UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

ENTRENAMIENTO Y EVALUACIÓN DE MODELOS PEQUEÑOS DE LENGUAJE NATURAL BASADO EN MÉTODOS DE AUTOATENCIÓN

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

SEBASTIÁN ALEJANDRO DONOSO BUSTOS

PROFESOR GUÍA:
JORGE PÉREZ ROJAS

MIEMBROS DE LA COMISIÓN:
IVÁN SIPIRAN MENDOZA
DANIEL PEROVICH GEROSA

SANTIAGO DE CHILE
2021

RESUMEN

Actualmente Inteligencia Artificial es una de las áreas más populares en el uso de nuevas tecnologías. Dentro de este campo, se integran las redes de aprendizaje profundo que al ser utilizadas para el procesamiento del lenguajes natural, se ha promovido el desarrollo de distintas herramientas como la traducción automática entre idiomas, chatbot, asistentes virtuales entre otros. El avance en este tipo de tecnologías ha sido impulsado gracias a las grandes cantidades de datos que existen actualmente junto con el aumento del poder computacional. Sin embargo hoy en día solo algunas instituciones con abundantes recursos económicos han monopolizado la utilización y el desarrollo de estas tecnologías. Si bien estas mismas instituciones han realizado esfuerzos por democratizar el acceso a las redes de aprendizaje profundo dejándolas a libre disposición, el problema es que estas requieren grandes cantidades de poder computacional para poder utilizarlas. Junto con lo anterior este tipo de tecnologías se desarrolla principalmente en el idioma inglés, lo que aumenta las dificultades para usar estas metodologías en contextos particulares de países como Chile.

Con el objetivo de aumentar la disponibilidad de modelos de aprendizaje profundo en español, en esta memoria presentamos la evaluación de 7 modelos tipo ALBERT en las tareas de GLUES para determinar su comprensión del lenguaje. Además, presentamos un nuevo modelo de menor tamaño llamado DistilBETO, el cual entrenamos usando la técnica de destilación desde un modelo tipo BERT.

El mejor resultado de las evaluaciones en GLUES lo obtuvo s-ALBERT base con un promedio de 77.03 % que es un poco menor al mejor resultado (BETO cased 79.46 %), los resultados de los demás modelos estuvieron relativamente cerca a excepción de s-ALBERT tiny que solo logró 68 % de promedio en GLUES. DistilBETO logró mantener el 94 % del desempeño de BETO unacesd con un tamaño 40 % menor y 50 % más rápido en hardware. Creemos que DistilBETO es la mejor opción a usar si se necesita un modelo liviano manteniendo los buenos resultados. Se esperaba que los modelos ALBERT obtuvieran mejores resultados pero creemos que esto se puede lograr realizando una mejor búsqueda de *hiperparámetros*.

a Catalina,
sin ti no podría haber realizado este trabajo.

Agradecimientos

A mi papá y mamá, por su amor y apoyo incondicional durante toda la carrera.

A mi hermana, por creer siempre en mi.

A Catalina, por ayudarme siempre en los momentos más difíciles.

A mi familia, por ayudarme siempre que lo necesité.

A mis amigos y amigas por tener siempre palabras de ánimo.

A mis amigos de la universidad por ayudarme durante la carrera.

A Jorge Pérez, por sus correcciones y buena disposición ante cualquier duda.

A José Cañete, por su ayuda y buena disposición desde el comienzo de este trabajo.

A Vijay, por darme todas las facilidades necesarias para realizar este trabajo.

Y por último al 42, por siempre tener un espacio de confianza y amistad durante este proceso.

Tabla de Contenido

1. Introducción **1**

2. Antecedentes **3**

2.1. Marco teórico . . . 3

2.1.1. Aprendizaje de máquina . . . 3

Hiperparámetros . . . 4

Step . . . 4

Batch size . . . 4

Epoca . . . 4

Acumulación de gradiente . . . 4

Learning rate . . . 4

Warm up . . . 4

Softmax . . . 5

Representación de palabras para modelos de PLN . . . 5

Conjunto de Datos . . . 5

2.1.2. Métricas de evaluación . . . 5

2.2. Estado del arte . . . 6

2.2.1. AutoAtención . . . 6

2.2.2. Pre entrenamiento y entrenamiento específico. . . . 8

2.2.3. BERT: Bidirectional Encoder Representations from Transformers . . 9

2.2.4. ALBERT: A Lite BERT . . . 10

2.2.5. DistilBERT . . . 11

2.2.6. GLUES . . . 12

Natural Language Inference: XNLI . . . 12

Paraphrasing: PAWS-X . . . 12

Named Entity Recognition: NER . . . 12

Part-of-Speech Tagging: POS . . . 13

Document Classification: MLDoc . . . 13

Question Answering: MLQA, TAR and XQuAD . . . 13

3. Problema **14**

3.1. Objetivos . . . 16

3.1.1. Objetivo General . . . 16

3.1.2. Objetivos Específicos . . . 16

4. Solución **17**

4.1. Evaluación de modelos s-ALBERT . . . 17

4.2. Pre entrenamiento y evaluación de DistilBETO . . . 18

4.3. Evaluación en Hardware de común acceso 19

5. Resultados y análisis **20**
5.1. Evaluación en GLUES 20
Tiempo de entrenamiento específico 20
Evaluación en GLUES 21
DistilBETO 23
5.2. Rendimiento en Hardware 24

6. Conclusiones **25**

Bibliografía **27**

Anexo A. **30**
A.1. Hiperparámetros 30
A.2. Tiempos de inferencia 33

Índice de Tablas

2.1. Métricas de evaluación 6
2.2. Número de parámetros en millones de los modelos ALBERT [12] 11
2.3. Ejemplo de etiquetas BIO-FORMAT en una oración. 13
3.1. Estimación de costos de entrenamiento en USD.(Tabla sacada de [19]) 15
3.2. Emisiones estimadas de CO_2 para el entrenamiento de modelos comunes PLN comparado con consumo familiar.(Tabla sacada de [19]) 15
5.1. Tiempo de entrenamiento en tareas de GLUES. 20
5.2. Comparación de modelos s-ALBERT y DistilBETO con los resultados reportados en [3]. Superíndice indica los resultados obtenidos por (a) Wu & Dredze [25], (b) Yang et al. [28]. Todos los resultados de BETO son de Cañete et al. [3] . . 21
5.3. Comparación de modelos s-ALBERT y DistilBETO con los resultados reportados en [3] en la tarea de QA. Los resultados corresponden a *F1* / emparejamiento exacto, cada encabezado de las columnas indica el conjunto de entrenamiento a la derecha y de prueba a la izquierda. Superíndice indica los resultados obtenidos por (c) Lewis et al. [13], (d) Carrino et al. [2] . Todos los resultados de BETO son de Cañete et al. [3] 22
5.4. Promedio de los resultados reportados en GLUES. 23
5.5. Promedio del tiempo de inferencia al predecir 100 ejemplos del conjunto de datos de NER. 24
A.1. *Hiperparámetros* para el entrenamiento en la tarea XNLI 30
A.2. *Hiperparámetros* para el entrenamiento en la tarea MLDoC 31
A.3. *Hiperparámetros* para el entrenamiento en la tarea PAWS-X 31
A.4. *Hiperparámetros* para el entrenamiento en la tarea POS 31
A.5. *Hiperparámetros* para el entrenamiento en la tarea NER 32
A.6. *Hiperparámetros* para el entrenamiento en la tarea QA-MLQA 32
A.7. *Hiperparámetros* para el entrenamiento en la tarea QA-TAR 32
A.8. Tiempo de inferencia de las 5 ejecuciones de los modelos en 100 ejemplos del conjunto de prueba de NER. 33

Índice de Ilustraciones

2.1. Diagrama de *encoder* del Transformer. 7
2.2. Visualización de atención por el modelo BETO, para la entrada *el perro corre en el parque en busca de la pelota*. Las imágenes (a) y (b) muestran atención de distintas cabezas/capas en los *token perro* y *busca* respectivamente. En la imagen (c) se visualiza múltiples atenciones en la misma capa de (b) para el *token corre*. Imagen generada con el programa realizado por Vig [22]. 8
2.3. Diagrama *"Masked Language Model"* BERT. 10
3.1. Número de parámetros de modelos. (Figura tomada de [17]) 14

Capítulo 1

Introducción

El Procesamiento del Lenguaje Natural (PLN desde ahora) a través de modelos de aprendizaje profundo ha ido en auge los últimos años. Sin embargo su desarrollo ha sido realizado solo por instituciones con grandes recursos tecnológicos como lo son Google, Facebook, Microsoft, entre otras. La utilización de redes de aprendizaje profundo, diseñadas especialmente para PLN por las grandes empresas mencionadas, han conseguido importantes resultados en tareas como la traducción entre idiomas, implementaciones en chatbot, reconocimiento de voz, análisis de sentimiento, corrección ortográfica, entre otros. Es por ello que la aplicación de estos nuevos estudios representa una gran oportunidad para generar avances no solo en el área de la computación, sino que también tienen el potencial de ser aprovechadas por diversas organizaciones de la sociedad civil que trabajan temáticas variadas.

Aún cuando se han realizado grandes avances en esta área, existen pocas aplicaciones directas en países en desarrollo, como es el caso de Chile. Esto se debe a que los estudios se llevan a cabo principalmente por grandes instituciones, las cuales cuentan con los recursos tecnológicos necesarios. De este modo, la investigación tiende a centralizarse en entidades que son parte de países con altos niveles de desarrollo de habla inglesa. En esta línea, uno de los problemas que se presentan en esta área de investigación está directamente relacionado con el idioma utilizado. Debido a la alta demanda de recursos para estos procesamientos, el español ha quedado relegado del plano de estas investigaciones, lo que ha limitado la posibilidad de hacer uso de estas metodologías para la aplicación de soluciones y entendimiento de problemas relativos a este contexto particular. Esta limitante repercute en un menor avance de la tecnología al servicio de las comunidades de habla hispana, lo que generalmente implica menos posibilidades de resolver problemas locales en conjunto con la innovación.

A partir de lo anterior en esta memoria evaluamos los modelos de aprendizaje profundo del tipo ALBERT [12] entrenados previamente en español por Cañete et al. [3]. Además, entrenamos un nuevo modelo más pequeño, usando el método de *"destilación"* [9] desde el primer modelo tipo BERT [5] de gran tamaño entrenado desde un conjunto de datos en español. Para realizar las evaluaciones utilizamos las tareas descritas en GLUES [3], y en la utilización de la técnica de *"destilación"* nos guiamos en el trabajo realizado por Sanh et al. [17]. Finalmente evaluamos el modelo pequeño en GLUES para comparar los resultados obtenidos por los modelos. Además, evaluamos el rendimiento de los modelos en Hardware de común acceso a fin de analizar el desempeño de estos según su tamaño. De este modo, a partir de los modelos integrados en esta memoria buscamos contribuir al desarrollo del PLN

en español y a la aplicación de estas tecnologías de forma más accesible.

Capítulo 2

Antecedentes

2.1. Marco teórico

A continuación presentamos algunos conceptos relevantes asociados al Aprendizaje Profundo y el Procesamiento del Lenguaje Natural, los cuales están a la base del desarrollo y entendimiento del presente trabajo.

2.1.1. Aprendizaje de máquina

Como mencionan Goodfellow, Bengio & Courville [8], los algoritmos de aprendizaje de máquina se definen por su capacidad de aprender de los datos para resolver una tarea específica. Estos tipos de algoritmos permiten abordar problemas que son altamente complejos de resolver con programas fijos, diseñados y escritos por seres humanos.

Un tipo específico de aprendizaje de máquina es el aprendizaje profundo. De manera general se puede decir que los algoritmos de aprendizaje profundo están formados por capas con parámetros conectados por funciones no lineales. Estos tienen una fase de entrenamiento en la que dado un número fijo de ejemplos $X = x_1, x_2, ..., x_n$ con sus etiquetas correspondientes $Y = y_1, y_2, ..., y_n$ se desea obtener la función $f(x) = y$, en consecuencia, el algoritmo busca obtener una función f' tal que $f' \approx f$. Se espera entonces que dado $f'(x; \theta) = \hat{y}$, con θ los parámetros que definen a la función f' se cumpla que $\hat{y} = y$, de ahí que el entrenamiento tiene como propósito encontrar los θ que cumplan con lo anterior. Para lograr esto se utiliza una función de pérdida o función objetivo $L(\hat{y}, y)$ que indica que tan lejos esta f' del resultado esperado. Con esto el algoritmo calcula el gradientes de los parámetros y los ajusta usando descenso estocástico de gradiente para minimizar L, el cual calcula una estimación del error de f' sobre los ejemplos. En la práctica se toma al azar un grupo de ejemplos del conjunto de datos de entrenamiento llamado *batch* y se calcula el promedio de la función de pérdida sobre estos ejemplos. Luego se obtiene el gradiente de los parámetros para realizar un paso de descenso de gradiente y actualizar θ a los valores que minimizan el error. Esto se repite con nuevos *batch* hasta que se hayan utilizados todos los datos de entrenamiento generalmente más de una vez.

Al finalizar el entrenamiento se espera que el algoritmo sea capaz de generalizar lo aprendido a casos que no ha sido expuesto con anterioridad. Los algoritmos de aprendizaje profundo, denominados modelos o redes de aprendizaje profundo, se diferencian entre sí según la tarea

que buscan resolver y el método en particular que utilizan para realizarla. En esta memoria nos centramos en los modelos del tipo BERT para los cuales se realizará una explicación más adelante.

Hiperparámetros

Los *hiperparámetros* gobiernan la capacidad de la red de aprendizaje profundo y se ajustan manualmente utilizando el conjunto de datos de evaluación, es preciso señalar que a continuación solo se definirán los más relevantes para este trabajo.

Step

Al realizar un descenso de gradiente en el entrenamiento de un modelo, se dice que se hizo un *step* .

Batch size

El *batch size* determina el número de ejemplos en paralelo que se entregan al modelo. En general este se elige según la cantidad total de datos y la capacidad de la memoria del hardware a utilizar.

Epoca

Al recorrer todos los ejemplos del conjunto de datos de entrenamiento por un modelo se dice que este realizó una *época*.

Acumulación de gradiente

Debido a las limitaciones de memoria que se encuentran asociadas al entrenamiento de modelos de aprendizaje profundo se utiliza la *acumulación de gradiente* para simular un *batch size* de mayor tamaño. De esta manera se realiza un *step* solo cuando han pasado por la red el número de *batch* que determina la acumulación de gradiente.

Learning rate

Define el factor de cambio de los parámetros en el descenso de gradiente.

Warm up

Para evitar que al comienzo del entrenamiento los ejemplos que se pasan a través de la red afecten demasiado el cambio de los parámetros, se utiliza *warm up*; de esta manera se realiza un incremento lineal del *learning rate* desde una fracción de este hasta su valor original.

Softmax

Cuando un modelo realiza una clasificación genera una distribución de probabilidades sobre las posibles clases; para esto generalmente se utiliza la función *softmax*. Como se muestra en 2.1 esta función toma los valores reales z_j que asigna un modelo a las diferentes clases i y los distribuye entre 0 y 1.

$$p_i = \frac{exp(z_j)}{\sum_j exp(z_j)} \tag{2.1}$$

Representación de palabras para modelos de PLN

PLN corresponde al área que estudia la representación del lenguaje natural humano mediante algoritmos [6]. Los computadores utilizan lenguajes de programación diseñados especialmente para que no exista ambigüedad al ser leídos por un programa, mientras que los lenguajes que utilizamos los humanos para comunicarnos tienden a ser poco eficientes y ambiguos, lo que dificulta poder definirlos de manera formal [8]. Como menciona Goldberg [7] el lenguaje humano es altamente ambiguo y variable; a pesar de que las personas somos los principales usuarios del lenguaje, no somos tan buenos para entenderlo y describirlo formalmente.

Al querer aplicar algoritmos de aprendizaje profundo en el lenguaje natural es necesario tener una forma de representar las palabras para que los modelos sean capaces de procesarlas. Los modelos trabajados en esta memoria utilizan técnicas de *"Text Tokenization"*. En esta el *tokenizer* es el encargado de convertir las secuencias de texto en una entrada para el modelo separando cada secuencia en palabras o subpalabras llamadas *tokens*, luego se transforman en ids correspondientes al vocabulario del modelo. Los modelos utilizados para PLN tienen un vocabulario finito, que se define aplicando un algoritmo al corpus de texto con el que se realiza el pre entrenamiento.

Conjunto de Datos

Para entrenar un modelo de aprendizaje profundo en una tarea se necesita un conjunto de datos, los cuales generalmente se dividen en tres partes: entrenamiento, validación y prueba. Si bien las proporciones de estos conjuntos pueden variar se recomienda que el conjunto de entrenamiento sea mayor a los otros dos. El conjunto de validación se utiliza para probar distintos *hiperparámetros* con el objetivo de buscar la mejor configuración para el modelo. Mientras que el conjunto de prueba se usa solamente para evaluar la capacidad del modelo.

2.1.2. Métricas de evaluación

Para evaluar la capacidad de un modelo de aprendizaje profundo es necesaria una forma cuantitativa de medirla. Para ello generalmente la medición es específica para la tarea realizada por el modelo [8]. A continuación, en la tabla 2.1 se definen las métricas que utilizamos para la evaluación de los modelos de este trabajo, las siglas utilizadas se describen a continuación:

- VP número de ejemplos donde el modelo indicó que eran positivos y acertó.

- VN número de ejemplos que el modelo indicó que eran negativos y acertó.
- PT número total de ejemplo predichos por el modelo.
- FP número de ejemplos donde el modelo indicó que eran positivos y se equivocó.
- FN número de ejemplos que el modelo indicó que eran negativos y se equivocó.

Tabla 2.1: Métricas de evaluación

accuracy	$\frac{VP+VN}{PT}$
precision	$\frac{VP}{VP+FP}$
recall	$\frac{VP}{VP+FN}$
F1	$\frac{2}{recall^{-1}+precision^{-1}}$
PT	$(VP+VN+FP+FN)$

2.2. Estado del arte

Actualmente los modelos de PLN que mejores resultados entregan en la práctica, están basados en las arquitecturas del tipo Transformer [21]. Los grandes resultados que se han logrado alcanzar en las diferentes tareas de PLN, se deben a que estos modelos logran aprender sobre la relación sintáctica y semántica de las palabras entre sí, dado que al procesar un *token* son capaces de mirar todos los *tokens* que conforman la oración [3], a diferencia de otras arquitecturas donde los modelos solo pueden ver los *tokens* anteriores al que está siendo procesado.

Gracias a que estos modelos pueden ser entrenados con datos no etiquetados, es que se ha logrado hacer uso provechosamente de las grandes cantidades de texto que existen disponibles en la web, como es el caso de miles de libros, páginas como Wikipedia, entre otros.

2.2.1. AutoAtención

Los modelos de PLN que se usaron en este trabajo son conocidos como *"modelos de representación del lenguaje"*; estos se caracterizan por recibir una secuencia de texto y entregar una distribución de probabilidades sobre un vocabulario definido. Todas las arquitecturas que se presentan en este trabajo utilizan solo el *encoder* del Transformer [21] que consta de dos partes, capa de multiple-autoatención y red *feed-forward* como se ejemplifica en la figura 2.1. La autoatención se aplica a cada *token* de la oración y tiene como objetivo obtener vectores que contengan la información de la función de los *tokens* en la sentencia. Para esto al aplicar autoatención en un *token* se consideran todos los *tokens* de la oración, poniendo mayor énfasis en aquellos que entregan mejor información para generar el vector del *token* referido. Gracias a lo anterior los modelos que utilizan autoatención tienen la ventaja de poder ver toda la entrada para cada paso que realizan.

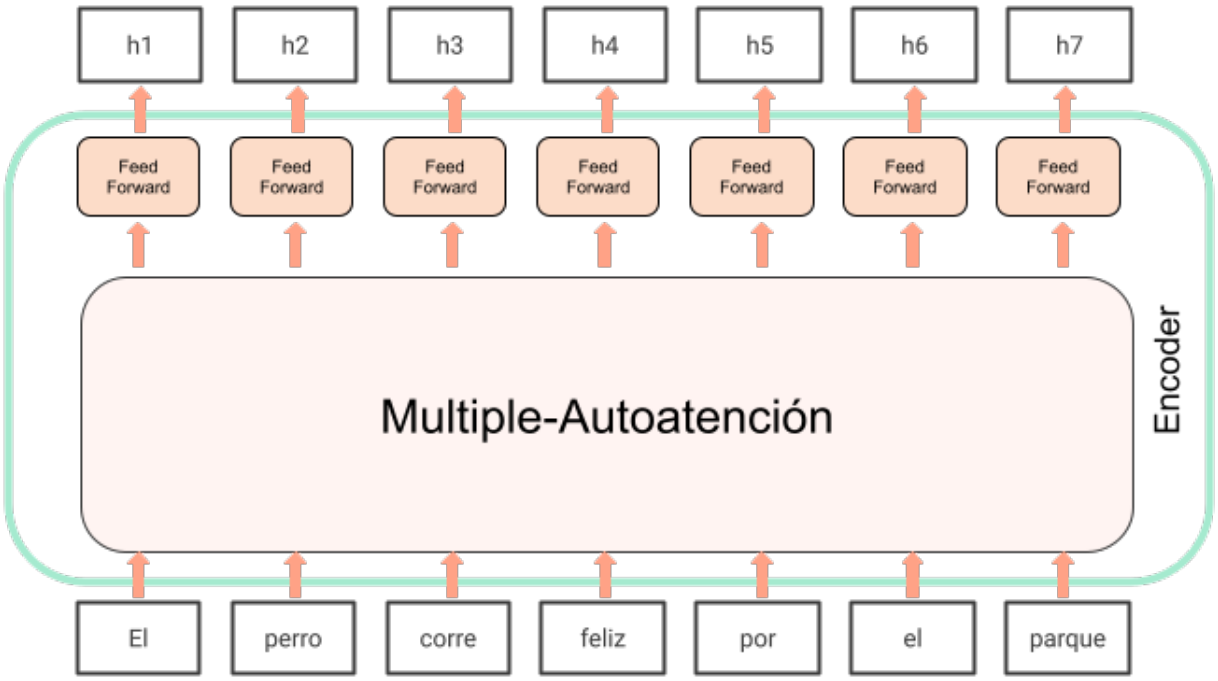

Figura 2.1: Diagrama de *encoder* del Transformer.

Por ejemplo, al considerar una oración W con n palabras la capa de autoatención de un modelo genera los vectores q_i: query; k_i: key; v_i: value para cada *token* $w_1, w_2, ..., w_n$ de la oración. Al aplicar autoatención en un w_i particular se obtiene el vector h_i, para esto se realiza el producto punto de q_i con los $k_1, k_2, ..., k_n$ vectores asociados a los *tokens* de la oración; luego los resultados se dividen por la raíz cuadrada del número de dimensión del vector k y se aplica la función *softmax*. A partir de lo anterior, se generan los valores $s_1, s_2, ..., s_n$ para cada *token*, los que se pueden entender como un puntaje que representa la importancia que tiene cada palabra en relación con w_i. Finalmente, se multiplica cada s_j con el vector v_j respectivo a cada *token*, los vectores resultantes se suman y este resultado se pasa a través de la red feed forward de donde se obtiene el h_i. Cuando se habla de multi atención se realiza lo mismo pero en cada *token* se aplican múltiples autoatenciones con las denominadas *cabezas de atención*. En la figura 2.2, se muestran distintos ejemplos donde la autoatención es representada como líneas que conectan el *token* visitado por esta con los *tokens* referenciados por la apliación de autoatención [22]. Los colores de la parte superior representan las distintas cabezas de atención mientras que la intensidad de los colores en las líneas conectoras representan el puntaje asignado a los *tokens* por la atención.

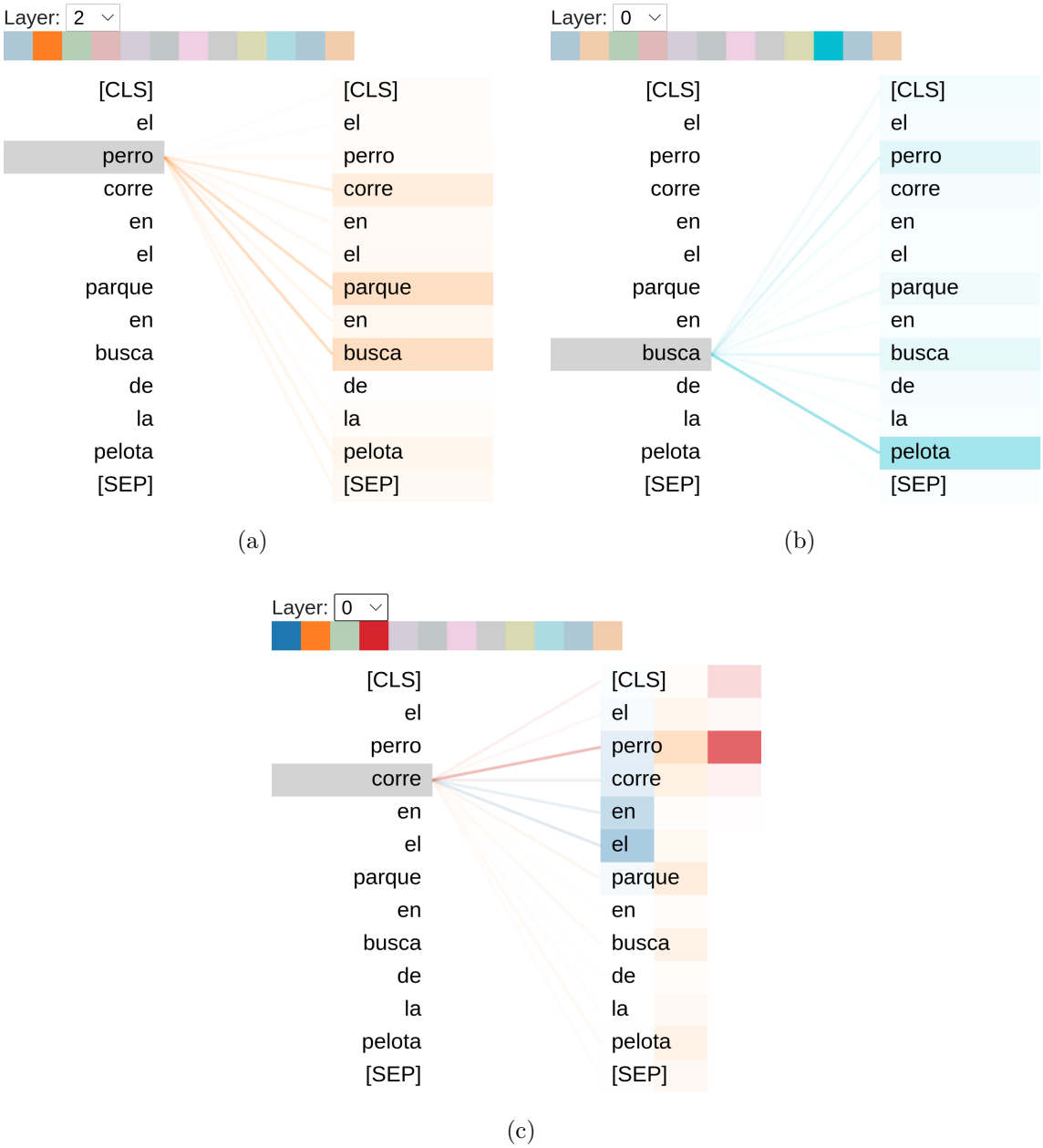

Figura 2.2: Visualización de atención por el modelo BETO, para la entrada *el perro corre en el parque en busca de la pelota*. Las imágenes (a) y (b) muestran atención de distintas cabezas/capas en los *token perro* y *busca* respectivamente. En la imagen (c) se visualiza múltiples atenciones en la misma capa de (b) para el *token corre*. Imagen generada con el programa realizado por Vig [22].

2.2.2. Pre entrenamiento y entrenamiento específico.

De acuerdo a lo planteado por Cañete et al. [3], una de las principales características de los modelos actuales de PLN, es que se trabaja llevando a cabo un pre entrenamiento aprovechando las grandes cantidades de textos disponibles. Dado que es necesario que los modelos tengan una función objetivo para poder ser entrenados, se utilizan tareas que definen

el resultado esperado de un ejemplo desde el mismo texto que se usa como entrada. Algunas de las tareas más comunes usadas en estos casos consisten en predecir las palabras que faltan en una oración. Por ejemplo si tomamos la oración *"María corre todas las mañanas por el parque"* y reemplazamos la palabra *"correr"* por el *token* [**predecir**], generando de esta manera la entrada *"María **[predecir]** todas las mañanas por el parque"* y la etiqueta *"correr"*, de ahí que podemos definir la tarea del modelo con el objetivo de predecir las palabras que deben ir en las posiciones que aparece el *token* [**predecir**]. Gracias a estas tareas previas los modelos como BERT [5] logran aprender la estructura del lenguaje a un nivel más abstracto sin aún realizar una tarea en específico. Finalmente luego del pre entrenamiento estos tipos de modelos son entrenados con conjuntos de datos etiquetados propios de una tarea específica como es el caso de traducción automática.

La evaluación del desempeño de estos modelos se realiza mediante las tareas definidas por "Evaluación de comprensión del lenguaje general", desde ahora GLUE por sus siglas en inglés [23]. De acuerdo a lo planteado por Wang y otros autores [23] esta evaluación consiste de nueve tareas de comprensión del lenguaje en oraciones, construida desde el conjunto de datos establecido por la misma evaluación. Con esto se busca poder estandarizar la evaluación de modelos de PLN, dado que el conjunto de datos está diseñado para lograr analizar y evaluar el rendimiento del modelo en una amplia gama de fenómenos lingüísticos que se encuentran en el lenguaje natural. Siguiendo la finalidad del conjunto de evaluaciones anteriores, Cañete y otros autores [3] proponen la "Evaluación de comprensión del lenguaje general en español" desde ahora GLUES.

2.2.3. BERT: Bidirectional Encoder Representations from Transformers

En octubre del 2018 Devlin et al. [5] presentan BERT, una de las arquitecturas de PLN más importantes, ya que logró el estado del arte en 11 tareas y un $7,7$ pp de mejora absoluta en GLUE [5], la cual está formada por múltiples encoders apilados en capas y presenta dos ideas claves para el pre entrenamiento y utilización de texto no etiquetado. La primera refiere al uso de la función objetivo llamada *"Masked Language Model"* (MLM), la que consiste en reemplazar aleatoriamente algunos de los *tokens* de la oración de entrada por el *token* [**mask**], esperando que el modelo prediga los *tokens* originales para así reconstruir la oración como se muestra en la figura 2.3. Gracias a la utilización de esta función objetivo en el pre entrenamiento, este tipo de modelo logra aprender buenas representaciones del lenguaje, ya que para realizar una predicción correcta solo puede utilizar el contexto de la oración.

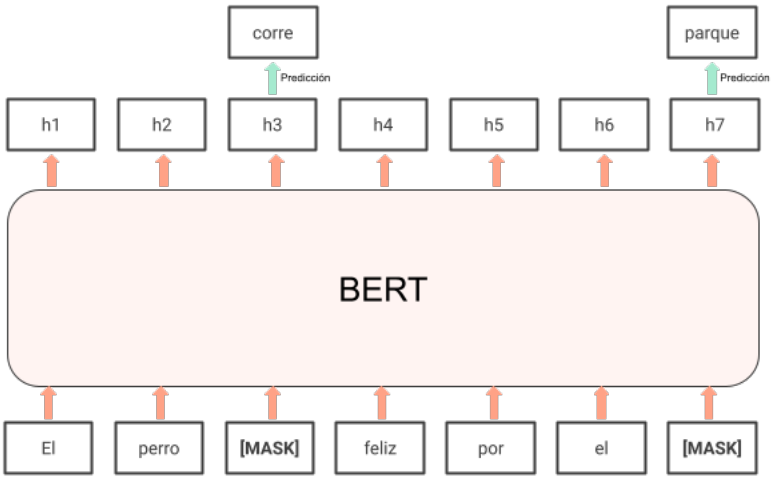

Figura 2.3: Diagrama *"Masked Language Model"* BERT.

Por otro lado, Devlin et al. [5] proponen la tarea *"Next Sentence Prediction"*(NSP) donde el objetivo es clasificar si es que dadas dos oraciones la segunda precede o no a la primera. De esta manera, se busca que el modelo sea capaz de identificar la coherencia y temática entre oraciones. Si bien se habla de entrenamiento con texto sin etiquetas, se puede decir que para las tareas mencionadas las etiquetas son generadas desde los ejemplos que se usan, ya que en MLM los *tokens* cambiados funcionan como tal, mientras que en NSP basta con tomar oraciones contiguas de un texto para tener casos positivos y cambiar la segunda oración para tener casos negativos. Los autores de BERT presentan dos modelos según su tamaño, la versión base que tiene 110M de parámetros y la version large con 340M, siendo esta última la que alcanzó los mejores resultados, cada una fue entrenada durante 4 días en los servidores de Google.

En base a lo anterior BETO es presentado el 2020 por Cañete et al. [3], el cual es el primer modelo tipo BERT entrenado específicamente con un conjunto de datos completamente en español. A diferencia de BERT utiliza *Dynamic Masking* que consiste en duplicar un número de veces determinado cada ejemplo para que cada entrada del modelo tenga diferentes *token* con masking. El trabajo realizado por Cañete et al. [3] muestra mejores resultados para la mayoría de las tareas de GLUES en comparación a *"multiligual BERT"* [5]; este último es un modelo de múltiples idiomas ya que está entrenado simultáneamente en 104 idiomas diferentes.

2.2.4. ALBERT: A Lite BERT

A pesar de que en general mientras más grande es un modelo es posible obtener mejores resultados, el trabajo propuesto por Lan et al. [12] presenta una arquitectura que reduce el número de parámetros y aumenta la velocidad de entrenamiento respecto a BERT. Ya que de esta manera se pueden superar las limitaciones en tiempo y hardware encontradas al momento de buscar mejores resultados solo aumentando el tamaño de los modelos. Este nuevo modelo llamado ALBERT presentan tres cambios claves, primero realiza una factorización de los *embeddings* en matrices más pequeñas, lo que disminuye la cantidad de parámetros del modelo. En segundo lugar, a diferencia de BERT se comparten los parámetros entre los *encoders* de distintas capas, visto de otra forma el mismo *encoder* se repite varias veces. Y por último, se

cambia la tarea *"Next Sentence Prediction"* por *"Sentence-order Prediction"*(SOP). El problema con NSP es que a pesar de tener como objetivo aprender distinciones de coherencia y temática entre oraciones, termina enfocándose solamente en la última [12]. En cambio, SOP apunta a la coherencia entre oraciones ya que a diferencia NSP, el modelo siempre recibe dos oraciones consecutivas de un texto por lo tanto del mismo tema, pero se busca que identifique si estas se encuentran en orden o han sido intercambiadas de posición. El trabajo original en el que se presenta ALBERT entrega cuatro modelos de distintos tamaños como se muestra en la tabla 2.2, para este trabajo agregamos la versión *"tiny"* [26] pre entrenada por Cañete et al. [3].

Tabla 2.2: Número de parámetros en millones de los modelos ALBERT [12]

Modelo	**N° Parámetros**
tiny	5M
base	12M
large	18M
xlarge	60M
xxlarge	235M

2.2.5. DistilBERT

En busca de modelos que requieran menos recursos pero que mantengan los buenos resultados de las redes de gran tamaño es que Sanh et al. [17] presentan DistilBERT; este logró en comparación a BERT base reducir el tamaño en un 40 %, manteniendo el 97 % de la capacidad de comprensión del lenguaje siendo un 60 % más rápido.

DistilBERT sigue la arquitectura de BERT base pero con algunas modificaciones. En primer lugar, removieron el *"token-type embeddings"* que sirve para representar las oraciones distintas en la tarea NSP y disminuyeron el número de capas a la mitad, con esto se logra una disminución desde los 110M de parámetros del modelo original a 66M. Por otro lado, para disminuir los recursos necesarios en el pre entrenamiento de DistilBERT utilizaron la técnica de *"destilación"* [9]. Esta metodología plantea que en vez de entrenar modelos más pequeños de manera tradicional, se aprovechen las capacidades de generalización de los modelos de mayor tamaño, transfiriendo este aprendizaje a un modelo más reducido. De esta manera, es posible entrenar modelos que requieren menos recursos mediante la *"destilación del conocimiento"* de un modelo ya pre entrenado, obteniendo resultados cercanos a los modelos grandes y mejor a los que se obtienen de manera clásica.

Por ejemplo si tomamos la oración *"para sentirte mejor deberías **[MASK]** más seguido"* y la pasamos como entredata a BETO [3], vemos que los primeros 5 *tokens* que se obtienen para reemplazar [**MASK**] son: *"venir"*, *"hacerlo"*, *"hacer"*, *"correr"* y *"comer"*. Dado que todas las palabras entregadas por BETO tienen sentido para completar la oración se infiere que no solamente el *token* con mayor probabilidad puede ser correcto sino que existen otros igualmente validos según el contexto. De esta manera, aprovechando la distribución de probabilidades sobre el vocabulario que entregan estos modelos al realizar la predicción de un *token* es que resulta posible "tranferirle" este conocimiento a otro modelo en su pre entrenamiento.

Siguiendo la idea de que el profesor es el encargado de enseñar a un estudiante es que al

utilizar *"destilación"* se denomina *profesor* al modelo pre entrenado y *estudiante* al que se desea entrenar. Para el pre entrenamiento Sanh et al. [17] utilizaron la función de pérdida 2.2 denominada *"distillation loss"*

$$L_{ce} = \sum_i t_i * log(s_i) \tag{2.2}$$

en la que t_i es la probabilidad estimada por el *profesor* y s_i por el *estudiante*. De esta manera, se calcula la diferencia a la que esta el *estudiante* del *profesor* al momento de entregar una predicción. Además, como función de salida se usó 2.3 *"softmax-temperature"* [9]

$$p_i = \frac{exp(z_j/T)}{\sum_j exp(z_j/T)} \tag{2.3}$$

en la cual T controla la suavidad de la distribución entregada y z_j corresponde al puntaje que le asigna el modelo a la clase i. Además de L_{ce} se agregan dos funciones de pérdida más, *"Masked Language Model"* L_{mlm} que corresponde a la utilizada en BERT y *"cosine embedding loss"* L_{cos} la que ayuda a alinear los vectores del *estudiante* y *profesor*.

2.2.6. GLUES

Con el fin de tener una mayor comprensión de las tareas definidas en GLUES que utilizamos en esta memoria para evaluar los modelos, a continuación se realiza una descripción de estas basada en el trabajo de Cañete et al. [3].

Natural Language Inference: XNLI

Dadas dos oraciones, premisa e hipótesis, la tarea consiste en predecir si la premisa es acorde, contraria o neutra respecto a la hipótesis. Para el entrenamiento se utiliza la traducción del conjunto de datos de MNLI [24], mientras que para evaluación y prueba se utilizan los datos de Conneau et al. [4]. Esta tarea se evalúa mediante *accuracy*.

Paraphrasing: PAWS-X

La tarea consiste en predecir si dos oraciones son semánticamente equivalentes o no. El conjunto de datos es el propuesto por Yang et al. [27] y se evalúa con simple *accuracy*.

Named Entity Recognition: NER

El problema consiste en determinar si cada palabra en una oración corresponde a una entidad o no. Las entidades nombradas (*named entities*) son frases que contienen nombres de personas, organizaciones, lugares, tiempo y cantidades. Para este caso solo se utilizaron las primeras tres (personas, organizaciones, lugares) y hay una cuarta categoría para misceláneos de entidades. El conjunto de datos usado en esta tarea utiliza el etiquetado *"BIO format"*[16], que marca la diferencia entre las partes de una entidad inicio, dentro o fuera de esta como se muestra en la tabla 2.3.

Tabla 2.3: Ejemplo de etiquetas BIO-FORMAT en una oración.

Daniel	B-PER
,	O
es	O
periodista	O
en	O
Argentina	B-LOC
,	O
jugo	O
ayer	O
con	O
Del	B-PER
Bosque	I-PER

Para esta tarea *precision* corresponde al porcentaje de entidades nombradas encontradas correctas y *recall* es el porcentaje de entidades nombradas presentes en el corpus que fueron encontradas. Los datos que se utilizan son los propuestos por Tjong Kim Sang [20]; la evaluación es con *F1*.

Part-of-Speech Tagging: POS

Es la tarea de etiquetar una palabra en un texto como sustantivo, adjetivo, verbo, preposición o conjunción. Se utiliza el conjunto de datos en español de *Universal Dependencies (v1.4) TreeBank* [14] y la evaluación es el *accuracy* de las etiquetas predichas.

Document Classification: MLDoc

Esta tarea consiste en clasificar documentos en cuatro categorías *CCAT (Corporate/Industrial), ECAT(Economics), GCAT(Goverment/Social), y MCAT(Markets)*. Se utiliza la parte más grande del conjunto de datos en español de MLDoc [18] y se evalúa con *accuracy*.

Question Answering: MLQA, TAR and XQuAD

Dado un contexto y una pregunta la tarea consiste en encontrar la secuencia contigua de letras dentro del contexto que responde a la pregunta. Se utilizan distintos conjuntos de datos, los cuales son versiones traducidas de SQuAD V1.1 [15]. Estos son MLQA [13] y TAR [2] para el entrenamiento, mientras que para la evaluación y prueba se usan MLQA y XQuAD [1]. Se evalúa utilizando el promedio de dos métricas sobre todas las preguntas, porcentaje de respuestas que son un emparejamiento exacto y *F1*, donde las respuestas son tratadas como bolsa de palabras.

Capítulo 3

Problema

La gran demanda de recursos de hardware que se requieren al trabajar con los actuales modelos de aprendizaje profundo en PLN conlleva que principalmente instituciones con grandes cantidades de recursos económicos tengan acceso al desarrollo y utilización de este tipo de tecnología. Como indica Strubell et al. [19] es necesario equiparar el acceso de los/as investigadores/as a los recursos computacionales, puesto que actualmente el desarrollo ocurre sucesivamente por los mismos grupos de investigadores/as, dejando fuera ideas que no estén alineadas con ellos.

Considerando que el tamaño de los nuevos modelos desarrollados sigue aumentando con el tiempo, no tiene mucha utilidad liberarlos a la comunidad si para poder utilizarlos se necesitan recursos al cual solo grandes empresas pueden acceder. En la figura 3.1 se muestra el número de parámetros de algunos modelos, acá se puede apreciar la magnitud de este tipo de tecnología y quiénes son los que tienen la capacidad para desarrollarlas. Además, aparece DistilBERT [17], modelo en el que basamos parte de la solución presentada en esta memoria.

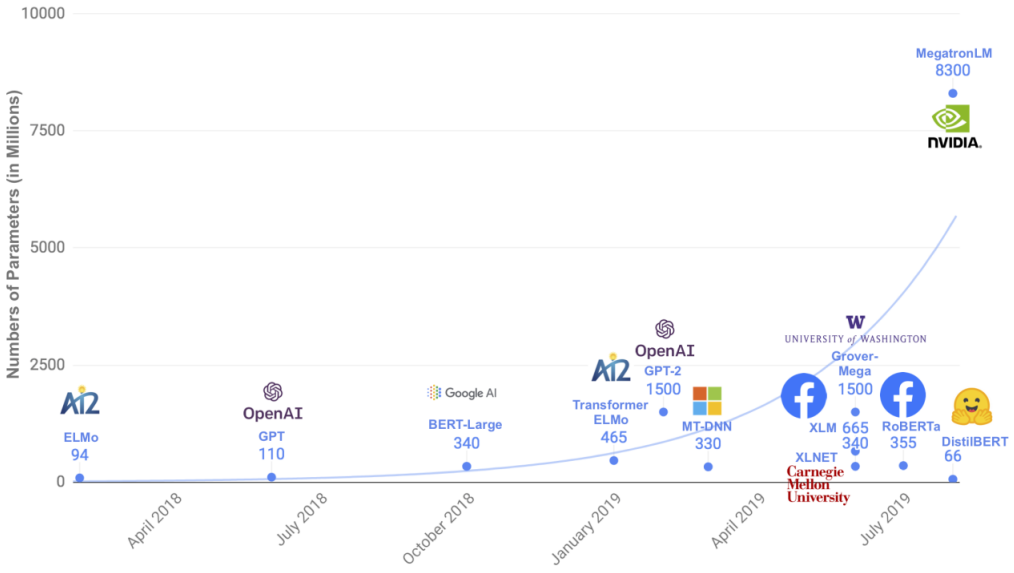

Figura 3.1: Número de parámetros de modelos. (Figura tomada de [17])

Para tener una idea de los costos monetarios que requiere el desarrollo de este tipo de investigación en la tabla 3.1 se muestra un estimado de cuánto dinero se necesita para entrenar distintos modelos en la nube, por ejemplo para entrenar BERT en su versión base el costo puede llegar hasta 6912 USD [19].

Tabla 3.1: Estimación de costos de entrenamiento en USD.(Tabla sacada de [19])

Modelos	**Costo Nube en USD**
Transfomer base	$41-$140
Transformer big	$289-$981
ELMo	$433-$1472
BERT base	$3751-$12,571
BERT base (hardware diferente)	$2074-$6912
NAS	$942,973-$3,201,722
GPT-2	$12,902-$43,008

Junto con lo anterior el avance tecnológico de las redes de aprendizaje profundo ha traído consigo un gran impacto medio ambiental, como muestra el estudio realizado por Strubell et al. [19] en la tabla 3.2 la cantidad CO_2 generadas por el entrenamiento de modelos de PLN puede llegar a sobrepasar las emisiones producidas por un automóvil en su vida útil.

Tabla 3.2: Emisiones estimadas de CO_2 para el entrenamiento de modelos comunes PLN comparado con consumo familiar.(Tabla sacada de [19])

Consumo Cotidiano	CO_{2e} (lbs)
Viaje Aereo, 1 pasajero, NY $\longleftrightarrow$ SF	1984
Persona, 1 año	11,023
Persona de EEUU, 1 año	36,156
Vida útil automóvil combustible incl.	126
Entrenamiento de modelos PLN	
Pipeline de PLN (Parsing, SRL)	39
con tuning y experimentación	78,468
Transformer big	192
con neural architecture search	626,155

Además del costo monetario y medioambiental, el desarrollo de estas tecnologías se enfocan principalmente en el idioma inglés, es por esto que se han realizado esfuerzos, por grupos más pequeños y con menos recursos, para intentar aprovechar estas tecnologías en contextos locales. Sin ir más lejos BETO [3] es un ejemplo del trabajo realizado para incluir al idioma español; este evidencia que si bien se puede realizar buenos avances en PLN enfocados en el contexto de países hispano hablantes, estos modelos requieren de muchos recursos. Por

ejemplo, BETO tiene 110 millones de parámetros, y los modelos tipo ALBERT, que se pueden entender como una mejora de BERT, en cuanto a desempeño y disminución en parámetros [12], tienen una cantidad de parámetros en el rango de 5M a 235M.

Continuando con el trabajo realizado por Cañete et al. [3] y con el objetivo de poder democratizar el acceso a este tipo de tecnología es que en esta memoria evaluamos los modelos tipo ALBERT denominados s-ALBERT y utilizamos la técnica de *"destilación"* para entrenar una versión más pequeña de BETO, denominado DistilBETO. Se espera que tanto los modelos ALBERT como DistilBETO tengan resultados cercanos a los reportados por BETO en GLUES.

3.1. Objetivos

3.1.1. Objetivo General

Evaluar modelos del tipo ALBERT entrenados previamente en español usando diversas tareas así como entrenar un nuevo modelo más pequeño en términos de parámetros, usando el método de *"destilación"* desde un modelo BERT de gran tamaño.

3.1.2. Objetivos Específicos

- Evaluar los modelos tipo ALBERT -previamente entrenados- en las tareas de GLUES.
- Entrenar un modelo de tamaño reducido mediante la técnica de *"destilación"* desde un modelo de tipo BERT considerablemente más grande.
- Evaluar el modelo pequeños entrenado con *"destilación"* en las mismas tareas que fueron evaluados los modelos de mayor tamaño.
- Evaluar el desempeño de modelos en hardware de común acceso para la población chilena.
- Comparar los resultados obtenidos en las evaluaciones de los modelos de gran y menor tamaño.

Capítulo 4

Solución

Para desarrollar la solución que se presenta a continuación, utilizamos como guía el trabajo de entrenamiento y evaluación realizado por Cañete et al. [3], por lo que se hará uso de los modelos BETO *uncased* y s-ALBERT entrenados en ese trabajo. El código que usamos se encuentra en un repositorio[1] de acceso público; todo el trabajo fue realizado con el lenguaje de programación Python 3.8 y los entrenamientos se realizaron en una GPU "NVIDIA RTX 3090".

4.1. Evaluación de modelos s-ALBERT

Si bien son cinco los modelos s-ALBERT según tamaño que entrenamos en las tareas de GLUES, para el tamaño base evaluamos tres iguales en configuración pero pre entrenados con distintos *hiperparámetros*. Además, realizamos 42 entrenamientos específicos; para los hechos en las tareas XNLI, QA, POS, PAWS-X y NER usamos los recursos publicados por la biblioteca Transformers[2] [17] que contiene código de libre acceso para usar varios modelos basados en la arquitectura Transformer [21] y realizar tareas del estado del arte en PLN. Para MLDoc[3] usamos como base el código implementado para evaluar BETO.

Primero transformamos los modelos desde la biblioteca TensorFlow a Pytorch ya que esta última entrega más flexibilidad para realizar modificaciones de los modelos.

Para cada entrenamiento de un modelo en una tarea específica se deben elegir *hiperparámetros*; es ideal probar distintos valores de estos ya que de esta manera se pueden obtener mejores resultados. Sin embargo, debido a que esto requiere tratar con cada modelo en cada una de las tareas decidimos que no era factible realizarlo por falta de tiempo. Los *hiperparámetros* se eligieron siguiendo lo hecho por Cañete et al. [3] y Lan et al. [12] finalmente estos fueron distintas combinaciones de: *batch size* {2, 4, 8, 16, 32} ,*learning rate* {5e-5, 3e-5, 3e-6, 1e-6}, *épocas* {3,4} y máximo largo de secuencia {512, 384}. Además, todos usaron un *warm up* del 10 % de los *steps* del modelo.

Debido a que los modelos large, xlarge y xxlarge, ocupaban demasiado espacio de la RAM de GPU, al intentar entrenarlos el código se detenía por falta de memoria, en consecuencia se debieron elegir *batch sizes* pequeños y ocupar *acumulación de gradiente*. Los valores espe-

[1] Repositorio código memoria https://github.com/sdonoso/glues_for_alberts_and_distilbeto
[2] Biblioteca Transformers https://huggingface.co/transformers/
[3] Código base para la tarea MLDoC https://github.com/gchaperon/spanish-bert-benchmarks

cíficos de los *hiperparámetros* de cada entrenamiento se pueden encontrar en las sección A.1 del Anexo.

El código para realizar el entrenamiento específico de cada tarea, en primer lugar carga el conjunto de datos, el modelo pre entrenado y el *tokenizer*. Este último ya fue entrenado por Cañete et al. [3] mediante el algoritmo SentencePiece [10]. Además, la biblioteca utilizada agrega una capa de salida a los modelos; esto es necesario debido a que las tareas de GLUES presentan distintos tipos de clasificaciones, por ejemplo en la tarea PAWS-X se realiza una clasificación binaria mientras que en la tarea NER el modelo hace una clasificación multiclase para cada token. Durante el entrenamiento cuando se termina un recorrido del modelo por todos los ejemplos guardamos una versión del modelo en ese punto de entrenamiento y lo utilizamos para realizar predicciones sobre el conjunto de datos de evaluación. Esto se realiza al finalizar cada *época*; de esta manera tenemos versiones del modelo en distintos puntos del entrenamiento y podemos elegir la versión con el menor valor de la función de pérdida en el conjunto de validación. Al finalizar por completo el entrenamiento en una tarea especifica, se guarda la versión con mejores resultados, se realizan las predicciones en el conjunto de prueba y se calcula la métrica correspondiente a la tarea, siendo este el valor que se reporta en los resultados. Es importante recalcar que para cada tarea sea realiza el entrenamiento con el modelo pre entrenado, y a su vez para cada tarea se obtiene un modelo distinto a los demás.

4.2. Pre entrenamiento y evaluación de DistilBETO

Como se mencionó en la sección 2.2.5 para realizar *"destilación"* se necesita el modelo *profesor* y el *estudiante*; en este caso estos corresponden a BETO *uncased* y DistilBETO. El modelo *estudiante* sigue la misma arquitectura de BETO con las modificaciones mencionadas en la sección 2.2.5. La eliminación de estos parámetros se justifica como menciona la investigación de Sanh et al. [17], debido a que de esta forma se logra la mejor optimización en desempeño y rapidez del modelo según lo mostrado en sus experimentos.

En este proceso utilizamos el código[4] publicado por Sanh et al.[17]. En primer lugar, realizamos un pre procesamiento del conjunto de datos *"Spanish Corpora"* [5] usado en el pre entrenamieno de BETO [3], la idea de esto es ahorrar el tiempo de tokenización al ir pasando los ejemplos al modelo. Originalmente el código recibía un solo archivo de texto que contenía una sentencia por línea; cambiamos esto ya que el peso total del conjunto de datos era de 18GB en disco y al momento de realizar el pre procesamiento el uso de memoria RAM superaba los 90GB por lo que no era viable realizarlo de esta manera. En consecuencia, procesamos los archivos del conjunto de datos de forma separada; cada secuencia fue tokenizada y guardada en archivos en formato binario. Luego contamos el número de ocurrencias de cada *tokens* puesto que la función de pérdida de MLM que se usa en el pre entrenamiento pone mayor énfasis en en los *token* menos frecuentes siguiendo la idea de Lample et al. [11].

Antes de comenzar el entrenamiento copiamos los pesos de la mitad de las capas de BETO

[4] Código DistilBERT https://github.com/huggingface/transformers/tree/master/examples/research_projects/distillation

[5] Conjunto de datos pre entrenamietno BETO https://github.com/josecannete/spanish-corpora

en el *estudiante* ya que esto ayuda a la convergencia del modelo en el pre entrenamiento [17]. Finalmente Realizamos 3 *épocas* de entrenamiento con 120M de secuencias, fijamos un *batch size* de 16 con *acumulación de gradiente* igual a 256 lo que nos entrega un paso del descenso de gradiente cada 4k ejemplos. A diferencia del entrenamiento en BERT no utilizamos "Next Sentence Prediction", ya que fueron removidas las partes necesarias para esta tarea.

Durante el entrenamiento se entregan los mismos ejemplos por separado a las redes *profesor* y *estudiante* junto con la máscara para los *tokens* que deben predecir. Primero calculamos la función de pérdida *"distillation loss"*; para esto tomamos los valores de las predicciones de los *tokens* con *masking* que entrega cada modelo sobre el vocabulario. Aplicamos las funciones *log-softmax* al *estudiante* y *softmax* al *profesor*, ambas con temperatura igual a 2 para suavizar la distribución de portabilidades sobre el vocabulario según lo señalado en la sección 2.2.5. A continuación calculamos la Divergencia de Kullback-Leibler entre los valores resultantes y guardamos el resultado. En segundo lugar, calculamos la función de perdidad MLM del *estudiante*; luego calculamos la función coseno para medir la distancia entre los vectores de la última capa del los modelos *estudiante* y *profesor*. Luego cada resultado de las tres funciones de pérdidas es multiplicado por un factor que determina cuánto será el aporte a la pérdida del *estudiante* y se suman obteniendo el valor de pérdida final con el que se actualizarán los parámetros de la red *estudiante*. Siguiendo las recomendaciones del código decidimos que la función *"distillation loss"* tuviera un mayor peso que las demás. Es preciso destacar que el pre entrenamiento descrito duró aproximadamente 25 días en completarse, por lo que no tuvimos la posibilidad de probar otros valores para los *hiperparámetros*.

En el entrenamiento y evaluación en las tareas de GLUES, solo realizamos modificaciones a algunas partes del código, considerando que DistilBETO *uncased*, utiliza un *tokenizer* distinto a s-ALBERT y se debe señalar que es un modelo entrenado solo en letras minúscula. Los *hiperparámetros* que utilizamos fueron distintas combinaciones de: *batch size* {16, 32}, *learning rate* {5e-5, 3e-5, 3e-6}, *épocas* {3, 4}, máximo largo de secuencia {512, 384} y *warm up* del 10 % de los *steps* del entrenamiento. Los otros pasos de la evaluación en GLUES fueron iguales a los explicados en la sección anterior.

4.3. Evaluación en Hardware de común acceso

Realizamos una prueba para calcular el tiempo que se demoran los modelos en predecir 100 ejemplos del conjunto de datos de prueba de la tarea NER. Para todos los modelos usamos *batch size* 1 y máximo largo de secuencia igual a 128. El equipo que utilizamos en los experimentos considerando que fuera de "común" acceso en Chile fue un Notebook "ASUS x409fa-ek114t" del año 2019, cuyas características son: 16GB de ram, procesador Intel Core i7-8655U. Repetimos 5 veces la prueba para cada modelo, de esta forma nos aseguramos que no existieran tiempos inconsistentes.

Capítulo 5

Resultados y análisis

5.1. Evaluación en GLUES

Tiempo de entrenamiento específico

El tiempo total que duró el entrenamiento específico de los modelos s-ALBERT y DistilBETO en las tareas de GLUES es aproximadamente 11,5 días completos, en la tabla 5.1 se muestra el detalle de la duración por modelo. Debemos aclarar que algunos tiempos no fueron bien registrados por lo que para hacer el cálculo total estimamos los valores faltantes con los valores de conjunto de datos y modelos cercanos en tamaño.

La tarea que nos tomó más tiempo fue XNLI, la cual duró aproximadamente 8,5 días para todos los modelos, mientras que la que se demoró menos fue NER con 1 hora y 11 minutos. La diferencia de tiempo depende en gran parte del número de ejemplos de los conjuntos de entrenamiento; esto varía según la tarea. Por ejemplo XNLI tiene 390k en cambio NER cuenta solo con 8k, siendo los con mayor y menor tamaño respectivamente.

Tabla 5.1: Tiempo de entrenamiento en tareas de GLUES.

Tiempos de Entrenamiento (s)							
N° ejemplos	392702	14305	49401	8324	10000	89194	89194
Modelo	XNLI	POS	PAWS-X	NER	MLDoC	MLQA	TAR
s-ALBERT tiny	6646	89	1242	100		199	9385
s-ALBERT base	45064	254	3001	163		485	11644
s-ALBERT base v2	38109	254	5125	158		489	7670
s-ALBERT base v3	37712	256	5682	163		476	20027
s-ALBERT large	201187	1024	4349	416		13805	4391
s-ALBERT xlarge		2514	13976	1526		38863	5079
s-ALBERT xxlarge		2549	27643	1654		38863	10446
DistilBETO *uncased*	13705	180	2053	112		2743	2496

Evaluación en GLUES

En las tablas 5.2 y 5.3 se muestran los resultados que obtuvimos en las tareas de GLUES; junto con los modelos que entrenamos están los resultados de BETO y mBERT. Todos los modelos comparados utilizaron los conjunto de datos en español para el entrenamiento descritos en las tareas de GLUES. Los valores más altos de cada tarea se encuentran remarcados en negro.

La tabla 5.3 muestra solo los resultados para la tarea *"Question Answering"* ya que como se indicó en la sección 2.2.6 esta presenta dos conjuntos de datos para el entrenamiento y prueba. Los valores de la derecha corresponden al *accuracy* del emparejamiento exacto y los de la izquierda a *F1*.

Tabla 5.2: Comparación de modelos s-ALBERT y DistilBETO con los resultados reportados en [3]. Superíndice indica los resultados obtenidos por (a) Wu & Dredze [25], (b) Yang et al. [28]. Todos los resultados de BETO son de Cañete et al. [3]

Modelo	XNLI	POS	PAWS-X	NER	MLDoC
Best mBert	$78,50^a$	$97,10^a$	$89,00^b$	$87,38^a$	$95,70^a$
BETO *uncased*	80,15	98,44	**89,55**	82,67	**96,12**
BETO cased	**82,01**	**98,97**	89,05	**88,43**	95,60
s-ALBERT tiny	71,23	97,12	80,50	78,20	83,87
s-ALBERT base v1	77,88	98,43	88,65	85,36	92,85
s-ALBERT base v2	78,46	98,22	87,80	85,21	93,75
s-ALBERT base v3	77,24	98,28	88,30	85,19	92,62
s-ALBERT large	75,18	94,77	72,45	84,08	81,52
s-ALBERT xlarge	76,62	97,23	86,65	74,75	80,50
s-ALBERT xxlarge	79,46	97,88	87,15	82,39	93,70
DistilBETO *uncased*	78,48	87,61	80,84	82,69	95,67

Tabla 5.3: Comparación de modelos s-ALBERT y DistilBETO con los resultados reportados en [3] en la tarea de QA. Los resultados corresponden a *F1* / emparejamiento exacto, cada encabezado de las columnas indica el conjunto de entrenamiento a la derecha y de prueba a la izquierda. Superíndice indica los resultados obtenidos por (c) Lewis et al. [13], (d) Carrino et al. [2] . Todos los resultados de BETO son de Cañete et al. [3]

Modelo	MLQA,MLQA	TAR,XQuAD	TAR,MLQA
Best mBERT	53,90 / $37,40^c$	**77,60** / $61,80^d$	68,10 / $48,30^d$
BETO *uncased*	67,85 / **46,03**	77,52 / 55,46	68,04 / 45,00
BETO cased	**68,01** / 45,88	77,56 / 57,06	**69,15** / 45,63
s-ALBERT tiny	49,76 / 28,3	62,50 / 41,76	54,11 / 32,05
s-ALBERT base v1	65,41 / 42,12	75,69 / 55,29	65,69 / 41,97
s-ALBERT base v2	65,86 / 42,49	74,95 / 54,78	65,07 / 41,55
s-ALBERT base v3	65,67 / 42,62	74,99 / 54,95	65,28 / 41,69
s-ALBERT large	63,35 / 39,65	73,00 / 53,27	63,51 / 40,07
s-ALBERT xlarge	67,83 / 43,46	74,52 / 53,61	63,14 / 39,00
s-ALBERT xxlarge	67,55 / 42,07	76,83 / 55,63	66,32 / 42,83
DistilBETO *uncased*	63,37 / 40,90	72,46 / 51,17	63,65 / 40,92

Se puede ver en ambas tablas que no obtuvimos ningún resultado mejor de los reportados con anterioridad. De los modelos entrenados por nosotros el que obtuvo mejores resultados fue s-ALBERT base v1 con un promedio total de 77,03 % lo que no se encuentra tan lejos del 79,46 % obtenido en promedio por BETO cased considerando la menor cantidad de parametros del tamaño base. Por otro lado s-ALBERT tiny obtuvo el peor resultado con un promdedio de 68,15 %. El promedio de los resultados en GLUES de todos los modelos se puede ver en la tabla 5.4 .

Los malos resultados obtenidos por los modelos en QA se pueden explicar en parte a que el conjunto de datos MLQA es de mala calidad como indica Cañete et al. [3]. El modelo s-ALBERT tiny obtuvo en promedio el peor resultado, esto se debe a que solo tiene 4 capas y la mitad de los parámetros de s-ALBERT base, lo que representa una menor capacidad para resolver las tareas. Sin embargo destacamos que sin contar la tarea de QA, s-ALBERT tiny solo obtuvo el peor resultado en la tarea POS.

Entre las distintas versiones de los modelos s-ALBERT base el que mejor resultados obtuvo fue la version v1, la diferencia con las versiones v2 y v3 fueron de solo 0,19 pp y 0,25 pp, esto se debe a que en general fueron entrenados con los mismos *hiperparámetros*. Creemos que sería interesante entrenarlos con valores de *hiperparámetros* distintos para ver la incidencia del pre entrenamiento y el entrenamiento específico en los resultados de las tareas de GLUES.

El modelo s-ALBERT xxlarge obtuvo el segundo mejor resultado, solo a 0,01 pp de s-ALBERT base v1, teniendo en cuenta que el primero es el modelo con mayor número de parámetros que evaluamos y que esta arquitectura fue la que obtuvo mejores resultados que BERT large [12], creemos que si se realiza un trabajo enfocado en la búsqueda de buenos *hiperparámetros* para cada tarea, se podrían obtener mejores resultados que BETO. En cambio, los modelos de tamaño s-ALBERT large y xlarge obtuvieron menos 5,23 pp y 3,72 pp que el modelo s-ALBERT base v1. Creemos que los perores resultados de estos se debe a una mala selección de *hiperparámetros*, ya que entrenamos los tres modelos más grandes con las mismas configuraciones sin considerar que las redes large y xlarge a pesar de tener menos parámetros que la red xxlarge tienen el doble de capas.

Tabla 5.4: Promedio de los resultados reportados en GLUES.

Modelo	Promedios en GLUE
Best mBert	77,65
BETO *uncased*	78,36
BETO cased	79,46
s-ALBERT tiny	68,15
s-ALBERT base v1	77,03
s-ALBERT base v2	76,97
s-ALBERT base v3	76,78
s-ALBERT large	71,80
s-ALBERT xlarge	73,32
s-ALBERT xxlarge	77,02
DistilBETO	73,94

DistilBETO

Respecto a todos los modelos que entrenamos en esta memoria DistilBETO está en la cuarta posición con un promedio de 73,94 % en GLUES. En MLDoc obtuvo mejor resultado que todos los modelos s-ALBERT, en cambio en POS obtuvo el peor desempeño con 87,61 %. Para la tarea PAWS-X a pesar de que el modelo *profesor* BETO *uncased* obtuvo el mejor resultado, DistilBETO tuvo una diferencia de 8,7 pp con este. Creemos que esto se debe a que el pre entrenamiento con *"destilación"* no considera las tareas con dos secuencias como lo es PAWS-X.

Ya que en la mayoría de las tareas DistilBETO no se alejó más de 5 pp de BETO *uncased* e incluso obtuvo 0,02 pp más en NER, creemos que si se realiza una búsqueda de *hiperparámetros* para POS se puede mejorar considerablemente el resultado. Finalmente podemos decir que DistilBETO mantuvo el 94 % de rendimiento de BETO unacsed en GLUES, con un tamaño 40 % menor.

5.2. Rendimiento en Hardware

En la tabla 5.5 se muestra de menor a mayor el promedio de los tiempos de inferencia de cada modelo, además incluimos el número de parámetros de estos, los tiempos de cada experimento se pueden revisar en la tabla A.8 en el anexo.

El modelo con el menor tiempo fue s-ALBERT tiny con 1,62 segundos, esto se debe a que junto con tener la menor cantidad de parámetros solo tiene 4 capas. DistilBETO logró ser aproximadamente 50 % más rápido que BETO *uncased*, lo que comprueba la buena selección de parámetros removidos.

Es interesante notar que a pesar de que los modelos s-ALBERT base, large y xlarge tienen al menos 60M de parámetros menos que BETO *uncased*, este es más rápido. Estas diferencias de tiempo también son señaladas por los autores de ALBERT [12]. La versión base es la más cercana en tiempo a BETO *uncased*. Creemos que esto se debe a que si bien existe una gran diferencia en el número de parámetros, estos modelos tienen la misma cantidad de capas, mientras que las versiones large y xlarge tienen el doble de capas. Por ultimo s-ALBERT xxlarge fue el que demoró más tiempo, como indica Lan et al. [12] esto era esperable ya que el modelo era muy grande.

Tabla 5.5: Promedio del tiempo de inferencia al predecir 100 ejemplos del conjunto de datos de NER.

	N° Parámetros	Tiempo (s)
s-ALBERT tiny	5M	1,62
DistilBETO *uncased*	66M	5,25
BETO *uncased*	110M	10,46
s-ALBERT base	12M	13,83
s-ALBERT large	18M	49,45
s-ALBERT xlarge	60M	224,08
s-ALBERT xxlarge	235M	484,45

Capítulo 6

Conclusiones

En el presente trabajo presentamos el nuevo modelo DistilBETO que entrenamos utilizando la técnica de *"destilación"* desde el modelo BETO *uncased* [3]. Además, realizamos la evaluación de los modelos s-ALBERT y DistilBETO en las tareas de GLUES para determinar la comprensión del lenguaje que tienen este tipo de redes de aprendizaje profundo. Finalmente, realizamos experimentos para diferenciar el rendimiento de estos modelos en hardware de común acceso.

Considerando que el pre entrenamiento de DistilBETO duró 90K *steps* mientras que según lo indicado por Cañete et al. [3] BETO fue pre entrenado con alrededor de 2M de *steps*, DistilBETO logró retener un 94 % de desempeño, siendo 40 % mas pequeño y 50 % más rápido. Esto demuestra que es posible obtener modelos que requieran menos recursos y mantengan buenos resultados. Si bien los modelos s-ALBERT obtuvieron buenos resultados en la mayoría de los casos, algunos modelos cómo s-ALBERT large obtuvieron resultados lejanos a modelos con peores rendimiento en las demás tareas. Creemos que esto se debió a la mala selección de hiperámetros en esas tareas. Además, esperábamos que el modelo de mayor tamaño lograra superar en alguna tarea a BETO, consideramos que esto se puede lograr realizando una búsqueda de *hiperparámetros* más exhaustiva pero esto requiere de más recursos tanto de hardware como de tiempo debido al número de parámetros del modelo.

Es claro que si bien DistilBETO requiere menos recursos para ser usado, de igual manera utilizamos una gran cantidad de recursos para entrenarlo. Ya que para usar la técnica de *"destilación"* es necesario tener un modelo grande previamente entrenado, junto con esto se debe considerar que el pre entrenamiento de DistilBETO tomó 25 días en un servidor que a pesar de no ser de categoría industrial contaba con hardware de alto costo. Se debe tener en cuenta que pese a que el trabajo que realizamos fue a una escala mucho menor en comparación con grandes empresas, el pre entrenamiento, entrenamiento y evaluación de todos los modelos duró en total alrededor de 36 días, lo que significa una gran cantidad de energía utilizada; esto recalca la necesidad por disminuir el impacto medio ambiental de este tipo de trabajos a través de por ejemplo, la utilización de energías renovables para el funcionamiento de los servidores. Por último, creemos que los 8 modelos entrenados y evaluados en este trabajo significan un gran aporte para el desarrollo de PLN en español y para la aplicación de este tipo de tecnologías en contextos locales.

Durante la realización de esta memoria aprendimos la importancia de la colaboración para

el aprendizaje y la creación de nuevas herramientas, ya que si no fuera por la libre disposición de los recursos desarrollados por otros, no hubiera sido posible la realización de este trabajo. Además, creemos que es sumamente significativo el enfocar los trabajos desarrollados en la universidad a entornos locales, ya que de esta manera se puede realizar un aporte a la sociedad.

Como trabajo futuro queda la posibilidad de mejorar los resultados de las evaluaciones en GLUES realizando una búsqueda más minuciosa de *hiperparámetros*. Además, se pueden entrenar versiones más pequeñas de los modelos s-ALBERT con mejores resultados utilizando la técnica de *"destilación"*. Finalmente sería interesante probar el desempeño de DistilBETO y los modelos s-ALBERT más pequeños en teléfonos celulares, ya que este tipo de aparatos son los más utilizados por las personas. Con esto se espera que mejore la accesibilidad de este tipo de tecnologías en países de habla hispana para el desarrollo de nuevas aplicaciones que ayuden a solucionar problemas de países menos desarrollado como Chile.

Bibliografía

[1] Artetxe, M., Ruder, S., and Yogatama, D. On the cross-lingual transferability of monolingual representations. *arXiv preprint arXiv:1910.11856* (2019).

[2] Carrino, C. P., Costa-jussà, M. R., and Fonollosa, J. A. Automatic spanish translation of the squad dataset for multilingual question answering. *arXiv preprint arXiv:1912.05200* (2019).

[3] Cañete, J., Chaperon, G., Fuentes, R., Ho, J.-H., Kang, H., and Pérez, J. Spanish pre-trained bert model and evaluation data. In *PML4DC at ICLR 2020* (2020).

[4] Conneau, A., Lample, G., Rinott, R., Williams, A., Bowman, S. R., Schwenk, H., and Stoyanov, V. Xnli: Evaluating cross-lingual sentence representations. *arXiv preprint arXiv:1809.05053* (2018).

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

[6] Eisenstein, J. *Natural Language Processing*. MIT Press, 2018.

[7] Goldberg, Y. A Primer on Neural Network Models for Natural Language Processing. *Journal of Artificial Intelligence Research 57* (oct 2016), 345–420.

[8] Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[9] Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[10] Kudo, T., and Richardson, J. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226* (2018).

[11] Lample, G., and Conneau, A. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291* (2019).

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

[13] Lewis, P., Oğuz, B., Rinott, R., Riedel, S., and Schwenk, H. Mlqa: Evaluating cross-lingual extractive question answering. *arXiv preprint arXiv:1910.07475* (2019).

[14] Nivre, J., Agić, Ž., Ahrenberg, L., Aranzabe, M. J., Asahara, M., Atutxa, A., Ballesteros, M., Bauer, J., Bengoetxea, K., Berzak, Y., Bhat, R. A.,

Bick, E., Börstell, C., Bosco, C., Bouma, G., Bowman, S., Cebiroğlu Eryiğit, G., Celano, G. G. A., Chalub, F., Çöltekin, Ç., Connor, M., Davidson, E., de Marneffe, M.-C., Diaz de Ilarraza, A., Dobrovoljc, K., Dozat, T., Droganova, K., Dwivedi, P., Eli, M., Erjavec, T., Farkas, R., Foster, J., Freitas, C., Gajdošová, K., Galbraith, D., Garcia, M., Gärdenfors, M., Garza, S., Ginter, F., Goenaga, I., Gojenola, K., Gökirmak, M., Goldberg, Y., Gómez Guinovart, X., Gonzáles Saavedra, B., Grioni, M., Grūzītis, N., Guillaume, B., Hajič, J., Hà Mỹ, L., Haug, D., Hladká, B., Ion, R., Irimia, E., Johannsen, A., Jørgensen, F., Kaşikara, H., Kanayama, H., Kanerva, J., Katz, B., Kenney, J., Kotsyba, N., Krek, S., Laippala, V., Lam, L., Lê Hồng, P., Lenci, A., Ljubešić, N., Lyashevskaya, O., Lynn, T., Makazhanov, A., Manning, C., Mărănduc, C., Mareček, D., Martínez Alonso, H., Martins, A., Mašek, J., Matsumoto, Y., McDonald, R., Missilä, A., Mititelu, V., Miyao, Y., Montemagni, S., Mori, K. S., Mori, S., Moskalevskyi, B., Muischnek, K., Mustafina, N., Müürisep, K., Nguyễn Thị, L., Nguyễn Thị Minh, H., Nikolaev, V., Nurmi, H., Osenova, P., Östling, R., Øvrelid, L., Paiva, V., Pascual, E., Passarotti, M., Perez, C.-A., Petrov, S., Piitulainen, J., Plank, B., Popel, M., Pretkalniņa, L., Prokopidis, P., Puolakainen, T., Pyysalo, S., Rademaker, A., Ramasamy, L., Real, L., Rituma, L., Rosa, R., Saleh, S., Saulīte, B., Schuster, S., Seeker, W., Seraji, M., Shakurova, L., Shen, M., Silveira, N., Simi, M., Simionescu, R., Simkó, K., Šimková, M., Simov, K., Smith, A., Spadine, C., Suhr, A., Sulubacak, U., Szántó, Z., Tanaka, T., Tsarfaty, R., Tyers, F., Uematsu, S., Uria, L., van Noord, G., Varga, V., Vincze, V., Wallin, L., Wang, J. X., Washington, J. N., Wirén, M., Žabokrtský, Z., Zeldes, A., Zeman, D., and Zhu, H. Universal dependencies 1.4, 2016. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

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

[16] Ramshaw, L., and Marcus, M. Text Chunking using Transformation-Based Learning. In *Third Workshop on Very Large Corpora* (1995), pp. 157–176.

[17] Sanh, V., Debut, L., Chaumond, J., and Wolf, T. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).

[18] Schwenk, H., and Li, X. A Corpus for Multilingual Document Classification in Eight Languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)* (Paris, France, may 2018), N. C. C. Chair), K. Choukri, C. Cieri, T. Declerck, S. Goggi, K. Hasida, H. Isahara, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk, S. Piperidis, and T. Tokunaga, Eds., European Language Resources Association (ELRA).

[19] Strubell, E., Ganesh, A., and McCallum, A. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243* (2019).

[20] Tjong Kim Sang, E. F. Introduction to the CoNLL-2002 shared task: Language-independent named entity recognition. In *COLING-02: The 6th Conference on Natural Language Learning 2002 (CoNLL-2002)* (2002).

[21] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems* (2017), pp. 5998–6008.

[22] Vig, J. A multiscale visualization of attention in the transformer model. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations* (Florence, Italy, July 2019), Association for Computational Linguistics, pp. 37–42.

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

[24] Williams, A., Nangia, N., and Bowman, S. R. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426* (2017).

[25] Wu, S., and Dredze, M. Beto, bentz, becas: The surprising cross-lingual effectiveness of bert, 2019.

[26] Xu, B. L. albert_zh, 2019. https://github.com/brightmart/albert_zh.

[27] Yang, Y., Zhang, Y., Tar, C., and Baldridge, J. Paws-x: A cross-lingual adversarial dataset for paraphrase identification. *arXiv preprint arXiv:1908.11828* (2019).

[28] Yang, Y., Zhang, Y., Tar, C., and Baldridge, J. Paws-x: A cross-lingual adversarial dataset for paraphrase identification, 2019.

Anexo A

A.1. Hiperparámetros

Las siglas usadas en las tablas son:

- bs: *batch size*.
- ag: *acumulación de gradiente*.
- msl: máximo largo de secuencia.
- lr: *learning rate*.
- e: *épocas*.

Tabla A.1: *Hiperparámetros* para el entrenamiento en la tarea XNLI

XNLI					
Modelo	bs	ag	msl	lr	e
s-ALBERT tiny	32	1	512	5e-5	4
s-ALBERT base	16	2	512	3e-5	4
s-ALBERT base v2	16	2	512	3e-5	4
s-ALBERT base v3	16	2	512	3e-5	4
s-ALBERT large	8	2	512	1e-6	4
s-ALBERT xlarge	4	4	512	1e-6	4
s-ALBERT xxlarge	2	8	512	1e-6	4
DistilBETO *uncased*	32	1	512	5e-5	4

Tabla A.2: *Hiperparámetros* para el entrenamiento en la tarea MLDoC

MLDoC					
Modelo	bs	ag	msl	lr	e
s-ALBERT tiny				0	
s-ALBERT base	16	1	512	3e-5	3
s-ALBERT base v2	16	1	512	3e-5	3
s-ALBERT base v3	16	1	512	3e-5	3
s-ALBERT large	8	2	512	3e-6	3
s-ALBERT xlarge	4	4	512	3e-6	3
s-ALBERT xxlarge	4	4	512	3e-6	3
DistilBETO *uncased*	32	1	512	3e-5	3

Tabla A.3: *Hiperparámetros* para el entrenamiento en la tarea PAWS-X

PAWS-X					
Modelo	bs	ag	msl	lr	e
s-ALBERT tiny					
s-ALBERT base	16	2	512	5e-5	4
s-ALBERT base v2	32	1	512	5e-5	4
s-ALBERT base v3	32	1	512	5e-5	4
s-ALBERT large	8	4	512	1e-6	4
s-ALBERT xlarge	4	4	512	1e-6	4
s-ALBERT xxlarge	4	4	512	1e-6	4
DistilBETO *uncased*	32	1	512	5e-5	4

Tabla A.4: *Hiperparámetros* para el entrenamiento en la tarea POS

POS					
Modelo	bs	ag	msl	lr	e
s-ALBERT tiny	32	1	512	3e-5	4
s-ALBERT base	32	1	512	5e-5	4
s-ALBERT base v2	16	2	512	3e-5	4
s-ALBERT base v3	16	2	512	3e-5	4
s-ALBERT large	8	4	512	1e-6	4
s-ALBERT xlarge	4	4	512	1e-6	4
s-ALBERT xxlarge	4	4	512	1e-6	4
DistilBETO *uncased*	32	1	512	3e-5	4

Tabla A.5: *Hiperparámetros* para el entrenamiento en la tarea NER

NER					
Modelo	bs	ag	msl	lr	e
s-ALBERT tiny					
s-ALBERT base	16	2	512	5e-5	4
s-ALBERT base v2	16	2	512	5e-5	4
s-ALBERT base v3	16	2	512	5e-5	4
s-ALBERT large	8	4	512	3e-5	4
s-ALBERT xlarge	2	8	512	1e-6	4
s-ALBERT xxlarge	2	8	512	1e-6	4
DistilBETO *uncased*	32	1	512	5e-5	4

Tabla A.6: *Hiperparámetros* para el entrenamiento en la tarea QA-MLQA

QA-MLQA					
Modelo	bs	ag	msl	lr	e
s-ALBERT tiny	16	2	384	3e-5	4
s-ALBERT base	16	2	384	3e-5	4
s-ALBERT base v2	16	2	384	3e-5	4
s-ALBERT base v3	16	2	384	3e-5	4
s-ALBERT large	4	4	384	3e-6	4
s-ALBERT xlarge	4	4	384	3e-6	4
s-ALBERT xxlarge	4	4	384	3e-6	4
DistilBETO *uncased*	16	2	384	3e-5	4

Tabla A.7: *Hiperparámetros* para el entrenamiento en la tarea QA-TAR

QA-TAR					
Modelo	bs	ag	msl	lr	e
s-ALBERT tiny	32	1	384	3e-5	4
s-ALBERT base	32	1	384	3e-5	4
s-ALBERT base v2	16	2	384	3e-5	4
s-ALBERT base v3	16	2	384	3e-5	4
s-ALBERT large	16	1	384	3e-6	4
s-ALBERT xlarge	8	2	384	3e-6	4
s-ALBERT xxlarge	8	2	384	3e-6	4
DistilBETO *uncased*	32	1	384	3e-5	4

A.2. Tiempos de inferencia

Tabla A.8: Tiempo de inferencia de las 5 ejecuciones de los modelos en 100 ejemplos del conjunto de prueba de NER.

Tiempos de inferencia (s)						
Modelo	1°	2°	3°	4°	5°	Promedio
s-ALBERT tiny	1,76	1,48	1,84	1,68	1,36	1,62
DistilBETO *uncased*	5,10	5,29	5,25	5,29	5,31	5,25
BETO *uncased*	10,19	10,05	10,56	10,86	10,63	10,46
s-ALBERT base	14,16	13,36	13,59	13,84	14,22	13,83
s-ALBERT large	49,19	50,80	47,93	48,74	50,61	49,45
s-ALBERT xlarge	211,52	212,58	237,24	233,32	225,75	224,08
s-ALBERT xxlarge	459,05	507,19	461,55	491,32	503,15	484,45