UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Modelos contextuales e incontextuales de palabras para Twitter

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Jose Arias Moncho

Tutor: Lluís Felip Hurtado, José Ángel González

Curso 2019-2020

Resum

En aquest treball es tracta amb models incontextuals i contextuals per a generar representacions vectorials contínues de paraules i utilitzar-les en tasques de processament del llenguatge natural. En concret, s'estudia l'entrenament i l'aplicació de models Word2Vec i TWilBERT a l'anàlisi de sentiments i a la detecció d'emocions. Per a validar els resultats de tots dos enfocaments, s'ha participat en el "Taller de Análisis Semántico de la Sociedad Española de Procesamiento del Lenguaje Natural", comparant aquests resultats amb els obtinguts per altres universitats i equips d'investigació.

Finalment, cal destacar la importància de les comparatives realitzades en aquest treball respecte als dos tipus de models utilitzats, mostrant les diferències en la potència i també la diferència que existeix entre les tasques de processament del llenguatge natural, de tal forma que els models competitius en una tasca poden no ser-ho tant en una altra.

Paraules clau: Twitter, detecció d'emocions, anàlisi de sentiments, word embeddings incontextuals, word embeddings contextuals

Resumen

En este trabajo se exploran modelos incontextuales y contextuales para generar representaciones vectoriales continuas de palabras y utilizarlas en tareas de procesamiento del lenguaje natural. En concreto, se estudia el entrenamiento y la aplicación de modelos Word2Vec y TWilBERT al análisis de sentimientos y a la detección de emociones. Para validar los resultados de ambos enfoques, se ha participado en el Taller de Análisis Semántico de la Sociedad Española de Procesamiento del Lenguaje Natural, comparando dichos resultados con los obtenidos por otras universidades y equipos de investigación.

Por último, cabe destacar la importancia de las comparativas realizadas en este trabajo con respecto a los dos tipos de modelos utilizados, mostrando las diferencias en la potencia de ambos y también lo distintas que son las tareas de procesamiento del lenguaje natural entre ellas, de tal forma que los modelos competitivos en una tarea pueden no serlo tanto en otra.

Palabras clave: Twitter, detección de emociones, análisis de sentimientos, word embeddings incontextuales, word embeddings contextuales

Abstract

In this work we explore context-free and contextualized models to generate continuous vector representations of words and use them in Natural Language Processing tasks. Specifically, we study the training and the application of Word2Vec and TWilBERT models to sentiment analysis and emotion detection. To validate the results of both approaches, we participated in the "Taller de Análisis Semántico de la Sociedad Española de Procesamiento del Lenguaje Natural", comparing our results with the ones obtained from another universities and investigation groups.

At last, we have to remark the importance of the comparisons made in this work regarding to the two type of models, showing the difference in capacity and also the difference in the Natural Language Processing tasks, meaning that competitive models for a given task can perform poorly in another one.

Key words: Twitter, emotion detection, sentiment analysis, context-free word embeddings, contextual word embeddings

Índice general

Apéndices

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

El procesamiento del lenguaje natural (PLN) o *NLP* en inglés es un campo de investigación que lleva explorándose desde hace muchos años. Sin embargo, el uso de modelos neuronales dentro del campo es bastante reciente, remontándose al año 2001. Desde ese momento hemos visto como se han ido sucediendo diferentes cambios y mejoras a los modelos hasta que recientemente ha habido una explosión. Este campo dentro de la Inteligencia Artificial (IA) está en el punto de mira actualmente gracias a los avances producidos por los modelos de lenguaje basados en la arquitectura *Transformer*. Algunos ejemplos son: GPT-3 (OpenAI) o los *Bidirectional Encoder Representations from Transformers* (*BERT*), que explicaremos más adelante. Todos estos avances han propiciado que aparezcan aplicaciones como es el *chatbot* Blender [1] en Abril de este año. Así, Facebook presenta en su trabajo a Blender, un *chatbot* de una calidad muy superior a todos sus predecesores (ver Figura 1.1), pudiendo hablar de una gran variedad de temas y no solo eso, sino aceptando como entrada que nosotros le indiquemos como tiene que comportarse e indicándole su personalidad, como por ejemplo que sea una persona con una pasión desmedida por el ajedrez. Este *chatbot* se basa en modelos de lenguaje como los que indicaremos más adelante, no en una base de datos con frases prediseñadas como se venía utilizando hasta este momento.

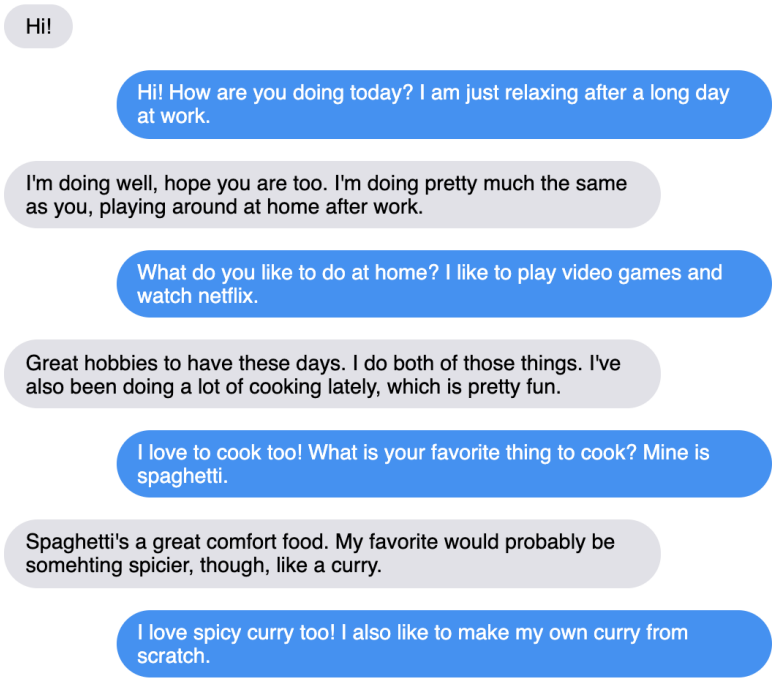

Figura 1.1: Conversación de un humano (izquierda) con Blender (derecha). Fuente.

Tras este *chatbot* y un mes más tarde, se publica en Mayo GPT-3 [2] , un modelo creado por OpenAI y que cuenta con 175 mil millones de parámetros. GPT-3 es un modelo de lenguaje entrenado capaz de generar texto muy realista, como podemos ver en la Figura 1.2, hacer traducción automática y muchas otras tareas gracias a aportar al modelo un contexto inicial como OpenAI indica en su trabajo.

Title: United Methodists Agree to Historic Split
Subtitle: Those who oppose gay marriage will form their own denomination
Article: After two days of intense debate, the United Methodist Church has agreed to a historic split - one that is expected to end in the creation of a new denomination, one that will be "theologically and socially conservative," according to The Washington Post. The majority of delegates attending the church's annual General Conference in May voted to strengthen a ban on the ordination of LGBTQ clergy and to write new rules that will "discipline" clergy who officiate at same-sex weddings. But those who opposed these measures have a new plan: They say they will form a separate denomination by 2020, calling their church the Christian Methodist denomination.
The Post notes that the denomination, which claims 12.5 million members, was in the early 20th century the "largest Protestant denomination in the U.S.," but that it has been shrinking in recent decades. The new split will be the second in the church's history. The first occurred in 1968, when roughly 10 percent of the denomination left to form the Evangelical United Brethren Church. The Post notes that the proposed split "comes at a critical time for the church, which has been losing members for years," which has been "pushed toward the brink of a schism over the role of LGBTQ people in the church." Gay marriage is not the only issue that has divided the church. In 2016, the denomination was split over ordination of transgender clergy, with the North Pacific regional conference voting to ban them from serving as clergy, and the South Pacific regional conference voting to allow them.

Figura 1.2: Noticia generada por GPT-3 incluida en su trabajo. Fuente.

Siguiendo con este ritmo, en Junio de 2020 OpenAI presenta un prototipo de consola semántica, de tal forma que el usuario puede indicarle en lenguaje natural qué quiere hacer y la consola le propone los comandos que tiene que ejecutar para dicho objetivo. Y no sólo esto, este mismo mes Facebook presenta también en [3] un sistema no supervisado capaz de traducir código de Java, C++ o Python entre ellos.

Todas estas aplicaciones nos muestran lo que estamos viviendo, un tiempo sin precedentes para las tareas del *NLP* en el que cada vez se ve implicada una mayor parte de la sociedad. Todos estos cambios se ven potenciados por el hecho de que hoy en día se genera una cantidad masiva de datos de forma diaria en las diferentes redes sociales, lo que está consiguiendo que haya muchos datos para entrenar a estos modelos con tantos millones de parámetros y que se apliquen en las ideas anteriormente presentadas. Y no sólo están estas aplicaciones, sino que existen muchas otras que van a ir surgiendo, esto es simplemente el comienzo.

Dentro de todas estas tareas relacionadas con el lenguaje natural, nosotros nos centraremos en el análisis de sentimientos y la detección de emociones, un campo que interesa a una gran cantidad de organismos con una motivación que va desde económica hasta científica. Las empresas buscan obtener información acerca de la población para así poder medir sus decisiones en función de cuánto agradan a su público o cuanto atraen a sus posibles compradores. Los científicos quieren investigar este campo para obtener modelos que sean capaces de distinguir las emociones de un texto tal y como lo haría un humano. Y por último los gobiernos quieren fomentar la investigación, apoyándola con la concesión de diferentes proyectos como en este caso es AMIC [4] [5], un proyecto coordinado denominado Análisis Afectivo de Información Multimedia con Comunicación Inclusiva

y Natural del que son partícipes las personas implicadas en el grupo de investigación Enginyeria del Llenguatge i Reconeixement de Formes (ELiRF), en el que se ha desarrollado este trabajo. Dentro del proyecto se abordan tareas del lenguaje natural como son:

- Procesado de Audio, Habla, y Lenguaje para análisis de Información Multimedia
- Tecnología para la interacción conversacional persona-máquina con aprendizaje dinámico
- Sistemas basados en la interacción oral dinámicamente mejorables y adaptables a nuevos contextos
- Métodos de Aprendizaje para Minería de Textos en Dominios Específicos
- Comprensión del habla y diálogo: estudio de aproximaciones basadas en aprendizaje automático para la adaptación a nuevos dominios.

Tras esto se ha tratado de informar al lector sobre lo importante que es y será este campo de investigación para la sociedad en un futuro. En las siguientes secciones procederemos a introducir la motivación que ha propiciado la realización de este trabajo, los objetivos que se buscaba conseguir y por último una estructuración del resto de puntos que formarán esta memoria. Por último, se ha incluido un glosario de términos relevantes en relación a este trabajo.

1.1 Motivación

En este punto hablaré de mi motivación a la hora de optar por este trabajo y cómo me interesó cuando solo tenía una idea vaga del proyecto, pero también hablaré de cómo me ha impulsado durante toda la etapa de trabajo para realizar un buen trabajo.

Como estudiante en computación y también como individuo, el estudio de los datos y tareas de procesamiento del lenguaje natural a través de herramientas de machine learning es un tema que considero muy relevante dentro de la investigación actual, estando en pleno auge y desarrollo, además de ser vital en la situación actual que vivimos y primordial en el avance y desarrollo de las sociedades modernas. Por otra parte, me parece una de las áreas más interesantes de la informática debido a mi experiencia personal en la carrera con las diferentes asignaturas, además de considerarlo un tema preocupante de cara al futuro por los posibles usos que se pueden hacer con tanta información que se puede obtener de cada uno de nosotros de forma individual.

Como profesional, siento que este trabajo se acerca mucho a las salidas profesionales que más me interesan y deseaba obtener más experiencia en el campo del aprendizaje automático y el lenguaje natural para poder considerar que mi formación en la titulación ha sido totalmente plena.

Cuando hablé con mis cotutores, ambos profesionales que me han impartido las asignaturas que he considerado más interesantes a lo largo de la carrera y considero personas de las que puedo aprender a nivel académico y profesional acerca de esta tema, decidí optar por esta opción, ya que me parece un lujo tanto poder trabajar con Twitter como poder hacerlo con estos cotutores. De forma personal y tras este proyecto, considero firmemente que esta es también la mejor red social pública para estudiar las opiniones y la forma de expresarse de la gente.

Conforme se fue desarrollando el proyecto, surgió la posibilidad de utilizar mi trabajo con la ayuda de mis cotutores para medir los resultados en una competición a nivel

internacional como es el Taller de Análisis Semántico en la SEPLN (TASS), lo me ha parecido un aliciente muy inspirador y motivador para esforzarse buscando obtener buenos resultados.

1.2 Objetivos

En este punto vamos a introducir los objetivos que nos planteamos al principio del proyecto, desarrollando un poco el contexto en el que se enmarcan. En este trabajo se requería trabajar con una gran cantidad de datos, que consideramos es uno de los puntos positivos que ofrece Twitter al tratar con una cantidad de datos tan grandes. De esta forma, nuestra atención se centraba en utilizar muchos datos para entrenar modelos con muchos millones de parámetros, siendo antes necesario evaluar la calidad de los datos mediante modelos más sencillos. Por este motivo, nuestros objetivos se basan en este mismo concepto, teniendo:

- En primer lugar, queremos obtener datos (*tweets*) de forma continuada a partir de dejar un *crawler* durante la realización del trabajo para obtener la mayor cantidad de tweets posibles.
- En segundo lugar, queremos realizar una comparativa entre un modelo contextual y otro incontextual y observar los diferentes resultados que ambas obtienen. Esperamos que un modelo contextual obtenga mejores resultados al ser modelos mucho más grandes y que pueden expresar mejor el lenguaje si son entrenados con datos de calidad.
- A continuación y en tercer lugar, queremos también hacer otra comparativa cambiando la cantidad de datos con la que se entrena a un modelo incontextual para ver cómo afecta el número de datos a estos modelos. Por ello, compararemos un modelo entrenado con los tweets que tenemos de forma inicial (70M) con otro modelo que utilice una cantidad de tweets que consideremos adecuada según los plazos de tiempo que tengamos.
- Finalmente, en cuarto lugar, y como objetivo personal, se quiere familiarizarse con las tareas del *NLP*, aprender a trabajar con modelos incontextuales más sencillos y trabajar aunque sea de forma escasa con modelos más complejos y actuales como son los modelos contextuales.

1.3 Estructura de la memoria

Como ya hemos indicado anteriormente, procedemos a introducir los diversos capítulos que forman este trabajo. Entre ellos, la memoria se estructura de la siguiente manera:

- **Capítulo 1, Introducción:** En este capítulo se introduce este trabajo, la motivación que ha impulsado su elección y realización, se establece que estructura tendrá, cómo se relaciona con las diferentes asignaturas y se proporciona una definición de conceptos específicos y técnicos de la especialización y de Twitter.
- **Capítulo 2, Estado del arte:** En el segundo capítulo hablaremos de la evolución de los modelos neuronales aplicados a tareas del lenguaje natural, introduciendo los avances que nos han llevado a las tecnologías y técnicas actuales que hacen posibles aplicaciones como las que se han presentado al comienzo de esta memoria.

- **Capítulo 3, Redes neuronales y *word embeddings*:** En el tercer capítulo se introduce todos los conceptos teóricos relevantes a las redes neuronales y todo lo que las envuelve, los *word embeddings* y los diferentes modelos contextuales e incontextuales que se consideran relevantes así como también se explican las *Deep Averaging Networks* (*DAN*) debido a que posteriormente las utilizaremos de forma práctica.
- **Capítulo 4, Herramientas y metodología:** En este capítulo se explican los elementos fundamentales en relación al trabajo realizado así cómo la forma en que se ha procedido en relación al tratamiento de datos y las herramientas con que se han diseñado los modelos. Por último se ha comentado el uso de recursos computacionales.
- **Capítulo 5, Trabajo experimental:** En este quinto capítulo se ha establecido el trabajo realizado, dividiéndolo en las dos tareas con las que se ha tratado. En cada una de ellas se ha comentado el estado del arte, la distribución de los datos, la metodología en la fase de entrenamiento y las comparativas marcadas en el apartado de objetivos ya introducido previamente.
- **Capítulo 6, Conclusiones y trabajo futuro:** Finalmente, en este último capítulo se comenta las dificultades observadas a lo largo de la realización del trabajo. Además, se relaciona el trabajo con los objetivos establecidos en un principio y finalmente se presentan posibles líneas de estudio para una continuación de este trabajo.

1.4 Asignaturas relacionadas

En este punto vamos a introducir una correlación entre el trabajo realizado y las diferentes asignaturas cursadas en el grado de Ingeniería Informática y que han sido especialmente útiles para la realización del presente trabajo.

En primer lugar, cabe destacar la relevancia de la asignatura de "Base de datos y sistemas de información", en la cual se introduce el concepto de base de datos así como las técnicas y metodologías para trabajar con estas a la hora de encontrar información relevante en bases de datos relacionales, concretamente SQL, de tal forma que este conocimiento lo hemos aplicado a una base de datos no relacional con bastante facilidad.

Después están las asignaturas conocidas como "Percepción" y "Aprendizaje automático", que juntas introducen las redes neuronales y diferentes técnicas después de haber introducido conceptos más básicos y que han sido una base teórica imprescindible para la realización de este trabajo.

Por último, la asignatura "Sistemas de almacenamiento y recuperación de información" nos expone de forma básica el concepto de corpus, trabajar con ellos, cómo almacenar la información de estos, tratar a las palabras como vectores continuos, el preprocesado de texto y el *web crawling*, lo que convierte a esta asignatura en la asignatura más cercana y relevante a este trabajo.

1.5 Glosario de términos

- *Stopword*: Son palabras muy frecuentes, utilizadas en un lenguaje, que generalmente no aportan ninguna información semántica como por ejemplo serían las preposiciones "a", "en" o "de", los determinantes "el", "la", "los", "las", etcétera.
- Bolsa de palabras: Modelo muy utilizado de forma histórica en muchos campos como recuperación de información (*information retrieval*) en que las frases se representan por el número de apariciones de las palabras en la frase.
- Corpus: Colección o conjunto de datos específicos sobre un tema concreto.
- *Crawler*: Programa utilizado para obtener datos de forma automática de una página web concreta basándose en la estructura de esta.
- Regresión: En estadística, se entiende como el proceso que estudia y estima la relación entre variables.
- Regresión logística: Tipo de análisis de regresión que consiste en predecir el resultado de una variable categórica, es decir, una variable que puede adoptar un número ilimitado de categorías, en función de las variables independientes o predictoras.
- N-grama: Secuencia continua de n elementos de un dato o texto. Estos elementos pueden ser sílabas, letras, palabras u otros elementos que se deseen utilizar según el objetivo deseado.
- Distancia de Levenshtein: Medida de distancia entre dos secuencias en que se considera la mínima cantidad de cambios (inserciones, eliminaciones y sustituciones) de carácteres necesarias para que las secuencias sean iguales.
- *Token*: Símbolo perteneciente a una oración o texto, tras haber separado este aplicándole un preproceso concreto.
- *Backend*: Parte de un sistema o programa informático no accesible de forma directa por parte del usuario.
- *JSON*: Formato específico de ficheros que suele poder comprenderse y utilizado para intercambiar o transmitir información.
- *API*: Conocida como Application Programming Interface, este concepto hace referencia el conjunto de variables y funciones que una aplicación o librería muestra a los usuarios.
- *Tweet*: Mensaje de máximo 280 caracteres escrito en una cuenta de Twitter por un usuario y que define la identidad de Twitter como plataforma.
- Usuario: Propietario de la cuenta de Twitter al que puede referenciarse por su nombre con una "@" seguida de su nombre de usuario.
- *Hashtag*: Elemento que se añade a los *tweets* para clasificarlos en cierta categoría y que pueden ser buscados por esta. Se utilizan con un "#" seguido del nombre del tema.
- Me gusta: Acción con la que indicas que te ha gustado un *tweet*, el cual se añadirá a tu lista de me gustas que puede verse en tu perfil.
- *Retweet*: Acción que puedes realizar por la que muestras a tus seguidores el *tweet* del usuario que te interesa.

CAPÍTULO 2
Estado del arte

En este capítulo vamos a introducir los diferentes cambios que se han dado a lo largo de estos últimos años y cómo nos han llevado a las técnicas con redes neuronales utilizadas actualmente. Además, revisaremos las principales tareas de procesamiento del lenguaje natural y los resultados que se pueden observar hoy en día. Este apartado utiliza el post [6] de Sebastian Ruder en su blog como hilo conductor debido a que considero que es una aproximación bastante fiel a la historia neuronal de los modelos y tareas de *NLP*.

Este campo de la investigación es bastante actual y data del 2001, año en el que surge la primera estructura neuronal para una tarea de *NLP* conocida como *Language Modeling* (*LM*). Esta tarea consiste en predecir que palabra será la siguiente, dadas todas las palabras anteriores. Para ello, Yoshua Bengio propuso en [7] una red neuronal *feed-forward* que podemos observar en la Figura 2.1, en la que el modelo recibe un vector de entrada con las n palabras previas y miraba en una tabla C que mapea las palabras del vocabulario con un vector de reales $C(i) \in \mathbb{R}^m$. Estos vectores de reales que se obtienen de la tabla o matriz C son lo que actualmente se conocen como *word embeddings*.

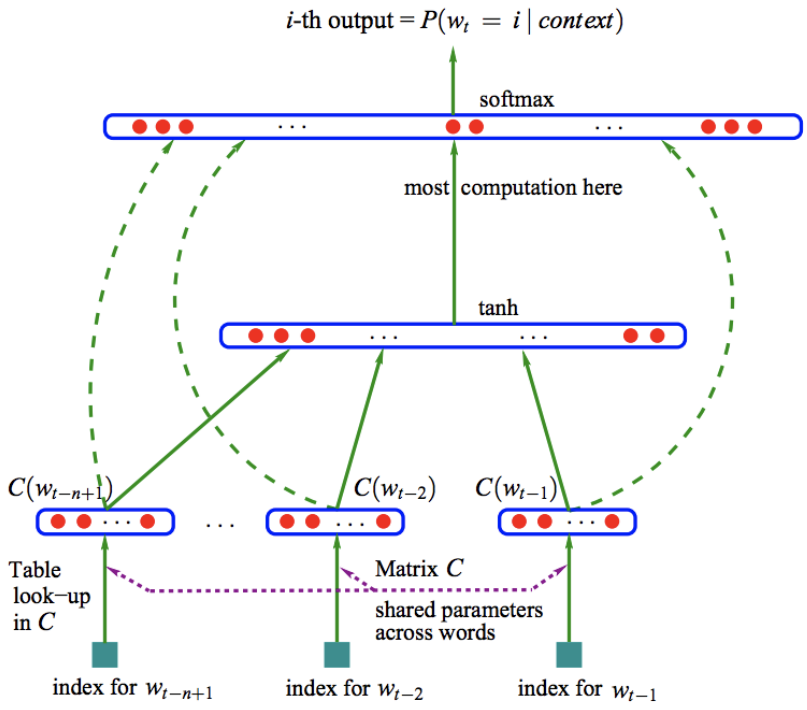

Figura 2.1: Arquitectura utilizada por Bengio en su trabajo. Fuente.

Este modelo propuesto por Bengio en 2001 fue bastante relevante por los buenos resultados que obtuvo en su momento, superando a un modelo basado en trigramas en un 35 % utilizando el Brown corpus y en un 20 % con el Hansard corpus.

Estas mejoras que obtuvo el modelo se midieron utilizando la Perplejidad (PPL), que podemos observar en la Fórmula 2.1.

$$PPL = \frac{1}{\hat{P}(w_t \mid w_1^{t-1})} \tag{2.1}$$

Posteriormente, ya en 2008, se empezó a centrar la atención más en el aprendizaje multi-tarea, intentando entrenar modelos de forma general, sin centrarse en tareas de *NLP* específicas para así aprender representaciones que fueran generales y aplicables a una gran parte de tareas de *NLP*. Es por tanto que en este año Collobert y Weston utilizan en su trabajo [8] una misma tabla de búsqueda C para dos tareas diferentes como podemos observar en la Figura 2.2. También indican las diferentes tareas de *NLP* que ellos consideran, separándolas en seis y luego entrenan a sus modelos en estas tareas que han indicado para posteriormente comprobar como mejora el modelo tras haber sido entrenado en diferentes combinaciones, con sus mejores modelos obteniendo resultados en el corpus de PropBank que superaban a 9 de los 11 modelos de la época.

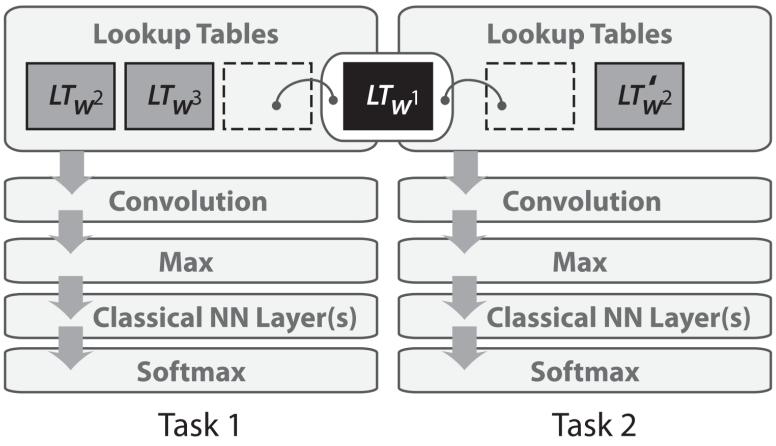

Figura 2.2: Uso de la misma tabla LT_{w^1} en dos tareas distintas. Fuente.

Un poco más tarde, más concretamente en 2010, Mikolov propuso en [9] utilizar redes neuronales recurrentes (*RNN* en inglés) en vez de redes neuronales *feed-forward* con el fin de no tener un tamaño fijo de palabras previas establecido como parámetro de entrenamiento, sino que se podía trabajar con un contexto de tamaño variable debido a las conexiones recurrentes que forman ciclos en la red neuronal recurrente. Debido a la complejidad que las *RNN* pueden tener, Mikolov utilizó en su trabajo una red Elman (Figura 2.3), la estructura más simple que puede utilizarse.

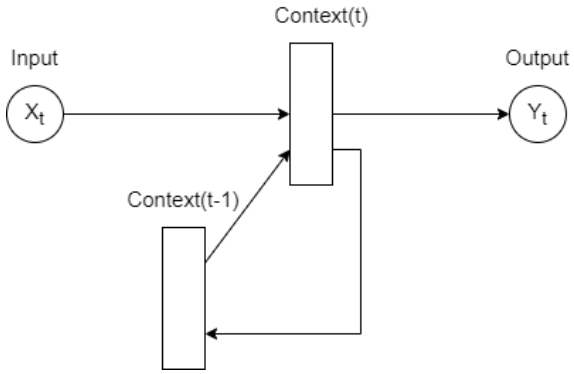

Figura 2.3: Esquema de una *RNN* simple o red Elman.

Los resultados supusieron una mejora de casi el 50 % en PPL comparado con un KN5 (5-grama con un suavizado Kneser-Ney) y también mejoró los resultados en un 12 % al compararse con modelos backoff entrenados con 5 veces más palabras en reconocimiento del habla para el dataset del DARPA WSJ.

Esta mejora del 12 % se refiere al *Word error rate* (*WER*), que se deriva de la distancia de Levenshtein. Podemos observar cómo se calcula el *WER* en la Fórmula 2.2.

$$WER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C} \tag{2.2}$$

donde S es el número de sustituciones, D es el número de borrados, I el número de inserciones, C el número de palabras correctas y N es el número de palabras en la referencia.

Unos años más tarde, en 2013, se producen grandes avances en este campo. En primer lugar aparece el uso de las *Long Short-term memory* (*LSTM*) [10] en *LM* gracias al trabajo de Graves en [11], que son modelos basados en *RNN* que añaden células de memoria (Figura 2.4) para mejorar el problema del ruido cuando hay mucho contexto de por medio entre dos palabras relevantes. Este problema del ruido dentro de la tarea de *LM* en teoría no debería existir para una *RNN* debido a sus conexiones recurrentes, pero como ya había indicado Yoshua Bengio y sus compañeros en 1994 en [12], en la práctica este problema sí se da, y por ello, aparece el uso de estas *LSTM* que solucionan el problema.

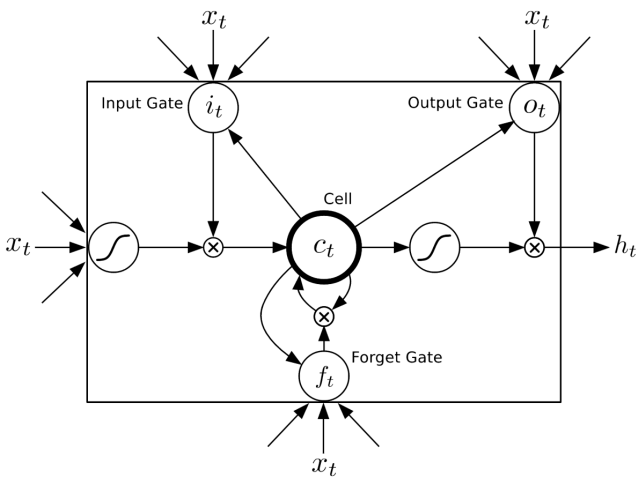

Figura 2.4: Esquema de una célula de memoria. Fuente.

Añadido a lo anterior, en 2013 también apareció lo que se conoce como *word embeddings*. Desde el 2001 y hasta este año, ya se habían usado vectores densos con valores reales para representar palabras en una gran cantidad de modelos. Sin embargo, fue con la publicación de Mikolov en [13] y en [14], donde él y sus compañeros decidieron eliminar la capa oculta que se venía utilizando en todos los modelos anteriores y crearon dos modelos (*CBOW* y *Skip-gram*) con nuevos objetivos de entrenamiento. La eliminación de la capa oculta permitió reducir el tiempo de entrenamiento del modelo de forma drástica, lo que llevó a que se entrenase estos modelos con muchos más datos. Esta posibilidad de entrenar con muchos más datos permitió que se pudieran obtener mejores resultados de los que había hasta ese momento.

Por último, en este año se empezaron a fijar los modelos neuronales como el estándar para tareas de *NLP* en contraposición a otros que se habían utilizado tradicionalmente como serían por ejemplo los n-gramas, las máquinas de vectores de soporte (*SVM*), etc. De esta manera, se consolidaron tres modelos neuronales como fundamentales en las tareas de *NLP*:

- ***Recurrent Neural Networks*:** Estas redes fueron la elección más evidente dentro de las redes neuronales, utilizándose las *LSTM* que podemos observar en la Figura 2.6, que demostraron obtener mejores resultados que una *RNN* simple (Figura 2.5).

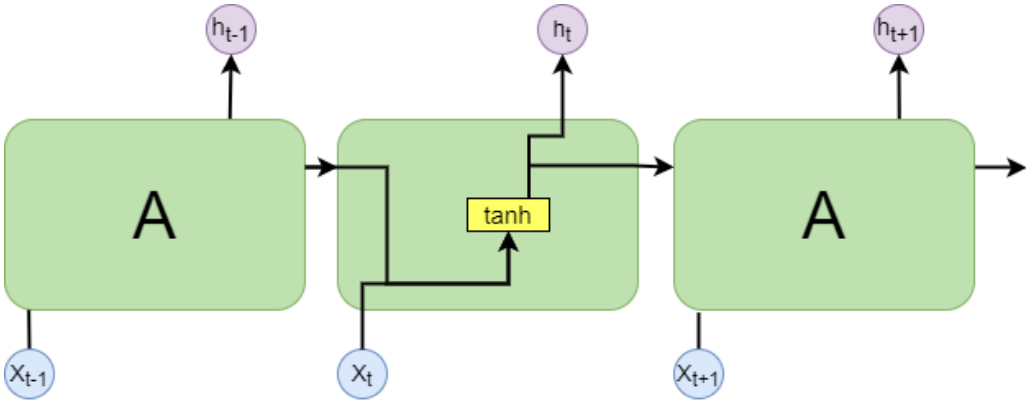

Figura 2.5: Esquema de una *RNN*. Fuente.

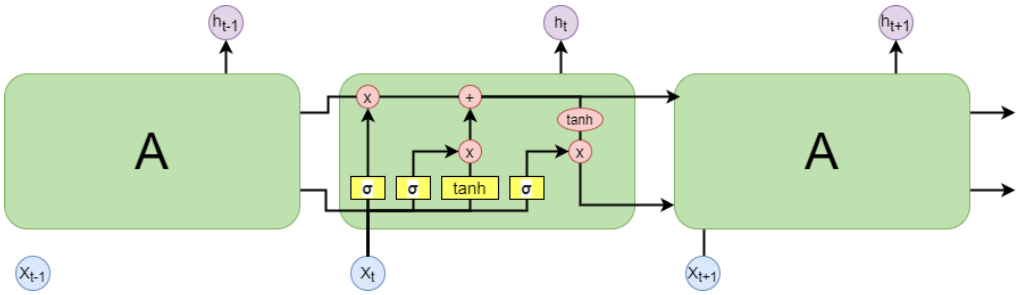

Figura 2.6: Esquema de una *LSTM*. Fuente.

- ***Convolutional Neural Networks*:** El uso masivo de estas redes en visión por computador hizo que fueran adoptadas para el campo del lenguaje. De este modo, en una red neuronal Convolucional tenemos capas convolucionales, que aplican un filtro al vector de entrada para obtener una salida de tamaño reducido en comparación a la entrada. En estas redes, suele aplicarse varios filtros y con la salida se suele utilizar una función de activación como podrían ser la tangente hiperbólica (tanh) o *Rectified Linear units* (*RELu*). Además de estas capas, las *CNN* también tienen otro tipo de capas conocido como capas de agrupación (*pooling*), que se centran en aplicar un filtro para seleccionar una cantidad de valores fijos de la entrada, como por ejemplo sería aplicar el máximo en diferentes grupos de la matriz para reducir su tamaño. Estas características convierten a las *CNN* en la elección óptima para tratar con imágenes, pero su uso en tareas de *NLP* no parece tan evidente. Sin embargo, en 2014 Kim trató en [15] como un modelo basado en *CNN* utilizando *word embeddings*, proceso observable en la Figura 2.7, obtenía mejores resultados que el estado del arte en 4 de 7 tareas, incluyendo tareas como análisis de sentimientos o clasificación de preguntas y todo esto sin ajustar demasiado los parámetros de la red para cada tarea específica.

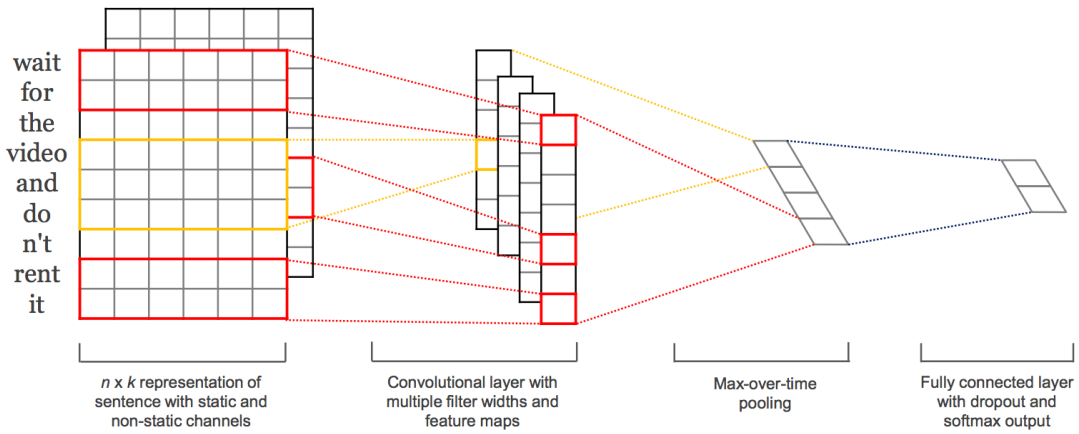

Figura 2.7: Esquema de una *CNN*. Fuente.

- ***Recursive Neural Networks*:** A diferencia de las redes anteriores, que tratan el lenguaje como una secuencia, las redes neuronales recursivas (*RecNN*) tienen la capacidad de establecer relaciones jerárquicas que toman forma de árbol que podemos comprobar en la Figura 2.8. De esta forma, en estas redes se utiliza un n-grama para que sea parseado y convertido en un árbol binario en que cada nodo es una palabra. Así, el resultado para cualquier tarea de *NLP* como podría ser análisis de sentimientos se calcula desde las hojas hasta la raíz. Estos modelos obtuvieron muy buenos resultados en el trabajo realizado por Socher y sus compañeros en [16], donde muestran tres modelos distintos basados en redes neuronales recursivas que acaban obteniendo mejores resultados que los modelos no neuronales más utilizados (*SVM*, *Naive Bayes* y *Naive Bayes* con una bolsa de bigramas (*BiNB*)). Adicionalmente, experimientaron el impacto de la negación, mediante la inclusión de *"contrastive conjunctions"* como "pero", sobre *BiNB* y los modelos basados en *RecNN* y comprobaron que la precisión de su mejor modelo superaba al *BiNB* en un 14 %.

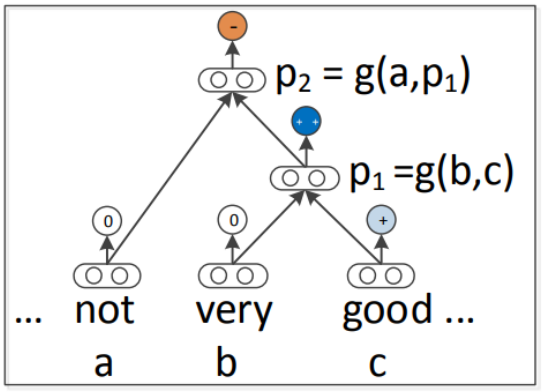

Figura 2.8: Ejemplo de un árbol en una *RecNN* para la tarea de análisis de sentimientos. Fuente.

Tras todas estas líneas de investigación paralelas, pasamos a ver en 2014 como en [17] se propone un modelo de aprendizaje basado en un *mapping* de una secuencia a otra diferente a partir de redes neuronales. Para ello, deciden utilizar una red neuronal codificadora (Figura 2.9) que procesa los símbolos de una frase y los convierte en un vector de tamaño fijo y después otra red neuronal decodificadora se encarga de predecir la salida símbolo a símbolo. En la práctica, se decantaron por usar varias capas de *LSTMs* y sus resultados sobre el BLEU en traducción del Inglés al Francés demostró ser un 1 % mejor que el resto de modelos, y sobretodo, ser bastante capaz a la hora de tratar con frases largas. Este modelo supuso un gran cambio dentro del campo de la traducción automática.

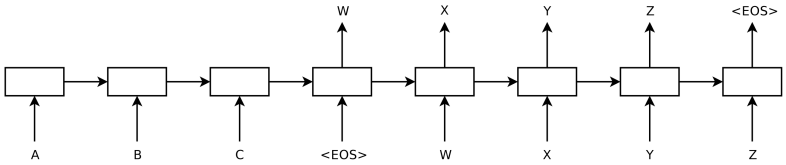

Figura 2.9: Modelo codificador decodificador de cadenas. Fuente.

Un año después, en 2015, Bahdanau comentó en [18] el reciente uso que se estaba haciendo de redes neuronales en traducción automática y comentaba como los modelos propuestos hasta ese momento, incluido el de Sutskever en [17] tenían el problema de codificar la frase de entrada (x) en un vector de tamaño fijo. De esta forma, su trabajo se basa en una red neuronal recurrente bidireccional (tiene en cuenta el contexto de izquierda a derecha y de derecha a izquierda) como codificadora y, a parte, también utilizan un vector de contexto (c_i) en su modelo que se calcula utilizando las salidas ($h_1, h_2, ...h_{Tx}$) que tiene el codificador. Además, a cada salida se le asocia un peso (α_{ij}), de tal manera que se calcula el contexto como una suma ponderada de las salidas con los pesos de atención. Este mecanismo puede ser visto como un modelo de alineamiento entre las salidas del decodificador y las del codificador. Todos estos nuevos elementos lo que consiguen es que el decodificador tenga un mecanismo de atención, que podemos observar en la Figura 2.10, en el que con las salidas del codificador se decide que partes son las realmente importantes y permite que nuestro codificador no tenga que representar toda la información en un vector de tamaño fijo. Los resultados que obtienen son bastante consistentes, superando al modelo de codificador decodificador que se utilizaba hasta el momento por un gran margen y obteniendo mejores resultados (0,5 %) que el sistema de traducción automática Moses que había sido entrenado con 418M de palabras más.

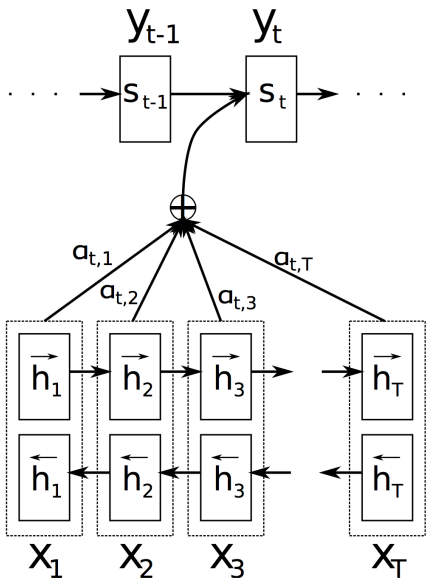

Figura 2.10: Decodificación de una cadena utilizando atención. Fuente.

El concepto de atención se convirtió en un elemento indispensable para muchas tareas de *NLP* en las que hubiera que centrar el foco en una parte concreta de la entrada, y fue en 2017 cuando surgió gracias al trabajo de Vaswani en [19] el *Transformer*, un modelo que dejaba atrás las *RNN* y *CNN* para estar completamente formado por atención y *feed-forward networks* y cuya estructura se puede apreciar en la Figura 2.11.

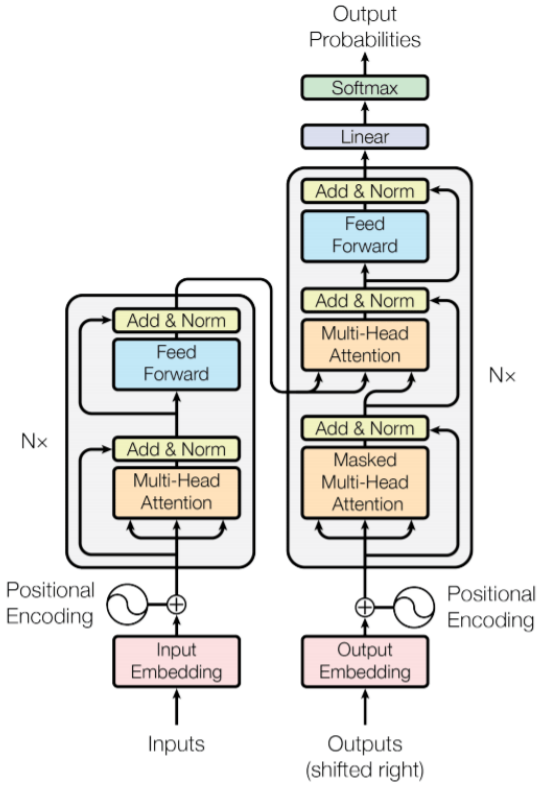

Figura 2.11: Arquitectura del *Transformer*. Fuente.

Además de este detalle, en su trabajo también introducen el concepto de *self-attention*, que se refiere a que el codificador mira el resto de palabras cuando está procesando una concreta para así establecer relaciones complejas entre ellas. Este concepto es muy útil porque si por ejemplo queremos traducir *"The animal didn't cross the street because it was too tired."* (frase obtenida gracias a Jay Alammar en [20]) y nos encontramos con la palabra *"it"* en nuestro decodificador, queremos que nuestro modelo entienda la importancia de relacionar la palabra *"it"* con *"animal"* por hacer referencia a ella. Este concepto se calcula utilizando vectores de *query* (Q), *key* (K) y *value* (V) en el codificador, que podemos observar a continuación:

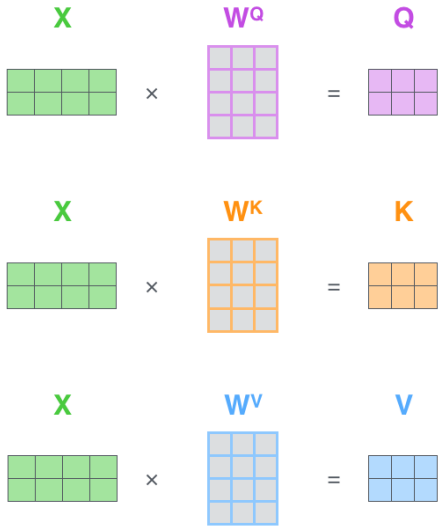

Figura 2.12: Matrices necesarias. Fuente.

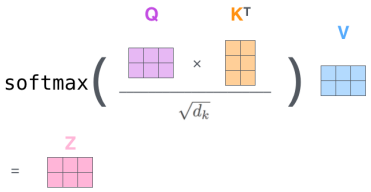

Figura 2.13: Cálculo del *self-attention*. Fuente.

Este concepto de *self-attention* lo refinaron usando varios cabezales de atención en paralelo, permitiendo que la atención tenga diferentes matrices de *querys, keys* y *values* y por tanto, diferentes subespacios vectoriales. Estas mejoras aparecen en la Figura 2.11 y se corresponden con las capas de *Multi-head attention*.

Su trabajo con este modelo les permitió mejorar el estado del arte en el BLEU en la tarea de traducir del inglés al alemán en un 2 % y del inglés al francés en un 0,5 % con un tiempo de ejecución en entrenamiento muy inferior a los modelos con los que se comparaban.

Tras todos estos avances, surgen a partir de 2018 los *word embeddings* basados en modelos contextuales gracias a los *Embeddings from Language Models* (*ELMo*), una arquitectura fruto del trabajo de Peters en [21] en la que se utiliza un modelo de lenguaje bidireccional (*biLM*), más concretamente un *LSTM* bidireccional que se muestra en la Figura 2.14 del que se aprenden diferentes *word embeddings* por palabra según su contexto lingüístico en el corpus de entrenamiento. Este *LSTM* bidireccional nos permite fijarnos en el contexto anterior y posterior a la palabra concreta. Este modelo es entrenado en *LM*, tarea que consistía en predecir la siguiente palabra y que obtiene *word embeddings* muy buenos y generales, sin estar orientados a tareas de *NLP* concretas. Este modelo que presentaron puede añadirse a otros modelos centrados en tareas específicas simplemente preentrenándolo y luego usándolo como un componente encargado del lenguaje.

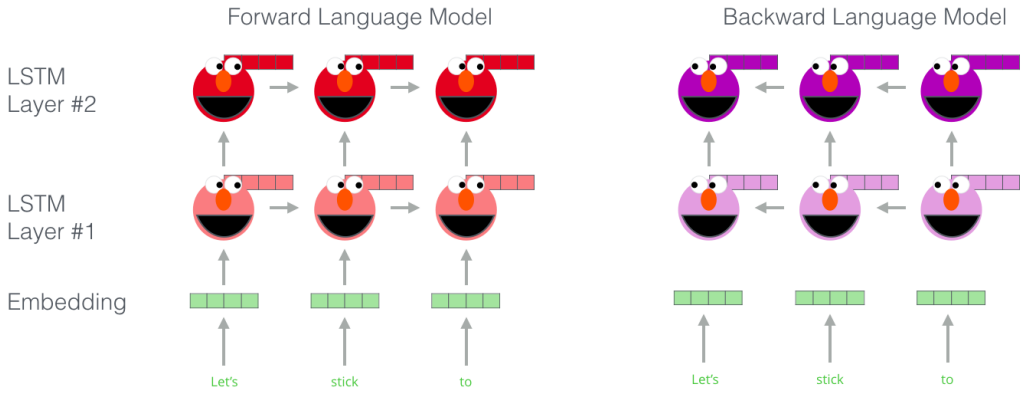

Figura 2.14: Representación del modelo *ELMo* con el *LSTM* bidireccional. Fuente.

Este mismo año se publica también *Universal Language Model Fine-tuning* (*ULMFiT*), un trabajo realizado por Howard y Ruder [22] en que incorporan un método de transferencia del aprendizaje entre tareas de *NLP* además de otras técnicas de *fine-tuning*. *Fine-tuning* consiste en utilizar un modelo entrenado para una tarea y reajustarlo o configurarlo para otra tarea totalmente distinta. Su modelo es entrenado en tres fases:

- En primer lugar entrenan a la red con la tarea de *LM* ya vista anteriormente con un corpus de carácter general.
- En segundo lugar ajustan ese modelo en la tarea concreta usando *fine-tuning* discriminativo (Discr) y *Slanted Triangular Learning Rates* (*STLR*).
- En tercer y último lugar se ajusta el clasificador en la tarea concreta gracias a *Gradual Unfreezing*, Discr y *STLR* para así adaptar las representaciones de alto nivel manteniendo las de bajo nivel intactas.

Sus resultados fueron bastante positivos, superando los resultados del estado del arte en 6 tareas de clasificación de texto y comparándose con modelos entrenados con 100x más datos.

Posteriormente en Mayo de 2019 aparecen los *Bidirectional Encoder Representations from Transformers* (*BERT*), un modelo creado por Google AI Language en [23] donde consiguen mejorar los resultados del estado del arte en 11 tareas. En este trabajo hacen uso de los modelos *Transformer* que hemos introducido anteriormente para crear *BERT*, teniendo dos tamaños: el primero ($BERT_{BASE}$) con 110M y el segundo ($BERT_{LARGE}$) con 340M de parámetros.

Tras *BERT* han ido surgiendo variaciones y mejoras como *A Robustly Optimized BERT Pretraining Approach* (*RoBERTa*) [24] en Julio del 2019, donde explican que una correcta configuración de los hiperparámetros de *BERT* obtiene mejores resultados que *BERT* o cualquiera de los modelos posteriores a su publicación.

La segunda mejora de *BERT*, que aparece en Febrero de 2020, denominada *A Lite BERT for self-supervised learning of language representations* (*ALBERT*) [25], donde utilizan 2 técnicas de reducción de parámetros y se centran en modelar mejor la coherencia entre frases para así obtener mejores resultados que el estado del arte 8 meses después de la publicación de *BERT* y teniendo menos parámetros que $BERT_{LARGE}$.

Por último está *ELECTRA* [26], un modelo que surge en Marzo de 2020, donde sus autores utilizan una técnica de entrenamiento diferente a la de *BERT* y que permite un mejor uso de los datos y mejora de manera significativa los resultados obtenidos con *BERT*, especialmente en modelos de tamaño pequeño. Sus resultados son comparables a *RoBERTa* utilizando una cuarta parte del tiempo de entrenamiento o superando los resultados con el mismo tiempo de entrenamiento.

Con esto, ya hemos introducido lo que se considera como la evolución del *Deep Learning* aplicado al lenguaje natural y poniendo un poco de contexto vamos a utilizar los resultados del GLUE [27] [28] para poner en contexto el avance que ha habido desde el principio y donde nos encontramos ahora.

El modelo que aparece en último lugar y del que hemos hablado en esta sección es el *Continuous Bag of Words* (*CBOW*) por Mikolov donde el score es de 58,6. A continuación y yendo hacia arriba podemos encontrar muchos otros hasta que finalmente llegamos al último previo a los *word embeddings* contextuales que es un *biLSTM + Attention* (*Attn*), obteniendo un 65,6. Tras esto tenemos ya modelos que utilizan *word embeddings* contextuales como las GLUE Baselines en el puesto 34 que utilizan un *BiLSTM + ELMo + Attn* con un score de 70. Después tenemos varios modelos basados en *BERT*, entre los que tenemos *BERT: 24-layers, 16-heads, 1024-hidden* que llega a un 80,5. A continuación está *RoBERTa* en la onceava posición con un 88,1, *ELECTRA* en séptimo lugar y un score de 89,4 y el primer puesto pertenece a una configuración de *ALBERT + DAAF + NAS* que llega al score de 90,6.

Como podemos observar, en tan solo 7 años este campo ha aumentado en el GLUE casi en 30 puntos el score de los modelos y el cambio más grande ha sido gracias a *BERT*, que aumentó en un score de 10 los resultados que había hasta la fecha.

De todo lo presentado anteriormente, se deduce la potencia que los modelos contextuales de *word embeddings* suponen para este campo y que pueden considerarse el cambio revolucionario de estos años.

CAPÍTULO 3
Redes neuronales y *word embeddings*

En este capítulo se introducen diferentes conceptos con la intención de que el lector pueda entender estos conceptos y el trabajo realizado. De esta forma, introduciremos las redes neuronales así como los mecanismos utilizados para entrenarlas y que aprendan, para posteriormente comentar los *word embeddings*, típicamente basados en redes neuronales y se pueden obtener a partir de modelos contextuales e incontextuales, en los que nos adentraremos para explicarlos detalladamente. Finalmente comentaremos las *Deep averaging networks* (*DAN*) que tendrán un papel importante en el posterior trabajo práctico realizado con *word embeddings*.

3.1 Redes neuronales

En primer lugar vamos a explicar de forma detallada el concepto y contexto de las redes neuronales. Este modelo surgió a mitades del siglo pasado, pero diferentes cambios y apariciones de algoritmos han posibilitado su uso a día de hoy. Las redes neuronales son modelos basados en unidades muy sencillas: neuronas.

Una neurona es simplemente una función lineal que recibe unos parámetros de entrada ($x_1, x_2, ...x_N$) y como salida se obtiene un resultado (y). Esta función que tiene la neurona (ecuación 3.1) es una suma ponderada, en la que la neurona tiene unos pesos ($w_1, w_2, ...w_N$) y un parámetro denominado sesgo (b). Estos pesos son relativos a las diferentes conexiones con cada parámetro de entrada y el sesgo actúa como el término independiente de una función. De esta manera, estos pesos que tiene cada neurona de nuestro modelo son los parámetros que debemos aprender para ajustar el modelo a la tarea en la que lo utilicemos.

$$w_1 * x_1 + w_2 * x_2 + ... + w_N * x_N + b = y \tag{3.1}$$

Sin embargo, la neurona necesita a parte de todo esto una función de activación (f). Esto es debido a que si nuestras neuronas carecieran de esta función, estaríamos aplicando a unos parámetros de entrada una serie de funciones lineales y esto matemáticamente equivale a realizar una sola función lineal, de tal forma que nuestro modelo solo podría hacer fronteras de separación lineales. De esta manera, las funciones de activación son funciones matemáticas que transformarán no linealmente nuestros valores de salida evitando así la linealidad de nuestro modelo. Algunas de estas funciones son:

Nombre	Función Matemática	Rango
Escalón	$y = sign(x)$	$\{-1, +1\}$
	$y = H(x)$	$\{0, +1\}$
RELu	$y = max(0, x)$	$[0, x]$
GeLu	$y = 0{,}5x(1 + \frac{2}{\sqrt{\pi}} \int_0^{\frac{x}{\sqrt{2}}} e^{-t^2} dt)$	$[0, x]$
Sigmoide	$y = \frac{1}{1+e^{-x}}$	$[0, +1]$
	$y = tgh(x)$	$[-1, +1]$
Gaussiana	$y = Ae^{-Bx^2}$	$[0, +1]$
Softmax	$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$	$[0, 1]$

Tabla 3.1: Tabla con diferentes funciones de activación.

De esta forma, el esquema final de una neurona lo podemos observar a continuación, en la Figura 3.1.

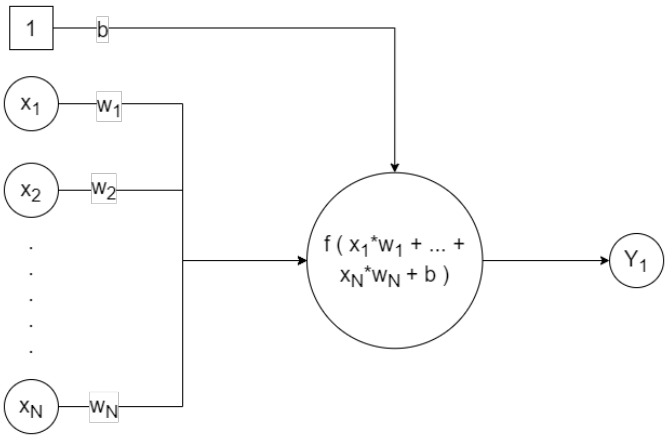

Figura 3.1: Esquema de una neurona.

Establecido ya el concepto de neurona, pasamos a hablar de la red neuronal como conjunto y modelo, ya que este no es más que un conjunto de neuronas estructuradas por capas, de tal forma que tenemos una primera capa de entrada que recibe los parámetros de entrada, los cuales son procesados por esta capa y los resultados que estas neuronas obtienen son transmitidos a las capas posteriores. La última capa se conoce como capa de salida y todas las que se encuentran entre la capa de entrada y de salida se conocen como capas ocultas.

El número de capas ocultas así como el número de neuronas que tendrá cada capa de nuestro modelo son parámetros que se establecen al crear un modelo concreto que se quiere utilizar y que deben comprobarse para la tarea concreta que se esté llevando a cabo para encontrar así la mejor configuración. Así, en la Figura 3.2 tenemos un ejemplo de un esquema de la estructura de una red neuronal.

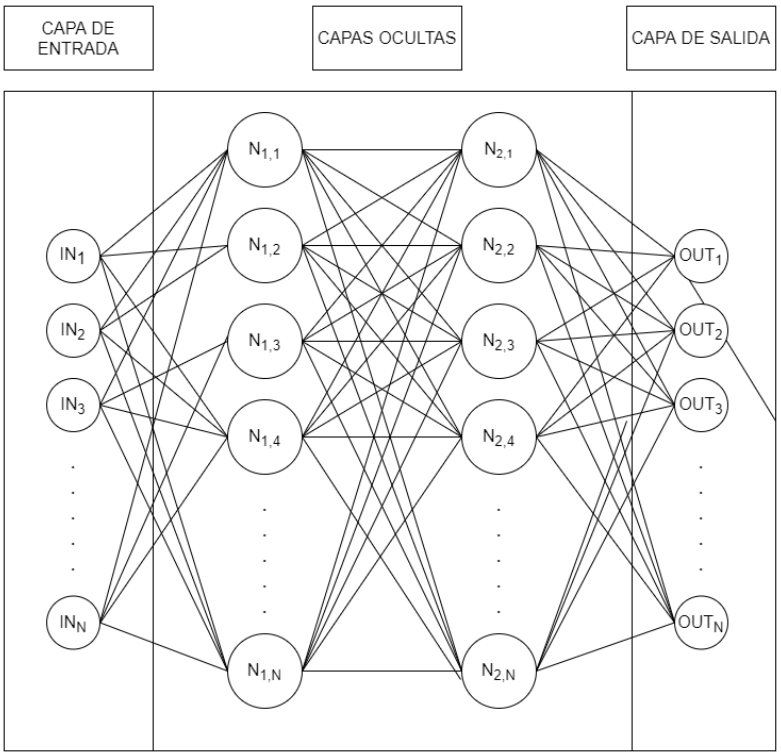

Figura 3.2: Estructura de una red neuronal.

Todos estos conceptos ya se habían establecido en el modelo conocido como Perceptrón, el precursor de las redes neuronales y que consiste en una neurona con función de activación escalón y por tanto, un clasificador lineal. Sin embargo, el algoritmo de aprendizaje aplicado sobre el perceptrón no era aplicable a las redes neuronales con capas ocultas y por tanto, no teníamos ningún algoritmo que entrenase a estas redes neuronales.

Durante estos años, se utilizaba una técnica conocida como perturbación aleatoria para entrenar a las redes neuronales, que era un algoritmo de fuerza bruta en que se calculaba para cada neurona cómo variaba el coste al introducir un cambio, cálculo que se realizaba muchas veces para muchos caminos hacia delante, siendo esto muy ineficiente por tener que considerar todos los caminos desde la neurona actual hasta la capa de salida.

Por ello y tras una época de estancamiento, apareció el algoritmo de aprendizaje que se viene utilizando desde el año 1986 hasta la actualidad: *backpropagation*.

Backpropagation es un algoritmo que permite actualizar los parámetros de las neuronas de una red neuronal utilizando otro algoritmo más sencillo conocido como el descenso del gradiente. La idea consiste en utilizar el error final de la capa de salida para ir corrigiendo los parámetros de las capas anteriores, capa a capa, hasta llegar a la capa de entrada yendo hacia atrás. Esta técnica es muy eficiente ya que yendo capa a capa podemos actualizar todos los pesos de nuestra red neuronal, obteniendo un algoritmo lineal con el número de capas y parámetros.

3.2 Descenso del gradiente

En esta sección se presenta el algoritmo del descenso del gradiente, el cual busca minimizar o maximizar una función variando sus variables.

A continuación se presenta un ejemplo sencillo comparativo con el funcionamiento de este algoritmo para que el lector pueda entender más fácilmente esta técnica. Supongamos que nos encontramos en un terreno montañoso con los ojos totalmente tapados y sin ningún tipo de mapa y nuestro objetivo es llegar al punto más bajo de todo el terreno. Como no tenemos ningún tipo de información global del terreno y estamos ciegos, nuestra mejor opción es tantear con nuestras piernas el terreno, considerar hacia que dirección desciende la pendiente, caminar en esa dirección y luego volver a pararnos para repetir esta serie de pasos hasta que todas las direcciones tengan una pendiente ascendente y hayamos llegado a un punto mínimo.

Este ejemplo ilustra de forma sencilla la formulación de este algoritmo, de tal forma que empezamos con una configuración de parámetros ($\theta(1)$) totalmente arbitraria, calculamos las derivadas parciales ($\frac{\partial error}{\partial \theta_1}$, $\frac{\partial error}{\partial \theta_2}$, ... $\frac{\partial error}{\partial \theta_N}$) de cada parámetro con respecto a la función de coste y estos valores de las derivadas forman el vector gradiente (∇f), que nos indicará la dirección hacia la que asciende nuestra función. De esta forma, lo que haremos será ir en contra del gradiente, actualizando nuestros parámetros restándole el vector gradiente multiplicado por un parámetro extra del algoritmo, el factor de aprendizaje (α). Esto se repite hasta la convergencia o hasta que se supere un número máximo de iteraciones.

Todo esto forma un algoritmo con resultados muy distintos, consiguiendo que el número de iteraciones, el factor de aprendizaje y los valores iniciales de los parámetros definan totalmente cómo actúa el algoritmo, obteniendo muchas posibilidades como que la convergencia sea rápida, pero se llegue a un mínimo local, o que por el contrario sea lenta y llegue al mínimo global o incluso, que el algoritmo no converja.

Como podemos observar, el algoritmo utiliza las siguientes fórmulas:

$$\theta(1) = arbitrario$$

$$\theta(k+1) = \theta(k) - \alpha \nabla f$$

Este algoritmo no puede ser aplicado directamente sobre una red neuronal porque el cálculo del gradiente de un parámetro de una neurona puede afectar al resultado final por todos los caminos hacia delante desde esa conexión hasta todas las neuronas de salida. Esto sin contar con el hecho de que también depende en cada camino de los valores de los parámetros posteriores, que si varían, también variará el error. Todo esto en resumen acaba implicando que el uso de este algoritmo sobre redes neuronales sin más sea totalmente inviable en la práctica. Y es así que se utiliza *Backpropagation* basado en descenso del gradiente.

3.3 *Word embeddings*

Los *word embeddings* son una forma de representación del lenguaje que surgió como una alternativa continua a representaciones discretas como *bag of words* y que resultaban bastante ineficientes debido al uso de vectores dispersos del tamaño de tu vocabulario. De esta forma, los *word embeddings* son un modelo de representación del lenguaje en que los vectores, de dimensionalidad reducida, buscan preservar relaciones semánticas entre palabras, de tal forma que al representar las palabras se cumplen algunas propiedades interesantes como por ejemplo que la distancia entre una capital y su país sea igual o muy semejante para todas las que aparezcan en los datos de entrenamiento. Esto también deriva en otra propiedad muy interesante que implica que $V(\text{“Moscú”}) - V(\text{“Rusia”}) + V(\text{“España”}) = R$ se cumple, siendo cada uno de estos $V()$ el *word embedding* correspondiente a las palabras indicadas y el vector R el vector resultado de esta operación y que según indica la teoría, tendría en el espacio vectorial a Madrid como la palabra más cercana. Esto quiere decir que la distancia entre Moscú y Rusia es muy parecida a la que hay entre Madrid y España, tal como se puede observar en la Figura 3.3.

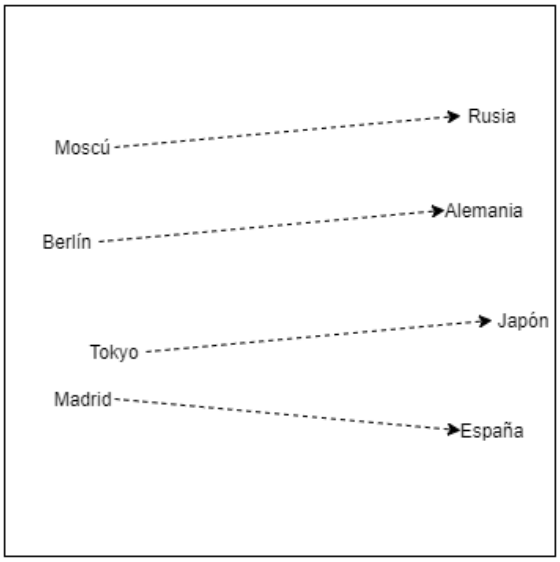

Figura 3.3: Propiedad de los *word embeddings* en un espacio bidimensional simplificado.

Introducida la teoría y propiedades de los *word embeddings*, pasamos a hablar de los dos tipos de modelos que tenemos para obtener esta representación del lenguaje. En primer lugar tenemos los modelos incontextuales, de tal forma que cada palabra que aparece en entrenamiento tendrá una sola representación vectorial obtenida colapsando toda la información que se tiene de esta palabra con todas sus apariciones en los datos. Dentro de estos modelos tenemos implementaciones como pueden ser *FastText* [29], *Word2Vec* o *GloVe* [30]. Por otro lado tenemos los modelos contextuales, los cuales tienen una complejidad añadida debido a que permiten que las palabras tengan una representación vectorial para cada contexto en el que aparecen. Entre estos modelos tenemos *ELMo*, *BERT*, *ALBERT* o *ELECTRA*.

3.3.1. Modelos incontextuales

Definidos ya los dos tipos de modelos, vamos a introducir ahora cómo se generan los *word embeddings* incontextuales, los cuales son bastante sencillos en su forma más básica, pero tienen detalles concretos de implementación de los que hablaremos y que van a introducir una capa de complejidad. En esencia, lo que vamos a hacer es entrenar una red neuronal con una única capa oculta, la cual tendrá un número de neuronas concreto según el parámetro (D) que nosotros le indicaremos al modelo. Una vez realizado el entrenamiento, cogeremos los pesos de la capa oculta de esta red neuronal entrenada y los utilizaremos como las representaciones vectoriales de nuestras palabras como podemos observar en la Figura 3.4.

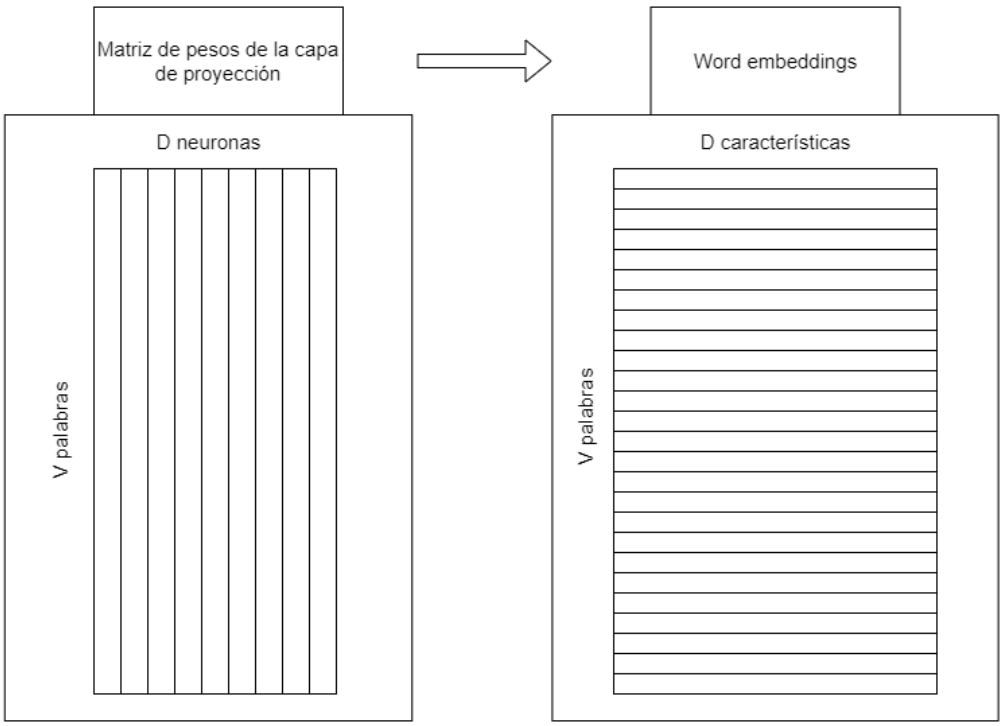

Figura 3.4: Equivalencia entre los pesos de nuestra red y el resultado.

De este modo y en primer lugar, se genera un vocabulario con las palabras del corpus de entrenamiento. Con esto tendremos V palabras únicas que forman nuestro vocabulario y así, representaremos a las palabras como un vector one-hot de dimensionalidad V. Este vector será un vector disperso con un único 1 en la posición de la palabra concreta que estamos representando.

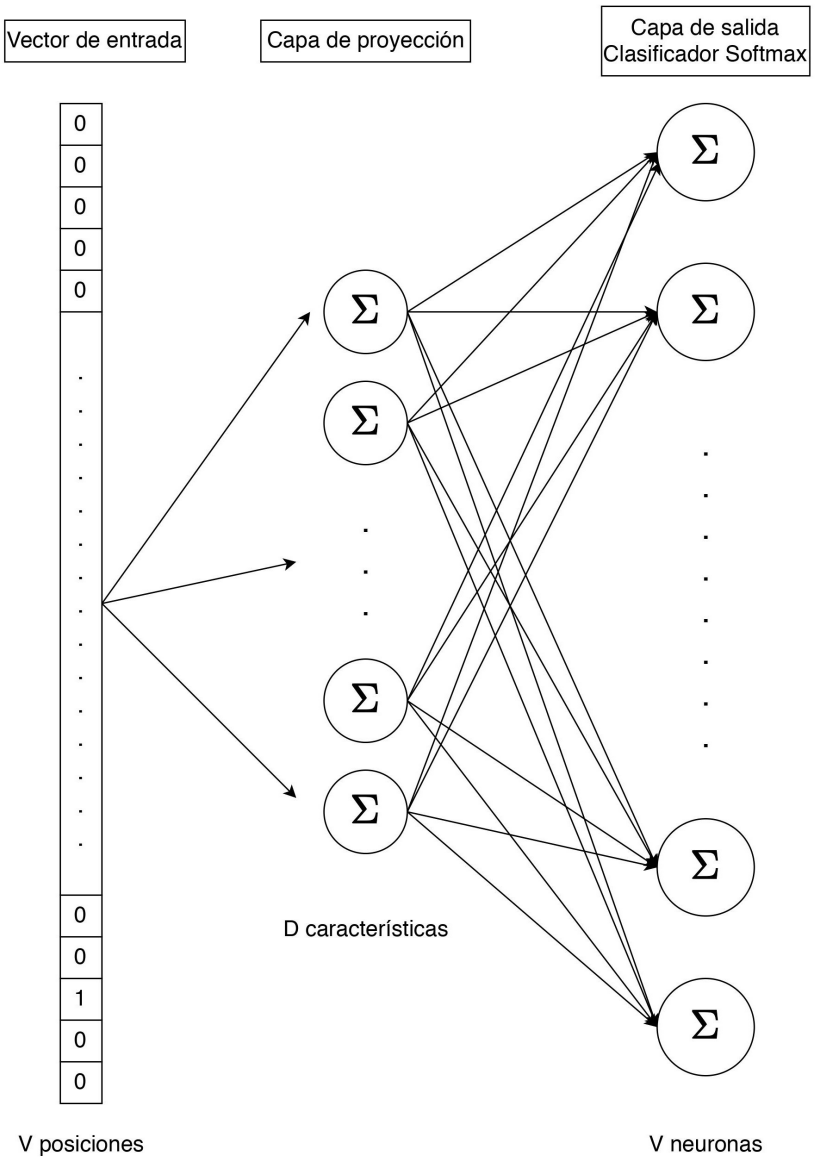

Figura 3.5: Estructura del modelo *Skip-gram*.

Como podemos ver en el modelo (Figura 3.5), tenemos una capa oculta que no tiene ninguna función de activación y por tanto se le llama capa de proyección. Por otro lado, se emplea una función de activación *softmax* en la capa de salida, de esta forma, la salida de la red será un vector con la probabilidad para cada palabra del vocabulario de ser la palabra del contexto elegida aleatoriamente.

Esta tarea en la que entrenaremos a la red neuronal depende del tipo de modelo in-contextual que se utilice, y así vamos a introducir las que Mikolov propuso en [13] y que podemos observar en la Figura 3.6. De este modo tenemos:

- En primer lugar, el modelo *CBOW* o *Continuous Bag of Words*, en que la tarea será predecir una palabra concreta (w_t) dadas las palabras de contexto alrededor de esta ($w_{t-N}, ..., w_{t-1}, w_{t+1}, ..., w_{t+N}$).

- Y en segundo lugar, en el modelo *Skip-gram* (*SG*), la tarea con la que se entrena a la red neuronal consiste en seleccionar una palabra (w_t) y intentamos predecir las palabras alrededor de w_t. Para ello, utilizamos un clasificador log-linear para obtener las probabilidades de que una palabra del vocabulario este dentro del contexto de w_t.

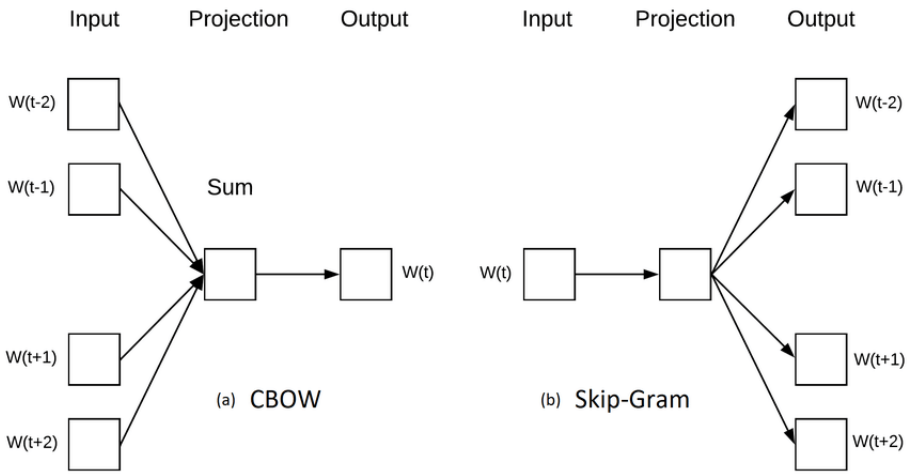

Figura 3.6: Esquema de los dos modelos propuestos por Mikolov. Fuente.

A partir de este momento nos centraremos en el modelo *Skip-gram*, que es el que utilizaremos posteriormente en el caso de estudio y por tanto, el que nos interesa ver más en profundidad. Así pues, formalizando este modelo, lo que se propone es que dada una secuencia de palabras ($w_1, w_2, ...w_N$), se maximice la probabilidad del logaritmo, que podemos observar en la Fórmula 3.2.

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j} \mid w_t) \tag{3.2}$$

donde c es el tamaño del contexto de entrenamiento. Como nos indica la bibliografía [14] y observando la fórmula, aumentar el tamaño del contexto implica aumentar el número de datos a predecir, lo que significa utilizar más cada frase y por tanto puede llevar a una mejor precisión. Sin embargo, esto también significa que el tiempo de entrenamiento aumenta de forma irremediable. Otro detalle importante a resaltar de este modelo y que podemos observar en la formulación es que no se tiene en cuenta la distancia entre la palabra actual y la palabra que se escoge aleatoriamente.

La definición que este modelo hace de la probabilidad $P(w_{t+j} \mid w_t)$ usando la función *softmax*, teniendo la siguiente fórmula:

$$P(w_{wo} \mid w_t) = \frac{\exp(v'^{T}_{wo} v_{wI})}{\sum_{w=1}^{V}\exp(v'^{T}_{w} v_{wI})} \tag{3.3}$$

donde v_w y v'_w son los vectores de entrada y salida correspondientes a w y V es el número de palabras del vocabulario. Este enfoque no es viable debido a que la computación de cada probabilidad condicionada, $\nabla log p(wO|wI)$, equivale a la talla del diccionario, que suele ser bastante grande.

Además, si consideramos el tamaño, podemos comprobar que tenemos $2 * V * D$ parámetros en nuestra red neuronal entre la capa de proyección y la capa de salida, siendo esto una gran cantidad de parámetros a ajustar. El proceso de entrenamiento no sólo será lento, sino que se necesitará una cantidad de datos bastante grande para ajustar todos los valores de los parámetros, lo que también llevará mucho tiempo al tener muchos datos con los que entrenar.

Para remediar este problema de obtener $\nabla log p(wO|wI)$ y entrenar una red con una cantidad tan grande de parámetros se establecen dos aproximaciones, que se discuten en las siguientes subsecciones.

Negative sampling

Primero tenemos ***negative sampling***, propuesto por Mikolov y sus compañeros en su segundo trabajo [14], y donde solucionaron ambos problemas introduciendo las siguientes mejoras:

En primer lugar, el modelo pasó a procesar pares de palabras muy comunes o incluso frases enteras que se repetían mucho como una única palabra en lo que al modelo refería. Esta mejora permite obtener mejores resultados con palabras compuestas, en las que el significado individual es diferente al que tienen las palabras juntas, por ejemplo palabras como Nueva York, Real Madrid o la Real Academia Española.

Además de esto, también se mejoró la calidad del entrenamiento teniendo en cuenta aquellas palabras comunes o *stopwords* que aparecen millones de veces y no benefician al entrenamiento de *embeddings*. Palabras como por ejemplo "de", "a" o "en" no aportan ningún valor a la representación que la palabra Madrid debería tener en el espacio vectorial. Del mismo modo, estas palabras comunes tampoco cambian significativamente a lo largo del entrenamiento con sus millones de ocurrencias. Esto les llevó a evitar este desequilibrio utilizando una probabilidad de descarte, utilizando la Fórmula 3.4.

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \tag{3.4}$$

donde $f(wi)$ es la frecuencia de la palabra wi y t es un umbral seleccionado que ellos consideran debería estar cerca de 10^{-5}.

Según indican en la fuente, la elección de este cambio y fórmula es debido a que querían actuar de forma agresiva con las palabras más frecuentes y que fue elegida heurísticamente para después comprobar que en la práctica daba muy buenos resultados. Además, este cambio consigue acelerar bastante el entrenamiento y mejora los vectores de las palabras con un número de ocurrencias reducido.

Por último, tenemos que hablar de su trabajo a partir de lo que se conoce como *Noise Contrastive Estimation* (*NCE*), que indica que los modelos deberían poder diferenciar entre los datos y el ruido. De esta forma, mientras que *NCE* maximiza la probabilidad del logaritmo *softmax*, a nosotros solo nos interesa obtener vectores que representen correctamente a las palabras, con lo que decidieron simplificar *NCE* a lo que denominaron *Negative Sampling* (*NEG*), definiéndolo a partir de su objetivo (ver Fórmula 3.5).

$$\log \sigma(v'^{T}_{wo} v_{wI}) + \sum_{i=1}^{k} \mathbb{E}_{wi \sim P_n(w)} [\log \sigma(-v'^{T}_{wi} v_{wI})] \tag{3.5}$$

con el que sustituyeron todas las probabilidades condicionadas del objetivo del modelo *Skip-gram*. De esta forma, la tarea se convierte en distinguir la palabra objetivo (wo) de la distribución de ruido $P_n(w)$ utilizando regresión logística, teniendo un parámetro k que indica el número de muestras negativas que se utilizan para cada dato. Según indica la fuente, los valores de k de forma empírica se consideran relativos al tamaño del corpus de entrenamiento, siendo un valor en el rango de 5 y 20 bueno para corpus pequeños y entre 2 y 5 para corpus de gran tamaño.

Softmax jerárquico

En segundo lugar tenemos el uso de ***softmax* jerárquico**, una aproximación eficiente introducida por Morin y Bengio en [31]. En esta mejora, solo se evalúan $log2(V)$ nodos de la capa de salida.

Para ello se utiliza un representación de árbol binario de la capa de salida con V palabras como hojas y para cada nodo (q) se representa específicamente las probabilidades relativas de los nodos hijos (qL y qR). Así se acaba definiendo un paseo aleatorio por el árbol que asigna probabilidades a las diferentes palabras.

Introduciendo nomenclatura y una definición más detallada, cada palabra (w) puede ser alcanzada por un camino desde la raíz, teniendo además que $n(w, j)$ es el nodo j del camino desde la raíz a w y $L(w)$ es la longitud del camino, podemos afirmar que $n(w, 1) = root$ y que $n(w, L(w)) = w$. Además, cualquier nodo dentro de n, siendo $ch(n)$ un hijo arbitrario de n y siendo $[[x]]$ igual a 1 si x es verdadero y -1 en caso contrario, podemos definir $P(w_o \mid w_i)$ de la siguiente manera:

$$P(w \mid w_I) = \prod_{j=1}^{L(w)-1} \sigma([[n(w, j+1) = ch(n(w, j))]] \cdot v'^{T}_{n(w,j)} v_{wI}) \tag{3.6}$$

donde σ es la función sigmoide ($P(t) = \frac{1}{1+e^{-t}}$). Esto acaba implicando que computar $\log p(w_O \mid w_I)$ y $\nabla \log P(w_O \mid w_I)$ es proporcional a $L(wo)$, que de media no será mayor a $log(W)$. También cabe destacar el hecho de que con esta formulación tenemos una representación (vw) para cada palabra (w) y una representación ($v'n$) para cada nodo interno (n) del árbol binario.

3.3.2. Modelos contextuales

Bidirectional Encoder Representations from Transformers (*BERT*) [23] es un modelo que surge a partir de los conceptos ya establecidos anteriormente y que mejoró el estado del arte considerablemente. Para ello, los autores indican las limitaciones que el modelo GPT y el *Transformer* de Vaswani [19] tienen por utilizar una arquitectura que procesa el lenguaje de izquierda a derecha y deciden entrenar a un modelo usando el codificador del *Transformer* [19].

Este codificador se preentrena con dos tareas:

- La primera será *Masked Language Model* (*MLM*), lo que significa que de un dato de entrada se enmascará una cantidad y unos *tokens* totalmente aleatorios, tal que la tarea de preentrenamiento del modelo es predecir el identificador de la palabra que había originalmente en la entrada.
- Y la segunda tarea será *Next Sentence Prediction* (*NSP*).

El uso de estas dos tareas introduce varias ventajas. La principal y más importante es que el uso de este lenguaje enmascarado les permite que su modelo entrene fusionando el contexto de la izquierda y la derecha, obteniendo así representaciones bidireccionales. Además, utilizar pares de texto también les permite que su modelo aprenda a relacionar frases, preparándolo mejor para tareas como *question answering*.

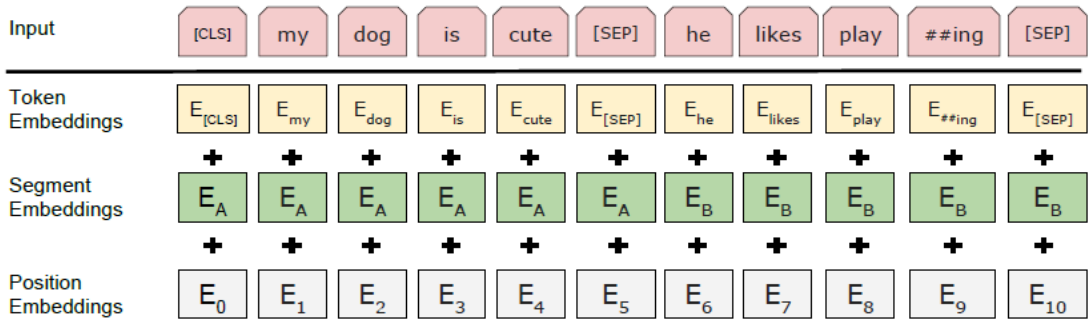

Figura 3.7: Representación de la entrada según *BERT*. Fuente.

Como podemos ver en la Figura 3.7, a parte de enmascarar algunos *tokens*, también tienen el símbolo especial [CLS] para indicar el inicio de un dato de entrada y el símbolo [SEP] para separar dentro del dato de entrada dos frases para cuando sea necesario.

Entrando en detalles de implementación concretos de la tarea de *MLM*, los autores optaron por enmascarar cada *token*, con 15 % de probabilidad, mediante alguno de los siguientes *tokens*:

- El *token* [MASK] con una probabilidad del 80 %.
- Un *token* aleatorio con una probabilidad del 10 %.
- El mismo *token* con una probabilidad del 10 %.

Al mismo tiempo, cuando hablamos de la tarea *NSP*, los autores optaron por balancear el número de muestras positivas y negativas, de tal manera que, un 50 % de las veces B sea realmente la frase siguiente a A y el otro 50 % de las veces sea una frase aleatoria del corpus de entrenamiento.

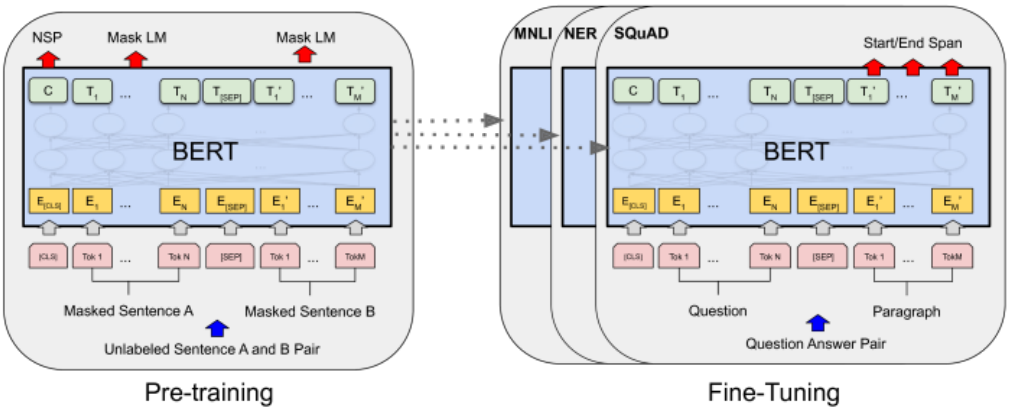

Figura 3.8: Esquema global del modelo. Fuente.

Una vez tenemos ya nuestro modelo preentrenado, observable en la Figura 3.8 a la izquierda, simplemente tendríamos que usarlo para tareas específicas cogiendo la entrada de esta tarea y configurando los parámetros para esa tarea concreta, que aparece a la derecha de la Figura 3.8. Según indican, este proceso es mucho menos costoso y más simple que la fase de preentrenamiento.

3.4 *Deep averaging networks*

Una vez presentados los modelos contextuales e incontextuales es relevante para el trabajo desarrollado hacer las siguientes aclaraciones:

- En primer lugar tenemos el modelo incontextual *skip-gram*, que utilizamos para preentrenar nuestros *word embeddings*. Sin embargo, conforme tenemos estos, ahora solo podemos utilizarlos como entrada a otro sistema que los utilice para la tarea concreta del lenguaje en que queremos trabajar. Estos modelos son denominados modelos *feature-based*.
- Por otro lado tenemos *BERT*, que como hemos indicado antes se preentrena para que aprenda el lenguaje y luego se le hace un reajuste para la tarea concreta. Estos modelos se conocen como modelos *fine-tuning*.

Debido a esto, es necesario introducir el modelo que utilizaremos con nuestros *word embeddings* incontextuales posteriormente en nuestro trabajo. Con esto se introduce las *Deep averaging networks* (*DAN*), un tipo de red neuronal presentado por Iyyer y sus compañeros en el año 2015 [32], donde presentan una red neuronal simple que iguala o mejora los resultados de modelos más complejos en análisis de sentimientos con un tiempo de entrenamiento menor al resto de opciones.

Este modelo no tiene en cuenta el orden relativo de las palabras en una oración, sino que sigue una metodología de trabajo bastante simple. Primero se utiliza el vector media de los *word embeddings* de los *tokens* de entrada, luego se va propagando por una o más capas hacia delante y finalmente se hace una clasificación lineal en la última capa.

En su trabajo indican que para análisis de sentimientos ellos consideran más importante el uso de buenos *word embeddings* preentrenados que el uso de redes muy sofisticadas y proceden a comparar modelos que no tienen ordenación de palabras con otros que sí tienen en cuenta la composición sintáctica de la frase. Así, introducen las redes neuronales recursivas, que podemos ver a la izquierda de la Figura 3.9, y las comparan con una bolsa de palabras neuronal (*NBOW*) y tras ello indican que el objetivo es conseguir un modelo rápido como el *NBOW* y con buenos resultados como las redes neuronales recursivas.

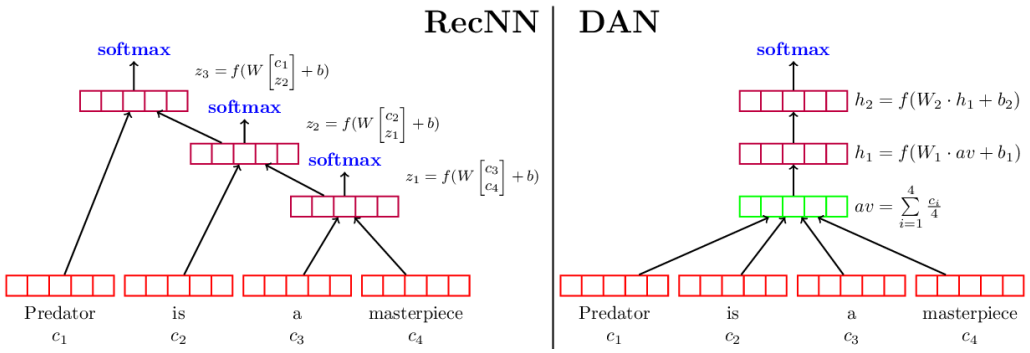

Figura 3.9: Comparativa entre las *RNN* y las *DAN*. Fuente.

De este modo presentan sus *DAN* (derecha de la Figura 3.9), una red que creen puede obtener buenos resultados gracias a que las redes neuronales *feed-forward* capturan una representación más abstracta de la entrada que la anterior. De esta forma, para calcular el vector media (z) de una entrada (X), utilizan la media de los *word embeddings* $v_{w \in X}$ y en vez de pasar este vector a una capa de salida, lo procesan mediante n capas ($z_{1...n}$), que calcularán:

$$z_i = g(z_{i-1}) = f(W_i \cdot z_{i-1} + b_i) \tag{3.7}$$

Finalmente, cuando se obtiene la representación de la última capa (z_n), se emplea como entrada de una capa *softmax* que hará la tarea de predicción.

Este modelo fue optimizado gracias al concepto de *Dropout* [33], que consiste en abandonar o ignorar ciertos valores de salida de algunos nodos de alguna capa, como podemos comprobar en la Figura 3.10. Este concepto es muy relevante porque evita un problema bastante importante cuando se trata con redes neuronales: *overfitting*. El *overfitting* o sobreentrenamiento hace referencia a cuando un modelo no es capaz de generalizar y está demasiado ajustado a los datos de entrenamiento, haciendo que sus resultados al tratar con otros ejemplos esté muy marcado por el entrenamiento y obtenga malos resultados.

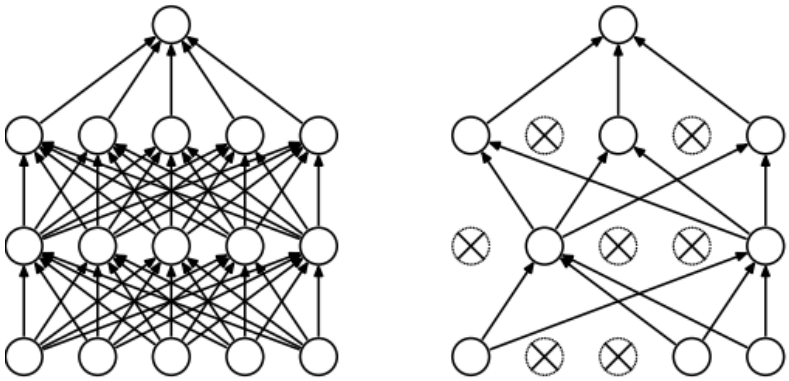

Figura 3.10: Ejemplo del *dropout* clásico aplicado a una red neuronal. Fuente.

Al utilizar *dropout* el modelo acaba siendo más robusto, ya que si tenemos una red con n neuronas, gracias al *dropout* tendremos 2^n posibles configuraciones de redes con los mismos parámetros y diferentes combinaciones de neuronas ignoradas.

Este concepto de *dropout* es utilizado por las *DAN* con una diferenciación bastante concreta. En vez de ignorar ciertas salidas de nodos concretos, las *DAN* ignoran ciertos *word embeddings* de algunos *tokens* para el cálculo del vector media. Esto implica que gracias a este *word dropout* la red observará teóricamente $2^{|X|}$ secuencias de *tokens* por cada entrada X.

Finalmente, sus resultados fueron bastante buenos, ya que obtuvieron el segundo mejor resultado con el corpus Rotten Tomatoes (un set de datos de críticas de películas) por detrás del modelo *CNN-MC* (una red neuronal convolucional multicanal), el cual necesita 18 veces más tiempo de entrenamiento. Además de esto, sus resultados con otros sets de datos fueron bastante cercanos a los del estado del arte de ese momento y el tiempo de entrenamiento era mucho menor que los modelos con mejores resultados, habiendos realizado todo el entrenamiento del modelo en un portátil con un solo core.

CAPÍTULO 4
Herramientas y metodología

En este capítulo se presenta algunas de las características principales de Twitter, la plataforma sobre la que vamos a trabajar, dónde almacenaremos nuestros *tweets*, cómo son estos datos que vamos a almacenar, cómo los vamos a obtener centrándonos en librerías concretas y las herramientas que utilizaremos para obtener los *word embeddings* contextuales e incontextuales.

4.1 Componentes principales

4.1.1. Twitter

Twitter es una red social enfocada al *microblogging* formada por *tweets*, en los que los usuarios pueden dar su opinión o escribir aquello que deseen. Esta red social se considera de *microblogging* debido a que los *tweets* tienen una limitación en longitud (280 caracteres desde Noviembre de 2017, siendo 140 caracteres hasta ese momento). Un *tweet* puede contener menciones a otros usuarios, *hashtags*, *emojis*, imágenes, enlaces, GIFs, vídeos, localización y encuestas. Además de todo esto, Twitter posee otras muchas funcionalidades como crear hilos, mandar mensajes privados, etcétera. Además tiene muchas opciones de configuración y personalización que no se consideran relevantes en este trabajo.

4.1.2. MongoDB

MongoDB es un sistema de bases de datos multiplataforma y NoSQL, concretamente de tipo orientado a documentos y de código abierto en que los datos son almacenados en formato *BSON* (*Binary JSON*). Dentro de una base de datos, MongoDB nos permite tener distintas colecciones, lo que permite tener diferentes tipos de datos en una misma base de datos sin ningún problema. En nuestro caso nos decantamos por usar este tipo de base de datos ya que nos ofrece una mayor flexibilidad a la hora de tratar con los diferentes *tweets* y debido a que tratar los *tweets* como documentos resulta bastante cómodo al utilizar el lenguaje de programación Python.

4.1.3. Python

Python [34] es un lenguaje de programación interpretado, dinámico y multiplataforma que admite diferentes paradigmas de programación y posee una ventaja con respecto al resto de lenguajes de programación en su gran cantidad de librerías, las cuales permiten trabajar con redes neuronales y aportan una gran cantidad de opciones entre las cuales elegir.

Cabe destacar que para el uso de Python nos hemos decantado por utilizar Anaconda [35], una distribución gratuita de código abierto para diferentes tareas de computación científica tanto en Python como R que permite simplificar los problemas que surgen al trabajar con entornos de trabajo y librerías con diferentes versiones. De esta forma, Anaconda nos ha servido para establecer el entorno de trabajo y poder instalar las diferentes librerías de Python sin tener problemas de librerías ni ser administrador en la máquina en la que se ha realizado este trabajo.

4.1.4. Keras

Keras [36] es una librería de código abierto de *Machine Learning* escrita en Python y que hemos utilizado para modelar nuestras *DAN* además de TWilBERT, nuestra implementación de *BERT* que mencionaremos con más detalles en el apartado 4.5. Además, el entrenamiento de los modelos también se ha llevado a cabo con Keras y para realizarlo utiliza una librería de bajo nivel que hace la función de *backend* como pueden ser TensorFlow [37], PlaidML [38], etc. En nuestro caso esta librería de bajo nivel ha sido TensorFlow.

4.2 Datos

Los datos utilizados para realizar la evaluación de los modelos presentados en este trabajo serán *tweets*. Estos *tweets* pueden tener características muy diferentes, desde utilizar buena ortografía y seguir las reglas gramaticales y de puntuación hasta no seguir ninguna regla y utilizar diferentes abreviaciones, de tal forma que tendremos que considerarlo a la hora de trabajar con estos datos. A parte de la estructura textual y la expresión lingüística del *tweet*, también tendremos que entender cómo es la estructura del *tweet* de forma computacional al recolectarlos de Twitter.

Así, vamos a utilizar Python con diversas librerías que nos permiten obtener estos tweets y almacenarlos en nuestra MongoDB. De esta forma, utilizaremos la librería Tweepy [39] para así poder obtener los datos gracias a una *API* sencilla con la que desarrollamos un *streamer* que obtiene todos los *tweets* en castellano y de forma asíncrona, en tiempo real, que tengan al menos una *stopword* de una lista concreta. Realizamos la recolección de esta manera debido a que Tweepy nos permite añadir un parámetro adicional de idioma para filtrar así los *tweets*, ya que un *tweet* geolocalizado en España no tiene por qué estar en español. Así, los *tweets* que recibimos de Tweepy tienen todos una estructura semejante, y la de un *retweet* se correspondería con la que podemos observar en la Figura 4.1.

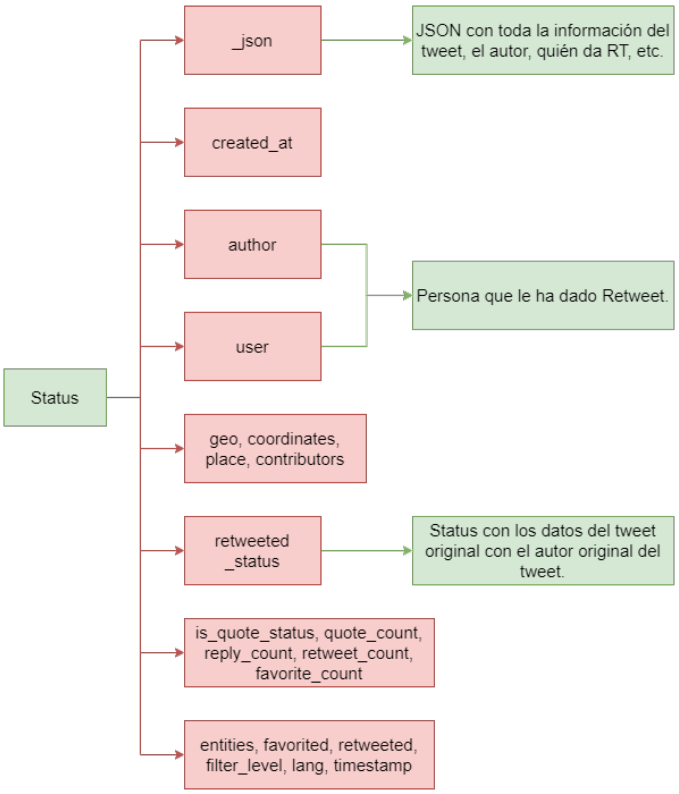

Figura 4.1: Ejemplo de la estructura de un *retweet* en Tweepy.

Dentro de toda esta cantidad de datos y objetos dentro de objetos, nosotros queremos almacenar el campo _json en MongoDB, para lo que utilizaremos PyMongo [40], una librería que nos permite trabajar con MongoDB y nos ofrece una forma sencilla de acceder a una base de datos en MongoDB. Una vez obtenemos el *tweet* con nuestro *streamer*, nos quedamos con el campo _json y al ser un *JSON* y tener este formato, Python lo interpreta como un diccionario y así le creamos un campo al diccionario que será el _id que tendrá el valor del id de nuestro *tweet*. Esto se debe a que MongoDB trabaja con documentos que indexa con el campo _id y como queremos que nuestra base de datos esté indexada por el id de los *tweets* (clave única por cada dato) para poder buscarlos de esta forma. Finalmente introducimos el *tweet* en la colección de la base de datos que queremos al tener ya guardados los datos de acceso a esta base de datos.

4.3 Word2Vec

Una vez tenemos estos datos almacenados, el siguiente paso será obtener los *word embeddings*. Word2Vec es una herramienta que implementa las estrategias *Skip-gram* y *CBOW*, para obtener *word embeddings* incontextuales. Tenemos diferentes parámetros a escoger dentro de esta herramienta entre los que podemos encontrar:

- La tarea de entrenamiento, pudiendo escoger entre *CBOW* o *SG*. Según indican los autores, el modelo obtenido con la tarea *CBOW* se entrena más rápido, mientras que el obtenido gracias a *SG* es más preciso con palabras poco frecuentes.
- El algoritmo de entrenamiento, donde nos encontramos con *negative sampling* y *softmax* jerárquico. Aquí, los autores explican que *softmax* jerárquico funciona mejor con palabras poco frecuentes y por el contrario, *negative sampling* es más útil con palabras frecuentes y vectores con pocas dimensiones.

- El submuestreo de palabras frecuentes para reducir el tiempo de entrenamiento.
- El tamaño o dimensionalidad de los *word embeddings* que queremos obtener, que suelen ser mejores cuanto mayor sea su tamaño hasta cierto punto.
- El tamaño de ventana de contexto para el entrenamiento de los *embeddings*.

Esta herramienta está implementada en una gran cantidad de lenguajes de programación, pero nosotros nos decantamos por utilizar Python y por ello, utilizaremos la librería Gensim [41] para la utilización de Word2Vec en Python. Esta librería tiene una implementación eficiente que permite trabajar con datos sin requerir que se carguen en memoria principal, pudiendo trabajar con más datos de los que podríamos si se cargasen en memoria principal. También podemos trabajar de forma más eficiente y rápida con máquinas multicore gracias al uso de workers o hilos. Finalmente esta librería cuenta con parámetros adicionales como el número mínimo de ocurrencias de una palabra para considerarla en el vocabulario de entrenamiento o el factor de aprendizaje.

Esta librería guarda los *word embeddings* en disco y con el entrenamiento se generan tres objetos. El primero es el modelo de *embeddings* y los otros dos son ficheros auxiliares con formato npy denominados **trainables.npy** (la red neuronal usada para entrenar a los *embeddings*) y **wv.npy** (el diccionario que traduce las palabras a sus correspondientes *embeddings*). Debido a la extensión de estos ficheros, se genera una dependencia con otra librería de Python conocida como NumPy [42], una librería utilizada en computación ciéntifica con una gran cantidad de funcionalidades entre la que destaca el uso eficiente de vectores y matrices.

4.4 Preproceso de los datos para *word embeddings* incontextuales

Una vez tenemos los datos almacenados y hemos determinado la librería que utilizaremos para generar nuestros *word embeddings* incontextuales el siguiente paso es considerar qué preproceso vamos a realizar sobre nuestros datos. Para ello hemos decidido generar un fichero al que pondremos en cada línea el identificador del *tweet* seguido de una tabulación y el texto considerando si se trata de un *tweet* extendido o no y sustituyendo los múltiples espacios en blanco por uno solo.

Tras la generación de este documento que podemos considerar un primer fichero con el corpus inicial, generaremos otro fichero en el que realizaremos el preproceso concreto que hemos establecido para este trabajo. En primer lugar obtenemos el texto del *tweet* y lo separamos por espacios en blanco para obtener las diferentes palabras. Debido a que estamos tratando con *tweets*, las palabras que empiecen por "@" serán sustituidas por el término "user", las que empiecen por "#" serán sustituidas por "hashtag" y las que empiecen por "http" o "www." serán sustituidas por "url". Tras esto, el resto de palabras son normalizadas de tal forma que se eliminan los acentos para evitar el problema de personas que puedan escribir con acentos mientras que otros no lo hagan y se convierte toda la frase a minúsculas. Así generamos nuestro corpus de entrenamiento que utilizaremos para obtener los *word embeddings* correspondientes. Todo este proceso está esquematizado en la Figura 4.2.

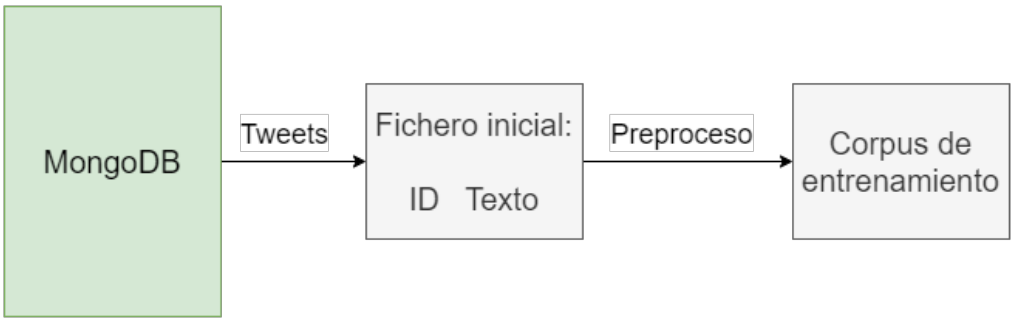

Figura 4.2: Generación del corpus de entrenamiento.

Para realizar este preproceso hemos utilizado la librería estándar de Python para expresiones regulares conocida como re [43], así como unicodedata [44] para tratar con la normalización de acentos, otra librería estándar dentro de Python. A parte de esto, para la generación de los ficheros hemos utilizado otra librería del estándar conocida como logging [45], para así poder observar las líneas procesadas al recorrer nuestra colección de *tweets* en MongoDB y poder monitorizar el tiempo de ejecución.

4.5 TWilBERT

Por último tenemos la preparación de los datos para el uso de nuestra implementación de *BERT*, TWilBERT [46] [47], la cual es una adaptación del modelo *BERT* con la intención de ser utilizada con *tweets* en español. Debido a cómo entrena el modelo *BERT*, que hemos explicado en la sección 3.3.2, el modelo necesita pares de frases para entrenar. Por ello, el fichero que hemos generado para obtener nuestros *word embeddings* incontextuales no nos sirve para entrenar a nuestro modelo de *word embeddings* contextuales. Así, y debido a que estamos tratando con *tweets*, hemos decidido utilizar un *tweet* y su respuesta como dato positivo y utilizar la respuesta y el *tweet* como dato negativo para entrenar al modelo. De esta forma, tendremos que considerar diferentes casos a la hora de generar este fichero y tratar con los *tweets*:

- Si nuestro *tweet* no es respuesta de otro *tweet* o tiene menos de 3 palabras, lo descartamos.
- Si nuestro *tweet* es respuesta de otro *tweet*, miramos si tenemos el *tweet* al que responde en nuestra base de datos y si lo tenemos utilizamos el par para entrenamiento. Sino, lo buscamos con Tweepy por Twitter. En caso de que el *tweet* ya no exista, se descarta el *tweet* respuesta y si sí existe, se utiliza el par para entrenamiento.

Con este proceso obtuvimos un corpus con un total de 94M de parejas de *tweets*, siendo la mitad de pares muestras positivas (*tweet*, respuesta) y la otra mitad muestras negativas (respuesta, *tweet*).

4.6 Recursos computacionales

Todo este proceso se ha realizado en la máquina Alzira, proporcionada por el departamento y que cuenta con un i7-6700 *quadcore*, una gráfica HD Graphics 530 y 16 GB de memoria principal.

Sin embargo, y debido a la cantidad de memoria, para obtener y trabajar con los *word embeddings* generados con 270M de tweets hemos tenido que utilizar una segunda máquina denominada Foto y también proporcionada por el departamento con un Intel Xeon E5-1660 *sixcore*, una gráfica GK106GL y la parte relevante, 64 GB de memoria principal.

CAPÍTULO 5
Trabajo experimental

En este capítulo se comenta el trabajo realizado dividiéndolo en las dos tareas del *NLP* sobre las que se ha trabajado: análisis de sentimientos y detección de emociones. Dentro de cada una se comentará los resultados del estado del arte, los datos con los que se ha trabajado para realizar el entrenamiento de los diferentes modelos con los que se ha trabajado. Finalmente, se comentarán los resultados obtenidos en la competición.

5.1 Análisis de sentimientos

El análisis de sentimientos o minería de opinión es una tarea del *NLP* centrada en identificar y extraer información subjetiva de diferentes datos sobre una entidad.

Utilizando terminología de la literatura [48], una entidad es un producto, servicio, tema, problema, persona, organización o evento. De este modo, una opinión está formada por una tupla de 5 elementos $(e_i, a_{ij}, s_{ijkl}, h_k, t_l)$, donde e_i es la entidad sobre la que se opina, a_{ij} es un aspecto concreto de la entidad e_i, s_{ijkl} es el sentimiento del aspecto a_{ij} que tiene el que opina (h_k) en el momento en que opina (t_l). Esta opinión s_{ijkl} puede ser medida como positiva, neutra o negativa o con diferentes niveles/intensidades, teniendo como ejemplo el uso de una puntuación de 1 a 5 como lo más usual al puntuar restaurantes, hoteles o aplicaciones.

Dentro de esta tarea, es bastante relevante el concepto de subjetividad. La subjetividad de una frase hace referencia a si la información presentada describe información del mundo o por el contrario, muestra los sentimientos, creencias o visión del sujeto acerca de este mundo. Este concepto marcará diferencias importantes entre que un dato concreto pertenezca a una clase o a otra y tenemos que considerarlo a la hora de considerar esta tarea.

Tras haber introducido conceptos relevantes y una introducción de la tarea, procedemos a comentar resultados del estado del arte para tener una referencia cuando analicemos los resultados que obtengamos en esta tarea con nuestros modelos.

5.1.1. Resultados del estado del arte

Centrándonos en resultados obtenidos en el análisis de sentimientos en español, vamos a introducir el Taller de Análisis Semántico en la SEPLN (TASS) [49], una competición que se lleva organizando desde el 2012 y en la que participan diversas universidades presentando sus mejores modelos cada año.

A continuación se detallan los resultados del TASS en estos últimos años, en concreto serán los resultados desde la edición del año 2017 hasta la del año pasado, 2019.

De esta forma, en primer lugar tenemos los resultados del año 2017 en [50], donde utilizaron 3 corpus distintos (InterTASS, corpus general del TASS y corpus general del TASS 1k) para la tarea de análisis de sentimientos, que contaba con 4 clases (Positivo, Neutro, Negativo y None). Para el corpus InterTASS el equipo ELiRF-UPV obtuvo un *F1* de 0.493 y una *accuracy* de 0.607. En el corpus general del TASS el mejor resultado fue de INGEOTEC-evodag con un *F1* de 0.577 y una *accuracy* de 0.645. Por último, en el corpus general del TASS 1k el mejor sistema fue el de RETUYT-svm con un *F1* de 0.562 y una *accuracy* de 0.7.

Situándonos en el 2018, el TASS [51] decidió introducir cambios, de tal forma que decidieron separar el análisis de sentimientos en diferentes variantes, introduciendo la variante de Español (ES), Costa Rica (CR) y Perú (PE). Así, este año tuvieron estos tres corpus de diferentes variantes con las mismas clases que el año anterior. Para la variante ES, el mejor sistema, de ELiRF-UPV, obtuvo un *F1* de 0.503 y una *accuracy* de 0.612. En cuanto a la variante CR, el mejor sistema fue el de RETUYT-LSTM con un *F1* de 0.504 y una *accuracy* de 0.537. Finalmente está la variante PE, para la que RETUYT-CNN obtuvo el mejor resultado con un *F1* de 0.472 y una *accuracy* de 0.494.

Por último tenemos el 2019 [52], año en que añadieron la variante de uruguay (UY) y méxico (MX) a las tres variantes que ya habían introducido anteriormente en 2018. Este año utilizaron las cuatro clases que habían utilizado anteriormente. De esta forma, en la variante ES ganó ELiRF-UPV con un *F1* de 0.507, en la variante CR el mejor resultado fue de RETUYT-InCo con un *F1* de 0.512 y en la variante PE Atalaya obtuvo un *F1* de 0.454. Además, en las nuevas variantes ganó ELiRF-UPV con un *F1* de 0.515 para la variante UY y un *F1* de 0.501 en MX.

La tabla 5.1 muestra los mejores resultados obtenidos en los diferentes corpus del TASS en ediciones anteriores:

Corpus	*F1-Score*
InterTASS	0.493
General	0.577
General 1k	0.562
Variante ES	0.507
Variante CR	0.512
Variante PE	0.472
Variante UY	0.515
Variante MX	0.501

Tabla 5.1: Tabla con los mejores resultados en ediciones anteriores del TASS.

5.2 Experimentación

Tras haber introducido los resultados obtenidos anteriormente en análisis de sentimiento y antes de adentrarnos en el trabajo realizado, debemos definir las medidas de evaluación que se han utilizado y se utilizan hoy en día para medir la calidad de los diferentes modelos.

En primer lugar tenemos el *recall* (ecuación 5.1) y la *precision* (ecuación 5.2), que son las primeras medidas y se basan en utilizar la clasificación de datos según cuatro tipos: *True Positive* (*TP*), *True Negative* (*TN*), *False Positive* (*FP*) y *False Negative* (*FN*).

$$Recall = \frac{TP}{TP + FN} \tag{5.1}$$

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

Además de estas dos, tenemos la *accuracy*, con la siguiente fórmula:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.3}$$

La última medida, que se basa en el *recall* y la *precision*, es el *F1-Score*, que es una media armónica de estas dos y se calcula con la siguiente fórmula:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{5.4}$$

Por último, y dentro de cualquier medida, se puede calcular utilizando la micro o macro media. La macro media calcula la media tratando a todas las clases por igual, mientras que la micro media tiene en cuenta la contribución de cada clase, funcionando mejor cuando tenemos clases desequilibradas.

Ahora que ya hemos introducido las medidas de evaluación, pasamos a establecer los resultados que hemos obtenido en este trabajo. En primer lugar, vamos a centrarnos en el corpus del TASS para la tarea de análisis de sentimiento, que ha sido utilizado para entrenar y evaluar los modelos.

Clase	Español	Costa Rica	Perú	Uruguay	México
N	475	310	228	367	505
NEU	297	246	522	286	172
P	354	221	216	290	313
Σ	1126	777	966	943	990

Tabla 5.2: Distribución de los datos de entrenamiento del TASS para la tarea de análisis de sentimientos.

Como podemos ver en la tabla 5.2, este año solo tenemos tres clases en las que clasificar nuestros datos, habiendo unificado la clase NONE a la clase NEU en una única. Esto se debe a que había mucha controversia en la utilidad práctica de distinguir entre estas dos clases. Además, en la Tabla 5.3 podemos ver ejemplos concretos de cada clase.

Texto	Clase
@morbosaborealis jajajaja... eso es verdad... aquí no hay uno cuerdo	N
@Adriansoler espero y deseo que el interior te cause lo mismo	NEU
@daniacal aún no, pero si estará jugable en el TGS no creo que tarde demasiado.	P

Tabla 5.3: Ejemplos concretos de datos del corpus de entrenamiento en Español para la tarea de análisis de sentimientos del TASS.

Tras haber introducido la distribución del corpus de entrenamiento y sus características, además de algunos ejemplos de cada clase, se procede a explicar el proceso de entrenamiento con el que se han obtenido los mejores modelos para cada variante.

Para explicar la metodología en el proceso de entrenamiento, primero tengo que establecer un problema que surge con la metodología que estamos utilizando. En este caso, estoy hablando de la aleatoriedad que tenemos en nuestro modelo. En primer lugar tenemos el hecho de que algunas operaciones con matrices son no deterministas para ser más eficientes en GPU y en segundo lugar tenemos el concepto de *dropout* que hemos establecido anteriormente y que nos aportará también diferentes resultados para diferentes ejecuciones. Estos dos conceptos son vitales para el buen funcionamiento de nuestros modelos, pero también implican que nuestro modelo puede dar resultados distintos con la misma configuración de hiperparámetros.

Con todo esto, hemos utilizado una configuración de malla para la búsqueda de los hiperparámetros de las *DAN* y hemos ejecutado 5 veces cada configuración de la malla. De esta forma, hemos obtenido para cada configuración de la malla su mejor resultado tras las 5 iteraciones y luego hemos seleccionado los 6 modelos con los mejores resultados. Tras esto, hemos ejecutado otras 3 iteraciones con las 6 configuraciones seleccionadas, actualizando los modelos para finalmente utilizar el modelo con el mejor resultado y utilizarlo en el TASS. Los parámetros a configurar en nuestras *DAN* son:

- En primer lugar tenemos el número de capas y número de neuronas por capa, que definen la topología de nuestra *DAN*. Para estos hiperparámetros hemos utilizado las tres siguientes configuraciones: 1 capa con 128 neuronas, 2 capas con 64 neuronas y 2 capas con 128 neuronas. Esto se debe a que las pruebas con otras configuraciones no obtuvieron resultados muy prometedores y que el número de muestras de la tarea no era muy elevado.

- En segundo lugar tenemos el *hidden dropout* y el *input dropout*, que hacen referencia a la probabilidad de ignorar la salida de una neurona y la probabilidad de ignorar la entrada de una palabra para el cálculo del vector media respectivamente. Ambos hiperparámetros se han ido variando desde un 10 % a un 50 %, habiendo probado con un 10 %, 15 %, 20 %, 30 %, 40 % y 50 %.

- Después tenemos la normalización, que hace referencia al proceso de escalar los datos de entrada a cada capa oculta para que todos ellos tengan una magnitud parecida y así el proceso de entrenamiento por descenso del gradiente actualice los parámetros de la red a la hora de minimizar el error. Dentro de la literatura consideramos imprescindible en relación a este concepto, introducir el trabajo de Sergey Ioffe y Christian Szegedy [53], en el que introducen el *Batch Normalization*, una técnica con la cual buscan minimizar el *"internal covariate shift"*, de forma que el entrenamiento se vea acelerado y estabilizado, permitiendo que se utilice un factor de aprendizaje más grande y que la inicialización de los pesos del modelo sea más irrelevante y no dañe el aprendizaje del modelo. Además, también indican que esta técnica actúa como un regularizador, permitiendo eliminar el *dropout* sin dañar los resultados o el aprendizaje del modelo en algunos casos. Con todo esto, y centrándonos en nuestros modelos, hemos optado por utilizar ambas opciones, modelos con y sin normalización.
- Y por último tenemos el tamaño del *batch*, que es el número de muestras o datos que se procesan antes de que el modelo actualice sus pesos. En este caso hemos utilizado los tamaños 8, 16, 32 y 64.

Con estos parámetros y sus posibles combinaciones obtenemos un total de 144 configuraciones posibles.

5.3 Análisis de resultados

En este apartado vamos a establecer los resultados obtenidos en entrenamiento y en test. Después compararemos nuestros modelos con el resto de modelos del TASS en test y también específicamente nuestro modelo incontextual con nuestro modelo contextual. Finalmente haremos una comparativa de los resultados en entrenamiento de nuestro modelo incontextual utilizado en el TASS con el generado posteriormente con 270M de *tweets*.

5.3.1. Comparativa con TWilBERT

Esta comparativa es uno de los objetivos marcados dentro de este trabajo y se corresponde con comprobar la diferencia entre un modelo contextual y uno incontextual, para así considerar el avance que suponen los modelos incontextuales en el campo del *NLP*.

De este modo, podemos observar en la tabla 5.4, 5.5 y 5.6 los resultados obtenidos en el TASS por nuestros modelos.

Variante	*F1-Score*	*Precision*	*Recall*
ES	0.583382	0.583173	0.583593
CR	0.557537	0.556248	0.558832
PE	0.564979	0.596210	0.536857
UY	0.577443	0.578054	0.576832
MX	0.556748	0.556039	0.557460

Tabla 5.4: Tabla con los resultados en el TASS para la tarea de análisis de sentimientos con nuestro modelo Word2Vec + *DAN*.

Variante	*F1-Score*	*Precision*	*Recall*
ES	0.654101	0.656368	0.651849
CR	0.626343	0.630497	0.622243
PE	0.635577	0.672373	0.602600
UY	0.626769	0.635285	0.618479
MX	0.618262	0.614121	0.622458

Tabla 5.5: Tabla con los resultados en el TASS para la tarea de análisis de sentimientos con nuestro primer modelo TWilBERT.

Variante	*F1-Score*	*Precision*	*Recall*
ES	0.671147	0.672677	0.669624
CR	0.646364	0.646510	0.646218
PE	0.633887	0.635119	0.632659
UY	0.654733	0.667486	0.642457
MX	0.634451	0.637033	0.631889

Tabla 5.6: Tabla con los resultados en el TASS para la tarea de análisis de sentimientos con nuestro segundo modelo TWilBERT.

Estos resultados nos muestran la capacidad que tienen los modelos contextuales para representar el lenguaje, así como justifican el avance que ha habido en el estado del arte estos últimos años. Sin embargo, y observando los resultados obtenidos por nuestro modelo incontextual, podemos ver resultados muy competitivos que superan ampliamente a los mejores resultados de años anteriores en todas las variantes. Como ya hemos indicado anteriormente, esto se debe en parte a la unión de la clase NONE y NEU, que implica una simplificación de la tarea e implica una obtención de mejores resultados.

Por último tenemos la siguiente tabla, en la que hemos indicado el puesto en que han quedado nuestros modelos con respecto al TASS, habiendo en este un total de 12 modelos presentados por las diferentes universidades.

Variante	Word2Vec + *DAN*	TWilBERT 1	TWilBERT 2
ES	5/12	4/12	1/12
CR	5/12	4/12	1/12
PE	5/12	1/12	2/12
UY	5/12	4/12	3/12
MX	5/12	4/12	1/12

Tabla 5.7: *Ranking* de nuestros modelos, para las diferentes variantes del Español, en la tarea de análisis de sentimientos del TASS.

5.3.2. Comparativa con Word2Vec 270M

Esta comparativa es otro de los objetivos marcados en este trabajo. Así, vamos a comprobar la mejora obtenida por el modelo incontextual al aumentar el número de datos con los que se genera los *word embeddings*, comprobando cómo de relevantes son y cuantificando la potencia que pueden obtener.

De este modo, los resultados de nuestros modelos incontextuales en entrenamiento fueron:

Variante	*F1-Score*	*Precision*	*Recall*	*Accuracy*
ES	0.5918	0.5937	0.5945	0.6179
CR	0.6152	0.6182	0.6170	0.6205
PE	0.5902	0.5912	0.5914	0.6305
UY	0.6216	0.6247	0.6219	0.6336
MX	0.6330	0.6396	0.6302	0.6784

Tabla 5.8: Tabla con los resultados en entrenamiento de los mejores modelos con 70M de *tweets*.

Variante	F1-Score	Precision	Recall	Accuracy
ES	0.6146	0.6179	0.6125	0.6403
CR	0.6377	0.6439	0.6368	0.6410
PE	0.6074	0.6039	0.6114	0.6406
UY	0.6503	0.6504	0.6510	0.6573
MX	0.6385	0.6477	0.6326	0.6863

Tabla 5.9: Tabla con los resultados en entrenamiento de los mejores modelos con 270M de tweets.

Como se puede observar, el uso de una cantidad de datos casi 4 veces mayor solo obtiene un incremento máximo de 0.03 para la variante UY. Estos resultados nos muestran como los modelos incontextuales están limitados por la reducida capacidad (parámetros) de los modelos más que por la cantidad de datos de entrenamiento, a diferencia de los modelos contextuales basados en arquitecturas mucho más complejas.

5.4 Detección de emociones

La detección de emociones es una tarea del *NLP* consistente en determinar la emoción o emociones presentes en un texto, en nuestro caso en un tweet.

Una emoción es un pensamiento o sentimiento subjetivo. Las emociones han sido estudiadas en muchos campos, como podría ser en la psicología, la filosofía o la sociología. Además de estudiarse en diversos campos, también se ha contemplado desde diferentes aspectos, considerando respuestas emocionales, reacciones fisiológicas, expresiones faciales, gestos, posturas, etcétera. Sin embargo, cuando nos centramos en las emociones desde el punto de vista del aprendizaje automático y el procesamiento del lenguaje natural, nos centramos más en la expresión que se hace de estos plasmada en un texto.

Dentro de las emociones, muchos científicos han intentado clasificar las emociones generando las diferentes emociones que hay, sin embargo nunca se ha llegado a un consenso, lo que complica la tarea ya que las emociones pueden ser divididas en muchos grupos distintos. A pesar de esto, hay emociones que siempre suelen considerarse como podrían ser la tristeza, el enfado, el miedo o la alegría.

Las emociones también pueden ser consideradas en una escala de intensidad como se hacía con los sentimientos, separando las emociones en una evaluación racional, en la que se especifica creencias tangibles o actitudes utilitarias y en una evaluación emocional,

que hacen referencia a emociones no tangibles que se forman a partir del estado mental de las personas.

Con esto hemos introducido la tarea de detección de emociones además de algunas definiciones formales íntimamente relacionadas y ahora procedemos a establecer el estado del arte en esta tarea.

5.4.1. Resultados del estado del arte

En este apartado hablaremos del SemEval (*Semantic Evaluation*) [54], una competición de sistemas de análisis semántico computacional con una gran cantidad de tareas diferentes y que cuenta con 14 ediciones hasta la fecha (2020), siendo la primera de ellas en 1998. Entre estas tareas, nos centraremos en la tarea 1 realizada en el año 2018 [55] [56], denominada *"Affect in Tweets"* en la que se presentaron 5 subtareas relacionadas con detección de emociones. Especialmente relacionada con el trabajo desarrollado, resulta de interés la última, denominada *Emotion Classification* (*E-c*) y que consistía en clasificar un *tweet* como neutral (sin emoción) o dentro de una o más de las 11 emociones que representaba el estado mental del usuario que había *tweeteado* el *tweet* concreto. Estas once emociones que ellos consideraron para la tarea fueron:

- **Enfado**: que también incluye rabia o molestia.
- **Anticipación**: considerando también interés o vigilancia.
- **Asco**: incluyendo desinterés, disgusto o aversión.
- **Miedo**: uniéndolo a ansiedad, preocupación, terror y aprensión.
- **Alegría**: además de serenidad y éxtasis.
- **Amor**: y también afección.
- **Optimismo**: o también confianza o esperanza.
- **Pesimismo**: incluyendo falta de confianza o escepticismo.
- **Tristeza**: uniéndola a estar pensativo y al dolor o pena.
- **Sorpresa**: que va unido a la distracción o el asombro.
- **Confianza**: junto a la admiración, la aceptación o el agrado.

Para evaluar los sistemas de la competición, se emplearon como medidas de evaluación la *accuracy*, y la *Micro-F1* y la *Macro-F1* como medidas secundarias. En la Tabla 5.10 se muestran los resultados obtenidos por los tres mejores modelos en español.

Nombre del equipo	*Accuracy*	*Micro-F1*	*Macro-F1*
MILAB_SNU	0.469	0.558	0.407
ELiRF-UPV	0.458	0.535	0.440
Tw-StAR	0.438	0.520	0.392

Tabla 5.10: Resultados de los 3 mejores sistemas en la tarea *Emotion Classification* del SemEval 2018.

5.5 Experimentación

Tras hablar de resultados obtenidos en años anteriores en el SemEval, vamos a centrarnos en el corpus del TASS para la tarea de detección de emociones, que ha sido utilizado para entrenar y evaluar los modelos.

Sentimiento	Nº de *tweets*
Alegría	1270
Tristeza	706
Enfado	600
Sorpresa	241
Asco	113
Miedo	67
Otros	2889
Total	5886

Tabla 5.11: Distribución de los datos de entrenamiento en la tarea de detección de emociones del TASS.

Como podemos ver en la Tabla 5.11, la distribución de datos por clases está bastante más desequilibrada que la distribución que teníamos en análisis de sentimientos. Por último y en relación a los datos, en la Tabla 5.12 tenemos algunos ejemplos de cada clase.

Texto	Sentimiento
Messi, maldito animal!! Que golazo! HASHTAG	Alegría
"¿Qué si estoy triste por HASHTAG? Cómo no voy a estar triste si ahí fue donde me bautizaron... (Me dueles Paris) ...Snif snif"	Tristeza
Hay dembele hijo mío que malo eres!!#ChampionsLeague	Enfado
Acabo de ver el mejor capítulo de la historia de juego de tronos. Sin palabras. HASHTAG	Sorpresa
"HASHTAG 8 navajazos en el pecho dolerían menos que ver esto"	Asco
Tengo miedo del capítulo de hoy HASHTAG	Miedo
Y si, siempre se cumple la inexorable “ley del ex”, el 9 USER HASHTAG	Otros

Tabla 5.12: Ejemplos concretos de datos del corpus de entrenamiento en Español para la tarea de detección de emociones del TASS.

De esta forma, el proceso de entrenamiento de los modelos será el mismo que en análisis de sentimientos y utilizaremos las mismas variaciones de los parámetros de las *DAN*. Obtendremos después de 5 iteraciones de entrenamiento los 6 mejores modelos y los ejecutaremos otras 3 veces para así obtener nuestro mejor modelo y hacer las predicciones de los datos de test con este modelo.

5.6 Análisis de resultados

En este apartado hablaremos de los resultados obtenidos en detección de emociones. Primero compararemos los resultados en test de nuestro modelo incontextual con nuestro modelo contextual para después comparar ambos con el resto de modelos del TASS. En segundo y último lugar, compararemos nuestro modelo incontextual en entrenamiento con el modelo generado posteriormente con 270M de tweets.

5.6.1. Comparativa con TWilBERT

En primer lugar tenemos la comparación entre nuestro modelo contextual e incontextual, teniendo también el *ranking* de nuestros modelos según su desempeño en el TASS y observaciones de los resultados que hemos obtenido.

Así, los resultados en test obtenidos en el TASS para detección de emociones están en la siguiente tabla:

Modelo	*F1-Score*	*Precision*	*Recall*
Word2Vec + DAN	0.431075	0.446321	0.416836
TWilBERT	0.446582	0.443422	0.449788

Tabla 5.13: Resultados en test de nuestros modelos en la tarea de detección de emociones del TASS.

Como se puede observar en la Tabla 5.13, los resultados de ambos modelos son bastante similares, al contrario de lo que observábamos en análisis de sentimientos. En la Tabla 5.14 se muestra el *ranking* de nuestros sistemas sobre el conjunto de test de la tarea de detección de emociones, considerando que había 8 modelos que participaban en total.

Tarea	Word2Vec + *DAN*	TWilBERT
Detección de emociones	2/8	1/8

Tabla 5.14: *Ranking* de nuestros sistemas sobre el conjunto de test en la tarea de detección de emociones.

Podemos comprobar que nuestros modelos son los mejores en esta tarea, llevándose el primer y segundo puesto en comparación al resto de modelos del TASS. Esto nos muestra que un modelo basado en *word embeddings* incontextuales puede conseguir resultados comparables a un modelo contextual en algunos casos.

Como detalle a recalcar, aunque podríamos considerar que los resultados de esta tarea deberían superar ampliamente a los del SemEval 2018 debido a que el número de sentimientos, y por tanto, clases, es menor, en la práctica no es así. De esta forma, si miramos la única medida que comparten (el Macro-F1 de la Tabla 5.10 se corresponde con el F1-Score de la Tabla 5.13) nos indica que a pesar de que la tarea del SemEval del 2018 es más compleja y se realizó hace 2 años, los resultados que se obtuvieron son similares a los que obtiene nuestro mejor modelo contextual, teniendo este un *F1* de 0.446 y el mejor modelo en esta medida del SemEval (ELiRF-UPV) un *F1* de 0.440.

5.6.2. Comparativa con Word2Vec 270M

En esta otra subsección del trabajo comparamos al modelo incontextual con otra versión ampliada, igual que hacíamos con la tarea de análisis de sentimientos. Así, podemos comprobar en la Tabla 5.15 los resultados sobre el conjunto de entrenamiento de nuestros dos modelos incontextuales.

Modelo	*F1-Score*	*Precision*	*Recall*	*Accuracy*
70M	0.5438	0.5915	0.5277	0.6534
270M	0.5430	0.5947	0.5185	0.6593

Tabla 5.15: Tabla comparativa entre los resultados en entrenamiento de nuestros modelos incontextuales para detección de emociones.

En este caso observamos como utilizar más *tweets* para generar los *word embeddings* incontextuales no mejora el resultado en detección de emociones, al igual que utilizar *word embeddings* contextuales tampoco resultaba una diferencia muy grande con un modelo incontextual.

CAPÍTULO 6
Conclusiones y trabajo futuro

En este último capítulo comentaremos las conclusiones de este trabajo, además de especificar ciertas propuestas y mejoras relacionadas con el trabajo realizado, que resultan de gran interés para futuros proyectos.

6.1 Conclusiones

El trabajo que se presenta ha cumplido con los objetivos planteados a su inicio. Se ha obtenido una colección de *tweets* que en su primera versión consta de 70M de *tweets* y finalmente se ha conseguido ampliar a 270M. Durante este proceso hemos tenido algunas dificultades relacionadas con el mecanismo elegido para almacenar los *tweets* en la base de datos inicial. Finalmente se ha conseguido una representación adecuada para el volumen de datos necesitado. En ese sentido, ha sido de gran utilidad los conocimientos previos que se tenían sobre bases de datos y su aplicación en concreto a una base de datos no relacional.

Se ha realizado las comparativas experimentales planteadas de forma exitosa, cumpliendo los objetivos establecidos. La realización de estas comparativas han fortalecido conocimientos relacionados con el preproceso de datos, además de diferentes técnicas del procesamiento del lenguaje natural.

Por último, consideramos que no se ha logrado totalmente el cuarto objetivo consistente en trabajar con modelos contextuales. A pesar de que sí que se han utilizado en dos tareas del *NLP*, no se ha realizado todo el proceso de entrenar los modelos contextuales, sino que el trabajo realizado se ha limitado a generar el corpus que ha servido de entrenamiento para estos modelos.

6.2 Trabajo futuro

Del desarrollo de este proyecto, se desprenden posibles líneas de trabajo para que futuros alumnos puedan continuar con este trabajo.

En primer lugar, queda pendiente la liberación del corpus de test, formado por pares (*tweet, gold standard*), lo que nos impide comprobar el desempeño que tendría nuestro modelo incontextual generado con 270M de *tweets* sobre este conjunto, para poder compararlo con el resto de enfoques de la competición. Por tanto, resulta de gran interés continuar analizando el comportamiento de los modelos incontextuales cuando estos son expuestos a cantidades crecientes de datos.

En segundo lugar, abrimos la puerta a la utilización de modelos contextuales para estas tareas de *NLP* en Twitter, más allá de TWilBERT, siendo intresante estudiar el comportamiento de más modelos contextuales. De esta manera, consideramos que la utilización de modelos incontextuales sería interesante para marcar una base de referencia o comprobar la calidad de los datos.

En tercer y último lugar, consideramos interesante el estudio de modelos contextuales en otras tareas de *NLP* en Twitter español, en las que aún no se han evaluado y analizado la mayor parte de los modelos contextuales, publicados recientemente para el idioma inglés y para dominios más generales.

Bibliografía

[1] Stephen Roller, Emily Dinan, Naman Goyal, Da Ju, Mary Williamson, Yinhan Liu, Jing Xu, Myle Ott, Kurt Shuster, Eric M. Smith, Y-Lan Boureau y Jason Weston. *Recipes for building an open-domain chatbot*. 2020. arXiv: 2004.13637 [cs.CL].

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever y Dario Amodei. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[3] Marie-Anne Lachaux, Baptiste Roziere, Lowik Chanussot y Guillaume Lample. *Unsupervised Translation of Programming Languages*. 2020. arXiv: 2006.03511 [cs.CL].

[4] Alfonso Ortega Giménez y Eduardo Lleida Solano. *Proyectos de EXCELENCIA y Proyecytos RETOS AGENCIA ESTATAL DE INVESTIGACIÓN: Análisis afectivo de información multimedia con comunicación inclusiva y natural*. http://dihana.cps.unizar.es/~alfonso/amic/index.html. 2017.

[5] Industria y Competitividad Ministerio de Economía. *Propuesta de resolución provisional del procedimiento de concesión de ayudas a proyectos de I+D+I correspondientes al programa estatal de investigación, desarrollo e innovación orientada a los retos de la sociedad, en el marco del plan estatal de investigación científica y técnica y de innovación 2013 -2016. CONVOCATORIA 2017*. https://sede.micinn.gob.es/stfls/eSede/Ficheros/2017/Propuesta_Resolucion_Provisional_Proyectos_Retos_2017.pdf. [Página 225]. 2017.

[6] Sebastian Ruder. *A Review of the Neural History of Natural Language Processing*. https://ruder.io/a-review-of-the-recent-history-of-nlp/. 2017.

[7] Departement Operationnelle, Y. Bengio, R Ducharme, Pascal Vincent y Centre Mathematiques. «A Neural Probabilistic Language Model». En: (oct. de 2001).

[8] Ronan Collobert y Jason Weston. «A unified architecture for natural language processing: Deep neural networks with multitask learning». En: ene. de 2008, págs. 160-167. DOI: 10.1145/1390156.1390177.

[9] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocký y Sanjeev Khudanpur. «Recurrent neural network based language model». En: vol. 2. Ene. de 2010, págs. 1045-1048.

[10] Sepp Hochreiter y Jürgen Schmidhuber. «Long Short-Term Memory». En: *Neural Comput.* 9.8 (nov. de 1997), 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[11] Alex Graves. *Generating Sequences With Recurrent Neural Networks*. 2013. arXiv: 1308.0850 [cs.NE].

[12] Y. Bengio, P. Simard y P. Frasconi. «Learning long-term dependencies with gradient descent is difficult». En: *IEEE Transactions on Neural Networks* 5.2 (1994), págs. 157-166.

[13] Tomas Mikolov, Kai Chen, Greg Corrado y Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[14] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado y Jeffrey Dean. *Distributed Representations of Words and Phrases and their Compositionality*. 2013. arXiv: 1310.4546 [cs.CL].

[15] Yoon Kim. «Convolutional Neural Networks for Sentence Classification». En: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, oct. de 2014, págs. 1746-1751. DOI: 10.3115/v1/D14-1181. URL: https://www.aclweb.org/anthology/D14-1181.

[16] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng y Christopher Potts. «Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank». En: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, oct. de 2013, págs. 1631-1642. URL: https://www.aclweb.org/anthology/D13-1170.

[17] Ilya Sutskever, Oriol Vinyals y Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014. arXiv: 1409.3215 [cs.CL].

[18] Dzmitry Bahdanau, Kyunghyun Cho y Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2014. arXiv: 1409.0473 [cs.CL].

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser e Illia Polosukhin. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[20] Jay Alammar. *The Illustrated Transformer*. https://jalammar.github.io/illustrated-transformer/. 2018.

[21] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee y Luke Zettlemoyer. *Deep contextualized word representations*. 2018. arXiv: 1802.05365 [cs.CL].

[22] Jeremy Howard y Sebastian Ruder. *Universal Language Model Fine-tuning for Text Classification*. 2018. arXiv: 1801.06146 [cs.CL].

[23] Jacob Devlin, Ming-Wei Chang, Kenton Lee y Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. arXiv: 1810.04805 [cs.CL].

[24] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer y Veselin Stoyanov. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: 1907.11692 [cs.CL].

[25] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma y Radu Soricut. *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations*. 2019. arXiv: 1909.11942 [cs.CL].

[26] Kevin Clark, Minh-Thang Luong, Quoc V. Le y Christopher D. Manning. *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators*. 2020. arXiv: 2003.10555 [cs.CL].

[27] New York University, ML^2, Washington University y DeepMind. *General Language Understanding Evaluation*. https://gluebenchmark.com/.

[28] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy y Samuel R. Bowman. *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding*. 2018. arXiv: 1804.07461 [cs.CL].

[29] Piotr Bojanowski, Edouard Grave, Armand Joulin y Tomas Mikolov. *Enriching Word Vectors with Subword Information*. 2016. arXiv: 1607.04606 [cs.CL].

[30] Jeffrey Pennington, Richard Socher y Christopher D. Manning. «GloVe: Global Vectors for Word Representation». En: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, págs. 1532-1543. URL: http://www.aclweb.org/anthology/D14-1162.

[31] Frederic Morin y Yoshua Bengio. «Hierarchical probabilistic neural network language model». En: *AISTATS'05*. 2005, págs. 246-252.

[32] Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber y Hal Daumé III. «Deep Unordered Composition Rivals Syntactic Methods for Text Classification». En: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, jul. de 2015, págs. 1681-1691. DOI: 10.3115/v1/P15-1162. URL: https://www.aclweb.org/anthology/P15-1162.

[33] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever y Ruslan Salakhutdinov. «Dropout: A Simple Way to Prevent Neural Networks from Overfitting». En: *J. Mach. Learn. Res.* 15.1 (ene. de 2014), 1929–1958. ISSN: 1532-4435.

[34] Guido van Rossum. *Python*. https://www.python.org/. 1991.

[35] Inc Anaconda. *Anaconda*. https://www.anaconda.com/. 2012.

[36] François Chollet. *Keras*. https://keras.io/. 2015.

[37] Google Brain. *TensorFlow*. https://www.tensorflow.org/. 2015.

[38] Vertex.AI. *PlaidML*. https://github.com/plaidml/plaidml. 2017.

[39] *Tweepy*. https://www.tweepy.org/. 2009.

[40] *PyMongo*. https://pymongo.readthedocs.io/en/stable/. 2009.

[41] Radim Řehůřek. *Gensim*. https://radimrehurek.com/gensim/models/word2vec.html. 2011.

[42] Travis Oliphant. *NumPy*. https://numpy.org/. 2006.

[43] *Re Documentation*. https://docs.python.org/3/library/re.html.

[44] *Unicodedata Documentation*. https://docs.python.org/3/library/unicodedata.html.

[45] *Logging Documentation*. https://docs.python.org/3/library/logging.html.

[46] José-Ángel González, Llu'ıs-F Hurtado y Ferran Pla. *TWilBERT*. https://github.com/jogonba2/TWilBert. 2020.

[47] José-Ángel González, Llu'ıs-F Hurtado y Ferran Pla. «TWilBert: Pre-trained Deep Bidirectional Transformers for Spanish Twitter». En: *To be decided* (Forthcoming).

[48] Bing Liu. *Sentiment Analysis and Opinion Mining*. 2012.

[49] Universidad de Jaén, Sistemas Inteligentes de Acceso a la Información (SINAI), Meaning Cloud y Sociedad Española para el Procesamiento del Lenguaje Natural (SEPLN). *TASS*. http://tass.sepln.org/. 2012.

[50] Eugenio Martínez-Cámara, Manuel Díaz-Galiano, Miguel García-Cumbreras, Manuel García-Vega y Julio Villena-Román. «Overview of TASS 2017». En: sep. de 2017.

[51] Eugenio Martínez-Cámara, Yudivián Almeida-Cruz, Manuel Carlos Díaz-Galiano, Suilan Estévez-Velarde, Miguel Ángel García Cumbreras, M. Vega, Yoan Gutiérrez, Arturo Montejo Ráez, A. Montoyo, R. Muñoz, Alejandro Piad-Morffis y Julio Villena-Román. «Overview of TASS 2018: Opinions, Health and Emotions». En: *TASS@SEPLN*. 2018.

[52] Manuel Carlos Díaz-Galiano, M. Vega, E. Casasola, Luis Chiruzzo, Miguel Ángel García Cumbreras, Eugenio Martínez-Cámara, D. Moctezuma, Arturo Montejo Ráez, Marco Antonio Sobrevilla Cabezudo, Eric Sadit Tellez, Mario Graff y Sabino Miranda-Jiménez. «Overview of TASS 2019: One More Further for the Global Spanish Sentiment Analysis Corpus». En: *IberLEF@SEPLN*. 2019.

[53] Sergey Ioffe y Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[54] ACL-SIGLEX. *SemEval*. https://www.aclweb.org/anthology/venues/semeval/. 1998.

[55] Saif Mohammad, Felipe Bravo-Marquez, Mohammad Salameh y Svetlana Kiritchenko. «SemEval-2018 Task 1: Affect in Tweets». En: *Proceedings of The 12th International Workshop on Semantic Evaluation*. New Orleans, Louisiana: Association for Computational Linguistics, jun. de 2018, págs. 1-17. DOI: 10.18653/v1/S18-1001. URL: https://www.aclweb.org/anthology/S18-1001.

[56] *SemEval-2018 Task 1: Affect in Tweets*. URL: https://competitions.codalab.org/competitions/17751.