Universidad Internacional de Valencia

MÁSTER UNIVERSITARIO
EN INTELIGENCIA ARTIFICIAL

Implementación y evaluación de aplicaciones usando BERT, para los problemas de análisis de sentimiento y respuesta a preguntas.

Curso académico 2021-22 - Matrícula abril 2021
Tercera convocatoria: mayo 2022

Por el alumno **Izaguirre Viera, Luis Arturo**
DNI (Chile): 24.767.484-5
lsizaguirre@gmail.com

Dirigido por **Díaz Pinto, Andrés**
andresyesid.diaz@campusviu.es

“It’s not artificial intelligence I’m worried about,
it’s human stupidity!”

Neil Jacobstein

Agradecimientos

A mis padres, por su inconmesurable amor, por siempre guiarme, por inspirarme a seguir sus pasos y ser cada día mejor.

A Nancy Victoria, quien ha sabido enseñarme a luchar por mis metas y a nunca rendirme. Gracias por tu apoyo incondicional.

A todo el cuerpo docente de la Universidad Internacional de Valencia por su dedicación y enseñanza. En especial al director de este trabajo, Andrés Díaz Pinto.

Índice general

Índice de figuras

Índice de tablas

Resumen

BERT, siglas para ***B**idirectional **E**ncoder **R**epresentations from **T**ransformers* es un modelo de lenguaje preentrenado basado en la arquitectura de transformers que desde su aparición en octubre del 2018 ha avanzado en de forma acelerada en el estado del arte de diversas tareas comprendidas en el campo del procesamiento del lenguaje natural, como el resumen de textos, clasificación, similaridad semántica, entre otras.

El objetivo de este trabajo de investigación es evaluar el uso, la facilidad de implementación y el rendimiento que puede obtener este modelo en la solución de tareas para las que en un principio no fue entrenado.

Se eligió trabajar con dos tareas, la primera de ellas fue un problema de clasificación a través del análisis de sentimiento sobre un conjunto de datos que contiene reseñas de películas y la segunda un problema de comprensión lectora donde se intenta identificar una respuesta dada una pregunta y un contexto. Para resolver ambos problemas se realizó un proceso de ajuste o fine-tunning sobre el modelo base de BERT.

Los resultados mostraron además de la facilidad de implementación, un excelente rendimiento al compararlo con los modelos que definen el estado del arte para cada una de estas tareas. Con un ligero proceso de ajuste se logró tener para el primer problema una precisión cercana al 92 % y para el problema de preguntas y respuestas un score F1 cercano al 77 %.

Los resultados sugieren que BERT es un excelente punto de partida para la construcción de soluciones a distintos problemas en el campo del procesamiento de lenguaje natural.

Introducción 1

1.1. Procesamiento Natural del Lenguaje

Los últimos años hemos sido testigos del auge de la inteligencia artificial y del impacto que esta ha generado en distintos aspectos de nuestras vidas, convirtiéndose en una tecnología que ha cambiado la forma en la que interactuamos y vivimos. Hemos presenciado sorprendentes avances de la inteligencia artificial en distintos ámbitos y la evolución que ha tenido en la resolución de muchas tareas específicas, logrando en varias de ellas superar las capacidades cognitivas propias de los seres humanos.

El Procesamiento del Lenguaje Natural o *Natural Language Processing* (NLP, por sus siglas en inglés) es una rama de la inteligencia artificial que habilita a las computadoras a comprender, entender y procesar el lenguaje humano. El NLP provee de un conjunto de técnicas y metodologías para analizar, interpretar y darle significado al lenguaje y de este modo, poder generar conocimiento a partir de este.

El NLP se ha convertido en una de las tareas más complejas a resolver en la búsqueda de recrear las capacidades cognitivas propias de los seres humanos a través de las máquinas. Y es que, sin lugar a duda, el lenguaje humano es un proceso bastante complejo que ha sido el resultado de la propia evolución a lo largo de miles de años y sustentado por un motor biológico con más de cien mil millones de neuronas que interactúan entre sí.

Para hacerlo aún más complejo, nuestro lenguaje está lleno de ambigüedades, de palabras con distintas acepciones, giros y diversos significados según el contexto y además elementos que aún son de difícil comprensión como la ironía o el sarcasmo. Esto hace que el procesamiento de lenguaje natural haya sido una de las tareas más difíciles de dominar en el mundo de la inteligencia artificial.

1.2. El aprendizaje automático y el NLP

En este camino del entendimiento y comprensión del lenguaje humano por las maquinas, se han explorado distintas soluciones desde algoritmos o arquitecturas basadas en reglas donde se intenta modelar instrucción a instrucción y a través de reglas complejas, hasta soluciones donde dejamos que algoritmos puedan inferir estas reglas a través del Aprendizaje Automático o *Machine Learning* (ML, por sus siglas en inglés).

El *machine learning* a pesar de ser una disciplina con muchos años de investigación, ha ganado una relevancia considerable en los últimos años debido al aumento cada vez mayor de la capacidad de cómputo y con ello la habilidad de poder procesar grandes cantidades de datos. Esto ha posibilitado el desarrollo acelerado del *machine learning* y los modelos de Aprendizaje Profundo o *Deep Learning* (DL, por sus siglas en inglés). Las redes neuronales artificiales ocupan una posición muy prometedora en el área del *machine learning* y han transformado la forma en que se desarrollan las aplicaciones de NLP.

Para este tipo de problemas donde el dato de entrada es un texto de naturaleza secuencial, se han explorado distintas arquitecturas que van desde las redes neuronales recurrentes (ver sección 1.2.1), pasando por las redes *long short-term memory* (ver sección 1.2.2), hasta las poderosas redes basadas en la novedosa arquitectura de Transformer, concepto sobre el que se profundizará en la sección 1.3.1.

1.2.1. Redes Neuronales Recurrentes (RNN)

Las Redes Neuronales Recurrentes o *Recurrent Neural Networks* (RNN, por sus siglas en inglés), son un tipo de redes que tienen un excelente desempeño para el análisis de datos que tienen una naturaleza secuencial, es decir, datos donde el orden tiene importancia relevante. Es por ello, que generalmente es empleada en los problemas de procesamiento de audio, vídeo o texto. La idea es que los datos de entrada tengan una cierta secuencia adjunta como en los textos donde la secuencia de palabras genera un contexto que agrega significado a la oración.

Como es explicado por Torres (2019), las redes neuronales recurrentes fueron concebidas en la década de 1980. Pero estas redes fueron en un principio difíciles de entrenar por sus requerimientos en computación y no es solo hasta estos últimos años, en los que se generaron grandes avances en temas de capacidades de cómputo y procesamiento, que este tipo de redes se han hecho más accesibles y se ha popularizado su uso.

El aspecto más importante de las RNN es que funcionan en bucles, los datos fluyen circularmente, en consecuencia aprenden no solo la entrada del estado actual sino también la entrada anterior. Se dice entonces que este tipo de red incorpora la retroalimentación lo

que consigue crear un concepto de temporalidad, permitiendo a la red tener “memoria”.

En un modelo de red típico, la función de activación solo actúa en una dirección, hacia delante, desde la capa de entrada hacia la capa de salida, es decir, que no recuerdan valores previos. Una red RNN es parecida, pero incluye conexiones que apuntan “hacia atrás”, una especie de retroalimentaciones entre las neuronas dentro de las capas.

Siguiendo esta misma idea, una capa de neuronas recurrentes se puede implementar de tal manera que, en cada instante de tiempo, cada neurona recibe dos entradas, la entrada correspondiente de la capa anterior y a su vez la salida del instante anterior de la misma capa, como es mostrado en la figura 1.1.

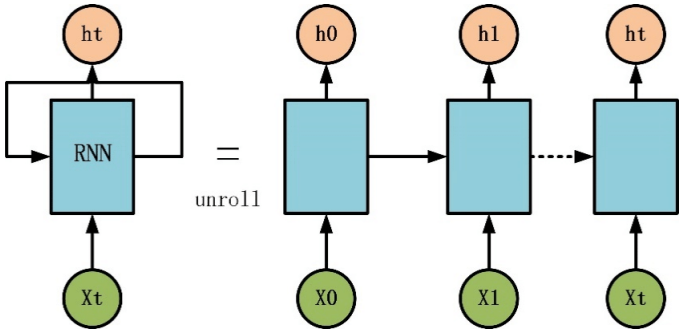


Figura 1.1: ***Arquitectura típica de una red neuronal recurrente, donde podemos observar la secuencia y la relación de cada celda con los estados anteriores. Fuente: (Hao et al., 2020)***

Precisamente esta “memoria interna” es lo que hace a este tipo de redes muy adecuadas para problemas de aprendizaje automático que involucran datos secuenciales. Gracias a su “memoria interna”, las RNNs pueden recordar información relevante sobre la entrada que recibieron, lo que les permite ser más precisas en la predicción de lo que vendrá después, manteniendo información de contexto a diferencia de los otros tipos de redes que no pueden recordar acerca de lo que ha sucedido en el pasado, excepto lo reflejado en su entrenamiento a través de sus pesos.

Dos conceptos importantes que afectan a las RNN son los gradientes explosivos (*exploding gradient*, en inglés) y el desvanecimiento del gradiente (*vanishing gradients*, en inglés), aunque ambos efectos son propios en general de cualquier tipo de red muy grande en números de parámetros, sea o no sea recurrente.

Estos conceptos son bien descritos por Torres (2019), recordando que el gradiente indica el cambio a realizar en todos los pesos con respecto al cambio en el error, en tal sentido, hablamos de “gradientes explosivos” cuando el algoritmo asigna una importancia exageradamente alta a los pesos, sin mucha razón y esto genera un problema en el entrenamiento y hablamos de “gradientes desaparecidos” cuando los valores de un gradiente son demasiado pequeños y el modelo deja de aprender o requiere demasiado tiempo de-

bido a ello.

1.2.2. Redes Long Short-Term Memory (LSTM)

Como se describe en la sección anterior, las redes neuronales recurrentes convencionales presentan problemas en su entrenamiento debido a que los gradientes retropropagados tienden a crecer enormemente o a desvanecerse con el tiempo debido a que el gradiente depende no solo del error presente sino también los errores pasados. La acumulación de errores provoca dificultades para memorizar dependencias a largo plazo.

Las Redes de Gran Memoria a Corto Plazo o *Long Short-Term Memory* (LSTM, por sus siglas en inglés), son una extensión de las redes neuronales recurrentes, que como indica Thomas (2020), fue propuesta por Sepp Hochreiter y Jürgen Schmidhuber en 1997 para abordar el problema de los gradientes explosivos y desaparecidos. Este tipo de redes básicamente amplían su memoria para aprender de experiencias importantes que han sido analizadas hace mucho tiempo en la secuencia. Las LSTM permiten a las RNNs recordar sus entradas durante un largo período de tiempo. Esto se debe a que LSTM contiene su información en la memoria, que puede considerarse similar a la memoria de un ordenador, en el sentido que una neurona de una LSTM puede leer, escribir y borrar información de su memoria.

En una neurona LSTM hay tres puertas que controlan el modo en que la información fluye dentro o fuera de la unidad: puerta de entrada (*input gate*), puerta de olvidar (*forget gate*) y puerta de salida (*output gate*), como muestra la figura 1.2. Estas puertas determinan si se permite o no una nueva entrada, se elimina la información porque no es importante o se deja que afecte a la salida en el paso de tiempo actual.

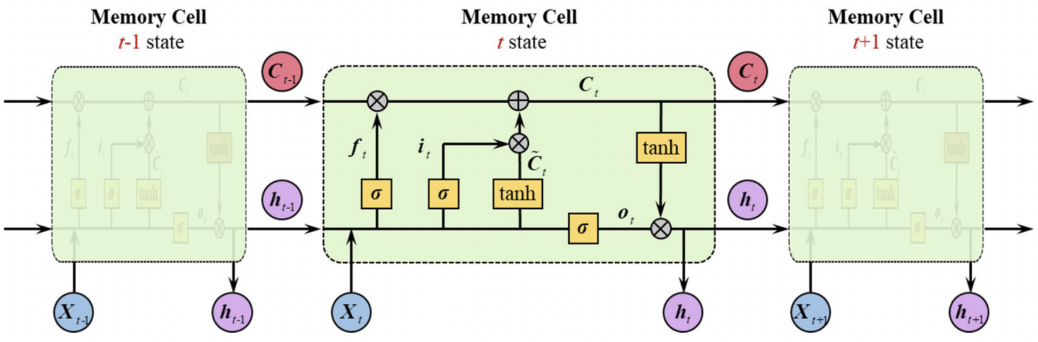


Figura 1.2: ***Mirada interna a una celda de memoria en una red de tipo Long Short-Term Memory (LSTM). En la imagen se pueden observar los tres tipos de puertas presentes en la neurona. Fuente: (Olah, 2015)***

1.3. Bidirectional Encoder Representations from Transformer

BERT siglas para ***B**idirectional **E**ncoder **R**epresentations from **T**ransformers* es un poderoso modelo de lenguaje publicado por investigadores de Google en octubre de 2018 que usa dos etapas, el preentrenamiento y el ajuste fino (*fine-tunning*, en inglés), para crear modelos muy eficientes capaces de resolver una amplia gama de tareas relacionadas al procesamiento de lenguaje natural (Devlin et al., 2018).

Desde su aparición BERT ha causado revuelo en la comunidad del *machine learning* producto de su característica arquitectura que permite que el mismo modelo preentrenado se pueda ajustar para resolver una variedad de tareas finales que pueden no ser similares a las tareas para las que en un principio se entrenó y obtener resultados similares a los que definen el estado del arte para estas tareas en el campo del NLP (ver figura 1.3), lo que define el objetivo principal de este trabajo (ver sección 2.1). Los modelos como BERT y sus similares han superado varios puntos de referencia importantes en tareas de procesamiento de lenguaje natural.

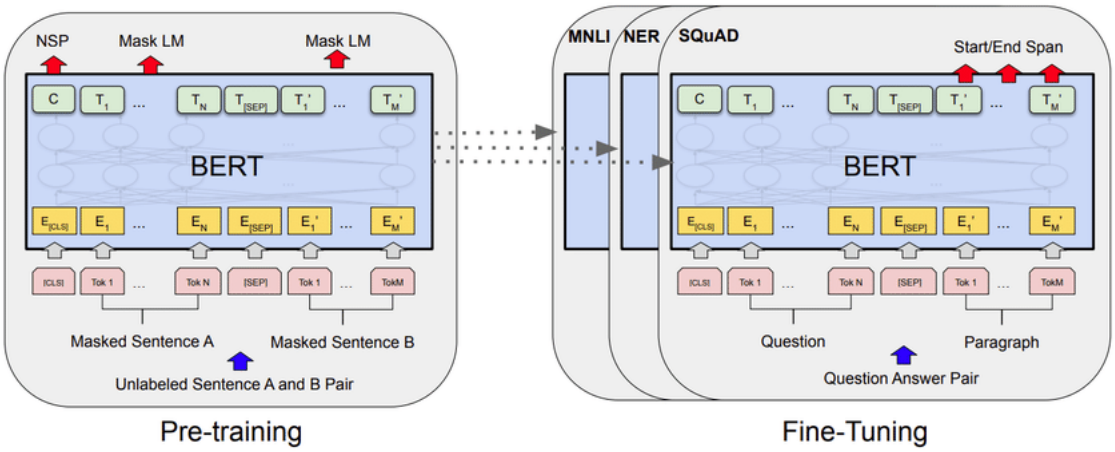


Figura 1.3: ***Procedimientos generales de preentrenamiento y fine-tunning en BERT. BERT utiliza la misma arquitecturas tanto en el preentrenamiento como en el fine-tuning. Los parámetros del modelo preentrenado se utilizan para inicializar modelos para resolver otras tareas. Fuente: (Devlin et al., 2018)***

1.3.1. Mecanismos de Atención y Transformer

La base de BERT es el modelo Transformer, un tipo de red neuronal propuesta recientemente, basada en mecanismos de atención, que acepta como entrada datos secuenciales (por ejemplo, una secuencia de palabras) y produce algún resultado (por ejemplo,

la predicción de sentimientos). A diferencia de las redes recurrentes tradicionales, como las LSTM, que procesan cada elemento de secuencia uno a la vez, con Transformer se procesan todos los elementos simultáneamente (autoatención) formando conexiones directas entre todos los elementos. Esto no solo permite una mayor paralelización, sino que también da como resultado una mayor precisión en una variedad de tareas, lo que le ha permitido impulsar importantes avances en el procesamiento del lenguaje natural.

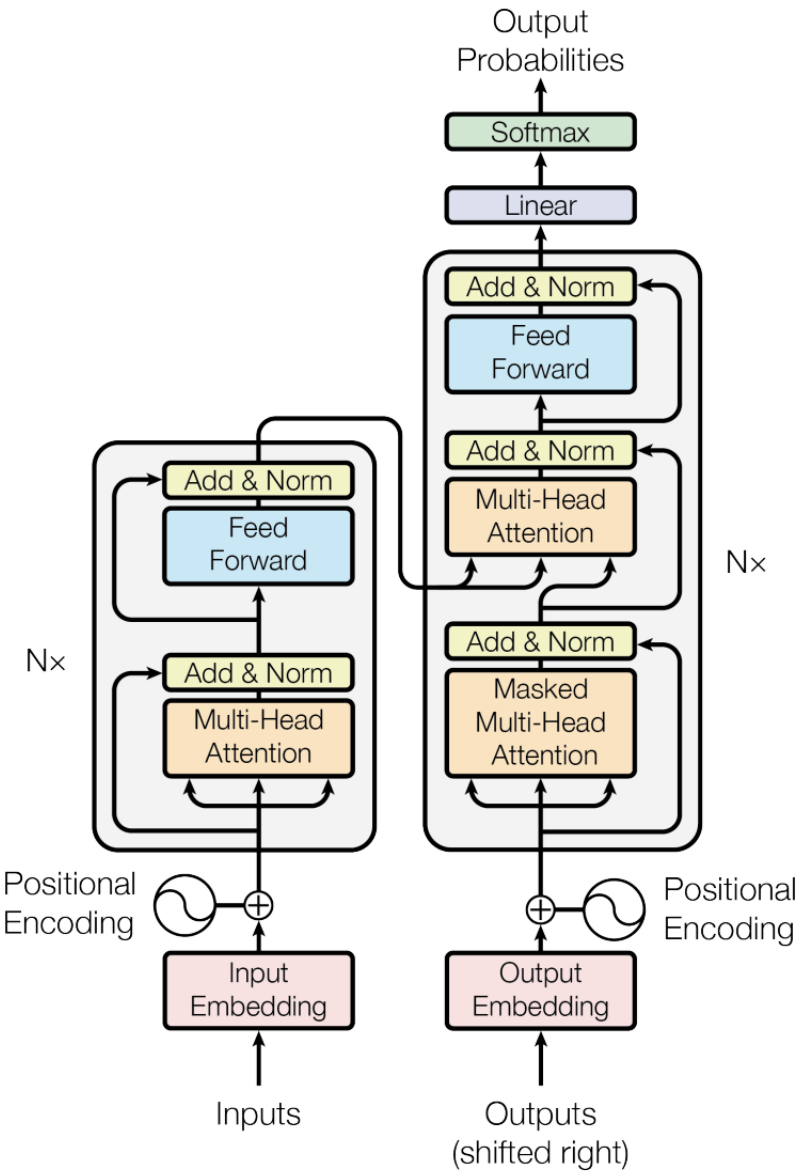


Figura 1.4: ***Arquitectura del modelo Transformer propuesto en el paper "Attention is all you need!". Fuente: (Vaswani et al., 2017)***

La publicación del *paper "Attention is all you need"* (Vaswani et al., 2017) representó una completa revolución en el campo del procesamiento del lenguaje natural. En este *paper* se propone por primera vez la arquitectura Transformer (figura 1.4). La clave de

esta novedosa arquitectura es el mecanismo de autoatención y el uso de autoencoders para lograr un entrenamiento más ágil y producir resultados eficientes. En la actualidad, años después de la publicación del *paper*, esta arquitectura sigue siendo estado del arte en el mundo del NLP y la base de implementaciones como BERT o GPT.

Aunque las RNNs permiten observar palabras anteriores de la secuencia, sufren de memoria cortoplacista. Esto provoca que cuando trabajamos con una secuencia larga las RNNs no pueden observar palabras muy antiguas. Las LSTMs tienen una ventana de memoria más grande que las RNNs, pero siguen teniendo una capacidad limitada. Esta secuencialidad es un obstáculo para la paralelización del proceso. Además, cuando estas secuencias son demasiado largas, el modelo tiende a olvidar el contenido de las posiciones distantes en la secuencia o a mezclarlo con el contenido de las posiciones siguientes. El mecanismo de autoatención resuelve esto ya que, permiten modelar dependencias sin considerar su distancia en las secuencias de entrada o salida. En la mayoría de los casos los mecanismos de atención se utilizan en conjunto con una red recurrente.

El Transformer utiliza dos tipos diferentes de funciones de atención (figura 1.5):

- **Atención de producto punto escalado** o *scaled dot-product attention*, que calcula la función de atención en un conjunto de consultas simultáneamente, empaquetadas en una matriz.

- **Atención Multi-Cabeza** o *Multi-Head Attention*, que permite que el modelo atienda de manera conjunta la información de diferentes subespacios de representación en diferentes posiciones.

En cuanto a las entradas, el *paper* de Vaswani et al. (2017) menciona que las secuencias de entrada se representan utilizando alguna forma de *word embeddings*, es decir, una representación vectorial de cada palabra que guarda información semántica. Esto se hace tanto para el *encoder* como para el *decoder*. Los *word embeddings* por sí solos carecen de cualquier información posicional que se logra en las RNNs en virtud de su naturaleza secuencial. Mientras tanto, en la autoatención se pierde cualquier información posicional debido a la aplicación de la función *softmax*. Para preservar la información posicional, el Transformer inyecta un vector a los embeddings de entrada individuales. Este vector inyectado se denomina “codificación posicional” y se agrega a los *embeddings* de entrada en la parte inferior de las pilas del *encoder* y *decoder*.

Los autores compararon los resultados de la arquitectura con otras soluciones consideradas estado del arte en 2017. La arquitectura Transformer obtuvo un desempeño mejor que el de otros modelos. (figura 1.6)

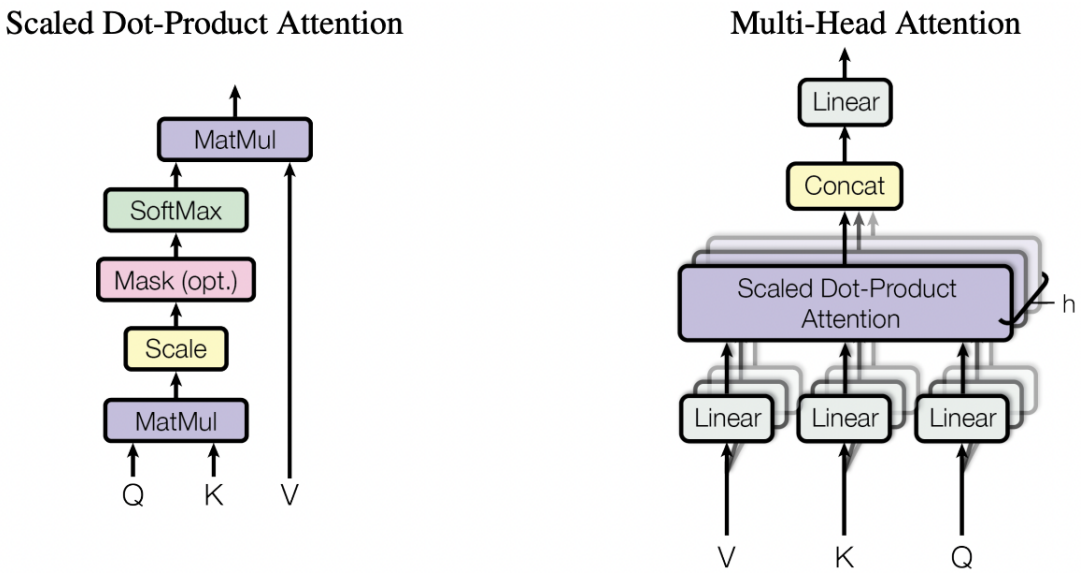


Figura 1.5: ***A la izquierda podemos observar el mecanismo de atención basado en el producto punto escalado y a la derecha el mecanismo de atención multi-head, el cual consiste en varias capaz de atención corriendo en paralelo. Fuente: (Vaswani et al., 2017)***

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Figura 1.6: ***La imagen compara el resultado de la implementación de la arquitectura Transformer en un problema de traducción superando el desempeño de otros modelos considerados el estado del arte antes de la publicación del paper "All you Need is Attention". En la comparación se observa además de un mejor desempeño, un costo de procesamiento menor al del resto de modelos. Fuente: Vaswani et al. (2017)***

1.3.2. Arquitectura de BERT

La arquitectura de BERT es básicamente una pila de *encoders* de la arquitectura Transformer.

En el *paper* original de BERT Devlin et al. (2018) hablan de dos modelos o versiones de BERT que se pueden elegir en términos de implementación:

- **BERT$_{BASE}$:** L=12, H=768, A=12, 110 millones de parámetros.
- **BERT$_{LARGE}$:** L=24, H=1024, A=16, 340 millones de parámetros.

donde:
L: número de capas de encoders
H: tamaño de la capa oculta (dimensión del embedding)
A: número de encabezados de auto atención

La innovación técnica clave de BERT es aplicar el entrenamiento bidireccional de Transformer al modelamiento del lenguaje. Esto es diferente a los enfoques de modelos anteriores que analizaban las secuencias de texto en una sola dirección. Los resultados obtenidos por Devlin et al. (2018) en su *paper*, muestran que un modelo de lenguaje entrenado bidireccionalmente tiene un sentido más profundo del contexto y flujo del lenguaje que los modelos de lenguaje de una sola dirección.

El preentrenamiento de BERT está basado en dos tareas no supervisadas:

1. **Modelo de Lenguaje Enmascarado** o *Masked Language Model* (MLM): esta tarea es la que habilita el aprendizaje bidereccional profundo en el modelo. En esta tarea se enmascara un porcentaje de los tokens de entrada (reemplazando los tokens por el token especial [MASK]) al azar y el modelo intenta predecir los tokens que han sido enmascarados. Esos tokens predichos del modelo luego son introducidos a una función softmax de salida sobre el vocabulario para obtener las palabras de salida finales.
2. **Predicción de la Siguiente Oración** o *Next Sentence Prediction* (NSP): esta tarea es la utilizada para que el modelo sea capaz de resolver tareas donde es importante la relación entre dos oraciones. En este caso, el modelo es entrenado recibiendo pares de frases como entrada y aprendiendo a predecir si la segunda frase del par es la subsiguiente dentro del documento original de entrada.

El modelo esta entrenado en ambas tareas mencionadas anteriormente de forma simultánea. Esto es posible debido a la forma en la que el modelo procesa las entradas y las salidas, las cuales son compatibles para ambos problemas.

En cuanto a la entrada (ver figura 1.6), BERT necesita recibir información tanto para una sola oración como para dos oraciones agrupadas sin ambigüedades en una secuencia de token. Los autores del *paper* oficial señalan que una secuencia de entrada puede ser un tramo arbitrario de texto contiguo, en lugar de una oración lingüística real. El token [SEP] se usa para separar dos oraciones.

A diferencia de las RNNs en las que las entradas se alimentan secuencialmente, el modelo BERT no puede conservar el orden de los tokens de entrada. El orden de las palabras en cada idioma es significativo, tanto semántica como sintácticamente. Para realizar correctamente la tarea de predicción de la siguiente oración, el modelo debe ser capaz de distinguir entre las dos oraciones. Ambos problemas se resuelven agregando *embeddings* que contienen la información requerida a los tokens correspondientes a la entrada original y usando el resultado como entrada a nuestro modelo BERT.

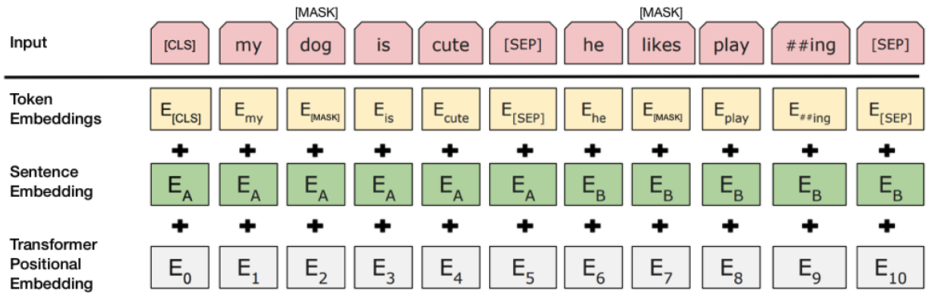


Figura 1.7: ***Representación de la entrada del modelo de BERT. El input del modelo es la suma de los token embeddings, el embedding de segmentación y el embedding posicional. Fuente: Vaswani et al. (2017)***

Los siguientes *embeddings* se agregan a los *embeddings* de tokens:

- **Embedding de segmentos** o *Segment Embedding*: Proporciona información sobre la oración de la que forma parte un token en particular.
- **Embedding posicional** o *Position Embedding*: Proporciona información sobre el orden de las palabras de entrada.

En cuanto a la salida, también está diseñada para predecir el resultado de dos tareas diferentes de forma simultánea. Esto lo hace mediante el uso de diferentes capas FFNN y *softmax* construidas sobre las salidas del último *encoder*. El primer token de entrada es siempre un token de clasificación especial [CLS]. El estado final correspondiente a este token se usa como la representación de secuencia agregada para las tareas de clasificación y se usa para la predicción de la siguiente oración. Los estados finales correspondientes a los tokens [MASK] se introducen en la FFNN y *softmax* para predecir la siguiente palabra del vocabulario.

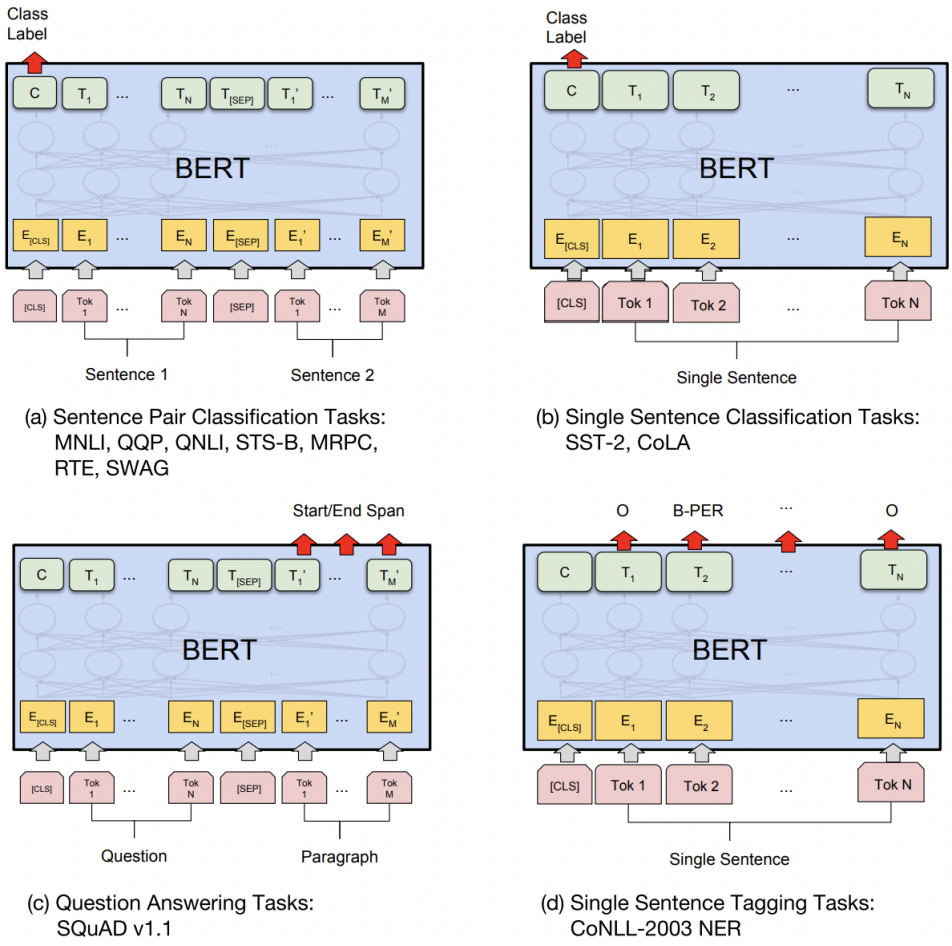
1.3.3. BERT y Fine Tunning

La idea de la transferencia de aprendizaje o *transfer learning* (en inglés), es entrenar un modelo en una tarea y luego aprovechar el conocimiento adquirido para mejorar el rendimiento del modelo en una tarea relacionada. El *fine-tunning* es un enfoque para transferir

el aprendizaje en el que se agrega una capa a la salida del modelo para que se ajuste a una nueva tarea y se entrena el modelo resultante.

Una consecuencia positiva de agregar capas (entrada/salida) y no cambiar el modelo BERT es que solo se necesita aprender una cantidad mínima de parámetros desde cero, lo que hace que el procedimiento sea rápido, rentable y eficiente en recursos.

En la clasificación de pares de oraciones y la clasificación de oraciones únicas, el estado final correspondiente al token [CLS] se usa como entrada para las capas adicionales que hacen la predicción. (ver figura 1.8)



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Figura 1.8: ***Esta imagen ilustra el proceso de fine-tunning para BERT en diferentes tareas y como se amolda la arquitectura unificada para distintas tareas. Fuente: Vaswani et al. (2017)***

En las tareas de preguntas y respuestas, se introducen un vector de inicio (S) y uno final (E) durante el *fine-tunning*. La pregunta se introduce al modelo como oración A y la respuesta como oración B. La probabilidad de que la palabra i sea el comienzo del intervalo de respuesta se calcula como un producto escalar entre Ti (estado final correspondiente al i-ésimo token de entrada) y S (vector de inicio) seguido por un *softmax* sobre todas las

palabras del párrafo. Se utiliza un método similar para el tramo final. (ver figura 1.8)

Entre las ventajas del proceso de *fine-tunning* podemos mencionar:

- **Desarrollo más rápido** En primer lugar, los pesos del modelo BERT preentrenados ya codifican mucha información sobre el lenguaje. Como resultado, lleva mucho menos tiempo entrenar el modelo ajustado, es como si ya se hubiera entrenado las capas inferiores de nuestra red de forma exhaustiva y solo se necesita ajustar levemente mientras se usa su salida como características para la tarea de clasificación. De hecho, los autores recomiendan solo de 2 a 4 épocas de entrenamiento para ajustar BERT en una tarea específica de NLP, en comparación con los cientos de horas de Unidad de Procesamiento Gráfico o *Graphic Processing Unit* (GPU, por sus siglas en inglés) necesarias para entrenar el modelo BERT original o una LSTM desde cero.

- **Menos datos** Además, y quizás igual de importante, debido a los pesos previamente entrenados, este método permite ajustar una tarea en un conjunto de datos mucho más pequeño que el que se requeriría en un modelo construido desde cero. Un inconveniente importante de los modelos NLP creados *from scratch* es que a menudo se requiere de un conjunto de datos prohibitivamente grande para entrenar nuestra red con una precisión razonable, lo que significa que se tiene que dedicar mucho tiempo y energía a la creación del conjunto de datos. Al ajustar BERT, ahora se puede entrenar un modelo para que tenga un buen rendimiento con una cantidad mucho menor de datos de entrenamiento.

- **Mejores resultados** Finalmente, se demostró que este sencillo procedimiento de ajuste fino (por lo general, agregar una capa completamente conectada encima de BERT y entrenar durante algunas épocas) logra resultados de vanguardia con ajustes mínimos específicos de tareas para una amplia variedad de tareas: clasificación, inferencia de lenguaje, similitud semántica, respuesta a preguntas, etc. En lugar de implementar arquitecturas personalizadas, se ha demostrado que funciona bien en una tarea específica, simplemente ajustar BERT. Esta opción se muestra como una alternativa mejor (o al menos igual).

1.3.4. BERT, variantes y otros modelos

BERT no es el único modelo que está produciendo estos resultados innovadores. Existe otro modelo desarrollado por OpenAI llamado GPT-3 (Brown et al., 2020) que está causando bastante revuelo en el mundo del NLP.

GPT ha causado bastante revuelo en Internet últimamente. Este modelo tiene algo en común con BERT, y es que ambos modelos reutilizan un componente de Transformer. Como se indicó anteriormente, BERT apila la parte del encoder del Transformer como su bloque de construcción. Mientras tanto, GPT usa la parte del decoder del Transformer como su bloque de construcción.

Las conexiones bidireccionales en BERT se deben a la autoatención bidireccional del *encoder*. Mientras tanto, las conexiones en GPT solo están en una sola dirección, de izquierda a derecha, debido al diseño del *decoder* para evitar mirar predicciones futuras.

Adicionalmente, hay algunas variantes de BERT donde la arquitectura no es diferente, pero presentan ciertas adecuaciones. (ver tabla 1.1)

Paste Latex Here

Modelo	**Creador**	**Descripción**
Albert Lan et al. (2019)	Google	Este paper describe técnicas de reducción de parámetros para disminuir la reducción de memoria y aumentar la velocidad de entrenamiento de los modelos BERT.
RoBERTa Liu et al. (2019)	Facebook	Este paper cree que los modelos BERT originales estaban poco entrenados y muestra que con más entrenamiento/ajuste puede superar los resultados iniciales.
ERNIE Sun et al. (2019b)	-	Representación mejorada a través de la integración del conocimiento de Baidu: se inspira en la estrategia de enmascaramiento de BERT y aprende la representación del lenguaje mejorada mediante estrategias de enmascaramiento del conocimiento, que incluyen el enmascaramiento a nivel de entidad y el enmascaramiento a nivel de frase.
DistilBERT Sanh et al. (2019)	-	BERT más pequeño. Reduce el número de capas a 6 y elimina el embedding de segmentación

Tabla 1.1: ***Lista de modelos basados en BERT. Fuente: Brown et al. (2020)***

Objetivos

2

La publicación de BERT por parte de Google marcó el inicio de una nueva era en el machine learning obteniendo con este modelo excelentes resultados que definieron un nuevo estado del arte en muchas tareas y benchmarks en el campo del procesamiento de lenguaje natural.

El hecho de ser un modelo accesible, con una arquitectura unificada que permite un ajuste rápido y fácil, lo convierte en un modelo de referencia.

2.1. Objetivo General

Basado en los fundamentos antes expuestos, el objetivo principal de este trabajo es: **Evaluar el uso y desempeño de BERT (Biderectional Encoder Representations for Transformers) como modelo en la resolución de tareas relacionadas al procesamiento del lenguaje natural, distintas a las que para en un principio fue preentrenado.** Específicamente, se trabajará en dos tareas, el análisis de sentimiento y el problema de respuesta a preguntas.

2.2. Objetivo Específicos

Para evaluar el uso y desempeño de este modelo en la resolución de problemas asociados al procesamiento de lenguaje natural, trabajaremos en los objetivos específicos citados a continuación:

1. Identificar problemas típicos del procesamiento del lenguaje natural donde, el uso de modelos basados en *transformers* hayan tenido un aporte considerable en su solución.

2. Identificar y seleccionar un conjunto de datos para cada tarea que permita construir

una solución a través del ajuste del modelo. Estudiar, limpiar y dividir los datos en los subconjuntos de entrenamiento, prueba y validación.

3. Evaluar el estado del arte para cada uno de estos problemas, entendiendo el desempeño de distintos algoritmos empleados en su solución.

4. Desarrollar soluciones basadas en el ajuste de BERT, adecuándose para resolver los problemas seleccionados.

5. Comparar el desempeño de las soluciones desarrolladas con el estado del arte haciendo uso de las métricas apropiadas en cada benchmark.

Metodología y Desarrollo

3

La idea general del *transfer learning* es preentrenar un modelo usando una red neuronal de gran tamaño sobre un dataset de gran volumen para después ajustar la red (*fine-tunning*) usando un dataset mucho más reducido y ajustado a la tarea especifica que se intenta resolver. El aprendizaje por transferencia marca una gran diferencia y nos permite resolver nuevos problemas mucho más fácilmente, ya que normalmente podemos cargar pesos previamente entrenados de un modelo sin requerir de un gran conjunto de datos para resolver una tarea específica.

BERT es un modelo de lenguaje preentrenado que ha demostrado que el concepto de la transferencia del aprendizaje ha resultado ser una herramienta poderosa no solo en los problemas de visión por computacdora (*computer vision*, en inglés), sino también en el procesamiento natural del lenguaje y que podría ser suficiente con un ligero ajuste (*fine-tunning*) implementar un nuevo modelo que resuelva de forma satisfactoria una nueva tarea.

El *transfer learning* se convirtió en la estrategia predeterminada en la visión por computadora alrededor del año 2014. A menudo usaban modelos que estaban entrenados previamente para la clasificación de imágenes en Imagenet Russakovsky et al. (2015), lo que significaba que usaban aprendizaje supervisado y un gran conjunto de datos con etiquetas para entrenar previamente el modelo. Durante varios años se estuvo investigando sin éxito formas de usar el aprendizaje por transferencia en el procesamiento del lenguaje natural y se estuvo tratando de encontrar tareas y conjuntos de datos con las mismas propiedades atractivas que tenía Imagenet para la visión por computadora. Finalmente, en 2018 al menos tres artículos diferentes (BERT Devlin et al. (2018), ELMo Peters et al. (2018), GPT Radford et al. (2018)) demostraron que se puede usar el *transfer learning* también en el procesamiento natural del lenguaje.

Entre estos modelos, se puede decir que BERT fue el más limpio, ya que requirió la menor cantidad de *fine-tunning*, un conjunto de datos más pequeño y, en general, también dio los mejores resultados.

Basados en estos conceptos de *transfer learning* y *fine-tunning* procederemos a eva-

luar el uso y el desempeño de BERT en la resolución de problemas de procesamiento natural del lenguaje siguiendo los siguientes pasos:

1. **Selección de problemas**
   En esta primera etapa elegimos un par de problemas típicos del procesamiento natural del lenguaje pero que a su vez sean distintos a los problemas para los que BERT fue entrenado inicialmente. La idea es poder demostrar la adaptabilidad del modelo para resolver cualquier clase de problemas basado en la arquitectura propia de BERT. Los problemas con los que trabajaremos son el Análisis de Sentimiento (clasificación) y la tarea de Respuesta a Preguntas.

2. **Selección de los conjuntos de datos (*datasets*)**
   En cuanto a la selección de los *datasets* se decidió trabajar con conjuntos de datos asociados a cada uno de los problemas antes descritos, procurando que los *datasets* elegidos permitan establecer indicadores de comparación y que existan estudios de comparación del desempeño con otros modelos y soluciones. Para el caso de análisis de sentimiento se trabajará con un *dataset* de reseñas de películas de IMDB creado por investigadores de la Universidad de Stanford, para el problema de preguntas y respuestas se trabajó con el conocido *dataset* SQuAD en su versión 1.0.

3. **Fine Tuning de BERT**
   Para cada uno de los problemas elegidos se hizo un ajuste del modelo inicial de BERT usando para el entrenamiento un *dataset* etiquetado. La intención es hacer las adecuaciones necesarias al modelo para intentar encontrar una solución simple a cada uno de los problemas elegidos.

4. **Evaluación de los resultados**
   Se compararon los resultados obtenidos en cada problema con otros modelos implementados para resolver el problema. La idea es validar que simplemente haciendo un pequeño *fine-tunning* del modelo original podemos obtener resultados similares o cercanos a los del estado del arte en cada problema.

3.1. Problema de Análisis de Sentimiento (Clasificación)

3.1.1. Descripción del problema

El análisis de sentimiento es un subcampo del procesamiento de lenguaje natural que ha tenido un gran desarrollo en los últimos años. Este interés ha sido alimentado en gran parte por el auge de las redes sociales que ha dado lugar a la aparición de blogs, foros y aplicaciones en línea que le permiten a los usuarios discutir sobre cualquier tema y compartir sus opiniones. Este tipo de análisis es ampliamente usado por distintas empresas para entender la opinión de sus clientes y conocer la aceptación de sus productos y servicios.

A través del análisis de sentimiento las empresas pueden estudiar los comentarios de los clientes para prestar una mejor atención, crear mejores productos o mejorar sus estrategias de marketing para atraer nuevos clientes. En líneas generales los resultados del análisis de sentimiento ayudan a conocer mejor los intereses y opiniones de los clientes sobre las tendencias del mercado.

El objetivo principal del problema es analizar una secuencia de texto y a partir de ella identificar o extraer información subjetiva que permita inferir si el sentimiento asociado a la frase es positivo o negativo. Desde otro punto de vista, se trata de una tarea de clasificación de la polaridad asociada a los textos que pueden estar presentes en documentos, oraciones u opiniones de forma masiva y automática.

En el estudio de Bhavitha et al. (2017), los autores señalan que las técnicas de análisis de sentimientos se pueden clasificar en tres enfoques distintos en cuanto a la forma de solución.

1. **Enfoque de aprendizaje automático:** Cuando utilizamos modelos de *machine learning* en la solución del problema y generalmente nos basamos en un modelo especifico a partir del entrenamiento sobre un gran corpus de documentos.

2. **Enfoque basados en léxico:** Mediante este enfoque, la orientación semántica de las palabras de un texto se calcula obteniendo las polaridades de las palabras a partir de un léxico predefinido. Es decir, habrá palabras en el texto que por si sola denotan ya una polaridad. Ejemplo, palabras como *magnifico, bueno, grandioso, espectacular*, tienen un sentido positivo, mientras que palabras como *horrible, malo, decepción*, tienen un significado negativo.

3. **Enfoque híbrido:** Se basa en la combinación de información lingüística y estadística, reglas semánticas junto con el uso de redes neuronales profundas.

Más recientemente, se utilizan técnicas de aprendizaje profundo, como BERT, T5 y

GPT, para entrenar clasificadores de sentimientos de alto rendimiento que se evalúan mediante métricas como F1, recuperación y precisión.

3.1.2. Selección del Dataset

Para evaluar los sistemas de análisis de sentimientos, se utilizan generalmente conjuntos de datos de referencia como:

- **SST (Stanford Sentiment Treebank):** es un corpus con árboles de análisis sintáctico completamente etiquetados que permite un análisis completo de los efectos compositivos del sentimiento en el lenguaje. El corpus se basa en el conjunto de datos presentado por (Pang y Lee, 2005) y consta de 11.855 oraciones individuales extraídas de reseñas de películas. Se analizó con el analizador de Stanford e incluye un total de 215154 frases únicas de esos árboles de análisis, cada una anotada por 3 jueces humanos. Cada frase está etiquetada como negativa, algo negativa, neutral, algo positiva o positiva. El corpus con las 5 etiquetas se denomina SST-5 o SST de grano fino. Los experimentos de clasificación binaria en oraciones completas (negativas o algo negativas versus algo positivas o positivas con oraciones neutras descartadas) se refieren al conjunto de datos como SST-2 o SST binario.

- **GLUE (General Language Understanding Evaluation benchmark):** es una colección de nueve tareas de comprensión del lenguaje natural, que incluyen tareas de una sola oración CoLA y SST-2, tareas de similitud y paráfrasis MRPC, STS-B y QQP, y tareas de inferencia del lenguaje natural MNLI, QNLI, RTE y WNLI.

- **Large Movie Review Dataset IMDB:** es el conjunto de datos que se seleccionó para este experimento y que se procederá a describir a continuación.

El ***Large Movie Review Dataset IMDB*** es un conjunto de datos con 50.000 reseñas de películas creado para la clasificación de sentimientos binarios que contiene sustancialmente más datos que los otros conjuntos de datos de referencia.

Como describen los creadores de este *dataset* en su *paper* del 2019 *Learning Word Vectors for Sentiment Analysis* Maas et al. (2011), este conjunto de datos fue creado a partir de reseñas de películas realizadas por la audiencia y extraídas del sitio público IMDB. Para la construcción del mismo consideraron no incluir más de 30 reseñas por película.

Este conjunto de datos se encuentra balanceado, es decir, contiene un mismo número de reseñas positivas y negativas. Los autores para construirlo consideraron solo críticas altamente polarizadas en base al sistema de rating que tiene IMDB y de esa forma consi-

deraron una crítica negativa a aquellas reseñan que tenían una puntuación ≤ 4 sobre 10 y positiva si la puntuación era ≥ 7 sobre 10. Las reseñas neutrales no fueron incluidas

Para el propósito de este trabajo se usó una versión del *dataset* original transformado y disponible en la plataforma *kaggle.com* (Lakshmipathi, 2019). Esta versión ha sido transformada a un archivo de valores separados por comas, CSV (del inglés comma-separated values), en el cual se presentan dos columnas una para la reseña y otra para el sentimiento.

La razón por la que se escogió este *dataset* y no otro es la amplia cantidad de soluciones desarrolladas tomando como base de entrenamiento el mismo, esto permitirá poder hacer un estudio del resultado obtenido y compararlo con otros modelos.

3.1.3. Estado del arte

Para evaluar el estado del arte relacionado a la solución de análisis de sentimiento sobre el *dataset* de reseñas de IMDB se utilizaron diversas fuentes.

En el trabajo de Dang et al. (2020) los autores analizaron y realizaron una comparación entre distintos estudios que usan el aprendizaje profundo como las Redes Neuronales Profundas o *Deep Neural Networks* (DNN, por sus siglas en inglés), Redes Neuronales Recurrentes o *Recurrent Neural Networks* (RNN, por sus siglas en inglés) y Redes Neuronales Convolucionales o *Convolutional Neural Networks* (CNN, por sus siglas en inglés), para resolver diferentes problemas asociados al análisis de sentimiento. En este estudio no se incluyeron las recién aparecidas (para ese momento) redes basadas en Transformers. Los autores usaron distintos datasets para la evaluación entre ellos el mismo utilizado en este trabajo 3.1.2 y utilizaron en el preprocesamiento de los datos dos técnicas distintas de procesamiento de texto (*word embeddings* y TF-IDF). En sus conclusiones, los autores deducen una fiabilidad ligeramente superior en la mayoría de conjunto de datos con las redes neuronales recurrentes, sin embargo, igual mencionan un tiempo de cálculo mucho más largo, como se puede evidenciar en las imágenes 3.1 y 3.2

Es importante mencionar con respecto a este estudio que si bien las RNN evidenciaron tener un mejor desempeño con todos los datasets esto solo ocurrió con el uso de Word Embedding como técnica de preprocesado de los datos y no con TF-IDF.

Adicionalmente nos basamos en el sitio web NLP-progress (`https://nlpprogress.com/`) Ruder (2022), el cual es un repositorio en linea que hace un seguimiento del progreso en distintos problemas en el procesamiento del lenguaje natural NLP, incluidos los datasets y el estado del arte actual para las tareas de NLP más comunes.

Este repositorio muestra un ranking de modelos 3.1 que resuelven el problema de análisis de sentimiento haciendo uso del dataset de IMDB 3.1.2. En este ranking dos de los

Datasets	TF-IDF			Word Embedding		
	DNN	CNN	RNN	DNN	CNN	RNN
Sentiment140	0.76497407	0.76688544	0.56957939	0.78816761	0.80060849	0.82819948
Tweets Airline	0.85936944	0.85451457	0.82809226	0.8979309	0.90373439	0.90451624
Tweets SemEval	0.83674669	0.81377485	0.54857318	0.83674748	0.84313431	0.85172402
IMDB Movie Reviews (1)	0.85232000	0.82300000	0.56392000	0.84572000	0.86072000	0.87052000
IMDB Movie Reviews (2)	0.85512000	0.80628002	0.58724000	0.80252000	0.82624000	0.86688000
Cornell Movie Reviews	0.70437264	0.67867751	0.50787764	0.70221434	0.71365671	0.76693790
Book Reviews	0.75876443	0.72741509	0.5169437	0.74560455	0.76630924	0.73347052
Music Reviews	0.76850000	0.69200000	0.5170000	0.70800000	0.74450000	0.73100000

Figura 3.1: *Comparación del accuracy obtenido para datasets con dos clases (positivo y negativo), en el artículo Sentiment Analysis Based on Deep Learning: A Comparative Study.* ***Fuente:*** ***Dang et al. (2020)***

Dataset	TF-IDF			Word Embedding		
	DNN	CNN	RNN	DNN	CNN	RNN
Sentiment140	11 min 55 s	3 8min 17 s	**1h48 min 52 s**	4 min 18 s	7 min 3 s	**1 h 4 min 16 s**
Tweets Airline	1 min	34.41 s	1 h 54 s	30.66 s	1 min 22 s	2 min 41 s
Tweets SemEval	20.53 s	24.5 s	23 min 52 s	26.75 s	1 min 11 s	2 min 43 s
IMDB Movie Reviews (1)	1 min 11 s	1 min 7 s	**1 h25 min 48 s**	21.13 s	32.66 s	7 min 42 s
IMDB Movie Reviews (2)	17.78 s	22.05 s	30 min 21 s	31.32 s	36.81 s	8 min 23 s
Cornell Movie Reviews	23.2 s	16.83 s	31 min 55 s	12.9 s	21.26 s	4 min 40 s
Book Reviews	11.93 s	10.12 s	21 min 9 s	16.21 s	20.6 s	2 min17 s
Music Reviews	26.48 s	17.35 s	29 min 50 s	13.94 s	16.89 s	4 min 42 s

Figura 3.2: ***Comparación de los tiempos de cómputo experimentados para el entrenamiento de modelos con GPU, en el artículo Sentiment Analysis Based on Deep Learning: A Comparative Study. Fuente: Dang et al. (2020)***

primeros tres modelos están basados en una implementación de BERT descrita por Sun et al. (2019a) en su *paper* "*How to Fine-Tune BERT for Text Classification?*", y el modelo que presenta el mejor performance, XLNet Yang et al. (2019) es un método de preentrenamiento autorregresivo generalizado similar a BERT que permite aprender contextos bidireccionales mediante la maximización de la probabilidad esperada de todas las permutaciones del orden de factorización.

Modelo	Accuracy	Paper
XLNet Yang et al. (2019)	96.21	XLNet: Generalized Autoregressive Pretraining for Language Understanding
BERT_large+ITPT Sun et al. (2019a)	95.79	How to Fine-Tune BERT for Text Classification?
BERT_base+ITPT Sun et al. (2019a)	95.63	How to Fine-Tune BERT for Text Classification?

Tabla 3.1: ***Ranking de modelos según NLPProgress (http://nlpprogress.com/english/sentiment_analysis.html) en la resolución de la tarea de análisis de sentimiento. Fuente NLPProgress***

Y por último citamos el ranking de otro reconocido sitio web *Papers with Code* (https://paperswithcode.com/), el cual es un recurso de referencia para *papers* científicos que representan el más actual estado del arte en distintas casuísticas del *machine learning*.

La plataforma está compuesta de casi 5.000 benchmarks, 2.300 tareas y 50.000 *papers* que permiten comparar distintos enfoques en la resolución de distintos problemas.

Particularmente para el análisis de sentimiento sobre el dataset de IMDB, la plataforma compara distintas investigaciones rankeadas según el accuracy obtenido con este conjunto de datos. Es interesante observar que entre las 10 mejores soluciones encontramos 7 trabajos basados en Transformers y la mitad de ellos son modelos basados en BERT. (ver figura 3.3)

Rank	Model	Accuracy	Extra Training Data	Paper	Code	Result	Year	Tags
1	**NB-weighted-BON + dv-cosine**	97.4	✓	Sentiment Classification Using Document Embeddings Trained with Cosine Similarity			2019	
2	**XLNet**	96.21	✓	XLNet: Generalized Autoregressive Pretraining for Language Understanding			2019	Transformer
3	**EFL**	96.1	×	Entailment as Few-Shot Learner			2021	Transformer
4	**GraphStar**	96.0	✓	Graph Star Net for Generalized Multi-Task Learning			2019	
5	**BERT large finetune UDA**	95.8	✓	Unsupervised Data Augmentation for Consistency Training			2019	Transformer
6	**BERT_large+ITPT**	95.79	✓	How to Fine-Tune BERT for Text Classification?			2019	Transformer
7	**L MIXED**	95.68	✓	Revisiting LSTM Networks for Semi-Supervised Text Classification via Mixed Objective Function			2020	LSTM
8	**BERT_base+ITPT**	95.63	✓	How to Fine-Tune BERT for Text Classification?			2019	Transformer
9	**BERT large**	95.49	✓	Unsupervised Data Augmentation for Consistency Training			2019	Transformer
10	**ULMFiT**	95.4	✓	Universal Language Model Fine-tuning for Text Classification			2018	LSTM

Figura 3.3: ***Esta imagen muestra el ranking de modelos organizados por el accuracy obtenido en el problema de análisis de sentimiento sobre el dataset de reseñas de IMDB. En este ranking podemos ver 7 modelos basados en transformers y la mitad de los modelos en el top 10 son modelos basados en BERT. Fuente: https://paperswithcode.com/sota/sentiment-analysis-on-imdb***

3.1.4. Arquitectura de la solución

Como se mencionó anteriormente en la sección 1.3.2, la arquitectura de BERT ha sido diseñada para adaptarse a la resolución de distintos problemas a través de un leve ajuste. Este caso en particular se trata de un problema de clasificación de una frase en dos clases objetivos, en otras palabras, dada una frase clasificarla como sentimiento positivo o sentimiento negativo.

La forma de afrontar un problema de clasificación con BERT es en base al token es-

pecial [CLS]. Como indican los autores en el *paper* de BERT, “El primer token de cada secuencia es siempre un token de clasificación especial [CLS]. El estado oculto final correspondiente a este token se utiliza como representación de secuencia agregada para tareas de clasificación."(Devlin et al. (2018))

En la imagen 3.4 se describe la arquitectura general de la solución propuesta para este problema, donde podemos observar las 12 capas de Transformer que componen el modelo. Cada una de estas capas toma un *embedding* de tokens como entrada y produce la misma cantidad de *embeddings* en la salida. En la salida del último Transformer se construye un clasificador a través del primer *embedding* correspondiente al token [CLS].

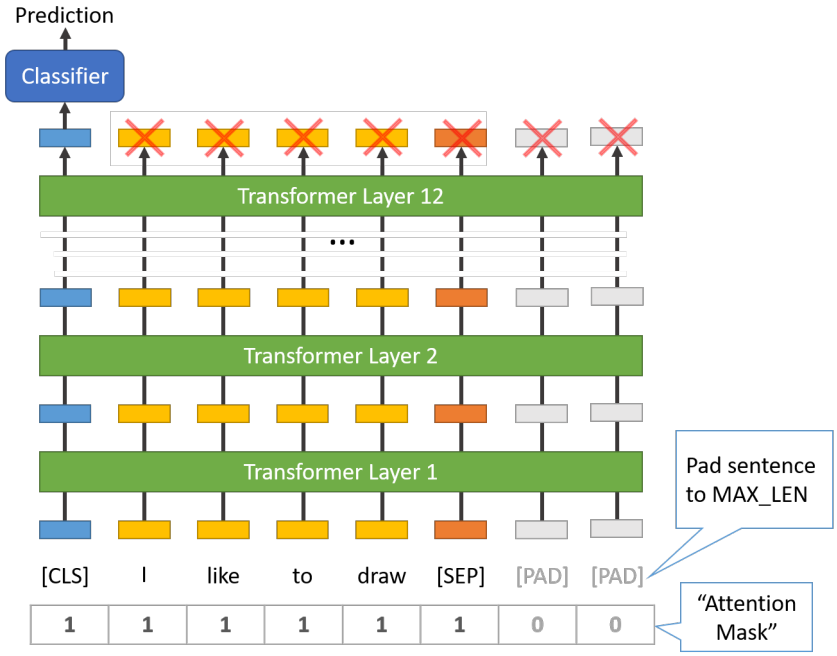


Figura 3.4: ***Adecuación de la arquitectura de BERT para la solución de problemas de clasificación basados en una frase de entrada. Fuente: Ryan (2019)***

3.1.5. Preparación de los datos

Durante el proceso de preparación de los datos se abordaron dos etapas. La primera de ellas de exploración y saneamiento de los datos. Durante esta primera etapa:

- Se identificó y valido que tal como indica la documentación Maas et al. (2011), el dataset está compuesto por 50000 registros bien balanceados (25000 datos etiquetados como positivos y 25000 datos etiquetados como negativos) como se pudo confirmar una vez fueron cargados los datos.

- Se pudo notar que ciertos registros se encontraban duplicados. Estos registros duplicados fueron removidos.
- Se creó y aplicó una función de limpieza que permite para cada una de las reseñas limpiar algunos elementos que podrían causar ruido y carecen de valor en el análisis y entrenamiento del modelo como las etiquetas HTML, texto entre corchetes y algunos signos de puntuación.

La segunda etapa se basó en un proceso de normalización de los datos de entrada para BERT. Al tratarse BERT de un modelo preentrenado, este espera que los datos de entrada tengan un formato especifico, tal como se indica en la sección 1.3.3. La normalización de los datos de entrada se basa en la construcción adecuada de los *embeddings* para que el modelo pueda entender cada una de las sentencias de entrada, esto consiste en:

1. Como se describe en la arquitectura de la solución (ver figura 3.4), el modelo de BERT tiene dos restricciones:

   a) Todas las sentencias de entrada deben tener la misma longitud, por ese motivo las sentencias son truncadas o rellenadas con un token especial denominado [PAD]

   b) La longitud máxima de la sentencia de entrada es de 512 tokens.

   Al tener sentencias de entrada de longitud variable se optó en este caso por truncar los datos de entrada a una longitud máxima de 512 tokens. En otras palabras estamos basando el entrenamiento en los primeros 512 tokens de cada reseña, desechando el resto de la información.

2. Añadir sentencias de entrada que tengan agregados los tokens especiales al comienzo y al final de cada sentencia.

   a) El token [SEP] es un token especial que se utiliza para demarcar el fin de una sentencia o la separación entre dos sentencias. En este caso se utiliza como delimitador final de la única sentencia de entrada.

   b) El token [CLS] se debe anteponer a cada sentencia de entrada. Este token es usado en las tareas de clasificación, pero indistintamente del problema a resolver el modelo BERT siempre espera recibirlo como el comienzo de la entrada. La importancia de este token radica en que actúa como una representación agregada para las atareas de clasificación. La última capa oculta de este token se usa como una representación de la sentencia para la clasificación de secuencias.

3. Agregar dos embeddings de información para el modelo: Al embedding de tokens producto del tokenizador de BERT (Token IDs), se deben agregar dos *embeddings* adicionales:

   a) Mask IDs o máscara de atención que diferencia cuales elementos de la secuencia son tokens y cuales son tokens de relleno o padding. Se trata simplemente una matriz de 1s y 0s que indica qué tokens están rellenando y cuáles no. Esta máscara le dice al mecanismo de auto-atención en BERT que no incorpore estos tokens [PAD] en su interpretación de la sentencia.

   b) Segment IDs usado para distinguir diferentes sentencias. En este caso en el que tenemos una sola sentencia se trata de un vector de 0s.

Adicionalmente es importante mencionar que existen técnicas y trabajos enfocados en cómo manejar la limitación que presenta BERT en cuanto a la extinción de los tokens de la secuencia de entrada.

Existen tres enfoques distintos basados en el costo computacional:

- **Bajo costo computacional:** Seleccionar una parte del texto original. Los ejemplos incluyen elegir los primeros n tokens, los últimos n tokens o compilar una nueva instancia de texto a partir del principio y el final de la instancia de texto original.
- **Costo computacional medio a alto:** Usar modelos de Transformers recientes (como Longformer) que tienen un límite de 4096 tokens en lugar de 512. Estos nuevos métodos tienen mecanismos de atención modificados que reducen el costo computacional.
- **Alto costo computacional:** Dividir la instancia de texto en fragmentos que se ajusten a un modelo como BERT con un límite estándar de 512 tokens por instancia e implementar el modelo en cada parte por separado, luego unir las representaciones vectoriales resultantes.

En el artículo de Fiok et al. (2021) además de proponer un nuevo método llamado Text Guide, que es un método de bajo coste computacional el cual pretende hacer un truncamiento mas inteligente de la instancia de entrada.

Por último mencionar que Sun et al. (2019a) en su *paper* hace una comparación usando distintas técnicas para truncar el texto, tomando en algunos casos el principio, en otros casos el final y en algunos casos haciendo combinaciones de distintos segmentos del texto. De este trabajo igual podemos concluir que la diferencia del performance del modelo no es significativa entre las distintas técnicas.

Por las razones antes expuestas y entendiendo que el m´todo elegido no impactará mucho el performance del modelo, para este trabajo se decidió utilizar un truncamiento simple tomando los primeros 512 tokens de la sentencia de entrada.

3.1.6. Modelo y Fine Tuning

El modelo de BERT se obtuvo desde Tensorflow Hub (https://www.tensorflow.org/hub). TensorFlow Hub es un repositorio de modelos de aprendizaje automático pre-entrenados, listos para poder implementar un fine-tunning.

Se usó como modelo BERT preentrenado la versión bert_multi_cased_L-12_H-768_A-12. Para hacer el clasificador se optó por usar el API Keras de Tensorflow tomando las salidas del modelo de BERT, específicamente el pooled_output (embedding del token [CLS]) y creando una capa única para la tarea de clasificación, para ayudar a prevenir el sobre-ajuste se hizo un droput de 10 %. En esta última capa, se aplica la función softmax para obtener las probabilidades predichas del modelo entrenado.

Para entrenar el modelo propuesto se utilizo como valores de los hiperparámetros los recomendados en el *paper* de BERT. Según los autores del *paper* BERT no necesita muchas epocas para ser entrenado y recomiendan que sean entre 2 y 4. Devlin et al. (2018):

- Tamaño del lote = 8
- Tasa de aprendizaje = 2e-5
- Longitud máxima de secuencia = 512
- Épocas = 3

El entrenamiento se realizó usando Google Colab y utilizando la GPU (Graphic Processing Unit). La duración del entrenamiento ha sido:

- Primera Época = 3368 segundos.
- Segunda Época = 3373 segundos.
- Tercera Época = 3372 segundos.
- Entrenamiento total = 10113 segundos, aproximadamente 2 horas y 50 minutos.

El código fuente del entrenamiento se encuentra en el repositorio Github https://github.com/lsizaguirre/09MIAR-TFM

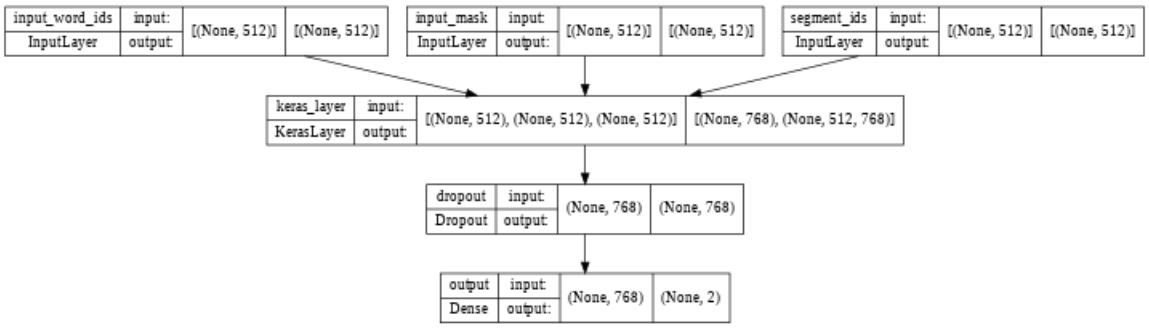


Figura 3.5: ***Modelo de BERT propuesto para la solución de este problema de clasificación. En el modelo se observan como entrada las tres embeddings que la conforman, el token embedding, el de segmentación y el posicional, estos van directamente a la capa preentrenada de BERT, luego se agrega una capa de dropout y una última capa que recibe el dropout del embedding del token [CLS] y como salida tiene solo dos neuronas, una para cada clase a predecir. Fuente: Imagen propia***

3.1.7. Resultados y Conclusiones.

Podemos observar que con una configuración de red muy sencilla y con los parámetros recomendados para el fine-tunning se obtienen resultados bastante buenos y cercanos a los que definen el estado del arte. Se podría intentar hacer un ajuste más sofisticado y con ello obtener mejores resultados.

Las siguientes imágenes muestran algunas de las métricas obtenidas durante la evaluación del experimento.

```
Loss: 0.4330078959465027
Accuracy: 0.9225572347640991
```

Figura 3.6: ***Accuracy y Loss obtenidos con el modelo. Fuente: Imagen propia***

	precision	recall	f1-score	support
0	0.92	0.92	0.92	4939
1	0.92	0.92	0.92	4978
accuracy			0.92	9917
macro avg	0.92	0.92	0.92	9917
weighted avg	0.92	0.92	0.92	9917

Figura 3.7: ***Métricas de clasificación obtenidas en el modelo. Fuente: Imagen propia***

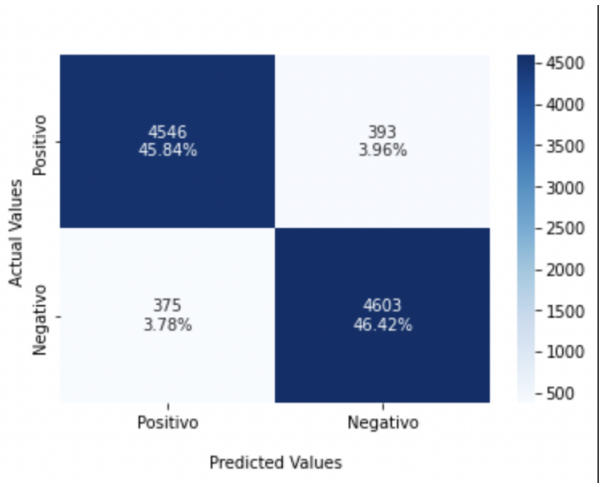


Figura 3.8: ***Matriz de confusión resultado de la evaluación del modelo. Fuente: Imagen propia***

3.2. Problema de Respuesta a Preguntas

3.2.1. Descripción del problema

Vivimos en una época donde la actividad digital forma parte de nuestro día a día. Todos los días generamos una cantidad enorme de datos e información, interactuando a través de redes sociales, trabajando conectados a Internet, etc. A medida que aumenta la información, se vuelve más difícil recuperar una pieza de información relevante de manera eficiente. En este sentido, la tarea de Respuesta a Preguntas o *Question Answering* (QA, por sus siglas en inglés) es un campo de la ciencia de la computación y el NLP que tiene como objetivo crear sistemas que puedan responder de forma automática preguntas en lenguaje natural.

Los sistemas QA intentan encontrar automáticamente la respuesta contextual y semánticamente correcta para una pregunta proporcionada en un texto. En general, los tres componentes asociados con los sistemas QA son:

1. Clasificación de preguntas
2. Recuperación de información
3. Extracción/generación de respuestas

En este trabajo se usará BERT como modelo base para resolver este problema. La idea en este caso es que se le dará un párrafo al modelo de BERT que contendrá información acerca de un tema, y vamos a buscar respuestas con respecto a preguntas relacionadas con ese tema. Para ello, se generarán preguntas cuyas respuestas se encuentren dentro de los textos del párrafo.

3.2.2. Selección del Dataset

Para este experimento se usó **The Stanford Question Answering Dataset (SQuAD)**. Este es un conjunto de datos de respuesta a preguntas publicado en el año 2016 por investigadores de la Universidad de Stanford, junto al *paper* "*SQuAD: 100000+ Questions for Machine Comprehension of Text*" (Rajpurkar et al., 2016)

Existe una segunda versión de SQuAD (v2.0) publicada por los mismos investigadores en el año 2018 en el *paper* "*Know What You Don't Know: Unanswerable Questions for SQuAD*" (Rajpurkar et al., 2018). Esta versión se diferencia de la primera en que en algunas oportunidades la respuesta no se encuentra en el texto original y está enfocada a ser

utilizada en soluciones donde el modelo debe fabricar la respuesta de algún modo. Por simplicidad en este trabajó se utilizó la versión 1.1 de este conjunto de datos.

Este conjunto de datos contiene un total de 107785 preguntas extraídas de un conjunto de 536 artículos seleccionados de la Wikipedia, en un amplio rango de tópicos. Los autores del *paper* mencionan que este conjunto de datos se diferencia a los que se habían publicado con anterioridad en que las respuestas son de formato libre y no están acotadas a un conjunto de opciones lo que hace el problema más interpretativo y con una cantidad bastante grande de opciones.

Otro aspecto relevante es que los archivos que componen este conjunto de datos se encuentran en *formato .json* y vienen en las siguientes particiones, generadas aleatoriamente:

- **train-v1.1.json:** Es un *archivo .json* que contiene el 80 % de los datos y contiene el conjunto de entrenamiento o training set. Es el subconjunto del dataset donde están almacenadas las secuencias que se utilizarán para hacer el entrenamiento.
- **dev-v1.1.json:** Es un *archivo .json* que contiene el 10 % de los datos y contiene el conjunto de validación o validation test. Es el subconjunto de datos que se utilizarán para hacer la evaluación.
- El 10 % restante de registros son usados para la validación final usada en el benchmark y este conjunto de datos no lo hacen disponible. testing set.

3.2.3. Estado del arte

Hay dos métricas dominantes utilizadas por muchos conjuntos de datos de respuesta a preguntas, incluido SQuAD: la coincidencia exacta (EM) y puntaje F1. Estos puntajes se calculan en pares individuales de pregunta y respuesta. Cuando son posibles varias respuestas correctas para una pregunta determinada, se calcula la puntuación máxima sobre todas las posibles respuestas correctas. Las puntuaciones generales de EM y F1 se calculan para un modelo promediando las puntuaciones de los ejemplos individuales.

- **Coincidencia exacta o Exact Match (EM, por sus siglas en inglés):** esta métrica es tan simple como parece. Para cada par de pregunta y respuesta, si los caracteres de la predicción del modelo coinciden exactamente con los caracteres de (una de) las respuestas verdaderas, EM = 1; de lo contrario, EM = 0. Esta es una métrica estricta de todo o nada, estar equivocado por un solo carácter da como resultado una puntuación de 0. Al evaluar contra un ejemplo negativo, si el modelo predice cualquier texto, automáticamente recibe un 0 para ese ejemplo.

- **La puntuación F1 o F1 score (en inglés):** es una métrica común para los problemas de clasificación y se usa ampliamente en la tarea de QA. Es apropiado cuando nos preocupamos por igual por la precisión y la memoria. En este caso, se calcula sobre las palabras individuales de la predicción frente a las de la respuesta verdadera. La cantidad de palabras compartidas entre la predicción y la verdad es la base de la puntuación F1: la precisión es la relación entre la cantidad de palabras compartidas y la cantidad total de palabras en la predicción, y el recuerdo o memoria es la relación entre la cantidad de palabras compartidas. al número total de palabras en la verdad fundamental.

Es indudable que BERT ha provocado un avance considerable en el estado del arte de varias tareas de NLP. Específicamente en la tarea de QA podemos ver tanto en el leaderboard de SQuAD como en el de otros conjuntos de datos similares como CoQA (Reddy et al., 2018) que la mayoría de modelos que lideran están basados en BERT.

	Long Answer Dev			Long Answer Test			Short Answer Dev			Short Answer Test		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
DocumentQA	47.5	44.7	46.1	48.9	43.3	45.7	38.6	33.2	35.7	40.6	31.0	35.1
DecAtt + DocReader	52.7	57.0	54.8	54.3	55.7	55.0	34.3	28.9	31.4	31.9	31.1	31.5
$\text{BERT}_{\text{joint}}$ (this work)	**61.3**	**68.4**	**64.7**	**64.1**	**68.3**	**66.2**	**59.5**	**47.3**	**52.7**	**63.8**	**44.0**	**52.1**
Single Human	80.4	67.6	73.4	-	-	-	63.4	52.6	57.5	-	-	-
Super-annotator	90.0	84.6	87.2	-	-	-	79.1	72.6	75.7	-	-	-

Figura 3.9: ***En este estudio Alberti et al. (2019) comparan el desempeño de BERT contra un humano común y un humano experto. Se evidencia el mejor desempeño de BERT en ciertos casos. Fuente: Alberti et al. (2019)***

Para evaluar el estado del arte relacionado a la solución de preguntas y respuestas sobre el dataset SQuAD en su versión 1.0 se utilizaron diversas fuentes.

En el *paper* realizado por Alberti et al. (2019) titulado "*A BERT Baseline for the Natural Questions*" los autores concluyen que BERT debe ser la nueva línea base y que constituye un buen punto de partida para los modelos de preguntas y respuestas y *datasets* con características similares. En este mismo estudio los autores comparan el desempeño de BERT con otros modelos que resolvían la tarea en ese momento y además lo compararon con el desempeño de la tarea realizada por un humano común y un humano experto. (ver figura 3.9). Los autores concluyen que los resultados obtenidos por los modelos de respuesta a preguntas basados en BERT también se están acercando rápidamente al rendimiento humano informado para estos conjuntos de datos.

Existen otros estudios, como el presentado en el *paper* "*Comparative Study of Machine Learning Models and BERT on SQuAD*" de Patel et al. (2020) donde realizan un análisis comparativo del rendimiento de ciertos modelos populares en el aprendizaje automático y

el modelo de BERT sobre SQuAD, brindando BERT una mayor precisión que otros modelos.

En el ranking de la página oficial de SQuAD (https://rajpurkar.github.io/SQuAD-explorer/) podemos observar muchos modelos basados en ALBERT Lan et al. (2019). ALBERT es una versión de BERT mucho más ligera con 12 millones de parámetros en vez de 110 millones como los que contiene BERT. En el *paper* original de ALBERT llamado "*ALBERT: A Lite BERT for Self-supervised Learning of Language Representations*" comparan el desempeño de ALBERT con BERT en las tareas de SQuAD obteniendo resultados bastante similares (ver figura 3.10)

Model		Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg	Speedup
BERT	base	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3	4.7x
	large	334M	92.2/85.5	85.0/82.2	86.6	93.0	73.9	85.2	1.0
ALBERT	base	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1	5.6x
	large	18M	90.6/83.9	82.3/79.4	83.5	91.7	68.5	82.4	1.7x
	xlarge	60M	92.5/86.1	86.1/83.1	86.4	92.4	74.8	85.5	0.6x
	xxlarge	235M	**94.1/88.3**	**88.1/85.1**	**88.0**	**95.2**	**82.3**	**88.7**	0.3x

Figura 3.10: ***Comparación de los resultados obtenidos en distintas tareas de NLP entre BERT y su variante ALBERT. Para las tareas SQuAD los dos números equivalen a F1 y EM. Fuente: Lan et al. (2019)***

3.2.4. Arquitectura de la solución

Como ya se mencionó en la sección 1.3.2, la arquitectura de BERT ha sido diseñada para adaptarse a la resolución de distintos problemas a través de un leve ajuste, por este motivo es que en BERT tenemos dos formas diferentes de manejar la entrada y la salida del modelo.

Con respecto a la entrada, una de las formas es cuando se tiene un vector que se supone representa toda una frase en su completitud y es básicamente la forma que se utiliza en las tareas de clasificación. La segunda forma de entrada es una lista de vectores, cada uno de los cuales es una representación de las palabras en el contexto que las rodea. Esta segunda forma de entrada es particularmente la que se utiliza en este tipo de problemas al no tratarse de un problema de clasificación como en el experimento anterior.

Como entrada se definieron dos frases, la primera frase representa el texto en el que hay que buscar la información. La segunda frase representa la pregunta que se quiere resolver. En la primera frase, lo que se necesita es buscar dónde empieza y dónde termina la hipotética respuesta a nivel de tokens y ese nivel de tokens luego es traducido a nivel de palabras. De esa forma se busca la respuesta dentro de la primera oración.

Con respecto a la salida, para este caso no se usa el *Pooled Output*, el cual es una representación del token de clasificación [CLS], un embedding de toda la frase. Por el

contrario, se usa el *Sequence Output*, el cual está representado por una lista de vectores (embedding), uno para cada una de las palabras, de esta forma se podrá localizar las más probables de ser la el inicio y el final de la respuesta.

Casi siempre que se utiliza BERT para generar la solución de una tarea, lo único que hay que hacer es añadir una capa densa al final antes de hacer ningún ajuste de los parámetros. Estas capas densas se le aplica a cada uno de los elementos, a cada vector y nos va a dar dos posibles respuestas, dos neuronas en la capa de salida.

La primera nos representará una puntuación de qué tan probable es que esa palabra sea la posición inicial en la que empieza la respuesta y la segunda neurona representará una puntuación de qué tan probable es que esa palabra sea la la que finaliza la respuesta De esa forma, simplemente aplicando una capa densa con dos neuronas a cada vector, a cada representación vectorial de las palabras, obtendremos un puntaje de qué tan susceptible es de ser la palabra que inicia o que finaliza la respuesta a la pregunta que estamos buscando.

Finalmente se debe construir un par de listas de todos los puntajes de qué tan probable es que sean el inicio o el final de la respuesta para cada una de las palabras. Una lista para el inicio y otro para el final. La respuesta será la frase o secuencia de palabras contenidas entre el token que tenga el mayor puntaje o probabilidad de ser el inicio de la respuesta y el token con mayor puntaje de ser el final de la respuesta.

Como el proceso de tokenización de BERT tiende a romper palabras, estas se tendrán que reconstruir correctamente.

3.2.5. Preparación de los datos

La entrada del modelo BERT para problemas de *question answering* está compuesta por dos frases, la pregunta y el contexto. La forma de manejar la entrada es a través de la estructura mostrada en la figura 3.11, donde podemos ver:

1. Al igual que el experimento anterior y en todos los problemas donde usemos BERT, el token [CLS] se debe anteponer a cada sentencia de entrada. Este token es usado en las tareas de clasificación, pero indistintamente del problema a resolver el modelo BERT siempre espera recibirlo como el comienzo de la entrada.

2. Seguido al token [CLS] se colocarán los tokens correspondientes a las palabras que componen la pregunta.

3. A esto le procederá el token especial [SEP] que en este caso tiene un significado distinto dentro de la entrada al experimento anterior. La idea en este caso es que el

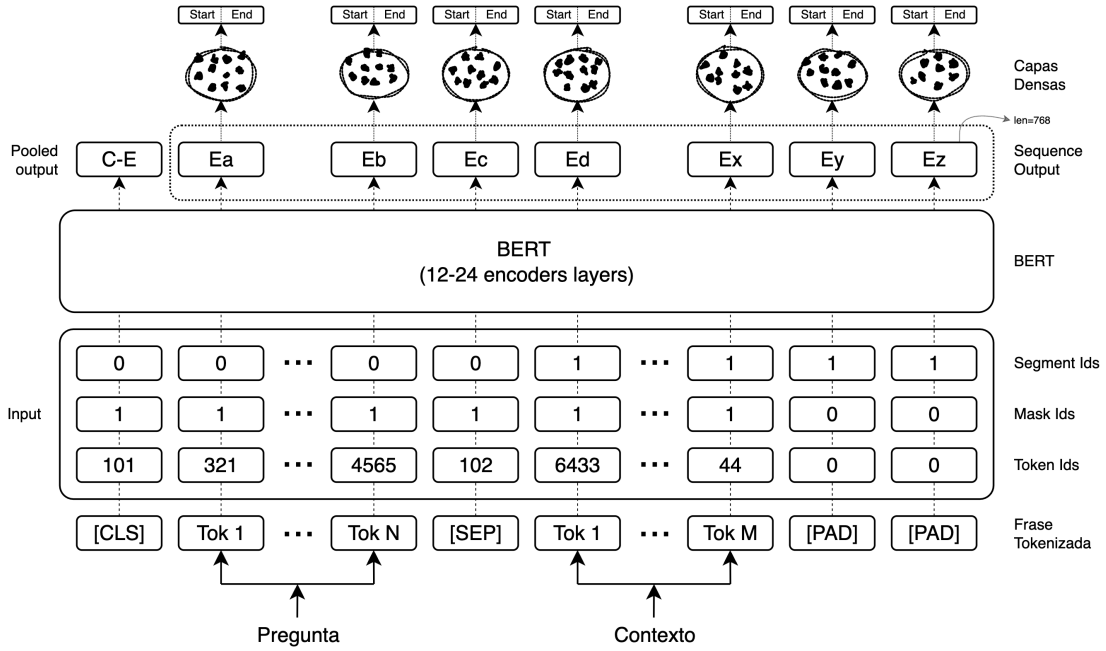


Figura 3.11: ***Adecuación de la arquitectura de BERT para la solución de problemas de preguntas y respuestas. Fuente: Imagen propia.***

token [SEP] separará la pregunta del contexto.

4. En el segment embedding se debe marcar desde el principio de la frase de entrada hasta el token [SEP] como el segmento inicial.

5. En el resto del token embedding se colocarán los tokens que correspondan al contexto o referencia.

6. Al igual que en el experimento anterior BERT presenta algunas restricciones con respecto a la entrada:

   a) Todas las sentencias de entrada deben tener la misma longitud.

   b) La longitud máxima de la sentencia de entrada es de 384 tokens. Esta longitud es un estándar en el conjunto de datos de SQuAD por dos razones, tener la mayoría de las preguntas y contextos tokenizados sin ningún truncamiento y mantener una longitud limitada para acelerar el entrenamiento.

A diferencia del problema anterior (ver sección 3.1), el conjunto de datos asociado a este problema se encuentra en un *archivo .json* que necesita ser preprocesado. Sin embargo, esta tarea de preprocesado de datos es una tarea que se simplifica considerablemente comparado con el proceso del experimento anterior debido a un conjunto de funciones que provee Google para trabajar con el conjunto de datos de SQuAD que ya se encuentran escritas y disponibles en su librería Official.nlp.bert.

Lo que se logra con estas librerías es transformar los datos de entrenamiento de SQuAD a datos utilizables por las librerías de Tensorflow. Para ello se utilizará la función ***generate_tf_record_from_json_file***, dejando escrita, su salida, en un archivo llamado ***train_meta_data***, divido en cuatro partes. Este archivo ya estará listo para ser utilizado, también se inicializa el tamaño del batch para el entrenamiento, en este caso será de tamaño 4.

Una vez dado este paso ya se obtienen los datos "tokenizados", luego a través de la función ***create_squad_dataset*** y a partir de los archivos mencionados con anterioridad, se crea el *dataset* de entrenamiento. De esa forma ya se tendrán listos los datos para entrenar el modelo.

3.2.6. Modelo y Fine Tuning

Al igual que el experimento anterior, el modelo de BERT se obtuvo desde Tensorflow Hub (https://www.tensorflow.org/hub). Se usó como modelo BERT preentrenado la versión bert_en_uncased_L-12_H-768_A-12 al principio se usó la versión bert_multi_cased_L-12_H-768_A-12, pensando que se obtendrían mejores resultados usando entradas en distintos idiomas, pero los resultados no fueron buenos.

Adicionalmente y haciendo uso del API Keras de Tensorflow se creó una "capa SQuAD", esta es una capa compuesta por redes neuronales densamente conectadas, la cual será entrenada para devolver dos valores numéricos.

Estos valores representarán dos posiciones en el contexto, que determinarán de dónde se obtendrán el resultado, una vez obtenido el resultado. El resultado calculado se podrá comparar con el resultado esperado y, de esta forma, en base a la evaluación del error cometido, empezar el aprendizaje de las neuronas de la red densamente conectada.

Como recomienda Tensorflow en su página oficial (https://www.tensorflow.org/text/tutorials/fine_tune_bert), la función de perdida se implementó a través del método ***sparse_categorical_crossentropy*** de ***tf.keras.backend***.

Para entrenar el modelo propuesto se utilizó como valores de los hiperparámetros los recomendados en el *paper* de BERT. Según los autores del *paper* BERT no necesita muchas épocas para ser entrenado y recomiendan que sean entre 2 y 4. Devlin et al. (2018):

- Tamaño del entrenamiento = 88641
- Número de lotes = 3000
- Tamaño del lote = 4
- Tasa de aprendizaje = 2e-5

- Longitud máxima de secuencia = 512
- Épocas = 3

El entrenamiento se realizó usando Google Colab y utilizando la Unidad de Procesamiento Tensorial o *Tensor Processing Unit* (TPU, por sus siglas en inglés). La duración del entrenamiento ha sido:

- Primera Época = 12863 segundos.
- Segunda Época = 10381 segundos.
- Tercera Época = 11303 segundos.
- Entrenamiento total = 34537 segundos, aproximadamente 9 horas y 40 minutos.

El código fuente del entrenamiento se encuentra en el repositorio Github https://github.com/lsizaguirre/09MIAR-TFM

3.2.7. Resultados y Conclusiones.

Este experimento es distinto al anterior en cuanto a la fase de evaluación de los resultados debido a que no se puede hacer de forma directa como en el problema de análisis de sentimiento.

Para evaluar los modelos, primero se obtienen las predicciones haciendo uso también de un conjunto de funciones que Google pone a disposición para tal fin y estas se usan como entrada para un script de evaluación que se encuentra en la página oficial del *dataset* SQuAD (https://rajpurkar.github.io/SQuAD-explorer/).

Este *script* de evaluación en *Python* nos permite evaluar las predicciones bajo las mismas condiciones que son aplicadas en el *paper* de SQuAD (como remover los artículos y signos de puntuación de las frases) y obtener las métricas de Exact Match (EM) y el score F1.

En este caso realizamos dos entrenamientos:

1. **Primer entrenamiento:**
   Donde se uso como modelo de BERT la versión bert_multi_cased_L-12_H-768_A-12, versión base multilenguaje. Con este modelo obtuvimos las siguientes métricas:
   - Exact Match (EM) = 29,33 %
   - F1 Score = 41 %

2. **Segundo entrenamiento:**
   Donde se mantuvieron todos los hiperparámetros con los mismos valores pero se entrenó con va versión bert_en_uncased_L-12_H-768_A-12

   - Exact Match (EM) = 71,34 %
   - F1 Score = 80,58 %

Con respecto a los resultados podemos deducir:

- En el primer entrenamiento no se obtuvieron buenos resultados debido a que no seleccionamos el mejor modelo de BERT con respecto a los datos de entrenamiento. Se eligió trabajar con un modelo multilenguaje cuando en los datos de entrenamiento hay solo frases en ingles, además usamos un modelo entrenado con minúsculas y mayúsculas cuando el script de evaluación transforma cada frase a minúscula en el proceso de evaluación.
- En el segundo entrenamiento haciendo uso de un modelo adecuado los resultados fueron bastantes satisfactorios, acercándose a los estándares que definen el estado del arte y muy similares a las capacidades humanas.

Resultados y Discusión

4

Los resultados obtenidos en ambos experimentos realizados en este trabajo y descritos en la sección 3, sugieren que BERT es una línea de base sólida para el desarrollo de soluciones a distintas tareas de procesamiento del lenguaje natural. Las soluciones que fueron propuestas, tanto al problema de clasificación como en el problema de respuesta a preguntas, se basaron en ligeros ajustes al modelo preentrenado obteniendo en ambos casos resultados muy cercanos a los que representan el estado del arte de estos problemas.

Esta capacidad de adaptación de BERT y la obtención de buenos resultados en distintos problemas es también confirmada por otros autores en distintos estudios. Como en el trabajo realizado por Ghavidel. et al. (2020), quienes compararon el desempeño de BERT con otro modelo similar (XLNet) en un problema de preguntas y respuestas.

Topal et al. (2021), compararon 3 modelos basados en transformers, BERT, XLNet y GPT-3. En el trabajo mencionan las implicaciones significativas que han tenido todos estos modelos en el campo del procesamiento natural del lenguaje y de los buenos resultados que se obtienen a partir de ellos incluso cuando no son ajustados para propósitos específicos.

Es importante por último resaltar que los ajustes realizados en cada uno de los experimentos fueron mínimos y que para el problema de respuesta a preguntas no se usó la totalidad del conjunto de entrenamiento por el costo de procesamiento. A pesar de haber obtenido buenos resultados, la oportunidad de mejora es amplia.

Conclusiones

5

En este trabajo se evaluó el uso y se comparó el desempeño de BERT en la resolución de dos tareas, el análisis de sentimiento y el problema de respuesta a preguntas. En base a al estudio realizado y a los experimentos desarrollados, se rinden las siguientes conclusiones:

1. La arquitectura unificada de BERT le permite al modelo poder adaptarse fácilmente a la solución de distintos problemas en el ámbito del procesamiento del lenguaje natural, obteniendo resultados cercanos a los que define el estado del arte para esos problemas.

2. Resulta más fácil, rápido y eficiente usar un modelo preentrenado en un gran volumen de datos como punto de partida para la construcción de soluciones a tareas de NLP. En el caso de BERT, este ha sido entrenado con el corpus de Wikipedia y libros de Google Books, en su versión base tiene 110 millones de parámetros, lo que lo hace un modelo muy grande. Adaptar el modelo es mucho más fácil y barato que entrenarlo from scratch (desde cero).

3. Para que BERT funcione de forma adecuada se debe hacer un preprocesamiento del conjunto de datos adaptándolo a lo que el modelo espera recibir como entrada. Este preprocesamiento consiste en transformar las frases a un vector de token (token embeddings) y añadir ciertos metadatos para marcar el principio y el final de las oraciones, así como para distinguir las diferentes oraciones o la posición de las palabras en una oración.

4. Es importante considerar el proceso de limpieza de los datos para obtener un buen resultado. En tal sentido, el detectar, corregir y eliminar registros imprecisos o que no son de utilidad para el modelo, genera un mejor rendimiento y un impacto positivo en la solución.

5. Sin profundizar mucho en el desarrollo de la arquitectura de las capas densas que

se crean en los modelos de BERT para hacer el proceso de ajuste, se pueden conseguir buenos resultados. Se podría siempre optimizar el modelo y profundizar en la complejidad de estas últimas capas para mejorar el rendimiento, pero con configuraciones y ajustes básicos es suficiente para conseguir un rendimiento aceptable.

6. Existen diferentes versiones de BERT. Es importante seleccionar una versión apropiada entre BASE o LARGE, según las capacidades de computo y una versión multilenguaje o de lenguaje especifico según los datos de entrenamiento con los que se cuente. Si utilizamos un modelo multilenguaje y nuestros datos no lo son, no se obtendrá un buen rendimiento.

Limitaciones y Perspectivas de Futuro

6

1. **Escalar el enfoque:** El enfoque de este trabajo se realizó sobre BERT, pero a la fecha existen otros modelos de lenguaje preentrenados basados en la arquitectura de transformers que están teniendo mucha relevancia en el campo del procesamiento del lenguaje natural como XLNet, T5 o GPT-3. En el mismo sentido, existen distintas variaciones de BERT (tabla 1.1) que también están teniendo excelentes resultados en la resolución de problemas específicos. Algunas de estas variaciones prometen mejoras, otras simplifican y hacen el entrenamiento aún más liviano. Se recomienda explorar otros modelos y variaciones de BERT.

   Adicionalmente en este trabajo se desarrollan dos aplicaciones que resuelven dos tareas bien concretas y especificas dentro del NLP, el análisis de sentimiento a través de un problema de clasificación y el problema de preguntas y respuestas. Se recomienda ampliar el estudio con otras tareas.

2. **Mejorar fine-tunning:** Las soluciones planteadas en este trabajo se basan en un fine-tunning muy simple. Es muy probable que se puedan realizar mejoras sustanciales a los resultados obtenidos utilizando técnicas de fine-tunning más complejas, como las exploradas en el paper “Universal Language Model Fine-tuning for Text Classification” (Howard y Ruder, 2018), donde plantean un método de transfer learning efectivo que se puede aplicar a cualquier tarea en NLP, e introduce técnicas que son clave para obtener mejores resultados en el proceso de fine-tunning para un modelo de lenguaje.

3. **Capacidades de cómputo:** La simplicidad de las soluciones propuestas se debe en gran parte a las capacidades de cómputo disponible para la implementación. El entrenamiento y fine-tunning de los modelos propuestos se realizó utilizando la versión gratuita de Google Colab (`https://colab.research.google.com/`) bajo un entorno donde los recursos no están garantizados y son limitados. Por este motivo se optó por usar la versión BERT$_{BASE}$ y no la versión BERT$_{LARGE}$ (ver 1.3.2) en las soluciones propuestas.

Lista de Acrónimos

CNN Redes Neuronales Convolucionales o *Convolutional Neural Networks*.

DL Aprendizaje Profundo o *Deep Learning*.

DNN Redes Neuronales Profundas o *Deep Neural Networks*.

FFNN Redes Neuronales Prealimentadas o *Feed Forward Neural Networks*.

GPU Unidad de Procesamiento Gráfico o *Graphic Processing Unit*.

HTML Lenguaje de Marcas de Hipertexto o *HyperText Markup Language*.

LSTM Redes de Gran Memoria a Corto Plazo o *Long Short-Term Memory*.

ML Aprendizaje Automático o *Machine Learning*.

NLP Procesamiento del Lenguaje Natural o *Natural Language Processing*.

RNN Redes Neuronales Recurrentes o *Recurrent Neural Networks*.

SQuAD The Stanford Question Answering Dataset.

TPU Unidad de Procesamiento Tensorial o *Tensor Processing Unit*.

Glosario

Archivo JSON Notación de Objeto Javascript o JavaScript Object Notation (JSON) es un formato de archivo sencillo para el intercambio de información. El formato JSON permite representar estructuras de datos (arrays) y objetos (arrays asociativos) en forma de texto.

Computer Vision Disciplina científica que incluye métodos para adquirir, procesar, analizar y comprender las imágenes del mundo real con el fin de producir información numérica o simbólica para que puedan ser tratados por un ordenador..

softmax La función Softmax es una función de activación que transforma las salidas a una representación en forma de probabilidades, de tal manera que el sumatorio de todas las probabilidades de las salidas de 1. Devuelve la distribución de probabilidad de cada una de las clases soportadas en el modelo.

TF-IDF TF-IDF (del inglés Term frequency – Inverse document frequency), frecuencia de término – frecuencia inversa de documento, es una medida numérica que expresa cuán relevante es una palabra para un documento en una colección, calculada a través de la frecuencia de ocurrencia del término en la colección de documentos.

Transformer son una clase reciente de redes neuronales para datos de tipo secuenciales, basadas en la autoatención, que han demostrado estar bien adaptadas al texto y actualmente están impulsando importantes avances en el procesamiento del lenguaje natural.

Word Embeddings es una técnica del procesamiento del lenguaje natural que consiste, básicamente, en asignar un vector a cada palabra. Este vector guarda información semántica, lo que permite que pueda ser asociado o disociado a otros vectores (palabras) según distintos contextos gramaticales.

Herramientas de Desarrollo y Código Fuente

1. Se utilizó Python como lenguaje de programación. Especificamente los exprimentos se desarrollaron en Notebooks de Python.

2. Para el desarrollo de los experimentos propuestos en este trabajo se utilizó Google Colab (https://colab.research.google.com/) como plataforma de desarrollo.

3. Adicionalmente, se obtuvieron los modelo preentrenados de BERT a traves de Tensorflow Hub (https://www.tensorflow.org/hub)

4. El control de versiones del código fuente se hizo a través de Github, donde se encuentran los notebooks de este proyecto. Repositorio: https://github.com/lsizaguirre/09MIAR-TFM

Bibliografía

Alberti, C., Lee, K., y Collins, M. (2019). A bert baseline for the natural questions.

Bhavitha, B. K., Rodrigues, A. P., y Chiplunkar, N. N. (2017). Comparative study of machine learning techniques in sentimental analysis. In *2017 International Conference on Inventive Communication and Computational Technologies (ICICCT)*, pages 216–221.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., y Amodei, D. (2020). Language models are few-shot learners.

Dang, N. C., Moreno-García, M. N., y la Prieta, F. D. (2020). Sentiment analysis based on deep learning: A comparative study. *Electronics*, 9(3):483.

Devlin, J., Chang, M.-W., Lee, K., y Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding.

Fiok, K., Karwowski, W., Gutierrez-Franco, E., Davahli, M. R., Wilamowski, M., Ahram, T., Al-Juaid, A., y Zurada, J. (2021). Text guide: Improving the quality of long text classification by a text selection method based on feature importance. *IEEE Access*, 9:105439–105450.

Ghavidel., H., Zouaq., A., y Desmarais., M. (2020). Using bert and xlnet for the automatic short answer grading task. In *Proceedings of the 12th International Conference on Computer Supported Education - Volume 1: CSEDU,*, pages 58–67. INSTICC, SciTePress.

Hao, F., Man, J., Jingyan, L., Gong, Q., Tao, Peifu, T., Yan, Y., Yun, L., y Liu (2020). A lstm algorithm estimating pseudo measurements for aiding ins during gnss signal outages. *Remote Sensing*, 12:256.

Howard, J. y Ruder, S. (2018). Universal language model fine-tuning for text classification.

Lakshmipathi, N. (2019). Imdb dataset of 50k movie reviews. recuperado de https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews. recuperado el 22 de abril de 2022.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., y Soricut, R. (2019). Albert: A lite bert for self-supervised learning of language representations.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., y Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., y Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

Olah, C. (2015). Understanding lstm networks.

Pang, B. y Lee, L. (2005). Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales.

Patel, D., Raval, P., Parikh, R., y Shastri, Y. (2020). Comparative study of machine learning models and bert on squad.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., y Zettlemoyer, L. (2018). Deep contextualized word representations.

Radford, A., Narasimhan, K., Salimans, T., y Sutskever, I. (2018). Improving language understanding by generative pre-training. *OpenAI*.

Rajpurkar, P., Jia, R., y Liang, P. (2018). Know what you don't know: Unanswerable questions for squad.

Rajpurkar, P., Zhang, J., Lopyrev, K., y Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text.

Reddy, S., Chen, D., y Manning, C. D. (2018). Coqa: A conversational question answering challenge.

Ruder, S. (2022). NLP-progress.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., y Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

Ryan, C. M. a. N. (2019). BERT Fine-Tuning Tutorial with PyTorch. Chris McCormick . retrieved from http://www.mccormickml.com.

Sanh, V., Debut, L., Chaumond, J., y Wolf, T. (2019). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Sun, C., Qiu, X., Xu, Y., y Huang, X. (2019a). How to fine-tune bert for text classification?

Sun, Y., Wang, S., Li, Y., Feng, S., Chen, X., Zhang, H., Tian, X., Zhu, D., Tian, H., y Wu, H. (2019b). Ernie: Enhanced representation through knowledge integration.

Thomas, A. (2020). *Natural Language Processing with Spark NLP: Learning to Understand Text at Scale*. O'Reilly Media.

Topal, M. O., Bas, A., y van Heerden, I. (2021). Exploring transformers in natural language generation: Gpt, bert, and xlnet.

Torres, J. (2019). *Deep learning: introducción práctica con Keras : segunda parte*. Watch this space. Kindle Direct Publishing.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., y Polosukhin, I. (2017). Attention is all you need.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., y Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding.