Transformers BERT para *Question-Answering* sobre COVID-19

Trabajo Fin de Máster

Bernardo Sigüenza Moreno

Trabajo de investigación para el

Máster Universitario en Tecnologías del Lenguaje

Universidad Nacional de Educación a Distancia

Dirigido por el

Prof. Dr. D. Anselmo Peñas Padilla

Septiembre 2021

A mi familia

Agradecimientos

Quiero agradecer al Dr. Anselmo Peñas Padilla que haya dirigido este Trabajo Final de Máster. Gracias a su orientación, mi propuesta inicial basada en teorías de procesamiento ya superadas se ha convertido en esta investigación sobre modelos de lenguaje modernos aplicados a la resolución de un problema de actualidad.

Resumen

La sobrecarga de información debido al ritmo de publicación de artículos científicos requiere sistemas *question-answering* que proporcionen acceso eficiente al conocimiento adecuando las respuestas al tipo de usuario.

Para el desarrollo de sistemas *question-answering* son necesarios *datasets* de entrenamiento/evaluación anotados por expertos. Sin embargo, los *datasets* existentes para comprensión lectora en áreas de conocimiento especializadas como medicina no tienen un volumen de muestras suficiente para usarlos con métodos de aprendizaje supervisado. BioASQ v9b, un *dataset* biomédico, contiene 3.742 preguntas; COVID-QA-2019 (Möller et al., 2020) 2.019 ternas de pregunta, artículo, respuesta; COVID-QA-147 (Tang et al., 2020) 147 ternas, COVID-QA-111 (Lee et al., 2020) 111 ternas mientras que la versión 2 del *Stanford Question Answering Dataset* SQuAD v2 (Rajpurkar et al., 2018), un *dataset* genérico creado a partir de artículos de Wikipedia, contiene 130.319 muestras de entrenamiento, 11.873 de validación y 8.862 de pruebas.

Una solución a la falta de *datasets question-answering* específicos del dominio con tamaños suficientes de muestras consiste en inducir el modelo de lenguaje en un *dataset* de dominio general y aplicarlo al dominio específico.

Este trabajo estudia el rendimiento de modelos BERT (Devlin et al., 2018) y SBERT (Reimers et Gurevych, 2019) entrenados en corpus de

dominio general SQuAD v2, QuAC (Choi et al., 2018) y MS MARCO (Nguyen et al., 2016) cuando se utilizan para obtener respuestas en el dominio COVID-19 mediante el corpus CORD-19 (Wang et al., 2020).

Abstract

Information overload due to the increase in scientific literature requires question-answering systems that provides efficient access to knowledge, adapting the answers to the user.

One of the most important requirements for the development of a question-answering system is an expert annotated training/validation dataset. However, existing machine reading comprehension datasets for question-answering in specialized knowledge areas such as biomedicine are not large enough to be used in supervised learning models; e.g., BioASQ v9b, a biomedical dataset containing 3742 questions; COVID-QA-2019 (Möller et al., 2020) consisting of 2019 question-article-answer triples; COVID-QA-147 (Tang et al., 2020) 147 triples, COVID-QA-111 (Lee et al., 2020) 111 triples. The Stanford Question Answering Dataset SQuAD v2 (Rajpurkar et al., 2018), a reading comprehension dataset created by crowdworkers on a set of Wikipedia articles is composed of 130319 training samples, 11873 validation samples and 8862 test samples.

To address the lack of biomedical dataset, the language representations are pretrained and fine-tuned on large generic corpora, e.g., SQuAD, and evaluated in COVID-19 domain.

We evaluate performance of BERT (Devlin et al., 2018) and SBERT (Reimers et Gurevych, 2019) models trained in SQuAD v2, QuAC

(Choi et al., 2018) and MS MARCO (Nguyen et al., 2016) to obtain answers in the COVID-19 domain using CORD-19 dataset (Wang et al., 2020)

Índice general

Índice de Figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación

Desde el descubrimiento del coronavirus SARS-COV-2 a finales de 2019 hasta el 15 de febrero de 2021 se publicaron 111.228 artículos[1] sobre el virus y la enfermedad asociada, COVID-19; 2.223 trabajos nuevos en la semana previa. Esta producción bibliográfica causa una sobrecarga de información que dificulta que los investigadores se mantengan al día sobre nuevos hallazgos. Dado que los avances se realizan sobre el conocimiento existente, es necesario disponer de herramientas que den acceso a este volumen de literatura. Con el fin de potenciar el desarrollo de tales herramientas, la comunidad científica desarrolla el conjunto de datos CORD-19 (Wang et al., 2020) y propone varios retos compartidos:

- **Kaggle CORD-19 research challenge**[2], los participantes debían extraer respuestas a preguntas científicas sobre COVID-19 a partir de los documentos del corpus CORD-19. El desafío se desarrolló en dos rondas. La primera planteó nueve preguntas abiertas sobre transmisión, diagnóstico y tratamiento de COVID-19. A partir de las respuestas de esta primera ronda se planteó una segunda consistente en completar un esquema de

1 https://covid19primer.com/dashboard (consultado por última vez el 16 de febrero de 2021).

2 https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge

respuesta único para cada pregunta y se pidió a los participantes que llenaran el esquema extrayendo las respuestas desde los documentos de CORD-19. Por ejemplo, para factores de riesgo se debían extraer métricas de gravedad y mortalidad de la enfermedad; mientras que para incubación del virus debían incluirse rangos de tiempo.

Los resultados del reto Kaggle se han utilizado para crear por *bootstrapping* conjuntos de entrenamiento para métodos de aprendizaje supervisado. Los primeros sistemas que utilizaban aproximaciones supervisadas usaban conjuntos de datos sin preguntas específicas sobre COVID-19, por ejemplo, procedentes del corpus BioASQ[3].

- **TREC-COVID document retrieval challenge**[4], evaluaba la capacidad de los sistemas para recuperar documentos de CORD-19 y clasificarlos en base a la relevancia de éstos sobre un conjunto de temas COVID-19 predefinidos. Los resultados del desafío proporcionaron una evaluación de técnicas óptimas de recuperación para desarrollos futuros de sistemas de recuperación de información.

- **EPIC-QA**[5], *Epidemic Question Answering* promovía el desarrollo de sistemas de respuestas a preguntas biomédicas sobre COVID-19 fomentando la investigación en el diseño de soluciones capaces de dar respuestas ajustadas al nivel de conocimiento de los usuarios; ofreciendo respuestas de nivel experto precisas, como lo esperan las comunidades científicas y médicas así como respuestas en un lenguaje amigable para el público general. Propone dos tareas:

  a) QA para expertos. Las respuestas extraídas deben proporcionar información a preguntas formuladas por investigadores, científicos, virólogos o médicos;
  b) QA para consumidores. Las respuestas extraídas deben proporcionar información comprensible para el público general.

3 http://bioasq.org/
4 https://ir.nist.gov/covidSubmit/index.html
5 https://bionlp.nlm.nih.gov/epic_qa/

Cada tarea tiene su propio conjunto de preguntas aunque varias de ellas se comparten para explorar si las mismas soluciones se pueden aplicar a diferentes tipos de usuarios. El reto evalúa la capacidad de los sistemas para extraer y clasificar fragmentos de texto que formen las respuestas de las preguntas en lugar de devolver documentos completos.

1.2 Objetivo general y propuesta

El objetivo del trabajo es estudiar la eficiencia de modelos de lenguaje BERT para *question-answering* entrenados en *datasets* de dominio general SQuAD v2 (Rajpurkar et al., 2018), QuAC (Choi, Eunsol, et al. 2018) y MS MARCO (Nguyen et al., 2016) cuando se aplican a la tarea de obtener respuestas a preguntas del dominio COVID-19 utilizando el *dataset* CORD-19 (Wang et al., 2020).

Tomando como referencia un *pipeline* QA de tres etapas: recuperación de información → obtención de respuestas → reranking de respuestas, el trabajo se centra en el módulo de obtención de respuestas analizando dos aproximaciones a la obtención de respuestas: (i) QA extractiva con diferentes modelos BERT (Devlin et al., 2018) y (ii) búsqueda semántica con modelos SentenceBERT para determinar el modelo más efectivo.

En concreto, se plantean las siguientes cuestiones de investigación: ¿es mejor utilizar una aproximación basada en q*uestion-answering* o en *sentence similarity*? Para *question-answering*, ¿es mejor utilizar BERT, BioBERT, ClinicalBERT o SciBERT? ¿Con qué *dataset* de *fine-tuning* SQuAD v2, QuAC o ambos? ¿Cómo afectan los hiperparámetros de *fine-tuning* al rendimiento de los modelos?

Para que la recuperación de información no afecte a las conclusiones, se parte de los documentos relevantes dados por la organización del reto EPIC-QA. El estudio se realiza en condiciones ideales de recuperación.

El módulo de obtención de respuestas recibe como entradas una pregunta en lenguaje natural y una lista de documentos relevantes todos

con el mismo valor de relevancia y, devuelve un conjunto de *nuggets*, fragmentos que forman la respuesta. La corrección de las respuestas se mide con la métrica *Normalized Discount Novelty Score (*NDNS).

La metodología de evaluación compara los resultados obtenidos con los resultados alcanzados por los sistemas presentados a la tarea A: *expert QA* del desafío EPIC-QA.

1.3 Estructura del documento

El capítulo 2 desarrolla el marco teórico en el que se fundamenta este trabajo: arquitectura de *Transformers* (Vaswani et al., 2018), mecanismo de *self-attention* y modelos BERT y analiza varios trabajos participantes en el reto EPIC-QA.

El capítulo 3 define el marco de evaluación: metodología, métricas NDNS y corpus EPIC-QA.

El capítulo 4 realiza la experimentación. En primer lugar determina qué modelo para *extractive question-answering es* más efectivo en función del *dataset* con el que se realiza *fine-tuning.* Una vez identificado el modelo se analiza si es mejor utilizar una aproximación basada en *question-answering* o en *sentence similarity*.

El capítulo 5 enumera las conclusiones que se obtienen a partir del análisis de resultados y enuncia varias líneas de investigación hacia donde podría dirigirse un trabajo futuro.

Capítulo 2

Marco teórico y trabajos relacionados

2.1 Marco teórico

2.1.1 BERT: Bidirectional Encoder Representations from Transformers

BERT (Devlin et al., 2019) es un modelo de lenguaje que aprende representaciones bidireccionales usando la arquitectura de *Transformers* (Vaswani et al., 2018). Modelos previos como ELMO (Peters et al., 2018) o GPT (Radford et al., 2018) utilizan representaciones de lenguaje unidireccionales de izquierda a derecha; cada *token* solo atiende a *tokens* previos. Las representaciones unidireccionales no son óptimas en tareas a nivel de *tokens* como QA donde es esencial incorporar el contexto de ambas direcciones.

La arquitectura de BERT es una red bidireccional de *Transformers*. Si se denota por L el número de capas o bloques de *Transformers*, por H la dimensión de las capas ocultas y por A el número de bloques de autoatención, el modelo $\text{BERT}_{\text{BASE}}$ emplea L = 12, H = 768 y A = 12,

aproximádamente 110M de párametros y el modelo $BERT_{LARGE}$ utiliza L = 24, H = 1024 y A = 16, 340M de parámetros en total. Con esta estructura, BERT ha establecido nuevos resultados en varias tareas de procesamiento de lenguaje natural incluyendo QA, clasificación y regresión de pares de oraciones.

BERT utiliza la arquitectura de *Transformer* para computar representaciones de las entradas y salidas sin usar redes neuronales convolucionales (CNN) o recurrentes (RNN) evitando las desventajas de este tipo de redes:

- CNN: las redes neuronales convolucionales computan representaciones ocultas en paralelo para posiciones de entrada y salida. El número de operaciones necesarias para relacionar señales de dos posiciones arbitrarias de entrada o de salida crece con la distancia entre posiciones. El *Transformer* reduce dichas operaciones a un número constate a cambio de disminuir la resolución del modelo debido al promediado de las posiciones ponderadas de atención. Este efecto se contrarresta utilizando varios bloques de atención (*multi-head attention*).

- RNN: las redes neuronales recurrentes alinean las posiciones de los símbolos de entrada y salida iterativamente en tiempo de computación generando una secuencia de estados ocultos h_t dependiente del estado oculto anterior h_{t-1} y la entrada en el paso t. Esta naturaleza secuencial impide la paralelización de las muestras de entrenamiento y condiciona la longitud de las secuencias debido a limitaciones de memoria en el procesamiento por lotes.

Representaciones de entrada y salida. La representación de entrada utiliza secuencias de *tokens* para manejar sin ambigüedad tanto oraciones individuales como pares de oraciones. En BERT, una sentencia es un fragmento arbitrario de texto contiguo el cual no siempre coincide con el concepto lingüístico de oración. Los pares de sentencias se concatenan juntos en una única secuencia de *tokens*: x_1, …, x_N e y_1, …, y_M con tokens especiales que los delimitan: [CLS] x_1, …, x_N, [SEP], y_1, …, y_M, [EOS]. Las longitudes de M y N están limitadas a $M + N < T$ donde T define la longitud máxima de la secuencia de entrada.

Las sentencias, además de por el *token* especial [SEP], se diferencian añadiendo un *embedding* de segmento a cada *token* que indica si pertenece a la sentencia A o a la sentencia B. BERT usa *embeddings WordPiece* (Wu et al., 2016) con un vocabulario de alrededor de 30.000 tokens. El primer token de cada secuencia siempre es el carácter especial [CLS]. El estado oculto final correspondiente a este *token* se utiliza en tareas de clasificación. Para un *token* dado, su representación de entrada se construye sumando los *embeddings* del *token*, el segmento y la posición correspondientes:

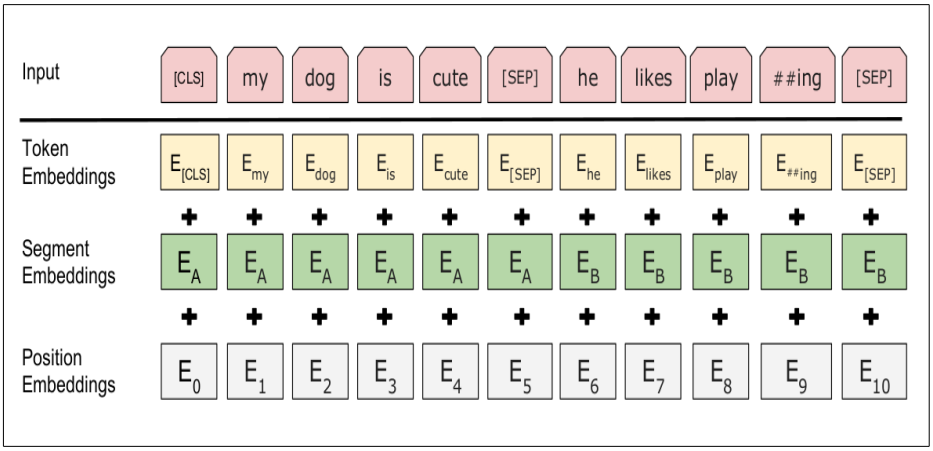

Figura 1: Preprocesamiento de una entrada BERT (del trabajo original Devlin et al., 2018)

Entrenamiento La aplicación de un modelo BERT a una tarea de procesamiento de lenguaje natural consta de dos etapas: preentrenamiento (*pre-training*) y ajuste (*fine-tuning*). El modelo primero se preentrena en un gran corpus de texto sin etiquetar y posteriormente se ajusta (*fine-tuning*) usando datos etiquetados dependientes de la tarea final. Un rasgo distintivo de BERT es su arquitectura unificada para diferentes tareas de procesamiento.

Preentrenamiento Durante el preentrenamiento, el modelo usa grandes volúmenes de datos no etiquetados para aprender dos objetivos: (1) un modelo de lenguaje enmascarado y, (2) la predicción de la siguiente sentencia. (Baevski et al., 2019) demuestra que incrementar el tamaño del corpus de datos resulta en mejoras del rendimiento de la tarea final. BERT se entrenó en una combinación de BookCorpus (Zhu et al., 2015) junto con Wikipedia inglesa, en total 16GB de texto sin comprimir.

Objetivos de preentrenamiento:

1. Aprendizaje de un modelo de lenguaje enmascarado (MLM, *Masked Language Model*). Se enmascaran el 15% de los *tokens Wordpiece* de entrada al azar para posteriormente predecirlos. La finalidad de enmascarar un porcentaje de los *tokens* es evitar que la bidireccionalidad del modelo permita que cada palabra se "vea" indirectamente a sí misma y que el modelo prediga trivialmente la palabra objetivo en un contexto multicapas.

   Aunque esta tarea está definida para obtener un modelo preentrenado bidireccional, tiene la desventaja de que se crea un desajuste entre las etapas de *pre-training* y *fine-tuning* porque el *token* [MASK] no aparece en la fase de ajuste. Para mitigar este efecto, no siempre se reemplazan los *tokens* enmascarados con el *token* especial [MASK]. El generador de muestras de entrenamiento, elige el 15% de las posiciones de los *tokens* al azar para la predicción. Si se elige el enésimo *token*, se reemplaza este *token* con (1) el *token* [MASK] el 80% de las veces; (2) un *token* aleatorio el 10% de las veces; (3) el *token* enésimo sin cambios el 10% de las veces restantes.

   El objetivo de aprender un modelo de lenguaje enmascarado como se denomina en terminología BERT es la definición de la tarea Cloze (Taylor, 1953) consistente en completar un texto del que se han suprimido palabras de forma sistemática para probar la comprensión lectora del modelo. BERT, a diferencia de los *auto-encoders* (Vincent et al., 2008) no reconstruye la entrada completa, sólo predice los *tokens* enmascarados.

2. Predicción de la sentencia siguiente (NSP, *Next Sentence Prediction*). Varias tareas finales como respuesta a preguntas (QA, *Question-Answering*) e inferencia de lenguaje natural (NLI, *Natural Language Inference*) se basan en entender la relación entre dos sentencias. Para entrenar un clasificador que entienda tales relaciones, se construye el conjunto de entrenamiento etiquetando pares de sentencias A y B. El 50% de las veces, B es la sentencia siguiente a A; el ejemplar resultante se anota como (*is_next*). El otro 50% de las ocasiones, la sentencia B se extrae del corpus de forma aleatoria; la muestra obtenida se etiqueta como (*not_next*).

> Aunque los autores de BERT consideran que el preentrenamiento en esta tarea es beneficioso tanto para QA como inferencia de lenguaje natural (NLI), existe discusión en la comunidad de procesamiento de lenguaje natural sobre la utilidad de esta tarea. Posteriormente, al desarrollar los modelos RoBERTa (Liu et al., 2019) y ALBERT (Lan et al., 2019) se profundizará en esta cuestión.

BERT se optimiza con Adam (Kingma and Ba, 2015) usando los siguientes parámetros: $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1\text{e-}6$ y decaimiento de peso L2 de 0.01. El ratio de aprendizaje se eleva durante los primeros 10.000 pasos hasta un valor máximo de 1e-4 y luego decae linealmente. BERT se entrena con un *dropout* de 0.1 en todas las capas y pesos de atención, y una función de activación GELU (Hendrycks y Gimpel, 2016). Los modelos se preentrenan durante 1M de iteraciones, con lotes de 256 secuencias y longitud máxima de las secuencias de 512 *tokens*.

Ajuste (*fine-tuning*). En la etapa de *fine-tuning*, el modelo se inicia con los parámetros de preentrenamiento y a continuación se ajustan éstos usando datos etiquetados para la tarea final. Cada tarea final tiene su propio modelo ajustado pero todos se inician con los mismos parámetros.

El proceso de *fine-tuning* es directo dado que el mecanismo de autoatención (*self-attention)* del *Transformer* permite que BERT modele tareas de procesamiento conectando entradas y salidas específicas y ajustando los parámetros de extremo a extremo. En la entrada, la oración A y la oración B del preentrenamiento tienen diferente significado en función de la tarea final: (1) pares de oraciones en tareas de parafraseo; (2) pares de hipótesis-premisa en tareas de implicación; (3) pares de pregunta-pasaje en tareas de *question-answering* y; (4) un par texto-$\varnothing$ en clasificación de texto o etiquetado de secuencia. En la salida, las representaciones de *tokens* alimentan una capa de salida dependiente de la tarea. Comparado con el preentrenamiento, el proceso de *fine-tuning* tiene un coste computacional relativamente bajo.

Formulación de la tarea de QA. Dado un pasaje de texto y una pregunta, la tarea *extractive question-answering* consiste en seleccionar

un fragmento continuo de texto en el pasaje como respuesta a la pregunta.

La tarea representa la pregunta y la respuesta como una única secuencia de entrada. Empaqueta ambas usando el *embedding* A para la pregunta y el *embedding* B para el pasaje. Durante la etapa de *fine-tuning*, se introducen dos nuevos parámetros: un vector inicial $S \in R^H$ y un vector final $E \in R^H$. Se denota el vector oculto final de BERT para el i-ésimo token de entrada como $T_i \in R^H$. Se puede calcular la probabilidad de que la palabra c sea el comienzo del intervalo de respuesta como un producto escalar entre T_c y S seguido de una función softmax sobre todas las palabras del párrafo:

$$P_c = \frac{e^{S \cdot T_c}}{\sum_j e^{S \cdot T_j}}$$

análogamente, se puede calcular la probabilidad de que la palabra f sea el final del intervalo de respuesta como la softmax del producto escalar entre T_f y E:

$$P_f = \frac{e^{E \cdot T_f}}{\sum_j e^{E \cdot T_j}}$$

El objetivo del entrenamiento es la probabilidad logarítmica de la posición inicial y final correcta. La puntuación de un fragmento candidato desde la posición c hasta la posición f se define como $S \cdot T_c + E \cdot T_f$. El fragmento con máxima puntuación donde f ≥ c se usa como predicción. El objetivo de entrenamiento es la suma de las probabilidades logarítmicas de las posiciones correctas de comienzo y fin.

Las siguiente figuras muestran las probabilidades de que los *token* t_c y t_f sean, respectivamente, el comienzo y el final de la respuesta a la pregunta: "*what are the transmission routes of coronavirus?"* en el contexto: "*Transmission -Current information suggests that the two main routes of transmission of the COVID-19 virus are respiratory droplets and contact. Vertical transmission has also been reported. 6 Respiratory droplets are generated when an infected person coughs or sneezes. Any person who is in close contact (within 1 m) with someone who has respiratory symptoms (coughing, sneezing) is at risk of being*

exposed to potentially infective respiratory droplets. Droplets may also land The virucidal efficacy of chemical germicides against coronavirus has been investigated. A study of disinfectants against human coronavirus 229E found several disinfectants were effective after a 1-minute contact time; these included sodium hypochlorite (at a free chlorine concentration of 1,000 ppm and 5,000 ppm), 70% ethyl alcohol, and povidone-iodine (1% iodine). A study also showed complete inactivation of the SARS coronavirus by 70% ethanol and povidoneiodine with an exposure time of 1 minute and 2.5% glutaraldehyde with an exposure time of 5 minute. 11 Although Glutaraldehyde (GA) is an effective disinfectant but it potential to cause severe allergic dermatitis and should be used cautiously... [el pasaje continúa]".

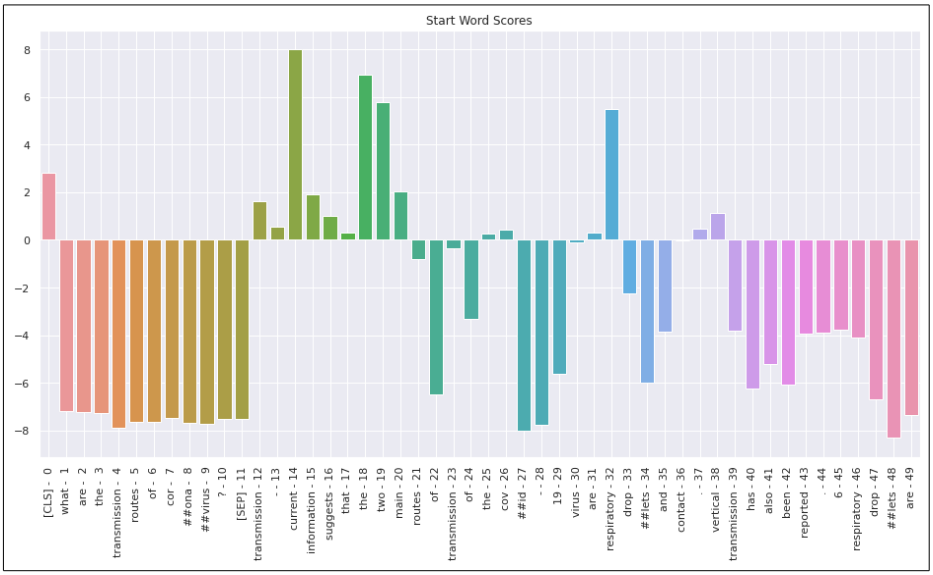

Figura 2: Probabilidad de los tokens para el comienzo de respuesta

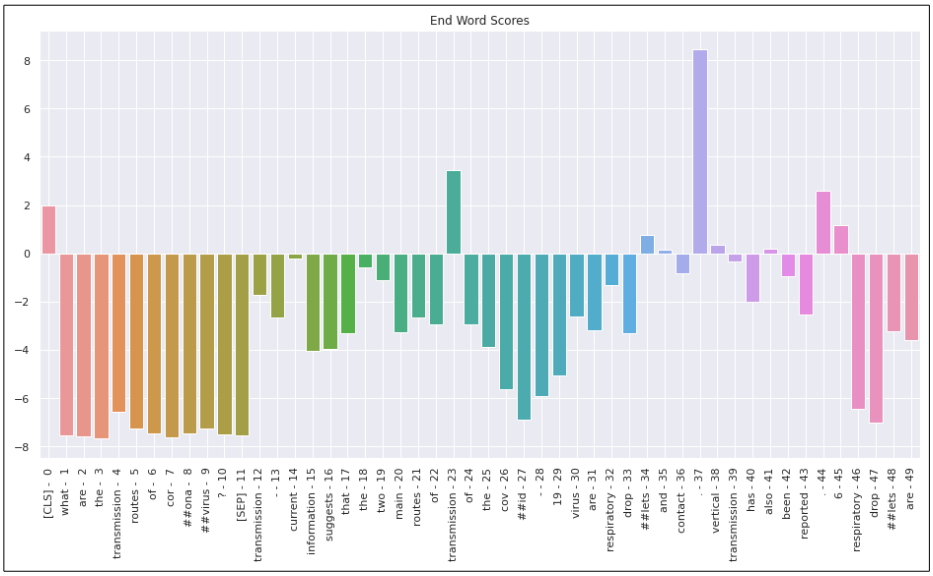

Figura 3: Probabilidad de los tokens para el final de respuesta

La respuesta extraída por el modelo para la pregunta y el contexto dados es la secuencia de *tokens* que se encuentran entre el *token* t_c = "*current*" y el token t_f = "." ambos incluidos: "*current information suggests that the two main routes of transmission of the cov ##id – 19 virus are respiratory drop ##lets and contact* .".

Si el pasaje no contiene ninguna respuesta, se predicen las posiciones inicial y final como 0 y se responde el *token* [CLS]. El espacio de probabilidad se extiende para incluir la posición del *token* [CLS]. En tiempo de inferencia, se compara la puntuación del fragmento sin-respuesta: $s_{null} = S \cdot T_{[CLS]} + E \cdot T_{[CLS]}$ con la puntuación del mejor fragmento no-nulo: $\hat{s_{c,f}} = max_{f \geq c} S \cdot T_c + E \cdot T_f$. Se predice una respuesta no nula cuando $\hat{s_{c,f}} > s_{null} + \tau$, donde el parámetro τ es un valor umbral determinado empíricamente durante el entrenamiento para maximizar el rendimiento del modelo.

La siguiente figura muestra la respuesta del modelo para la pregunta: "*what are best practices in hospitals and at home in maintaining quarantine?*" cuando se utiliza el mismo contexto que en el ejemplo anterior:

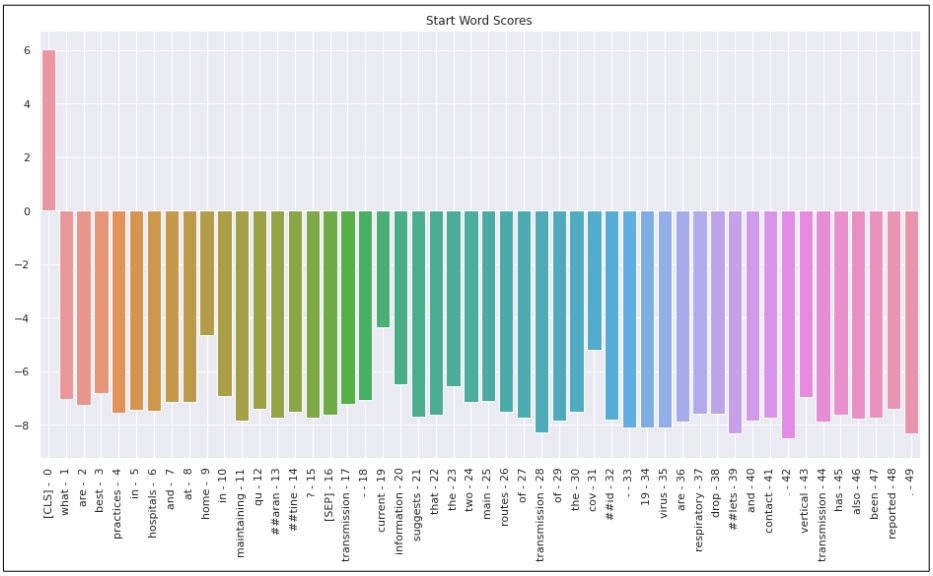

Figura 4: Probabilidad de los tokens cuando la respuesta no está en el contexto

Resultados

■ **GLUE** El *benchmark* GLUE *General Language Understanding Evaluation* (Wang et al., 2018a) es una colección de diversas tareas de comprensión del lenguaje natural. Para el ajuste en GLUE, se representa la secuencia de entrada como se ha descrito anteriormente: [CLS] A [SEP] B [SEP], como representación agregada se usa el vector oculto final $C \in \mathbb{R}^H$ correspondiente al primer *token* de entrada [CLS]. Los únicos parámetros introducidos durante el ajuste son los pesos de la capa de clasificación $W \in \mathbb{R}^{K \times H}$, donde K es el número de etiquetas. Se computa una función de pérdida de clasificación estándar con C y W, por ejemplo, $\log(softmax(C\,W^T))$.

Se utilizó un tamaño de lote de 32 y se entrenó durante 3 épocas sobre la colección de datos para tareas GLUE. En cada tarea, se seleccionó el mejor ratio de aprendizaje (entre 5e-5, 4e-5, 3e-5 y 2e-5). Se encontró que $\text{BERT}_{\text{LARGE}}$ era inestable algunas veces en conjuntos con pocos datos así que se reinició aleatoriamente varias veces y se seleccionó el modelo con los mejores resultados en el conjunto dev.

Modelo	MNLI 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5	Promedio
$BERT_{BASE}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$BERT_{LARGE}$	86.7/85.9	72.1	92.7	94.9	60.5	86.5	89.3	70.1	82.1

Tabla 1: Comparativa $BERT_{BASE}$ frente a $BERT_{LARGE}$ en el benchmark GLUE

Al entrenar varios modelos BERT con diferente número de capas, unidades ocultas y bloques de atención manteniendo los hiperparámetros y procedimiento de ajuste descrito previamente se observa que los modelos más grandes conducen a mejoras en la precisión en los cuatro conjuntos de datos, incluso para MRPC, que solo tiene 3.600 ejemplos de entrenamiento etiquetados y es sustancialmente diferente de las tareas previas al entrenamiento. Se sabe que el incremento en el tamaño del modelo llevará a mejoras continuas en tareas a gran escala como traducción máquina y modelado del lenguaje.

■ Question-answering extractivo

SQuAD v1.1 El conjunto de datos *Stanford Question Answering Dataset* (SQuAD v1.1) es una colección de 100K pares de pregunta/respuesta construido mediante colaboración (Rajpurkar et al., 2016). Dada una pregunta y un pasaje de Wikipedia que contiene la respuesta, la tarea es predecir el fragmento de texto en el pasaje que responde la pregunta.

SQuAD v2.0 La tarea SQuAD 2.0 extiende la definición del problema SQuAD 1.1 permitiendo que no exista respuesta en el pasaje proporcionado y haciendo el problema más realista. Se ajustan los modelos durante dos épocas con un ratio de aprendizaje de 5e-5 y un tamaño de lote de 48.

Dataset	Modelo	EM	F1
SQuAD v 1.1	$BERT_{BASE}$	80.8	88.5
	$BERT_{LARGE}$	84.1	90.9
SQuAD v2	$BERT_{BASE}$	80.4	77.6
	$BERT_{LARGE}$	78.7	81.9

Tabla 2: Comparativa $BERT_{BASE}$ frente a $BERT_{LARGE}$ en SQuAD v1.1 y SQuAD v2

2.1.2 *Transformers*

El *Transformer* (Vaswani et al., 2017) reproduce la arquitectura codificador-decodificador de los modelos de transducción neuronal secuenciales (Bahdanau et al., 2014); el codificador mapea una secuencia de símbolos de entrada $x = (x_1, \ldots, x_n)$ a una secuencia de representaciones continuas $z = (z_1, \ldots, z_n)$ y, a partir de z, el decodificador genera una secuencia de salida $y = (y_1, \ldots, y_m)$ de símbolos un elemento a la vez. En cada paso el modelo es autoregresivo consumiendo los símbolos generados previamente como entrada adicional cuando se produce la siguiente salida.

El *Transformer* utiliza la arquitectura general de pila de capas de auto-atención completamente conectadas para codificador y decodificador.

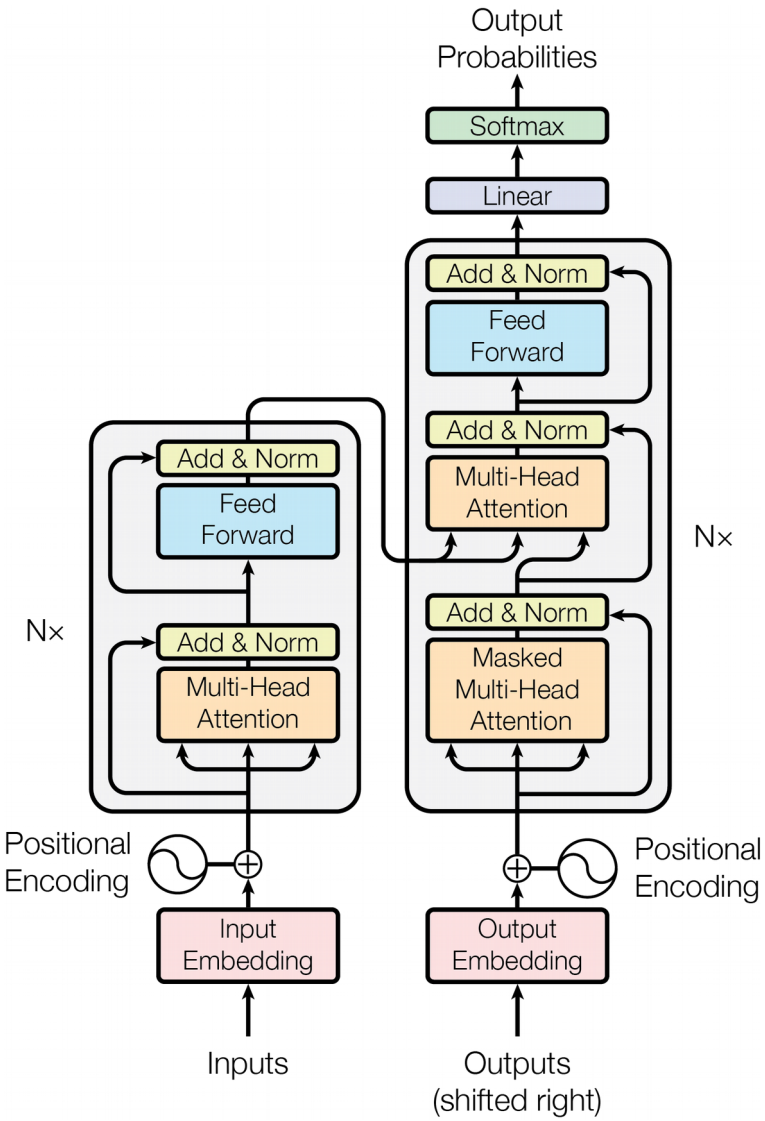

Figura 5: Arquitectura del Transformer (del original Vaswani et al., 2017)

2.1.2.1 Pilas de codificadores y decodificadores

■ **Codificador.** El codificador está compuesto de una pila de N=6 capas idénticas. Cada capa tiene dos subcapas. La primera es un mecanismo de varios bloques de autoatención, en la literatura se denominan *multi-head self-attention* y la segunda es una red *feed-forward* (FFNN) posicional completamente conectada. Se usa una conexión residual alrededor de cada una de las dos subcapas seguida por una operación de normalización de capa. La salida de cada subcapa es LayerNorm(x + Sublayer(x)), donde Sublayer(x) es la función implementada por la propia subcapa. Para facilitar las conexiones residuales, todas las

subcapas del modelo así como las capas de *embeddings* producen salidas de dimensión $d_{model} = 512$.

■ **Decodificador.** El decodificador está también compuesto de una pila de N=6 capas idénticas. Además de las dos subcapas empleadas en el codificador, el decodificador inserta una tercera subcapa que realiza *multi-head attention* sobre la salida de la pila de codificación. Al igual que en el codificador, se emplean conexiones residuales alrededor de cada subcapa seguida por una operación de normalización de capa. También se modifica la subcapa de autoatención de la pila del decodificador para evitar que las posiciones previas afecten a las posiciones posteriores. Este enmascaramiento combinado con el hecho de que la salida de *embeddings* se desplaza por una posición, asegura que las predicciones para la posición i sólo dependan de las salidas conocidas en las posiciones menores que i.

2.1.2.2 Atención

El mecanismo de atención permite que el decodificador acceda a la historia completa del codificador logrando una representación más rica de la sentencia fuente. Una función de atención se define como una aplicación de una consulta y un conjunto de pares clave-valor a una salida; donde la consulta, las claves, los valores y la salida son vectores. La salida se computa como la suma ponderada de los valores. El peso asignado a cada valor se calcula a partir de una función de compatibilidad entre la consulta y la clave correspondiente.

■ **Atención producto escalar escalado.** La entrada consiste de consultas y claves de dimensión d_k, y valores de dimendión d_v. Se calculan los productos escalares de la consulta con todas las claves, se divide cada una de ellas entre $\sqrt{d_k}$ y se aplica la función softmax para obtener los pesos de los valores.

En la práctica, se computa la función de atención en un conjunto de consultas a la vez mediante álgebra de matrices. Sean Q, K y V las matrices de las consultas, las claves y los valores respectivamente, la matriz de atención se calcula como:

$$Atención(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Las dos funciones de atención comúnmente empleadas son atención aditiva y atención producto escalar. La atención aditiva calcula la función de compatibilidad usando una red *feed-forward* con una capa oculta única. Mientras las dos son similares en complejidad teórica, en la práctica, la atención producto escalar es mucho más rápida y más eficiente en espacio dado que se puede implementar usando código optimizado para multiplicación de matrices.

Para valores pequeños de d_k, los dos mecanismos rinden de forma similar. La atención aditiva mejora al producto escalar sin escalado para grandes valores de d_k. Se sospecha que para grandes valores de d_k, el producto escalar crece más rápido en magnitud empujando la función softmax a regiones con gradientes extremadamente pequeños. Para contrarrestar este efecto, se escala el producto escalar por $1/\sqrt{d_k}$.

■ ***Multi-head attention.*** En lugar de realizar una función de atención simple con claves, valores y consultas de dimensión d_{model} se ha demostrado beneficioso proyectar linealmente las consultas, las claves y los valores h veces con diferentes proyecciones lineales para dimensiones d_k, d_k y d_v respectivamente. Las consultas y claves tienen dimensión d_k, y los valores tienen dimensión d_v. En cada una de estas versiones proyectadas de consultas, claves y valores se calcula la función de atención en paralelo, creando valores de salida de dimensión d_v. Estos valores se concatenan y una vez más se proyectan, resultando en los valores finales.

Disponer de varios bloques de atención permite al modelo atender conjuntamente a información procedente de diferentes subespacios de representación en diferentes posiciones. Con un mecanismo de atención simple al promediar se inhibe este efecto.

$$MultiHead(Q,K,V)=Concat(head_1,\ldots,head_h)W^O$$

donde

$$head_i=Attention(QW_i^Q,KW_i^K,VW_i^V)$$

y las proyecciones W son matrices de tamaño:

$$W_i^Q \in \mathbb{R}^{d_{model} \times d_k}, W_i^K \in \mathbb{R}^{d_{model} \times d_k}, W_i^V \in \mathbb{R}^{d_{model} \times d_v}$$

En la arquitectura de *Transformer* original se define una estructura con h = 8 bloques de atención en paralelo o *heads*. Para cada una de estos bloques se usa $d_k = d_v = d_{model}/h = 64$. Debido a las reducidas dimensiones de cada bloque, el coste computacional total es similar al de una capa de atención simple de dimensión completa.

■ **Aplicaciones de la atención en el modelo.** El *Transformer* emplea atención *multi-head* de tres formas diferentes:

- En las subcapas de atención codificador-decodificador, las consultas vienen desde la capa de decodificador previa y las claves y valores vienen desde la salida del codificador. Esto permite a cada posición del decodificador atender todas las posiciones de la secuencia de entrada. De este modo se mimetizan los mecanismos de atención de codificador-decodificador en modelos de secuencia a secuencia.
- El codificador contiene capas de autoatención. En una capa de autoatención todas las claves, valores y consultas proceden del mismo origen, en este caso, la salida de la capa previa del codificador. Cada posición en el codificador puede atender a todas las posiciones de la capa previa del codificador.
- Del mismo modo, las capas de autoatención en el decodificador permiten a cada posición del decodificador atender a todas las posiciones previas a esa posición. Para preservar la propiedad autoregresiva es necesario prevenir el flujo de información hacia la izquierda en el decodificador. Este comportamiento se implementa con la atención producto escalar escalado enmascarando (estableciendo en $-\infty$) todos los valores de la entrada de la función softmax que corresponden a conexiones ilegales.

■ **Por qué la autoatención.** El uso de autoatención está motivado por tres cualidades deseable en el modelo: (1) La complejidad computacional total por capa. (2) La cantidad de computación que puede ser paralelizada en relación con el número mínimo de operaciones secuenciales requeridas. (3) La longitud de la ruta entre las dependencias de larga distancia en la red. Aprender dependencias de larga distancia es necesario en muchas tareas de transducción de

secuencias. Un factor clave que afecta a la habilidad para aprender tales dependencias es la longitud de las rutas de las señales hacia adelante y hacia atrás que tienen que atravesar la red. Cuanto más cortas son las rutas entre cualquier combinación de posiciones en las secuencias de entrada y salida más fácil es aprender dependencias de largo rango (Hochreiter et al., 2001).

La capa de autoatención conecta todas las posiciones con un número constante de operaciones ejecutadas secuencialmente mientras que una capa recurrente requiere O(n) operaciones secuenciales. En términos de complejidad computacional, las capas de autoatención son más rápidas que las capas recurrentes cuando la longitud de la secuencia n es más pequeña que la dimensión d de la representación, que es el caso más frecuente si se representan las secuencias utilizando W*ordpiece* o B*yte-Pair*. Para mejorar el rendimiento en tareas que involucran secuencias muy largas, la autoatención puede limitarse a una ventana de tamaño r de la secuencia de entrada centrada alrededor de la respectiva posición de salida. Esto incrementaría la longitud de la ruta máxima a O(n/r).

2.1.2.3 Position-wise feed-forward networks

Junto a las subcapas de atención, cada una de las capas en el codificador y decodificador contiene una red neuronal *feed-forward* completamente conectada que se aplica a cada posición separada e idénticamente. Esto consiste de dos transformaciones lineales con una activación ReLU entre ellas.

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

Mientras que las transformaciones lineales son las mismas a través de las diferentes posiciones, se usan diferentes parámetros de capa a capa. Otro modo de describir esto es como dos convoluciones con tamaño de núcleo 1 (*point-wise*). La dimensionalidad de entrada y salida es $d_{model} = 512$ y la capa interior tiene dimensionalidad $d_{ff} = 2048$.

2.1.2.4 Embeddings y Softmax

Se usan *embeddings* aprendidos para convertir los *tokens* de entrada y los *tokens* de salida a vectores de dimensión d_{model}. También se usa una transformación lineal aprendida y función softmax para que la salida del

decodificador prediga las probabilidades del siguiente *token*. En el modelo, se comparte la misma matriz de pesos entre las dos capas de *embeddings* y la transformación lineal previa a la softmax. En las capas de *embeddings*, se multiplica estos pesos por $\sqrt{d_{model}}$.

2.1.2.5 Codificación posicional

Dado que el modelo no contiene recurrencia ni convolución, se debe inyectar información acerca de la posición relativa o absoluta de los *tokens* en la secuencia. Para este fin, se añade "codificación posicional" a los *embeddings* de entrada en las bases de las pilas de codificador y decodificador. Las codificaciones posicionales tienen la misma dimensión d_{model} que los *embeddings* así que ambos pueden sumarse. Para las codificaciones posicionales se usan funciones seno y coseno de diferentes frecuencias:

$$PE_{(pos,2i)} = \sin\left(pos / 10000^{2i/d_{model}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(pos / 10000^{2i/d_{model}}\right)$$

donde *pos* es la posición e *i* es la dimensión. Es decir, cada dimensión de la codificación posicional corresponde a una sinusoide. Las longitudes de onda forman una progresión geométrica desde 2л hasta 10000 · 2л. Se elige esta función porque el modelo aprenderá fácilmente a atender a posiciones relativas para cualquier desplazamiento fijo k; PE_{pos+k} puede representarse como una función lineal de PE_{pos}.

2.1.2.6 Entrenamiento

La arquitectura de T*ransformer* usa el optimizador Adam con β_1=0.9, β_2=0.98 y $\varepsilon=10^{-9}$ variando el ratio de aprendizaje durante el entrenamiento según la fórmula:

$$lrate = d_{model}^{-0.5} \cdot min\left(\text{step\_num}^{-0.5}, \text{step\_num} \cdot \text{warmup\_steps}^{-1.5}\right)$$

Esto corresponde a incrementar el ratio de aprendizaje linealmente para los primeros *warmup_steps* pasos de entrenamiento y decrementarlo a partir de entonces proporcionalmente a la inversa del cuadrado de la raíz del número de pasos. Se usa *warmup_steps* = 4000.

Durante el entrenamiento se utilizan dos formas de regularización:

1. **Desconexión de conexiones residuales (*residual dropout*).** (Srivastava, 2014) Se aplica *dropout* a la salida de cada subcapa con $P_{drop} = 0.1$. Previamente, se suma a la subcapa de entrada y se normaliza. Además, se aplica desconexión a las sumas de los *embeddings* y las codificaciones posicionales en ambas pilas de codificadores y decodificadores manteniendo el valor de $P_{drop} = 0.1$.
2. **Suavizado de etiquetas** Durante el entrenamiento, se emplea suavizado de etiquetas de valor $\varepsilon_{ls} = 0.1$ (Szegedy et al., 2016) Esto daña la perplejidad porque el modelo aprende a ser más inseguro pero mejora la precisión.

2.1.3 Optimizaciones del modelo BERT

2.1.3.1 RoBERTa: Robustly Optimized BERT

(Liu et al., 2019) analiza el preentrenamiento de BERT para realizar una elección cuidadosa de hiperpárametros que optimice los resultados del modelo en relación al coste computacional. Los autores concluyen que BERT está subentrenado y proponen RoBERTa que supera el rendimiento de BERT. Las modificaciones introducidas son simples:

1) cambiar dinámicamente el patrón de enmascaramiento aplicado a los datos de entrenamiento;
2) emplear sentencias completas eliminando el objetivo de predecir la siguiente sentencia;
3) entrenar el modelo durante más tiempo en conjuntos de datos mayores y con un número menor de lotes pero aumentando el número de secuencias por lote.
4) utilizar un vocabulario BPE a nivel de bytes de aproximadamente 50K entradas.

■ **Enmascarado estático frente a enmascarado dinámico.** La implementación original de BERT realiza enmascarado una vez durante el preprocesamiento de los datos dando como resultado una máscara estática única. RoBERTa genera una máscara dinámica cada vez que se alimenta el modelo con una secuencia. Esta estrategia es crucial cuando

se preentrena por más pasos o con *datasets* mayores ya que evita que el modelo vea las secuencias de entrenamiento con las mismas máscaras múltiples veces.

■ **Formato de entrada y predicción de la siguiente sentencia.** En el procedimiento original de preentrenamiento de BERT, el modelo observa dos segmentos concatenados que son bien muestras contiguas del mismo documento (con P=0.5) o de distintos documentos. Además del objetivo de crear un modelo de lenguaje enmascarado, el modelo se entrena para predecir si los segmentos del documento proceden del mismo o de distinto documento a través de una función de pérdida auxiliar denominada predicción de la siguiente sentencia (NSP, *Next Sentence Prediction*).

En el trabajo original de (Devlin et al., 2019) se supone que la predicción de la siguiente sentencia es un factor importante en el entrenamiento del modelo observando que eliminar la tarea NSP degrada el rendimiento en QNLI, MNLI y SQuAD 1.1. Sin embargo, algunos trabajos posteriores cuestionan la necesidad de la función de pérdida NSP (Lample and Conneau, 2019; Yang et al., 2019; Joshi et al., 2019). Para entender estas discrepancias los autores de RoBERTa compararon diferentes alternativas de formatos de entrenamiento encontrando que utilizar sentencias individuales daña el rendimiento en tareas finales porque el modelo no es capaz de aprender dependencias de larga distancia.

RoBERTa se preentrena sin función de pérdida NSP y con fragmentos de textos procedentes del mismo documento encontrando que esta configuración iguala o ligeramente supera los resultados originales de $\text{BERT}_{\text{BASE}}$ en tareas finales.

■ **Entrenamiento con grandes volúmenes de datos.** Trabajos en Traducción Automática Neuronal (NMT, *Neural Machine Translation*) (Ott et al., 2018) han mostrado que entrenar con grandes volúmenes mejora la velocidad de optimización y el rendimiento de la tarea final cuando el ratio de aprendizaje se incrementa adecuadamente. El modelo original $\text{BERT}_{\text{BASE}}$ entrena durante 1M de pasos con tamaño de lote de 256 secuencias. Esto es equivalente en coste computacional, mediante acumulación del gradiente, a entrenar por 125K pasos con tamaños de lotes de 2K secuencias o por 31K pasos con tamaños de lotes de 8K.

(You et al., 2019) también prueba que BERT es sensible a lotes de entrenamiento grandes usando tamaños de 32K secuencias. El entrenamiento con grandes lotes controlando el número de pasos mejora la perplejidad para el objetivo de modelado de lenguaje enmascarado así como la precisión de la tarea final. Además, lotes de mayor longitud son más fáciles de paralelizar mediante arquitecturas paralelas distribuidas.

La versión RoBERTa que emplea la arquitectura $\text{BERT}_{\text{LARGE}}$ (L = 24, H = 1024, A = 16, 355M de parámetros) necesitó 1024 GPUs V100 durante aproximadamente veinticuatro horas para completar un preentrenamiento de 100K pasos sobre un corpus comparable a BookCorpus + Wikipedia. En la fecha de publicación de este trabajo, el precio por hora de una GPU V100 en el cloud de Google[6] era de 2,48 USD: $2{,}48\,\$ \times 24\,\text{h} \times 1024 = 60.948{,}48\,\$$.

■ **Codificación del texto.** La codificación de pares de bytes (BPE) (Sennrich et al., 2016) permite manejar los grandes vocabularios comunes en los corpus de lenguaje natural. En lugar de palabras completas, BPE se basa en unidades de subpalabras que se extraen realizando un análisis estadístico del corpus de entrenamiento. Es un híbrido entre representaciones a nivel de caracteres y palabras.

La implementación original de BERT utiliza un vocabulario BPE a nivel de caracteres de 30K entradas aprendido mediante preprocesamiento de la entrada con reglas de *tokenización* heurísticas. Esta aproximación tiene el inconveniente de que los caracteres *unicode* pueden ocupar una porción importante del vocabulario cuando se modelan corpus grandes. (Radford et al., 2019) propone una implementación de BPE que cambia los caracteres *unicode* por bytes como unidades básicas de subpalabras. El uso de bytes hace posible aprender un vocabulario de subpalabras de un tamaño modesto (50K unidades) que puede codificar cualquier texto de entrada sin introducir ningún *token* [UNKNOWN] cuando es necesario representar subpalabras desconocidas.

■ **Resultados.** Los resultados muestran que RoBERTa mejora a $\text{BERT}_{\text{LARGE}}$ en todas las tareas reafirmando la importancia de las elecciones de diseño durante el preentrenamiento:

6 https://cloud.google.com/compute/gpus-pricing

1) Tamaño del dataset. RoBERTa se entrena con 160GB de texto durante 100K pasos.
2) Número mayor de iteraciones. RoBERTa mejora las tareas finales al pasar de 100K a 300K y posteriormente 500K iteraciones. A pesar del mayor número de iteraciones el modelo no sobreajusta.

Los resultados mostrados a continuación corresponden a RoBERTa entrenado por 500K pasos.

GLUE. Se realiza *fine-tuning de* RoBERTa separadamente para cada tarea GLUE usando solo el subconjunto de entrenamiento correspondiente a la tarea. Se realiza un barrido de hiperparámetros para cada tarea con los siguientes valores: tamaño de lotes {16, 32} y ratios de aprendizaje {1e-5, 2e-5, 3e-5} con un *warmup* lineal para los primeros 6% de pasos seguido de un decaimiento lineal hasta 0. Se ajusta durante 10 épocas y se realiza detención temprana en base a las métricas de evaluación en el subconjunto *dev*. El resto de hiperpárametros se mantienen constantes durante el preentrenamiento.

Modelo	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI
$BERT_{LARGE}$	86.6	92.3	91.3	70.4	93.2	88.0	60.6	90.0	
RoBERTa	90.2	94.7	92.2	86.6	96.4	90.9	68.0	92.4	91.3

Tabla 3: Comparativa $BERT_{LARGE}$ frente a RoBERTa en el benchmark GLUE

SQuAD. Para SQuAD v1.1 se sigue el mismo procedimiento de ajuste que en BERT. Para SQuAD v2.0 además se clasifica si la pregunta dada tiene respuesta en el contexto asociado; se entrena este clasificador conjuntamente con el predictor de fragmentos sumando las funciones de pérdida de ambos términos: clasificación y fragmento.

Modelo	SQuAD v1.1		SQuAD v2.0	
	EM	F1	EM	F1
$BERT_{LARGE}$	84.1	90.9	79.0	81.8
RoBERTa	88.9	94.6	86.5	89.4

Tabla 4: Comparativa $BERT_{LARGE}$ frente a RoBERTa en SQuAD v1.1 y SQuAD v2

El procedimiento de entrenamiento propuesto en RoBERTa mejora los resultados publicados por BERT en las tareas GLUE y SQuAD. Cuando se entrena por más tiempo sobre datos adicionales, el modelo

obtiene una puntuación de 88.5 en GLUE estableciendo nuevos valores en cuatro de las nueve tareas GLUE: MNLI, QNLI, RTE y STS-B.

■ **Conclusiones.** RoBERTa demuestra que el rendimiento mejora entrenando modelos más grandes sobre más datos, con un número mucho mayor de lotes y secuencias mayores, eliminando el objetivo de predecir la siguiente sentencia y cambiando el patrón de enmascaramiento dinámicamente.

2.1.3.2 ALBERT: A Lite BERT for self-supervised learning of language representations

ALBERT (Lan et al., 2019) realiza varias modificaciones sobre la arquitectura BERT. Mantiene la arquitectura de T*ransformer* (Vaswani et al., 2017) pero utiliza activaciones no lineales de error gausiano (GELU, *Gaussian Error Linear Units*) (Hendryck y Gimpel, 2016) en lugar de las unidades lineales rectificadas (ReLU, *Rectified Linear Units*) de la propuesta original.

ALBERT factoriza la matriz de *embeddings* en dos matrices más pequeñas logrando independizar el tamaño de las capas ocultas de la dimensión de los embeddings del vocabulario. Esta parametrización permite aumentar el tamaño de las capas ocultas sin aumentar significativamente la dimensión de los *embeddings* del vocabulario. En BERT y RoBERTa el tamaño E del vocabulario de *embeddings* WordPiece está asociado con la dimensión de las capas ocultas: $E \equiv H$. Los *embeddings* WordPiece aprenden representaciones independientes del contexto, mientras que los *embeddings* en las capas ocultas aprenden representaciones dependientes del contexto. Al desvincular el tamaño E de los *embeddings* Wordpiece del tamaño H de la dimensión de las capas ocultas, se vuelve más eficiente el uso de los parámetros del modelo.

Desde una perspectiva práctica, el procesamiento del lenguaje requiere vocabularios V de tamaños superiores a las 30.000 entradas. Si $E \equiv H$, incrementar H incrementa el tamaño de la matriz de *embeddings* que tiene tamaño $V \times E$. Esto puede resultar fácilmente en modelos con miles de millones de parámetros. ALBERT reduce significativamente el número de parámetros del modelo proyectando los

vectores *one-hot* sobre un espacio de *embeddings* de tamaño E, donde $H \gg E$; el resultado se proyecta sobre las capas ocultos. Al usar esta descomposición, se reducen los parámetros de embeddings de $O(V \times H)$ a $O(V \times E + E \times H)$.

La segunda técnica se basa en compartir parámetros entre capas evitando que el número de parámetros crezca con la profundidad de la red. ALBERT comparte los parámetros de la red neuronal *Feed-Forward* entre las capas y los parámetros de atención. Se observa que las transiciones de capa a capa son mucho más suaves en ALBERT que en BERT; compartir pesos tiene el efecto de estabilizar los parámetros de la red.

Ambas técnicas reducen el número de parámetros del modelo sin dañar significativamente el rendimiento. Una arquitectura ALBERT similar a $\text{BERT}_{\text{LARGE}}$ tiene 18 veces menos parámetros y se puede entrenar alrededor de 1,7 veces más rápido.

Para mejorar aún más el rendimiento, ALBERT reemplaza la ineficiente función de pérdida (Yang et al., 2019; Liu et al., 2019) del modelo BERT original consistente en predecir la siguiente sentencia (NSP, *Next Sentence Prediction*) por una función de pérdida auto-supervisada de predicción del orden de las oraciones (SOP, *Sentence-Order Prediction*).

Los autores de ALBERT suponen que la razón principal detrás de la ineficacia de NSP es su falta de dificultad comparada con la tarea principal de modelado de lenguaje enmascarado (MLM, masked language modeling). No obstante, consideran que el modelado entre sentencias es un aspecto importante de la comprensión del lenguaje y proponen una función de pérdida basada en la coherencia entre oraciones. La función de pérdida SOP genera ejemplos positivos con la misma técnica que BERT (dos segmentos consecutivos del mismo documento) y como ejemplos negativos los mismos segmentos pero con el orden intercambiado. Esto fuerza al modelo a aprender distinciones de grano fino sobre las propiedades de coherencia a nivel de discurso.

Debido a todas las decisiones de diseño descritas, los modelos ALBERT tienen menos parámetros que los correspondiente modelos

BERT. Por ejemplo, $ALBERT_{LARGE}$ tiene 18 veces menos parámetros comparados a $BERT_{LARGE}$, 18M frente a 334M. Una configuración de $ALBERT_{XLARGE}$ con H=2048 tiene sólo 60M de parámetros y $ALBERT_{XXLARGE}$ con H=4096 tiene 233M de parámetros. Esta mejora en la eficiencia de parámetros es la ventaja más importante en las elecciones de diseño de ALBERT.

■ **Preentrenamiento.** Al igual que BERT, ALBERT emplea BookCorpus y Wikipedia para preentrenar los modelos usando un vocabulario de 30.000 tokens generado con SentencePiece. Las entradas tienen la forma [CLS] x_1, …, x_N, [SEP], y_1, …, y_M, [SEP] con una longitud máxima M + N < 512 *tokens.* Aleatoriamente se generan secuencias de entrada más cortas con probabilidad del 10%.

El modelo produce entradas enmascaradas para el objetivo de modelado de lenguaje enmascarado (MLM) utilizando mascaras de n-gramas (Joshi et al., 2019). La longitud de cada máscara de n-gramas se selecciona de forma aleatoria. La probabilidad para la longitud n está dada por:

$$p(n)=\frac{1/n}{\sum_{k=1}^{N} 1/k}$$

La longitud máxima de los n-gramas se establece en un valor N, por ejemplo, 3. Es decir, el objetivo MLM puede consistir de hasta 3 palabras completas, por ejemplo “respiratory syndrome coronavirus”.

El modelo se actualiza utilizando un tamaño de lote de 4096 y un optimizador LAMB con un ratio de aprendizaje de 0.00176 (You et al., 2019). Si no se especifica otro valor, todos los modelos se entrenan durante 125.000 pasos

■ **Resultados.** Como resultado de las decisiones de diseño descritas es posible escalar ALBERT teniendo aún menos parámetros que $BERT_{LARGE}$ pero alcanzando mejoras de rendimiento significativas. ALBERT alcanza 89.4 F1 en GLUE y 92.2 F1 en SQuAD 2.0.

Model		Params	SQuAD v1.1	SQuAD v2.0	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

Tabla 5: Comparativa de modelos BERT frente a modelos ALBERT en SQuAD y GLUE

Si bien ALBERT-xxlarge tiene menos parámetros que BERT-large y obtiene resultados significativamente mejores, es computacionalmente más costoso debido a su estructura más grande.

2.1.3.3 DistilBERT: una versión destilada de BERT

DistilBERT (Sanh et al., 2019) es un método para preentrenar modelos de representación de lenguaje que aprovecha los sesgos inductivos aprendidos por modelos más grandes durante el preentrenamiento utilizando una función de pérdida triple que combina el modelado de lenguaje, la destilación y la distancia coseno. El modelo resultantes es más pequeño, rápido y con un coste menor de preentrenamiento.

Destilación de conocimiento (*Knowledge distillation*) (Hinton et al., 2015) es una técnica de compresión en la que un modelo compacto, el estudiante, se entrena para reproducir el comportamiento de un modelo mayor o un conjunto de modelos, el profesor.

Para predecir la clase a la que pertenece un ejemplo, se entrena un clasificador mediante aprendizaje supervisado para maximizar las probabilidades estimadas de las etiquetas de referencia. Un objetivo de entrenamiento estándar consiste en minimizar la entropía cruzada entre la distribución predicha del modelo y la distribución empírica de las etiquetas de entrenamiento. Un modelo que rinda correctamente en el conjunto de entrenamiento predirá una distribución de salida con una alta probabilidad en la clase correcta y probabilidades próximas a cero en el resto de clases. Algunas de estas probabilidades “casi cero” son mayores que otras y reflejan, en parte, las capacidades de

generalización del modelo y cómo de bien rendirán en el conjunto de pruebas.

Función de pérdida durante el entrenamiento El estudiante se entrena con una función de pérdida de destilación sobre las probabilidades objetivo del maestro: $L_{ce} = \sum_i t_i * \log(s_i)$ donde t_i (respectivamente s_i) es una probabilidad estimada por el maestro (respectivamente el estudiante). Este objetivo da como resultado una señal de entrenamiento que aprovecha la distribución de probabilidades completa del maestro. Siguiendo a (Hinton et al., 2015) se utiliza una softmax: $p_i = \frac{e^{(z_i/T)}}{\sum_j e^{(z_j/T)}}$ donde T (temperatura) controla la suavidad de la distribución de salida y z_i es la puntuación del modelo para la clase i. Se aplica la misma temperatura T al alumno y al profesor durante el entrenamiento. En tiempo de inferencia, T se establece a 1 para obtener una función softmax estándar.

El objetivo de entrenamiento final es una combinación lineal de la pérdida de destilación L_{ce} con la pérdida de entrenamiento supervisado, en el caso de DistilBERT la pérdida de modelado de lenguaje enmascarado L_{mlm} (Devlin et al., 2018). Es beneficioso agregar una pérdida de *embedding* coseno (L_{cos}) que tiende a alinear las direcciones de los vectores de estado oculto del alumno y del maestro.

■ **Arquitectura del estudiante.** El estudiante, DistilBERT, tiene la misma arquitectura general que BERT pero el número de capas se ha reducido por un factor de 2 y se han eliminado los e*mbeddings* para especificar los tipos de *token* y la capa de salida dependiente de la tarea final. Muchas de las operaciones utilizadas en la arquitectura de *Transformer* (capa lineal y normalización de capa) se han optimizado mediante álgebra lineal y los autores muestras que las variaciones en la última dimensión del tensor (tamaño oculto) tienen un impacto más pequeño en la eficiencia computacional (para un conjunto fijo de parámetros) que variaciones en otros factores como el número de capas. Por tanto, el esfuerzo está dirigido a reducir el número de capas.

■ **Inicialización del estudiante.** Además de las optimizaciones descritas previamente y las elecciones arquitectónicas, un elemento importante en el procedimiento de entrenamiento es encontrar la

inicialización correcta para que la subred converja. Tomando ventaja de la dimensionalidad común entre las redes de profesor y estudiante, se inicia el estudiante desde el profesor tomando una capa de cada dos.

■ **Destilación.** Se aplican las mejores prácticas de entrenamiento de modelos BERT propuestas en (Liu et al., 2019). Así que DistilBERT se destila en lotes muy grandes aprovechando acumulación de gradiente (hasta 4K muestras por lote) usando enmascaramiento dinámico y sin el objetivo de predicción de la siguiente oración.

■ **Datos y potencia de cálculo.** DistilBERT está entrenado en el mismo corpus que el modelo BERT original: una concatenación de Wikipedia en inglés y Toronto Book Corpus (Zhu et al., 2015). DistilBERT se entrenó en 8 GPU V100 de 16 GB durante aproximadamente 90 horas. El modelo RoBERTa (Liu et al., 2019) requirió 1 día de entrenamiento en 1024 GPU V100 de 32GB.

Model	Score	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI
$BERT_{BASE}$	79.5	56.3	86.7	88.6	91.8	89.6	69.3	92.7	89.0	53.5
DistilBERT	77.0	51.3	82.2	87.5	89.2	88.5	59.9	91.3	86.9	56.3

Tabla 6: Comparativa $BERT_{BASE}$ frente a DistilBERT en el benchmark GLUE

Para SQuAD 1.1 el modelo $BERT_{BASE}$ alcanza puntuaciones 81.2/88.5 en EM/F1 respectivamente. Mientras que DistilBERT obtiene 77.7/85.8 en EM/F1 respectivamente.

■ **Conclusiones.** DistilBERT es una versión preentrenada de uso general de BERT, un 40% más pequeña, un 60% más rápida, que retiene el 97% de las capacidades de comprensión de lenguaje. (Sanh et al., 2019) demuestra que un modelo de lenguaje de propósito general se puede entrenar con éxito mediante destilación.

2.1.4 Versiones específicas de BERT para el dominio científico-médico

2.1.4.1 BioBERT

BioBERT, Bidirectional Encoder Representations from Transformers for Biomedical Text Mining (Lee, Yoon et al., 2020) es un modelo de representación del lenguaje específico del dominio biomédico preentrenado en un corpus científico de gran tamaño. BioBERT supera significativamente a BERT en tareas de procesamiento de lenguaje biomédico.

BioBERT se inicia con los pesos del modelo BERT original entrenado en un corpus de dominio general (Wikipedia inglesa y BookCorpus). A continuación se preentrena en un corpus de dominio científico (resúmenes PubMed y artículos completos de PMC) y se ajusta y evalúa en tres tareas de procesamiento de lenguaje: reconocimiento de entidades (NER), extracción de relaciones (RE) y *question-answering* (QA). El modelo BioBERT mejora los resultados en tareas de procesamiento de textos biomédicos respecto del modelo BERT equivalente entrenado en un corpus de dominio general.

■ **Preentrenamiento.** BioBERT utiliza el vocabulario WordPiece (Wu et al., 2016) sensible a mayúsculas de $\text{BERT}_{\text{BASE}}$. Se ha probado que emplear un vocabulario que discrimine entre mayúsculas y minúsculas obtiene rendimientos ligeramente mejores en tareas finales en el dominio biomédico. BioBERT se preentrenó durante 23 días en 8 GPUs NVIDIA V100 de 32GB. La longitud máxima de la secuencia de entrada se fijó en 512 *tokens* y el tamaño del lote en 192 secuencias, lo que resulta en 98.304 palabras por iteración.

■ **Fine-tuning para QA.** Se usa la misma arquitectura BERT utilizada para SQuAD pero con el *datasets* de preguntas factuales BioASQ. Se computan las probabilidades de los *tokens* de comienzo y fin del fragmento de respuesta mediante una capa simple. Sin embargo, alrededor del 30% de las preguntas factuales de BioASQ están sin responder porque las respuestas exactas no aparecen en los pasajes dados. Como (Wiese et al., 2017) se excluyeron del conjunto de entrenamiento las muestras con preguntas sin responder. También se utiliza el mismo proceso de preentrenamiento de (Wiese et al., 2017) que utiliza SQuAD y que mejora el rendimiento tanto de BERT como de BioBERT. Para comparar entre modelos, emplea la métrica de evaluación *Mean Reciprocal Rank* (MRR).

Se usa una sola GPU NVIDIA Titan XP de 12GB para ajustar BioBERT en cada tarea final. Para el *fine-tuning*, se exploraron tamaños

de lote de 10, 16, 32 y 64, y tasas de aprendizaje de 5e-5, 3e-5 y 1e-5. El ajuste de BioBERT en tareas de QA tomó menos de una hora ya que el tamaño de los datos de ajuste es mucho menor que el de los datos de ajuste utilizados por (Devlin et al., 2019).

■ **Resultados.** BioBERT obtiene puntuaciones F1 mayores en NER biomédico (+0.62) y RE biomédico (+2.80) y una puntuación MRR superior (+12.24) en QA biomédico.

2.1.4.2 ClinicalBERT

ClinicalBERT (Alsentzer, et al., 2019) es el resultado de aplicar BERT a textos clínicos procedentes de 2 millones de notas de la base de datos MIMIC-III v1.4 (Johnson et al., 2016). Existen dos versiones de ClinicalBERT: (1) ClinicalBERT, iniciado desde $\text{BERT}_{\text{BASE}}$ y (2) ClinicalBioBERT, iniciado desde $\text{BioBERT}_{\text{BASE}}$.

■ **Preentrenamiento** Se utiliza un tamaño de lote de 32 secuencias, una longitud de secuencia máxima de 128 y un ratio de aprendizaje de 5e-5. Los modelos se entrenaron durante 150.000 pasos. Se experimentó también con modelos preentrenados durante 300.000 pasos pero no se encontraron diferencias significativas en el rendimiento de las tareas finales al aplicar estos modelos.

■ **Fine-tuning para NER y NLI** Para tareas finales, se exploraron los siguientes hiperparámetros: ratio de aprendizaje de {2e-5, 3e-5, 5e-5}, tamaño de lote {16, 32} y épocas {2, 3, 4}. La longitud de secuencia máxima se estableció en 150 *tokens* para todas las tareas. Los *embeddings* resultantes se pasan a través de una capa lineal para clasificación, bien a nivel de *tokens* para tareas de identificación como NER o usando el *token* centinela de comienzo de sentencia [CLS] para MedNLI.

■ **Coste computacional** El preprocesamiento y entrenamiento de BERT en MIMIC toma una cantidad significativa de recursos computacionales. Se estima que el procesamiento del modelo de *embeddings* entero llevó entre 17 y 18 días de tiempo de computación en una única GPU GeForce GTX TITAN X 12GB además de la potencia de CPU y memoria para las tareas de preprocesamiento. 18 días de ejecución continuada es una inversión significativa.

■ **Resultados** Los modelos ClinicalBERT y ClinicalBioBERT se han aplicado en tareas de inferencia de lenguaje natural en el ámbito médico, MedNLI (Romanov and Shivade, 2018) y en cuatro i2b2 tareas de reconocimiento de entidades NER, todas en formato IOB. ClinicalBioBERT ajustado en MedNLI alcanza una precisión de 82.7%.

2.1.4.3 SCIBERT[7]: A Pretrained Language Model for Scientific Text

SCIBERT (Beltagy et al., 2019) es un modelo de lenguaje preentrenado basado en BERT (Devlin et al., 2019) para tratar la falta de datos científicos etiquetados a gran escala y de alta calidad. Aprovecha el preentrenamiento no supervisado sobre un gran corpus multidominio de publicaciones científicas para mejorar el rendimiento en tareas de procesamiento de dominio científico incluyendo etiquetado de secuencias, clasificación de oraciones y análisis de dependencias. Los autores demuestran estadísticamente mejoras significativas sobre BERT.

Se entrena SCIBERT en una muestra aleatoria de 1.14M de artículos de Semantic Scholar (Ammar et al., 2018). Este corpus consiste de un 18% de artículos sobre ciencia de la computación y 82% del dominio biomédico. Se utilizan los artículos completos, no sólo los resúmenes. La longitud media de los artículos es de 154 oraciones (2.769 *tokens*) resultando en un corpus de 3.17B de *tokens*, similar en tamaño al corpus de 3.3B de *tokens* empleado en el entrenamiento de BERT. Se separan las oraciones utilizando ScispaCy[8] (Neumann et al., 2019).

■ **Vocabulario.** BioBERT usa WordPiece (Wu et al., 2016) para *tokenización* del texto de entrada. El vocabulario contiene las palabras o subpalabras más frecuentemente utilizadas. Para SCIBERT, los autores crean SCIVOCAB, un nuevo vocabulario WordPiece construido en el mismo corpus científico que SCIBERT utilizando SentencePiece[9]. Se producen dos vocabularios, uno sensible a mayúsculas y otro sólo en minúsculas con tamaños de 30K entradas para hacerlo similar al WordPiece original. El solapamiento de entradas entre el vocabulario original de WordPiece utilizado por BERT y SCIVOCAB usado por BioBERT es del 42% ilustrando las diferentes frecuencias en el uso de

7 https://github.com/allenai/scibert/

8 https://github.com/allenai/SciSpaCy

9 https://github.com/google/sentencepiece

las palabras entre textos de dominio científico y textos de dominio general.

Aunque un vocabulario específico del dominio es útil, SCIBERT se beneficia más de ser preentrenado desde cero en corpus científicos.

■ **Preentrenamiento.** Se utiliza el procedimiento original de BERT para entrenar SCIBERT con la misma configuración y tamaño que $BERT_{BASE}$. Se entrenan cuatro versiones diferentes de SCIBERT: (i) *cased* o *uncased* y (ii) BASEVOCAB o SCIVOCAB. Los dos modelos que utilizan BASEVOCAB son iniciados desde los modelos $BERT_{BASE}$ correspondientes. Los otros dos modelos que usan el nuevo SCIVOCAB son entrenados desde cero. Se establece una longitud de sentencia máxima de 128 *tokens* y se entrena el modelo hasta que la función de pérdida deja de decrecer. Después se continúa entrenando el modelo permitiendo longitudes de oraciones de hasta 512 tokens.

Se utiliza una única TPU v3 con 8 núcleos. Entrenar los modelos SCIVOCAB desde cero en el corpus toma 1 semana (5 días con longitud máxima de 128, después 2 días con longitud máxima 512). Los modelos BASEVOCAB toman 2 días menos de entrenamiento porque no se entrenan desde cero.

■ **Fine-tuning.** Se utiliza la misma arquitectura, optimización e hiperparámetros utilizados en (Devlin et al., 2019). Para clasificación de textos, se usa el *token* [CLS] para alimentar una capa de clasificación lineal. Para etiquetado secuencial, por ejemplo, NER, se utiliza cada token BERT en una capa de clasificación lineal con salida softmax.

En todas las configuraciones, se aplica un *dropout* de 0.1 y entropía cruzada usando Adam (Kingma and Ba, 2015) como optimizador. Se ajusta para 2 y 5 épocas usando un tamaño de lote de 32 y ratios de aprendizaje de 5e-6, 1e-5, 2e-5 o 5e-5 con un *warmup lineal* seguido de un decaimiento lineal (Devlin et al., 2019). Para cada *dataset* y variante BERT, se toma el mejor ratio de aprendizaje y número de épocas en el conjunto de desarrollo y se reportan los resultados correspondientes del conjunto de pruebas. Se encuentra que la configuración que trabaja mejor a través de muchos conjuntos de prueba y modelos es 2 o 5 épocas y un ratio de aprendizaje de 2e-5. Mientras que depende de la tarea, los hiperparámetros óptimos para cada tarea son a menudo los mismos que en las variantes BERT.

■ **Resultados.** Siguiendo (Devlin et al., 2019) se usan los modelos *cased* para NER y los modelos *uncased* para las otras tareas. También se utilizan los modelos *cased* para *parsing*. Algunos experimentos mostraron que los modelos *uncased* tienen mejores resultados que los modelos *cased* incluso en tareas a nivel de *token* como NER.

SCIBERT mejora $BERT_{BASE}$ en tareas científicas (+2.11 F1). En particular, dentro del dominio biomédico se observa que SCIBERT mejora $BERT_{BASE}$ en tareas biomédicas (+1.92 F1).

2.1.5 SpanBERT

SpanBERT (Joshi et al., 2019) es un modelo preentrenado diseñado para mejorar la representación y predicción de fragmentos de texto. Extiende BERT de dos formas: (1) enmascarando fragmentos continuos aleatorios en lugar de *tokens* aleatorios y (2) entrenando representaciones de límites de fragmentos de texto para predecir el contenido completo del fragmento enmascarado. SpanBERT mejora BERT en tareas de selección de fragmentos como *question answering* y resolución de coreferencia. En particular, con el mismo conjunto de datos de entrenamiento y mismo tamaño que el modelo $BERT_{LARGE}$, SpanBERT obtiene 94.6% y 88.7% F1 en SQuAD 1.1 y SQuAD 2.0 respectivamente.

Para implementar SpanBERT, se usa una réplica ajustada de BERT que en sí misma supera sustancialmente al BERT original. El preentrenamiento en segmentos individuales, en lugar de dos segmentos concatenados para predecir la siguiente oración (NSP) mejora considerablemente el rendimiento en la mayoría de las tareas finales.

■ **Modelo.** Se diferencia de BERT en tres aspectos: (i) utiliza un proceso de enmascarado aleatorio diferente; (ii) introduce una función objetivo distinta, *Span Boundary Objetive* (SBO) que trata de predecir fragmentos enmascarados completos utilizando sólo los *tokens* extremos del fragmentos; (iii) muestrea un segmento contiguo de texto para cada ejemplar de entrenamiento en lugar de construir la sentencia de entrada concatenando dos segmentos y, por tanto, no emplea el objetivo de predecir la siguiente sentencia.

■ **Máscara de fragmento.** Dada una secuencia de tokens X = (x_1, x_2, …, x_n), se selecciona un subconjunto de tokens $Y \subseteq X$ mediante muetreo iterativo de fragmentos de texto hasta alcanzar el porcentaje de enmascaramiento, por ejemplo, 15% de X. En cada iteración, primero se muestrea la longitud de un fragmento siguiendo una distribución geométrica $l \sim Geo(p)$ sesgada hacia fragmentos más cortos. A continuación se selecciona uniformemente el punto de comienzo del fragmento a enmascarar. Siempre se muestrea una secuencia completa de palabras en lugar de *tokens* de subpalabras y el punto de comienzo debe estar en el comienzo de una palabra. Después de las pruebas preliminares, se establece p=0.2 y la longitud máxima del fragmento l_{max}=10 palabras. Esto produce una longitud media de fragmento de 3.8 palabras.

Como en BERT, en total se enmascaran el 15% de los *tokens* reemplazando 80% de los tokens enmascarados con el *token* [MASK], 10% con *tokens* aleatorios y 10% con los *tokens* originales. Sin embargo, se realiza este reemplazo a nivel de fragmentos y no para cada *token* individual. Por ejemplo, todos los *tokens* del fragmento se reemplazan con [MASK].

■ **Span Bondary Objective.** Los modelos de selección de fragmentos (Lee et al., 2016, 2017; He et al., 2018) suelen crear una representación de longitud fija de un fragmento utilizando sus *tokens* de inicio y fin. Idealmente, las representaciones de final de fragmento deberían resumir la mayor cantidad posible del contenido interno del fragmento. En SpanBERT se hace introduciendo el objetivo SBO que implica predecir cada *token* de un fragmento enmascarado utilizando solo las representaciones de los *tokens* observados en los extremos.

Formalmente, sea la salida del codificador del transformador para cada *token* de la secuencia: x_1, ..., x_n. Dado un fragmento enmascarado de *tokens* (x_s, ..., x_e) $\in$ Y, donde (s, e) indican las posiciones inicial y final, se representa cada *token* x_i del intervalo utilizando las codificaciones de salida de los *tokens* externos x_{s-1} y x_{e+1}, así como el *embedding* de posición del *token* objetivo p_{i-s+1}:

$$y_i = f(x_{s-1}, x_{e+1}, p_{i-s+1})$$

donde los *embeddings* de posición p_1, p_2, ... marcan las posiciones relativas de los *tokens* enmascarados con respecto al *token* de límite izquierdo x_{s-1}. Se implementa la función de representación f(·) como una red *feed-forward* de 2 capas con activaciones GeLU (Hendrycks y Gimpel, 2016) y normalización de capa (Ba et al., 2016):

$$h_0 = [x_{s-1}; x_{e+1}; p_{i-s+1}]$$
$$h_1 = LayerNorm(GeLU(W_1 h_0))$$
$$y_1 = LayerNorm(GeLU(W_2 h_1))$$

Se utiliza la representación del vector y_i para predecir el *token* x_i y computar la entropía cruzada exactamente como en un objetivo MLM. SpanBERT suma las pérdidas de ambos objetivos: *span boundary* y MLM para cada *token* x_i en el fragmento enmascarado (x_s, …, x_e), mientras reutiliza el *embedding* de entrada para los *tokens* objetivos en ambos MLM y SBO:

$$loss(x_i) = loss_{MLM}(x_i) + loss_{SBO}(x_i) = -\log P(x_i|X) - \log P(x_i|Y)¿$$

■ **Entrenamiento de secuencia simple.** Las muestras de BERT contienen dos secuencias de texto (X_A, X_B) y un objetivo que entrena el modelo para predecir si están conectadas (NSP). Los autores de SpanBERT encontraron que esta configuración es casi siempre peor que usar una secuencia simple sin el objetivo NSP. Hipotizaron que el entrenamiento de secuencia única es superior al entrenamiento de pares de secuencia con NSP porque (a) el modelo se beneficia de contextos completos y (b) el condicionamiento de contextos, a menudo sin relación, añade ruido al modelo de lenguaje enmascarado. Por ello, en SpanBERT se eliminan ambos, el objetivo NSP y el procedimiento de muestreo con dos segmentos y simplemente se muestrea un segmento continuo de hasta n=512 *tokens* en lugar de dos segmentos cuya suma sea n tokens juntos.

En resumen, SpanBERT preentrena representaciones de fragmento mediante: (1) enmascarado de fragmentos de palabras completas usando una distribución geométrica, (2) optimizando un objetivo de límite de fragmento además del MLM mediante un *pipeline* de datos de secuencia única.

■ **Resultados.** SpanBERT alcanza valores F1 de 94.6% y 88.7% en SQuAD1.1 y SQuAD 2.0 respectivamente. Mientras que otros trabajos han mostrado los beneficios de agregar más datos (Yang et al., 2019) y aumentar el tamaño del modelo (Lample y Conneau, 2019), La estrategia de SpanBERT consistente en diseñar objetivos de preentrenamiento significativos para las tareas finales demostrando que tienen un impacto notable en el rendimiento del modelo.

Extractive Question Answering. Siguiendo BERT, se utiliza la misma arquitectura de modelo QA para todos los *datasets*. Primero se convierten el pasaje P = (p_1, p_2, …, p_l) y la pregunta Q = (q_1, q_2, …, $q_{l'}$) en una secuencia simple X = [CLS] p_1 p_2 … p_l [SEP] q_1 q_2 … $q_{l'}$ [SEP] se pasa al codificador preentrenado y se entrenan dos clasificadores independientemente para predecir los límites del fragmento respuesta (comienzo y fin). Para preguntas sin respuesta en SQuAD 2.0, simplemente se establece el fragmento respuesta al token especial [CLS].

	SQuAD 1.1		SQuAD 2.0	
	EM	F1	EM	F1
$BERT_{BASE}$	84.3	91.3	80.0	83.3
SpanBERT	88.8	94.6	85.7	88.7

Tabla 7: Comparativa $BERT_{BASE}$ frente a DistilBERT en el SQuAD v1.1 y SQuAD v2

Para todas las tareas QA se usa longitud de secuencia máxima = 512 y una ventana de tamaño 128 cuando las longitudes son mayores de 512. Se eligen ratios de aprendizaje de {5e-6, 1e-5, 2e-5, 3e-5, 5e-5} y tamaños de lotes de {16, 32} ajustando durante 4 épocas sobre todos los *datasets*.

2.1.6 Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks (SBERT)

BERT (Devlint et al., 2018) y RoBERTa (Liu et al., 2019) han establecido nuevos resultados en tareas de regresión de pares de oraciones tales como similitud textual semántica (STS, Semantic Textual Similarity). Sin embargo, es necesario que ambas oraciones alimenten la red lo que causa sobrecarga computacional. Encontrar la pareja de oraciones más similares en una colección de n = 10.000 sentencias requiere que BERT realice n(n-1)/2 = 49.995.000

inferencias. En una GPU V100 este cómputo necesita alrededor de 65 horas demostrando que la arquitectura de BERT no es adecuada para búsqueda de similitud semántica o tareas no supervisadas como *clustering*.

SBERT (Reimers et Gurevych, 2019) utiliza codificadores BERT duales, redes BERT siamesas en la terminología de SBERT. El modelo emplea dos codificadores separados E_Q y E_P para la pregunta y el pasaje respectivamente. Cada codificador es un modelo BERT (base, *uncased*) que produce la representación del token [CLS] como salida. La similitud entre una pregunta y un pasaje es el productos escalar de la salida de los codificadores:

$$\text{sim}(q, p) = E_Q(q)^T E_P(p)$$

SBERT reduce el esfuerzo de encontrar la pareja más similar de 65 horas con BERT/RoBERTa a alrededor de 5 segundo con SBERT, manteniendo la precisión de BERT.

■ **Introducción.** Una solución común para tratar el *clustering* y las búsquedas semánticas consiste en proyectar cada oración sobre un espacio vectorial de modo que las oraciones semánticamente similares estén próximas. Para aplicar BERT en esta tarea, se calculan los *embeddings* de sentencias de tamaño fijo y se utiliza la representación del *token* [CLS]. Sin embargo, esta práctica crea *embeddings* de oraciones pobres a menudo peores que promediar *embeddings* GloVE (Pennington et al., 2014).

Para mitigar esta limitación, se desarrolló SBERT, una arquitectura de red siamesa que utiliza dos modelos BERT con pesos acoplados y vectores de tamaño fijo como sentencias de entrada. Se pueden encontrar oraciones semánticamente similares utilizando una medida de similitud como el coseno o la distancia euclídea. Estas métricas pueden ser realizadas de forma extremadamente eficiente en hardware permitiendo que SBERT se utilice para búsqueda de similitud semántica también como para *clustering*.

■ **Modelo.** SBERT añade una operación de *pooling* a la salida de BERT / RoBERTa para derivar *embeddings* de sentencias de tamaño fijo. Experimenta con tres estrategias de *pooling*: usando la salida del *token*

[CLS], computando la media de todos los vectores de salida (MEAN-strategy) o computando el máximo en el tiempo de los vectores de salida (MAX-strategy). La configuración por defecto es MEAN.

Para ajustar BERT / RoBERTa se usan redes siamesas dobles y triples (Schroff et al., 2015) que actualizan simultáneamente los pesos de los *embeddings* de sentencias para que sean semánticamente similares. La estructura de red depende de los datos de entrenamiento disponible. Se experimenta con las siguientes estructuras y funciones objetivos:

Función objetivo clasificación. Se concantenan los *embeddings* de las sentencias *u* y *v* con la diferencia elemento a elemento de $|u-v|$ y se multiplica por los pesos de entrenamiento $W_t \in \mathbb{R}^{3n \times k}$:

$$o = softmax(W_t(u, v, |u-v|))$$

donde *n* es la dimensión de *embeddings* de sentencias y *k* el número de etiquetas. Se optimiza con la función de pérdida de entropía cruzada.

Función objetivo regresión. Se computa la similitud coseno entre los dos *embeddings* de oraciones *u* y *v*. Se usa la función de pérdida error cuadrado medio como función objetivo.

Función objetivo tripleta. Dada una sentencia ancla *a*, una sentencia positiva *p* y una sentencia negativa *n*, la pérdida de tripleta ajusta la red de modo que la distancia entre a y p sea más pequeña que la distancia entre a y n. Matemáticamente, se minimiza la siguiente función de pérdida:

$$max(\|s_a - s_p\| - \|sa - s_n\| + \epsilon, 0)$$

con s_x el *embedding* de sentencia para a/n/p, $\|\cdot\|$ una métrica de distancia y un margen ε. El margen ε asegura que s_p está al menos más ε cercano a s_a que a s_n. Como métrica se utiliza la distancia euclídea y se fija ε=1 en los experimentos.

Se entrena SBERT en la combinación de *datasets* SNLI (Bowman et al., 2015) y Multi-Genre NLI (Williams et al., 2018). El SNLI es una colección de 570.000 pares de oraciones anotados con las etiquetas

contradiction, *entailment* y *neutral*. MultiNLI contiene 430.000 pares de oraciones y cubre un rango de géneros de textos transcrito y escrito. Se ajusta SBERT durante una época con una función objetivo softmax para 3 clases. Se usa un tamaño de lote de 16, optimizador Adam con ratio de aprendizaje 2e-5, y un *warmup* lineal sobre el 10% de los datos de entrenamiento. La estrategia de *pooling* por defecto es MEAN.

■ **Evaluación de la Similitud textual semántica.** Se evalúa el rendimiento de SBERT para tareas comunes de similitud textual semántica. Los métodos SOTA a menudo aprenden una función de regresión compleja que hace corresponder los *embeddings* de sentencia a la puntuación de similitud. Sin embargo, estas funciones de regresión trabajan por parejas y debido a la explosión combinatoria no son escalables si la colección de oraciones alcanza cierto tamaño. Por ello siempre se usa similitud coseno para comparar la similitud entre dos *embeddings* de sentencias. Los autores experimentaron también con la distancia negativa de Manhattan y la distancia negativa euclídea pero los resultados para las tres distancias fueron aproximadamente los mismos.

Se analizaron las diferentes estrategias de pooling (MEAN, MAX y CLS). Para la función objetivo de clasificación, se evaluaron diferentes métodos de concatenación. Para cada posible configuración se entrenó SBERT con 10 semillas aleatorias diferentes y se promediaron los rendimientos.

La función objetivo (clasificación o regresión) depende del *dataset* anotado. Para la función objetivo clasificación, se entrena SBERT-base en los datasets SNLI y Multi-NLI. Para la función objetivo regresión, se utiliza el *dataset* STS.

Cuando se entrena con la función objetivo de clasificación en datos NLI, la estrategia de *pooling* tiene un impacto menor. La influencia del modo de concatenación es mucho mayor. Se utiliza como entrada al clasificador softmax (u, v, |u – v|). El componente más importante es la diferencia escalar |u – v|. Nótese, que el modo de concatenación es sólo relevante durante el entrenamiento del clasificador softmax. En inferencia, cuando se predicen las similitudes para el *dataset* STS sólo se utilizan los *embeddings* de las sentencias u y v en combinación con

la similitud semántica. La diferencia escalar |u – v| mide la distancia entre las dimensiones de los *embeddings* de las dos oraciones asegurando que pares similares están más cerca y pares diferentes están más separados.

Cuando se entrena con la función objetivo de regresión, se observa que la estrategia de *pooling* tiene un gran impacto. El rendimiento de la estrategia MAX obtiene peores resultados que las estrategia MEAN o CLS.

■ **Resultados.** SBERT entrenado en datos NLI con función objetivo de clasificación, estrategia de *pooling* media (MEAN) y función de concatenación (u, v, |u – v|) alcanza una puntuación de 80.78. SBERT entrenado en datos STSb con función objetivo de regresión y estrategia de *pooling* (MEAN) obtiene una puntuación de 87.44. Las configuraciones se evaluaron en el conjunto de desarrollo del *benchmark* STSb utilizando similitud coseno y correlacción Spearman.

■ **Conclusiones.** SBERT se puede utilizar para tareas que computacionalmente no son viables de formular con BERT. Por ejemplo, el *clustering* jerárquico de 10,000 oraciones requiere alrededor de 65 horas con BERT ya que deben calcularse 50 millones de combinaciones de oraciones aproximadamente . Con SBERT, se puede reducir el esfuerzo a unos 5 segundos.

2.1.7 ColBERT: Contextualized Late interaction over BERT

ColBERT (*Contextualized Late interaction over BERT*) (Khattab and Zaharia, 2020) propone un nuevo paradigma denominado interacción tardía para estimar la relevancia entre una consulta *q* y un documento *d*. En la interacción tardía, *q* y *d* se codifican por separado en dos conjuntos de *embeddings* contextuales y se evalúa la relevancia entre ambos conjuntos mediante cálculos con baja demanda computacional y fáciles de podar obteniendo un ranking sin necesidad de realizar una evaluación exhaustiva de todos los posibles candidatos.

La arquitectura de ColBERT consiste en (a) un codificador de consultas f_Q, (b) un codificador de documento f_D y (c) el mecanismo de

interacción. Dados una consulta *q* y un documento *d*, f_Q codifica la consulta en una bolsa de *embeddings* de tamaño fijo E_q mientras que f_D codifica *d* en otra bolsa E_d. Todos los *embeddings* en E_q y E_d están contextualizados en base a los términos de *q* y *d*, respectivamente.

Utilizando E_q y E_d, ColBERT computa la puntuación de relevancia entre *q* y *d* mediante el mecanismo denominado interacción tardía que se define como un sumatorio de similitudes maximales. En partircular, se calcula la similitud coseno maximal de cada vector $v \in E_q$ con cada vector en E_d y se combinan las salidas mediante un sumatorio. Además de la similitud coseno, los autores también evalúan el cuadrado de la norma L2 como medida de similitud entre vectores.

El mecanismo de interacción tardía busca la puntuación de similitud mayor entre cada término t_q de la consulta y términos del documentos. Los términos t_q se representan por *embedding* contextuales en la consulta e_q, y los términos t_d por *embeddings* contextuales del documento. Se estima la relevancia del documento sumando las similitudes de todos los términos de la consulta.

■ **Codificadores de consultas y documentos.** Antes de la interacción tardía, ColBERT codifica cada consulta o documento en una bolsa de embeddings empleando codificadores BERT. Los codificadores de consultas y documentos comparten un modelo BERT. Para distinguir las secuencias de entrada que corresponden a las consultas y documentos se preceden mediante un token especial [Q] para consultas y [D] para documentos.

Codificador de consultas. Dada una consulta q, se *tokeniza* en sus *tokens* WordPiece $q_1\ q_2\ \ldots\ q_l$. Se antepone a los *tokens* el *token* especial [Q]. La secuencia resultante se precede del *token* de comienzo [CLS]. Si la consulta tiene menos de un número predefinido de *tokens* N_q, se rellena con el *token* especial [MASK] hasta la longitud N_q (en caso contrario, se truncan los N_q primeros *tokens*). Esta secuencia de *tokens* de entrada se pasa a BERT que computa una representación contextualizada de cada *token*.

Dada la representación BERT de cada *token*, el codificador pasa la salida contextualizada a través de una capa lineal sin activaciones. Esta capa sirve para controlar la dimensión de los *embeddings* de ColBERT

produciendo *embeddings* de dimensión m (la dimensión de la capa de salida). Mientras que la dimensión de los *embeddings* de ColBERT tiene un impacto limitado en la eficiencia de la codificación de la consulta, este paso es crucial para controlar el espacio requerido para representar los documentos y el tiempo de ejecución de la consulta. En particular, el tiempo que lleva transferir las representaciones de los documentos a la GPU desde la memoria del sistema. Finalmente, los *embeddings* de salida son normalizados para que la norma L2 sea igual a uno. El resultado es que el producto escalar de cualquier par de *embeddings* es equivalente a su similitud coseno perteneciendo al intervalo [-1, 1].

Codificador de documentos. El codificador de documentos tiene una arquitectura similar. Primero se segmenta un documento d en sus *tokens* constituyentes d_1 d_2 … d_m, se precede la secuencia del *token* de comienzo BERT [CLS] y del *token* especial [D] que indica que se trata de la secuencia de un documento. A diferencia de las consultas, no se añaden *tokens* [MASK] a los documentos. Después de pasar la secuencia de entrada a través de BERT y de la posterior capa lineal, el codificador de documentos filtra los *embeddings* correspondientes a signos de puntuación mediante una lista predefinida. Este filtrado reduce el número de *embeddings* por documentos. Los autores afirman que los *embeddings* de signos de puntuación no son necesarios para la eficacia de ColBERT.

En resumen, dado q = q_0 q_1 … q_l y d = d_0 d_1 … d_n, se computan las bolsas de embeddings E_q y E_d según el siguiente esquema:

E_q := Normalize(CNN(BERT(“[Q] q_0 q_1 … q_l # # … #”))) donde # representa el token [mask]
E_d := Filter(Normalize(CNN(BERT(“[D] d_0 d_1 … d_n”)))

■ **Interacción tardía.** Dada la representación de una consulta q y un documento d, la puntuación de relevancia entre d y q, denotada como $S_{q, d}$, se estima mediante la interacción entre sus bolsas de *embeddings* contextualizados calculando una suma de similitudes maximales (implementada como productos escalares debido a la normalización de *embeddings*) o el cuadrado de la distancia L2.

$$S_{q,d} := \sum_{i \in [|E_q|]} \max_{j \in [|E_d|]} E_{q_i} \cdot E_{d_j}^T$$

ColBERT es diferenciable de extremo a extremo. Se ajustan los codificadores BERT y se entrenan desde cero los parámetros adicionales, la capa lineal y los *embeddings* de los marcadores [Q] y [D] usando optimizador Adam. El mecanismo de interacción de ColBERT no tiene parámetros entrenables. Dada una terna (q, d^+, d^-) con la consulta q, un documento positivo d^+ y un documento negativo d^-, ColBERT se usa para producir una puntuación para todos los documentos individualmente y está optimizado mediante una función de pérdida de entropía cruzada softmax por pares sobre las puntuaciones calculadas de d^+ y d^-.

2.2 Trabajos relacionados

En el marco del *track* para TAC2020: *Epidemic QuestionAnswering* (EPIC-QA), once equipos desarrollaron sistemas capaces de responder preguntas sobre la enfermedad COVID-19, el virus que la causa SARS-CoV-2, otros coronavirus relacionados y la respuesta recomendada a la pandemia. Las siguientes tablas muestran los resultados obtenidos para las tareas *expert* y *consumer* en el ciclo de evaluación *preliminary* por los sistemas presentados:

Expert Preliminary

	NDNS-Partial	NDNS-Relaxed	NDNS-Exact
1	0.305 ixa-3	0.307 ixa-3	**0.341 ixa-3**
2	0.303 ixa-2	0.304 ixa-2	0.338 ixa-2
3	0.302 HLTRI-1	0.295 IBM-3	0.327 IBM-1
4	0.294 IBM_3	0.294 IBM_1	0.327 IBM_2
5	0.294 IBM_1	0.293 IBM_2	0.325 IBM_3
6	0.293 IBM_2	0.293 HLTRI_1	0.306 ixa_1
7	0.276 ixa_1	0.277 ixa_1	0.300 vigicovid_2
8	0.266 vigicovid_2	0.267 vigicovid_2	0.297 vigicovid_3
9	0.263 vigicovid_3	0.265 vigicovid_3	0.289 HLTRI_1

Tabla 8: Resultados de la evaluación preliminary en la tarea A usando Normalized Discounted Novelty Score (NDNS)

Consumer Preliminary

	NDNS-Partial	NDNS-Relaxed	NDNS-Exact
1	0.488 HLTRI_1	0.482 HLTRI_1	**0.475 HLTRI_1**
2	0.398 IBM_1	0.396 IBM_1	0.413 IBM_1
3	0.394 IBM_3	0.394 IBM_3	0.409 IBM_3
4	0.374 IBM_2	0.372 IBM_2	0.390 IBM_2

Tabla 9: Resultados de la evaluación preliminary en la tarea B usando Normalized Discounted Novelty Score (NDNS)

2.2.1 ixa - Ixa NLP Group at the University of the Basque Country (Otegi et al., 2020)

El sistema presentado por el Grupo de Procesamiento de Lenguaje Natural de la Universidad del País Vasco se compone de dos subsistemas principales:

1. **Módulo de recuperación de información IR.** Indexa el texto completo de los artículos siguiendo las acciones de procesamiento habituales consistentes en *tokenizar, lematizar*, pasar a minúsculas el texto y eliminar las *stopwords*. La unidad de indexación es el párrafo. El corpus de documentos indexados es un subconjunto obtenido al filtrar el *dataset* CORD-19 mediante una lista de sinónimos para los términos relevantes. Por ejemplo, para COVID-19: “coronavirus 2019”, “ncov 2019”, “sarscov19”, etc; se comprueba si alguno de los sinónimos aparece en el título o en el resumen de un artículo.

   La recuperación de documentos relevantes a partir de una pregunta en lenguaje natural se realiza con el algoritmo de búsqueda BM25F (Zaragoza et al., 2004).

2. **Módulo de extracción de respuestas QA.** Dada una pregunta en lenguaje natural y un párrafo de un documento candidato, el módulo QA retorna la respuesta a la pregunta en el párrafo o *no answer* en otro caso. El módulo emplea SciBERT ajustado en los *datasets* SQuAD 2.0 y QuAC.

   La utilización de QuAC para el ajuste proporciona mejores respuestas que si solo se utiliza SQuAD. Ofrece respuestas de mayor longitud. Según los autores, los sistemas entrenados en SQuAD 2.0 pueden alcanzar métricas de evaluación automáticas

altas pero fallan al atender las preferencias de usuario en casos de usos más próximos al desempeño real. Demuestran mediante un test A/B adicional que los usuarios prefieren sistemas entrenados también en QuAC, es decir, SQuAD 2.0 + QuAC.
No obstante, añadir el *dataset* QuAC a la fase de ajuste causa que los resultados del sistema se reduzcan en 20 puntos cuando se evalúan con la métrica F1. Estos valores contradicen el análisis manual previo.

El sistema final combina ambos módulos IR y QA en un *pipeline*. Para cada una de las preguntas, se recuperan los veinte párrafos más relevantes usando el módulo IR. A continuación, se ejecuta el módulo QA sobre los párrafos para extraer respuestas candidatas. Si el 85% de estas 20 respuestas son del tipo *no answer*, la pregunta en curso no se responde para no sobrecargar al usuario con malas respuestas.

2.2.2 IBM Submissions to EPIC-QA Open Retrieval Question Answering on COVID-19

Para cada tarea, experto o consumidor, IBM presentó tres propuestas con un *pipeline* estándar de tres etapas: recuperación de información, extracción de fragmentos y *reranking* de respuestas cambiando las implementaciones de los módulos de recuperación de pasajes como se describe a continuación:

IBM-1 Utiliza Elastic Search (BM25) para recuperación de pasajes. Reordena los pasajes usando un clasificador BERT entrenado en datos y evaluaciones de relevancia procedentes del conjunto de entrenamiento de la tarea 8b de BioASQ y la ronda de evaluación *preliminary* de EPIC-QA.

IBM-2 El módulo de recuperación de información de este sistema es diferente al empleado en el sistema anterior; el resto del *pipeline* se mantiene. La recuperación de documentos relevantes se realiza ensamblando el módulo IR del sistema IBM-1 con un recuperador de pasajes denso (DPR, Dense Passage Retriever) (Karpokhin et al., 2020) entrenado en datos etiquetados manualmente. Siguiendo el trabajo de (Reddy et al., 2020) se ajusta BART (Lewis et al., 2020) en ejemplos SQuAD 2.0 (Rajpurkar et al., 2018) y se usan los artículos de la colección CORD-19 (Wang et al., 2020) para generar pares sintéticos de

pregunta-pasaje. Con el modelo BART entrenado, se codifican los pasajes de las colecciones *expert* y *consumer* de EPIC-QA y se almacenan en un índice usando FAISS (Johnson et al., 2019). En tiempo de inferencia se utiliza vecinos más cercanos para encontrar similitud entre la pregunta del usuario y las preguntas generadas extrayendo así los pasajes relevantes.

IBM-3 Este sistema usa un tercer algoritmo de recuperación de pasajes ensamblando BM25 y un recuperador de pasajes denso adaptado al dominio COVID-19. Las etapas de extracción de respuestas y reranking no se modifican.

Los módulos IR de IBM-2 e IBM-3 son sistemas ensamblados que computan una combinación de las puntuaciones asignadas a un pasaje. El conjunto IBM-2 con mejor desempeño ponderó la puntuación BM25 por 0.4 y la puntuación DPR por 0.6. En IBM-3, el mejor rendimiento se logró al ponderar el clasificador neuronal por 0.7 y la puntuación combinada BM25 + DPR de IBM-2 por 0.3.

En las tres ejecuciones, el componente IR devuelve los primeros 3.000 pasajes que después se hacen corresponder con sus documentos de origen. Los 1.000 documentos mejor puntuados pasan a través del módulo de extracción de respuestas.

A partir de los pasajes candidatos, el módulo de extracción de respuestas obtiene fragmentos de texto usando el sistema de comprensión lectora (MRC) GAAMA (Chakravarti et al., 2020). El módulo final emplea los fragmentos de las respuestas extraídas de todos los pasajes candidatos para seleccionar y reordenar la lista de respuestas que se muestra al usuario.

GAAMA ajusta un modelo RoBERTa$_{\text{LARGE}}$ (Liu et al., 2019) para QA. Los autores prueban con dos ejemplares de GAAMA. Un primer modelo de dominio abierto ajustado en SQuAD 2.0 (Rajpurkar et al., 2018) durante 2 épocas y después en el *dataset Natural Questions* (Kwiatkowski et al., 2019) durante 1 época. El segundo modelo GAAMA se ajusta específicamente para el dominio COVID-19, primero entrenando sobre texto sin procesar de la colección de documentos CORD-19 y luego con muestras sintéticas generadas a partir de los documentos CORD-19 usando BART (Lewis et al., 2020).

Para EPIC-QA, se experimentó con ambos modelos GAAMA. En los experimentos con el conjunto de preguntas del ciclo de evaluación *preliminary*, se encontró que la puntuación NDNS promedio del sistema GAAMA de dominio abierto era aproximadamente 1,5 puntos superior a la puntuación NDNS promedio del sistema GAAMA adaptado al dominio COVID-19.

Previamente al reranking de respuestas se eliminan respuestas duplicadas con técnicas simples para medir la similitud de oraciones. Las oraciones contiguas se combinan y verifican que están dentro de los límites de los pasajes especificados en la colección de documentos. La puntuación final del fragmento extraído es la media ponderada de la puntuación IR de su documento y la puntuación MRC más alta en el conjunto de oraciones que forman el fragmento. Se establecieron los pesos de 0.7 para IR y 0.3 para MRC mediante el análisis del conjunto de desarrollo COVID-QA (Reddy et al., 2020).

2.2.3 The University of Texas at Dallas HLTRI's Participation in EPIC-QA

El sistema, denominado SER4EUNOVA, se estructura en cuatro módulos: módulo de ranking de contextos, módulo de generación de preguntas, módulo de reconocimiento de vinculación entre preguntas y módulo de re-ranking de respuestas.

Dada una pregunta y la colección de documentos utilizadas en una de las tareas EPIC-QA, el módulo de ranking de contextos recupera oraciones ordenadas a partir del conjunto de documentos candidatos a contener las respuestas a la pregunta. A partir de cada oración candidata, el módulo de generación de preguntas genera preguntas equivalentes. Los pares de preguntas se pasan a través del módulo de reconocimiento de vinculación entre preguntas para descubrir qué preguntas están textualmente vinculadas. Los pares de preguntas constan bien de preguntas que fueron generadas desde las oraciones candidatas o de la pregunta original emparejada con una de las preguntas generadas. Las relaciones de vinculación descubiertas permiten la generación de un grafo de vinculación entre preguntas (EQG, *Entailment Question Graph)* donde los nodos son preguntas y los vértices indican las relaciones de vinculación descubiertas.

El módulo de reranking de respuestas busca en el grafo EQG las preguntas que responden los *nuggets* de la pregunta original. Esto es posible determinando en el grafo EQG los componentes conectados más grandes: las preguntas que (1) están vinculadas por la pregunta original y (2) tienen la mayor conectividad en cada componente conectado. Basándose en esta suposición, las respuestas son reordenadas para favorecer aquellas oraciones que contienen el mayor número de *nuggets* de respuesta. Esta lista es la salida del sistema SER4EQUNOVA.

■ **Módulo de ranking de contextos.** El primer módulo de SER4EQUNOVA es el módulo de ranking de contextos que, para una pregunta dada, se encarga de encontrar respuestas candidatas en la colección de documentos *expert* o *consumer*. Los corpus experto y consumidor están indexados con el software Lucene y la búsqueda de contextos relevantes con BM25 (Robertson et al., 1994). A continuación, se consideran todas las oraciones dentro de cada contexto y mediante un sistema BERT-RERANK se calcula una puntuación de reordenación para cada pregunta seleccionando las 1000 primeras oraciones de este ranking como entrada al módulo de generación de preguntas.

El sistema BERT-RERANK se entrenó de forma similar al modo descrito en el trabajo previo de (Nogueira y Cho, 2020). Se consideran las oraciones que contienen un *nugget* para una pregunta dada como una oración “relevante” y aleatoriamente se muestrean las oraciones “no relevantes” de las 1000 oraciones seleccionadas. El entrenamiento se realiza en la colección preliminar proporcionada por los anotadores del reto. Se entrenaron modelos separados para cada una de las tareas A y B. Se inicializaron los pesos BERT a un modelo de reranking que se entrenó en MS MARCO (Nguyen et al., 2016) usando BioBERT (Lee et al., 2019). Además, la función de pérdida se ponderó multiplicando la perdida de las oraciones “relevantes” por el número de *nuggets* en la oración. De este modo, en el reranking se prioriza las oraciones con más *nuggets.*

■ **Módulo de generación de preguntas.** El módulo de generación de preguntas produce preguntas generadas para cada respuesta candidata proporcionada por el módulo de ranking de contextos. Las preguntas son generadas con el modelo docTTTTTquery (Nogueira, 2019)

entrenado en MS MARCO. Para cada respuesta candidata se generan k preguntas. La generación de preguntas puede realizarse completamente *offline* del corpus o ejecutarse ad hoc como un proceso paralelo con respecto a cada respuesta. Un ejemplo de la colección *Expert QA*. Para la pregunta q_0: "What is the origin of COVID-19" se ilustra un subconjunto de las respuestas candidatas producidas por el módulo de ranking de contextos y un subconjunto de preguntas generadas para cada respuesta:

q0. What is the origin of COVID-19
- a1. The outbreak of COVID-19 originated from Wuhan City, Hubei province, in China.
- a2. The causative agent of COVID-19 es a novel coronavirus known as SARS-CoV-2.
- a3. …

Cada respuesta se conecta a una o más preguntas que fue generada desde la respuesta. Por ejemplo, la respuesta "The outbreak of COVID-19 originated from Wuhan City, Hubei province, in China." produce la pregunta "Where did COVID-19 originate?".

■ **Módulo de reconocimiento de vinculación de preguntas.** El grafo de preguntas vinculadas requiere un sistema de reconocimiento de vinculación entre preguntas (RQG, *Recognizing Question Entailment*) (Ben Abacha and Demner-Fushman, 2019) para identificar relaciones entre pares de preguntas vinculadas. Los autores experimentan con dos tipos de sistemas RQE: BERT-RQE y QBERT-RQE. BERT-RQE utiliza un modelo de lenguaje BERT (Devlin et al., 2019) preentrenado con una capa lineal simple que predice relaciones entre dos pares de preguntas con dos etiquetas: "vinculado", "no vinculado". El segundo sistema RQE, QBERT-RQE utiliza la misma arquitectura, pero inicia sus pesos a un modelo MSMARCO-BERT-RERANK entrenado en pares de pregunta-respuesta desde MSMARCO para predecir "relevante", "no relevante" (Nogueira y Cho, 2020) con la suposición de que la relevancia entre pregunta-respuesta es similar a la vinculación entre pregunta-pregunta. Ambos modelos fueron ajustados en el conjunto de preguntas duplicadas Quora[10] y alcanzaron valores de precisión (%) de 88.94 BERT-RQE y 89.55 QBERT-RQE.

10 https://www.kaggle.com/c/quora-question-pairs

La secuencia: [CLS] *tokens wordpiece* de la pregunta A [SEP] *tokens wordpiece* de la pregunta B [SEP] se pasa al modelo MSMARCO-BERT-RERANK para obtener el *embedding* contextualizado del *token* [CLS]. Como en otras tareas de clasificación, se utiliza el *token* [CLS] para alimentar una capa simple, en este caso de detección de vinculación que calcula la probabilidad de vinculación entre las preguntas A y B.

■ **Módulo de reranking de respuestas.** El módulo de reranking de respuestas se encarga de ordenar las respuestas proporcionadas por el módulo de ranking de contextos en base a la novedad de los *nuggets* de cada respuesta. Las preguntas generadas por el módulo de generación de preguntas se utilizan para construir un grafo de preguntas vinculadas empleado para determinar si un *nugget* responde preguntas. La presencia de un *nugget* que responde preguntas proporciona la información necesaria para reordenar las respuestas. La salida del módulo de reranking de respuestas es una lista ordenada de respuestas de mayor a menor puntuación determinada en función de si contienen más o menos *nuggets* nuevos en relación con otras respuestas ya incluidas en los resultados.

■ **Grafo de preguntas vinculadas.** La generación del grafo de preguntas vinculadas confía en la predicción de la probabilidad de vinculación entre pares de preguntas. Los nodos del EQG son las preguntas generadas por el módulo de generación de preguntas que también comparte algunas relaciones de vinculación. La pregunta inicial q_0 se considera externa al grafo. Sin embargo, se tienen en cuenta las relaciones vinculadas entre q_0 y cualquiera de los nodos de preguntas del EQG.

El EQG se utiliza para descubrir las preguntas generadas automáticamente que son respondidas por los *nuggets* de q_0, la pregunta original. Para descubrir estas preguntas, primero se buscan todos los componentes conectados del EQG. Desde cada componente del EQG, se seleccionan aquellas preguntas que (1) se conectan a través del número mayor de relaciones de vinculación a otras preguntas desde el mismo componente; y (2) también están vinculadas por q_0. Después, las preguntas seleccionadas se filtran eliminando aquellas cuya probabilidad de vinculación con q_0 está por debajo de un valor umbral preestablecido.

■ **Reordenación de respuestas.** El módulo de reordenación de respuestas toma en cuenta el *nugget* de la respuesta y vuelve a clasificar las sentencias de la respuesta recuperadas por el módulo de clasificación de contexto. El objetivo no es solo contar el número de *nuggets* de respuestas que se generaron a partir de cada oración, sino también tener en cuenta la novedad de los *nuggets* de respuesta. Eso implica realizar un seguimiento de cada *nugget* de respuesta que ya se conoce y distinguir los *nuggets* de respuesta novedosos. Cuando las preguntas se generan automáticamente en el módulo de generación de preguntas, para cada pregunta se conoce el fragmento de texto de la respuesta. Se realiza un seguimiento de cada fragmento de respuesta para considerar posibles *nuggets* para la pregunta original q_0.

Cada respuesta a partir de la cual se generó el *nugget* puede contener *nuggets* conocidos que se han observado en respuestas previas, o *nuggets* novedosos que aún no se han visto. Se prefieren respuesta que contienen *nuggets* que tienen poca o ninguna superposición con los *nuggets* vistos anteriormente. Teniendo esto en cuenta, se modifica la clasificación producida por el módulo de clasificación de contextos haciendo uso del mismo algoritmo NDNS utilizado por los organizadores de EPIC-QA para calcular una clasificación óptima de respuestas basada en *nuggets* dentro de cada respuesta.

2.3.4 Covidex: Neural Ranking Models and Keyword Search Infrastructure for the COVID-19 Open Research Dataset

Covidex (Zhang et al., 2020) es un motor de búsqueda sobre el *dataset* CORD-19. Utiliza Anserini IR toolkit (Yang et al., 2017, 2018) para la recuperación inicial de documentos candidatos y MonoT5 como *reranker*. La unidad de indexación es el párrafo. Como módulo de obtención de respuestas emplea BioBERT como se describe en (Zhang et al., 2020b)

MonoT5. A pesar del éxito de BERT para clasificación de documentos (Dai and Callan, 2019; MacAvaney et al., 2019; Yilmaz et al., 2019) existen evidencias de que la clasificación con modelos secuencia-a-secuencia puede alcanzar incluso mejores resultados, particularmente en *zero-shot* y otras configuraciones con datos de entrenamiento limitados (Nogueria et al., 2020); monoT5 está basado en T5 (Raffel et al., 2019).

Dada una consulta q y un conjunto de documentos candidatos D, para cada $d \in D$ se construye la siguiente secuencia de entrada para alimentar el modelo:

$$Query: q, Document: d, Relevant$$

El modelo se ajusta para producir *true* o *false* dependiendo de si el documento es relevante o no a la consulta. Es decir, *true* y *false* son las palabras objetivo a predecir en la tarea de secuencia-a-secuencia.

En tiempo de inferencia, se computan las probabilidades de cada pareja consulta-documento, se aplica softmax sólo a los *logits* de los *tokens true* y *false*. Se reordenan los documentos candidatos de acuerdo a las probabilidades asignadas al *token true*. (Nogueira et al., 2020) contiene detalles adicionales sobre la normalización *logit* y los efectos de diferentes palabras *logit*.

Dado que no se dispone de datos de entrenamiento específicos para ranking de documentos COVID-19, se ajusta el modelo en el conjunto de pasajes MS MARCO (Nguyen et al., 2016) que consta de 8.8M de pasajes obtenidos desde el top 10 de resultados recuperados por el motor de búsqueda Bing (basado en alrededor de 1M de consultas). El conjunto de entrenamiento contiene aproximadamente 500K pares de consultas y documentos relevantes; en promedio cada consulta tiene un pasaje relevante. También se proporcionan documentos no relevantes como parte del conjunto de datos de entrenamiento. (Nogueira et al., 2020) y (Yilmaz et al., 2019) han demostrado que modelos entrenados en MS MARCO pueden aplicarse directamente a otras tareas de ranking.

Se ajusta el modelo monoT5 con un ratio de aprendizaje constante de 10e-3 para 10K iteraciones con lotes de tamaño 128 balanceados por clases. Se usa un máximo de 512 *tokens* de entrada y una salida de un *token* {*true*, *false*}. En el conjunto de pasajes MS MARCO, ninguna de las entradas necesita ser truncada cuando se emplea este límite de longitud. Las variantes de entrenamiento basadas en T5-base y T5-3B toman aproximadamente 4 y 40 horas, respectivamente, en una TPU v3-8.

En tiempo de inferencia, dado que los documentos recuperados son mayores que las restricciones de longitud del modelo, no es posible alimentar con el texto completo el modelo. Para tratar esta cuestión, primero se segmenta cada documento en fragmentos aplicando una ventana de 10 sentencias con un deslizamiento de 5. Se obtiene una probabilidad de relevancia para cada ventana realizando inferencias independientes y se selecciona la probabilidad más alta entre los fragmentos como la puntuación de relevancia del documento.

Para reducir la latencia de extremo a extremo se reordenan sólo los 96 documentos top de cada consulta y se trunca la entrada del reranker a 256 *tokens* como máximo. La latencia media de extremo a extremo es de 2 segundo.

El sistema se despliega en un pequeño *cluster* de servidores cada uno con dos GPUs NVIDIA V100 ya que el *pipeline* requiere inferencia neuronal en cada consulta. Cada servidor ejecuta la pila de software completa en una configuración simple replicada, no existe particionado.

2.3.5 Vigicovid Question Answering system at EPIC-QA

VIGICOVID QA (Otegi et al., 2020) tiene una arquitectura de tres etapas: recuperación de documentos; extracción de respuestas y ranking de respuestas.

■ **Recuperación de documentos.** El módulo de recuperación de información sigue tres pasos: filtrado e indexación, recuperación preliminar, reordenación de documentos candidatos.

1. Filtrado e indexación. Los documentos relevantes se obtienen mediante filtrado por palabra clave. Las palabras claves son variantes del término “COVID-19”. La unidad de indexación es el documento completo.

2. Recuperación preliminar. A partir de la colección del texto completo de los artículos científicos, se obtiene un ranking preliminar para la consulta. Se utiliza una aproximación no neuronal.

3. Reranking. El ranking obtenido en el paso anterior se reordena con un clasificador BERT siguiendo una estrategia similar a la propuesta por (Nogueira and Cho, 2019).

Después del proceso de reranking se selecciona el top 400 de documentos para ser escaneado por el módulo de extracción de respuestas.

■ **Módulo de extracción de respuestas.** Emplea SciBERT como modelo de lenguaje ajustado para QA usando los *datasets*: SQuAD 2.0 (Rajpurkar et al., 2018) y QuAC (Choi et al., 2018).

Dado que los documentos son demasiado grandes para alimentar la red, se emplea una estrategia de ventana deslizante con solapamiento de pasajes. Se utilizan los valores por defecto de la implementación (Wolf et al., 2020) para los parámetros longitud máxima de secuencia y deslizamiento (*stride*). Después de escanear el documento completo, se conservan las 10 respuestas más probables a la pregunta para cada documento dado. La puntuación final de los fragmentos es la suma de la puntuación de la respuesta dada por la red neuronal y la puntuación del documento asignada por el motor de búsqueda.

Capítulo 3

Marco de evaluación

3.1 Metodología de evaluación

La evaluación se realiza comparando los resultados del sistema propuesto con las puntuaciones obtenidas por los sistemas participantes en la tarea *expert* del reto EPIC-QA. La métrica de evaluación empleada en el reto es NDNS, *Normalized Discount Novelty Score*, una métrica que penaliza la repetición de respuestas. Para la evaluación, la organización desarrolló un *script* en *Python* que toma como entrada un archivo con el siguiente formato:

```
QUESTION_ID Q0 START_SENTENCE_ID:END_SENTENCE_ID RANK SCORE RUN_NAME
```

donde QUESTION_ID es el identificador de la pregunta en el archivo json de preguntas: EQ001, EQ002, ...; Q0 es una constante necesaria por motivos de compatibilidad; START_SENTENCE_ID y END_SENTENCE_ID son los identificadores de comienzo y fin de una serie de oraciones contiguas pertenecientes al mismo contexto del documento; RANK es el número de orden [1, 1000] de la respuesta en la lista de respuestas recuperadas para la pregunta QUESTION_ID; SCORE es un valor numérico que representa la puntuación asociada a la respuesta y RUN_NAME es un nombre para identificar la ejecución.

3.2 Métrica de evaluación

NDNS, ***Normalized Discount Novelty Score*** es una versión modificada de Normalized Discount Cumulative Gain, NDCG que favorece la exploración de respuestas en la colección de documentos promoviendo que el sistema proporcione una lista diversa de respuestas. Una respuesta se considera relevante si y sólo si describe un conjunto de hechos (*nugget*) que responde la pregunta y no ha sido incluida en ninguna de las respuestas previas de la lista ordenada de resultados. Las respuestas se penalizan en base a sus longitudes en número de sentencias.

NDNS se define como:

$$NS_i = \frac{n_i \times (n_i + 1)}{n_i + SF_i}$$

donde n_i es el número de *nuggets* nuevos en una respuesta i y SF_i es un término de penalización denominado factor de sentencia. Se definen tres factores de sentencia diferentes que determinan tres métricas NDNS:

- NDNS-Relajada: las respuestas sólo deberían contener oraciones con *nuggets* novedosos:

$$SF_i = na_i + sn_i + min(nn_i, 1)$$

- NDNS-Parcial: las respuestas no deberían contener oraciones sin *nuggets:*

$$SF_i = na_i + min(nn_i, 1)$$

- NDNS-Exacta: las respuestas deberían ser cortas:

$$SF_i = na_i + sn_i + nn_i$$

donde na_i es el número de oraciones sin *nuggets*, sn_i es el número de oraciones con *nuggets* ya vistos, y nn_i es el número de oraciones con *nuggets* nuevos.

■ Normalized Discount Cumulative Gain (NDCG)

La Ganancia Acumulada Descontada Normalizada (NDCG, Normalized Discount Cumulative Gain) se define a partir de:

1) un espacio de muestras X con n objetos a ordenar: $x_{1,}\ldots, x_n(x_i \in X)$;
2) un conjunto finito Y de grados de relevancia. El caso más simple es $Y=\{0,1\}$ donde 0 corresponde a "irrelevante" y 1 corresponde a "relevante";
3) una función f de ranking. Se asume que f es una aplicación de $X \rightarrow \mathbb{R}$.

Para cada objeto $x \in X$, f asigna una puntuación f(x). Para cada n objetos x_1, …, x_n, f los clasifica de acuerdo a sus puntuaciones f(x_1), …, f(x_n). La lista ordenada resultante, denotada como $x^f_{(1)},\ldots,x^f_{(n)}$ satisface que $f(x^f_{(1)}) \geq \ldots \geq f(x^f_{(n)})$. Sea $y_1,\ldots,y_n(y_i \in Y)$ el grado de relevancia asociado con x_1, …, x_n. Se denotará por $S_n=\{(x_1,y_1),\ldots,(x_n,y_n)\}$ el conjunto de datos a ordenar.

Sean $D(r)(r \geq 1)$ una función de descuento, f una función de clasificación y S_n un conjunto de datos. La Ganancia Acumulada Descontada (DCG, Discounted Cumulative Gain) de f en S_n con descuento D se define como

$$DCG_D(f,S_n)=\sum_{r=1}^{n} y^f_{(r)} D(r)$$

Sea la DCG ideal el valor de DCG de la mejor función de clasificación en S_n definida como

$$IDCG_D(S_n)=max_{f'}\sum_{r=1}^{n} y^{f'}_{(r)} D(r)$$

Se define la Ganancia Acumulada Descontada Normalizada (NDCG) de f en S_n con descuento D como:

$$NDCG_D(f, S_n) = \frac{DCG_D(f, S_n)}{IDCG_D(S_n)}$$

Se denomina NDCG estándar cuando la función de descuento asociada es la inversa del decaimiento logarítmico $D(r) = \frac{1}{\log(1+r)}$. La base del logaritmo no es importante para NDCG dado que el factor de escalado se cancela debido a la normalización.

Una propiedad importante de NDCG es que si la función de clasificación f' preserva el orden de la función de clasificación f, entonces $NDCG_D(f', S_n) = NDCG_D(f, S_n)$ para todo S_n. Preservar el orden significa que $\forall x, x' \in X, f(x) > f'(x) \Rightarrow f'(x) > f'(x')$ y viceversa. Por tanto, la métrica de ranking NDCG no está solo definida en una función simple f sino que define una clase de equivalencia de funciones de ranking que preservan el orden entre ellas.

3.3 Colecciones de evaluación

EPIC-QA El sistema se evalúa en las colecciones de documentos *expert* de EPIC-QA; *expert* procede de la colección de documentos CORD-19. Todos los documentos de la colección siguen una versión modificada del esquema CORD-19 JSON, descrito a continuación.

CORD-19, *COVID-19 Open Research Dataset* (Wang et al, 2020) es un conjunto de datos abiertos en continuo crecimiento para la investigación sobre COVID-19 y coronavirus relacionados SARS y MERS. Está gestionado por el instituto Allen para la Inteligencia Artificial (AI2) en colaboración con la Oficina de Política Científica y Tecnológica de la Casa Blanca (OSTP), la Biblioteca Nacional de Medicina (NLM), la Iniciativa Chan Zuckerburg (CZI), Microsoft Research y Kaggle, coordinados por el Centro de Seguridad y Tecnología Emergente de la Universidad Georgetown (CSET).

CORD-19 fue diseñado para facilitar el desarrollo de sistemas de recuperación de información a partir de varias colecciones de documentos y metadatos. La mayor parte de los artículos proceden de la Iniciativa COVID-19 de Emergencia de Salud Pública[11] del Centro PubMed y en menor medida de los servidores bioRxiv y medRxiv así como de la base de datos COVID-19 de la Organización Mundial de la Salud. CORD-19 normaliza los contenidos de las diferentes fuentes y genera un corpus sin duplicados donde los artículos se representan mediante un esquema JSON denominado JSON CORD-19 diseñado para preservar la estructura de los artículos: secciones, encabezamientos, párrafos, citas o referencias.

Un documento en JSON CORD-19 tiene asociado un conjunto de metadatos normalizados: DOI, título, autores, lugar de publicación, fecha de publicación, identificador en PubMed o identificador de revisores entre otros campos:

```
{
  "document_id": <str>,     # 40-character sha1 of the URL or document
  "metadata": {
    "title": <str>,         # Title of the document
    "url": <str>,           # URL of the page (for webpages)
    "authors": [...]        # Authors of the page (for CORD-19)
  },
  "contexts": [             # List of context(s) in the document
  {
    "section": <str>,       # Name of the section (if any)
                            # containing the context
    "text": <str>,          # The full text (without markup) in the
                            # section
    "context_id": <str>,    # Globally unique context identifier
    "sentences": [
    {
      "start": 0,           # Inclusive character start offset of the
                            # sentence
      "end": 27,            # Exclusive character end offset of the
                            # sentence
      "sentence_id": <str>, # Globally unique identifier for
                            # the sentence
    },
    {                       # Second sentence in the context
      "start": 28,
      "end": 56,
```

11 https://www.ncbi.nlm.nih.gov/pmc/about/covid-19/

```
      "sentence_id": <str>,
    },
    {...},                  # Third sentence in the context
    …
    ]
  },
  {…},                      # Second context in the document
  …
  ]
}
```

Capítulo 4

Experimentación

Este capítulo evalúa la aplicación de modelos BERT entrenados en *datasets question-answering* de dominio general: SQuAD v2 (Rajpurkar et al., 2018) y QuAC v1.1 (Choi et al., 2018) a la tarea de responder preguntas del dominio COVID-19. En concreto, estudia dos aproximaciones a la obtención de respuestas para la tarea A: *Expert QA* del TAC 2020 Epidemic Question Answering:

1. Mediante extracción de fragmentos de texto usando modelos BERT.
2. Mediante similitud semántica entre la pregunta y el pasaje de la respuesta utilizando modelos SentenceBERT.

Las preguntas de investigación formuladas son las siguientes:

1. Modelos BERT: ¿qué modelo de lenguaje: BERT, BioBERT, ClinicalBERT o SciBERT demuestra ser más efectivo en la tarea de extracción de respuesta?

2. *Dataset* de *fine-tuning*: ¿qué conjunto de datos de entrenamiento obtiene mejores resultados: SQuAD, QuAC o SQuAD + QuAC?

3. Análisis de hiperparámetros: ¿cómo afectan los parámetros de *fine-tuning* en los modelos?

4. Aproximación a la obtención de respuestas: ¿es mejor utilizar una estrategia basada en q*uestion-answering o* en *sentence similarity*?

4.1 Metodología

4.1.1 Modelos BERT

El proceso general para responder las preguntas de investigación sobre modelos BERT emplea el siguiente esquema:

1. *Fine-tuning* del modelo bajo estudio en el conjunto de entrenamiento del *dataset* con los parámetros definidos en el experimento.
2. Evaluación con las métricas EM/F1 en el subconjunto de evaluación del *dataset* empleado para *fine-tuning*.
3. Evaluación con las métricas NDNS en la colección de documentos *preliminary expert* del corpus EPIC-QA.

El código[12] para implementar los procesos anteriores está escrito en Python y utiliza las librerías Transformers[13] y Datasets[14] de 🤗 Hugging Face[15]. La ejecución se realiza utilizando recursos del *cloud* de Google[16].

4.1.1.1 *Fine-tuning*

Consta de los siguientes pasos: (i) formateado del *dataset*, (ii) preprocesamiento del conjunto de entrenamiento, (iii) entrenamiento.

Formateado del *dataset.* El *dataset* SQuAD tiene la siguiente estructura de campos:

id: identificador de la pregunta, de tipo *string*;

12 El código de esta sección se basa en el ejemplo Question Answering on SQuAD publicado por Hugging Faces: https://colab.research.google.com/github/huggingface/notebooks/blob/master/examples/question_answering.ipynb
13 https://github.com/huggingface/transformers
14 https://github.com/huggingface/datasets
15 https://huggingface.co
16 https://cloud.google.com

title: título de la pregunta, de tipo *string*;
context: contexto desde donde se extrae la respuesta, de tipo *string*;
question: enunciado de la pregunta, de tipo *string*;
answers: diccionario de *arrays* con dos entradas:

- *text*: un array de fragmentos de respuestas, de tipo lista de *string*;
- *answer_start*: un array de comienzos de fragmentos de respuesta, de tipo lista de *int32*.

Un ejemplo extraído del conjunto de entrenamiento,

```
{
    "id":                  "56ddde6b9a695914005b9629",
    "title":               "Normans"
    "context":              "\"The Normans (Norman: Nourmands; French: Normands; Latin: Normanni) were the people who in the 10th and 11th centuries gave thei...",
    "question":            "When were the Normans in Normandy?",
    "answers": {
           "answer_start": [94, 87, 94, 94],
           "text":         ["10th and 11th centuries", "in the 10th and 11th centuries", "10th and 11th centuries", "10th and 11th centuries"]
    }
},
```

Si la pregunta no tiene respuestas, el campo *answers* contiene:

```
    "answers": {
           "answer_start": [0]
           "text":         []
    }
```

El modelo espera los índices de las palabras de comienzo y fin de la respuesta. Es necesario calcular el final de la respuesta *answer_end* a partir del comienzo *answer_start* más la longitud del texto *len(text)* para todos los campos *answers* de cada ejemplo de entrenamiento. Campo *answer_end* del ejemplo anterior:

```
"answer_end": [117, 117, 117, 117]
```

Para utilizar el *dataset* QuAC, es necesario realizar un paso previo que convierta su estructura de campos: *answers*, *background*, *context*, *dialogue_id*, *followups*, *wikipedia_page_title*, *yesno* a la estructura de SQuAD: *answers*, *context*, *id*, *question*, *title* perdiendo la información

de contexto y dando como resultado que los enunciados de algunas preguntas sean semánticamente incompletos debido a la pérdida de información que era deducible desde el contexto. Una vez realizada la conversión de esquemas, se formatea el *dataset* resultante como si fuera SQuAD v2.

Preprocesamiento del conjunto de entrenamiento. Consiste en:

1) *tokenizar* los pares de entrada (pregunta, contexto) convirtiendo las palabras en *wordpieces* y éstas en los ID del vocabulario de preentrenamiento;
2) generar las secuencias de entrada en el formato esperado por el modelo.

Cada modelo tiene su propio *tokenizador* porque el vocabulario se crea durante el preentrenamiento. El *tokenizador* necesita conocer la longitud máxima de la secuencia de entrada en el modelo y qué hacer en caso de que la longitud de la pregunta más el contexto sea superior. La estrategia seguida ha sido truncar el contexto. Cuando es necesario truncar un contexto, el *tokenizador* genera otra secuencia de entrada emparejando la pregunta con el resto del contexto truncado. Ejemplo:

pregunta y contexto:

```
question = "what is the origin of COVID-19?"
context = "The Severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2) pandemic offers a unique opportunity to study the introduction and evolution of a pathogen into a completely naive human population"
```

secuencia de *wordpieces* de entrada:

```
[CLS] what is the origin of co ##vid - 19 ? [SEP] the severe acute respiratory syndrome corona ##virus 2 ( sar ##s - co ##v - 2 ) pan ##de ##mic offers a unique opportunity to study the introduction and evolution of a pathogen into a completely naive human population [SEP]
```

ids de la secuencia de entrada (*input_ids)*:

```
[101, 2054, 2003, 1996, 4761, 1997, 2522, 17258, 1011, 2539, 1029, 102, 1996, 5729, 11325, 16464, 8715, 21887, 23350, 1016, 1006, 18906, 2015, 1011, 2522, 2615, 1011, 1016, 1007, 6090, 3207, 7712, 4107, 1037, 4310, 4495, 2000, 2817, 1996, 4955, 1998, 6622, 1997, 1037, 26835, 2046, 1037, 3294, 15743, 2529, 2313, 102]
```

El valor 101 corresponde al token [CLS] y el 102 al token [SEP]. La longitud de la secuencia de entrada del ejemplo anterior es de 52 *tokens*.

El paso intermedio que muestra los *wordpieces* queda oculto, aquí se ha mostrado para ilustrar el ejemplo.

El *tokenizador* recibe la pregunta y el contexto junto a los parámetros de *tokenizado* y devuelve la secuencia de entrada.

```
max_length = 384
doc_stride = 128

tokenized_example = tokenizer(
        question,
        context,
        truncation="only_second",
        max_length=max_length,
        stride=doc_stride
)
input_ids = tokenized_exampled[‘input_ids’]
```

El atributo *stride* define un valor de solapamiento entre contextos en caso de que sea necesario truncar la entrada.

Entrenamiento. El código para completar la tarea de preprocesamiento del *dataset* junto a las operaciones necesarias para definir los argumentos de *fine-tuning* y lanzar el entrenamiento se incluyen en un *script.* La ejecución se realiza en una máquina virtual n1-standard con 8 CPUs y 30GB de RAM que distribuye la carga de trabajo a una TPU v3-8.

Los modelos $\text{BERT}_{\text{BASE}}$ requieren alrededor de 30 minutos cuando se ajustan durante 4 épocas con tamaño de lotes de 48 secuencias. Los modelos $\text{BERT}_{\text{LARGE}}$ necesitan 90 minutos aproximadamente para el mismo número de épocas y tamaño de lotes de 16 secuencias. Durante el entrenamiento, se generan los modelos para 1, 2 y 3 épocas como *check-points* del proceso de *fine-tuning.*

El *fine-tuning* en SQuAD v2 + QuAC v1.1 consiste en continuar el entrenamiento del modelo ajustado en SQuAD v2 durante 4 épocas con el entrenamiento en QuAC durante 4 épocas más.

4.1.1.2 Evaluación de modelos con las métricas EM/F1

Para la evaluación se dispone de dos conjuntos, el conjunto de evaluación del *dataset* empleado durante *fine-tuning* y el conjunto de predicciones realizadas por el modelo ya entrenado. El modelo devuelve tres valores para cada entrada: la puntuación asignada a la respuesta y los *logits* de comienzo y fin. Es necesario, mapear los *logits* sobre el contexto original para extraer la respuesta. Con esta operación

es posible calcular la coincidencia exacta, la cobertura, la precisión y la puntuación F1 del modelo.

El *script* de evaluación no tiene una demanda computacional tan alta como el *script* de *fine-tuning* y se ejecuta en una GPU NVIDIA Tesla T4.

4.1.1.3 Evaluación de modelos con las métricas NDNS

La evaluación de modelos en el conjunto de documentos relevantes TREC-COVID se realiza en las siguientes condiciones:

1) la recuperación de información se supone en condiciones ideales. Los documentos candidatos procesados por los modelos son documentos relevantes. Todos ellos con la misma puntuación de relevancia;

2) aunque los documentos están segmentados en contextos o pasajes, la extracción de respuestas se realiza a nivel de sentencias. La secuencia de entrada a los modelos se forma concatenando la pregunta como sentencia A con cada una de las oraciones del documento candidato como sentencia B.

   No se utiliza el párrafo porque las respuestas extraídas por los modelos no se extienden más allá de una oración dentro del párrafo. Si el párrafo contiene varias oraciones que son parte de la respuesta correcta, se perderán todas menos la de mayor puntuación degradando el rendimiento del modelo;

3) para obtener los *nuggets* que forman la respuesta a una pregunta se consideran todas las respuestas del modelo diferentes del *token* [CLS], del orden de 4000 fragmentos de texto por pregunta. Los fragmentos se extienden hasta la sentencia original. Si el modelo devuelve el fragmento de texto: “COVID-19 patients presented predominantly with fever and cough” que forma parte de la sentencia con identificador k5y1qc82-C020-S000, la respuesta se extiende a la sentencia completa: “As expected from initial observations in China [4, 5, 11] , COVID-19 patients presented predominantly with fever and cough, which

appears to be more frequent in adults than children, as well as dyspnea, and myalgia, among other clinical features";

4) la lista de respuestas se ordena de mayor a menor puntuación y se seleccionan las 400 mejores. El valor 400 se ha establecido mediante un estudio empírico con el modelo $BioBERT_{LARGE}$ ajustado en SQuAD v2 + QuAC para maximizar los resultados:

#res	NDNS-Partial	NDNS-Relaxed	NDNS-Exact
100	0.258	0.257	0.273
200	0.277	0.273	0.273
300	0.300	0.291	0.301
350	0.303	0.293	0.303
400	**0.313**	**0.301**	**0.307**
450	0.314	0.301	0.306

Tabla 10: Resultados NDNS de $BioBERT_{LARGE}$ en SQuAD+QuAC en función del número de respuestas

Las respuestas que pertenecen al mismo contexto y son consecutivas se concatenan asignando al resultado la puntuación mayor de las respuestas originales.

Como se menciona en el capítulo 3 Marco de evaluación, la organización del desafío EPIC-QA proporciona un *script* de evaluación escrito en Python que toma como entrada un archivo de respuestas con el siguiente formato:

```
 Q_ID  Q0   START_SENTENCE_ID:END_SENTENCE_ID   RANK  SCORE   RUN_NAME
EQ001 Q0   dx3jywv7-C000-S000:dx3jywv7-C000-S000    0  0.907EXAMPLE_RUN
EQ001 Q0   aazntg4c-C008-S000:aazntg4c-C008-S000    1  0.740EXAMPLE_RUN
EQ001 Q0   eec649x4-C018-S000:eec649x4-C018-S000    2  0.579EXAMPLE_RUN
EQ001 Q0   5fe8rmhm-C002-S000:5fe8rmhm-C002-S000    3  0.469EXAMPLE_RUN
EQ001 Q0   pn9nwl64-C000-S000:pn9nwl64-C000-S000    4  0.396EXAMPLE_RUN
EQ001 Q0   7v5aln90-C011-S000:7v5aln90-C011-S000    5  0.301EXAMPLE_RUN
…
EQ002 Q0   mdyojac2-C004-S000:mdyojac2-C004-S000  400  0.122EXAMPLE_RUN
EQ002 Q0   hadnxjeo-C010-S000:hadnxjeo-C010-S000  401  0.093EXAMPLE_RUN
EQ002 Q0   lysvg3vw-C006-S000:lysvg3vw-C006-S000  402  0.005EXAMPLE_RUN
EQ002 Q0   qz2joxys-C027-S000:qz2joxys-C027-S000  403  0.002EXAMPLE_RUN
EQ002 Q0   5kyx6ycq-C015-S000:5kyx6ycq-C015-S000  404  0.002EXAMPLE_RUN
…
EQ045 Q0   y2vgo55k-C005-S000:y2vgo55k-C005-S000 6855  0.897EXAMPLE_RUN
EQ045 Q0   omv50b7j-C024-S000:omv50b7j-C024-S000 6856  0.578EXAMPLE_RUN
EQ045 Q0   26yvryhw-C005-S000:26yvryhw-C005-S000 6857  0.503EXAMPLE_RUN
EQ045 Q0   55mxgkmy-C010-S000:55mxgkmy-C010-S000 6858  0.391EXAMPLE_RUN
EQ045 Q0   y2vgo55k-C024-S000:y2vgo55k-C024-S000 6859  0.349EXAMPLE_RUN
…
```

4.1.2 Modelos SentenceBERT

Los modelos SBERT están entrenados para búsqueda semántica en el corpus MS MARCO. No se realiza *fine-tuning*. La evaluación con las métricas NDNS en la colección de documentos *preliminary expert* se hace en las condiciones definidas en 4.1.1.3.

4.2 Experimentos

Para responder las preguntas de investigación se realizan los siguientes experimentos:

4.2.1 ¿Qué modelo BERT: $BERT_{BASE}$, $BioBERT_{BASE}$, ClinicalBERT o SciBERT demuestra ser más efectivo en la tarea de extracción de respuesta?

■ Experimento:

1) Se ajustan los modelos $BERT_{BASE}$, $BioBERT_{BASE}$, ClinicalBERT y SciBERT en el dataset SQuAD v2 (Rajpurkar et al., 2018) durante 4 épocas con tamaño de lotes de 48 secuencias, longitud máxima de la secuencia de entrada de 512 *tokens* y ratio de aprendizaje de 5e-5.

2) Se evalúan los modelos en el subconjunto de validación de SQuAD v2 con las métricas EM y F1 para verificar que los resultados alcanzados están dentro de los valores considerados de estado del arte para modelos similares ajustados en la tarea SQuAD v2. La tabla siguiente muestra los resultados:

Modelo	EM	F1
$BERT_{BASE}$	67.236	70.492
$BioBERT_{BASE}$	74.320	77.776
ClinicalBERT	68.643	72.531
SciBERT	70.605	74.231

Tabla 11: Evaluación de modelos en el subconjunto validation de SQuAD v2.

3) Se evalúan los modelos con las métricas NDNS en el conjunto de preguntas *preliminary* de la tarea *expert* de EPIC-QA. Los resultados obtenidos se muestran a continuación:

Modelo	NDNS-Partial	NDNS-Relaxed	NDNS-Exact
$BERT_{BASE}$	**0.252**	**0.243**	**0.249**
$BioBERT_{BASE}$	0.222	0.215	0.224
ClinicalBERT	0.217	0.210	0.223
SciBERT	0.233	0.231	0.233

Tabla 12: Evaluación en el conjunto de preguntas *preliminary expert* de modelos entrenados en SQuAD v2.

4.2.2 ¿Qué fine-tuning da mejores resultados: SQuAD, QuAC o SQuAD + QuAC?

■ Experimento:

1) Se repite el experimento 4.1 realizando *fine-tuning* de los modelos $BERT_{BASE}$, $BioBERT_{BASE}$, ClinicalBERT y SciBERT en el subconjunto *train* de QuAC durante 4 épocas con los mismos parámetros de entrenamiento: tamaño de lotes de 48 secuencias, longitud máxima de la secuencia de entrada de 512 *tokens* y ratio de aprendizaje de 5e-5.

2) Se evalúan los modelos en el subconjunto de validación de QuAC con las métricas EM y F1 para constatar que los modelos rinden como un modelo basado en regresión logística que no tiene en consideración la información de contexto (Choi et al., 2018). La tabla muestra los resultados:

Modelo	EM	F1
$BERT_{BASE}$	18.330	39.085
$BioBERT_{BASE}$	16.209	36.090
ClinicalBERT	13.312	32.808
SciBERT	14.251	34.581

Tabla 13: Evaluación de modelos en el subconjunto *validation* de QuAC.

3) Se evalúan los modelos con las métricas NDNS en el conjunto de preguntas *preliminary* de la tarea *expert* de EPIC-QA con las mismas condiciones que en el experimento 4.1.

Modelo	NDNS-Partial	NDNS-Relaxed	NDNS-Exact
$BERT_{BASE}$	**0.296**	**0.287**	**0.295**
$BioBERT_{BASE}$	0.060	0.060	0.067
ClinicalBERT	0.183	0.182	0.191
SciBERT	0.229	0.223	0.233

Tabla 14: Evaluación en el conjunto de preguntas *preliminary expert* de modelos entrenados en QuAC.

4) Para SQuAD v2 + QuAC se continúa el ajuste de los modelos ya entrenados en el experimento (4.1) con el *dataset* QuAC y las condiciones descritas: 4 épocas, tamaño de lotes de 48 secuencias, longitud máxima de las secuencias de entrada de 512 *tokens* y ratio de aprendizaje de 5e-5.

5) La evaluación de los modelos en el subconjunto de validación de QuAC con las métricas EM y F1 obtiene los resultados:

Modelo	EM	F1
$BERT_{BASE}$	18.616	38.603
$BioBERT_{BASE}$	16.862	37.041
ClinicalBERT	15.910	37.210
SciBERT	16.086	37.296

Tabla 15: Evaluación de modelos en el subconjunto *validation* de QuAC.

6) Evaluación de los modelos con las métricas NDNS en el conjunto de preguntas *preliminary expert* de EPIC-QA como se describe en 4.1.3):

Modelo	NDNS-Partial	NDNS-Relaxed	NDNS-Exact
$BERT_{BASE}$	0.237	0.232	0.232
$BioBERT_{BASE}$	**0.277**	**0.267**	**0.273**
ClinicalBERT	0.268	0.260	0.263
SciBERT	0.211	0.205	0.211

Tabla 16: Evaluación en el conjunto de preguntas *preliminary expert* de modelos entrenados en SQuAD v2 + QuAC

7) (Devlin et al., 2019) observa que modelos con un número mayor de parámetros obtienen mejores resultados a cambio de un coste computacional mayor. Los siguientes modelos se han entrenado durante 4 épocas, con tamaños de lotes de 16 secuencias,

longitud máxima para la secuencia de entrada de 512 *tokens* y ratio de aprendizaje de 5e-05.

8) Evaluación en las métricas EM/F1:

Dataset	Modelo	EM	F1
SQuAD v2	$BERT_{LARGE}$	76.282	79.774
	$BioBERT_{LARGE}$	80.502	83.969
QuAC	$BERT_{LARGE}$	19.649	41.509
	$BioBERT_{LARGE}$	18.289	40.741
SQuAD v2 + QuAC	$BERT_{LARGE}$	19.160	28.264
	$BioBERT_{LARGE}$	19.717	29.031

Tabla 17: Evaluación de modelos large con las métricas EM y F1

9) Evaluación de los modelos con las métricas NDNS en el conjunto de preguntas *preliminary expert* de EPIC-QA como se describe en 4.1.3):

Dataset	Modelo	NDNS-Partial	NDNS-Relaxed	NDNS-Exact
SQuAD v2	$BERT_{LARGE}$	0.204	0.199	0.207
	$BioBERT_{LARGE}$	0.255	0.245	0.249
QuAC	$BERT_{LARGE}$	**0.325**	**0.315**	**0.324**
	$BioBERT_{LARGE}$	0.259	0.250	0.257
SQuAD v2 + QuAC	$BERT_{LARGE}$	0.280	0.266	0.274
	$BioBERT_{LARGE}$	**0.313**	**0.301**	**0.307**

Tabla 18: Evaluación en el conjunto de preguntas *preliminary expert* de modelos large.

■ Resultados:

La siguiente tabla agrupa los resultados NDNS de todos los modelos *base* analizados:

Dataset	Modelo	NDNS-Partial	NDNS-Relaxed	NDNS-Exact
SQuAD v2	$BERT_{BASE}$	0.252	0.243	0.249
	$BioBERT_{BASE}$	0.222	0.215	0.224
	ClinicalBERT	0.217	0.210	0.223
	SciBERT	0.233	0.231	0.233
QuAC	$BERT_{BASE}$	**0.296**	**0.287**	**0.295**
	$BioBERT_{BASE}$	0.060	0.060	0.067
	ClinicalBERT	0.183	0.182	0.191
	SciBERT	0.229	0.223	0.233
SQuAD v2 + QuAC	$BERT_{BASE}$	0.237	0.232	0.232
	$BioBERT_{BASE}$	**0.277**	**0.267**	**0.273**
	ClinicalBERT	0.268	0.260	0.263
	SciBERT	0.211	0.205	0.211

Tabla 19: Resumen de las evaluaciones en el conjunto de preguntas *preliminary expert* de modelos base.

■ **Conclusiones**:

1. Cuando se realiza *fine-tuning* en QuAC, los modelos BERT obtienen los mejores resultados: $NDNS_{Exact}$ = 0.295 y $NDNS_{Exact}$ = 0.324 para $BERT_{BASE}$ y $BERT_{LARGE}$ respectivamente.

2. Cuando el *fine-tuning* es en SQuAD v2 + QuAC, los modelos BioBERT alcanzan los mejores resultados: $NDNS_{Exact}$ = 0.273 para $BioBERT_{BASE}$ y $NDNS_{Exact}$ = 0.307 para $BioBERT_{LARGE}$.

4.2.3 Análisis de hiperparámetros: ¿cómo afectan los parámetros de *fine-tuning* en los resultados de los modelos?

■ Experimento:

El siguiente experimento aísla los parámetros de *fine-tuning* para analizar cómo afectan a los resultados de un modelo $BERT_{LARGE}$ entrenado en QuAC.

1) Número de épocas.

Se realiza *fine-tuning* de modelos $\text{BERT}_{\text{LARGE}}$ en el *dataset* QuAC durante 2, 3 y 4 épocas manteniendo constantes el resto de parámetros: tamaño de lotes de 16 secuencias, longitud máxima para la secuencia de entrada de 512 *tokens* y ratio de aprendizaje de 5e-05.

Se evalúan los modelos anteriores en el conjunto de preguntas *preliminary expert* con las métricas NDNS:

Épocas	NDNS-Partial	NDNS-Relaxed	NDNS-Exact
2	0.336	0.327	0.341
3	**0.346**	**0.335**	**0.344**
4	0.325	0.315	0.324

Tabla 20: Evaluación en el conjunto de preguntas *preliminary expert* de modelos $\text{BERT}_{\text{LARGE}}$ entrenados durante 2, 3 y 4 épocas en QuAC.

2) Tamaño del lote de entrenamiento.

El tamaño del lote está relacionado con la memoria consumida. Los modelos *large* no se pueden ajustar en una TPU v3-8 con tamaños de lotes mayores de 16 secuencias porque agotan la memoria disponible. El estudio se realiza en modelos $\text{BERT}_{\text{BASE}}$ ajustados en QuAC durante 3 épocas con secuencias de longitud 512, ratio de aprendizaje 5e-05 y tamaños de lotes de 16, 32, y 48 secuencias respectivamente.

Lotes	NDNS-Partial	NDNS-Relaxed	NDNS-Exact
16	**0.306**	**0.292**	0.297
32	0.240	0.232	0.243
48	0.301	0.291	**0.298**

Tabla 21: Evaluación en el conjunto de preguntas *preliminary expert* de modelos $\text{BERT}_{\text{LARGE}}$ entrenados en QuAC con lotes de tamaño 16, 32 y 48 secuencias.

3) Longitud de la secuencia de entrada.

Se realiza *fine-tuning* de modelos $\text{BERT}_{\text{LARGE}}$ en el *dataset* QuAC durante 2, 3 y 4 épocas cambiando la longitud de la secuencia de entrada y manteniendo constantes el resto de parámetros: tamaño de lotes de 16 secuencias y ratio de aprendizaje de 5e-05.

Longitud	Épocas	NDNS-Partial	NDNS-Relaxed	NDNS-Exact
384	2	0.308	0.301	0.303
	3	0.290	0.983	0.293
	4	0.309	0.298	0.308
512	2	0.336	0.327	0.341
	3	**0.346**	**0.335**	**0.344**
	4	0.325	0.315	0.324

Tabla 22: Evaluación en el conjunto de preguntas *preliminary expert* de modelos $\text{BERT}_{\text{LARGE}}$ entrenados en QuAC durante 2, 3 y 4 épocas y longitudes de secuencias de entrada 384 y 512

Cuando la longitud de la secuencia de entrada es superior a la longitud máxima, se trunca el contexto en n subcontextos, que generan n secuencias de entrada válidas concatenando la pregunta con cada subcontexto. Si el fragmento de respuesta comienza en un subcontexto y acaba en otro el modelo no lo identifica como respuesta. De ahí que modelos con longitudes de entrada mayores obtengan mejores rendimientos.

4) Ratio de aprendizaje.

Se ajustan modelos $\text{BERT}_{\text{LARGE}}$ en el *dataset* QuAC durante 3 épocas con ratios de aprendizaje 2e-05 y 5e-05 manteniendo constantes el resto de parámetros: tamaño de lotes de 16 secuencias y longitudes de secuencias de 512 *tokens*.

lr	NDNS-Partial	NDNS-Relaxed	NDNS-Exact
2e-05	0.312	0.303	0.310
5e-05	**0.346**	**0.335**	**0.344**

Tabla 23: Comparación de los resultados de dos modelos BERTLARGE entrenados en QuAC con ratios de aprendizaje 2e-05 y 5e-05.

5) Longitudes de las respuestas en función del *dataset* de ajuste.

La siguiente tabla lista las longitudes medias de las respuestas extraídas por modelos $\text{BioBERT}_{\text{LARGE}}$ entrenados durante 3 épocas con tamaños de lotes de 16 secuencias, longitudes de secuencias de 512 *tokens* y ratio de aprendizaje 5e-05.

Dataset	Longitud media (caracteres)
SQuAD v2	28
QuAC	39
SQuAD v2 + QuAC	42

Tabla 24: Longitudes medias de los fragmentos extraídos en función del *dataset* de *fine-tuning* utilizado

Como observa (Otegi et al., 2020), modelos ajustados con SQuAD v2 + QuAC producen respuestas más extensas. La longitud de la respuesta está condicionada por el *dataset* de entrenamiento.

4.2.4 Una vez determinado cuál es el mejor modelo BERT para realizar extracción de respuestas, ¿es mejor utilizar una aproximación basada en q*uestion-answering* o en *sentence similarity*?

■ Experimento:

Se compara el mejor modelo $BERT_{LARGE}$ ajustado en QuAC durante 3 épocas con tamaño de lotes de 16 secuencias, longitud máxima de la secuencia de entrada de 512 t*okens* y ratio de aprendizaje de 5e-05 con los siguientes modelos SentenceBERT entrenados en MS MARCO:

Modelo	MRR@10 dev	PSS[17]
msmarco-bert-base-dot-v5	38.08	52.11
msmarco-MiniLM-L6-cos-v5	32.27	42.16
msmarco-MiniLM-L12-cos-v5	32.75	43.89
msmarco-distilbert-cos-v5	33.79	44.98

Tabla 25: Rendimiento de modelos SentenceBERT para búsqueda semántica (datos tomados de https://www.sbert.net/docs/pretrained_models.html#semantic-search)

17 Performance Semantic Search

Modelo	NDNS-Partial	NDNS-Relaxed	NDNS-Exact
$BERT_{LARGE}$	0.346	0.335	**0.344**
msmarco-bert-base-dot-v5	0.325	0.312	0.304
msmarco-MiniLM-L6-cos-v5	0.341	0.327	0.333
msmarco-MiniLM-L12-cos-v5	0.322	0.307	0.318
msmarco-distilbert-cos-v5	**0.366**	**0.347**	0.341

Tabla 26: Comparación de los resultados del mejor modelo $BERT_{LARGE}$ entrenados en QuAC con modelos SentenceBERT entrenados en MS MARCO

■ Conclusiones:

En condiciones ideales de recuperación de información, un modelo $BERT_{LARGE}$ ajustado en QuAC durante 3 épocas con tamaño de lotes de 16 secuencias, longitud máxima de la secuencia de entrada de 512 *tokens* y ratio de aprendizaje de 5e-05 obtiene puntuaciones $NDNS_{Partial}$ = 0.346, $NDNS_{Relaxed}$ = 0.335 y $NDNS_{Exact}$ = 0.344 cuando se evalúa en el conjunto de preguntas *preliminary expert*. Un modelo SentenceBERT *msmarco-distilbert-cos-v5* entrenado en 500K pares (consulta, respuesta) del *dataset* MS MARCO alcanza puntuaciones $NDNS_{Partial}$ = 0.366, $NDNS_{Relaxed}$=0.347 y $NDNS_{Exact}$ = 0.341 *out-of-the-box*.

Aunque los sistemas alcanzan valores de estado del arte en relación a los sistemas presentados a la tarea A: *Expert QA*, no son comparables dado que obtenemos los resultados en condiciones ideales de recuperación.

	NDNS-Partial	NDNS-Relaxed	NDNS-Exact
1	0.305 ixa-3	0.307 ixa-3	**0.341 ixa-3**
2	0.303 ixa-2	0.304 ixa-2	0.338 ixa-2
3	0.302 HLTRI-1	0.295 IBM-3	0.327 IBM-1
4	0.294 IBM_3	0.294 IBM_1	0.327 IBM_2
5	0.294 IBM_1	0.293 IBM_2	0.325 IBM_3
6	0.293 IBM_2	0.293 HLTRI_1	0.306 ixa_1
7	0.276 ixa_1	0.277 ixa_1	0.300 vigicovid_2
8	0.266 vigicovid_2	0.267 vigicovid_2	0.297 vigicovid_3
9	0.263 vigicovid_3	0.265 vigicovid_3	0.289 HLTRI_1

Tabla 27: Resultados de la evaluación preliminary en la tarea A usando Normalized Discounted Novelty Score (NDNS)

Capítulo 5

Conclusiones y trabajo futuro

5.1 Conclusiones

Este trabajo ha evaluado el rendimiento de modelos *Transformers* BERT (Devlin et al., 2018) y SBERT (Reimers et Gurevych, 2019) entrenados en *datasets* de dominio general SQuAD v2 (Rajpurkar et al., 2018), QuAC (Choi et al., 2018) y MS MARCO (Nguyen et al., 2016) cuando se aplican a la tarea de obtención de respuestas a preguntas de dominio COVID-19 a partir del conjunto de documentos relevantes TREC-COVID del corpus de artículos biomédicos CORD-19 (Wang et al., 2020).

Cuatro han sido las preguntas de investigación:

1) ¿Qué modelo de lenguaje: $BERT_{BASE}$, $BioBERT_{BASE}$, ClinicalBERT o SciBERT demuestra ser más efectivo en la tarea de extracción de respuestas?

   Apartado 4.2.1) Cuando los resultados se midieron con las métricas NDNS, el modelo que obtuvo las puntuaciones más

altas fue $BERT_{BASE}$. $NDNS_{Partial}$ = 0.252, $NDNS_{Relaxed}$ = 0.243 y $NDNS_{Exact}$ = 0.249 (Tabla 12).

2) ¿Qué *fine-tuning* da mejores resultados: SQuAD v2, QuAC o SQuAD v2 + QuAC?

Apartado 4.2.2) El *dataset* que obtuvo mejores resultados fue QuAC cuando se usó para *fine-tuning* de un modelo $BERT_{BASE}$. Si el *fine-tuning* se realiza en SQuAD v2 + QuAC, la puntuación más alta la obtuvo el modelo $BioBERT_{BASE}$.

Dataset	Modelo	NDNS-Partial	NDNS-Relaxed	NDNS-Exact
QuAC	$BERT_{BASE}$	0.296	0.287	0.295
SQuAD v2 + QuAC	$BioBERT_{BASE}$	0.277	0.267	0.273

Tabla 28: Resultados NDNS de los modelos base más efectivos en función del dataset de fine-tuning.

Este comportamiento se mantiene en las arquitecturas *large*:

Dataset	Modelo	NDNS-Partial	NDNS-Relaxed	NDNS-Exact
QuAC	$BERT_{LARGE}$	0.325	0.315	0.324
SQuAD v2 + QuAC	$BioBERT_{LARGE}$	0.313	0.301	0.307

Tabla 29: Resultados NDNS de los modelos large más efectivos en función del dataset de fine-tuning.

3) ¿Cómo afectan los parámetros de *fine-tuning* a los resultados de los modelos?

Se aisló cada uno de los parámetros de *fine-tuning* para analizar el efecto sobre los resultados:

Apartado 4.2.3.1) Número de épocas. El modelo que obtiene la mejor puntuación fue el entrenado durante 3 épocas. $NDNS_{Partial}$ = 0.346, $NDNS_{Relaxed}$ = 0.335 y $NDND_{Exact}$ = 0.344 (Tabla 20).

Apartado 4.2.3.2) Tamaño del lote de entrenamiento. El modelo que obtiene la mejor puntuación en la métrica $NDNS_{Exact}$ fue el entrenado con lotes de 48 secuencias. Sin

embargo, el modelo con lotes de 16 secuencias alcanzó las mayores puntuaciones en NDNS parcial y relajada. (Tabla 21).

Apartado 4.2.3.3) Longitud de la secuencia de entrada. El mejor modelo fue $BERT_{LARGE}$ entrenado durante 3 épocas con longitud máxima de secuencia de entrada de 512 *tokens:* $NDNS_{Partial}$ = 0.346, $NDNS_{Relaxed}$ = 0.335 y $NDNS_{Exact}$ = 0.344 (Tabla 22).

Apartado 4.2.3.4) Ratio de aprendizaje. El mejor modelo empleó el ratio de aprendizaje de 5e-05. $NDNS_{Partial}$ = 0.346, $NDNS_{Relaxed}$ = 0.335 y $NDNS_{Exact}$ = 0.344 (Tabla 23).

Apartado 4.2.3.5) Longitudes de las respuestas en función del *dataset* de *fine-tuning*. Las respuestas más largas las devolvió el modelo ajustado en SQuAD v2 + QuAC con una longitud media de 42 caracteres por respuesta.

En condiciones ideales de recuperación de información, el modelo BERT más efectivo para realizar extracción de respuestas fue $BERT_{LARGE}$ ajustado en QuAC durante 3 épocas con tamaño de lotes de 16 secuencias, longitud máxima de la secuencia de entrada de 512 *tokens* y ratio de aprendizaje de 5e-05.

4) ¿Es mejor utilizar una aproximación basada en *question-answering* o *sentence similarity*?

Apartado 4.2.4) Una aproximación *sentence similarity* basada en un modelo SBERT entrenado en MS MARCO utilizando codificadores duales DistilBERT y empleando la distancia coseno como medida de distancia alcanza resultados $NDNS_{Partial}$ = 0.366, $NDNS_{Relaxed}$= 0.347 y $NDNS_{Exact}$ = 0.341 con la latencia de un modelo DistilBERT y sin la necesidad de un proceso de *fine-tuning* previo a la utilización del modelo.

El modelo $BERT_{LARGE}$ ajustado en QuAC durante 3 épocas con tamaño de lotes de 16 secuencias, longitud máxima de la secuencia de entrada de 512 *tokens* y ratio de aprendizaje de 5e-05 solo mejora la métrica $NDNS_{Exact} = 0.344$ del modelo SBERT.

5.2 Trabajo futuro

1. La recuperación de información con índices densos e híbridos es una línea de investigación con entidad propia. La salida del módulo de recuperación de información condiciona todo el *pipeline*.

2. Durante la elaboración del trabajo presentado se valoró en varias ocasiones la utilización de técnicas de generación de texto basadas en *Transformers* secuencia a secuencia. Esta idea se abandonó debido al coste computacional de la misma:

   a) En el módulo de recuperación de información, la generación de secuencias se utiliza para extender los documentos del corpus generando preguntas que se indexan junto a los documentos para posteriormente utilizar RQE o búsqueda semántica por similitud con el fin de mejorar los resultados de recuperación. El corpus EPIC-QA tiene 4M de documentos cuando se utiliza el contexto como unidad de indexación. Extender cada uno de ellos con varias preguntas generadas con BART (Lewis et al., 2019) o T5 (Raffel et al., 2019) tiene un coste computacional a tener en consideración.

   b) Uno de los requisitos más importantes para el desarrollo de un modelo QA es un *dataset* anotado con preguntas, pasajes y respuestas. La falta de tales *datasets* en dominios específicos de conocimiento como COVID-19 motiva el desarrollo de *datasets* sintéticos mediante técnicas de generación de texto. Es necesario estudiar la calidad del *dataset* de entrenamiento generado (+100K muestras) para QA extractivo. El coste humano de elaborar un corpus anotado de calidad se traslada al coste computacional para generar el *dataset* sintético.

3. El reranking de respuestas es una tarea más compleja que simplemente ordenar una lista de respuestas a partir de las puntuaciones. Una de las líneas de investigación en *rerankers* formula la tarea como un problema de clasificación (Nogueira y Cho, 2020) y enseña a modelos BERT a reordenar las respuestas en base a la novedad de los *nuggets*.

4. Establecer una metodología de análisis, diseño, implementación y despliegue de sistemas QA específicos del dominio a partir de corpus de artículos científicos sin anotar que obtenga la aprobación de los usuarios finales y logre un nivel de madurez suficiente que permita llegar al consumidor en forma de servicio. En este sentido, si el coste de los recursos no fuera un factor limitante, preentrenar modelos desde cero en dominios específicos con corpus sin anotar como CORD-19 y ajustar mediante *datasets* QA generados sintéticamente podría ser una línea de investigación prometedora como base a una metodología de desarrollo.

Referencias

[Alsentzer et al., 2019]

E. Alsentzer, J. R. Murphy, W. Boag, W.-H. Weng, D. Jin, T. Naumann, and M. B. A. McDermott. Publicly Available Clinical BERT Embeddings. In: arXiv:1904.03323 (2019).

[Arman et al., 2019]

Arman Cohan, Waleed Ammar, Madeleine van Zuylen, and Field Cady. 2019. Structural scaffolds for citation intent classification in scientific public In NAACL-HLT, pages 3586–3596, Minneapolis,
Minnesota. Association for Computational Linguistics.

[Bahdanau et al., 2014]

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly
learning to align and translate. CoRR, abs/1409.0473, 2014.

[Baevski et al., 2019]

Alexei Baevski, Sergey Edunov, Yinhan Liu, Luke Zettlemoyer, and Michael Auli. 2019. Cloze-driven pretraining of self-attention networks. arXiv preprint arXiv:1903.07785.

[Beltagy et al., 2019]

Beltagy, I., Lo, K., & Cohan, A. (2019). Scibert: A pretrained language model for scientific text. *arXiv preprint arXiv:1903.10676*.

[Ben Abacha and Demner-Fushman, 2019]

Asma Ben Abacha and Dina Demner-Fushman. 2019. A question-entailment approach to question answering. BMC Bioinformatics, 20:511, 10.

[Chakravarti et al., 2020]

Rishav Chakravarti, Anthony Ferritto, Bhavani Iyer, Lin Pan, Radu Florian, Salim Roukos, and Avi Sil. 2020. Towards building a robust industry-scale question answering system. In Proceedings of the 28th International Conference on Computational Linguistics: Industry Track, pages 90–101, Online. International Committee on Computational Linguistics.

[Choi et al., 2018]

Choi, E., He, H., Iyyer, M., Yatskar, M., Yih, W. T., Choi, Y., ... & Zettlemoyer, L. (2018). Quac: Question answering in context. *arXiv preprint arXiv:1808.07036*.

[Dai and Callan, 2019]

Zhuyun Dai and Jamie Callan. 2019. Deeper text understanding for IR with contextual neural language modeling. In Proceedings of the 42nd Annual International ACM SIGIR Conference on

Research and Development in Information Retrieval (SIGIR 2019), pages 985–988, Paris, France.

[Devlint et al., 2018]

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[He et al., 2018]

Luheng He, Kenton Lee, Omer Levy, and Luke Zettlemoyer. 2018. Jointly predicting predicates and arguments in neural semantic role labeling. In Association for Computational Linguistics (ACL), pages 364–369.

[Hendrycks y Gimpel, 2016]

Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. arXiv preprint arXiv:1606.08415, 2016.

[Hinton et al, 2015]

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. ArXiv, abs/1503.02531, 2015

[Hochreiter et al., 2001]

Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

[Joshi et al., 2019]

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2019.
SpanBERT: Improving pre-training by representing and predicting spans. arXiv preprint arXiv:1907.10529.

[Johnson et al., 2019]

Jeff Johnson, Matthijs Douze, and Herve J'egou. 2019. Billion-scale similarity search with gpus. IEEE Transactions on Big Data.

[Kalchbrenner et al., 2016]

Kalchbrenner, N., Espeholt, L., Simonyan, K., Oord, A. V. D., Graves, A., & Kavukcuoglu, K. (2016). Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*.

[Karpukhin et al., 2020]

Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., ... & Yih, W. T. (2020). Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*.

[Khattab and Zaharia, 2020]

Khattab, O., & Zaharia, M. (2020, July). Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval* (pp. 39-48).

[Kingman and Ba, 2015]

D. P. Kingma and J. L. Ba, Adam: A Method for stochastic Optimization. San Diego: The International Conference on Learning Representations (ICLR), 2015.

[Kitaev et al., 2020]

Kitaev, N., Kaiser, Ł., & Levskaya, A. (2020). Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*.

[Kwiatkowski et al., 2019]

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: A benchmark for question answering research. Transactions of the Association for Computational Linguistics, 7:452–466.

[Lample and Conneau, 2019]

Guillaume Lample and Alexis Conneau. 2019. Crosslingual language model pretraining. ArXiv preprint arXiv:1901.07291.

[Lan et al., 2019]

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2019). Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

[Lee et al., 2016]

Kenton Lee, Shimi Salant, Tom Kwiatkowski, Ankur Parikh, Dipanjan Das, and Jonathan Berant. 2016. Learning recurrent

span representations for extractive question answering. arXiv preprint arXiv:1611.01436.

[Lee et al., 2017]

Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. 2017. End-to-end neural coreference resolution. In Empirical Methods in Natural Language Processing (EMNLP), pages 188–197.

[Lee et al., 2020]

Lee, J.; Yi, S. S.; Jeong, M.; Sung, M.; Yoon, W.; Choi, Y.; Ko, M.; and Kang, J. 2020. Answering Questions on COVID-19 in Real-Time. arXiv preprint arXiv:2006.15830

[Lee, Yoon et al., 2020]

Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., & Kang, J. (2020). BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, *36*(4), 1234-1240.

[Lewis et al., 2019]

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., ... & Zettlemoyer, L. (2019). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

[Liu et al., 2019]

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... & Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

[MacAvaney et al., 2019]

MACAVANEY, Sean, et al. CEDR: Contextualized embeddings for document ranking. En *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2019. p. 1101-1104.

[Möller et al., 2020]

Möller, T.; Reina, G. A.; Jayakumar, R.; and Livermore, L. 2020. COVID-QA: A Question Answering Dataset for COVID-19.

[Mosbach et al., 2020]

Mosbach, M., Andriushchenko, M., & Klakow, D. (2020). On the stability of fine-tuning bert: Misconceptions, explanations, and strong baselines. *arXiv preprint arXiv:2006.04884*.

[Newmann et al., 2019]

Mark Neumann, Daniel King, Iz Beltagy, and Waleed Ammar. 2019. ScispaCy: Fast and robust models for biomedical natural language processing. In arXiv:1902.07669.

[Nogueira, 2020]

Rodrigo Nogueira. 2019. From doc2query to doctttttt-query.

[Nogueira, et al., 2019]

Nogueira, Rodrigo, et al. Multi-stage document ranking with bert. *arXiv preprint arXiv:1910.14424*, 2019.

[Nogueira et al., 2020]

Nogueira, Rodrigo; JIANG, Zhiying; LIN, Jimmy. Document ranking with a pretrained
sequence-to-sequence model. *arXiv preprint arXiv:2003.06713*, 2020.

[Nogueira et Cho, 2020]

Rodrigo Nogueira and Kyunghyun Cho. 2020. Passage re-ranking with bert.

[Nguyen et al., 2016]

Nguyen, T., Rosenberg, M., Song, X., Gao, J., Tiwary, S., Majumder, R., & Deng, L. (2016, January). MS MARCO: A human generated machine reading comprehension dataset. In *CoCo@ NIPS*.

[Otegi et al., 2020]

Otegi, A., Campos, J. A., Azkune, G., Soroa, A., & Agirre, E. (2020). Automatic Evaluation vs. User Preference in Neural Textual QuestionAnswering over COVID-19 Scientific Literature.

[Ott et al., 2018]

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling neural machine translation. In Proceedings of the Third Conference on Machine Translation (WMT).

[Pennington et al., 2014]

Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global vectors for word representation. In *Proceedings*

of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).

[Peters et al.2018]

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 2227–2237, New Orleans, LA, USA.

[Radford et al., 2018]

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

[Radford et al., 2019]

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. Technical report, OpenAI.

[Raffel et al., 2019]

Raffel, Colin, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[Rajpurkar et al., 2016]

Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

[Rajpurkar et al., 2018]

Rajpurkar, P., Jia, R., & Liang, P. (2018). Know what you don't know: Unanswerable questions for SQuAD. *arXiv preprint arXiv:1806.03822*.

[Reddy et al., 2020]

Revanth Gangi Reddy, Bhavani Iyer, Md Arafat Sultan, R. Zhang, Avi Sil, V. Castelli, Radu Florian,
and S. Roukos. 2020. End-to-end qa on covid-19: Domain adaptation with synthetic training. ArXiv,
abs/2012.01414.

[Reimers et Gurevych, 2019]

Reimers, N., & Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

[Robertson et al., 1994]

Robertson, S. (2004). Understanding inverse document frequency: on theoretical arguments for IDF. *Journal of documentation*.

[Romanov and Shivade, 2018]

Alexey Romanov and Chaitanya Shivade. 2018. Lessons from Natural Language Inference in the Clinical Domain. arXiv:1808.06752 [cs]. ArXiv: 1808.06752.

[Sanh, et al., 2019]

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

[Schroff et al., 2015]

Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. FaceNet: A Unified Embedding for Face Recognition and Clustering. arXiv preprint arXiv:1503.03832, abs/1503.03832.

[Sennrich et al., 2016]

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In Association for Computational Linguistics (ACL), pages 1715–1725.

[Srivastava et al., 2014]

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929-1958.

[Szegedy et al., 2016]

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2818-2826).

[Tang et al., 2020]

Tang, R.; Nogueira, R.; Zhang, E.; Gupta, N.; Cam, P.; Cho, K.; and Lin, J. 2020. Rapidly Bootstrapping a Question Answering Dataset for COVID-19. arXiv preprint arXiv:2004.11339.

[Taylor, 1953]

Taylor, W. L. (1953). “Cloze procedure”: A new tool for measuring readability. *Journalism quarterly*, *30*(4), 415-433.

[Vaswani et al., 2018]

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008).

[Vincent, et al., 2008]

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoen-coders. In Proceedings of the 25th international conference on Machine learning, pages 1096–1103. ACM.

[Wang et al., 2018]

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. R. (2018). GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

[Wang et al, 2020]

Wang, L. L., Lo, K., Chandrasekhar, Y., Reas, R., Yang, J., Eide, D., ... & Kohlmeier, S. (2020). Cord-19: The covid-19 open research dataset. *ArXiv*.

[Wiese et al, 2017]

Wiese,G. et al. (2017) Neural domain adaptation for biomedical question answering. In: Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017), Vancouver, Canada. pp. 281–289. Association for Computational Linguistics. https://www.aclweb.org/anthology/K17-1029.

[Wolf et al., 2020]

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 38–45, Online. Association for Computational Linguistics.

[Wu et al., 2016]

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Gregory S. Corrado, Macduff Hughes, and Jeffrey
Dean. 2016. Google’s neural machine translation system: Bridging the gap between human and machine translation. Abs/1609.08144

[Yang et al., 2017]

Yang, P., Fang, H., & Lin, J. (2017, August). Anserini: Enabling the use of lucene for information retrieval research. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 1253-1256).

[Yang et al., 2018]

Yang, P., Fang, H., & Lin, J. (2018). Anserini: Reproducible ranking baselines using Lucene. *Journal of Data and Information Quality (JDIQ)*, *10*(4), 1-20.

[Yang et al., 2019]

Yang You, Jing Li, Jonathan Hseu, Xiaodan Song, James Demmel, and Cho-Jui Hsieh. 2019. Reducing bert pre-training time from 3 days to 76 minutes. arXiv preprint arXiv:1904.00962.

[Yilmaz et al., 2019]

YILMAZ, Zeynep Akkalyoncu, et al. Cross-domain modeling of sentence-level evidence for document retrieval. En *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 2019. p. 3481-3487.

[You et al., 2019]

Yang You, Jing Li, Jonathan Hseu, Xiaodan Song, James Demmel, and Cho-Jui Hsieh. 2019. Reducing bert pre-training time from 3 days to 76 minutes. arXiv preprint arXiv:1904.00962.

[Zaragoza et al., 2004]

Robertson, S., Zaragoza, H., & Taylor, M. (2004, November). Simple BM25 extension to multiple weighted fields. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management* (pp. 42-49).

[Zhang et al., 2020]

Zhang, E., Gupta, N., Tang, R., Han, X., Pradeep, R., Lu, K., ... & Lin, J. (2020). Covidex: Neural ranking models and keyword search infrastructure for the covid-19 open research dataset. *arXiv preprint arXiv:2007.07846*.

[Zhang et al., 2020a]

Edwin Zhang, Nikhil Gupta, Rodrigo Nogueira, Kyunghyun Cho, and Jimmy Lin. 2020a. Rapidly
deploying a neural search engine for the COVID-19 Open Research Dataset: Preliminary thoughts and lessons learned. ArXiv:2004.05125.

[Zhang et al., 2020b]

Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q. Weinberger, and Yoav Artzi. 2020b. Revisiting few-sample BERT fine-tuning. ArXiv:2006.05987.

[Zhu et al., 2015]

Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In Proc. of ICCV.