UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE CUENCA

CARRERA DE INGENIERÍA ELECTRÓNICA

DISEÑO DE UN SISTEMA PROTOTIPO DE DIÁLOGO PERSONA-MÁQUINA BASADO EN LA ARQUITECTURA BERT

Trabajo de titulación previo a la obtención
del título de Ingeniero Electrónico

AUTORES: JUAN FERNANDO AUQUILLA VICUÑA
JUAN CARLOS MORA ALVAREZ

TUTOR: ING. CHRISTIAN RAÚL SALAMEA PALACIOS, Ph.D.

Cuenca - Ecuador
2022

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Nosotros, Juan Fernando Auquilla Vicuña con documento de identificación N° 0105264907 y Juan Carlos Mora Alvarez con documento de identificación N° 0105736920; manifestamos que:

Somos los autores y responsables del presente trabajo; y, autorizamos a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Cuenca, 08 de abril del 2022

Atentamente,

Juan Fernando Auquilla Vicuña
0105264907

Juan Carlos Mora Alvarez
0105736920

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Nosotros, Juan Fernando Auquilla Vicuña con documento de identificación Nº 0105264907 y Juan Carlos Mora Alvarez con documento de identificación Nº 0105736920, expresamos nuestra voluntad y por medio del presente documento cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autores del Proyecto Técnico: "Diseño de un sistema prototipo de diálogo persona-máquina basado en la arquitectura BERT", el cual ha sido desarrollado para optar por el título de: Ingeniero Electrónico, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribimos este documento en el momento que hacemos la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Cuenca, 08 de abril del 2022

Atentamente,

Juan Fernando Auquilla Vicuña
0105264907

Juan Carlos Mora Alvarez
0105736920

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, Christian Raul Salamea Palacios con documento de identificación N° 0102537180, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: DISEÑO DE UN SISTEMA PROTOTIPO DE DIÁLOGO PERSONA-MÁQUINA BASADO EN LA ARQUITECTURA BERT**,** realizado por Juan Fernando Auquilla Vicuña con documento de identificación N° 0105264907 y por Juan Carlos Mora Alvarez con documento de identificación N° 0105736920, obteniendo como resultado final el trabajo de titulación bajo la opción Proyecto Técnico que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Cuenca, 08 de abril del 2022

Atentamente,

_____

Ing. Christian Raúl Salamea Palacios, Ph.D.

0102537180

AGRADECIMIENTO

Quiero agradecer a mis padres y a mi familia por haberme brindado su apoyo incondicional en cada etapa de mi carrera universitaria y por la confianza que me ofrecieron en cada una de mis decisiones. Al Dr. Christian Salamea PhD, por la orientación y motivación brindada para el desarrollo de este proyecto.

Juan Fernando Auquilla

AGRADECIMIENTO

Quiero expresar mi gratitud a Dios por las bendiciones diarias de la vida, por jamás abandonarme y permitirme culminar esta etapa.

Gracias a mi padre Milton y mi madre Ruth (+), que, aunque no esté presente en vida, ha sido una gran fortaleza para seguir adelante; por el esfuerzo, paciencia y apoyo de parte de ustedes para culminar mi carrera universitaria.

Agradezco a mi hermano Xavier, por el apoyo constante para no decaer cuando todo parecía complicado e imposible, por ayudarme siempre en todo sentido sin importar la situación, por ser una excelente persona y mi gran confidente de toda la vida.

Al Dr. Christian Salamea PhD; gracias por la dirección, consejos y orientación brindada a lo largo de este trabajo de titulación.

A mi compañero Juan Fernando por la paciencia, el compañerismo y amistad a lo largo de estos años.

Infinitas gracias a todos mis familiares, amigos y profesores por el apoyo y ayuda a lo largo de todo este tiempo para cumplir este gran objetivo, sin su presencia nada de esto sería posible.

Mi cariño y aprecio para cada uno de ustedes.

Juan Carlos Mora Alvarez

DEDICATORIA

El presente trabajo de titulación se lo quiero dedicar a mis padres Teodoro y Lourdes, por todo el apoyo, paciencia y confianza depositadas a lo largo de mi formación, y por estar ahí siempre de una manera incondicional. A mi Tía la Dra. Martha Auquilla, por todo el apoyo y la motivación que recibí de su parte, al haberme brindado su confianza, permitiéndome llegar a culminar esta etapa en mi vida.

Juan Fernando Auquilla

DEDICATORIA

Este trabajo de titulación se lo dedico a mis padres Ruth y Milton, por el apoyo, paciencia, sacrificio, esfuerzo y el aporte en mi formación humana y profesional; gracias de corazón.

A mi hermano Xavier, por su ayuda, consejos, apoyo incondicional y jamás abandonarme.

Muchos de mis logros se los debo a ustedes.

A las personas que nos compartieron sus conocimientos y facilitaron el desarrollo de este trabajo.

Se cumple una meta más, gracias a todos ustedes.

Juan Carlos Mora Alvarez

INDICE GENERAL

INDICE DE FIGURAS

GLOSARIO

PLN: Procesamiento Natural de Lenguaje - Natural Language Processing.

BERT: Representación Bidireccional Codificada de Transformers - Bidirectional Encoder Representations from Transformers.

LSTM: Memoria a Corto Plazo - Long Short-Term Memory.

SQuAD: Conjunto de datos de respuesta a preguntas de Stanford - Stanford Question Answering Dataset.

TPU: Unidad de Procesamiento Tensorial – Tensor Processing Unit.

HCI: Interacción Persona Máquina – Human Computer Interaction.

IR: Recuperación de información.

EI: Extracción de información.

CAPÍTULO 1

1. Introducción

1.1. Motivación

Hoy en día, resulta común que las personas se relacionen con máquinas de forma consciente o inconsciente, a través de diversos medios. Por tal motivo, es necesario crear mecanismos que establezca un canal de comunicación humano/máquina, lo más intuitivo y natural posible, logrando una interacción práctica, sencilla y eficaz. En consecuencia, se están diseñando modelos de diálogo que representan formalmente procesos de diálogo recurriendo a un procesamiento de lenguaje natural y tecnologías semánticas.

Partiendo de lo anterior, en este trabajo el objetivo es diseñar e implementar una interfaz de comunicación que refleje un modelo de diálogo persona-máquina basado en la arquitectura BERT. Dicho modelo, se encuentra pre-entrenado para describir relaciones bidireccionales de palabras, a partir de texto sin etiquetar que permite obtener resultados de mejor precisión en el procesamiento de lenguaje natural. Por otra parte, con respecto a los sets de datos, se busca brindar al modelo información de diferentes características por lo que se utilizará un set de datos propio y otro con licencia libre.

1.2. Planteamiento del trabajo

Recientemente, gracias a la disminución del costo de los dispositivos de almacenamiento y el aumento del ancho de banda de las conexiones a internet, se está experimentando un auge de la computación en la nube. Esto facilita en gran medida compartir información y contenidos entre usuarios para crear y modificar información de manera colaborativa [6].

Los sistemas de diálogo persona-máquina estudian la forma en que las personas realizan el diseño, la implementación y utilizan sistemas informáticos participativos y cómo las diferentes máquinas afectan a las personas, las organizaciones y en general la sociedad. Esto abarca, no sólo la facilidad de uso, sino también nuevas técnicas de interacción para apoyar las tareas del usuario, proporcionar un mejor acceso a la información y crear formas de comunicación más poderosas.

A lo largo del tiempo, dentro del contexto del procesamiento natural del lenguaje (PLN), se empleó el Test de Turing como criterio para medir la inteligencia de una

máquina. Luego se desarrolló el sistema de traducción automática estadística sustentado siempre en un aprendizaje basado en reglas. Seguido, se incorporaron algoritmos, como los árboles de decisión y sistemas de sentencias "si-entonces" análogas a las pautas escritas a mano [1]. Con el desarrollo de las redes neuronales artificiales, se facilitó la resolución de problemas complejos, sobre todo en procesamiento de imágenes y de reconocimiento visual [2]. Lo siguiente fue la rama del procesamiento del lenguaje natural, donde se utilizó la capacidad de extracción de información producida dentro de las redes neuronales para dar paso al concepto de Words embeddings (grupo de lenguajes de modelado y procesos de aprendizaje en PLN), en la cual las palabras o frases son representadas como vectores para ejecutar análisis semánticos y sintácticos [3]. Hoy en día, los Words embeddings permiten convertir un corpus (de palabras) a un conjunto de entrenamiento numérico que, a su vez, puede alimentar modelos como LSTM para aplicaciones como la comprensión o la predicción de texto [4]. El último peldaño de desarrollo va de la mano con el uso de modelos pre-entrenados (transfer learning) que analizan el contexto de palabras, entre los que destaca BERT, del cual existen versiones adaptas para diferentes idiomas [5].

En la actualidad, existen modelos pre-entrenados para la comprensión de texto, por lo que en este trabajo se pretende aprovechar todo el camino recorrido en torno a la tecnología de los sistemas de interacción persona-máquina y generar interfaces de usuario amigables.

1.3. Estructura de capítulos

1. Introducción
   1.1. Motivación
   1.2. Planteamiento del trabajo
   1.3. Estructura de capítulos
2. Contexto del trabajo
   2.1. Procesamiento del lenguaje natural
   2.2. Sistemas de interacción humano-maquina
   2.3. Estado del arte
3. Fundamento Teórico
   3.1. Tokenización.
   3.2. Representación vectorial de palabras (Embedding).
   3.3. Word2Vec vs BERT Embedding.

CAPÍTULO 2

2. Contexto del trabajo

En el presente trabajo se utilizará un sistema de diálogo persona - máquina para la interacción del usuario con la máquina. Se sabe que el modelo de sistema es una representación formal del proceso de diálogo entre usuario y máquina que se aplicará para el procesamiento de lenguaje natural y las tecnologías semánticas.

A partir de esto se va a diseñar e implementar una interfaz que permita aplicar el modelo de diálogo persona - máquina basada en la arquitectura BERT, el cual se encuentra entrenado previamente para presentaciones bidireccionales profundas, a partir de texto sin etiquetar mediante el condicionamiento conjunto del contexto izquierdo y derecho de todas las capas, con esto se obtendrá resultados de mejor precisión en el procesamiento de lenguaje natural.

2.1. Procesamiento del lenguaje natural

El procesamiento del lenguaje natural (PLN) se refiere a la capacidad de una máquina para procesar información a partir del lenguaje, por lo que consiste en una serie de mecanismos, técnicas y modelos que procesan texto para proporcionar conocimiento [7]. De esta manera, es común encontrar esquemas que realizan trabajos concernientes con el lenguaje o desarrollan guías que ayudan a entender los mecanismos de las personas [9]. Las áreas en las que el PLN se está desarrollando abren un gran abanico de investigación, pudiendo destacar las siguientes: recuperación y extracción de información, traducción automática, minería de datos, generación de resúmenes automáticos, análisis de sentimientos, sistemas de búsquedas de respuestas, entre otras [8].

2.1.1. Recuperación y extracción de información.

La recuperación de información (IR) es el procedimiento de descubrir en un gran repositorio de información, datos de naturaleza estructurada, no estructurada o semiestructurada, que satisfaga una necesidad de información. Cuando se refiere a una naturaleza estructurada, suele relacionarse a documentos y cuando se refiere a una naturaleza no estructurada son textos planos (archivos de texto utilizando la extensión

.txt), en cambio un material semiestructurado se refiere a páginas web mirándolo como un referente. Por lo tanto, la información no estructurada, al no apoyarse en una representación clara, no se encuentran preparados para procesarse. Por su parte, la información semiestructurada se encuentra en documentaciones de esquema estructurado, muy similares a las que se descubren en las bases de datos.

Por lo tanto, la recuperación de la información implica el cambio del texto en representaciones apropiadas según modelos específicos que cumplen los objetivos de búsqueda.

A su vez el proceso de extracción de información (EI), radica en obtener los segmentos del texto que corresponden para convertirlas en un formato de base de datos, es decir, un formato estructurado [8].

2.1.2. Minería de datos.

La minería de datos tiene como objetivo sacar información de un grupo de datos y convertirla en una forma entendible para su siguiente utilización.

La minería de datos suministra instrumentos para revelar patrones y relaciones ocultas en datos estructurados. También supone que los datos ya están recopilados en un formato estructurado. Por ello, su pre-procesamiento radica en limpiar y normalizar los datos y generar numerosos vínculos entre las tablas de la base de datos [12].

2.1.3. Traducción automática.

La traducción automática toma un texto escrito en una lengua y lo traduce a otra, manteniendo el mismo significado. En general, el texto de la lengua de origen se transforma primero en una representación intermedia, luego, según la morfología de la lengua final, se ejecutan transformaciones en esta representación intermedia y, finalmente, se convierte en la lengua de destino [13].

2.1.4. Sistemas de búsquedas de respuestas.

Son sistemas planteados para responder a una consulta en lenguaje natural. Dichos sistemas se levantan sobre los motores de investigación y pretenden el material como fuente para revelar las respuestas. Hay tres pasos: el entrenamiento, fragmentación y sólo entonces se determina la clase de respuesta.

Para levantar un sistema de respuestas es necesario emplear metodologías de PLN como: recuperación de información, extracción de información con algoritmos de

reconocimiento de entidades y etiquetado, comparación de secuencias de caracteres, entre varias más [14].

2.1.5. Generación de resúmenes automáticos.

La generación de resúmenes automáticos se llega a definir a medida de documento y a medida de grupo de documentos. Estos consiguen tener una visión extractiva o abstractiva. El extractivo gravitan en una recopilación de términos, frases o párrafos reveladores que puntualizan el significado del argumento original. Por otro lado, los abstractivos reconoce de metodologías de parafraseo para originar los resúmenes [15].

2.1.6. Análisis de sentimientos.

El análisis de sentimientos en texto es extraer e identificar la información subjetiva. La manera sencilla de analizar los sentimientos es una categorización de sentimientos polarizada que logra determinar evaluaciones en el rango de -10 a 10 y que se fundamenta en el aprendizaje para valorar las emociones negativas y positivas en corpus de entrenamiento etiquetados. Las técnicas avanzadas reconocen el análisis gramatical y la descomposición de frases [12].

2.2. Sistemas de interacción humano - máquina

2.2.1. Sistemas persona – máquina

El término sistemas hombre-máquina se refiere a todas las condiciones en las que los seres humanos (tanto individuos como grupos) utilizan, controlan o supervisan herramientas, máquinas o sistemas tecnológicos. Se encuentran ejemplos de condiciones que involucran sistemas humano-máquina en casi todos los dominios tecnológicos y muchos dominios relacionados. En consecuencia, las aplicaciones de los sistemas hombre-máquina son multidisciplinarios. La investigación en sistemas hombre-máquina es fundamental, es decir, buscar principios genéricos relacionados con la interacción hombre-máquina que se apliquen a muchos dominios, y operacional, es decir, diseñar y evaluar aplicaciones para un dominio específico [16].

2.2.2. Sistemas de Diálogo persona - máquina

Los sistemas de diálogo son técnicas que tienen por finalidad construir un diálogo con las personas con una disposición relacionada. Es común que, los sistemas de diálogo permiten utilizar texto, voz, imágenes o gráficos, gestos o demás formas para relacionarse

entre un y otro sentido de comunicación (ingreso y respuesta). De esta manera, un sistema de diálogo entiende tanto las operaciones de reconocimiento como las de sinopsis, independientemente del contorno de comunicación que manejen en el diálogo [10].

El esquema clásico del sistema de diálogo tiene la siguiente estructura:

1. **Natural Language Understanding (NLU):** es un módulo de reconocimiento del lenguaje natural. La tarea principal de este módulo es identificar entidades, identificar el sujeto de la oración de entrada y preparar los datos detectados para su posterior procesamiento [17].
2. **Dialogue Manager (DM):** es el módulo que sigue al módulo de reconocimiento de lenguaje natural que tiene como objetivo coordinar el flujo de diálogo y comunicarse con otros subsistemas y componentes [18].
3. **Natural Language Generation (NLG):** recibe una especificación de un acto comunicativo del administrador de diálogo y genera una representación de texto correspondiente. Definir las dos funciones que debe realizar el módulo de generación de respuestas: planificación de contenidos y generación de lenguaje, pero reconociendo que la primera se puede atribuir al administrador de diálogos.

El módulo de reconocimiento de lenguaje natural es una parte muy importante del sistema de diálogo, ya que depende del modelo al que se accederá a los datos en el futuro y de cómo se generará la salida del sistema [19].

2.3. Estado del arte

En esta sección, se dará a conocer cómo se desarrollan los sistemas de interacción humano máquina (IHM) y la eficacia que tienen estos para interactuar con el usuario mediante un diálogo persona-máquina. También se podrá ver el desarrollo que se ha dado con el avance del tiempo y como estos sistemas se han ido adaptando a las necesidades del usuario mediante la aplicación de algoritmos para diferentes propósitos. El objetivo principal es describir la arquitectura de BERT enfocada en el desenvolvimiento de un sistema de diálogo, mostrando la diferencia y la eficiencia que tiene esta arquitectura al ser utilizada como un sistema de interacción humano-máquina (IHM).

Desde los años sesenta, el rápido crecimiento de los sistemas de información ha dado lugar a un amplio desarrollo de la investigación sobre la interacción persona-ordenador (HCI), cuyo objetivo es diseñar interfaces persona-ordenador que presenten propiedades ergonómicas como la amabilidad, la facilidad de uso, la transparencia, etc. Se abarcaron diversas situaciones de trabajo: trabajo clínico, programación informática, diseño, etc. Sin embargo, eran principalmente estáticas en el sentido de que el usuario controla totalmente el ordenador. Más recientemente, las organizaciones públicas y privadas se han lanzado a la empresa de gestionar sistemas cada vez más complejos y acoplados mediante la automatización. Las máquinas modernas no sólo procesan información, sino que actúan en situaciones dinámicas como lo hacían los humanos en el pasado, gestionando la bolsa, las plantas industriales, los aviones, etc. Estas situaciones dinámicas no están totalmente controladas y se ven afectadas por factores inciertos. Por ello, hay que mantener grados de libertad que permitan a los humanos y a la máquina adaptarse a contingencias imprevistas. Es necesario un enfoque de cooperación hombre-máquina (HMC) para hacer frente a los nuevos retos que introduce esta tendencia. A continuación, se describe la posible mejora de la HCI mediante la HMC, la necesidad de una nueva concepción de la asignación de roles entre humanos y máquinas, y los principales problemas encontrados en las nuevas formas de relación hombre-máquina.

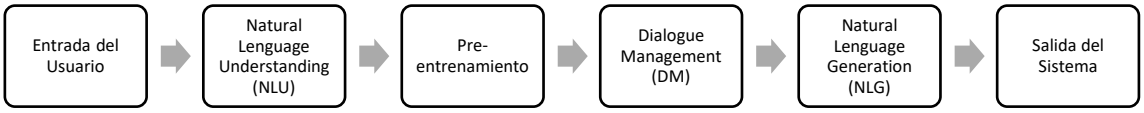

Figura 1.Estructura de un sistema de diálogo

En la figura 1, se tiene desglosado la estructura que conlleva un sistema de diálogo, donde se ve comprometido por diferentes bloques para obtener un sistema completo entre uno de ellos esta el pre-entrenamiento de los sistemas de diálogo para este se puede utilizar diferentes técnicas y obtener un resultado similar, en (Zaib, M., Sheng, Q., & Emma Zhang, W. 2020) nos habla de los modelos lingüísticos preformados que se han introducido recientemente y que tienen el potencial de resolver el problema de la escasez de datos y aportan ventajas considerables al generar incrustaciones de palabras contextualizadas (Zaib, M., Sheng, Q., & Emma Zhang, W. 2020), nos dice que con estos modelos lingüísticos se puede obtener relaciones jerárquicas, la dependencia a largo plazo y lo que conlleva a que se hable sobre cómo se puede aprovechar los puntos fuertes de estos modelos lingüísticos para diseñar agentes conversacionales más atractivos y elocuentes. De esta manera deliberan si estos modelos pre-entrenados son capaces de superar los retos propios de los sistemas de diálogo, y cómo se puede aprovechar su arquitectura para superar los mismos.

(E. Levin, R. Pieraccini and W. Eckert, 2000.) Se afirma que el problema del diseño de diálogos puede formalizarse como uno de optimización con una función objetivo que refleja diferentes magnitudes de diálogo llamativas para una aplicación determinada. También muestra que cualquier sistema de diálogo puede describirse formalmente como un procedimiento de sentencia secuencial en términos de su espacio de estados, conjunto de decisiones y estrategias (E. Levin, R. Pieraccini and W. Eckert, 2000). De la misma manera nos dice que un sistema de diálogo puede ser mapeado a un patrón estocástico conocido como proceso de decisión de Markov (MDP). En el marco del MDP existen diversos algoritmos basados en datos para encontrar la estrategia óptima basados en el aprendizaje por refuerzo. Para un uso eficaz de los datos de entrenamiento disponibles, y proponen una combinación de aprendizaje supervisado y de refuerzo (E. Levin, R. Pieraccini and W. Eckert, 2000).

Según las aplicaciones, los sistemas de diálogo pueden clasificarse en dos grupos: (1) sistemas orientados a tareas y (2) sistemas no orientados a tareas. Los sistemas orientados a tareas buscan como meta facilitar al individuo a realizar determinadas tareas. Los enfoques más aplicados a los sistemas orientados a las tareas consisten en buscar la réplica del diálogo como un canal. Los sistemas entienden primeramente el mensaje entregado por el ser humano, lo muestran como un estado interno, después agarran ciertas acciones según la política con referencia al estado del diálogo, y por último la acción se modifica en su forma superficial como lenguaje natural. Aun cuando el

lenguaje comprendido se trabaja mediante modelos estadísticos, casi la totalidad de los sistemas de diálogo empleados siguen utilizando características manuales o reglas elaboradas a mano para las representaciones del espacio de estados y acciones, la detección de intenciones y el llenado de espacios (Chen, H., Liu, X., Yin, D., & Tang, J. 2017).

Por lo tanto, al hablar de pre-entrenamiento y algoritmos para el proceso de un sistema de diálogo humano-máquina se muestra un nuevo modelo de representación del lenguaje llamado BERT, que significa Representaciones Codificadoras Bidireccionales de Transformadores. A diferencia de los modelos de representación lingüística recientes (Peters et al., 2018a; Rad- ford et al., 2018), BERT es una arquitectura que esta diseñada para entrenar previamente representaciones bidireccionales profundas a partir en texto sin etiquetar mediante la restricción conjunta de los contextos izquierdo y derecho en todas las capas. Por lo tanto, los modelos BERT previamente entrenados se pueden adaptar en una sola capa de salida adicional para generar modelos de vanguardia para una amplia gama de tareas, como la respuesta a preguntas y la inferencia lingüística, sin realizar cambios significativos en la arquitectura específica de la tarea.

BERT es conceptualmente sencillo y empíricamente potente. La puntuación GLUE aumentó al 80,5% (mejora absoluta 7,7%), la precisión MultiNLI aumentó al 86,7% (mejora absoluta ,4.6%), SQuAD v1.1 F1 (mejora absoluta 1,5) en la respuesta de la prueba a 93,2. Logro nuevos resultados avanzados en once tareas de procesamiento del lenguaje natural, incluyendo mejoras. Y al realizar con SQuAD v2.0 F1, 83, 1 (mejora absoluta de 5.1) siendo el porcentaje en la respuesta de prueba.

El pre-entrenamiento del modelo lingüístico ha demostrado ser eficaz ya que este a permitido el mejoramiento evidente de tareas que están basadas en el procesamiento del lenguaje natural (Dai y Le, 2015; Peters et al., 2018a; Radford et al., 2018; Howard y Ruder, 2018). Las mismas que están dentro del rango de nivel de oración, como la deducción del lenguaje natural (Bowman et al., 2015; Williams et al., 2018) y la paráfrasis (Dolan y Brockett, 2005), que tienen como objetivo predecir las conexiones entre las oraciones analizando la oración completa y predecir actividades a nivel de token, como reconocer entidades nombradas y responder preguntas, en los cuales requiere que los

modelos produzcan una salida detallada a nivel de token (Tjong Kim Sang y De Meulder, 2003; Rajpurkar et al., 2016).

Existen dos estrategias para aplicar las representaciones lingüísticas pre-entrenadas a las tareas de flujo descendente: la basada en características y la de ajuste fino. El enfoque basado en características, como ELMo (Peters et al., 2018a), maneja cierto tipo de arquitecturas que son especificas de la tarea con representaciones entrenadas previamente como funcionalidades adicionales. Haciendo mención del ajuste fino como un planteamiento, tal como el Generative Pre-trained Transformer (OpenAI GPT) (Radford et al., 2018), que simplemente ingresa los parámetros mínimos específicos para la actividad y ajusta todos los parámetros previamente entrenados para entrenar las siguientes actividades. Ambos enfoques utilizan un modelo de lenguaje unidireccional para aprender una expresión lingüística general y compartir una función objetiva en común con el previo aprendizaje.

En este sentido, se argumenta que las técnicas actuales restringen la potencia de las representaciones pre-entrenadas, especialmente en el caso de los enfoques de ajuste fino. La principal limitación es que los modelos lingüísticos estándar son unidireccionales, lo que limita la elección de las arquitecturas que pueden utilizarse durante el pre-entrenamiento. Por ejemplo, en OpenAI GPT, los autores utilizan una arquitectura de izquierda a derecha, en la que cada token solo puede tender a tokens anteriores en las capas de auto-atención del transformer (Vaswani et al., 2017). La limitación es que se aplican no son óptimas para dichas acciones que se encuentran en un nivel de frase, dando la posibilidad a ser perjudiciales cuando se utiliza en un planteamiento que se basa en el ajuste fino en actividades que se les clasificaría como en nivel de token, siendo estas el responder preguntas donde es de suma importancia poder tener un entendimiento visto desde ambas direcciones, para de esta manera obtener un contexto mucho más acertado.

Por lo tanto, al hablar de la arquitectura de BERT esta nos modera la limitación de una unidireccionalidad de la cual ya se hablo antes utilizando un objetivo de pre-entrenamiento de "modelo de lenguaje enmascarado" (MLM), inspirado en la tarea Cloze (Taylor, 1953). Cuando se habla de MLM se concluye que este enmascara aleatoriamente determinados tokens de la entrada, y el propósito es poder identificar la raíz del léxico de la palabra enmascarada basándose únicamente en el contexto.

A diferencia del pre-entrenamiento del modelo lingüístico de izquierda a derecha, el objetivo de MLM permite que la representación fusione el contexto izquierdo y el

derecho, lo que nos permite pre-entrenar un Transformador bidireccional profundo. Además del modelo de lenguaje enmascarado, también utilizamos una tarea de "predicción de la siguiente frase" que pre-entrena conjuntamente las representaciones de pares de texto.

Entonces se demuestra la importancia del pre-entrenamiento bidireccional de las representaciones lingüísticas. Al igual que Radford et al. (2018), que utiliza modelos lingüísticos unidireccionales para el pre-entrenamiento, BERT utiliza modelos lingüísticos enmascarados para permitir representaciones bidireccionales profundas pre-entrenadas. Esto también contrasta con Peters et al. (2018a), que utiliza una concatenación superficial de LSTMs entrenados independientemente de izquierda a derecha y de derecha a izquierda.

Dándose a conocer que las representaciones pre-entrenadas reducen la necesidad de muchas arquitecturas específicas de tareas fuertemente diseñadas. BERT es el primer modelo de representación basado en la sintonía fina que logra una utilidad de alto progreso en un amplio conjunto de actividades a nivel de frase y de token, superando a muchas arquitecturas de tareas específicas.

CAPÍTULO 3

3. Fundamento Teórico

Todo lo que el ser humano expresa, ya sea de forma verbal o escrita, contiene una gran cantidad de información. El tópico del que se habla, el tono de la voz e incluso la selección de las palabras añade algún tipo de dato que se interpreta para extraer valor, lo cual permite entender y predecir cualquier tipo de comportamiento [20]. Bajo este contexto, existe una característica particular: una persona puede generar miles de palabras en cada oración, añadiendo una alta complejidad de interpretación que va de la mano con la incorporación de modismos geográficos [21].

Desde un punto de vista computacional, esta característica particular hace que los datos generados por medio de conversaciones ya sean orales o escritas, carezcan de estructura y estén desordenadas, con lo cual son difíciles de manejar [22]. Sin embargo, gracias a los constantes avances tecnológicos en los campos del aprendizaje automático, específicamente en la inteligencia artificial, se está dando paso a técnicas revolucionarias de procesamiento del lenguaje que ayudan a entender el significado que existe detrás de cada idea u oración expresada por medio de un conjunto de palabras [23]. De este modo, es posible realizar, desde, un análisis de sentimientos hasta implementar modelos comunicacionales humano-máquina que tomen en consideración el contexto y las figuras retóricas.

Como puede notarse, las ideas expresadas reflejan los conceptos clave de lo que se conoce como procesamiento del lenguaje natural (PLN), hablando de un campo que se basa en la inteligencia artificial que otorga a las máquinas la habilidad de leer/escuchar, entender y extraer conocimiento a partir del lenguaje humano [24]. En la actualidad, el PLN está en auge gracias a las enormes mejoras en el acceso a los datos y al aumento de la potencia computacional, consintiendo el desarrollo de un sinnúmero de aplicaciones. Entre los ejemplos más significativos, se pueden mencionar las siguientes [25]:

- Reconocimiento y predicción de enfermedades a partir de las historias clínicas electrónicas y el propio discurso del paciente. Esta capacidad se está explorando en condiciones de salud que van desde las enfermedades cardiovasculares hasta la depresión y la esquizofrenia.

- Detección de opiniones sobre un servicio o producto mediante la identificación y extracción de información de redes sociales.
- Desarrollo de motores de búsqueda personalizados que cumplen funciones de asistentes virtuales para aprender del usuario y recordarle cualquier tipo de información cuando lo necesite.
- Filtrado de correo electrónico para detectar spam anticipando la llegada a la bandeja de entrada del usuario.
- Detección de noticias falsas a través del análisis del origen del texto, sesgando grupos que no son de confianza.
- Implementación de asistentes virtuales basados en el reconocimiento de la voz para responder a las peticiones del usuario.
- Sistemas de diálogo humano-máquina que buscan entender lo que una persona expresa a través del lenguaje para proporcionar una respuesta inteligente, dando paso a una nueva clase de asistente virtual.

Si bien el PLN representa un panorama prometedor en beneficio del desarrollo de la humanidad, existen ciertas dificultades que se deben superar antes de ofrecer una máquina que pueda mantener una conversación inteligente en tiempo real. Sobre todo, al hablar de sistemas de diálogo humano-máquina, se debe tener en cuenta la incorporación de un léxico de amplia cobertura, el análisis de sentimientos y la incorporación de un razonamiento que permita encadenar una conversación según las expresiones que manifieste cada usuario [26].

Como solución, a lo largo de los últimos años se han diseñado y establecido una serie de técnicas de pre-procesamiento del lenguaje que, sumadas al entrenamiento de modelos robustos y complejos de inteligencia artificial, mejoran de forma notable el rendimiento de los sistemas de diálogo humano –máquina [27]. En las siguientes subsecciones, se explicará a detalle diferentes métodos de pre-procesamientos y modelos de entrenamiento que intervienen en la obtención de un modelo de lenguaje de alta precisión.

3.1. Normalización.

Dentro del PLN, es común que el texto a procesar presente un alto grado de aleatoriedad, debido a que el lenguaje, aunque tenga reglas comunes según el idioma, obedezca en su mayoría a la trama en el que se desarrolla. De ahí que, se conoce como normalización al proceso que intenta estandarizar las palabras/frases que conforman un

corpus, antes de entrenar un modelo [28]. Como parte de los subprocesos que se implementan como parte de la normalización, se encuentra la tokenización, eliminación de signos de puntuación y abreviaciones, remoción de palabras de parada, stemming y lematización, los cuales se describirán a continuación.

3.1.1. Tokenización

Es el proceso de segmentar un texto en frases (tokenización frases) o palabras (tokenización de palabras) conocidos como tokens, al mismo tiempo que se desechan ciertos caracteres que no aportan información relevante como los signos de puntuación. De esta manera, es posible entender el contexto y analizar la secuencia de palabras/frases para alimentar a un modelo de PLN [29]. Existen diferentes técnicas de tokenización, entre las que se pueden destacar:

- Tokenización de espacios en blanco: es la técnica más sencilla y rápida de implementar, pues las frases o párrafos se dividen cada vez que se encuentra un espacio en blanco. Sin embargo, únicamente funciona en idiomas en los que los espacios en blanco segmentan el texto en palabras/frases significativas.
- Tokenización basada en diccionario: en este método, los tokens se extraen a partir de un conjunto de palabras/frases previamente almacenadas a manera de diccionario. En caso de no localizar el token, se emplean reglas especiales para proceder con la segmentación del texto.
- Tokenización basada en reglas: la tokenización se ejecuta siguiendo un conjunto de reglas. Lo más común es que estas reglas se establezcan en función de la gramática de un idioma o a través de expresiones regulares.
- Tokenización espacial (spacy): se trata de una técnica moderna, rápida y personalizable que incluye tokens especiales que necesitan o no ser segmentados. Por ejemplo, si se desea mantener el signo # como un token, se le dará prioridad sobre otras operaciones.

3.1.2. Remoción de palabras de parada

Las palabras de parada son aquellas palabras que no añaden ningún significado a la frase, por lo que su eliminación no afectará al procesamiento, ayudando a reducir el ruido y la dimensión del corpus [30]. Si bien no existe una lista universal de palabras de

parada, ya que dependen en gran medida del idioma, se puede generar un diccionario tomando en consideración las siguientes reglas:

- Eliminar aquellas palabras que tienen una frecuencia de aparición mayor al 80%.
- Eliminar los artículos, pronombres, preposiciones, y las conjunciones.
- Eliminar algunos verbos, adverbios y adjetivo.
- Eliminar modismos propios de cada idioma.

El proceso de eliminación es simple: después de tokenizar un texto, se busca cada token en el conjunto de palabras de parada; cada coincidencia implica que se quite el token bajo análisis del corpus. Sin embargo, se debe tener en consideración que la remoción de palabras de parada puede excluir información relevante y modificar el contexto. Por ejemplo, bajo un escenario de análisis de sentimientos, al remover la palabra "no", se perdería por completo el sentido de lo que se quiere comunicar y el entrenamiento sería erróneo.

3.1.3. Stemming

Hace alusión al proceso de cortar el inicio o el final de una palabra con el objetivo de eliminar sus afijos (prefijos y sufijos). Bajo esta perspectiva, el problema radica en que los afijos pueden crear o ampliar nuevas formas de la misma palabra (afijos flexivos), o incluso crear palabras nuevas (afijos derivativos), afectando directamente al modelo que se entrene. Así, una posible solución va de la mano con implementar un stemming basado en una lista de afijos y reglas comunes. Sin embargo, el resultado de este proceso puede arrojar palabras inexistentes o cambiar el significado de lo que se está expresando [31].

3.1.4. Lematización

A través de este pre-procesamiento, se pretende reducir una palabra a su forma base, agrupando diferentes formas de la misma. Por ejemplo, los verbos en cualquier tiempo se transforman a su infinitivo y los sinónimos se unifican, estandarizando el corpus. Vale recalcar que, aunque la lematización parece relacionarse directamente con el stemming, ésta utiliza un enfoque diferente para llegar a la forma base de las palabras: toma en cuenta el contexto para proveer un significado más exacto de lo que se desea comunicar, discriminando entre palabras idénticas. Por último, a pesar de los beneficios que refleja, es una tarea que consume una cantidad importante de recursos y exige una potencia computacional elevada [32].

3.2. Etiquetado de partes-de-la-oración

Al momento de procesar cualquier tipo de texto es común que las palabras reflejen significados ambiguos. Por ejemplo, si se normalizan las frases "él bebió un vaso de agua" y "su bebida se enfrió", las palabras "bebió" y "bebida" podrían reducirse a "beb", creando un grave problema de entendimiento: la primera es un verbo, mientras que la segunda es un sustantivo. Como solución, la técnica del etiquetado de partes-de-la-oración (del inglés Part-Of-Speech Tags) es un método que etiqueta una palabra según la posición que ocupe dentro del texto, basándose en su contexto, definición y posición: sustantivo, verbo, adjetivo, adverbio, preposición, entre otras [33]. Como es de esperarse, no es una tarea sencilla, por lo que puede llevarse a cabo según diferentes enfoques:

- Métodos basados en el léxico: asignan la etiqueta que aparece con más frecuencia para una determinada palabra.
- Métodos basados en reglas: las etiquetas se colocan según reglas predefinidas. Por ejemplo, se especifica que todas las palabras terminadas en ar-er-ir son verbos.
- Métodos probabilísticos: colocan una etiqueta según la probabilidad de aparición de una secuencia de palabras en concreto.
- Métodos de deep learning: emplean redes neuronales recurrentes para clasificar y asignar una etiqueta.

3.3. Vectorización o word embeddings

Con el objetivo de permitir que una computadora procese lenguaje, es necesario que el texto con el que se trabaja sea convertido a un vector continuo de valores numéricos. Para realizar este procedimiento, se puede emplear un sin número de métodos que se apoyan en la vectorización o incrustación de palabras (del inglés word embeddings). De manera general, las alternativas más simples buscan representar a cada documento de texto como una matriz de una sola dimensión o vector; mientras que, los más complejos, a más de realizar una representación vectorial, reflejan la estructura semántica y contextual que guardan las palabras entre sí [34]. En las siguientes subsecciones, se explicarán a detalle las tres opciones más conocidas para vectorizar texto: one-hot encoding, bag of words y word2vec.

3.3.1. One hot-encoding

En este método, las palabras que conforman un texto son escritas en forma de un vector constituido únicamente por 0 y 1. Así, cada palabra se codifica como un *hot-vector* único que puede identificarse a posteriori con facilidad, pues no existen dos palabras que se representen con un mismo *hot-vector*. Por consiguiente, al convertir una frase completa se obtendría una matriz conformada por varios *hot-vectors*. Por último, aunque es una técnica fácil de implementar, expone dificultades relacionadas con el procesamiento y costo computacional, pues las dimensiones de cada vector dependerán del tamaño total del corpus [35]. En la Figura 2, se ejemplifica este tipo de codificación.

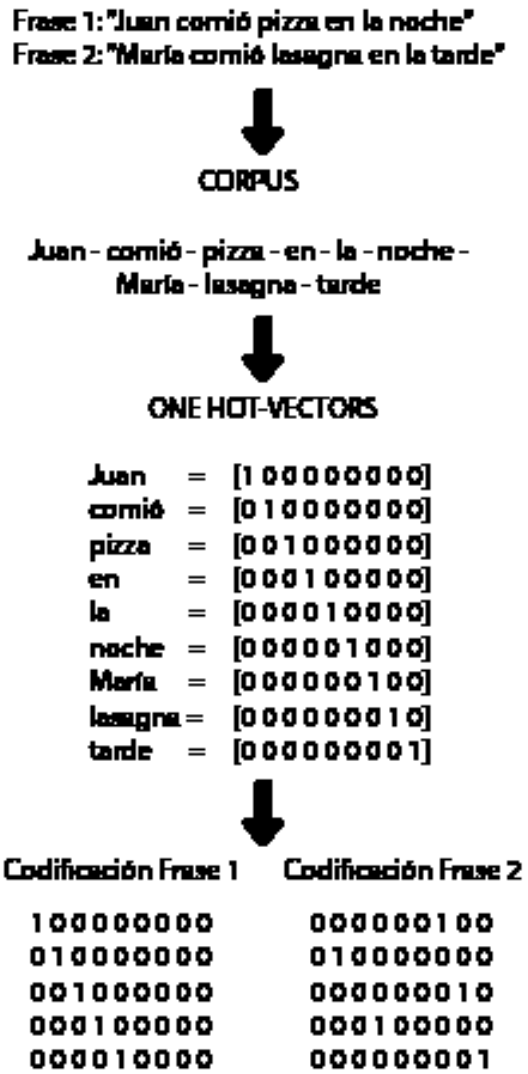

Figura 2.Ejemplo de codificación one hot-vector

Si bien el one hot-vector es un método sencillo de implementar, presenta dificultades relacionadas con la eficiencia computacional que supone trabajar con corpus de altas dimensiones. Por ejemplo, al tener un corpus de 50,000 palabras, cada una se

representa con 49,999 ceros y 1 solo uno, necesitando alrededor de 2,500 millones de unidades de espacio de memoria. En definitiva, la mayor parte de la matriz que codifica una frase está ocupada por ceros, reflejando una escasez de datos útiles.

Por otro lado, resulta difícil extraer información estructural y de contextualización, pues los vectores que representan una frase son independientes, implicando que cada palabra este aislada. Por ejemplo, al analizar las palabras "naranja", "plátano" y "sandía" es fácil que una persona encuentre una similitud entre todas: son frutas o se escriben después del verbo “comer”. No obstante, es imposible que una máquina determine esa similitud empleando hot-vectors.

3.3.2. Bag of Words

Una bolsa de palabras (del inglés Bag of Words – BoW) es una manera simple y efectiva de extraer y representar las características de un texto, pues se basa en la ocurrencia de cada palabra. Sin embargo, no toma en consideración ninguna información sobre el orden o estructura que mantiene el texto, de ahí el nombre de “bolsa”. Así, la codificación permite conocer el número de veces que se repitió una palabra en el corpus, más no el lugar en el que apareció [36]. En la Figura 3, se expone un ejemplo que permite entender de mejor manera este método.

Figura 3.Ejemplo de codificación por Bag of Words.

Debido a la falta de estructura y contextualización que puede suponer el empleo de una BoW, es común que se agrupen dos o más palabras para la codificación formando lo que se conoce como n-grama. De esta manera, se puede mantener la relación secuencial que existe y, por ende, considerar estructuras gramaticales que permiten inferir una mayor cantidad de información. Sin embargo, se siguen manteniendo los problemas de eficiencia computacional, consecuencia de las altas dimensiones que podría tener el corpus con el que se trabaje.

3.3.3. Word2Vec

Word2Vec es una red neuronal de dos capas que procesa el texto a través de la vectorización de las palabras. Como entrada toma un corpus para obtener como salida un conjunto de vectores característicos que permiten considerar el contexto y relación entre palabras. Esto se consigue gracias a que se crean vectores que reflejan similitudes matemáticas, ayudando a distribuir numéricamente las características de las palabras [37].

Conceptualmente, el funcionamiento de Word2Vec es sencillo, pues realiza predicciones de alta precisión sobre el significado de una palabra basándose en sus apariciones y la similitud del coseno entre ellas para establecer una relación. La ausencia de similitud se expresa como un ángulo de 90 grados (0), mientras que la similitud total como un ángulo de 0 grados (1), reflejando un solapamiento completo; por ejemplo, "niño" es a "hombre" como "niña" es a mujer. Por otro lado, en un contexto más amplio interviene en la agrupación y clasificación de documentos según tópicos, lo cual es de gran importancia para desarrollar búsquedas semánticas, análisis de sentimientos y recomendaciones. A su salida arroja un vocabulario en el que cada elemento tiene un vector asociado, conocido como incrustación neuronal, que permite que el lenguaje natural sea legible por una máquina al realizar potentes operaciones matemáticas que detectan similitudes [38].

Ahora bien, computacionalmente Word2vec es similar a un autocodificador, que representa cada palabra en un vector que se entrena en base a palabras vecinas. Así, cuando dicho vector no puede usarse para predecir con precisión el contexto se ajustan sus componentes, logrando que palabras consideradas similares mantengan una relación cercana que expresa una idea (información). Para ello, el tratamiento del corpus de entrada sigue un pre-procesamiento basado en pares que refleja la co-ocurrencia de los datos, lo cual se conoce como ventana de contexto [37]. Por ejemplo, tomando el texto "El procesamiento del lenguaje natural es divertido" con una ventana de contexto de 2, se iteran 2 palabras antes y 2 después del objetivo, consiguiendo un total de 4 asociaciones como se puede observar en la Figura 4.

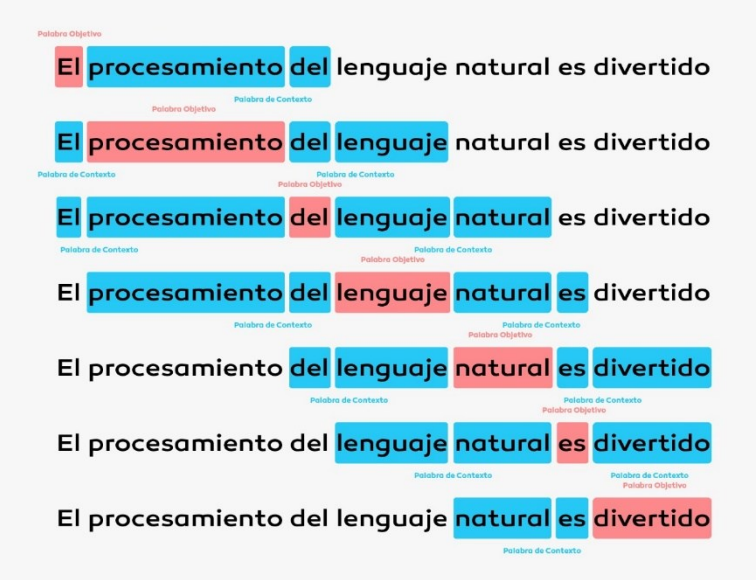

Figura 4.Ejemplo de co-ocurrencia de palabras sobre un objetivo con una ventana de contexto igual a 2.

Como se puede notar, las palabras más cercanas al inicio/final de la frase no cumplen con la ventana de contexto, por lo que su valor será inferior a 4. Al finalizar con la ejecución, se encuentran todos los pares de destino para formar un corpus según la relación palabra objetivo/palabra de contexto que se puede expresar de la siguiente manera:

$$Primera\ ventana\ de\ pares: (El, procesamiento)(El, del)$$
$$Segunda\ ventana\ de\ pares: (procesamiento, El)(procesamiento, del)(procesamiento, lenguaje)$$
$$Tercera\ ventana: (del, El)(del, procesamiento)(del, lenguaje)(del, natural)$$

Al final, se consigue el corpus (El, procesamiento) (El, del) (procesamiento, El) (procesamiento, del) (procesamiento, lenguaje) (del, el) (del, procesamiento) (del, lenguaje) (del, natural) (lenguaje, procesamiento) (lenguaje, del) (lenguaje, natural) (lenguaje, es) (natural, del) (natural, lenguaje) (natural, es) (natural, divertido) (es, lenguaje) (es, natural) (es, divertido) (divertido, natural) (divertido, es).

Con el corpus conformado, se puede seguir dos enfoques para vectorizar el texto: (i) un skip-gram que emplea una palabra para predecir un contexto objetivo; y (ii) un bag of words continuas (del inglés Continuos Bag of Words-CBoW) que utiliza el contexto para predecir una palabra objetivo.

3.3.3.1. Skip-gram

En el modelo Skip Gram, se intenta predecir una palabra contexto dada una palabra objetivo a través de una red neuronal. La entrada y salida serán una versión de

codificación one-hot de la palabra objetivo y de la palabra contexto, respectivamente, por lo que el tamaño de las capas de entrada y salida es igual al tamaño del corpus. Esta red neuronal sólo tiene una capa oculta en la mitad, la cual determina el tamaño de los vectores de palabras que se desea tener al final, a más de que solo existen dos matrices de peso, W1 y W2, como se muestra en la Figura 5 [39].

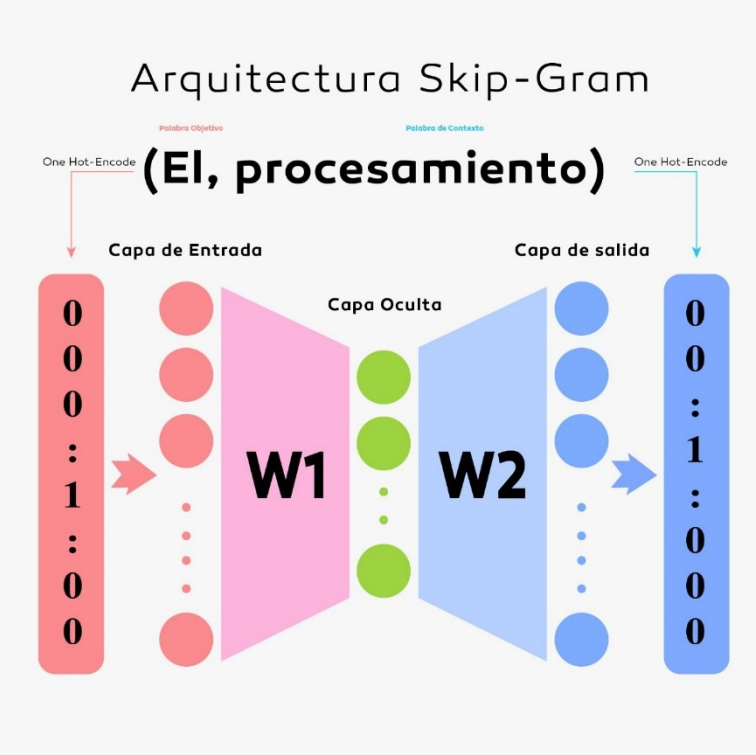

Figura 5.Arquitectura del Skip-Gram.

En resumen, todo el corpus (palabra objetivo, palabra de contexto) pasará en pares por la red neuronal para el entrenamiento y, posteriormente, predecir qué palabras de contexto pueden aparecer a partir de una palabra objetivo. Al finalizar el entrenamiento, si se introduce cualquier palabra objetivo se generará un vector de salida que representa las palabras que tienen una alta probabilidad de co-ocurrencia con el contexto.

3.3.3.2. CBoW

El funcionamiento de este enfoque es contrario al Skip-Gram, por lo que la única diferencia es que se intenta predecir la palabra objetivo a partir de las palabras de contexto. Por tanto, la red neuronal de entrenamiento guarda características similares, como se visualiza en la Figura 6. Entonces, cuando se ingresa una representación vectorial de un grupo de palabras de contexto, se obtendrá la palabra objetivo más próxima a ellas. Por ejemplo, con la frase “El _____ del lenguaje natural”, en la que el vector ["El”, "del",

"lenguaje", "natural"] representa las palabras de contexto, la red neuronal debería reflejar "aprendizaje" como palabra objetivo [40].

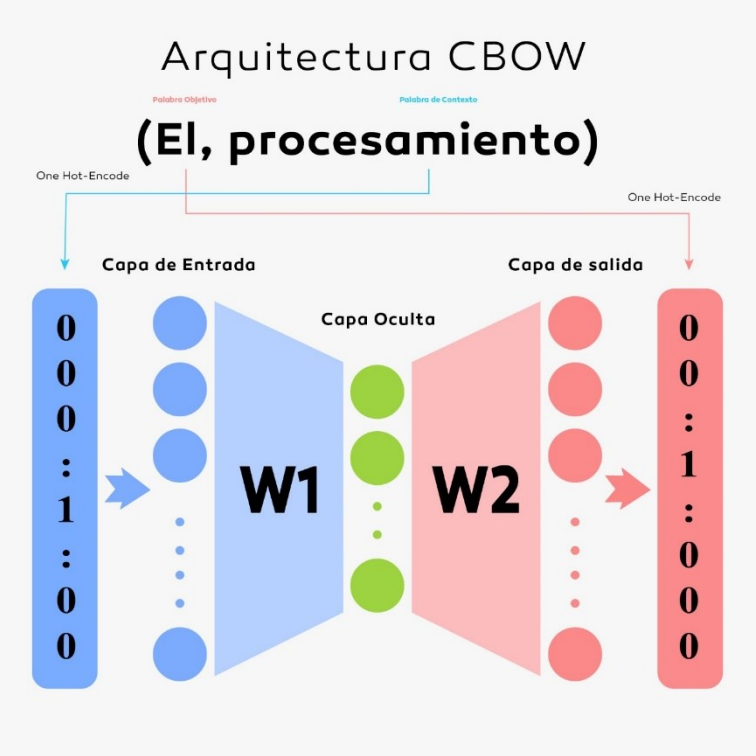

Figura 6.Arquitectura del CBoW.

3.4. Transformers

Con el constante desarrollo de la inteligencia artificial, es posible implementar diferentes modelos neuronales que permiten incrementar de forma notable las tasas de aprendizaje y predicción. Estos modelos, conocidos comúnmente como transformes, buscan resalta dinámicamente las características de los datos de entrada, los cuales, en el campo del PLN, suelen ser elementos de texto secuenciales. De esta manera, se trabaja con una arquitectura sequence-to-sequence (Seq2Seq), en la que una secuencia de elementos de entrada, como las palabras de una oración, se transforman en una secuencia de salida. Así, al ser dependientes del orden de los datos, pueden otorgar un significado a las palabras de salida, dependiendo de la importancia que se les de esta recordara u olvidara las palabras de entrada [41]. En definitiva, la idea principal es calcular una distribución de pesos en la secuencia de entrada, asignando valores más altos a los elementos más relevantes.

Los modelos Seq2Seq están conformados por un codificador que toma una sucesión de entrada y la mapea en un espacio de mayor dimensión (vector n-dimensional)

que se introduce en un decodificador para convertirlo en una sucesión de salida. Para entender mejor su funcionamiento se puede imaginar que el codificador y el decodificador son traductores humanos en el que su habla solo esta basada en dos tipos de idioma: su lengua materna, diferente entre ambos, y una lengua imaginaria [42]. En la traducción de alemán a francés, el codificador transforma una frase alemana a lengua imaginaria para que el decodificador sea capaz de entender y traducir al francés.

Ahora bien, suponiendo que la lengua imaginaria no sea fluida para los traductores, codificador y decodificador, es necesario que aprendan a través de un entrenamiento. Debido a que el procesamiento del lenguaje no sigue los mismos lineamientos de conjuntos de datos numéricos, es imprescindible recurrir a nuevas técnicas y mecanismos, entre los que se puede destacar "attention" [43].

3.4.1. Attention

Es un mecanismo que observa una sucesión de entrada la cual determina cada paso de entrenamiento las palabras que son más importantes. Aplicando attention al ejemplo del traductor alemán/francés, el codificador en lugar de proporcionar únicamente la traducción al decodificador, también le pasa palabras clave que guardan la semántica. De esta manera, se facilita la traducción para el decodificador, pues conoce de antemano las partes importantes de la frase y los términos clave que le otorgan contexto. En definitiva, attention analiza al mismo tiempo todos los datos de una secuencia y les atribuye peso según su importancia.

Partiendo de los enfoques de attention, nace lo que se conoce como self-attention para relacionar diferentes posiciones de una misma oración, calcular su representación y crear conexiones similares entre las palabras. Con las ideas descritas, es posible abordar la arquitectura de attention en la siguiente subsección.

3.4.2. Arquitectura de attention

La mayoría de los modelos utilizados para PLN secuenciales reflejan una estructura de codificador-decodificador. El codificador mapea una secuencia de entrada de representaciones de símbolos $x = (x_1, \ldots, x_n)$ a una secuencia de representaciones continuas $z = (z_1, \ldots, z_n)$. Dado z, el decodificador genera una secuencia de salida $y = (y_1, \ldots, y_m)$ de símbolos, un elemento cada vez. En cada paso, el modelo es autorregresivo, apoyándose en los símbolos generados anteriormente como entrada adicional al generar el siguiente. Como se puede observar en la Figura 7, la arquitectura

de attention también está conformada por un codificador y un decodificador. Ambos se componen de módulos que pueden apilarse entre sí varias veces, lo que se describe con Nx.

En primer lugar, el codificador describe una pila de $N = 6$ capas idénticas, las cuales poseen dos subcapas: i) un mecanismo multi-head attention, y la segunda es una red simple, totalmente conectada en función de tipo feed-forward, que se comunican entre sí gracias a una conexión residual seguida de una normalización. Para facilitar estas conexiones, todas las subcapas del modelo, así como las capas embeddings producen salidas de dimensión $d_{model} = 512$. Por su parte, el decodificador también consta de una pila de $N = 6\, capas$ idénticas. Sin embargo, en comparación con las subcapas de codificador, se inserta una tercera multi-head attention sobre la salida de la pila del codificador y se añade un mecanismo de enmascaramiento que garantice que las predicciones para una posición determinada sólo pueden depender de las salidas conocidas en posiciones inferiores.

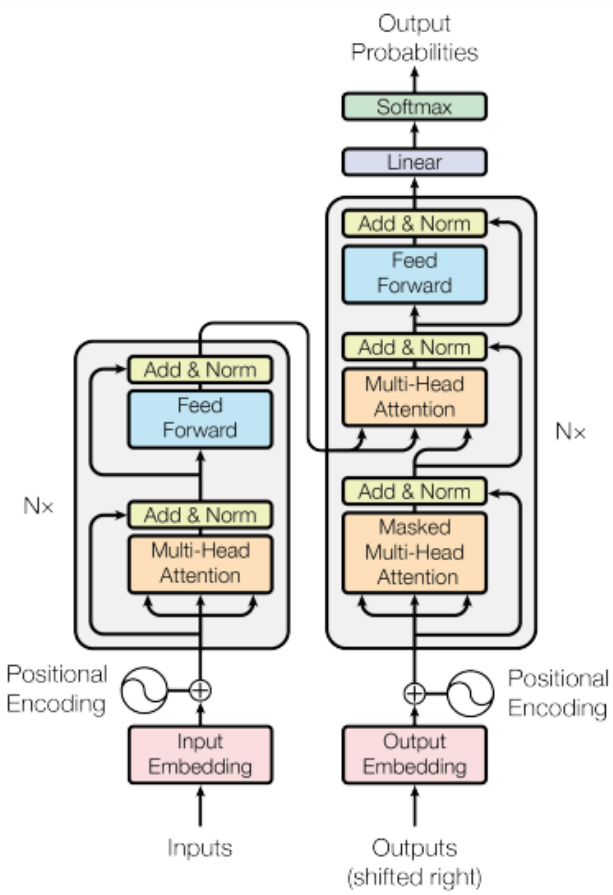

Figura 7.Arquitectura de attention [43].

Antes de describir el funcionamiento de la capa multi-head attention, es fundamental conocer los mecanismos que están por detrás de attention, los cuales se describen en la siguiente ecuación:

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

Q es la matriz encargada de incluir un query (representación vectorial de una palabra de la sucesión), K son todas las claves (representación vectorial de todas las palabras de la secuencia) y V son los valores (representación vectorial de todas las palabras de la secuencia a las que se ha añadido una puntuación). En otras palabras, a través de attention se compara el query con las claves y se obtienen puntuaciones para los valores. Cada puntuación representa la relevancia entre la consulta y cada clave. Para los módulos multi-head attention del codificador y decodificador, V llega a ser una sucesión de palabras idéntica que Q. No obstante, para el módulo que tiene presente las sucesiones del codificador y del decodificador, V es distinta de la secesión representada por Q. Así,

con fines de simplificación, se quiero expresar que los valores de V son multiplicados y sumados con unos pesos attention a, los cuales se definen por:

$$a = softmax\left(\frac{QK^T}{\sqrt{d_K}}\right)$$

Es decir, los pesos a se definen en función de cómo cada una de las palabras de la sucesion (Q) está influenciada por todas las demás palabras de la sucesion (K) y se encuentran entre 0 y 1 por la aplicación de la función softmax. Entonces, estos pesos son aplicados a todas las palabras de la sucesión que se introduce en V.

Ahora bien, un multi-head attention paraleliza múltiples mecanismos que se ejecutan uno al lado del otro, mediante proyecciones lineales de Q, K y V, como se puede apreciar en la Figura 8. Así, el sistema aprende de diferentes representaciones de Q, K y V, en beneficio del modelo. Las representaciones lineales se desarrollan al multiplicar Q, K y V por matrices de peso W que se aprenden durante el entrenamiento.

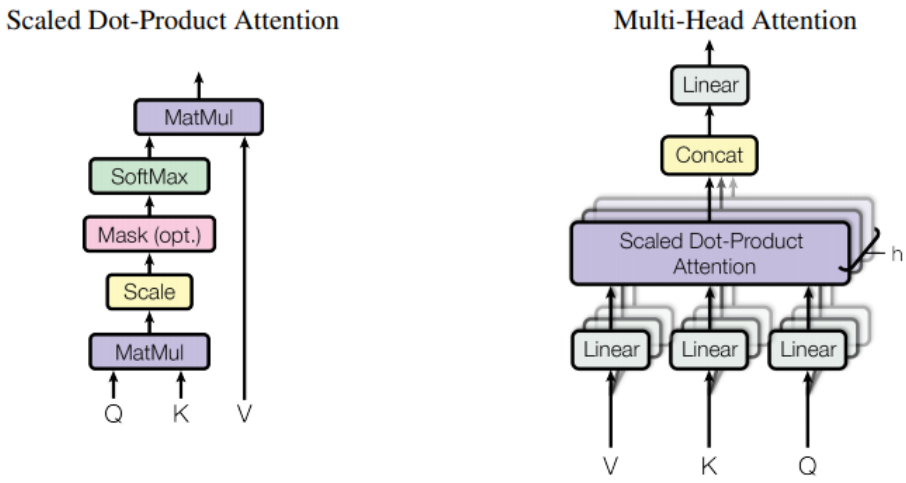

Figura 8.Arquitectura del multi-head attention [43].

La ventaja de attention se fundamenta en que solo presta atención a algunas palabras de entrada en las que se concentra la información más relevante, en lugar de una frase entera. Esto incrementa de gran manera el rendimiento del entrenamiento en comparación con modelos tradicionales en los que toda la frase de entrada se codifica en un vector de longitud fija.

3.5. Bidirectional Encoder Representation from Transformer (BERT)

A finales de 2018, los investigadores de Google diseñaron un nuevo modelo de NPL, llamado BERT (del inglés Bidirectional Encoder Representation from Transformer)

[44], como una alternativa para solucionar el problema de la escasez de conjuntos de entrenamiento. En general, si bien existe una gran cantidad de corpus disponibles, al momento de trabajar sobre una tarea en específico se reducen de manera notable los ejemplos de entrenamiento etiquetados por humanos, lo cual crea una dificultad relacionada con la cantidad de datos que demandan los modelos de aprendizaje profundo para llegar a resultados satisfactorios. BERT es un mecanismo de pre-procesamiento que, a partir de representaciones del lenguaje de propósito general, se afina para cumplir con los requerimientos de tareas más específicas (transfer learning).

Para entender la idea principal detrás de BERT, es fundamental comprender que el objetivo de una solución de PLN es predecir una palabra basándose en un contexto. De manera tradicional, esta predicción se realiza analizando una secuencia de texto durante un entrenamiento que sigue un flujo en una sola dirección: izquierda-derecha o una combinación derecha-izquierda e izquierda-derecha. De esta manera, es posible obtener buenos resultados para aplicaciones en las que se requiera completar una palabra al final de la secuencia, por ejemplo, la creación de frases. Sin embargo, resultado complicado obtener predicciones adecuadas tomando el contexto que rodea a una palabra intermedia.

En consecuencia, BERT surge como un modelo que se entrena de forma bidireccional para extraer un sentido más profundo del contexto. Para ello, se realiza un enmascaramiento de la palabra a predecir y se extrae la información del contexto de la frase tanto hacia la izquierda como hacia la derecha, apoyándose en mecanismos attention que intervienen en el aprendizaje de las relaciones entre las palabras. Vale destacar que, previo al entrenamiento, el texto de entrada es convertido en una secuencia de tokens a la que se añaden ciertas etiquetas y metadatos como se puede observar en la Figura 9.

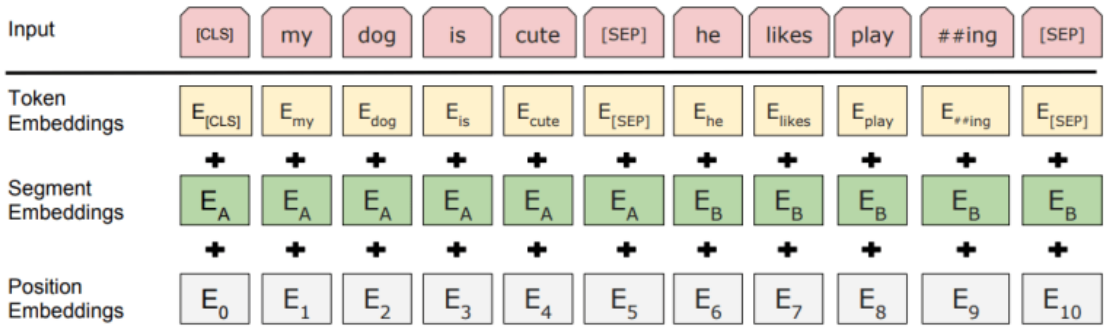

Figura 9.Representación de una secuencia de entrada para BERT [45].

Entre las etiquetas y metadatos mencionados, se detallan los siguientes:

- Embeddings de token: se añade una etiqueta [CLS] como token al principio de la primera oración y una etiqueta [SEP] como token al final de cada oración.
- Embeddings de segmento: se añade una etiqueta como token que indica la pertenencia de una palabra a cada oración.
- Embeddings de posición: se añade una etiqueta de posición como token que señala la ubicación de cada palabra en la oración.

BERT puede ser entrenado para que se adapte a tareas particulares, para ello se tienen dos alternativas: modelado del lenguaje enmascarado (del inglés Masked Lenguage Model - MLM) y predicción de la siguiente oración (del inglés Next Sentence Prediction), las cuales se explican en las siguientes subsecciones.

3.5.1. Modelado del Lenguaje Enmascarado (MLM)

BERT enmascara aleatoriamente el 15% de las palabras de entrada, sustituyéndolas por un token [MASK]. De esta manera, se pasa la secuencia completa por el codificador y se predicen las palabras enmascaradas mediante el contexto proporcionado por las palabras no enmascaradas. Sin embargo, el modelo intenta realizar una predicción únicamente cuando el token [MASK] está presente en la entrada, ocasionando un problema, puesto que se desea predecir los tokens correctos independientemente de lo que se encuentre en la entrada. Como solución, del 15% de los tokens, el 80% de se sustituyen realmente por el token [MASK], el 10% se sustituyen por un token aleatorio y el 10% restante se dejan sin cambiar. Por último, la desventaja de este proceso es que se ignora la predicción de las palabras no enmascaradas, reflejando una convergencia más lenta que modelos tradicionales.

3.5.2. Predicción de la siguiente oración

BERT permite relacionar dos oraciones a través de la predicción de la siguiente frase. Para ello, como parte de las entradas se recibe un par de frases separadas por token [SEP] para aprender a predecir si la siguiente frase es la lo que le sigue en el texto original. En ese sentido, vale destacar que un 50% de las veces la segunda frase viene después de la primera y el otro 50% se coloca una frase aleatoria del corpus. Al final, se debe predecir si es que existe una conexión entre ellas. Como parte del pre-procesamiento a nivel de frase se coloca una etiqueta de pertinencia, lo cual se puede observar en el siguiente ejemplo:

$Entrada_1$: $[CLS]$ $El\ hombre\ vino\ a$ $[MASK]$ $tienda$ $[SEP]$ $él\ compró\ un\ galón$ $[MASK]$ $leche.$
$Etiqueta_1$: $IsNext$

$Entrada_2$: $[CLS]$ $El\ hombre\ vino\ a$ $[MASK]$ $tienda$ $[SEP]$ $pinguino$ $[MASK]$ $pájaro\ no\ volador.$
$Etiqueta_2$: $NotNext$

Para predecir si la segunda frase está conectada a la primera, la secuencia de entrada completa pasa por el codificador, la salida del token [CLS] se transforma en un vector 2×1 utilizando una capa de clasificación simple y la etiqueta $IsNext$ es asignada a través de una capa softmax.

3.6. Word2Vect Vs BERT Embedding

Las representaciones lingüísticas pre-entrenadas pueden considerar o no el contexto, dando paso a una diferencia crucial entre un embedding tradicional en comparación con BERT. Los modelos libres de contexto, como word2vec, generan una única representación a través de un vector de números para el embbeding de palabras. Por ejemplo, la palabra "banco" tendría la misma representación en la frase "cuenta bancaria" y "orilla del río". En cambio, los modelos basados en el contexto arrojan una representación que se basa en todas las palabras de la frase. Por ejemplo, al analizar la frase "accedí a la cuenta del banco" con BERT, se representaría la palabra "banco" utilizando tanto su contexto anterior como el siguiente: "accedí a la... cuenta" [45].

Una vez entendidos los conceptos claves de BERT y NLP, en la siguiente sección se explicará brevemente lo relacionado con los datasets.

3.6.1. Datasets en NLP

Dentro de la inteligencia artificial, los datasets se conocen como un conjunto de instancias que comparten atributos comunes, a través de los cuales se entrena y valida un modelo. Específicamente, al tratarse de NLP, los datasets están conformados por texto que ha sido extraído de diferentes fuentes e idiomas, y dependerá del tipo de entrenamiento la codificación y el pre-procesamiento que se realice. Entre las alternativas más conocidas, se puede citar las siguientes:

- Blog Authorship Corpus: está conformado por los posts de 19.320 blogueros recogidos en blogger.com en agosto de 2004. Incorpora un total de 681.288 posts y más de 140 millones de palabras, es decir, aproximadamente 35 posts y 7250 palabras por persona. Cada blog se presenta como un archivo

independiente, cuyo nombre indica un identificador de blogger y el género, la edad, la industria y el signo astrológico que el propio blogger proporciona [46].

- Amazon Product Dataset: contiene reseñas y metadatos de productos de Amazon, incluyendo 142,8 millones de frases que abarcan desde mayo de 1996 hasta julio de 2014. Incluye reseñas (valoraciones, texto, votos de utilidad), metadatos del producto (descripciones, información de la categoría, precio, marca y características de la imagen) y enlaces (gráficos de "también visto/también comprado") [47].
- LibriSpeech: es un corpus de aproximadamente 1000 horas de habla inglesa leída a 16kHz. Los datos proceden de la lectura de audiolibros del proyecto LibriVox y han sido cuidadosamente segmentados y alineados [48].
- Free Spoken Digit Dataset: es una colección de datos de audio/voz que consiste en grabaciones en archivos WAV a 8kHz. Las grabaciones se recortan para que tengan un silencio casi mínimo al principio y al final [49].
- Stanford Question Answering Dataset (SQUAD v1): es un dataset de comprensión lectora que consiste en preguntas planteadas por crowd-workers sobre un conjunto de artículos de Wikipedia. La respuesta a cada pregunta es un segmento de texto, un spam del pasaje de lectura correspondiente, o texto vacío. Al final se combinan 100.000 preguntas respondidas con más de 50.000 nuevas preguntas sin respuesta escritas de forma adversa para que se parezcan a las que tienen respuesta [50].

CAPÍTULO 4

4. Metodología del trabajo

4.1. Objetivo General

Diseñar un sistema prototipo de diálogo persona-máquina basado en la arquitectura BERT.

4.2. Objetivo Especifico

- Analizar el funcionamiento de la arquitectura BERT en los sistemas de diálogo persona-máquina.
- Diseñar una interfaz de usuario en un software propietario con la que se pueda controlar un sistema de diálogo persona-máquina basado en la arquitectura BERT.
- Diseñar una interfaz de usuario con un software de libre licencia con la que se pueda controlar un sistema de diálogo persona-máquina basado en la arquitectura BERT.
- Probar el funcionamiento de las interfaces del sistema por medio de la medición del tiempo de conversación coherente entre la persona y la máquina.

4.3. Metodología

Estructura de los Sistemas de Diálogo Persona-Máquina

Los sistemas de diálogo persona-máquina consisten en varios componentes que interactúan para conseguir la comunicación entre humanos y máquinas. Entre los componentes están: el procesamiento del habla que consiste en el análisis de señales para caracterizar los patrones de voz, utilizando técnicas de procesamiento digital de señales [51, 52]. Según el enfoque de análisis se pueden realizar conversiones de voz a texto, llamados reconocedores de voz que consisten en la transformación automática, asistida por computadores, del lenguaje humano hablado al texto escrito correspondiente y conversiones de texto a voz a lo que se le llama síntesis del habla que se usa en aplicaciones donde los recursos audibles son transformados en base a textos escritos y

que estos recursos audibles sean capaces de transmitir emociones [53, 54]. En base de los reconocedores de voz se ha llevado el procesamiento de habla al reconocimiento de idioma o de locutor o incluso a procesos de diarización, etc. [55,56]. Por otro lado, se puede obtener una representación semántica del mensaje mediante la transformación de la voz, esto se logrará con el uso de Speech Understanding, y a su vez puede ser utilizado para el desarrollo de una acción o para trasladarla [57, 58].

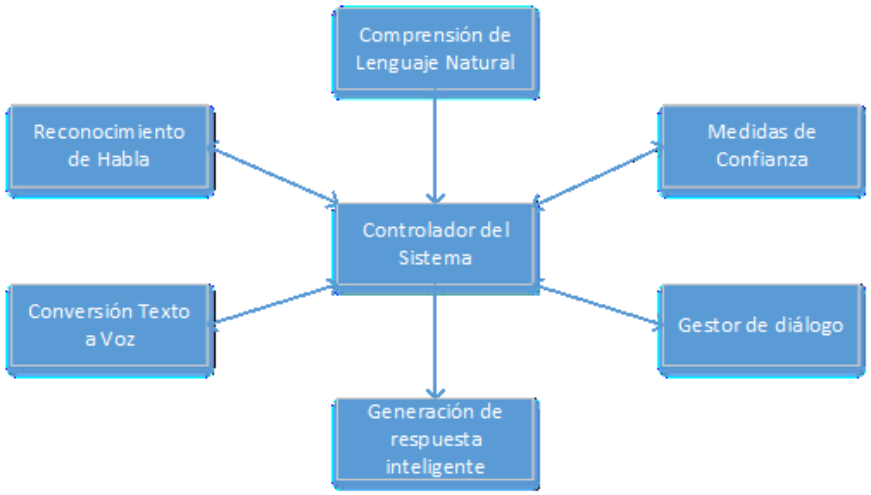

Figura 10.Componentes de un sistema de diálogo persona – máquina [11].

La metodología a aplicarse para el desarrollo del sistema de diálogo persona-máquina es a través del uso de la arquitectura BERT. Al tener una menor carga computacional se va a utilizar el modelo de BERT Small; como se conoce el algoritmo de BERT, es una concatenación de múltiples codificadores que están entrenados previamente. Esto le permite refinar con solo una capa adicional de salida para crear modelos de vanguardia para una amplia gama de actividades, como la respuesta a preguntas y la inferencia lingüística [59].

El proceso para el desarrollo de nuestro trabajo se detalla a continuación. Como primer punto se realiza el análisis del funcionamiento de la arquitectura BERT en los sistemas de diálogo de persona – máquina, esto con la finalidad de poder acondicionar el algoritmo a las necesidades de nuestro trabajo. A continuación, se realizará el análisis para la selección de los softwares (libre y propietario) para el desarrollo e implementación de las interfaces de usuario. Al tener los softwares necesarios se procederá a diseñar las interfaces de usuario, las cuales nos permitirán el ingreso del texto, su acondicionamiento de señal, procesar dicha entrada de información a través del algoritmo de BERT Small y

obteniendo en la salida la respuesta que se requiera. Finalmente se realizará las pruebas de funcionamiento, las mismas que estarán basadas en la medición de tiempo de conversación coherente entre la persona y la máquina. El proceso resumido a seguir se muestra en la figura 11.

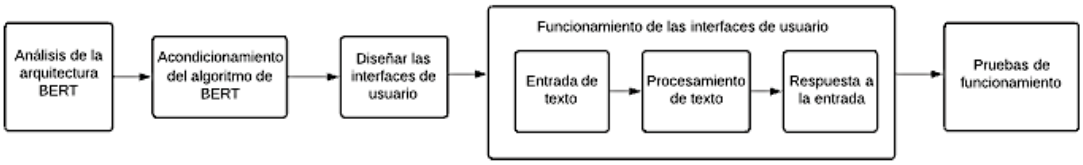

Figura 11. Diagrama sistema de diálogo persona-máquina a implementar.

CAPÍTULO 5

5. Desarrollo del trabajo

En el presente capítulo se aborda los temas sobre la implementación del sistema, que consiste en ajustar el modelo de BERT a las interfaces diseñadas en los distintos softwares detallados al inicio de este trabajo. Además, se realiza una revisión a la estructura del dataset SQUAD que es un conjunto de datos para comprensión lectora basado en los artículos disponibles en Wikipedia. Finalmente, se realiza el análisis de los resultados obtenidos en base a las diferentes métricas de evaluación planteadas.

Para el desarrollo del presente trabajo, se realiza la implementación de la arquitectura BERT en una interfaz en el software Anaconda Python y Matlab que permita el correcto funcionamiento de la misma. Esta a su vez, permitirá la búsqueda de respuestas concretas a una pregunta realizada por el usuario sobre un escrito o documento que se desee analizar; esto se da a través del entrenamiento de una red neuronal que utiliza el dataset SQUAD y aplicando la arquitectura BERT, permitiendo realizar un análisis del texto de una manera bidireccional; de este modo se obtiene una respuesta concreta y precisa a la pregunta realizada por el usuario que maneja la interfaz realizada.

5.1. Contribuciones del trabajo.

Como principal contribución se tiene la investigación y el análisis de la arquitectura utilizada en este trabajo de titulación. Para obtener el resultado esperado, en este caso la respuesta precisa y concreta a preguntas relacionadas con documentos de hasta 200 páginas con la ayuda de la arquitectura y los softwares utilizados.

A continuación, se tiene las interfaces desarrolladas en los diferentes softwares como es Anaconda Python y Matlab, para obtener una respuesta amigable e intuitiva por parte del usuario.

- Interfaz en el software Anaconda Python:

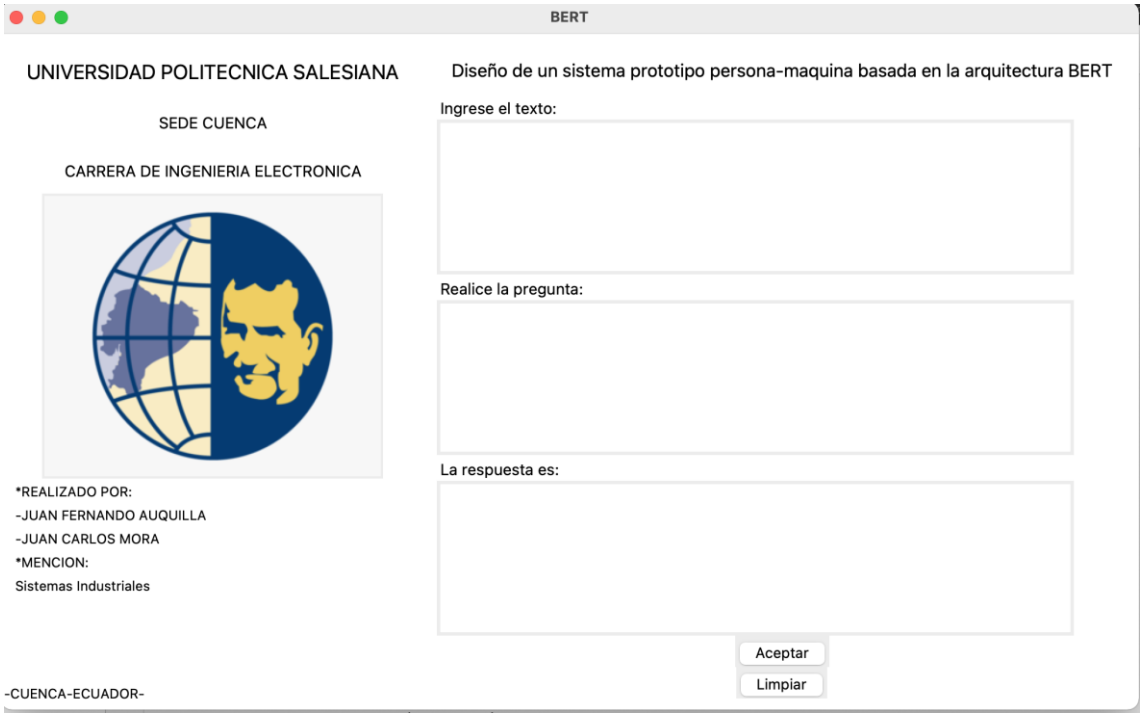

Figura 12. Interfaz realizada en el software Python Anaconda.

Al decir que la interfaz de usuario va a ser intuitiva, nos referimos a que ésta sea fácil de entender y de usar a la vez, como se puede observar en la interfaz realizada. Se han implementado tres cuadros, los cuales permitirán la ejecución de la arquitectura BERT, tenemos un apartado donde nos permitirá ingresar el texto a analizar.

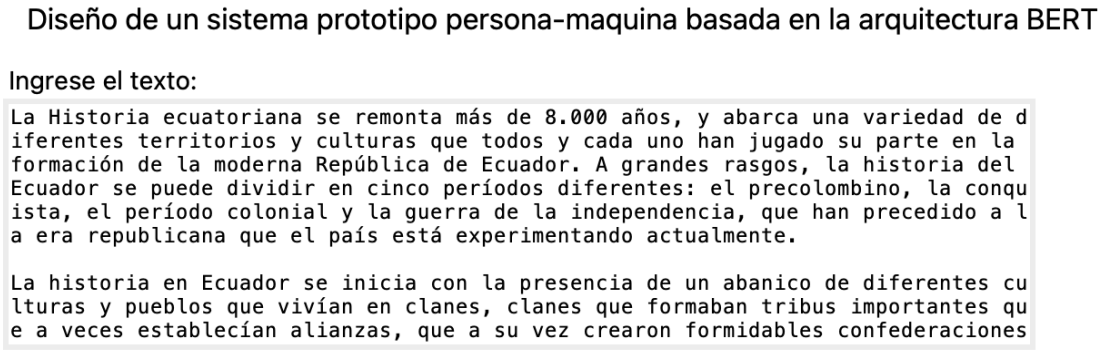

Figura 13.Cuadro para ingresar el texto a ser analizado en Python.

En este caso, como manera de ejemplo se ingresó un párrafo donde se habla de la historia del Ecuador, siendo este el texto que el código analizará en base a la pregunta que se le realice para la obtención de una respuesta precisa basada en este párrafo.

De esta manera, una vez ingresado dicho párrafo a continuación se debe realizar la consulta, por lo cual se tiene el espacio para que se realice la misma; y así se podrá analizar y buscar lo que se desea dentro del texto que se ingresó.

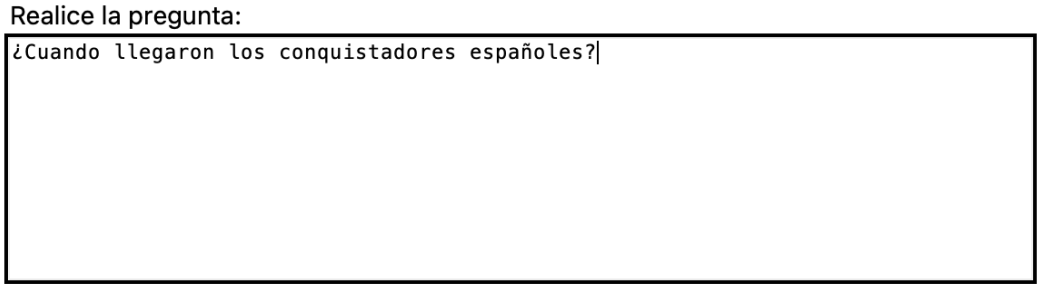

Figura 14.Cuadro para ingresar la pregunta en Python.

Y a su vez el apartado donde nos muestra una respuesta. Como se detalló anteriormente, la arquitectura BERT al tener un análisis bidireccional del texto nos permite obtener una respuesta concisa y precisa a la pregunta que se realiza, como se puede observar a continuación.

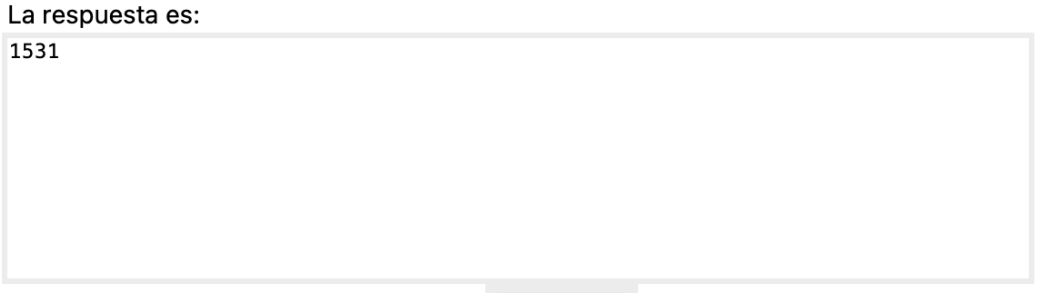

Figura 15.Cuadro para visualizar la respuesta en Python.

En este caso como se puede observar en el texto que se ingresó, es un párrafo corto de una parte de la Historia del Ecuador como un ejemplo, al ingresar la pregunta, *"¿Cuándo llegaron los conquistadores españoles?",* la arquitectura BERT entregó como respuesta *"1531"*, que según el texto que se ingresó es la respuesta correspondiente a la pregunta que se realizó para que se analice.

Y al ver una parte del párrafo que se analizó para este ejemplo podemos observar que *"Los conquistadores españoles liderados por Francisco Pizarro llegaron a Ecuador en 1531.",* siendo esta la línea donde se encuentra la respuesta a dicha pregunta y comprobando que la respuesta es correcta.

- Interfaz en el software Matlab:

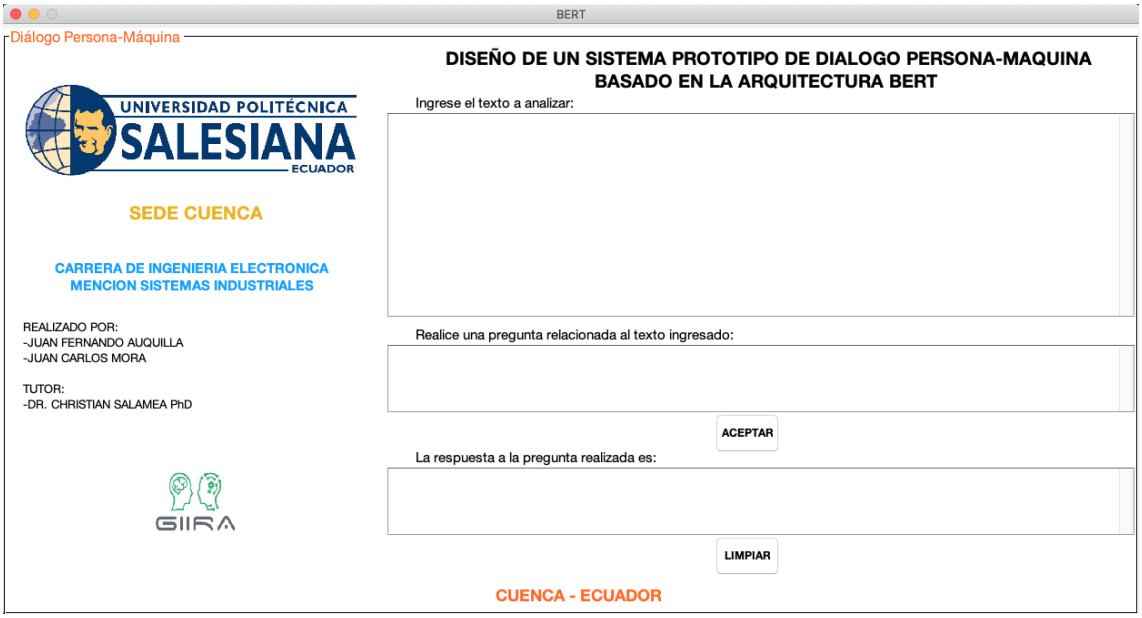

Figura 16.Interfaz realizada en el software Matlab.

En la figura anterior se tiene la interfaz realizada en el software Matlab para el respectivo funcionamiento de la arquitectura BERT. Se debe tener en cuenta que se trata de una interfaz de usuario intuitiva y de fácil uso, la misma debe ser clara y entendible. En este caso se ha implementado tres cuadros para ingresar y visualizar el respectivo texto en los diferentes casos, además de incluir dos botones, el primero para Aceptar la pregunta que se realiza y el segundo para Limpiar todos los cuadros e ingresar nueva información. En las figuras siguientes se tiene una descripción detallada de la función que cumple cada cuadro.

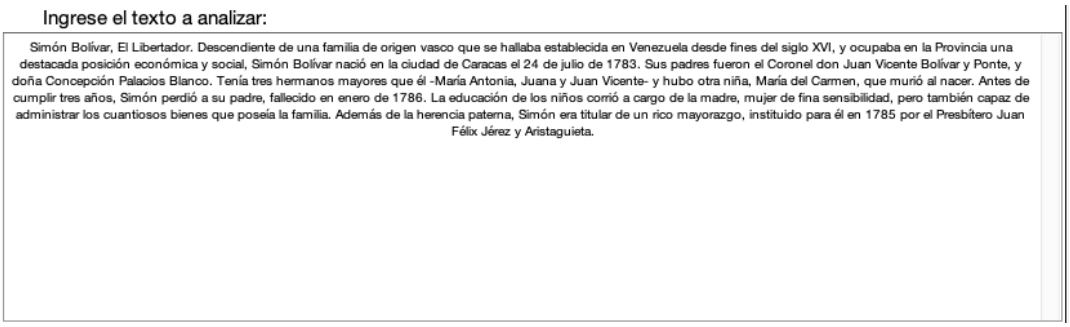

Figura 17. Cuadro para ingresar el texto a ser analizado en Matlab.

Como se puede visualizar en la figura anterior, se ingresó un texto a manera de ejemplo, el mismo que se trata de un párrafo de la biografía de Simón Bolívar; siendo éste el texto que el código analizará en base a la pregunta que se le realice para la obtención de una respuesta precisa basada en este párrafo.

De esta manera, una vez ingresado dicho párrafo a continuación se debe realizar la consulta, por lo cual, se tiene el cuadro intermedio para realizar la misma; y así se podrá analizar y buscar lo que se desea dentro del texto que se ingresó.

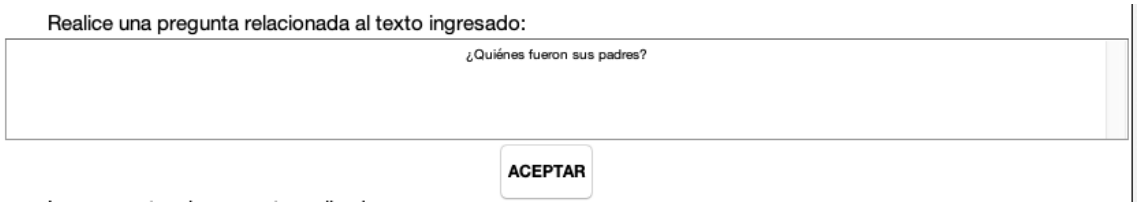

Figura 18.Cuadro para ingresar la pregunta en Matlab.

Y por último, se tiene el cuadro donde muestra la respuesta a la pregunta planteada. Como se detalló anteriormente, la arquitectura BERT al tener un análisis bidireccional del texto, permite obtener una respuesta concisa y precisa a la pregunta que se realiza, siempre y cuando sea relacionada al texto que se ingresó, como se puede observar a continuación.

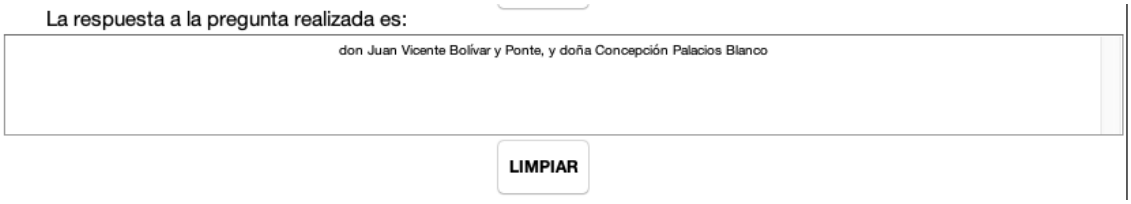

Figura 19.Cuadro para visualizar la respuesta a la pregunta en Matlab.

En este ejemplo planteado, se puede observar que el texto que se ingresó es un párrafo corto de la biografía de Simón Bolívar, El Libertador como un ejemplo. Al ingresar la pregunta, *"¿Quiénes fueron sus padres?",* la arquitectura BERT dio como respuesta *"don Juan Vicente Bolívar y Ponte, y doña Concepción Palacios Blanco"*, que según el texto que se ingresó, es la respuesta correspondiente a la pregunta que se realizó para el análisis.

Y al observar un fragmento del párrafo que se analizó para este ejemplo, se puede observar que *"Sus padres fueron el Coronel don Juan Vicente Bolívar y Ponte, y doña Concepción Palacios Blanco",* siendo esta la línea donde se encuentra la respuesta a dicha pregunta; comprobando que la misma es correcta.

5.2. Estructura Dataset SQUAD.

Para definir la estructura de como se forma el dataset SQuAD, se tiene en cuenta que esto se desarrolló a partir de conjuntos de datos ya existentes en el ámbito de comprensión lectora y respuesta a preguntas (QA) basados en los artículos disponibles en

Wikipedia, siendo SQuAD el que tiene una mayor capacidad de compresión lectora y respuesta a preguntas, a diferencia de otros conjuntos de datos.

Por lo tanto, tenemos los siguientes parámetros que definen la estructura del Dataset SQuAD:

- **Comprensión lectora**

Como enfoque para la compresión lectora por parte de SQuAD, se define que las preguntas requerirán dos tipos de razonamiento siendo estos uno de sentido común y uno a través de múltiples frases.

Para llegar a esto, se tuvo que pasar de un conjunto de datos basado en 600 preguntas reales y a su vez mejorado por un sistema que se basa en reglas y de un modelo de regresión logística.

- **Respuesta a preguntas de dominio abierto.**

El dataset tiene como objetivo responder a una pregunta a partir de varios documentos previamente recolectados, es decir, SQuAD utiliza pasajes que son extraídos de Wikipedia como una fuente de respuesta, siendo su tarea u objetivo la elección de frases, y en este caso viene a ser un tramo o parte específica de dicha frase.

Por lo tanto, las respuestas en SQuAD suelen incluir no entidades y llegan a ser frases mucho más largas, centrándose en las preguntas cuyas respuestas están ya de por si implícitas en el pasaje.

- **Recogida de preguntas y respuestas.**

Para el desarrollo de la recolección de preguntas y respuestas, se utilizó crowdworkers (grupos de trabajo) los mismos que son los encargados de crear y responder cinco preguntas sobre el contenido de dicho párrafo.

- **Análisis del conjunto de datos**

Una vez recopilado el conjunto de datos necesario para el desarrollo de SQuAD y las propiedades del mismo, se analiza las preguntas y respuestas del conjunto de desarrollo. De esta manera se llega a explorar: la diversidad de tipos de respuesta, la dificultad de las preguntas dependiendo del tipo de razonamiento que se necesita para contestarlas y el grado de divergencia sintáctica entre las frases de la pregunta y la respuesta.

- **Diversidad de respuestas.**

Al referirnos al término de diversidad de respuestas se da a entender que se tendrán diferentes tipos de respuestas, de esta manera se clasificará en dos tipos, siendo que estos

sean numéricas o no numéricas, donde se utiliza un análisis de constituyentes y etiquetas POS que son generadas por Stanford CoreNLP.

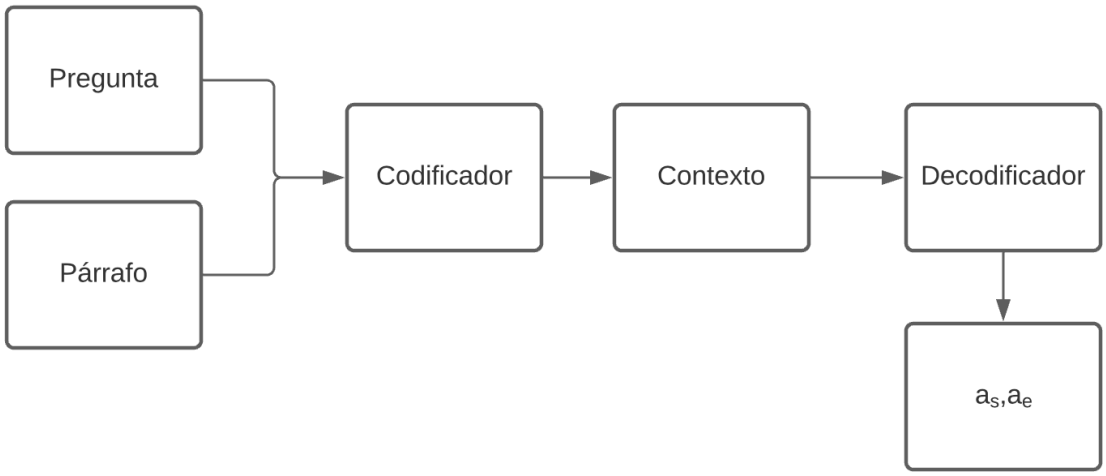

Figura 20.Funcionamiento SQuAD

En la Figura 20, se puede observar como el modelo de SQuAD tiene como variable de entrada la pregunta y el párrafo, de esta manera para cumplir con una secuencia que le permita con los datos ingresados obtener una respuesta precisa de lo que se quiere saber.

5.3. Descripción del experimento.

El experimento consiste en implementar un sistema prototipo de diálogo persona-máquina basado en la arquitectura BERT, el mismo que consiste en realizar una interfaz en dos softwares diferentes, siendo estos Anaconda Python y Matlab para la respectiva ejecución de la arquitectura antes mencionada. Cabe recalcar que para la interfaz en el software Anaconda Python se realiza directamente dentro del mismo y para la interfaz en el software de Matlab se necesita llamar a la arquitectura desde Anaconda Python.

Para dichos sistemas en los que se implementó es necesario la ayuda de herramientas como códigos externos y librerías con un fin específico; y así para que de esta forma se pueda trabajar de manera óptima en el desarrollo del sistema ya sea en cualquiera de los dos softwares que se utilizan.

5.3.1. Implementación del sistema

- **Interfaz gráfica en el software Anaconda Python**

Para la implementación del sistema en el software Anaconda Python, se utilizó el código de la arquitectura BERT en el notebook Jupyter, una vez que se ejecutó dicho

código y se tiene el correcto funcionamiento, se procedió a utilizar la librería tkinter, que viene a ser en otras palabras el python GUI (GUIde), esta librería permite el uso de un kit de herramientas denominadas “Tk” que es un conjunto multiplataforma de elementos de control gráfico o denominados “widgets”, los cuales permiten el desarrollo de la interfaz gráfica para un manejo fácil y amigable de la arquitectura BERT.

- **Interfaz gráfica en el software Matlab**

Para la implementación del sistema en el software Matlab se debe convertir el código de la arquitectura BERT del notebook Jupyter (extensión *.ipynb*) a Python (extensión .py), el archivo tiene por nombre prediccion.py. Una vez que se tiene este archivo, se debe usar una función (extensión .m) y una clase (extensión .m) en el software de Matlab que permite llamar al código de la arquitectura BERT que se encuentra en el software Python; específicamente se utilizan dos archivos llamados python.m (función) y conda.m (clase), ya que estos permiten el funcionamiento de la arquitectura BERT y las librerías necesarias para la ejecución, además que estos archivos se deben encontrar en el mismo entorno. A continuación, se procede a diseñar y programar los diferentes botones y cuadros de texto de la interfaz gráfica, llamada GUIDE en el software Matlab para ejecutar la arquitectura BERT; a través de un script se realiza la programación de todas las funciones que se agreguen a la interfaz gráfica.

5.3.2. Métricas de evaluación

Para evaluar el sistema se utilizó:

- El tiempo de análisis dependiendo el tamaño de texto ingresado.
- El tamaño máximo de texto que se puede ingresar para ser analizado.

5.3.3. Fase de entrenamiento

Para el entrenamiento, se realizó varias pruebas en base a lo siguiente.

1. La longitud del texto ingresado para analizar.
2. Tiempo de respuesta a la pregunta según la longitud de texto ingresada para el análisis.

5.4. Resultados y Discusión.

- Interfaz en el software Anaconda Python

Longitud de texto	Tiempo de respuesta
1 párrafo	7,4 segundos
2 párrafos	9 segundos
3 párrafos	9 segundos
6 párrafos	9 segundos
3 páginas	9 segundos
5 páginas	30 segundos
10 páginas	30 segundos

Tabla 1. Datos de evaluación de la interfaz en Python.

Para el análisis de cada longitud de texto de ingreso, se realizó 5 mediciones por cada longitud para obtener un tiempo promedio de la respuesta que tiene para dicho texto de ingreso.

- En el caso de la longitud de 1 párrafo, se desarrolló el análisis con 5 párrafos con una extensión de entre 3 y 15 líneas, obteniendo un tiempo promedio entre si de 7,4 segundos como tiempo de respuesta.
- En el caso de la longitud de 2 párrafos, se desarrolló el análisis con 5 párrafos con una extensión de entre 3 y 15 líneas, de igual manera se obtuvo un tiempo promedio de 9 segundos como tiempo de respuesta.
- En el caso de la longitud de 3 párrafos, se desarrolló el análisis con 5 párrafos con una extensión de entre 3 y 15 líneas, de igual manera se obtuvo un tiempo promedio de 9 segundos como tiempo de respuesta.
- En el caso de la longitud de 6 párrafos, se desarrolló el análisis con 5 textos, con una extensión de entre 3 y 15 líneas de cada párrafo; obteniendo un tiempo promedio de 9 segundos como tiempo de respuesta.

Como se puede observar en la tabla 1, en el caso de párrafos mayores a uno el tiempo de respuesta por parte del software terminó siendo el mismo para las longitudes de texto de 2, 3 y 6 párrafos, logrando una cierta estabilidad de tiempo promedio de respuesta en 9 segundos. Cabe acotar que se tomo un número entero, ya que los tiempos que se obtuvieron de manera individual en el momento de análisis sí variaban por milésimas de segundo.

En el caso de tener un ingreso mucho mayor en cuanto a la extensión del texto, es decir, páginas completas con un numero *n* de párrafos, en ella se realiza una sola prueba por cada métrica de evaluación.

- Interfaz en el software Matlab

Longitud de texto	Tiempo de respuesta
1 párrafo	39 segundos
2 párrafos	62 segundos
3 párrafos	66 segundos
6 párrafos	83 segundos
3 páginas	135 segundos
5 páginas	138 segundos
10 páginas	145 segundos

Tabla 2. Datos de evaluación de la interfaz en Matlab.

En el caso de la interfaz realiza en el software Matlab se evidencia claramente que tiempo de respuesta es mayor a la interfaz en el software Anaconda Python, esto se debe a que el software Matlab debe trabajar en conjunto con Anaconda, razón por la cual tiende a tomar un mayor tiempo para responder y también varía según la longitud del texto que se ingrese.

Una vez obtenidos los tiempos de respuesta en las diferentes interfaces, los resultados son los siguientes:

- Al tener un texto de aproximadamente 500 palabras, es decir, un texto no extenso relativamente; el tiempo de respuesta es inferior a los 10 segundos.
- Al ir aumentando la longitud de texto, se evidencia claramente que el tiempo de respuesta es mayor; ya que la arquitectura tiene un rango más grande por analizar para responder a la pregunta adecuadamente.
- No se recomienda enviar textos demasiado extensos, ya que puede tardar varios minutos en lograr responder a la pregunta que se formule; esto genera que la búsqueda o la respuesta que se espera se torne cansada al no tener un tiempo rápido.
- El tiempo de respuesta ideal es de un minuto aproximadamente; esto con el objetivo de que no se vuelva aburrido o tedioso para la persona que utiliza el sistema.

Los resultados anteriores también se ven afectados por las características que pueda ofrecer un equipo, en este caso los computadores utilizados tienen diferente procesador y antigüedad; razón por la cual en algunas máquinas se evidenciará un cambio en los tiempos de respuesta.

CAPÍTULO 6

6. CONCLUSIONES

Esta sección se encuentra dividida en dos partes, una relacionada con la arquitectura BERT y la otra con las interfaces en los diferentes softwares.

El modelo pre entrenado de BERT facilita totalmente el trabajo en la búsqueda bidireccional que realiza en un determinado texto para responder a la pregunta. Además, que a lo largo de este tiempo han venido trabajando en varios idiomas y se pudo aplicar al idioma español sin problema alguno. Una vez que se comprende el funcionamiento de la arquitectura y la base de datos, aplicar BERT en una interfaz gráfica es relativamente simple.

Con respecto al diseño de las interfaces gráficas no es un trabajo con un gran nivel de complejidad, en el mundo de la ingeniería los softwares de Anaconda Python y Matlab son muy utilizados, razón por la cual no se tuvo problema con esta parte.

Para ejecutar la arquitectura BERT en el software Matlab se debe realizar un determinado proceso y en cierto punto lo complicado en un principio, fue poder llamar desde este software al modelo de BERT. Recordar que el modelo se encuentra en Anaconda y deben trabajar conjuntamente.

Otro problema identificado fue al momento de realizar las pruebas para obtener los tiempos de respuesta y en esta parte vale recalcar que importa mucho las características de los equipos que se utilicen.

Además, al tener longitudes de texto muy extensas, el modelo de BERT tarda más tiempo en responder a la pregunta y más aún si se trabaja desde un tercer software (Matlab).

Finalmente, el uso de estas interfaces resulta de manera fácil y sencilla para la interacción de una persona que no tiene conocimiento a cerca de los temas abordados a lo largo de este trabajo.

7. ANEXOS

- Código utilizado en PYTHON.

*** REQUERIMIENTOS**

```
import os
import torch
import time
from torch.utils.data import DataLoader, RandomSampler, SequentialSampler

from transformers import (
    AutoTokenizer,
    AutoModelForQuestionAnswering,
    squad_convert_examples_to_features
)

from transformers.data.processors.squad import SquadResult, SquadV2Processor,
SquadExample
from transformers.data.metrics.squad_metrics import
compute_predictions_logits
```

***MODELO UTILIZADO**

```
def run_prediction(question_texts, context_text):
    """Setup function to compute predictions"""
    examples = []

    for i, question_text in enumerate(question_texts):
        example = SquadExample(
            qas_id=str(i),
            question_text=question_text,
            context_text=context_text,
            answer_text=None,
            start_position_character=None,
            title="Predict",
            is_impossible=False,
            answers=None,
        )

        examples.append(example)

    features, dataset = squad_convert_examples_to_features(
        examples=examples,
        tokenizer=tokenizer,
        max_seq_length=384,
        doc_stride=128,
        max_query_length=64,
        is_training=False,
        return_dataset="pt",
        threads=1,
    )

    eval_sampler = SequentialSampler(dataset)
    eval_dataloader = DataLoader(dataset, sampler=eval_sampler,
batch_size=10)
```

```
    all_results = []

    for batch in eval_dataloader:
        model.eval()
        batch = tuple(t.to(device) for t in batch)

        with torch.no_grad():
            inputs = {
                "input_ids": batch[0],
                "attention_mask": batch[1],
                "token_type_ids": batch[2],
            }

            example_indices = batch[3]

            outputs = model(**inputs)

            for i, example_index in enumerate(example_indices):
                eval_feature = features[example_index.item()]
                unique_id = int(eval_feature.unique_id)

                output = [to_list(output[i]) for output in outputs]

                start_logits, end_logits = output
                result = SquadResult(unique_id, start_logits, end_logits)
                all_results.append(result)

    output_prediction_file = "predictions.json"
    output_nbest_file = "nbest_predictions.json"
    output_null_log_odds_file = "null_predictions.json"

    predictions = compute_predictions_logits(
        examples,
        features,
        all_results,
        n_best_size,
        max_answer_length,
        do_lower_case,
        output_prediction_file,
        output_nbest_file,
        output_null_log_odds_file,
        False,  # verbose_logging
        True,  # version_2_with_negative
        null_score_diff_threshold,
        tokenizer,
    )

    return predictions

model_name_or_path = "mrm8488/distill-bert-base-spanish-wwm-cased-finetuned-
spa-squad2-es"

output_dir = ""

# Config
n_best_size = 1
max_answer_length = 30
do_lower_case = True
null_score_diff_threshold = 0.0
```

```
def to_list(tensor):
    return tensor.detach().cpu().tolist()

# Setup model
model_class, tokenizer_class = (AutoModelForQuestionAnswering, AutoTokenizer)
tokenizer = tokenizer_class.from_pretrained(
    model_name_or_path, do_lower_case=True)
model = model_class.from_pretrained(model_name_or_path)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model.to(device)

processor = SquadV2Processor()
```

*INTERFAZ GRAFICA Y FUNCIONAMIENTO DEL MODELO

```
import tkinter as tk
from tkinter import *
from PIL import ImageTk, Image

root = Tk()
root.title("BERT")
root.config(bg="white")
#Creacion Frames
frame1 = Frame(root)
frame1.config(bg="white")
frame1.pack(side=RIGHT, fill=X, padx=10, pady=10)

frame3 = Frame(root)
frame3.config(bg="white")
frame3.pack(side=BOTTOM, fill=X, pady=5)

frame2 = Frame(root)
frame2.config(bg="white")
frame2.pack(side=LEFT, fill=Y, padx=10, pady=10)


miLabel=Label(frame2,text="UNIVERSIDAD POLITECNICA
SALESIANA",bg="white",fg="black",font=("Times New Roma",18))
miLabel.grid(row=0,column=0, sticky="n", padx=10,pady=10)

miLabel2=Label(frame2,text="SEDE CUENCA",fg="black",bg="white",font=("Times
New Roma",14))
miLabel2.grid(row=1,column=0, sticky="n", padx=10,pady=10)

miLabel3=Label(frame2,text="CARRERA DE INGENIERIA
ELECTRONICA",bg="white",fg="black",font=("Times New Roma",14))
miLabel3.grid(row=2,column=0, sticky="n", padx=10,pady=10)

miLabel4=Label(frame1,text="Diseño de un sistema prototipo persona-maquina
basada en la arquitectura BERT",bg="white",fg="black",font=("Times New
Roma",16))
miLabel4.grid(row=0,column=1, sticky="ne", padx=10,pady=10)

Label4=Label(frame2, text="*REALIZADO POR: ",bg="white",font=("Times New
Roma",12))
Label4.grid(row=4,column=0, sticky= "w")
```

```
Label5=Label(frame2, text="-JUAN FERNANDO AUQUILLA ",bg="white",font=("Times
New Roma",12))
Label5.grid(row=5,column=0, sticky= "w")
Label6=Label(frame2, text="-JUAN CARLOS MORA ",bg="white",font=("Times New
Roma",12))
Label6.grid(row=6,column=0, sticky= "w")
Label7=Label(frame2, text="*MENCION: ",bg="white",font=("Times New Roma",12))
Label7.grid(row=8,column=0, sticky= "w")
Label8=Label(frame2, text="Sistemas Industriales ",bg="white",font=("Times
New Roma",12))
Label8.grid(row=9,column=0, sticky= "w")
Label8=Label(frame3, text="-CUENCA-ECUADOR- ",bg="white",font=("Times New
Roma",12))
Label8.grid(row=0,column=2, sticky= "ne")

#************************IMAGEN*********************
#Setting it up
img = tk.PhotoImage(file='logo-2.png')
#Displaying it
Logo= tk.Label(frame2,image=img).grid(row=3, column=0,sticky="n")
#****************************************************
textSalida=StringVar()
result=StringVar()

def codigoBoton():
    result=textEntrada.get(1.0, tk.END+"-1c")#guarda en una variable el
resultado que se ingreso en el texto
    context=textContext.get(1.0, tk.END+"-1c")
    tam=len(result)
   # print(result)


    questions = [result]

    questions

    # Run method
    predictions = run_prediction(questions, context)

    # Print results
    print("Results:")
    for i, key in enumerate(predictions.keys()):
      print(questions[i],predictions[key])

    textSalida.insert(tk.END, predictions[key])# Permite imprimir un
resultado en un TEXT diferente


def Borrar():
    textEntrada.delete("1.0","end")
    textSalida.delete("1.0","end")
#********************** TEXT-EDIT
*****************************************************
Label1=Label(frame1, text="Ingrese el texto: ",bg="white",font=("Times New
Roma",14))
Label1.grid(row=1,column=1, sticky= "w")

textContext=tk.Text(frame1, height=10)
textContext.grid(row=2,column=1, sticky= "w")
```

```
Label2=Label(frame1, text="Realice la pregunta: ",bg="white",font=("Times New
Roma",14))
Label2.grid(row=3,column=1, sticky= "w")

textEntrada=tk.Text(frame1, height=10)
textEntrada.grid(row=4,column=1, sticky= "w")


Label3=Label(frame1, text="La respuesta es: ",bg="white",font=("Times New
Roma",14))
Label3.grid(row=5,column=1, sticky= "w")

textSalida=tk.Text(frame1, height=10)
textSalida.grid(row=6,column=1, sticky= "w")


#**********************BOTONES*****************************************
**********

BotonEnvio=Button(frame1, text = "Aceptar", command=codigoBoton)#agrega un
boton a la interfaz
BotonEnvio.grid(row=11,column=1, sticky= "n")

botonBorrar=tk.Button(frame1, text = "Limpiar", command=Borrar)
botonBorrar.grid(row=12,column=1, sticky= "n")


root.mainloop()
```

- Código utilizado en MATLAB.

Código de la arquitectura BERT en Python (extensión .py)

***REQUERIMIENTOS**

```
import sys
import torch
import time
from torch.utils.data import DataLoader, RandomSampler, SequentialSampler

from transformers import (
AutoTokenizer,
AutoModelForQuestionAnswering,
squad_convert_examples_to_features
)

from transformers.data.processors.squad import SquadResult, SquadV2Processor,
SquadExample
from transformers.data.metrics.squad_metrics import compute_predictions_logits
```

***MODELO UTILIZADO**

```
def to_list(tensor):
	return tensor.detach().cpu().tolist()

def run_prediction(question_texts, context_text):
	"""Setup function to compute predictions"""
	examples = []

	for i, question_text in enumerate(question_texts):
		example = SquadExample(
		    qas_id=str(i),
		    question_text=question_text,
		    context_text=context_text,
		    answer_text=None,
		    start_position_character=None,
		    title="Predict",
		    is_impossible=False,
		    answers=None,
		)

		examples.append(example)
```

```
            features, dataset = squad_convert_examples_to_features(
                examples=examples,
                tokenizer=tokenizer,
                max_seq_length=384,
                doc_stride=128,
                max_query_length=64,
                is_training=False,
                return_dataset="pt",
                threads=1,
            )

            eval_sampler = SequentialSampler(dataset)
            eval_dataloader    =    DataLoader(dataset,    sampler=eval_sampler,
batch_size=10)

            all_results = []

            for batch in eval_dataloader:
                model.eval()
                batch = tuple(t.to(device) for t in batch)

                with torch.no_grad():
                    inputs = {
                        "input_ids": batch[0],
                        "attention_mask": batch[1],
                        "token_type_ids": batch[2],
                    }

                    example_indices = batch[3]

                    outputs = model(**inputs)

                    for i, example_index in enumerate(example_indices):
                        eval_feature = features[example_index.item()]
                        unique_id = int(eval_feature.unique_id)

                        output = [to_list(output[i]) for output in outputs]

                        start_logits, end_logits = output
```

```
                        result    =    SquadResult(unique_id,    start_logits,
end_logits)
                        all_results.append(result)

            output_prediction_file = "predictions.json"
            output_nbest_file = "nbest_predictions.json"
            output_null_log_odds_file = "null_predictions.json"

            predictions = compute_predictions_logits(
                examples,
                features,
                all_results,
                n_best_size,
                max_answer_length,
                do_lower_case,
                output_prediction_file,
                output_nbest_file,
                output_null_log_odds_file,
                False,  # verbose_logging
                True,  # version_2_with_negative
                null_score_diff_threshold,
                tokenizer,
            )

            return predictions

if __name__ == '__main__':
       model_name_or_path    =    "mrm8488/distill-bert-base-spanish-wwm-cased-
finetuned-spa-squad2-es"
       output_dir = ""

       # Config
       n_best_size = 1
       max_answer_length = 30
       do_lower_case = True
       null_score_diff_threshold = 0.0

       # Setup model
       model_class,    tokenizer_class    =    (AutoModelForQuestionAnswering,
AutoTokenizer)
```

```
	tokenizer	=	tokenizer_class.from_pretrained(model_name_or_path,
do_lower_case=True)
	model = model_class.from_pretrained(model_name_or_path)

	device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
	model.to(device)
	processor = SquadV2Processor()

	context = sys.argv[1]
	questions  = [sys.argv[2]]
	predictions = run_prediction(questions, context)

	# Print results
	print("Results:")
	for i, key in enumerate(predictions.keys()):
		print(predictions[key])
```

* INTERFAZ GRAFICA Y FUNCIONAMIENTO DEL MODELO

Código de la interfaz gráfica en MATLAB (extensión .m)

```
function varargout = BERT(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @BERT_OpeningFcn, ...
                   'gui_OutputFcn',  @BERT_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

function BERT_OpeningFcn(hObject, eventdata, handles, varargin)
axes(handles.axes1);
ups=imread('ups.jpg');
imshow(ups);

axes(handles.axes2);
giira=imread('giira.jpg');
imshow(giira);

handles.output = hObject;
```

```
guidata(hObject, handles);

function varargout = BERT_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;

function eliminar_Callback(hObject, eventdata, handles)
set(handles.entrada,'String','')
set(handles.pregunta,'String','')
set(handles.respuesta,'String','')

function aceptar_Callback(hObject, eventdata, handles)
entrada=get(handles.entrada,'String'); %guarda los datos de entrada
pregunta=get(handles.pregunta,'String');
set(handles.figure1, 'pointer', 'watch');
drawnow;
set(handles.respuesta,'String','');
prediccion=python('prediccion.py',entrada,pregunta);
index=strfind(prediccion,'Results');
set(handles.figure1, 'pointer', 'arrow');
s=
set(handles.respuesta,'String',prediccion(index+9:length(prediccion)-
1)); %muestra en la salida

function entrada_Callback(hObject, eventdata, handles)

function entrada_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function pregunta_Callback(hObject, eventdata, handles)

function pregunta_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function respuesta_Callback(hObject, eventdata, handles)

function respuesta_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Código de la clase conda (extensión .m)

```
classdef conda
properties
end

methods
end

methods (Static)
```

```
    function varargout = conda(varargin)

        varargout = {};

        if nargin == 2
            if strcmp(varargin{1},'env') && strcmp(varargin{2},'list')
                conda.getenv();
                return
            elseif strcmp(varargin{1},'activate')
                conda.setenv(varargin{2})
                return
            else
                error('Unknown argument syntax')
            end
        else
            error('Unknown argument syntax')
        end

    end

    function init()

        try
            getpref('condalab','base_path');
        catch
            str = input('Enter the path to your conda installation:
\n','s');
            str = strrep(str,[filesep 'conda'],'');
            setpref('condalab','base_path',str)
        end

         conda.addBaseCondaPath()

    end

    function addBaseCondaPath()

        try
            condalab_base_path = getpref('condalab','base_path');
        catch
            conda.init()
            condalab_base_path = getpref('condalab','base_path');
        end

        [e,o]=system('which conda');
        if e == 0
            %conda está en algún lugar
            return
        end

        P = strsplit(getenv('PATH'),pathsep);
        add_to_path = true;
        for i = 1:length(P)
            if strcmp(P{i},condalab_base_path)
                add_to_path = false;
            end
        end
        if add_to_path
```

```
            setenv('PATH',[condalab_base_path pathsep getenv('PATH')
]);
        end

    end

    function varargout = getenv()
        conda.addBaseCondaPath;
        [~,envs] = system('conda env list');
        envs = strsplit(envs,'\n');

        p = strsplit(getenv('PATH'),pathsep);

        % quite el asterisco porque siempre está al inicio (root)
        for i = length(envs):-1:1
            envs{i} = strrep(envs{i},'*',' ');
            if isempty(envs{i})
                continue
            end
            if strcmp(envs{i}(1),'#')
                continue
            end

            this_env = strsplit(envs{i});
            env_names{i} = this_env{1};
            env_paths{i} = this_env{2};

            active_path = 0;
            for j = 1:length(env_paths)
                this_env_path = [env_paths{j} filesep 'bin'];
                if any(strcmp(this_env_path,p))
                    active_path = j;
                end
            end
         end

        if nargout
            varargout{1} = env_names;
            varargout{2} = env_paths;
            varargout{3} = active_path;
        else
            fprintf('\n')
            for i = 1:length(env_names)
                if isempty(env_names{i})
                    continue
                end
                if active_path == i
                    disp(['*' env_names{i} '      ' env_paths{i}])
                else
                    disp([env_names{i} '      ' env_paths{i}])
                end
            end
        end
    end

    function setenv(env)
        conda.addBaseCondaPath;
        [~,envs] = system(['conda env list']);
        envs = strsplit(envs,'\n');
```

```
        [env_names, env_paths] = conda.getenv;

        %revisa que los entornos existan en la lista
        assert(any(strcmp(env_names,env)), 'env you want to activate
is not valid')

        p = getenv('PATH');
        %elimina cada conda entorno del camino
        p = strsplit(p,pathsep);
        rm_this = false(length(p),1);
        for i = 1:length(p)
            %elimina "bin" del final
            this_path = strtrim(strrep(p{i}, [filesep 'bin'],''));
            if any(strcmp(this_path,env_paths))
                rm_this(i) = true;
            end
        end
        p(rm_this) = [];

        %agregar el entorno de la vía que deseas
        this_env_path = [env_paths{strcmp(env_names,env)} filesep
'bin'];
        p = [this_env_path p];
        p = strjoin(p,pathsep);
        p = [p pathsep getpref('condalab','base_path')];

        setenv('PATH', p);

    end

    %pregunta al interpretador de python donde está ubicada
    function test()
        system(['python ' fileparts(which(mfilename)) filesep
'test.py']);
    end
end
end
```

Código de la función Python (extensión .m)

```
function [result, status] = python(varargin)

if nargin > 0
    [varargin{:}] = convertStringsToChars(varargin{:});
end

cmdString = '';

for i = 1:nargin
    thisArg = varargin{i};
    if ~ischar(thisArg)
        error(message('MATLAB:python:InputsMustBeStrings'));
    end
    if i==1
        if exist(thisArg, 'file')==2
            if isempty(dir(thisArg))
                thisArg = which(thisArg);
            end
```

```
        else
            error(message('MATLAB:python:FileNotFound', thisArg));
        end
    end

    if isempty(thisArg) || any(thisArg == ' ')
        thisArg = ['"', thisArg, '"'];
    end

    cmdString = [cmdString, ' ', thisArg];
end

if isempty(cmdString)
    error(message('MATLAB:python:NopythonCommand'));
end

if ~ispc || isdeployed
    if ispc
        checkCMDString = 'python -v';
    else
        checkCMDString = 'which python';
    end
    [cmdStatus, ~] = system(checkCMDString);
    if cmdStatus ~=0
        error(message('MATLAB:python:NoExecutable'));
    end
end

cmdString = ['python' cmdString];
if ispc && ~isdeployed
    pythonInst = 'opt\anaconda3\';
    cmdString = ['set PATH=',pythonInst, ';%PATH%&' cmdString];
end

[status, result] = system(cmdString);

if nargout < 2 && status~=0
    error(message('MATLAB:python:ExecutionError', result, cmdString));
end
```

8. REFERENCIAS

[1] Mariani, Joseph; Francopoulo, Gil; Paroubek, Patrick (2019), «The PLN4PLN Corpus (I): 50 Years of Publication Collaboration and Citation in Speech and Language Processing», Frontiers in Research Metrics and Analytics

[2] [url=https://www.frontiersin.org/research-topics/4817/artificial-neural-networks-as-models-of-neural-information-processing «Artificial Neural Networks as Models of Neural Information Processing | Frontiers Research Topic»] (en inglés).

[3] Mikolov, Tomas; Sutskever, Ilya; Chen, Kai; Corrado, Greg; Dean, Jeffrey (2013). «Distributed Representations of Words and Phrases and their Compositionality». arXiv:1310.4546

[4] Sepp Hochreiter; Jürgen Schmidhuber (1997). "Long short-term memory". Neural Computation. 9 (8): 1735–1780. doi:10.1162/neco.1997.9.8.1735

[5] Martínez, Merche (5 de noviembre de 2019). «Google BERT: actualización para entender el lenguaje natural». Human Level.

[6] Casilla, D. (2020). Diseño, implementación y evaluación de un sistema de diálogo persona-máquina en un espacio inteligente (Master). Universidad de Alcalá (UAH) Madrid.

[7] Alexander Gelbukh (2). Procesamiento de Lenguaje Natural y sus Aplicaciones. Komputer Sapiens, 1.

[8] Hernández, M. B., & Gómez, J. M. (2013). Aplicaciones de Procesamiento de Lenguaje Natural. Revista Politécnica, 32. Recuperado a partir de https://revistapolitecnica.epn.edu.ec/ojs2/index.php/revista_politecnica2/article/view/32

[9] Cortez Vásquez, A., Vega huerta, H., Pariona Quispe, J., & Huayna, A. M. (2009). Procesamiento de lenguaje natural. Revista De investigación De Sistemas E Informática, 6(2), 45–54. Recuperado a partir de https://revistasinvestigacion.unmsm.edu.pe/index.php/sistem/article/view/5923

[10] Casilla, D. (2020). Diseño, implementación y evaluación de un sistema de diálogo persona-máquina en un espacio inteligente (Master). Universidad de Alcalá (UAH) Madrid.

[11] Salamea, C. (2018). Diseño y Evaluación de Técnicas de Reconocimiento de idioma mediante la Fusión de Información Fonotáctica y Acústica. (Ingeniero). Universidad Politécnica de Madrid, Escuela técnica superior técnica de ingenieros de telecomunicación.

[12] R. Feldman and J. Sanger, "The text mining handbook", Cambridge University Press, December 11, 2006. [Also Online]. Available:www.safaribooksonline.com

[13] D. Bikel and I. Zitouni, "Multilingual natural language processing applications: from theory to practice", IBM Press, May 10, 2012. [Also Online].Available:www.safaribooksonline.com

[14] C. D. Manning, Prabhakar Raghavan, and Hin-rich Schütze, "Introduction to information retrieval",Cambridge University Press, July 7, 2008. [Also On-line]. Available: www.safaribooksonline.com

[15] Bandyopadhyay, S. Naskar and A. Ekbal, "Emerg-ing applications of natural language processing", IGIGlobal, October 31, 2012. [Also Online]. Available:www.safaribooksonline.com

[16] Wieringa, Peter & Stassen, Henk. Human-machine systems. Transactions of The Institute of Measurement and Control - TRANS INST MEASURE CONTROL. 21. 139-150. 10.1177/014233129902100402.

[17] Zeng, Z., Wang, J.: Advances in Neural Network Research and Applications. Springer- Verlag, Berlin, Heidelberg (2010). DOI: 10.1007/978-3-642-12990-2

[18] Anastassiou, G.A.: Intelligent Systems II: Complete Approximation by Neural Network Operators. Springer, Switzerland (2016). DOI: 10.1007/978-3-319-20505-2.

[19] Da Silva, I.N., Hernane Spatti, D., Andrade Flauzino, R., Liboni, L.H.B., dos Reis Alves, S.F.: Artificial Neural Networks. Springer, Switzerland (2017). DOI: 10.1007/978-3-319-43162-8.

[20] Boyd, R. L., & Pennebaker, J. W. (2017). Language-based personality: a new approach to personality in a digital world. Current opinion in behavioral sciences, 18, 63-68.

[21] Mark, D. M., Turk, A. G., Burenhult, N., & Stea, D. (Eds.). (2011). Landscape in language: transdisciplinary perspectives (Vol. 4). John Benjamins Publishing.

[22] Gharehchopogh, F. S., & Khalifelu, Z. A. (2011, October). Analysis and evaluation of unstructured data: text mining versus natural language processing. In 2011 5th International Conference on Application of Information and Communication Technologies (AICT) (pp. 1-4). IEEE.

[23] Matuszek, C. (2018, July). Grounded language learning: Where robotics and nlp meet (invited talk). In Proceedings of the International Joint Conference on Artificial Intelligence.

[24] Cambria, E., & White, B. (2014). Jumping NLP curves: A review of natural language processing research. IEEE Computational intelligence magazine, 9(2), 48-57.

[25] Zhao, W., Peng, H., Eger, S., Cambria, E., & Yang, M. (2019). Towards scalable and reliable capsule networks for challenging NLP applications. arXiv preprint arXiv:1906.02829.

[26] Hung, V. (2014). Context and NLP. In Context in Computing (pp. 143-154). Springer, New York, NY.

[27] Etaiwi, W., & Naymat, G. (2017). The impact of applying different preprocessing steps on review spam detection. Procedia computer science, 113, 273-279.

[28] Kaufmann, M., & Kalita, J. (2010, January). Syntactic normalization of twitter messages. In International conference on natural language processing, Kharagpur, India (Vol. 16).

[29] Kannan, S., Gurusamy, V., Vijayarani, S., Ilamathi, J., Nithya, M., Kannan, S., & Gurusamy, V. (2014). Preprocessing techniques for text mining. International Journal of Computer Science & Communication Networks, 5(1), 7-16.

[30] Saif, H., Fernández, M., He, Y., & Alani, H. (2014). On stopwords, filtering and data sparsity for sentiment analysis of twitter.

[31] Jivani, A. G. (2011). A comparative study of stemming algorithms. Int. J. Comp. Tech. Appl, 2(6), 1930-1938.

[32] Halácsy, P., & Trón, V. (2006, September). Benefits of deep NLP-based Lemmatization for Information Retrieval. In CLEF (Working Notes).

[33] Etaiwi, W., & Naymat, G. (2017). The impact of applying different preprocessing steps on review spam detection. Procedia computer science, 113, 273-279.

[34] Li, J., Chen, X., Hovy, E., & Jurafsky, D. (2015). Visualizing and understanding neural models in nlp. arXiv preprint arXiv:1506.01066.

[35] Rodríguez, P., Bautista, M. A., Gonzalez, J., & Escalera, S. (2018). Beyond one-hot encoding: Lower dimensional target embedding. Image and Vision Computing, 75, 21-31.

[36] Zhang, Y., Jin, R., & Zhou, Z. H. (2010). Understanding bag-of-words model: a statistical framework. International Journal of Machine Learning and Cybernetics, 1(1-4), 43-52.

[37] Church, K. W. (2017). Word2Vec. Natural Language Engineering, 23(1), 155-162.
[38] Goldberg, Y., & Levy, O. (2014). word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. arXiv preprint arXiv:1402.3722.
[39] McCormick, C. (2016). Word2vec tutorial-the skip-gram model. Apr-2016.[Online]. Available: http://mccormickml. com/2016/04/19/word2vec-tutorial-the-skip-gram-model.
[40] Jang, B., Kim, I., & Kim, J. W. (2019). Word2vec convolutional neural networks for classification of news articles and tweets. PloS one, 14(8), e0220976.
[41] Tetko, I. V., Karpov, P., Van Deursen, R., & Godin, G. (2020). State-of-the-art augmented NLP transformer models for direct and single-step retrosynthesis. Nature communications, 11(1), 1-11.
[42] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. arXiv preprint arXiv:2005.11401.
[43] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).
[44] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.
[45] Konstantinov, A., Moshkin, V., & Yarushkina, N. (2020, December). Approach to the Use of Language Models BERT and Word2vec in Sentiment Analysis of Social Network Texts. In International Scientific and Practical Conference in Control Engineering and Decision Making (pp. 462-473). Springer, Cham.
[46] Howe, R. (2012). The use of Fuck: A sociolinguistic approach to the usage of Fuck in the BNC and Blog Authorship Corpus. Eastern Michigan University.
[47] Mukherjee, A., Mukhopadhyay, S., Panigrahi, P. K., & Goswami, S. (2019, October). Utilization of Oversampling for multiclass sentiment analysis on Amazon Review Dataset. In 2019 IEEE 10th International Conference on Awareness Science and Technology (iCAST) (pp. 1-6). IEEE.
[48] Panayotov, V., Chen, G., Povey, D., & Khudanpur, S. (2015, April). Librispeech: an asr corpus based on public domain audio books. In 2015 IEEE international conference on acoustics, speech and signal processing (ICASSP) (pp. 5206-5210). IEEE.

[49] Salehinejad, H., Wang, Z., & Valaee, S. (2019, November). Ising dropout with node grouping for training and compression of deep neural networks. In 2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP) (pp. 1-5). IEEE.

[50] Rondeau, M. A., & Hazen, T. J. (2018, July). Systematic error analysis of the Stanford question answering dataset. In Proceedings of the Workshop on Machine Reading for Question Answering (pp. 12-20).

[51] Mendoza, L. E., Peña, J., Muñoz-Bedoya, L. A., & Velandia-Villamizar, H. J. (2013). Procesamiento de señales provenientes del habla subvocal usando Wavelet Packet y Redes Neuronales. TecnoLógicas, 655-667.

[52] López Zorrilla, A. (2017). Introducción al procesamiento del habla mediante técnicas de deep learning.

[53] Esquerra, I. (2006). Síntesis de habla emocional por selección de unidades. Proceedings of the IV Jornadas en Tecnologa del Habla, 161-165.

[54] Barrobés, H., & Ruiz, M. (2017). Síntesis del habla.

[55] Milone, D. H. (2005). Reconocimiento automático del habla con redes neuronales artificiales. Ciencia, Docencia y Tecnología, 16(31), 261-322.

[56] González Sigüenza, Beatriz. «BATVOX: sistema automático de reconocimiento de locutor». Estudios de fonética experimental, [en línia], 2008, Vol. 17, p. 303-16, https://www.raco.cat/index.php/EFE/article/view/140074 [Consulta: 9-11-2020].

[57] Kompe, R., & Kompe, R. (1997). Prosody in speech understanding systems (Vol. 1307). Berlin: Springer.

[58] Greenberg, S. (1996, July). Understanding speech understanding: Towards a unified theory of speech perception. In Proceedings of the ESCA Tutorial and Advanced Research Workshop on the Auditory Basis of Speech Perception (pp. 1-8). Keele, England.

[59] Devling, J., Chang, M., Lee, k., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [Ebook] (pp. 1-6). Retrieved from https://www.aclweb.org/anthology/N19-1423.pdf

[60] E. Levin, R. Pieraccini and W. Eckert, "A stochastic model of human-machine interaction for learning dialog strategies," in IEEE Transactions on Speech and Audio Processing, vol. 8, no. 1, pp. 11-23, Jan. 2000, doi: 10.1109/89.817450.

[61] *Hongshen Chen, Xiaorui Liu, Dawei Yin, and Jiliang Tang. 2017. A Survey on Dialogue Systems: Recent Advances and New Frontiers. SIGKDD Explor. Newsl. 19, 2 (December 2017), 25–35. DOI:https://doi.org/10.1145/3166054.3166058*

[62] Chen, H., Liu, X., Yin, D., & Tang, J. (2017). A Survey on Dialogue Systems: Recent Advances and New Frontiers. *SIGKDD Explor. Newsl., 19(2), 25–35.*

[63] Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. 1992. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479.

[64] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large anno- tated corpus for learning natural language inference. In *EMPLN*. Association for Computational Linguis- tics.

[65] Andrew M Dai and Quoc V Le. 2015. Semi-supervised sequence learning. In *Advances in neural informa- tion processing systems*, pages 3079–3087.

[66] William B Dolan and Chris Brockett. 2005. Automati- cally constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

[67] Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *ACL*. Association for Computational Linguistics

[68] Erik F Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *CoNLL*.

[69] Wilson L Taylor. 1953. Cloze procedure: A new tool for measuring readability. *Journalism Bulletin*, 30(4):415–433.

[70] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language under- standing with unsupervised learning. Technical re- port, OpenAI.

[71] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Pro- cessing Systems*, pages 6000–6010.

[72] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018a. Deep contextualized word rep- resentations. In *NAACL*.