UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

JIZT

Generación de resúmenes abstractivos en la nube mediante Inteligencia Artificial

Presentado por Diego Miguel Lozano
en la Universidad de Burgos — 29 de enero de 2021
Tutores: Dr. Carlos López Nozal y
Dr. José Francisco Díez Pastor

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. nombre tutor, profesor del departamento de nombre departamento, área de nombre área.

Expone:

Que el alumno D. Diego Miguel Lozano, con DNI 71307413-F, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado "JIZT - Generación de resúmenes abstractivos en la nube mediante Inteligencia Artificial.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 29 de enero de 2021

Vº. Bº. del Tutor:	Vº. Bº. del Tutor:
D. Carlos López Nozal	D. José Francisco Díez Pastor

Resumen

En este primer apartado se hace una **breve** presentación del tema que se aborda en el proyecto.

Descriptores

Palabras separadas por comas que identifiquen el contenido del proyecto Ej: servidor web, buscador de vuelos, android ...

Abstract

A **brief** presentation of the topic addressed in the project.

Keywords

keywords separated by commas.

Índice general

Índice de figuras

Índice de tablas

Introducción

El término Inteligencia Artificial (IA) fue acuñado por primera vez en la Conferencia de Dartmouth [1] hace ahora 65 años, esto es, en 1956. Sin embargo, ha sido en los últimos tiempos cuando su presencia e importancia en la sociedad han crecido de manera exponencial.

Uno de los campos históricos dentro de la AI, es el Procesamiento del Lenguaje Natural (NLP, por sus siglas en inglés), cuya significación se hizo patente con la aparición del célebre Test de Turing [2], en el cual un interrogador debe discernir entre un humano y una máquina conversando con ambos por escrito a través de una terminal.

Hasta los años 80, la mayor parte de los sistemas de NLP estaban basados en complejas reglas escritas a mano [3], las cuales conseguían generalmente modelos muy lentos, poco flexibles y con baja precisión. A partir de esta década, como fruto de los avances en Aprendizaje Automático (*Machine Learning*), fueron apareciendo modelos estadísticos, consiguiendo notables avances en campos como el de la traducción automática.

En la última década, el desarrollo ha sido aún mayor debido a factores como el aumento masivo de datos de entrenamiento (principalmente provenientes del contenido generado en la *web*), avances en la capacidad de computación (GPU, TPU, ASIC...) y el progreso dentro del área de la Algoritmia [4].

No obstante, ha sido desde la aparición del concepto de "atención" en 2015 [5, 6, 7] cuando el campo del NLP ha comenzado a lograr resultados cuanto menos sorprendentes [8, 9].

Con todo, la mayor parte de estos avances se han visto limitados al ámbito académico y empresarial. Los modelos cuyo código ha sido publicado, o bien no están entrenados, o bien requieren para ser usados conocimientos

avanzados de matemáticas o programación, o simplemente son demasiado grandes para ser ejecutados en ordenadores convencionales.

Con esta idea en mente, el objetivo de JIZT se centra en acercar los modelos NLP estado del arte tanto a usuarios expertos, como no expertos.

Para ello, JIZT proporciona:

- Una API REST destinada a los usuarios con conocimientos técnicos, a través de la cual se pueden llevar a cabo tareas de NLP.
- Una aplicación multiplataforma que consume dicha API, y que proporciona una interfaz gráfica sencilla e intuitiva. Esta aplicación puede ser utilizada por el público general, aunque no deja de ofrecer opciones avanzadas para aquellos usuarios con mayores conocimientos en la materia.

En un principio, dado el alcance de un Trabajo de Final de Grado, la única tarea de NLP implementada ha sido la de generación de resúmenes. La motivación para esta decisión se ha fundamentado principalmente en la relativa menor popularidad de esta área frente a otras como la traducción automática, el análisis de sentimientos, o los modelos conversacionales. Para estas últimas tareas existe actualmente una amplia oferta de grandes compañías como Google [10], IBM [11], Amazon [12], o Microsoft [13], entre muchas otras. Nuestra mayor limitación reside en que los modelos preentrenados que utilizaremos para la generación de los resúmenes funcionan únicamente en inglés. Esperamos que en un futuro próximo aparezcan modelos que admitan otros idiomas.

En un mundo en el que en cinco años se producirán globalmente 463 exabytes de información al día [14], siendo mucha de esa información textual, las generación de resúmenes aliviara en cierto modo el tratamiento de esos datos.

Sin embargo, gran parte del esfuerzo de desarrollo de JIZT se ha centrado en el diseño de su arquitectura, la cual se describirá con detalle en el capítulo de Conceptos Teóricos. Por ahora adelantaremos que ha sido concebida con el objetivo de ofrecer la mayor escalabilidad y flexibilidad posible, manteniendo además la capacidad de poder añadir otras tareas de NLP diferentes de la generación de resúmenes en un futuro cercano.

Por todo ello, el presente TFG conforma el punto de partida de un proyecto ambicioso, desafiante, pero con la certeza de que, independientemente de su recorrido, habremos aprendido, disfrutado, y ojalá ayudado a alguien por el camino.

Objetivos del proyecto

2.1. Objetivos generales

- Ofrecer la capacidad de llevar a cabo tareas de NLP tanto al público general, como al especializado. Como se ha mencionado con anterioridad, la única tarea NLP que implementará el presente TFG será la de generación de resúmenes.
- Emplear modelos pre-entrenados estado del arte para la generación de resúmenes abstractivos. Los resúmenes abstractivos se diferencian de los extractivos en que el resumen generado contiene palabras o expresiones que no aparecen en el texto original [15]. Dicho de forma más técnica, existe cierto nivel de paráfrasis.
- Diseñar una arquitectura con aspectos como la flexibilidad, la escalabilidad y la alta disponibilidad como principios fundamentales.
- Poner en práctica lo aprendido a lo largo de la carrera en áreas como Ingeniería del Software, Sistemas Distribuidos, Programación, Minería de Datos, Algoritmia y Bases de Datos.
- Ofrecer la totalidad del proyecto bajo licencias de *Software* Libre.

2.2. Objetivos técnicos

- Los modelos pre-entrenados de generación de texto admiten parámetros específicos para configurar dicha generación, por lo que se deberá

implementar una interfaz que permita a los usuarios establecer dichos parámetros de manera opcional. Por defecto, se proporcionarán los valores que mejores resultados ofrecen, extraídos mayoritariamente de manera experimental.

- Los modelos pre-entrenados de generación estado del arte presentan frecuentemente limitación en la longitud de los textos de entrada que reciben, derivada de la longitud de las secuencias de entrada con las que han sido entrenados. Esta longitud llega a ser tan baja como 512 *tókenes*[1] [16]. Por tanto, se deberá establecer algún mecanismo que permita sortear esta limitación para poder generar resúmenes de textos arbitrariamente largos.

- Gestionar el pre-procesado de los textos a resumir para ajustarlos a la entrada que los modelos pre-entrenados esperan.

- Algunos modelos pre-entrenados generan textos enteramente en minúsculas. Se deberá, por tanto, incluir mecanismos en la etapa de post-procesado que permitan recomponer la correcta capitalización de los resúmenes generados.

- Con el fin de cumplir con el objetivo general referente a la arquitectura, desarrollar una arquitectura de microservicios, basada en la filosofía *Cloud Native* [17, 18]. Este objetivo se divide a su vez en dos puntos:

  - Encapsular cada microservicio en un contenedor Docker.
  - Implementar la orquestración y balanceo de los microservicios a través de Kubernetes.

- Complementariamente al punto anterior, implementar una arquitectura dirigida por eventos [19]. La motivación detrás de la utilización de este patrón arquitectónico se justifica en el capítulo de Conceptos Teóricos.

- Implementar una API REST escrita en Python empleando el *framework web* Flask. Dicha API será el punto de conexión con el servicio de generación de resúmenes en la nube.

- Desplegar PostgreSQL como servicio en Kubernetes mediante el Operador PostgreSQL de Crunchy [20]. Esta base de datos cumplirá la doble función de (a) servir como caché para no volver a producir resúmenes ya generados con anterioridad, incrementando la velocidad

[1] Este término se definirá posteriormente. Por ahora, el lector puede considerar que un *tóken* es equivalente a una palabra.

de respuesta, y (b) almacenar los resúmenes generados con fines de evaluación de la calidad de los mismos y extracción de métricas.

- Desarrollar, con ayuda de Flutter, una aplicación multiplataforma con soporte nativo para Android, iOS, y *web*. Esta aplicación consumirá la API y proporcionará una interfaz gráfica sencilla e intuitiva para que usuarios regulares puedan hacer uso del servicio de generación de resúmenes.

Conceptos teóricos

En este capítulo, detallaremos de forma teórica el proceso de generación de resúmenes, desde el momento que recibimos el texto a resumir, hasta que se le entrega al usuario el resumen generado. En el siguiente capítulo, explicaremos las herramientas que hacen posible que todo este proceso se pueda llevar a cabo de forma distribuida «en la nube».

La generación de resúmenes se divide en cuatro etapas fundamentales:

- Pre-procesado.
- Codificación.
- Generación del resumen.
- Post-procesado.

Veamos en detalle en qué consiste cada una de ellas.

3.1. Pre-procesado del texto

El principal objetivo de esta etapa es adecuar el texto de entrada para que se aproxime lo máximo posible a lo que el modelo espera. Adicionalmente, se separa en texto de entrada en frases. Esta separación parecer *a priori* una tarea trivial, pero involucra una serie de dificultades que se detallarán a continuación.

Cabe destacar que, como mencionábamos en la Introducción, los modelos pre-entrenados de los que hacemos uso solo admiten textos en inglés, por

lo que algunas de las consideraciones que tomamos en el pre-procesado del texto solo son aplicables a este idioma.

A grandes rasgos, en la etapa de pre-procesado se divide a su vez en los siguientes pasos:

- Eliminar retornos de carro, tabuladores (`\n`, `\t`) y espacios sobrantes entre palabras (p. ej. `"I  am"` → `"I am"`).

- Añadir un espacio al inicio de las frases intermedias (p. ej.: `"How's it going?Great!"` → `"How's it going? Great!"`. Esto es especialmente relevante en el caso de algunos modelos, como por ejemplo BART [21], los cuales tienen en cuenta ese espacio inicial para distinguir entre frases iniciales y frases intermedias en la generación de resúmenes[2].

- Establecer un mecanismo que permita llevar a cabo la ya mencionada separación del texto en frases. Esto es importante dado que los modelos tienen un tamaño de entrada máximo. Dos estrategias comunes para eludir esta limitación consisten en (a) truncar el texto de entrada, lo cual puede llevar asociado pérdidas notables de información, o (b) dividir el texto en fragmentos de menor tamaño. En nuestro caso, la primera opción quedó rápidamente descartada ya que los textos que vamos a recibir, por lo general, superarán el tamaño máximo (en caso contrario tendría poco sentido querer generar un resumen). Refiriéndonos, por tanto, a la segunda opción, es frecuente llevar a cabo dicha separación de manera ingenua, únicamente atendiendo al tamaño de entrada máximo. Sin embargo, en nuestro caso decidimos refinar este proceso e implementamos un algoritmo original[3] en el que dicha separación se realiza de tal modo que ninguna frase queda dividida. Para garantizar el éxito de este algoritmo, es fundamental que las frases estén correctamente divididas; el porqué se clarificará en la siguiente sección, referente a la codificación del texto.

A continuación, nos centraremos en el proceso de división del texto en frases. A la hora de llevar a cabo este proceso, debemos tener en cuenta que el texto de entrada podría contener errores ortográficos o gramaticales,

[2] Por el momento, no hacemos uso de este modelo, aunque podría incluirse en el futuro.

[3] Utilizamos el término «original» porque no encontramos ningún recurso en el que se tratara este problema, por lo que tuvimos que resolverlo sin apoyos bibliográficos. Esto no quiere decir, sin embargo, que no se hayan implementado estrategias similares en otros problemas diferentes al aquí expuesto.

por lo que debemos tratar de realizar el mínimo número de suposiciones posibles.

No obstante, la siguiente consideración se nos hace necesaria: el punto (.) indica el final de una frase solo si la siguiente palabra empieza con una letra *y* además mayúscula.

Por ejemplo: `"Your idea is interesting. However, I would [...]."` se separaría en dos frases, dado que la palabra posterior al punto empieza con una letra mayúscula. Sin embargo: `"We already mentioned in Section 1.1 that this example shows [...]."` conformaría una única frase, ya que tras el punto no aparece una letra. Procedemos de igual modo en el caso de los signos de interrogación (?) y de exclamación (!). Por ejemplo: `"She asked 'How's it going?', and I said 'Great!'."` se tomará correctamente como una sola frase; tras la interrogación, la siguiente palabra comienza con una letra *minúscula*.

Con la suposición anterior, también se agruparían correctamente los puntos suspensivos.

Sin embargo, fallaría en situaciones como: `"NLP (i.e. Natural Language Processing) is a subfield of Linguistics, Computer Science, and Artificial Intelligence."`, en la que la división sería: `"NLP (i.e."` por un lado, y `"Natural Language Processing) is a subfield [...]."`, por otro, ya que `"Natural"` empieza con mayúscula y aparece tras un punto.

Asimismo, la razón principal por la que no podemos apoyarnos únicamente en reglas predefinidas, reside en las llamadas Entidades Nombradas (*Named Entities*, en inglés), esto es, palabras que hacen referencia a personas, lugares, instituciones, empresas, etc. Existe toda una disciplina dedicada la identificación de este tipo de palabras, conocida como Reconocimiento de Entidades Nombradas (NER, por sus siglas en inglés), y pese a los buenos resultados conseguidos por algunos de los modelos propuestos, se considera un problema lejos de estar resuelto [22].

En nuestro caso emplearemos un modelo pre-entrenado para solucionar, al menos en parte, el problema de las Entidades Nombradas. Este modelo también solventa situaciones como la descrita anteriormente, en las que las reglas escritas a mano se quedan cortas. En el capítulo de Técnicas y Herramientas, hablaremos de dicho modelo y de la implementación concreta en código de los procedimientos expuestos anteriormente.

3.2. Codificación del texto

En esta etapa, se lleva a cabo lo que se conoce en inglés como *word embedding*[4]. Los modelos de IA trabajan, por lo general, con representaciones númericas. Por ello, las técnicas de *word embedding* se centran en vincular texto (bien sea palabras, frases, etc.), con vectores de números reales [23]. Esto hace posible aplicar a la generación de texto arquitecturas comunes dentro de la IA (y especialmente, del *Deep Learning*), como por ejemplo las Redes Neuronales Convolucionales (CNN) [24].

Esta idea, conceptualmente sencilla, encierra una gran complejidad, dado que los vectores generados deben retener la máxima información posible del texto original, incluyendo aspectos semánticos y gramaticales. Por poner un ejemplo, los vectores correspondientes a las palabras «profesor» y «alumno», deben preservar cierta relación entre ambos, y a su vez con la palabra «educación» o «escuela». Además, su vínculo con las palabras «enseñar» o «aprender» será ligeramente distinto, dado que en este caso se trata de una categoría gramatical diferente (verbos, en vez de sustantivos). A través de este ejemplo, podemos comprender que se trata de un proceso complejo.

Dado que los modelos pre-entrenados se encargan de realizar esta codificación por nosotros, no entraremos en más detalle en los algoritmos concretos empleados, dado que consideramos que se sale del alcance de este trabajo[5].

Lo que sí que hemos tenido que implementar en esta etapa, ha sido la división del texto en fragmentos a fin de no superar el tamaño máximo de entrada del modelo.

De este modo, podremos realizar resúmenes de textos arbitrariamente largos, siguiendo los siguientes pasos:

1. Dividimos el texto en fragmentos.
2. Generamos un resumen de cada fragmento.
3. Concatenamos los resúmenes generados.

Anteriormente, habíamos mencionado el término *token*. Este concepto se puede traducir al español como «símbolo». En nuestro caso concreto, un *token* es el vector numérico asociado a una palabra al realizar la codificación.

[4] En el presente documento, hemos traducido este término como «codificación del texto».

[5] En cualquier caso, el lector curioso puede explorar los algoritmos más populares de codificación, los cuales, ordenados cronológicamente, son: word2vec [25, 26], GloVe [27], y más recientemente, ELMo [28] y BERT [29].

Más concretamente, en modelos más actuales, como el modelo T5 [16], los *tókenes* pueden referirse a palabras completas o a *fragmentos* de las mismas.

Por lo general, las palabras que aparecen en el vocabulario con el que ha sido entrenado el modelo van a generar un único *token*. Sin embargo, las palabras desconocidas, se descompondrán en varios *tókenes*. Lo mismo sucede con palabras compuestas o formadas a partir de prefijación o sufijación. En la siguiente figura, podemos ver un ejemplo de ello:

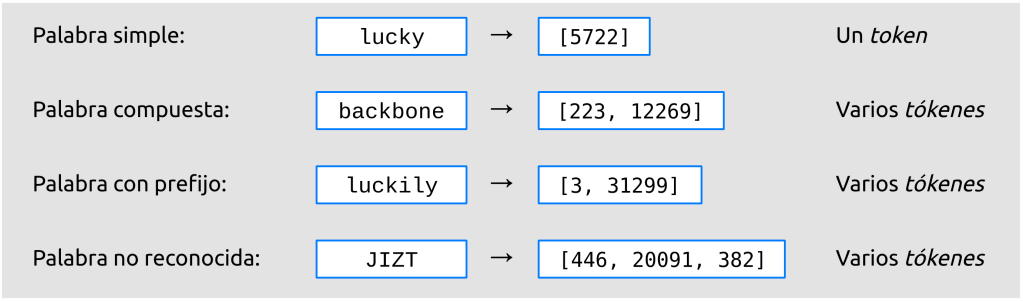

Figura 3.1: Ejemplo de *tokenización* con el modelo T5.

En el anterior ejemplo, si decodificamos los *tókenes* correspondientes a la palabra `"backcone"`, esto es, `[223, 12269]`, obtenemos los fragmentos `"back"`, y `"bone"`, respectivamente.

La idea detrás de esta fragmentación se basa en la composición, uno de los mecanismos morfológicos de formación de palabras más frecuentes [30] en muchos idiomas, como el inglés, español o alemán. Por tanto, presupone que dividiendo las palabras desconocidas en fragmentos menores, podemos facilitar la comprensión de las mismas. Naturalmente, habrá casos en los que esta idea falle; por ejemplo, en la figura anterior, la palabra `"JIZT"` se descompone en `"J"`, `"IZ"`, `"T"`, lo cual no parece hacerla mucho más comprensible.

Una vez explicado el concepto de *token*, volvamos al problema ya mencionado con anterioridad: los modelos de generación de texto admiten un tamaño de entrada máximo, determinado en función del número de *tókens*. Debido a que la unidad de medida es el número de *tókenes*, y no el número de palabras, o de caracteres, debemos tener en cuenta algunos detalles, entre ellos el hecho de que los modelos generan *tókenes* especiales para marcar el inicio y/o el final de la secuencia de entrada.

El modelo T5 (el cual como mencionábamos anteriormente, es el único modelo que utilizamos por ahora), genera un único *token* de finalización de

secuencia (EOS, *end-of-sequence*), que se coloca siempre al final del texto de entrada, una vez codificado, y en el caso de de este modelo siempre tiene el *id* 1. En la siguiente figura podemos ver un ejemplo con un texto de entrada:

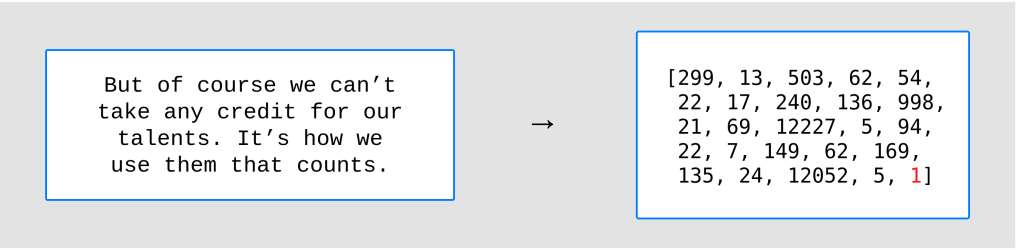

Figura 3.2: Pasaje del libro *A Wrinkle in Time*. El *tóken* EOS se ha marcado en rojo.

Como podemos ver, el *tóken* EOS aparece una única vez por cada texto de entrada, y es independiente de las frases que este contiene.

Otro aspecto a tener en cuenta, reside en que este modelo no solo es capaz de generar resúmenes, si no que puede ser empleado para otras tareas como la traducción, respuesta de preguntas [16], etc. Para indicarle cuál de estas es la tarea que queremos que desempeñe, curiosamente se lo tenemos que indicar tal y cómo lo haríamos en la vida real; en nuestro caso, simplemente precedemos el texto a resumir con la orden «resume» («*summarize*»). Por poner otro ejemplo, si quisiéramos resumir de alemán a español, le señalaríamos: «traduce de alemán a español» seguido de nuestro texto («*summarize German to Spanish*»).

Por consiguiente, este prefijo deberá aparecer al principio de cada una de las subdivisiones generadas y, del mismo modo, lo tenemos que tener en cuenta a la hora de calcular el número de *tókenes* de las mismas.

Con las anteriores consideraciones en mente, el objetivo principal será llevar a cabo la división del texto de entrada de forma que el número de *tókenes* varíe lo mínimo posible entre las diferentes subdivisiones, y todo ello sin partir ninguna frase.

Esta es una tarea más compleja de lo que puede parecer. En el capítulo de Técnicas y Herramientas se propone un algoritmo que emplea una estrategia voraz para llevar a cabo una primera división del texto; posteriormente procede al *balanceo* de las subdivisiones generadas en el paso anterior, de forma que el número de *tókenes* en cada subdivisión sea lo más parecido posible. Y esto, evidentemente, sin superar el máximo tamaño de entrada del modelo en ninguna de las subdivisiones.

Técnicas y herramientas

Esta parte de la memoria tiene como objetivo presentar las técnicas metodológicas y las herramientas de desarrollo que se han utilizado para llevar a cabo el proyecto. Si se han estudiado diferentes alternativas de metodologías, herramientas, bibliotecas se puede hacer un resumen de los aspectos más destacados de cada alternativa, incluyendo comparativas entre las distintas opciones y una justificación de las elecciones realizadas. No se pretende que este apartado se convierta en un capítulo de un libro dedicado a cada una de las alternativas, sino comentar los aspectos más destacados de cada opción, con un repaso somero a los fundamentos esenciales y referencias bibliográficas para que el lector pueda ampliar su conocimiento sobre el tema.

Aspectos relevantes del desarrollo del proyecto

Mencionar Cloud Native.

Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.

Bibliografía

[1] "Daniel Crevier. AI: The tumultuous history of the search for artificial intelligence. NY: Basic Books, 1993. 432 pp. (Reviewed by Charles Fair)". En: *Journal of the History of the Behavioral Sciences* 31.3 (1995), págs. 273-278. DOI: https://doi.org/10.1002/1520-6696(199507)31:3<273::AID-JHBS2300310314>3.0.CO;2-1. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/1520-6696%28199507%2931%3A3%3C273%3A%3AAID-JHBS2300310314%3E3.0.CO%3B2-1. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/1520-6696%28199507%2931%3A3%3C273%3A%3AAID-JHBS2300310314%3E3.0.CO%3B2-1.

[2] A. M. Turing. "Computing Machinery and Intelligence". En: *Mind* LIX.236 (oct. de 1950), págs. 433-460. ISSN: 0026-4423. DOI: 10.1093/mind/LIX.236.433. eprint: https://academic.oup.com/mind/article-pdf/LIX/236/433/30123314/lix-236-433.pdf. URL: https://doi.org/10.1093/mind/LIX.236.433.

[3] Pamela McCorduck. *Machines Who Think*. USA: W. H. Freeman y Co., 1979. ISBN: 0716710722.

[4] Joachim Rahmfeld. *Recent Advances in Natural Language Processing*. Sep. de 2019. URL: https://venturebeat.com/2021/01/06/ai-models-from-microsoft-and-google-already-surpass-human-performance-on-the-superglue-language-benchmark/. Último acceso: 26/01/2020.

[5] Thang Luong, Hieu Pham y Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". En: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational

Linguistics, sep. de 2015, págs. 1412-1421. DOI: 10.18653/v1/D15-1166. URL: https://www.aclweb.org/anthology/D15-1166.

[6] Dzmitry Bahdanau, Kyunghyun Cho y Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL].

[7] Ashish Vaswani y col. "Attention Is All You Need". En: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[8] Thomas Macaulay. *Someone let a GPT-3 bot loose on Reddit — it didn't end well*. Oct. de 2020. URL: https://thenextweb.com/neural/2020/10/07/someone-let-a-gpt-3-bot-loose-on-reddit-it-didnt-end-well. Último acceso: 26/01/2020.

[9] Kyle Wiggers. *AI models from Microsoft and Google already surpass human performance on the SuperGLUE language benchmark*. Ene. de 2021. URL: https://venturebeat.com/2021/01/06/ai-models-from-microsoft-and-google-already-surpass-human-performance-on-the-superglue-language-benchmark/. Último acceso: 26/01/2020.

[10] Google. *Cloud Natural Language*. URL: https://cloud.google.com/natural-language. Último acceso: 26/01/2020.

[11] IBM. *Watson*. URL: https://www.ibm.com/watson/about. Último acceso: 26/01/2020.

[12] Amazon. *Comprehend*. URL: https://aws.amazon.com/es/comprehend. Último acceso: 26/01/2020.

[13] Microsoft. *Text Analytics*. URL: https://azure.microsoft.com/es-es/services/cognitive-services/text-analytics. Último acceso: 26/01/2020.

[14] Raconteur. *A Day in Data*. 2019. URL: https://www.raconteur.net/infographics/a-day-in-data/. Último acceso: 26/01/2020.

[15] Abigail See, Peter J. Liu y Christopher D. Manning. "Get To The Point: Summarization with Pointer-Generator Networks". En: *CoRR* abs/1704.04368 (2017), pág. 1. arXiv: 1704.04368. URL: http://arxiv.org/abs/1704.04368.

[16] Colin Raffel y col. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". En: *CoRR* abs/1910.10683 (2019), pág. 11. arXiv: 1910.10683. URL: http://arxiv.org/abs/1910.10683.

[17] Microsoft. *Defining Cloud Native*. Mayo de 2020. URL: https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/definition. Último acceso: 27/01/2020.

[18] John Arundel y Justin Domingus. *Cloud Native DevOps with Kubernetes*. O'Reilly Media, Inc., mar. de 2019. ISBN: 9781492040767.

[19] Adam Bellemare. *Building Event-Driven Microservices: Leveraging Organizational Data at Scale*. O'Reilly Media, Inc., 2020. ISBN: 9781492057895.

[20] Crunchy Data. *Crunchy PostgreSQL Operator*. Mayo de 2021. URL: https://access.crunchydata.com/documentation/postgres-operator/latest. Último acceso: 27/01/2020.

[21] Mike Lewis y col. "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension". En: *CoRR* abs/1910.13461 (2019). arXiv: 1910.13461. URL: http://arxiv.org/abs/1910.13461.

[22] Wikipedia. *Reconocimiento de entidades nombradas - Wikipedia, La enciclopedia libre*. 2020. URL: https://es.wikipedia.org/wiki/Reconocimiento_de_entidades_nombradas. Último acceso: 27/01/2020.

[23] Christopher Manning - Stanford University. *Stanford CS224N: NLP with Deep Learning. Winter 2019. Lecture 13. Contextual Word Embeddings*. 2019. URL: https://www.youtube.com/watch?v=S-CspeZ8FHc. Último acceso: 28/01/2020.

[24] Linlin Hou y col. *Method and Dataset Entity Mining in Scientific Literature: A CNN + Bi-LSTM Model with Self-attention*. 2020. arXiv: 2010.13583 [cs.AI].

[25] Tomas Mikolov y col. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[26] Tomás Mikolov y col. "Distributed Representations of Words and Phrases and their Compositionality". En: *CoRR* abs/1310.4546 (2013). arXiv: 1310.4546. URL: http://arxiv.org/abs/1310.4546.

[27] Jeffrey Pennington, Richard Socher y Christopher Manning. "GloVe: Global Vectors for Word Representation". En: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, abr. de 2014, págs. 1532-1543. DOI: 10.3115/v1/D14-1162. URL: https://www.aclweb.org/anthology/D14-1162.

[28] Matthew E. Peters y col. "Deep contextualized word representations". En: *CoRR* abs/1802.05365 (2018). arXiv: 1802.05365. URL: http://arxiv.org/abs/1802.05365.

[29] Jacob Devlin y col. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. En: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[30] Bożena Cetnarowska. “Ingo Plag, Word-formation in English (Cambridge Textbooks in Linguistics). Cambridge: Cambridge University Press, 2003. Pp. xiv 240.” En: *Journal of Linguistics* 41.1 (2005). DOI: 10.1017/S0022226704303233.