UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

Verificación automática de hechos utilizando un modelo de aprendizaje profundo interpretable

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:
LUIS RAMÓN CASILLAS PÉREZ SOTO

DIRECTOR DE TESIS:
DRA. HELENA MONTSERRAT GÓMEZ ADORNO
Instituto de Investigaciones en Matemáticas Aplicada y en Sistemas

C.U. Ciudad de México. Agosto 2021

Jurado asignado:

Presidente:	Dr. Gibrán Fuentes Pineda
Vocal:	Dra. Helena Montserrat Gómez Adorno
Secretario:	Dra. Gemma Bel Enguix
Suplente:	Dr. Noé Alejandro Castro Sánchez
Suplente:	Dra. Wendy Elizabeth Aguilar Martínez

Directora de tesis:

_____

Dra. Helena M. Gómez Adorno

Agradecimientos

Ciudad Universitaria, IIMAS, 2021

En primer lugar quiero agradecer a mis padres por haberme apoyado desde que inicie mi preparación académica. También agradezco a Leslie por haberme apoyado y aguantado estos dos años que sin duda no fueron fáciles. También a Mariana, Ana y Luciano que me escucharon cuando los necesitaba y me sacaban una sonrisa cuando más falta hacía.

A la Dra. Helena le agradezco mucho su excelente labor al guiarme en un mundo que para mí era nuevo, el procesamiento de lenguaje natural. También le agradezco por apoyarme para desarrollarme en otras áreas en el ámbito académico.

Muchas gracias a la UNAM y en particular al Posgrado en Ciencia en Ingeniería de la Computación por darme la oportunidad de continuar desarrollándome y conocer gente muy valiosa.

Por último quisiera agradecer al CONACYT por el apoyo recibido durante mi estancia en el posgrado. También agradezco los recursos de cómputo brindados a través de la Plataforma de Aprendizaje Profundo para Tecnologías del Lenguaje del Laboratorio de Supercómputo del INAOE, que sin duda fueron fundamentales para completar el presente trabajo.

Índice general

Índice de figuras

Índice de tablas

Verificación automática de hechos utilizando un modelo de aprendizaje profundo interpretable

Tesis de Maestría

Luis Ramón Casillas Pérez Soto
Posgrado en Ciencia e Ingeniería de la Computación
Universidad Nacional Autónoma de México

Resumen

La verificación de hechos es una iniciativa que tiene la intención de ser una herramienta de apoyo en la lucha contra la creación y distribución de información falsa. La información falsa puede tener muchas formas, se puede presentar como texto, imágenes, video o cualquier combinación de estos recursos. Muchas veces el poder detectar este tipo de información requiere de considerar diferentes aspectos. En este trabajo nos enfocamos en la verificación de hechos con datos textuales, se verifican afirmaciones escritas en lenguaje natural. Existen varias propuestas y vertientes tratando de resolver esta tarea. Algunos métodos se enfocan en detectar el estilo particular que utilizan los generadores de información falsa, otros se centran en la detección de patrones específicos en la interacción de la información con usuarios de redes sociales. Otro enfoque, al cual dedicamos los esfuerzos de este trabajo, son los métodos basados en el conocimiento, se utiliza una base de conocimiento para comprobar la veracidad de la afirmación.

En este trabajo presentamos una arquitectura novedosa de red neuronal basada en un modelo preentrenado de lenguaje. Se proponen dos modelos, el primero tiene el objetivo de extraer de un conjunto de documentos la evidencia que sea relevante para verificar una afirmación. El segundo modelo se centra en la verificación de una afirmación considerando la evidencia extraída. Estos modelos tienen la ventaja de que cuentan con una capa de atención que computa la relevancia de los elementos de entrada, mejorando el desempeño de los modelos, y otorgando a los mismos la capacidad de ser interpretados. Con las ponderaciones calculadas en la capa de atención se puede obtener una explicación de qué es lo que se está tomando en cuenta para evaluar la veracidad de la afirmación.

Capítulo 1

Introducción

Internet y las redes sociales han cambiado la forma en que las personas interactúan de formas que difícilmente se podrían haber imaginado en el pasado. Es muy simple realizar actividades como comunicarse con contactos en lugares muy alejados, reencontrarse con viejas amistades o mantenerse al tanto sobre el acontecer diario en el país y en el mundo. Un aspecto muy relevante de las redes sociales, que ha generado una verdadera revolución, es cómo se ha democratizado la generación de información. Anteriormente, para publicar información de forma masiva era necesario participar en un periódico, televisora o emisora radiofónica. Hoy sólo es necesario acceso a internet, y en unos cuantos teclazos y un par de clics se puede emitir una opinión acerca de cualquier tema. Prácticamente cualquier persona puede crear publicaciones que expresan sentimientos, juicios u opiniones. Actualmente es una realidad que la gran mayoría de usuarios de internet utilizan las redes sociales para mantenerse informados.

Dentro de la inmensa cantidad de datos que fluyen diariamente en los medios digitales, se puede encontrar información de calidad muy variada. Existen piezas de información que son apegadas a la realidad, otras que lo son parcialmente y otras tantas son distorsiones completas. Este tipo de verdades a medias e información falsa se presenta en forma de fenómenos como noticias falsas, notas satíricas, bromas, teorías conspirativas, desinformación y manipulación mediática. Cada uno de estos fenómenos tiene un origen particular y afectan de diversas formas a la sociedad. Ya sea que una pieza informativa haya sido creada intencionadamente o no, puede ser potencialmente muy perjudicial. En años recientes el problema de la información falsa ha crecido de forma muy importante, fomentado por el hecho de que muchas personas se han dado cuenta de que difundir información falsa en redes sociales tiene la capacidad de ser beneficioso política y económicamente.

Al mismo tiempo, se ha reconocido la importancia que tiene el combatir la generación y distribución de información falsa. Sin embargo, se trata de un problema de una gran complejidad. ¿Cómo saber si un texto es falso o no? En la práctica un texto difícilmente será completamente falso, más bien existen matices y niveles de falsedad. Entonces es importante poder segmentar el problema para poder clasificar la información con una granularidad más adecuada. La estrategia más común para la detección de información falsa es la comprobación manual, un conjunto de verificadores humanos tienen la tarea de buscar evidencia en fuentes confiables que permita determinar la veracidad de una pieza de información. Estos grupos de verificadores han surgido como respuesta a la gran proliferación de noticias falsas, buscando disminuir el impacto de la desinformación. No obstante, no todas las piezas de información son fáciles de verificar. Algunas quizá son tan absurdas que cualquier humano con conocimiento

promedio puede detectar que la información no es real. Pero otras piezas de información que intentan engañar a propósito utilizan un estilo bastante sofisticado por lo cual es bastante complicado decidir sobre la veracidad. En estos casos es necesario investigar y extraer evidencia que apoye o refute la información en cuestión.

La tarea manual de verificación de la información no es sencilla, requiere de mucho tiempo y recursos, lo cual lo hace costoso y poco rentable, considerando el volumen de información de dudosa procedencia que transita en medios digitales. Por esta razón, es relevante contar con herramientas tecnológicas que apoyen a esta labor.

1.1. Motivación

Verificar si una pieza de información es verdadera o falsa es una tarea complicada incluso para los humanos. Primero se requiere extraer del texto los hechos o afirmaciones que vale la pena verificar, no todos los hechos presentados en un texto tienen la misma relevancia para ser considerados. Una vez que se han extraído los hechos en forma de afirmaciones, entonces el grupo de verificadores debe iniciar el proceso de investigación utilizando fuentes que son de una confiabilidad previamente evaluada. Posteriormente, es necesario ponderar la evidencia encontrada y los verificadores deben decidir la veracidad de cada afirmación. La complejidad para los humanos es en gran medida causada por la enorme cantidad de información susceptible a ser verificada. En teoría cualquier publicación en medios digitales y redes sociales debería pasar por un proceso de evaluación. Pero mantener el ritmo de los publicadores es prácticamente imposible, ya que mientras se está verificando una afirmación, surgen otros cientos de miles de publicaciones con afirmaciones igualmente susceptibles a ser verificadas.

Recientemente, las labores de verificación de información ya se apoyan en herramientas tecnológicas tratando de hacer más ágil su proceso [16]. Sistemas de recuperación de información facilitan la labor de búsqueda de evidencia, ya que permiten extraer un subconjunto muy reducido de documentos que potencialmente pueden tener información de interés, facilitando el proceso de investigación. A pesar de las bondades de estos sistemas, aún es necesario filtrar todavía más la información. Los verificadores humanos lo que hacen es leer detenidamente el documento recuperado y seleccionar las partes de él que podrán servir para verificar la afirmación. Pero el problema del volumen de información sigue siendo latente en esta tarea de encontrar evidencia, es necesario contar con una herramienta automatizada que permita extraer la evidencia relevante para verificar la afirmación.

Una vez que se cuenta con toda la evidencia necesaria, el verificador humano requiere agrupar toda la evidencia, y con base en ella, decidir si la afirmación que se está evaluando es verdadera, falsa o parcialmente falsa. En este punto también es recomendable tener un apoyo tecnológico que sea capaz de decidir si una afirmación es verdadera o falsa dada la evidencia encontrada en pasos anteriores.

Como se observa, la tarea de verificación de hechos normalmente se ataca en 3 pasos, y en cada uno de estos pasos se debe de lidiar con la velocidad con la que está surgiendo nueva información. No importa qué tan grande y capaz sea un equipo de verificación, difícilmente le podrán ganar a este flujo descontrolado de publicaciones. Implementar un modelo eficaz de

verificación automática de hechos es de gran relevancia considerando el impacto que tiene en la sociedad.

1.2. Objetivos

El objetivo general de este trabajo consiste en proponer una arquitectura de red neuronal interpretable para la verificación de hechos basada en el conocimiento preexistente, aprovechando las virtudes que tiene el utilizar un modelo preentrenado de lenguaje.

1.2.1. Objetivos particulares

Los objetivos particulares que se plantean son los siguientes:

- Proponer una arquitectura neuronal siamesa que pueda determinar si cada una de las porciones de un texto es relevante con respecto a una afirmación.
- Proponer una arquitectura neuronal siamesa que tome como entrada una afirmación y evidencia, para determinar si la afirmación es verdadera o falsa.
- Establecer cuál es la mejor forma de ajustar un modelo preentrenado de lenguaje, a la tarea de evaluación de relevancia y a la verificación de afirmación.
- Proponer un mecanismo de atención en la red neuronal que permita la interpretación de la decisión tomada por el modelo con respecto a una afirmación.
- Aplicar los modelos propuestos e integrarlos en un método completo de verificación automática de hechos, para apoyar en la labor del combate a la desinformación.

1.3. Hipótesis

Los modelos del lenguaje permiten representar oraciones en un espacio vectorial lo cual ayuda al proceso de clasificación de textos. Sin embargo, su interpretabilidad es bastante compleja. Utilizar una red siamesa, codificando por separado la afirmación y la evidencia, y utilizando una capa de atención, permite construir un modelo igualmente eficaz en la tarea de verificación de hechos, pero a la vez proporciona una interpretación más simple e informativa acerca de los factores considerados para evaluar la veracidad de una afirmación.

1.4. Preguntas de investigación

Las preguntas de investigación que surgen en este trabajo son:

1. ¿Qué ventajas tiene utilizar una red siamesa en la verificación de hechos, comparado con el actual estado del arte?
2. ¿Cuál es la mejor estrategia de entrenamiento para ajustar un modelo preentrenado de lenguaje, a la tarea de verificación de hechos?
3. ¿Cuál es el mecanismo de atención más adecuado para evaluar la relevancia de una evidencia?

4. ¿Qué estrategia para resumir una secuencia de palabras, representadas como vectores, es la más adecuada para ajustarse a la tarea de interés?

5. ¿Cómo aprovechar una capa de atención, en una red siamesa para obtener una interpretación de la clasificación hecha por la red neuronal?

1.5. Contribución

1. Análisis de los modelos preentrenados de lenguaje que han marcado el estado del arte en procesamiento del lenguaje en años recientes.

2. Análisis del estado actual de los métodos especializados en la tarea de verificación automática de hechos.

3. Propuesta de una arquitectura de red neuronal siamesa que aprovecha un modelo preentrenado de lenguaje, y es capaz de verificar la veracidad de una afirmación con una efectividad cercana al estado del arte.

4. Una capa de atención que permite ponderar la relación entre una afirmación y la evidencia. Esta capa puede ser aprovechada para obtener una interpretación del modelo.

1.6. Organización de la tesis

En este trabajo se presenta una propuesta de arquitectura de red neuronal siamesa basada en un modelo preentrenado de lenguaje. El objetivo de esta arquitectura es tratar con el problema de evaluar si una oración contiene información relevante con respecto a una afirmación. También esta arquitectura se adapta a la tarea de verificar si una afirmación es verdadera o no considerando la evidencia encontrada. En el capítulo 2 se presentan los antecedentes teóricos en los cuales se basa el trabajo. Se inicia con la presentación de los modelos de lenguaje, sus antecedentes desde los métodos probabilísticos hasta los modelos neuronales. Posteriormente se discute la importancia del concepto de atención aplicada al aprendizaje profundo, así como los distintos mecanismos que se utilizan para implementar estas capas. Como seguimiento a los mecanismos de atención, se describe la arquitectura *transformer*, que justamente aprovecha las virtudes de la atención para lograr un modelo muy robusto que ha demostrado ser de gran utilidad en procesamiento del lenguaje natural (PLN). Después se presentan modelos preentrenados basados en esta arquitectura. En particular, nos enfocamos en *BERT* [8] que es el modelo preentrenado de lenguaje que se utiliza para implementar la arquitectura propuesta.

En el capítulo 3, se define formalmente el problema de la verificación de hechos. Se inicia con una discusión acerca del problema de la información falsa, que es una situación que incluye a otros fenómenos como las noticias falsas, rumores, noticias satíricas, bromas, etc. Se presentan las estrategias que se han desarrollado hasta ahora para detectar información falsa, y los aspectos que toma en cuenta cada estrategia. Nuestro trabajo se centra en un método basado en conocimiento, y nos enfocamos en describir la tarea de verificación automática de hechos. Finalmente, se listan las principales métricas que se utilizan para evaluar un modelo de verificación de hechos, estas métricas serán relevantes para comparar cuantitativamente cómo es que se están desempeñando las diferentes arquitecturas que se están comparando

para obtener la mejor propuesta de solución.

En el capítulo 4 se presenta la arquitectura de red neuronal siamesa que se está proponiendo en este trabajo de tesis para tratar con el problema mencionado. Se describe su estructura general y los elementos que la componen. Para cada uno de sus elementos se proponen algunas variantes en su configuración con la intención de experimentar y obtener de la red el mejor rendimiento.

El capítulo 5 resume la configuración experimental planteada para este trabajo. Se comienza con la descripción del conjunto de datos usado, y el preprocesamiento requerido para facilitar el entrenamiento. También se muestra cómo se llevó a cabo el entrenamiento en cada una de las etapas que el modelo debe de tratar. Los resultados de la etapa de experimentación están disponibles en el capítulo 6. Se muestran los resultados obtenidos en las tareas individuales, usando datos ideales para las tareas de evaluación de relevancia y verificación de afirmación. Se comparan todas las configuraciones propuestas y se obtienen los mejores modelos buscando compararlos ya en la tarea completa de verificación de hechos. Finalmente, se comparan las configuraciones propuestas con el estado del arte y un modelo de referencia.

El capítulo 7 tiene la intención de aplicar la arquitectura propuesta a un problema real que está aconteciendo actualmente: la desinformación en tiempos de la COVID-19. Se construye un conjunto de datos de afirmaciones sobre COVID-19, y junto con evidencia recolectada de artículos científicos, se etiqueta cada una de las afirmaciones como verdadera o falsa. Utilizando este conjunto se pueden entrenar los modelos para adaptarse a este dominio y ser capaces de verificar afirmaciones sobre esta enfermedad. El modelo implementado para este fin, demuestra ser un apoyo relevante en el combate a la desinformación durante la pandemia de la COVID-19.

Finalmente se concluye con algunas consideraciones que resultaron de este trabajo, aspectos importantes que quedan después de experimentar con estrategias de entrenamiento, distintas arquitecturas y su correspondiente evaluación contra el estado del arte actual. Se analizan las ventajas de la propuesta, sus limitantes como una herramienta contra la información falsa y las áreas de mejora que se pueden atacar en el trabajo futuro que se plantea.

Capítulo 2

Antecedentes

En este capítulo se describen conceptos y términos que se utilizarán en el desarrollo de esta tesis. Se inicia con un análisis de los modelos de lenguaje y cómo es que se han implementado hasta antes de la aparición de la arquitectura *transformer*. Posteriormente se presentan los mecanismos de atención y su importancia en PLN. Estos mecanismos de atención invariablemente están relacionados con la arquitectura *transformer*, sobre la cual se explica su funcionamiento básico y el por qué ha sido tan exitosa en PLN. Por último se expondrá el caso de *BERT* un modelo preentrenado de lenguaje, que utilizando aprendizaje por transferencia, estableció el estado del arte en varias tareas.

2.1. Modelos de lenguaje

Un modelo de lenguaje es aquel que es capaz de estimar la distribución de probabilidad de una palabra n en una secuencia, dadas las $n-1$ palabras anteriores en la secuencia. Debido a que es capaz de estimar la secuencia de palabras más probable dentro de un conjunto de opciones, el modelado de lenguaje tiene aplicaciones tales como reconocimiento del habla, corrección de errores ortográficos y gramaticales. Por ejemplo, en el caso del reconocimiento del habla, es posible que se identifiquen palabras que tienen un sonido muy similar, sin embargo, considerando el contexto y las palabras predichas anteriormente, se puede establecer cuál es la palabra más probable que se está escuchando. Estos modelos también tienen otras aplicaciones en tareas específicas de PLN gracias a que pueden aprender características del lenguaje y relaciones entre las palabras. Por esta razón es común que se utilicen modelos pre-entrenados de lenguaje para resolver otro tipo de tareas [25] [8] [30].

2.1.1. Modelos de lenguaje probabilísticos

Existen métodos probabilísticos que modelan la probabilidad conjunta de una secuencia de palabras [13]. Entonces la probabilidad de la secuencia $w_{1:n}$ utilizando la regla de la cadena queda:

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})...P(w_n|w_{1:n-1})$$

Utilizar esta regla para computar la probabilidad de una secuencia puede ser complicado. Si esta secuencia es muy larga, obtener la probabilidad de una secuencia de gran tamaño, incluso en un corpus muy grande, dará un valor muy pequeño o incluso igual a cero. Esto ocurre porque las secuencias se vuelven muy específicas mientras aumenta su longitud, lo

cual reduce la probabilidad de encontrarla en un corpus. Por esta razón esta probabilidad conjunta se aproxima asumiendo que una palabra depende únicamente de la palabra anterior. Esta simplificación se conoce como asunción de Markov. Esta asunción tiene diferentes grados dependiendo de cuántos elementos previos se consideran. En general se asume que utilizar un único elemento previo es una buena aproximación.

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1})$$

La probabilidad de la secuencia completa queda como:

$$P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$$

Para obtener esta probabilidad condicional se cuentan las apariciones del bigrama $w_{n-1}w_n$ y se divide entre todos los bigramas que tienen a w_{n-1}.

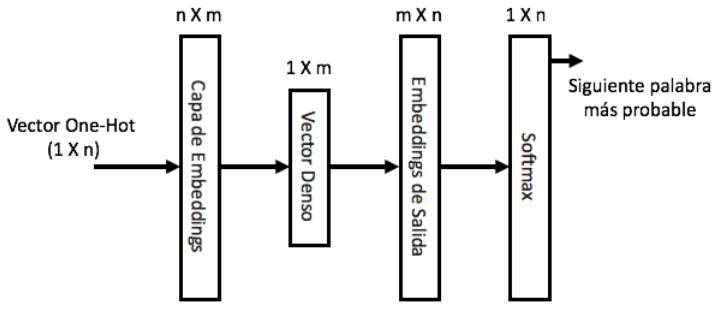
$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

Con estas ecuaciones se puede obtener la palabra más probable dada una secuencia previa.

2.1.2. Modelo de lenguaje neuronal básico

Si bien los métodos probabilísticos suelen dar buenos resultados en el modelado de lenguaje, el estado del arte se encuentra en las redes neuronales. De forma muy sencilla se puede utilizar una capa completamente conectada para crear un modelo de este tipo [2]. Se puede utilizar una arquitectura como en la Figura 2.1. Las palabras de entrada se codifican como vectores *one-hot*. Este tipo de codificación utiliza vectores de dimensión igual al tamaño de vocabulario. Para representar a la palabra n de este vocabulario, todas las dimensiones del vector se ponen a cero, excepto la dimensión n que se pone a 1.

Figura 2.1: Modelo de lenguaje utilizando una red neuronal *feed-forward*. La palabra de entrada se codifica como un vector one-hot. A la salida se obtiene una distribución de probabilidad sobre el vocabulario, que indica cuál es la siguiente palabra más probable.



Estos vectores se hacen pasar por una capa de *embeddings*, la cual transforma el vector *one-hot* a una representación densa de dimensión m. En la práctica esto se implementa multiplicando el vector de entrada por una matriz de $n \times m$ donde n es el tamaño del vocabulario y m la dimensión del vector denso. Este vector denso se hace pasar por otra capa de *embeddings* de

salida. Esta matriz será de $m \times n$, con lo cual se obtendrá un vector del tamaño del vocabulario.

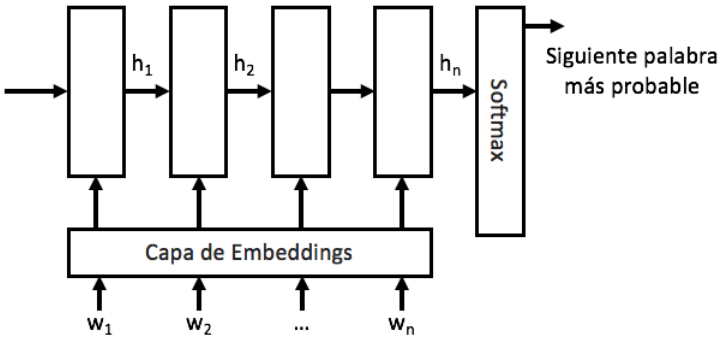
Como último paso, se utiliza una función *softmax* para crear una distribución de probabilidad sobre el vocabulario y así obtener la palabra que más probablemente sigue a la palabra de entrada. Para entrenar esta red neuronal es posible utilizar la entropía cruzada como función de pérdida para calcular la diferencia entre la distribución de probabilidad predicha y la distribución objetivo.

Este es un modelo muy sencillo, pero considera únicamente una palabra anterior. Es posible extender este modelo para tomar más palabras previas de forma similar a lo que se hace en el modelo CBOW [23], promediando los *embeddings* de entrada de las palabras anteriores. Pero esta técnica provoca que el orden de las palabras se pierda y se obtenga una representación equivalente a una bolsa de palabras. Utilizar una mayor cantidad de palabras previas ayuda a mejorar las predicciones de la siguiente palabra.

2.1.3. Modelo de lenguaje neuronal recurrente

Una forma más natural de modelar una secuencia de palabras utilizando una red neuronal es a través de redes recurrentes [24]. Este tipo de red considera celdas de tipo recurrente que por cada paso de tiempo toma una entrada y el estado oculto anterior para producir una salida y el estado oculto actual. Este nuevo estado oculto es alimentado para el siguiente paso, junto con la siguiente entrada de la secuencia. Este mecanismo se repite hasta que se acaben los elementos de la secuencia de entrada (Figura 2.2).

Figura 2.2: Modelo de Lenguaje utilizando una red neuronal recurrente. En cada paso de tiempo se alimenta la red con una la palabra de la secuencia. Al final el último estado oculto se hace pasar por una capa *softmax* para obtener la siguiente palabra más probable.



Para el caso de modelos de lenguaje, las entradas se procesan de la misma manera que con la capa completamente conectada, se codifica cada palabra como un vector *one-hot* y se pasa por una capa de *embeddings*. Con este *embedding* se alimenta la celda recurrente y se van generando n estados ocultos. Las salidas de la celda se ignoran. Al terminar de procesar todas las palabras de la secuencia, se toma el último estado oculto. Este estado oculto se pasa por una capa de *embeddings* de salida que produce un vector de tamaño n, igual al tamaño del vocabulario. De forma similar que en el caso anterior, se aplica una función *softmax* al vector

de salida para determinar la palabra con mayor probabilidad de ser la siguiente en la secuencia.

Este tipo de red neuronal tiene la ventaja de modelar naturalmente las secuencias de entrada, además de que para el caso del modelo de lenguaje permite tomar secuencias más largas de palabras sin perder la información del orden. No obstante, este modelo considera sólo el último estado oculto para codificar toda la secuencia para después predecir la siguiente palabra. Esta representación puede ser problemática, especialmente en secuencias muy largas, ya que se puede estar "olvidando" información importante de elementos al inicio de la secuencia y se favorece artificialmente a los últimos elementos.

Para subsanar este problema se han propuesto soluciones como los mecanismos de atención y arquitecturas más robustas para el modelado de secuencias, como la arquitectura *transformer*. Estos temas se tratarán en secciones posteriores.

2.2. Atención en redes neuronales

Los mecanismos de atención en redes neuronales se basan en la intuición de cómo los humanos perciben las imágenes y cómo se interpreta o entiende una secuencia de palabras. En el caso de las imágenes, la visión humana percibe sólo una pequeña porción en alta resolución, mientras que el resto de la imagen puede parecer borrosa [12]. Esto quiere decir, que se está enfocando o que la atención se concentra en un punto. En el caso del texto, las personas utilizan el contexto alrededor de una palabra para comprender su significado y su función en la oración [11]. Sin embargo, no todas las palabras del contexto tienen el mismo nivel de influencia. El punto por considerar es que en ambas tareas la atención se enfoca en distintos puntos de la entrada a distintos niveles.

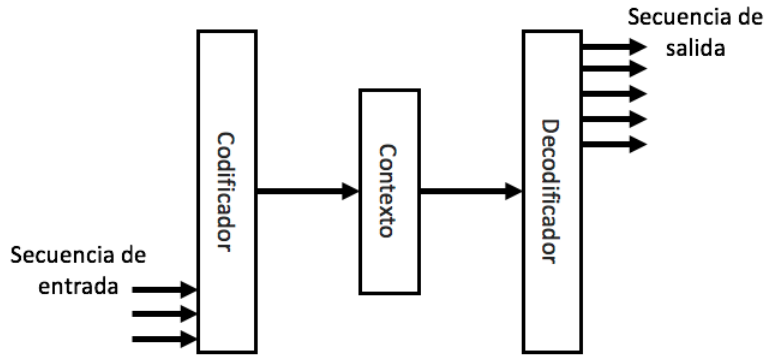
Con esta intuición, se puede definir la atención en redes neuronales como el mecanismo que genera un vector de pesos que permite ponderar los diferentes elementos de la entrada, basado en la importancia que tiene cada entrada para el elemento actual. En el caso del procesamiento del lenguaje, los mecanismos de atención tuvieron su origen en la arquitectura codificador-decodificador [36].

2.2.1. Arquitectura codificador-decodificador

Esta arquitectura de red neuronal tiene el propósito de recibir una secuencia de entrada y producir una secuencia de salida. Las secuencias pueden ser de una longitud arbitraria y no necesariamente deben de ser del mismo tamaño entre ellas. Estos modelos también se conocen como modelos secuencia-secuencia. Como su nombre lo indica, esta arquitectura está formada por dos componentes principales, el codificador y el decodificador como se observa en la Figura 2.3.

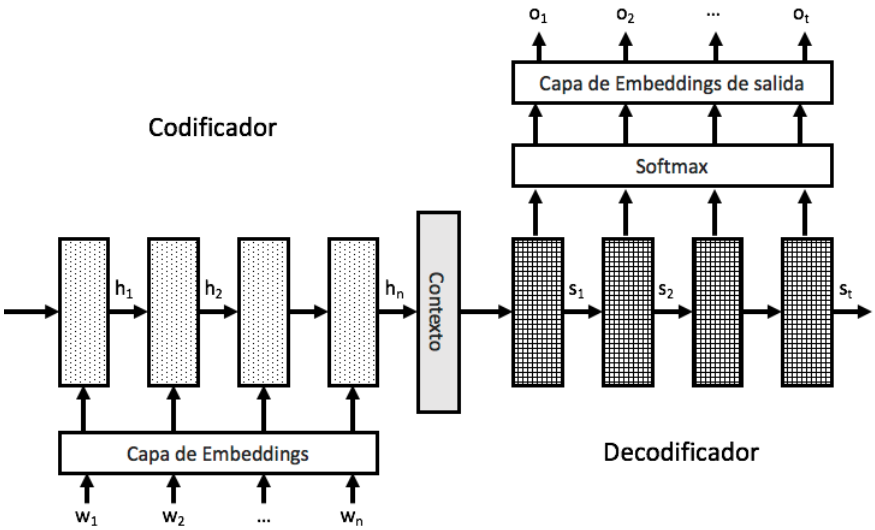
El codificador recibe como entrada una secuencia, y su tarea consiste en generar un vector de contexto que sintetice adecuadamente el contenido de la secuencia, en el caso del texto, este vector es una representación semántica. El decodificador toma como entrada el vector de contexto generado por el codificador. Considerando esta entrada, se produce una secuencia de salida hasta que se obtenga un elemento que indique el final de la secuencia.

Figura 2.3: Arquitectura codificador-decodificador. Toma una secuencia de entrada y produce una secuencia de tamaño arbitrario y no necesariamente igual al tamaño de la entrada.



Tanto el codificador como el decodificador se pueden implementar usando redes recurrentes. En este caso, el vector de contexto corresponderá al último estado oculto del codificador (Figura 2.4). Esta estrategia para crear el vector de contexto tiene desventajas en la representación de secuencias largas: normalmente se pierden detalles de la secuencia, principalmente al inicio de esta. Entre más elementos tenga la secuencia de entrada es más complicado que un solo vector de dimensión fija sea capaz de capturar todos los aspectos y relaciones implícitas de la entrada.

Figura 2.4: Arquitectura codificador-decodificador implementada con una red neuronal recurrente. El último estado oculto del codificador corresponde al vector contexto.

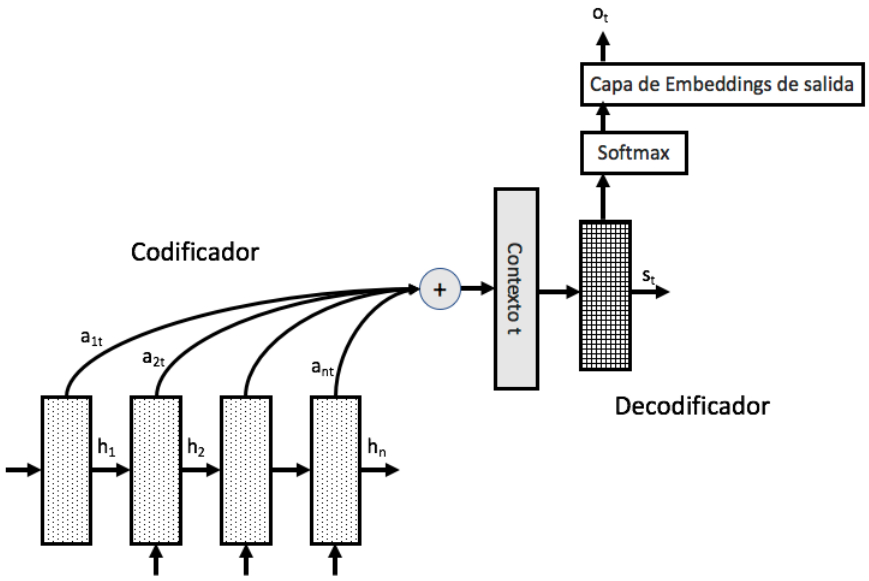


2.2.2. Atención en modelos codificador-decodificador

Considerando las limitantes de la codificación de un único vector contexto, se propuso un mecanismo de atención [1]. La idea de la atención es que el vector de contexto no sea

únicamente el último estado oculto del codificador, si no que para cada salida del decodificador se crea un contexto personalizado que toma en cuenta todos los estados ocultos del codificador. De esta forma toda la secuencia de entrada influye, en distinta medida, para la creación del vector de contexto (Figura 2.5).

Figura 2.5: Mecanismo de atención en redes recurrentes. Propone crear un vector contexto diferente para cada estado en el decodificador, tomando en cuenta todos los estados ocultos del codificador.



El codificador puede estar implementado con una red recurrente. Cada elemento de la secuencia de entrada genera un estado oculto h_i. Cada estado oculto en el decodificador corresponde a la palabra de salida j y está dada por $s_j = f(s_{j-1}, y_{j-1}, c_j)$. Donde c_j es el vector contexto. El vector contexto es la suma pesada de los estados ocultos de la secuencia de entrada. Los pesos de esta suma corresponden a *scores* de alineación.

Los pesos a_{ij} asignan un valor dependiendo qué tan bien alineados están la entrada i y la salida j, basado en una métrica específica. En [1] el *score* se calcula utilizando una capa completamente conectada parametrizada con pesos entrenables. La matriz de *scores* se puede visualizar para identificar la correlación entre las palabras de entrada y las de salida. En la Figura 2.6 (tomada de [1]) se muestra un ejemplo de mapa de atención calculado para una traducción automática. Se tiene una oración en francés y su traducción al inglés. En cada eje se listan los *tokens* del par de oraciones, la intersección entre los *tokens* muestra el peso entre ese par de *tokens*. Entre más claro es el color mayor el peso calculado para esa pareja. El mapa de atención resalta los *tokens* que se tomaron en cuenta para traducir cada palabra de un idioma al otro.

La atención significa que la relación entre las palabras ya no sólo depende de la distancia entre ellas, si no que se pueden representar relaciones más complejas y lejanas. Se han propuesto diversos mecanismos para establecer el *score* de alineación de los elementos de una secuencia, en la Tabla 2.1 se detallan algunos de estos mecanismos.

Figura 2.6: Visualización de los pesos calculados en la tarea de traducción automática. Estos pesos indican qué tan importante es un elemento de la entrada durante la generación de un elemento de la salida. Imagen tomada de [1].

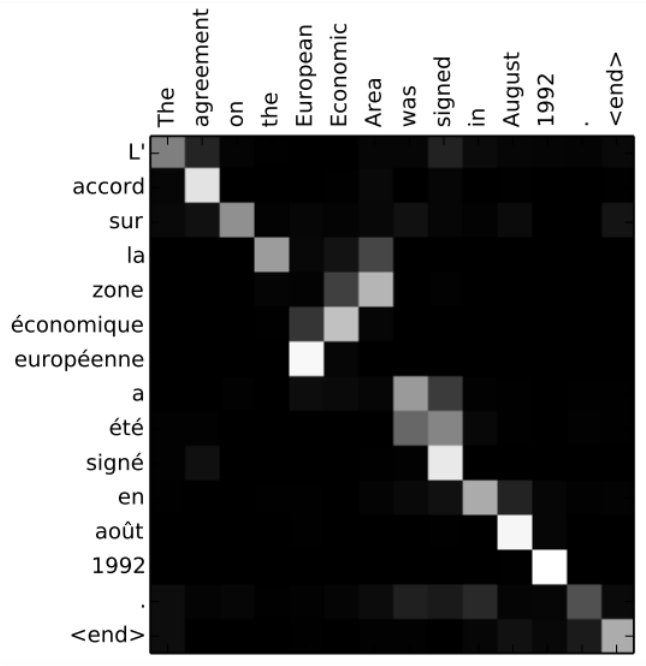


Tabla 2.1: Algunas funciones para calcular el *score* entre un par de elementos de secuencias distintas.

	Score	Artículo
Contenido	$score(s_t, h_i) = cosine[s_t, h_i]$	[14]
Sumativo	$score(s_t, h_i) = v_a tanh(W_a[s_t; h_i])$	[1]
Localización	$softmax(W_a s_t)$	[20]
General	$score(s_t, h_i) = s_t W_a h_i$	[20]
Multiplicativo	$score(s_t, h_i) = s_t h_i$	[20]
Multiplicativo escalado	$score(s_t, h_i) = \frac{s_t h_i}{\sqrt{n}}$	[38]

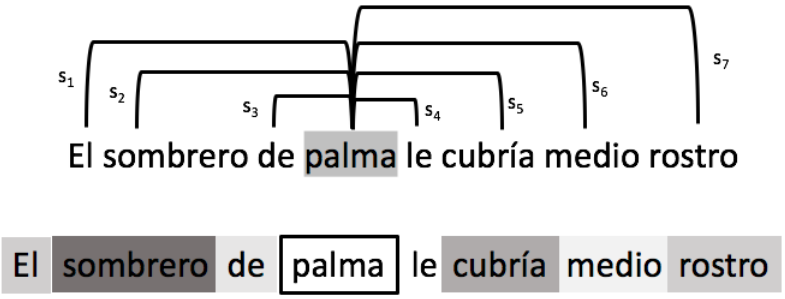
También se pueden clasificar los mecanismos de atención considerando los elementos de la secuencia que se evalúan para calcular los *scores*:

- Global. Se atiende a la secuencia entera.
- Local. Se atiende a una porción de la secuencia.
- Auto-atención. Evalúa los pesos considerando la misma secuencia. Se puede utilizar cualquier mecanismo para evaluar los *scores*, simplemente se evalúa sobre la misma secuencia.

2.2.3. Auto-atención

Este mecanismo consiste en calcular un *score* de atención tomando en cuenta un elemento de una secuencia y evaluándolo contra otro elemento de la misma secuencia, incluido el mismo elemento en cuestión (Figura 2.7). En el caso del texto, este mecanismo permite evaluar las relaciones que existen entre palabras de la misma secuencia, no solo entre la de entrada y la de salida. Con esto se obtiene más información de las relaciones semánticas y sintácticas dentro de la secuencia, lo cual resulta en representaciones más ricas de una oración.

Figura 2.7: Auto-atención. Este mecanismo obtiene los *scores* de alineamiento entre una palabra y las demás palabras de la misma secuencia, incluyendo la palabra en cuestión. De esta forma se sabe que tan importantes son las demás palabras para cada una de ellas.



2.3. Transformers

Las redes recurrentes son de mucha utilidad en tareas de procesamiento del lenguaje natural, modelan de forma natural la esencia secuencial de un texto. Sin embargo, tienen algunas limitaciones, es complicado modelar relaciones entre palabras, la distancia entre las palabras es linear y dada su naturaleza secuencial no se pueden paralelizar. Los mecanismos de atención ayudan a reducir los dos primeros problemas.

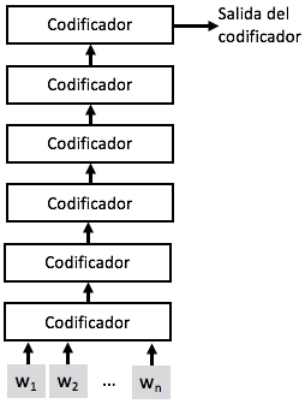
La arquitectura *transformer* fue propuesta por [38] para resolver el problema de secuencia-secuencia, permitiendo paralelizar el procesamiento de la entrada. Este modelo se basa en auto-atención y también en la atención entre las salidas del codificador y las salidas del decodificador. Como método para calcular el *score* de alineamiento, se utiliza el producto punto escalado entre la dimensión de los vectores de entrada.

La arquitectura está formada por un codificador y un decodificador. El codificador recibe una secuencia de entrada y produce una secuencia de salida. El decodificador tiene un mecanismo de atención sobre las salidas del codificador para producir la secuencia de salida del modelo.

El codificador está formado por varios bloques codificadores apilados. El primer codificador toma la entrada del modelo que puede provenir de una capa de *embeddings*. Codificadores subsecuentes toman como entrada la salida del codificador anterior. En el artículo original [38] se apilan 6 codificadores. El último bloque codificador produce la salida del codificador

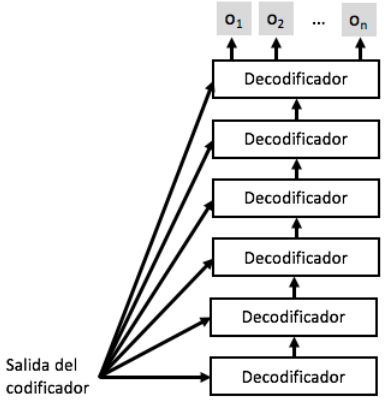
(Figura 2.8).

Figura 2.8: Codificador *transformer*, está formado por 6 bloques codificadores apilados.



Por su parte, el decodificador también se forma por bloques decodificadores apilados. Cada bloque decodificador toma las salidas del codificador y la salida del bloque decodificador anterior (Figura 2.9).

Figura 2.9: Decodificador *transformer*, está formado por 6 bloques decodificadores apilados.

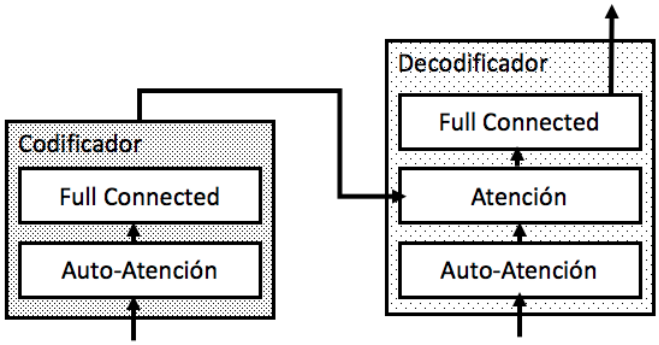


Cada bloque codificador está formado por la misma estructura, una capa de auto atención seguida de una capa completamente conectada (Figura 2.10). Sin embargo, los bloques no comparten los mismos pesos.

En el caso de los decodificadores, estos están formados también por una capa de auto-atención y una capa completamente conectada. Adicionalmente, entre las capas mencionadas, se encuentra una capa de atención entre las salidas de codificador y cada uno de los bloques decodificadores (Figura 2.10). Esta configuración permite al decodificador atender a los elementos de la secuencia de entrada mientras se genera un elemento de salida.

En el codificador los elementos de la entrada son procesados en paralelo. Pasan por cada

Figura 2.10: Estructura de bloques codificadores y de los bloques decodificadores.

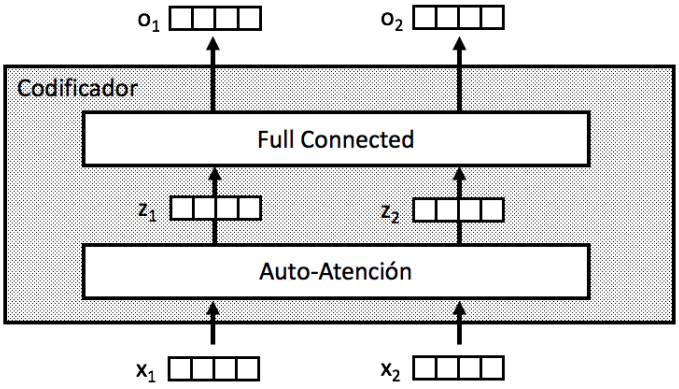


bloque codificador de forma independiente y a la salida del codificador se obtiene una secuencia del mismo tamaño, pero cada elemento de la secuencia puede ser de una dimensión distinta a los *embeddings* de entrada.

2.3.1. Capa de auto-atención

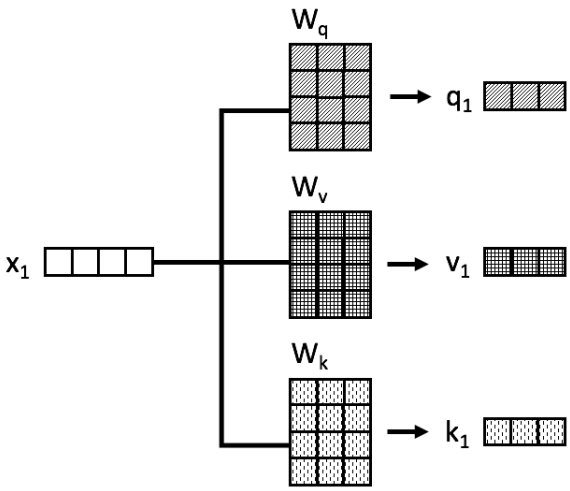
En esta capa se obtiene un vector por cada elemento de la secuencia (Figura 2.11). Este vector es el resultado de una suma pesada de todos los elementos de la secuencia, utilizando un conjunto de pesos que se calculan tomando como entrada el elemento en cuestión y cada uno de los demás elementos, incluido el actual. De esta forma, se calcula un peso para cada elemento de la secuencia. El artículo [38] propone un conjunto de abstracciones para calcular los pesos. Al evaluar el elemento i de la secuencia este elemento fungirá como *query* Q_i. Cada uno de los demás elementos, incluido el actual, tomarán el rol de *Key* K_j y valor V_j.

Figura 2.11: Detalle del bloque codificador. Cada entrada es procesada en paralelo de forma independiente a las demás entradas. Primero se procesan en la capa de auto-atención y luego pasan por la capa completamente conectada.



Cada vector, dependiendo del rol que le corresponda, será transformado utilizando una matriz de pesos entrenables, existe una matriz para transformar a *query*, otra para *key* y otra para valores. Este conjunto de matrices lo que pretenden es transformar el espacio vectorial de la entrada original para que se ajuste al rol que le corresponda (Figura 2.12).

Figura 2.12: Matrices de pesos. Estas matrices transforman el espacio de los vectores de entrada, para que cumplan con cada uno de los roles Q, K y V.



$$Q_i = x_i W_q$$

$$K_i = x_i W_k$$

$$V_i = x_i W_v$$

Una vez que se calculan estos vectores, se debe obtener el *score* de atención. Este *score* indica qué tan importantes son los elementos de la secuencia con respecto al elemento actual. El *score* se calcula obteniendo el producto punto del *query* Q_i y el *key* K_j. Posteriormente este *score* se debe normalizar dividiéndolo entre $\sqrt{n}$ donde n es la dimensión de los vectores. A continuación, se aplica una función *softmax* a todos los pesos asociados al vector x_i.

$$a'_{ij} = \frac{Q_i K_j^\top}{\sqrt{n}}$$

$$a_{ij} = \frac{exp\ a'_{ij}}{\sum_j exp\ a'_{ij}}$$

El siguiente paso es hacer la suma pesada de los vectores valor V_j considerando los pesos calculados en el punto anterior. Esto da como resultado un vector Z_i con el que se alimentará la capa completamente conectada.

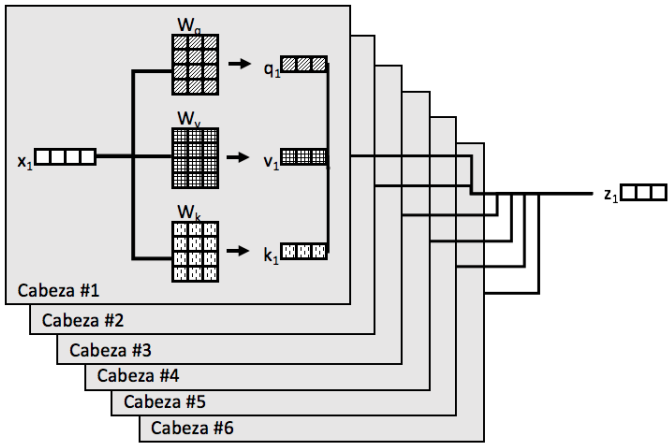
$$Z_i = \sum_{j=1}^{n} a_{ij} V_j$$

2.3.2. Atención multi-cabeza

El vector de salida Z_i de la capa de auto-atención está formada por la suma pesada de los demás vectores de la secuencia. Este vector logra capturar características específicas de las relaciones entre las palabras.

El artículo [38] propone utilizar múltiples cabezas de atención. Estas cabezas consisten en el mismo mecanismo de auto-atención descrito anteriormente, cada cabeza tiene sus propias matrices para la transformación de los vectores de entrada a vectores Q, K y V. Se calculan en paralelo los vectores de auto-atención y se obtienen n vectores Z, uno por cada una de las n cabezas. Para obtener un vector único por cada elemento de la secuencia, se concatenan los vectores de las n cabezas y utilizando una matriz de pesos de $(n \times d) \times d$, se obtiene un vector único. Este será el vector Z con el que se alimenta la capa completamente conectada (Figura 2.13).

Figura 2.13: Mecanismo de atención multi-cabeza. Cada cabeza tiene un mecanismo de atención independiente, tiene sus propias matrices de pesos W_q, W_k y W_v.



2.3.3. Orden de la secuencia

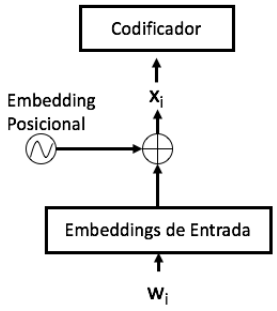
Hasta ahora, se ha presentado el procesamiento dentro de los bloques codificadores que implica que este puede ser llevado a cabo de forma paralela. Cada elemento de la secuencia puede ser procesado de forma independiente sin necesidad de esperar al anterior. Pero esto significa que el orden de los elementos se pierde, para el modelo es indiferente la posición que guardan en la secuencia. Por esta razón el modelo *transformer* propone utilizar un *embedding* posicional. Este *embedding* es un vector que debe reflejar la distancia existente entre los elementos de la secuencia. El artículo que presenta la arquitectura *transformer* propone utilizar una función senoidal para crear los vectores posicionales. Estos vectores se suman a los generados en la capa de *embeddings* (Figura 2.14). De esta manera el modelo tiene información de la posición que guardan los elementos de la secuencia y puede aprender las relaciones entre ellos tomando en cuenta este factor, además de otras variables.

2.3.4. Decodificador

El decodificador funciona de forma muy similar al codificador. Toma como entrada las palabras generadas anteriormente en el mismo decodificador. La capa de auto-atención funciona como se describió anteriormente.

Cada bloque decodificador tiene una capa extra de atención (Figura 2.10). Esta capa de atención se enfoca en computar la relación entre la salida del codificador y los elementos de

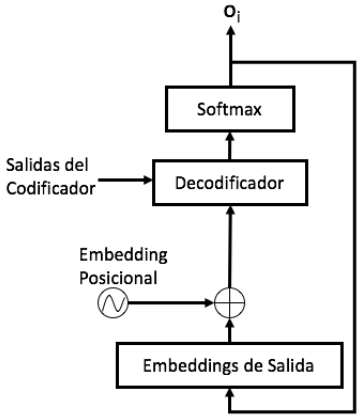
Figura 2.14: *Embedding* posicional. Se suma al *embedding* de entrada un vector que indica la posición del elemento de la secuencia de entrada.



entrada del decodificador. Funciona de forma muy parecida a la capa de auto-atención, excepto que los valores y las *keys* (V_j y K_j) corresponden a elementos de salida del decodificador.

A la salida del decodificador se tiene una capa *softmax* para determinar la palabra que se debe generar. La palabra generada se alimenta como entrada al decodificador (Figura 2.15). Este proceso secuencial se repite hasta que se genere una palabra especial que indica el final de la oración.

Figura 2.15: Generación de la secuencia de salida en el decodificador.



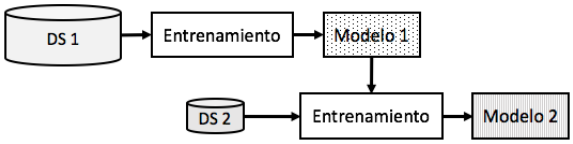
2.4. Aprendizaje por transferencia

Un modelo de aprendizaje automático se construye para resolver una tarea específica en un domino determinado. Para este propósito, es necesario contar con ejemplos que le permitan aprender patrones en los datos. No obstante, el aprendizaje profundo es muy demandante en la cantidad de datos necesaria para entrenamiento, para que un modelo de aprendizaje profundo pueda generalizar requiere necesariamente de una gran cantidad de datos anotados de forma manual. Estos modelos pueden tener un excelente rendimiento en un dominio, pero es muy común que al tratar de resolver el mismo problema pero en distinto dominio,

se sufra una degradación importante de su comportamiento. También es probable que en el nuevo dominio no se cuente con los datos necesarios para entrenar. Por estas limitaciones en la cantidad de datos disponibles, es de utilidad poder aprovechar un modelo entrenado previamente para aplicarlo en una tarea diferente o en un dominio distinto.

Lo que se busca en el aprendizaje por transferencia es reutilizar el conocimiento que tiene un modelo para poder aplicarlo a una tarea similar. De esta forma, si en la tarea origen se cuenta con la cantidad de datos necesaria para entrenar el modelo, este se puede aprovechar en tareas o dominios que no tienen tantos datos disponibles. En este caso sólo es necesario ajustar mínimamente el modelo pre-entrenado para adaptarse a la nueva tarea (Figura 2.16).

Figura 2.16: Aprendizaje por transferencia. Se aprovecha el conocimiento de un modelo entrenado en otra tarea.



Existen dos formas de aprovechar el conocimiento de un modelo pre-entrenado:

- Extracción de características. Se aprovecha el modelo pre-entrenado para obtener una representación de los datos y esta representación se utiliza en un modelo de aprendizaje automatizado para resolver una tarea específica.
- Ajuste fino. Se integra el modelo pre-entrenado en un nuevo modelo que tiene el objetivo de resolver una tarea diferente. Se entrena este nuevo modelo y se ajustan los pesos del modelo pre-entrenado para adaptarse a la tarea específica. Normalmente se utiliza una tasa de aprendizaje muy pequeña y apenas unas pocas épocas para ajustar el nuevo modelo.

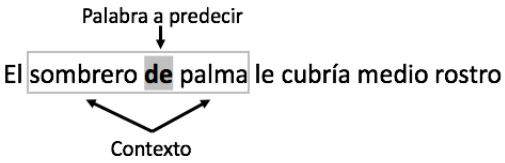
2.4.1. Modelos pre-entrenados de lenguaje

Si bien el aprendizaje por transferencia se plantea como la posibilidad de reusar un modelo que originalmente se creó para resolver una tarea específica, en la actualidad en PLN se han propuesto diversos modelos de lenguaje que no tienen el objetivo de resolver una tarea, ni tienen por sí solos una aplicación específica, más bien estos modelos de lenguaje lo que pretenden es recolectar conocimiento lingüístico, y utilizar lo aprendido durante el pre-entrenamiento para resolver otras tareas. Para este propósito es necesario definir un conjunto de tareas pretexto que servirán para delimitar el conocimiento que tendrá este modelo y que será de utilidad al momento de ajustarlo a una tarea. A continuación, se listan algunas de las tareas pretexto que se han propuesto para el pre-entrenamiento de modelos del lenguaje en PLN.

2.4.1.1. Predicción de una palabra central

Esta tarea plantea obtener de un texto una ventana con n palabras y se debe predecir la palabra central de dicha ventana. Para lograr esta predicción, se consideran las palabras que están antes y después (Figura 2.17). Esta tarea se utiliza en [23] para entrenar el algoritmo CBOW.

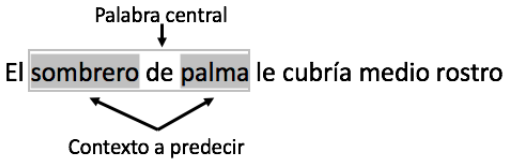
Figura 2.17: Tarea de predicción de palabra central.



2.4.1.2. Predicción del contexto

Se selecciona una ventana de texto y se debe seleccionar una palabra central. El objetivo es predecir el contexto basado en esta palabra central (Figura 2.18). Esta estrategia se utiliza en [23] en el algoritmo *skip-gram*.

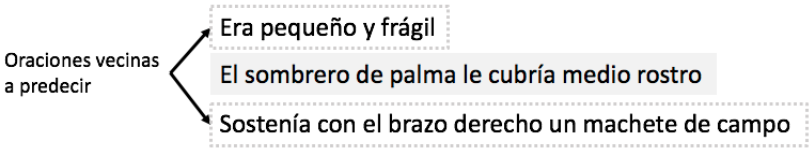
Figura 2.18: Tarea de predicción de contexto utilizando la palabra central.



2.4.1.3. Predicción de oraciones vecinas

Se seleccionan 3 oraciones consecutivas de un texto. Se deben predecir las dos oraciones circundantes dada la oración central (Figura 2.19).

Figura 2.19: Predicción de oraciones vecinas considerando una oración central.



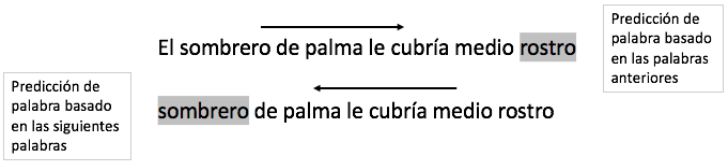
2.4.1.4. Modelado de lenguaje

Esta tarea consiste en predecir la siguiente palabra dada una secuencia previa de palabras. También existe la variante de predecir una palabra considerando las siguientes palabras de la secuencia (Figura 2.20).

2.4.1.5. Modelado enmascarado de lenguaje

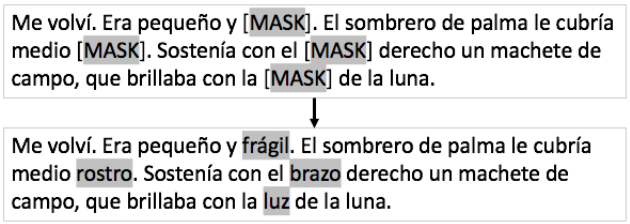
Dado un texto, en esta tarea se propone enmascarar aleatoriamente un subconjunto de palabras. Se deben predecir las palabras enmascaradas considerando el contexto que las rodea

Figura 2.20: Predicción de próxima palabra. También se considera el caso en el cual se predice una palabra considerando las siguientes palabras.



(Figura 2.21).

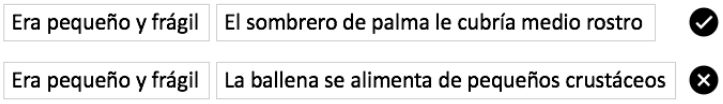
Figura 2.21: Modelo de lenguaje enmascarado. Se predicen palabras que fueron enmascaradas aleatoriamente.



2.4.1.6. Predicción de próxima oración

De un texto se extraen un par de oraciones consecutivas, este par de oraciones representará un ejemplo positivo. Para generar un ejemplo negativo se selecciona una pareja de oraciones al azar que no sean adyacentes. El objetivo es predecir si un par de oraciones son consecutivas o no (Figura 2.22).

Figura 2.22: Predicción de próxima oración. Dado un par de oraciones, se debe predecir si son consecutivas o no.



2.4.1.7. Predicción del orden de las oraciones

Se seleccionan un par de oraciones consecutivas como ejemplos positivos. Para crear ejemplos negativos se invierten las parejas de oraciones de como aparecen realmente en el texto. Se debe predecir si una pareja de oraciones está en el orden adecuado. (Figura 2.23).

2.4.1.8. Predicción de oraciones

En un texto se eliminan al azar algunas oraciones. Esta tarea consiste en predecir las oraciones perdidas. (Figura 2.24).

Figura 2.23: Predicción orden de oraciones. Predecir si un par de oraciones están en el orden correcto.

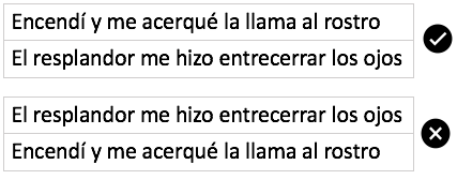
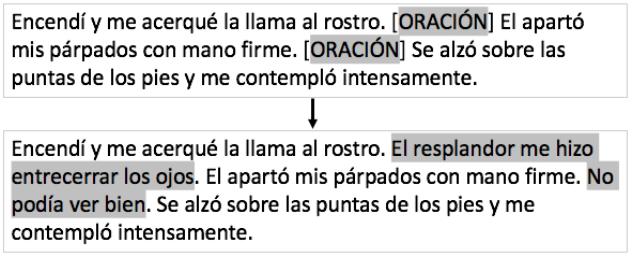


Figura 2.24: Predicción de oraciones.



Todas estas tareas pretexto comparten la característica de que son auto-supervisadas. Los datos de entrada no están etiquetados manualmente, pero se pueden generar ejemplos etiquetados de forma automática utilizando únicamente el texto disponible en un corpus. Esto es en especial útil considerando que actualmente existe una gran cantidad de datos textuales. Muchos de los modelos pre-entrenados del estado del arte se entrenan en cantidades masivas de texto, aprovechando las bondades de este tipo de aprendizaje. Esto permite que los modelos generados capturen una gran variedad de conocimiento lingüístico.

2.5. BERT

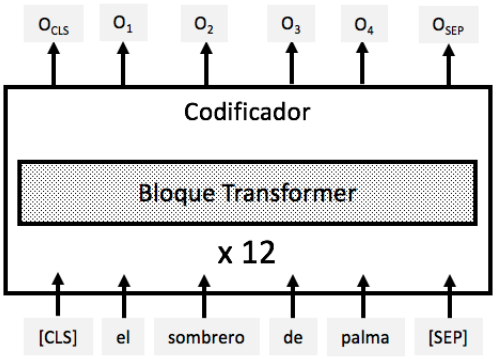
BERT es un modelo de lenguaje pre-entrenado, propuesto por *Google* en 2018 [8]. Se basa en la arquitectura *transformer*, utiliza solamente el codificador. Está pre-entrenado en *Wikipedia* y el *BookCorpus*. Este pre-entrenamiento le permite ajustarse a diversas tareas de PLN y desde su aparición se colocó como el estado del arte en muchas de estas tareas.

2.5.1. Arquitectura

BERT consiste en un conjunto de bloques codificadores apilados, tal como se propone en el artículo de la arquitectura *transformer*. En [8] se proponen dos versiones de la arquitectura, la versión base y la grande. La versión base consiste en 12 capas, 12 cabezas de atención y una dimensión de estado oculto de 768, tiene un total de 110 millones de parámetros. La versión grande tiene 24 capas, un estado oculto de dimensión 1024 y 16 cabezas de atención, esto da un total de 340 millones de parámetros (Figura 2.25).

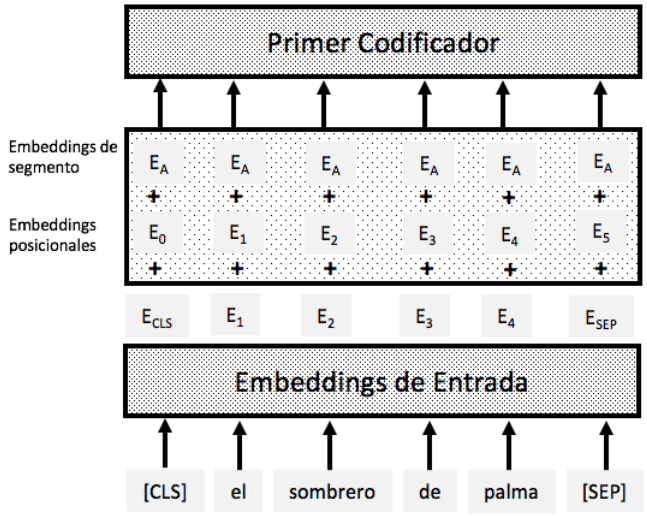
Las entradas consisten en una secuencia de *tokens*. Los *tokens* pueden corresponder a una o dos oraciones. Al inicio de la secuencia se usa el *token* especial *CLS* y para marcar la separación entre las dos oraciones y el final de la secuencia, se utiliza el *token* especial *SEP*.

Figura 2.25: Arquitectura de *BERT*. Consiste en 12 bloques codificadores (24 en la versión grande) apilados.



Las palabras y los *tokens* de control pasan por una capa de *embeddings*. A estos *embeddings* se les suma un *embedding* posicional para que el modelo pueda distinguir el orden de las palabras. Adicional a este *embedding*, se adiciona un *embedding* de segmento que sirve para indicar qué *tokens* pertenencen a la primera o segunda oración (Figura 2.26).

Figura 2.26: Entradas al modelo *BERT*. Además de los *embeddings* de entrada y los posicionales, se agrega un *embedding* de segmento que indica si el *token* pertenece a la primera o segunda oración.



En el último bloque codificador, se genera el último estado oculto, que es útil para tareas específicas y para el pre-entrenamiento.

2.5.2. Pre-entrenamiento

BERT es pre-entrenado en 2 tareas pretexto. La primera es el modelado enmascarado de lenguaje. Se selecciona el 15 % de los *tokens* en un texto y el modelo debe predecir las palabras enmascaradas utilizando las palabras previas y posteriores. Este tipo de pre-entrenamiento le

permite a *BERT* aprovechar la auto-atención bidireccional y aprender de las relaciones entre las palabras.

La segunda tarea de pre-entrenamiento es la predicción de la próxima oración. Se eligen parejas de oraciones de tal forma que la mitad sean oraciones consecutivas y la otra mitad no lo sea. Se debe predecir si un par de oraciones son consecutivas o no. Esta tarea le permite al modelo aprender las relaciones entre un par de oraciones.

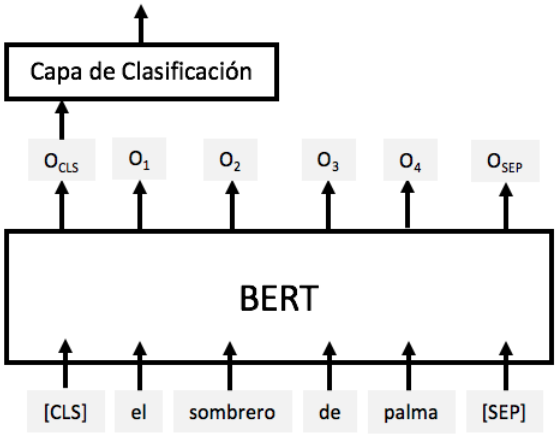
El pre-entrenamiento se llevó a cabo con *Wikipedia*, con 2.5 mil millones de palabras y *BookCorpus* que contiene 800 millones de palabras. Este proceso se realizó en 4 días utilizando un conjunto de *TPUs*.

2.5.3. Ajuste fino

BERT se puede utilizar en diversas tareas de PLN, utilizando los pesos aprendidos durante el pre-entrenamiento. Solo es necesario agregar capas a la entrada o a la salida. Después de realizar estas modificaciones se debe hacer un ajuste en los pesos de las nuevas capas y de *BERT*, entrenando con los datos de la tarea en cuestión, por unas pocas épocas. Normalmente, se recomienda usar una tasa de aprendizaje muy baja, de otra forma se corre el riesgo de perder el conocimiento adquirido durante el pre-entrenamiento.

Para tareas de clasificación o que requieren un vector que resuma la secuencia de palabras, se recomienda utilizar la salida correspondiente al *token CLS*. Esto es debido a que precisamente esta salida es la que se utiliza para la clasificación durante el pre-entrenamiento, este vector es capaz de capturar el contenido de toda la secuencia. Sin embargo, es necesario ajustar para obtener mejores resultados en tareas específicas (Figura 2.27).

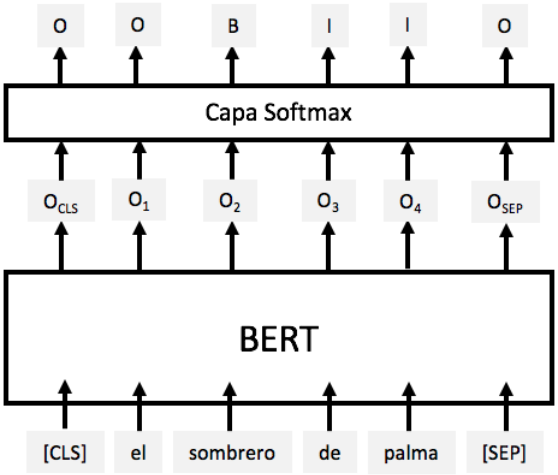
Figura 2.27: Para crear un clasificador con *BERT* se puede usar la salida del *token CLS*. Esta salida se utiliza en una capa de clasificación.



Utilizando las demás salidas se puede crear un modelo de etiquetado de secuencias, útil para tareas como *POS* y *NER* (Figura 2.28). También se puede definir una distribución de probabilidad sobre toda la secuencia de salida, para definir el inicio y el fin de la porción de

texto que contiene la respuesta buscada en la tarea de pregunta-respuesta.

Figura 2.28: Agregando una capa *softmax* sobre las salidas de *BERT* es posible crear un etiquetador de secuencias. En este caso se muestra un etiquetador de entidades.



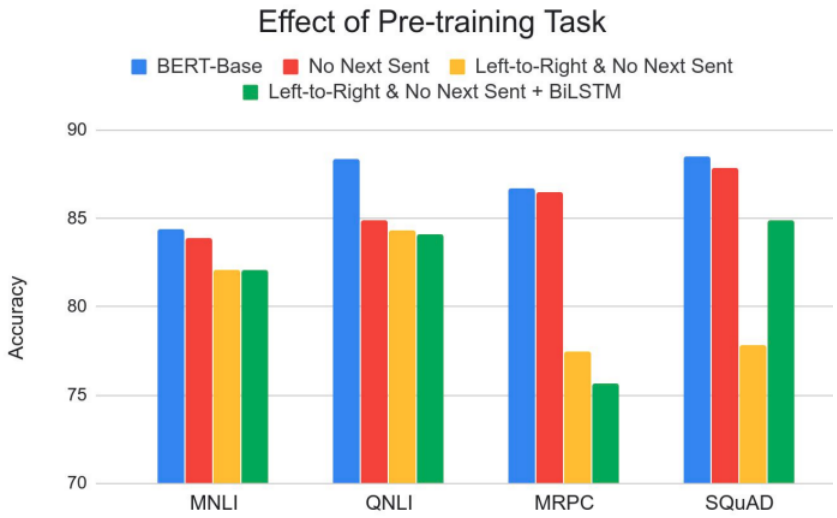
2.5.4. Otros *transformers*

Existen diversas variantes de *BERT* que pretenden mejorar algunas debilidades del modelo original, ajustando hiperparámetros, mejorando las tareas de pre-entrenamiento o proponiendo otras tareas pretexto. Se ha visto que utilizando diferentes tareas pretexto el modelo pre-entrenado se ajusta más fácilmente a diferentes tareas, depende qué tan intuitivamente cercana es la tarea pretexto a la tarea final. Un buen ejemplo de cómo afectan las tareas pretexto en diversas tareas lo presentan los autores de [8]. En la Figura 2.29 se puede observar que el usar distintas tareas pretexto influye en qué tan bien se ajusta *BERT* a las diversas tareas específicas de PLN. No todas las tareas pretexto impactan de la misma manera en todas las tareas específicas, algunas son muy trascendentes y otras si bien ayudan a mejorar el rendimiento final, no afecta demasiado el hecho de prescindir de estas tareas pretexto durante el pre-entrenamiento.

Hay algunas propuestas que lo que buscan es reducir el uso de recursos de *BERT*, tal es el caso de *DistilBERT* [33] que utiliza un método conocido como destilación de una red neuronal, para aproximar las salidas de un modelo base, en este caso *BERT*, con un modelo con menos parámetros. *DistilBERT* es 40 % más pequeño y 60 % más rápido que el modelo base de *BERT* y mantiene 95 % del rendimiento original.

Otras variantes de *BERT* experimentan con diversas tareas pretexto de aprendizaje auto-supervisado, algunas de las cuales fueron mencionadas anteriormente. Como se mencionó, cada tarea pretexto aprende un aspecto distinto del lenguaje. Por esta razón, cada tarea pretexto es adecuada en distinto nivel para resolver diferentes tareas de PLN. Incluso combinar varias tareas pretexto tiene un efecto positivo en diversas tareas. Como resultado, los modelos basados en la arquitectura *transformer* que utilizan el pre-entrenamiento con aprendizaje auto-supervisado, han tenido excelentes resultados, pero es complicado que el

Figura 2.29: Efecto de diferentes tareas de pre-entrenamiento en tareas específicas. Imagen tomada de [8]



modelo pre-entrenado se adapte a todos los escenarios.

También se pueden encontrar otros modelos basados en la arquitectura *transformer* que proponen mejorar su rendimiento a través de un modelo mucho más grande y el uso de grandes cantidades de información durante el preentrenamiento. Al momento de la escritura de este trabajo el modelo que lleva esta idea al extremo es *GPT-3* [3]. Este modelo es la tercera iteración del modelo llamado *Generative Pre-trained Transformer* 3 que fue creado por el laboratorio de investigación en inteligencia artificial *OpenAI*. Los autores proponen un modelo que cuenta con 96 capas, 96 cabezas de atención en cada capa y una dimensión de estado oculto de 12,288, dando un total de 175 mil millones de parámetros. Fue entrenado con un grupo de corpus que en total tienen 300 mil millones de *tokens*. Trabajos anteriores mostraron los beneficios de utilizar más datos para el pre-entrenamiento. En el caso de *GPT-3* utilizan esta estrategia, pero también proponen un modelo de dimensiones no vistas anteriormente. Estos dos factores colocan a este modelo de lenguaje como el estado del arte en varias de las tareas de PLN. Además, en varios demos que están disponibles se muestra que *GPT-3* es capaz de realizar diversas tareas sin ningún ajuste previo, puede clasificar documentos, responder a preguntas e incluso resolver operaciones aritméticas sencillas presentadas como lenguaje natural. Considerando estos hechos es evidente que tanto el tamaño del modelo como la cantidad de información usada durante el entrenamiento ayudan a que se aprendan muchos aspectos del lenguaje que pueden ser utilizados de forma exitosa en tareas de PLN.

2.6. Resumen

En este capítulo se expusieron la terminología y conceptos relacionados con los modelos de lenguaje neuronales. Estos han demostrado ser una gran aportación en el área de PLN, utilizando diversas arquitecturas de redes neuronales son capaces de aprender varias características del lenguaje y permiten que este conocimiento extraído de grandes cantidades de texto pueda ser usado para resolver otras tareas no directamente relacionadas con los objetivos durante el pre-entrenamiento.

Un avance importante en aprendizaje profundo son los mecanismos de atención. Estos mecanismos tienen la intención de ponderar la entrada a un modelo, con lo cual se pueden establecer los puntos que son más importantes para la toma de decisión de una red neuronal. Estos mecanismos han sido de gran utilidad no sólo en PLN, si no también en el procesamiento de imágenes y de audio.

Agrupando los conceptos anteriores, nos encontramos con la arquitectura *transformer* que, utilizando mecanismos de atención más robustos, es capaz de aprender varios aspectos del lenguaje, como dependencias sintácticas o roles semánticos de las palabras. Entrenando estos modelos en cantidades masivas de texto utilizando hardware especializado, es posible crear modelos pre-entrenados de lenguaje que logran mejorar sustancialmente el estado del arte en varias tareas. Modelos como *BERT* y *GPT-2* son dos buenos ejemplos de este tipo de modelos. En la actualidad se continúa explorando cómo mejorar estos modelos, algunos intentan ajustar mejor los hiper-parámetros otros utilizan tareas pretexto diferentes o con una configuración distinta. Pero una estrategia que ha sido explorada ampliamente es el aumentar más y más la cantidad de texto en el pre-entrenamiento, lo cual aumenta exponencialmente los requerimientos computacionales, todo esto para conseguir mejoras modestas en el rendimiento de los modelos resultantes.

Capítulo 3

Verificación automática de hechos

En este capítulo se presenta un panorama global del problema de la detección de información falsa en medios digitales utilizando métodos automáticos. Primero se plantea el fenómeno general de la información falsa, que se presenta en forma de noticias falsas, bromas, rumores, etc. Posteriormente, se plantean varias formas que se han propuesto para atacar este problema, que no necesariamente ocupan herramientas tecnológicas. Por último, se define la tarea de la verificación automática de hechos, algunos métodos para resolverla y por supuesto el estado del arte. Se finaliza este capítulo exponiendo los métodos para evaluar la tarea que se estará tratando en este trabajo.

3.1. El problema de la información falsa

La distribución de información falsa o engañosa no es un fenómeno nuevo, ha ocurrido desde que aparecieron fuentes noticiosas como los periódicos. No obstante, la llegada de medios como el internet y las redes sociales, han provocado que la información se democratice, es decir, cualquier persona puede generar contenido cuyo objetivo sea informar. Ya sea que se busque crear un artículo informativo o que se desee expresar una opinión sobre un tema específico, solamente es necesario teclear unas cuantas palabras y publicar este texto en una red social. Este tipo de medios tienen muchos beneficios, pero también han provocado que el problema de la desinformación crezca exponencialmente. Este problema es de particular interés ya que puede tener consecuencias catastróficas [1], pero es muy difícil de atacar debido a fenómenos propios de las redes sociales y a los intereses económicos y políticos creados alrededor de la distribución de información falsa.

Dentro de las piezas de información que abundan en redes sociales se pueden identificar dos tipos principales: desinformación y mala información. Desinformación son piezas informativas falsas creadas intencionalmente, y dependiendo del objetivo de esta información en esta categoría se pueden identificar las noticias falsas, noticias satíricas y bromas. Por su parte, la mala información es información falsa pero que fue creada sin intención, este tipo de información generalmente ocurre en forma de rumores, y surgen muchas veces por mala interpretación de noticias y por las creencias populares.

Este tipo de información se encuentra en las redes sociales, una tierra fértil para su rápida distribución. En [34] se identifican algunos factores que facilitan la distribución de información

[1] `https://www.elfinanciero.com.mx/mundo/que-es-el-pizzagate-y-como-surgio`

falsa. Por un lado, se tienen los factores psicológicos, la información falsa aprovecha vulnerabilidades de las personas para favorecer su distribución:

- Realismo ingenuo. Las personas tienden a pensar que su visión u opinión de la realidad es la única acertada.
- Confirmación de tendencia. Los consumidores prefieren recibir información que confirma sus opiniones preexistentes.

También existen factores sociales que fomentan este fenómeno:

- Los generadores de información tienden a preferir el beneficio a corto plazo por sobre su reputación como medio confiable. Esto quiere decir que si la creación de contenido falso o engañoso le retribuye al medio de comunicación en el corto plazo, no tienen tanto problema con el impacto que seguramente traerá en su credibilidad.
- Los consumidores de información demandan información veraz y objetiva, pero al mismo tiempo tienen la necesidad de confirmar sus opiniones y creencias. Es por esta razón que medios de comunicación con una tendencia clara, tienen consumidores que demandan información con esta tendencia, lo que impulsa a estos medios de comunicación a seguir generando este tipo de contenido.

También las redes sociales cuentan con factores facilitadores. La simplicidad y bajo costo para crear un perfil en redes sociales ha favorecido la proliferación de cuentas dedicadas a atacar perfiles específicos y a la distribución de noticias falsas con el objetivo de obtener una retribución económica y política. La realidad es que existen organizaciones que se dedican a este tipo de tareas que son especialmente demandadas, por ejemplo, por actores políticos en tiempos electorales. Otro aspecto importante es que los usuarios de redes sociales usualmente siguen perfiles que piensan muy similar a ellos. En el artículo [29] se identifica este fenómeno como cámaras de eco. Dentro de las redes sociales se forman grupos de usuarios que comparten filias y fobias, y como resultado en estas comunidades se distribuye de forma acelerada información que cumple con sus necesidades. Esto genera pequeños ecosistemas en los cuales la información es muy limitada, normalmente sólo se comparte una visión única de un acontecimiento. Esta situación genera invariablemente que se acentúe la polarización en redes sociales.

3.2. Detección de información falsa

Por los motivos presentados anteriormente, es de suma importancia detectar información falsa en las redes sociales. Su detección oportuna puede limitar su distribución y mitigar el impacto negativo que estos elementos pueden tener en la sociedad. Sin embargo, esta tarea de detección temprana es una labor muy complicada, ya que la información falsa se presenta en varias formas: texto, imágenes que modifican la connotación, videos o todas estas formas juntas o en diferentes combinaciones. Además, estas piezas de información se han vuelto muy sofisticadas, tratando de atrapar la mayor cantidad posible de lectores. Esto hace que sea complicado diferenciar entre información verdadera y la que no lo es, incluso para un humano. Por este motivo hay muchos aspectos que se deben tomar en cuenta en esta tarea. El presente trabajo se enfoca en detectar información falsa en forma de texto. Estos textos pueden ser artículos noticioso, artículos científicos, publicaciones en redes sociales, mensajes de texto y otras presentaciones.

Para lograr la detección de información falsa se han propuesto diferentes soluciones, estas se pueden categorizar dependiendo del aspecto que toman en cuenta. A continuación, se presentan algunas de las estrategias identificadas por [34].

3.2.1. Métodos basados en el conocimiento

Estos métodos se basan en la idea de que la información falsa se estructura como un texto en el cual se pueden identificar afirmaciones las cuales no están apegadas a la realidad. Entonces, para verificar la información es necesario detectar estas afirmaciones y compro-barlas contra una base de conocimiento que servirá como fuente de verdad fundamental. Dependiendo del sentido que tenga esta fuente de verdad se podrá determinar si la afirmación es verdadera o no. Esta forma de clasificar la información también se conoce como verificación de hechos. Actualmente es muy común que esta tarea de detección se haga de forma manual, para lo cual es necesario que un grupo de expertos se encargue de encontrar las afirmaciones susceptibles de ser verificadas. Una vez que se tienen estas afirmaciones los verificadores deben buscar en diferentes fuentes las posibles evidencias que apoyen o refuten la afirmación en cuestión. Con las afirmaciones detectadas y la evidencia, para el verificador humano será posible determinar si la información es falsa. No obstante, esta forma de verificación es muy demandante de recursos humanos, que deben estar capacitados, y también de tiempo, por lo que esta estrategia suele ser muy cara. Aún así, esta es la forma más común para la verificación de información, existen sitios especializados para tal fin [23].

Otra forma de aprovechar el conocimiento colectivo es utilizar una estrategia llamada *crowd-sourcing*. Lo que se propone es que la comunidad es la que ayuda a verificar la información a través de sitios de discusión de noticias.

Por último, están los métodos computacionales los cuales proponen realizar la tarea de verificación de forma automática. Esto se logra utilizando diversas herramientas tecnológicas que consideran diferentes aspectos de la información presentada. El trabajo actual se centra en la verificación automática de hechos, más adelante se define formalmente esta tarea.

3.2.2. Métodos basados en el estilo

Es bastante común que el texto que contiene información falsa utilice un lenguaje muy particular, este tipo de lenguaje busca atraer la mayor cantidad de personas. Una de las formas que se utiliza es intentar aprovecharse del morbo, con títulos con información incompleta o sensacionalista que invitan a leer un texto informativo que no siempre cumple con las expectativas del título.

Una estrategia para detectar información falsa es precisamente detectando el estilo que normalmente se usa en este tipo de texto. Extrayendo las características como el vocabulario usado, los calificativos, la intensidad sentimental de las palabras usadas, entre otras, se puede tener una buena aproximación para clasificar los textos. En [28] se utiliza justamente un modelo basado en estilo para la detección de noticias falsas.

[2] `https://www.politifact.com/`
[3] `https://www.snopes.com/`

3.2.3. Métodos basados en aspectos sociales

Estos métodos se enfocan en las interacciones que tiene una pieza informativa en redes sociales, observando cómo es que reaccionan los usuarios a la información, o cómo se ha ido compartiendo esta a través de diferentes usuarios, se puede dar una buena idea si es que se está tratando con información falsa. Un primer enfoque considera un fenómeno que se estableció anteriormente, un usuario con una postura bien definida normalmente escribirá o compartirá publicaciones que se alineen a sus creencias y que también confirmen su forma de pensar. Este tipo de perfiles son muy susceptibles a compartir desinformación relacionada con sus creencias, esto es debido a que sus filtros informativos son mucho más bajos cuando se trata de información que le da la razón, pero en caso contrario, los usuarios tienden a ser más cuidadosos o incluso ni siquiera comparten información contraria a sus creencias, un fenómeno que es descrito en [29]. Como resultado, se puede considerar la postura de un usuario hacia un tema como un factor que determine su tendencia a compartir información falsa.

También se puede considerar la trayectoria que ha tenido la información en una red social, perfilando usuarios que comúnmente comparten información falsa. Si un artículo informativo fue compartido o tiene su origen en un perfil que ha distribuido en algún momento piezas de desinformación, es muy probable que pueda compartir más publicaciones de este tipo.

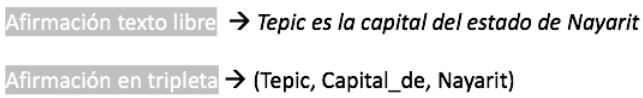
Adicionalmente, observar cuál es el comportamiento de los usuarios de la red social con respecto a una publicación también es de utilidad para clasificar dicha publicación. Si los usuarios reaccionan de forma negativa, o comentando la falta de validez de la información presentada, es muy probable que se esté tratando con información no real. Este enfoque es muy similar a lo que ocurre en plataformas de *crowd-sourcing*, en los cuales se utiliza las valoraciones colectivas para determinar la naturaleza de una publicación.

Trabajos tales como [18] y [21] consideran aspectos sociales para la detección de rumores en redes sociales.

3.3. Definición de verificación automática de hechos

Esta tarea la podemos entender como el proceso para determinar si una descripción de un hecho es verdadera o no, utilizando conocimiento previo para tomar esta decisión. Esta descripción del hecho se denominará afirmación en este trabajo. La afirmación puede tener varios formatos, por ejemplo, se pueden utilizar tripletas sujeto-predicado-objeto que es una representación muy usada en PLN. Pero utilizar estas tripletas requiere que estas deban ser extraídas previamente de las fuentes textuales que las establecen, lo cual es propiamente una tarea aparte por resolver. Otra representación de una afirmación es simplemente texto libre (Figura 3.1), el cual contiene palabras que establecen un sujeto, un objeto y la relación entre ellos en forma de predicado. Usualmente es texto concreto que es obtenido de textos más amplios que establecen la afirmación presentada. En la realidad pocas veces nos encontraremos directamente con afirmaciones susceptibles a ser verificadas, si no que será necesario obtener primero aquellas oraciones de un documento que valga la pena evaluar. Hay toda un área de investigación que se encarga de desarrollar métodos para extraer afirmaciones de discursos políticos, debates, noticias, etc. determinando su relevancia para ser evaluada, considerando si es de interés público, si no es demasiado absurda o si ya ha sido verificada anteriormente.

Figura 3.1: Representación de una afirmación. Esta puede ser presentada como una tripleta sujeto-predicado-objeto o como texto libre.



El segundo requerimiento para un modelo de verificación de hechos es el conocimiento que permitirá razonar si la afirmación es cierta. Este conocimiento de igual forma se puede representar de muchas maneras, por ejemplo, como un grafo de conocimiento, como es el caso de *DBpedia* [4] que es una base de conocimiento extraída de *Wikipedia* [5]. También es posible representar el conocimiento con texto libre, utilizando un conjunto de documentos que contendrán información que puede ser relevante para la verificación de una afirmación. Es importante, señalar que es necesario evaluar el nivel de confianza de la base de conocimiento que servirá como verdad fundamental para el modelo, si la información que se utiliza para verificar una afirmación no es confiable, entonces las salidas del modelo tampoco podrán ser confiables y por lo tanto la utilidad de dicho modelo será muy pobre. Un trabajo que propone un método para evaluar la confiabilidad de un hecho obtenido [9], considera la página donde está publicado y la forma cómo se hizo la extracción. Este conocimiento que se utilizará para el proceso de verificación lo denominaremos evidencia.

Entonces, un modelo de verificación automática de hechos tomará como entrada una afirmación C y la evidencia que será un conjunto de oraciones $E = \{e_1, e_2, \ldots, e_n\}$. Como salida se obtendrá una clasificación de la afirmación dada la evidencia (Figura 3.2). La forma más simple es realizar una clasificación binaria, verdadera o falsa, pero también es posible agregar una tercera clase que indicará si es que la evidencia presentada no cuenta con la información necesaria para llevar a cabo la verificación. Esta tercera clase es de utilidad en aplicaciones reales, porque normalmente hay una fase previa de extracción de evidencia, lo cual implica que pueda existir un error y que como consecuencia se presente durante la verificación de la afirmación una evidencia que no sea útil; es una manera de tratar con este posible error. Adicionalmente, también se pueden establecer varios niveles de veracidad o falsedad de la afirmación. Esto corresponde a cómo es que muchos verificadores humanos llevan a cabo esta tarea: clasifican una afirmación como verdadera, parcialmente verdadera, parcialmente falsa, falsa.

3.4. Métodos para la verificación de hechos

Primeramente, es importante mencionar que existen métodos de verificación de hechos que no consideran evidencia alguna. Simplemente se fijan en características lingüísticas de la afirmación para determinar su naturaleza, [31] usa este método. Esta es una estrategia muy similar a la planteada por los métodos basados en el estilo, pero en este caso está limitado a una afirmación concreta, no a todo un documento. Para detectar estas características en el lenguaje, se utilizan métodos supervisados, en los cuales se utilizan *datasets* con afirmaciones

[4] `https://www.dbpedia.org/`
[5] `https://es.wikipedia.org/`

Figura 3.2: Vista general del proceso de verificación automática de hechos. Se proporciona una afirmación, se busca evidencia y se procede a clasificar la afirmación.

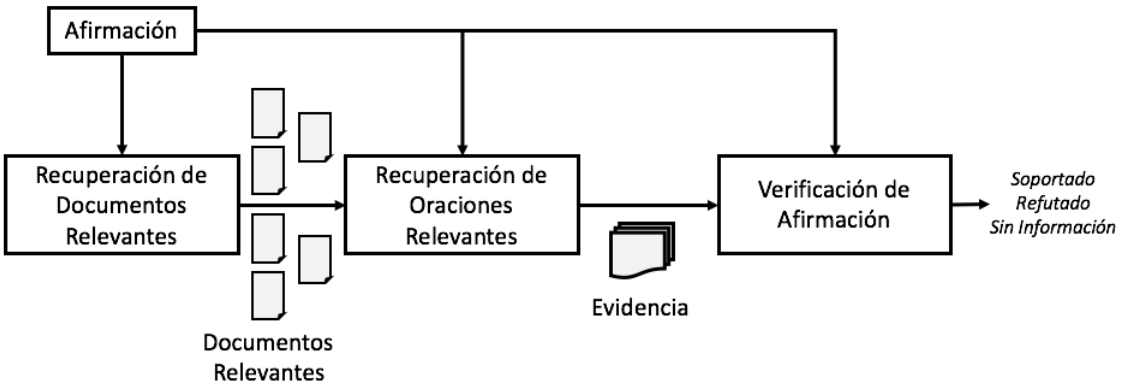
etiquetadas basado en verificaciones hechas por sitios especializados en esta labor. De esta forma se puede establecer si una afirmación es creíble u objetiva lo cual suele ser indicativo de veracidad, aunque no siempre ocurre de esta manera. Estos métodos suelen dar resultados aceptables, pero para poder realizar una verificación de hechos más certera, es necesario contar con conocimiento del mundo real.

Cuando se cuenta con suficiente evidencia, hay otros métodos que modelan este problema como reconocimiento de vinculación textual, RTE por sus siglas en inglés (Recognizing Textual Entailment) [7]. RTE es una tarea que se utiliza como una prueba de referencia para modelos de PLN, con la cual se puede evaluar qué tan bien un sistema es capaz de razonar dado un texto [26]. RTE consiste en una premisa y una hipótesis, y se debe determinar si la hipótesis soporta, refuta o es neutral con respecto a la premisa. Entonces adaptando esta noción los modelos basados en RTE deben determinar si la evidencia soporta o refuta una afirmación, considerando que la evidencia es una verdad fundamental, entonces se puede inferir que, si la evidencia soporta, entonces la afirmación es verdadera, en caso contrario es falsa.

3.4.1. Método de 3 pasos

Como se ha mencionado, para resolver este problema es necesario contar con suficiente evidencia para evaluar una afirmación. Muchos métodos de verificación automática de hechos segmentan esta tarea en 3 subtareas secuenciales: recuperación de documentos relevantes, recuperación de oraciones relevantes y verificación de afirmación (Figura 3.3).

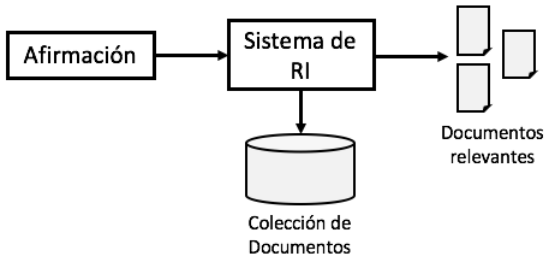
Figura 3.3: Verificación de hechos en 3 pasos.



3.4.1.1. Recuperación de documentos relevantes

Este es el primer paso, y consiste básicamente en un sistema de recuperación de información (RI). Este tipo de sistemas tienen el objetivo de encontrar documentos (capítulos, párrafos, oraciones) que sean relevantes dada una necesidad de información [22] (Figura 3.4). Esta tarea se debe de realizar de forma eficiente y en un tiempo razonable. Es necesario contar con una colección de documentos sobre los cuales se realizará la búsqueda, esta colección puede ser tan grande como se necesite. Una técnica estándar para llevar a cabo estas búsquedas eficientemente es el uso de una estructura conocida como índice invertido. Este índice tiene el objetivo de mapear los términos existentes en la colección con los documentos en los cuales aparecen los términos. De esta forma, al introducir una consulta se puede determinar en qué documentos aparecen los términos de la consulta.

Figura 3.4: Recuperación de documentos relevantes. Un sistema de recuperación de información es capaz de extraer documentos relevantes de una colección dada una necesidad de información (la afirmación).



La importancia de este paso es que, para el caso de la verificación de hechos, se puede contar con una base de conocimiento muy grande, lo cual favorece que se puedan verificar gran cantidad de afirmaciones. Sin embargo, evaluar uno a uno los documentos para conocer su relevancia respecto a la afirmación, puede ser computacionalmente muy demandante. Por este motivo, la recuperación de documentos relevantes es una forma de realizar un filtrado previo que podrá no ser tan exacto semánticamente pero que se realiza de forma muy eficiente; es una buena aproximación para reducir significativamente el tamaño de la ventana de búsqueda.

A continuación, se listan algunas implementaciones para este paso en la tarea de recuperación de información.

Anserini [6] es una herramienta de recuperación de información que fue construida sobre la biblioteca *Apache Lucene* [7]. Esta herramienta cuenta con un *API* que permite realizar indexado de documentos de forma eficiente, de tal manera que es posible construir un motor de búsqueda escalable, tiene la capacidad de realizar búsquedas en colecciones de incluso millones de documentos en unos cuantos milisegundos.

Un método que fue presentado como *baseline* en el reto *FEVER* [37] es la construcción de un índice invertido sobre la colección de documentos de *Wikipedia*, y para los pesos de los

[6] `https://github.com/castorini/anserini`
[7] `https://lucene.apache.org`

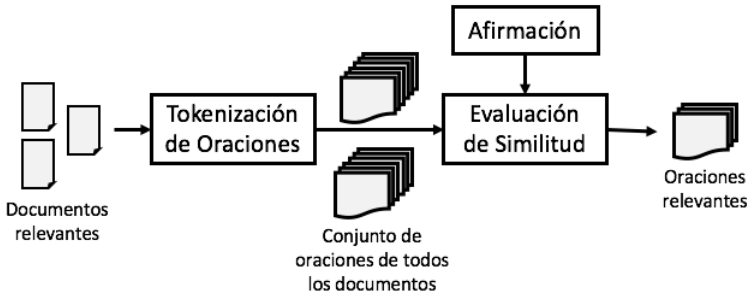
términos en los documentos se utiliza *TF-IDF*.

En [15] proponen un método para recuperar documentos de *Wikipedia*. Primero, realiza un preprocesamiento de la afirmación buscada, con ayuda de un parser se extraen las frases nominales ya que estas frases pueden contener conceptos que son relevantes para la búsqueda. Posteriormente utiliza un motor de búsqueda [8] para encontrar los documentos relevantes dadas las frases nominales extraídas. Varios trabajos [5] [35] utilizan esta misma metodología para extraer documentos relevantes.

3.4.1.2. Recuperación de oraciones relevantes

El paso anterior da como resultado un subconjunto de documentos que son relevantes para la afirmación presentada, esto quiere decir que parte de cada uno de ellos contiene texto que de alguna forma está relacionado con la afirmación. En el segundo paso, que consiste en la recuperación de oraciones relevantes, se deben de encontrar las porciones de texto importantes para realizar la verificación. Una buena aproximación para encontrar estos segmentos de texto es evaluar una a una las oraciones de cada documento, esto se puede visualizar en la Figura 3.5. De esta manera se tiene una granularidad adecuada para realizar la búsqueda ya que las oraciones normalmente tienen información concreta acerca de un hecho. Como resultado de esta etapa, se tendrá un conjunto E de n oraciones $\{e_1, e_2, \ldots, e_n\}$ que servirá como evidencia en el proceso de verificación.

Figura 3.5: Recuperación de oraciones relevantes (evidencia). Se extraen todas las oraciones de los documentos relevantes. Luego se evalúa la relevancia de cada una de ellas. Evaluar la similitud semántica es una forma de hacerlo.



Una estrategia para encontrar la evidencia relacionada con una afirmación es encontrar las oraciones que sean semánticamente más cercanas. Utilizar una representación vectorial de documentos como la propuesta en [32] suele dar buenos resultados calculando la similitud coseno de los vectores que representan las oraciones. También se ha propuesto utilizar *TF-IDF* como en el caso del modelo *baseline* de [37].

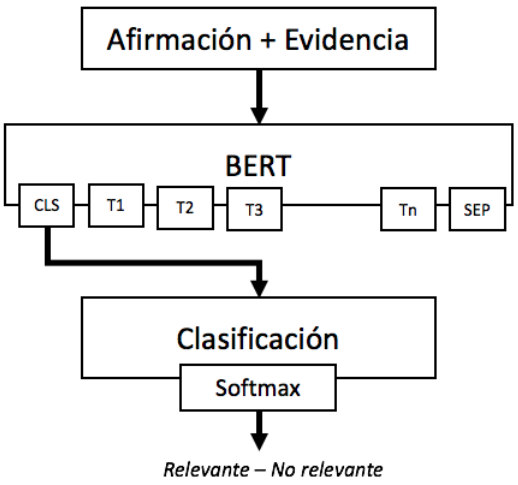
Otro grupo de propuestas utilizan redes neuronales para clasificar la relevancia de las oraciones. El modelo propuesto por [4] llamado *ESIM* utiliza celdas recurrentes *LSTM* para tratar la tarea de *RTE*, y considerando que es posible estimar la relevancia de una oración utilizando modelos pensados para *RTE*, existen modelos como [15] que son variantes

[8] `https://www.mediawiki.org/wiki/API`

precisamente de *ESIM*.

No obstante, los buenos resultados de estos modelos, actualmente el estado del arte está en utilizar modelos preentrenados del lenguaje como *BERT* [8]. Estos modelos también aproximan esta tarea con *RTE*, creando un clasificador que indica si una oración es relevante para una afirmación presentada. En [35] se presenta un modelo que utiliza *BERT* para extraer evidencia, y es a la fecha el estado del arte utilizando redes neuronales. En la Figura 3.6 se muestra la arquitectura de este modelo.

Figura 3.6: Propuesta de [35] para la clasificación de oraciones relevantes. Se utiliza *BERT* [8] para codificar la pareja de oraciones, se agrega una capa extra de clasificación.



3.4.1.3. Verificación de afirmación

En el tercer y último paso para la verificación de hechos, ahora se debe tomar la evidencia resultante del paso anterior. Evaluando en conjunto la evidencia y la afirmación, el modelo debe decidir si la afirmación es apoyada, refutada o no tiene información suficiente (Figura 3.7). Al igual que en la recuperación de oraciones relevantes, la verificación se aproxima utilizando *RTE*. Por esta similitud, los métodos presentados para el paso previo son aplicables también para verificar la afirmación. Modelos basados en *ESIM* como [15] han dado resultados buenos en esta tarea, pero también el uso de modelos de lenguaje ha alcanzado el estado del arte. El modelo propuesto por [35] se presenta como el estado del arte, utilizando *BERT* y agregándole una capa de clasificación de 3 clases (Figura 3.8).

La salida de este paso es la etiqueta definitiva que tendrá la afirmación. Cada etapa de este método de 3 pasos es susceptible de generar un error, no todos los documentos presentados como relevantes realmente lo serán, no todas las evidencias obtenidas serán verdaderamente relevantes. Como resultado el error se va arrastrando desde etapas tempranas del proceso, es importante evaluar este método en su totalidad utilizando las salidas de las etapas iniciales tanto para la evaluación como para el entrenamiento.

Figura 3.7: Verificación de afirmación. Toma como entrada la evidencia encontrada y la afirmación.

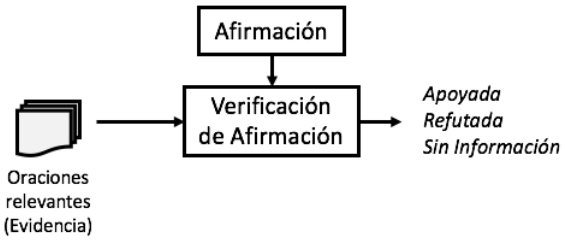
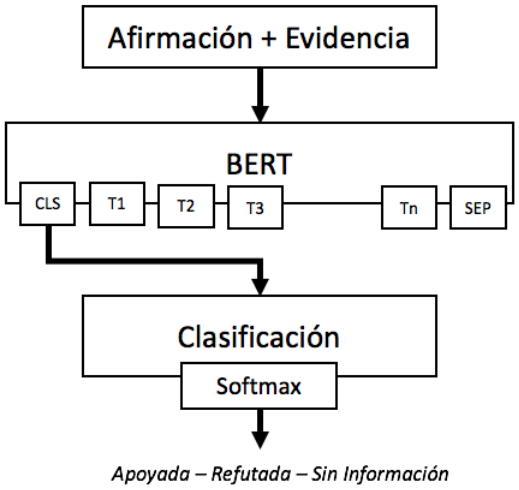


Figura 3.8: Propuesta de [35] para la verificación de afirmaciones. Se utiliza BERT [8] para codificar la pareja de afirmación y evidencia, se agrega una capa extra de clasificación de 3 clases.



3.5. Evaluación de la verificación automática de hechos

Cada una de las etapas de la verificación automática de hechos debe ser evaluada para tener una idea de qué tan bien se está desempeñando y para estar en posibilidad de comparar la estrategia actual con otros métodos que se han propuesto.

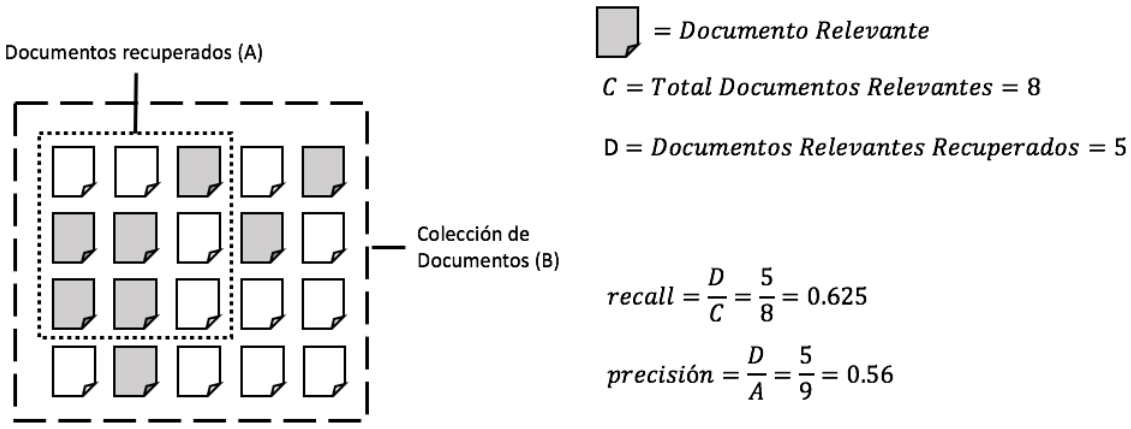
3.5.1. Evaluación de la recuperación de documentos relevantes

Esta tarea corresponde a un sistema de recuperación de información. Para evaluar este tipo de sistemas se requiere de una colección de prueba, que es una colección que tiene un conjunto de necesidades de información y para cada una de ellas tiene anotado los documentos que cumplen con dicha necesidad de información.

Las métricas más utilizadas para evaluar RI son la precisión y el *recall*. El *recall* es la proporción de los documentos relevantes que se pudieron recuperar del total de documentos relevantes. Por su parte la precisión es la proporción de documentos relevantes recuperados entre el total de documentos recuperados, es decir, esta métrica indica cuántos de los

documentos recuperados son realmente relevantes (Figura 3.9).

Figura 3.9: Precisión y *recall*. Se muestra una colección de documentos (B) de los cuales se recuperó una ventana de 9 documentos (A).



La tarea de RI puede ser vista como una clasificación binaria, un documento es relevante o no lo es. En tareas de clasificación es común evaluar utilizando la exactitud. Sin embargo, en RI la clase no relevante es mucho más grande que la relevante, un sistema que clasifique todos los documentos como no relevantes obtendría muy buenos resultados. Además, con la exactitud es complicado ver una diferencia real entre sistemas. Por esta razón la exactitud no se utiliza para evaluar RI.

Entonces, un sistema con alta precisión es más cuidadoso de los resultados que arroja. Un sistema con alto *recall* se preocupa por devolver todos o casi todos los documentos relevantes. En general, el *recall* y la precisión tienen una relación inversa, entre más documentos devuelve un sistema, tiene más probabilidades de encontrar todos los documentos relevantes, como consecuencia, aumenta el *recall*. Clasificar todos los documentos de la colección como relevantes resulta en un *recall* igual a 1. Pero esto provoca que muchos documentos no relevantes sean clasificados como relevantes, esta situación reduce de forma sustancial la precisión. Este tipo de relación provoca que los sistemas deban buscar un compromiso, un punto medio que se adecúe a las necesidades. Hay sistemas que pueden necesitar una alta precisión, como las búsquedas web, en las cuales hay mucha redundancia de información y no es necesario encontrar todos los documentos relevantes, si no que dentro de los documentos clasificados como relevantes esté la información requerida. Otras aplicaciones necesitan un mayor *recall*, como en la búsqueda de patentes, en este dominio todos los documentos relevantes deben ser encontrados [22].

Estimar el rendimiento del sistema requiere tomar en cuenta el *recall* y la precisión en conjunto. Si se grafica *recall* contra precisión se obtiene una curva que idealmente se debe aproximar a la esquina superior derecha. Por este motivo, entre más grande sea el área bajo la curva, mejor rendimiento se puede estimar del sistema.

3.5.1.1. *F-score*

Esta es una medida que permite obtener la relación entre la precisión y el *recall*. Es la media armónica pesada entre estas medidas.

$$F_{\beta^2} = \frac{PR}{R\alpha + P(1-\alpha)} = \frac{(\beta^2+1)PR}{\beta^2 P + R}$$

$$\beta^2 = \frac{1-\alpha}{\alpha}$$

El parámetro α define qué tanto peso tiene el *recall* y la precisión. Cuando ambas tienen el mismo peso $\alpha = 0.5$ y $\beta^2 = 1$ a este valor de F se le conoce como F1. El valor máximo de F1 es un buen indicador del mejor compromiso entre *recall* y precisión. Se utiliza una media armónica para evitar que una de las métricas haga que el *score* final no sea tan bajo aún cuando alguna de las métricas tenga valores cercanos a cero. Esta medida se acerca más al valor mínimo cuando los valores son muy diferentes.

$$F1 = \frac{2PR}{P+R}$$

3.5.2. Evaluación de la recuperación de oraciones relevantes

Esta evaluación es bastante similar a la planteada para la tarea de recuperación de documentos relevantes. Es similar en el sentido de que se desean clasificar de forma binaria el conjunto de oraciones existente en el conjunto de documentos clasificados como relevantes. En este caso, también la mayoría de las oraciones corresponderán a la clase negativa, mientras que sólo unas cuantas de ellas pertenecerán a la clase relevante. Por estos motivos, esta etapa se evalúa también utilizando *recall*, precisión y F1.

3.5.3. Evaluación de la verificación de afirmación

La verificación de afirmación es una tarea de clasificación que en general se trata con tres clases, soportada, refutada y sin información. Para evaluar esta tarea, se utiliza la exactitud. Algunos retos como el propuesto por *FEVER* [37] utilizan una métrica que considera también los pasos anteriores, se toma la exactitud de la clasificación de las etiquetas, siempre y cuando se haya obtenido por lo menos una evidencia correcta. Para este caso la exactitud simple es el límite superior de los valores que puede tomar la exactitud considerando la evidencia. En algunos casos esta evaluación será la más conveniente ya que el clasificador podría estar obteniendo la etiqueta correcta a pesar de que se le presente evidencia que no tiene que ver con la afirmación en cuestión.

Para la evaluación de todas las etapas (recuperación de documentos relevantes, recuperación de oraciones relevantes y verificación de afirmación) se pueden utilizar las entradas ideales o usando como entrada las salidas del paso anterior. Como se mencionó, cada paso trae consigo un error en su tarea individual, por lo que ese error se propaga a etapas posteriores en el método de verificación de hechos. Esta forma de medir da una idea más realista de qué tan bien trabaja todo el proceso en conjunto. No obstante, durante el proceso de experimentación también es útil saber qué tan bien se comporta cada uno de los componentes de forma individual, utilizando para esto entradas ideales.

3.6. Resumen

En este capítulo se presenta la tarea de verificación automática de hechos. El propósito de esta tarea es apoyar en las labores de verificación manual que están enfocadas en identificar piezas de información presuntamente falsas. El problema de detección de información falsa es una actividad de mucha dificultad ya que por un lado este tipo de información se presenta de muchas maneras: imágenes, texto, video o audio. En este trabajo nos centramos en detección de información falsa en fuentes textuales. Aún limitando este problema al texto, sigue siendo de gran dificultad porque hay muchos aspectos que se pueden considerar para la detección. Se pueden tomar características del texto como el estilo, vocabulario o sentimiento, o también se pueden considerar aspectos externos como las interacciones sociales que este texto ha tenido, o también se puede utilizar conocimiento externo para evaluar el texto sospechoso de ser falso.

El foco de este capítulo es precisamente los métodos de detección de información falsa textual que utilizan fuentes de conocimiento externas para cumplir con la tarea. Si bien esta verificación de información en la práctica se realiza de forma manual, existen esfuerzos para proponer e implementar modelos que sean capaces de realizar la verificación de hechos de forma automática.

Se presentaron algunos modelos que han sido creados para verificación de hechos (expresados como afirmaciones), la gran mayoría de ellos utiliza un proceso de 3 pasos para uno, extraer documentos que puedan ser útiles; dos, extraer de estos documentos oraciones que sean relevantes y tres, verificar la afirmación considerando la evidencia encontrada.

Por último, se presentan las métricas que serán útiles para evaluar los modelos construidos y compararlos con otras propuestas.

Capítulo 4

Metodología propuesta

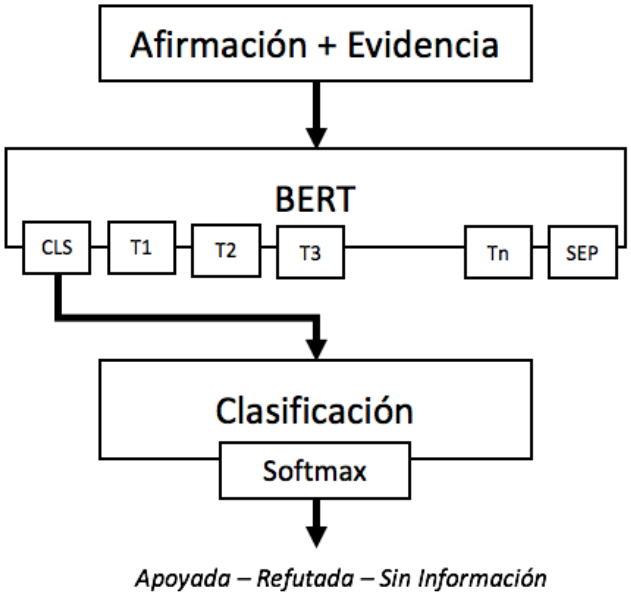
En el capítulo 3 se presentó el método de tres pasos [37] que ha sido ampliamente utilizado por diversos trabajos con la intención de llevar a cabo una verificación automática de hechos. Este método evalúa una afirmación y como resultado debe de clasificarla como apoyada, refutada o sin información, tomando en cuenta la evidencia existente. Este método se divide en tres etapas: recuperación de documentos relevantes, extracción de oraciones relevantes y verificación de la afirmación (Figura 3.3). En este capítulo se presenta nuestra arquitectura para realizar la extracción de oraciones relevantes que posteriormente servirán como evidencia, y también se propone aplicar esta misma arquitectura con pequeñas adaptaciones para verificar la afirmación utilizando la evidencia recolectada en la etapa previa. La intención de esta arquitectura es utilizar modelos preentrenados de lenguaje que han dado resultados muy positivos tanto en otras tareas de PLN, como en la propia verificación de hechos, aprovechando las virtudes de dichos modelos para generar una interpretación de la salida.

4.1. Arquitectura propuesta

Las arquitecturas basadas en modelos preentrenados de lenguaje han demostrado ser implementaciones bastante competitivas en la tarea de verificación de hechos, y es de hecho en este tipo de modelos en donde se encuentra el estado del arte. En este trabajo de tesis utilizamos como arquitectura de comparación el modelo propuesto por [35], que tiene resultados de estado del arte en verificación de hechos. Este modelo está basado en *BERT* [8], en la Figura 4.1 se muestra su arquitectura. Lo que propone este artículo es hacer un ajuste fino al modelo preentrenado, agregando una capa de clasificación al final. Esta capa de clasificación toma el vector de salida correspondiente al *token* de control *CLS*. Como se ha mencionado esta salida de *BERT* es utilizada por tareas específicas para obtener una representación vectorial de una secuencia de palabras, es decir, una oración. Las arquitecturas que proponen los autores para la tarea de relevancia y para la verificación son iguales, sólo varía la cantidad de clases de cada uno. Este modelo obtiene los mejores resultados para el *dataset* de *FEVER*. A pesar de los aspectos positivos, este modelo carece de una interpretación para las decisiones tomadas. Y esto es principalmente debido a que *BERT* cuenta con múltiples mecanismos de atención que funcionan en paralelo y a través de todas sus capas. Hay investigación [6] acerca de cómo interpretar lo que están aprendiendo las cabezas de atención, durante el preentrenamiento y el ajuste fino. En la Figura 4.2 se muestran, por ejemplo, los mapas de atención calculados para 2 secuencias de prueba, dichos mapas correspoden a las capas 7-12 de *BERT*. En estos mapas, en el eje X se encuentran los *tokens* de la primera secuencia, y en el eje Y los *tokens*

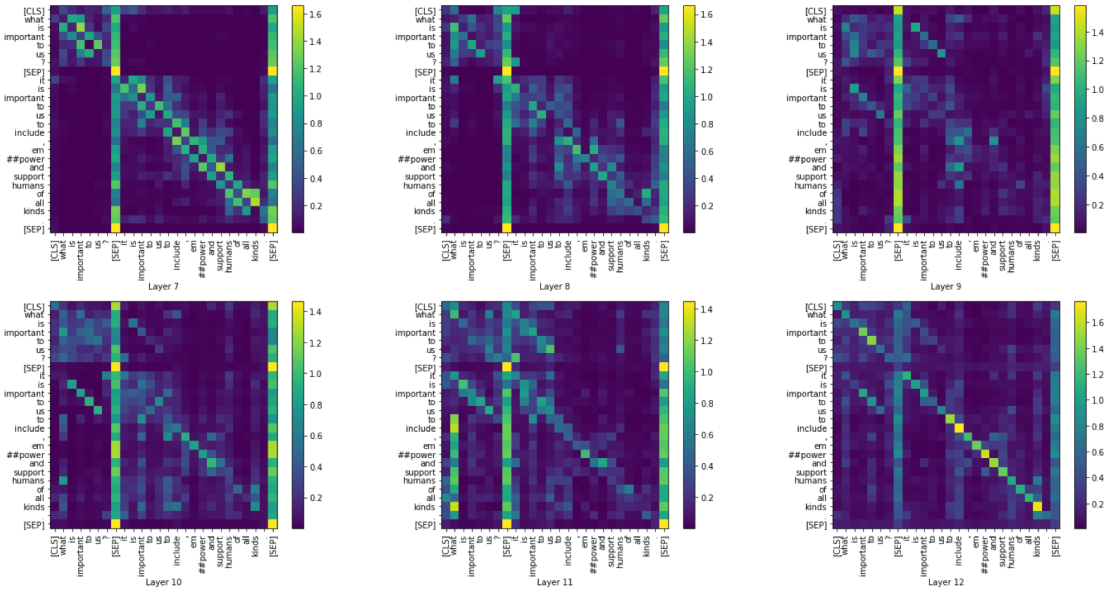
de la segunda. La intersección de cada elemento representa el peso que se calculó para cada *token*. En este ejemplo, los pesos se cargan mucho a *tokens* de control, que muchas veces se entienden como atenciones nulas (*no operation*). La hipótesis de estas investigaciones es que cada capa se centra en un aspecto diferente del lenguaje, algunas se enfocan en la sintaxis, otras en las relaciones semánticas, y en cada una de las cabezas de cada capa se enfoca en un aspecto particular de estos niveles del lenguaje. En la Figura es evidente que no hay una clara interpretación para los pesos. El problema se acrecienta con modelos más grandes como *GPT-3*, que tiene 96 capas y 96 cabezas, lo que resulta en un total de 9216 mecanismos independientes de atención. Cada uno de estos mecanismos se enfoca en particularidades del lenguaje, pero es tal su granularidad que se pueden tener n cabezas enfocándose en aspectos relacionados con la sintaxis. Obtener una interpretación de semejante cantidad de mapas de atención no es trivial y es algo que se sigue investigando.

Figura 4.1: Arquitectura propuesta por [35] para ajustar *BERT* para la tarea de verificación de hechos.



La propuesta presentada en este trabajo de tesis pretende simplificar la interpretabilidad del sistema con un mecanismo de atención único que aprenda a calcular las relaciones que tienen cada una de las palabras. En esencia lo que se busca es codificar por separado la afirmación y la evidencia en cuestión. De esta codificación resultarán n vectores (que representan los *tokens* de la oración) de dimensión d para la afirmación, y otros n vectores de dimensión d para la evidencia. La dimensión d de los vectores está dada por el tamaño del estado oculto de *BERT*, 768 para el modelo base. Utilizando estos vectores se empleará un mecanismo de atención para calcular la importancia que tiene cada uno de los elementos de una secuencia (afirmación) con respecto a todos los elementos de la otra secuencia (evidencia). En la Figura 4.3 se muestra la arquitectura propuesta.

Figura 4.2: Mapas de atención obtenidos de distintas capas de *BERT*. Estos corresponden al promedio de todas las cabezas de atención de capa.

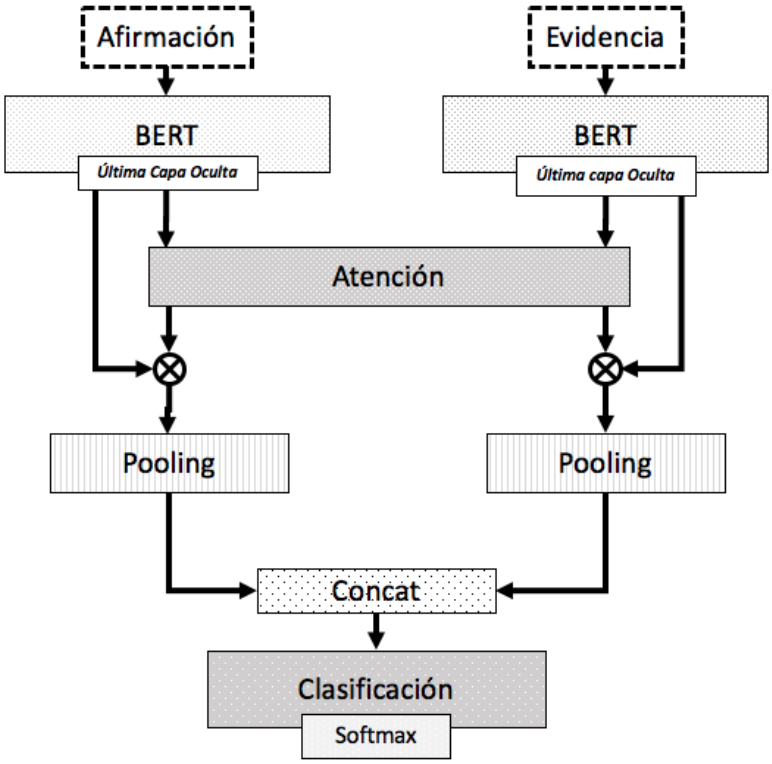


4.1.1. Codificación de las oraciones

Como entrada al modelo se tienen un par de oraciones, la afirmación y la evidencia (esto aplica tanto a la evaluación de la relevancia como a la verificación). Como se observa en la Figura 4.3 se tiene un par de bloques *BERT* que toman las oraciones tokenizadas y en su último estado oculto generan los n vectores para cada oración. Como se ha presentado anteriormente, estos vectores son representaciones semánticas de cada uno de los *tokens* de las oraciones, varían según su significado y también dependiendo de su rol dentro de la oración. Entonces estas representaciones son muy útiles para evaluar uno contra uno los vectores de las dos secuencias para conocer qué tan relevante son los demás vectores para el vector actual. Las secuencias de las oraciones se limitaron a un máximo de 128 *tokens*, en caso de ser de longitud mayor el tokenizador trunca la secuencia. Para los casos en los cuales la oración tiene menos *tokens* se agregan *tokens* de *padding* al final. El bloque codificador entrega secuencias de 128 vectores. Este tamaño máximo de secuencia corresponde con el valor utilizado en [35]. Verificando los tamaños que se tienen en el conjunto de datos se encontró que para el caso de las afirmaciones, la longitud máxima es de 65 *tokens*, el promedio es 8.11 y el 95 % de las afirmaciones está por debajo de 14 *tokens*. En la Figura 4.4 se muestra la distribución de la longitud de *tokens* para las afirmaciones. Para la evidencia, la longitud máxima es de 400 *tokens*, con un promedio de 37.12 y el 95 % de la evidencia está por debajo de 83 *tokens*. En la Figura 4.5 se muestra la distribución de la longitud de *tokens* para la evidencia.

En este punto es importante mencionar que se experimentó con dos configuraciones para estos bloques codificadores (Figura 4.6). Primero se evalúo tener un par de bloques con pesos independientes. Esta configuración provoca que el número de parámetros del modelo crezca mucho. Vale la pena recordar que el modelo base de *BERT* cuenta con 110 millones de parámetros. Esto quiere decir que, al utilizar este esquema con pesos separados, el modelo final aumenta en este número la cantidad de parámetros, resultando en un uso de memoria

Figura 4.3: Arquitectura propuesta tanto para la evaluación de la relevancia como para la verificación.



considerablemente mayor y por supuesto mayor tiempo de entrenamiento e inferencia. Por esta razón también se experimentó con pesos compartidos. Los pesos compartidos dieron mejores resultados en todos los casos, además de ser más eficiente en el uso de memoria. Como consecuencia, se decidió usar una configuración de pesos compartidos en todos los experimentos. Estos resultados son consistentes con los presentados en [32], ellos proponen una red siamesa para entrenar un modelo que calcula representaciones semánticas de oraciones, y obtuvieron mejores resultados usando pesos compartidos.

4.1.2. Mecanismo de atención

Se utilizó un mecanismo de atención con el objetivo de ponderar los elementos de las secuencias correspondientes a la afirmación y a la evidencia. Se probaron diferentes configuraciones de este mecanismo. El mecanismo toma como entrada las 2 secuencias s_1 y s_2 que son el resultado de la codificación del bloque *BERT*. Como resultado se obtendrá una matriz A de $n \times n$ donde n es la longitud de la secuencia de entrada, en este caso $n = 128$. A continuación, se describen los mecanismos de atención utilizados en los experimentos.

4.1.2.1. Atención multiplicativa

Este mecanismo está basado en la propuesta de [20]. Cada elemento de la matriz A estará dado por:

Figura 4.4: Distribución de cantidad de *tokens* en la lista de afirmaciones.

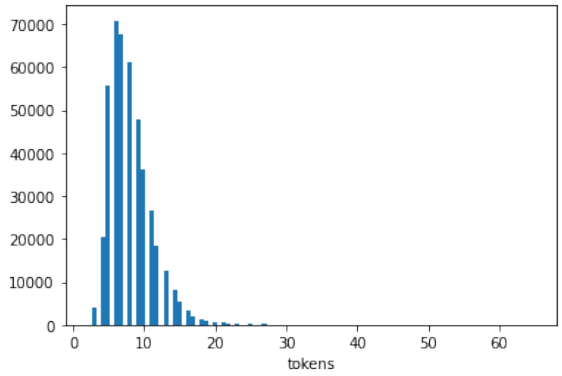
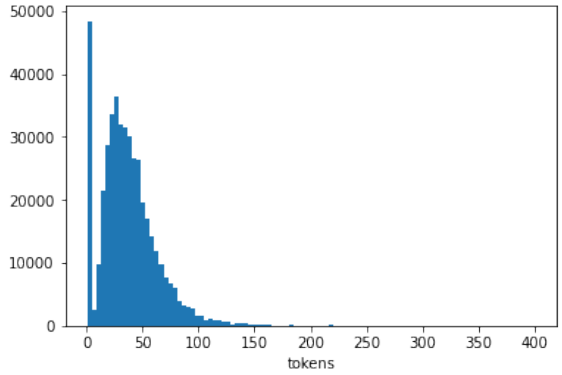


Figura 4.5: Distribución de cantidad de *tokens* en la evidencia.



$$A_{ij} = s_{1,i} \cdot s_{2,j}$$

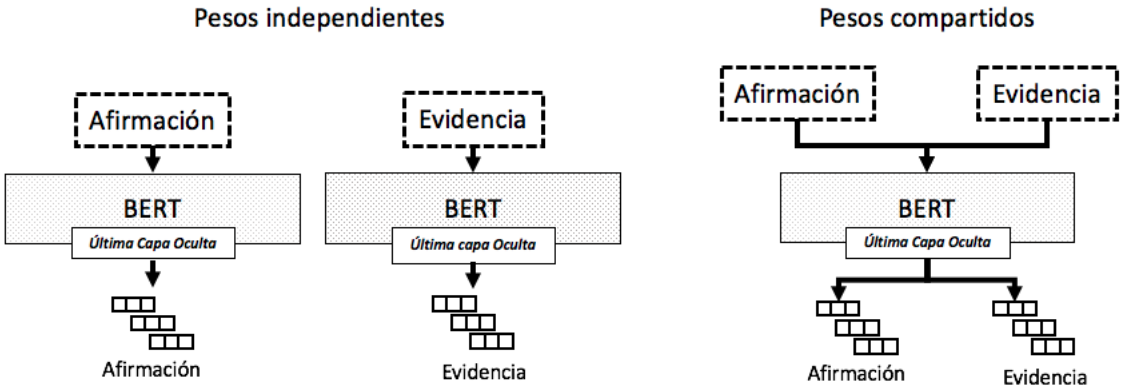
Esta función para calcular la medida de similitud entre un par de vectores funciona ya que el producto punto tiene la capacidad de evaluar qué tan parecidos son dos vectores. Es justamente esta similitud la que se busca plasmar en la matriz A. En [10] se propone agregar una función *softmax*, calculada sobre una dimensión de la matriz A. Nosotros experimentamos con la función *softmax* y con la sigmoide. La función sigmoide se aplica a cada elemento de la matriz para establecer un peso que se encontrará entre 0 y 1. Mientras que la *softmax* se aplica a todo el renglón de la matriz A, de tal forma que cada renglón representa una distribución de probabilidad.

4.1.2.2. Atención coseno

En este tipo de atención se utiliza la función coseno para obtener la similitud entre los vectores de las secuencias, se basa en el mecanismo descrito en [14]. La matriz A se calcula como:

$$A_{ij} = \frac{s_{1,i} \cdot s_{2,j}}{\|s_{1,i}\| \|s_{2,j}\|}$$

Figura 4.6: Configuraciones de la codificación de las 2 oraciones. Al final se obtienen representaciones vectoriales independientes tanto de la afirmación como de la evidencia.



4.1.3. *Pooling*

Después de calcular la matriz de atención A esta se multiplica por cada una de las secuencias de vectores s_1 y s_2. De esta forma, se ponderan los vectores de las secuencias, dando más peso a aquellos que son más cercanos a sus contrapartes en la otra secuencia. Como resultado se obtienen un par de secuencias de vectores u_1 y u_2, ambas secuencias con la configuración propuesta estarán compuestas por 128 vectores.

$$u_1 = As_1$$
$$u_2 = A^\intercal s_2$$

Ahora es necesario obtener un único vector que representará tanto a la afirmación como a la evidencia. Para este propósito, se realiza un *pooling* sobre las secuencias de vectores u_1 y u_2. Como resultado de este proceso, se obtendrá un vector por cada una de las oraciones de entrada, la dimensión de este par de vectores corresponderá al tamaño de la representación del estado oculto de *BERT*, para el caso de la versión base es de 768 dimensiones y para la versión grande de 1024. A continuación, se describen las estrategias de *pooling* probadas durante los experimentos.

4.1.3.1. Promedio

Utilizando los 128 vectores de la secuencia, se obtiene el centroide de este conjunto. Este centroide es el promedio de cada una de las dimensiones de todos los vectores. El vector resultante estará dado por la siguiente expresión, donde n es el tamaño de la secuencia:

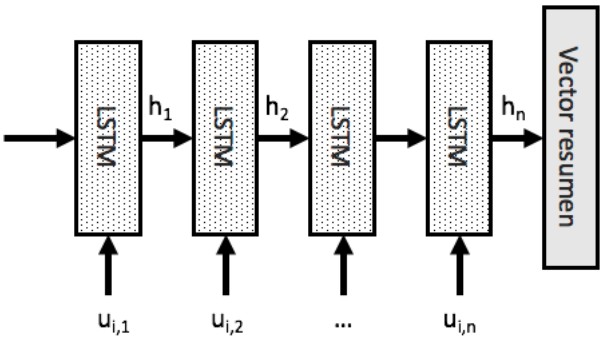
$$p_{i,j} = \frac{1}{n}\sum_{k=1}^{n} u_{i,j,k}$$
$$i = [1, 2]$$

Esta es una estrategia que suele dar buenos resultados, en [32] utilizan este tipo de *pooling* para resumir toda la secuencia de salida del último estado oculto de *BERT*.

4.1.3.2. Celda *LSTM*

Las celdas *LSTM* [17] son un tipo de red neuronal recurrente, que permite procesar secuencias, pero de forma más inteligente que una celda recurrente simple. Es capaz de "recordar" u "olvidar" detalles de la secuencia según su nivel de importancia. Al final de la secuencia se obtiene un estado oculto que refleja el contenido de dicha secuencia. Aprovechando estas características, se propuso utilizar una celda *LSTM* para resumir las secuencias u_1 y u_2. Ambas secuencias se hacen pasar por la celda recurrente y de cada una se toma el último estado oculto, este será el vector único que representará la secuencia completa (Figura 4.7).

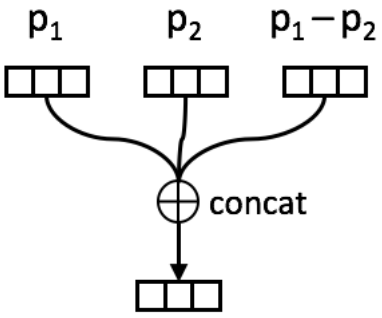
Figura 4.7: Red recurrente *LSTM* para obtener una representación vectorial de las secuencias u_1 y u_2.



4.1.4. Unión de las secuencias

Como resultado de la capa de *pooling*, se obtiene el par de vectores p_1 y p_2. Dichos vectores fueron producidos por cada una de las ramas de la red siamesa. Ahora es necesario mezclar estos vectores con la intención de hacer pasar el vector resultante a través de una capa de clasificación. Para llevar a cabo esta mezcla, se utilizó el método propuesto por [32]. Lo que se propone en este trabajo es concatenar el p_1 y p_2, y adicionalmente también se concatena el vector diferencia entre p_1 y p_2 (Figura 4.8). Se utilizó este método debido a que fue la estrategia que mejores resultados produjo en el artículo citado.

Figura 4.8: Unión de los vectores que representan las 2 secuencias de entrada.



4.1.5. Clasificación

La última etapa de la arquitectura consiste en un bloque de clasificación. La estructura de este bloque fue determinada de forma empírica, probando diversas configuraciones. En la Figura 4.9 se puede apreciar la estructura de este elemento de la arquitectura. Está formada por 3 capas completamente conectadas apiladas. La primera tiene una función de activación sigmoide, la segunda una tangente hiperbólica. La última capa utiliza la función *softmax* para calcular la clase más probable dada las entradas. En esta capa es donde se diferencia el modelo encargado de evaluar la relevancia de una oración y el modelo que verifica una afirmación. En el caso del modelo de la relevancia la última capa completamente conectada tiene una salida de 2 neuronas, sobre las cuales se calcula la función *softmax*. Esto corresponde con las clases que se están buscando: relevante y no relevante. En el segundo modelo, evaluación de afirmación, las salidas de la última capa son 3 neuronas correspondientes a las 3 clases, apoyada, refutada y sin información. En esencia estos modelos son iguales, sólo se diferencian en la cantidad de neuronas de salida en la última capa (Figura 4.10).

Figura 4.9: Estructura del bloque de clasificación.

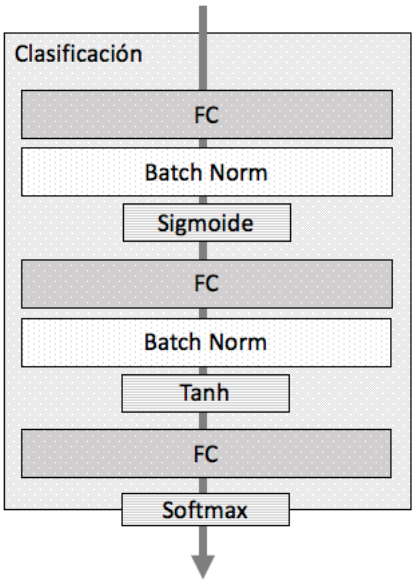
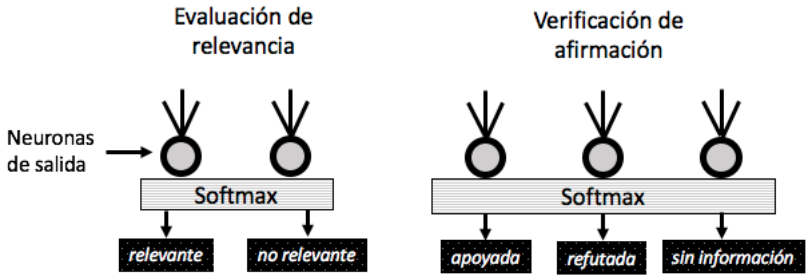
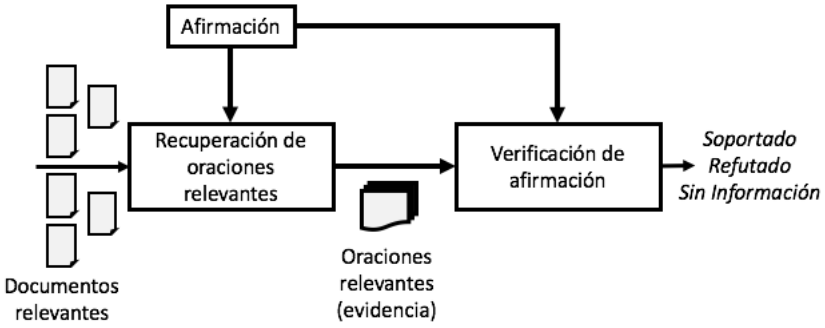


Figura 4.10: Clasificación para cada una de las tareas que se está tratando.



Con la arquitectura descrita se crearán dos modelos, el primero toma como entrada un conjunto de documentos relevantes para una afirmación, como salida produce el subconjunto de oraciones que son relevantes para la afirmación. El segundo modelo toma como entrada las oraciones recuperadas por el primero, con estas oraciones como evidencia el modelo clasifica la afirmación como apoyada, refutada o sin información. El detalle de esta arquitectura general se puede ver en la Figura 4.11,

Figura 4.11: Arquitectura general para extracción de evidencia y verificación de afirmación.



4.2. Resumen

En este capítulo se describió la metodología propuesta para realizar las tareas 2 y 3 del método para la verificación automática de hechos. Se describieron los modelos que permitirán primero, tomando un conjunto de documentos relevantes, extraer de ellos el subconjunto de oraciones que son consideradas relevantes dada una afirmación; segundo, verificar si la afirmación es apoyada, refutada o no tiene información, usando la evidencia obtenida. La arquitectura básica para estos modelos consiste en una red neuronal siamesa, que codifica de forma independiente la afirmación y la evidencia utilizando *BERT*. Con los vectores obtenidos de *BERT* es posible calcular la importancia de cada *token* en una secuencia, con respecto a cada uno de los *tokens* de la otra secuencia. El objetivo de esta configuración es crear una capa de atención que permita ponderar las entradas. La ponderación permitirá al modelo concentrarse en los aspectos, tanto de la afirmación como de la evidencia, que son más importantes para la clasificación. Las ponderaciones de este modelo son mucho más sencillas que las implementadas a través de las múltiples cabezas de atención de *BERT* y otras arquitecturas basadas en *transformer*, y serán de utilidad para obtener una interpretación de la decisión que tomó el modelo.

Capítulo 5

Configuración experimental

En este capítulo se describen las etapas que fueron necesarias para realizar la experimentación que permitiera seleccionar el modelo óptimo para atender las tareas de interés que se han descrito anteriormente. Para los experimentos se utilizó *FEVER* [37], un *dataset* enfocado en la evaluación de métodos de extracción de evidencia y verificación de hechos. En la primera parte del capítulo se describe el procesamiento previo que requirió este *dataset*. El conjunto de datos original ofrece un conjunto de afirmaciones y los identificadores de documentos (artículos de *Wikipedia*) en los cuales hay evidencia para verificar la afirmación. Para facilitar el entrenamiento de la red neuronal fue necesario crear un archivo con parejas afirmación-evidencia.

En la segunda parte del capítulo se describe el proceso de entrenamiento de la arquitectura propuesta. Debido a que se está ajustando un modelo preentrenado de lenguaje, existen varias opciones para realizar dicho ajuste. Estas opciones se pusieron a prueba buscando generar los mejores modelos de clasificación. Adicionalmente, como se mencionó en el capítulo anterior, hay varias configuraciones de la arquitectura que vale la pena explorar, buscando cumplir con las tareas presentadas y también con el objetivo de contar con una interpretación sencilla.

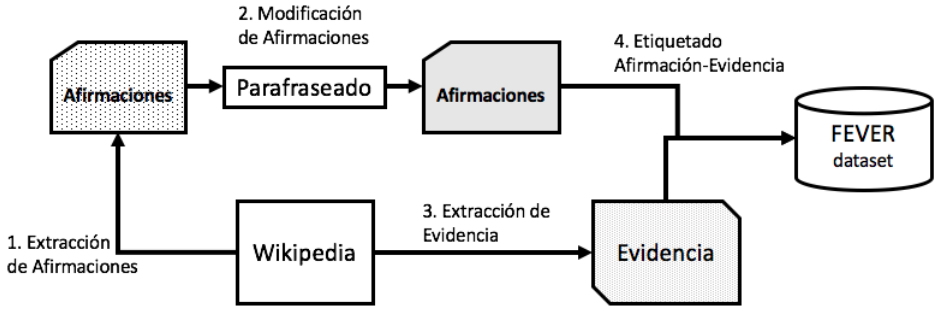
5.1. Conjunto de datos

En tareas de aprendizaje supervisado es necesario contar con una cantidad suficiente de datos de entrenamiento. Con el reciente interés en la investigación en verificación automática de hechos, han surgido competencias que buscan que los participantes propongan modelos sobre nuevos conjuntos de datos. Sin embargo, estos conjuntos de datos normalmente constan de apenas unos cientos de afirmaciones y un par de miles de oraciones que sirven como evidencia. Con conjuntos de estas dimensiones es complicado tener una prueba de referencia que sirva para comparar métodos de verificación de hechos, además de que los modelos basados en redes neuronales necesitan idealmente de una gran cantidad de datos para entrenar. Motivados por esta situación, en [37] proponen un *dataset* solamente en inglés de grandes dimensiones llamado *FEVER*. Está pensado en las tres etapas principales de la tarea de verificación automática de hechos. Para evaluar el modelo que se está presentando en este trabajo se utilizará precisamente *FEVER*. El *dataset* es parte de un reto que invita a crear modelos de verificación de hechos.

Los autores construyeron el *dataset* a través de un conjunto de anotadores que obtuvieron de forma manual oraciones sencillas que describen un hecho, es decir, afirmaciones. Como base

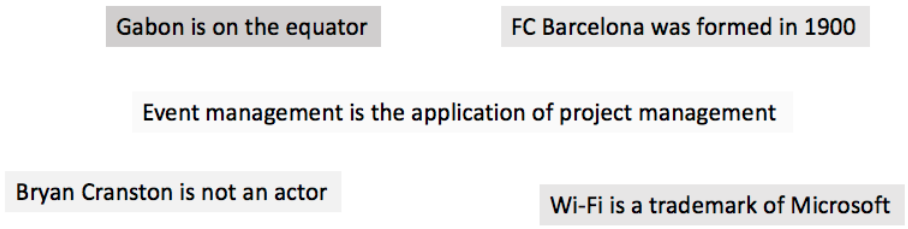
para extraer las afirmaciones se tomaron artículos de *Wikipedia* en inglés. Posteriormente, cada una de estas afirmaciones fue modificada con el objetivo de que no fueran exactamente igual al texto que aparece en el artículo, es decir, se parafrasearon. De otra forma se hubieran obtenido afirmaciones muy sencillas de verificar, simplemente buscando coincidencias exactas en todos los artículos de *Wikipedia*. Una vez que se tuvo todo el conjunto de afirmaciones, otro grupo de anotadores se encargó de buscar evidencia relacionada con cada una de las afirmaciones. Contando con esta evidencia ahora el anotador debe indicar si la afirmación es apoyada o refutada por el texto encontrado. En caso de que no se encuentre evidencia relevante para una afirmación, esta será etiquetada como sin suficiente información. Este proceso se puede visualizar en la Figura 5.1.

Figura 5.1: Proceso de creación del *dataset FEVER*.



Como resultado de esta recolección de afirmaciones y sus respectivas evidencias los autores lograron juntar 185,445 afirmaciones que se encuentran distribuidas como se muestra en la Tabla 5.1. La Figura 5.2 presenta ejemplos de afirmaciones. Para la extracción de la evidencia se proporciona un *dump* de *Wikipedia* que contiene alrededor de 5 millones de artículos correspondientes al año 2017. Con estos elementos es posible crear un sistema de verificación de hechos.

Figura 5.2: Ejemplos de afirmaciones extraídas.



El método que se está proponiendo en este trabajo sigue la estrategia de 3 pasos para verificación de hechos. Entonces es necesario crear 1 componente que recupere los documentos relevantes, otro componente que extraiga las oraciones relevantes de estos documentos y por último un componente que clasifique las parejas de afirmación-evidencia. Para entrenar los modelos que corresponden a los elementos del método de verificación de hechos, es necesario crear un archivo con los ejemplos relevantes para cada caso.

Tabla 5.1: Distribución del número de clases en cada partición de *FEVER*.

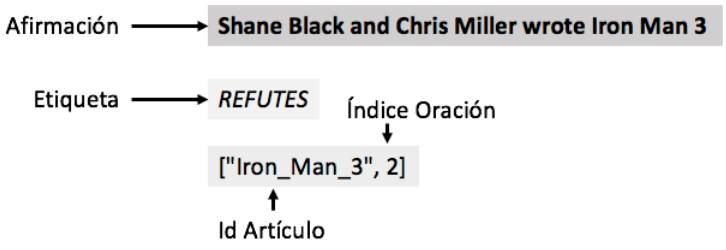
	Apoyada	Refutada	Sin información
Entrenamiento	80,035	29,775	35,639
Desarrollo	3,333	3,333	3,333
Pruebas	3,333	3,333	3,333
Reservado	6,666	6,666	6,666

Las arquitecturas que se están proponiendo se centran únicamente en la identificación de oraciones relevantes y en la verificación de la afirmación. Para obtener los documentos relevantes se utilizó el modelo de recuperación de información de [15]. Como se describió anteriormente, este modelo extrae frases nominales de las afirmaciones para crear búsquedas más concretas, después utiliza el *API* de *Wikipedia* para obtener los documentos más relevantes. Se eligió utilizar esta propuesta porque obtuvo un *recall* de 93 % en *FEVER*. Los archivos con los documentos recuperados por dicho modelo, para cada afirmación en cada una de las particiones de los datos, se pueden encontrar en el repositorio de *Github* de los autores[1].

5.2. Preprocesamiento

El conjunto de datos de *FEVER* estructura cada uno de sus registros de la forma que se muestra en la Figura 5.3. Se muestra la afirmación, y junto con ella aparece su etiqueta (soportada, refutada o sin información). También se acompaña con la evidencia que llevó a los anotadores a elegir la etiqueta mencionada. La evidencia se especifica con el identificador del artículo de *Wikipedia* y el número de oración dentro del artículo. Con este identificador y número de oración, se puede extraer el texto que corresponde a la evidencia, usando el *dump* de *Wikipedia* adjunto al *dataset*.

Figura 5.3: Ejemplo de registro original de *FEVER*.



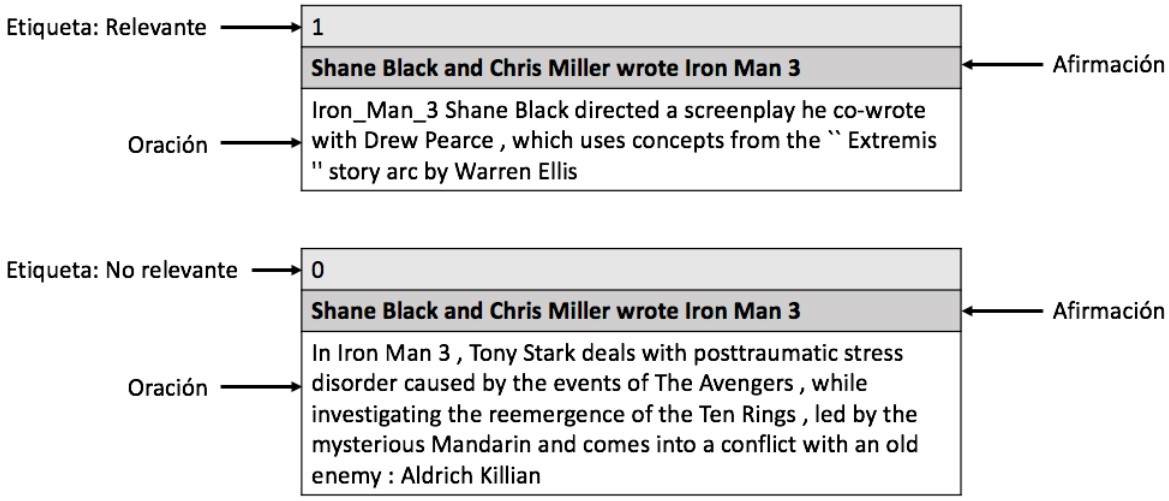
Para el caso del modelo que se está proponiendo, es conveniente llevar a cabo un preprocesamiento de estos datos con el propósito de facilitar el entrenamiento y evaluación. El objetivo es crear un archivo con los siguientes campos: etiqueta-afirmación-oración. El significado de cada uno de estos campos varía según la etapa del proceso con la que se está tratando.

[1] `https://github.com/UKPLab/fever-2018-team-athene`

5.2.1. Preprocesamiento para la relevancia de oraciones

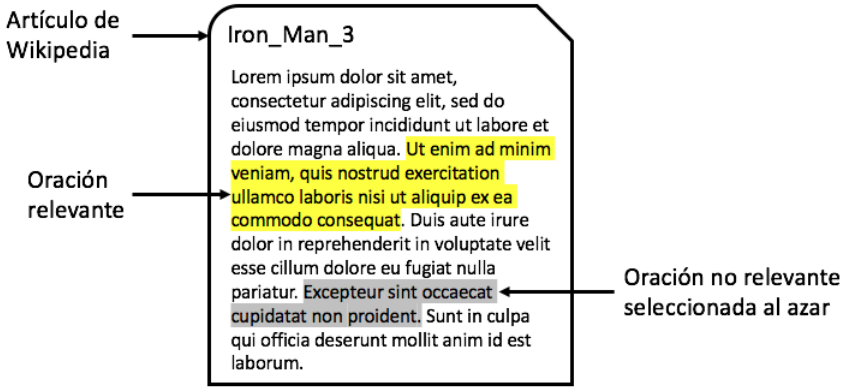
La intención de este preprocesamiento es generar los datos necesarios para la creación del modelo encargado de clasificar si una oración es relevante dada una afirmación, de tal forma que se simplifique la tarea de entrenamiento. Como resultado se obtuvo un archivo con el formato etiqueta-afirmación-oración. Para esta etapa la etiqueta se refiere a si la oración es relevante con respecto a la afirmación, le corresponde "1" si la oración es relevante o un "0" si no es relevante. En la Figura 5.4 se muestran ejemplos de registros generados.

Figura 5.4: Ejemplos de registros generados para la tarea de evaluación de relevancia de oraciones.



Para crear los ejemplos relevantes se toma del *dataset* las afirmaciones que están etiquetadas como refutada y apoyada, se extraen del *dump* de *Wikipedia* las oraciones que están marcadas como evidencia. Con esto se tienen el texto de la afirmación y la evidencia para los ejemplos relevantes. Para el caso de los ejemplos no relevantes, se utilizan las mismas afirmaciones y utilizando los artículos de *Wikipedia* que contienen evidencia, se selecciona al azar una oración del artículo, que tenga una longitud mayor a 10 caracteres y que no esté marcada como evidencia. De esta forma se obtienen ejemplos negativos que se encuentran en los artículos con evidencia, por lo cual el texto es de un tema similar y por lo tanto, para el modelo resultará más complicado saber si es relevante o no (Figura 5.5). Esto es importante ya que también se experimentó creando ejemplos negativos seleccionando al azar oraciones de cualquier artículo de *Wikipedia*. Pero durante los experimentos resultó claro que estos ejemplos resultan muy triviales para el modelo, simplemente fijándose en el léxico es evidente que el tema que se está tratando es totalmente diferente a lo que se establece en la afirmación. Muestrear los ejemplos negativos de artículos con evidencia, además de ofrecer ejemplos más difíciles, está más apegado a lo que ocurre en la práctica al momento de evaluar las oraciones de un documento para encontrar evidencia relevante.

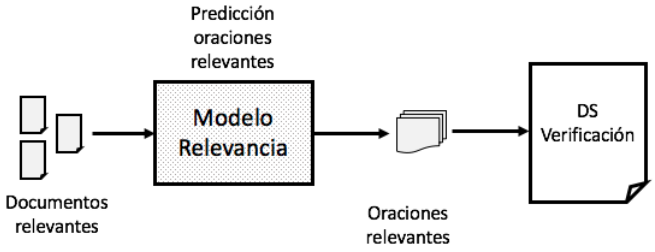
Figura 5.5: Extracción de ejemplos positivos y negativos. Los ejemplos negativos se extraen al azar de un artículo que contiene un ejemplo positivo.



5.2.2. Preprocesamiento para la verificación de afirmaciones

El preprocesamiento requerido para el paso de verificación de afirmaciones utiliza la salida del paso anterior. Esto quiere decir que se entrena el modelo encargado de clasificar la relevancia de las oraciones, y posteriormente con este modelo se predice la evidencia extraída de los documentos relevantes. Como resultado de esta clasificación se tendrá el conjunto de afirmaciones y para cada una de ellas, un conjunto de oraciones que fueron predichas como relevantes (Figura 5.6). Con estos datos como entrada, se generan los registros para los datos de entrenamiento de la verificación de afirmaciones. Estos registros tendrán los campos etiqueta-afirmación-oración, etiqueta corresponde a "0" si la afirmación está etiquetada como refutada en *FEVER*, "1" si tiene la etiqueta apoyada y "2" en caso de sin información.

Figura 5.6: El *dataset* de ejemplos para la tarea de verificación se crea a partir de las predicciones hechas por el modelo entrenado para la tarea de relevancia de oraciones.



5.3. Configuración experimental

El objetivo de este trabajo es crear un par de modelos que se encarguen de tratar la tarea de identificación de oraciones relevantes y la verificación de afirmaciones. Estas

tareas corresponden a los pasos 2 y 3 de la verificación automática de hechos, presentadas anteriormente (ver sección 3.4.1).

Las redes neuronales pueden ser entrenadas sin ningún problema utilizando exclusivamente la CPU, sin embargo, las operaciones llevadas a cabo durante el entrenamiento y posteriormente en la inferencia, se adaptan naturalmente al procesamiento en paralelo que ofrece una *GPU*. En la arquitectura propuesta se está utilizando un modelo preentrenado que se basa en la arquitectura *transformer*. Este tipo de arquitecturas suelen ser computacionalmente muy demandantes en el entrenamiento, e inclusive la inferencia es habitualmente muy lenta en una CPU. Estos modelos no sólo son demandantes de procesamiento, también utilizan una cantidad importante de memoria. En el caso de *BERT* esto es muy notable cuando se utiliza el modelo grande o se utilizan secuencias de *tokens* cercanas al límite de 512. También otro aspecto a tomar en cuenta es el tamaño del lote de entrenamiento, la documentación de *BERT* [2] sugiere utilizar tamaños de lote superiores a 32 para ajustar, en especial, el modelo grande. De otra forma se corre el riesgo de perder los pesos preentrenados, incluso con tasas de aprendizaje muy pequeñas. Por las razones expuestas, para la experimentación presentada se utilizó una *GPU* V100 con 32GB de memoria. Esta capacidad de hardware permitió hacer el entrenamiento con lotes de tamaño 64, atendiendo de esta forma la recomendación del tamaño del lote para ajustar un modelo basado en *BERT*.

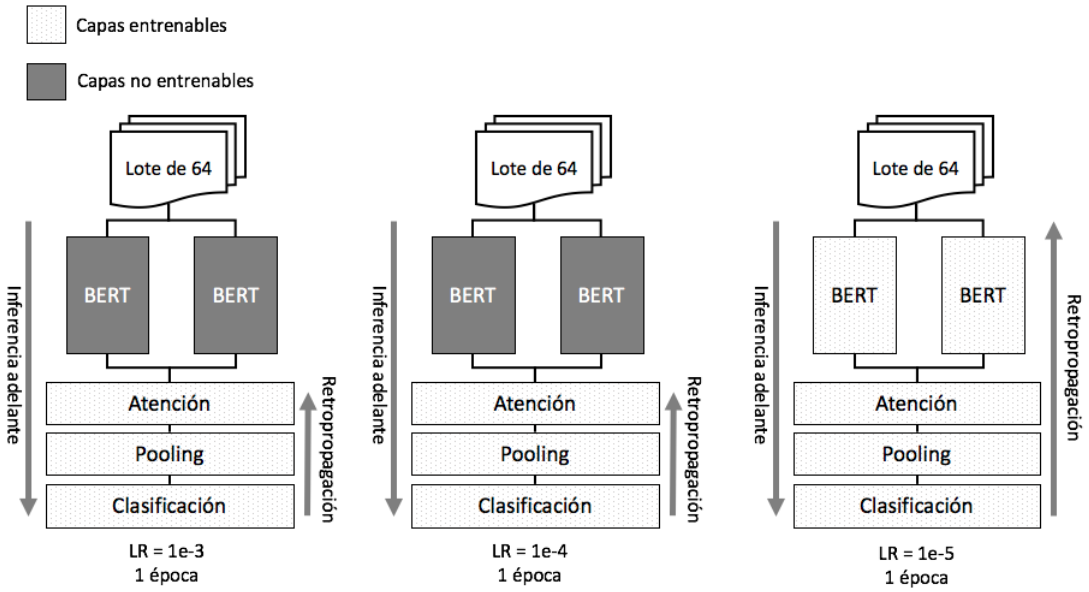
5.3.1. Modelo para evaluación de relevancia

Como resultado del preprocesamiento del conjunto de datos, se obtuvo una lista con parejas de afirmación-oración y una etiqueta que indica si la oración es relevante o no para la afirmación. Este preprocesamiento se llevó a cabo en el conjunto de datos de *FEVER* [37] en la partición de entrenamiento y en la de pruebas, con el objetivo de llevar a cabo dichas tareas de aprendizaje automatizado sobre los modelos propuestos. Los datos resultaron en aproximadamente 381k ejemplos para entrenamiento, y 45k para evaluación. Con estos datos como entrada se configuró el entrenamiento del modelo.

Dado que se está utilizando el modelo preentrenado de lenguaje *BERT*, una buena práctica al trabajar con este tipo de aprendizaje por transferencia es realizar un ajuste fino sobre los parámetros de *BERT*, ajustando al mismo tiempo las capas que se agregaron para llevar a cabo la tarea específica. Durante la experimentación, el entrenamiento se llevó a cabo de la siguiente manera: primero se congeló el bloque de *BERT*, con el objetivo de entrenar únicamente las capas de atención, *pooling*, concatenación y clasificación. Se inició con una tasa de aprendizaje de $1e-3$. Con este valor de tasa se entrenó el modelo por 1 época. Posteriormente, se redujo el valor de la tasa a $1e-4$ y se volvió a entrenar por 1 época. La intención de estas etapas previas de entrenamiento es que las capas agregadas después de *BERT* se ajusten poco a poco a la tarea que se pretende resolver. La reducción de la tasa de aprendizaje permite que la optimización de los pesos sea más fina y se vaya acercando más a los valores esperados. Una vez que se completaron estas dos épocas, la tasa de aprendizaje se reduce a $1e-5$ y se descongelan los parámetros de *BERT*. Este valor es el que se recomienda para realizar un ajuste fino de *BERT*. Entonces, a continuación se realiza un ajuste de los pesos del modelo de lenguaje para adaptarse aún más a la tarea de evaluación de la relevancia, se entrena por una época más. El detalle de este entrenamiento se puede visualizar en la Figura 5.7.

[2] `https://github.com/google-research/bert`

Figura 5.7: Etapas del entrenamiento, reduciendo la tasa de aprendizaje. Inicialmente el bloque de *BERT* está congelado y sólo se ajustan los pesos de las capas extras.



Se probaron distintas estrategias para el entrenamiento como, por ejemplo, ajustar *BERT* desde un inicio con una tasa muy baja, no ajustar *BERT* en ninguna etapa, entre otras. Pero la configuración de 3 épocas, reduciendo la tasa de aprendizaje fue la que mejores resultados entregó. Todos los experimentos en adelante se trabajaron de esta manera.

Para el entrenamiento se utilizó una *GPU* V100, y en promedio los experimentos realizados demoraron 12 horas cada uno en ejecutarse.

5.3.2. Modelo para verificación de afirmación

Para el entrenamiento del modelo encargado de realizar la verificación de la afirmación tomando en cuenta la evidencia, se utiliza la salida de la etapa de extracción de oraciones relevantes. Como salida de esta etapa previa se obtiene una lista de afirmaciones, y para cada una de ellas un conjunto de oraciones que fueron predichas como relevantes. Estas predicciones se realizaron utilizando el modelo entrenado para la evaluación de la relevancia de una oración. Se tomaron las 5 oraciones más relevantes para formar la evidencia que posteriormente se usa para la tarea de verificación. Para el conjunto de entrenamiento se obtuvieron un total de 724k parejas afirmación-evidencia, con su respectiva etiqueta (apoyada, refutada o sin información). Para el conjunto de validación resultaron un total de 99k ejemplos con la misma estructura descrita para el entrenamiento. La distribución de las etiquetas para ambos conjuntos de datos se muestra en la Tabla 5.2.

Como se puede observar en la Tabla 5.2 las clases se encuentran muy desbalanceadas, con una cantidad considerablemente mayor para la clase apoyada, la clase refutada es la que menos ejemplos tiene. Durante la experimentación, nos dimos cuenta de que el modelo se estaba sesgando de forma importante hacia la clase con mayor cantidad de datos. Por este motivo, se decidió balancear de forma artificial el *dataset*, obteniendo aleatoriamente un

Tabla 5.2: Distribución de los ejemplos en las 3 clases, después de predecir las oraciones relevantes para cada afirmación.

	Apoyada	Refutada	Sin información	Total
Entrenamiento	398,818	148,148	177,443	724,409
Validación	33,083	33,154	33,121	99,358

número de ejemplos para cada clase, de tal manera que todos tuvieran el mismo tamaño. En este caso, todas las clases quedaron con la cantidad de ejemplos que originalmente tenía la clase refutada. Finalmente, este *dataset* balanceado cuenta con 444k registros, distribuidos de forma equitativa entre las tres clases.

Una vez construido el *dataset*, se entrenó el modelo propuesto durante un par de épocas. Para esta tarea, la reducción de la tasa de aprendizaje y el congelamiento de *BERT* no dio mejores resultados como en el caso de la evaluación de la relevancia. El entrenamiento se llevó a cabo realizando un ajuste fino por 2 épocas, utilizando una tasa de aprendizaje de $1e-5$.

Para el entrenamiento se utilizó una *GPU* V100, y en promedio los experimentos realizados demoraron 6 horas cada uno en ejecutarse.

5.4. Resumen

El *dataset* que se utilizó para la experimentación (*FEVER* [37]) requirió de un preprocesamiento, buscando simplificar el entrenamiento. Este proceso previo necesitó buscar en un conjunto de más de 5 millones de artículos de *Wikipedia*, la evidencia anotada en el *dataset* original. Esto sólo fue necesario para el entrenamiento del modelo de evaluación de relevancia. Para el caso de la verificación de afirmaciones, los datos de entrenamiento se obtuvieron de las predicciones del modelo ya entrenado para la relevancia. Como resultado se generaron un par de archivos correspondientes a cada una de las tareas en cuestión.

Entrenar un modelo que ocupa aprendizaje por transferencia permite utilizar diferentes configuraciones. Por ejemplo, se puede mantener congelado el modelo preentrenado buscando que simplemente se haga una extracción de características y en este caso sólo es necesario ajustar las capas extras que se agregan a la arquitectura. Por otro lado, también se puede experimentar ajustando todo el modelo desde un inicio utilizando una tasa de aprendizaje muy baja, tratando de evitar destruir los pesos aprendidos anteriormente. En nuestro caso, un entrenamiento mixto fue lo que mejores resultados entregó, entrenando con una tasa más grande, pero manteniendo congelado los parámetros de *BERT*. Posteriormente se reduce la tasa de aprendizaje, se descongela *BERT*, y ahora sólo es necesario ajustar un poco más todos los pesos para optimizar las salidas. La intuición detrás de esta forma de entrenar es que inicialmente el error producido por las capas extras es muy grande, y este error se retropropaga por todo el modelo pudiendo afectar el preentrenamiento de *BERT*. Pero si se ajustan inicialmente las capas extras, entonces cuando se descongela *BERT*, ya usando una tasa muy pequeña, se puede evitar tocar más de lo necesario los pesos de *BERT*.

Capítulo 6

Resultados experimentales

En este capítulo se detallan los experimentos que se realizaron y las distintas configuraciones que se pusieron a prueba, así como los resultados obtenidos. Cualquier arquitectura de aprendizaje profundo tiene muchas variantes que permiten evaluar empíricamente el efecto que pueden tener en el desempeño final del modelo. Como consecuencia, las redes neuronales tienen una gran flexibilidad, pero al mismo tiempo se dificulta la experimentación. La gran cantidad de hiperparámetros y configuraciones que puede tener una arquitectura, hace prácticamente imposible probar cada una de estas variantes. La razón principal es que, por ejemplo, el *dataset* para entrenar la verificación de la afirmación tiene en total 724k ejemplos de parejas afirmación-evidencia. Entrenar una red basada en *transformer* es por sí misma una tarea demandante de recursos, ahora si se considera un corpus del tamaño como el que se describe, cada experimento puede demorar más de 12 horas.

Para tratar de cubrir una cantidad mayor de configuraciones se planteó utilizar un conjunto de datos de menor tamaño para poder experimentar más ágilmente y reducir el espacio de búsqueda a un menor número de configuraciones posibles. El *dataset* reducido se creó seleccionando aleatoriamente ejemplos del conjunto de entrenamiento y del de validación. Como resultado se tienen 30k ejemplos para entrenamiento y 10k para validación. A continuación, se presentan los experimentos que se realizaron sobre el conjunto reducido. Como resultado se seleccionaron las configuraciones que tuvieron mejor rendimiento.

También es importante mencionar que la evaluación de los experimentos se realizó de forma individual. Esto quiere decir que para cada una de las tareas (relevancia y verificación) se evalúa la exactitud que el modelo tiene para predecir las etiquetas de cada caso. Una vez que se encontraron los modelos que mejor rendimiento tienen de forma individual en cada una de las tareas, se eligieron los mejores y entonces se evalúo de forma conjunta todo el método de verificación de hechos, para lo que se utilizó precisión, *recall* y F1 para medir y comparar los resultados.

6.1. Experimentos con bloques de *BERT*

La red siamesa utiliza esquemáticamente un par de bloques de codificación *BERT* que tienen el objetivo de tomar como entrada, uno de ellos la afirmación y el otro la evidencia. Es posible entrenar un modelo basado en *BERT* utilizando únicamente los vectores que genera sin modificar sus pesos, es decir, manteniendo congelados los parámetros preentrenados.

No obstante, estos bloques participan del proceso de inferencia y tienen influencia en el error que se produce en la capa de salida, por lo cual, es recomendable ajustar los pesos preentrenados de este modelo de lenguaje. Se experimentó entrenando el modelo con *BERT* siempre congelado, pero se observó que los resultados estaban muy alejados del estado del arte. Ajustar estos parámetros de forma fina permitió que el modelo tuviera un rendimiento más cercano al estado del arte.

Otra estrategia para ajustar el modelo, que ya fue previamente presentada en el capítulo anterior, es utilizar un esquema mixto. La intención es ajustar inicialmente sólo las capas extras del modelo (atención, *pooling* y clasificación) de tal manera que el error (que es al comienzo muy grande) se reduzca, y los parámetros de las capas extra se acerquen más a los valores óptimos. Para esta primera etapa, *BERT* está congelado y se usa una tasa de aprendizaje de 1-e3. Después de reduce la tasa a 1-e4 sin descongelar *BERT* y se continúan ajustando las capas extra, la intuición de este paso es que el modelo se vaya acercando más lentamente al punto óptimo, sin provocar que el error se dispare. En este punto el modelo ya no mejora, aunque se continúe entrenando por más épocas. Entonces ahora sí, es necesario ajustar los pesos de *BERT* para tratar de reducir aún más el error. Como recomienda la documentación [1] se utiliza una tasa aún más pequeña 1e-5. Se entrena el modelo por una época más y como resultado el rendimiento mejora sustancialmente. Estos experimentos se realizaron de forma previa con el objetivo de encontrar la mejor estrategia para ajustar el modelo en conjunto, no se presentan los resultados concretos. Valdría la pena explorar con mayor profundidad el efecto que tiene esta estrategia de entrenamiento en diferentes arquitecturas que utilizan aprendizaje por transferencia. Para todos los experimentos descritos en adelante se utilizó la estrategia planteada.

Una variante adicional que es importante explorar es si es conveniente utilizar pesos compartidos en el bloque *BERT*. En [32] utilizan una red siamesa para ajustar el modelo de *BERT* para que sea capaz de generar una representación vectorial única de una oración (*Sentence-BERT*), y justamente exploran esta variante en la red siamesa. Para la tarea de similitud semántica resultó que los mejores resultados son cuando el bloque *BERT* comparte los pesos. Para la tarea que nos interesa en este trabajo, se pusieron a prueba este par de configuraciones. De forma similar al artículo citado, encontramos que el mejor rendimiento durante el entrenamiento y en la validación resultó de utilizar un bloque *BERT* con pesos compartidos. En la Tabla 6.1 se pueden observar las diferencias que se obtuvieron en la exactitud en la tarea de evaluación de la relevancia. La interpretación que le damos a este resultado es que utilizar pesos separados provoca que exista una divergencia entre los espacios vectoriales generados en cada una de las ramas. Entonces, si bien las representaciones entregadas por *BERT* siguen siendo semánticamente significativas, estas diferencias entre la representación de la afirmación y la evidencia provoca que los vectores no sean comparables. Para que la capa de atención funcione como se plantea, es requisito que se pueda medir la similitud entre las palabras de cada una de las oraciones. Y esto se consigue precisamente, si el espacio de representación de las oraciones se mantiene constante, a través de mantener un bloque codificador *BERT* único.

[1] `https://github.com/google-research/bert`

Tabla 6.1: Comparación de utilizar pesos compartidos de *BERT*. Se compara utilizando un modelo con atención multiplicativa y otro con atención coseno (conjunto de datos reducido).

Modelo	Entrenamiento	Validación
(Atn Mul) + BERT no compartido	0.9019	0.8564
(Atn Cos) + BERT no compartido	0.9118	0.8222
(Atn Mul) + BERT compartido	**0.9278**	**0.8724**
(Atn Cos) + BERT compartido	0.9406	0.8622

6.2. Experimentos con la capa de atención

En la capa de atención se probaron 3 tipos de mecanismos de atención: multiplicativo con función *softmax*, multiplicativo con función *sigmoide* y similitud coseno. Para mayor referencia de estos métodos revisar el capítulo 2. El objetivo de la capa de atención es calcular una matriz de dimensión 128×128 (128 corresponde al tamaño máximo de la secuencia) cuyos elementos reflejen la importancia que tienen cada uno de los elementos de la segunda secuencia con respecto al cada elemento de la primera secuencia, la ponderación se calcula todos contra todos. Cada mecanismo propuesto tiene características que reflejan de una manera distinta la similitud entre un par de vectores. Se tomó como base un bloque *BERT* con los pesos compartidos y una capa de clasificación simple para evaluar los mecanismos. En la Tabla 6.2 se muestran los resultados para la tarea individual de clasificación de relevancia. El mejor resultado lo tuvo la atención multiplicativa aplicándole una función *softmax* sobre la segunda dimensión de la matriz. Con este tipo de atención, los pesos para un elemento específico modelan una distribución de probabilidad, por lo que suman 1 y es posible identificar aquellos elementos con mayor ponderación. Esta característica será útil para comprender la decisión tomada por el modelo acerca de la verificación de la afirmación.

Tabla 6.2: Experimentos con diferentes mecanismos de atención (conjunto de datos reducido). Se presenta la exactitud obtenida.

Modelo	Entrenamiento	Validación
Multiplicativa + Softmax	**0.9278**	**0.8724**
Multiplicativa + Sigmoide	0.8971	0.8243
Coseno	0.9406	0.8622

6.3. Experimentos *pooling*

La capa de *pooling* busca reducir las secuencias de entrada de tal manera que un vector único represente a esta secuencia. Se planteó anteriormente utilizar dos estrategias para realizar la tarea. Primero, utilizando el promedio de los vectores de la secuencia, para lo cual es necesario sumar todos lo vectores y obtener el promedio de cada una de sus dimensiones, este tipo de método suele dar buenos resultados [32]. La otra estrategia para el *pooling* es utilizar una red recurrente *LSTM*. Utilizar redes recurrentes da la posibilidad de experimentar con otros hiperparámetros, como el tamaño del estado oculto y también la cantidad de capas. Se probó con 1, 5 y 10 capas en la red recurrente. La representación final de la secuencia se toma

del último estado oculto. En la Tabla 6.3 se muestran los resultados de utilizar diferentes estrategias para el *pooling*. Estos resultados corresponden al entrenamiento de la tarea de clasificación de relevancia. Los mejores resultados se obtuvieron utilizando una red *LSTM* de 5 capas, utilizar más de este número de capas no ayudó a mejorar el rendimiento, incluso están por debajo de la exactitud que se logró con el promedio de los vectores.

Tabla 6.3: Experimentos con estrategias para representar la secuencia con un vector único (conjunto de datos reducido). Se presentan las exactitudes.

Modelo	Entrenamiento	Validación
Promedio	0.9278	0.8724
LSTM 1 capa	0.8857	0.8322
LSTM 5 capas	**0.9270**	**0.8746**
LSTM 10 capas	0.7559	0.7041

6.4. Experimentos capa de clasificación

Inicialmente se probaron las configuraciones con una capa de clasificación sencilla, simplemente se agrega una capa completamente conectada (FC) después de la concatenación de los vectores, y se agrega una función *softmax* como activación. Sin embargo, también se exploró la conveniencia de robustecer este bloque. Se agregaron más capas completamente conectadas, normalización por lote y también *dropout*. En la Tabla 6.4 se muestra la exactitud obtenida al entrenar el modelo con las diferentes configuraciones. Agregar más de dos capas *FC* provoca que el rendimiento empiece a caer, por lo que se decidió utilizar solamente 2 capas, además de la de salida. Adicionalmente, se probaron distintas funciones de activación para cada una de las 2 capas ocultas. Se probaron distintas combinaciones de ReLU, sigmoide y tangente hiperbólica (Tanh). La combinación óptima fue utilizar la función sigmoide y la Tanh. Así mismo, se agregó normalización por lote, lo cual trajo mejoras en el rendimiento. Vale la pena mencionar que también se probó agregar una capa de *dropout*, pero esta modificación no mejoró la exactitud lograda. En la Figura 4.9 se presenta la configuración resultado de estos experimentos.

Tabla 6.4: Exactitud obtenida de los experimentos con diferentes configuraciones en el bloque de clasificación (conjunto de datos reducido).

Clasificación	Entrenamiento	Validación
1 lineal extra	0.8462	0.8049
2 lineales extras	0.8429	0.8079
3 lineales extra	0.8433	0.8061
2 lineales extra + RELU	0.8182	0.7816
2 lineales extra + Tanh	0.8693	0.8216
2 lineales extra + Relu + Tanh	0.8586	0.8169
2 lineales extra + Tanh + Tanh	0.8510	0.8053
2 lineales extra + Sigm + Tanh	0.8608	0.8222
2 lineales extra + Sigm + Tanh + Norm	**0.8977**	**0.8394**

6.5. Experimentación con otros modelos preentrenados

BERT es un modelo preentrenado de lenguaje que se basa en la arquitectura *transformer*. *BERT* se refiere no sólo a una arquitectura, sino también a las tareas pretexto y técnicas usadas durante el preentrenamiento. Fundamentados en esta idea, existen muchos otros modelos basados en *transformer* que intentan mejorar las debilidades de *BERT*, o se enfocan en tareas más específicas de PLN. Como bloque codificador en nuestra arquitectura se probó con modelos distintos a *BERT*, como *Roberta* [19] y *GPT-2* [30]. En la Tabla 6.5 se encuentra la exactitud obtenida utilizando diferentes modelos preentrenados para la verificación de afirmación. *Roberta* ajustado para la tarea de similitud semántica (*SentBert-Roberta*) alcanzó la exactitud más alta. Para la tarea de clasificación de relevancia utilizar otros modelos no mostró mejoría alguna, en la Tabla 6.6 se puede observar la comparación de la exactitud obtenida para la clasificación de relevancia. Entonces, para la tarea de clasificación de relevancia se continuó utilizando *BERT* base. Para la verificación de afirmación se utilizó *SenteceBert-Roberta*, aunque también se continuaron pruebas con *BERT* base para esta tarea.

Tabla 6.5: Exactitud obtenida en la tarea de verificación de afirmación utilizando diferentes modelos preentrenados además de *BERT* base (conjunto de datos reducido).

Modelo preentrenado	Entrenamiento	Validación
BERT base	75.55	60.74
Roberta	77.06	61.74
GPT-2	50.4	48.88
SentenceBERT	77.92	61.30
SentenceBERT (Roberta)	**76.82**	**61.86**

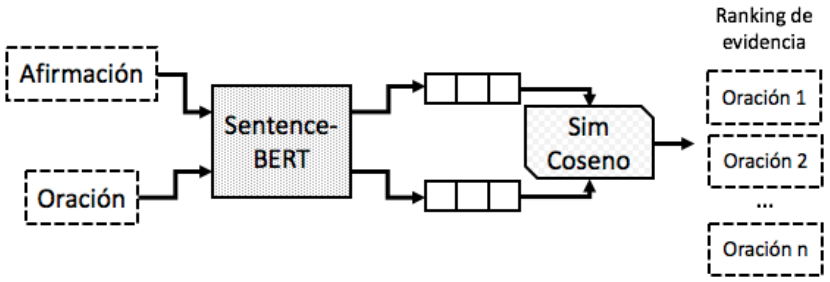
Tabla 6.6: Exactitud obtenida en la tarea de clasificación de relevancia utilizando diferentes modelos preentrenados además de *BERT* base (conjunto de datos reducido).

Modelo preentrenado	Entrenamiento	Validación
BERT base	**92.58**	**86.43**
Roberta	90.02	84.76
GPT-2	84.23	78.41
SentenceBERT	88.10	84.72
SentenceBERT (Roberta)	89.92	84.12

Todos los experimentos y resultados descritos hasta ahora, se llevaron a cabo en el *dataset* reducido. De esta experimentación se obtuvo un subconjunto de modelos que tuvieron los mejores resultados en el conjunto de datos reducido. Con dichos modelos, se realizaron posteriormente experimentos en el conjunto completo.

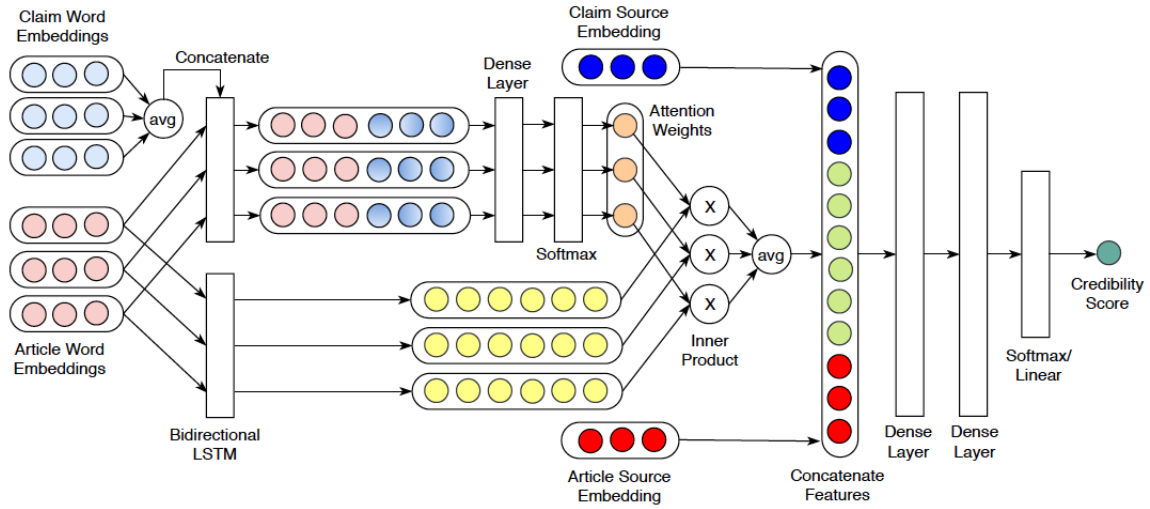
Como *baseline* para la tarea de clasificación de relevancia, se utilizó *Sentence-BERT* [32]. Este es un modelo basado en *BERT* que obtiene una representación vectorial de una oración. El vector generado es de utilidad debido a que es semánticamente significativo, lo cual quiere decir que es posible medir la similitud semántica entre un par de oraciones a través de diversas operaciones. Para el caso que se está presentando, la similitud se mide con la función coseno. El modelo de referencia evalúa una a una las oraciones de los documentos relevantes, y ordena de mayor a menor similitud. Al final se devuelven las n oraciones más similares a la afirmación, que serán por lo tanto las más relevantes (Figura 6.1). Para comparar este modelo con las arquitecturas propuestas, se realizó la clasificación de la evidencia considerando su similitud con la afirmación, si la similitud entre la afirmación y la evidencia es menor a 0.4 entonces la evidencia se considera como relevante, en el caso contrario se clasifica como no relevante.

Figura 6.1: Modelo de referencia para la clasificación de relevancia.



Como *baseline* para la tarea de verificación de la afirmación se utiliza la arquitectura propuesta por [27]. Los autores de este artículo proponen un modelo basado en *LSTM* que evalúa la veracidad de una afirmación considerando evidencia, la fuente de la afirmación y la fuente de la evidencia. Para el *baseline*, adaptamos este modelo quitando las entradas referentes a las fuentes, ya que la credibilidad de la evidencia se da por sentada en nuestro modelo (Figura 6.2).

Figura 6.2: Modelo de referencia para la verificación de afirmaciones. Imagen tomada de [27].



6.6. Clasificación de relevancia

Para la clasificación de relevancia se entrenaron las 4 configuraciones de arquitectura que mejores resultados dieron con el conjunto de datos reducido, pero ahora se utiliza el conjunto de datos completo. En la Tabla 6.7 se muestran las exactitudes obtenidas por las 4 arquitecturas, y se comparan con el modelo de [35] y el *baseline* que utiliza *SentenceBERT* [32]. El modelo con la exactitud más alta durante la validación, es el que utiliza atención multiplicativa, y una red *LSTM* de 5 capas para obtener la representación de la secuencia. Se observa que este modelo no supera al estado del arte [35], pero se acerca bastante. Nuestro mejor modelo para esta tarea mejora considerablemente al *baseline*. En la misma Tabla se muestran unas etiquetas que utilizaremos para referirnos a nuestros 3 mejores modelos de clasificación de relevancia, R1 para el mejor modelo, R2 para el segundo y R3 para el tercero.

Tabla 6.7: Exactitud de las 4 arquitecturas propuestas para la tarea de clasificación de relevancia (conjunto de datos completo).

	Modelo	Entrenamiento	Validación
Soleimani 2020		0.9298	0.8996
Bert (Compartido) + Multiplicativo + Softmax	R2	0.9261	0.8892
Bert (Compartido) + Coseno	R3	0.9419	0.8858
Bert (Compartido) Multiplicativo + Sigmoide		0.9241	0.8651
Bert (Compartido) + Multiplicativo + Pool(LSTM-5L)	**R1**	**0.9322**	**0.8937**
Baseline		0.6649	0.6553

6.7. Verificación de afirmación

Para la verificación de afirmación se hizo lo mismo que se describió anteriormente, con las 4 arquitecturas que mejores resultados tuvieron en el conjunto de datos reducido, se entrenó con el conjunto completo. La Tabla 6.8 compara la exactitud obtenida durante la validación de las 4 arquitecturas, el estado del arte [35] y el *baseline* [27]. El modelo con la mayor exactitud fue el que utiliza *SentenceBERT-Roberta*, atención multiplicativa y *pooling* por promedio. De forma similar que con la tarea anterior se aprecia que el modelo no supera al estado del arte [35], aunque supera considerablemente al *baseline* [27]. Para esta tarea, *SentenceBERT-Roberta* demostró un mejor rendimiento comparado con *BERT*.

6.8. Evaluación del método completo

El método de 3 pasos que se está implementando en este trabajo implica que inicialmente se deben recuperar n documentos relevantes de una colección. Posteriormente, de estos documentos se extraen las oraciones que son relevantes para la afirmación. Por último, se verifica la afirmación considerando la evidencia encontrada. Cada una de las etapas planteadas, implica un grado de incertidumbre, puede existir un error desde la recuperación de los documentos, el cual se propaga a la extracción de evidencia, que a su vez agrega su propio error. Al llegar a la tercera etapa ya se viene arrastrando el error de las dos anteriores, y claro esta última etapa también tiene potencialmente su propio

Tabla 6.8: Exactitud de las 4 arquitecturas propuestas para la verificación de afirmación (conjunto de datos completo).

	Entrenamiento	Validación
Soleimani 2020	0.8320	0.6576
Bert (Compartido) + Multiplicativo + Softmax	0.8102	0.6338
Bert (Compartido) + Multiplicativo + Pool(LSTM-5L)	0.7620	0.5582
SentBert-Roberta (Compartido) + Multiplicativo + Softmax	**0.7936**	**0.6378**
SentBert-Roberta (Compartido) + Multiplicativo + Pool(LSTM-5L)	0.7878	0.6343
Baseline	0.5693	0.5233

error. Por estas razones, es importante que se pueda evaluar de forma integral todo el método, para medir el impacto que tiene el error de los primeros pasos en los posteriores. Como se comentó anteriormente, para la recuperación de los documentos relevantes se utilizan las predicciones del modelo creado por [15]. Con estas predicciones, se obtienen los artículos de *Wikipedia*, y con el modelo entrenado para la clasificación de relevancia, se extraen las oraciones que son relevantes. Con las predicciones de las oraciones relevantes, en conjunto con la afirmación, se realiza la verificación usando el modelo entrenado para dicho fin.

Para cada etapa se seleccionaron los 3 modelos que mejores resultados mostraron individualmente, por lo tanto, en total se tienen nueve combinaciones diferentes entre los modelos seleccionados.

Para la etapa de clasificación de relevancia, se presenta la Tabla 6.9 la cual detalla precisión, *recall* y F1 considerando una ventana de recuperación de oraciones de 5. Esto significa que se están recuperando las 5 oraciones que resultaron ser más relevantes y se está evaluando cuántas de estas 5 oraciones son realmente relevantes (precisión), y cuántas oraciones relevantes del total se pudieron extraer (*recall*). El artículo [35] refiere que para la tarea de verificación es importante enfocarse en el *recall*, con el objetivo de que la etapa de verificación tenga suficiente evidencia. Esto se puede entender ya que esta métrica se enfoca en medir qué tanta evidencia existente fue posible recuperar, en el caso de que no se haya podido recuperar una buena parte de la evidencia, entonces la tarea de verificación se complica porque no se cuenta con la suficiente evidencia para comparar la afirmación. Nuestro mejor modelo en esta tarea, nombrado como R1, supera al modelo de [35] en *recall*, pero debido a que la precisión es considerablemente más baja, resulta en un F1 menor.

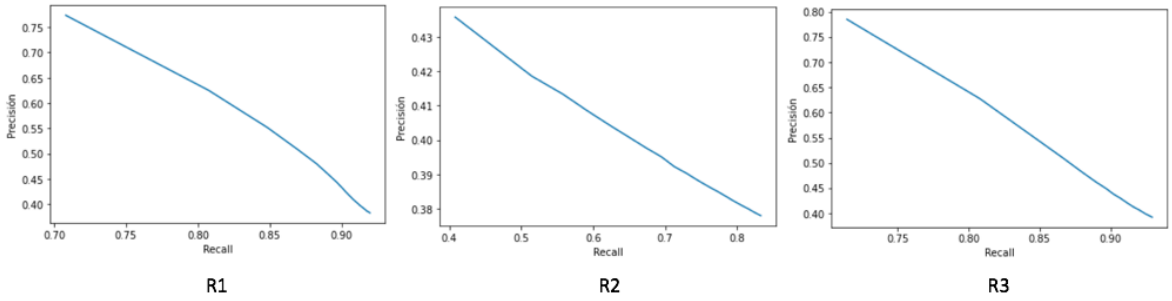
En la Figura 6.3, se presentan las gráficas precisión contra *recall* de los tres modelos considerados para la clasificación de relevancia. Se observa que el modelo R3 tiene un mejor equilibrio entre precisión y *recall*, lo cual se ve reflejado en un mayor *score* F1. Considerando el nivel de compromiso entre estas métricas, es claro que el R3 es superior. Pero basándonos en la hipótesis de [35] el R1 es más adecuado dado su mayor *recall*.

También es relevante observar el comportamiento de los modelos con distintos tamaños de ventana de búsqueda. En la Figura 6.4 es posible visualizar cómo es que la precisión cae rápidamente conforme se amplía la ventana. Para el caso del modelo R2 es evidente que su precisión es baja desde un inicio, y sólo sigue empeorando conforme se extraen más oraciones.

Tabla 6.9: Resultados de la clasificación de relevancia para una ventana de 5 oraciones recuperadas.

	Precisión@5	Recall@5	F1@5
Soleimani 2020	0.5828	0.8504	0.6299
Modelo R1 (Bert (Compartido) + Multiplicativo + Pool(LSTM-5L))	**0.4800**	**0.8823**	**0.5503**
Modelo R2 - Bert (Compartido) + Multiplicativo + Pool(Promedio)	0.4000	0.5900	0.4300
Modelo R3 - Bert (Compartido) + Coseno + Pool(Promedio)	0.4825	0.8796	0.5519
Baseline	0.3609	0.3951	0.3652

Figura 6.3: Gráficas *recall* vs precisión de los modelos de clasificación de relevancia.

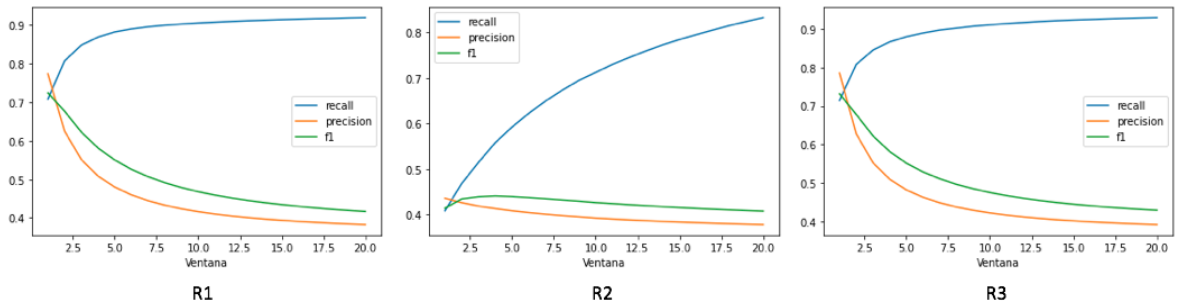


Los modelos R1 y R3 tienen un comportamiento similar en cuanto a *recall*, pero la precisión cae más rápidamente en el R1. Esto confirma lo expresado en la Tabla 6.9, el modelo R3 tiene un mejor compromiso entre la precisión y el *recall* lo cual se ve reflejado en un F1 superior.

La verificación de la afirmación toma las predicciones de la identificación de oraciones relevantes, cada uno de los 3 modelos de verificación puede tomar las predicciones de cualquiera de los 3 modelos de relevancia. En la Tabla 6.10 se presentan las exactitudes obtenidas de las 9 combinaciones distintas de modelo de relevancia y modelo de verificación. La exactitud más alta resulta de la combinación del modelo R1 y *Roberta* + Multiplicativo + Promedio. La mayoría de las configuraciones obtiene un resultado en un rango pequeño, es decir, no existe mucha diferencia entre las distintas configuraciones. Solamente el *baseline* se encuentra muy por debajo de estos valores. El estado del arte [35] tiene mayor exactitud, pero se encuentra bastante cercano a nuestro mejor modelo.

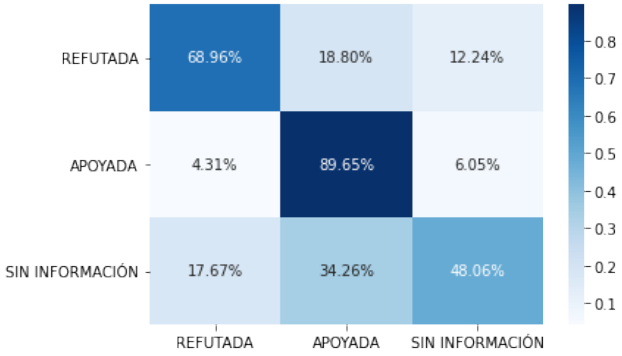
En las Figuras 6.5 y 6.6 están disponibles las matrices de confusión que comparan las clasificaciones del estado del arte [35] y nuestro mejor modelo respectivamente. En estas matrices se expresan los porcentajes del total de cada clase y cómo es que fueron clasificadas. Por ejemplo, en la Figura 6.5 de los ejemplos que realmente pertenecen a la clase refutada, el 68.96 % se clasificó como refutada, el 18.80 % como apoyada y el 12.24 % como sin información. Se aprecia en la Tabla 6.10 que el modelo de [35] tiene una exactitud superior a nuestros modelos, y en la matriz de confusión la distribución de las predicciones entre ambos modelos es bastante

Figura 6.4: Gráficas con diferentes tamaños de ventana en la recuperación de oraciones relevantes para los modelos R1, R2 y R3.



diferente. En la matriz de confusión correspondiente al modelo de [35] se observa un sesgo importante hacia la clase soportada, la mayoría de los ejemplos se clasifican de esta manera. Esto a pesar de que el conjunto de entrenamiento está completamente balanceado. Nuestro modelo en cambio se nota más balanceado, no tan sesgado a ninguna de las 3 clases; aunque la clase que más dificultad le impone es la de sin información. Esta reducción en el sesgo es relevante para la tarea de verificación de hechos, en la práctica clasificar buena parte de las afirmaciones como apoyadas (y por lo tanto se puede inferir que son verdaderas) puede no ayudar mucho en el combate a la desinformación porque si se clasifica como verdadera mucha información falsa o sin suficiente información, las afirmaciones que realmente son falsas requerirán aún una verificación manual para evitar que se sigan distribuyendo. Aunque también es claro que es necesario mejorar la exactitud en la verificación final de la afirmación. Ambos factores se deben considerar cuando se crea un modelo de verificación automática de hechos.

Figura 6.5: Matriz de confusión de la clasificación del modelo de [35].



6.9. Interpretabilidad

Los mejores resultados de nuestro modelo son bastante cercanos al estado del arte, pero se encuentran aún por debajo. Uno de los principales objetivos planteados inicialmente, es crear una arquitectura que sea capaz de clasificar una afirmación pero que también pueda dar retroalimentación al usuario final acerca de la decisión tomada. Esta parte es relevante ya

Tabla 6.10: Resultados de combinar distintas configuraciones de modelos de relevancia y modelos de verificación.

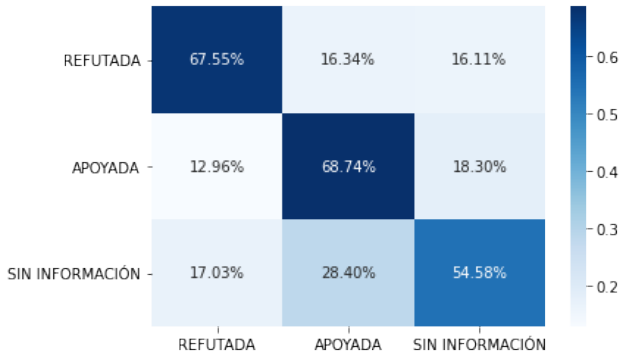
Relevancia	Verificacion	Exactitud
Soleimani 2020	Soleimani 2020	0.6889
Bert (Compartido) + Multiplicativo + Pool(LSTM-5L)	**SentBert-Roberta (Compartido) + Multiplicativo + Softmax**	**0.6362**
Bert (Compartido) + Multiplicativo + Pool(LSTM-5L)	SentBert-Roberta (Compartido) + Multiplicativo + Pool(LSTM-5L)	0.6336
Bert (Compartido) + Multiplicativo + Pool(LSTM-5L)	Bert (Compartido) + Multiplicativo + Softmax	0.6356
Bert (Compartido) + Multiplicativo + Softmax	SentBert-Roberta (Compartido) + Multiplicativo + Softmax	0.6358
Bert (Compartido) + Multiplicativo + Softmax	SentBert-Roberta (Compartido) + Multiplicativo + Pool(LSTM-5L)	0.6317
Bert (Compartido) + Multiplicativo + Softmax	Bert (Compartido) + Multiplicativo + Softmax	0.6330
Bert (Compartido) + Coseno	SentBert-Roberta (Compartido) + Multiplicativo + Softmax	0.6360
Bert (Compartido) + Coseno	SentBert-Roberta (Compartido) + Multiplicativo + Pool(LSTM-5L)	0.6327
Bert (Compartido) + Coseno	Bert (Compartido) + Multiplicativo + Softmax	0.6352
Baseline	Baseline	0.5233

que normalmente las redes neuronales son vistas como cajas negras, no se tiene mucha noción acerca de lo que ocurre dentro del modelo o cuáles son los factores que toma en cuenta para clasificar. En este sentido, un mecanismo de atención puede ser una buena pista acerca de lo que está ocurriendo en la red, o también de los elementos de la entrada que está considerando son importantes.

Para otorgarle interpretabilidad al modelo planteamos utilizar la capa de atención que pondera la importancia que tienen los *tokens* de la evidencia con respecto a la afirmación. La capa de atención computa una matriz cuyos elementos son precisamente estas ponderaciones, las cuales se calculan todos contra todos. Con estas ponderaciones la arquitectura le da más importancia a algunos vectores de la secuencia de entrada. En las Figuras 6.7, 6.8 y 6.9 se muestran tres ejemplos de parejas afirmación-evidencia que son evaluados por el modelo. Al evaluar estas parejas, se obtiene el mapa de atención calculado, en el cual se ven reflejadas las ponderaciones. En las Figuras es posible observar a grandes rasgos, a qué *tokens* de la evidencia le da más importancia el modelo con respecto a cada *token* de la afirmación. La importancia de los *tokens* se muestra a través de un código de colores, los *tokens* más importantes aparecen de un tono azul más intenso. Los *tokens* resaltados son las partes de la evidencia que son relevantes para la verificación de la afirmación.

La matriz de ponderaciones es útil para concentrarse en algunos vectores de la secuencia de entrada. Para aprovechar la información contenida en ella se calculó el promedio de las ponderaciones de cada uno de los *tokens* de la evidencia. De esta forma se puede mostrar

Figura 6.6: Matriz de confusión de la clasificación del modelo R1 + [SentBert-Roberta (Compartido) + Multiplicativo + Softmax].



cuáles son los aspectos de la evidencia que la red neuronal consideró para clasificar la evidencia. Como resultado de este promedio en el eje 0 de la matriz, se obtiene un vector de una sola dimensión que contiene los pesos de la evidencia. En las Figuras 6.10, 6.11 y 6.12 se visualiza las ponderaciones calculadas para la evidencia, en los mismos tres ejemplos planteados. Se puede apreciar la importancia que calculó el modelo para clasificar la afirmación, y de esta manera contamos con una interpretación a la salida obtenida. En el texto de la evidencia se muestran los *tokens* más relevantes de un color azul más intenso, la intensidad de este color refleja la importancia del *token* para la verificación de la afirmación.

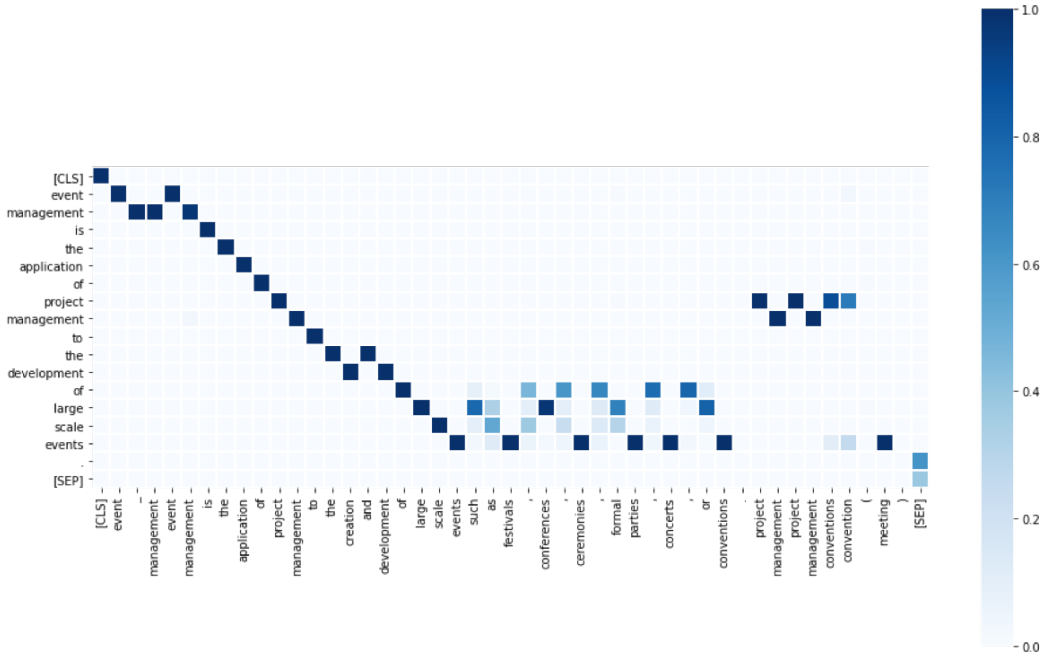
6.10. Resumen

A partir de los resultados mostrados en este capítulo, se puede confirmar que la arquitectura propuesta es capaz de recuperar evidencia que es relevante para una afirmación. Además, utilizando la evidencia encontrada es posible clasificar la afirmación como apoyada, refutada o sin información. Todo lo anterior manteniendo un rendimiento bastante cercano al estado del arte en la tarea de verificación automática de hechos.

Se experimentó con diversas configuraciones en cada uno de los elementos de la arquitectura, buscando mejorar las predicciones del modelo. También, el utilizar una estrategia mixta durante el entrenamiento, permitió que el ajuste al modelo preentrenado de lenguaje sea únicamente lo necesario para adaptarse a la tarea de verificación de hechos. Esto permite conservar lo más posible los pesos aprendidos durante el entrenamiento, y esto se ve reflejado en mejores métricas durante la validación. Adicionalmente, en el bloque de *BERT*, utilizar pesos compartidos en ambas ramas de la red siamesa demostró ser de gran utilidad, ya que de esta forma las representaciones tanto de la afirmación como de la evidencia se mantienen constantes. El que no fuera constante esta representación significaría que para una misma secuencia de *tokens* se obtendrían vectores diferentes dependiendo si la secuencia es afirmación o evidencia.

En la capa de atención, el mecanismo multiplicativo aplicando una función *softmax* obtuvo los mejores resultados, comparado con la función coseno y la sigmoide. La atención multiplicativa tiene la capacidad de medir qué tan parecidos son un par de vectores, y

Figura 6.7: Mapa de atención obtenido del modelo. Ejemplo 1.



aplicando la función *softmax*, mantiene estos *scores* de similitud en un rango bien delimitado el cual permite ponderar de forma controlada qué tan importantes son los elementos de la evidencia con respecto de la afirmación. Esto sin duda ayudó a la arquitectura a realizar una clasificación más precisa.

En el cuanto al bloque de clasificación, utilizar un par de capas ocultas entregó mejores resultados que utilizar sólo una capa de salida. El utilizar una combinación de funciones de activación diferentes entre estas dos capas fue también de utilidad.

Un aspecto importante que se describió en este capítulo fue la utilidad que tiene la capa de atención. La capa de atención calcula una matriz de ponderaciones entre los elementos de ambas secuencias de entrada. En cada proceso de inferencia se genera una matriz de atención particular para cada caso. La matriz guarda la información de los aspectos en los cuales la red neuronal se está enfocando más, y cuáles está dejando de lado. Por este motivo, el mapa de atención generado se utiliza como una aproximación para entender que es lo qué está tomando en cuenta el modelo para decidir si una afirmación es apoyada, refutada o sin información. Poder visualizar estas ponderaciones le da un valor agregado a nuestro modelo porque da información útil al usuario final.

Figura 6.8: Mapa de atención obtenido del modelo. Ejemplo 2.

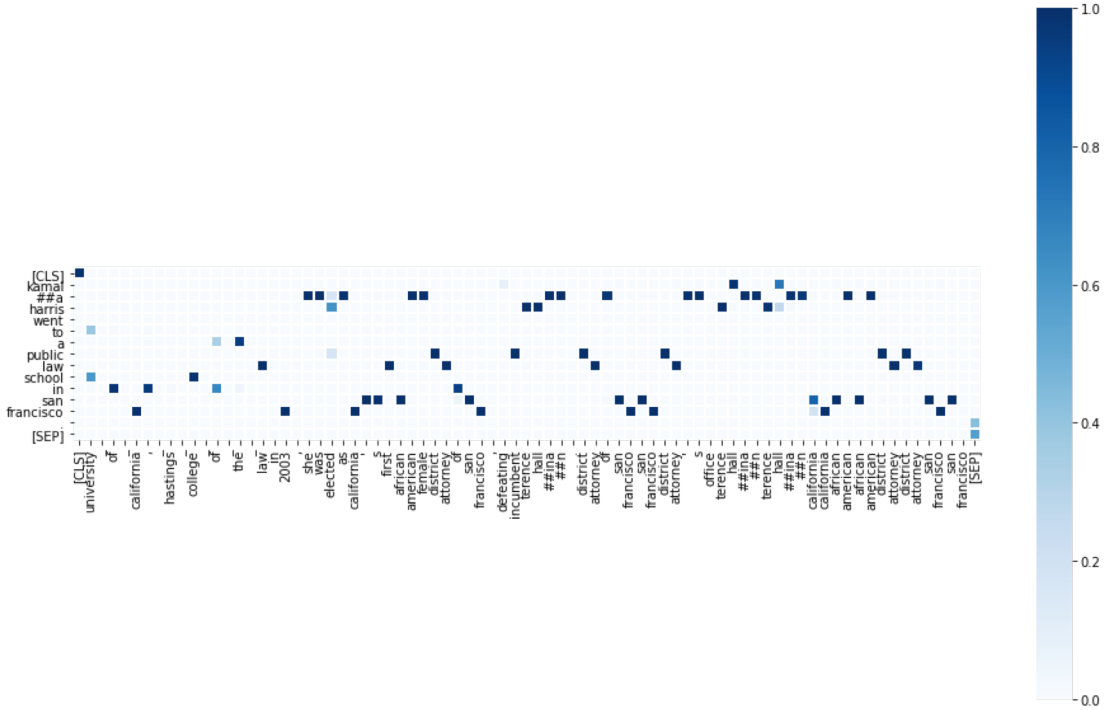


Figura 6.9: Mapa de atención obtenido del modelo. Ejemplo 3.

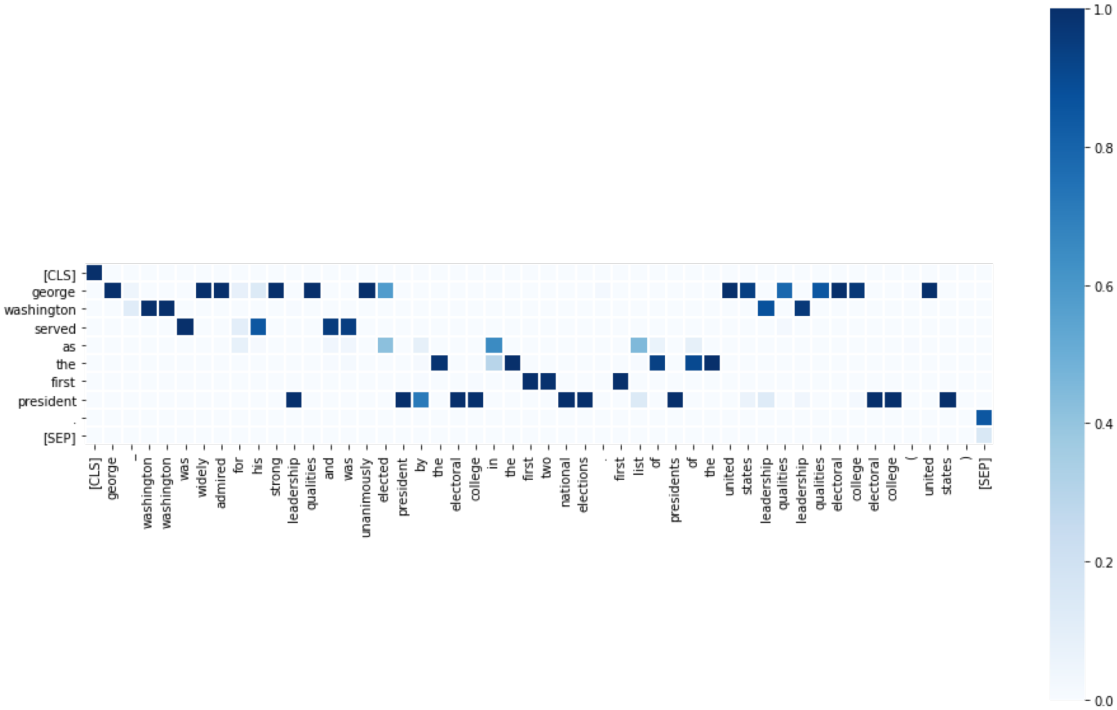


Figura 6.10: Afirmación y evidencia con los *tokens* más importantes resaltados, tomando las ponderaciones calculadas por la capa de atención. Ejemplo 1.

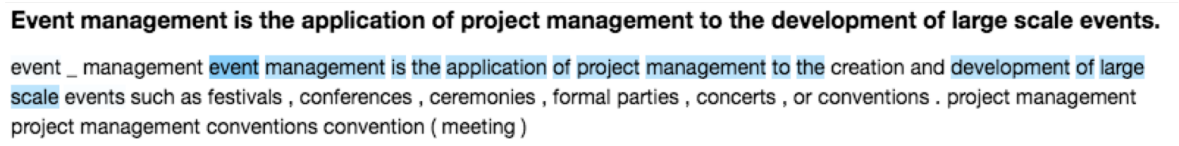


Figura 6.11: Afirmación y evidencia con los *tokens* más importantes resaltados, tomando las ponderaciones calculadas por la capa de atención. Ejemplo 2.

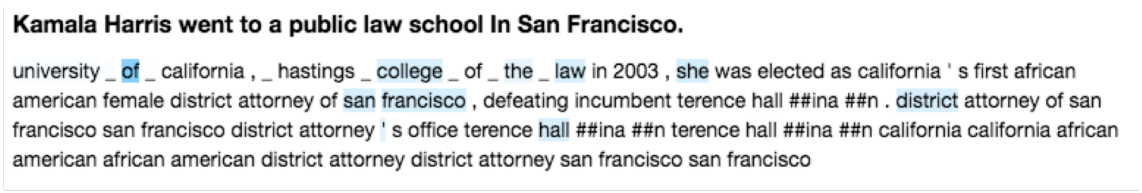
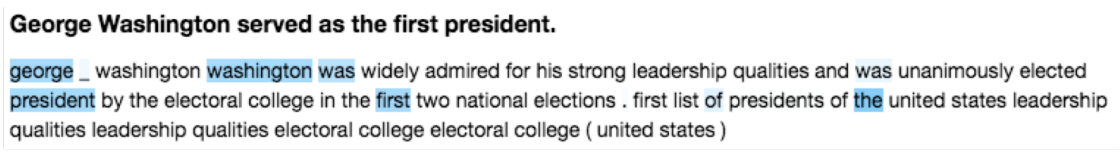


Figura 6.12: Afirmación y evidencia con los *tokens* más importantes resaltados, tomando las ponderaciones calculadas por la capa de atención. Ejemplo 3.



Capítulo 7

Caso de estudio: Verificación de información sobre COVID-19

En este capítulo se presenta un caso de estudio en el cual se aplican los modelos propuestos de evaluación de relevancia y verificación de afirmación. La intención es mostrar la utilidad de dichos modelos como herramienta de apoyo en el combate a la desinformación. Si bien las herramientas automatizadas no son una solución total a esta problemática, se busca que sean un apoyo en las tareas manuales de verificación de hechos.

Como se describió en el capítulo 3, la desinformación es un problema que no es nuevo y suele afectar a muchos ámbitos de la vida cotidiana. Con la pandemia que se vivió a partir del año 2020 a nivel mundial surgió una gran necesidad de información: conocer los peligros de esta enfermedad, los factores de riesgo, cómo protegerse, entre otros. En un inicio se conocía realmente muy poco de este tema, mucha de la información que se compartía en etapas tempranas se extrapolaba de enfermedades muy parecidas o relacionadas. Fue debido a esta gran necesidad global de información que empezaron a surgir piezas de desinformación en medios digitales. Muchas veces estas piezas surgieron no por mala intención, si no como consecuencia de que algunas personas compartieran información basada en creencias populares, que no necesariamente son apegadas a la realidad. En esta categoría caen, por ejemplo, remedios caseros para curar o prevenir la enfermedad, que la gran mayoría de las veces no tienen ningún sustento científico. En otras ocasiones la desinformación surge de fuentes noticiosas que dan una mala interpretación de algún artículo científico. Por ejemplo, algunos medios de comunicación dan por un hecho comprobado, información que apenas se está investigando, o que requiere de recolectar más datos para tener una conclusión más concreta. Estas verdades a medias permean en la sociedad y muchas veces es complicado que estas ideas falsas desaparezcan por completo, dando lugar a otros fenómenos como las teorías conspirativas.

No fue hasta que empezaron a surgir más y más estudios sobre el COVID-19, que estuvo disponible una mayor cantidad información confiable al respecto. Aunque aún está pendiente el facilitar canales de comunicación con el público en general para informar de manera oportuna y certera.

Toda la información que ha surgido de la investigación sobre el COVID-19 y temas relacionados, puede ser una buena fuente para comprobar las afirmaciones que abundan acerca de esta enfermedad. Aprovechando los modelos presentados, se propone crear un método de 3

pasos con el objetivo de realizar la verificación automática de afirmaciones sobre COVID-19. Para este propósito es necesario contar con los siguientes elementos:

- Una base de datos con documentos que pueden servir como fuente de evidencia.
- Un sistema de recuperación de información para obtener los documentos relacionados con la afirmación.
- Un modelo para extraer las oraciones relevantes de los documentos recuperados. Estas oraciones relevantes servirán como evidencia.
- Un modelo para verificar la afirmación considerando la evidencia encontrada.
- Un conjunto de datos de afirmaciones relacionadas con COVID-19 con su respectiva etiqueta de veracidad.

A continuación, se detalla cómo es que se implementó cada uno de los elementos del método de verificación de afirmaciones sobre COVID-19.

7.1. Base de datos

Como base de datos se utilizó *Covid-19 Open Research Dataset* [39] (CORD-19, por sus siglas en inglés). Este conjunto de datos surgió como una iniciativa del gobierno de Estados Unidos junto con varios grupos de investigación. Contiene más de 300 mil artículos de investigación sobre COVID-19 y temas relacionados. Tiene el objetivo de fomentar la cooperación e investigación sobre esta enfermedad, a través de hacer públicos estos datos y compartir el conocimiento existente hasta ahora. La comunidad de investigadores y entusiastas pueden explorar estos documentos para extraer y generar nuevo conocimiento. En nuestro modelo, esta base de datos es la fuente de conocimiento que servirá como verdad fundamental. En la verificación automática de hechos es muy importante considerar la confiabilidad de las fuentes de información. Debido a que CORD-19 contiene artículos científicos damos por un hecho la confiabilidad de estos documentos.

7.2. Sistema de recuperación de información

Un sistema de recuperación de información tiene el objetivo de extraer documentos de una colección de tamaño arbitrario. Estos documentos deben cumplir con una necesidad de información, expresada como una consulta. La recuperación debe ser realizada en un tiempo razonable, de tal forma que los documentos no pierdan su relevancia por la demora. En el caso que estamos presentando, se está tratando con una base de datos de artículos científicos que contiene cientos de miles de documentos. Es necesario poder obtener los documentos relevantes rápidamente. Como motor de búsqueda se utilizó *Anserini* [1] que es una herramienta de recuperación de información construida sobre Apache Lucene [2]. Lucene es una biblioteca que se ha convertido en un estándar de facto para la construcción de motores de búsqueda en aplicaciones personalizadas. Consiste de una interfaz de programación que permite indexar fácilmente una colección de documentos, y además cuenta con la interfaz para acceder rápidamente a estos documentos. Las características que han hecho tan popular a *Lucene* son

[1] `https://github.com/castorini/anserini`
[2] `https://lucene.apache.org/`

su simplicidad, rendimiento y gran capacidad de escalamiento, es capaz de buscar documentos en colecciones de millones de documentos en apenas unos cuantos milisegundos. *Anserini* agrega a estas capacidades información extra que es necesaria en la tarea de recuperación de información.

Nosotros utilizamos *Anserini* para realizar un filtrado previo de documentos que pueden contener información relevante relacionada con la afirmación. Este filtrado es realizado en los miles de documentos contenidos en CORD-19. Como primer paso para utilizar esta herramienta es necesario construir un índice sobre la colección. En nuestro modelo se utilizó un índice preconstruido[3] que implementa un indexado a nivel párrafo. Consideramos que este nivel de indexado otorga una granularidad adecuada para la búsqueda que se pretende realizar, considerando que los párrafos tienen información muy relacionada, y no son tan dispersos como un artículo completo. Es una forma de limitar las búsquedas que se realizarán en etapas posteriores. En nuestro modelo, la recuperación de información entregará 100 documentos relacionados con la afirmación.

7.3. Evaluación de relevancia

Para esta etapa se utilizó el modelo presentado en el capítulo 4 que fue entrenado sobre *FEVER*. Dicho modelo recibirá los 100 documentos (párrafos en nuestro caso) que se consideraron relevantes con respecto a la afirmación. Un proceso posterior se encarga de tokenizar las oraciones de los documentos. Cada una de estas oraciones es entonces evaluada contra la afirmación. Por cada una de las oraciones se obtendrá un *score* entre 0 y 1 que indicará la relevancia. Finalmente se ordenan las oraciones por relevancia en orden descendente, y se seleccionarán las 10 oraciones con mayor *score*. Este proceso de evaluación es bastante más preciso que la recuperación de información realizada en el paso anterior, pero es considerablemente más demandante de recursos computacionales. Por esta razón es importante realizar el filtrado previo con una herramienta de recuperación de información altamente eficiente.

7.4. Conjunto de datos de afirmaciones sobre COVID-19

Para entrenar el modelo de verificación es necesario contar con un conjunto de afirmaciones etiquetadas que permitan clasificar una pareja de afirmación-evidencia como verdadera o falsa. Con este objetivo, se construyó un conjunto de datos con parejas afirmación-evidencia y junto con cada una de ellas, su respectiva etiqueta que indica la veracidad. El conjunto de afirmaciones y evidencia se encuentra en idioma inglés.

Las afirmaciones se extrajeron de algunos sitios que se especializan en combatir la desinformación relacionada con la COVID-19. Las fuentes de las afirmaciones son las siguientes:

- Cazadores de mitos de la OMS [4].
- Mitos contra hechos, Universidad Johns Hopkins [5].

[3] `https://github.com/castorini/anserini/blob/master/docs/experiments-cord19.md#pre-built-indexes-all-versions`

[4] `https://www.who.int/emergencies/diseases/novel-coronavirus-2019/advice-for-public/myth-busters`

[5] `https://www.hopkinsmedicine.org/health/conditions-and-diseases/coronavirus/2019-novel-coronavirus-myth-versus-fact`

- Preguntas y respuestas sobre el coronavirus. CNN [6].

En total se recolectaron 112 afirmaciones diferentes. Posteriormente, utilizando el motor de búsqueda indexado sobre CORD-19, se extrajeron documentos basados en cada una de las afirmaciones. Para cada afirmación se tiene un conjunto de documento potencialmente con evidencia. Entonces un conjunto de anotadores se encargó de decidir si en los documentos recuperados hay información relevante, si no la hay el documento se descarta. Con los documentos que sí tienen información relevante se construyeron parejas afirmación-evidencia y el anotador decidió si la afirmación es verdadera o falsa, anotando la pareja acorde a esto. Como resultado de este proceso se obtuvieron 759 parejas. En la Figura 7.1 se muestran algunos ejemplos de estas parejas. En la Tabla 7.1 se muestra el número promedio de evidencias por afirmación, el máximo y el mínimo.

Tabla 7.1: Número promedio de evidencias por afirmación, el mínimo y el máximo.

	evidencia
min	1
max	10
promedio	6.77

Figura 7.1: Ejemplo de parejas pertenecientes al conjunto de afirmaciones sobre COVID-19.

Afirmación	Evidencia	Etiqueta
covid can not be cured	Excellent therapeutic effect: Among 101 hospitalized patients with confirmed COVID 19 all were cured and discharged except for one death. No secondary hospital infection no pipeline infection and no pressure sore were found in all patients.	**Falso**
best way to protect against COVID is frequently cleaning hands	The CDC recommends multiple steps to prevent the transmission and risk of SARS CoV 2. Frequent hand washing lasting at least 20 seconds by using soap and water is advised. Hand sanitizers with at least 60% alcohol can also be used as an alternative.	**Verdadero**

Finalmente, para acrecentar los datos, las afirmaciones originales se negaron modificando el texto original para cambiar su sentido. Como consecuencia las etiquetas de las negaciones también se modificaron para adecuarse al nuevo sentido de la afirmación. Con esta estrategia, se duplicó el número de parejas (Tabla 7.2). La distribución de las etiquetas es completamente balanceada.

7.5. Verificación de afirmación

El modelo enfocado en esta etapa se basa en la arquitectura propuesta en este trabajo. Pero para enfocarse en el dominio de los artículos sobre COVID-19 se entrenó la red neuronal

[6] `https://edition.cnn.com/interactive/2020/health/coronavirus-questions-answers/`

Tabla 7.2: Número de afirmaciones y evidencia recolectada para los conjuntos de datos.

Dataset	Afirmaciones	Evidencia
Original	112	759
Original + negaciones	224	1518

utilizando un conjunto de afirmaciones sobre este tema. Dichas afirmaciones se encuentran acompañadas de evidencia y una etiqueta que indica si la afirmación es verdadera o no. El modelo se diferencia con el implementado para *FEVER* en que la clasificación es binaria, no se está considerando la clase sin información.

7.5.1. Entrenamiento

El modelo de verificación se entrenó usando el conjunto de afirmaciones de COVID-19. Como base, se reutilizó el modelo preentrenado en *FEVER*, sólo se ajustó al nuevo dominio. Se evaluó el modelo utilizando validación cruzada de 5 particiones, entrenando por una época, resultando en una exactitud de 0.9179. También se probó con el modelo de [35] y de [27], comparando entrenar los modelos solamente en nuestro *dataset* y entrenándolos primero en *FEVER* y posteriormente en el nuestro. En todos los casos se mostró una mejoría pequeña derivado de entrenar el modelo sobre *FEVER* y después sobre el el *dataset* de afirmaciones sobre COVID-19. Las exactitudes obtenidas de esta validación se pueden observar en la Tabla 7.3.

Tabla 7.3: Validación cruzada de 5 particiones de nuestro modelo, el estado del arte y el *baseline* sobre el *dataset* de afirmaciones sobre COVID-19. Se muestra la exactitud resultante.

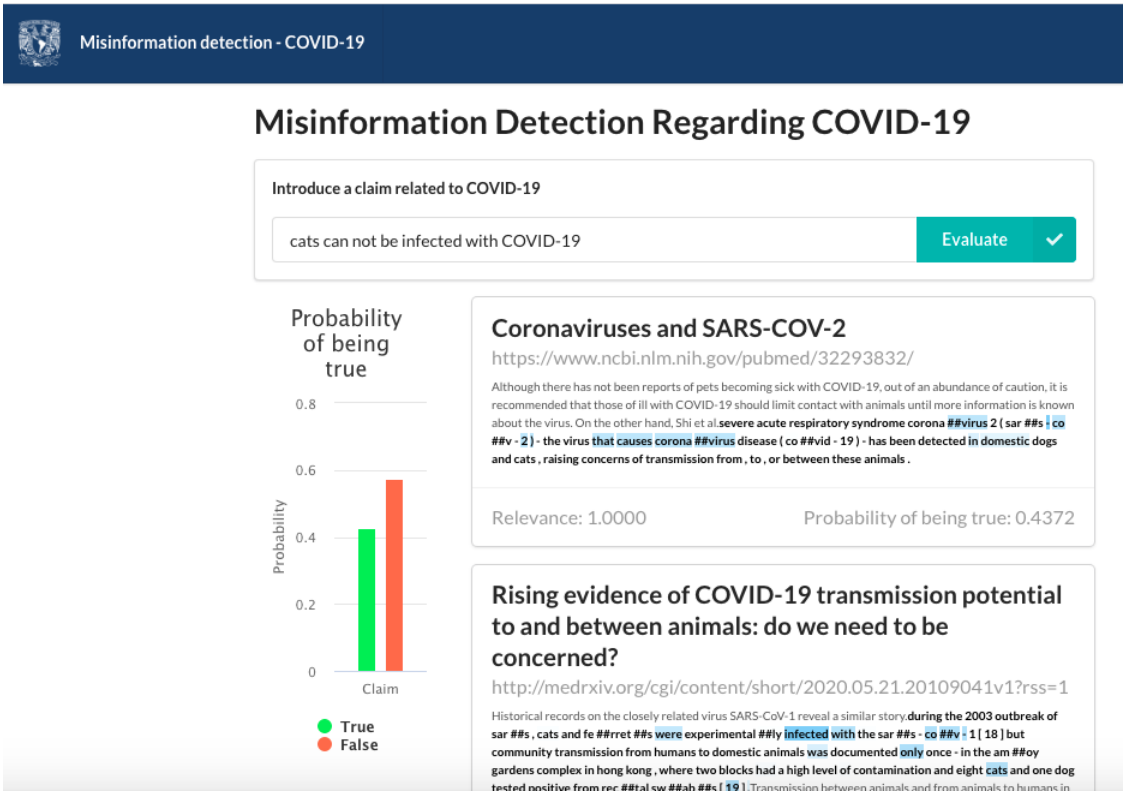
Modelo	Exactitud
Nuestro modelo	0.8920
Nuestro modelo + FEVER	0.9179
Soleimani [35]	0.9230
Soleimani [35] + FEVER	0.9324
Baseline [27]	0.8490
Baseline [27] + FEVER	0.8500

7.6. Aplicación completa

Agregando todos los elementos del método de 3 pasos, se construyó una interfaz web (Figura 7.2) para facilitar la interacción con el modelo de verificación de afirmaciones sobre el COVID-19. En esta interfaz se presenta un campo de texto en el cual se introduce la afirmación en inglés. Al invocar la evaluación de la afirmación se ejecutan los siguientes pasos secuenciales:

- Se extraen 100 párrafos del CORD-19.
- Se evalúan cada una de las oraciones provenientes de los 100 párrafos. Se obtienen las 10 oraciones con mayor *score* de relevancia.
- Se verifica la afirmación contra la evidencia.

Figura 7.2: Interfaz web para interactuar con el modelo de verificación de afirmaciones sobre COVID-19.



- Por cada evidencia se obtiene un *score* de veracidad entre 0 y 1. Para calcular el *score* final se obtiene una suma pesada de los *scores* de veracidad considerando la relevancia calculada para cada evidencia.

- Se presenta el *score* de veracidad estimado (Figuras 7.4 y 7.6).

- Adicionalmente se muestra el texto que se consideró como evidencia. Utilizando el mecanismo de atención del modelo de relevancia, se presenta la oración relevante, así como las palabras de ella que se tomaron en cuenta para considerarla como tal (Figuras 7.3 y 7.3).

La evidencia presentada como resultado de la verificación de la afirmación permite comprender por qué se clasificó la afirmación como verdadera o falsa. Además, permite informar de los hechos encontrados dentro de artículos relacionados con el COVID-19, lo cual tiene el potencial de ser una herramienta para informar de forma más precisa acerca de las inquietudes existentes acerca de la pandemia. En las Figuras 7.3 y 7.5 se presentan 2 ejemplos de afirmaciones relacionadas con COVID-19. En la parte inferior de cada una de estas Figuras se muestra evidencia que fue posible recuperar, además de que se resaltan las partes de la evidencia que son más relevantes para determinar si la afirmación es verdadera o no. En las Figuras 7.4 y 7.6 se muestra el *score* final de veracidad calculado para cada una de las afirmaciones.

7.7. Resumen

Se presentó cómo es que se puede aprovechar el modelo que se está proponiendo en este trabajo. El objetivo fue crear una aplicación que permita evaluar si una afirmación relacionada con la enfermedad COVID-19 es verdadera o falsa. Para dicho fin, se utilizó el método de tres pasos presentado previamente. Utilizando un conjunto de artículos científicos fue posible obtener documentos que pueden ser considerados a priori como verdades fundamentales.

El modelo de evaluación de relevancia construido permitió obtener las oraciones más relacionadas con la afirmación. Para la etapa de verificación de afirmación, se creó un conjunto de datos de afirmaciones sobre la enfermedad. El conjunto tiene la finalidad de ajustar el modelo de verificación, previamente entrenado en *FEVER*, para mejorar su rendimiento en el dominio en el cual se está trabajando.

Finalmente se mostró la interfaz web que se implementó buscando hacer más amigable la interacción con el modelo de verificación de afirmaciones. Esto convierte a la aplicación en su conjunto, en una herramienta muy útil que permite conocer la veracidad de la información que circula diariamente en medios digitales. Y no sólo eso, si no que además proporciona explicaciones del por qué la afirmación fue considerada como tal. Esta información extra es sin duda un medio informativo que permite conocer mejor al problema que se enfrenta durante la pandemia. Adicionalmente el uso de las ponderaciones calculadas en la capa de atención del modelo permite enriquecer el texto que se presenta como evidencia del *score* de veracidad calculado.

Figura 7.3: Ejemplo 1 de afirmación sobre COVID-19 *cats can not be infected with COVID-19*. En la parte inferior se muestra evidencia relevante para la verificación de la afirmación.

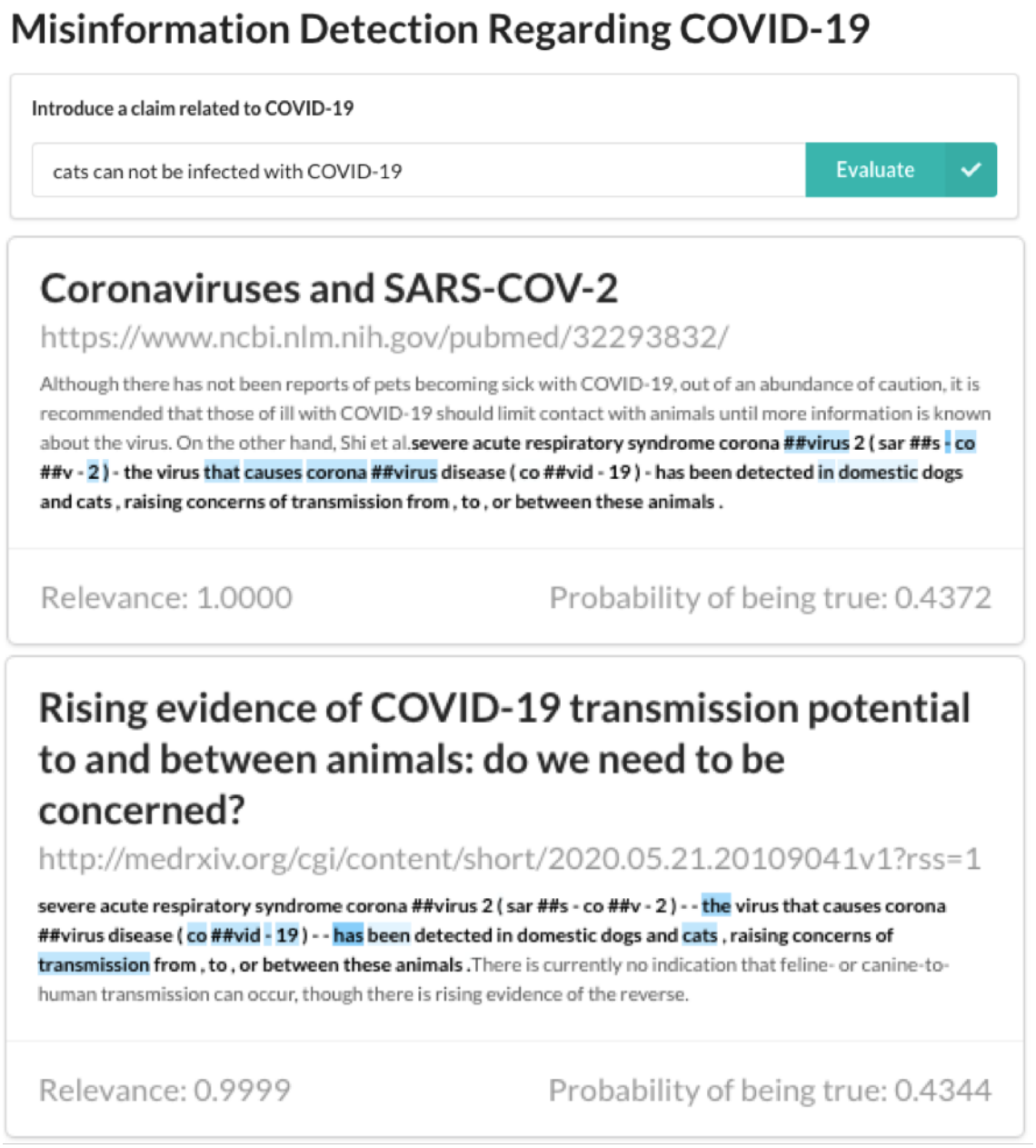


Figura 7.4: *Score* final de veracidad para la afirmación del ejemplo 1. En este caso el modelo considera que la afirmación tiene más probabilidad de ser falsa.

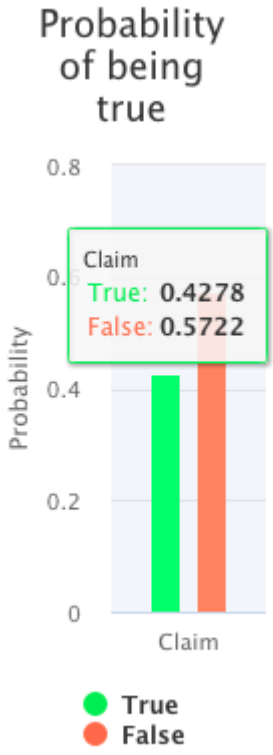


Figura 7.5: Ejemplo 2 de afirmación sobre COVID-19: *covid-19 virus is affected by high temperatures*. En la parte inferior se muestra evidencia relevante para la verificación de la afirmación.

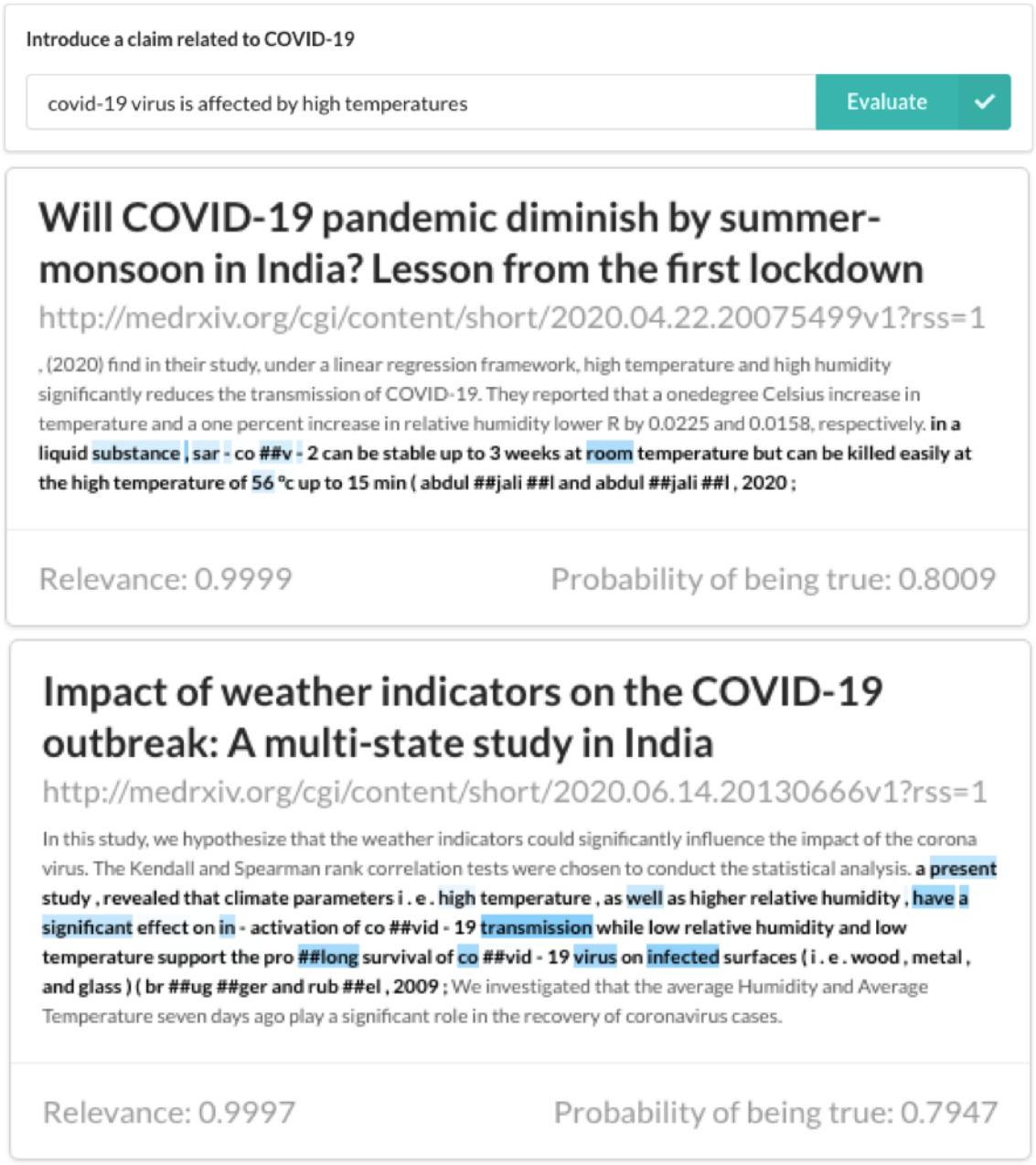
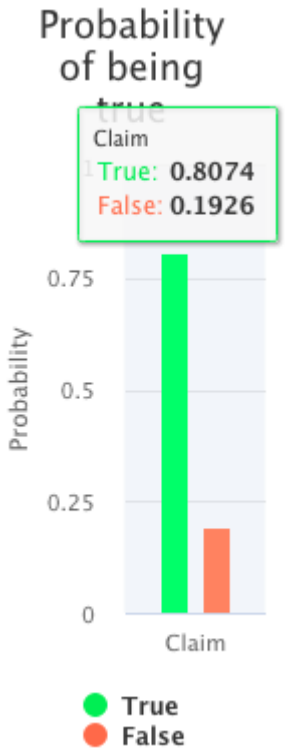


Figura 7.6: *Score* final de veracidad para la afirmación del ejemplo 2. En este caso el modelo considera que la afirmación tiene más probabilidad de ser verdadera.



Capítulo 8

Conclusiones y trabajo futuro

En capítulos anteriores se discutió el problema de la información falsa. Este tipo de información es creada indistintamente con o sin intención, pero de igual forma puede ser muy perjudicial, es capaz de manipular las creencias de la gente y su toma de decisiones. Esto puede tener sin duda graves consecuencias en varios contextos, por ejemplo, el económico y el político [34]. Y es justamente en estos dominios, en donde la información falsa es utilizada por grupos que intentan obtener un beneficio. Las piezas informativas falsas pueden venir en muchas formas, imágenes, videos o texto. En nuestro caso, nos concentramos en la detección de este fenómeno en datos textuales. Como se ha discutido ya, este problema es muy complicado de solucionar, hoy en día no existe una solución contundente a esta problemática. Algunos esfuerzos se han realizado para tratar de verificar afirmaciones que se hacen en discursos políticos y en artículos noticiosos.

Nosotros utilizamos un enfoque basado en conocimiento, que requiere de una base de datos con información confiable que se considera a priori como una verdad fundamental. En este trabajo se proponen un par de modelos que tratan con la evaluación de la relevancia y la verificación de la afirmación. En este par de tareas, el estado del arte en años recientes fue tomado por modelos basados en la arquitectura *transformer*, en particular los modelos preentrenados de lenguaje como *BERT* [8]. Como muchas otras tareas de procesamiento de lenguaje, este tipo de modelos preentrenados han mejorado el desempeño del estado del arte, superando incluso en muchas tareas al ser humano. Es incuestionable los beneficios de utilizar aprendizaje por transferencia, reducen el tiempo de entrenamiento, ya que se aprovecha el conocimiento lingüístico extraído durante el preentrenamiento, y ayudan a mejorar el rendimiento en dominios donde existen muy pocos datos. Ya algunos trabajos, como [35], han explorado la aplicación de estos modelos para la verificación automática de hechos. Nosotros también utilizamos un modelo preentrenado de lenguaje para este fin, pero le damos un enfoque distinto al ajuste fino realizado por otros trabajos, buscando obtener un modelo interpretable.

8.1. Conclusiones finales

¿Qué ventajas tiene utilizar una red siamesa en la verificación de hechos, comparado con el actual estado del arte?

El estado del arte se encuentra en utilizar modelos preentrenados de lenguaje para ajustarlos a la verificación de hechos. Esta tarea consiste, en pocas palabras, en una

clasificación de un par de oraciones. Regularmente lo que se hace en estos casos es hacer un ajuste fino del modelo preentrenado agregando una capa de clasificación y entrenando por unas pocas épocas. Esta estrategia es bastante efectiva, los modelos logran clasificar correctamente una buena proporción de las afirmaciones presentadas. Donde tienen una carencia importante estos modelos es en el momento de buscar una interpretación a las decisiones resultantes. Las arquitecturas basadas en *transformer* han encontrado la forma de mejorar su desempeño a través de aumentar el número de capas y de cabezas de atención, además de utilizar conjuntos de datos mucho más grandes que sus predecesores. Como consecuencia, los modelos más recientes ya superan los miles de millones de parámetros.

Un elemento de las arquitecturas *transformer* que puede ser utilizado para tratar de comprender qué es lo que está ocurriendo dentro de la red neuronal, son las cabezas de atención. Recordando, estas cabezas son mecanismos de atención paralelos e independientes que se encuentran además distribuidos en todas las capas. Las cabezas aprenden aspectos distintos del lenguaje según en la capa que se encuentren, pero incluso se enfocan en detalles diferentes aún perteneciendo a la misma capa. Entonces, las cabezas de atención sirven para dar una interpretación al modelo, pero debido a su número y a las diferencias en lo que están aprendiendo, no es tan simple extraer ese conocimiento de estos elementos.

Una ventaja que tiene nuestra propuesta es precisamente el simplificar la interpretabilidad del modelo, esto se logra a través de una arquitectura de red siamesa. En cada una de las ramas de la red se codifican de forma independiente las dos entradas, la afirmación y la evidencia. Después de esta codificación se utiliza una capa de atención que tiene el objetivo de calcular la importancia que tiene cada una de las palabras de la evidencia, con respecto a la afirmación. Un aspecto relevante de esta capa de atención es que el modelo se puede enfocar en las partes de la entrada que tienen más importancia, y omitir o reducir el enfoque en las partes menos importantes. Y esta es justamente la ventaja de los mecanismos de atención, mejorar el rendimiento de las redes neuronales a través de concentrarse en los segmentos importantes de los datos. Utilizar esta capa en nuestro modelo ayudó a mejorar sustancialmente el desempeño. Adicionalmente, la atención también puede ser utilizada para saber en que se está enfocando la red, y consecuentemente estas ponderaciones se pueden utilizar para dar una interpretación al modelo. Como nuestra arquitectura tiene sólo una capa de atención, la interpretación es mucho más sencilla de extraer. En el caso de estudio se demostró la importancia de otorgar una explicación a la tarea de verificación de hechos, da información extra acerca de la evaluación hecha sobre la afirmación: se muestra la evidencia que se consideró para evaluar la veracidad y además se resaltan las partes más relevantes dentro del texto de la evidencia. Esto sin duda es de gran utilidad como herramienta de apoyo para combatir la desinformación.

¿Cuál es la mejor estrategia de entrenamiento para ajustar un modelo preentrenado de lenguaje, a la tarea de verificación de hechos?

Ajustar un modelo preentrenado de lenguaje a una tarea particular (en este caso la verificación automática de hechos) es una tarea relativamente sencilla. Se agregan capas al inicio o al final del modelo dependiendo del objetivo que se tenga y se entrenan por unas cuantas épocas. Al ajustar un modelo de *BERT* lo que recomienda la documentación es utilizar una tasa de aprendizaje muy baja, regularmente el valor que se usa es 2e-5. Esto es necesario ya que si no de otra forma se corre el riesgo de perder lo aprendido en el preentrenamiento y prácticamente se estaría entrenado de cero el bloque *transformer*. Esto

es particularmente relevante cuando se está trabajando con el modelo grande de *BERT*. Otra recomendación que se hace es el tamaño del lote, el cual no debe ser muy pequeño, se recomiendan tamaños superiores a 32. La recomendación surge del hecho de que si se usa un tamaño de lote muy pequeño se corre el mismo riesgo de destruir los pesos preentrenados, lo cual se ve reflejado en una caída importante en el desempeño.

Las arquitecturas *transformer* suelen requerir mucha memoria en *GPU* para el entrenamiento, y utilizar tamaños de lote muy grandes empeora la situación. A esto se le puede sumar que también el tamaño de la secuencia aumenta considerablemente el uso de este recurso. En nuestro caso nos limitamos a utilizar 128 tokens para economizar el uso de memoria, y además este valor es adecuado considerando que ni la longitud de las afirmaciones ni de las evidencias lo superan en promedio. Ante limitaciones en el espacio en memoria, aún se pueden utilizar tamaños de lote más grandes, se pueden utilizar estrategias como la acumulación de gradiente, la cual busca obtener el gradiente promedio de varios pasos de entrenamiento, para después realizar la retro-propagación con estos valores. Esto es equivalente a utilizar un tamaño de lote más grande, pero sin la necesidad de tanta memoria.

Para ajustar nuestro modelo, utilizamos una técnica de entrenamiento mixta. Primero se congela el bloque *BERT*, y se entrenan las capas adicionales utilizando una tasa de aprendizaje de 1e-3. Se entrena el modelo por un par de épocas, llevarlo por más de 3 épocas ya no lo hacía mejorar. Posteriormente, se reduce la tasa a 1e-4 y se entrena por un par de épocas más. Lo que se busca con este ajuste previo del modelo, es reducir la magnitud del error provocado por las capas adicionales del modelo. Primero se ajustan de forma importante estas capas, y posteriormente se hace un ajuste más fino de estas mismas capas. En este punto el error retro-propagado es mucho menor que al inicio. Continuar entrenando con *BERT* congelado ya no significa una mejoría. Por este motivo, en la tercera etapa del entrenamiento se descongela *BERT* buscando hacer un ajuste todavía más fino del modelo. Aquí se empiezan a tocar los pesos del modelo de lenguaje para ajustarse a la tarea de verificación de información, pero ese ajuste es mucho más cuidadoso. Esta estrategia demostró ser más eficiente que otras, como el mantener congelado *BERT* siempre o utilizar desde el comienzo una tasa baja. Con este entrenamiento mejoró el rendimiento de los modelos.

¿Cuál es el mecanismo de atención más adecuado para evaluar la relevancia de una evidencia?

En un mecanismo de atención se busca realizar una ponderación en los datos de entrada, detectando la porción de ellos que es más relevante para llevar a cabo su tarea. En el caso de nuestra arquitectura, se realiza una ponderación entre los elementos de las dos secuencias de entrada, se busca encontrar los elementos que deben pesar más en la verificación de la afirmación. Existen varios métodos para implementar la capa de atención, se experimentó con algunos de ellos.

El que mejor resultado dio fue la atención multiplicativa, este mecanismo calcula la similitud entre un par de vectores utilizando el producto punto. Esta función tiene la capacidad de medir qué tan parecidos son dos vectores, y es especialmente útil en representaciones vectoriales binarias. Para tareas de similitud semántica, la función que se usa más frecuentemente es el coseno, que básicamente es un producto punto normalizado. Es efectivo por que no depende de la magnitud de los vectores, sino que se concentra en evaluar las

diferencias entre las direcciones de estos.

El producto punto entrega un valor que representa la similitud de los vectores, pero tiene un rango de valores bastante amplio. Para controlar esto, y que sea una ponderación más certera, se agregó una función *softmax* en la dimensión que corresponde a la secuencia de la evidencia. De esta manera, los valores que se obtienen modelan una distribución de probabilidad y los pesos obtenidos pueden ser utilizados para ponderar los vectores de ambas secuencias.

A pesar de la popularidad de la similitud coseno en tareas de similitud semántica, para la capa de atención del modelo de verificación de hechos, la atención multiplicativa aplicándole una función *softmax* dio mejores resultados durante la experimentación.

¿Qué estrategia para resumir una secuencia de palabras, representadas como vectores, es la más adecuada para ajustarse a la tarea de interés?

Después de la capa de atención, se obtienen un par de secuencias de vectores. Para poder clasificar estas secuencias se necesita un mecanismo para obtener una representación significativa de estas secuencias. En la literatura algunos usan promedio, otros el valor máximo o mínimo de cada dimensión. También es posible generar una representación de la secuencia utilizando redes recurrentes, la secuencia se procesa elemento a elemento y al final el último estado oculto se puede considerar como una representación o resumen de la secuencia entera. Este tipo de redes tienen sus limitantes, que se discutieron en el capítulo 2, pero vale la pena explorar su utilidad en esta tarea.

En la tarea de evaluación de relevancia, el promedio entregó buenos resultados, pero utilizar una red *LSTM* tuvo un desempeño superior. Con la *LSTM* fue posible explorar con diferentes hiperparámetros como el número de capas. Utilizar 5 capas fue la configuración óptima. Esta representación de la secuencia demostró ser superior a utilizar simplemente el promedio, aunque tiene el inconveniente que la red *LSTM* agrega más parámetros al modelo y por lo tanto aumenta su complejidad y requerimientos computacionales tanto en entrenamiento como en inferencia.

¿Cómo aprovechar una capa de atención, en una red siamesa para obtener una interpretación de la clasificación hecha por la red neuronal?

La capa de atención implementada en la red siamesa aprovecha las representaciones vectoriales creadas por *BERT* para computar la relevancia de los elementos de la secuencia. Esta capa entrega una matriz de ponderaciones de tamaño 128×128. Por cada token en la afirmación hay un vector con 128 pesos, cada uno de los pesos corresponde a la importancia de cada token en la evidencia. Entonces la matriz es una tabla de ponderaciones de todos contra todos. Los pesos de cada token de la afirmación suman 1 siempre, por la función *softmax* que se utiliza. Entonces conviene enfocarse en la otra dimensión para obtener la importancia de cada token de la evidencia. Para aprovechar estos pesos y dar una interpretación al modelo, lo que se hizo fue obtener un promedio de los pesos que tiene cada token de la evidencia. Como resultado se obtiene un vector con pesos en el rango de 0-1. El modelo entrega este vector de pesos como salida, además de la clasificación que corresponda en cada caso (relevancia o verificación). En el caso de estudio se utilizó este vector de pesos de la evidencia para mostrar en la

salida, los elementos que la red neuronal consideró son más importantes y así dar información extra.

8.2. Contribuciones del trabajo

En este trabajo se propuso una arquitectura novedosa, basada en un modelo preentrenado del lenguaje, que se enfoca en la segunda y tercera etapa del método de verificación de hechos. Esta arquitectura obtiene resultados competitivos con respecto al estado del arte, no lo alcanza a mejorar, pero es bastante cercano. Una ventaja que tiene esta arquitectura sobre el estado del arte es que las clasificaciones en la tarea de verificación de afirmación están mucho menos sesgadas hacia la clase apoyada, en el conjunto de datos *FEVER*. Tiene un desempeño similar, pero está mucho más balanceado en la distribución de sus predicciones.

Experimentamos con diferentes estrategias para hacer ajuste fino del modelo de *BERT*. Se propuso un método de decaimiento en la tasa de aprendizaje, congelando inicialmente el modelo preentrenado y luego descongelándolo. Esto demostró dar mejores resultados que hacer un ajuste fino simple.

Otra contribución es la información extra que se puede obtener del modelo de verificación de hechos, fijándose en los pesos entregados por la matriz de ponderaciones de la capa de atención. En el caso de estudio fue posible darse cuenta de la diferencia que hay entre sólo entregar una clase (verdadero-falso) a entregar un *ranking* de veracidad y además presentar la evidencia encontrada, resaltando las porciones de la evidencia que resultaron ser relevantes para evaluar la afirmación.

En el área de procesamiento de lenguaje, es muy importante contar con conjuntos de datos que permitan experimentar con diversos modelos y configuraciones, buscando los mejores resultados para varias tareas de este campo de conocimiento. El conjunto de datos construido con afirmaciones sobre la COVID-19 precisamente puede ser de utilidad para trabajos futuros en la tarea de verificación de hechos o temas relacionados, además de enfocarse en un tema que sin discusión es de mucha relevancia hoy en día.

8.3. Trabajo futuro

Un área de oportunidad para explorar mejoras a los modelos propuestos es tratar de comprender mejor que es lo que está ocurriendo en cada una de las capas de los modelos preentrenados, así también como en las cabezas de atención. Ya hay investigación al respecto [6], pero aún falta camino por recorrer para realmente obtener información útil de los mecanismos de atención de estas arquitecturas. Existen hipótesis acerca de qué es lo que aprende cada cabeza en las distintas capas. Pero como ya se ha referido, la complejidad de la interpretación de estas capas crece de forma directamente proporcional conforme se van agregando más capas y más cabezas. La tendencia actual es mejorar el rendimiento de los modelos de lenguaje sumando parámetros y usando más datos de preentrenamiento. No hay duda de que esta tendencia ha funcionado, aunque la magnitud de las mejoras ya no corresponde con la cantidad de datos y el cómputo extra que se necesita. Comprender que es lo que están haciendo estos modelos no es sencillo.

Los modelos de verificación de hechos basados en el conocimiento necesitan, como es de esperarse, contar con una base de datos que servirá para obtener el conocimiento contra el que se verificarán las afirmaciones. Para que esta estrategia funcione se debe tener la certeza de que la información en dicha base de datos sea confiable y esté apegada a la realidad. Hablar de una verdad es complicado, es un término que puede desencadenar discusiones filosóficas al respecto. Por lo tanto, este tipo de métodos de verificación dependen mucho de la calidad de los datos con los que se cuenta. En realidad, lo que se está haciendo es aproximar la evaluación de la veracidad utilizando inferencia del lenguaje natural (*NLI* o *RTE*, por sus siglas en inglés). Así sólo se está verificando si la afirmación es apoyada o refutada por la evidencia. Para considerar la veracidad de la afirmación es necesario también evaluar la veracidad de la evidencia. Hoy en día existe investigación enfocada en esta labor, ser capaces de evaluar que tan confiable es una fuente de información y de esta manera utilizar los documentos de esta fuente como una verdad fundamental. En el caso de estudio, se asumió que los hechos presentados en la base de datos CORD-19 son verdaderos ya que se trabaja con artículos científicos. Pero la realidad, es que incluso en los artículos científicos es posible encontrar información contradictoria. Un aspecto que el modelo podría tomar en cuenta para tener predicciones más acertadas es considerar también la fecha de publicación, para dar más peso a los artículos más recientes en caso de encontrar una contradicción.

La verificación de hechos no sólo se da en el dominio científico, también es necesario en otras áreas. En áreas diferentes a las ciencias no es tan sencillo encontrar fuentes confiables de información, como podrían ser considerados los artículos científicos. Se requiere trabajar en estrategias para evaluar eficazmente la confiabilidad de una fuente o generar nuevas fuentes de información confiable.

Bibliografía

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *ArXiv*, 1409, 09 2014.

[2] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. *Neural Probabilistic Language Models*, volume 3, pages 137–186. Springer, 05 2006.

[3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, and D. Amodei. Language models are few-shot learners. -, 05 2020.

[4] Q. Chen, X. Zhu, Z.-H. Ling, S. Wei, H. Jiang, and D. Inkpen. Enhanced lstm for natural language inference. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1657–1668, 01 2017.

[5] A. Chernyavskiy and D. Ilvovsky. Extract and aggregate: A novel domain-independent approach to factual data verification. In *Proceedings of the Second Workshop on Fact Extraction and VERification (FEVER)*, pages 69–78, 01 2019.

[6] K. Clark, U. Khandelwal, O. Levy, and C. Manning. What does bert look at? an analysis of bert's attention. In *Proceedings of the Second BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pages 276–286, 06 2019.

[7] R. Cooper, R. Crouch, J. van Eijck, C. Fox, J. Genabith, J. Jaspars, H. Kamp, D. Milward, M. Pinkal, M. Poesio, S. Pulman, T. Briscoe, H. Maier, and K. Konrad. Using the framework. -, 03 1996.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 10 2018.

[9] X. Dong, E. Gabrilovich, K. Murphy, V. Dang, W. Horn, C. Lugaresi, S. Sun, and W. Zhang. Knowledge-based trust: Estimating the trustworthiness of web sources. *Proceedings of the VLDB Endowment*, 8, 02 2015.

[10] X. Feifei, Z. Shuting, and T. Yu. Bert-based siamese network for semantic similarity. *Journal of Physics: Conference Series*, 1684:012074, 11 2020.

[11] J. Firth. A synopsis of linguistic theory 1930-1955, 05 1957.

[12] R. Gonzalez and R. Woods. *Digital Image Processing (2nd Edition)*. Addison-Wesley Pub, 01 2002.

[13] J. Goodman. A bit of progress in language modeling. *Computer Speech & Language*, 15:403–434, 10 2001.

[14] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines, 10 2014.

[15] A. Hanselowski, H. Zhang, Z. Li, D. Sorokin, B. Schiller, C. Schulz, and I. Gurevych. Ukp-athene: Multi-sentence textual entailment for claim verification, 09 2018.

[16] N. Hassan, B. Adair, J. Hamilton, C. Li, M. Tremayne, J. Yang, and C. Yu. The quest to automate fact-checking. *Proceedings of the 2015 Computation + Journalism Symposium*, 10 2015.

[17] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[18] S. Kwon, M. Cha, K. Jung, W. Chen, and Y. Wang. Prominent features of rumor propagation in online social media. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 1103–1108, 12 2013.

[19] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach, 07 2019.

[20] M.-T. Luong, H. Pham, and C. Manning. Effective approaches to attention-based neural machine translation, 08 2015.

[21] J. Ma, W. Gao, Z. Wei, Y. Lu, and K.-F. Wong. Detect rumors using time series of social context information on microblogging websites. In *CIKM '15: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 10 2015.

[22] C. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*, volume 13. Cambridge University Press, 01 2008.

[23] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, 2013, 01 2013.

[24] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur. Recurrent neural network based language model. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association, INTERSPEECH 2010*, volume 2, pages 1045–1048, 01 2010.

[25] M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. pages 2227–2237, 01 2018.

[26] A. Poliak. A survey on recognizing textual entailment as an nlp evaluation, 10 2020.

[27] K. Popat, S. Mukherjee, A. Yates, and G. Weikum. Declare: Debunking fake news and false claims using evidence-aware deep learning, 09 2018.

[28] M. Potthast, J. Kiesel, K. Reinartz, J. Bevendorff, and B. Stein. A stylometric inquiry into hyperpartisan and fake news, 02 2017.

[29] W. Quattrociocchi, A. Scala, and C. Sunstein. Echo chambers on facebook. *SSRN Electronic Journal*, 01 2016.

[30] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners, 02 2019.

[31] H. Rashkin, E. Choi, J. Jang, S. Volkova, and Y. Choi. Truth of varying shades: Analyzing language in fake news and political fact-checking. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2931–2937, 01 2017.

[32] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[33] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 10 2019.

[34] K. Shu, A. Sliva, S. Wang, J. Tang, and H. Liu. Fake news detection on social media: A data mining perspective. *ACM SIGKDD Explorations Newsletter*, 19, 08 2017.

[35] A. Soleimani, C. Monz, and M. Worring. *BERT for Evidence Retrieval and Claim Verification*, pages 359–366. ECIR, 04 2020.

[36] I. Sutskever, O. Vinyals, and Q. Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 4, 09 2014.

[37] J. Thorne, A. Vlachos, C. Christodoulopoulos, and A. Mittal. Fever: a large-scale dataset for fact extraction and verification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 04 2018.

[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 06 2017.

[39] L. Wang, K. Lo, Y. Chandrasekhar, R. Reas, J. Yang, D. Eide, K. Funk, R. Kinney, Z. Liu, W. Merrill, P. Mooney, D. Murdick, D. Rishi, J. Sheehan, Z. Shen, B. Stilson, A. Wade, K. Wang, C. Wilhelm, and S. Kohlmeier. Cord-19: The covid-19 open research dataset. *ArXiv*, 04 2020.