Escuela Politécnica Superior de Jaén
Grado en Ingeniería Informática

Universidad de Jaén
Escuela Politécnica Superior
de Jaén

TRABAJO FIN DE GRADO

GENERADOR DE POESÍA CON DEEP LEARNING

Alumno

Sergiu Stoia

Tutores

Arturo Montejo Ráez
(Departamento de Informática)

Flor Miriam Plaza Del Arco
(Departamento de Informática)

Noviembre, 2022

(Página intencionalmente en blanco)

Universidad de Jaén

Departamento de Informática

Don Arturo Montejo Ráez y Doña Flor Miriam Plaza Del Arco, tutores del Trabajo Fin de Grado titulado: '**Generador de poesía con Deep Learning**', que presenta Don Sergiu Stoia, otorgan el visto bueno para su entrega y defensa en la Escuela Politécnica Superior de Jaén.

Jaén, Noviembre de 2022

El alumno:

STOIA SERGIU - X8836146Y

Firmado digitalmente por STOIA SERGIU - X8836146Y
Fecha: 2022.11.15 10:19:19 +01'00'

Sergiu Stoia

Los tutores:

Firmado digitalmente por MONTEJO RAEZ ARTURO - ***2653** el día 15/11/2022

PLAZA DEL ARCO FLOR MIRIAN - 15519730C

Digitally signed by PLAZA DEL ARCO FLOR MIRIAN - 15519730C
Date: 2022.11.15 13:04:19 +01'00'

Arturo Montejo Ráez

Flor Miriam Plaza Del Arco

(Página intencionalmente en blanco)

Agradecimientos

En primer lugar, me gustaría agradecer a mi familia y amigos el apoyo que me han brindado durante todos estos años.

También quisiera agradecer a mis tutores el proporcionarme la idea y la guía para la resolución de este trabajo.

Por último, y no menos importante, agradecer a todos los profesores que me han dado clase durante todos estos años de carrera.

FICHA DEL TRABAJO FIN DE TÍTULO	
Titulación	Grado en Ingeniería Informática
Modalidad	Trabajo Teórico/Experimental
Especialidad	Tecnologías de la Información
Mención	Sin mención
Idioma	Español
Tipo	Específico
TFT en equipo	No
Autor/a	Sergiu Stoia
Fecha de asignación	25/10/2021
Descripción corta	Con el paso de los años, la IA ha avanzado gracias al aprendizaje automático y el *deep learning,* a partir del cual surgen modelos del lenguaje mediante los cuales generar texto. La finalidad de este trabajo es la de implementar un sistema generador de poesía que haga uso de un modelo del lenguaje *pre-entrenado* al español y que se ajuste a un *corpus* de poesía hispánica.

NORMAS APLICADAS EN ESTE DOCUMENTO	
LOCALES	
TFT-UJA:2017	Normativa de Trabajos Fin de Grado, Fin de Máster y otros Trabajos Fin de Título de la Universidad de Jaén (Normativa marco UJA aprobada en Consejo de Gobierno)
TFT-EPSJ:2017	Normativa sobre Trabajos Fin de Grado y Fin de Máster en la Escuela Politécnica Superior de Jaén (Normativa EPSJ aprobada en Junta de Escuela)
TFT-EPSJ	Criterios de evaluación y normas de estilo para TFG y TFM de la Escuela Politécnica Superior de Jaén
NACIONALES E INTERNACIONALES	
ISO 2145:1978	Documentación - Numeración de divisiones y subdivisiones en documentos escritos
UNE 50132:1994	Traducción de la ISO 2145
APA 6ª edición	Estilo de referencias y citas de APA (American Psychological Association)

NORMAS UTILIZADAS COMO BASE O REFERENCIA	
NACIONALES	
UNE 157001:2014	Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico
UNE 157801:2007	Criterios generales para la elaboración de proyectos de sistemas de información
*Estas normas se han utilizado **como base o referencia** para la inclusión de algunos contenidos y definiciones sobre elaboración de proyectos, entendiendo como **proyecto** la documentación consensuada entre una empresa y un cliente, que da lugar al perfeccionamiento de un contrato para la elaboración de una obra o la prestación de un servicio. Por consiguiente, no debe esperarse la aplicación de estas normas en cuanto a la completitud de los contenidos ni a la organización de los mismos.*	

Contenido

Índice de ilustraciones

1 ESPECIFICACIÓN DEL TRABAJO

En este capítulo se presenta la especificación del trabajo, con una estructura y contenidos **inspirados** en los criterios y recomendaciones que establece la norma UNE 157801:2007 - "*Criterios Generales para la elaboración de proyectos de Sistemas de Información*".

A lo largo del documento se utilizarán términos y acrónimos cuya descripción aparece en el apartado 7 (Definiciones y abreviaturas).

1.1 Introducción

Con el paso de los años, la Inteligencia Artificial (IA) ha ido tomando presencia en muchos ámbitos, todo esto impulsado gracias al uso de técnicas de aprendizaje automático (también conocido como *Machine Learning o ML)* y, más recientemente, el *Deep Learning* (DL), que hace uso de redes neuronales profundas.

Una de las áreas que más relevancia ha tomado dentro de la IA es la del Procesamiento del Lenguaje Natural (PLN), un área interdisciplinaria entre la Informática y la Lingüística cuyo objetivo principal es el desarrollo de sistemas de software pensados para procesar el lenguaje humano. Gracias a esta área se consiguen sistemas que analizan, comprenden y generan lenguaje natural.

Algunos ejemplos de aplicaciones son los traductores, los sistemas de recuperación de información, sistemas de extracción del conocimiento y, concretamente, la generación automática de textos[1]. En este tipo de generadores se utilizan modelos de lenguaje, los cuales han pasado de ser estadísticos a ser modelos de *Deep Learning* gracias al avance en cuanto a las capacidades de cómputo y almacenamiento.

Este trabajo tiene como fin el desarrollo de un sistema de generación de poesía, para ello se hace uso del modelo de lenguaje *GPT2-large-bne* (Asier Gutiérrez-Fandiño, 2021) , basado en *GPT2* (OpenAI, 2019) y el cual será ajustado con un

[1] https://plantl.mineco.gob.es/tecnologias-lenguaje/Paginas/tecnologias-lenguaje.aspx

corpus de poemas en español. Este generador respetará reglas inherentes de este género literario tales como rima, métrica y estructura.

Este documento contiene un total de 9 capítulos entre los que se incluyen este mismo, donde se presenta la especificación del trabajo, junto a los objetivos del mismo y todo lo ocurrido hasta llegar hasta el punto de partida. A continuación está el resto de capítulos que conforman la memoria:

- Capítulo 2 Diseño inicial: Correspondiente a la especificación de aspectos relacionados con la arquitectura del sistema propuesto, diseño y metodología empleada.
- Capítulo 3 Desarrollo: Se explica todo lo relativo a la implementación del sistema.
- Capítulo 4 Experimentación, resultados y discusión: Se muestran distintos resultados obtenidos tras la ejecución del sistema y un análisis sobre los mismos.
- Capítulo 5 Conclusiones y trabajos futuros: Se presentan las conclusiones obtenidas tras terminar el proyecto además de propuestas para mejorar y líneas de trabajo futuras.
- Capítulo 6 Apéndices: Se incluye la documentación adicional que pueda resultar de utilidad para comprender correctamente el proyecto.
- Capítulo 7 Definiciones y abreviaturas: Definición de toda la terminología empleada a lo largo de la memoria que pueda resultar muy específica y requiera aclaraciones.
- Capítulo 8 Bibliografía: Referencias a textos, artículos y otros documentos utilizados como fuente de conocimiento a lo largo del desarrollo del proyecto.

1.2 Objetivos del trabajo

Como hemos comentado anteriormente, el resultado a conseguir es un sistema generador de poesía en español utilizando técnicas de PLN. Este partirá de una serie de entradas realizadas por el usuario: la estructura deseada del poema y el texto a

partir del cual se empezarán a generar los versos correspondientes. Para cumplir con este objetivo general, ha sido necesario abordar los siguientes objetivos específicos:

1. Ajustar el modelo de lenguaje de *GPT2-large-bne* con el *corpus* de poesía hispánica (proceso también conocido como *fine-tuning*).
2. Implementar un programa que sea capaz de calcular la métrica de un verso, teniendo en cuenta las diferentes excepciones y recursos literarios que existen.
3. Crear un programa que dadas dos palabras encuentre la rima entre ambas, ya sea consonante o asonante.
4. Diseñar un algoritmo de generación del lenguaje con restricciones (*constrained language generation)* que sea capaz de explorar secuencias posibles dentro de las restricciones impuestas. En nuestro caso, estas restricciones son la métrica y la rima
5. Finalmente se busca juntar los objetivos anteriores con el fin de implementar el sistema de generación de poesía, este consistirá en la generación de posibles versos gracias a un modelo del lenguaje para el español basado en la arquitectura GPT-2 y la posterior validación de los mismos mediante el medidor de métrica y rima.

1.3 Antecedentes y estado del arte

Podemos definir la IA como una disciplina dentro de las ciencias de la computación cuyo objetivo es entender los principios que hacen posible un comportamiento inteligente, aunque esta área de la computación no tenga una definición, podemos destacar las siguientes:

1. Sistemas que piensan como humanos: *"El nuevo y excitante esfuerzo de hacer que los computadores piensen … máquinas con mente, en el más amplio sentido literal"* (Haugeland, 1985)
2. Sistemas que piensan racionalmente: *"El estudio de las facultades mentales mediante el uso de modelos computacionales"* (McDermott, 1985)

3. Sistemas que actúan como humanos: *"El arte de desarrollar máquinas con capacidad para realizar funciones que cuando son realizadas por personas requieren inteligencia"* (Kurzweil, 1990)
4. Sistemas que actúan de forma racional: *"La inteligencia computacional es el estudio del diseño de agentes inteligentes"* (Poole, 1998)

Dentro de la IA existe un paradigma llamado *Machine Learning* (ML), una rama cuyo objetivo es la capacidad de que una máquina *aprenda* a realizar una tarea a partir de un conjunto de datos, lo que en otros casos se realizaría con instrucciones explícitas indicadas por un experto (BBVA, 2019).

En el ML se ingresan tanto las cuestiones a resolver como sus respuestas, de modo que el algoritmo de aprendizaje encuentre patrones y devuelva un modelo como resultado. Esta rama tiene muchos usos en varios ámbitos, por ejemplo, en la visión por ordenador y los motores de recomendación (BBVA, 2019).

Una de las mayores confusiones que ocurren es la de utilizar indistintamente los conceptos de *Machine Learning* y *Deep Learning*, cuando en realidad el DL es un subcampo del ML (Kavlakoglu, 2020).

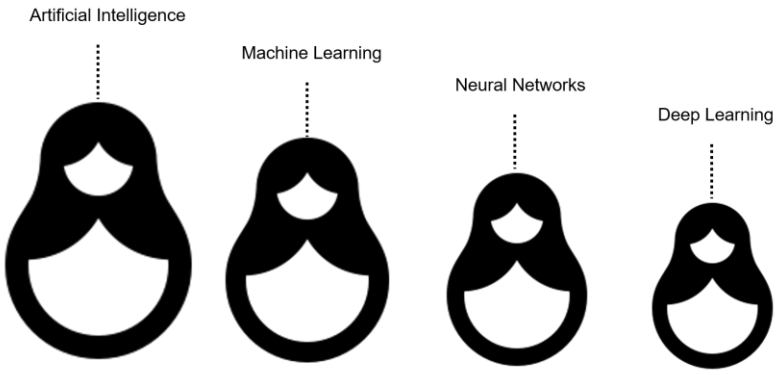

Ilustración 1 Representación de los subcampos de la IA

La principal diferencia entre *Machine Learning* y *Deep Learning* se basa en las diferencias que tienen ambos algoritmos a la hora de *aprender* y la cantidad de datos que maneja cada uno. DL automatiza en gran parte la extracción de características, lo que evita la intervención humana, mientras que ML depende más de expertos humanos ya que suelen ser estos los que determinen el conjunto de características a extraer.

Deep Learning funciona gracias a las redes neuronales, que conforman la estructura principal de este paradigma. El *"Deep"* en DL se refiere a la profundidad de capas de la red neuronal. Estas redes neuronales imitan el modo de aprendizaje del ser humano a través de distintos algoritmos (Kavlakoglu, 2020).

En la *Ilustración 2* podemos observar la estructura de una red profunda, donde podemos ver las capas de entrada (*input*) y salida (*output*) además de múltiples capas intermediarias

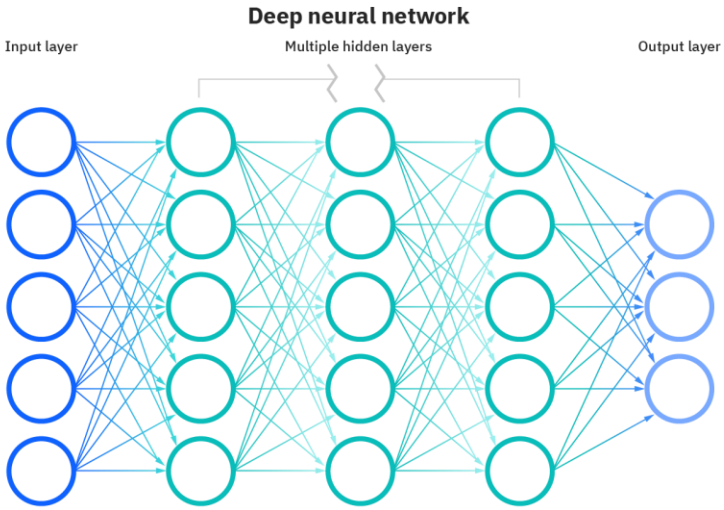

Ilustración 2: Ejemplo de red neuronal profunda con diferentes capas

Gracias a los avances en los algoritmos de entrenamiento de estas redes y la velocidad de cómputo de los procesadores actuales, el área del PLN ha progresado considerablemente en poco tiempo. Anteriormente, uno de los mayores inconvenientes que existían era la incapacidad de tener en cuenta el orden de las palabras a la hora de analizar un texto, esto era a causa del uso de variables tipo *Bag of Words* (BoW), donde los textos se representaban por vectores que contabilizaban el número de veces que aparecía una palabra[2].

Esta representación fue sustituida por los *Word embeddings*, donde las palabras de un texto son representadas mediante un vector. En esta metodología, los vectores de las palabras con un contexto similar estarán más cercanos en el espacio vectorial (por ejemplo "perro" y "gato").

[2] https://kryptonsolid.com/que-es-el-modelo-de-bolsa-de-palabras-modelo-bow/

Gracias a esto, era posible mantener el significado y orden dentro de una frase en las redes neuronales recurrentes (RNN) (Santander, 2021). En estas redes se va generando una representación del documento completo a base de ir mezclando *embeddings*, como podemos ver en la *Ilustración 3* donde "*h*" es el texto de entrada, *"x"* la representación de la palabra con *Word embedding* e *"y"* el resultado final (Vaca, s.f.).

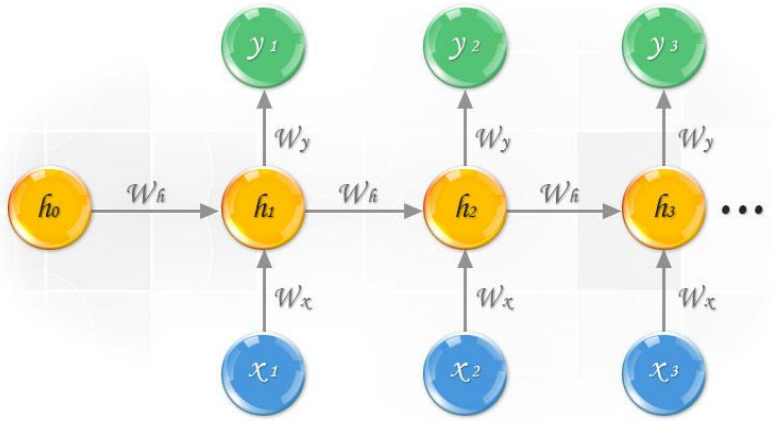

Ilustración 3 Funcionamiento de una RNN con Word embedding

En 2017, la arquitectura *Transformers* fue presentada por *Ashish Vaswani*, un modelo que sustituyó las capas recurrentes de las redes neuronales por las denominadas capas de atención, gracias a estas capas se consiguen analizar secuencias mucho más largas y de forma paralela, a diferencia de las RNN, las cuales tienen una memoria más a corto plazo y analizan las secuencias palabra a palabra (Ashish Vaswani, 2017).

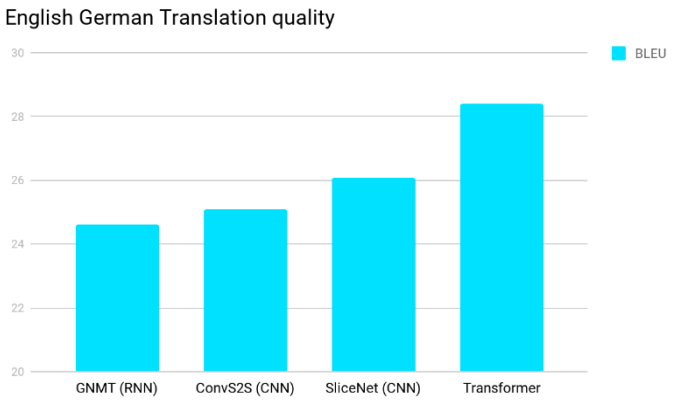

Ilustración 4 Comparación entre distintos modelos para una tarea de PLN (Mayor es mejor)

La arquitectura Transformer se basa en la estructura codificador-decodificador (*encode-decode).* En este tipo de redes el codificador extrae las características de la entrada para que posteriormente el decodificador las utilice para generar una salida. En la *Ilustración 5* podemos ver una representación en forma de diagrama de bloques que explica el funcionamiento de este tipo de redes, la parte izquierda corresponde con el *encoder* y la derecha con el *decoder* (Sotaquirá, 2020).

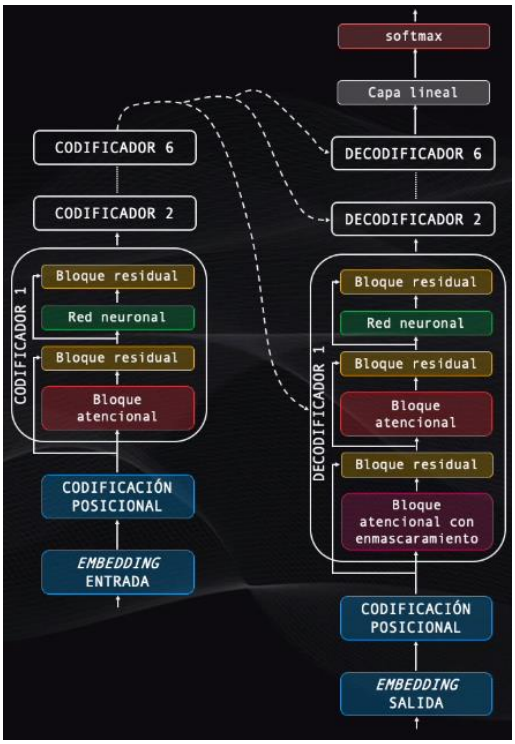

Ilustración 5 Diagrama de bloques general de una Red Transformer

Existen varios módulos dentro de la red, siendo el primero de estos el *embedding* de entrada y salida, donde el texto de entrada será convertido a una serie de vectores los cuales son llamados *tokens*.

Ya que el análisis de la secuencia de los textos no se realiza de forma serial, es necesario indicar el orden en el que se encuentran las palabras dentro del texto, esto se consigue con la codificación posicional, que genera una serie de vectores que se sumarán a los *tokens,* utilizando funciones senoidales para las posiciones pares y cosenoidales para las impares, resultando en vectores con un patrón numérico único para cada posición.

Tras conseguir *tokenizar* todos los elementos de la entrada junto a su respectivo orden, el siguiente paso a realizar es el análisis de la totalidad de la secuencia, encontrando relaciones entre varias palabras de la misma. La manera de conseguir esto es mediante bloque de atención dentro de los codificadores, en este paso se expresan estas relaciones de forma numérica, teniendo así el contexto de cada *token* representado de forma matemática. Normalmente existen varios bloques de atención para que, en consecuencia, las asociaciones entre palabras se estudien a distintos niveles.

En el caso del decodificador, existe una excepción en los bloques de atención, ya que las palabras de la salida se generan secuencialmente, solo se tendrán en cuenta las palabras anteriores y la generada (Sotaquirá, 2020).

La arquitectura *Transformers* ha sentado las bases de muchos de los modelos de lenguaje por todas las ventajas que ofrece. A la hora de utilizar estos modelos existen dos fases (Vaca, s.f.):

1. *Pre-training:* El modelo aprende la estructura del lenguaje de forma general, además de adquirir un conocimiento genérico del significado de las palabras
2. *Fine-tuning:* Esta fase se realiza con modelos ya pre-entrenados, se añaden ciertas capas a la arquitectura para adaptar los modelos a ciertas tareas.

Un ejemplo de modelo perteneciente a esta familia es *Generative Pre-Training Transformer* (*GPT),* publicado en el año 2018 por *OpenAI.* Anteriormente a este modelo, los datos de entrenamiento debían ir acompañados de una gran cantidad de anotaciones, lo que provoca una mayor intervención humana y por lo tanto un mayor coste, además de ser imposible clasificar una gran cantidad de datos en la mayoría de los casos. Este modelo tenía como propuesta la utilización de datos sin etiquetar para posteriormente ajustar el modelo mediante *fine-tuning*. (Alec Radford, 2018)

Para el entrenamiento de este modelo se utilizó el *dataset BooksCorpus* que contenía 7000 libros sin publicar, haciendo uso de 117 millones de parámetros para la red neuronal. Gracias a todo lo que aprendió el modelo en el pre-entrenamiento, no era necesario un ajustado demasiado costoso, haciendo que este modelo superara a los ya existentes en esa época.

No obstante, aunque *GPT* consiguiera superar a todos los modelos existentes, *GPT-2* llegó al año siguiente y lo superó con creces. La principal tarea de este modelo era la de adivinar la siguiente palabra dadas las anteriores (OpenAI, 2019). Fue entrenado con un *dataset* de textos recogidos de *Reddit* el cual ocupaba 40GB, con un número total de 1.5 billones de parámetros.

Una de las grandes cualidades de *GPT-2* son los buenos resultados que se obtienen en ciertos campos a pesar de no haber realizado *fine-tuning* con un *dataset* más específico, lo que se denomina *Zero Shot Learning*.

Debido a las grandes capacidades de este modelo, *OpenAI* decidió publicar una versión menos potente del mismo, ya que podía ser usado para fines maliciosos, la primera versión contenía 117 millones de parámetros, a diferencia de los 1500 millones que tiene la versión final. (OpenAI, 2019)

A pesar de todo, a lo largo de los años este campo ha seguido evolucionando, y por lo tanto, *OpenAI* no se quedó atrás con *GPT-2.* En el año 2020 surgió *GPT-3*, un modelo incluso más ambicioso que su predecesor, y con resultados aún mejores. El objetivo a conseguir por parte de *OpenAI* era el de hacer el procesamiento del lenguaje más potente y rápido sin necesitar un ajustado, el cuál en la mayoría de casos no es necesario con este modelo del lenguaje. Para conseguir un modelo más potente que *GPT-2* se utilizó un *dataset* de 500 billones de palabras llamado *Common Crawl*, además de 175 billones de parámetros, lo que resulta ser 100 veces más grande que el anterior modelo (Tom B. Brown, 2020).

Es la arquitectura GPT-2 y un modelo pre-entrenado para el español realizado por la Biblioteca Nacional el que usaremos en nuestros experimentos.

1.4 Requisitos iniciales

Tal y como se explica en el *apartado 1.2*, se desea implementar un sistema generador de poesía con dos tipos de entrada, la estructura del texto y el texto inicial a partir del cual se generará el resto de la secuencia. Teniendo esto en cuenta, se definen los siguientes requisitos iniciales:

1. Se debe realizar *fine-tuning* al modelo *GPT2-large-bne* (Asier Gutiérrez-Fandiño, 2021) de manera exitosa, devolviendo como resultado otro modelo entrenado con el corpus de poesía hispánica.
2. El sistema debe recoger datos sobre la estructura y texto de entrada por parte del usuario
3. El generador debe ser capaz de respetar la estructura que ha definido el usuario, lo que implica una medida de la métrica de cada verso generado y la detección de la rima entre versos.
4. El sistema debería generar un poema en un tiempo aceptable
5. La implementación debería ofrecer al usuario alguna forma de modificar los parámetros de generación, siendo algunos de estos los que están relacionados con el *decoder* del generador.

1.5 Alcance

Con la elaboración de este trabajo se pretenden obtener los siguientes resultados:

1. *Notebook* de *Google Colab* donde se puede encontrar el código para el ajuste del modelo a la generación de textos poéticos.
2. *Notebook* que contiene el código del sistema generador, donde encontraremos los métodos utilizados para la medida de la métrica y la rima, además del uso del modelo antes ajustado.
3. Archivo de texto que contendrá algunos resultados obtenidos donde se indicará la entrada utilizada y los parámetros de generación para cada poema.
4. Archivo de texto *README* que explicará el contenido de cada fichero de la entrega, además de incluir los repositorios donde se han almacenado tanto el código del sistema de generación como el modelo de lenguaje generado.
5. Memoria del trabajo

1.6 Hipótesis y restricciones

El TFG se define como una asignatura de 12 créditos, lo que supone que la duración total del proyecto será de 300 horas, incluyendo todas las etapas del ciclo de vida, con la excepción del mantenimiento. Por consiguiente, la principal restricción aplicable es la limitación de la duración del trabajo.

En cuanto a la calidad del resultado es importante resaltar lo altamente restrictiva que puede llegar a ser la poesía, teniendo una cantidad enorme de reglas a seguir, por lo que es comprensible que el sistema generador ocupe mucho tiempo para generar poemas, ya que se debe encontrar una salida del modelo que cumpla con todas las reglas establecidas, debe de seguir la rima, métrica y estructura acordados por el usuario.

1.7 Estudio de alternativas y viabilidad

A la hora de tomar la decisión sobre cómo realizar la implementación del sistema generador, se han tenido en cuenta dos alternativas:

1. Realizar *fine-tuning* del modelo para que este sea capaz de generar salidas que respeten la rima. Para este caso sería necesario un *corpus* más grande y más recursos.

2. Ajustar el modelo del lenguaje para generar salidas normales las cuales no respeten la rima, pero utilizar estas para implementar un sistema que a partir de las mismas pueda encontrar las salidas que cumplen las reglas necesarias y generar el poema verso a verso (generación del lenguaje con restricciones).

Finalmente se ha elegido la segunda salida, esto es debido a los recursos que se han conseguido para la implementación de este trabajo, estos recursos no son suficientes como para implementar la primera alternativa y que el resultado sea satisfactorio.

1.8 Descripción de la solución propuesta

Finalmente, como se ha explicado en el *apartado 1.8*, la solución final se trata de un sistema que generará el poema verso a verso donde, para cada verso del poema, se tendrá en cuenta las restricciones de métrica y rima.

En primer lugar es necesario ajustar el modelo para una tarea de PLN denominada *Casual Language Modeling*, para ello se estudiará el contenido del corpus y se eliminarán los elementos innecesarios para posteriormente realizar el *fine-tuning* de *GPT2-large-bne.*

Posteriormente se implementará el sistema, el cual recogerá las restricciones de métrica y rima a partir de una entrada realizada por el usuario. Estas entradas realizadas por el usuario se aplicarán a las salidas obtenidas por el modelo de lenguaje previamente ajustado con el corpus de poesía.

Mediante el uso de la estructura de datos llamada pila, el sistema será capaz de realizar un *"backtracking"*, es decir, volver a pasos anteriores si no se consiguen resultados favorables tras varios intentos del generador para encontrar versos adecuados para el poema, esto ahorra bastante tiempo de ejecución ya que evitamos que el sistema se quede bloqueado en la generación de un solo verso.

1.9 Material y métodos

A la hora de implementar este trabajo se ha empleado una tarea dentro del modelado del lenguaje llamada *Casual Language Modeling (CLM)*. En este tipo de metodología, los modelos deben predecir el siguiente token de una secuencia a partir de los tokens anteriores, es decir, el modelo calcula el *token i + 1* a partir de los *token i*, *i-1, i-2, …* (Gugger, 2021)

Uno de los modelos que están pensados para esta tarea es *GPT2,* un modelo basado en *Transformers* cuyo objetivo, como se ha mencionado en el *apartado 1.3*, es adivinar la siguiente palabra de una secuencia. Ya que este sistema generador debe crear poemas en español, se ha utilizado el modelo *GPT2-large-bne*, un modelo *pre-entrenado* con un *corpus* en español que consiste en *570 GB* de textos recogidos

por la *Biblioteca Nacional de España* gracias a *web crawlings* (Asier Gutiérrez-Fandiño, 2021).

A la hora de ajustar el modelo, se ha utilizado un *corpus* de poesía hispánica, una recopilación de varios poemas creada por el Grupo de Investigación SINAI[3] en el que se recogen los poemas de los autores más influyentes de 15 países de habla hispana, contando con un total de 109 autores, 137 ficheros de texto y una cantidad total de 774128 *tokens* tras realizar el preprocesamiento descrito en el *Apartado 3.1.1*.

1.10 Tecnologías utilizadas

En este apartado se enumerarán todas las tecnologías utilizadas en la implementación del proyecto:

1. Python: Se ha utilizado este lenguaje de programación ya que es el más utilizado para temas relacionados con el procesamiento del lenguaje natural

2. Google Collaboratory[4]: Se ha elegido esta herramienta tras intentar realizar el *fine-tuning* del modelo en un entorno local mediante *Visual Studio,* al intentar realizar esta tarea con la *GPU*, la memoria no era suficiente debido a la complejidad del modelo. Esta herramienta de *Google* ofrece los recursos de cómputo sin depender de los de nuestra máquina.

3. Se han utilizado las librerías de *transformers*[5] y *git-lfs*[6] para ajustar y utilizar el modelo. En cuanto a la implementación del sistema generador se han usado las librerías *syltippy*[7] la cual nos devolverá un vector que contiene las sílabas de una palabra y *spacy*[8] para analizar morfológicamente las palabras de los poemas.

[3] https://sinai.ujaen.es/sinai
[4] https://colab.research.google.com/?hl=es
[5] https://huggingface.co/docs/transformers/main/en/index
[6] https://git-lfs.github.com
[7] https://github.com/nur-ag/syltippy
[8] https://spacy.io

1.11 Metodología de desarrollo de software

En el desarrollo de software existen dos tipos de metodologías de desarrollo, las metodologías tradicionales y ágiles.

Las metodologías tradicionales dividen el proyecto en distintas fases, las cuales se completan y validan de forma secuencial, es decir, una etapa debe ser finalizada para pasar a la siguiente. Uno de los problemas de este tipo de metodologías es la poca flexibilidad que tienen.

La metodología ágil surgió como respuesta a los enfoques tradicionales, este tipo de metodologías son capaces de adaptar las formas de trabajo a las condiciones del proyecto, consiguiendo mayor flexibilidad (Santander becas, 2020).

Finalmente, tras analizar ambos tipos de metodologías, se ha tomado la decisión de seguir la metodología en cascada, una de las metodologías tradicionales más famosas, la cual divide el proyecto en distintas fases secuenciales (Stsepanets, 2021).

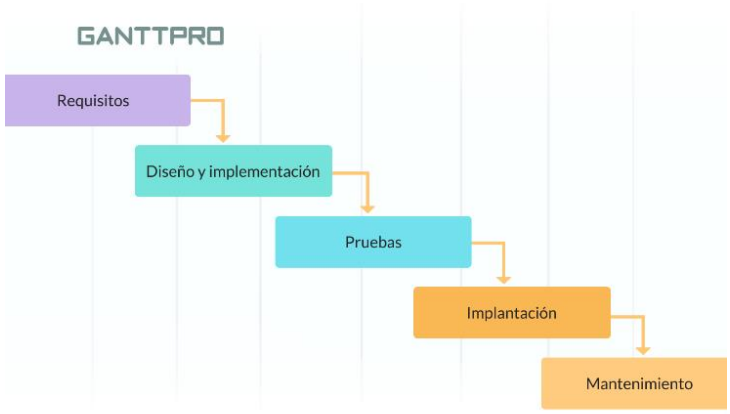

Ilustración 6 Modelo en cascada

Las distintas fases de este modelo en cascada son:

1. **Requisitos:** Es la fase más importante, donde se realizan reuniones e intercambio de opiniones para definir los requisitos que definirán el proceso de desarrollo y resultado final del proyecto.

2. **Diseño e implementación:** Se trabaja en el desarrollo del producto, trabajando en el diseño, implementación y código cumpliendo los requisitos anteriormente pactados.

3. **Pruebas:** Una etapa en la que los especialistas prueban el software y detectan los errores, asegurándose de cumplir los requisitos del cliente.

4. **Implantación:** Se instala el software, en esta fase es posible que existan algunos procesos de prueba.

5. **Mantenimiento:** El producto se entregará al cliente, y dependiendo del tipo de proyecto se pondrá en marcha el soporte para confirmar que el proyecto funciona correctamente.

En el caso de este proyecto, las fases de implantación y mantenimiento no son necesarias, ya que una vez conseguidos los resultados esperados no será necesario realizar una instalación del sistema ni ofrecer un servicio de soporte ni mantenimiento.

Se ha elegido esta metodología ya que solo contamos con un solo efectivo a la hora del desarrollo, por lo que resulta más sencillo e intuitivo trabajar con una metodología tradicional, además de contar con requisitos bastante claros, por lo que no existirán muchos cambios en el proceso de implementación.

1.12 Estimación del tamaño y esfuerzo

Ya que el presente proyecto es un TFG, no existen restricciones de tipo económico, sino de tipo temporal (un número aproximado de horas). Por consiguiente, los cálculos de tamaño del proyecto están supeditados al tiempo disponible. En cuanto al esfuerzo, se dispone de tan solo un efectivo (la persona autora del trabajo).

1.13 Planificación temporal

Teniendo en cuenta que el Trabajo de Fin de Grado en el grado de Ingeniería Informática equivale a 12 créditos y que un crédito equivale a 25 horas, podemos calcular que la realización de este proyecto es de máximo 300 horas, lo que equivale a 7,5 semanas laborales, es decir, semanas con 5 días laborales de 8 horas de duración para cada día.

Semana	Actividades
1	Revisión del estado del arte
2	Análisis de requisitos y diseño
3	Estudio del corpus de poesía
4	*Fine-tuning* del modelo
5	Implementación de scripts de medida de métrica y rima
6	Implementación del sistema de generación de poesía
7	Pruebas de solución final y resolución de errores
8	Finalización de memoria

Tabla 1 Planificación temporal

En la *Tabla 1* observamos que la implementación del proyecto se ha realizado en 8 semanas.

Las dos primeras semanas corresponden a la fase de análisis de requisitos donde se estudia el estado de arte, los requisitos funcionales y no funcionales, y también se realiza el diseño del sistema final, lo que corresponde a la siguiente fase del modelo en cascada. Las semanas 3, 4, 5 y 6 forman la fase de implementación y finalmente las dos últimas semanas corresponden con la fase de pruebas y la entrega del proyecto.

1.14 Presupuesto

A día de hoy, el sueldo medio de un ingeniero informático es de 2.208 euros al mes según *talent.com*[9], a la hora esto resultaría en 13,59 euros. Como hemos comentado en el apartado anterior, el proyecto supone una cantidad de 300 horas, por lo que el total del sueldo formaría un total de 4.077 euros ya que solo contamos con un efectivo.

Respecto al material informático, las herramientas de software utilizadas no tienen coste alguno ya que se ha utilizado la versión gratuita de *Google Collaboratory*. En cuanto al apartado hardware, se ha utilizado un ordenador portátil con un valor

9 https://es.talent.com/salary?job=ingeniero+inform%C3%A1tico#:~:text=%C2%BFCu%C3%A1nto%20gana%20un%20Ingeniero%20inform%C3%A1tico,%E2%82%AC%2013,59%20por%20hora.

estimado de 1200 euros y una vida útil de estimada en 5 años, por lo tanto, su cuota de amortización sería equivalente a 240 euros por año. Teniendo en cuenta los gastos del material informático y las 8 semanas de duración del proyecto, el coste equivale a 36’92 euros.

También se deben tener en cuenta gastos indirectos, siendo estos gastos derivados del uso de instalaciones o facturas de servicios, calculamos que ronda el 10% de sobrecoste del proyecto, es decir, un total de 411’39 euros.

Finalmente, tras haber definido los posibles gastos del proyecto, el total tras sumar todos estos gastos es de 4525’31 euros.

Concepto	€/hora	Coste
Sueldo ingeniero informático	13,59 €	4.077 €
Material informático		36’92 €
Gastos indirectos		411’39 €
TOTAL		4.525’31 €

Tabla 2 Costes del proyecto

2 DISEÑO INICIAL

2.1 Especificaciones del sistema

Se analizan y describen los requisitos tanto funcionales como no funcionales que definirán la implementación del sistema final. Es importante definir de forma clara estos requisitos ya que muchos de los problemas de los proyectos de software surgen a causa de no definir de forma correcta los requisitos que se quieren imponer a la implementación del proyecto.

2.1.1 Requisitos funcionales

Los requisitos funcionales se refieren a aquellas funciones que prestará el sistema. Define qué debe y que no debe hacer el sistema, definiendo la interacción con el usuario y con otros sistemas, respuestas automáticas, procesos predefinidos.

Como no existe un cliente para este proyecto, los requisitos serán definidos por el autor del trabajo:

1. El sistema debe aceptar entradas por parte del usuario, a partir de las cuales se definirán la estructura y texto inicial del poema.

2. A la hora de construir el poema se deben cumplir las reglas de rima y métrica establecidas por el usuario.

3. Se seguirán unas reglas de estilo para que la salida del poema sea más coherente.

2.1.2 Requisitos no funcionales

Los requisitos no funcionales se refieren a aquellas propiedades del sistema que no tienen que ver con "qué" hace el sistema sino "cómo" lo hace, estos requisitos son condiciones que impone el cliente en cuanto a la calidad del producto, envolviendo aspectos como el rendimiento, seguridad, disponibilidad.

1. Rendimiento: El sistema debe generar los poemas en el menor tiempo posible, siempre teniendo en cuenta lo restrictiva que es la poesía, por lo que es normal que el generador tarde un tiempo en encontrar un verso adecuado.

2. Escalabilidad: El programa debe ser capaz de generar poemas más grandes sin que esto afecte al rendimiento.

3. Usabilidad: El usuario debe ser capaz de utilizar el generador de poemas sin ningún tipo de complicación

2.2 Análisis y diseño del sistema

Tras haber analizado y estudiado los requisitos del sistema que queremos implementar, es necesario diseñarlo teniendo en cuenta el objetivo final y estos requisitos.

2.2.1 Clases implementadas

Se han diseñado dos clases las cuales serán utilizadas en la implementación del sistema de generación de poesía:

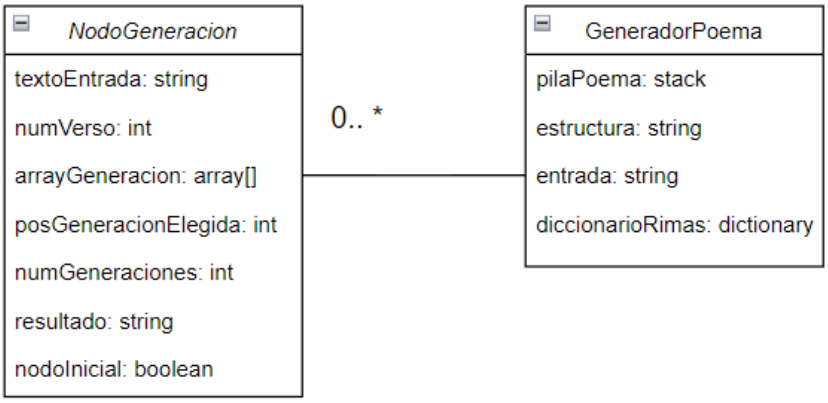

Ilustración 7 Diagrama de clases

1. **NodoGeneracion**: Es una clase que representa un paso dentro del proceso de generación del poema. Este nodo representa a un verso dentro de todo el poema, esta clase es la encargada de utilizar el modelo de lenguaje ajustado previamente para generar textos hasta encontrar uno que se adecúe a las reglas del poema.

2. **GeneradorPoema**: Se encarga de mantener una pila de objetos *NodoGeneracion* y de validar los textos obtenidos por cada nodo. Es la clase que se ocupa de recoger la estructura y el texto de entrada del cliente, y la que crea los objetos *NodoGeneracion* para avanzar en la creación del poema.

2.2.2 Flujo del sistema

En este apartado se analizará todo el proceso que sigue el sistema cuando se desea generar un poema. Para ello se ha creado un diagrama de flujo, representado en la *Ilustración 8.*

Tras recoger la estructura y el texto de entrada por parte del usuario, el sistema creará el poema verso a verso mediante el uso de la clase *NodoGeneracion*, gracias a la cual se utilizará el modelo para generar textos, los cuales serán validados mediante una comprobación de métrica y rima.

Si un nodo, tras generar varios textos, no da resultados favorables, se intentarán generar otros nuevos, a no ser que haya alcanzado el límite de número de veces que generará textos nuevos, por lo tanto, el sistema volverá al nodo anterior para intentar generar el poema por “otro camino”.

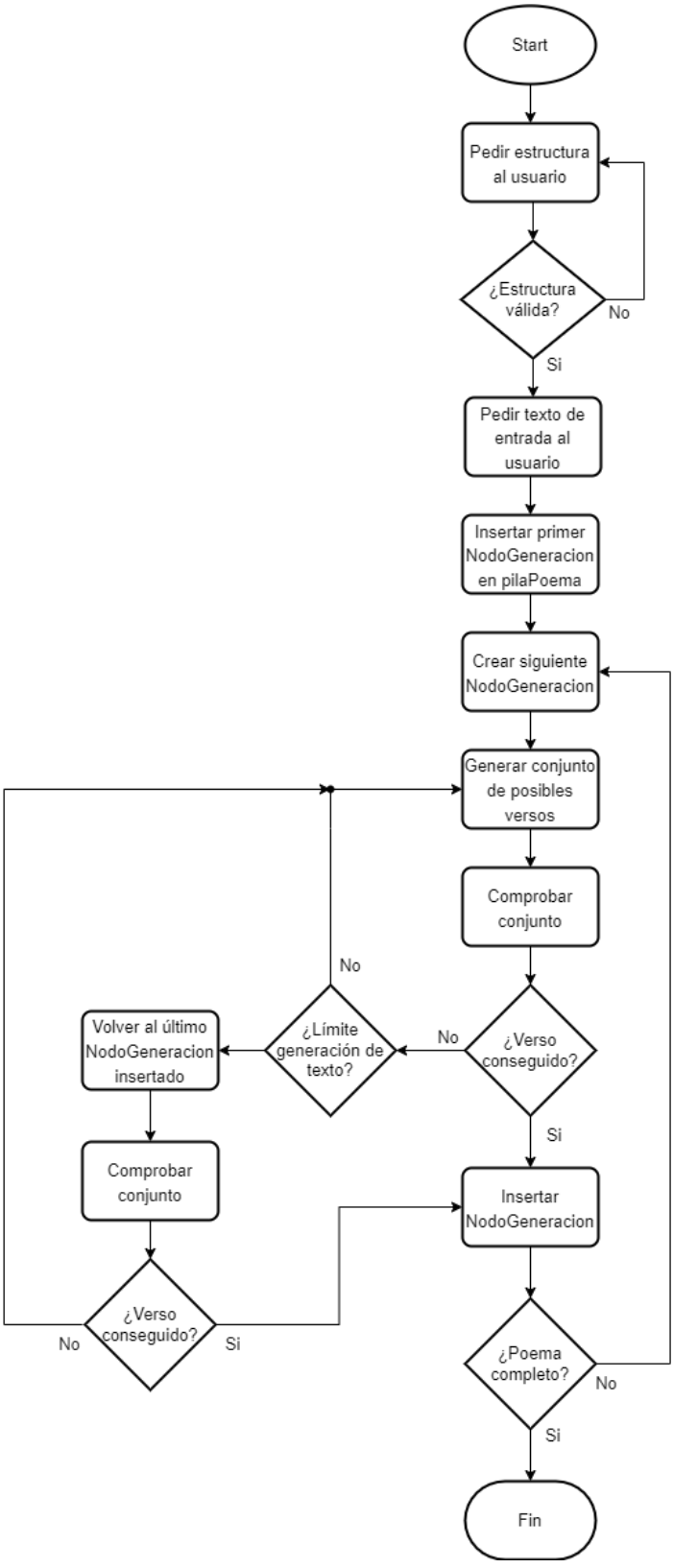

Ilustración 8 Diagrama de flujo correspondiente al sistema de generación de poemas

3 DESARROLLO

3.1 Implementación

3.1.1 *Fine-tuning* del modelo

La implementación de este proyecto se ha probado en dos tipos de entornos diferentes, en un entorno local y en *Google Collaboratory*.

A la hora de realizar el *fine-tuning* del modelo *GPT2-large-bne* en un entorno local se han detectado varios problemas, aunque el equipo utilizado cuente con una *RTX 2060 de Nvidia*, a la hora de ajustar utilizando la aceleración por *GPU,* el equipo no dispone de la memoria de vídeo para conseguir realizar esta tarea. En este entorno local se utilizó el entorno de desarrollo integrado (IDE) de *Visual Studio 2022.*

Tras haber fracasado en esta tarea, se pasó de un entorno local a *Google Collaboratory*, ya que este entorno es especialmente adecuado para tareas de aprendizaje automático. Tras adaptar el código realizado en el entorno local se ha conseguido el objetivo de ajustar el modelo con el *corpus* de poesía.

A la hora de utilizar el corpus, se han eliminado algunos contenidos del mismo, como por ejemplo los títulos de los poemas, ya que algunos de estos títulos eran simplemente números pertenecientes al sistema de numeración romano (*I, II, III, …*), siendo *tokens* no deseados a la hora de realizar un ajuste del modelo. Además, otra característica eliminada del corpus son las líneas en blanco, esto se realizó con el objetivo de acortar el tamaño del corpus tras no conseguir realizar un ajuste en un entorno local por falta de memoria.

Tras realizar varias pruebas generando texto con el modelo se observó un problema con los signos de puntuación, en muchos casos el modelo devolvía salidas con fragmentos tales como “,,,”, “-----“, etc … Por lo que se ha decidido eliminar los signos de puntuación del *corpus*, lo que solucionó este problema.

3.1.2 Métrica y rima

Tras haber conseguido ajustar el modelo, se continuó implementando el proyecto, esta vez a través del desarrollo de los métodos necesarios para medir la métrica de un verso y la rima entre dos palabras.

La métrica es la medida de los versos, representa la estructura y construcción del poema mediante un cómputo silábico que hace referencia al número de sílabas de cada verso. Este cómputo tendrá un conjunto de reglas que se deben cumplir a la hora de implementar:

1. Si un verso termina en palabra aguda se le sumará 1 a la métrica
2. Si un verso termina en palabra esdrújula se le restará 1 a la métrica
3. Si un verso termina en palabra llana se dejará la métrica igual
4. Sinalefa: Consiste en ajustar la métrica combinando la última sílaba de una palabra con la primera sílaba de la palabra siguiente siempre que terminen y comiencen en vocal respectivamente, esto hará que dos sílabas cuenten solamente como una.

Para obtener todas las sílabas que componen un texto se ha utilizado la librería *syltippy*, esta librería devolverá una tupla que contiene un array con las sílabas de la palabra y en la otra posición de la tupla nos marca la posición de la sílaba tónica. Para cada palabra de un verso se analizará si se forma sinalefa con la siguiente palabra y, si en el caso de ser la última palabra de la secuencia se comprobará si es aguda, llana o esdrújula. A continuación se muestra una representación en forma de pseudocódigo:

```
def metricaVerso(verso):
  silabasVerso = separarVersoSilabas
  for palabra in silabasVerso
    numSilabas += sinalefa(palabra, sigPalabra)
    if palabra es la última del verso
      numSilabas += palabraTonica(palabra)
    numSilabas += len(silabasPalabra)
```

El método *sinalefa* devolverá un -1 si se produce sinalefa con la palabra siguiente, y el método *palabraTonica* devolverá +1 si la palabra es aguda, 0 si es llana o -1 si es esdrújula.

Por otra parte, la rima según la Real Academia Española se define como una entidad de sonidos vocálicos y consonánticos o solo vocálicos a partir de la última vocal acentuada en dos o más versos. La repetición de fonemas se produce siempre a partir de la sílaba tónica.

La rima, al igual que la métrica, contiene una variedad de reglas las cuales se han necesitado implementar para desarrollar de manera correcta la detección de la rima:

1. Excepción de los diptongos: Cuando la zona de la rima contenga un diptongo, la vocal débil del diptongo se eliminará. Por ejemplo, "ac**ei**t**e**" puede rimar con "v**ete**" si eliminamos la vocal débil del diptongo ("i")

2. Excepción de las esdrújulas: A efectos de rima se puede ignorar la sílaba postónica en una palabra esdrújula. Por ejemplo, "c**án**t**i**c**o**" puede rimar con "z**anco**" si se elimina la vocal postónica ("ti").

A la hora de recoger los fonemas, extraemos la parte de la palabra a partir de la vocal tónica, por ejemplo "z**anco**" -> "anco". Se recogerán fonemas aparte para cada excepción anteriormente mencionada si es necesario, es decir, una palabra puede tener distintos fonemas. Posteriormente, se comparan todos los fonemas de una palabra con todos los fonemas de la otra palabra y si en uno de estas comparaciones se confirma que dos fonemas son iguales, existirá la rima. A continuación se muestra una representación en forma de pseudocódigo:

```
def rimaPalabras(p1,p2):
  posVocalTonicaP1  = recogerPosVocalTonica(p1)
  posVocalTonicaP2  = recogerPosVocalTonica(p2)
  fonemasP1 = recogeFonemas(p1)
  fonemasP2 = recogeFonemas(p2)
  comprobarFonemas(fonemasP1,  fonemasP2)
```

3.1.3 Sistema de generación de poemas

El funcionamiento del sistema viene descrito por el diagrama de flujo mostrado en la *Ilustración 8*. Se han utilizado todos los resultados obtenidos en los dos apartados anteriores.

Este sistema generará el poema verso a verso a través de la creación de objetos *NodoGeneracion* los cuales representan un paso dentro de todo este proceso.

La clase NodoGeneracion es la encargada de utilizar el modelo de lenguaje para generar posibles textos, aunque solo podrá realizar un número limitado de intentos, si se llega a este límite este nodo será descartado. A continuación se puede observar una representación en pseudocódigo de la función de generar texto.

```
def generarTexto(metricaObjetivo, metricaRestante, textoVerso):
  numGeneraciones += 1
  if numGeneraciones <= limite_generaciones
    outputs = localModel.generate(top_p, top_k, num_textos_generacion)
    for output in outputs:
      salidaMetrica = getTextoMetrica(output, metricaRestante)
      if metricaObjetivo == metricaVerso(salidaMetrica)
        arrayGeneracion.append(salidaMetrica)
        return True
      else:
        return False
```

Como podemos observar, se utilizará el modelo ajustado para generar una cantidad de textos estos textos serán recortados para ajustarse a la métrica que buscamos, es decir, si buscamos un verso de métrica 12, la función *getTextoMetrica* cortará el texto hasta conseguir una métrica de 12 o menor (si no es posible obtener un texto con métrica 12). Si el texto obtenido tiene una métrica igual a la del objetivo, se añadirá a los posibles versos los cuales se comprobarán posteriormente.

Si el *NodoGeneracion* ha llegado al límite de intentos de generación, devolverá un *False* para indicar a la clase *GeneradorPoema* que este nodo no es útil y debe volver hacia atrás.

La clase *GeneradorPoema* será la encargada de analizar los textos generados por cada *NodoGeneracion* y comprobar si existe alguno que cumpla con las reglas de métrica y rima anteriormente implementadas.

```
def generarPoema()
  estructura = pedirEstructura()
  if estructura ¡= None:
    entrada = pedirEntrada()
    primerNodo = NodoGeneracion(entrada)
    pilaPoema.append(primerNodo)
    while poemaNoCompleto:
      resultado = siguientePaso()
      if resultado == None:
        while resultado = None:
        resultado = anteriorPaso()
```

Este método pedirá la estructura y la entrada al usuario, posteriormente generará el primer nodo a partir del cual empezará a generar los siguientes versos del poema. Si al intentar dar un paso no se consigue un verso válido, el sistema retrocederá al nodo anterior para probar otra alternativa. A continuación se mostrarán las representaciones de los métodos *siguientePaso* y *anteriorPaso* en forma de pseudocódigo.

```
def siguientePaso(textoEntrada, numVerso)
  metricaObjetivo = estructura[numVerso]
  nodo = NodoGeneracion(textoEntrada)
  resultado = None
  while resultado == None:
    if nodo.generarTexto(metricaObjetivo, metricaRestante, textoVerso) == True
      arrayGeneracion = nodo.getArrayGeneracion()
      resultado = buscaGeneracion(arrayGeneracion)
    else
      return None
  return resultado
```

A la hora de dar un paso, el sistema generará un nodo nuevo con el que empezará a generar textos hasta encontrar un verso válido o hasta llegar al límite de intentos para ese nodo, lo que devolvería un resultado nulo.

El método *buscaGeneracion* comprobará todos los textos generados aplicando las reglas de rima y métrica anteriormente implementadas y ciertas reglas de estilo que dictan que un verso no debe terminar en conjunción, determinante o preposición, ya que así los resultados de los poemas son más coherentes.

```
def anteriorPaso()
  nodo = pilaPoema.getUltimoNodo()
  metricaObjetivo  = estructura[numVerso]
  resultado = None
  arrayGeneracion  = nodo.getArrayGeneracion()
  resultado = buscaGeneracion(arrayGeneracion)
  while resultado == None:
   if nodo.generarTexto(metricaObjetivo, metricaRestante, textoVerso) == True
     arrayGeneracion  = nodo.getArrayGeneracion()
     resultado = buscaGeneracion(arrayGeneracion)
   else
     return None
```

Para dar un paso hacia atrás en el desarrollo del poema se escogerá el último nodo introducido en la pila del poema, se comprobará si contiene algún verso que cumpla las reglas o se seguirá intentando generar texto con este nodo.

Finalmente, tras haber implementado todos estos objetivos, el sistema quedará completo para su uso.

3.1.3.1 Constantes utilizadas en la implementación

A lo largo del desarrollo se han definido constantes que pueden ser modificadas a gusto del usuario antes de la ejecución del generador:

1. **LIMITE_GENERACIONES:** Representa el número máximo de veces que se generarán posibles versos en un *NodoGeneracion.* En la implementación se

ha asignado un valor de *10* a esta constante con el fin de evitar que el sistema se retrase más de lo debido en un *NodoGeneracion* inservible.

2. **NUM_TEXTOS_GENERACION:** El número de salidas que generará el modelo de lenguaje en un intento. El sistema generará 10 textos en cada intento realizado por el *NodoGeneracion*.

3. **MIN_TOKENS_A_GENERAR:** Mínimo de tokens que se generarán por el modelo. El mínimo de tokens a generar por el modelo es de 20.

4. **MAX_TOKENS_A_GENERAR:** Mínimo de tokens que se generarán por el modelo. El máximo de tokens a generar por el modelo es de 25.

5. **CORTA_MITAD:** Un booleano que activará una función en el generador que consiste en dividir un nodo por la mitad al dar un paso hacia atrás en la creación del poema. Esta función está desactivada en el resultado final, por lo que este booleano tiene un valor *False*.

6. **CONST_TOP_P:** Es una variable dentro de la generación de texto por parte del modelo, se refiere al mínimo de probabilidad acumulada que debe tener un set de palabras para ser considerado como una salida. Teniendo en cuenta lo restrictivas que son las reglas de métrica y rima, esta constante tendrá un valor de *0.75,* este valor no es tan restrictivo y por lo tanto el sistema será capaz de encontrar versos adecuados con más facilidad.

7. **TOP_K:** Es una variable dentro de la generación de texto, es el número de sets de palabras que se escogen como posibles salidas (las k mejores), se puede utilizar en combinación con *CONST_TOP_P*. Esta constante equivale a 50, por lo que el modelo devolverá las 50 mejores salidas obtenidas.

4 EXPERIMENTACIÓN, RESULTADOS Y DISCUSIÓN

Esta sección está enfocada en los resultados obtenidos tras la implementación de este proyecto. Al ejecutar el *notebook* de *Google collaboratory* que contiene el sistema generador de poemas, en primer lugar debemos introducir una estructura y una entrada:

```
Primero de todo, deberá introducir una estructura, se mostrará el formato con un ejemplo:
-     6a/6b/ /7a/7b/ /5-/6a
cada número seguido de una letra es un verso, los espacios significan cambio de estrofa y el guión (-) significa que ese verso no tiene rima
el mínimo silábico es 2 y el máximo 14
Introduzca una estructura: 7-/11A/7A/7-/11B/7B/11-
['7-', '11a', '7a', '7-', '11b', '7b', '11-']
DiccionarioRimas: {'-': '', 'a': '', 'b': ''}
[(7, '-'), (11, 'a'), (7, 'a'), (7, '-'), (11, 'b'), (7, 'b'), (11, '-')]
Ahora debe introducir una entrada a partir de la cual generar el resto del poema
Introduzca una entrada: Sueños
```

Ilustración 9 Primeros pasos de la ejecución del programa

Tras insertar la estructura y la entrada que se desea en el poema resultante, el sistema mostrará los pasos que da a lo largo de la generación del poema.

```
GENERANDO VERSO 2
Estado de diccionario de rimas:
{'-': '', 'a': '', 'b': ''}
Estado de la pila del poema:
0 Sueños -->numVerso: 0
1 son los que nacen -->numVerso: 0
2 en mi vida de lo que ya es pasado -->numVerso: 1
Estado poema:
0 Sueños son los que nacen  <--- Metrica del verso: 7 - Rima:
1 en mi vida de lo que ya es pasado  <--- Metrica del verso: 11 a Rima:
2
3
4
5
6
```

Ilustración 10 Ejemplo de paso en la generación de un poema

Finalmente, el sistema mostrará por pantalla el resultado obtenido tras haber conseguido dar todos los pasos necesarios. Se mostrará la métrica de cada verso y su rima:

```
RESULTADO:
0 Sueños son los que nacen  <--- Metrica del verso: 7 - Rima: None
1 de los ojos de la noche la ausencia  <--- Metrica del verso: 11 a Rima: enca
2 de luz y la existencia  <--- Metrica del verso: 7 a Rima: enca
3 de mundos que se mueven  <--- Metrica del verso: 7 - Rima: None
4 en círculos en torno de otro mundo  <--- Metrica del verso: 11 b Rima: uo
5 de sombras de otros mundos  <--- Metrica del verso: 7 b Rima: uo
6 y de otras realidades que giran  <--- Metrica del verso: 11 - Rima: None
```

Ilustración 11 Resultado final de una ejecución del sistema generador

4.1 Resultados y discusión

Se presentarán algunos resultados obtenidos, los cuales se comentarán y discutirán, los poemas obtenidos se han guardado en el fichero *ResultadosFinales.txt* el cual se encontrará en la entrega de este trabajo. Se han utilizado estructuras típicas de la poesía hispánica a la hora de ejecutar el sistema:

-*Entrada*: Mirada

-*Estructura*: 8a/8-/8a

0 Mirada de muerte al borde <--- Metrica del verso: 8 a Rima: oe

1 de los fuegos del desierto <--- Metrica del verso: 8 - Rima:

2 En las sombras de la noche <--- Metrica del verso: 8 a Rima: oe

-*Tiempo de ejecución: 6 min 32 s*

-*Entrada*: Noches

-*Estructura*: 12A/12B/12A/ /11B/11C/11B/ /11C/11D/11C

0 Noches y días y noches y mañanas <--- Metrica del verso: 12 A Rima: aa

1 de sol y playa con un sol que ilumina <--- Metrica del verso: 12 B Rima: ia

2 mi alma Todo lo recuerdo en alboradas <--- Metrica del verso: 12 A Rima: aa

3

4 nocturnas con las olas en la orilla <--- Metrica del verso: 11 B Rima: ia

5 de la mar sin más patria que mi amor <--- Metrica del verso: 11 C Rima: o

6 propio y mi soledad infinita <--- Metrica del verso: 11 B Rima: ia

7

8 Todo eso fue un sueño que fue ayer y hoy <--- Metrica del verso: 11 C Rima: o

9 es una pesadilla que me hace <--- Metrica del verso: 11 D Rima:

10 temblar y sentir el aliento de un dios <--- Metrica del verso: 11 C Rima: o

- *Tiempo de ejecución: 56 min 57s*

-*Entrada*: Días

-*Estructura*: 12-/12-/12-/ /11-/11-/11-/ /11-/11-/11-

0 Días en los que a la memoria se aferran <--- Metrica del verso: 12 - Rima: None

1 los corazones humanos y no saben <--- Metrica del verso: 12 - Rima: None

2 qué hacer ni con qué amarlo Aún los siglos <--- Metrica del verso: 12 - Rima: None

3

4 de horror y horror que la historia <--- Metrica del verso: 11 - Rima: None

5 ha tatuado en el pecho son mayores <--- Metrica del verso: 11 - Rima: None

6 que los años que un día el mar anduvo <--- Metrica del verso: 11 - Rima: None

7

8 a sus orillas Anoche la tierra <--- Metrica del verso: 11 - Rima: None

9 en su lecho de arenas movedizas <--- Metrica del verso: 11 - Rima: None

10 y roídas de lodo levantó <--- Metrica del verso: 11 - Rima: None

- *Tiempo de ejecución: 5min 0s*

-*Entrada:* Ventana

-*Estructura:* 7a/11B/7a/7b/11B

0 Ventana de otro cielo <--- Metrica del verso: 7 a Rima: eo

1 abierto y al otro lado del mar <--- Metrica del verso: 11 b Rima: a

2 la luna el sol el viento <--- Metrica del verso: 7 a Rima: eo

3 la luz la brisa la sal <--- Metrica del verso: 7 b Rima: a

4 el vino el fuego y el azufre el mar <--- Metrica del verso: 11 b Rima: a

- *Tiempo de ejecución: 19min 42s*

-Entrada: Atardecer

-Estructura: 8a/8-/8a/8-/8a/8-/8a/8-/8a/8-

0 Atardecer es de plata <--- Metrica del verso: 8 a Rima: aa

1 y oro un cielo de perlas <--- Metrica del verso: 8 - Rima: None

2 con una estrella dorada <--- Metrica del verso: 8 a Rima: aa

3 que brilla en el mar de su alma <--- Metrica del verso: 8 - Rima: None

4 y en los ojos del alma <--- Metrica del verso: 8 a Rima: aa

5 mía que a mi amor pone el sol <--- Metrica del verso: 8 - Rima: None

6 en su frente y se apaga <--- Metrica del verso: 8 a Rima: aa

7 en la noche de mi vida <--- Metrica del verso: 8 - Rima: None

8 Un cielo que se derrama <--- Metrica del verso: 8 a Rima: aa

9 en mi alma que no tiene fin <--- Metrica del verso: 8 - Rima: None

- Tiempo de ejecución: 23min 39s

Observamos que todos los resultados consiguen respetar las reglas de rima y estructura impuestas por el usuario al introducir los parámetros de ejecución necesarios.

El tiempo de ejecución varía bastante, siendo el mayor de estos un tiempo de *56 min 57s*, tras el cual, se ha obtenido un tiempo de *5min 0s* en un poema con la misma estructura pero con versos libres, lo que da a entender que la rima es la causa de que el tiempo de ejecución varíe tanto, ya que es la regla más restrictiva de las que se aplican en la poesía.

Otra observación es la baja presencia de rimas consonantes, esto se debe a que es el tipo de rima más restrictivo en comparación a la rima asonante, ya que debe de hacer coincidir más caracteres para conseguir la repetición de fonemas.

5 CONCLUSIONES Y TRABAJOS FUTUROS

Tras haber implementado el proyecto y observar los resultados obtenidos se puede decir que el proyecto ha sido finalizado con éxito, obteniendo resultados acordes a los objetivos propuestos en el *apartado 1.2* de esta memoria. Se ha conseguido ajustar un modelo de lenguaje con el fin de realizar poemas y se ha desarrollado un sistema que implemente todas las reglas necesarias para la obtención de textos poéticos que contengan métrica y rima.

Durante todo el desarrollo de este trabajo se han adquirido conocimientos relacionados con el PLN tales como el uso de modelos del lenguaje basados en la arquitectura *Transformers* y la tarea de generar texto a través del *Casual Language Modeling*. También se ha adquirido mayor experiencia como programador utilizando *Python* en un entorno como *Collaboratory*.

Es importante señalar que es igual de necesario tener conocimientos a la hora de la implementación de un sistema en cuanto a la programación como a la hora de realizar un diseño que se ajuste a los objetivos y al alcance del proyecto, lo que podría haberse realizado de mejor manera, siendo uno de los mayores problemas a la hora del desarrollo la insuficiente organización.

En cuanto a los posibles trabajos futuros, es cierto que existen apartados en los que mejorar, como por ejemplo la poca eficiencia que tiene la implementación a la hora de encontrar rimas, lo que provoca que el tiempo de ejecución varíe bastante y pueda llegar a tardar más de una hora.

Otra mejora posible es la de orientar los versos a una temática con la intención de mejorar la coherencia de los mismos. Esto podría llevarse a cabo con la introducción de restricciones adicionales que aseguren una mínima afinidad semántica entre los distintos versos o que mantengan una misma tónica emocional.

6 APÉNDICES

6.1 Guía original del Trabajo Fin de Título

Descripción corta del TFG

Usando modelos de redes neuronales profundas, se pretende, a partir de un modelo pre-entrenado para el español, ajustar el modelo a la generación de poesía, entrenando con un amplio repertorio de obras. El proyecto explorará la posibilidad de uso de un algoritmo Beam Search para generar versos ajustados en métrica y rima.

El resultado debería ser un sistema de inteligencia artificial de generación del lenguaje capaz de componer poemas completos a partir de un texto semilla.

Conocimientos/Requisitos previos recomendados

- Programación en Python
- Procesamiento del lenguaje natural

Objetivos del TFG

- Ajustar un modelo pre-entrenado para el español a la generación de versos
- Explorar el algoritmo Beam Search como solución a la generación de texto con restricciones.
- Definir restricciones de rima y métrica e integrarlas en Beam Search
- Evaluar los resultados obtenidos y proponer trabajo futuro

Metodología a desarrollar

- Dado el carácter experimental del proyecto, se llevará a cabo un primer análisis de los modelos pre-entrenados disponibles y su viabilidad para el ajuste a la generación de textos poéticos
- Tras ese primer análisis, se investigará la implementación del algoritmo Beam Search para permitir "backtracking" en el proceso de generación de tokens, de acuerdo a las restricciones de métrica y rima consideradas

- Se implementará un sistema final, con una interfaz web básica, que permita demostrar la solución final
- Durante todo el proyecto, se llevará a cabo iteraciones de diseño experimental, experimentación y evaluación

6.2 Poemas generados

En este apartado se mostrarán todos los poemas que se encuentran en el fichero *ResultadosFinales.txt* sin mostrar datos sobre la rima y métrica para facilitar su lectura.

Resultado 1:

Mirada de muerte al borde

de los fuegos del desierto

En las sombras de la noche

Resultado 2:

Noches y días y noches y mañanas

de sol y playa con un sol que ilumina

mi alma Todo lo recuerdo en alboradas

nocturnas con las olas en la orilla

de la mar sin más patria que mi amor

propio y mi soledad infinita

Todo eso fue un sueño que fue ayer y hoy

es una pesadilla que me hace

temblar y sentir el aliento de un dios

Resultado 3:

Días en los que a la memoria se aferran

los corazones humanos y no saben

qué hacer ni con qué amarlo Aún los siglos

de horror y horror que la historia

ha tatuado en el pecho son mayores

que los años que un día el mar anduvo

a sus orillas Anoche la tierra

en su lecho de arenas movedizas

y roídas de lodo levantó

Resultado 4:

Lluvia de ternura en los labios Anoche

la vio pasar y el río suspirando

y se detuvo un instante entre sus brazos

y sus piernas como un río que se rompe

Resultado 5:

Vientos de invierno de la tierra en el mar

El día llega a sus primeros puntos

de nieve El sol es un viejo astro que ya

no alumbra la vida En las aguas del mar

del Norte la última aurora está a punto

Resultado 6:

Raíces viejas y ramas

viejas se hunden en mi alma

y en el polvo del tiempo

y la muerte En el silencio

de la noche las campanas

Resultado 7:

Alegres los que en ti creen

y en tus alas unieron

los muertos en la historia

del alma mía Los muertos

Resultado 8:

Ventana de otro cielo

abierto y al otro lado del mar

la luna el sol el viento

la luz la brisa la sal

el vino el fuego y el azufre el mar

Resultado 9:

Prados y praderas verdes

de fresnos y olivares

centenarios

encinas y robles crecen

y crecen en el paisaje

castellano

Resultado 10:

Atardecer es de plata

y oro un cielo de perlas

con una estrella dorada

que brilla en el mar de su alma

y en los ojos del alma

mía que a mi amor pone el sol

en su frente y se apaga

en la noche de mi vida

Un cielo que se derrama

en mi alma que no tiene fin

Resultado 11:

Jaén es un reino rico y generoso

que tiene por tesoro el cielo y sus arcas

ricas y hondas llenas de plata y de oro

llenas del oro y el zafiro que guarda

el sol y los astros ricos y bellos ojos

de los cielos y la gracia de las montañas

Resultado 12:

Sueños son los que nacen

de los ojos de la noche la ausencia

de luz y la existencia

de mundos que se mueven

en círculos en torno de otro mundo

de sombras de otros mundos

y de otras realidades que giran

Resultado 13:

Cielo tormentoso todo se espesea

Todo se cubre de estrellas

de hermosura ignota y en su velo

se proyecta el azul de la tierra

Por todas partes el viento

lleva la fragancia de lo eterno

que en los brazos de las piedras

nos ciñe y nos encierra

Resultado 14:

Estrellas que se apagan y se encienden

cuando se quiere dar luz a la tierra

y luego se queman en la hoguera

del sol que la encierra y que no vuelve

a salir nunca mas y es de un día

para otro la hora nueva que a veces

me hace temblar como una herida

7 DEFINICIONES Y ABREVIATURAS

- IA (Inteligencia Artificial): Disciplina que intenta replicar y desarrollar la inteligencia y sus procesos implícitos a través de computadoras.
- ML (Machine Learning): Rama de la IA que busca que una máquina *aprenda* a realizar una tarea a partir de un conjunto de datos, lo que en otros casos se realizaría con instrucciones explícitas indicadas por un experto
- DL (Deep Learning): Subcampo del ML, se diferencia de este por la automatización de extracción de características y el uso de redes neuronales.
- BoW (Bag of Words): tipo de representación donde los textos se representaban mediante vectores que contabilizaban el número de veces que aparecía una palabra
- RNN (Redes neuronales recurrentes): es un tipo de red neuronal artificial especializada en procesar datos secuenciales o series temporales cuya arquitectura permite que la red obtenga memoria artificial.
- GPT (Generative Pre-Training Transformer): modelo de lenguaje basado en la arquitectura *Transformer,* publicado en 2018 por *OpenAI.*
- GPT2 (Generative Pre-Training Transformer 2): Sucesor de GPT.
- CLM (Casual Language Modeling): Tarea de PLN en la que los modelos deben predecir el siguiente token de una secuencia a partir de los tokens anteriores, es decir, el modelo calcula el *token i + 1* a partir de los *token i*, *i-1, i-2, …*

8 BIBLIOGRAFÍA

- Alec Radford, K. N. (2018). *Improving Language Understanding by Generative Pre-Training.*
- Arrabales, R. (29 de Marzo de 2016). *Deep Learning: qué es y por qué va a ser una tecnología clave en el futuro de la inteligencia artificial*.
- Ashish Vaswani, N. S. (12 de Junio de 2017). *Attention Is All You Need.*
- Asier Gutiérrez-Fandiño, J. A. (15 de Julio de 2021). *MarIA: Spanish Language Models.*
- BBVA. (8 de Noviembre de 2019). *'Machine learning': ¿qué es y cómo funciona?*
- Berkeley school of information. (26 de Junio de 2020). *What is Machine Learning (ML)?*
- Burns, E. (Marzo de 2021). *What is deep learning?*
- ComputerWeekly. (Enero de 2017). *Aprendizaje automático (machine learning)*.
- Gugger, S. (2021). *Fine-tuning a language model.*
- Haugeland, J. (1985). *Artificial Intelligence: the very idea.*
- IBM. (15 de Julio de 2020). *Machine Learning*.
- IIC. (s.f.). *Evolución del PLN y los Transformers en Spain AI*.
- Kavlakoglu, E. (2020). *AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the Difference?*
- Kurzweil, R. (1990). *The Age of Intelligent Machines.*
- McDermott, C. y. (1985). *Introduction to Artificial Intelligence.*
- Medium. (20 de Abril de 2018). *Requerimientos Funcionales y No Funcionales, ejemplos y tips.*

- OpenAI. (14 de Febrero de 2019). *Better Language Models and Their Implications*.
- OpenAI. (2019). *GPT-2: 1.5B Release.*
- Poole, D. (1998). *Computational Intelligence: A Logical Approach.*
- Pressman, R. (2010). *Ingeniería del Software.* McGraw-Hill.
- Santander becas. (21 de Diciembre de 2020). *Metodologías de desarrollo de software: ¿qué son?*
- Santander, C. (14 de Abril de 2021). *¿Qué son los Transformers?*
- Sotaquirá, M. (30 de Junio de 2020). *Redes Transformer (... o el fin de las Redes Recurrentes)*.
- Stsepanets, A. (29 de Octubre de 2021). *Modelo de cascada ('Waterfall'): qué es y cuándo conviene usarlo.*
- Tom B. Brown, B. M.-V. (2020). *Language Models are Few-Shot Learners.*
- Uszkroeit, J. (31 de Agosto de 2017). *Transformer: A Novel Neural Network Architecture for Language Understanding*.
- Vaca, A. (s.f.). *Transformers en Procesamiento del Lenguaje Natural*.