UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Máster en tecnologías del lenguaje

Extracción de respuestas con detección de preguntas sin respuesta

Gonzalo Pérez Fernández

Director: Anselmo Peñas Padilla

<junio, 2020>

Resumen

Actualmente los sistemas de respuestas a preguntas (QA), están en proceso de desarrollo y perfeccionamiento. Lo podemos ver aplicado en asistentes en la actualidad como "Google Home" o "Alexa", pero está claro que todavía necesitan mejorar las respuestas que nos proporcionan en la mayor parte de los casos.

Por ello, investigadores de *Google AI Language* han desarrollado un modelo para procesamiento de lenguaje natural (NLP) denominado "BERT", que aplica un entrenamiento bidireccional de transformadores, por lo que tendrá una comprensión más profunda del contexto del lenguaje, más que los sistemas que solo utilizan entrenamiento en una sola dirección. Además, este modelo aplica una técnica de enmascaramiento denominada Masked LM (MLM) que permite el entrenamiento bidireccional en modelos en los que antes no era posible.

Debido a este preentrenamiento del modelo sobre grandes cantidades de corpus de datos, ha sido posible utilizarlo en soluciones como "Análisis de sentimiento" y, sobre todo, para sistemas de "Respuestas a preguntas o Question Answering". Para hacer un proceso de investigación del modelo BERT en el ámbito de Question Answering, vamos a utilizar el dataset de Stanford denominado SQuAD. Dicho dataset contiene un conjunto de contextos o documentos de Wikipedia con una serie de preguntas y sus correspondientes respuestas plausibles en dicho documento. En un principio, se creo una primera versión de SQuAD (v1.1), en la que dado un contexto y una pregunta, siempre existía una respuesta plausible para cada una de dichas preguntas. Este dataset está formado por más de 100.000 preguntas con sus respectivas repuestas en más de 500 artículos, por lo que cuando se realizó un entrenamiento del modelo BERT con esta versión del dataset, los resultados fueron muy alentadores, obteniendo un 93.2 % en la métrica de evaluación F1.

Sin embargo, se empezó a contemplar la idea de que en un escenario real, muchas de las preguntas que se realizan pueden no tener respuesta en los documentos. De aquí, nació la idea de crear una nueva versión del dataset SQuAD, la versión 2.0, que combina las 100.000 preguntas de la versión 1.1 con aproximadamente 50.000 preguntas sin respuestas, formuladas de manera similar a las preguntas con repuestas. Aquí se pudo ver una bajada considerable del rendimiento del sistema, resultando un 83.1 % en la métrica F1.

Por lo tanto, llegado a este punto nos surgen un conjunto de preguntas que vamos a intentar responder en este trabajo. El modelo preentrenado BERT funciona de diferente manera dependiendo del entrenamiento que le aportemos. Por lo tanto, debemos de investigar como se comporta dicho modelo y de que forma podríamos mejorarlo, entrenándolo con distintos tipos de preguntas y distintas cantidades de estas preguntas. Además, a la hora de predecir este modelo, podemos también sacar conclusiones sobre como actúa y si está actuando realmente como debería.

Abstract

Question Answering systems (QA) are currently under development and improvement processing. We can see nowadays in assistents such as “Google Home” and “Alexa”, but it’s clear that it needs to improve the answers provided in most cases.

Because of that, the researchers of “Google AI Language” have developed a Natural Language Proccesing (NLP) model called “BERT” which applies a bidirectional training of the transformers, therefore it will have a deeper understanding of the language context, more than systems that use a unidirectional training. In addition, this model applies a masking technique named Masked LM (MLM) that allows bidirectional training in models where it was not possible before.

Due to this pre-trained model against a large amount of corpus data, it makes possible to be used in solutions such as “sentiment analysis” and, especially in “Question Answering” systems. Therefore, to investigate the BERT model in the field of Question Answering, we will use the Stanford dataset named SQuAD. This dataset contains an amount of contexts or Wikipedia consisting of questions and the appropriate plausible answer to each of them. A first version of SQuAD (v1.1) was initially created, in which given a context and a question, this question always has a plausible answer. This dataset contains more than 100.000 questions and the corresponding answer in more than 500 articles, so when BERT was trained with this dataset version, the results were very promising, obtaining about 93.2% in the F1 evaluation metric.

However, a new idea began to be contemplated, since a large amount of the questions have no answers in the documents. Because of that, the idea of creating a new SQuAD dataset version, version 2.0, combining the above 100.000 questions in v1.1 with 50.000 new unanswerable questions, which are written in the same way as the answerable ones. Here we could see a big drop to 83.1% in the F1 metric.

Therefore, at this point a number of questions arise which we are going to try to answer in this paper. The BERT pretrained model works in different ways depending on the training it receives. Therefore, we must investigate the model’s behavior and how we can improve it by training it with different kind of questions and different sets of these types. In addition, at the prediction time we can get conclusions about how the model acts and if it is working as intended.

AGRADECIMIENTOS

En primer lugar quiero agradecerles a mis padres Jose Luis y Rosa y a mi hermana Gema, ya que sin ellos no habría conseguido nada de esto. Gracias por el apoyo incondicional que me han dado y por las largas charlas y regañinas para llegar a ser quien soy ahora mismo. Sin ellos nunca podría haber llegado a conseguir todo lo que tengo, ni a superarme día a día, marcándome nuevos objetivos. Porque gracias a ellos se que soy capaz de lograr todo lo que me proponga.

También se lo quería agradecer a mi pareja Paula, ya que me ha apoyado en mis estudios desde el principio, ayudándome con todas mis necesidades, haciéndome mis estudios más fáciles. Gracias a su infinita compañía, que ha convertido cada momento difícil en una carga más llevadera.

Agradecer también la ayuda recibida por mi tutor Anselmo Peñas, por aguantar mis múltiples correos. Gracias a su ayuda en otras asignaturas, supe desde el principio que sería un excelente tutor para este proyecto, por ello, le pedí que fuera mi guía y mentor en el Trabajo de Fin de Máster, papel que aceptó sin ninguna duda.

Por último, dar las gracias a la UNED y a todos los grandes profesionales que forman parte de este Máster, ya que sacar unos estudios de forma no presencial es más fácil gracias a su gran labor, esfuerzo y dedicación.

Gonzalo Pérez Fernández

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE LISTADOS

CAPÍTULO 1

INTRODUCCIÓN

1.1 MOTIVACIÓN

Con el paso de los años, la informática está siendo incorporando cada vez más en nuestra vida cotidiana, intentando facilitarnos la vida todo lo posible. Una de las investigaciones que más impacto está teniendo entre grupos de investigadores, es el campo de Question Answering (QA).

Sin embargo, este campo de investigación todavía no está del todo perfeccionado, ya que hay que tener en cuenta muchos factores a la hora de mejorar dichos sistemas. Los investigadores de Google AI Language han inventado un nuevo modelo denominado BERT[1], el cual ha sido preentrenado utilizando texto no etiquetado con un entrenamiento en ambos sentidos, es decir, de forma bidireccional (Peters y *et al.*, 2018), de está forma el modelo puede entender el contexto en ambos lados del escrito.

Este sistema puede ser reajustado con una única capa de salida adicional para, de esta forma, poder crear modelos de última generación como Question Answering e Inferencia del lenguaje, sin necesidad de cambiar la arquitectura de la tarea.

Para realizar un proceso de investigación de este modelo en el campo de Question Answering, hemos optado por utilizar el dataset SQuAD[2] de Stanford. Este dataset ha sido diseñado por expertos en lenguaje, compuestos por un documento de Wikipedia (contexto) y un conjunto de preguntas con las correspondientes respuestas en este documento. De está forma, se creó la primera versión del dataset (v1.1), pero actualmente, existe una segunda versión (v2.0) que además contiene preguntas a las que no se pueden responder dentro de ese mismo contexto o documento (Rajpurkar y *et al.*, 2018).

De aquí, nace la motivación de este trabajo, ya que en esta segunda versión del modelo SQuAD el rendimiento bajó considerablemente con respecto al primero. Por ello, debemos de investigar mediante una pila de pruebas, que puede hacer que el modelo trabaje mejor a la hora de entrenarlo con este dataset o quizás preguntarnos que es lo que está haciendo que el modelo baje su rendimiento tan drásticamente.

1.2 OBJETIVOS

Este punto consistirá en explicar cuales van a ser nuestras metas del trabajo. Por ello, explicaremos dichos objetivos de forma general, abstrayéndonos de datos más específicos que detallaremos más adelante.

El objetivo principal de este trabajo es el de investigar el funcionamiento del modelo BERT utilizando como dataset de entrenamiento SQuAD. Para realizar nuestro proceso de investigación

[1] Bidirectional Encoder Representations from Transformers
[2] Stanford Question Answering Dataset

nos vamos a realizar una serie de preguntas sobre dicho modelo. Intentaremos responderlas de la mejor forma posible, aportando pruebas de nuestras predicciones pero, para ello, debemos de:

- Entender como funciona el sistema de preentrenamiento del modelo BERT.
- Estudiar dicho modelo ajustado para sistemas de Question Answering, entradas, salidas, ...
- Comprender el dataset SQuAD y como debemos de formatearlo de una manera entendible como entrada al modelo, siendo capaz de darnos la salida deseada.
- Encontrar el script que más se adapte a nuestras necesidades y aprender como funciona.
- Realizar distintas pruebas de entrenamiento y predicción para tener el conocimiento necesario para proceder.
- A partir de las anteriores pruebas, del conocimiento del modelo y del dataset realizar distintos puntos de investigación que nos ayude a responder unas preguntas que nos indique como el modelo está actuando.

1.3 PREGUNTAS DE INVESTIGACIÓN

En esta sección veremos las preguntas que se han planteado en este trabajo y que vamos a intentar responder. En apartados posteriores explicaremos como se van a tratar cada una de estas preguntas.

1. ¿Qué efecto tiene ver preguntas sin respuesta en el entrenamiento?
2. ¿Cómo afecta ver durante el entrenamiento distintas proporciones de preguntas sin respuesta?
3. ¿Cómo actuaría el modelo entrenado con el dataset SQuAD 2.0 si al evaluarlo solo recibiera preguntas sin respuesta?

1.4 METODOLOGÍA

Una vez que hemos visto de forma general cuales serán nuestros objetivos, procederemos a explicarlos de una forma más detallada y precisa, dando de esta forma un nivel de detalle más amplio.

La propuesta para este trabajo es la de realizar un proceso de investigación sobre el modelo BERT desarrollado por el equipo de inteligencia artificial de Google. Entrenando el modelo, de tal forma, que podamos utilizarlo y adaptarlo de manera que sirva como solución a problemas relacionados con el campo del procesamiento del lenguaje natural.

En este caso, en el trabajo realizado aquí, nos vamos a centrar únicamente en un campo del procesamiento del lenguaje natural, el cual tiene que ver con las respuestas a preguntas teniendo claro un contexto o documento.

Por ello, dicho modelo nos va a ser de mucha utilidad ya que está preentrenado de forma bidireccional (Peter Dun, 2018) para entender el contexto de las oraciones en ambos sentidos, por lo tanto, hemos optado por llevar la investigación hacia este modelo.

Como ya tenemos escogido el modelo, la siguiente propuesta será conocer que datasets se escogen para entrenarlo y adaptarlo de tal forma que nos sea de utilidad para el objetivo principal de respuestas a preguntas. En este caso, existen una gran cantidad de datasets que nos podrían ser de utilidad para dicho objetivo, pero dado el gran número de puntos de investigación existentes del modelo BERT utilizando el dataset de Stanford denominado SQuAD, hemos inclinado la balanza hacia este mismo.

Ya que existen muchos trabajos de investigación relacionados con el modelo BERT y el dataset SQuAD, podremos recopilar una gran cantidad de información y conocimiento para poder tratar de obtener nuestros objetivos de la mejor forma posible además, entendiendo el problema desde un principio.

Una vez que conocemos cuales son las herramientas que vamos a utilizar en este trabajo, nuestra propuesta es la de investigar a fondo el proceso de entrenamiento del modelo BERT con dicho dataset, comprobando en que momentos el entrenamiento conlleva a un mayor éxito y en cuales nos encontramos con que la predicción no es del todo correcta o dicho de otra forma, un bajo porcentaje de éxito.

Los objetivos los vamos a explicar de forma estructurada y en el orden en el que se irán realizando para conseguir los resultados finales. Por ello, empezaremos desde lo más básico, como sería el proceso de recopilación de información, hasta alcanzar el proceso de obtención de resultados.

1.4.1 Question Answering

Uno de los conceptos fundamentales que debemos de conocer es el Question Answering o respuestas a preguntas, dado que el trabajo se va a relacionar con este subcampo del procesamiento del lenguaje natural.

Por lo tanto, para realizar este trabajo, debemos de poner como objetivo el aprender sobre este concepto, entender el uso que se le da a los sistemas de respuestas a preguntas, saber que utilidades aporta, comprender cual es la arquitectura general de un sistema de estas características y una vez adquirida de forma general esta información, también debemos de conocer la forma de evaluarlos.

La finalidad de entender dicho concepto y sus usos, es poder enfocar el estudio en la dirección correcta, teniendo los conocimientos necesarios para poder seleccionar los métodos que nos ayuden a lograr el objetivo final.

1.4.2 Modelo BERT y preentrenamiento

Una vez que tenemos una visión general sobre como funciona un sistema de respuestas a preguntas, debemos de analizar si el modelo que vamos a utilizar en este proceso de investigación nos va a ser útil para realizar este tipo de trabajos y, si es así, debemos de averiguar como hacerlo.

Una vez que sabemos si es posible continuar con este modelo, debemos de conocer cual es su estructura, o más bien, como funciona el modelo, para ser capaz de ejecutarlo como un sistema de respuestas a preguntas. Por ello, como entendemos de antemano que es un modelo preentrenado, debemos de saber como realizara este entrenamiento para enfocar de una manera u otra nuestro punto de investigación.

Con respecto a su estructura, investigaremos por que capas está formado y que finalidad tiene cada una de ellas. Además, tenemos que identificar cuales van a ser las entradas y salidas de este modelo para adaptarnos a él, obteniendo los resultados deseados.

Otro proceso interesante a tener en cuenta es la recopilación de información sobre otros puntos de investigación de este modelo, dándonos un enfoque más realista y claro de como funciona o como podría ser mejorado.

Una vez que nos hemos empapado de conocimiento sobre este modelo preentrenado, nos quedaría investigar posibles desarrollos que nos ayuden a conseguir que este modelo sea utilizado para crear sistemas de respuestas a preguntas y que es lo que necesitaríamos para lograrlo.

1.4.3 Datasets existentes para Question Answering

Como ya hemos comentado, sabemos que queremos conseguir que nuestro modelo funcione como un sistema de respuestas a preguntas, por lo tanto, llegados a este punto, debemos de conocer cuales son los conjuntos de datos que pueden ser de utilidad para lograr dichos objetivos, desde estudiar sus características hasta entender de que forma pueden ser adaptados a nuestra solución.

En este trabajo nos vamos a centrar en el dataset SQuAD, pero antes de elegirlo, debemos conocer los beneficios que nos aporta con respecto a los demás ya existentes, sabiendo de su utilidad como punto de investigación en este trabajo.

1.4.4 Dataset SQuAD

En el apartado anterior hemos logrado llegar a la conclusión sobre con que dataset vamos a trabajar, pero antes de empezar con él y realizar la pila de experimentos debemos, del mismo modo, conocer la estructura del conjunto de datos, cuales son los beneficios que nos aportan con respecto al resto de datasets existentes e identificar el margen de investigación que nos deja su uso.

Otro objetivo a tener en cuenta en este apartado es conocer la historia, así cómo, cuál ha sido la motivación para diseñar dicho corpus de datos. Esto nos va a ayudar a tener un concepto más general sobre él y nos va a permitir hacer suposiciones sobre los resultados que vamos a obtener.

1.4.5 Interacción entre el modelo BERT con el dataset SQuAD

Ya conocemos como están diseñados tanto el modelo BERT como el dataset SQuAD, por lo tanto, debemos de indagar en el funcionamiento de ambos juntos. De esta forma, debemos de adaptar el dataset de manera que el formato que se le da a cada uno de los registros sea comprendido por la capa de entrada del modelo BERT.

Una vez que conocemos la entrada, debemos de interpretar la salida del modelo, adaptando los resultados a aquello que queremos obtener y a partir de ahí, hacer suposiciones sobre su funcionamiento.

También, debemos de investigar otros puntos realizados por la comunidad de investigadores que hayan elegido el modelo BERT con SQuAD como punto de estudio. Este enfoque, nos permitirá conocer mejor el funcionamiento de su interacción, además de que es lo que provoca que el sistema se comporte de mejor o peor manera.

1.4.6 Definir experimentos

Uno de los grandes problemas con los que la comunidad de investigadores se está enfrentando actualmente en el campo de respuestas a preguntas, es conocer cuando una pregunta que se va a realizar no tiene respuesta entre los datos a los cuales el sistema tiene acceso. Por lo tanto, nuestro objetivo aquí es centrar nuestra estudio en este tópico, observando como se comporta el modelo BERT con este tipo de preguntas y en que momentos el problema se va a agravar o disminuir.

Por ello, vamos a definir unos experimentos que van a ser realizados mediante las respuestas a unas preguntas que nos vamos a plantear para, posteriormente, analizar los resultados.

Por lo tanto, vamos a comprobar como va a actuar nuestro modelo cuando en el entrenamiento se va a encontrar con este tipo de preguntas, es decir, evaluar el modelo, ver que tipo de resultados nos está dando y si estos son mejores o peores que los resultados que nos da cuando el modelo no tiene que lidiar con este tipo de preguntas.

Otro objetivo es comprobar si el modelo actúa mejor, igual o peor con diferentes cantidades de preguntas (con o sin respuestas). Para ello, lo entrenaremos de diferentes formas, dando distintos versiones del conjunto de datos al modelo. Para realizar este enfoque, debemos de diseñar de forma cuidadosa estas nuevas versiones que nos van a servir para estudiar este problema.

Por último, el objetivo final sería comprobar como se comporta el modelo cuando únicamente va a recibir preguntas sin respuesta en el momento de evaluar el modelo. De esta forma, podemos comprobar los resultados y lanzar nuestras teorías.

1.5 ESTRUCTURA DE LA MEMORIA

En este apartado podremos ver de una manera resumida y general, las diferentes partes de las que esta compuesto dicho trabajo, así, de un solo vistazo, conoceremos el contenido de cada sección ya que cada una de ellas tratará de explicar una parte importante del proceso.

1.5.1 Introducción

En este punto se ha hablado de cual es la motivación que nos lleva a realizar este trabajo de investigación, así como, de nuestra propuesta a seguir para conseguirlo. Además, hemos explicado cuales van a ser los objetivos que queremos conseguir en este trabajo, a través de las preguntas de investigación propuestas que nos ayuden a realizar nuestros experimentos y sacar unas conclusiones. También se ha hablado de las metodologías que se van a seguir con cada uno de los pasos y conocimientos que se deben de tener para proceder con el trabajo de investigación.

1.5.2 Antecedentes

En esta sección, se tratará de dar una visión general sobre el conjunto de conceptos y tecnologías necesarios a conocer, además de los estudios que se han realizado sobre este tema y que conclusiones se han obtenido de ellos.

1.5.3 Definición de experimentos

También, explicaremos el conjunto de pasos que debemos de seguir para lograr nuestros objetivos, que van desde entender nuestro problema hasta definir cuales van a ser los experimentos a llevar a cabo y como lo vamos a hacer.

1.5.4 Resultados

Trataremos, de igual manera, de responder a las preguntas que nos hemos planteado para este trabajo, para ello realizaremos algunas pruebas de investigación y métodos que nos ayuden a responderlas de la mejor forma posible .

1.5.5 Conclusiones

Una vez hemos realizado nuestro apartado de investigación, debemos de obtener algunas conclusiones sobre ello, que nos ayudarán a expresar dichas conclusiones y posibles desarrollos futuros que nos permitan a nosotros u otros investigadores continuar con dicho estudio.

CAPÍTULO 2

ANTECEDENTES

En este apartado el lector podrá entender las diferentes tecnologías y conocimientos que van a ser requeridos para la realización de este trabajo. Además de conocer los antecedentes, u otros puntos de investigación, que nos ayudarán a tener una base de conocimiento para continuar el proyecto de investigación.

De esta forma podremos visualizar los problemas que nos podemos encontrar durante este proceso y, cuales son las posibles soluciones o maneras de mitigación.

2.1 QUESTION ANSWERING

El procesamiento de lenguaje natural es un subcampo de la inteligencia artificial que ayuda a las computadoras a poder entender, interpretar y manipular el lenguaje humano desde diferentes fuentes de datos.

El procesamiento del lenguaje natural ayuda en la comunicación entre humanos y computadoras. Por ejemplo, puede ayudar a las computadoras a leer un conjunto de datos, interpretar estos datos y, a partir de ahí, sacar conclusiones, como puede ser, el sentimiento del texto escrito o que partes de este son las importantes.

El lenguaje es muy complejo de interpretar debido a los múltiples puntos de vista y maneras de expresar el mismo contenido. Por ello, estos datos que se encuentran a partir de fuentes desconocidas se denominan datos no estructurados, ya que en ellos pueden existir inconsistencias e incluso errores ortográficos que harían aún más difícil su interpretación (Kwiatkowski y *et al.*, 2019).

Para el procesamiento del lenguaje se necesita también un entendimiento del análisis sintáctico, como es la forma en la que las palabras se enlazan unas con otras y, además, el análisis semántico que nos ayuda a conocer si las frases que estamos analizando cobran o no sentido, es decir, tienen significado.

Por lo tanto, el procesamiento del lenguaje natural puede ser de utilidad para un gran repertorio de aplicaciones útiles. Pueden ir desde simples hasta más complejas, por ejemplo:

- Búsqueda de palabras claves, sinónimos ...
- Extracción de información de diferentes sitios, como puede ser de la web.
- Máquinas de traducción, reconocimiento del habla, asistentes personales ...
- Chats, agentes de soporte.
- Análisis de sentimiento, localizando cual es el sentimiento que se expresa en cada texto analizado.

- Question answering, sobre el que se basará dicho trabajo de investigación, que nos permite encontrar respuestas a preguntas dentro de un corpus de datos.

Los sistemas de respuestas a preguntas, en inglés Question Answering, responden a preguntas realizadas por humanos en lenguaje natural. Este tipo de sistemas funcionan traduciendo las sentencias en una representación que haga capaz al ordenador de entenderlas de forma que el sistema pueda localizar y responder a estas frases de forma comprensible para el ser humano.

Para poder generar una respuesta entendible y plausible, el sistema debe primero analizar tanto de forma sintáctica como de forma semántica.

2.1.1 Arquitectura general

En este apartado vamos a conocer como funciona, de una forma general, la arquitectura de un sistema de respuestas a preguntas. Para ello, nos vamos a basar en la Figura 2.1 explicando cada uno de los pasos que vemos en ella:

1. Primero se descomponen las preguntas en eventos más simples, de esta forma podemos generar preguntas menos complejas con las que trabajar (subpreguntas). De la misma forma, podemos conocer el orden que lleva cada una de estas subpreguntas. Para realizar la descomposición se debe de conocer la taxanomía[1] de las preguntas, ya que dependiendo del tipo de pregunta, se debe de manejar de una manera u otra. Una vez identificado podremos realizar la división de las preguntas complejas en preguntas simples.

2. Una vez tenemos las subpreguntas deben de ser enviadas a un sistema de respuesta de preguntas general, pero para ello, debemos de la misma forma conocer cuál va a ser el formato de salida del sistema.

3. En este paso, obtendremos el conjunto de respuestas para todas las subpreguntas analizadas en el paso anterior mediante un sistema de respuestas a preguntas de propósito general.

4. Por último, cogemos todas las respuestas del paso anterior y realizamos una serie de filtros. Por ejemplo, eliminar preguntas irrelevantes y, de esta forma, construimos una respuesta definitiva más compleja que sería la respuesta a la primera pregunta.

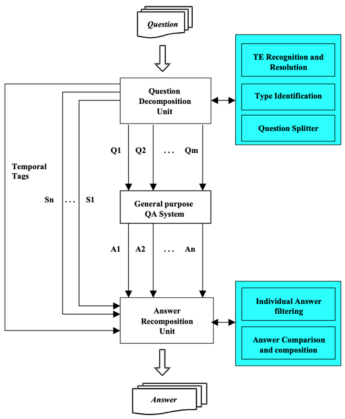

Figura 2.1: Figura arquitectura sistema QA

[1]Taxonomía: Ciencia que trata de los principios, métodos y fines de la clasificación.

2.1.2 Tipos de evaluación

Para conocer un poco más sobre este tipo de sistemas vamos a observar algunos de los tipos de evaluación o criterios que más se emplean en estos sistemas (Radev y *et al.*, 2002).

1. FHS o éxito del primer suceso: este tipo de evaluación comprueba si la primera respuesta dada por el sistema, el cual estamos evaluando, es la correcta. De este modo, si es correcta devuelve un 1 y de otro modo devuelve un 0.

2. FARR o primera respuesta de clasificación recíproca: comprueba todas las respuestas hasta que encuentra cual es la respuesta correcta, siguiendo un orden ascendente, de esta forma, calcula el éxito dividiendo 1 entre la posición de la respuesta correcta. Por ejemplo, si entre el conjunto de respuestas, la correcta es la número 4, el sistema de evaluación devolverá 1/4 y si ninguna de las respuestas proporcionadas es la correcta, se devolverá un 0.

3. FARWR o primera respuesta de clasificación recíproca de palabras: este criterio se basa en devolver el resultado contando desde la palabra que contiene la respuesta. Por ejemplo, si la respuesta dada es "ha nacido en el año 1940*z* la respuesta correcta es "1940", el sistema de evaluación devolvería un 6/6.

4. TRR o clasificación recíproca total: en muchas ocasiones existen más de una respuesta correcta, por lo que solo tener en cuenta la primera respuesta correcta sería un error. Este sistema de evaluación tendrá en consideración todas las respuestas aportadas asignándoles un "peso", clasificándolas de la forma que si de 10 respuestas la 2 y la 5 son correctas, el sistema daría como resultado, 1/2 + 1/5 = 7/10.

5. PREC o Precisión: este tipo de criterios, evalúan mediante la longitud total de caracteres que contienen todas las respuestas correctas, dividido entre la longitud total de caracteres de todas las respuestas aportadas en el sistema.

2.1.3 Metodología de evaluación

Varios sistemas de Question Answering (QA) responden a un set de preguntas a través de documentos ya dados y, a partir de ahí, son evaluadas las respuestas con respecto a las esperadas. La medida de evaluación utilizada depende de los objetivos de dicha evaluación (Rodrigo y Peñas, 2017).

Medidas centradas en preguntas correctas

El Accuracy es aquella medida que representa únicamente la proporción de preguntas correctamente respondidas, dejando así a un lado las preguntas respondidas de forma incorrecta y, además, está limitada a una respuesta por pregunta.

$$Accuracy = nc/n \tag{2.1}$$

En donde:
nc: *es el número de respuestas correctas*
n: *es el número total de preguntas*

La medida Mean Reciprocal Rank (MRR) se usa cuando se pueden dar varias respuestas por pregunta. La puntuación dado a cada pregunta es la recíproca del rango en el que se da la primera

respuesta correcta ó, 0 si no se da una respuesta correcta. La puntuación final es la media de todas los rangos recíprocos dados a todas estas preguntas.

MRR también intenta hacer ver el impacto de una respuesta errónea, ya que si se le aplica el mismo valor que a una no respuesta, a veces utilizar respuestas aleatorias pueden acertar en la predicción, dando valor a la puntuación final.

Medidas basadas en la puntuación de self-confidence

Las utilizadas para este tipo de medidas son dos:

- Confidence Weighted Score (CWS): está basada en Average precision. Los sistemas devuelven solo una respuesta por pregunta y el ranking de las respuestas se basa en la self confidence del sistema. CWS recompensa los sistemas que devuelven respuestas correctas al principio del ranking.

$$CWS = 1/n * SUM(C(i)/i) \tag{2.2}$$

$$C(i) = SUM(i(j)) \tag{2.3}$$

  En donde:
  n: *es el número de preguntas*
  i: *es la posición de la pregunta en el ranking*
  C(i): *es el número de respuestas correctas hasta el ranking i*

  CWS da más valor a preguntas en posiciones altas, ya que contribuyen más a la puntuación final.

- Correlación de pearson: utiliza dos medidas, K y K1, devolviendo 1 si la respuesta es correcta y -1 si no lo es. Ambas medidas pesan los valores con la puntuación de confianza dada por el sistema.

Reconociendo la falta de respuestas

Un buen sistema de QA con buena self-confidence debería de ser capaz de detectar y reducir respuestas incorrectas, a la vez que mantiene un buen nivel de respuestas correctas. Este método premia los sistemas que reducen respuestas incorrectas y mantienen las correctas. Es decir, si un sistema no está seguro sobre una respuesta, se evaluará más el no responder que el dar una respuesta incorrecta.

2.1.4 Question Answering usando Information Retrieval

En los años 90, la gran cantidad de documentos atrajo la idea de extraer información de textos de dichos documentos. La arquitectura típica de un sistema QA era una pipeline que abarca el análisis de la pregunta, obtención de documentos candidatos, extracción de la respuesta y selección de la respuesta.

Por lo tanto, mientras Information Retrieval (IR) devuelve el conjunto de documentos donde la gente busca por información, Question Answering (QA) busca en esos documentos las respuestas a las preguntas que pueden surgir.

Se sugirieron unos cambios en la arquitectura QA, en lugar de la clásica. La descrita anteriormente en forma de pipeline lleva a propagación de errores debido a la dependencia entre módulos. El cambio

sugerido fue el de introducir validación en QA. Después de generar todas las posibles respuestas, un módulo de validación tomará la respuesta definitiva.

Los tipos de validación existentes son los siguientes:

- Validación de la respuesta: un módulo de validación de respuesta recibe el par de <pregunta,respuesta>y devuelve un valor indicando si la respuesta es correcta o no. De esta manera, se reduce la cantidad de respuestas incorrectas dadas en un sistema de QA. Answer validation se puede realizar de dos formas:
  - Validación: descarta las respuestas incorrectas de un set de candidatos.
  - Selección: devuelve no más de una respuesta final de un set de candidatos.
- Validaciones basadas en redundancia: el paso de validación asume que la respuesta es aquella que más se repite, ya que tiene más probabilidades de ser la correcta.
- Validación basadas en similitud del texto: este método mide la similitud entre una combinación de respuestas con la pregunta. Este método funciona al nivel léxico, sintáctico y semántico.
- Validación basada en el análisis del texto: este método analiza conexiones entre las preguntas y las respuestas. Si las conexiones devuelven pruebas de que la respuesta fue encontrada, entonces la respuesta se considera como correcta.

2.1.5 La importancia del background en sistemas de QA

En una conversación normal en la que el emisor requiere que el receptor le de una respuesta, el emisor omite información en la pregunta ya que da por echo de que el receptor tiene el contexto necesario para entenderlo (Choi y *et al.*, 2018). El receptor, utiliza dicho contexto y el background para obtener una interpretación del mensaje. La solución que se ha intentado dar es la de proporcionar dicho background utilizando recursos externos. Sin embargo, es un campo todavía por investigar al que no se ha llegado a una conclusión.

Principales problemas que un sistema de QA puede encontrar

Este tipo de preguntas que utilizan un background puede darse principalmente en estos dos tipos de preguntas:

- Preguntas complejas:
  Este tipo de preguntas pueden ser aquellas que necesiten obtener respuestas de varias fuentes para poder responder a la pregunta final. Por ejemplo, si decimos "¿Qué temperatura hace en la zona más cálida de España?".

  Aquí necesitaríamos responder a varias preguntas que pueden estar en diferentes fuentes:

  Zona más cálida de España ->X

  Temperatura en X ->Y (la respuesta)
- Preguntas con inferencia textual:
  Este tipo de preguntas se da en aquellas que utilizan distintas formas de redacción a la hora de plantear la pregunta. Por ejemplo, si decimos "La vida es más sencilla que 3 años atrás", este tipo de preguntas es fácilmente entendible por un agente humano, pero no para un sistema automático, dado que el sistema automático no es capaz de reconocer el año al que el emisor hace referencia. Estos tipos de preguntas necesitarían de un contexto y un background como el que hemos mencionado en puntos anteriores.

2.2 MODELO BERT

El modelo preentrenado BERT es uno de los primeros sistemas de representación contextual bidireccional que utiliza como entrenamiento corpus de texto plano en lenguaje natural. Este modelo esta preentrenado de forma que controla tanto el contexto izquierdo como el derecho, de ahí que se denomine bidireccional, utilizando enmascaramiento, lo que le ayuda a prestar atención tanto al lado derecho como al izquierdo. Para comprender mejor este tipo de flujo bidireccional hemos consultado el artículo de la Universidad de Washington y el instituo de Inteligencia Artifical de Allen (Seo y *et al.*, 2016).

Para comprender un poco más como funciona BERT debemos también de conocer cuál es su arquitectura (Devlin y *et al.*, 2019). La cual analizaremos en el siguiente punto. Además, debemos de conocer de que capas está formado el modelo y de que forma se comporta (Lalande, 2019).

2.2.1 Arquitectura

El modelo BERT hace uso de transformadores con un mecanismo que aprende el contexto en ambas direcciones por las relaciones entre las palabras de un texto. Los transformadores incluyen dos mecanismos separados: - Un codificador que lee el texto de entrada. - Un decodificador que produce la predicción para la tarea.

El transformador codifica la entrada leída en una secuencia de palabras de una sola vez (derecha-izquierda e izquierda-derecha). Está característica bidireccional permite al modelo aprender el contexto de las palabras.

La entrada al modelo es una secuencia de tokens, los cuales en un principio están embebidos en vectores y son procesados en la red neuronal. La salida es una secuencia de vectores de tamaño H, en el cual, cada vector corresponde al token de entrada con el mismo índice.

La entrada en la Figura 2.2 al modelo, está diseñada de tal forma que entra un conjunto de tokens de hasta 512. En lugar de una oración en lenguaje natural, la entrada puede ser el contenido de una o dos oraciones. La entrada al modelo BERT se forma sumando los siguientes componentes:

- El token de entrada de WordPiece utilizando un vocabulario de 30.000 tokens.
- Un segmento de aprendizaje A para cada token de la primera oración, y de la misma manera, para una segunda oración si la hubiese.
- Las posiciones de cada token hasta los 512 que puedan existir.

En estas frases, se utilizan también tokens especiales para designar:

- Comienzo de frase [CLS]
- Final de frase [SEP]

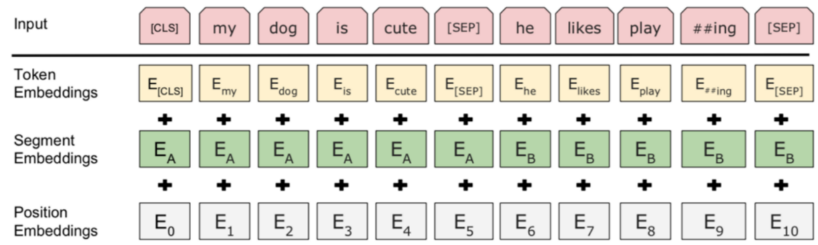

Figura 2.2: Figura input modelo BERT

2.2.2 Capas del modelo

Ya que hemos comentado cuales son las características de este modelo y su entrenamiento contextual bidireccional, vamos a conocer cuales son las capas que hacen posible dicha funcionalidad.

Masked LM (MLM)

Antes del entrenamiento de las secuencias de palabras, un 15 % de las palabras en cada una de estas secuencias es sustituida por un token [MASK]. Por lo tanto, el modelo intenta predecir cuál es el valor original de cada una de las palabras enmascaradas basándose en el contexto que provee el resto de las que no lo están.

La predicción de la salida requiere:

1. Añadir una capa de clasificación al principio del codificador de salida.
2. Multiplicar la salida de los vectores embebiendo la matriz, transformándola en la dimensión del vocabulario.
3. Calcular la probabilidad de cada palabra, utilizando la función “softmax”.

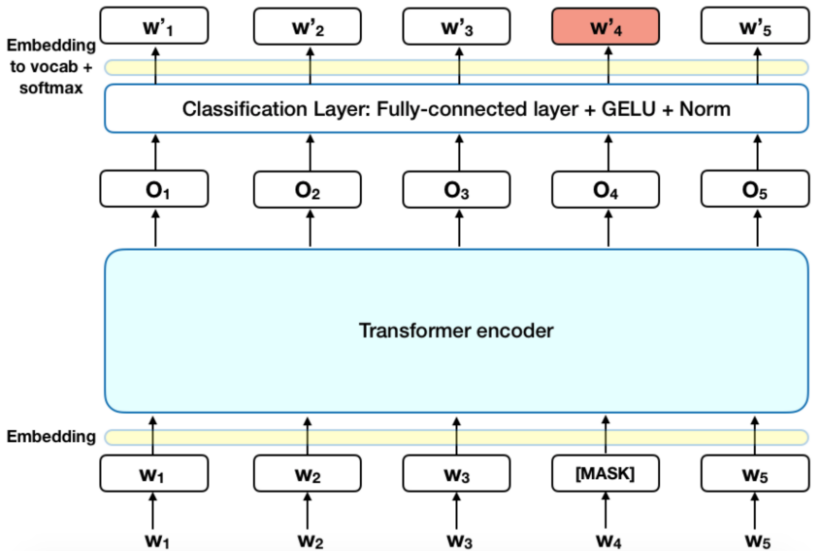

Figura 2.3: Figura masked LM

La función de pérdida del modelo BERT, toma solo en consideración los valores enmascarados e ignora las predicciones de las palabras no enmascaradas.

Next sentence prediction (NSP)

En el proceso del entrenamiento del modelo BERT, recibe como entrada un par de sentencias y aprende a predecir si la segunda de las secuencias es una subsecuencia del documento original.

Durante el entrenamiento, el 50 % de la entrada son un par, en la que la segunda frase es la subsecuencia de la sentencia original del documento y la otra mitad una sentencia aleatoria del mismo. La suposición es que la sentencia aleatoria será desconectada de la primera.

Para ayudar al modelo a distinguir entre las dos sentencias, la entrada se procesa de la siguiente forma:

1. Un token [CLS] se inserta al principio de la primera oración y el token [SEP] al final de la secuencia.
2. Una sentencia embebida es añadida a cada sentencia A y B, indicando a cada uno de los tokens a que oración pertenece.
3. Una posición es añadida a cada token para indicar la colocación de cada sentencia.

Para saber si la primera oración va después de la segunda, los pasos a seguir son:

1. Toda la entrada va a través del transformador del modelo.
2. La salida del token [CLS] es transformado en un vector de 2x1, usando una capa de clasificación.
3. Calcula la probabilidad de próxima secuencia utilizando la función softmax.

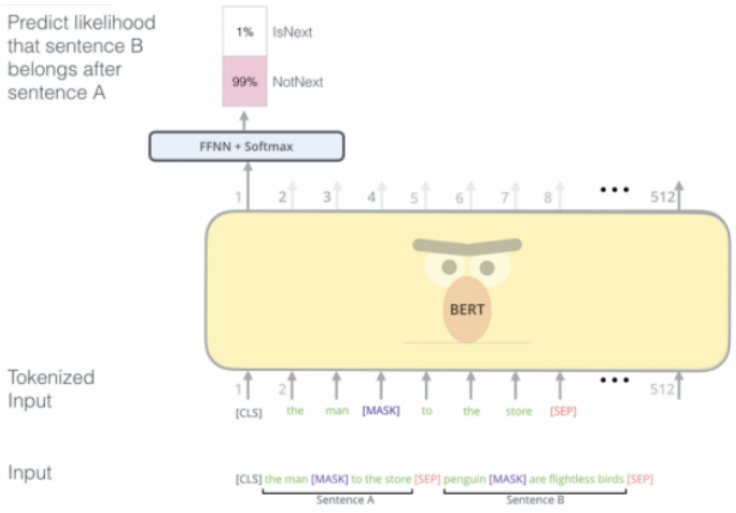

Figura 2.4: Figura NSP layer

Las dos capas anteriores son entrenadas juntas con el objetivo de minimizar la función de perdida.

2.2.3 Dataset existentes para QA

Como hemos comentado anteriormente, para obtener nuestros resultados sobre respuestas a preguntas utilizando el modelo preentrenado BERT necesitamos hacer uso de un buen dataset que nos ayude a entrenar el modelo y que pueda ser formateado de tal forma que nos sirva como entrada para este. Llegados a este punto, veremos algunos de los más utilizados para dichas tareas (Ruder, s.f.), (CHOUDHURY, s.f.).

Natural Questions (NQ)

Este dataset (Kwiatkowski y *et al.*, 2019) es un nuevo corpus para entrenar y evaluar sistemas de respuestas a preguntas, que tengan un dominio abierto. Diseñado por Google, es el primero en presentar un proceso end to end en el que las personas pueden encontrar las respuestas a las preguntas. Contiene 300.000 preguntas en lenguaje natural con respuestas anotadas por agentes humanos en páginas de Wikipedia. Además, se anotaron otros 16.000 ejemplos donde las respuestas a una pregunta eran dadas por 5 agentes diferentes, los cuales eran muy útiles, sobre todo, a la hora de evaluar el sistema.

Question Answering in Context

Question Answering in Context (Choi y *et al.*, 2018) contiene un conjunto de datos que sirven para modelar, comprender y participar en una especia de diálogo de búsqueda de información, en la que participan dos agentes, alumno y profesor. El papel del alumno consiste en realizar las preguntas con el objetivo de obtener la máxima información posible y aprender de forma libre. El papel del profesor consiste en responder a dichas preguntas proporcionando extractos de texto.

Este dataset está formado por 14k de diálogo de búsqueda de información, que incluye 100k pares QA en total.

HOTPOTQA

Este dataset (Z. Yang y *et al.*, 2018) contiene 113k pares (documentos Wikipedia y pregunta-respuesta). Estas son preguntas que requieren encontrar y razonar sobre múltiples documentos para responder a la pregunta, por lo tanto, esto ayuda al sistema de QA a realizar comparaciones y extraer hechos relevantes con la similitud necesaria entre los documentos.

MS MARCO

MS MARCO o Human Generated MAchine Reading COmprehension dataset (Bajaj y *et al.*, 2016), es un dataset creado por Microsoft AI Research. Este dataset contiene 1.010.916 de preguntas extraídas de los logs de Bing, cada una formada por unas 182.669 respuestas escritas por agentes humanos. El propósito de este dataset es totalmente no comercial, si no para mejorar e investigar sobre el campo de la inteligencia artificial y las áreas relacionadas.

TWEETQA

Este dataset (W. Xiong y *et al.*, 2019) está únicamente enfocado a las respuestas a preguntas en las redes sociales. Ha sido creado por investigadores de IBM y la universidad de California. Está compuesto por 10.898 artículos con 17.794 tweets y 13.757 pares de preguntas y respuestas.

NEWSQA

NewsQA (Trischler y *et al.*, 2016) esta centrado en el dominio de las noticias dadas por la CNN con aproximadamente 100.000 pares de preguntas-respuestas, basados en 10.000 artículos. Este dataset contiene 119.633 preguntas en lenguaje natural creadas por trabajadores sobre 12.744 noticias de los artículos de la CNN.

SQuAD

Este dataset (Rajpurkar y *et al.*, 2016) es el más importante en este trabajo y en el cuál pondremos más atención. Está diseñado para comprensión lectora que incluye preguntas creadas por trabajadores en el campo del lenguaje, sobre un conjunto de documentos de artículos escritos en Wikipedia y las respuestas a cada una de las preguntas es una parte del texto de estos documentos. Este dataset está formado por dos principales versiones, 1.1 y 2.0. En apartados futuros, veremos cuales son las principales diferencias entre ambos.

2.3 BERT COMO SISTEMA DE QA

Como hemos visto anteriormente, el modelo BERT permite la entrada de dos sentencias separadas cada una de ellas por una serie de tokens como son [CLS] y [SEP]. Conociendo está característica del modelo, vamos a utilizarlo de tal forma que nos ayude a dado un contexto[2] y una pregunta, que nos prediga tanto la posición inicial como la final en la que se encuentra la respuesta en dicho contexto (Alberti y *et al.*, 2019), (Hu, 2019).

Para entrenarlos, vamos a utilizar la arquitectura descrita anteriormente, en los que ambos pares de sentencias (contexto y pregunta) serán modificadas de tal forma que la entrada sea de la forma *[CLS]contexto[SEP]pregunta[CLS].*

Una vez dada la entrada, la salida contendrá una posición de inicio, llamémosla *I* y otra de final, *F*. Para conocer la probabilidad de que esos tokens son la posición de inicio y final de la respuesta a la pregunta, se calcula el productor softmax entre el Token de la posición inicial T_I y el token de la posición final T_F. Además, debe de existir una restricción clara que indique que la posición final siempre debe de ir después que la inicial.

Debido a las limitaciones que hemos ido observando en el modelo BERT utilizando como dataset la versión 2.0 de SQuAD, la comunidad de investigadores ha realizado algunas modificaciones o nuevos desarrollos sobre él, intentando mejorar los resultados (L. Yang y Song, 2019). En este punto veremos algún ejemplo (Peter Dun, 2018) de ellos y, así cómo, el objetivo de esa modificación.

Pointer Net

En esta aproximación se realizó una modificación en la capa de la línea base BERT por una capa del tipo Pointer Net, la cual, es una RNN[3] que genera como salida tamaños dinámicos.

La salida dada por esta capa da una distribución softmax sobre el tamaño de la entrada. La salida de esta capa da como lugar la posición inicial de la respuesta y la posición final de la misma. Cuando la posición inicial es mayor que la posición final se deduce que dicha respuesta no es plausible, es decir, no tiene respuesta para dicho contexto facilitado.

Dynamic Pointing Decoder

La mayoría de los sistemas de respuestas a preguntas no tienen una funcionalidad para recuperarse de preguntas que no tienen respuestas. Por lo tanto, este decodificador itera sobre los posibles índices de fin de respuesta y encuentra cuál puede ser el verdadero rango de la respuesta que estamos buscando. Para comprender mejor el funcionamiento de este decodificador el artículo facilitado por

[2]Contexto: porción de texto plano que utilizamos para encontrar la respuesta a nuestras pregunta. Este contexto puede venir, por ejemplo, de Wikipedia.

[3]RNN: Recurrent Neural Network

salesfore research (C. Xiong y *et al.*, 2017) te ayuda a comprenderlo y a ver cómo se ha utilizado en este problema.

Utilizando una multicapa maxout, el decodificador coge el estado oculto para generar nuevas estimaciones de los estados inicio y final.

Answer Chunk Ranker

La última capa del modelo BERT es una versión modificada de Answer Chunk Ranker, para comprender más el uso de esta capa, especialmente utilizando el modelo BERT, se puede consultar el artículo facilitado por el grupo de investigación de IBM (Yu y *et al.*, 2016). Una de las versiones de esta capa se utiliza para extraer fragmentos secuenciales como posibles respuestas potenciales. Las subsecuencias dadas por esta capa tienen un tamaño de longitud N, donde N está preestablecido por el usuario. Este diseño ha sido realizado para evitar la sobrecarga.

La segunda capa es la Ranker, la cual compara las representaciones de las respuestas que se han ido extrayendo y se utiliza la similitud coseno para comprobar cuál es la respuesta que más concuerda con la pregunta.

2.4 DATASETS PARA LOCALIZAR PREGUNTAS SIN RESPUESTA

Existen diversos conjuntos de datasets que se han creada principalmente para localizar preguntas sin respuestas[4]. Las respuestas a preguntas deben de ser a partir del contexto de un documento o artículo, por ello, para conocer cuando una pregunta no tiene respuesta existen los siguientes datasets:

- Zero-shot Relation Extraction (Levy y *et al.*, 2017): el cuál contiene un 65 % de preguntas sin respuesta que no tienen una respuesta plausible.
- TriviaQA (Joshi y *et al.*, 2017): obtiene los documentos por medio de la web o Wikipedia para cada una de las preguntas y, de la misma forma, muchos de estos documentos tienen preguntas que no se pueden responder por medio de él.
- SQuAD: para este dataset se realizó el método TF-IDF para localizar este tipo de preguntas sin respuestas en la que relacionaba la pregunta de un documento o artículo con los párrafos de dicho documento. Mediante esta técnica se pudieron localizar tanto las respuestas correctas como el posible ruido que se podía generar con ellas.
- NewsQA: este dataset (explicado anteriormente) también obtuvo preguntas sin respuestas, ya que los trabajadores daban cuestiones relacionadas con resúmenes de artículos en lugar de citar el mismo al completo.

2.5 CONCLUSIONES

Los trabajos realizados sobre el modelo BERT están mayormente centrados a intentar modificar el mismo, añadiendo nuevas capas, modificando la estructura del modelo ... Sin embargo, nadie se ha parado a pensar si dicho modelo funcionaría mejor únicamente cambiando la proporción de preguntas que usamos en un Dataset o el tipo de preguntas que necesitamos para entrenar nuestro modelo.

[4]Preguntas sin respuestas: son aquellos párrafos emparejados con alguna pregunta que no tiene respuesta en dicho párrafo.

Pudimos ver un cambio notable del rendimiento de este modelo cuando se intentaron introducir preguntas sin respuestas en el entrenamiento y la predicción, tanto que fue necesario realizar una modificación en el modelo que hiciera reconocer este tipo de preguntas. Aquí es donde nuestro trabajo cobra sentido, está claro que ese cambio mejoró notablemente el rendimiento en el modelo, pero nos debemos parar a pensar e investigar si el modelo no es capaz de reconocer todas estás preguntas debido a una falta de entrenamiento de ellas o, si por el contrario, podemos encontrarnos con un caso de overfitting [5]. Llegando así a un punto en el que nuestro modelo se haya entrenado con un número tal de preguntas sin respuestas que únicamente sea capaz de reconocer aquellas que son de alguna forma muy similares a estas.

Por ello, en este trabajo nos ha sido necesario conocer todos los puntos anteriormente vistos, de forma que podamos realizar nuestros experimentos y llegar a conclusiones o a un punto en el que otro grupo de investigación pueda utilizar estas deducciones para sacar nuevas teorías.

[5]Overfitting: en español conocido como "sobre-entrenamiento", se debe a cuando el modelo se ha entrenado de forma que su ajuste será únicamente capaz de reconocer aquello casos idénticos con los que se ha entrenado y no será capaz de reconocer nuevos datos de entrada.

CAPÍTULO 3

DEFINICIÓN DE LOS EXPERIMENTOS

Para lograr conseguir los objetivos anteriormente descritos, primero fue necesario analizar y encontrar un script que nos ayudara a estudiarlo de tal forma que pudiéramos comprender las entradas y las salidas del modelo para procesos de Question Answering. Por ello, vamos a utilizar el código subido a la plataforma GitHub (team, Oct 21,2018-May 11,2020) desarrollado con TensorFlow.

El modelo BERT, que hay en dicha plataforma, está ya preparado para ser entrenado con los dataset SQuAD, tanto para la versión 1.1 como para la versión 2.0. Pero dicho modelo necesita de tecnología TPU[1] para poder entrenarlo y evaluarlo en un tiempo razonable con el fin de realizar nuestras pruebas.

Una vez tenemos la tecnología necesaria para entrenar nuestro modelo, debemos de comprender como funciona y que acepta como entradas para obtener la deseada salida, en este caso el índice de inicio y fin de la respuesta en el texto.

Como entrada al modelo debemos de formatear cada registro del dataset SQuAD y combinar tanto las preguntas como las respuestas en un mismo valor. Para ello, la entrada se formará de tres capas diferenciadas:

- Token Embeddings: la entrada en esta capa estará formado por el conjunto de tokens del contexto, así como el de la pregunta, separados por un token [SEP] y el token [CLS] que indica inicio de contexto y final de pregunta.
- Position Embeddings: BERT ha sido diseñado para procesar entradas de secuencias de hasta 512 tokens de tamaño. Esta capa es un vector que representa cada posición. De esta forma, en frases del estilo “I think, therefore I am” diferencia la palabra “I”, de forma que la primera “I” no tenga la misma representación de posición que la segunda.
- Segment Embeddings: BERT utiliza esta capa para aprender a distinguir entre oraciones, de forma que, si tenemos un par (contexto, pregunta), cada token del contexto será sustituido por un “0” y cada token de la pregunta será sustituida por un “1”.

Estos elementos producirán una única entrada para el modelo BERT. A partir de esta entrada, el modelo predecirá el índice inicial y final de la respuesta en el contexto dado.

De está manera podemos ver la forma de utilizar el modelo preentrenado BERT para, con la ayuda del dataset SQuAD, crear un sistema que sea capaz de predecir la respuesta a una pregunta dentro de un contexto dado. Pero la complejidad empieza cuando existen preguntas que no tienen respuestas en el contexto, aquí es donde nuestra investigación toma forma para tratar de entender el funcionamiento del modelo con este tipo de datasets.

[1] Tensor Processing Unit

Para entender dicho funcionamiento hemos contemplado tres distintas preguntas, donde cada una conllevará un punto de investigación con unos resultados a ser evaluados. Estas preguntas son:

1. ¿Qué efecto tiene ver preguntas sin respuesta en el entrenamiento?

   Para responder esta pregunta entrenaremos el sistema con la versión 2.0, la cual contiene preguntas sin respuestas. Luego evaluaremos dicho modelo con ambas versiones v1.1 y v2.0.

2. ¿Cómo afecta ver durante el entrenamiento distintas proporciones de preguntas sin respuesta?

   Cogeremos el dataset SQuAD 2.0 y modificaremos el número de preguntas sin respuesta que aparecen creando una nueva versión. Realizaremos el mismo procedimiento con aquellas preguntas que si tienen respuesta. Una vez tenemos estas dos nuevas versiones, trataremos de evaluar el sistema para ver como ha afectado en él dicho cambio.

3. ¿Cómo actuaría el modelo entrenado con el dataset SQuAD 2.0 si al evaluarlo solo recibiera preguntas sin respuesta?

   Entrenaremos el modelo con el dataset SQuAD 2.0 y lo evaluaremos únicamente con una versión del dataset que solo tenga preguntas sin respuestas. De esta forma, veremos si dichas preguntas están afectando de forma muy negativa en el entrenamiento.

3.1 DATASET SQUAD

Debido a la gran cantidad de estudios realizados a la par entre el modelo BERT y el dataset de Stanford denominado SQuAD, nos hemos basado en dicho dataset para continuar con nuestra investigación. En este punto veremos cuál fue la motivación para crearlo, así como las diferencias fundamentales entre ambas versiones disponibles actualmente. De este modo podremos entender mejor cuál es nuestro objetivo llegados a este punto.

3.1.1 Motivación

Para la realización de este dataset se hizo uso de trabajadores especializados en el campo lingüista, otorgando una mayor fiabilidad a la hora de generar preguntas y decir cual es la respuesta plausible para cada una de ellas.

Cada uno de estos trabajadores escogía un párrafo de un articulo de Wikipedia. Partiendo de aquí, generaba cinco preguntas que solo podían ser respondidas mediante el artículo de Wikipedia que se les había facilitado.

El siguiente paso consistía en eliminar las preguntas de aquellos trabajadores que habían escrito en total menos de 25 preguntas en el artículo, eliminando así el posible ruido que pudiera crearse en este dataset. Digamos que es una especia de filtro para aquellos trabajadores que no habían entendido completamente el objetivo de la tarea. Este filtro no fue solo utilizado sobre las nuevas preguntas generadas, sino que también se realizó sobre aquellas ya existentes en el dataset.

Para la creación de la versión 2.0 de este dataset, se eliminaron también aquellos registros (artículos de Wikipedia) de las particiones de desarrollo y test que no se encontraban preguntas sin respuesta.

Una vez terminado con los filtros, se intentó dar otro punto de vista al dataset, en la que entraron a formar parte nuevos trabajadores que intentarían responder a todas las preguntas del dataset mediante los artículos existentes. Para reducir el ruido, se mantuvieron únicamente aquellas respuestas seleccionadas por la mayoría de los trabajadores.

3.1.2 SQuAD v1.1 vs SQuAD v2.0

El dataset SQuAD, como hemos visto anteriormente, está formado por un conjunto de preguntas hechas por especialistas del lenguaje basándose en artículos de Wikipedia donde la respuesta a cada una de las preguntas se encontraba en dichos artículos.

Este fue el primer objetivo a conseguir con este dataset, es decir, la versión 1.1. Dicho dataset estaba formado por más de 100.000 preguntas con sus correspondientes respuestas sobre más de 500 artículos de Wikipedia. En otras palabras, cada pregunta realizada en dicho dataset, contiene una respuesta plausible en una parte del artículo de Wikipedia que se le ha asignado para tal fin.

Sin embargo, se empezó a pensar como funcionarían los sistemas de respuestas a preguntas cuando se encontraran con preguntas que no tuviesen respuestas en los artículos facilitados con anterioridad. De esa idea nació la segunda versión de este dataset, SQuAD 2.0. En esta nueva versión se combinaron los 100.000 pares de preguntas con respuesta de la primera versión con otras 50.000 preguntas que no tienen una respuesta en el artículo de Wikipedia dado para tal fin. Para realizar dicha tarea los trabajadores crearon estas preguntas sin respuesta haciéndolas lucir de forma muy similar a aquellas que si tienen respuesta. El propósito de este dataset, es que los sistemas de respuesta a preguntas en un caso real, se van a encontrar continuamente con preguntas que no tienen respuesta en los textos. Por lo tanto, deben aprender a diferenciarlas y dar una respuesta lo más correcta posible.

En la siguiente tabla 3.1, podemos ver un ejemplo del contenido del dataset en su versión 2.0 y como están organizados los datos (Group, s.f.).

Tabla 3.1: Ejemplo SQuAD 2.0

Context	Question	Answer	Position	Answerable
Since at least the time of the Ancient Greeks, a proportion of Jews have assimil	When did the advent of the Jewish Enligtenment occur?	{18th century}	{427}	True
	What was a result of the Jewish Enlightenment?	{growing tren of assimilation}	{691}	True
	Name a Jewish community that disappeared entirely?	{Kaifeng Jews of China}	{335}	True
	Where did assimilation never take place?	{in all areas}	{246}	False
	What Jewish community has never disappeared?	{Kaifeng Jews of China}	{335}	False

En la tabla 3.1 podemos encontrar un registro formado por los siguientes elementos:

- Context: está formado por el contenido del documento del cual vamos a extraer únicamente la porción necesaria para encontrar nuestra respuesta.
- Question: contiene la pregunta que se ha realizado sobre el contexto anterior, esperando que la respuesta esté localizada en él.
- Answer: engloba la respuesta a la pregunta realizada situada en el contexto, que puede ser plausible o no.
- Position: muestra en que posición del contexto se encuentra la respuesta que se ha dado.
- Answerable: indica si la respuesta que se ha facilitado a la pregunta es plausible o no, es decir, si se puede responder a la pregunta con el contexto que se ha facilitado.

La versión SQuAD 2.0 se creó con el objetivo de tener un tamaño grande, diverso y, además, con poco ruido. Por ello, también se deben de tener en cuenta dos cosas:

- Relevancia: las preguntas sin respuesta deben también de ser relevantes en el tema sobre el que trata el artículo. Mediante heurística, podemos conocer cuando tiene o no respuesta plausible.
- Existencia de respuestas plausibles: en el párrafo donde se ha dado la respuesta, debe de existir algún dato que concuerde con el tipo de respuesta que se debe de generar para la pregunta realizada. Por ejemplo, para la pregunta *¿En que año nació Miguel de Cervantes?* debe de existir algún dato que indique una fecha que puede ser interpretada como fecha de nacimiento.

En la Figura 3.1 podemos ver una tabla con las principales diferencias estructurales entre ambas versiones del dataset.

	SQuAD 1.1	SQuAD 2.0
Train		
Total examples	87,599	130,319
Negative examples	0	43,498
Total articles	442	442
Articles with negatives	0	285
Development		
Total examples	10,570	11,873
Negative examples	0	5,945
Total articles	48	35
Articles with negatives	0	35
Test		
Total examples	9,533	8,862
Negative examples	0	4,332
Total articles	46	28
Articles with negatives	0	28

Figura 3.1: Figura comparación versiones SQuAD

3.1.3 Resultados BERT sobre SQuAD

En un principio, el modelo BERT fue probado utilizando los datos de entrenamiento del datset SQuAD v1.1, dando unos resultados muy alentadores al obtener un accuracy muy alto. Dicha evaluación fue dada mediante la métrica F1 que podremos ver más adelante en la sección de resultados.

Sin embargo, cuando la versión 2.0 fue publicada y se probó con el modelo BERT, los resultados bajaron considerablemente llegando a conseguir un accuracy de 73.1 F1 cuando la versión 1.1 llegó hasta el 88.5 F1. Como podemos comprobar en estos resultados, se observan unos cambios bastante radicales que dejan unas cuantas preguntas abiertas sin responder. Por lo tanto, aquí es donde debemos de obtener nosotros los resultados o intentar, por lo menos, mediante puntos de investigación acercarnos al posible problema que produce esta bajada de rendimiento.

La principal diferencia que podemos observar entre ambos datasets es que la versión 2.0 está formada, además, por preguntas que no tienen respuesta en los contextos dados para cada una de ellas. Por tanto, la investigación irá dirigida a este punto en concreto.

3.1.4 Datasets utilizados

En este apartado vamos a dar a conocer el conjunto de datasets que se han utilizado en este proyecto para intentar dar respuesta a las preguntas que nos hemos planteado. También argumentaremos que beneficios nos van a aportar estos datasets a la hora de la búsqueda de dichas respuestas.

Los dos principales datasets existentes actualmente son:

1. SQuAD 1.1: versión inicial del dataset SQuAD el cual contenía un conjunto de preguntas que podían ser respondidas dadas un contexto o documento de Wikipedia.
2. SQuAD 2.0: segunda versión de este dataset, a la cual se le añadieron algunas modificaciones, como saber si la pregunta tiene respuesta o no dado un documento o contexto. Este dataset nos será útil a la hora de realizar nuestros experimentos, dando mayor posibilidad de juego.

Ambos datasets tienen las correspondientes versiones para entrenamiento y evaluación. Los datasets de evaluación han sido los que se han utilizado para desarrollar nuestra línea base, en la que comprobaremos como funciona el modelo cuando no está entrenado, evaluándolo a través de uno de esos datasets.

Para este proyecto vamos a jugar con los datasets mencionados arriba. Además, vamos a generar nuestras propias versiones de ellos que nos ayudarán a responder las preguntas que nos hemos planteado.

Las diferentes versiones que nos vamos a encontrar sobre este dataset son:

- SQuAD 2.0 (modificado para que contenga más preguntas con respuesta que sin respuesta): este dataset nos va a ayudar con la segunda pregunta planteada para ver como actúa el modelo cuando se encuentra con más preguntas con respuesta que sin respuesta.
- SQuAD 2.0 (modificado para que contenga más preguntas sin respuesta que con respuesta): del mismo modo que el anterior dataset, este nos ayudará con la segunda pregunta que trata diferentes porciones de preguntas, haciéndonos ver el comportamiento del modelo cuando la porción de preguntas sin respuesta es mayor que el grupo de preguntas con respuesta.
- SQuAD 2.0 (modificado para tener únicamente preguntas sin respuesta): este dataset nos va a ayudar a responder a la tercera pregunta, en la que comprobaremos como se comporta el modelo si al evaluarlo solo se encuentra con preguntas sin respuesta.

Los datasets expuestos anteriormente son aquellos que utilizaremos para dar respuestas a nuestras preguntas. Estos datasets han sido diseñados una vez conocíamos que preguntas queríamos responder y, de que forma, podíamos hacerlo. Por lo tanto, el siguiente paso a dar es probarlos utilizando el código oficial de GitHub mencionado en apartados anteriores, haciendo las modificaciones y pruebas necesarias.

3.2 IMPLEMENTACIÓN

Antes de empezar con el proceso de desarrollo de los resultados, debemos de centrarnos en como los vamos a realizar y que herramientas o plataformas vamos a emplear. Para llevar a cabo el procedimiento de búsqueda de respuesta, primero debíamos de encontrar el código que nos ayudase a realizar el experimento de investigación. De este modo llegamos a la conclusión de utilizar el código del modelo preentrenado BERT realizado con TensorFlow [2].

En un principio, TensorFlow conlleva un proceso de investigación y entendimiento del código antes de poder proceder con la ejecución. Una vez encontrado el código, hubo que realizar un proceso de aprendizaje en TensorFlow y entender el código para a partir de ahí realizar las ejecuciones necesarias.

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

Uno de los fallos que se cometieron al principio es que al ejecutar el código, no se tuvieron en cuenta la búsqueda de los pesos preentrenados del modelo. Por lo tanto, se empezaron a lanzar experimentos que no daban para nada los resultados esperados. Tras un largo periodo de investigación, pudimos llegar a la conclusión que nosotros únicamente teníamos el script que cargaba el modelo, al igual que lanzaba el entrenamiento y las predicciones.

Tras buscar en diferentes fuentes, al final pudimos encontrar el modelo preentrenado [3] con los pesos actualizados llevados a cabo mediante un entrenamiento exhaustivo de diferentes fuentes de datos. Estos pesos nos sirvieron para poder empezar con nuestra línea base, la cual consistiría únicamente en ejecutar nuestro modelo preentrenado sin llegar a entrenarlo con nuestros datasets de investigación.

De esta forma, como en cualquier trabajo de investigación, tendríamos un punto de partida del cual poder empezar a realizar diferentes pruebas y obtener las correspondientes conclusiones a partir de los resultados.

Ya teníamos dos puntos importantes de salida, el script para entrenar nuestro modelo con los datasets necesarios y el modelo preentrando de diferentes fuentes con los pesos actualizados. Ahora debíamos de saber cuales iban a ser los parámetros que utilizaríamos para entrenar nuestro modelo, haciendo que quedaran equilibrados entre ellos, no sobrecargando la memoria de la GPU. Del mismo modo, tras consultar varias fuentes de investigación, llegamos a la conclusión de utilizar los siguientes parámetros:

Listado 3.1: Parámetros para entrenar nuestro modelo

```
# Required parameters
flags.DEFINE_string(
    "bert_config_file",
        "gs://<bucket>/bert_squad_config/bert_config.json",
    "The␣config␣json␣file␣corresponding␣to␣the␣pre-trained␣BERT␣
        model.␣"
    "This␣specifies␣the␣model␣architecture.")

flags.DEFINE_string("vocab_file",
    "gs://<bucket>/bert_squad_config/vocab.txt",
                    "The␣vocabulary␣file␣that␣the␣BERT␣model␣was␣
                        trained␣on.")

flags.DEFINE_string("bucket", "<bucket>",
                    "bucket␣to␣store␣and␣read␣the␣files")
# Other parameters
flags.DEFINE_string("train_file", "train-vX.X.json",
                    "SQuAD␣json␣for␣training.␣E.g.,␣
                        train-v1.1.json")

flags.DEFINE_string(
    "predict_file", "dev-vX.X.json",
    "SQuAD␣json␣for␣predictions.␣E.g.,␣dev-v2.0.json␣or␣
        test-v1.1.json")

flags.DEFINE_string(
    "init_checkpoint",
        "gs://mybucketgon/bert_squad_config/bert_model.ckpt",
    "Initial␣checkpoint␣(usually␣from␣a␣pre-trained␣BERT␣model).")

flags.DEFINE_bool("do_train", False, "Whether␣to␣run␣training.")

flags.DEFINE_bool("do_predict", True, "Whether␣to␣run␣eval␣on␣the␣
    dev␣set.")
```

[3] Modelo: https://storage.googleapis.com/bert$_{m}odels/2018_10_18/uncased_L-12_H-768_A-12.zip$

```
27
28 flags.DEFINE_bool("do_eval", False, "Whether␣to␣run␣eval␣on␣the␣
       dev␣set.")
29
30 flags.DEFINE_integer("train_batch_size", 16, "Total␣batch␣size␣for␣
       training.")
31
32 flags.DEFINE_integer("predict_batch_size", 8,
33                       "Total␣batch␣size␣for␣predictions.")
34
35 flags.DEFINE_float("learning_rate", 5e-5, "The␣initial␣learning␣
       rate␣for␣Adam.")
36
37 flags.DEFINE_float("num_train_epochs", 2.0,
38                    "Total␣number␣of␣training␣epochs␣to␣perform.")
39 flags.DEFINE_bool(
40     "version_2_with_negative", True,
41     "If␣true,␣the␣SQuAD␣examples␣contain␣some␣that␣do␣not␣have␣an␣
          answer.")
```

Siguiendo los anteriores pasos, fuimos ya capaces de obtener los resultados que estábamos buscando como punto inicial. Sin embargo, nos encontramos con otro gran problema, debido a la cantidad de datos que debíamos utilizar y el número de pruebas que se debían de lanzar, el tiempo que el algoritmo tardaba en realizar, tanto el entrenamiento como la predicción, era demasiado alto para realizar nuestro proceso de investigación.

De este modo, la única forma de acelerar este proceso era utilizando la GPU. Posteriormente nos percatamos además, que el script que seleccionamos para entrenar nuestro modelo contenía una serie de parámetros que nos ayudaría a encontrar la solución al problema con el que nos estábamos enfrentando en este momento:

Listado 3.2: Parámetros para entrenar nuestro modelo usando TPU

```
1 flags.DEFINE_bool("use_tpu", True, "Whether␣to␣use␣TPU␣or␣
      GPU/CPU.")
2 flags.DEFINE_integer(
3     "num_tpu_cores", 8,
4     "Only␣used␣if␣‘use_tpu‘␣is␣True.␣Total␣number␣of␣TPU␣cores␣to␣
          use.")
```

Utilizando los parámetros del listado 3.2 nuestro proceso de entrenamiento y de evaluación llevaría menos tiempo, de esta forma, tendríamos más tiempo para realizar diferentes pruebas y obtener los resultados deseados. La TPU [4] se denomina al conjunto de circuitos integrados personalizados específicos de aplicaciones (ASIC) de Google, que se utilizan para acelerar las cargas de trabajo de aprendizaje automático.

Ya conocemos que es la TPU y para que nos va a servir. Sin embargo, para utilizar las TPUs necesitaríamos de algún tipo de plataforma que nos proveyera de máquinas con este tipo de tecnología, además de permitirnos ejecutar nuestros procesos de entrenamiento y de predicción. De este modo, nos decantamos por la utilización de Google Cloud Platform[5], la cual nos podía, tanto proveer de este tipo de máquinas con la tecnología TPU[6], como con una herramienta denominada AI Platform[7], que ejecutara nuestro código en la nube utilizando dichas máquinas, almacenándonos los resultados en buckets que nos permitirán más adelante manipularlos.

Ya hemos conseguido identificar los diferentes pasos que debemos de seguir para realizar nuestro

[4] Unidades de procesamiento tensorial: https://cloud.google.com/tpu/docs/tpus?hl=es-419
[5] GCP: https://cloud.google.com/
[6] Tipos de TPU: https://cloud.google.com/tpu/docs/tpus?hl=esavailability
[7] AI Platform: https://cloud.google.com/ai-platform/?hl=es-419

proceso de entrenamiento y predicción, utilizando nuestros datasets seleccionados. No obstante, los resultados obtenidos por el script utilizado son predicciones basadas en el entrenamiento, pero de un simple vistazo, nosotros no podíamos conocer si los resultados obtenidos eran buenos o malos, lo que quiere decir que no teníamos ninguna métrica que nos evaluara dichos resultados.

Para ello, utilizamos otro script [8] que mediante los valores obtenidos en la predicción y los resultados que se deben de obtener que están localizados en el dataset de evaluación, realizaba las operaciones necesarias para obtener resultados entendibles y de un vistazo poder comprender si dichos resultados han sido buenos o malos.

Los principales resultados recibidos al ejecutar dicho script son los siguientes:

exact	Porcentaje de los resultados exactos obtenidos en el proceso de predicción.
f1	La métrica F1 que combina tanto la precisión como el recall en un único valor.
total	El número total de valores que se han evaluado.

[8] Evaluación: https://worksheets.codalab.org/rest/bundles/0x6b567e1cf2e041ec80d7098f031c5c9e/contents/blob/

CAPÍTULO 4

RESULTADOS

En este capítulo nos vamos a centrar en explicar los procedimientos que hemos seguido para realizar el estudio sobre el modelo BERT entrenado y evaluado con el dataset SQuAD . Además, mediante los resultados que hemos ido obteniendo, podemos comprobar o hacer suposiciones sobre el porque de esos resultados y si llegamos a conclusiones sobre como podríamos mejorar dichos resultados mediante los análisis realizados aquí.

Para explicar este punto nos vamos a plantear una serie de cuestiones que nos van a ayudar a entender mejor como funciona el modelo. También veremos como actúa cuando se ve expuesto a distintos datasets y formas de entrenamiento. Para realizar estas preguntas, debemos primero de conocer la línea base en la que nos vamos a centrar.

La línea base corresponde con el entrenamiento del modelo BERT preentrenado, utilizando el dataset SQUAD. Al principio, solo existía un estudio basado en la versión SQuAD 1.1, pero conforme esto fue evolucionando se creo una segunda versión con preguntas que no pueden ser respondidas por medio de los documentos disponibles, comparando los resultados con la primera versión dando como consecuencia una bajada en el rendimiento del modelo. Sin embargo, se sabe que el rendimiento es peor que con la versión anterior, pero no sabemos exactamente en que proceso o el motivo por el que el modelo está actuando de forma errónea.

Para ello, antes de plantearnos una serie de preguntas, vamos a realizar nuestra propia linea base, es decir, vamos a evaluar el modelo pero sin entrenarlo con ningún dataset. La evaluación la realizaremos con el Dataset 2.0, viendo los resultados obtenidos mediante este test. Al no estar entrenado, el rendimiento debe de ser muy bajo, pero esto nos va a servir como punto de partida para empezar a realizar nuestros experimentos.

De aquí nace nuestro proyecto, queremos realizar varios puntos de investigación que nos ayuden a responder esta pregunta principal, dividiéndola en otras preguntas más específicas que nos lleven, de forma objetiva, a comentar el motivo de esos resultados.

Las preguntas que nos hemos planteado en este trabajo son las siguientes:

- ¿Qué efecto tiene ver preguntas sin respuesta en el entrenamiento? Aquí debemos de tener claro cuál es nuestro objetivo. Nuestro objetivo es encontrar el comportamiento del modelo cuando se encuentra con preguntas a las que no debería de poder responder. Dependiendo del resultado podemos sacar nuestras propias conclusiones de como se comporta el modelo cuando tiene que hacer frente a este tipo de preguntas.

- ¿Cómo afectaría durante el entrenamiento ver distintas proporciones de preguntas sin respuestas? Nuestra finalidad en este punto es comprobar si el modelo funciona mejor cuando el número de preguntas sin respuesta es mayor que el número de preguntas con respuesta o, si por el contrario, tendría un mejor comportamiento con un número más pequeño de estas preguntas.

- ¿Cómo actuaría el modelo entrenado con el dataset que combina ambos tipos de preguntas y se evalúa únicamente utilizando preguntas sin respuesta? Aquí nos vamos a poder dar cuenta si el resultado obtenido por el modelo es muy bajo, alto o tiene unos resultados normales. De este modo podemos evaluar en primer plano que efecto tienen este tipo de preguntas en el modelo.

Para darle respuestas a las preguntas vamos a utilizar el código que podemos encontrar en GitHub, modificándolo de tal forma que nos ayude con nuestro objetivo. Para ello, podemos modificar los dataset cambiando el número de preguntas que aparecen en cada uno (preguntas con respuesta y sin respuesta), la forma de entrenarlo, (utilizando los datasets ya existentes y los modificados) y la forma de evaluarlos, comprobándolo, de igual manera, entre los datasets existentes y los modificados.

Los resultados se evaluarán de tal forma que podamos ver si los cambios que hemos realizado, intentando dar una respuesta a nuestra subpregunta, superan los resultados esperados o, si por el contrario, son aún peores que los de la línea base. Sin embargo, aunque los resultados sean peores que la línea base, esto nos puede ayudar de igual forma a comprobar la fuente del problema y en futuras investigaciones ayudarnos a mejorar nuestro modelo.

Ya que tenemos la información necesaria, como son las preguntas que queremos responder y los datasets que vamos a utilizar para responder estas preguntas, únicamente nos queda realizar los experimentos y obtener los resultados. Una vez tenemos estos resultados seremos capaces de sacar nuestras conclusiones. Antes de realizar los experimentos sobre las preguntas que nos hemos planteado hemos lanzado la línea base del experimento, como explicamos en apartados anteriores. De esta forma podemos, a partir de ella, ramificar las distintas preguntas con sus respectivas pruebas.

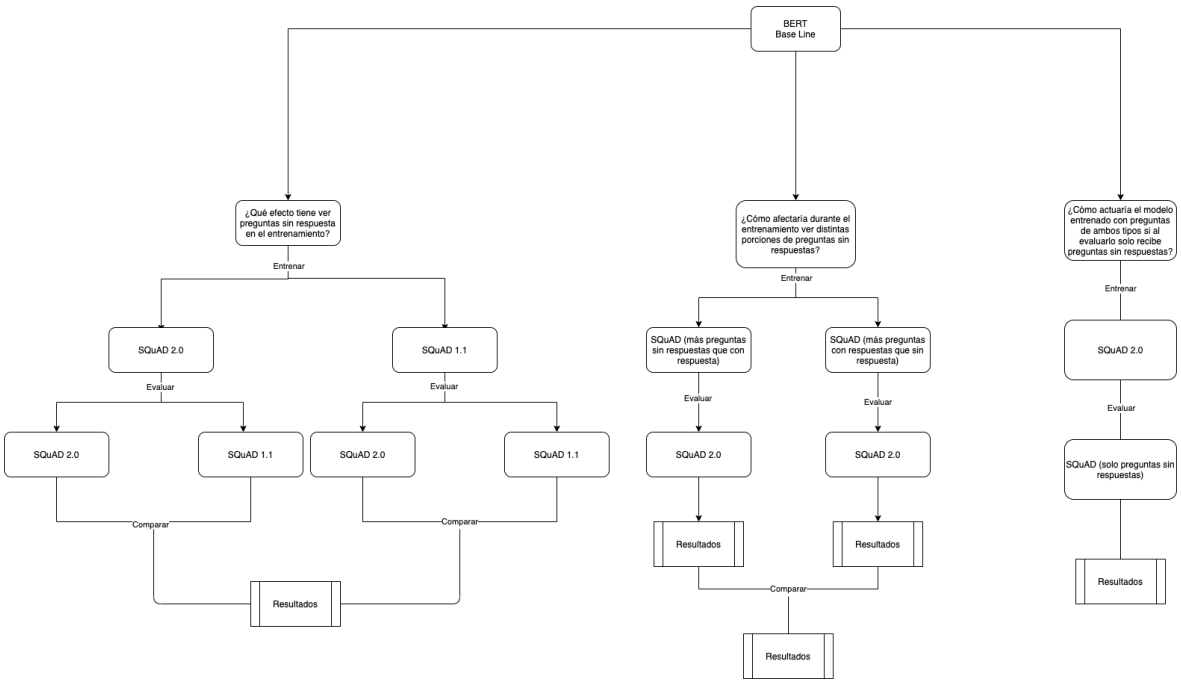

Figura 4.1: Figura del diagrama

Para lanzar la línea base, lo que hemos hecho es coger el código que contiene el diseño del modelo BERT. Sin entrenarlo, hemos lanzado directamente la evaluación con el dataset SQuAD 2.0. Los resultados obtenidos en esta prueba son los esperados con una evaluación en la métrica F1 de 4.029712229935384 %. A partir de estos resultados vamos a lanzar los distintos experimentos intentando mejorarlos y obtener nuestras conclusiones como podemos ver en el diagrama 4.1.

Este apartado se va a plantear de forma que vamos a intentar responder a cada pregunta. Dicho esto, formularemos la pregunta y explicaremos que pasos hemos seguido para intentar responderla.

Por ejemplo, que entrenamiento le hemos dado, con que dataset y cuál hemos utilizado para evaluarlo del mismo modo. También lanzaremos una teoría sobre la pregunta y los resultados que creemos que podemos obtener, de esta forma, mediante los resultados verdaderos, veremos si se acercan o no a nuestras suposiciones, sacando así nuevas conclusiones.

4.1 ¿QUÉ EFECTO TIENE VER PREGUNTAS SIN RESPUESTA EN EL ENTRENAMIENTO?

Para realizar este experimento vamos a dividirlo en dos ramas, uno para entrenarlo con el dataset SQuAD 1.1 y otro con el SQuAD 2.0. Partiendo de ahí, los evaluaremos al igual con ambos datasets, pudiendo ver como actúa el modelo cuando se encuentra con las preguntas sin respuesta en el entrenamiento y en la evaluación. Veremos si entrenarlo con un dataset que tenga este tipo de preguntas sin respuestas ayudará a la mejora en el rendimiento en la evaluación.

Antes de realizar el experimento, vamos a lanzar nuestras suposiciones sobre el resultado que vamos a obtener. Al entrenar el modelo únicamente con preguntas que tienen respuesta, el modelo va a estar preparado para enfrentarse a este tipo de preguntas. Pero, de igual forma, cuando se le facilite alguna pregunta sin respuesta el sistema, al no estar entrenado para reaccionar ante este tipo de situaciones, intentará buscar una respuesta incluso cuando esta no exista, bajando el rendimiento de este. Con el fin de resolver este problema, lanzamos la primera prueba de nuestra bifurcación entrenando el modelo con el SQuAD 1.1, evaluándolo con el mismo, suponiendo que reaccionará de forma correcta al estar entrenado con este tipo de preguntas. Más adelante, lo evaluaremos con el SQuAD 2.0, suponiendo que actuará erróneamente en la mayoría de los casos en los que se encuentre con preguntas sin respuesta.

Los resultados del anterior experimento fueron los siguientes:

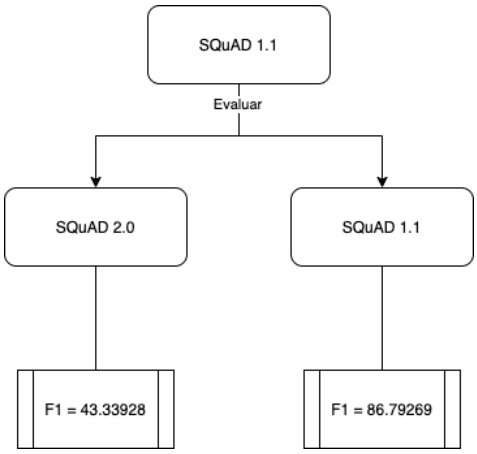

Figura 4.2: Figura de Resultados diagrama entrenado SQuAD 1.1

Como podemos observar en la Figura 4.2 los resultados son los esperados. Podemos observar que nuestras suposiciones son correctas. Dado que el modelo nunca se ha tenido que enfrentar en el entrenamiento a este tipo de preguntas, en el momento que se enfrenta a ellas, como podemos ver en la evaluación, no sabe que respuestas seleccionar, conduciendo el modelo al error, bajando notablemente la métrica F1 con la que se está evaluando.

Después de este experimento, ya nos queda más claro la razón que condujo al cambio de estrategia en el diseño del modelo. Realizando de este modo el entrenamiento del mismo con preguntas de ambos tipos, ayudando al modelo en la toma de decisiones. Por ello, vamos a entrenar ahora el modelo utilizando el dataset SQuAD 2.0. El modelo acepta nuevas entradas en las que nos dirá si la

pregunta tiene una respuesta posible o no (is_possible). Antes de realizar el experimento volveremos a hacer nuestras suposiciones sobre como va a reaccionar. Debido a que ahora le vamos a meter más complejidad al modelo, ya que tiene que saber si puede o no responder a una pregunta, suponemos que bajará su rendimiento en la respuesta con respecto al modelo entrenado y evaluado con SQuAD 1.1. Al ser capaz de reconocer este tipo de preguntas sin respuesta, ahora el modelo dará mejores resultados que anteriormente siendo entrenado únicamente con preguntas con respuesta.

Además, veremos como actúa el modelo entrenado con SQuAD 1.1 cuando se enfrenta otra vez únicamente a preguntas con respuesta. Veremos del mismo modo si el nuevo diseño del modelo ha afectado de alguna forma a los anteriores resultados. Al ser un diseño un poco más complejo, en mi opinión, el modelo bajará su rendimiento aunque no de forma muy notable, ya que seguirá siendo capaz de reconocer las preguntas con respuesta, pero ahora se le añade la nueva incógnita de si en realidad tiene o no respuesta.

Por lo tanto, los resultados que hemos obtenido con este experimento son los siguientes:

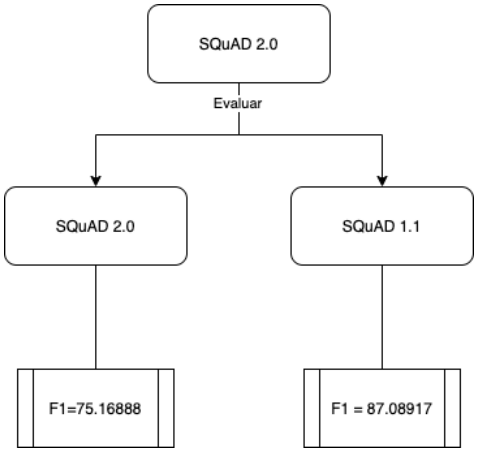

Figura 4.3: Figura de Resultados diagrama entrenado SQuAD 2.0

Como podemos ver en la Figura 4.3, para nuestra sorpresa, dichos resultados obtenidos al evaluar este nuevo diseño con el dataset SQuAD 1.1 son muy similares o incluso mejores que en el anterior experimento, por lo que de alguna forma este nuevo dataset ha mejorado el rendimiento o por lo menos lo ha mantenido. Además, como esperábamos, al evaluarlo con el SQuAD 2.0 los resultados son mejores que con el anterior experimento pero, no llegan a ser tan buenos como los resultados con el dataset que tiene únicamente preguntas con respuestas. Estos resultados son alentadores, ya que esto nos conduce más cerca de la mejora del rendimiento cuando nos encontramos con este tipo de preguntas a las cuales el modelo se enfrentará en un escenario real.

4.2 ¿CÓMO AFECTARÍA DURANTE EL ENTRENAMIENTO VER DISTINTAS PROPORCIONES DE PREGUNTAS SIN RESPUESTA?

De igual modo que en el apartado anterior, vamos a dividir nuestro experimento sobre esta pregunta en dos partes bien diferenciadas. Estas dos partes por separado nos ayudarán a obtener unos resultados que luego, al compararlos entre sí, extraeremos el conocimiento deseado. En este punto, queremos observar como va a actuar el modelo cuando se entrena con distintas proporciones de preguntas sin respuesta, ya que en algunos casos, los modelos pueden llegar al sobreaprendizaje y dar peores resultados. Queremos comprobar si el número de preguntas sin respuesta actual en el dataset SQuAD 2.0 son las óptimas o, si por el contrario, podría haber menos o más cantidad de preguntas para un mejor funcionamiento.

Con este fin, hemos creado dos datasets distintos a partir del ya existente SQuAD 2.0. Uno que contenga aproximadamente un 20 % del total de preguntas con respuesta y otro, que contenga un 20 % del total de preguntas sin respuesta. El código que hemos diseñado para crear estos datasets ha sido pensado de tal forma que separe en dos partes las preguntas con respuesta de las preguntas sin respuesta. Una vez las tiene diferenciadas todas, coge el 20 % de una de las dos, obteniendo como resultado final el 20 % de las preguntas de un tipo y el total de las preguntas del otro.

Listado 4.1: Algoritmo de creación de nuevos datasets

```
'''
Created on Jan 6, 2020

@author: pgonzalo
'''
import json

list_new_data = []
new_data = {'version':'2.1', 'data':list_new_data}
num_without = 0
num_with = 0
with open('train-v2.0.json') as json_file:
    data = json.load(json_file)['data']
    for par in data:
        new_par = {}
        list_par = []

        new_par['paragraphs'] = list_par
        new_par['title'] = par['title']
        for quest in par['paragraphs']:
            list_new = []
            list_qas = []
            new_q = {'context': quest['context'], 'qas': list_qas}
            without = []
            witha = []
            for qas in quest['qas']:
                if qas['is_impossible'] == False:
                    without.append(qas)
                else:
                    witha.append(qas)
                    num_with += 1
            tot = len(without) + len(witha)
            with_per = int(tot * 0.2)
            if len(without) < with_per:
                list_qas += without
                num_without += len(without)
            else:
                list_qas += without[:with_per]
                num_without += len(without[:with_per])
            list_qas += witha
            list_par.append(new_q)
        list_new_data.append(new_par)


with open('train-v2.1.json','w') as new_squad:
    new_squad.write(json.dumps(new_data))
print("WITHOUT %d", num_without)
print("WITH %d", num_with)
```

Ya tenemos nuestros dos datasets bien separados, ahora lo que vamos a hacer es entrenar nuestro modelo con los distintos datasets y obtener nuestros resultados.

SQuAD 2.0 (menos preguntas con respuestas)

En esta sección utilizaremos el dataset que contiene menos preguntas con respuesta que sin respuestas, en otras palabras, un mayor número de preguntas que no se pueden responder. En este tipo de experimentos es difícil suponer que es lo que va a pasar, ya que el mejor modo de comprobar su funcionamiento es mediante este tipo de experimentos, es decir, prueba y error. Sin embargo, el número de preguntas con respuesta es demasiado pequeño para entender cuál es la respuesta que corresponde a cada pregunta, por lo tanto, mi suposición sería decir que los resultados no van a ser muy alentadores.

Los resultados que obtuvimos fueron los siguientes:

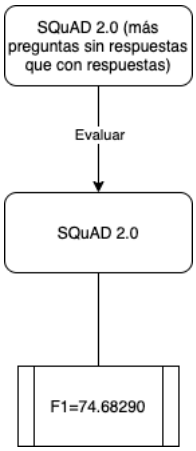

Figura 4.4: Figura de Resultados diagrama entrenado con SQuAD 2.0 (más preguntas sin respuestas)

En este caso, nos vuelven a sorprender los resultados obtenidos, incluso podemos ver que los resultados son muy cercanos a los obtenidos con el modelo entrenado y evaluado con el dataset SQuAD 2.0. Esto nos hace pensar que es muy probable que nos tengamos que centrar más en el número de preguntas sin respuesta que en el número de preguntas con respuesta, ya que el modelo está actuando incluso con un número bastante más bajo de preguntas con respuesta de un modo casi similar al original. Para ver si nuestra conclusión es correcta, vamos a realizar el mismo proceso, pero ahora cambiando los roles, dicho de otra manera, poniendo más preguntas con respuesta que sin respuesta.

SQuAD 2.0 (menos preguntas sin respuesta)

Una vez que hemos visto el resultado anterior, vamos a utilizar el mismo código que hemos empleado en el Listado 4.1, pero para generar un nuevo dataset que contenga el 20 % del total de preguntas sin respuestas y todas las preguntas con respuestas del dataset SQuAD 2.0. Una vez generado dicho dataset, vamos a coger nuestro modelo y entrenarlo utilizando dicho dataset, para evaluarlo más adelante con la versión dev del dataset SQuAD 2.0 y comprobar los resultados obtenidos.

Viendo los resultados obtenidos en el anterior experimento, podemos pensar que los resultados que vamos a obtener ahora deben de ser peores que el original y el anterior, ya que parece ser que para que el modelo tenga un rendimiento más óptimo necesita de un gran número de preguntas sin respuesta que le haga entender a que tipo de preguntas se está enfrentando para poder actuar de una forma más correcta.

Siguiendo la línea de nuestro anterior proceso, antes de realizar el experimento diría que el modelo tiene malos resultados pero no solo debido a que necesite más preguntas sin respuesta, si no que el modelo no será capaz de entender cuando una pregunta tiene respuesta o no debido al poco

número de ellas en el paso del entrenamiento. Por tanto, siguiendo los resultados del experimento anterior, como el conocimiento que tenemos sobre el funcionamiento del modelo, los resultados que obtendríamos son peores que muchos de los resultados anteriores, aunque mejores que cuando el modelo no entendía de preguntas sin respuesta.

Los resultados obtenidos son:

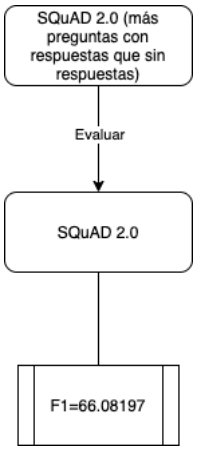

Figura 4.5: Figura de Resultados diagrama entrenado con SQuAD 2.0 (más preguntas con respuestas)

En este caso parece que estábamos en lo cierto, los resultados son peores que los obtenidos en la anterior investigación, por lo que podemos decir que la realización del mencionado experimento nos ha ayudado a sacar conclusiones correctas sobre este proceso, acercándonos más a la teoría de que el modelo reacciona de forma más correcta cuando se entrena con un número mayor de preguntas sin respuesta.

Como investigación a futuro, sería buen punto de partida crear nuevas preguntas sin respuesta, con la ayuda de expertos lingüístas, como los utilizados en la creación de los dataset SQuAD. Estas preguntas deben tener el mismo formato que las anteriores, lanzando nuevas pruebas con distinto número de preguntas sin respuesta, siempre mayores que el número actual. De esta forma quizás podamos alcanzar el número óptimo de preguntas y mejorar considerablemente nuestro modelo.

Por lo tanto, al comparar los resultados de ambos datasets con porciones distintas de tipos de preguntas podemos confirmar que el modelo está actuando mejor cuando mayor es el número de preguntas sin respuesta. Para saber de seguro el motivo, deberíamos de coger como línea base este apartado de investigación y seguir haciendo pruebas para llegar a más conclusiones, obteniendo el resultado de este comportamiento.

4.3 ¿COMO ACTUARÍA EL MODELO ENTRENADO CON EL DATASET QUE COMBINA AMBOS TIPOS DE PREGUNTAS Y SE EVALÚA ÚNICAMENTE UTILIZANDO PREGUNTAS SIN RESPUESTA?

Por último, otro campo a explorar sería comprobar como actuaría un modelo ya entrenado con el dataset SQuAD 2.0 ante la entrada de preguntas en la evaluación sin respuesta. Este experimento nos va a ayudar a probar si el modelo es capaz de localizar este tipo de preguntas y actuar acorde al entrenamiento que se le ha proporcionado. Viendo los anteriores experimentos realizados nos han quedado muchas cosas claras, pero tenemos también que conocer si el diseño del modelo no es del todo correcto o necesita algún ajuste para que reconozca de forma correcta este tipo de preguntas. Por ende, aquí pueden ocurrir dos cosas distintas y bien diferenciadas:

1. Buenos resultados, por encima del 70 %: el modelo es capaz de diferenciar este tipo de preguntas, de esta forma, puede que nuestro modelo esté bien diseñado y necesitemos mejorar otros aspectos, como puede ser los datasets o el número de preguntas en cada uno (como hemos visto en las investigaciones anteriores).

2. Malos resultados: el modelo necesitaría ser revisado, de forma que al entrenarlo con este tipo de preguntas sea capaz de diferenciarlas de forma correcta.

Para crear el dataset que nos va a servir en este experimento vamos a escoger otra vez el dataset SQuAD 2.0, reutilizando el código que hemos visto en el Listado 4.1. Vamos a quedarnos esta vez únicamente con las preguntas sin respuesta, descartando totalmente las que tienen respuesta.

Listado 4.2: Algoritmo de creación de nuevos datasets

```
'''
Created on Jan 6, 2020

@author: pgonzalo
'''
import json

list_new_data = []
new_data = {'version':'2.3', 'data':list_new_data}
num_without = 0
num_with = 0
with open('dev-v2.0.json') as json_file:
    data = json.load(json_file)['data']
    for par in data:
        new_par = {}
        list_par = []

        new_par['paragraphs'] = list_par
        new_par['title'] = par['title']
        for quest in par['paragraphs']:
            list_new = []
            list_qas = []
            new_q = {'context': quest['context'], 'qas': list_qas}
            without = []
            witha = []
            for qas in quest['qas']:
                if not qas['is_impossible'] == False:
                    without.append(qas)
                    num_without += 1

            list_qas += without

            list_par.append(new_q)
        list_new_data.append(new_par)


with open('dev-v2.3.json','w') as new_squad:
    new_squad.write(json.dumps(new_data))
print("WITHOUT %d", num_without)
```

Una vez hemos creado nuestro dataset, vamos a reutilizar también el modelo que entrenamos con el dataset SQuAD 2.0. Lo evaluaremos con nuestro nuevo dataset, por lo que los resultados que hemos obtenido en este experimento son los siguientes:

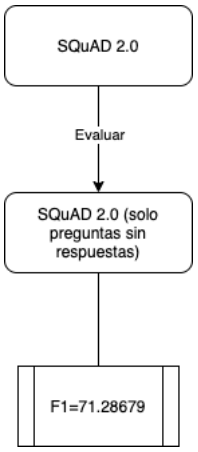

Figura 4.6: Figura de Resultados diagrama evaluado con preguntas sin respuestas

Los resultados obtenidos aquí nos dejan ver que el rendimiento está por encima del 70 % y, por lo tanto, podemos suponer que el modelo está desarrollado de forma correcta para localizar estas preguntas sin respuesta, hecho que nos hace pensar que los experimentos futuros deben de ir enfocados a otros puntos de investigación, como el anterior, utilizando diferentes porciones de preguntas. Además si es necesario, utilizar distintos datasets para comprobar cuales funcionan mejor e ir descartando opciones y posibilidades.

4.4 TIEMPO EMPLEADO

En este apartado vamos a centrarnos en explicar de un vistazo el conjunto de tareas que se han llevado a cabo para la realización de este trabajo de investigación y las horas aproximadas que nos ha llevado cada una de estas tareas.

Tarea	Descripción	Tiempo
Lectura de artículos BERT	Tiempo empleado en leer documentación sobre el modelo, entendiendo su estructura y funcionamiento	150 h
Lectura de artículos SQuAD	Tiempo empleado en leer documentación sobre el dataset, para conocer cuál fue la motivación para crearlo y cómo se hizo	70 h
Lectura de artículos BERT y SQuAD	Tiempo empleado para leer artículos y documentos en los que el modelo y los datasets trabajan juntos	90 h
Búsqueda del script	Tiempo empleado en buscar el script que se utilizará para obtener los resultados	12 h
Aprendizaje TensorFlow	Tiempo empleado en aprender las funciones necesarias de TensorFlow y entender de esta forma el script	72 h
Aprendizaje del formato de SQuAD	Tiempo empleado en entender el formato de ambas versiones del dataset SQuAD para poder aplicarlo a nuestro script	30 h
Búsqueda modelo preentrenado	Tiempo empleado en buscar el modelo BERT preentrenado que mejor funcionará para nosotros	12 h
Aprendizaje AI Platform usando TPU	Tiempo empleado en entender y aplicar la tecnología TPU, utilizando el producto de GCP AI Platform	72 h
Pruebas iniciales	Tiempo empleado en realizar pruebas antes de realizar los experimentos finales	160 h
Ejecución de la línea base	Tiempo empleado en obtener los resultados de nuestra línea base	10 h
Dar respuesta a la pregunta 1	Tiempo empleado en entrenar, predecir y evaluar los resultados para dar respuesta a la pregunta 1	30 h
Dar respuesta a la pregunta 2	Tiempo empleado en entrenar, predecir y evaluar los resultados para dar respuesta a la pregunta 2	65 h
Dar respuesta a la pregunta 3	Tiempo empleado en entrenar, predecir y evaluar los resultados para dar respuesta a la pregunta 3	20 h
Creación de nuevos scripts	Tiempo empleado en la creación de nuevos scripts que nos ayudaron a dar respuesta a algunas preguntas	4 h
Total	Tiempo total empleado en las tareas	797 h

Tabla 4.1: Tabla que muestra el tiempo empleado en cada tarea

CAPÍTULO 5

CONCLUSIONES Y TRABAJO FUTURO

En este apartado vamos a ver una tabla que nos va a mostrar todos los resultados obtenidos de forma global, mostrando qué pregunta nos ha ayudado a responder, qué dataset hemos utilizado para entrenar nuestro modelo, cuál hemos utilizado para realizar el proceso de predicción y, por último, qué resultados hemos obtenido.

Pregunta	Entrenamiento	Predicción	Evaluación
Baseline	SQuAD 2.0	SQuAD 2.0	exact: 0.10949212498947192 f1: 4.029712229935384 total: 11873
¿Qué efecto tiene ver preguntas sin respuesta en el entrenamiento?	SQuAD 1.1	SQuAD 1.1	exact: 78.12677388836329 f1: 86.79269446573387 total: 10570
		SQuAD 2.0	exact: 38.72652236166091 f1: 43.339288173929354 total: 11873
	SQuAD 2.0	SQuAD 1.1	exact: 78.63765373699148 f1: 87.08917129825087 total: 10570
		SQuAD 2.0	exact: 71.32990819506443 f1: 75.16888254020685 total: 11873
¿Cómo afectaría durante el entrenamiento ver distintas proporciones de preguntas sin respuesta?	SQuAD 2.0 (menos preguntas con respuesta que sin respuesta)	SQuAD 2.0	exact: 70.59715320475027 f1: 74.68290191575093 total: 11873
	SQuAD 2.0 (menos preguntas sin respuestas que con respuestas)	SQuAD 2.0	exact: 63.581234734271035 f1: 66.08197405570465 total: 11873
¿Como actuaría el modelo entrenado con el dataset que combina ambos tipos de preguntas y se evalúa únicamente utilizando preguntas sin respuesta?	SQuAD 2.0	SQuAD 2.0 (solo preguntas sin respuesta)	exact: 71.28679562657696 f1: 71.28679562657696 total: 5945

Tabla 5.1: Tabla de Resultados obtenidos de forma global

Como podemos apreciar en la Tabla 5.1, hemos realizado diferentes procedimientos de entrenamiento, predicción y evaluación para responder a nuestras preguntas. Gracias a ello, tenemos unos resultados que nos sirven de conocimiento y como forma de obtener unas conclusiones sobre ellos.

La conclusión obtenida en este caso es que en un principio el modelo preentrenado BERT no tiene suficiente información para ser capaz de obtener buenos resultados en una tarea de respuestas a preguntas. De ahí nace la idea de buscar un dataset que nos permita transformar dicho modelo de tal forma que sea capa de realizar estas tareas.

En el mundo actual no todas las preguntas tienen una respuesta, por lo tanto, los resultados obtenidos por el dataset SQuAD 1.1 únicamente son de relevancia para compararlos con los resultados obtenidos en las versión 2.0, versión que contiene preguntas sin respuesta. Del mismo modo, al igual que hubo que adaptar el dataset para este tipo de preguntas, se debe de hacer lo mismo con el modelo, ya que debe de ser capaz de entender este tipo de preguntas. Dicha mejora la podemos ver de forma clara en la Tabla 5.1 respondiendo a nuestra primera pregunta, en la cuál el modelo entrenado con SQuAD 2.0 tiene una mejora considerable con respecto al entrenado únicamente con SQuAD 1.1.

Podemos decir por tanto, que el cambio ha sido para mejor y que esto ha ayudado a mejorar los resultados.

Sin embargo, los resultados no son siempre los deseados y, necesitamos comprender como podríamos mejorar dicho modelo o, por lo menos, encontrar en que partes es más vulnerable. Hemos visto que antes ha obtenido una mejora considerable al poder entrenarse con preguntas sin respuesta, por ello, una buena prueba es ver cuantas de estas preguntas necesitaría el modelo para reconocerlas y funcionar mejor. De aquí nace la segunda pregunta, y la respuesta podría ser que cuantas más preguntas sin respuesta existen, mejor funciona el modelo ya que es capaz de identificarlas desde un principio, evitando de esta forma buscar una respuesta cuando en realidad no la hay. Este punto podría ser considerada una buena línea base por donde empezar un nuevo camino de investigación y descubrir cuál es el número idóneo de preguntas sin respuesta con las que el modelo se debe de entrenar para obtener mejores resultados.

La última pregunta nos ayudará a conocer si el modelo es capaz de reconocer este tipo de preguntas. Viendo los resultados podemos deducir que el modelo si es capaz de reconocer hasta en un 71 % este tipo de preguntas y reaccionar de forma correcta.

No solo vamos a centrarnos en conclusiones sobre los resultados que hemos obtenido, si no que también, como conclusiones, podemos llegar a explicar como objetivos, haber conseguido el aprendizaje de varios temas que al empezar este trabajo no conocíamos. Por lo tanto, hemos conseguido meternos dentro del modelo y conocer más a fondo las capas que lo forman y cómo funciona dicho modelo. De esta manera, nos ha ayudado de forma muy beneficiosa a proponernos unos objetivos y preguntas para poder obtener los resultados precisos y necesarios para este trabajo. También hemos podido ver como este modelo dependiendo del contenido puede reaccionar de una forma u otra, además de poder diseñarlo para distintas soluciones, no únicamente para sistemas de respuestas a preguntas.

También, hemos aprendido a utilizar TensorFlow y hemos sido capaces de ejecutar nuestro modelo de forma más rápidas utilizando máquinas en la nube que nos han ayudado a acelerar el proceso de nuestros experimentos y llegar a conclusiones de una forma más activa. Todos estos objetivos nos han servido de forma conjunta a conseguir nuestro objetivo final. Pero además, nos quedamos con todo lo que hemos aprendido realizando este trabajo, y la forma en la que nos ha ayudado a realizar una investigación para que en un futuro podamos hacer lo mismo y sepamos que pasos o pautas debemos de seguir para conseguirlo.

Como trabajo futuro, se debería de escoger los anteriores puntos de investigación y utilizarlos como línea base para, desde ahí, realizar más experimentos y plantearnos más preguntas que nos lleven a nuevos puntos de partida.

Antes mencionamos que un buen punto de partida sería la pregunta número dos, pero esta, podría ser combinada con la pregunta número 3 para ir viendo la evolución del modelo en cuanto a reconocer este tipo de preguntas y mejorarlo desde esa perspectiva.

De igual modo, nuevos desarrolladores, serían capaces de adaptar tanto el modelo como los datasets de forma que los resultados fueran mejorando poco a poco. Así, nuevos investigadores podrían lanzar nuevas hipótesis e intentar demostrarlas mediantes nuevos y más avanzados en el tiempo puntos de investigación, creando de este modo un ciclo continuo de mejora. El dataset que hemos utilizado en este caso es el de Stanford SQuAD. Sin embargo, estas mismas pruebas pueden ser realizadas utilizando otros de los Datasets existentes para tareas de Question Answering y ver si los resultados son mejores, iguales o peores que los realizados en este trabajo.

ANEXO A

TOPOLOGÍA MODELO BERT

Una red neuronal como la de BERT, es una red neuronal multicapa que es una generalización de una red neuronal monocapa, ya que mientras la red monocapa únicamente se compone de una capa de neuronas de entrada y una capa de neuronas de salida, la red multicapa, como su propio nombre indica, está formada además por un conjunto de capas intermedias o capas ocultas entre la capa de entrada y la capa de salida (Calvo, s.f.).

Dependiendo del número de conexiones que presente la red esta puede estar total o parcialmente conectada.

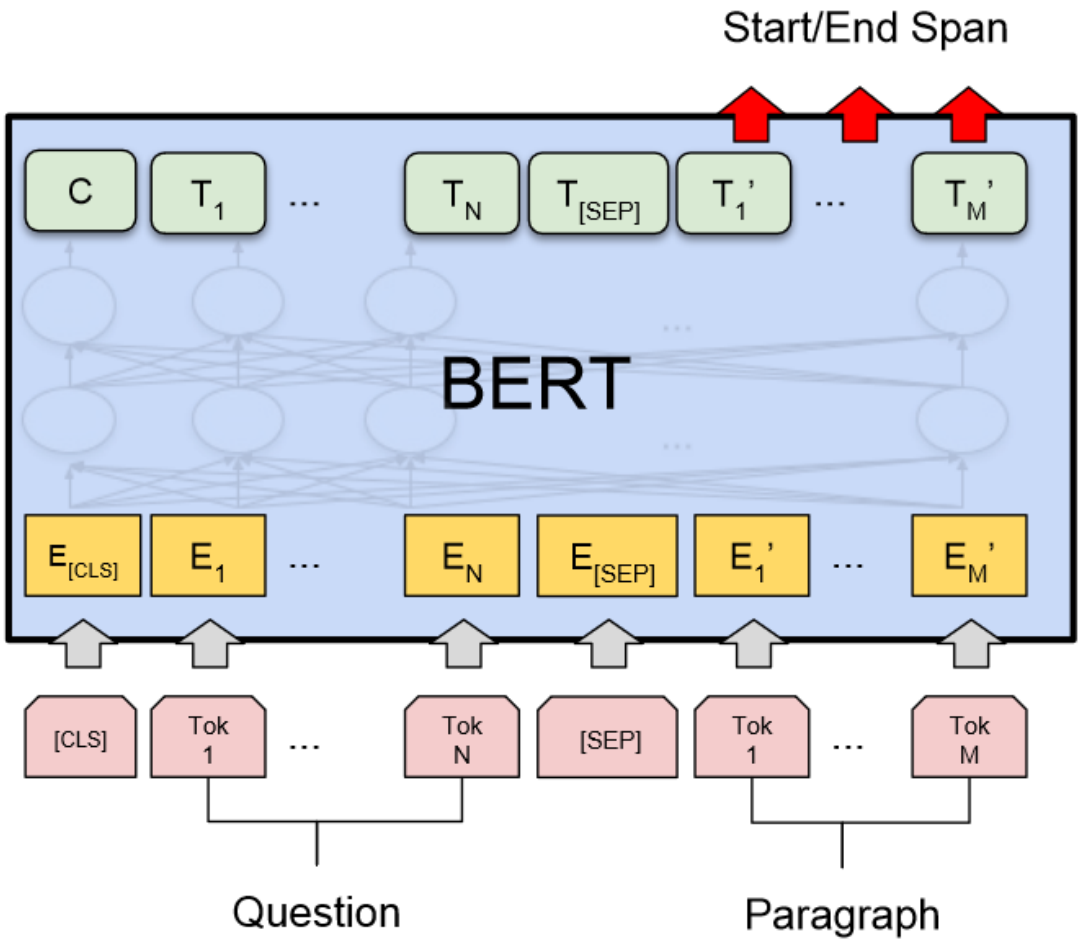

Figura A.1: Figura que muestra la topología del modelo BERT

ANEXO B

CONFIGURACIÓN NECESARIA

En este Anexo veremos el formato del bert_config.json, parámetro requerido a la hora de lanzar nuestros procesos de entrenamiento y predicción. Otros dos archivos fundamentales a tener en cuenta es el vocab file (vocab.txt) y el modelo bert preentrenado, pero estos archivos son demasiado grandes o el formato no permite mostrarlo aquí en este trabajo, pero lo mencionamos de forma que sepamos que existen y son importantes.

Listado B.1: Fichero $bert_config$

```
{
  "attention_probs_dropout_prob": 0.1,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "max_position_embeddings": 512,
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "type_vocab_size": 2,
  "vocab_size": 30522
}
```

ANEXO C

PARÁMETROS UTILIZADOS

En este Anexo haremos una visión global de los parámetros más importantes que se han utilizado en el script para ejecutar nuestro entrenamiento y predicciones.

Listado C.1: Parámetros más importantes

```
# Required parameters
flags.DEFINE_string(
    "bert_config_file",
        "gs://<bucket>/bert_squad_config/bert_config.json",
    "The config json file corresponding to the pre-trained BERT
        model. "
    "This specifies the model architecture.")

flags.DEFINE_string("vocab_file",
    "gs://<bucket>/bert_squad_config/vocab.txt",
                    "The vocabulary file that the BERT model was
                        trained on.")

flags.DEFINE_string("bucket", "<bucket>",
                    "bucket to store and read the files")
# Other parameters
flags.DEFINE_string("train_file", "train-vX.X.json",
                    "SQuAD json for training. E.g., 
                        train-v1.1.json")

flags.DEFINE_string(
    "predict_file", "dev-vX.X.json",
    "SQuAD json for predictions. E.g., dev-v2.0.json or
        test-v1.1.json")

flags.DEFINE_string(
    "init_checkpoint",
        "gs://mybucketgon/bert_squad_config/bert_model.ckpt",
    "Initial checkpoint (usually from a pre-trained BERT model).")

flags.DEFINE_bool("do_train", False, "Whether to run training.")

flags.DEFINE_bool("do_predict", True, "Whether to run eval on the
    dev set.")

flags.DEFINE_bool("do_eval", False, "Whether to run eval on the
    dev set.")

flags.DEFINE_integer("train_batch_size", 16, "Total batch size for
    training.")

flags.DEFINE_integer("predict_batch_size", 8,
                     "Total batch size for predictions.")

```

```
flags.DEFINE_float("learning_rate", 5e-5, "The initial learning rate for Adam.")

flags.DEFINE_float("num_train_epochs", 2.0,
                   "Total number of training epochs to perform.")
flags.DEFINE_bool(
    "version_2_with_negative", True,
    "If true, the SQuAD examples contain some that do not have an answer.")
```

ANEXO D

CREACIÓN DEL MODELO

En este Anexo veremos la clase del modelo BERT y el método que crea dicho modelo.

D.1 CLASE BERT

Listado D.1: Clase del modelo BERT

```
class BertModel(object):
  """BERT model ("Bidirectional Encoder Representations from
      Transformers")."""


  def __init__(self,
               config,
               is_training,
               input_ids,
               input_mask=None,
               token_type_ids=None,
               use_one_hot_embeddings=False,
               scope=None):

    config = copy.deepcopy(config)
    if not is_training:
      config.hidden_dropout_prob = 0.0
      config.attention_probs_dropout_prob = 0.0

    input_shape = get_shape_list(input_ids, expected_rank=2)
    batch_size = input_shape[0]
    seq_length = input_shape[1]

    if input_mask is None:
      input_mask = tf.ones(shape=[batch_size, seq_length],
          dtype=tf.int32)

    if token_type_ids is None:
      token_type_ids = tf.zeros(shape=[batch_size, seq_length],
          dtype=tf.int32)

    with tf.variable_scope(scope, default_name="bert"):
      with tf.variable_scope("embeddings"):
        # Perform embedding lookup on the word ids.
        (self.embedding_output, self.embedding_table) =
            embedding_lookup(
             input_ids=input_ids,
             vocab_size=config.vocab_size,
             embedding_size=config.hidden_size,
             initializer_range=config.initializer_range,
```

```
37              word_embedding_name="word_embeddings",
38              use_one_hot_embeddings=use_one_hot_embeddings)
39
40            # Add positional embeddings and token type embeddings, then layer
41            # normalize and perform dropout.
42            self.embedding_output = embedding_postprocessor(
43                input_tensor=self.embedding_output,
44                use_token_type=True,
45                token_type_ids=token_type_ids,
46                token_type_vocab_size=config.type_vocab_size,
47                token_type_embedding_name="token_type_embeddings",
48                use_position_embeddings=True,
49                position_embedding_name="position_embeddings",
50                initializer_range=config.initializer_range,
51                max_position_embeddings=config.max_position_embeddings,
52                dropout_prob=config.hidden_dropout_prob)
53
54        with tf.variable_scope("encoder"):
55
56            attention_mask = create_attention_mask_from_input_mask(
57                input_ids, input_mask)
58
59            # Run the stacked transformer.
60            self.all_encoder_layers = transformer_model(
61                input_tensor=self.embedding_output,
62                attention_mask=attention_mask,
63                hidden_size=config.hidden_size,
64                num_hidden_layers=config.num_hidden_layers,
65                num_attention_heads=config.num_attention_heads,
66                intermediate_size=config.intermediate_size,
67                intermediate_act_fn=get_activation(config.hidden_act),
68                hidden_dropout_prob=config.hidden_dropout_prob,
69                attention_probs_dropout_prob=
70                    config.attention_probs_dropout_prob,
71                initializer_range=config.initializer_range,
72                do_return_all_layers=True)
73
74        self.sequence_output = self.all_encoder_layers[-1]
75
76        with tf.variable_scope("pooler"):
77            first_token_tensor = tf.squeeze(self.sequence_output[:,
                0:1, :], axis=1)
78            self.pooled_output = tf.layers.dense(
79                first_token_tensor,
80                config.hidden_size,
81                activation=tf.tanh,
82                kernel_initializer=create_initializer(
83                    config.initializer_range))
```

D.2 CREACIÓN MODELO BERT

Listado D.2: Método creación modelo BERT

```
def create_model(bert_config, is_training, input_ids, input_mask,
    segment_ids,
                 use_one_hot_embeddings):
  """Creates a classification model."""
  model = modeling.BertModel(
      config=bert_config,
      is_training=is_training,
      input_ids=input_ids,
      input_mask=input_mask,
      token_type_ids=segment_ids,
      use_one_hot_embeddings=use_one_hot_embeddings)

  final_hidden = model.get_sequence_output()

  final_hidden_shape = modeling.get_shape_list(final_hidden,
      expected_rank=3)
  batch_size = final_hidden_shape[0]
  seq_length = final_hidden_shape[1]
  hidden_size = final_hidden_shape[2]

  output_weights = tf.get_variable(
      "cls/squad/output_weights", [2, hidden_size],
      initializer=tf.truncated_normal_initializer(stddev=0.02))

  output_bias = tf.get_variable(
      "cls/squad/output_bias", [2],
          initializer=tf.zeros_initializer())

  final_hidden_matrix = tf.reshape(final_hidden,
                                   [batch_size * seq_length,
                                      hidden_size])
  logits = tf.matmul(final_hidden_matrix, output_weights,
      transpose_b=True)
  logits = tf.nn.bias_add(logits, output_bias)

  logits = tf.reshape(logits, [batch_size, seq_length, 2])
  logits = tf.transpose(logits, [2, 0, 1])

  unstacked_logits = tf.unstack(logits, axis=0)

  (start_logits, end_logits) = (unstacked_logits[0],
      unstacked_logits[1])

  return (start_logits, end_logits)
```

ANEXO E

LECTURA DEL DATASET SQUAD

En este Anexo veremos como se lee del dataset SQuAD y se transforma al formato que entienda más adelante el modelo. Este código fue modificado para que leyera este dataset guardado en un bucket en GCS, ya que íbamos a lanzar los procesos de entrenamiento y predicción desde Google Cloud Platform.

E.1 LECTURA DEL DATASET DESDE GOOGLE STORAGE

Listado E.1: Método de lectura desde GCS

```
def readFromGCS(is_training):
    from google.cloud import storage
    bucket = storage.Client().get_bucket(FLAGS.bucket)
    blob = None
    if is_training:
        blob = bucket.get_blob(FLAGS.train_file)
    else:
        blob = bucket.get_blob(FLAGS.predict_file)
    data = blob.download_as_string()

    return json.loads(data)
```

E.2 FORMATEADO DEL DATASET

Listado E.2: Método que formatea cada registro de SQuAD

```
def read_squad_examples(input_file, is_training):
  """Read a SQuAD json file into a list of SquadExample."""
  '''with tf.gfile.Open(input_file, "r") as reader:
    input_data = json.load(reader)["data"]'''
  input_data = readFromGCS(is_training)["data"]
  def is_whitespace(c):
    if c == "␣" or c == "\t" or c == "\r" or c == "\n" or ord(c)
        == 0x202F:
      return True
    return False

  examples = []
  for entry in input_data:
    for paragraph in entry["paragraphs"]:
      paragraph_text = paragraph["context"]
      doc_tokens = []
      char_to_word_offset = []
      prev_is_whitespace = True
      for c in paragraph_text:
        if is_whitespace(c):
          prev_is_whitespace = True
        else:
          if prev_is_whitespace:
            doc_tokens.append(c)
          else:
            doc_tokens[-1] += c
          prev_is_whitespace = False
        char_to_word_offset.append(len(doc_tokens) - 1)

      for qa in paragraph["qas"]:
        qas_id = qa["id"]
        question_text = qa["question"]
        start_position = None
        end_position = None
        orig_answer_text = None
        is_impossible = False
        if is_training:

          if FLAGS.version_2_with_negative:
            is_impossible = qa["is_impossible"]
          if (len(qa["answers"]) != 1) and (not is_impossible):
            raise ValueError(
                "For␣training,␣each␣question␣should␣have␣exactly␣1␣
                    answer.")
          if not is_impossible:
            answer = qa["answers"][0]
            orig_answer_text = answer["text"]
            answer_offset = answer["answer_start"]
            answer_length = len(orig_answer_text)
            start_position = char_to_word_offset[answer_offset]
            end_position = char_to_word_offset[answer_offset +
                answer_length -
                                                1]
            # Only add answers where the text can be exactly recovered from the
            # document. If this CAN'T happen it's likely due to weird Unicode
            # stuff so we will just skip the example.
            #
            # Note that this means for training mode, every example is NOT
            # guaranteed to be preserved.
```

```
57              actual_text = " ".join(
58                  doc_tokens[start_position:(end_position + 1)])
59              cleaned_answer_text = " ".join(
60                  tokenization.whitespace_tokenize(orig_answer_text))
61              if actual_text.find(cleaned_answer_text) == -1:
62                tf.logging.warning("Could not find answer: '%s' vs. "
                      "'%s'",
63                                   actual_text, cleaned_answer_text)
64                continue
65            else:
66              start_position = -1
67              end_position = -1
68              orig_answer_text = ""
69
70          example = SquadExample(
71              qas_id=qas_id,
72              question_text=question_text,
73              doc_tokens=doc_tokens,
74              orig_answer_text=orig_answer_text,
75              start_position=start_position,
76              end_position=end_position,
77              is_impossible=is_impossible)
78          examples.append(example)
79
80    return examples
```

ANEXO F

EJECUTAR SCRIPT EN AI-PLATFORM

F.1 ARCHIVO SETUP

Este archivo es necesario para instalar aquellas dependencias que no están por defecto instaladas en las maquinas que utiliza AI-platform para realizar el entrenamiento.

En este caso nosotros instalamos la librería de Google Cloud Storage, ya que leemos de aquí nuestros datasets.

Listado F.1: Archivo setup

```
import setuptools

setuptools.setup(
    name ="main",
    version="1.0",
    install_requires=['google-cloud-storage'],
    packages=setuptools.find_packages())
```

F.2 CONFIGURACIÓN DE LAS MAQUINAS

Con este archivo indicamos todos los parámetros necesarios que se van a utilizar cuando entrenemos nuestro modelo, como puede ser, tipos de maquinas, workers, cores, ...

Listado F.2: Archivo config.yaml

```
trainingInput:
  scaleTier: CUSTOM
  masterType: n1-highcpu-16
  workerType: cloud_tpu
  workerCount: 1
  workerConfig:
    acceleratorConfig:
      type: TPU_V2
      count: 8
```

F.3 COMANDO DE DESPLIEGUE

En esta sección veremos el comando que se ha utilizado para desplegar nuestro código a AI-platform y en el que indicamos los archivos necesarios que debe de recoger, como el anterior config.yaml.

En cuanto al setup.py no es necesario de indicarlo, ya que automáticamente es leído y recogido cuando se encuentra en el mismo directorio que el script.

Listado F.3: Comando AI-Platform

```
gcloud ai-platform jobs submit training jobname --package-path
    main   --module-name main.main   --job-dir gs://<bucket>/bert
    --region us-central1   --config config.yaml --runtime-version
    1.14
```

BIBLIOGRAFÍA

FUENTES ONLINE

Calvo, Diego (s.f.). *Perceptrón Multicapa – Red Neuronal*. URL: http://www.diegocalvo.es/perceptron-multicapa/.

CHOUDHURY, AMBIKA (s.f.). *10 QUESTION-ANSWERING DATASETS TO BUILD ROBUST CHATBOT SYSTEMS*. URL: https://analyticsindiamag.com/10-question-answering-datasets-to-build-robust-chatbot-systems/.

Group, Stanford NLP (s.f.). *Script modelo BERT*. Stanford. URL: https://rajpurkar.github.io/SQuAD-explorer/.

H., Mood Shukri (s.f.). *How the Embedding Layers in BERT Were Implemented*. URL: https://medium.com/@_init_/why-bert-has-3-embedding-layers-and-their-implementation-details-9c261108e28a.

Ruder, Sebastian (s.f.). *NLP-progress*. URL: http://nlpprogress.com/english/question_answering.html.

team, Googe research (Oct 21,2018-May 11,2020). *Script modelo BERT*. Google. URL: https://github.com/google-research/bert.

FUENTES NO ONLINE

Alberti, Chris, Kenton Lee y Michael Collins (2019). «A BERT Baseline for the Natural Questions». En: *CoRR* abs/1901.08634. arXiv: 1901.08634. URL: http://arxiv.org/abs/1901.08634.

Bajaj, Payal y *et al.* (2016). *MS MARCO: A Human Generated MAchine Reading COmprehension Dataset*. arXiv: 1611.09268 [cs.CL].

Choi, Eunsol, He He, Mohit Iyyer, Mark Yatskar, Wen-tau Yih, Yejin Choi, Percy Liang y Luke Zettlemoyer (2018). *QuAC : Question Answering in Context*. arXiv: 1808.07036 [cs.CL].

Devlin, Jacob, Ming-Wei Chang, Kenton Lee y Kristina Toutanova (2019). «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding». En: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Ed. por Jill Burstein, Christy Doran y Thamar Solorio. Association for Computational Linguistics, págs. 4171-4186. DOI: 10.18653/v1/n19-1423. URL: https://doi.org/10.18653/v1/n19-1423.

Hu, Zhangning (2019). *Question Answering on SQuAD with BERT*. Stanford University.

Joshi, Mandar, Eunsol Choi, Daniel S. Weld y Luke Zettlemoyer (2017). «TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension». En: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. Vancouver, Canada: Association for Computational Linguistics.

Kwiatkowski, Tom y *et al.* (2019). «Natural Questions: a Benchmark for Question Answering Research». En: *Transactions of the Association of Computational Linguistics*.

Lalande, Kevin M. (2019). *Final Project: SQuAD 2.0 with BERT*. Department of Computer Science.

Levy, Omer, Minjoon Seo, Eunsol Choi y Luke Zettlemoyer (2017). «Zero-Shot Relation Extraction via Reading Comprehension». En: *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017), Vancouver, Canada, August 3-4, 2017*. Ed. por Roger Levy y Lucia Specia. Association for Computational Linguistics, págs. 333-342. DOI: 10.18653/v1/K17-1034. URL: https://doi.org/10.18653/v1/K17-1034.

Peter Dun Lauren Zhu, David Zhao (2018). *Extending Answer Prediction for Deep Bidirectional Transformers*. Stanford University.

Peters, Matthew E., Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee y Luke Zettlemoyer (2018). «Deep Contextualized Word Representations». En: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*. Ed. por Marilyn A. Walker, Heng Ji y Amanda Stent. Association for Computational Linguistics, págs. 2227-2237. DOI: 10.18653/v1/n18-1202. URL: https://doi.org/10.18653/v1/n18-1202.

Radev, Dragomir R., Hong Qi, Harris Wu y Weiguo Fan (2002). «Evaluating Web-based Question Answering Systems». En: *Proceedings of the Third International Conference on Language Resources and Evaluation, LREC 2002, May 29-31, 2002, Las Palmas, Canary Islands, Spain*. European Language Resources Association. URL: http://www.lrec-conf.org/proceedings/lrec2002/sumarios/301.htm.

Rajpurkar, Pranav, Robin Jia y Percy Liang (2018). «Know What You Don't Know: Unanswerable Questions for SQuAD». En: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*. Ed. por Iryna Gurevych y Yusuke Miyao. Association for Computational Linguistics, págs. 784-789. DOI: 10.18653/v1/P18-2124. URL: https://www.aclweb.org/anthology/P18-2124/.

Rajpurkar, Pranav, Jian Zhang, Konstantin Lopyrev y Percy Liang (2016). *SQuAD: 100,000+ Questions for Machine Comprehension of Text*. arXiv: 1606.05250 [cs.CL].

Rodrigo, Álvaro y Anselmo Peñas (2017). «A study about the future evaluation of Question-Answering systems». En: *Knowl. Based Syst.* 137, págs. 83-93. DOI: 10.1016/j.knosys.2017.09.015. URL: https://doi.org/10.1016/j.knosys.2017.09.015.

Seo, Min Joon, Aniruddha Kembhavi, Ali Farhadi y Hannaneh Hajishirzi (2016). «Bidirectional Attention Flow for Machine Comprehension». En: *CoRR* abs/1611.01603. arXiv: 1611.01603. URL: http://arxiv.org/abs/1611.01603.

Trischler, Adam, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman y Kaheer Suleman (2016). *NewsQA: A Machine Comprehension Dataset*. arXiv: 1611.09830 [cs.CL].

Xiong, Caiming, Victor Zhong y Richard Socher (2017). «Dynamic Coattention Networks For Question Answering». En: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=rJeKjwvclx.

Xiong, Wenhan, Jiawei Wu, Hong Wang, Vivek Kulkarni, Mo Yu, Xiaoxiao Guo, Shiyu Chang y William Yang Wang (2019). «TweetQA: A Social Media Focused Question Answering Dataset». En: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Yang, Liu y Lijing Song (2019). «Contextual Aware Joint Probability Model Towards Question Answering System». En: *CoRR* abs/1904.08109. arXiv: 1904.08109. URL: http://arxiv.org/abs/1904.08109.

Yang, Zhilin, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov y Christopher D. Manning (2018). «HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering». En: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Yu, Yang, Wei Zhang, Kazi Hasan, Mo Yu, Bing Xiang y Bowen Zhou (2016). «End-to-End Answer Chunk Extraction and Ranking for Reading Comprehension». En: arXiv: 1610.09996.