UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

FACULTAD DE INGENIERÍA ELÉCTRICA

“DESARROLLO DE UNA APLICACIÓN ANDROID DE TRADUCCIÓN ESPAÑOL – PURÉPECHA”

TESIS

Autor: Jaime Kharim Calderón Espinosa

Tutor: MC. Miguel Ángel García Trillo

Morelia, Michoacán Diciembre, 2022

Agradecimientos

Agradezco a mi familia por brindarme su apoyo, paciencia y motivación durante toda la carrera, además de formarme en la persona que soy.

Agradezco a mis compañeros por apoyarme en el camino y brindarme su amistad.

También agradezco a mis profesores por ayudarme en mi desarrollo académico. Por brindarme enseñanzas, conocimiento y experiencia.

Resumen

Este proyecto tiene como objetivo principal el desarrollo de una aplicación para dispositivos móviles con sistema operativo Android, destinada a la traducción de texto del español al purépecha. Para lograr esto, se emplearán técnicas de machine learning y se utilizará un modelo seq2seq, reconocido como uno de los enfoques más efectivos en el procesamiento del lenguaje natural, especialmente en el ámbito de la traducción automática.

La motivación detrás de esta iniciativa radica en la importancia de promover la inclusión y brindar apoyo a un sector específico de la sociedad: las personas que hablan la lengua purépecha. Con frecuencia, el idioma se convierte en una barrera significativa en la integración social, ya que gran parte de la información cotidiana y los recursos están disponibles únicamente en español. A través de esta aplicación, se pretende eliminar esta barrera lingüística y proporcionar a los hablantes de purépecha una herramienta práctica que les permita acceder a la información y los servicios disponibles en la sociedad.

Es importante destacar que este proyecto aborda una necesidad poco atendida, ya que las lenguas minoritarias, como el purépecha, a menudo carecen de soluciones tecnológicas específicas. Al concentrarse en el desarrollo de una aplicación de traducción que se adapte a las particularidades del purépecha, se busca no solo facilitar la comunicación y el acceso a recursos para los hablantes de esta lengua, sino también promover su preservación y valoración cultural.

Además, es relevante mencionar que el enfoque geográfico de este proyecto se centra en el estado de Michoacán, México, donde el purépecha es una lengua originaria arraigada en la historia y la identidad de la región. Al trabajar en esta localidad, se pretende establecer una conexión más estrecha con la comunidad purépecha y garantizar que la aplicación sea relevante y efectiva en su contexto específico.

En resumen, el desarrollo de esta aplicación de traducción del español al purépecha no solo busca facilitar la comunicación y el acceso a la información para los hablantes de esta lengua, sino también fomentar la inclusión social, promover la valoración de las lenguas minoritarias y contribuir al enriquecimiento cultural de la comunidad purépecha en el estado de Michoacán.

Palabras clave: Android, traducción automática, inteligencia artificial, aplicación, lengua, neurona, lstm, red neuronal recurrente, integración social.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1 Introducción

La traducción automática (TA) es un campo en constante evolución que se centra en el desarrollo de sistemas de software capaces de traducir automáticamente textos de un lenguaje natural a otro. Es un proceso complejo que implica el uso de algoritmos y técnicas computacionales para analizar y comprender el significado y la estructura de un texto en el idioma de origen, y luego generar una traducción coherente en el idioma de destino.

Figura 1: Ilustración traducción automática.

A lo largo de la historia, la traducción ha sido realizada principalmente por traductores humanos, quienes aplican su conocimiento y experiencia lingüística para capturar el significado y la intención del texto original y expresarlos de manera efectiva en el idioma de destino. Sin embargo, con el advenimiento de la tecnología de la información y el aumento de la demanda de traducciones rápidas y precisas, la traducción automática ha ganado protagonismo.

La traducción automática ofrece una serie de ventajas, como la capacidad de procesar grandes volúmenes de texto en poco tiempo y la posibilidad de realizar traducciones en tiempo real. Además, los sistemas de traducción automática pueden adaptarse y mejorar a medida que se les proporciona más información y se los entrena con conjuntos de datos específicos.

Al procesar cualquier traducción, humana o automática, el significado del texto en el idioma original (origen) se debe restaurar totalmente en el de destino, es decir, en la traducción. Aunque en apariencia parezca sencillo, es mucho más complejo. La traducción no es una mera sustitución de una palabra por otra. Un traductor debe interpretar y analizar todos los elementos del texto y saber cómo influyen unas palabras en otras. Para ello se necesitan amplios conocimientos de gramática, sintaxis (estructura de las oraciones), semántica (significados), etc., de los idiomas de origen y de destino.

Tanto la traducción humana como la automática tienen sus propios desafíos. Por ejemplo, dos traductores individuales no pueden producir traducciones idénticas del mismo texto en el mismo par de idiomas, y es posible que se requieran varias rondas de revisiones para lograr la satisfacción del usuario. Pero el mayor desafío reside en cómo se pueden producir traducciones de calidad aptas para ser publicadas mediante la traducción automática.

En el caso específico de la traducción automática entre el inglés y el español, dos idiomas ampliamente utilizados y estudiados, se han logrado avances significativos. Sin embargo, la traducción automática entre idiomas menos comunes o en vías de extinción presenta desafíos adicionales. Un ejemplo de esto es el purépecha, una lengua indígena nacional de la familia lingüística tarasca.

Existen diversos enfoques para afrontar el problema de la traducción automática, como métodos estadísticos o traductores basados en reglas, pero en este proyecto se tratará únicamente la traducción basada en redes neuronales.

Por otro lado, tenemos nuestros idiomas objetivo: el purépecha y el español.

El purépecha, también conocido como tarasco, es hablado por aproximadamente 128,344 personas en 19 municipios del estado de Michoacán, México. Es considerado una lengua aislada, ya que no se ha establecido una relación genealógica clara con otros idiomas conocidos. El purépecha tiene una rica historia y cultura asociada, y ha desempeñado un papel importante en la región de Michoacán durante siglos.

Figura 2: El pueblo purépecha.

La ubicación principal de los hablantes de purépecha se encuentra en el noroeste de Michoacán, en las áreas de las ciudades de Uruapan y Pátzcuaro. Estas comunidades han conservado y transmitido el idioma a través de generaciones, pero también se enfrentan a desafíos para su preservación y revitalización. Aunque actualmente cuenta con una población significativa de hablantes, el purépecha está en riesgo de desaparición a largo plazo, lo que lo convierte en una lengua en peligro.

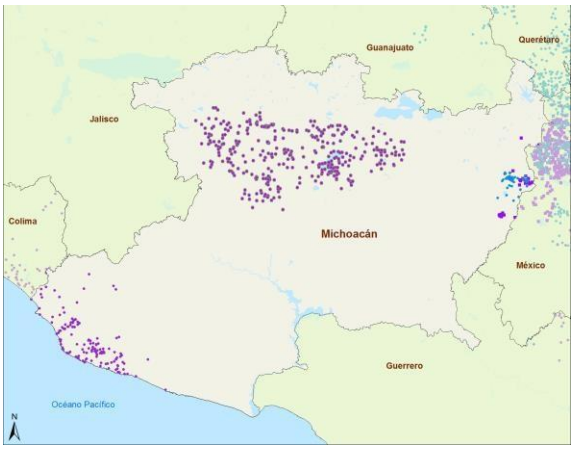

Figura 3: Regiones de Michoacán con población purépecha.

La cultura purépecha tuvo su inicio alrededor del año 1200 d.C. y su apogeo ocurrió hasta el año 1600. Durante este período, el pueblo purépecha estableció un gobierno monárquico y teocrático. Desarrollaron una sociedad altamente organizada, con avances significativos en la agricultura, la arquitectura, la cerámica y otras áreas. Los purépechas también son conocidos por su arte y artesanía distintivos, como las famosas "catrinas" y las máscaras de madera tallada.

En cuanto a la terminología utilizada para referirse a este grupo étnico, existe un debate en curso. Históricamente, se les ha llamado "tarascos", pero en las últimas décadas ha habido un esfuerzo por utilizar el término "purépechas", que es el autónimo preferido por muchos miembros de la comunidad. Aunque ambas denominaciones se utilizan, es importante reconocer y respetar la preferencia de autodenominación de los purépechas.

El presente proyecto se centrará en la traducción automática basada en redes neuronales para el par de idiomas purépecha-español. Las redes neuronales son modelos de aprendizaje automático inspirados en el funcionamiento del cerebro humano, capaces de capturar patrones complejos en los datos de entrenamiento y generar traducciones más precisas y naturales.

1.2 Objetivos

Los objetivos principales de este trabajo de investigación se describen a continuación:

- Desarrollar una aplicación móvil accesible y fácil de usar que permita a las personas que hablan purépecha traducir su lengua nativa al español.
- Investigar y recopilar una base de datos exhaustiva de vocabulario, frases comunes y expresiones en purépecha para asegurar una traducción precisa y completa en la aplicación.
- Realizar pruebas y evaluaciones exhaustivas de la aplicación con hablantes nativos de purépecha para garantizar su usabilidad, eficacia y adecuación cultural.
- Estudiar el impacto de la aplicación en la comunidad purépecha, tanto en términos de acceso a la información como en la preservación de la lengua y la cultura indígena.
- Analizar los desafíos tecnológicos, sociales y culturales asociados con el desarrollo y la implementación de la aplicación, y proponer recomendaciones para superarlos.
- Promover la conciencia y la valoración de las lenguas indígenas, en particular la lengua purépecha, entre la sociedad en general, destacando su importancia histórica y su contribución a la diversidad cultural.

1.3 Justificación

En la actualidad, se han desarrollado numerosos proyectos relacionados con la traducción automática. Sin embargo, la mayoría de estos proyectos se centran en idiomas ampliamente reconocidos como el inglés o el francés, y su aplicación se limita principalmente a entornos de escritorio. Por lo tanto, existe una brecha significativa en términos de atención y recursos dedicados a la traducción automática para otros idiomas, incluyendo aquellos que son de vital importancia en contextos regionales específicos, como es el caso del estado de Michoacán.

Este proyecto de tesis tiene como objetivo abordar dicha brecha y centrarse en la traducción automática para un idioma específico hablado en Michoacán. Al hacerlo, se busca no solo promover la inclusión de las personas de este origen, sino también reconocer y valorar la importancia cultural y lingüística de esta comunidad.

Al desarrollar un sistema de traducción automática específico para el idioma hablado en Michoacán, se espera facilitar la comunicación y el acceso a la información para las personas que lo utilizan como su lengua materna. Esto tiene el potencial de tener un impacto positivo en diversos ámbitos, como la educación, la atención médica, la administración pública y el desarrollo económico de la región.

Además, al ampliar el alcance de la traducción automática más allá de los idiomas convencionales, se contribuirá al avance de la tecnología lingüística y se explorarán nuevas posibilidades en el campo de la traducción automática. Esto podría sentar las bases para futuras investigaciones y proyectos que beneficien a otras comunidades con idiomas menos representados.

En resumen, este proyecto de tesis tiene como objetivo llenar un vacío importante en el campo de la traducción automática al enfocarse en un idioma específico hablado en Michoacán. Al hacerlo, se espera fomentar la inclusión, reconocer la importancia cultural y lingüística de esta comunidad y abrir nuevas posibilidades en el campo de la tecnología lingüística.

Capítulo 2

Marco Teórico

2.1 Redes neuronales

¿Qué es una red neuronal?

Una red neuronal es un método de la inteligencia artificial que enseña a las computadoras a procesar datos de una manera que está inspirada en la forma en que lo hace el cerebro humano. Se trata de un tipo de proceso de machine learning llamado aprendizaje profundo, que utiliza los nodos o las neuronas interconectados en una estructura de capas que se parece al cerebro humano. Crea un sistema adaptable que las computadoras utilizan para aprender de sus errores y mejorar continuamente. De esta forma, las redes neuronales artificiales intentan resolver problemas complicados, como la realización de resúmenes de documentos o el reconocimiento de rostros, con mayor precisión.

¿Qué es el aprendizaje profundo en el contexto de las redes neuronales?

La inteligencia artificial es el campo de las ciencias de la computación que investiga métodos para dar a las máquinas la capacidad de realizar tareas que requieren inteligencia humana. El machine learning es una técnica de inteligencia artificial que otorga a las computadoras acceso a conjuntos de datos muy grandes y les enseña a aprender de estos datos. El software de machine learning encuentra patrones en los datos existentes y los aplica a datos nuevos para tomar decisiones inteligentes. El aprendizaje profundo es un subconjunto del machine learning que utiliza las redes de aprendizaje profundo para procesar los datos.

¿Cómo funcionan las redes neuronales?

El cerebro humano es lo que inspira la arquitectura de las redes neuronales. Las células del cerebro humano, llamadas neuronas, forman una red compleja y con un alto nivel de interconexión y se envían señales eléctricas entre sí para ayudar a los humanos a procesar la información. De manera similar, una red neuronal artificial está formada por neuronas artificiales que trabajan juntas para resolver un problema. Las neuronas artificiales son módulos de software, llamados nodos, y las redes neuronales artificiales son programas de software o algoritmos que, en esencia, utilizan sistemas informáticos para resolver cálculos matemáticos.

Arquitectura de una red neuronal simple

Una red neuronal básica tiene neuronas artificiales interconectadas en tres capas:

- Capa de entrada

La información del mundo exterior entra en la red neuronal artificial desde la capa de entrada. Los nodos de entrada procesan los datos, los analizan o los clasifican y los pasan a la siguiente capa.

- Capa oculta

Las capas ocultas toman su entrada de la capa de entrada o de otras capas ocultas. Las redes neuronales artificiales pueden tener una gran cantidad de capas ocultas. Cada capa oculta analiza la salida de la capa anterior, la procesa aún más y la pasa a la siguiente capa.

- Capa de salida

La capa de salida proporciona el resultado final de todo el procesamiento de datos que realiza la red neuronal artificial. Puede tener uno o varios nodos. Por ejemplo, si tenemos un problema de clasificación binaria (sí/no), la capa de salida tendrá un nodo de salida que dará como resultado 1 o 0. Sin embargo, si tenemos un problema de clasificación multiclase, la capa de salida puede estar formada por más de un nodo de salida.

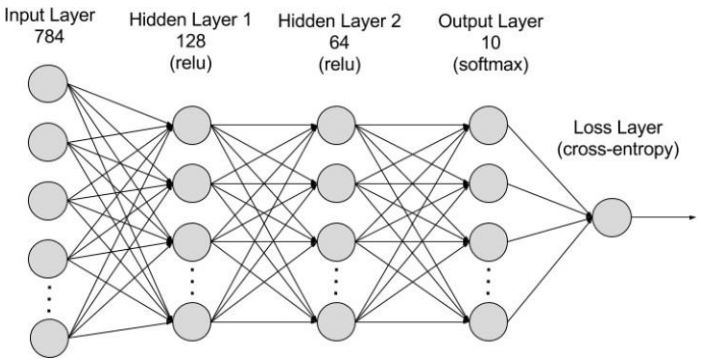

Figura 4: Conexiones red neuronal.

2.1 Redes Neuronales Recurrentes

¿Qué son las redes neuronales recurrentes?

Una red neuronal recurrente (RNN) es un tipo de red neuronal artificial que utiliza datos secuenciales o datos de series de tiempo. Estos algoritmos de aprendizaje profundo se utilizan comúnmente para problemas ordinales o temporales, como la traducción de idiomas, el procesamiento de lenguaje natural (nlp), el reconocimiento de voz y subtítulos de imágenes; están incorporados en aplicaciones populares como Siri, búsqueda por voz y Google Translate. Al igual que las redes neuronales feedforward y convolucionales (CNN), las redes neuronales recurrentes utilizan datos de entrenamiento para aprender. Se distinguen por su "memoria", ya que toman información de entradas anteriores para utilizarse en los datos de entrada y en los resultados. Si bien las redes neuronales profundas tradicionales asumen que los datos de entrada y los resultados son independientes entre sí, los resultados de las redes neuronales recurrentes dependen de los elementos anteriores dentro de la secuencia. Si bien los eventos futuros también serían útiles para determinar los resultados de una secuencia dada, las redes neuronales recurrentes unidireccionales no pueden tener en cuenta estos eventos en sus predicciones.

Tomemos un modismo, como "tener mala pata", que se usa comúnmente cuando alguien tiene mala suerte, para ayudarnos en la explicación de las RNNs. Para que la expresión tenga sentido, debe expresarse en ese orden específico. Como resultado, las redes recurrentes deben tener en cuenta la posición de cada palabra en dicho modismo y usan esa información para predecir la siguiente palabra en la secuencia.

Otra característica distintiva de las redes recurrentes es que comparten parámetros en cada capa de la red. Si bien las redes tipo "feedforward" tienen diferentes pesos en cada nodo, las redes neuronales recurrentes comparten el mismo parámetro de peso dentro de cada capa de la red. Dicho esto, estos pesos aún se ajustan en los procesos de retro propagación y descenso de gradiente para facilitar el aprendizaje por refuerzo.

Las redes neuronales recurrentes aprovechan el algoritmo de retro propagación a través del tiempo (BPTT) para determinar los gradientes, que es ligeramente diferente de la retro propagación tradicional, ya que es específico de los datos de secuencia. Los principios de BPTT son los mismos que los de la retro propagación tradicional, donde el modelo se entrena a sí mismo calculando errores desde su capa de salida hasta su capa de entrada. Estos cálculos nos permiten ajustar y adaptar los parámetros del modelo de manera adecuada. BPTT se diferencia del enfoque tradicional en que BPTT suma los errores en cada paso de tiempo, mientras que las redes "feedforward" no necesitan sumar errores, ya que no comparten parámetros en cada capa.

A través de este proceso, los RNN tienden a encontrarse con dos problemas, conocidos como gradientes explosivos y gradientes que desaparecen. Estos problemas se definen por el tamaño del gradiente, que es la pendiente de la función de pérdida a lo largo de la curva de error. Cuando el gradiente es demasiado pequeño, continúa haciéndolo más pequeño, actualizando los parámetros de peso hasta que se vuelven insignificantes, es decir, 0. Cuando eso ocurre, el algoritmo ya no está aprendiendo. Los gradientes explosivos suceden cuando el gradiente es demasiado grande, creando un

modelo inestable. En este caso, los pesos del modelo crecerán demasiado y, finalmente, se representarán como NaN. Una solución a estos problemas es reducir la cantidad de capas ocultas dentro de la red neuronal, eliminando parte de la complejidad en el modelo RNN. [IBM]

2.1 Long Short-Term Memory

Memoria a largo plazo a corto plazo (LSTM): Esta es una arquitectura RNN popular, que fue introducida por Sepp Hochreiter y Juergen Schmidhuber como una solución al problema del gradiente que desaparece. En su artículo científico (PDF, https://www.bioinf.jku.at/publications/older/2604.pdf), ellos abordan el problema de las dependencias a largo plazo. Es decir, si el estado anterior que influye en la predicción actual no es del pasado reciente, es posible que el modelo RNN no pueda predecir con precisión el estado actual. Como ejemplo, digamos que queríamos predecir las palabras en cursiva a continuación, "Alice es alérgica a las nueces. Ella no puede comer mantequilla de maní. " El contexto de una alergia a las nueces puede ayudarnos a anticipar que los alimentos que no se pueden comer contienen nueces. Sin embargo, si ese contexto fuera unas pocas oraciones antes, entonces sería difícil, o incluso imposible, para la RNN conectar la información. Para remediar esto, los LSTM tienen "celdas" en las capas ocultas de la red neuronal, que tienen tres puertas: una puerta de entrada, una puerta de salida y una puerta de olvido. Estas puertas controlan el flujo de información que se necesita para predecir el resultado en la red. Por ejemplo, si los pronombres de género, como "ella", se repitieron varias veces en oraciones anteriores, puede excluirlo del estado de celda. [IBM]

2.2 Modelo Seq2seq

¿Qué es un modelo seq2seq?

Los modelos de secuencia a secuencia (abrv. Seq2Seq) son modelos de aprendizaje profundo que han tenido mucho éxito en tareas como la traducción automática, resumen de texto y subtítulos. Google Translate comenzó a usar un modelo de este tipo en producción a finales de 2016. Estos modelos se explican en los dos artículos pioneros (Sutskever et al., 2014, Cho et al., 2014).

Un modelo Seq2Seq es un modelo que toma una secuencia de elementos (palabras, letras, series temporales, etc.) y genera otra secuencia de elementos a la salida.

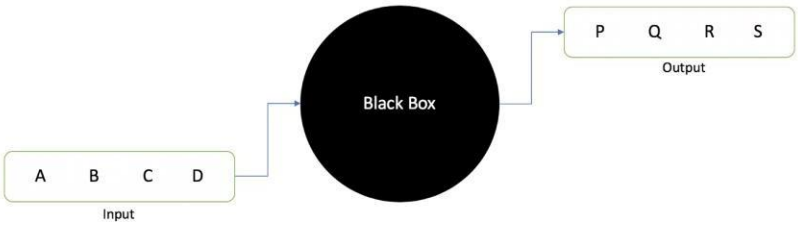

Figura 5: Modelo seq2seq.

En el caso de la traducción automática neuronal, la entrada es una serie de palabras y la salida es la serie de palabras traducidas.

El modelo está compuesto por un codificador y un decodificador. El codificador captura el contexto de la secuencia de entrada en forma de un vector de estado oculto y lo envía al decodificador, que luego produce la secuencia de salida. Dado que la tarea se basa en secuencias, tanto el codificador como el decodificador tienden a usar algún tipo de RNN (Red neuronal recurrente), LSTM (Long short-term memory), etc. El vector de estado oculto puede tener cualquier tamaño, aunque en la mayoría de los casos, se toma como una potencia de 2 y un gran número (256, 512, 1024) que puede representar de alguna manera la complejidad de la secuencia completa, así como el dominio.

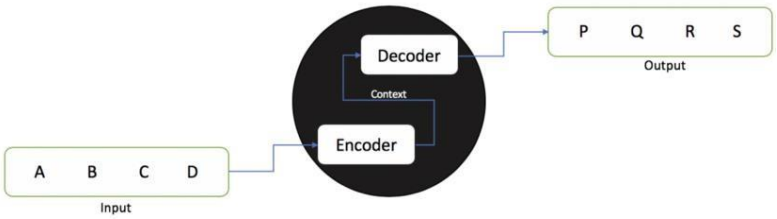

Figura 6: Modelo seq2seq encoder-decoder.

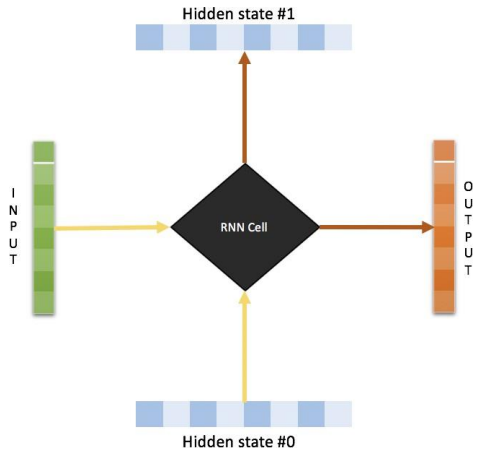

Figura 7: Celula RNN.

Los RNN por diseño toman dos entradas, el ejemplo actual que ven y una representación de la entrada anterior. Por lo tanto, la salida en el paso de tiempo t depende tanto de la entrada actual como de la entrada en el tiempo t-1. Esta es la razón por la que se desempeñan mejor cuando se les plantean tareas relacionadas con las secuencias. La información secuencial se conserva en un estado oculto de la red y se utiliza en la siguiente instancia.

El codificador, que consta de RNN, toma la secuencia como entrada y genera una incorporación final al final de la secuencia. Esto luego se envía al decodificador, que luego lo usa para predecir una secuencia y, después de cada predicción sucesiva, usa el estado oculto anterior para predecir la siguiente instancia de la secuencia.

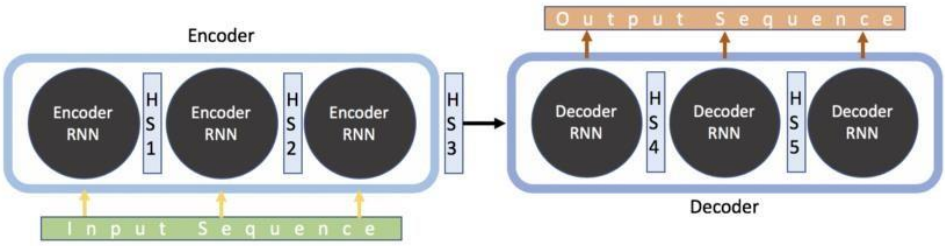

Figura 8: Modelo codificador-decodificador para modelado seq2seq.

Capítulo 3

Descripción del sistema y herramientas

3.1 Medios digitales

3.1.1 Android

Android es un sistema operativo móvil basado en núcleo Linux y otro software de código abierto. Fue diseñado para dispositivos móviles con pantalla táctil, como teléfonos inteligentes, tabletas, relojes inteligentes (Wear OS), automóviles con otros sistemas a través de Android Auto, al igual los automóviles con el sistema Android Automotive y televisores Leanback. [WAndr]

Inicialmente fue desarrollado por Android Inc., que adquirió Google en 2005. Android fue presentado en 2007 junto con la fundación del Open Handset Alliance (un consorcio de compañías de hardware, software y telecomunicaciones) para avanzar en los estándares abiertos de los dispositivos móviles. El código fuente principal de Android se conoce como Android Open Source Project (AOSP), que se licencia principalmente bajo la Licencia Apache. Android es el sistema operativo móvil más utilizado del mundo, con una cuota de mercado superior al 90 % al año 2018, muy por encima de IOS. [WAndr]

3.1.2 Android Studio

Android Studio es el entorno de desarrollo integrado oficial para la plataforma Android. Fue anunciado el 16 de mayo de 2013 en la conferencia Google I/O, y reemplazó a Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android. La primera versión estable fue publicada en diciembre de 2014. [WAS]

Está basado en el software IntelliJ IDEA de JetBrains y ha sido publicado de forma gratuita a través de la Licencia Apache 2.0. Está disponible para las plataformas GNU/Linux, macOS, Microsoft Windows y Google Chrome OS. Ha sido diseñado específicamente para el desarrollo de Android. [WAS]

Estuvo en etapa de vista previa de acceso temprano a partir de la versión 0.1, en mayo de 2013, y luego entró en etapa beta a partir de la versión 0.8, lanzada en junio de

2014. La primera compilación estable, la versión 1.0, fue lanzada en diciembre de 2014. [WAS]

Desde el 7 de mayo de 2019, Kotlin es el lenguaje preferido de Google para el desarrollo de aplicaciones de Android. Aun así, Android Studio admite otros lenguajes de programación, como Java y C ++. [WAS]

3.1.3 Java

Java es un lenguaje de programación y una plataforma informática que fue comercializada por primera vez en 1995 por Sun Microsystems. Hay muchas aplicaciones y sitios web que no funcionarán, probablemente, a menos que tengan Java instalado y cada día se crean más. Java es rápido, seguro y fiable. Desde ordenadores portátiles hasta centros de datos, desde consolas para juegos hasta computadoras avanzadas, desde teléfonos móviles hasta Internet, Java está en todas partes, si es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con unos diez millones de usuarios reportados. [WJ]

El lenguaje de programación Java fue desarrollado originalmente por James Gosling, de Sun Microsystems (constituida en 1983 y posteriormente adquirida el 27 de enero de 2010 por la compañía Oracle), y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son compiladas a bytecode (clase Java), que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente. [WJ]

La compañía Sun desarrolló la implementación de referencia original para los compiladores de Java, máquinas virtuales y librerías de clases en 1991, y las publicó por primera vez en 1995. A partir de mayo de 2007, en cumplimiento de las especificaciones del Proceso de la Comunidad Java, Sun volvió a licenciar la mayoría de sus tecnologías de Java bajo la Licencia Pública General de GNU. Otros han desarrollado también implementaciones alternas a estas tecnologías de Sun, tales como el Compilador de Java de GNU y el GNU Classpath. [WJ]

3.1.4 Python

Python es un lenguaje de programación ampliamente utilizado en las aplicaciones web, el desarrollo de software, la ciencia de datos y el machine learning (ML). Los desarrolladores utilizan Python porque es eficiente y fácil de aprender, además de que se puede ejecutar en muchas plataformas diferentes. El software Python se puede descargar gratis, se integra bien a todos los tipos de sistemas y aumenta la velocidad del desarrollo.[PY]

3.1.5 TensorFlow Lite

TensorFlow Lite es un conjunto de herramientas para ayudar a los desarrolladores a ejecutar modelos de TensorFlow en dispositivos incorporados, móviles o de IoT. Permite la inferencia de aprendizaje automático en dispositivos con una latencia baja y un tamaño de objeto binario pequeño. [TFL]

TensorFlow Lite consta de dos componentes principales:

- El intérprete de TensorFlow Lite ejecuta modelos especialmente optimizados en varios tipos de hardware diferentes, entre los que se incluyen teléfonos celulares, dispositivos Linux incorporados y microcontroladores.

- El conversor de TensorFlow Lite convierte los modelos de TensorFlow en un formato eficiente para que los use el intérprete y además, puede implementar optimizaciones para mejorar el tamaño y el rendimiento de los objetos binarios.

Aprendizaje automático en el perímetro

TensorFlow Lite se diseñó para facilitar el aprendizaje automático en dispositivos que están "en el perímetro" de la red, y así, no tener que de enviar o recibir datos mediante un servidor. Para los desarrolladores, el uso del aprendizaje automático en dispositivos puede mejorar los siguientes aspectos:

- Latencia: no hay ida y vuelta con un servidor.
- Privacidad: no es necesario que los datos salgan del dispositivo.
- Conectividad: no se requiere una conexión a Internet.
- Consumo de energía: las conexiones de red requieren mucha energía.

TensorFlow Lite funciona en una gran variedad de dispositivos, desde microcontroladores pequeños hasta teléfonos celulares potentes. [TFL]

3.1.6 Keras

Keras es una biblioteca de código abierto (con licencia MIT) escrita en Python, que se basa principalmente en el trabajo de François Chollet, un desarrollador de Google, en el marco del proyecto ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System). La primera versión de este software multiplataforma se lanzó el 28 de marzo de 2015. El objetivo de la biblioteca es acelerar la creación de redes neuronales: para ello, Keras no funciona como un framework independiente, sino como una interfaz de uso intuitivo (API) que permite acceder a varios frameworks de aprendizaje automático y desarrollarlos. Entre los frameworks compatibles con Keras, se incluyen Theano, Microsoft Cognitive Toolkit (anteriormente CNTK) y TensorFlow.[KS]

3.2 Medios Físicos

Como medios físicos se utiliza una computadora DELL modelo Inspiron 5570 que cuenta con las siguientes características:

- Nombre del SO: Microsoft Windows 10 Home Single Language
- Tipo de sistema: x64-based PC
- Procesador: Intel(R) Core (TM) i3-8130U CPU @ 2.20GHz, 2208 Mhz, 2 procesadores principales, 4 procesadores lógicos
- Memoria física instalada (RAM): 4.00 GB

Y las pruebas y ejecución de la aplicación se realizan en un teléfono inteligente Samsung modelo A52 con las características:

- Versión de Android: Android 13 Tiramisu
- Procesador. Qualcomm SM7125 Octa Core (2 x 2.3GHz + 6 x 1.8GHz)
- Memoria. RAM: 6 GB

Capítulo 4

Implementación

4.1 Modelo y exportación a TFLite

Importar las librerías requeridas

```
import tensorflow as tf
import os
os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = "true"

import string
import re
from numpy import array, argmax, random, take
import pandas as pd
import matplotlib.pyplot as plt
import os

%matplotlib inline
pd.set_option('display.max_colwidth', 200)
```

Figura 9: Importar librerias.

Leer los datos de entrenamiento

Nuestros datos se encuentran en un archivo .txt que contiene pares de frases en español – purépecha. Lo primero es leer el archivo usando la siguiente función.

```
# function to read raw text file
def read_text(filename):
        # open the file
        file = open(filename, mode='rt', encoding='utf-8')

        # read all text
        text = file.read()
        file.close()
        return text
```

Figura 10: Leer líneas.

Ahora definimos otra función para separar cada línea del archivo y obtener las parejas de frases. Una vez hecho esto vamos a separar esas parejas en frases en español y frases en purépecha respectivamente

```
# split a text into sentences
def to_lines(text):
      sents = text.strip().split('\n')
      sents = [i.split('\t') for i in sents]
      return sents
```

Figura 11: Separar frases.

Ya podemos usar esas funciones para leer el archivo y guardarlo en un arreglo con el formato que necesitamos

```
data = read_text("esp_pur.txt")
beng_eng = to_lines(data)
beng_eng = array(beng_eng)
```

Figura 12: Leer archivo.

Preprocesamiento de texto

Un paso bastante importante en cualquier proyecto, especialmente en PNL. La mayoría de los datos con los que trabajamos no están estructurados, por lo que hay ciertas cosas de las que debemos ocuparnos antes de saltar a la parte de creación de modelos.

Nos desharemos de los signos de puntuación y luego convertiremos todo el texto a minúsculas.

```
beng_eng[:,0] = [s.translate(str.maketrans('', '', string.punctuation)) for s in beng_eng[:,0]]
beng_eng[:,1] = [s.translate(str.maketrans('', '', string.punctuation)) for s in beng_eng[:,1]]

beng_eng
```

```
# convert text to lowercase
for i in range(len(beng_eng)):
    beng_eng[i,0] = beng_eng[i,0].lower()
    beng_eng[i,1] = beng_eng[i,1].lower()

beng_eng[0,1]
```

Figura 13: Limpiar texto.

Luego, vectorizamos nuestros datos de texto usando la clase Tokenizer() de Keras. Esta convertirá nuestras oraciones en secuencias de números enteros. Luego podemos rellenar esas secuencias con ceros para hacer que todas las secuencias tengan la misma longitud.

Debemos tener en cuenta que prepararemos tokenizadores para ambas oraciones, español y purépecha:

```
# function to build a tokenizer
def tokenization(lines):
      tokenizer = tf.keras.preprocessing.text.Tokenizer()
      tokenizer.fit_on_texts(lines)
      return tokenizer
```

```
# prepare Español tokenizer
eng_tokenizer = tokenization(beng_eng[:, 0])
eng_vocab_size = len(eng_tokenizer.word_index) + 1

eng_length = 8
print('Tamaño de Vocabulario en Español: %d' % eng_vocab_size)

import json
with open( 'word_dict1.json' , 'w' ) as file:
    json.dump( eng_tokenizer.index_word , file )
```

```
# prepare Purépecha tokenizer
deu_tokenizer = tokenization(beng_eng[:, 1])
deu_vocab_size = len(deu_tokenizer.word_index) + 1

ben_length = 8
print('Tamaño de Vocabulario en Purépecha: %d' % deu_vocab_size)

import json
with open( 'word_dict.json' , 'w' ,encoding='utf8') as file:
    json.dump( deu_tokenizer.index_word , file, ensure_ascii=False )
```

Figura 14: Tokenizar.

El siguiente bloque de código contiene una función para preparar las secuencias. También realizará relleno de la secuencia a una longitud máxima como se mencionó anteriormente.

```
# encode and pad sequences
def encode_sequences(tokenizer, length, lines):
        # integer encode sequences
        seq = tokenizer.texts_to_sequences(lines)
        # pad sequences with 0 values
        seq = tf.keras.preprocessing.sequence.pad_sequences(seq, maxlen=length, padding='post')
        return seq
```

Figura 15: Creación de secuencias.

Construcción del modelo

Ahora dividiremos los datos en conjuntos de entrenamiento y prueba para el entrenamiento y la evaluación del modelo, respectivamente.

```
from sklearn.model_selection import train_test_split

# split data into train and test set
train, test = train_test_split(beng_eng, test_size=0.2, random_state = 12)
```

Figura 16: División de conjuntos

Es hora de codificar las oraciones. Codificaremos oraciones en español como secuencias de entrada y oraciones en purépecha como secuencias de destino. Esto debe hacerse tanto para el entrenamiento como para los conjuntos de datos de prueba.

```
# prepare training data
trainY = encode_sequences(deu_tokenizer, ben_length, train[:, 1])
trainX = encode_sequences(eng_tokenizer, eng_length, train[:, 0])

# prepare validation data
testY = encode_sequences(deu_tokenizer, ben_length, test[:, 1])
testX = encode_sequences(eng_tokenizer, eng_length, test[:, 0])
```

Figura 17: Codificar frases.

Ahora comenzaremos definiendo la arquitectura de nuestro modelo Seq2Seq:

Para el codificador, usaremos una capa de incrustación y una capa LSTM
Para el decodificador, usaremos otra capa LSTM seguida de una capa densa

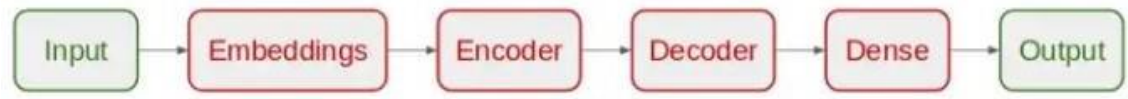

Figura 18: Arquitectura seq2seq.

```
# build NMT model
def define_model(in_vocab,out_vocab, in_timesteps,out_timesteps,units):
      model = tf.keras.Sequential()
      model.add(tf.keras.layers.Embedding(in_vocab, units, input_length=in_timesteps, mask_zero=True))
      model.add(tf.keras.layers.LSTM(units))
      model.add(tf.keras.layers.RepeatVector(out_timesteps))
      model.add(tf.keras.layers.LSTM(units, return_sequences=True))
      model.add(tf.keras.layers.Dense(out_vocab, activation='softmax'))
      return model
```

Figura 19: Definición del modelo.

Veamos el resumen de nuestro modelo para utilizar esta información en la implementación en Android Studio.

```
model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 8, 512)            460800

 lstm (LSTM)                 (None, 512)               2099200

 repeat_vector (RepeatVector  (None, 8, 512)           0
 )

 lstm_1 (LSTM)               (None, 8, 512)            2099200

 dense (Dense)               (None, 8, 986)            505818

=================================================================
Total params: 5,165,018
Trainable params: 5,165,018
Non-trainable params: 0
_________________________________________________________________
```

Figura 20: Resumen del modelo.

Usamos el optimizador RMSprop en este modelo, ya que suele ser una buena opción cuando se trabaja con redes neuronales recurrentes.

```
rms = tf.keras.optimizers.RMSprop(learning_rate=0.001)
model.compile(optimizer=rms, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Figura 21: Optimización del modelo.

Tengamos en cuenta que hemos utilizado 'sparse_categorical_crossentropy' como función de pérdida. Esto se debe a que la función nos permite usar la secuencia de destino tal cual, en lugar del formato codificado one-hot. La codificación one-hot de las secuencias de destino utilizando un vocabulario tan grande podría consumir toda la memoria de nuestro sistema.

Ahora que tenemos todo listo para comenzar el entrenamiento de nuestro modelo, lo entrenaremos durante 30 épocas y con un tamaño de batch de 512 con una división de validación del 20 %. El 80% de los datos se utilizarán para entrenar el modelo y el resto para evaluarlo. Podemos cambiar y jugar con estos hiperparámetros.

También usaremos la función ModelCheckpoint() para guardar el modelo con la menor pérdida de validación.

```
filename = 'model_traductor'
checkpoint = tf.keras.callbacks.ModelCheckpoint(filename, monitor='val_loss', verbose=1, save_best_only=True, mode='min')

history = model.fit(trainX, trainY.reshape(trainY.shape[0], trainY.shape[1], 1),
          epochs=100, batch_size=128,
          validation_split = 0.1,
          callbacks=[checkpoint], verbose=1)

model.save(filename)
```

Figura 22: Entrenar el modelo.

Una vez tenemos entrenado nuestro modelo podemos exportarlo a el formato TFlite para poder utilizarlo en dispositivos, esto lo aremos con el siguiente código.

```
from tensorflow import lite
converter = lite.TFLiteConverter.from_keras_model(model)

converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.experimental_new_converter=True
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS,
tf.lite.OpsSet.SELECT_TF_OPS]

tfmodel = converter.convert()
open('nmt_05_12_19_test_beng.tflite', 'wb').write(tfmodel)
```

Figura 23: Exportar modelo.

4.2 Implementación en Android Studio

Dentro de la función onCreate() vamos a inicializar los elementos de nuestra interfaz de usuario para poder ser utilizados en la aplicación, también inicializaremos nuestro interprete de TFLite y cargaremos los archivos JSon que contienen los vocabularios.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    translateButton =findViewById(R.id.button2);
    userInput = findViewById(R.id.editText);
    outputSentence = findViewById(R.id.textView);

    try {
        tfLite = new Interpreter(loadModelFile(this.getAssets(), modelFilename: "nmt_05_12_19_test_beng.tflite"));
    } catch (IOException e) {
        e.printStackTrace();
    }


    loadJson();
```

Figura 24: Función onCreate().

Después vamos a definir la función setOnClickListener() para esperar el click en el botón de traducir.

Con esta función vamos a separar las palabras en un arreglo y convertirlo a entero buscando el equivalente de cada palabra en el diccionario de palabras que definimos.

Una vez tenemos el arreglo en el formato apropiado para ser una entrada para correr nuestro modelo ejecutamos la función runModel().

```
translateButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

            String[] words = userInput.getText().toString().split( regex: "\\s+");

            int wordsLength = words.length;

            float[][] inputArray = new float[1][8];

            if(wordsLength > 8)
            {
                for(int i = 0; i < 8; i++)
                    inputArray[0][i] = getTokenNumber(englishTokenList,words[i]);
            }
            else
            {
                for(int i = 0; i < 8; i++) {
                    if(i >= wordsLength)
                        inputArray[0][i] = 0.0f;
                    else
                        inputArray[0][i] = getTokenNumber(englishTokenList, words[i]);
                }
            }

            String res = runModel(inputArray,bengaliTokenList);

            outputSentence.setText(res);
    }
});
```

Figura 25: Función onClick().

El la función runModel() creamos un arreglo con el formato de salida de nuestro modelo, nosotros definimos una longitud de palabras de 8 y el valor devuelto de la función modal.summary() nos dio una dimensión de 986.

Una vez que tenemos los arrays de entrada y salida para nuestro modelo podemos ejecutar la función run() y mandar esos parámetros.

Ya que tenemos el resultado formaremos la cadena resultante obteniendo los índices equivalentes a nuestro archivo JSon.

Después solo retornamos la traducción para que sea pintada en el campo de texto que definimos para mostrar al usuario el resultado.

```
private String runModel(float[][] inputVal,ArrayList<String> list){

//       float[][][] outputVal = new float[1][8][3329];
    float[][][] outputVal = new float[1][8][986];

    tfLite.run(inputVal,outputVal);

    StringBuilder stringBuilder = new StringBuilder();

    for (float[][] floats : outputVal) {
        for (float[] aFloat : floats) {
            Log.d( tag: "Test",  msg: "deep arr: " + Integer.toString(argMax(aFloat)));

            stringBuilder.append(getWordFromToken(list,argMax(aFloat)));
            stringBuilder.append(" ");
        }

    }

    return stringBuilder.toString();
}
```

Figura 26: Función runModel().

Capítulo 5

Pruebas, resultados y limitaciones

5.1 Pruebas

Realizando la prueba con la palabra serpiente vemos que el resultado coincide con la traducción equivalente como se muestra en la siguiente figura.

Figura 27: Prueba con palabra "serpiente".

Si hacemos una prueba con una palabra que no se encuentra en el diccionario, nos muestra como resultado un valor erróneo como se muestra en este ejemplo con la palabra lanzar:

Figura 28: Prueba con palabra lanzar.

5.2 Resultados y limitaciones.

Como podemos observar, la aplicación es capaz de detectar correctamente las palabras que se encuentran en el diccionario sin embargo nos muestra resultados erróneos con palabras fuera de él.

Esta aplicación solo permite realizar traducciones de máximo 8 palabras ya que al incrementar la cantidad cada vez nos da resultados más alejados de la traducción real.

Además, debido a los datos de entrenamiento, hasta ahora solo nos da resultados de una palabra. esto se puede corregir entrenando el modelo con frases en vez de solo palabras

Capítulo 6

Conclusiones y trabajos futuros

6.1 Conclusiones

Las redes neuronales recurrentes y en general las técnicas de la inteligencia artificial son grandes herramienta para diversas aplicaciones que se encuentran presentes en nuestro día a día, cada vez es más necesario formarse para al menos entender un poco de estas tecnologías, ya que el mundo avanza a una gran velocidad y nosotros como ingenieros debemos estar apegados al cambio.

Con el desarrollo planteado se logró aplicar estos conocimientos y se obtuvieron resultados satisfactorios, aunque aún tiene mucho campo de mejora ya que en este tipo de aplicaciones siempre tiene algo que se podría mejorar.

Fue un gran reto ya que integrar los modelos programados para computadora en una aplicación para dispositivos móviles no es sencillo, aunque ya existen librerías que nos exportan los modelos, a veces no son compatibles entre si ya sea por las versiones de las librerías o porque los modelos exportados no son soportados.

Es un hecho que esta área tiene mucha "tela de la que cortar" además de ser un área que en la actualidad es bien remunerada con amplias áreas de oportunidad.

Aunque no es un conocimiento relativamente nuevo, el impacto que está teniendo en la actualidad va a hacer que no se termine el desarrollo de nuevas tecnologías mejoradas en las que podamos desenvolvernos profesionalmente.

6.2 Trabajos futuros.

Existen diversas mejoras que se podrían hacer al presente trabajo, entre las cuales se encuentran:

- Mejora en la interfaz gráfica de usuario para que sea más amigable y con un mejor diseño.
- Mejorar los datos con los que fue entrenamiento el modelo, ya que los datos utilizados son muy pocos además no son frases de muchas palabras lo cual hace que no se explote al máximo la capacidad de este tipo de modelos, es mejor tener datos con frases cortas.
- Otra de las mejoras seria desarrollar aún más la aplicación para que sea capaz de cambiar entre idiomas, es decir, que pueda traducir de español a purépecha o de purépecha a español. incluso integrar más lenguas de México.
- Finalmente se podría implementar algoritmos con mayor desempeño, por ejemplo, el modelo Transformer de la librería TensorFlow.
  Este modelo de transformador maneja entradas de tamaño variable utilizando pilas de capas de autoatención en lugar de RNN o CNN.
  Esta arquitectura general tiene una serie de ventajas:

  - No hace suposiciones sobre las relaciones temporales/espaciales entre los datos. Esto es ideal para procesar un conjunto de objetos (por ejemplo, unidades de StarCraft).
  - Las salidas de capa se pueden calcular en paralelo, en lugar de una serie como un RNN.
  - Los elementos distantes pueden afectar la salida de los demás sin pasar por muchos pasos RNN o capas de convolución.
  - Puede aprender dependencias de largo alcance. Este es un desafío en muchas tareas de secuenci

Referencias

[WAndr] *Wikipedia*. Android.
Disponible [Internet]: <https://es.wikipedia.org/wiki/Android> [26 may 2021]

[WAS] *Wikipedia*. Android Studio.
Disponible [Internet]: <https://es.wikipedia.org/wiki/Android_Studio> [28 may 2021]

[WJ] *Wikipedia*. Java (Lenguaje de programacion).
Disponible [Internet]:
<https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)> [24 may 2021]

[TFL] *Tensor Flow Lite*. Tensor Flow.
Disponible [Internet]:
<https://www.tensorflow.org/lite/guide?hl=es-419> [05 feb 2021]

[EE] *El Español*. Huawei y5 prime 2018.
Disponible [Internet]:
< https://www.elespanol.com/elandroidelibre/moviles-android/20180517/huawei-y5-prime-android-oreo-movil-ajustado/307970698_0.html> [2021]

[.NET] *Microsoft .NET*. Microsoft Corporation. 2002. Disponible [Internet]: <http://www.microsoft.com/net> [24 de enero de 2023]

[ABROSE] *Agent Based Brokerage Services in Electronic Commerce*. ACTS-316. Disponible [Internet]: <http://www.cordis.lu/infowin/acts/rus/projects/ac316.htm> [24 de enero de 2023]

[AUI] *Asociación Española de Usuarios de Internet*. Disponible [Internet]: <http://www.aui.es> [24 de enero de 2023]

[Ber96+] Berners-Lee, T., R. Fielding, UC Irvine y H. Frystyk. *Hypertext Transfer Protocol -- HTTP/1.0*. Internet Engineering Task Force. Request for Comments: 1945. Mayo 1996. Disponible [Internet]:
<http://www.ietf.org/rfc/rfc1945.txt> [24 de enero de 2023]

[Cha99] Chapel, David. *Simple Object Access Protocol and firewalls*. Diciembre 1999. Disponible [Internet]:
<http://msdn.microsoft.com/workshop/xml/general/SOAP_White_Paper.asp> [24 de enero de 2023]

[IBM] *IBM*. Disponible [Internet]:
https://www.ibm.com/mx-es/topics/recurrent-neural-networks#:~:text=Una%20red%20neuronal%20recurrente%20(RNN,datos%20de%20series%20de%20tiempo.> [24 de enero de 2023]

[KS] *Digital Guide*. Disponible [Internet]:
<https://www.ionos.mx/digitalguide/online-marketing/marketing-para-motores-de-busqueda/que-es-keras/
.> [24 de enero de 2023]

[PY] *Amazon*. Disponible [Internet]:
<https://aws.amazon.com/es/what-is/python/> [24 de enero de 2023]