del tema, sino para un médico de otra especialidad que lea el informe o el texto, quien puede no conocer toda la jerga y abreviaturas utilizadas en otra área médica. Como podemos imaginar, si esta casuística pone en riesgo la correcta interpretación por parte de un experto, imaginemos la dificultad en un sistema dentro del Procesamiento del Lenguaje Natural (PLN). Una solución clásica sería

que un experto de cada materia, se encargase de "traducir" cada acrónimo en su forma larga adecuada. Evidentemente, esto supone una gran inversión de tiempo y de mano de obra. Además del inconveniente de que habría que realizar esta "traducción" de cada acrónimo en cada texto nuevo. También pueden desarrollarse diccionarios que contengan cada acrónimo y su forma larga, sin embargo, siempre necesitaríamos la ayuda de un experto que sepa qué significado es el adecuado según el contexto.

Todo esto pone de manifiesto la necesidad de un sistema de PLN que automatice este proceso y sea capaz de hacerlo sobre cualquier texto nuevo médico y sobre cualquier acrónimo. En la literatura inglesa, estos sistemas han sido investigados y desarrollados con buenos resultados, sin embargo, en la literatura médica española, la investigación en este ámbito es mucho más escasa.

1.3 Hipótesis y Objetivos

A lo largo de los últimos 20 años se han ido desarrollando métodos que permiten desambiguar acrónimos. Sin embargo, la mayoría de las investigaciones y sistemas creados se basan en literatura inglesa.

Gracias a iniciativas como IberEval [1], se ha promovido el desarrollo de Tecnologías del Lenguaje para la identificación de acrónimos (IberEval 2017 [2]) y la identificación y desambiguación (IberEval 2018 [3]) en literatura médica

[1] *Evaluation of Human Language Technologies for Iberian Languages (Spanish, Portuguese, Catalan, Basque and Galician)*

[2] http://nlp.uned.es/IberEval-2017/index.php

[3] https://temu.bsc.es/BARR2/

española.

El desarrollo de aplicaciones de PLN basándose en aprendizaje profundo o *deep learning*, especialmente las contribuciones de Google AI desde los vectores de palabras (Le and Mikolov, 2014a) hasta los *Transformers* (Vaswani et al., 2017) y BERT (Devlin et al., 2018) han supuesto una revolución para los sistemas de PLN. La motivación principal de esta investigación es "usar" estas nuevas técnicas que han dado muy buenos resultados en literatura inglesa para la desambiguación de acrónimos en literatura médica española y analizar los resultados obtenidos para este idioma.

Tras analizar el estado del arte de la identificación y desambiguación de acrónimos, especialmente en literatura médica, haciendo un repaso desde las técnicas más clásicas hasta las más novedosas basadas en aprendizaje profundo, se abordan en este trabajo las siguientes hipótesis:

- El contexto en el que se encuentra el acrónimo es decisivo para poder desambiguarlo. De modo que el éxito de este sistema estará muy condicionado por la definición que se haga del contexto.
- La comprensión del lenguaje y del idioma por parte del sistema utilizado es crucial para diferenciar contextos.
- El entrenamiento de un sistema para que sea capaz de entender el lenguaje requiere un corpus muy grande y mucho tiempo de computación. Por ello, el uso de sistemas ya preentrenados a los que se les añade una capa de ajuste supone una solución a dicho problema.

- El sistema puede dar lugar a errores para aquellos significados menos frecuentes y por tanto puede ser interesante balancear los datos.

Tras plantear estas hipótesis se definen los siguientes **objetivos**:

- Aplicación de técnicas de *Transformers* a la desambiguación de acrónimos en textos del dominio biomédico en español.
- Estudiar diferentes configuraciones de arquitecturas tipo *Transformers* para encontrar aquella que da mejores resultados en nuestro problema.
- Crear y/o adaptar los recursos lingüísticos necesarios para la desambiguación.
- Evaluar el sistema con una colección abierta para poder compararlo con investigaciones anteriores.

1.4 Propuesta

El problema de la ambigüedad está muy presente en los sistemas que resuelven tareas de Procesamiento del Lenguaje Natural. En la literatura médica se abusa además del uso de acrónimos para acortar definiciones, los cuales dan lugar a muchas ambigüedades. Este problema de la desambiguación de acrónimos ha sido estudiado y tratado durante mucho tiempo. Algunas técnicas han sido exitosas pero con los inconvenientes de ser rígidas o necesitar mucho trabajo manual, o supervisión de un experto.

Uno de los factores más importantes para poder discernir que tipo de significado le corresponde al acrónimo en cada caso es el contexto en el que se encuentra. En función de la temática del texto, por ejemplo cardíaco u oncológico, las siglas del

acrónimo podrán referirse a una u otra definición. Por ello, se han desarrollado muchas investigaciones que han intentado recoger el contexto adecuado por acrónimo. En los últimos años se utilizaron *word embeddings* o vectores de palabras para poder representar estos contextos y buscar sus equivalentes mediante similitud de espacios vectoriales. También se han usado en los últimos años redes neuronales para resolver el problema. Pero en todos los casos el objetivo ha sido el mismo: intentar "comprender" el texto en el que se encuentra el acrónimo para poder asignarle la definición correcta. Muy recientemente se han desarrollado nuevas técnicas que han supuesto una revolución en las tareas de PLN. Estas se basan en la arquitectura de tipo *Transformer*, que se explicará en la sección 2.9 de esta memoria. Basándose en este tipo de arquitecturas, la gran revolución ha sido BERT, donde se entrenan estos sistemas de forma bidireccional, lo que les permite generar una comprensión del lenguaje mucho más profunda. Tal y como habíamos dicho, la clave para saber qué significado asignar a un acrónimo es entender su contexto, y por tanto, estos sistemas mejoran mucho la obtención y comprensión del mismo. Además, otras de las grandes ventajas de estos sistemas es que ya están preentrenados con grandes corpus del idioma, por lo que nos ahorramos la necesidad de recolectar grandes cantidades de textos y las horas de computación necesaria. Estos modelos permiten añadir una capa después del modelo preentrenado que realice la tarea que necesitamos. A este sistema se le llama ajuste fino y se detallará en la sección 2.2.7.

El siguiente trabajo propone y estudia el uso de BETO (sistema tipo BERT pero entrenado para el español) con ajuste fino en el que se añade una capa

de clasificación. De este modo, dado un acrónimo, un contexto y una definición del significado del acrónimo, el sistema deberá ser capaz de determinar si esa definición del acrónimo es la que le corresponde según ese contexto o no.

1.5 Estructura del trabajo

Este trabajo se encuentra dividido de la siguiente forma:

- **Capítulo 1:** En este capítulo se ha detallado la motivación que ha llevado al desarrollo de este trabajo así como la contextualización del problema a resolver y su objetivo. Por último, se ha resumido la propuesta a implementar.
- **Capítulo 2:** Se hace un revisión de los trabajos del estado del arte centrados en la identificación y desambiguación de acrónimos en la literatura médica, principalmente inglesa. Además, se explican los fundamentos de las técnicas más comunes utilizadas en dichos trabajos. Se compara el uso de las mismas en la literatura inglesa (campo más maduro) frente al uso en la literatura española, de forma que se identifiquen aquellas con más éxito en los trabajos desarrollados en inglés para poderlos aplicar en los textos en castellano.
- **Capítulo 3:** Se presentan los detalles del método propuesto, desde el conjunto de datos hasta la arquitectura definida para la experimentación.
- **Capítulo 4:** Se detalla la implementación del método propuesto en el capítulo anterior: desde las herramientas necesarias, el preprocesamiento de los datos hasta el modelado.
- **Capítulo 5:** Se analizan los distintos experimentos realizados sobre los métodos

propuestos con el objetivo de presentar la configuración que mejores resultados presente.

- **Capítulo 6:** Por último, en este capítulo se recopilan las conclusiones extraídas del trabajo y se proponen las posibles líneas futuras de trabajo.

2. Estado del arte

En el objetivo de este trabajo nos centraremos en la correcta desambiguación de acrónimos en la literatura médica. La desambiguación de acrónimos puede considerarse un caso particular de WSD (*Word Sense Disambiguation)*.
Esta tarea se divide en dos: en primer lugar, la correcta extracción de los acrónimos del texto (en inglés *short-form* o *SF*) y, en segundo lugar, su desambiguación, es decir, encontrar la definición (o *long-form*, *LF*) correcta según el contexto en el que se encuentre el acrónimo. Del éxito de la primera dependerá también el éxito de la segunda.

2.1 Extracción de acrónimos

La **tarea de extracción** podemos agruparla en tres bloques según el enfoque escogido: detección de acrónimos mediante expresiones regulares, utilizando aprendizaje supervisado, o búsqueda heurística. Además de la detección del acrónimo, esta tarea incluye la detección de su descripción si se encuentra en el mismo texto, o fuera del mismo.

2.1.1 Técnicas basadas en expresiones regulares

Las expresiones regulares (ER) consisten en patrones definidos manualmente y su posterior búsqueda en un conjunto de datos, en nuestro caso, en las palabras de un texto.

El uso de las mismas ha sido muy común para describir patrones que definen a los acrónimos: por ejemplo, detectar palabras o *tokens* cuyas letras sean todas mayúsculas (Cuadros et al., 2018), que tengan una cantidad determinada de mayúsculas (Charbonnier and Wartena, 2018), o que sean más estrictas, como una combinación específica de mayúsculas, minúsculas, números y signos de puntuación, como en (Larkey et al., 2000), (Montalvo et al., 2017), (Sanchez and Martínez, 2017), (Sánchez-León, 2018) y más recientemente en (Ciosici et al., 2019). Otro patrón muy usado en la búsqueda de acrónimos ha sido aquel en el que el acrónimo o *short form (SF)* se encuentra entre paréntesis: (Zhang et al., 2011), (Montalvo et al., 2018) e incluso (Kashyap et al., 2020) más recientemente. Esta búsqueda de construcciones entre paréntesis da muy buenos resultados, siempre y cuando no haya acrónimos en el texto fuera de esta estructura. Al igual que con las ER que definen los patrones más comunes, este método carece de flexibilidad: aquellos acrónimos con un patrón distinto o que no se encuentren entre paréntesis, no serán detectados. Por ello, este método sólo resulta útil si se quiere detectar un conjunto muy específico de *SF*, como en (Pakhomov et al., 2005) donde sólo se buscan 8 acrónimos concretos o como método complementario de otras alternativas

para la búsqueda de acrónimos ((Montalvo et al., 2017), (Sanchez and Martínez, 2017), (Cuadros et al., 2018)).

2.1.2 Técnicas basadas en aprendizaje supervisado

En analítica avanzada podemos diferenciar dos subconjuntos de modelos analíticos: los basados en el aprendizaje supervisado y los basados en el aprendizaje no supervisado.

En los primeros, el modelo es entrenado con un conjunto de datos que contiene la variable a predecir o *target*. Cuando esta variable es de tipo continua hablamos de modelos de regresión y cuando la variable es discreta hablamos de modelos de clasificación. En nuestro caso nos centraremos en los modelos de clasificación.

En la fase de entrenamiento, el modelo encuentra los patrones comunes que definen a cada valor de la variable objetivo. Una vez finalizada esta etapa, es decir, una vez que el modelo ha aprendido a diferenciar las clases de la *target*, se le proporciona un nuevo conjunto de datos donde la variable objetivo no estará cumplimentada, y el objetivo del modelo será predecir su valor. En la casuística de este trabajo, el modelo supervisado debería clasificar si un *token* del texto es un acrónimo o no lo es.

Los modelos no supervisados no predicen ningún tipo de etiqueta y, por tanto, no tendremos una fase de aprendizaje. Estos modelos detectan patrones comunes en los datos y los segmentan según los mismos.

Entre los modelos supervisados más comunes utilizados en este ámbito

encontramos los siguientes:

- **Naive Bayes:** Se basa en la premisa de que elige el tipo de clase más probable. Este algoritmo asume que las características son independientes entre sí, de ahí su nombre *naive*, ingenuo. De este modo, las probabilidades de cada clase serían independientes, lo que supone que la probabilidad total sería la multiplicación de todas ellas. Entrenar un clasificador de Naive Bayes consiste en estimar cada una de estas probabilidades. Aunque en las propuestas leídas no lo utilizan en identificación de acrónimos, en (Cuadros et al., 2018) lo utilizan, entre otros algoritmos, para desambiguarlos, tarea que se definirá más adelante.
- **Árboles de decisión:** Es un método supervisado basado en construcciones lógicas, similar a los modelos basados en reglas. En él cada hoja o nodo representa un atributo o característica y la rama que sale de ella una regla de decisión. Los dos nodos hijos de ambas ramas son el resultado para cada decisión. (Pakhomov et al., 2005) y (Pomares-Quimbaya et al., 2020) utilizan Árboles de Decisión para la desambiguación de acrónimos, que se detalla más adelante. Dentro de este tipo de algoritmos encontramos también el famoso algoritmo *Ramdom Forest*, que es un tipo de algoritmo de *bagging* basado en árboles. Consiste en la generación simultánea de varios árboles de decisión sobre diferentes subconjuntos de datos, para generar después una predicción como promedio de todos los modelos. Este último es utilizado por (Ronzano and Furlong, 2017) para la identificación de acrónimos y por (Cuadros et al., 2018) de nuevo para desambiguarlos, como se verá más

adelante.

- **SVM:** Las Máquinas de Vectores de Soporte o *Support Vector Machine* buscan el hiperplano frontera en el espacio n-dimensional (*n* se corresponde con el número de variables que tengamos) en el que se encuentran los datos. Este hiperplano separa los datos pertenecientes a una clase frente a los de otra. (Nadeau and Turney, 2005) y (Zhang et al., 2011) obtuvieron buenos resultados en la identificación de acrónimos con este algoritmo. Veremos como también (Finley et al., 2016) y (Wu et al., 2015) lo utilizan para desambiguarlos.
- **Regresión Logística:** Utiliza la función sigmoide para clasificar la variable dependiente en la clase 0 o la clase 1. La salida de este tipo de modelo es la probabilidad de que la clase sea positiva (es decir, valor 1). Más adelante se detalla cómo (Finley et al., 2016) lo utilizan también para la desambiguación de acrónimos.

Además de los modelos supervisados descritos anteriormente, las **redes neuronales** también pueden funcionar como un clasificador. Aunque los tipos y funcionamiento de las mismas se detallan en la sección 2.2.2, enumeraremos en este punto algunos de los trabajos que las utilizan para la identificación de acrónimos. (Unanue et al., 2017) utilizan una red neuronal recurrente con *character-embeddings* para reconocer entidades en textos médicos. (Agerri and Rigau, 2017) utilizan un reconocedor de entidades basado en un modelo de red neuronal básico, el perceptrón, y otro basado en una red neuronal recurrente LSTM (*Long Short-Term Memory*) para la identificación de los acrónimos y su expansión.

(Menaha and Jayanthi, 2020) , tras una búsqueda heurística de la posible expansión del acrónimo, utilizan un perceptrón para la clasificación de una *LF* como posible o no posible expansión del acrónimo.

Para poder trabajar con algoritmos supervisados como los descritos anteriormente, debemos primeramente realizar un preprocesado del texto. Las *features* o características relevantes respecto al tipo de *token* se extraen y almacenan en un vector de características, el cual codifica esta información de manera numérica o nominal. Este vector se utilizará como input del modelo supervisado.

En el caso de los acrónimos los tipos de *features* usadas son diversas, por ejemplo la densidad de letras en mayúscula, en minúscula, si contienen dígitos, longitud de la palabra, categoría gramatical de la misma (*POS tagging*), características de la forma de la palabra (Dai et al., 2015), incluso la frecuencia del *token* en bases de conocimiento como la Wikipedia (Ronzano and Furlong, 2017) o si la palabra termina de forma no común (Cuadros et al., 2018), entre otras. Los problemas de estas técnicas son básicamente dos: que necesitamos un conjunto previamente etiquetado con las palabras que son acrónimos y las que no, y la rigidez, ya que si existe algún tipo nuevo de acrónimo con un patrón que no se encuentra en el conjunto de entrenamiento, no será capaz de detectarlo como acrónimo y lo etiquetará como palabra común.

2.1.3 Técnicas basadas en búsqueda heurística

El tercer método se basa en generar reglas que permitan, o bien disminuir el espacio de búsqueda como (Nadeau and Turney, 2005), para luego aplicar otra técnica, en su caso un modelo supervisado, o como único camino para la extracción de acrónimos, lo cual ha sido utilizado en los trabajos más recientes como (Nithyashri et al., 2020) o (Menaha and Jayanthi, 2020). Esto consiste en añadir información, basándose en el espacio estudiado, de forma que se restrinja este espacio de búsqueda. Podemos definirlo como reglas para buscar las ramas del espacio de estados que son más probables de llegar a una solución aceptable. Las heurísticas se aplican en dos casos:

1. Un problema que puede no tener una solución exacta, como un diagnóstico médico.
2. Un problema con solución exacta, pero en el que el coste computacional para encontrarla puede ser demasiado alto.

En nuestro caso nos encontraríamos con la segunda opción. Podríamos entrenar un modelo supervisado que fuese capaz de detectar todos los acrónimos posibles, pero el coste de conseguir un corpus con todos los acrónimos posibles y etiquetarlos como tal sería muy costoso. Además de que necesitaría estar actualizado a nuevos acrónimos que puedan surgir en el lenguaje, lo que además supondría un reentrenamiento del modelo. Las reglas heurísticas nos permiten buscar estos acrónimos limitando el espacio de búsqueda, por ejemplo, limitándonos a palabras que contengan alguna mayúscula (característica que comparten la mayoría de los acrónimos). Así por ejemplo, las palabras solo con minúsculas son descartadas,

disminuyendo el espacio de palabras de posibilidades. La ventaja de la búsqueda heurística o por reglas es que es un método más laxo que los anteriores, y permite captar acrónimos sin un patrón de búsqueda tan estricto. La desventaja es que puede fallar más que las otras dos técnicas y detectar *SF* que en realidad no lo son.

2.1.4 Detección de la expansión del acrónimo

Detectado el acrónimo, debemos **extraer su expansión**, que puede encontrarse o no en el propio texto. (Schwartz and Hearst, 2003) afirmaban que la mayoría de los pares ocurren bajo el patrón de la expansión (también llamada *long form, LF*) seguida del acrónimo o *short form (SF)* entre paréntesis o al revés. Para ello, desarrollaron un algoritmo de extracción basado en ese patrón, donde busca que los caracteres de la *SF* coincidan con las letras de la *LF*. (Montalvo et al., 2017) introdujeron una extensión al algoritmo para que las palabras de la *LF* no tengan que aparecer necesariamente en el mismo orden que las letras en la *SF*. (Intxaurrondo and Krallinger, 2017) utilizan varios algoritmos desarrollados para literatura inglesa que buscan también este tipo de patrones en literatura española, con buenos resultados.

Siguiendo esta idea, otras técnicas menos rígidas buscan ventanas de N-gramas previas al acrónimo, y extraen la definición del mismo aplicando una serie de reglas: (Thakker et al., 2017), (Kashyap et al., 2020), (Menaha and Jayanthi, 2020). Estos utilizan la misma frase en la que se encuentra el acrónimo para

aplicar el conjunto de N-gramas. Sin embargo, (Nithyashri et al., 2020) afirman que puede darse el caso en el que la expansión no se encuentre en la misma frase y la búsqueda deba ampliarse a todo el texto. En la misma publicación además de buscar la expansión en un conjunto de N-gramas previos al acrónimo, se desarrolla un sistema en el que asigna una etiqueta numérica a cada palabra en función de si empieza por una de las letras de la *SF*, no empieza o alguno de los caracteres de la palabra coincide con uno de los caracteres del acrónimo. Con ello cada N-grama forma un vector de valores que luego es introducido a un clasificador. (Du et al., 2019) definen un modelo automático basado en reglas, que detecta la abreviatura y su expansión: dividen la abreviatura en bloques, las expansiones candidatas se generan restaurando los bloques (palabras) y por último, estas son filtradas y clusterizadas.

Otra técnica diferente es la aplicada por (Dale, 2018) en la que intenta reconstruir las abreviaturas con un modelo probabilístico, que devuelve la letra más probable siguiente a una de las letras de la abreviatura.

También puede ser útil el uso de diccionarios, en los que buscar la forma corta y extraer la definición, tal y como hacen con los diccionarios de FreelingMed en (Montalvo et al., 2017), o (Sánchez and Martínez, 2018) con el *Diccionario de Siglas Médicas*. En publicaciones más recientes, se ha buscado la expansión en la conocida web *acronymfinder.com* ((Premkumar et al., 2020), (Nithyashri et al., 2020)). La desventaja de estos métodos es el basto trabajo que requiere la construcción de este tipo de diccionarios y la rigidez, ya que sólo encontraremos

las *SF* que estén contenidas en el diccionario. De modo que esta técnica resulta útil si se utiliza de forma complementaria a otras.

A continuación, se recoge de forma esquemática en la Tabla 2.1 el conjunto principal de publicaciones estudiadas en la tarea de identificación de acrónimos y su expansión ordenadas cronológicamente:

Autores	**Título**	**Año**	**Detección SF**	**Detección par LF-SF**	**Idioma**
Larkey et al.	Acrophile: An Automated Acronym Extractor and Server	2000	Expresiones Regulares	Búsqueda de patrones	Inglés
Schwartz and Hearst	A simple Algorithm for identifying abbreviation definitions in biomedical test	2003	Algoritmo búsqueda patrones [patrón LF (SF), (SF) LF]	Algoritmo búsqueda patrones [patrón LF (SF), (SF) LF]	Inglés
Nadeau and Turney	A supervised Learning Approach to Acronym Identification	2005	Búsqueda heurística y de patrones	Búsqueda heurística y de patrones. Modelo supervisado.	Inglés
Zhang et al.	Entity Linking with Effective Acronym Expansion,Instance Selection and Topic Modeling	2011	Búsqueda de patrones SF (string)	-	Inglés
Dai et al.	Enhancing of chemical compound and drug name recognition using representative tag scheme and fine-grained tokenization	2015	Modelo supervisado	-	Inglés
Montalvo et al.	Biomedical Abbreviation Recognition and Resolution by PROSA-MED	2017	Búsqueda heurística de patrones	Búsqueda heurística ,Expresiones regulares Algoritmo Schwartz and Hearst	Español
Ronzano and Furlong	IBI-UPF at BARR-2017: learning to identify abbreviations in biomedical literature	2017	Modelo supervisado	Modelo supervisado de clasificación Búsqueda heurística	Español

Autores	Título	Año	Deteccion SF	Detección par LF-SF	Idioma
Sánchez and Martínez	A proposed system to identify and extract abbreviation definitions in Spanish biomedical texts for the Biomedical Abbreviation Recognition and Resolution	2017	Búsqueda heurística de patrones	Algoritmo búsqueda patrones	Español
Unanue et al.	Recurrent neural networks with specialized word embeddings for health domain named entity recognition	2017	Character-embeddings Redes Neuronales Recurrentes	-	Inglés
Aguerri and Rigau	Applying existing Named Entity taggers at BARR IBEREVAL 2017 task	2017	Reconocedor entidades basado en redes neuronales	-	Español
Intxarruondo and Krallinger	CNIO al BARR IberEval 2017: exploring three biomedical abbreviation identifiers for Spanish biomedical publications	2017	Algoritmo de búsqueda de patrones	Algoritmo de búsqueda de patrones	Español
Thakker et al.	Acronym Disambiguation: A Domain Independent Approach	2017	Es un input al modelo	Búsqueda basada en reglas	Inglés
Cuadros et al.	Vicomtech at BARR2: Detecting Biomedical Abbreviations with MLmethods and dictionary-based heuristics	2018	Modelo supervisado, Expresiones regulares Diccionarios	Ventana de 8 n-gramas antes de la LF. Diccionario. Búsqueda heurística.	Español
Montalvo et al.	MAMTRA-MED at Biomedical Abbreviation Recognition and Resolution IberEval 2018	2018	Búsqueda de patrones	Algoritmo de Schwartz and Hearst con modificación	Español
Sánchez León	ARBOREx: Abbreviation resolution based on regular expressions for BARR2	2018	Expresiones Regulares	-	Español
Castaño et al.	A simple approach to Abbreviation Resolution at BARR2, IberEval 2018	2018	Reconocedor de entidades (cTAKES)	-	Español

Autores	**Título**	**Año**	**Deteccion SF**	**Detección par LF-SF**	**Idioma**
Sánchez and Martínez	A simple method to extract abbreviations within a document usaing regular expressions	2018	Expresiones Regulares	Diccionario	Español
Dale	A machine learning model to understand fancy abbreviations, trained on Tolkien	2018	-	Modelo probabilístico	Inglés
Charbonnier and Wartena	Using Word Embeddings for Unsupervised Acronym Disambiguation	2018	Búsqueda de patrones (palabras en mayúsculas con al menos 3 letras)	Expresiones regulares	Inglés
Li et al.	Guess Me if You Can: Acronym Disambiguation for Enterprises	2018	Es un input al modelo	Búsqueda con regla heurísticas y patrones. Normalización las LF	Inglés
Ciosci et al.	Unsupervised Abbreviation Disambiguation	2019	Expresiones regulares	Algoritmo de Schwartz-Heartz Normalización de las LF	Inglés
Premkumar et al.	Acronym Disambiguation using Web Scraping	2020	Búsqueda de patrones (letras en máyusuclas)	Búsqueda en el repositorio acronymfinder.com	Inglés
Kashyapa et al.	The CLASSE GATOR (CLinical Acronym SenSE disambiGuATOR): A Method for predicting acronym sense from neonatal clinical notes	2020	Expresiones Regulares	Búsqueda basada en reglas en los n-gramas precedentes al acrónimo	Inglés
Nithyashri et al.	Web-Based Acronym Expansion Disambiguation Using Word2Vec and Doc2Vec Modeling	2020	Búsqueda de patrones con reglas	Búsqueda en el repositorio acronymfinder.com	Inglés
Pomares-Quinbaya et al.	Leveraging PubMed to Create a Specialty-Based Sense Inventory for Spanish	2020	Algoritmo de Schwartz-Heartz	Algoritmo de Schwartz-Heartz Normalización de las LF	Español

Autores	Título	Año	Deteccion SF	Detección par LF-SF	Idioma
Menaha and Jayanthi	A Hybrid Model for Finding Abbreviation-Definition Pairs form Biomedical Abstracts using Heuristics-based Sequence Labeling and Perceptron Linear Classifier	2020	Búsqueda heurística de patrones y posterior aplicación de reglas	Búsqueda heurística en la ventana de n-gramas precedente al acrónimo y redes neuronales	Inglés

Tabla 2.1: Resumen de publicaciones identificación de acrónimos y expansión

2.2 Desambiguación de acrónimos

2.2.1 Técnicas basadas en forma y aprendizaje supervisado

Dentro de la tarea de desambiguación podemos diferenciar dos grandes grupos: los basados en la forma y los basados en contexto. En los primeros se busca la forma correcta del acrónimo comparando las letras iniciales de la forma larga con las letras del acrónimo. La limitación de esta técnica es que no se podrá desambiguar un acrónimo con dos formas largas cuyas letras coincidan. Los métodos basados en contexto asignan al acrónimo aquella expansión cuyo contexto del texto sea más similar. El contexto puede ser definido de diferentes formas: palabras dentro de una ventana que rodean a la *SF* y su expansión, las palabras que componen el documento íntegro, todas las palabras pertenecientes a la frase en la que aparece el par acrónimo y su definición, etc. Una vez obtenido el contexto, la forma de desambiguar puede ser diversa.

Una de las técnicas, al igual que se usaba en la identificación de acrónimos, son los **modelos supervisados**. En este caso los contextos se desglosan en características que suponen la entrada al modelo, como por ejemplo, longitud del acrónimo, número de letras mayúsculas en el conjunto de palabras previas al acrónimo o la categoría gramatical de las palabras que rodean al acrónimo. Estas características sirven de variables de entrada a diferentes modelos, como los explicados en la sección 2.1.2 los cuales "aprenderán" patrones para determinar si la definición asociada a una *SF* es la correcta.

2.2.2 Métodos basados en aprendizaje profundo o *Deep Learning*

Las redes neuronales intentan emular el funcionamiento de las neuronas humanas en el procesamiento de información: cada neurona funciona como la suma de un campo receptivo dendrítico, una función de decisión no lineal (el soma) y una línea de transmisión de la respuesta (el axón). El modelo de neurona más simple se denomina McCulloch-Pitts, debido a quienes lo definieron. Esta neurona tendrá varias entradas con un peso asociado. La neurona se activa según una función de activación si el parámetro de la función supera un umbral también denominado polarización o bias. Si combinamos varias de estas neuronas simples formaremos una red neuronal denominada perceptrón simple, y si estas neuronas forman varias capas, hablaremos del perceptrón multicapa, cuyo esquema puede verse en la

Figura 2.1. Esta red está formada por un conjunto de neuronas de entrada que simplemente transmiten la información a la siguiente capa o capas ocultas . En cada una de las capas ocultas, cada unidad recibe como entradas las salidas de todas las unidades de la capa anterior, sin existir conexiones entre las unidades de la misma capa o entre capas consecutivas. Por último, la capa de salida la compone un conjunto de neuronas de McCulloch-Pitts, cuyo número será igual a la dimensión de la *target*.

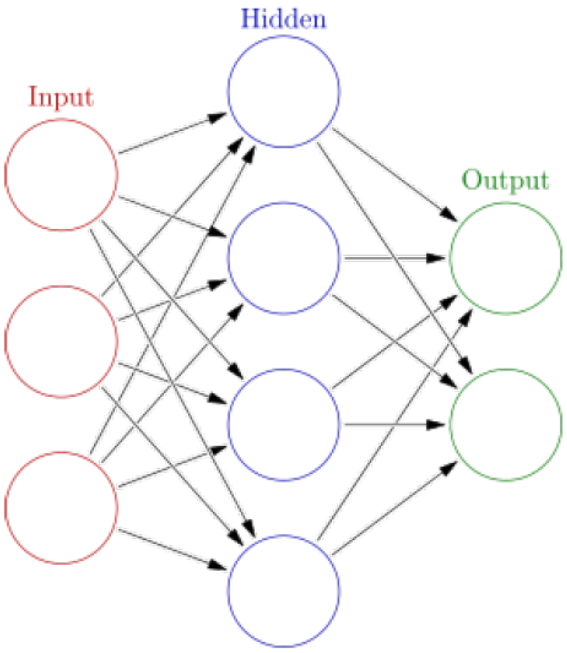

Figura 2.1: Perceptrón multicapa de una capa oculta

Las conexiones de entrada y salida entre las neuronas vienen representadas por pesos. La expresión de un modelo de perceptrón multicapa con una capa oculta es

la siguiente:

$$\hat{Y} = F_2(F_1(xW_1)W_2) \tag{2.1}$$

W_1 son los pesos entre la capa de entrada y la oculta y W_2 entre la capa oculta y la de salida. F_1 y F_2 son funciones de activación. Si nos encontramos en un problema de regresión F_2 será una función lineal mientras que en un problema de clasificación será una función logística o *softmax*. En el caso de un problema no lineal, se usará una función no lineal como la sigmoidal o la tangente hiperbólica.

Para que la predicción del modelo sea lo menos errónea posible habrá que modificar los pesos, el éxito de la red se basará en el algoritmo de ajuste de los pesos. Este algoritmo en un perceptrón multicapa es llamado algoritmo de retropropagación del error y consiste en inicializar los pesos con valores aleatorios, ver las salidas de la red con estos pesos y compararlas con las salidas que sabemos que deberían tener. Si no coinciden, se modifican los pesos de modo que disminuya la función error y así sucesivamente hasta que el error esté por debajo de una cota que hayamos impuesto. Se basa en el método de descenso de gradiente y se aplica secuencialmente a las capas de la red desde la salida hasta la entrada.

Las capas ocultas aprenden representaciones más sofisticadas a medida que se alejan de la entrada. Añadir capas proporciona mayor expresividad al modelo y permite mejorar los resultados. Podríamos entonces pensar que podemos resolver cualquier problema añadiendo cuantas más capas mejor, sin embargo, a partir de

cierto número, normalmente tres o cuatro, aparecen problemas de convergencia que impiden un entrenamiento adecuado del modelo con métodos tradicionales.

Existen otros tipos de redes neuronales más sofisticadas que pueden sacar partido a la estructura interna de los datos, como por ejemplo las imágenes. Una imagen tiene un número de píxeles organizados en filas y columnas y si es en color tiene también tres canales: rojo, verde y azul. El orden de estos datos es importante, contienen una tipología. No solo los píxeles por si mismos tienen información, sino que cómo están ordenados también lo tiene. En lugar de aplicar una multiplicación de matrices entre las activaciones de una capa y los pesos de la capa siguiente, lo que se hace es aplicar el operador de convolución. Esto permite mantener la información de la colocación entre los píxeles. Este tipo de redes se conocen como **Redes neuronales convolucionales** (***Convolutional Neural Networks (CNN)***). Están formadas por un *kernel* que podemos considerarlo como un filtro que se aplica para extraer ciertas características importantes. La convolución consiste en tomar un grupo de píxeles, en el caso de la imagen, e ir realizando un producto escalar con un kernel. El kernel se irá desplazando a lo largo de la imagen por el conjunto de píxeles y realizando el producto escalar. Un ejemplo esquemático de una capa convolucional puede verse en la Figura 2.2. Cada resultado de este producto se almacenará como valor en una matriz. En una capa de tipo convolucional, la multiplicación de matrices entre la entrada (las variables del conjunto de datos o las activaciones de la anterior capa oculta, según corresponda) y los pesos de la propia capa se sustituye por la convolución entre la entrada y distintos kernels.

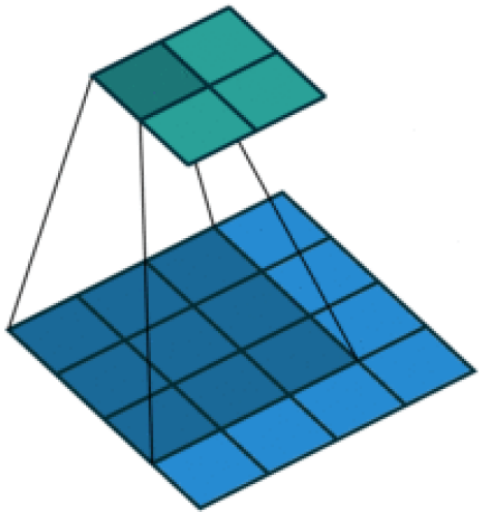

Figura 2.2: Ejemplo de capa convolucional

Es habitual que las capas convolucionales se intercalen con capas de *pooling*. Las capas de *pooling* disminuyen la dimensión de la entrada aplicando una operación, normalmente el máximo (max-pooling) o la media (meanpooling) a las regiones de la entrada.

Este modelo además de para imágenes puede ser utilizado para otros tipos de datos en los que hay una estructura espacial. Una aplicación de este tipo de redes a los textos es la clasificación de los mismos, donde la mínima unidad (al igual que el píxel en la imagen) serían los caracteres. Este método puede ser usado para la clasificación de textos o análisis de sentimiento de un texto.

Si trabajamos con datos secuenciales como frases de texto o series temporales, se utilizan conexiones recurrentes dentro de cada capa. Éstas son bucles que

realimentan cada unidad con su última activación dotando a cada unidad de una especie de "memoria" sobre los datos vistos anteriormente. Es decir, en la capa de este tipo de red se añaden conexiones de su salida hasta su entrada, alimentándose a si misma. En estos casos hablamos de ***Redes neuronales recurrentes*** **(RNN)**, cuyo funcionamiento está representado en la Figura 2.3.

Si nos centramos en el ámbito de textos, el *input* sera una vector *one-hot encoder* (explicado en la sección 2.2.3) de una palabra en un tiempo t. Consideramos un texto como un conjunto de datos secuencial, porque las palabras sucesivas dependen una de la otra. Estas redes permiten al dato de entrada $\bar{x_t}$ interactuar con el estado de la capa oculta creado a raíz de los datos de entrada previos al momento t. La clave es que hay un $\bar{x_t}$ de entrada en cada momento t y un estado oculto $\bar{h_t}$ que cambia en cada marca de tiempo a medida que llegan nuevos puntos de datos. Cada sello de tiempo también tiene un valor de salida $\bar{y_t}$. Esto nos permite predecir la siguiente palabra en un texto, técnica que se conoce como *language modeling*.

La principal arquitectura de este tipo de redes es LSTM (*Long Short-Term Memory*), las cuales tienen una memoria más amplia que las redes recurrentes clásicas, y es lo que las convierte en una arquitectura tan interesante, ya que subsana la desventaja de las RNN en las que la secuencia debe de ser relativamente corta para que las activaciones de momentos anteriores tengan relevancia en la predicción más actual. Este tipo de redes tienen una memoria mayor ya que incorporan el concepto celda de estado (que es la memoria de la red), siendo posible agregar o eliminar información de dicha celda usando una serie de "compuertas" (*forget*, *update* y *output*) que discrimina la información relevante de la irrelevante.

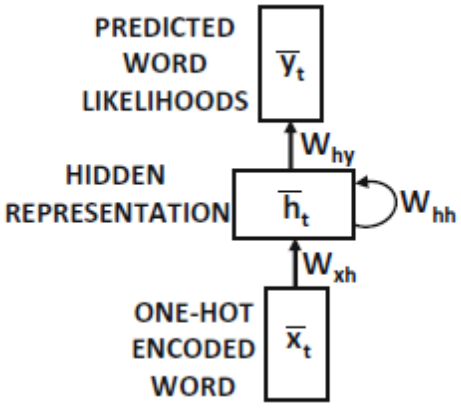

Figura 2.3: Ejemplo de red neuronal recurrente

En general hemos hablado de redes neuronales que pueden ser usadas en un problema supervisado. Sin embargo, si nos encontramos en un caso no supervisado, hay una última arquitectura a la que tenemos que hacer referencia: los *autoencoders*. Estos intentan a partir de una representación reducida de la entrada, reconstruirla. Se utilizan mucho en la reducción de dimensionalidad. La red tiene un codificador, que comprime los datos de entrada, estos pasan por un cuello de botella (capa que divide al *encoder* y *decoder*) y llegan al descodificador, que es la zona donde reconstruye los datos originales a partir de la representación comprimida.

2.2.3 Métodos basados en similitud de espacios vectoriales

En el **modelo de espacio vectorial** el contexto elegido se muestra como vectores cuyas componentes representan los términos existentes en ellos. Cada componente

representa un término que puede existir en la colección y su valor indica su presencia o ausencia en un documento dado.

Dentro de este tipo de vectores podemos diferenciar varios tipos:

- **One-hot encoding:** Cada palabra está representada por un vector cuya longitud es igual al total de palabras únicas en el corpus. Las coordenadas del vector representan cada palabra y toman el valor 1 o 0 indicando la presencia o ausencia del término en un contexto.
- **Term Frequency:** El tratamiento binario no permite distinguir aquellos casos en los que los términos tienen más peso que otros. Esto se resuelve dando a las componentes el valor del peso de cada término (importancia del término en el contexto). En este caso cada vector representa un documento que contiene la frecuencia de las palabras que aparecen en él.
- **Term Frecuency-Inverse Document Frequency (TF-IDF):** Basado en el modelo anterior. Suaviza la frecuencia de las palabras reduciendo el efecto de aquellas que son muy comunes en el corpus gracias al componente IDF: las palabras muy comunes tendrán menos peso que aquellas menos comunes pero más representativas

El problema de este tipo de modelos de espacio vectorial es que si el corpus es muy grande, los vectores también lo serán. Las representaciones vectoriales conocidas como *word embeddings* subsanan este error.

Word Embeddings

Un *word embedding* es una representación de una palabra formada por la representación de densidad de todos los vectores *one-hot-encoding* de cada palabra del corpus, en un espacio de dimensionalidad menor que la dimensión del vocabulario, y cuyos elementos son capaces de capturar la semántica del lenguaje. Así, dos palabras contextualmente similares, por ejemplo "rey" y "reina", serán dos vectores cercanos en el espacio vectorial. En la Figura 2.4 se puede ver un ejemplo de espacio vectorial de palabras.

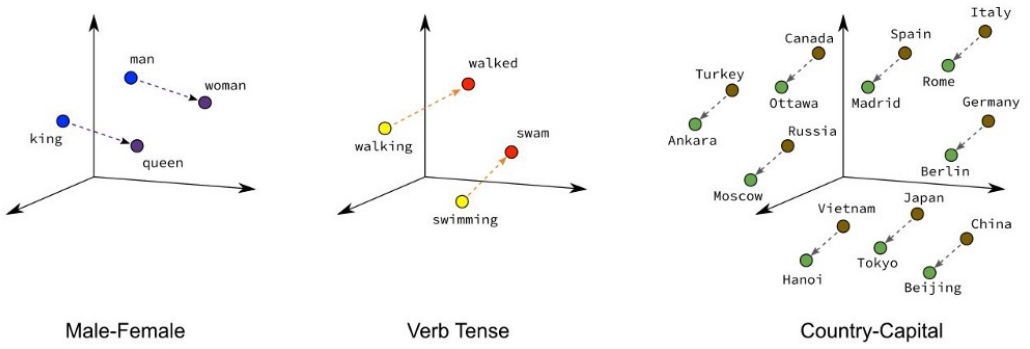

Figura 2.4: Imagen obtenida de Google Developers donde se muestra como aquellas palabras relacionadas se encuentran cerca en el espacio vectorial

Para la obtención de este tipo de vectores existen varias técnicas desarrolladas. Una de las más conocidas es ***word2vec***, modelo creado por (Le and Mikolov, 2014a) y (Mikolov et al., 2013) en Google. Este modelo es un modelo de redes neuronales poco profundas, en el que podemos diferenciar dos arquitecturas: *Continous Bag of Words (CBOW)* o *Skip-gram*. Este modelo aprende a mapear las palabras en el espacio vectorial. En el caso de *CBOW* el objetivo es predecir una palabra dadas

las de su contexto, mientras que *skip-gram* por el contrario predice las palabras del contexto dada una palabra.

La capa de entrada de la red recibe los vectores del contexto, para ello debemos decidir primeramente la longitud de la ventana del contexto, es decir, cuantas palabras antes y después de la palabra objetivo consideramos que definen su contexto. La capa de entrada tendrá tantas neuronas como palabras únicas haya en el vocabulario. El número de neuronas de la capa oculta sería el tamaño del embedding (que suele estar entre 100 y 1000, lo que es un valor mucho menor que el vocabulario de muchos corpus). El número de neuronas de salida es igual al número de entrada.

Otro método para obtener *word embeddings* es ***Global Vectors (GloVe)***. En este caso el proceso de aprendizaje es a raíz de una matriz de co-ocurrencia V x V, siendo V el tamaño del vocabulario, en lugar de una tarea de predicción de palabras como en el caso anterior. Cada entrada de la matriz se corresponde con el número de veces que dos n-gramas del vocabulario co-ocurren. *GloVe* utiliza información global y puede obtener dependencias con términos más alejados de la palabra objetivo que con *word2vec*. En (KS and Sangeetha, 2019) apuntan que empíricamente no hay ninguna ventaja de un método frente a otro, mas allá de que uno se adecúe mejor a los tipos de datos o a la tarea buscada.

Word2vec y *GloVe* tienen una limitación, y es que no pueden lidiar con términos que no aparezcan en el vocabulario, no podrán asignarle un *embedding*. ***FastText*** subsana este error, ya que se basa en los subcomponentes que forman una palabra, los caracteres. Construye el vector que representa a la palabra como la composición

de sus componentes morfológicos. Esto permite construir el vector de una palabra que puede no estar incluida en el vocabulario y que el modelo no ha visto nunca.

Sentence Embeddings

Hasta ahora hemos hablado de vectores de palabras, pero se puede también trabajar con vectores de frases o ***sentence embeddings***. En los casos en los que el corpus sea muy grande, puede ser tedioso trabajar únicamente con palabras, incluso aunque los *word embeddings* reduzcan la dimensionalidad de los vectores de tamaño igual al del vocabulario.

Mikolov y Le presentaron en 2014 su propuesta para *sentence embeddings* llamada *Doc2Vec* (Le and Mikolov, 2014b). Añaden al modelo *CBOW* de *word2vec* otro vector (*Paragraph ID*). De modo que cuando se entrenan los vectores de las palabras se entrena también el vector del documento. Este modelo se llama *Distributed Memory Version of Paragraph Vector (PV-DM)* y podemos ver un esquema del mismo en la Figura 2.5.

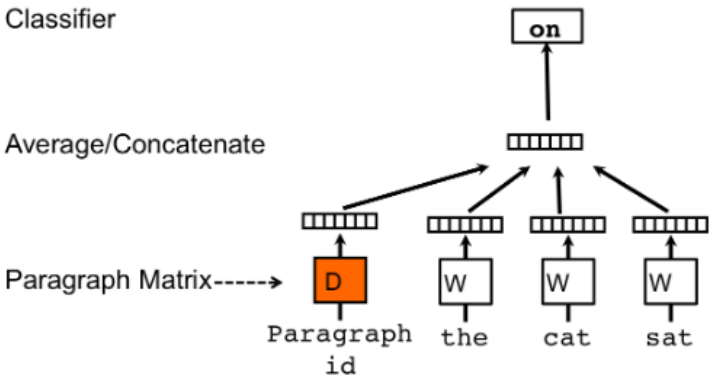

Figura 2.5: Imagen obtenida de (Le and Mikolov, 2014b) que muestra el esquema de la red para entrenar un modelo PVDM

También existe otro algoritmo que se asemeja al método *skip-gram* en *word2vec*, llamado *Distributed Bag of Words version of Paragraph Vector (PVDOBW)*. Es más rápido y consume menos memoria, ya que no tiene que almacenar los vectores de las palabras. En este caso se escogen palabras al azar de una oración y el modelo debe predecir de qué frase provienen, como puede verse en el ejemplo de la Figura 2.6.

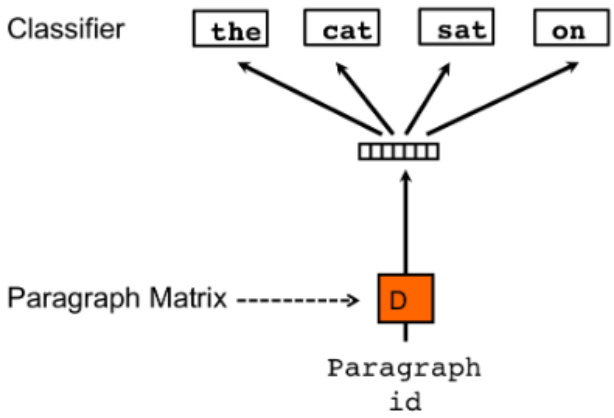

Figura 2.6: Imagen obtenida de (Le and Mikolov, 2014b) que muestra el esquema de la red para entrenar un modelo PVDOBW

Similitud de vectores

Sea cual sea el vector utilizado de los detallados anteriormente y sea cual sea la metodología usada para obtenerlo, la tarea de desambiguar utiliza estos vectores como *input* para la técnica que decida usarse. Si hablamos de vectores clásicos, *one-hot-encoder*, podemos comparar el vector de un acrónimo y el de cada expansión candidata mediante la suma de los términos comunes a ambos (que en realidad es el producto escalar de los dos vectores). Aquella *LF* que dé un valor mayor de producto escalar es la que compartirá más palabras del contexto y, por tanto, la más probable a ser la correcta.

Uno de los métodos más comunes para ver similitudes entre vectores es la similitud coseno. Se basa en la función coseno, la cual viene dada por :

$$cos(d_i, d_j) = \frac{d_i^t d_j}{\| d_i \| \| d_j \|} \tag{2.2}$$

Esta medida es 1 si los objetos son idénticos y -1 si no tienen nada en común.

2.2.4 Modelos codificador-descodificador

En (Cho et al., 2014) y (Sutskever et al., 2014) se presentó este tipo de arquitectura de forma simultánea, como la representada en la Figura 2.7. El primer caso consiste en dos redes neuronales recurrentes (RNN) que actúan como un codificador y un descodificador y en el segundo caso se usan concretamente LSTM. El modelo de Cho da un *score* de cómo de buena es una traducción, mientras que el modelo de Sutskever devuelve la propia traducción. Estas arquitecturas son las usadas actualmente por el traductor de Google.

El codificador mapea una secuencia de entrada de longitud variable a un vector de longitud fija, y el descodificador mapea de nuevo este vector a una secuencia de longitud variable. Se entrenan en conjunto.

El codificador es una RNN o una LSTM que lee cada símbolo de la entrada secuencialmente. A medida que lee cada símbolo, el estado oculto de la RNN/LSTM va cambiando, y cuando termina de leer, el estado oculto contiene un "resumen" de toda la secuencia de entrada. El descodificador es entrenado para generar la secuencia de salida, prediciendo el siguiente símbolo, dado un estado oculto. Este tipo de modelos también son conocidos como *seq2seq*: secuencia a secuencia.

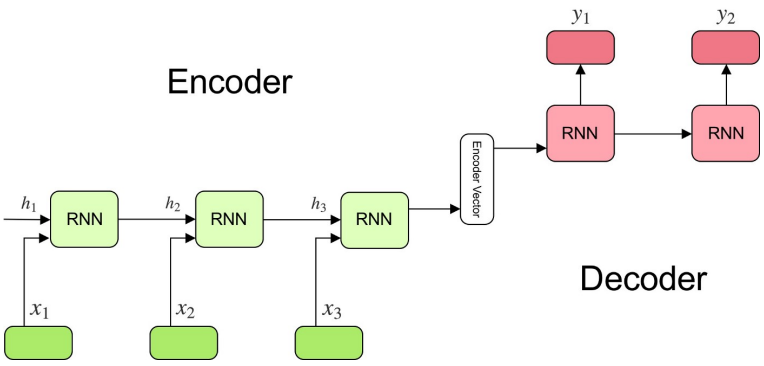

Figura 2.7: Ejemplo de una arquitectura codificador-descodificador con RNN, como en (Cho et al., 2014)

Este tipo de modelos pueden ser usados de dos maneras: generar una secuencia de salida dada una de entrada, o dar un *score* de probabilidad a un par entrada-salida. Una aplicación muy común de estos modelos es la traducción automática, donde la entrada sería el texto a traducir y la salida el texto traducido. Una de las grandes ventajas de este tipo de modelos es que la longitud de las secuencias de entrada y salida no tienen por qué ser iguales. Por ello es de gran ayuda en traducción, porque la secuencia *estoy de camino* en español tendría 3 elementos, mientras que la salida que sería su traducción al inglés *I am on my way* tendría 5.

2.2.5 Mecanismos de atención

Como vimos en la sección 2.2.4, después de que el codificador lea la entrada, la información se acumula en el estado oculto que llamaremos *h(T)*. Cuando el descodificador va a predecir la salida a raíz de este estado, le otorga el mismo peso a cada palabra. Sin embargo, una palabra del conjunto de entrada puede tener correlación con más de una de las palabras del conjunto de salida. Además, en

los modelos *seq2seq* tradicionales, se descartan todos los estados intermedios del encoder y se usan solo los finales (el vector almacenado en el estado *h(T)*) para inicializar el decoder. Estos sistemas funcionan bien cuando la longitud de la secuencia de entrada es pequeña, pero son incapaces de recordar secuencias excesivamente largas, tienden a olvidar las primeras partes una vez que se ha procesado la secuencia entera. Esto se debe a que las RNN tiene memoria corto-placista, y aunque las LSTM tienen una ventana de memoria más grande, esta también es limitada.

Para solucionar este problema se desarrollaron los llamados mecanismos de atención, los cuáles se basan en el comportamiento del cerebro, quien es capaz de priorizar la percepción de unos elementos sobre otros, es decir, no procesa toda la información que le llega a través de los sentidos, sino que presta más atención a una parte que considera más relevante.

¿Por qué es esto importante? Bien, imaginemos que queremos traducir la frase *Ana es una buena chica* al inglés, *Ana is a good girl*. Cuando nuestro cerebro hace la traducción a *good*, en la frase en español se está centrando en la palabra *buena*, cuando hace la de *girl* se centra en *chica* y así sucesivamente. Esto es lo que hace un mecanismo de atención, según la palabra que vaya a predecir, mostrará más interés en una de las palabras de la secuencia de entrada.

El sistema funciona de la siguiente manera, imaginemos que queremos predecir la primera palabra de la sentencia. El descodificador no tiene ningún estado interno en ese momento, por lo que consideraremos el último estado del codificador como

el estado previo del descodificador. Usando este estado y todo los estados del codificador entrenaremos una red neuronal prealimentada. Esta red aprenderá a identificar estados relevantes del codificador, generando puntuaciones altas para aquellos estados en los que se deben tomar atención y puntuaciones bajas para los que deben ser ignorados. Si suponemos que estamos prestando atención a la primera palabra de la sentencia, deberían tener más *score* los estados h_1 y h_2, por lo que las puntuaciones s_1 y s_2 serán mayores. Mediante una capa *softmax* estas puntuaciones se trasforman a probabilidad y por tanto, a pesos: α_1 y α_2. Una vez obtenidos estos pesos, obtendremos el vector de contexto, donde cada elemento es la multiplicación de cada estado de entrada h multiplicado por su peso α.

$$c(t) = \sum_{j=1}^{T} \alpha_{ij} h(t) \tag{2.3}$$

Tengamos en cuenta que estamos aún en el paso de querer calcular la predicción de esta primera palabra, por lo que este vector tendrá valores altos en los dos primeros términos de la secuencia y bajos en los restantes. Tras estos pasos, el descodificador predicirá la palabra correspondiente y generará el estado d_1. Este proceso se generará de manera iterativa para cada elemento de la secuencia, haciendo en cada caso atención en los estados correspondientes del codificador.
Al contrario que en los modelos tradicionales de *seq2seq* en el que el vector de contexto usado por el descodificador es fijo, en este caso se calculan vectores de contexto separados para cada paso de tiempo calculando los pesos de atención en cada momento.

Atención basada en producto escalar

Podemos describir una función de atención como una *query* o consulta y un conjunto de pares clave-valor como salida. La consulta q representa el vector de una palabra, las claves o *keys* k, son el resto de palabras de la secuencia, de dimensión d_k y v el valor vectorial de la palabra que se procesa en ese instante.

La función de atención se calcula de la siguiente forma

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_K}})V \tag{2.4}$$

donde Q, K y V son matrices que contienen todas las ejecuciones de las funciones de atención. Al dividir entre $\sqrt{d_k}$ normalizamos el resultado para evitar que la función *softmax* caiga en regiones de gradientes muy pequeños. La ventaja de este mecanismo frente a la atención aditiva es que este es más eficiente y rápido. Se puede ver un esquema en la Figura 2.8.

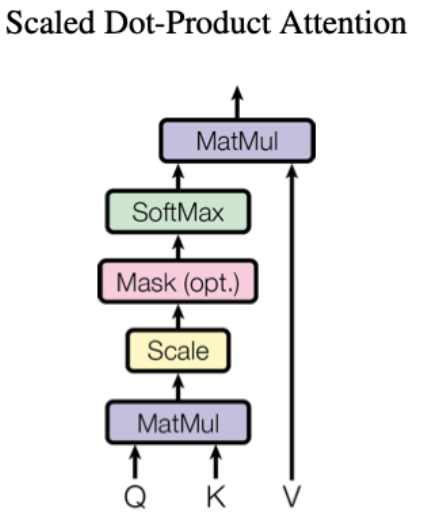

Figura 2.8: Esquema de atención multiplicativa o de producto escalar (Vaswani et al., 2017)

2.2.6 Transformers

En (Vaswani et al., 2017) se expuso la arquitectura de los *Transformer*, la cual se basa en un codificador-descodificador como se explicó anteriormente.

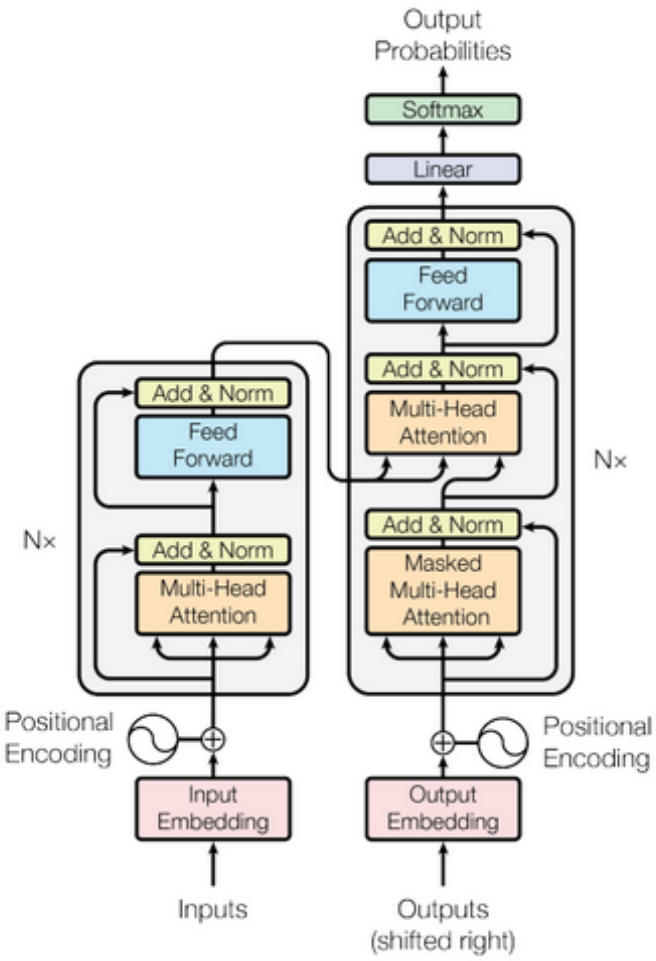

Figura 2.9: Aquitectura de un *Transformer* definida en (Vaswani et al., 2017)

El *encoder* o codificador (bloque izquierdo de la Figura 2.9), está compuesto por 6 capas. Cada una de ellas tiene dos subcapas. La primera es una *multi-head attention* con mecanismos de auto atención y la segunda es una simple red alimentada hacia delante, como se observa en la Figura 2.10. Ambas subcapas tienes conexiones residuales y están conectadas a una capa de normalización.

Cada elemento de la secuencia de entrada entra al codificador como un *embedding*.

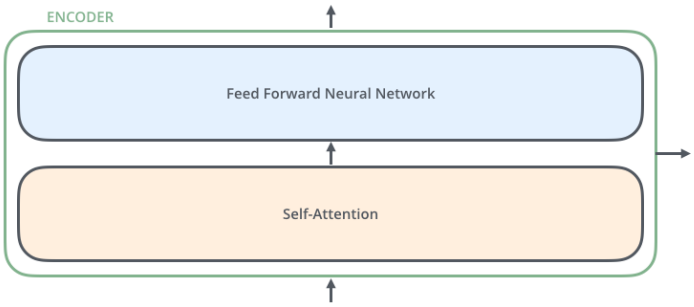

Figura 2.10: Subcapa del codificador esquematizada por (Alammar, 2018)

El *decoder* o descodificador (bloque derecho de la Figura 2.9) también está compuesto por 6 capas. Además de las dos subcapas como en el el codificador, tiene una tercera capa *multi-head attention* que atiende tanto a los estados internos del descodificador como a la salida del codificador. También tiene conexiones residuales seguidas de una capa de normalización.

Self-attention

Esta capa tiene un mecanismo similar a la atención multiplicativa explicada en la sección 2.2.5. Se inicializa con las 3 matrices de pesos correspondientes a las *queries*, las *keys* y los *valores*: W_q, W_k y W_v. Los pesos de estas matrices serán entrenados a la vez que entrenemos el modelo.

En la primera fase del cálculo, se crean las matrices Q, K y V multiplicando la secuencia de entrada con las matrices nombradas anteriormente. La salida se obtiene realizando la operación de la ecuación 2.4. Lo que se calcula en esta ecuación es básicamente lo siguiente:

- Se calcula para cada palabra de la secuencia de entrada, su *score* respecto al resto de palabras. Esto se calcula multiplicando la *querie* de la palabra por los *values* del resto de palabras de la secuencia.
- Dividimos entre la raíz cuadrada de la dimensión de las claves y después lo pasamos por una función *softmax* que normaliza entre 0 y 1.
- El siguiente paso es multiplicar la salida de esta función por el valor de cada palabra. Esto nos pemite identificar palabras importantes y desechar las que no lo son.

En el segundo cálculo, se aplica la función 2.4. Una vez que obtenemos la matriz de pesos con esta ecuación, la multiplicamos por la matriz de entrada. Lo que esto hace en realidad es reemplazar el *embedding* que representa la palabra inicialmente por el que contiene los pesos obtenidos mediante este mecanismo de atención. Se puede ver un esquema en la Figura 2.11.

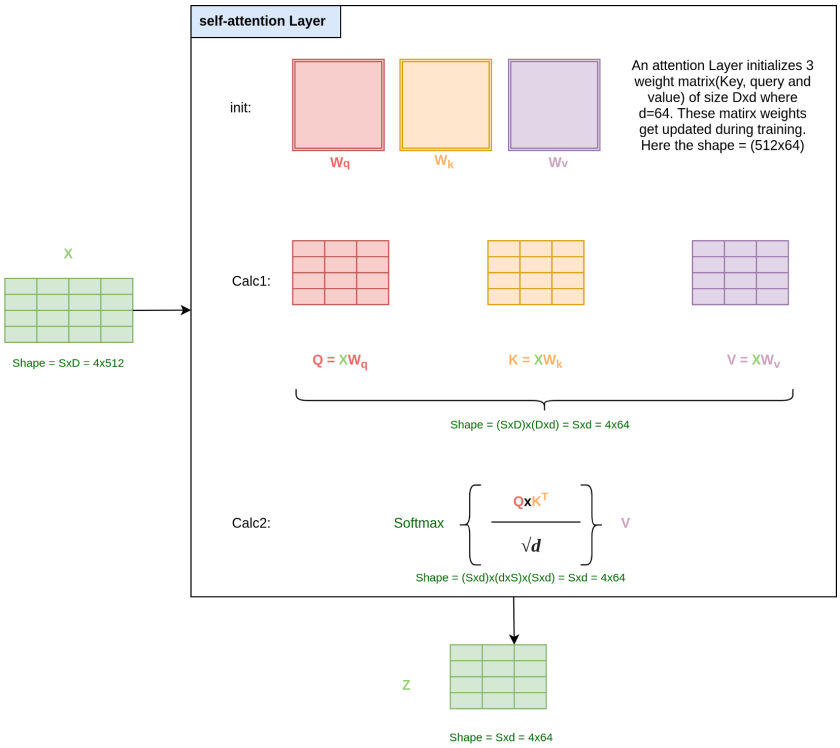

Figura 2.11: Esquema de capa *self-attention* propuesta por (Argwal, 2020)

En esta capa en lugar de relacionar las secuencia de entrada con la de salida, como se explicó enla sección 2.2.5, relaciona elementos de la misma secuencia entre sí y de ahí su nombre de *auto-atención*.

Multi-head attention

Esta idea se basa en añadir varias unidades de *self-attention* como se ve en la Figura 2.12. En lugar de ejecutar una función de atención sola cada vez, utiliza una proyección de Q, K y V en varios espacios lineales. En cada una de estas versiones proyectadas de consultas, claves y valores, se realiza la función de atención en

paralelo, produciendo vectores de salida de dimensión d_v que posteriormente se concatenan. Al tener varias capas de atención, tendremos varias matrices de pesos Q/K/V. Cada conjunto de matrices se inicializa de forma aleatoria y después del entrenamiento, cada conjunto de matrices se utiliza para proyectar los vectores de entrada en un subespacio diferente. Esto permite al modelo que preste atención conjuntamente a información de representaciones sobre diferentes subespacios y posiciones.

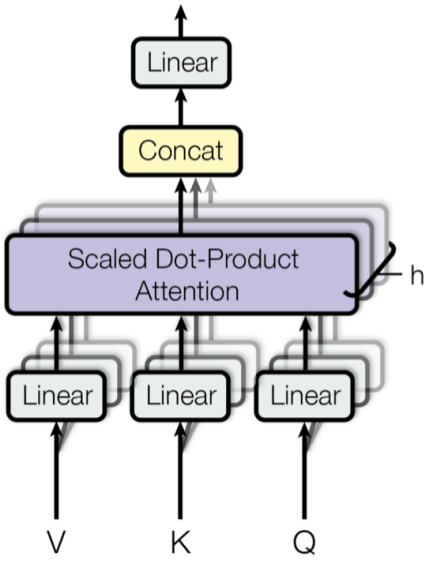

Figura 2.12: Esquema de atención multi unidad o cabeza (Vaswani et al., 2017)

Positional Encodings

El *Transformer* no tiene recursividad ni convolución. Para que el modelo guarde el orden de la secuencia, debemos añadir información sobre la posición de cada elemento en la misma. En caso contrario, el *Transformer* no sabría diferenciar entre

dos secuencias de entrada idénticas pero con orden diferente, como por ejemplo "Ana le ofreció pastel a María" o "María le ofreció pastel a Ana".

En primer lugar, se obtiene el vocabulario del conjunto de entrenamiento y a cada palabra se le asigna un número entero. A estos sencillos *embeddings* se les añade la información posicional dada por:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_model}) \tag{2.5}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_model} \tag{2.6}$$

Así obtenemos los llamados *positional encodings* que son añadidos a los *embeddings* de entrada tanto del codificador como del descodificador. Estos deben tener la misma dimensión que los *embeddings*. Este proceso esquematizado aparece en la Figura 2.13.

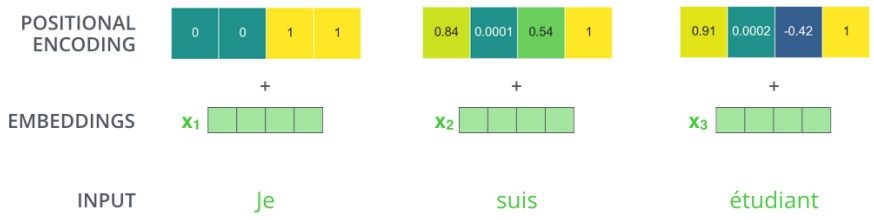

Figura 2.13: Ejemplo de *positional encodings* según (Alammar, 2018)

Descodificador

El descodificador contiene 6 capas apiladas según la publicación de (Vaswani et al., 2017), cuyo esquema aparece en la Figura 2.14. Cada capa contiene a su vez tres

capas: *masked multi-head self-attention*, *multi-head self-attention* y *position-wise fully connected feed-forward network*.

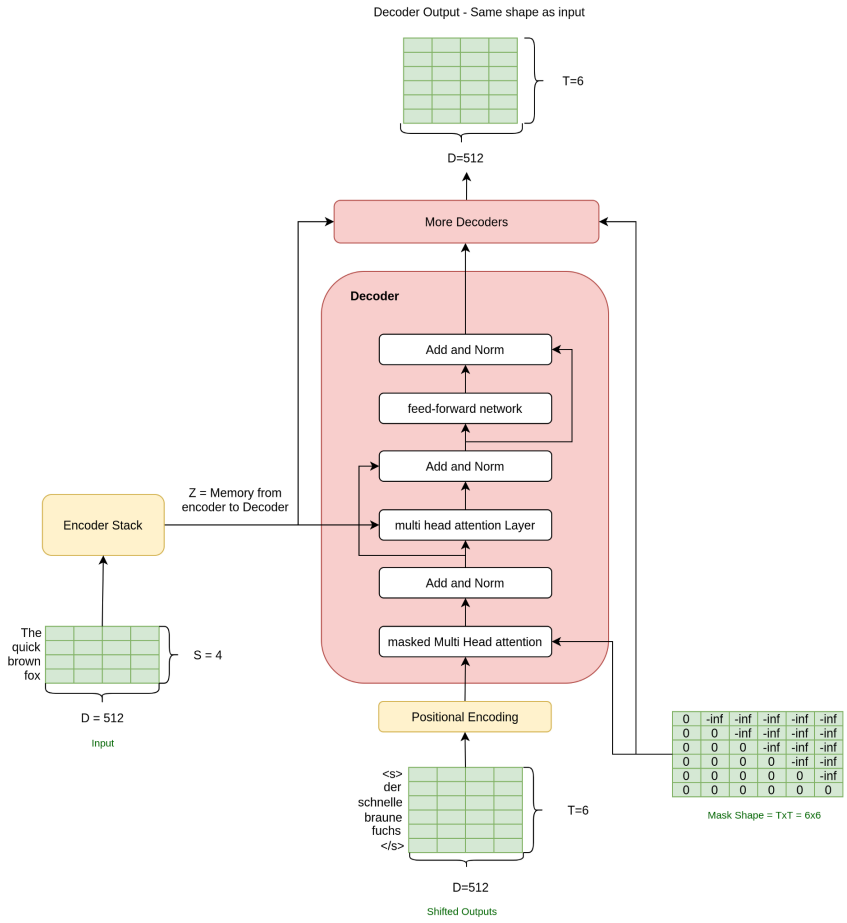

Figura 2.14: Esquema de la arquitectura del descodificador según (Argwal, 2020)

El descodificador funciona como un modelo condicional del lenguaje. Predice la siguiente palabra dada otra palabra o una frase. Pero lo más interesante es que no lo hace de manera secuencial, sino que usa el enmascaramiento o *masking* para hacer este cálculo, en la capa *masked multi-head self-attention*. Lo hace

"enmascarando" futuras posiciones, de modo que la red nunca pueda ver las palabras siguientes, ya que de lo contrario podría copiar dichas palabras durante el entrenamiento. Matemáticamente, se añade una matriz de enmascaramiento a la operación de matrices antes de la función *softmax*, tal y como podemos ver en la Figura 2.15

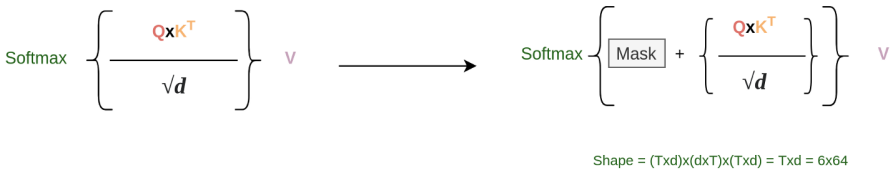

Figura 2.15: Diferencias entre el cálculo de la matriz de auto-atención en la capa de *self-attention* (izqda) y la capa *masked self-attention* según (Argwal, 2020)

Tras este paso pasa por las dos otras capas del descodificador. Al igual que se hacía en el codificador, se añaden los vectores de posición (o *positional encodings*) a las entradas del descodificador. Este proceso se repite hasta que se llega a un tipo de símbolo especial que indica que el descodificador ha completado la secuencia de salida. La salida de cada descodificador alimenta al siguiente descodificador tal y como pasaba en el codificador.

Capa de salida

El descodificador devuelve un vector de números en coma flotante. Para su descodificación en una palabra se utiliza una última capa lineal seguida por una capa *softmax*. La capa lineal es una red neuronal completamente conectada que proyecta el vector salida del descodificador en un vector que tendrá tantas neuronas como el tamaño del vocabulario del modelo, llamado *vector logits*. Cada neurona

tendrá la puntuación de cada palabra. Para pasar esta puntuación a probabilidad, usamos la capa *softmax*. Tras esto, se elegirá la neurona con probabilidad mayor produciendo la palabra asociada a la misma como resultado del cálculo en ese instante.

2.2.7 BERT

Los vectores de palabras detallados en la sección 2.2.3 dieron el comienzo a los modelos preentrenados con grandes conjuntos de datos sin etiquetar como la Wikipedia, o grandes corpus de libros. Esto permitió realizar tareas del procesamiento del lenguaje natural a partir de ellos, es decir, no necesitábamos tener todo el conjunto de datos con los que son entrenados, sino únicamente el modelo ya preentrenado con estas grandes cantidades y aplicarlos para los datos y la tareas que necesitásemos. El problema de modelos como Word2Vec o Glove es que generan un único vector para cada palabra, aunque la palabra sea polisémica y tenga varios significados. Por ejemplo, en las frases "Esta planta debe ser regada a diario" y "La oficina está en la tercera planta", planta tiene dos significados diferentes, sin embargo, ambos modelos le otorgarán un vector único que la define. ELMo (Peters et al., 2018) soluciona este problema, generando vectores contextualizados. Usa una red LSTM-bidireccional y es entrenado para predecir la siguiente palabra en una secuencia de palabras (lo que se conoce como Modelado del Lenguaje o *Language Modeling*). Aquí es donde se empezaron a ver las grandes ventajas de los modelos preentrenados en tareas de PLN.

ULMFit fue un paso más allá y dio lugar a lo que llamamos ajuste-fino o *fine-tuning*.

Es decir, utilizar un modelo preentrenado que aprende la representación general del lenguaje y con unos pocos datos posteriores puede realizar otra tarea con éxito, como la clasificación de documentos. A este método se le denomina Transferencia de aprendizaje o *Transfer Learning*.

Los modelos GTP de OpenAI (Radford et al., 2019) continuaron el desarrollo de este método, básicamente sustituyendo la arquitectura basada en LSTM para el Modelado del Lenguaje por la arquitectura de los *Transformers* explicada enla sección 2.2.6. Sin embargo, presentaban un incoveninente, así como ELMo era bi-direccional, el *Transformer* de OpenAI sólo entrena hacia delante, de modo que sólo puede ver las palabras anteriores para predecir la siguiente.

BERT (*Bidirectional Encoder Representations from Transformers*) (Devlin et al., 2018) soluciona este problema utilizando codificadores que aplican un entrenamiento bidireccional. Como ya se explicó anteriormente, el codificador del *Transformer* lee la secuencia entera de palabras de una sola vez (no lo hace secuencialmente). Por ello se dice que es bidireccional, aunque lo más correcto sería decir que es no direccional. Podemos ver un esquema comparativo de estas tres técnicas en la Figura 2.16.

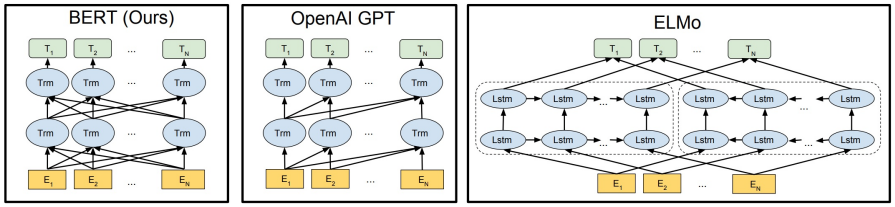

Figura 2.16: Esquema que muestra las diferencias entre ELMo, OpenIA GPT y BERT (Rizvi, 2019)

Masked LM (MLM)

Dado que el codificador lee la secuencia entera, podríamos temer que "viese" todas las palabras y por tanto, la predicción de la siguiente palabra estuviese trucada, pues ya la habría leído previamente. Para solucionar este problema, BERT usa "enmascaramientos". Antes de alimentar BERT con las secuencias de palabras, algunas de ellas son reemplazadas por el *token* [MASK] (en la publicación original los autores enmascaran el 15% aleatoriamente). El modelo entonces intentará predecir el valor original de estas palabras "enmascaradas" basándose en el contexto extraído por las palabras que la rodean. Para ello:

- Se añade una capa de clasificación al final de la salida del codificador.
- Se multiplican los vectores de salida por la matriz de *embeddings*, transformándolos a la dimensión del vocabulario.
- Se calcula la probabilidad de cada palabra aplicando una capa *softmax*.

En este caso no se reconstruirá toda la entrada sino que sólo se predecirán las palabras "enmascaradas". Este tipo de entrenamiento le permite a BERT entender la relación entre palabras.

Next Sentence Prediction (NSP)

En este caso el modelo recibe pares de frases como entrada e intenta predecir si la segunda frase en el par es continuación de la primera en el documento original. En este caso el objetivo es entender la relación entre frases de palabras.

El procesado de los pares es el siguiente:

- Se inserta el *token* [CLS] al principio de la primera frase y el *token* [SEP] al final de cada frase.
- A cada *token* se le añade un *embedding* que indica si es un *token* de la frase A o la B.
- Se añade también a cada *token* un vector de posición para indicar su posición en la frase.

El esquema de este tratamiento se representa en la Figura 2.17. Lo que se calcula al final es la probabilidad de que sea la segunda secuencia la siguiente frase respecto a la primera.

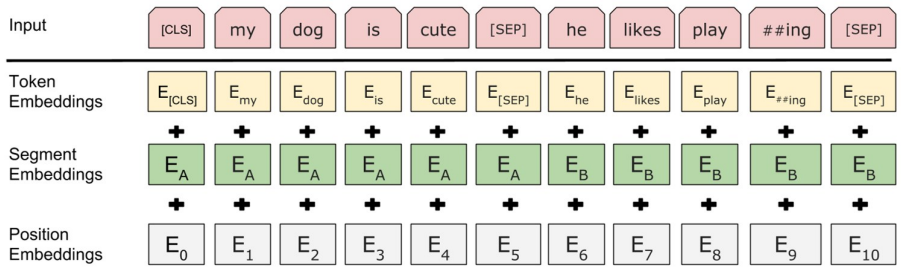

Figura 2.17: Representación del conjunto de entrada a BERT (Devlin et al., 2018)

Ajuste fino o *fine-tunning*

Como se ha explicado antes, *Transfer Learning* consiste en utilizar un modelo preentrnado con ajuste fino. Esto es, una vez que BERT es preentrenado dispondremos de un modelo que es capaz de representar secuencias lingüísticas y pares de secuencias (esquema izquierdo de la Figura 2.18). A continuación, entrenaremos este modelo ya preentrenado con un conjunto de datos para una tarea específica, por ejemplo, con datos de correos electrónicos para clasificar cuales de ellos serán o no *spam*. Para ello únicamente añadiremos una capa a la salida del modelo preentrenado. Esto es lo que se conoce como *fine-tunning* (esquema derecho de la Figura 2.18).

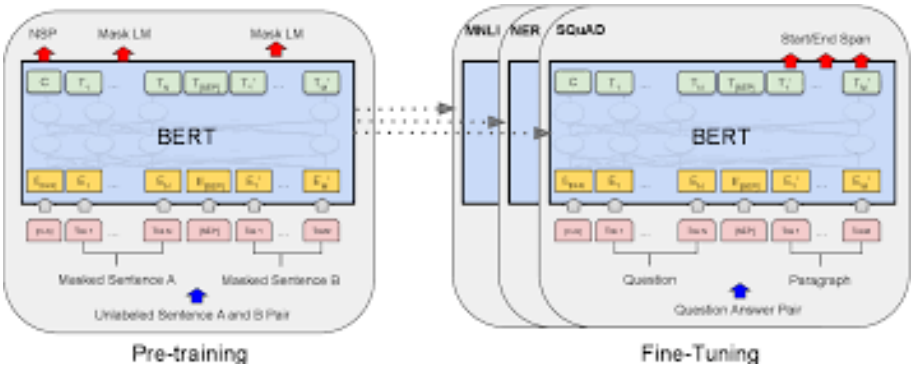

Figura 2.18: Esquema del pre-entrenamiento y ajuste fino en BERT (Devlin et al., 2018)

BETO

BERT ha sido entrenado con corpus en inglés principalmente. Para realizar tareas del procesamiento del lenguaje natural necesitaremos un BERT entrenado con corpus en español: este es BETO. Es entrenado mediante la técnica de

enmascaramiento. Este modelo pre-entrendo en español podemos encontrarlo en la libreria *Hugging Face* [1].

2.2.8 Trabajos de desambiguación basados en las técnicas anteriores

Inicialmente los trabajos basados en la forma, como en el caso de (Schwartz and Hearst, 2003), donde buscan la forma correcta del acrónimo comparando las letras iniciales de la forma larga con las letras de la *SF*, daban buenos resultados. Sin embargo, ya se describió en la sección 2.2.1 la limitación de estas técnicas.

Respecto al uso de modelos supervisados explicado en la sección 2.2.1, (Nadeau and Turney, 2005) obtuvieron buenos resultados utilizando como contexto las palabras obtenidas a la izquierda y a la derecha del acrónimo, en la misma frase en la que éste aparece. Con ellas generaron características que servían como entrada al modelo: número de letras de la *SF* que coinciden con la primera letra de la *LF*, número de letras mayúsculas en la definición, longitud del acrónimo, cantidad de letras de la expansión, entre otras. (Pakhomov et al., 2005) extrajeron el conjunto de palabras que ocurren en la misma frase que la *SF*, sin importar el orden de estas, técnica conocida como *bag-of-words*. Para desambiguar utilizaron dos algoritmos supervisados (árboles de decisión C5.0 y el algoritmo de entropía máxima) como modelos techo, es decir, como modelos ideales con los que comparar otra serie de modelos semi-supervisados. En el ámbito de la literatura biomédica española,

[1] https://github.com/dccuchile/beto

(Rubio-López et al., 2017) añadieron la categoría gramatical de las palabras anterior y posterior a la *SF* a las variables creadas para desambiguar el acrónimo mediante árboles de decisión C4.5. Más recientemente, en la literatura inglesa, (Kashyap et al., 2020) utilizan las palabras que ocurren en una ventana de 1000 caracteres a cada lado del par *SF-LF* y balancean el dataset antes de aplicar un modelo de regresión logística.

En general, exceptuando los trabajos basados en un enfoque puramente supervisado, nombrados con anterioridad, muchos trabajos han optado por el enfoque basado en espacios vectoriales. Una de las ventajas es que esta opción nos libra de la tediosa tarea de generar estas variables "a mano": es decir, calcular la categoría gramatical de cada palabra, o contabilizar el número de mayúsculas, calcular la longitud del acrónimo, etc., es decir, el *input* al modelo son un conjunto de variables que debemos definir y procesar en cada caso concreto. Se puede usar el enfoque más clásico de los vectores, como (Finley et al., 2016) donde simplemente cuentan la co-ocurrencia de la bolsa de palabras alrededor del acrónimo, ponderándola dando más peso a las palabras más cercanas a la *SF*. La desambiguación la realizan a través de varios modelos supervisados obteniendo el mejor resultado con la Regresión Logística.

(Li et al., 2015) proponen dos modelos de *word embeddings* basados en el de (Le and Mikolov, 2014a): asumiendo que la expansión del acrónimo se relaciona con el tema del texto y este es descrito por el top de las palabras *TF-IDF* del mismo, utilizan los *embeddings* de este top de palabras para describir el contexto, llamando a este modelo *TBE*. Por otro lado, suman los *embeddings* de las palabras

que rodean a una en concreto para enriquecer su información semántica, a lo que llaman *SBE*. Estos dos modelos los entrenan sustituyendo en el texto el par *SF-LF* por el acrónimo y un identificador único, obteniendo así un *embedding* que define el contexto de cada significado del mismo. La desambiguación la realizan haciendo similitud coseno de los vectores.

Basándose en este trabajo, (Wu et al., 2015) obtienen *embeddings* con alguna modificación: en lugar de obtener la suma de todas las palabras del contexto en la ventana definida (*SBE*), introducen el concepto de dirección: tiene en cuenta sólo las palabras de la izquierda, o sólo las de la derecha (*LR_SBE*). Por otro lado, generan *embeddings* quedándose con el valor máximo de cada dimensión, con la idea de que aquellas dimensiones que codifiquen el significado del contexto tendrán un valor más alto (*MAX_SBE*). Para asignar la definición correcta, en este caso utilizan un modelo supervisado (SVM), en el que el *input* son los vectores previamente descritos.

(Kai and Shi-Wen, 2016) utilizan *word embeddings* clásicos obtenidos con *word2vec* a raíz de textos sacados de Medline [2]. Desambiguan con un clasificador, pero en este caso, éste es una red neuronal convolucional.

(Charbonnier and Wartena, 2018) utilizan una variante del modelo *TBE* de (Li et al., 2015) que usa también un identificador del contexto, y difiere en que crea los vectores utilizando IDF como peso en lugar de TF-IDF. Desambiguan quedándose

[2]Base de Datos bibliográfica de ciencias de la salud y biomédicas. Incluye información bibliográfica de artículos de revistas científicas de diversas áreas: medicina, enfermería, farmacia y veterinaria. MedlinePlus contiene información de salud autorizada de la Biblioteca Nacional de Medicina (NLM), los Institutos Nacionales de Salud (NIH) y otras agencias gubernamentales y organizaciones relacionadas con la salud. `https://medlineplus.gov/`

con la definición candidata que da mayor valor en la similitud coseno. (Ciosici et al., 2019) diseñan un sistema de desambiguación no supervisado. Para ello sustituyen en el texto cada par *SF-LF* por un único elemento que contiene la *SF* y la *LF* normalizada. Las frases donde aparecen estos pares se utilizan para entrenar un modelo *word2vec*. Desambiguan con similitud coseno. El vector del contexto lo obtienen como el vector medio de las palabras que aparecen en el contexto. (Premkumar et al., 2020) utilizan las mismas técnicas (*word embeddings* con *word2vec* y similitud coseno) con la salvedad de que extraen los textos de internet, haciendo *web scrapping* para buscar publicaciones que contengan las expansiones del acrónimo (extraídas también de internet de la web acronymfinder.com). (Kashyap et al., 2020) extraen los contextos de los pares acrónimo-definición de textos de PubMed y predicen con una Regresión Logística.

Como vemos en muchos de los trabajos se combinan *word embeddings* de la frase en la que aparece la *SF*, la *LF* o el par. Por ello, algunas investigaciones han seguido la línea de trabajar con *sentence embeddings* los cuales ya codifican semánticamente una oración, un párrafo o un documento. (Thakker et al., 2017) utilizan un texto dado con el acrónimo y obtienen sus posibles expansiones haciendo *web scrapping*. Entrenan con ellos el modelo *doc2vec* (PVDM) de (Le and Mikolov, 2014b) para obtener los vectores que codifican el significado de cada texto (el que contiene el acrónimo y el que contiene el par) y desambiguan con similitud coseno de vectores. (Nithyashri et al., 2020) buscan las posibles expansiones del acrónimo de nuevo en la web Acronym Finder, y el contexto de las

mismas de fragmentos web. La representación vectorial de los contextos de la *SF* y la *LF* la obtienen entrenando un modelo *doc2vec* (PVDM). Desambiguan también haciendo similitud coseno de los vectores. (Li et al., 2019) utilizan un modelo del lenguaje, que en lugar de tener un *embedding* fijado para cada palabra, estudia el contexto entero de la frase en la que se encuentra la palabra, para asignarle así cada vez un *embedding*. Está basado en una arquitectura LSTM. Además, extraen la temática del texto mediante el modelo LDA (Latent Dirichlet Allocation): se le asigna de entrada la cantidad de temáticas a buscar, y una vez que segmenta los textos en ellas podemos obtener el top de palabras para cada temática. Desarrollan un modelo que combina los vectores obtenidos mediante ELMo y la información de la temática del texto, obteniendo resultados mejores que los obtenidos mediante técnicas supervisadas o puramente neuronales.

Existen también otro tipo de *embeddings* llamados *embeddings de conceptos*. Los conceptos se basan en identificadores únicos de concepto que están recogidos en la base de conocimiento UMLS (CUIs). Para crearlos, (Tulkens et al., 2016) entrenan primero un modelo *word2vec* con textos de Medline, BioASQ [3] y MIMIC-III [4] para obtener *word embeddings*. Reemplazan cada palabra en

[3] BioASQ es uan organización que propone desafíos sobre indexación semántica biomédica y sistemas pregunta respuesta (QA). Los desafíos incluyen tareas relevantes para la clasificación jerárquica de textos, aprendizaje automático, recuperación de información, control de calidad de textos y datos estructurados, resumen de varios documentos y muchas otras áreas. `http://bioasq.org/`

[4] *Medical Information Mart for Intensive Care*, es una gran base de datos que incluye información relacionada con los pacientes ingresados en las unidades de cuidados intensivos del hospital Beth Israel Deaconess Medical Center en Boston, Massachusetts.

la definición de un acrónimo con su representación vectorial. Aplicando una composición (multiplicación, suma o media), obtienen con ellos un vector de la definición. Estas definiciones son extraídas de UMLS. Con una segunda función de composición se combinan todos los vectores definición correspondiente a un concepto, lo cual da lugar al vector de concepto. Obtienen también un vector que defina el contexto en el que ocurre el acrónimo de la misma dimensionalidad que los vectores entrenados a raíz de los CUIs y desambiguan con similitud coseno de este vector y el vector de concepto. (Sabbir et al., 2017) trabajan también con *concepts embeddings* y desambiguan con varios métodos: similitud coseno, probabilidad basada en una base de conocimiento y el modelo de vecinos más próximos, siendo este último el que arroja un mejor *accuracy*.

(Skreta et al., 2019) obtienen *concepts embedings* entrenando el modelo *FastText* con el corpus MIMIC-III. Para cada *SF* aumentan las muestras de train para cada *LF* con frases que se encuentran cerca de los acrónimos en el espacio de los *embeddings*. Generan *Sentence embeddings* para incorporar también un contexto global, tomando la media ponderada por IDF de los vectores de cada palabra en el documento. Con el contexto local y global generan una representación del acrónimo. Desambiguan maximizando el producto escalar de los vectores.

Por último, nos quedan los *embebddings* formados por la unidad mínima: el carácter. Esto ya se describió previamente con el modelo *FastText*. (Kai and Shi-Wen, 2016) generan vectores *one-hot encoder* en lugar de palabras de

caracteres. La desambiguación la realizan utilizando una red neuronal bidireccional recurrente de memoria larga (Bi_LSTM). Las RNN clásicas sólo pueden visitar información del pasado, estas nuevas añaden un retraso de memoria entre la entrada y el objetivo, aumentando el contexto del pasado y del futuro, así podrá ver los caracteres anteriores al término objetivo y los posteriores. La construcción de la misma no es más que dos RNN conectadas a la capa de salida con mayor almacenamiento de información que una BRNN. La salida de esta red se clasifica mediante la función *softmax*. Este método es útil cuando las fuentes externas son limitadas y no disponemos de un gran corpus. Este método también es usado por (Unanue et al., 2017) no para desambiguar sino para reconocer entidades en textos médicos, tal y como se dijo en la sección de identificación de acrónimos. En este caso utilizan word embeddings obtenidos con *GloVe*, y *character-embebddings*. El uso de estos últimos tiene la utilidad de reconocer acrónimos que no haya visto antes por su estructura morfológica. Por ejemplo, la palabra "acelcististeína" se corresponde al nombre de un medicamento, y aquellas acabadas en "eina" en un texto biomédico tienen una alta de probabilidad de ser medicamentos también. Introducen el *word embedding* y los *character embeddings* de la palabra a una LSTM Bidireccional, y la salida a un clasificador (en este caso un modelo CRF, *Conditional Random Fields*).

Aunque los *Transformers* y BERT son técnicas aún novedosas, ya hay ejemplos de desambiguación donde se utilizan. (Pan et al., 2021) utilizan *adversarial trainning* (añaden ruido a los datos de entrenamiento del modelo) y varios modelos basados en BERT. En (Zhu et al., 2021) además de lo anterior utilizan un *ensemble*

de varios modelos basados en BERT. Este último obtuvo las mejores métricas de la tarea de desambiguación en *AAAI-21 Workshop on Scientific Document Understanding* [5].

Hay que matizar la importancia del conjunto de textos usados a la hora de la desambiguación. Entrenar con un dominio concreto mejora el rendimiento, como se demuestra en (Finley et al., 2016), cuando entrenan con un corpus y predicen sobre otro la predicción es peor. En (Charbonnier and Wartena, 2018) comparan la precisión del modelo entrenándolo con su propio corpus del dominio o utilizando modelos de *embeddings* ya entrenados, como GloVe o Google News, dando estos últimos peores resultados. En (Ciosici et al., 2019) disminuyen las posibles *LF* quitando las que no son del dominio reduciendo así la posibilidad del modelo de equivocarse. También debemos estar atentos al balanceo del dataset, si un término aparece pocas veces en el corpus, modelos como *word2vec* pueden no aprender bien la relación, y que su representación vectorial no sea la correcta. También en (Ciosici et al., 2019) añaden textos de *background* para solventar ese problema. En (Li et al., 2019) y (Kashyap et al., 2020) balancean las clases para que no haya pares acrónimo-expansión con pocas entradas.

En la Tabla 2.2 podemos ver un compedio de las publicaciones nombradas ordenadas cronológicamente y la metodología principal utilizada para la desambiguación de acrónimos. En color azul: métodos basados en enfoques supervisados, en color naranja: métodos basados en similitud de espacios

[5] https://sites.google.com/view/sdu-aaai21/shared-task

vectoriales, en color verde: métodos basados en aprendizaje profundo, en color morado: métodos basados en el contexto del documento, y por último en color amarillo los métodos basados en *Transformers*.

Autores	**Nombre**	**Año**	**Metodología desambiguación**	**Idioma**
Nadeau and Turney	A supervised Learning Approach to Acronym Identification	2005	Modelo Supervisado	Inglés
Parkhomov et al.	Abbreviation and Acronym Disambiguation in Clinical Discourse	2005	Modelo Supervisado	Inglés
Mc.Innes et al.	Using Second-order Vectors in a Knowledge-based Method for Acronym Disambiguation	2011	Similitud de espacios vectoriales	Inglés
Li et al.	Acronym Disambiguation Using Word Embedding	2014	Similitud de espacios vectoriales (word embeddings)	Inglés
Wu et al.	Clinical Abbreviation Disambiguation Using Neural Word Embeddings	2015	Modelo supervisado con word embeddings	Inglés
Finley et al.	Towards Comprehensive Clinical Abbreviation Disambiguation Using Machine-Labeled Training Data	2016	Similitud de espacios vectoriales y modelos supervisados	Inglés
Tulkens et al.	Using Distributed Representations to Disambiguate Biomedical and Clinical Concepts	2016	Similitud de espacios vectoriales (word y concept embeddings)	Inglés
Kai and Shi-Wen	Applying Convolutional Neural Network Model and Auto-expanded Corpus to Biomedical Abbreviation Disambiguation	2016	Word embeddings Clasificador con redes neuronales (CNN)	Inglés
Ganea and Hofmann	Deep Joint Entity Disambiguation with Local Neural Attention	2017	Word embeddings Clasificador con redes neuronales	Inglés

Autores	**Nombre**	**Año**	**Metodología desambiguación**	**Idioma**
Sabbir et al.	Knowledge-Based Biomedical Word Sense Disambiguation with Neural Concept Embeddings	2017	Similitud de espacios vectoriales (word embeddings). Modelos supervisados	Inglés
Rubio-López et al.	Acronym Disambiguation in Spanish Electronic Health Narrativres Using Machine Learning Techniques	2017	Modelos supervisados	Español
Thakker et al.	Acronym Disambiguation: A Domain Independent Approach	2017	Similitud de espacios vectoriales (sentence embeddings)	Inglés
Cuadros et al.	Vicomtech at BARR2: Detecting Biomedical Abbreviations with MLmethods and dictionary-based heuristics	2018	Modelos supervisados	Español
Castaño et al.	A simple approach to Abbreviation Resolution at BARR2, IberEval 2018	2018	Modelos supervisados	Español
Charbonnier and Wartena	Using Word Embeddings for Unsupervised Acronym Disambiguation	2018	Similitud de espacios vectoriales (word embeddings)	Inglés
Ciosci et al.	Unsupervised Abbreviation Disambiguation	2019	Similitud de espacios vectoriales (word embeddings)	Inglés
Skreta et al.	Training without training data: Improving the generability of aytomated medical abbreviation disambiguation	2019	Similitud de espacios vectoriales (word y sentence embeddings). Redes neuronales (CNN)	Inglés
Li et al.	A Neural Topic-Attention Model for MedicalTerm Abbreviation Disambiguation	2019	Modelo contextual vectorial (ELMo) y topic model	Inglés
Kai et al.	Disambiguation of Biomedical Acronyms Based on a Bidirectional Recurrent Neural Network of Character-level features	2019	Character embeddings. Redes neuronales (BI LSTM)	Inglés

Autores	Nombre	Año	Metodología desambiguación	Idioma
Premkumar et al.	Acronym Disambiguation using Web Scraping	2020	Similitud de espacios vectoriales (word embeddings)	Inglés
Kashyap et al.	The CLASSE GATOR (CLinical Acronym SenSE disambiGuATOR): A Method for predicting acronym sense from neonatal clinical notes	2020	Espacios vectoriales. Modelo supervisado.	Inglés
Nithyashri et al.	Web-Based Acronym Expansion Disambiguation Using Word2Vec and Doc2Vec Modeling	2020	Similitud de espacios vectoriales (sentence embeddings)	Inglés
Pan et al.	BERT-based Acronym Disambiguation with Multiple Training Strategies	2021	BERT	Inglés
Zhu et al.	AT-BERT: Adversarial Training BERT for Acronym Identification Winning Solution for SDU@AAAI-21	2021	BERT	Inglés

Tabla 2.2: Resumen de publicaciones enfocado a la desambiguación de acrónimos

2.3 Recapitulación y enfoque escogido

Tal y como se ha visto, las posibles técnicas son diversas. La mayoría de los trabajos nombrados han sido desarrollados para la literatura médica inglesa. El objetivo es aplicar las técnicas más exitosas a la literatura española.

Aunque en la revisión bibliográfica se recopilan también las investigaciones para la identificación de acrónimos, en este trabajo se desarrollará sólo la tarea de desambiguación. El corpus utilizado tiene ya identificados los acrónimos en el texto y se prescinde de la primera tarea de identificación para centrar la investigación en

la desambiguación.

En el campo de la desambiguación, el uso de técnicas en literatura médica española está poco maduro. Principalmente se ha desambiguado basándose en la frecuencia de la aparición de la *LF* (Montalvo et al., 2017), la que contiene un mayor número de co-ocurrencias entre las palabras del texto y las de la posible definición (Sánchez and Martínez, 2018), o entrenando modelos supervisados ((Castaño et al., 2018), (Cuadros et al., 2018), (Rubio-López et al., 2017)). (Sánchez-León, 2018) utiliza simplemente expresiones regulares. Debido a su prometedor éxito en lengua inglesa y que no han sido técnicas explotadas en la literatura médica española, para la tarea de desambiguación, este trabajo utilizará técnicas basadas en *Transformers* como en (Zhu et al., 2021) y (Pan et al., 2021), concretamente con BETO, BERT entrenado para el español.

Visto que el dominio y volumen del corpus es importante, además de los textos médicos de IberEval 2018 [6] se obtendrán de internet publicaciones de Medline que aumenten el corpus de cada acrónimo para no tener un conjunto de datos desbalanceado.

[6] `https://temu.bsc.es/BARR2/` Tarea de evaluación de desambiguación de acrónimos que pone a disposición de los participantes corpus biomédicos anotados por expertos.

3. Método propuesto

En este capítulo se detalla la aplicación de modelos linguísticos basados en mecanismos de atención y preentrenados (explicados de las secciones 2.2.4 a 2.2.7) para la desambiguación de acrónimos en literatura médica española.

3.1 Propuesta

Basándonos en los buenos resultados del uso de arquitecturas tipo *Transformers* en literatura biomédica inglesa para la desambiguación de acrónimos, estudiamos la adaptación de los mismos a nuestro idioma. El hecho de que estos modelos sean prometedores, es que nos abstraen de la ardua tarea de entrenamiento con grandes corpus para la comprensión el lenguaje, y nos permite simplemente añadir una capa posterior que será aquella que entrenemos con nuestros textos biomédicos. Esto nos ahorra un gran coste de tiempo y computación, porque estos modelos ya han sido entrenados para "comprender" la lengua española y simplemente tenemos que "completar" este conocimiento con textos específicos de una materia y entrenar un sistema de clasificación que sea capaz de predecir la correcta definición del

acrónimo.

3.2 Arquitectura

En el trabajo propuesto se utiliza una arquitectura codificador tipo BERT, como la explicada en el capítulo 2.2.7 y una capa de ajuste fino de clasificación. El esquema para un ejemplo concreto, puede verse en la Figura 3.1.

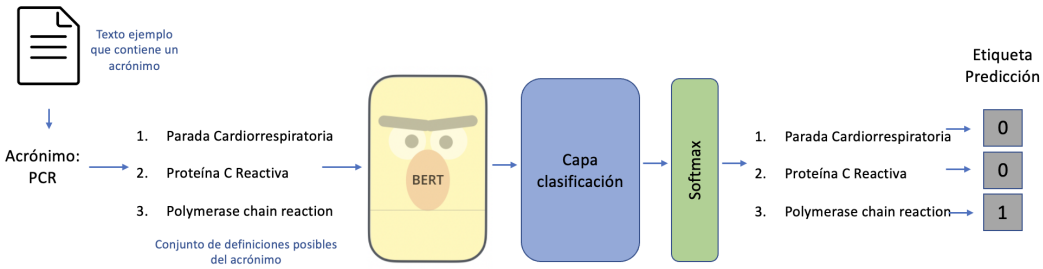

Figura 3.1: Esquema de la solución propuesta para el acrónimo PCR como ejemplo

Los datos, tras su preprocesamiento (que se explicará en la sección 4.2), entran a un modelo $BERT_{based}$ que ya está preentrenado con grandes corpus. Con los datos específicos de nuestro trabajo, este modelo es reentrenado con una capa de clasificación, lo que llamamos en el capítulo 2 ajuste fino o *fine tunning*. Esta capa devuelve un vector no normalizado con las predicciones. Para poder normalizar las probabilidades de esta predicción, añadimos una capa softmax, que en nuestro caso de clasificación binaria generará un valor entre 0 y 1 para cada clase.

3.2.1 Elección del tipo de modelo BERT

La elección del tipo de *Transformer* preentrenado con el idioma español es crucial para obtener buenos resultados. En la Tabla 3.1 podemos ver las diferentes opciones estudiadas en los diferentes experimentos.

Los modelos BERT ofrecidos por Google sólo están disponibles para el inglés, excepto $BERT_{multilingual}$ (Bert, 2019), modelo entrenado para más de 100 idiomas con textos de Wikipedia. Aunque este modelo será utilizado en los experimentos, podemos pensar acertadamente que supone un gasto de gran cantidad de recursos (la dimensión del vocabulario es de 110M) para sólo utilizar una pequeña parte (el español).

Más allá de los modelos entrenados por Google, (Cañete et al., 2020) de la Universidad de Chile, preentrenaron un modelo en español de un tamaño similar a BERT (Devlin et al., 2018), basado en el mismo aunque no oficial: BETO. Este tiene 12 capas de auto atención con 16 cabezas de atención, 1024 de tamaño oculto y un total de 110M de parámetros. Para entrenarlo utilizaron textos de Wikipedia además de fuentes del proyecto OPUS con textos en español (textos de las Naciones Unidas, charlas TED, publicaciones del gobierno, etc) [1]. El corpus final que utilizaron tenía un total de 3 billones de palabras.

[1] Puede consultarse el corpus en `https://github.com/josecannete/spanish-corpora`

Modelo	Características	Textos
BETO uncased	12 capas, 1024 ocultas, 16 cabezas, 110M parámetros	Entrenado con textos en español en minúscula
BETO cased	12 capas, 1024 ocultas, 16 cabezas, 110M parámetros	Entrenado con textos en español con mayúsculas
BERT Multilingual uncased	12 capas, 768 ocultas, 12 cabezas, 110M parámetros	Entrenado con textos en 104 idiomas de Wikipedia, en minúscula
BERT Multilingual cased	12 capas, 768 ocultas, 12 cabezas, 110M parámetros	Entrenado con textos en 104 idiomas de Wikipedia, conteniendo mayúsculas

Tabla 3.1: BERT y modelos $BERT_{based}$ entrenados con literatura española

En el capítulo 5 se detallan los experimentos realizados para determinar la configuración final de este sistema.

3.3 Datos de entrada

Aunque en el capítulo 2 se habló de la identificación de acrónimos, además de su desambiguación, en este trabajo nos centraremos únicamente en la desambiguación. Por ello, el conjunto de datos de entrada debe de tener los acrónimos ya localizados en el texto.

El corpus inicial está formado por textos biomédicos en castellano obtenidos de la tarea BARR2 de IberEval 2018 [2] descritos en la sección 5.1. Este corpus contiene:

[2] https://temu.bsc.es/BARR2/datasets.html

- Identificador del texto obtenido por SciELO [3]
- Texto médico en español
- Localización del acrónimo en el texto
- Acrónimo
- Definición correcta del acrónimo asignada por un experto basada en el contexto del texto
- Definición lematizada

En el capítulo 4 se detallará el procesamiento necesario de los datos previa entrada al modelo. Esta tarea consistirá en dos partes: en la primera los tratamientos necesarios al conjunto de datos para optimizarlos (normalización de definiciones, aumento del número de textos, definición del contexto por acrónimo) y una segunda tarea propia de tratamiento de texto en tareas del lenguaje natural, como la *tokenización*, además de la adaptación de los datos de entrada al formato necesario para un *Transformer*.

El texto médico original que se obtiene del corpus de BARR2, no se utilizará en su totalidad, ya que un texto muy amplio puede dar lugar a ruido y entorpecer al modelo en su predicción. Por ello, dentro de la primera parte de procesamiento de los datos se definirá un contexto para cada acrónimo. La elección del contexto adecuado se describe en la sección 4.2.2. Por otro lado, se analizaron las definiciones otorgadas por los expertos en dicho corpus, encontrando que contenían diferencias en la escritura, significando lo mismo. Por ello, estas definiciones se normalizan, cuyo proceso está detallado en la sección 4.2.1. Tras estas dos etapas

[3] https://scielo.org/

se analizó el balanceo de las diferentes definiciones para cada acrónimo, es decir, en cuántos contextos aparecía una definición concreta de una *SF*. Se observó que existía un gran desbalanceo, problema que se mitigó añadiendo textos obtenidos de Medline. Este proceso se detalla en la sección 5.1. Con estos procesos se completa la fase uno de preprocesamiento obteniendo de esta forma el corpus final con el que se va a trabajar.

Una vez obtenido el conjunto de datos para nuestro problema, entramos en la fase dos del preprocesamiento: adaptación de los datos de entrada al formato necesario para un *Transformer*. Esta etapa se describe de la sección 4.2.3 a la 4.2.5 e incluye la transformación del corpus para generar una etiqueta binaria (etiqueta a predecir por la capa de clasificación), unificación del contexto, acrónimo y definición en una única variable denominada *sentences* y la *tokenización*.

Este proceso se esquematiza en la Figura 3.2:

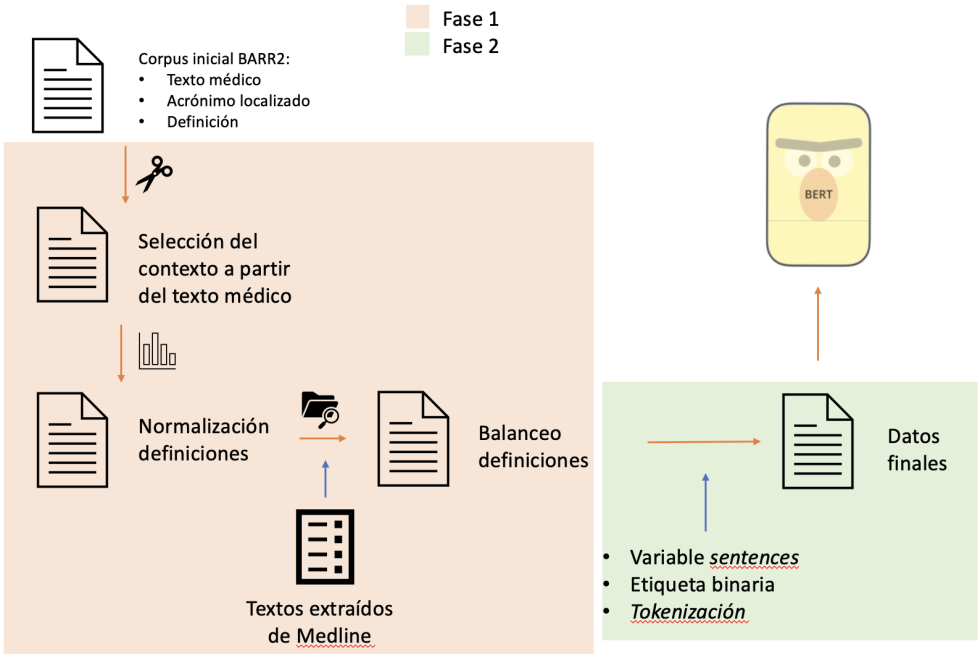

Figura 3.2: Esquema de las dos fases del preprocesado de datos previa entrada al modelo

Recordemos que el conjunto de datos de entrada esta formado por dos *datasets*: el de entrenamiento y el de test. El de entrenamiento o *train* se usará para entrenar la capa de clasificación y la capa de ajuste fino, y el de test para evaluar el rendimiento del modelo. En este segundo caso la variable objetivo (binaria, si la definición asociada al acrónimo es la correcta o no) no es usada por el modelo para realizar sus predicciones, simplemente se usa para evaluar cómo de correctas han sido las mismas.

4. Implementación del modelo propuesto

A continuación, se detallarán todos los procedimientos necesarios para la implementación del método propuesto.

4.1 Herramientas

Para la implementación de la solución se ha utilizado el lenguaje de programación Python. La principal razón es la amplia disponibilidad de librerías tanto para el tratamiento de datos como para el uso de modelos lingüísticos *Transformers*.
Las librerías más utilizadas han sido:

Para el tratamiento de datos:

- Numpy: librería para trabajar numéricamente con *arrays* o matrices de *n* dimensiones.
- Pandas: construido sobre la anterior, es una librería para trabajar y manipular datos estructurados. Es muy conocida para el trabajo de procesamiento de

datos y su análisis.

- Matplotlib: librería para visualización de datos.
- nltk: librería para el tratamiento de datos tipo texto y tareas de procesamiento del lenguaje natural.
- BeautifulSoup: librería para obtener información de páginas web mediante *scrapping*.

Para el modelado:

- Pytorch: librería para trabajar con aprendizaje profundo y arreglos de datos N-dimensionales o tensores.
- Tensorflow: librería desarrollada por Google para trabajar con aprendizaje profundo.
- transformers: librería de Huggingface (Wolf et al., 2020) para trabajar con modelos lingüísticos preentrenados *Transformers*. Se basa tanto en Pytoch como en TensorFlow2.

Para poder ejecutar modelos de este tipo necesitamos un hardware con recursos computacionales necesarios. Además, se debe de tener en cuenta la optimización de tiempo. La ejecución de estos modelos en un entorno local no es posible por ambos temas. Por ello, se usó una GPU de Google Colab conectándonos a ella mediante TensorFlow.

4.2 Preparación de datos

La preparación de datos consiste en dos tareas principales: la primera es optimizar el conjunto de datos disponible (normalización de definiciones, aumento del número de textos, definición del contexto por acrónimo) y la segunda consiste en el tratamiento de datos propio en tareas del lenguaje natural, como la *tokenización*, además de la adaptación de los datos de entrada al formato necesario para un *Transformer*.

El conjunto de datos de entrenamiento y test utilizados son informes médicos anónimos en español proporcionados por IberEval 2018 [1] que se detallarán en la sección 5.1. Estos dos conjuntos de datos están formados por dos archivos diferentes que deben de ser unidos: uno contiene los acrónimos asociados a cada documento, y otro los textos en bruto de cada uno. Cruzamos ambos archivos para tener toda la información unificada por documento, tanto en el conjunto de entrenamiento como en el de testeo. Podemos ver un ejemplo de ambos en las Figuras 4.1 y 4.2

	doc_id	StartOffset	EndOffset	Abbreviation	Definition	Definition_lemmatized
0	S0210-48062004000500008-1	1650	1652	ml	mililitro	mililitro
1	S0210-48062004000500008-1	708	709	l	litro	litro
2	S0210-48062004000500008-1	704	707	mEq	miliequivalente	miliequivalente
3	S0210-48062004000500008-1	677	681	pCO2	presión parcial de co2	presión parcial de co2
4	S0210-48062004000500008-1	2287	2290	HLA	human leucocyte antigen	human leucocyte antiger

Figura 4.1: Ejemplo del archivo de entrenamiento que contiene los acrónimos asociados a cada documento

[1] https://temu.bsc.es/BARR2/

	doc_id	texto
0	S1130-01082006000100014-1	Se trata de una mujer de 35 años, con antecedentes familiares de enfermedad de Crohn y sin antec...
1	S1130-01082009000300015-1	Varón de 70 años, fumador, con enfisema pulmonar y vitíligo al que en mayo de 2001 se realizó un...
2	S0210-56912010000200009-1	Se trata de una mujer de 70 años con antecedentes de HTA y diagnosticada recientemente de neopla...
3	S1130-01082008000900014-1	Varón de 41 años diagnosticado de adenocarcinoma medianamente diferenciado implantado sobre esóf...
4	S0210-48062004000500008-1	Paciente de 29 años de edad que acude al Servicio de Urgencias de nuestro Hospital ante la prese...

Figura 4.2: Ejemplo del archivo de entrenamiento que contiene los textos asociados a cada documento

4.2.1 Normalización de definiciones

En (Ciosici et al., 2019) se dieron cuenta de que en ocasiones una forma larga estaba escrita de dos formas diferentes significando lo mismo. Esto puede ser debido a un error humano en la escritura, a que haya dos formas de decirlo o a traducciones, por ejemplo del inglés al español. Inspirándonos en esta reflexión, se estudió qué sucedía en los conjuntos de entrenamiento utilizados para nuestro proyecto.

Por ejemplo, para el acrónimo TAC, el conjunto de datos utilizado tenía estas diferentes definiciones: "tomografía axial computarizada", "tomografía axial copmputarizada", "tomografía axial computarizada", "tomografía axial computerizada", "tomografía axial computarizada" y "tomografía axial computadorizada". Como vemos, algunas de ellas son exactamente lo mismo sin embargo tienen alguna variación en la escritura lo que da lugar a dos definiciones diferentes para el modelo. ¿Qué problema supone esto? Imaginemos que un texto contiene el acrónimo TAC y su definición asociada otorgada por el experto es "tomografía axial computarizada". Por otro lado el modelo aprende que para ese

tipo de contexto, TAC significa "tomografía axial computadorizada". Cuando evaluemos cómo de bien ha aprendido nuestro modelo con el conjunto de test, nos dirá que el modelo ha fallado a la hora de predecir esta definición, puesto que para él era "tomografía axial computarizada" y no "tomografía axial computadorizada", cuando significan lo mismo.

Para solucionar este problema, se realizó la misma solución utilizada en (Ciosici et al., 2019), que consiste en:

- Para todas las *long form* pertenecientes a un acrónimo calculamos la **distancia Levenshtein** entre ellas. Esta es el número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra, por ejemplo, si tenemos la palabra CASA y CATA, esta distancia sería 1, es decir, sólo habría que hacer una operación, cambiar la S por la T, para transformar una palabra en otra.
- Calculamos un ratio para cada par de definiciones de un mismo acrónimo que es

$$ratio = \frac{dist_{levenshtein}(def_1, def_2)}{len(def_1)} \tag{4.1}$$

- Si este ratio es menor a 0,2 suponemos que son la misma definición.
- Para aquellas con ratio menor a 0,2 reemplazamos la *long form* por la definición más frecuente para dicho acrónimo.

Tras realizar esta transformación, la definición para el acrónimo TAC son: "tomografía axial computarizada". Como vemos, hemos reducido el ruido y así evitamos errores.

4.2.2 Selección del contexto

Para cada acrónimo necesitamos extraer su contexto. Cada documento médico que tenemos puede tener una extensión variable y además contener varios acrónimos. De modo que para el registro del conjunto de datos asociado a un acrónimo y su definición, necesitamos no tener el texto médico completo sino acotar el contexto. Para ello, en primer lugar, lo que hacemos es extraer el texto que hay antes y después de cada acrónimo, ya que tenemos su *offset* de inicio y final en el texto, es decir, su posición en la cadena de caracteres.

Para que el acrónimo esté identificado en el contexto, este se acota entre los *tokens* <*star*> y <*end*>. De este modo, para el acrónimo *CT* podemos tener el contexto ejemplo que aparece en la Figura 4.3:

Varón de 36 años, sin antecedentes de interés, que fue estudiado en la consulta de medicina interna por presentar masa inguinoescrotal izquierda dolorosa a la palpación de dos meses de evolución, sin pérdida de peso ni síndrome miccional.
A la exploración, los testes eran de tamaño y consistencia normales, con un cordón espermático izquierdo indurado y muy doloroso. La ecografía testicular fue normal. La <start> CT <end> de abdomen-pelvis reveló masa de 6 x 3 centímetros en el trayecto del cordón espermático izquierdo sin objetivarse imágenes de afectación retroperitoneal. Con el diagnóstico de tumor paratesticular izquierdo fue intervenido, encontrándose una masa en cordón espermático y realizándose biopsia intraoperatoria informada como proliferación neoplásica de aspecto miofibroblástico no linfomatosa, por lo que se realizó orquiectomía radical izquierda reglada. La anatomía patológica fue de rabdomiosarcoma pleomórfico del cordón espermático, teste y epidídimo normales y negatividad de los márgenes de resección.
Posteriormente el paciente ha recibido varios ciclos de quimioterapia con adriamicina e ifosfamida + MESNA. En las pruebas de imagen de control a los cuatro meses de la cirugía, no se objetivan recidivas tumorales.

Figura 4.3: Ejemplo de texto médico dónde aparece el acrónimo CT acotado entre los *tokens* <*star*> y <*end*>

No el total del texto previo y posterior al acrónimo puede formar su contexto.

Imaginemos que tenemos un texto muy largo, estaríamos dando seguramente información de más y aumentando el ruido. El número de *tokens* que definen el contexto, basándonos en la literatura estudiada en el capítulo 2, se fijó a una ventana de 10 previos al acrónimo y 10 posteriores. El uso de esta ventana dio resultados exitosos, tal y como se verá en la sección 5.3.2.

4.2.3 Etiquetado binario

En este punto tenemos un acrónimo, su definición y su contexto. Sin embargo, nuestro modelo de *Transformer* con ajuste fino, introduce una capa de clasificación binaria. De modo, que debemos transformar nuestro conjunto de datos en un conjunto cuya etiqueta a predecir sea binaria. Para ello, por cada acrónimo y contexto, tendremos un registro con la definición correcta, y por tanto, su etiqueta será 1, y tantos registros como definiciones posibles tenga dicho acrónimo, con etiqueta 0. Para entenderlo mejor podemos ver el ejemplo de la Figura 4.4.

short_form	context	long_form	label
AST	transaminasas en el rango de hepatitis aguda (ALT y < start > AST < end > mayores de 20 veces e...	aspartato aminotransferasa	1
AST	transaminasas en el rango de hepatitis aguda (ALT y < start > AST < end > mayores de 20 veces e...	aspartate and alanine aminotransferase	0
AST	transaminasas en el rango de hepatitis aguda (ALT y < start > AST < end > mayores de 20 veces e...	aspartato transaminasa	0

Figura 4.4: Ejemplo de transformación del conjunto de datos para crear una etiqueta binaria para el modelo de clasificación.

4.2.4 *Sentences* y *labels*

Una vez tenemos el conjunto de datos hay que modificarlo una última vez en un formato aceptable de entrada al *Transformer*. Estos datos de entrada sólo deben tener un texto de entrada y una etiqueta. Por tanto, debemos unir la definición, el acrónimo y su contexto en una única cadena, lo que llamamos *sentences*
La cadena resultante será de este tipo:

```
[CLS] + definición + [SEP] + contexto + [SEP]
```

Recordemos que el contexto ya incluía el acrónimo delimitado por los *tokens* <start> y <end>.

Tenemos un ejemplo en la Figura 4.5:

short_form	context	long_form	label	sentences
GPT	de 68 . Los parámetros bioquímicos en sangre eran normales salvo la GOT 87 , < start > GPT < end > 51 , gamma-GT 67 y las proteínas totales que eran de 5,8 g/dl con albúmina	glutamic pyruvic transaminar	1	[CLS] glutamic pyruvic transaminar [SEP] de 68 . Los parámetros bioquímicos en sangre eran normales salvo la GOT 87 , < start > GPT < end > 51 , gamma-GT 67 y las proteínas totales que eran de 5,8 g/dl con albúmina [SEP]

Figura 4.5: Ejemplo de transformación del conjunto de datos en una única cadena de entrada al modelo tipo *Transformer* .

Por tanto, lo que se utilizará a partir de aquí será lo que contenga la variable *sentences* y la etiqueta que se asignó previamente tal y como se ha descrito en el apartado 4.2.3.

4.2.5 Formato de entrada al *Transformer*

Una vez obtenemos cada conjunto de datos como el almacenado en la variable *sentences* y su correspondiente etiqueta, tenemos que *tokenizar* el texto, es

decir, separarlo en unidades lingüísticas. El *tokenizador* identifica los diferentes elementos que hay en un texto: las palabras, los signos de puntuación, etc, y separa de esta forma cada unidad del mismo. Para poder identificar los diferentes elementos, necesita haber sido entrenado previamente con textos del idioma concreto, para que sepa identificarlos. Por ello, el tipo de *tokenizador* entrenado a utilizar en el modelo es importante, pues tiene que ser capaz de separar las diferentes unidades correctamente. En nuestro caso se usó $BETO_{uncased}$ y $BETO_{cased}$, ambos daban resultados similares. Los textos médicos contienen muchos tecnicismos, de modo que aquellas palabras que ha fallado más al *tokenizar* eran palabras médicas que el modelo no conocía, en ambos casos. Para las palabras generales, ambos modelos hacían un buen trabajo. Pensemos por ejemplo en la palabra *casa*. En $BETO_{cased}$ aparecerá tanto *casa* como *Casa* y en $BETO_{uncased}$ estas palabras las "verá" sin diferencia alguna. Sin embargo lo importante del *tokenizador* es que sea capaz de identificar las formas atómicas del texto, y ya sea con mayúscula o minúscula, si ha comprendido el lenguaje, sabrá identificar que es una unidad a la hora de *tokenizar*.

Hay que remarcar que en este caso no se ha hecho una transformación de todo el texto a minúsculas ni se han eliminado signos de puntuación, debido a que los acrónimos pueden contener ambos tipos de letras además de estos signos. En un primer estudio inicial, se observó que al ejecutar estos pasos de limpieza se "perdían" muchos acrónimos en el texto. Por ello, decidimos decantarnos por $BETO_{cased}$ como *tokenizador*, ya que este diferencia mayúsculas, minúsculas además de acentos y signos de puntuación.

Después de separar en *tokens* cada frase o *sentence* debemos fijar una longitud máxima. Dado que no trabajamos con contextos muy grandes (recordemos que limitamos la ventana de palabras a 10 por delante y por detrás del acrónimo) este valor fue 128. De este modo, a aquellas sentencias que no lleguen a dicha longitud, es decir, que no tengan 128 *tokens* se les añadirán ceros. Además, con el total de elementos o *tokens* diferentes del conjunto de entrada se forma un vocabulario. Este es almacenado en un objeto asignándole a cada uno un índice. Gracias a esto, cada sentencia, de una longitud de 128 elementos estará definida por los índices de cada elemento más los ceros necesarios para llegar a la longitud definida. Para cada sentencia, también se define una sentencia "enmascarada" en la que las posiciones donde había un elemento del texto se sustituye por un 1 y aquellas que fueron añadidas con un cero, se mantienen con un cero.
Por último, estas sentencias son transformadas a un Tensor de Pytorch, que será el formato de datos aceptado por el modelo *Transformer*.

4.3 Modelado

Tras el procesado de datos, se trabaja con los modelos definidos en la sección 3.2.1. Estos modelos se basan en sus correspondientes implementaciones de GitHub [2] [3].

Para ello, se cargan los modelos preentrenados con los que se vaya a trabajar. Posteriormente, se definen sus hiperparámetros (que ya veremos en el capítulo 5) y

[2] https://github.com/dccuchile/beto
[3] https://github.com/google-research/bert/blob/master/multilingual.md

a continuación se reentrenan estos modelos con nuestros datos. Tras este proceso, se realiza la predicción, obteniendo una probabilidad para cada clase normalizada gracias a añadir una capa *softmax*.

Tras obtener nuestras predicciones, los modelos son evaluados según una serie de métricas que se detallarán en el siguiente capítulo.

5. Resultados

En el siguiente capítulo se describen los datos con los que se trabaja, las medidas de evaluación y los experimentos realizados juntos con los resultados obtenidos.

5.1 Corpus

Los datos utilizados son informes médicos anónimos en español proporcionados por la tarea BARR2 de IberEval 2018 [1], ya que es la única colección de textos etiquetados en español para el objetivo de la desambiguación. Este corpus está formado gracias a artículos obtenidos de SciELO [2]. Esta iniciativa reúne publicaciones electrónicas de artículos completos de revistas científicas de América Latina, Sudáfrica y España. Para la creación del corpus se extrajeron los casos clínicos de los artículos de SciELO obteniendo así: 69 artículos de SciELO Argentina, 99 de SciELO Brasil, 744 de SciELO Chile, 65 de SciELO Colombia y

[1] Ibereval 2018: https://sites.google.com/view/ibereval-2018
BARR2: https://temu.bsc.es/BARR2/ textos sin licencia

[2] SciELO (*Scientific Electronic Library Online*) es una librería electrónica mantenida por *Sao Paulo Research Foundation (FAPESP)* y *Brazilian National Council for Scientific and Technological Development (BIREME)* https://scielo.org/

por último 2490 de SciELO España. Del total disponible en IberEval 2018 para la tarea BARR2 en este trabajo sólo se utilizaron los artículos correspondientes a los conjuntos de entrenamiento y testeo. El conjunto de test está formado por 220 casos clínicos y el conjunto de entrenamiento por 318 (para más información sobre el conjunto de datos consultar el artículo donde se describe la tarea de investigación BARR2 en la que se propone la desambiguación de acrónimos, (Intxaurrondo et al., 2018)).

Los casos clínicos contenidos en este corpus cubren un amplio rango de disciplinas médicas, incluyendo oftalmología, urología, enfermedades digestivas, cirugía, atención primaria, pediatría, medicina interna, nefrología, cirugía plástica, cuidados intensivos, farmacología y oncología.

El conjunto de datos contiene el informe médico, los acrónimos que aparecen en el y sus definiciones correctas anotadas por expertos. Este conjunto de datos será posteriormente procesado para la entrada al modelo, proceso que se detalló en el capítulo 4. En total disponemos de 768 acrónimos en el conjunto de entrenamiento, de los cuales la mayoría tienen un único significado, como se aprecia en la Figura 5.1.

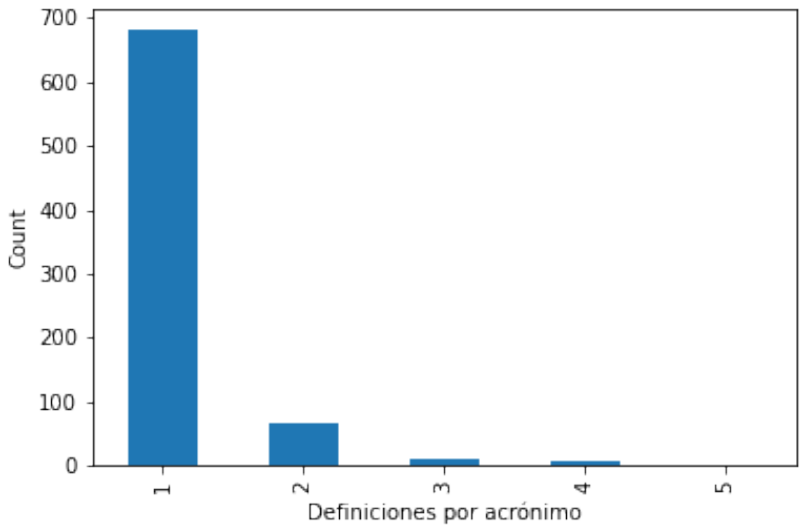

Figura 5.1: Cantidad de acrónimos según sus significados posibles

Los distintos significados de los acrónimos están bastante desbalanceados. Si representamos la cantidad de textos en los que aparece una definición se obtiene el gráfico representado en la Figura 5.2.

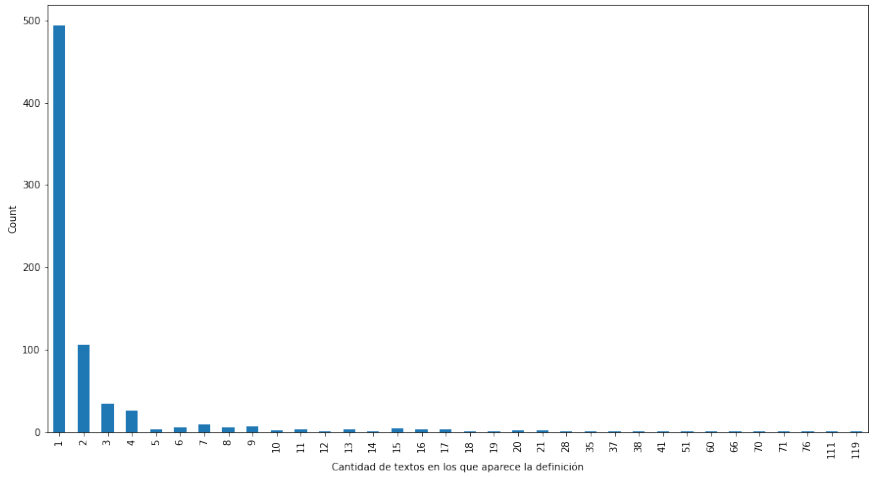

Figura 5.2: Cantidad de textos diferentes en los que aparece una definición

Para intentar balancear las diferentes definiciones, aquellas que aparece en tres textos o menos son buscadas en Medline, con la idea de poder aumentar la cantidad de textos que las contienen. Recordemos que en el capítulo 2 se vio como el factor más importante para saber distinguir la forma larga que representa el acrónimo en un texto se basa en el contexto del mismo. Si al modelo solo le damos como entrada, imaginemos, un texto para una definición, le será difícil generalizar un contexto en el que aparece esa definición, para poder ser capaz de detectarlo en el futuro.

En resumen, de los 768 acrónimos del conjunto de entrenamiento a los que les corresponden 735 definiciones, había 542 definiciones o formas largas con tres textos o menos en el conjunto de *train* o entrenamiento. Esas 542 definiciones son buscadas en Medline mediante *scrapping*, de las que sólo se encuentran 27.

Esto supuso añadir 564 textos a los datos de entrenamiento. La cantidad de textos diferentes en los que aparece una definición tras añadir los textos de Medline se representa en la Figura 5.3.

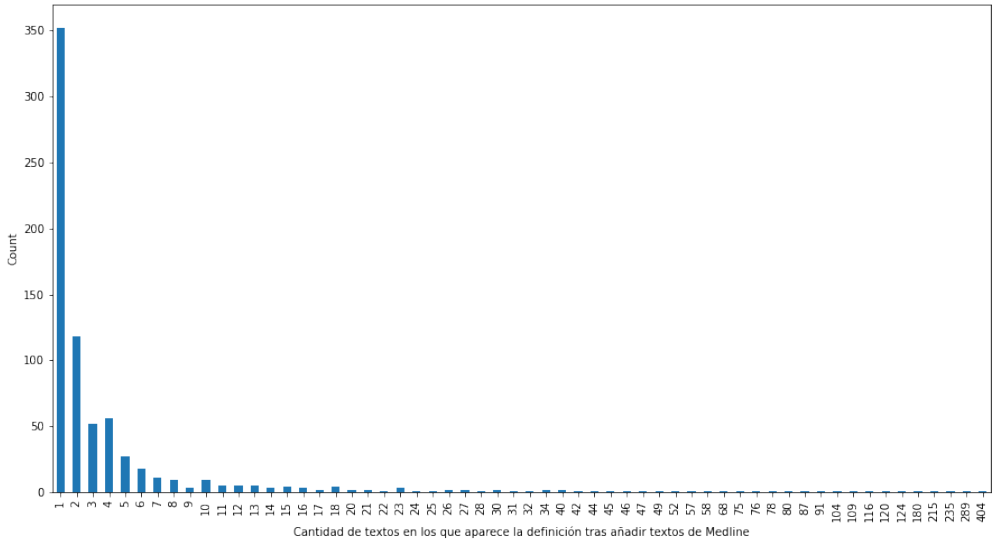

Figura 5.3: Cantidad de textos diferentes en los que aparece una definición tras añadir textos de Medline

La obtención de textos de Medline mediante la técnica de *scrapping* se divide en dos: búsqueda de información por concepto en la página de inicio de MedlinePlus [3] y búsqueda por letra del abecedario en la enciclopedia [4]. En el primer caso los textos no tienen derechos de autor, mientras que en el segundo si (Enciclopedia médica A.D.A.M.). En este trabajo sólo se han utilizado los textos extraídos de la página de inicio, sin derechos de autor.

[3] https://medlineplus.gov/spanish/
[4] https://medlineplus.gov/spanish/encyclopedia.html

5.2 Medidas de evaluación

Para comparar los diferentes experimentos se han usado medidas de evaluación propias de un sistema de clasificación binario, como se ha venido haciendo en los trabajos del estado del arte descritos en el capítulo 2.

Recordemos que en un sistema de clasificación binaria, la etiqueta a predecir será del tipo 1 o 0. En nuestro caso, será 1 cuando el acrónimo que aparece en el texto corresponda a la definición asignada, y 0 cuando esa definición sea propia del acrónimo pero no correcta para el contexto del mismo. Nuestro modelo de clasificación (que según se explicó en el capítulo 4 es una capa añadida al modelo $BERT_{based}$ por el método de ajuste fino) tendrá como objetivo predecir esta etiqueta, es decir, decidir si para un texto, un acrónimo y una definición dada, esta debe de ser 1 (definición correcta) o 0 (definición errónea).

La base para evaluar un sistema de clasificación binaria, es la matriz de confusión (ver Figura 5.4).

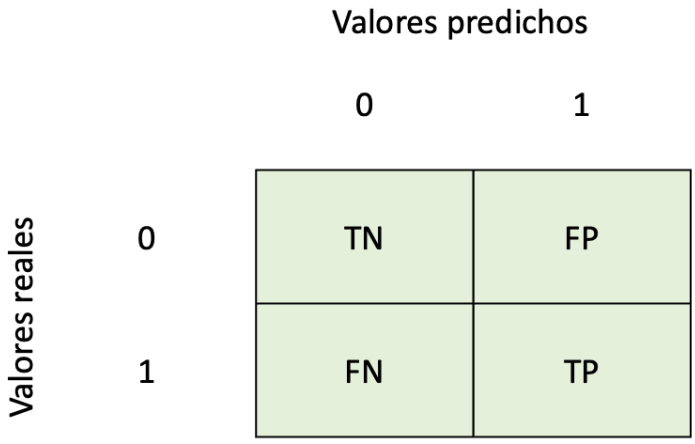

Figura 5.4: Matriz de confusión para un sistema binario

Esta tabla mide el rendimiento del modelo de la siguiente forma:

- **TN:** *True Negatives* o Verdaderos Negativos. Aquellos que son negativos y el modelo predice como negativos.
- **FN:** *False Negatives* o Falsos Negativos. Aquellos que son positivos y el modelo predice como negativos.
- **TP:** *True Positives* o Verdaderos Positivos. Aquellos que son positivos y el modelo predice como positivos.
- **FP:** *False Positives* o Falsos Positivos. Aquellos que son negativos y el modelo predice como positivos.

Basándonos en estas mediciones podemos definir diferentes métricas:

- **Accuracy**: mide el porcentaje de acierto del modelo, tanto de positivos como de negativos.

$$accuracy = \frac{TP + TN}{N} \tag{5.1}$$

- **Precision**: mide el total de predicciones positivas correctas del total de predicciones positivas.

$$precision = \frac{TP}{TP+FP} \tag{5.2}$$

- **Exhaustividad o recall**: mide el total de predicciones positivas correctas respecto al total de valores positivos.

$$recall = \frac{TP}{TP+FN} \tag{5.3}$$

- **F1 score**: es la media armónica de la precision y el recall.

$$F1score = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{5.4}$$

En nuestro caso, nos interesa que el modelo sea capaz de detectar la definición correcta del acrónimo, es decir, los unos. Por ello, la medida *accuracy* no es la más adecuada, ya que enmascara la cantidad de valores positivos predichos con los negativos. Esta métrica no es muy útil cuando las clases están muy desbalanceadas, como en nuestro caso. Es decir, podríamos tener un porcentaje de acierto muy alto, pero que sea debido a que acierta muchos negativos. Por ello, las medidas que más nos interesan son la *precision* y el *recall*. La *precision* nos da la calidad de la predicción mientras que el *recall* nos da la cantidad de aciertos positivos. Hay que tener en cuenta que la salida de un modelo de clasificación no es directamente la etiqueta (también llamado salida dura) sino que es la probabilidad de que sea

el valor positivo de la misma (o salida blanda). Por defecto, este valor es 0,5, es decir, si esta probabilidad es mayor o igual a 0,5 asignará la etiqueta 1, en el caso contrario será 0. Por tanto, si cambiamos este umbral de probabilidad, la cantidad de positivos o negativos predichos cambiará. Por ello, si queremos construir un modelo en el que las predicciones positivas sean muy seguras, podemos subir el umbral a por ejemplo 0,7. En este caso tendríamos un clasificador con una *precision* alta y un *recall* bajo, ya que disminuiría la cantidad de falsos positivos (las predicciones positivas serían más certeras) pero podría aumentar la cantidad de falsos negativos, pues por ejemplo, un positivo con probabilidad de 0,6 sería predicho como negativo, no siéndolo. Por otro lado, si quisiéramos un clasificador que no se deje ningún positivo, aunque se "cuele" algún negativo, deberíamos bajar el umbral, y en este caso tendríamos una baja *precision* y un alto *recall*. Con esto vemos que no es fácil optimizar ambas métricas, por ello una métrica que combinan ambas es *F1-score*. Dado que para este caso consideramos ambas métricas importantes, utilizaremos la media armónica de ambas, *F1-score*.

5.3 Experimentación y evaluación

En el capítulo 4 se habló de como el contexto del acrónimo era decisivo para la clasificación, además de la normalización de las definiciones, y de intentar balancear las muestras añadiendo textos obtenidos de Medline de aquellas definiciones minoritarias. Todas estas casuísticas han sido parámetros en la experimentación, para estudiar aquellas condiciones que dan lugar a mejores

resultados.

Pero además de estos parámetros, también han sido estudiados otros que veremos a continuación.

5.3.1 Hiperparámetros del *fine tuning*

Los hiperparámetros de un modelo son aquellos que nosotros podemos elegir, es decir, que no serán optimizados ni elegidos por el modelo, sino que son una de las entradas que debemos indicar. De normal cada modelo tendrá una serie de hiperparámetros predefinidos por defecto, pero que como ya hemos dicho, podemos modificar.

Como ya dijimos en el capítulo 4 utilizaremos un modelo BETO o $BERT_{based}$ con una capa de ajuste fino, es decir, utilizaremos el modelo ya entrenado pero añadiremos una capa de clasificación y entrenaremos esta última capa. Será en este punto en el que podemos decidir los valores de los hiperparámetros. Estos serán los siguientes:

- **Optimizador:** En una red neuronal, los pesos son inicialmente aleatorios. A medida que se entrena la red estos pesos se van ajustando. El optimizador es el encargado de cambiar los valores de los pesos según pasan los ciclos de entrenamiento.
- **Epoch:** Número de ciclos de entrenamiento.
- **Batch Size:** Número de datos que hay en cada ciclo de entrenamiento.
- **Learning Rate:** Indica cómo de extenso sera el camino que tome el algoritmo de optimización. Si el valor es muy pequeño se puede quedar

atrapado en un mínimo local y la red tardará mucho más tiempo en optimizar. Por otro, lado si los valores son muy altos puede "saltarse" el mínimo global y nunca encontrarlo, aunque en este caso el aprendizaje es más rápido.

- **Maximum length:** Número de elementos o *tokens* máximo en cada sentencia.

Basándonos en la solución ganadora propuesta por (Zhu et al., 2021) se experimentó con un *learning rate* de $1 \cdot e^{-5}$, el optimizador AdamW y un *batch size* de 32, obteniendo buenos resultados. Con el número de *epochs* se hicieron pruebas con 4, 8 y 15 (configuración ganadora de (Zhu et al., 2021)) ciclos. 8 *epochs* daban buenos resultados, así como 15 ciclos, aunque este último valor no mejoraba notablemente el modelo y consumía muchos recursos de computación. De modo que los hiperparámetros de nuestro modelo final fueron:

- **Optimizador:** AdamW
- **Epoch:** 8
- **Batch Size:** 32
- **Learning Rate:** $1 \cdot e^{-5}$
- **Maximum length:** 128

5.3.2 Preprocesamiento de datos

Como ya hemos dicho a lo largo de este trabajo, el contexto del acrónimo es bastante relevante para poder identificar el significado del mismo. Basándonos en la literatura se experimentó con una ventana de 10 *tokens* antes y después del acrónimo obteniendo buenos resultados.

Por otro lado, se experimentó normalizando las definiciones de los acrónimos, como se detalla en la sección 4.2.1, lo que también fue positivo. Además, se intentó balancear las muestras añadiendo textos obtenidos de Medline mediante *scrapping* para que aquellas definiciones con tres o menos textos en el conjunto de datos pudiesen aumentar su presencia para que el modelo fuese capaz de detectarlas.
En la Tabla 5.1 se puede ver los experimentos con los diferentes tratamientos de datos explicados. Además de estos parámetros también se han ido variando los modelos, lo cual se detallará en la sección 5.3.3. Las métricas utilizadas para medir el rendimiento del modelo son las explicadas en la sección 5.2. En estos experimentos se ha utilizado el conjunto de datos descrito en la sección 5.1, donde el conjunto de entrenamiento se ha usado para el proceso de aprendizaje del modelo y el conjunto de test para su evaluación, obteniendo así las métricas que aparecen en dicha tabla.

Modelo	**Tratamiento de datos**	**Precision**	**Recall**	**F1-score**
BETO uncased, BETO cased, BERT multilingual	Definiciones normalizadas y solo acrónimos ambiguos	0.53723	0.12752	0.2061
BETO uncased, BETO cased, BERT multilingual	Solo acrónimos que están en train y test Definiciones normalizadas y solo acrónimos ambiguos	0.5265	0.1755	0.2632
BETO uncased	Definiciones que están en train y test	0.989	0.159	0.2745
BETO uncased	Solo acrónimos ambiguos. Definiciones normalizadas. Textos de Medline añadidos	0.464	0.213	0.292
BETO uncased	Todos los acrónimos. Definiciones normalizadas. Textos de Medline añadidos	0.814	0.957	0.88
BETO uncased, BETO cased, BERT multilingual	Todos los acrónimos. Definiciones normalizadas. Textos de Medline añadidos	0.815	0.949	0.887
BETO uncased, BETO cased, BERT multilingual	Sólo acrónimos en test que aparecen en train. Definiciones normalizadas. Textos de Medline añadidos	**0.975**	**0.964**	**0.969**
BETO uncased	Todos los acrónimos Definiciones normalizadas. Textos de Medline añadidos	**0.815**	**0.968**	**0.884**
BETO cased	Todos los acrónimos Definiciones normalizadas. Textos de Medline añadidos	**0.82**	**0.955**	**0.884**
BETO uncased, BETO cased	Todos los acrónimos Definiciones normalizadas. Textos de Medline añadidos	**0.824**	**0.947**	**0.881**

Tabla 5.1: Experimentos realizados

En la Tabla 5.1 vemos que la primera línea cuyas métricas están remarcadas (modelos BETO uncased, BETO cased y BERT multilingual) da métricas muy buenas. Pero hay que tener en cuenta que sólo aparecen en el conjunto de test aquellos acrónimos que también estaban en el conjunto de entrenamiento. Este tema se abordará en la sección 5.3.5. A continuación veremos que en algunos casos aparecen los tres modelos del *ensemble* y en otros sólo uno de ellos.

5.3.3 *Ensemble* de modelos

Un *ensemble* de modelo consiste en combinar diferentes modelos bajo la premisa de que varios modelos pueden dar lugar a un modelo más robusto. Existen diferentes técnicas de *ensemble*:

- ***Bagging**:* se entrena el mismo tipo de modelo pero cada uno con una muestra aleatoria de los datos (con repetición). Los resultados se combinan en problemas de clasificación, quedándose con la opción que tiene mayor frecuencia.
- ***Boosting**:* aquí cada modelo se entrena prestando especial atención a los errores cometidos por el modelo anterior.
- ***Votación**:* en este caso se utilizan varios modelos y el resultado para la clasificación puede ser, o la clase que ha recibido más votos (votación dura) o la mayor suma de probabilidades (votación blanda).

En nuestro caso se usó un *ensemble* por votación de tres modelos: BETO *cased*, BETO *uncased* y BERT *multilingual*.

Tras estudiar el rendimiento de los tres modelos juntos en *ensemble* y por separado, se observó que con la mejor configuración obtenida el modelo que realmente estaba aportando las mejores métricas mediante el sistema de votos era BETO. En la Tabla 5.2 podemos observar las métricas de cada modelo por separado en el conjunto de **entrenamiento** y del *ensemble* evaluado con los datos de test, con los hiperparámetros definidos en la sección 5.3.1, las definiciones normalizadas, textos de Medline añadidos y sólo aquellas definiciones que están tanto en train como en test.

	BETO ***uncased***	**BETO** ***cased***	**BERT** ***multilingual***	***Ensemble***
precision	0,97	0,73	0,0	0,97
recall	0,93	1	0,0	0,96
f1 score	0,95	0,84	0,0	0,97
auc	0,93	0,73	0,27	0,94

Tabla 5.2: Métricas de evaluación con el conjunto de entrenamiento de cada modelo independiente y del *ensemble* de los mismos

Como se observa, las mejores métricas las aporta BETO, el modelo $BERT_{multilingual}$ tiene un mal rendimiento, no es capaz de aprender nada de los datos de entrenamiento, como se observa en la cuarta columna de la Tabla 5.2. Tal y como se predijo en la sección 3.2.1 este introduce mucho ruido al tener varios idiomas y no es una buena opción. Descartando $BERT_{multilingual}$, se realizó también un *ensemble* con ambos modelos de BETO para analizar los resultados. En la Tabla 5.3 podemos ver el rendimiento del modelo (sobre el conjunto de datos de testeo

descrito anteriormente) de ambos BETO por separado y en conjunto mediante *ensemble*.

	BETO *uncased*	**BETO *cased***	***Ensemble***
precision	0,814	0,823	0,824
recall	0,968	0,955	0,947
f1 score	0,884	0,884	0,881
auc	0,799	0,80	0,797

Tabla 5.3: Métricas de evaluación de cada modelo BETO independiente y del *ensemble* de los mismos

Se puede ver que las métricas son muy similares. Dado que entrenar un *ensemble* de modelos es mas costoso a nivel computacional y de tiempos, pues requiere entrenar y predecir los datos sobre cada modelo, no tiene sentido hacer este gasto cuando pueden obtenerse buenos resultados con un único modelo. Entre los dos modelos BETO podriamos usar uno u otro ya que tienen el mismo valor de *F1-score*. En un problema de detección de entidades, por ejemplo, $BETO_{cased}$ funcionaría mejor, dado que en español los nombres propios empiezan por mayúscula, y que el modelo haga esta distinción sería importante, además de que este modelo también detecta los acentos. Sin embargo en nuestro caso necesitamos diferenciar contextos, obtener patrones, y por tanto mientras el modelo comprenda el lenguaje, ambos serán igual de útiles. La ventaja de usar $BETO_{uncased}$ es que su vocabulario es más pequeño. Sin embargo, para líneas de trabajo futuras, nos decantamos por $BETO_{cased}$ ya que al diferenciar tanto minúsculas como mayúsculas puede ser de mayor utilidad identificando acrónimos. En la Tabla 5.4

aparecen un resumen de las métricas usando sólo este modelo con las condiciones definidas anteriormente.

	BETO *cased*
precision	0,823
recall	0,955
f1 score	0,884
auc	0,80

Tabla 5.4: Métricas de evaluación del modelo BETO *cased* sobre test

5.3.4 Configuración ganadora

Después de experimentar con las opciones anteriores, la configuración con mejores resutados es la que se muestra en la Tabla 5.5.

Modelo	**Hiperparámetros**	**Tratamiento de datos**	**Métricas**
BETO *cased*	• **Optimizador:** AdamW • **Epoch:** 8 • **Batch Size:** 32 • **Learning Rate:** $1 \cdot e^{-5}$ • **Maximum lenght:** 128	• Balanceado con textos de Medline • Normalización definiciones • Todos los acrónimos • *tokenizador*: $BETO_{cased}$	• **precision:** 0,823 • **recall:** 0,955 • **f1 score:** 0,884

Tabla 5.5: Configuración definitiva

Para el conjunto de datos utilizado, los mejores resultados obtenidos en la tarea

IberEval 2018 [5] fueron:

- Precision: 0.889 (Castaño et al., 2018)
- Recall: 0.811 (Sánchez-León, 2018)
- F1-score: 0.837 (Sánchez-León, 2018)

Comparando estos resultados con los obtenidos por el sistema propuesto se puede decir que nuestro modelo mejora las métricas de *recall* y *F1-score*.

5.3.5 Análisis de errores

En un primer momento, dado que el objetivo de este trabajo es detectar los acrónimos ambiguos, se eliminaron del conjunto de entrenamiento aquellos acrónimos con una única definición. Esto provocó un empeoramiento muy significativo de las predicciones del modelo, ya que en el conjunto de test había tanto acrónimos ambiguos como no ambiguos. Por esta razón, se decidió trabajar con todo el conjunto de acrónimos.

Por otro lado, se estudiaron en detalle aquellos acrónimos que el modelo no era capaz de predecir en el conjunto de *train* y se llegaron a dos conclusiones:

- La primera, es que fallaban aquellas acepciones del acrónimos que aparecían en test pero que no había visto en el conjunto de entrenamiento, lo que nos lleva a afirmar de que el modelo es muy dependiente del conjunto de datos de la fase de entrenamiento.
- Por otro lado, fallaban aquellos cuya definición estaba escrita de una manera en el conjunto de entrenamiento y otra en el conjunto de test. Cuando se

[5] https://temu.bsc.es/BARR2/

realizó la normalización de definiciones, se hizo de forma independiente para cada conjunto de datos. En un trabajo futuro se debería trabajar con acrónimos y definiciones unificadas. Por ejemplo para el acrónimo ALT, sus definición en train es "alanina aminotransferasa", mientras que en test esta es "alanina aminotransferar", es decir, la misma solo que no está lematizada. De modo que esto da lugar a una predicción errónea, cuando no es así. Ocurre lo mismo con las traducciones del español a inglés. Para el acrónimo AST, la definición en *train* es "aspartato aminotransferasa" y en test "aspartate aminotransferase".

- También ocurre que la definición es más completa en un conjunto de datos que en otro. Para el acrónimo C4 en *train* aparece como "complemento 4" y en test "cuarto componente complemento".

Por tanto, podemos ver que la mayoría de casos no detectados bien por el modelo se deben a errores humanos por parte de los expertos a la hora de asignar definiciones. Esto podría solventarse con un listado unificado de acrónimos y definiciones.

6. Conclusiones y líneas futuras

6.1 Conclusiones

La desambiguación de palabras para entender correctamente un texto en procesamiento del lenguaje natural sigue siendo todo un reto. Especialmente esto afecta a la literatura biomédica que se encuentra repleta de abreviaturas y acrónimos con más de un posible significado.

En este trabajo se ha presentado una propuesta de sistema para la desambiguación de acrónimos en literatura médica española. Tras la investigación teórica realizada en el Capítulo 2 se han podido tomar decisiones respecto al diseño del modelo. Debido a sus prometedores resultados en literatura inglesa biomédica, se ha optado por una arquitectura codificador-descodificador basada en *Transformers* cuyo codificador es BETO, un codificador $BERT_{based}$ entrenado con corpus en español, con una capa de clasificación y una función *softmax*. La clasificación ha sido binaria: la definición es la correcta o no para un acrónimo según el contexto en el que aparece.

La contribución de este trabajo es la desambiguación de acrónimos en **literatura médica española usando por primera vez *Transformers***, demostrando así que la aplicación en una lengua distinta al inglés ofrece también buenos resultados.

En los experimentos llevados a cabo para obtener la mejor configuración se estudió: el tratamiento previo de los datos, los hiperparámetros de la capa de clasificación y un posible *ensemble* de modelos tipo BERT.

Tras los mismos, se demostró que el sistema elegido da buenos resultados para la tarea a realizar. Respecto al *ensemble* de modelos, se estudiaron dos combinaciones posibles mediante un sistema de votos: BETO *cased*, BETO *uncased* y BERT *multilingual* y BETO *cased*, BETO *uncased*. En el primer caso se demostró que *multilingual* no daba buenos resultados, y este modelo fué descartado. En el segundo caso se observó esta combinación daba resultados similares a la que daban los modelos usados de forma independiente, por lo que se descartó el *ensemble*, ya que consume más recursos de computación y tiempos. Por otro lado, se vio la importancia del preprocesamiento de datos. La normalización de las definiciones de los acrónimos en el conjunto de entrenamiento, además de la adicción de más textos relativos a definiciones minoritarias para intentar balancear los datos, supusieron una mejora.

Comparando los resultados de este modelo con los trabajos que dieron los mejores resultados para el mismo conjunto de datos en la tarea IberEval 2018, se ha observado que el sistema propuesto mejora las métricas de *recall* y *F1-score*.

6.2 Líneas de trabajo futuras

Se ha visto como el contexto, la correcta normalización de las definiciones y el balanceo de las mismas es importante para un buen rendimiento del sistema definido. Por ello, a futuro, sería positivo realizar experimentos con un conjunto de datos mayor. Esto permitiría al modelo comprender más contextos y acrónimos.
Como se ha visto, algunas de las definiciones tenían muy pocos textos asociados a la misma. Esto dificulta al modelo poder detectar en qué tipo de contextos estas definiciones son las correctas, de modo que balancear estas definiciones más minoritarias ayudaría a mejorar la precisión del mismo.
Por otro lado, se vio que al asignar las definiciones diferentes expertos en ocasiones no estaba unificada la forma larga asignada: en ocasiones lematizada, en ocasiones en español en otras en inglés. Por este motivo sería útil poder estudiar otras técnicas de normalización, por ejemplo, utilizar un traductor de inglés a español que permita unificar aquellas definiciones que significan lo mismo.
En este trabajo se ha partido de la premisa de que el acrónimo ya estaba identificado y su definición asignada. Sin embargo, para ello ha sido necesario tener un conjunto de datos etiquetado por expertos. Para evitar tener que pasar por esta fase manual, sería muy interesante combinar el sistema definido para la desambiguación con un sistema capaz de identificar los acrónimos en un texto y sus respectivas definiciones.

Hemos visto que en la actualidad los *Transformers* entrenados en español son una minoría. Así como para el inglés hay mayor cantidad de modelos: ALBERT,

RoBERTA, ELECTRA... no es así para el español. El *ensemble* de modelos no funcionó correctamente. En (Zhu et al., 2021) utilizaron un *ensemble* formado por: RoBERTA, ALBERT, ELECTRA y $BERT_{LARGE}$ que dió muy buenos resultados. Por ello, si estos u otros codificadores se entrenan en un futuro con corpus en español, sería interesante probarlos sobre el conjunto de datos utilizado.

Bibliografía

R. Agerri and G. Rigau. Applying existing named entity taggers at barr ibereval 2017 task. *IberEval 2017*, 2017.

Alammar. The Illustrated Transformer. https://jalammar.github.io/illustrated-transformer/, 2018. *Alammar GitHub Blog*.

Aleixandre-Benavent, Alonso-Arroyo, González-Muñoz, and J. de Dios. Scientific communication (xxiii). medical language (1): The use and abuse of abbreviations and acronyms in the medical language and in pediatrics. *Acta Pediatrica Espanola*, 73:134–140, 05 2015.

R. Argwal. Understanding Transformers, the Data Science Way. https://www.kdnuggets.com/2020/10/understanding-transformers-data-science-way.html, 2020. *KDnuggets Blog*.

Bert. Bert Multilingual Repository. https://github.com/google-research/bert/blob/master/multilingual.md, 2019.

J. Castaño, P. Avila, D. Pérez, H. Berinsky, H. Parkm, L. Gambarte, and D. Luna. A simple approach to abbreviation resolution at barr2, ibereval 2018. *IberEval 2018*, 2018.

J. Cañete, G. Chaperon, R. Fuentes, and J. Pérez. Spanish pre-trained bert model and evaluation data, 2020.

J. Charbonnier and C. Wartena. Using word embeddings for unsupervised acronym disambiguation. 10 2018.

K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, Oct. 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. URL https://aclanthology.org/D14-1179.

M. Ciosici, T. Sommer, and I. Assent. Unsupervised abbreviation disambiguation contextual disambiguation using word embeddings. *ArXiv*, abs/1904.00929, 2019.

Cuadros, Perez, Montoya, and García-Pablos. Vicomtech at barr2: Detecting biomedical abbreviations with ml methods and dictionary-based heuristics. *IberEval 2018*, 10 2018.

H. Dai, P.-T. Lai, Y.-C. Chang, and R. T.-H. Tsai. Enhancing of chemical compound and drug name recognition using representative tag scheme and fine-grained

tokenization. *Journal of cheminformatics*, 7:S14, 01 2015. doi: 10.1186/1758-2946-7-S1-S14.

D. Dale. A machine learning model to understand fancy abbreviations, trained on tolkien. *Medium*, 2018.

J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

X. Du, R. Zhu, Y. Li, and A. Anjum. Language model-based automatic prefix abbreviation expansion method for biomedical big data analysis. *Future Generation Computer Systems*, 98:238–251, 2019. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2019.01.016. URL https://www.sciencedirect.com/science/article/pii/S0167739X18326529.

G. P. Finley, S. V. S. Pakhomov, R. McEwan, and G. B. Melton. Towards comprehensive clinical abbreviation disambiguation using machine-labeled training data. *AMIA ... Annual Symposium proceedings. AMIA Symposium*, 2016:560–569, 2016.

A. Intxaurrondo and M. Krallinger. Cnio al barr ibereval 2017: exploring three biomedical abbreviation identifiers for spanish biomedical publications. *IberEval 2017*, 2017.

A. Intxaurrondo, J. C. de la Torre, H. Rodríguez, M. Marimon, J. A. Lopez-Martín, A. Gonzalez-Aguirre, J. Santamaría, M. Villegas, and M. Krallinger. Resources,

guidelines and annotations for the recognition, definition resolution and concept normalization of spanish clinical abbreviations: the barr2 corpus. *IberEval 2018*, 2018.

D. Jurafsky and J. H. Martin. *Speech and Language Processing (2nd Edition)*. Prentice-Hall, Inc., USA, 2009. ISBN 0131873210.

R. Kai and W. Shi-Wen. Applying convolutional neural network model and auto - expanded corpus to biomedical abbreviation disambiguation. *Journal of Engineering Science and Technology Review*, 9:178–184, 12 2016. doi: 10.25103/jestr.096.27.

A. Kashyap, H. Burris, C. Callison-Burch, and M. R. Boland. The classe gator (clinical acronym sense disambiguator): A method for predicting acronym sense from neonatal clinical notes. *International Journal of Medical Informatics*, 137: 104101, 02 2020. doi: 10.1016/j.ijmedinf.2020.104101.

K. KS and Sangeetha. SECNLP: A survey of embeddings in clinical natural language processing. *CoRR*, abs/1903.01039, 2019. URL http://arxiv.org/abs/1903.01039.

Larkey, Ogilvie, Price, and B. Tamilio. Acrophile: an automated acronym extractor and server. In *DL '00*, 2000.

Q. Le and T. Mikolov. Distributed representations of sentences and documents. *31st International Conference on Machine Learning, ICML 2014*, 4, 05 2014a.

Q. Le and T. Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014b. URL http://arxiv.org/abs/1405.4053.

C. Li, L. Ji, and J. Yan. Acronym disambiguation using word embedding. 2015.

I. Li, M. Yasunaga, M. Y. Nuzumlali, C. Caraballo, S. Mahajan, H. Krumholz, and D. Radev. A neural topic-attention model for medical term abbreviation disambiguation. *CoRR*, abs/1910.14076, 2019. URL http://arxiv.org/abs/1910.14076.

H. Liu, Y. Lussier, and C. Friedman. A study of abbreviations in the umls. *Proceedings / AMIA ... Annual Symposium. AMIA Symposium*, pages 393–7, 02 2001.

Menaha and Jayanthi. A hybrid model for finding abbreviation–definition pairs from biomedical abstracts using heuristics-based sequence labeling and perceptron linear classifier. *Expert Systems with Applications*, 166:114049, 2020. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2020.114049.

T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 26, 10 2013.

S. Montalvo, M. Oronoz, H. Rodriguez, and R. Martinez. Biomedical abbreviation recognition and resolution by prosa med. *IberEval2017*, 2017.

S. Montalvo, R. Martinez, M. Almagro, and S. Lorenzo. Mamtra-med at biomedical abbreviation recognition and resolution ibereval 2018. *IberEval 2018*, 2018.

D. Nadeau and P. Turney. A supervised learning approach to acronym identification. *Proceedings of 8th Canadian Conference on Artificial Intelligence (AI'2005), Volume LNCS*, 3501, 05 2005. doi: 10.1007/11424918_34.

Nithyashri, Piruthika, Kanamani, and Menaha. Web-based acronym expansion disambiguation using word2vec and doc2vec modeling. *International Journal of Research in Engineering, Science and Managemen*, 3:265–270, 05 2020.

S. Pakhomov, T. Pedersen, and C. Chute. Abbreviation and acronym disambiguation in clinical discourse. *AMIA ... Annual Symposium proceedings / AMIA Symposium. AMIA Symposium*, 2005:589–93, 02 2005.

C. Pan, B. Song, S. Wang, and Z. Luo. Bert-based acronym disambiguation with multiple training strategies, 2021.

M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. 02 2018.

A. Pomares-Quimbaya, P. Úbeda, M. Oleynik, and S. Schulz. Leveraging pubmed to create a specialty-based sense inventory for spanish acronym resolution. *Studies in health technology and informatics*, 270:292–296, 06 2020. doi: 10.3233/SHTI200169.

Premkumar, Atchayaa, Idayavalli, and Gayathri. Acronym disambiguation using web scraping. *International Journal og Engineering and Advanced Technology (IJEAT)*, 9, 04 2020. doi: 10.35940/ijeat.D6812.049420.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.

M. S. Z. Rizvi. Demystifying BERT: A Comprehensive Guide to the Groundbreaking NLP Framework. https://www.analyticsvidhya.com/blog/2019/09/demystifying-bert-groundbreaking-nlp-framework/, 2019. *Analytics Vidhya Blog*.

F. Ronzano and L. Furlong. Ibi-upf at barr-2017: learning to identify abbreviations in biomedical literature. *IberEval 2017*, 2017.

I. Rubio-López, R. Costumero, H. Ambit, C. Gonzalo-Martin, E. Menasalvas, and A. González. Acronym disambiguation in spanish electronic health narratives using machine learning techniques. *Studies in Health Technology and Informatics*, 235:251–255, 01 2017. doi: 10.3233/978-1-61499-753-5-251.

A. Sabbir, A. Jimeno-Yepes, and R. Kavuluru. Knowledge-based biomedical word sense disambiguatio with neural concept embeddings. volume 2017, pages 163–170, 10 2017. doi: 10.1109/BIBE.2017.00-61.

C. Sanchez and P. Martínez. A proposed system to identify and extract abbreviation definitions in spanish biomedical texts for the biomedical abbreviation recognition and resolution. *IberEval 2017*, 2017.

A. Schwartz and M. Hearst. A simple algorithm for identifying abbreviation definitions in biomedical text. *Pacific Symposium on Biocomputing.*

Pacific Symposium on Biocomputing, 4:451–62, 02 2003. doi: 10.1142/9789812776303_0042.

M. Skreta, A. Arbabi, J. Wang, and M. Brudno. Training without training data: Improving the generalizability of automated medical abbreviation disambiguation. *ArXiv*, abs/1912.06174, 2019.

I. Sutskever, O. Vinyals, and Q. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014. URL http://arxiv.org/abs/1409.3215.

C. Sánchez and P. Martínez. A simple method to extract abbreviations within a document usaing regular expressions. *IberEval 2018*, 2018.

F. Sánchez-León. Arborex: Abbreviation resolution based on regular expressions for barr2. *IberEval 2018*, 2018.

A. Thakker, S. Barot, and S. Bagul. Acronym expansion: A domain independent approach. 11 2017.

S. Tulkens, S. Šuster, and W. Daelemans. Using distributed representations to disambiguate biomedical and clinical concepts. pages 77–82, 08 2016. doi: 10.18653/v1/W16-2910.

I. Unanue, E. Borzeshi, and M. Piccardi. Recurrent neural networks with specialized word embeddings for health-domain named-entity recognition. *Journal of Biomedical Informatics*, 76, 06 2017. doi: 10.1016/j.jbi.2017.11.007.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. 2017.

T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020.

Y. Wu, J. Xu, Y. Zhang, and H. Xu. Clinical abbreviation disambiguation using neural word embeddings. pages 171–176, 01 2015. doi: 10.18653/v1/W15-3822.

W. Zhang, Y. Sim, S. Jian, and C. L. Tan. Entity linking with effective acronym expansion, instance selection and topic modeling. pages 1909–1914, 01 2011. doi: 10.5591/978-1-57735-516-8/IJCAI11-319.

D. Zhu, W. Lin, Y. Zhang, Q. Zhong, G. Zeng, W. Wu, and J. Tang. At-bert: Adversarial training bert for acronym identification winning solution for sdu@aaai-21, 2021.

A. Apéndice A: Código

En este apéndice se describirá el código generado para este proyecto, describiendo cada archivo. Todo el código está escrito en lenguaje Python. El repositorio del mismo puede encontrarse en https://github.com/mariaega11/acronym_disambiguation_tfm.

A.1 Data

En este directorio se almacenan todos los datos para la ejecución del sistema. Contiene tres carpetas:

- *ibereval_data*: conjunto de datos para entrenamiento y testeo originales de IberEval (https://temu.bsc.es/BARR2/).
- *scrapping*: textos médicos obtenidos mediante *scrapping web* de Medline (https://medlineplus.gov/spanish/), tanto por búsqueda de concepto como búsqueda por letra del abecedario.
- *data_train*: datasets de entrenamiento y testeo modificados tras el preprocesamiento para poder funcionar como *input* al modelo.
- *data_predict*: salida con las predicciones del modelo.

A.2 Preprocessing

Directorio con los *scripts* necesarios para el preprocesado de los datos. Contiene dos archivos:

- *1_scrapping.ipynb*: su ejecución permite obtener información de la web de Medline buscando tanto por concepto como por letra.
- *2_preproces_input.ipynb*: *script* para ejecutar el preprocesado de los datos explicado en la sección 4.2 tanto del conjunto de entrenamiento como el de test.

A.3 Analytics

Directorio con el *script* necesario para la ejecución del modelo. Contiene el archivo:

- *3_model.ipynb*: ejecuta el modelo analítico descrito en esta memoria.