\chapter{Objetivos del trabajo y planteamiento de la solución}

\label{cap:objetivosDelTrabajoYPlanteamientoDeLaSolucion}

En el presente capítulo se detallarán los objetivos y requisitos clave del sistema a realizar en este trabajo junto con las restricciones para tener en cuenta. A continuación, se describirá el planteamiento inicial del sistema a través del estudio de una arquitectura general y la presentación de las herramientas que se van a necesitar durante el desarrollo. Para finalizar este punto, se realizará un exhaustivo estudio de las opciones disponibles para la implementación del sistema y se expondrán las cuestiones sobre las que se apoyan las decisiones de implementación tomadas.

\section{Objetivos y requisitos}

\label{sec:ObjetivosYRequisitos}

Pese a los avances informáticos de las últimas décadas, la composición de historias de vida sigue sustentando su proceso en medios rudimentarios.

La dificultad de composición es palpable debido a todas las tareas a realizar, generalmente, a través de medios humanos. Como paso previo a la redacción final de la historia de vida, debe realizarse una selección adecuada de los datos que se van a incluir sin dejar ninguna información relevante atrás. También hace falta concebir la estructura del escrito y que datos corresponden a cada una de dichas partes. Si se consigue hacer este proceso de manera satisfactoria, se puede comenzar con la escritura de la historia de vida. Quienes escriben estas historias suele ser personal terapéutico, y dado que escribir no es para nada una labor sencilla, pueden tener ciertas dificultades a la hora de buscar combinaciones perfectas de palabras que expresen ciertos conceptos, perder tiempo leyendo una y otra vez una misma oración que no parece convencerles o simplemente quedarse sin ideas. Si existiera un sistema de composición automático de estas historias, podrían ver su labor facilitada dado que la tarea de redacción humana sería sustituida por un procedimiento automático más rápido. Sin embargo, la existencia de sistemas de generación de historias de vida, o similares como notas biográficas, es prácticamente nula. Es por ello por lo que el sistema que se pretende desarrollar en este trabajo tratará de responder a este problema en particular.

Después de analizar y evaluar la problemática en cuestión, se puede resumir el objetivo del sistema como el desarrollo de un sistema de generación de lenguaje que tome como entrada unos datos correspondientes a información sobre la vida de una persona y sea capaz de transformarlos en un texto redactado en lenguaje natural. De la anterior descripción se derivan dos requisitos claros para nuestro sistema. Por una parte, la capacidad de conversión de datos a textos y por otra, la comprensión y representación de datos de tipo biográfico. Cabe mencionar que también deberán verse reflejados en el escrito final otros datos no puramente biográficos como pueden ser anécdotas personales, gustos, relatos de situaciones y un largo etcétera.

Finalmente, el sistema a desarrollar deberá estar libre de alucinaciones, de degeneraciones o, de cualquier otro tipo de fenómeno que perjudique la representación de los datos en lenguaje natural y la comprensión del escrito. También se considerará la generación de un texto fluido y coherente como resultado de la conversión de estos datos proporcionados como entrada a texto.

Para terminar esta sección, se plantean una serie de limitaciones con las que va a contar el sistema debido generalmente a restricciones ajenas al trabajo. Por una parte, como se comprobará más adelante, la consideración de generación de datos en castellano no se realizó debido a la inexistencia de estos, por lo que el sistema partirá y generará información en inglés, ya que en esta lengua el acceso a recursos es mucho más alcanzable. Otra limitación viene dada por la disponibilidad básica de recursos computacionales. En general, el entrenamiento de grandes cantidades de datos en un modelo neuronal necesita costosos computadores y una gran cantidad de tiempo que no se puede llegar a considerar en el ámbito de un trabajo de estas características. Por estas razones, algunas tecnologías deberán ser descartadas.

\section{Planteamiento inicial}

\label{sec:PlanteamientoInicial}

En este apartado se presenta toda aquella información imprescindible para comenzar a desarrollar el sistema de este trabajo. Primero, se considerará la estructura de la arquitectura del sistema general que va a estar compuesto necesariamente de varios componentes, cada uno dedicado a la realización de una tarea en concreto. Para finalizar, se expondrán las herramientas principales que se van a utilizar durante el desarrollo de este.

\subsection{Arquitectura del sistema}

Para construir el sistema completo que se pretende desarrollar en este trabajo, se han considerado las siguientes capas a implementar tal cual están representadas en la figura \ref{fig:arquitectura}.

Dado que los datos de entrada que componen la información a representar en la historia de vida, por lo general, no se encuentran en un formato adecuado, se implementará esta capa cuyo objetivo será precisamente la conversión de esta información a datos manejables por el modelo de lenguaje. Entre las tareas de esta capa destaca la creación, a partir de los datos de entrada, de un grafo de conocimiento del que posteriormente se extraerán tripletas semánticas que representen su información. Otras tareas que realizar por esta capa será la conversión de estas tripletas a un formato adecuado para que el modelo de lenguaje pueda procesarlas.

El modelo de lenguaje es el núcleo de nuestro sistema. Para que dicho modelo pueda desempeñar las tareas correspondientes de generación, se seleccionará un modelo pre-entrenado basado en redes neuronales idealmente alineado a los objetivos del trabajo y se ajustará bajo un conjunto de datos limpio y con un formato legible por el modelo. De esta manera, formaremos un modelo sobre el que se introducir unos datos de entrada y que potencialmente generará una información coherente y fluida en lenguaje natural.

Una vez generado el texto por el modelo de lenguaje, la información obtenida deberá ser transformada a un texto completo, coherente y legible por el usuario. Así, toda la información obtenida deberá unirse en una sola información a presentar si no estuviera ya estructurada de dicha manera. Esta capa podría contar con algún componente de traducción automática sencillo que tradujera los resultados al castellano, ya que el texto resultante de todo el proceso será inglés.

\end{itemize}

\subsection{Software y herramientas}

Antes de de desarrollar el sistema, es conveniente plantearnos que software y herramientas van a ser necesarias durante el proceso de desarrollo del sistema.

Como primera consideración, la biblioteca de software \textit{Hugging-Face} será utilizada para la implementación del modelo de lenguaje y la descarga de algunos de los conjuntos de datos utilizados, dado que pone a la disposición del usuario una gran cantidad de funcionalidades además de las ya mencionadas. Otro de los puntos a favor de usar esta biblioteca, es la integración de esta herramienta con las bibliotecas de aprendizaje profundo \textit{PyTorch} y \textit{TensorFlow} que nos permitirán realizar los cálculos del sistema a través de tensores en la GPU. Esta herramienta proporciona tecnologías de procesamiento de lenguaje natural de código abierto que giran especialmente en torno a la biblioteca \textit{Transformers}. Esta última funcionalidad permite el acceso a una gran cantidad de modelos pre-entrenados y sus variaciones en forma, tamaño y arquitectura para distintas tareas, facilitando la construcción del modelo.

Para finalizar, se utilizará el entorno virtual conocido \textit{Google Colaboratory} (\textit{Google Colab} en su forma abreviada), dada la facilidad de ejecución y de programación a través del navegador sin necesidad de configuración previa y

con acceso gratuito a GPUs. La versión utilizada de este software dispone de las siguientes especificaciones: procesador Intel(R) Xeon(R) CPU a 2.3GHz de frecuencia, 12 GB de memoria RAM y una GPU modelo Tesla k80.

\section{Decisiones de implementación}

\label{sec:DecisionesDeImplementacion}

Atendiendo a los módulos que componen la arquitectura de nuestro sistema, el núcleo del trabajo gira en torno a la generación de texto a través del modelado de lenguaje. Como se expuso en las conclusiones generales (sección \ref{sec:2conclusionesgenerales}), la estructura más conveniente para nuestro sistema corresponde a implementaciones neuronales de modelos de lenguaje, concretamente modelos pre-entrenados \textit{Transformers}.

La decisión de utilizar este tipo de modelo deriva esencialmente de las restricciones computacionales de las que partimos. Por una parte, si tuviéramos que entrenar una red neuronal bajo una gran cantidad de datos, el computador requerido necesitaría una serie de características computacionales enormes y no se dispone de ellas. Además, el tiempo necesario para llevar a cabo este entrenamiento sería considerable. Ejemplos de estas limitaciones podrían ser grandes modelos de lenguaje como \textit{GPT} que tomó un mes entero de entrenamiento sobre ocho GPUs.

Seleccionado el tipo de modelo, componente núcleo de nuestro sistema para la generación de texto, debemos escoger los datos que utilizaremos para ajustarlo y el modelo concreto que utilizaremos. Para ello se realizaron varias pruebas y se tomó la decisión en base a aquel que en mejores resultados originara. A continuación, se muestra este proceso de selección de conjuntos de datos y modelos considerados en la construcción del modelo de lenguaje final.

\subsection{Conjuntos de datos y preparación para el entrenamiento}

\label{sec:ConjuntosDeDatosYPreparacionParaElEntrenamiento}

En este apartado se presentan soluciones de datos efectivas para cubrir los requisitos de generación del sistema. Entre estos requisitos se encuentra la generación de textos que representen textualmente información bibliográfica de una persona. Otra de las necesidades a cubrir es la generación textual a partir de unos datos precisados como entrada al sistema.

Los modelos pre-entrenados si bien no necesitan enormes conjuntos de datos para realizar un entrenamiento desde cero, deben ser ajustados a tareas específicas haciendo uso de conjuntos de datos de tamaños mucho más pequeños.

Son muchas las \textit{datasets} existentes para realizar tareas de ajuste de modelos. Algunos corpus como \textit{News Aggregator}\footnote{Disponible en Kaggle https://www.kaggle.com/datasets/uciml/news-aggregator-dataset} pueden ser utilizados para la creación de noticias. Otra tarea puede ser la generación de recetas, utilizando para ello \textit{datasets} como \textit{recipe-box}\footnote{Disponible en GitHub https://github.com/rtlee9/recipe-box}.

Es importante realizar una correcta elección del conjunto de datos sobre el que se va a entrenar el modelo. La elección del conjunto de datos no es trivial, sino que obedece a las necesidades establecidas: en nuestro caso, generación de texto biográfico a partir de unos datos especificados como entrada.

Así, se realizó una búsqueda de conjunto de datos apropiados a las características necesarias para construir el sistema propuesto en este trabajo, resultando esta búsqueda en tres grandes conjuntos de datos: \textit{Wiki2bio}, \textit{WebNLG} y \textit{KELM}, que serán descritos a continuación.

\subsubsection{Wiki2bio}

Wiki2bio es un conjunto de datos propuesto por \citep{lebret-etal-2016-neural}. Fue creado debido a la necesidad de existencia de una gran base de datos que permitiera la redacción notas biográficas. Hasta entonces, las bases de datos con información bibliográfica eran demasiado pequeñas para entrenar un modelo de red neuronal, por lo que se ideó la construcción de un conjunto de una orden de magnitud superior. Compuesto por más de 700.000 ejemplos y un vocabulario de 400.000 palabras, extrajeron todos estos datos mapeando los datos contenidos en las tablas de información de Wikipedia con los textos descriptivos escritos en lenguaje natural.

Antes de realizar cualquier entrenamiento en el modelo bajo un conjunto de datos es necesario realizar un análisis y preprocesamiento previo que nos permita conocer más a fondo dicho conjunto.

Existen diferentes métodos para obtener datos de Wiki2bio. La manera más sencilla para descargar el conjunto de datos completo es hacer uso de las API de \textit{HuggingFace} o \textit{TensorFlow}, ya que devuelven los datos encapsulados en un objeto procesable con un conjunto de herramientas adecuadas para la tarea. En este caso, nos quedaremos con la primera opción, ya que posteriormente emplearemos los modelos de lenguaje de esta misma API.

El conjunto completo de datos se descarga finalmente a través de la biblioteca \textit{datasets} de HuggingFace\footnote{Disponible en https://huggingface.co/datasets/wiki\\_bio} que devuelve un objetivo de tipo \textit{DatasetDict}. El archivo final ocupa 738.19 MB y está compuesto por un total de 728.321 muestras.

El conjunto de datos se divide aleatoriamente en tres subconjuntos: conjunto para entrenamiento (80\%), conjunto para validación (10\%) y conjunto para prueba (10\%), constando cada uno de ellos de 582.659, 72.831 y 72.831 ejemplos, respectivamente.

Un ejemplo de los cualesquiera presentes en este conjunto de datos es representado en la figura~\ref{fig:ex\_wiki2bio}. Como se puede apreciar, la información de cada uno de los ejemplos se divide en dos grupos. Por una parte, el \textit{input\\_text} o texto de entrada especifica el contexto del que se están proporcionando los datos y el cuadro de información de Wikipedia en formato tabular. Los datos contenidos en esta tabla establecen asignaciones a características del contexto como son la nacionalidad, fecha de nacimiento o trabajo. El otro grupo corresponde a la información de destino o \textit{target\\_text} que representa en lenguaje natural la información contenida en el contexto y la tabla anterior.

Después de descargar con éxito la base de datos completa, se ha realizado una limpieza de los datos ya que se pudo apreciar que en numerosas ocasiones aparecen saltos de línea o algunos caracteres como paréntesis o corchetes codificados como símbolos en mitad de palabras u oraciones y esto podría llevar a la generación de datos incorrectos e imprecisos después del entrenamiento que se va a realizar. El resultado de esta limpieza de los datos llevada a cabo se muestra en la figura \ref{fig:ex\_wiki2bio\_clean}.

\subsubsection{WebNLG}

Introducido por \citep{gardent2017creating}, el corpus WebNLG se compone de un conjunto de tripletas semánticas, que describen hechos en forma de entidades y relaciones entre ellas, y los eventos correspondientes presentados en lenguaje natural. Este conjunto de datos nace de la explosión de desarrollo de bases de datos RDF (\textit{Resouce Description Format}), cuya entidad atómica de datos es precisamente la tripleta semántica. Una tripleta está formada por un conjunto de tres entidades que codifica una relación entre datos semánticos, en el caso de WebNLG, expresada de la forma <sujeto - predicado - objeto>. Para la creación de esta base de datos se parte de la base de conocimiento DBPedia construida a partir de datos estructurados de Wikipedia. Según mencionan sus autores, esta base de datos se crea con la finalidad de suplir las tareas de lexicalización, agregación de oraciones y realización de la arquitectura tradicional presentada en la sección~\ref{sec:arquitectura\_tradicional}.

Esta base de datos\footnote{Disponible en https://gitlab.com/shimorina/webnlg-dataset/-/tree/master/} está disponible en inglés, aunque la tercera versión del conjunto de datos también incluye una colección de datos en ruso (es necesario precisar que los conjuntos de datos en idiomas diferentes se encuentran separados). Un total de 45.050 oraciones componen WebNLG, dividiéndose a su vez en tres subconjuntos de datos: \textit{train}, \textit{dev} y \textit{test}, conteniendo cada uno 35.426, 4.464 y 5.150 oraciones, respectivamente.

El formato de un ejemplo del \textit{dataset} se encuentra en la figura~\ref{fig:webnlg\_ex}. Como se puede comprobar, cada ejemplo está formado por diversos campos. En primer lugar, aparece una serie de información acerca de los datos que nos vamos a encontrar en el ejemplo como el id, categoría, forma, tipos de estructura de las tripletas y número de estas. La categoría representa el tipo de datos de la entidad representada en la DBPedia y el 'eid' un id único en el subconjunto de datos y categoría para el ejemplo. Además, se muestra el número de tripletas en el conjunto, representado por el campo 'size'. Por otra parte, se tiene en cuenta la forma del árbol formado por el conjunto de tripletas en el campo 'shape', que representa dicho árbol en forma de cadena formado por paréntesis anidados donde X es un nodo, este formato se basa en la representación de árboles de Newick \citep{cardona2008extended}.%%TERMINAR

En el campo 'shape\\_type' se indica si el objeto de una tripleta es el sujeto de otra (\textit{chain}) si las tripletas tienen un tema compartido (\textit{siblings}) o si se dan ambos casos (\textit{mixed}).

Finalmente, se muestra la información más interesante de este conjunto de datos. Junto con unos ids y la calidad de las lexicalizaciones (\textit{good} o \textit{bad} en el campo 'comment') aparecen diferentes oraciones (campo 'text') que representan en lenguaje natural la información dada por los conjuntos de tripletas ('modified\\_triples\\_sets' y 'original\\_triple\\_sets'). Es destacable la existencia de varias oraciones para representar la misma información de un solo conjunto de tripletas y de varios conjuntos de tripletas representantes todas de la misma información.

Dado que el formato en el que descargamos los ficheros que componen esta \textit{dataset} es XML, se cargaron los datos y se pasaron a un objeto que pudiera entrenar el modelo, en este caso un \textit{DataFrame} que finalmente incluía 73.119 ejemplos para el entrenamiento. Posteriormente, se modificaron los datos para crear la entrada definitiva al modelo uniendo toda la información de entrada en una sola cadena de caracteres. Esta entrada, conformada por las tripletas, se estructura de la siguiente manera: objeto1 | relacion1 | atributo1 \&\& objeto2 | relacion2 | atributo2 ...; separando cada tripleta por dos \textit{ampersand} y cada entidad por una barra vertical.

A continuación, se utilizaron expresiones regulares para limpiar cada uno de los ejemplos que componían la base de datos ya que incluía demasiados símbolos que podrían dificultar su comprensión. Las tareas de limpieza llevadas a cabo comprendieron la eliminación de símbolos tales como guiones bajos, acentos circunflejos, comillas dobles, expresiones de identificación de tipos de números o alusiones a páginas web y la división de las relaciones según las letras mayúsculas y estableciéndolas en minúsculas. Después de procesar los datos para asignar a cada uno de los conjuntos de tripletas cada una de las soluciones en lenguaje natural, el resultado final se muestra en la figura~\ref{fig:webnlg\_ex\_own}.

KELM (\textit{Knowledge-Enhanced Language Model}) es un enorme corpus de datos en inglés construido específicamente para un sistema \textit{data-to-text} llamado TeKGen (\textit{Text from KG Generator}) propuesto por \cite{agarwal-etal-2021-knowledge}. Este sistema se basa en un modelo secuencia a secuencia para generar texto en lenguaje natural a partir de un grafo de conocimiento o \textit{Knowledge Graph} (KG) presentado a través de tripletas semánticas. La motivación de construcción de este conjunto de datos surge de la necesidad de variedad de entidades y relaciones que, según sus autores, WebNLG no tenía. De acuerdo con las cifras que mencionan, los datos de Wikidata (base de conocimiento con datos estructurados que sirve como base de datos secundaria para dar soporte a Wikipedia y como la que se ha construido el corpus KELM) podrían representarse en aproximadamente 6 millones de entidades y 1500 relaciones, frente a las 600 entidades y 20 relaciones de WebNLG.

Para crear este enorme corpus de datos, sus autores utilizaron supervisión a distancia para alinear las tripletas de Wikidata con el texto de Wikipedia. Para cada una de las entidades de Wikidata, se seleccionaron como oraciones candidatas aquellas que aparecen en la primera sección de su página de Wikipedia. A continuación, emparejaron con cada frase de esta sección las tripletas que contenían la entidad como sujeto. De esta manera, una tripleta puede alinearse con varias oraciones de la sección y a su vez, una sección puede alinearse con varias tripletas. Como observación, realizaron una heurística de sustitución de pronombres ya que en numerosas ocasiones las oraciones contenían este tipo de elemento sintáctico.

Finalmente, el conjunto de datos resultante está compuesto, según las cifras oficiales proporcionadas por sus creadores, por más de 18 millones de oraciones, 45 millones de tripletas y 1500 relaciones diferentes.

Al igual que las bases de datos anteriores, KELM se encuentra disponible para descargar desde la API de HuggingFace\footnote{Disponible en https://huggingface.co/datasets/kelm} y ocupa en total de 3.08 GB de tamaño y 7.964.073 ejemplos en total. Está dividida en 3 subconjuntos de datos: entrenamiento, prueba y validación. El número de ejemplos de cada uno de los subconjuntos mencionados anteriormente son 6.371.131, 796.471 y 796.471, que corresponde respectivamente al 80\%, 10\% y 10\% del número de ejemplos totales del conjunto de datos.

En la figura \ref{fig:kelm\_ex} se muestran dos ejemplos aleatorios del conjunto de datos KELM. Como se puede apreciar, cada ejemplo está formado por dos partes: la oración (\textit{sentence}), escrita en lenguaje natural; y la lista de tripletas (\textit{triple}), formada por las tripletas alineadas al texto de la oración separadas por una coma. Se muestran dos ejemplos diferentes ya que en el proceso de análisis de los datos se encontró que la lista de tripletas puede adoptar diversas estructuras. La primera forma, correspondiente al primer ejemplo mostrado (figura~\ref{fig:kelm\_ex1}), es representada de la manera (entidad\\_sujeto - nombre\\_relacion1 - valor\\_relacion1), (nombre\\_relacion2 - valor\\_relacion2) suponiendo que todas las tripletas se refieren a la entidad sujeto precisada en la primera de la lista. Otra estructura se muestra en la figura \ref{fig:kelm\_ex2} en la que, debido a la diferencia de la entidad sujeto en las tripletas representadas, cada una de las tripletas es independiente de la otra adquiriendo la forma (entidad\\_sujeto1 - nombre\\_relacion1 - valor\\_relacion1), (entidad\\_sujeto2 - nombre\\_relacion2 - valor\\_relacion2)... Cabe destacar el formato adoptado en el conjunto de datos original no establece ninguna separación entre los elementos perteneciente a una misma tripleta. Debido a lo confuso que puede ser su comprensión se ha añadido a cada una de ellas una separación entre los elementos de la forma (entidad\\_sujeto - nombre\\_relacion - valor\\_relacion) que no se encuentra en los datos originales.

Como se puede comprobar en los ejemplos presentados, esta base de datos no solo contiene información biográfica de personas, sino que también consta de conjuntos de ejemplos sobre otros temas. Es por esto por lo que puede resultar adecuada para el sistema que se pretende construir ya que podrá representar cualquier tipo de información que se desee sin limitarse al formato biográfico de Wiki2bio.

Con respecto al procesado de los datos, hay que destacar la limpieza de los datos originales. Únicamente hizo falta cambiar la codificación de los datos ya que se encontraban codificados en Windows-1252 por lo que algunos caracteres eran representados en un formato no apropiado para realizar el entrenamiento.

\subsection{Selección preliminar del modelo de lenguaje}

En este capítulo se realizará una aproximación a diferentes modelos de lenguaje basados en redes neuronales para la generación de texto. La motivación de este capítulo reside en la necesidad de comprender el funcionamiento de lo que podría ser considerado el núcleo de un sistema de generación: el modelo de lenguaje.

El cometido de este componente es la generación, a partir de una entrada dada, de una salida determinada acorde al propósito que se pretende conseguir con la construcción del sistema.

Se realizará un acercamiento a los modelos de lenguaje más empleados en la actualidad: GPT-2, BERT y T5. La selección de estos tres modelos no es una decisión arbitraria, sino que se escogieron aquellos modelos con mayor relevancia actualmente y que hubieran sido construidos con propósitos distintos de tal manera que se pueda discutir, según las características de cada uno de ellos, cuál/es serían los más apropiados para la tarea que se trata de realizar en este trabajo. Para poder considerar la adecuación del modelo se presentará el ajuste de cada uno de los modelos con respecto a un conjunto de datos de los presentados en la sección anterior \ref{sec:ConjuntosDeDatosYPreparacionParaElEntrenamiento}.

\subsubsection{Pre-entrenamiento del modelo}

Los modelos pre-entrenados han sido previamente capacitados para resolver una tarea mediante el aprendizaje de una serie de parámetros. Para realizar este pre-entrenamiento, generalmente, se utiliza aprendizaje auto-supervisado (\textit{Self-supervised Language Modelling}). Este proceso se realiza con los textos sin procesar, es decir, sin que los humanos etiqueten los datos de entrada de ninguna manera. La ventaja reside en que debido a la gran cantidad de datos que contienen los corpus utilizados en el entrenamiento, este procedimiento de etiquetado no se podría llevar a cabo de manera manual.

En los grandes modelos \textit{Transformers}, este proceso de entrenamiento es realizado en grandes computadores capaces de entrenar estos modelos sobre grandes conjuntos de datos. A continuación, vamos a centrarnos en los modelos de lenguaje que hemos considerado estudiar y extraeremos las principales claves de su pre-entrenamiento. También, se realizarán pruebas de funcionamiento de dichos modelos para comprobar sus resultados generales.

\paragraph{GPT-2 (\textit{Generative Pretrained Transformer})}

El pre-entrenamiento de GPT-2 se realizó sobre un gran corpus de texto en inglés siguiendo el entrenamiento auto-supervisado (\textit{self-supervised}). El objetivo de este modelo es la predicción de la palabra siguiente dada una secuencia de palabras u oración.

Para pre-entrenar el modelo se creó una base de datos, llamada \textit{WebText}, formada a partir de la extracción de todas las páginas web de los enlaces salientes en Reddit que recibieron una determinada puntuación mínima, para garantizar la calidad y significancia del enlace. Las páginas de Wikipedia relacionadas con estos enlaces se eliminaron. Es por esto por lo que GPT-2 no está entrenado bajo ningún texto de Wikipedia. El conjunto de datos resultante es un enorme corpus de 40GB de textos preparado para el entrenamiento de este modelo, GPT-2 \citep{radford2019language}.

Pese a todas las ventajas de este modelo, también tiene algunas limitaciones. Una de ellas, producida por el conjunto de datos seleccionado y por el propio algoritmo de entrenamiento aplicado, es la existencia de sesgos en la generación. Ya que la base de datos de partida está formada por una gran cantidad de contenido de internet sin filtrar, está influenciada por los sesgos representativos de los creadores de estos contenidos. En concreto, bajo la entrada ``El hombre blanco trabajaba como'', la salida generaba como posibles palabras de continuación a la oración ``periodista'' o ``conductor de autobús'', frente a la respuesta ``esclavo'' cuando se modificaba la raza de la persona de la oración de entrada.

Para comprender el proceso de generación de texto con este modelo se realizaron una serie de pruebas de funcionamiento básico. Para la obtención de todos los módulos necesario se utilizó la API \textit{Transformers} de la herramienta \textit{HuggingFace} que proporciona Python para la descarga y entrenamiento de modelos preentrenados. El modelo pre-entrenado utilizado es \textit{GPT2HeadModel}, una configuración del modelo GPT2 preparado para modelado de lenguaje. Por otra parte, el \textit{tokenizer} empleado es \textit{GPT2Tokenizer} basado en el algoritmo \textit{Byte-Pair-Encoding} visto anteriormente.

Como apunte, la tokenización, en el campo del Procesamiento de Lenguaje Natual, se refiere al proceso de transformación de una secuencia de palabras o símbolos a \textit{tokens} para que la máquina pueda comprender el lenguaje humano y contexto detrás de él. El proceso de tokenización en GPT-2, se basa en la obtención de subpalabras mediante un algoritmo de codificación de pares de bytes (\textit{Byte Pair Encoding} o \textit{BPE}).

El algoritmo BPE \citep{gage1994new} es un algoritmo de tokenización basado en subpalabras. Su objetivo principal es la resolución de los problemas de otros tipos de tecnologías basadas en palabras o en caracteres mediante un enfoque intermedio. De manera teórica, BPE es una forma simple de comprensión de datos en el que el par más común de bytes de datos consecutivos se reemplaza con un byte que no aparece en esos datos. Esta idea garantiza que las palabras más comunes se representen en el vocabulario como un solo \textit{token}, mientras que las palabras menos habituales se dividen en dos o más tokens de subpalabras.

Volviendo al tokenizador concreto utilizado: \textit{GPT2Tokenizer}; tiene en cuenta los espacios y por tanto asignará diferentes \textit{tokens} teniendo en cuenta también dicho carácter. Se puntualiza que con el parámetro $add\\_prefix\\_space=True$ se puede sortear este comportamiento, aunque no es lo recomendable ya que el modelo no está pre-entrenado de esa manera y podría derivar en una disminución del rendimiento. El resultado de aplicar a un texto inicial el \textit{GPT2Tokenizer} se puede comprobar en el código \ref{lst:tokenizer-gpt2}.

El resultado de aplicar la tokenización de \textit{GPT2Tokenizer} sobre una secuencia de palabras resulta en una lista denominada \textit{inputs\\_ids} que asigna un número identificador a cada uno de los \textit{tokens} encontrados en dicha secuencia. En el ejemplo \ref{lst:enc\_dec-gpt2} se puede comprobar el funcionamiento del proceso de codificar y decodificar. Dada una secuencia de entrada, en este caso \textit{'What is love?'}, se la pasamos al tokenizador y la codificamos. Este procedimiento devuelve los \textit{input\\_its} en forma de objeto \textit{tensor} y a continuación decodificamos. La secuencia original y la resultante después de aplicar ambos procesos son similares.

A continuación, se describe el proceso completo de generación de texto con GPT2 haciendo uso del tokenizador y del modelo. Para comenzar se codifica la secuencia de entrada al igual que se realizó en el ejemplo anterior. A continuación, se crea el modelo a partir del \textit{GPT2LMHeadModel} y se genera la salida con el método \textit{generate} (para generar el resultado final se emplearon los parámetros $max\\_lenght=50$ para establecer una longitud máxima, $num\\_beans=5$ y $no\\_repeat\\_ngram\\_size=2$). Para finalizar, se decodifica la salida del generador, ya que el resultado es un objeto \textit{tensor} similar al producido en la codificación. En el ejemplo~\ref{lst:model-gpt2} mostrado, se genera la continuación a la secuencia de entrada dada. El resultado es un texto coherente y cohesionado.

\paragraph{BERT (\textit{Bidirectional Encoder Representations from Transformers

BERT es un modelo pre-entrenado bajo dos grandes corpus de lengua inglesa con datos sin etiquetar. Por una parte, \textit{BookCorpus} \citep{Zhu\_2015\_ICCV} disponible en \textit{Hugging Face} \footnote{https://huggingface.co/datasets/bookcorpus} es un conocido corpus de texto a gran escala destinado especialmente al aprendizaje no supervisado de codificadores y decodificadores. \textit{BookCorpus}, está compuesto por 11038 libros (con alrededor de 74MB de oraciones y 1GB de palabras) de 16 diferentes subgéneros literarios. Otro de los corpus utilizados en el pre-entrenamiento del modelo es la Wikipedia inglesa, formada por textos de diversos temas y revisados por la comunidad de Wikipedia, lo que asegura una buena calidad y seguridad al entrenamiento del sistema.

Este modelo es capaz de realizar diversas tareas de procesamiento de lenguaje. Entre ellas destaca el Modelado de Lenguaje Enmascarado según se comentó en el apartado~\ref{sec:mlm}. Esta tarea de pre-entrenamiento del modelo se sustenta en un entrenamiento con una versión corrupta de los datos, generalmente enmascarando algunos tokens al azar y dejando que el modelo prediga el texto original. Este proceso garantiza la bidireccionalidad del modelo. Para llevar a cabo este procedimiento, antes de introducir una secuencia de palabras al modelo BERT, se reemplazan aproximadamente el 15\% de las palabras de dicha secuencia por el \textit{token} [MASK]. Seguidamente, el modelo trata de predecir el valor original de las palabras enmascaradas por el \textit{token} en función del contexto, proporcionado por el resto de las palabras no enmascaradas de la secuencia. Para poder realizar la predicción de la palabra enmascarada, se modifica ligeramente la arquitectura añadiendo una capa de clasificación a la salida del codificador. Después, se multiplican los vectores de salida por la matriz de incrustación, para transformar dicho vector en una matriz de la dimensión del vocabulario. Para terminar, se calcula la probabilidad de cada palabra en el vocabulario con la función \textit{softmax}. Esta nueva arquitectura se muestra en la figura ~\ref{fig:bert-MLM} de manera más detallada.

\figura{Bitmap/05ModeladoDeLenguaje/bert-MLM}{width=1\textwidth}{fig:bert-MLM}%

{Arquitectura BERT para MLM}

Otro de los procesos llevados a cabo durante el entrenamiento es la Predicción de la Siguiente Oración. mAproximadamente, el 50\% de estos pares de oraciones de entradas corresponden a dos pares seguidos de secuencias en el corpus original, mientras que el 50\% no son secuencias contiguas, sino que se realiza una elección aleatoria de cualquier otra oración del texto. Para que pueda realizar este procedimiento, se inserta el \textit{token} [CLS] al comienzo de la primera oración y el \textit{token} [SEP] para separar los pares de oraciones.

%Más cosas : Se agrega una incrustación posicional a cada token para indicar su posición en la secuencia. https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

Para predecir si la segunda oración está realmente contigua en el texto original a la primera, es necesario que toda la secuencia de entrada pase por el codificador del modelo. La salida del modelo del primer token [CLS] es un vector de dimensiones 2 x 1 y utilizando una capa de clasificación simple que se añade a la arquitectura, se calcula la probabilidad de la contigüidad de oraciones con la función \textit{softmax}.

Ambos procedimientos descritos se utilizan en conjunto durante el pre-entrenamiento del modelo con el objetivo de minimizar la función de pérdida combinada de ambas estrategias.

La entrada final al modelo se denomina \textit{input embeddings}. Este vector de entradas se constituye con la suma de los \textit{token embeddings}, \textit{segment embeddings} y \textit{position embeddings}. El primero de ellos se refiere a los \textit{tokens} resultantes de aplicar el tokenizador con los \textit{tokens} especiales [CLS] y [SEP]. El segundo \textit{embedding} indica la separación de oraciones. Por último, el \textit{position embedding} señala la posición de cada una de las palabras en la secuencia de entrada. Este concepto se muestra en la figura ~\ref{fig:bert-embeddings}.

A continuación, vamos a comprobar el funcionamiento del modelo BERT. Concretamente, se evaluará el proceso de predicción de la palabra más probable dada una secuencia con alguna palabra enmascarada con el \textit{token} [MASK].

Al igual que GPT-2, BERT necesita tokenizar los datos de entrada para poder manejarlos internamente. El tokenizador utilizado por este modelo de lenguaje es \textit{WordPiece} \citep{wordpiece} basado en subpalabras. Este algoritmo posee dos implementaciones: un enfoque ascendente de abajo hacia arriba y un enfoque descendente de arriba hacia abajo. El modelo BERT original utiliza el enfoque ascendente.

Este algoritmo no difiere demasiado del algoritmo BPE descrito anteriormente, ya que se trata de una versión modificada de dicho algoritmo. Sin embargo, \textit{WordPiece} trata de solucionar un problema común del BPE, limitado por la confusión de elección de un \textit{token} en el caso de las instancias que tiene más de una manera de ser codificadas. Debido a este problema, una misma entrada podría representarse mediante diferentes codificaciones pudiendo afectar a la precisión de las representaciones aprendidas.

Para realizar este proceso se utiliza el tokenizador \textit{BertTokenizer} que internamente

implementa el algoritmo \textit{WordPiece} descrito anteriormente. El primer ejemplo (código ~\ref{lst:tokenizer-bert}), muestra el resultado de tokenizar el texto de entrada. Se puede observar que la palabra ``thunderous'' no se encuentra en el vocabulario del tokenizador y por tanto la descompone en dos \textit{tokens}: ``thunder'' y ``\#\#ous''. Para indicar que estos \textit{tokens} no pertenecen a palabras separadas utiliza la doble almohadilla (\textit{\#\#}) como prefijo en el segundo \textit{token}.

A continuación, se muestra el proceso de predicción de una palabra enmascarada en una secuencia utilizada como entrada al modelo (código ~\ref{lst:model-bert}). El modelo utilizado es \textit{BertForMaskeLM}, una versión especial de BERT para la realización exclusiva de esta tarea. El proceso seguido para la predicción es muy sencillo: primero se obtienen los \textit{input\\_ids} de los \textit{tokens} que conforman la entrada y se codifican; a continuación, se obtiene el índice de las palabras enmascaradas (en el ejemplo mostrado hay un solo \textit{token} [MASK]); una vez obtenida la salida del modelo dada la secuencia de entrada, se aplica la función \textit{softmask} y finalmente se filtran las cinco palabras con mayor probabilidad. El resultado es una oración coherente mediante la generación de una palabra acorde con su entorno.

\paragraph{T5 (\textit{Text-to-Text Transfer Transformer})}

Para entrenar el modelo T5 se creó un enorme corpus de datos llamado \textit{``Colossal Clean Crawled Corpus''} (\textit{C4}). Este conjunto de datos contiene 750GB de información en lengua inglesa extraída mediante \textit{web scrappping} de la web. Ya que el corpus original son 20TB de datos no revisados y podían incluir lenguaje ofensivo, código fuente, textos en otros idiomas... en resumen, texto que no interesa para el entrenamiento, se siguieron una serie de pautas muy precisas para eliminar toda esta información, resultando un corpus limpio libre de todo dato no necesario.

Para realizar el preentrenamiento, se tiene en cuenta cada una de las tareas para las que se va a crear el modelo. Ya que se soporta en un enfoque de tipo \textit{Text-to-Text}, la entrada para cada una de las acciones a realizar será un texto, al igual que su salida. Para realizar tareas de traducción, sus autores precisan que la entrada al modelo fuese ``\textit{translate English to German: That is

good.}'' en el caso de querer traducir del idioma inglés al alemán ``\textit{That is

good}''. La salida del sistema sería ``\textit{Das ist gut.}''. En el caso de generación de resúmenes, la entrada estaría constituida por el texto a resumir seguida del texto ``\textit{TL;DR}'' (abreviación de \textit{too long, didn't read}). De esta manera se genera un resumen de un texto via decodificación autorregresiva.

Al igual que BERT, utiliza una entrada tokenizada como entrada al modelo. En el caso de T5, modifica el algoritmo de tokenización que utiliza el anterior modelo, basándose ahora en el algoritmo \textit{SentencePiece} que opera sobre regulación de subpalabras. Este algoritmo de tokenización, implementado en C++ es, increíblemente rápido, lo que resulta en un entrenamiento y generación muchísimo más veloz en comparación con los tokenizadores utilizados en GPT-2 (BPE) o BERT (WordPiece). Otra de las ventajas de este tokenizador es que se utiliza directamente sobre los datos sin la necesidad de almacenar los datos tokenizados en discos, por lo que utiliza menos memoria en el proceso. Por otra parte, es agnóstico respecto a los espacios en blanco, confiriendo a idiomas que en ocasiones no hacen uso de ellos, como el chino o japones, la misma facilidad de tokenización que a cualquier otro lenguaje. En general, se basa en la idea de que la codificación de pares de bytes no es óptima para el entrenamiento previo del modelo de lenguaje \citep{bostrom-durrett-2020-byte}.

%https://yukyunglee-github-io.translate.goog/Transformer-to-T5-4/?\_x\_tr\_sl=auto&\_x\_tr\_tl=es&\_x\_tr\_hl=es&\_x\_tr\_pto=wapp

Para pre-entrenar el modelo se adoptó el método de enmascaramiento MLM. Sin embargo, la diferencia con el método existente utilizado en BERT reside en que los \textit{tokens} consecutivos se reemplazan con una máscara sin enmascarar un \textit{token} aleatorio. Específicamente, si antes a cada una de las palabras enmascarados se les sustituía con el \textit{token} [MASK], este método los sustituye por <X>, <Y>, <Z>... y así sucesivamente hasta enmascarar todas las palabras introducidas. Estos \textit{tokens} se denominan \textit{tokens} centinela y tienen un tratamiento especial.

\subsubsection{\textit{Supervised fine-tuning}}

%https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language\_understanding\_paper.pdf

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%En este capítulo se presentan soluciones para cubrir los requisitos de generación del sistema.

%Por una parte, se trata la generación de textos que representen textualmente información bibliográfica de una persona.

La aparición de los modelos \textit{Transformers} como evolución de los modelos basados en redes neuronales recurrentes para la generación de lenguaje natural, supuso un cambio de paradigma en el modelado de lenguaje debido a la introducción de mecanismos de pre-entrenamiento y ajuste o \textit{finetune}. Después de un pre-entrenamiento sin supervisión del modelo bajo un gran conjunto de datos, dicho modelo puede ajustarse de manera mucho más rápida acorde a la tarea que se pretende lograr, utilizando para ello un conjunto de datos mucho más reducido y realizándose esta vez un entrenamiento supervisado.

Dependiendo de la tarea que se pretenda conseguir con el ajuste del modelo, se debe realizar una correcta elección del conjunto de datos sobre el que se va a entrenar el modelo. La elección de la base de datos no es trivial, sino que obedece a las necesidades establecidas: en nuestro caso, generación de texto biográfico a partir de unos datos especificados como entrada.

En los apartados siguientes, se mostrarán los resultados del ajuste realizado a diferentes modelos de lenguaje bajo los conjuntos de datos presentados en el apartado \ref{sec:ConjuntosDeDatosYPreparacionParaElEntrenamiento}.

\paragraph{Wiki2bio}\hfill

Para lograr un sistema de generación bibliográfica con esta \textit{dataset}, se escogió el modelo de lenguaje GPT-2 ya que el objetivo ideal de esta dataset es generar textos fluidos a partir de unos datos tabulares.

El proceso de ajuste de un modelo de lenguaje consta de una serie de pasos. En primer lugar, se debe escoger la versión adecuada del modelo de lenguaje seleccionado. En este caso, ya que se trata de una prueba se empleará la versión menos pesada (\textit{gpt2}) del modelo. Una vez seleccionada la versión se deben establecer los hiperparámetros que utilizará el modelo internamente para controlar el proceso de entrenamiento. La tasa de aprendizaje (\textit{learning rate}), el número de iteraciones de entrenamiento y el tamaño del lote (\textit{batch size}) son algunos ejemplos de hiperparámetros comunes. Los valores escogidos para estas variables tienen un impacto significativo en los parámetros aprendidos y, consecuentemente, en el rendimiento del modelo.

A continuación, se creó una clase denominada \textit{MyDataset} que agrupa una serie de funcionalidades básicas para el manejo del conjunto de datos utilizado. Esta clase hereda de la clase abstracta \textit{Dataset} de la librería \textit{torch} utilizada para representar grandes corpus. Según se define en su documentación, las clases que hereden de \textit{Dataset} deben sobreescribir los métodos \textit{\\_\\_getitem\\_\\_(index)} que devuelve el elemento en la posición \textit{index} del conjunto de datos y \textit{\\_\\_len\\_\\_()} que retorna el tamaño de dicho conjunto. También se incluyeron métodos de conversión de la información procedente de \textit{wiki2bio} en forma de diccionario con pares clave-valor a un formato comprensible por el modelo de lenguaje, en forma de texto.

Una vez realizados estos procedimientos básicos de gestión, se puede continuar el proceso de ajuste escogiendo un tokenizador adecuado. En este caso se utilizó \textit{GPT2-Tokenizer}, descrito anteriormente.

Para finalizar, se establece el modelo en modo de entrenamiento y se le asigna como datos de entrenamiento la porción de datos de \textit{MyDataset} establecida para ello. Para realizar esta prueba se realizará una selección de un subconjunto de los datos del corpus, esta decisión viene motivada por los límites de Google Colab, si utilizamos una gran cantidad de datos la memoria de la GPU no lo soportaría y se provocaría una interrupción durante el entrenamiento; también se podrían modificar algunos parámetros que permitirían entrenar una mayor cantidad de datos, pero aumentando considerablemente el tiempo de ejecución. En nuestro caso seleccionamos 10000 ejemplos de la base de datos original, de los cuales 8000 se utilizarán para realizar el entrenamiento del modelo y 2000 para la validación

El funcionamiento interno del modelo en este modo de \textit{finetune} consiste en para cada uno de los ejemplos establecidos como entrada, el modelo va aprendiendo a través de probabilidades cuales pueden ser las palabras siguientes a una secuencia. De esta manera se va ajustando el modelo a la tarea que se pretende conseguir, modificando los valores originales obtenidos después del pre-entrenamiento original.

Una vez realizado el entrenamiento que tomo en torno a dos horas, se puede comprobar con un ejemplo sencillo los resultados obtenidos. En la figura~\ref{fig:ex\_gpt2\_wiki2bio}, se muestra un ejemplo de resultado obtenido después de haber realizado el entrenamiento de ajuste. Como se puede apreciar algunos de los datos de salida son correctos y corresponden a los datos introducidos en la entrada. Sin embargo, se muestra un sobreajuste del modelo sobre los datos de entrenamiento ya que trata de generar un texto de longitud similar a los que han sido entrenados sin tener en cuenta el número de datos introducidos como entrada.

Por otra parte, se puede observar que este entrenamiento cae en alucinaciones y degeneraciones (conceptos introducidos en la sección \label{sec:efectos\_alu\_deg}) producidas en el momento en el que no sabe con qué información rellenar el texto. Estas alucinaciones se deben a que cuando se construyó esta dataset, los autores tomaron el cuadro de información de Wikipedia como fuente y la primera oración de la página de Wikipedia como referencia de verdad básica de texto de destino. Sin embargo, la primera oración del artículo de Wikipedia no es necesariamente equivalente al cuadro de información en términos de la información que contiene. De hecho \cite{dhingra2019handling}, señala que el 62 por ciento de las primeras frases de \textit{Wiki2bio} tienen información adicional no indicada en el infobox correspondiente. Este desajuste entre el origen y destino en los conjuntos de datos puede hacer que los modelos entrenados alucinen como ocurre en este caso.

Para evitar en gran medida estas incorrecciones en la generación se puede limitar el número de palabras generadas dando como resultado la figura~\ref{fig:ex2\_gpt2\_wiki2bio} que muestra un texto mucho más natural, aunque todavía mostrando algunas alucinaciones. Sin embargo, tiene como desventaja la limitación en la longitud del texto a generar y la necesidad de aproximación de dicha longitud dependiendo de la información introducida.

\paragraph{KEML}\hfill

KELM es un gran corpus de datos que trata de generar texto a partir de un grafo de conocimiento representado en forma de tripletas semánticas. Esta base de datos no solo contiene información en el formato propio de una biografía, sino que también contiene conjuntos de oraciones sobre otros temas. Es por esto por lo que puede resultar adecuada para el sistema que se pretende construir ya que podrá representar cualquier tipo de información que se desee sin limitarse al formato biográfico.

En este caso vamos a comprobar su funcionamiento bajo el modelo T5 descrito en el punto \label{sec:mtl}. El proceso de ajuste de este modelo es similar al utilizado para ajustar wiki2bio sobre GPT-2. Una vez realizado todo el proceso de limpieza y preprocesado de los datos descargamos un modelo básico del \textit{Transformer} T5 denominado \textit{T5-small} para a continuación realizar un entrenamiento con supervisión bajo el conjunto de datos escogido.

Para poder realizar el entrenamiento hace falta realizar una correcta selección de los hiperparámetros que se van a utilizar. El proceso de encontrar los mejores valores de hiperparámetros, que permite que el modelo descubra el mejor conjunto de parámetros para realizar una tarea determinada, recibe el nombre de \textit{optimización}. Para la optimización utilizaremos uno de los módulos disponibles a través de \textit{Hugging Face} conocido como AdaFactor, un ``método de https://www.overleaf.com/project/625fb2f8a25d988aade6b7daoptimización estocástica basado en Adam que reduce el uso de la memoria al tiempo que conserva los beneficios empíricos de la adaptabilidad'' \citep{shazeer2018adafactor}. Este método de optimización recibe una configuración inicial que hemos seleccionado según las recomendaciones de {Hugging Face} como se muestra en el código \ref{lst:adafactor}.

La entrada al modelo para realizar el entrenamiento está formada por ejemplos compuestos cada uno por su lista de tripletas separadas cada una de estas listas por una coma, un token separador \textit{</s>} y la oración de destino. Así, cada uno de los ejemplos resultantes del proceso de pre-procesado del conjunto de datos se va introduciendo en el modelo durante el pre-entrenamiento produciéndose continuamente un ajuste de sus parámetros internos. Cabe decir que se escogieron 30000 ejemplos para realizar el entrenamiento de prueba, aunque podrían haberse escogido más según se comprobó posteriormente ya que no nos acercamos al límite de uso de memoria de la GPU. Esto se debe a que T5 es un modelo que recorta bastante en el espacio requerido para el entrenamiento.

Ejemplos de los resultados obtenidos después de generar para diferentes tipos de datos las oraciones de salidas a través del modelo se encuentran en la figura \ref{fig:kelm\_result}. Se han seleccionado dos ejemplos representativos para estudiar la adecuabilidad de utilización de esta base de datos. El primer ejemplo muestra una correcta generacion de los datos proporcionados como entrada, al igual que el segundo de los ejemplos. Sin embargo, se puede apreciar una gran falta de representación de los datos de entrada ya que en ambos ejemplos falta información dada en la entrada que no se muestra en la salida: el año de nacimiento en el primer ejemplo y este mismo año y la profesión en el segundo.

WebNLG es una base de datos con una estructura parecida a la anterior. Por una parte, proporcionaba una lista de tripletas, esta vez separando cada uno de los elementos de dicha tripleta, al contrario que hacía KELM. La ventaja de esta base de datos puede residir precisamente en esta separación ya que proporcionará al modelo un mayor conocimiento de la estructura de datos que se trata de pasar como entrada. De esta manera el modelo aprendería los ejemplos con más precisión.

Para comprobar el funcionamiento de esta base de datos, se utilizaron dos modelos de los mencionados anteriormente: BERT y T5.

La elección de BERT como modelo de ajuste se realizó como alternativa a los modelos principalmente de generación casual que fueron entrenados en el resto de las pruebas. Ya que BERT es un modelo de lenguaje enmascarado utilizado especialmente para aprender una buena representación de los datos de entrada, pensamos que sería buena idea establecer otro planteamiento a la solución de modelado de lenguaje.

Para realizar el \textit{finetuning} de este modelo, una vez procesados todos los datos del conjunto WebNLG, como se describió en el apartado \label{sec:webnlg}, comenzamos descargando el modelo junto con el tokenizador. En nuestro caso hicimos uso, para ambos elementos, de la versión más básica: ``bert-base-uncased''. A continuación, transformamos la base de datos en un objeto de tipo \textit{dataset} de \textit{Hugging Face} y manipulamos este objeto para componer la entrada al modelo. Cada uno de los ejemplos que componen la entrada está formado por la unión de la columna \textit{input\\_text} y \textit{target\\_text}, junto con los tokens correspondientes de comienzo de texto, separadores y \textit{padding}:

A continuación, se extrajeron los identificadores de tokens a través del tokenizador. Un conjunto de 10000 ejemplos de esta base de datos tokenizada conformará la entrada final del modelo. La elección de este número de ejemplos viene motivada por el mismo principio de GPT-2, su alto coste de memoria.

Para terminar, se establecieron ciertos hiperparámetros básicos para el entrenamiento y se ajustó el modelo frente a los datos de entrada pasándole como parámetro un objeto de tipo \textit{DataCollator} que enmascarará un porcentaje (15\%) de los tokens de entrada para que el modelo aprenda su tarea de MLM.

Debido al mal funcionamiento del entrenamiento sobre este modelo de tipo lenguaje enmascarado, recurrimos a entrenar de nuevo los datos de Webnlg sobre el modelo t5 ya que anteriormente nos había proporcionado un buen resultado. El proceso de entrenamiento de Webnlg sobre este modelo es parecido al de KELM. Una vez preprocesados y descargados los datos y el modelo, se procedió a la composición de la entrada formada por los \textit{input\\_ids} devueltos por el tokenizador para cada uno de los ejemplos resultantes de la unión en una misma oración de las tripletas y el texto de destino, separados por un token de separación \textit{</s>}. Entrenando los datos en una sola iteración, el resultado fue el expuesto en la figura \ref{fig:ex\_t5\_webnlg} [TENGO AUN QUE HACER ESTA IMAGEN].

\subsection{Resolución final}

Habiendo comprobado los resultados de diferentes combinaciones de bases de datos y modelos se ha llegado a la conclusión de que la mejor combinación resultante ha sido la del ajuste del modelo T5 bajo el corpus WebNLG.

El problema del conjunto Wiki2bio era que debido al tamaño de sus ejemplos el modelo trata de generar textos de igual magnitud y si los datos proporcionados como entrada no son suficientes, caía en alucinaciones y degeneraciones imposibles de tratar. Por otra parte, el corpus KELM conseguía la resolución de los problemas anteriores, aunque introducía algunos nuevos como la falta de representación de los datos de entrada en bastantes casos. Esto podría llevar a que datos importantes necesarios de ser representados en la salida no se tuvieran en cuenta. Posteriormente, el modelo BERT de enmascaramiento de lenguaje se descartó al comprobar que el pre-conocimiento de la longitud de los datos de salida era importante para poder generar las oraciones y este pre-conocimiento en algunas ocasiones podría ser difícil de determinar. Finalmente, el ajuste del modelo T5 para la generación de lenguaje fluido resultó en muy buenos resultados sin ninguno de los efectos perjudiciales que en ocasiones las redes neuronales pueden llegar a provocar.