UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València

Traducción Automática de conjuntos de datos para la construcción de sistemas de Pregunta/Respuesta mediante Aprendizaje Automático

TRABAJO FIN DE MÁSTER

Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Autor: Juan Antonio López Ramírez
Tutor: Francisco Casacuberta Nolla

Curso 2019-2020

Resum

Les tasques d'aprenentatge automàtic basades en sistemes de pregunta/resposta requereixen de conjunts de dades per entrenar models predictius. Actualment, es compta amb aquests conjunts en idiomes molt parlats com l'anglès o el xinès, però no per a idiomes més locals com pot ser el basc.
L'objectiu d'aquest treball és, partint de conjunts de dades en anglès, crear les seves respectives versions en castellà, basc i català. Una vegada que estiguin creats aquests conjunts emprant tècniques de traducció automàtica, es procedeix a entrenar el model predictiu de pregunta/resposta per a cada idioma.
Cal tenir en compte no només aquells models entrenats amb un sol idioma, sinó aquells que fan servir mètodes multilingües (Per exemple, entrenament en un idioma i validació en un altre), ja que actualment aquests últims tenen prou pes en l'estat de l'art del processament del llenguatge natural i, específicament, en les tasques de pregunta/resposta.

Paraules clau: aprenentatge automàtic, traducció automàtica, pregunta/resposta, processament del llenguatge natural, Multilingüe

Resumen

Las tareas de aprendizaje automático basadas en sistemas de pregunta/respuesta requieren de conjuntos de datos para entrenar modelos predictivos. A día de hoy, se cuenta con dichos conjuntos en idiomas muy hablados como el inglés o el chino, pero no para idiomas más locales como pude ser el euskera.
El objetivo de este trabajo es, partiendo de conjuntos de datos en inglés, crear sus respectivas versiones en castellano, euskera y catalán. Una vez creados estos conjuntos empleando técnicas de traducción automática, se procede a entrenar el modelo predictivo de pregunta/respuesta para cada idioma.
Hay que tener en cuenta no solo aquellos modelos entrenados con un solo idioma, sino aquellos que emplean métodos multilingües (Por ejemplo, entrenamiento en un idioma y validación en otro), ya que actualmente estos últimos tienen bastante peso en el estado del arte del procesamiento del lenguaje natural y, específicamente, en las tareas de Pregunta/Respuesta.

Palabras clave: aprendizaje automático, traducción automática, pregunta/respuesta, procesamiento del lenguaje natural, multilingüe

Abstract

Machine Learning tasks based on QA systems require datasets to train predictive models. Nowadays, these sets are available in highly spoken languages such as English or Chinese, but not for regional languages such as Basque.
The aim of this work is to create datasets in Spanish, Basque and Catalan from the English versión. After creating these sets by Machine Translation techniques, we will proceed to train the predictive QA model for each language.
We must take into account that we can train models using just one or multiple languages (for example, training with one language and test with another one). We are taking into consideration multilingual models due to their importance in the NLP state of the art and, specifically, in QA tasks.

Key words: machine learning, machine translation, QA, NLP, multilingual

Índice general

Índice general v
Índice de figuras vii
Índice de tablas vii

1 Introducción 1
1.1 Traducción automática . . . 1
1.2 Comprensión de texto . . . 2
1.3 Objetivos . . . 3
1.4 Estructura de la memoria . . . 4
2 Traducción automática neuronal 5
2.1 Representación de palabras . . . 5
2.2 Redes Neuronales Recurrentes . . . 7
2.2.1 Redes neuronales recurrentes bidireccionales . . . 8
2.2.2 LSTM . . . 9
2.3 Arquitectura Encoder-Decoder . . . 11
2.3.1 Encoder . . . 12
2.3.2 Decoder . . . 12
2.3.3 Modelo de atención . . . 13
2.3.4 Fase de Decodificación . . . 14
2.4 Transformer . . . 14
2.4.1 Generalización del mecanismo de atención . . . 14
2.4.2 Normalización de capa . . . 16
2.4.3 Conexiones residuales . . . 16
2.4.4 Posición de las palabras . . . 17
2.4.5 Arquitectura . . . 17
2.4.6 Unión de pesos . . . 20
2.4.7 Suavizado de etiquetas . . . 21
2.5 Conclusiones . . . 22
3 Comprensión de texto para tareas de búsqueda de respuestas 23
3.1 Arquitectura de un sistema de búsqueda de respuestas . . . 24
3.2 Dimensiones del problema . . . 25
3.2.1 Usuarios . . . 25
3.2.2 Preguntas . . . 26
3.2.3 Respuestas . . . 27
3.2.4 Conocimiento necesario . . . 27
3.3 Clasificación de sistemas de búsqueda de respuestas . . . 28
3.3.1 Perspectiva general . . . 28
3.3.2 Sistemas que no utilizan técnicas de procesamiento del lenguaje natural . . . 29
3.3.3 Sistemas que emplean técnicas de análisis superficial . . . 29
3.3.4 Sistemas que utilizan técnicas de análisis profundo . . . 31
3.4 BERT . . . 31

3.4.1 Arquitectura . . . 32
3.4.2 Representaciones de la entrada y salida . . . 32
3.4.3 Preentrenamiento . . . 33
3.4.4 Afinación del modelo . . . 34
3.5 Conclusiones . . . 34
4 Implementación y experimentación **35**
4.1 Detalles de implementación . . . 35
4.2 Métricas empleadas . . . 35
4.2.1 BLEU . . . 35
4.2.2 Precisión . . . 36
4.2.3 *Recall* . . . 36
4.2.4 F1 . . . 36
4.3 Sistemas de traducción neuronal . . . 36
4.3.1 Datasets . . . 38
4.4 Sistemas de búsqueda de respuestas . . . 38
4.4.1 Datasets . . . 38
5 Resultados **41**
5.0.1 Traducción automática . . . 41
5.0.2 Búsqueda de respuestas . . . 42
5.1 Conclusiones . . . 43
6 Conclusiones finales y trabajos futuros **45**
6.1 Conclusiones finales . . . 45
6.2 Trabajos Futuros . . . 46
6.2.1 Nuevos datasets para la búsqueda de respuestas . . . 46
6.2.2 Enfoque multilingüe en la búsqueda de respuestas . . . 46
6.2.3 Mejora de resultados para euskera . . . 46
7 Agradecimientos **49**
Bibliografía **51**

Índice de figuras

2.1 Arquitectura neuronal para los *word-embeddings*. 6
2.2 Arquitectura de Elman para una red neuronal recurrente, desplegada en tres instantes de tiempo. 8
2.3 Arquitectura de una red neuronal bidireccional, desplegada en el tiempo. 9
2.4 Arquitectura de la LSTM. 10
2.5 Arquitectura del encoder-decoder. 12
2.6 Arquitectura Encoder-Decoder con un modelo de alineación. 15
2.7 Bloque encoder del Transformer. 18
2.8 Bloque decoder del Transformer. 18
2.9 Arquitectura completa del Transformer. 19

3.1 Arquitectura simple de un sistema de Búsqueda de Respuestas. 25
3.2 Procedimientos generales de preentrenamiento y afinación de BERT. . . . 32
3.3 Representación de la entrada para BERT. 33

4.1 Modelo de atención de red neuronal recurrente proporcionado por NMT-Keras. 37
4.2 Modelo de Transformer proporcionado por NMT-Keras. 37

6.1 Dos instancias de MLQA. 46

Índice de tablas

4.1 Datasets y cantidad de frases paralelas tomadas de cada uno de ellos para la traducción del inglés al español. 39
4.2 Datasets y cantidad de frases paralelas tomadas de cada uno de ellos para la traducción del español al catalán. 39
4.3 Datasets y cantidad de frases paralelas tomadas de cada uno de ellos para la traducción del español al euskera. 40

5.1 Resultados para el conjunto de test usando el modelo de atención de red neuronal recurrente de NMT-Keras. 41
5.2 Resultados para el conjunto de test usando el modelo de Transformer de NMT-Keras. 42
5.3 Resultados para el conjunto de test XQuAD usando BERT en Google Colab. 42

CAPÍTULO 1
Introducción

El lenguaje es el método de comunicación que ha permitido al ser humano desarrollarse en el máximo número de ámbitos posibles, desde el social y cultural hasta el científico. El lenguaje escrito es lo que ha permitido que el conocimiento haya podido ser transmitido con el paso de los años.

Ya sean manuscritos, impresos o digitales; la comprensión de textos es necesaria para que el lector pueda recibir el mensaje que se quiere transmitir. Debido a esto, dentro del área de la inteligencia artificial y el procesamiento del lenguaje natural, surge la comprensión del lenguaje, que pretende dotar a un sistema informático de aptitudes que le hagan capaz de entender aquello que se intenta transmitir en un texto determinado. A su vez, dentro de la comprensión de texto existen las tareas de búsqueda de respuestas, en las que el sistema informático ha de saber responder adecuadamente a unas determinadas preguntas que se le plantee, dado un cierto contexto donde se haya dicha respuesta.

Por otra parte, debido a que vivimos en un mundo multilingüe, existe una importante dificultad a la hora de que hablantes de diferentes idiomas puedan comunicarse. Es por esto por lo que surge la traducción automática, que permite que los ordenadores, de forma total o parcial, puedan llevar a cabo traducciones. Esta tarea se ha abordado desde diferentes puntos de vista, ya sea en la tecnología y fundamentos matemáticos utilizados como a la manera que tienen los usuarios de interactuar con estos sistemas.

1.1 Traducción automática

La Traducción automática puede verse como una aplicación de Reconocimiento de Formas que persigue el desarrollo de sistemas informáticos que sean capaces de traducir textos de forma automática. Cuando al sistema se le proporciona una frase, este produce una traducción de dicha frase en otro idioma, intentando preservar el significado original tanto como sea posible.

Moviéndonos a un plano más formal, una oración de tamaño N (es decir, que contiene N palabras) quedaría representada de la siguiente forma:

$$x = x_1, x_2, x_3..., x_N \tag{1.1}$$

Del mismo modo, la frase traducida a otro idioma estaría compuesta por M palabras (Donde M y N no tienen por qué ser iguales) y tendría la forma:

$$y = y_1, y_2, y_3..., y_M \tag{1.2}$$

Así, el objetivo principal es encontrar la frase objetivo y dada la frase original x, maximizando por tanto la probabilidad $p(y \mid x)$.

Para aprender esta probabilidad, se usan los corpus paralelos, que contienen oraciones en un idioma fuente junto con sus respectivas traducciones a un idioma objetivo. Es importante obtener tantas muestras paralelas como sea posible para que el sistema entrenado con ellas logre un buen rendimiento, además de que esos datos tengan una calidad adecuada. De esta forma, el sistema aprende la relación que existe entre el idioma de entrada y el de salida. Es necesario recalcar que los sistemas son capaces de traducir solo de un idioma a otro, de manera que si, por ejemplo, tenemos un sistema entrenado para traducir oraciones del español al catalán, haría falta capacitar a otro sistema para traducir del inglés al español.

Es necesario señalar que aprender una distribución de probabilidad condicional para las traducciones no es suficiente para un sistema de traducción automática totalmente funcional en tanto que saber qué tan buena es una oración traducida no nos proporciona suficiente información para saber si es la mejor, ya que podrían haber otras mejores.

Por último, está el paso de decodificación, en el que se prueban tantas traducciones como sea posible para así obtener la mejor de ellas. Debido a que hay muchas posibles frases en el espacio de búsqueda, esta se lleva a cabo en un subconjunto del espacio, con la esperanza de que la mejor de ellas, o al menos una que sea lo suficientemente correcta, esté presente en ese subconjunto.

1.2 Comprensión de texto

La comprensión del lenguaje natural es la parte del procesamiento del lenguaje natural que se encarga de interpretar un mensaje y entender su significado e intención, tal y como haría una persona. Para que construir un sistema funcione, independientemente de la tarea para la cuál esté pensado, necesita conjuntos de datos en el idioma específico, reglas de gramática, teoría semántica y pragmática (para entender el contexto e intencionalidad), etc. A partir de las muestras que proporcionan estos conjuntos se pueden recuperar partes específicas de los textos en base a palabras clave.

De manera similar al procesamiento del lenguaje natural, la comprensión del lenguaje natural usa algoritmos para reducir el habla humana en una ontología estructurada. A partir de aquí, existen algoritmos de inteligencia Artificial que detectan aspectos importantes como la intención, el tiempo, las ubicaciones, los sentimientos, etc. Una de las mayores diferencias con respecto al procesamiento del lenguaje natural es que va más allá de la comprensión de las palabras, ya que trata de interpretar el significado de los errores humanos comunes, como los pronunciamientos erróneos o las letras o palabras transpuestas.

Una hipótesis que impulsó la comprensión del lenguaje natural fue establecida por Noah Chomsky [2], y afirma que el objetivo fundamental en el análisis lingüístico para estudiar la estructura de las secuencias gramaticales es separar aquellas que son oraciones del lenguaje en cuestión de que aquellas que no lo son.

El análisis sintáctico se utiliza en múltiples tareas para evaluar cómo se alinea el lenguaje con las reglas gramaticales mediante la aplicación de dichas reglas a un grupo de palabras y derivando el significado de ellas en una serie de técnicas, como pueden ser la lematización, la segmentación morfológica o el salto de oraciones.

Sin embargo, la interacción humana permite producir errores en el texto y en el habla, compensándolos mediante un excelente reconocimiento de patrones y obteniendo información adicional del contexto. Esto muestra la desigualdad del análisis centrado en la

sintaxis y la necesidad de un enfoque más cercano en la semántica, que es el núcleo de la comprensión del lenguaje natural.

Este análisis semántico implica algoritmos informáticos para comprender el significado y la interpretación de las palabras; y es un problema que aún no se ha resuelto por completo. Algunas técnicas englobadas en este tipo de análisis son:

- Reconocimiento de entidades, en el que se identifican y clasifican partes de un texto en grupos predeterminados.
- Desambiguación del sentido de las palabras, ya que una palabra tiene un significado u otro en función del contexto.
- Generación de lenguaje natural mediante el uso de bases de datos para así convertir intenciones semánticas en lenguaje humano.

Sin embargo, para comprender completamente el lenguaje natural, las máquinas deben tener en cuenta no solo el significado literal que proporciona la semántica, sino también el mensaje deseado o la comprensión de lo que el texto está tratando de transmitir.

En el caso particular del presente trabajo, la comprensión del lenguaje natural se encuentra implícita en la tarea de búsqueda de respuestas. A grandes rasgos, esta tarea consiste en la recuperación y el análisis de información presente en documentos, para que el sistema sea capaz de responder a preguntas planteadas sobre dichos documentos. Así, los elementos más importantes y que sirven para que nuestro sistema pueda aprender a responder correctamente las preguntas que se le plantee son los siguientes:

- Los contextos, que suelen ser párrafos de varias líneas donde se encuentran las respuestas.
- Preguntas asociadas a esos contextos.
- Respuestas que satisfacen esas preguntas.

Por tanto, un conjunto de datos bien estructurado para la búsqueda de respuestas suele estar caracterizado por muestras, donde cada una de estas contiene un determinado número de párrafos, que a su vez tienen asignados una cierta cantidad de preguntas y sus correspondientes respuestas. Estas respuestas llevan asociadas un número que indica la posición en el párrafo donde comienza la respuesta.

1.3 Objetivos

El objetivo principal de este trabajo consiste en crear tres sistemas de búsqueda de respuestas que permitan, dado un determinado contexto, responder una pregunta que se les plantee. Cada sistema funciona en base a un determinado idioma, teniendo sistemas de búsqueda de respuestas para el español, el catalán y el euskera. Debido a la escasez de conjuntos de datos en esos idiomas para esta tarea, se ha planteado partir de dos datasets conocidos, como son **SQuAD2.0** [1] y **XQuAD** [62] y obtener sus correspondientes versiones en los tres idiomas. Para ello, se emplea la traducción automática.

Para ambas tareas (traducción y búsqueda de respuestas), se hará uso de los principales modelos de Deep Learning empleados en el campo del procesamiento del lenguaje natural, entre los que se encuentran las Redes Neuronales Recurrentes y el Transformer.

1.4 Estructura de la memoria

En este primer capítulo hemos presentado el problema que aborda el trabajo y los objetivos que se pretenden conseguir. En el capítulo 2 se introducirá el concepto de traducción automática neuronal, mientras que en el capítulo 3 se hará lo propio con la comprensión de texto orientada a las tareas de Pregunta/Respuesta. En ambos casos, se revisarán y explicarán sus respectivos enfoques, los conceptos más importantes y las arquitecturas más comunes, que serán utilizadas a lo largo de este trabajo.

Todas estas aproximaciones serán comparadas, probadas y evaluadas mediante conocidos conjuntos de datos en el capítulo 4 y 5.

Finalmente, en el capítulo 6 se expondrán las conclusiones extraídas de este trabajo y los posibles enfoques de trabajos futuros relacionados con este.

CAPÍTULO 2
Traducción automática neuronal

La traducción automática neuronal lleva a cabo el proceso de traducción mediante redes neuronales artificiales. Desde hace un tiempo, este método, enfocado en el marco estadístico, se ha erigido como el estado del arte en la traducción automática.

Las redes neuronales se caracterizan por realizar una combinación lineal de sus entradas y, posteriormente, aplicar una función no lineal a sus salidas. El uso de este método en la traducción automática se remonta a 1997 [3], aunque en su momento este enfoque presentaba una serie de problemas, como la propagación de los gradientes a través de la red y el alto coste computacional.

Mientras que el primer problema se solucionó con nuevas metodologías, como los entornos de trabajo informáticos de diferenciación automática [4] [5] [6]; el segundo se solventó al sustituir las unidades centrales de procesamiento (CPU) por las unidades gráficas de procesamiento (GPU), ya que estas permiten un mayor grado de paralelización, consiguiendo una aceleración del entrenamiento de las redes.

2.1 Representación de palabras

A pesar de que hay distintas opciones para representar las palabras en un sistema de traducción neuronal, el método estándar es el de utilizar una representación continua. Los *word-embeddings* son vectores continuos de características que sirven para ese propósito. En lugar de trabajar con vectores discretos de tamaño igual al vocabulario (si tenemos, por ejemplo, 400 palabras, los vectores son de tamaño 400), se trabaja con vectores continuos de menor tamaño (por ejemplo, 100). Si el tamaño del vocabulario es $|V|$ y el tamaño de los vectores es m, los *embeddings* se guardan en una matriz E, tal que:

$$E \in \Re^{|V| \cdot m} \tag{2.1}$$

de forma que se pueda indexar por palabras. Estas palabras, por lo general, se proyectan linealmente en el espacio de los *word-embeddings*.

Otra de las razones para descartar la representación discreta de palabras es el hecho de que no tienen en cuenta la similitud entre palabras, ya que es lógico suponer que palabras similares, desde un punto de vista léxico o semántico, han de tener vectores de características parecidos entre si.

El modelo de lenguaje propuesto por Bengio et al. [7] contiene un enfoque para estimar la probabilidad de las palabras en un espacio continuo. En él, las palabras son asignadas a un vector real v, tal que:

$$v \in \Re^m \tag{2.2}$$

Posteriormente, para obtener las probabilidades asociadas a cada palabra y su contexto, una función f asigna una secuencia de entrada de esos vectores a una distribución de probabilidad condicional sobre las palabras del vocabulario. Los parámetros de esta función f, junto con los vectores de características de las palabras, son entrenados empleando una red neuronal *feed-forward*, tal y como se muestra en la figura 2.1.

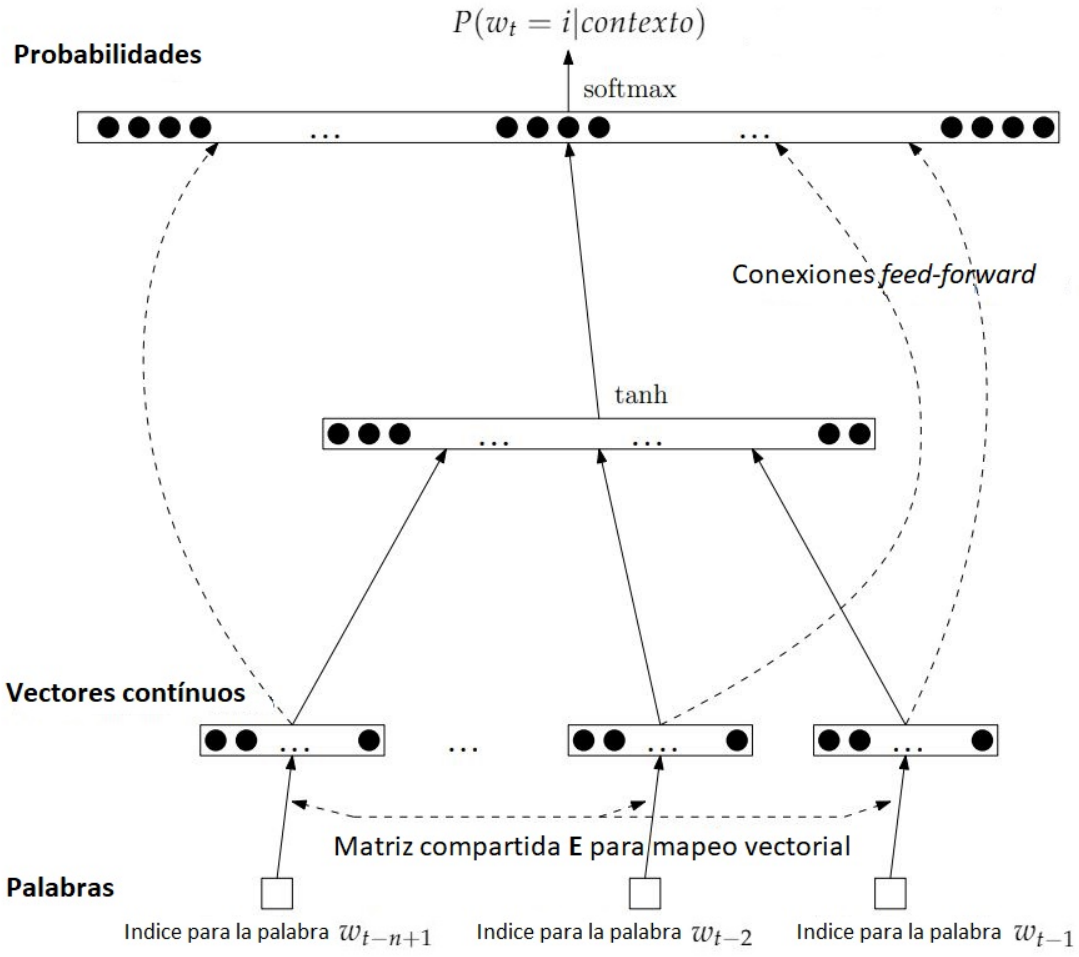

Figura 2.1: Arquitectura neuronal para los *word-embeddings*.

Todo este trabajo demostró un rendimiento sobresaliente en la predicción de palabras, pero también la necesidad de un modelo más eficiente desde una perspectiva computacional.

Mikolov et al. [8] propusieron un modelo que podría aprender la representación de palabras mucho más rápido y con mayor calidad que los enfoques anteriores. Mostró la relación aprendida entre palabras con álgebra simple simplemente sumando y restando vectores para obtener otras palabras. Por ejemplo, para obtener la palabra Reina se puede tomar la palabra Rey y restarle la palabra Hombre. Mikolov usó redes neuronales recurrentes para este propósito, en el que los *word-embeddings* adquieren más importancia, dejando en un segundo plano al modelo de lenguaje y sus probabilidades. Luego añadiría algunos cambios para mejorar el rendimiento general del sistema [9], entre los que se encuentra el uso de una capa *softmax* como una versión simplificada de la técnica de estimación del contraste de ruido. Esta capa se utiliza para obtener probabilidades y constituye una parte significativa del trabajo computacional. La capa *softmax* aplica la

función *softmax* a sus entradas y devuelve un vector en el que todos sus elementos suman uno. Dicha función tiene la siguiente forma:

$$\sigma(z_k) = \frac{exp(z_k)}{\sum_{m=1}^{K} exp(z_m)} \tag{2.3}$$

donde z es un vector k-dimensional que representa la entrada de la función *softmax*.

Los *word-embeddings*, más allá de la traducción neuronal o la comprensión de texto en tareas de búsqueda de respuestas, han demostrado ser útiles en otras áreas del procesamiento del lenguaje natural, como en el análisis sintáctico o el análisis de sentimientos.

2.2 Redes Neuronales Recurrentes

Una red neuronal recurrente es un tipo de red neuronal artificial donde las conexiones entre unidades forman un ciclo dirigido. Esto crea un estado interno de la red que le permite modelar un comportamiento temporal discreto.

Si tenemos, como entrada del sistema, una secuencia de T vectores:

$$x_1^T = x_1, ..., x_T \tag{2.4}$$

una red neuronal recurrente produce una secuencia de salida con la forma:

$$y_1^T = y_1, ..., y_T \tag{2.5}$$

que se calcula de la siguiente forma:

$$h_t = f_h(x_t, h_{t-1}) \tag{2.6}$$

$$y_t = f_o(h_t) \tag{2.7}$$

donde h_t es el estado oculto de la red en el instante de tiempo t, f_h es la función de estado oculto y f_o es la función de salida.

Existen distintas arquitecturas de redes neuronales recurrentes que utilizan diferentes funciones para h_t, f_h y f_o, como las redes de Jordan [10] o las redes de Elman [11]. Esta última arquitectura se puede apreciar en la figura 2.2, que consiste en una capa de entrada, una capa oculta conectada a sí misma y una capa de salida. Aquí, las funciones son definidas como:

$$h_t = f_h(x_t, h_{t-1}) = \phi(W_h^T h_{t-1} + W_x^T x_t) \tag{2.8}$$

$$y_t = f_o(h_t) = \sigma(W_y^T h_t) \tag{2.9}$$

donde $\mathbf{W}_h$, $\mathbf{W}_x$ y $\mathbf{W}_y$ son las matrices de peso recurrente, de entrada y de salida, respectivamente; ϕ es una función de activación no lineal, como la función sigmoide o la tangente hiperbólica; y σ es la función de activación de la salida, que suele ser la función *softmax* (Ecuación 2.3).

Estas tres matrices de pesos forman el conjunto de parámetros del modelo y normalmente se estiman mediante un método de descenso del gradiente estocástico, usando el

algoritmo de retropropagación a través del tiempo [12] para minimizar una función de coste bajo algún criterio de optimización, típicamente entropía cruzada entre la salida del sistema y la distribución de probabilidad de los datos de entrenamiento.

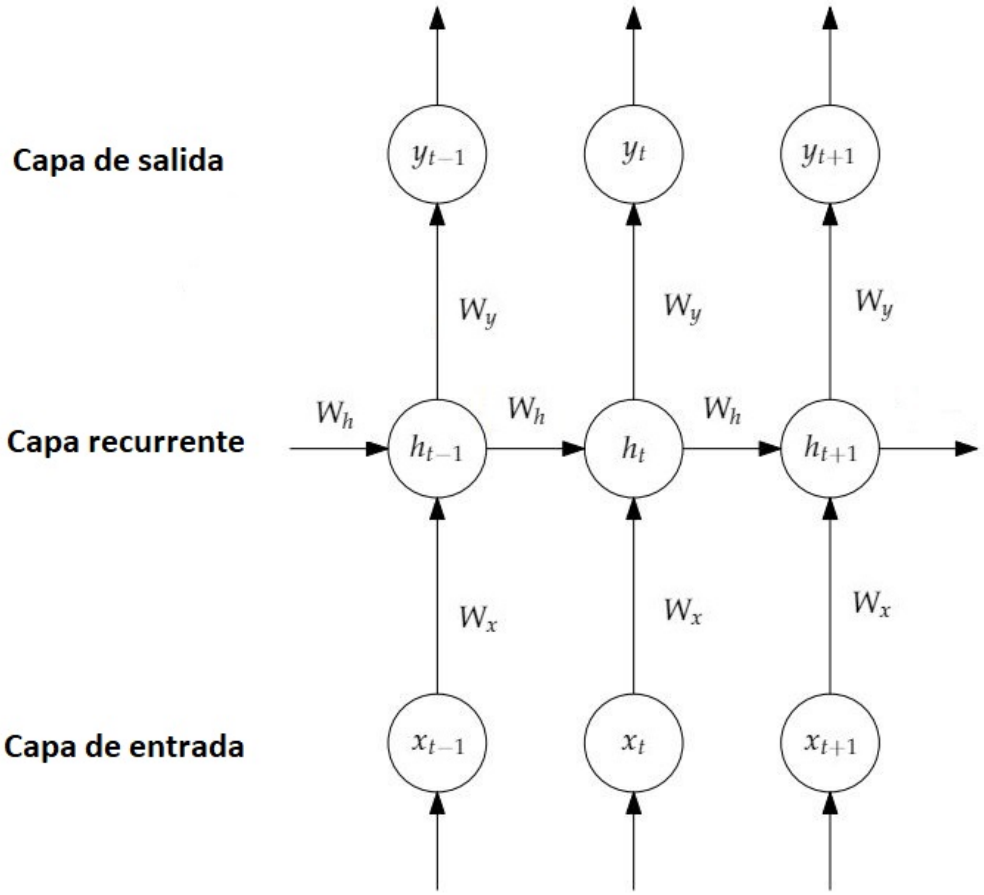

Figura 2.2: Arquitectura de Elman para una red neuronal recurrente, desplegada en tres instantes de tiempo.

2.2.1. Redes neuronales recurrentes bidireccionales

Las redes recurrentes regulares presentan el inconveniente de que solo se analizan en una dirección, normalmente hacia adelante, es decir, del pasado al futuro. Para comprender de forma completa el contexto de las secuencias, surgieron las redes neuronales recurrentes bidireccionales [13].

La idea principal de esta arquitectura es tener dos capas recurrentes independientes. Así, mientras una capa procesa la secuencia de entrada hacia adelante (de 1 a T), la otra lo hace de forma inversa, hacia atrás (de T a 1). Como las capas ocultas no tienen interacción entre ellas, las redes recurrentes bidireccionales se pueden entrenar utilizando los mismos

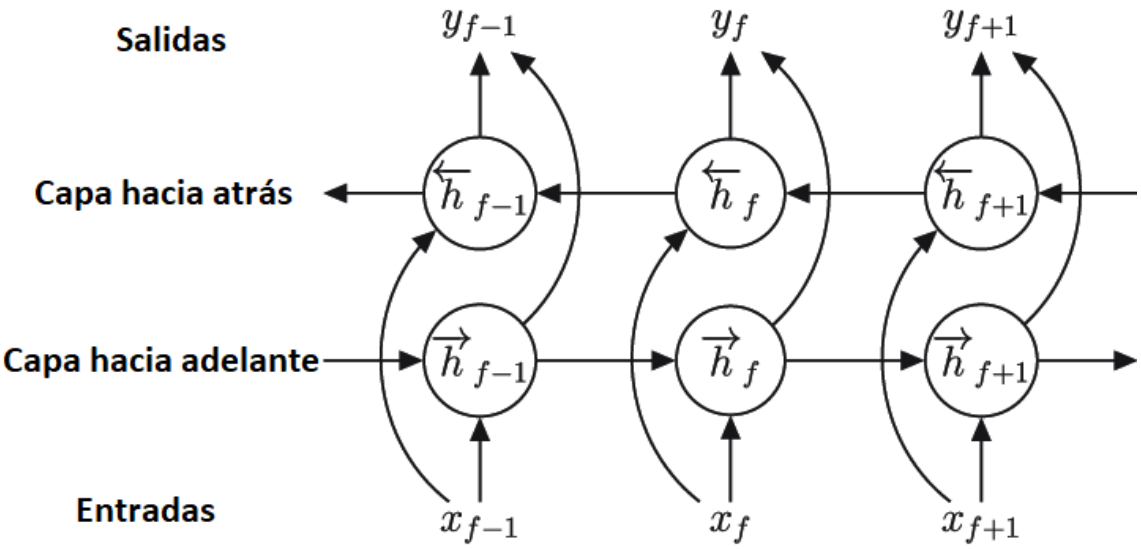

Figura 2.3: Arquitectura de una red neuronal bidireccional, desplegada en el tiempo.

algoritmos que los utilizados para las redes recurrentes simples. Siguiendo la notación previa, una red recurrente bidireccional se define como:

$$h_t^f = f_h(x_t, h_{t-1}^f) = \phi(W_f^T h_{t-1}^f + W_x^T x_t) \tag{2.10}$$

$$h_t^b = f_h(x_t, h_{t+1}^b) = \phi(W_b^T h_{t+1}^b + W_x^T x_t) \tag{2.11}$$

$$y_t = f_o(h_t^f, h_t^b) = \sigma(W_y^T(h_t^f + h_t^b)) \tag{2.12}$$

donde h_t^f es la capa que va en sentido hacia adelante y h_t^b la capa que va en sentido hacia atrás. Igual que antes, $\mathbf{W}_x$ y $\mathbf{W}_y$ son las matrices asociadas las capas de entrada y salida, mientras que $\mathbf{W}_f$ y $\mathbf{W}_b$ son las matrices de pesos de las capas hacia adelante y hacia atrás. La salida es una combinación producida por la función de salida de las capas hacia atrás y hacia adelante. Un esquema de esta arquitectura se puede apreciar en la figura 2.3.

2.2.2. LSTM

Un problema que presentan las redes neuronales recurrentes clásicas es el de construir modelos que tengan en cuenta dependencias a largo plazo [14]. Mientras más larga sea la dependencia, más difícil es de capturar.

Cuando la red produce una salida, obtiene el gradiente de la pérdida con respecto a los pesos que deben propagarse a lo largo de sus pasos. En cada paso que retropropagamos, el gradiente se vuelve más pequeño ya que estamos multiplicando números menores que uno. Dado que el gradiente se aplica a capas profundas de la red, es tan pequeño que no tiene ningún efecto sobre los parámetros que deben actualizarse. Esto se conoce como el desvanecimiento del gradiente.

Por otra parte, está el efecto contrario, es decir, si multiplicamos continuamente números mayores a uno obtendremos números demasiado grandes que afectarán demasiado a los parámetros de la red y producirá inestabilidad numérica.

Para solucionar estos problemas, se recortó el valor del gradiente a un número máximo y se propusieron nuevas funciones de activación que controlasen el flujo de información. De esta forma, se asegura que la derivada de la función recurrente se aproxime lo máximo posible a uno y se evita que el gradiente desaparezca.

Una opción para esto es utilizar las redes LSTM [15] [16], que comparten la estructura de una red recurrente clásica, pero cada celda simple se sustituye por una serie de puertas que tienen la capacidad de añadir o suprimir información del estado de la celda. Dicho estado actúa como una cinta transportadora entre diferentes neuronas y permite un fácil flujo de la información. Existen diferentes puertas, cada una con un objetivo específico, y todas están compuestas por una función sigmoide τ, un operador de multiplicación que consiste en una puerta XNOR $\odot$ y la suma de un vector de sesgo o *bias*. La función sigmoide se aplica para obtener un valor entre 0 y 1 que representa la cantidad de información que fluye. En la figura 2.4 se puede apreciar la estructura que presenta una celda LSTM.

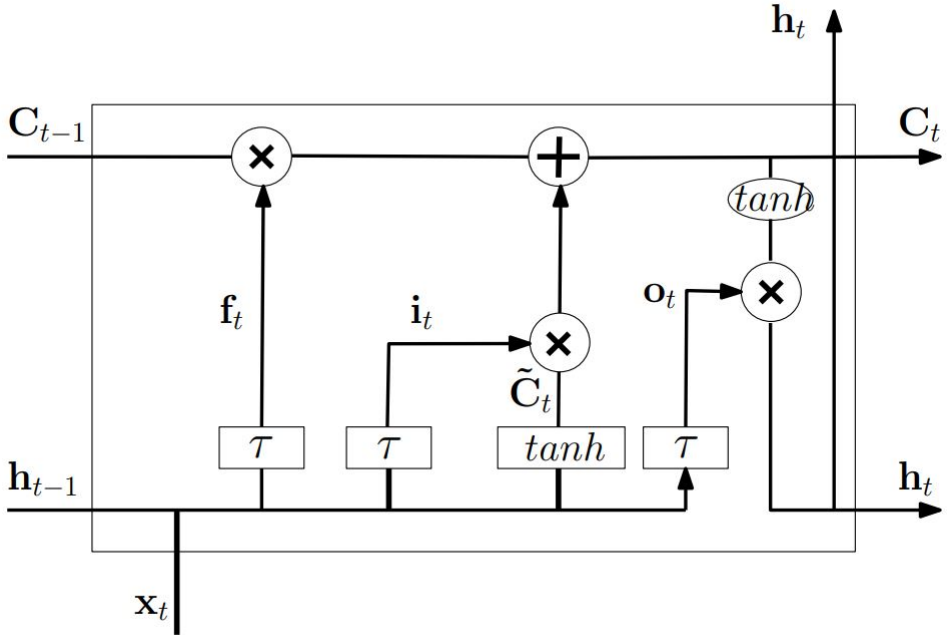

Figura 2.4: Arquitectura de la LSTM.

Las cuatro puertas distintas que constituyen una celda LSTM son las siguientes:

- **Forget gate (f_t):** Decide cuánta información se olvida del estado de la celda anterior (C_{t-1}). Para ello, la puerta utiliza el estado oculto anterior h_{t-1} y la entrada x_t, de forma que se calcula de la siguiente forma:

$$f_t = \tau(W_f x_t + U_f h_{t-1} + b_f) \tag{2.13}$$

  donde $\mathbf{W}_f$ y $\mathbf{U}_f$ son los pesos de la entrada y de las conexiones recurrentes para esta puerta.

- **Input gate (i_t):** Regula la cantidad que pasa de la entrada h_{t-1}, es decir, cuánta parte del contexto antiguo afecta al presente. Se calcula de la siguiente forma:

$$i_t = \tau(W_i x_t + U_i h_{t-1} + b_i) \tag{2.14}$$

donde $\mathbf{W}_i$ y $\mathbf{U}_i$ son los pesos de la entrada y de las conexiones recurrentes para esta puerta.

- **Memory gate (C_t):** Genera la nueva memoria junto con la puerta de entrada, decidiendo qué parte de la nueva información se almacenará en el estado de la celda, aplicando para ello la función tangente hiperbólica. Se calcula de la siguiente forma:

$$\bar{C}_t = tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{2.15}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \bar{C}_t \tag{2.16}$$

donde $\mathbf{W}_c$ y $\mathbf{U}_c$ son los pesos de la entrada y de las conexiones recurrentes para esta puerta, mientras que $\odot$ es, como se ha comentado anteriormente, un operador de multiplicación que consiste en una puerta XNOR.

- **Output gate (o_t):** Consiste en la salida de la celda y es una versión filtrada del estado de esta. Se calcula de la siguiente forma:

$$o_t = \tau(W_o x_t + U_o h_{t-1} + b_o) \tag{2.17}$$

donde $\mathbf{W}_o$ y $\mathbf{U}_o$ son los pesos de la entrada y de las conexiones recurrentes para esta puerta.

Por último, el estado oculto de la celda se calcula teniendo en cuenta la salida y la memoria, tal que:

$$h_t = o_t \odot tanh(C_t) \tag{2.18}$$

2.3 Arquitectura Encoder-Decoder

Hemos visto que cuando una red neuronal recurrente procesa una secuencia de longitud T, produce una secuencia de salida de la misma longitud. Sin embargo, en la traducción, las oraciones de origen y destino suelen tener distintos tamaños. Para solventar esto, se propusieron algunos modelos [17] [18] basados en un enfoque encoder-decoder. Luego, se extendió este modelo permitiendo que el decodificador busque dinámicamente en la oración fuente mientras decodifica una traducción [19].

El enfoque del encoder-decoder consiste en un proceso de dos partes. En primer lugar, se mapea la oración de origen en un vector de longitud fija y, posteriormente, este vector se decodifica para producir una oración de salida, probablemente de distinto tamaño. En la figura 2.5 se puede apreciar este concepto, donde $c_1, c_2, c_3, ..., c_N$ es el vector de contexto, que será explicado más adelante.

Este sistema está entrenado para maximizar la probabilidad condicional de verosimilitud sobre un conjunto de frases bilingües. Si dicho conjunto lo definimos como $\{(x_1, y_1), ..., (x_J, y_J)\}$, buscamos el conjunto óptimo de parámetros que maximice esa probabilidad sobre el conjunto de entrenamiento:

$$\arg\max_{\theta} \frac{1}{J} \sum_{j=1}^{J} log p_\theta(y_j | x_j) \tag{2.19}$$

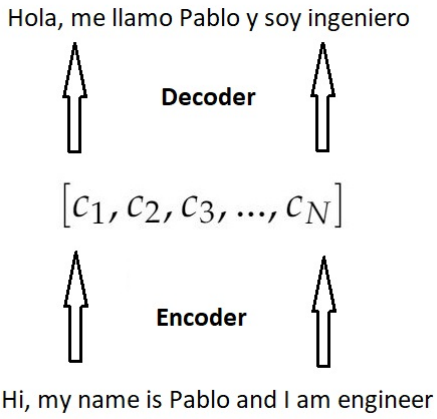

Figura 2.5: Arquitectura del encoder-decoder.

2.3.1. Encoder

El encoder, o codificador, es una red neuronal recurrente que lee cada secuencia de entrada de palabras $x_1^J = x_1, ..., x_J$ y lo encapsula en un vector de contexto **c**. Después de leer cada elemento de entrada x_j, el estado oculto de la red recurrente (h_j) cambia. Cuando se ha leído toda la secuencia de entrada, el estado oculto es un compendio de la secuencia. El vector de contexto **c** (que se puede observar en la figura 2.5) se construye aplicando una función no lineal a la secuencia de estado oculta:

$$h_j = f(x_j, h_{j-1}) \tag{2.20}$$

$$\mathbf{c} = q(\{h_1, ..., h_j\}) \tag{2.21}$$

donde h_j es el estado oculto en el momento j y f y q son funciones no lineales.

La arquitectura escogida para el encoder consiste en una red recurrente bidireccional, cuyos estados en dirección hacia adelante se alimentan con la secuencia de entrada ordenada (de x_1 a x_J) y sus estados hacia atrás leen la secuencia en orden inverso (de x_J a x_1). Por tanto, la capa hacia adelante (h_j^f) calcula una secuencia de estados ocultos hacia adelante (de h_1^f a h_J^f), mientras que la capa hacia atrás (h_j^b) hace lo propio con una secuencia de estados ocultos hacia atrás (de h_J^b a h_1^b). Para cada palabra x_j, se obtiene un h_j concatenando los estados hacia adelante con los estados hacia atrás:

$$h_j = \left[h_j^{f^T}; h_j^{b^T} \right]^T \tag{2.22}$$

Debido a las características de las redes neuronales recurrentes, el estado oculto h_j de una palabra x_j se centrará en las palabras circundantes de x_j. Así, h_j contiene una representación de las palabras anteriores y posteriores a x_j.

2.3.2. Decoder

El decoder, o decodificador, está entrenado para generar una frase de salida $y_1^I = y_1, ..., y_I$ dadas las palabras previas predichas y el vector de contexto **c**. Aplicando la regla de la cadena, se obtiene la siguiente expresión:

$$p(y_1, ..., y_I) = \prod_{i=1}^{I} p(y_i|y_1, ..., y_{i-1}, \mathbf{c}) \tag{2.23}$$

Para tener modelos que sean fáciles de tratar, se asume que existen dependencias directas con la palabra generada previamente (y_{i-1}). Desde la perspectiva de la redes neuronales recurrentes, cada probabilidad condicional se modela como:

$$p(y_i|y_1, ..., y_{i-1}, \mathbf{c}) = g(y_{i-1}, \mathbf{s}_i, \mathbf{c}) \tag{2.24}$$

donde g es una función no lineal y $\mathbf{s}_i$ es el estado oculto de la red neuronal recurrente del decoder.

Hay que tener en cuenta que, en este enfoque, cada frase se representa en el vector de tamaño fijo $\mathbf{c}$. Si la frase es larga, el vector de contexto no podrá capturar correctamente toda la información ni las relaciones existentes dentro de esta. Cho et al. [18] observaron esto cuando vieron que el rendimiento del sistema se reduce en gran medida cuando hay oraciones largas.

2.3.3. Modelo de atención

Para solventar el problema que se acaba de plantear, Bahdanau et al. [19] emplearon un vector de contexto diferente c_i para cada palabra objetivo y_i, es decir, un vector de contexto de longitud variable. De esta forma, las oraciones largas tendrán largas secuencia de vectores de contexto, siendo capaces de representar adecuadamente toda la información y relaciones existentes en la oración.

Así, teniendo en cuenta todo esto, la ecuación 2.24 se reescribe como:

$$p(y_i|y_1, ..., y_{i-1}, \mathbf{c}_i) = g(y_{i-1}, s_i, \mathbf{c}_i) \tag{2.25}$$

El vector de contexto $\mathbf{c}_i$ se calcula como una suma ponderada de la secuencia de anotaciones generadas por el encoder:

$$\mathbf{c}_i = \sum_{j=1}^{J} \alpha_{ij} h_j \tag{2.26}$$

donde α_{ij} es el peso asignado a cada anotación $\mathbf{h}_j$. Este peso es calculado siguiendo la función softmax:

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{J} exp(e_{ik})} \tag{2.27}$$

donde $e_{ij} = a(\mathbf{s}_{i-1}, \mathbf{h}_j)$ es una puntuación proporcionada por un modelo de alineación que puntúa como de bien coinciden las entradas alrededor de la posición j y la salida en la posición i.

Hay diferentes formas de implementar ese modelo de alineación, pero dado que tiene que puntuar J x I posibles alineaciones, interesa tener un sistema rápido y ligero. Bahdanau et al. [19] lo implementaron usando un perceptrón multicapa simple, basándose en la anotación de la oración fuente $\mathbf{h}_j$ y el estado anterior del decodificador $\mathbf{s}_{i-1}$:

$$a(\mathbf{s}_{i-1}, \mathbf{h}_j) = \mathbf{v}_a^T tanh(\mathbf{U}_a \mathbf{h}_j + \mathbf{W}_a \mathbf{s}_{i-1}) \tag{2.28}$$

Donde $\mathbf{U}_a \in \Re^{m' x m}$, $\mathbf{U}_a \in \Re^{m' x 2m}$ y $\mathbf{v}_a \in \Re^{m'}$ son las matrices de pesos. m es el número de unidades de la capa oculta del encoder y m' el número de unidades de la capa oculta del decoder. En la figura 2.6 se puede apreciar la estructura de un sistema de traducción neuronal con un modelo de atención.

2.3.4. Fase de Decodificación

El objetivo final de un sistema de traducción neuronal es generar traducciones. Dada la oración fuente f_i^J, la traducción t_1^I será la frase con máxima probabilidad a posteriori:

$$\hat{t}_1^I = \arg\max_{I, t_1^I} p(t_1^I | f_1^j, t_1^{I-1}) \tag{2.29}$$

El espacio de búsqueda son todas las frases posibles en el idioma destino. Obtener la solución óptima es intratable, por lo que se debe utilizar soluciones subóptimas para generar traducciones en un tiempo admisible. Por lo general, la estrategia es la búsqueda en haz (*Beam search*), una aproximación que considera las B mejores hipótesis en cada paso de tiempo. Las posibles hipótesis parciales se agregan en cada fase, pero la poda mantiene el número de mejores hipótesis siempre igual a B. Este proceso continúa hasta que se genera el símbolo de fin de secuencia.

2.4 Transformer

La arquitectura del Transformer [20] fue introducida recientemente para reemplazar a las capas recurrentes por un nuevo tipo de capa, denominadas capas de atención, además de añadir una serie de cambios en la estructura del encoder y el decoder. Este modelo logra importantes mejoras tanto en la velocidad como en la calidad de las traducciones.

Tanto el encoder como el decoder están compuestos por una serie de bloques de capas apilados uno encima del otro. Cada uno de estos bloques está formado por una serie de subcapas. Una subcapa implementa una función, como una capa oculta de una red neuronal, junto con conexiones residuales [21] y una normalización de capa [22]. A continuación se describirán cada una de estas técnicas.

2.4.1. Generalización del mecanismo de atención

Una función de atención se puede describir como la asignación de una consulta y un conjunto de pares clave-valor a una salida, donde la consulta, las claves, los valores y la salida son todos vectores. Habiendo introducido previamente la fórmula para el modelo de atención (véase la ecuación 2.28), es necesario recordar que, en una atención convencional, se usa $\mathbf{q} = \mathbf{s}_{i-1}$, $\mathbf{K}_j = \mathbf{h}_j$ y $\mathbf{V}_j = \mathbf{h}_j$.

Si deseamos calcular la atención para múltiples consultas, estas se pueden empaquetar en una matriz $\mathbf{Q}$, y el cálculo se puede expresar en términos de multiplicación de matrices. Esto significa que el proceso de atención se calcula como:

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \tag{2.30}$$

donde d_k es la dimensión de la clave.

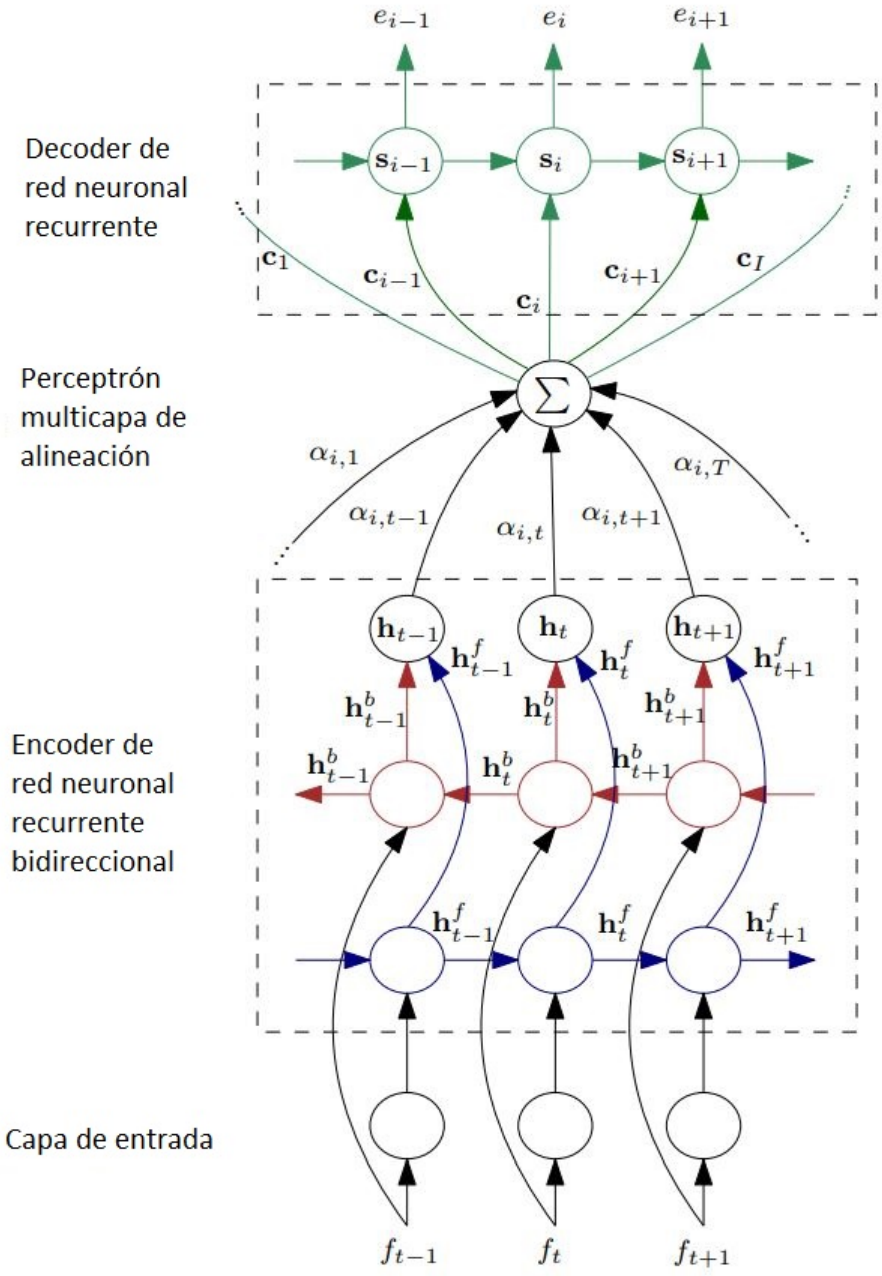

Figura 2.6: Arquitectura Encoder-Decoder con un modelo de alineación.

Hasta ahora, el mecanismo de Atención ha proporcionado una respuesta para cada consulta. El modelo Transformer introduce la atención *Multi-Head*, que amplía el mecanismo anterior para producir una respuesta que es la combinación de múltiples comparaciones clave-consulta. La atención *Multi-Head* consiste en realizar varias operaciones de atención en paralelo y combinar los resultados para obtener el vector de contexto final. Cada operación de atención individual o *head* se lleva a cabo aplicando una proyección lineal a la consulta, claves y valores, calculando la atención entre ellos y luego proyectando el resultado a un espacio común:

$$MultiHead(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = Concat(head_1, ..., head_h)\mathbf{W}^o \tag{2.31}$$

$$head_i = Attention(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \tag{2.32}$$

Las proyecciones se aplican mediante las medias de las matrices $\mathbf{W}^o$, $\mathbf{W}_i^Q$, $\mathbf{W}_i^K$ y $\mathbf{W}_i^V$. Estas matrices de proyección son parámetros aprendidos durante el entrenamiento. Una diferencia adicional de estas capas con respecto a la atención convencional es la forma en que se aplican en el modelo Transformer, algo que será explicado una vez que se describa la arquitectura.

2.4.2. Normalización de capa

Consiste en normalizar las salidas de las capas de la red neuronal, de modo que cada neurona se comporte siguiendo una distribución normal. Para normalizar una capa, se calcula la desviación típica y la media de las activaciones de cada neurona en esa capa, para posteriormente restar la media y dividir por la desviación típica. Esta normalización se aplica a la salida de las operaciones de la capa, $\mathbf{a}^{(i)}$, antes de que la función de activación $g(\cdot)$) sea aplicada. Por lo tanto, podemos aplicar la normalización a la capa i de la siguiente manera:

$$LayerNorm(\mathbf{a}^{(i)}) = \frac{\gamma}{\sigma^{(i)}} \odot (a^{(i)} - \mu^{(i)}) + \beta \tag{2.33}$$

$$\mu^{(i)} = \frac{1}{H}\sum_{h=1}^{H} a_h^{(i)} \tag{2.34}$$

$$\sigma^{(i)} = \sqrt{\frac{1}{H}\sum_{h=1}^{H}(a_h^{(i)} - \mu^{(i)})^2} \tag{2.35}$$

donde $\mu^{(i)}$ y $\sigma^{(i)}$ son, respectivamente, la media y la desviación típica, y tienen la misma dimensión que $\mathbf{a}^{(i)}$. γ y β son los parámetros aprendidos durante el entrenamiento, y se utilizan para que la capa pueda generar distribuciones normales que son diferentes de la distribución normal estándar.

2.4.3. Conexiones residuales

Es una técnica en la que la entrada de una capa omite una o más transformaciones y luego se agrega a la salida de esas transformaciones. Si suponemos que tenemos un vector **x** y una serie de capas que calculan la transformación **F(x)**, entonces la salida de ese bloque residual se calcularía como **x** + **F(x)**. El objetivo de esta técnica es facilitar el proceso de optimización para redes con muchas capas.

2.4.4. Posición de las palabras

Las capas de atención, por sí mismas, no ofrecen la posibilidad de conocer el orden de las palabras en una frase, ya que tratan todas las entradas de la misma manera. Un enfoque simple para este problema es codificar la posición de la palabra por medio de un vector *one-hot* p_i, de la misma manera que lo hacemos con las diferentes palabras que forman el vocabulario, w_i. Dada una matriz de *embedding* arbitraria $\mathbf{E}$, obtenemos la representación concatenando los vectores y multiplicándolos por dicha matriz:

$$e_n = [w_n; p_n]\mathbf{E} \tag{2.36}$$

$$e_n = w_n\mathbf{E}_w + p_n\mathbf{E}_p \tag{2.37}$$

donde E_w es la matriz para los *word-embeddings* y E_p es la matriz para las posiciones. Es decir, esta formulación es equivalente a tener 2 matrices de *embeddings* diferentes.

En lo que respecta a obtener las codificaciones para las posiciones, el enfoque sencillo es tratarlas como los *word-embeddings* y dejar que la red las aprenda durante el entrenamiento. Otra opción es utilizar una función predefinida que produzca estas codificaciones de posición.

En el caso del Transformer, las codificaciones de posición vienen dadas por dos funciones seno y coseno, calculadas de manera diferente para cada posición *pos* de la frase, $\forall pos \in [1, N]$. Estas funciones calculan un vector de *embedding* posicional para cada palabra cuyas entradas se calculan de la siguiente manera:

$$PE_{2i} = sin\left(pos\frac{1}{10000^{2i/d_{model}}}\right) \forall i \in [0, d_{model}/2 - 1] \tag{2.38}$$

$$PE_{2i+1} = cos\left(pos\frac{1}{10000^{2i/d_{model}}}\right) \forall i \in [0, d_{model}/2 - 1] \tag{2.39}$$

Los experimentos llevados a cabos en el artículo original [20] muestran que estas codificaciones posicionales fijas no conllevan una disminución del rendimiento, y ofrecen una capacidad de generalización ligeramente mayor que los *embeddings* aprendidos si tenemos que traducir frases que son más largas que las que aparecen en los datos de entrenamiento.

2.4.5. Arquitectura

En el modelo de Transformer, las conexiones residuales y la normalización de capas se combinan, y la salida de cada capa se calcula como *LayerNorm(x + Sublayer(x))*. Hay dos tipos de subcapas en la arquitectura Transformer, las capas *Feed Forward* y las capas de atención *Multi-Head*, previamente descritas. Las capas *Feed Forward* son capas de redes neuronales estándar que consisten en una multiplicación de matrices de pesos seguida de una función de activación.

Hasta ahora, las técnicas y capas descritas se pueden aplicar a cualquier modelo de traducción neuronal, pero el cambio en la arquitectura Transformer es cómo se aplican estas capas. Además, esta arquitectura, o al menos una versión modificada de esta, se empleará en la tarea de comprensión de texto para búsqueda de respuestas, tal y como veremos más adelante. La figura 2.7 muestra un bloque que el Transformer utiliza como encoder.

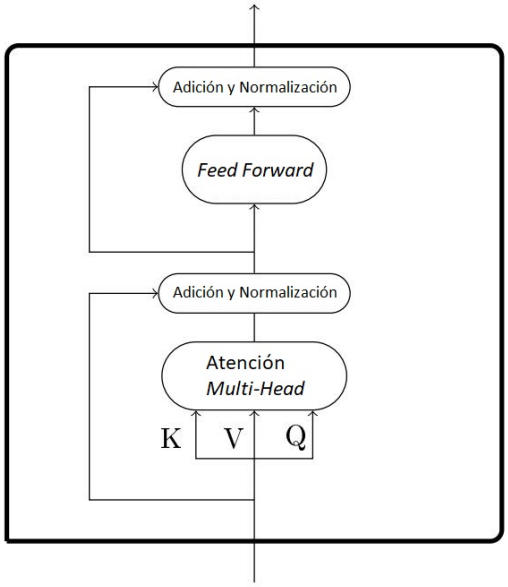

Figura 2.7: Bloque encoder del Transformer.

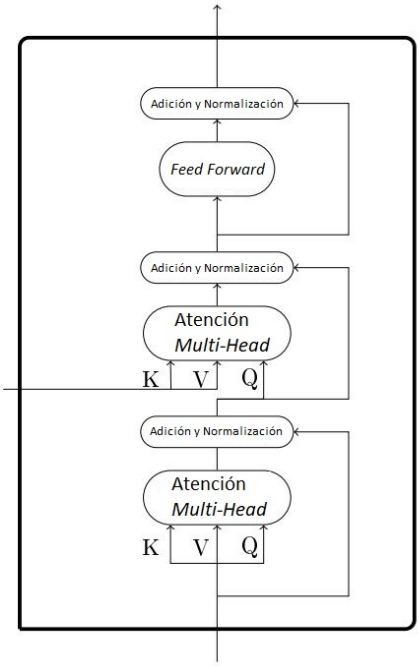

Figura 2.8: Bloque decoder del Transformer.

El bloque estándar del Transformer consiste en una subcapa de atención *Multi-Head* seguida de una subcapa *Feed Forward*. En lugar de alimentar una entrada a la vez como una red recurrente, la frase de entrada completa se envía al encoder y la representación de la oración de entrada se calcula toda al mismo tiempo. Las capas de atención en el encoder aplican la atención *Multi-Head* a la salida de la capa anterior, utilizando esa salida tanto como consulta como pares clave-valor. Por tanto, en cada capa, el encoder produce una representación para cada palabra, que puede incorporar información sobre cualquier otra palabra de la oración gracias al mecanismo de atención. Por lo tanto, la representación completa se puede producir en una sola pasada.

La figura 2.8 muestra un bloque que el Transformer utiliza como decoder. En comparación con los bloques del encoder, los bloques del decoder incluyen una subcapa de atención *Multi-Head* adicional que atiende la salida del encoder, lo que permite que el decoder acceda a las representaciones la oración de entrada. Al igual que el resto de modelos de traducción neuronal, el decoder produce una palabra a cada instante de tiempo, condicionada por las palabras emitidas previamente. El decoder es alimentado por las palabras que se hayan emitido hasta el momento y sus subcapas de atención *Multi-Head* realizan sus cálculos sobre la salida de la anterior capa del decodificador. En esas subcapas, la salida de la anterior capa del decodificador actúa como consulta, mientras que la salida del encoder actúa como par clave-valor.

En la figura 2.9 se puede apreciar la arquitectura completa del Transformer. Aunque la configuración puede variar, el modelo base del Transformer cuanta con 6 bloques encoder/decoder, una dimensión de *embedding* de 512, capas ocultas de tamaño 2048 y 8 *head* para el método de atención.

Hasta ahora, se han descrito los cambios introducidos por el modelo Transformer en los componentes del encoder y el decoder. Sin embargo, en el artículo en el que se presenta el Transformer también se introducen algunas consideraciones adicionales que se utilizan para entrenar el modelo que no están relacionadas con la arquitectura en sí y pueden considerarse de forma independiente.

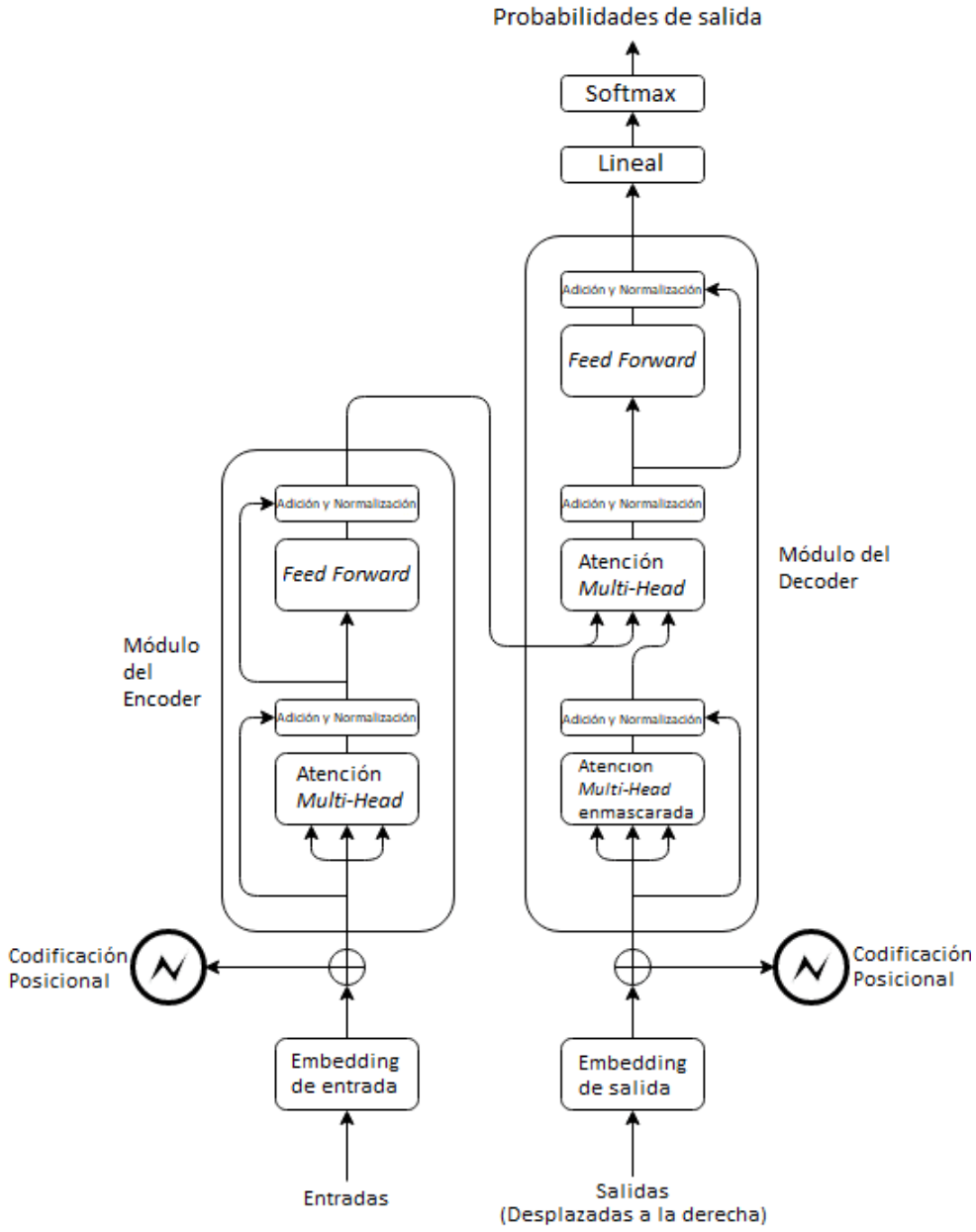

Figura 2.9: Arquitectura completa del Transformer.

2.4.6. Unión de pesos

Weight-tying [23] es una técnica propuesta para mejorar el rendimiento de los modelos de lenguaje de las redes neuronales mediante la manipulación de las diferentes capas de *embedding*, al mismo tiempo que proporciona una reducción significativa en el número de parámetros del modelo. Como se ha explicado previamente, los modelos de traducción neuronal utilizan matrices de *embeddings* para convertir el texto de entrada en un formato numérico apto para la red. En los sistemas Encoder-Decoder, tenemos 3 de esas matrices. Primero tenemos una matriz *embedding* del encoder, $\mathbf{E}_e$, que produce una representación para ser utilizada por el encoder a partir de los vectores *one-hot* de la frase de origen. En esta matriz, cada una de sus entradas se puede considerar como una representación codificada de la palabra de entrada correspondiente. Hay autores que han analizado la posibilidad de sustituir esta matriz aprendida durante el entrenamiento por algún método de *word-embedding* diferente, generalmente uno que haya sido previamente calculado por un modelo cuya tarea específica sea la de encontrar estos *word-embeddings*. Por ejemplo, un estudio [24] analizó los efectos de utilizar diferentes tipos de *embeddings* preentrenados.

Sin embargo, esa no es la única matriz de *embedding* utilizada en traducción neuronal. Dado que el decoder depende de y_{t-1}, también utilizamos una matriz *embedding* para el decoder, $\mathbf{E}_d$. Además, la salida de la última capa oculta del decoder debe proyectarse en un espacio que tenga la misma dimensión que el vocabulario de salida, de modo que obtengamos una distribución de probabilidad sobre todas las palabras posibles una vez que apliquemos la función *softmax*. Esto se logra mediante la multiplicación de la matriz *embedding* de salida, $\mathbf{E}_o$. Con todo eso, tenemos 3 matrices de *embedding* que caracterizan nuestros modelos. $\mathbf{E}_e$, $\mathbf{E}_d$ y $\mathbf{E}_o$ son el encoder, el decoder y las matrices de *embedding* de salida, respectivamente.

$\mathbf{E}_e$ tiene dimensión (dimensión del *embedding*, tamaño del vocabulario de entrada), $\mathbf{E}_d$ tiene dimensión (dimensión del *embedding*, tamaño del vocabulario de salida) y $\mathbf{E}_o$ tiene dimensión (tamaño del vocabulario de salida, tamaño de la capa oculta). Los autores de [23] se dieron cuenta de que, si tuviéramos el mismo vocabulario de entrada y salida, es decir, la misma dimensión, los *embeddings* del encoder y el decoder podrían realizarse mediante una única matriz con la misma dimensionalidad. Una vez que tenemos *embeddings* del encoder y el decoder compatibles, podemos ver que la traspuesta del *embedding* de salida, $\mathbf{E}_o^T$ también tiene la misma dimensión que las otras 2 matrices, siempre que la dimensión del *embedding* y el tamaño de la capa oculta sean también iguales. Por tanto, el *Weight-tying* consiste en juntar estas 3 matrices para que su trabajo sea realizado por una única matriz, considerando que $\mathbf{E}_e = \mathbf{E}_d = \mathbf{E}_o^T$. Experimentos realizados han demostrado que podemos llevar a cabo esta vinculación sin perder rendimiento, ya que *embedding* de salida comparte las mismas propiedades que el *embedding* de entrada al representar palabras parecidas de manera similar. Esta técnica tiene dos ventajas principales sobre el uso de diferentes *embeddings*. Por un lado, permite que las filas de la entrada del *embedding* se actualicen en cada paso de entrenamiento, en lugar de solo cuando su palabra correspondiente aparece en la entrada. Esto ayuda al modelo a entrenar más rápido. Por otro lado, y aun más importante, reduce enormemente la cantidad de parámetros que debe aprender el modelo. Podemos observar esto comparando el tamaño de las matrices ocultas con el tamaño de las matrices de *embedding*. Mientras que es habitual tener una dimensión de capa oculta de 1024 (lo que nos da matrices de dimensión 1024x1024), es muy común que el vocabulario contenga 20000 palabras o más, lo que se traduciría en matrices de *embedding* de dimensión 1024x20000. Es habitual ver modelos donde la mayoría de parámetros forman parte de las diferentes matrices de *embedding*. Si podemos usar solo una matriz de *embedding* en lugar de 3, logramos una reducción muy significativa en el número de

parámetros, con los beneficios asociados en el rendimiento del entrenamiento y el ahorro en memoria.

El modelo del Transformer utiliza esta técnica para obtener las ventajas mencionadas en términos de rendimiento y reducción de parámetros.

2.4.7. Suavizado de etiquetas

Cuando entrenamos un modelo de redes neuronales, generalmente se optimiza una función de coste J(θ) que depende de los parámetros del modelo (θ). Actualizamos los pesos utilizando el descenso del gradiente estocástico y procesando un lote de muestras en cada iteración de tiempo. Para un lote de N muestras, con y_n como salida de la red para esa muestra, la función de coste, generalmente, se define como el promedio de una función de pérdida aplicada a cada muestra:

$$J(\theta) = \frac{1}{N}\sum_{n=1}^{N} L(y_n, \hat{y}_n) = \frac{1}{N}\sum_{n=1}^{N} L(y_n, f(x_n;\theta)) \tag{2.40}$$

La elección de la función de pérdida es, por tanto, una de las decisiones que se deben tomar al entrenar un modelo que utiliza el descenso por gradiente. En los problemas de clasificación, la elección más común para la función de pérdida es la función de entropía cruzada H, que calcula una medida de disimilitud entre dos distribuciones de probabilidad, p y q. Si las distribuciones son discretas, esta función se puede calcular, para toda posible clase c, como:

$$H(p,q) = \sum_{c} p(c) log(q(c)) \tag{2.41}$$

Cuando trabajamos con problemas de clasificación, ya se ha establecido que se desea obtener un sistema que genere una distribución de probabilidad. Por tanto, la distribución emitida por la red p_{red} ocupará el lugar de la distribución q utilizada para calcular la entropía cruzada. La etiqueta, o la clase asignada a la muestra, también se puede considerar como una distribución de probabilidad p_{etiq}. Esto, generalmente, se hace estableciendo la probabilidad de la etiqueta a 1, por lo que $p(y) = 1$, y el resto de posibles valores tendrán una probabilidad de 0, de modo que $p(z) = 0, \forall z \neq y$. Habiendo obtenido estas distribuciones de probabilidad, la entropía cruzada se puede aplicar como función de pérdida L de la siguiente manera:

$$L(y, \hat{y}) = -\sum_{c} p_{etiq}(y = c|x) log(p_{red}(\hat{y} = c|x)) \tag{2.42}$$

En el caso de un sistema de traducción, para una sola muestra de entrenamiento y de longitud I, denotaremos el símbolo i-ésimo de la frase objetivo como y_i. Así, la pérdida se calcula como:

$$L(y, \hat{y}) = -\sum_{i=1}^{I}\sum_{c} p_{etiq}(y_i = c|x) log(p_{red}(\hat{y}_i = c|x)) \tag{2.43}$$

Debido a la forma en que se ha definido la distribución de probabilidad dada por la etiqueta, toda la masa de probabilidad se da a un solo valor y, al calcular la entropía cruzada, el término de la distribución de la etiqueta puede ser ignorado en todos menos uno de los posibles valores de salida. El cálculo, pues, se puede simplificar a:

$$L(y, \hat{y}) = log(p_{red}(\hat{y} = y|x)) \tag{2.44}$$

Si bien la simplificación anterior tiene sentido para algunas de las aplicaciones generales del reconocimiento de formas, no está del todo claro que este sea el mejor enfoque posible para la traducción automática. Si, en la oración destino, se describe algo como 'bueno', no sería del todo correcto asumir que esa palabra es la única traducción correcta, ya que 'bueno', en un determinado contexto, puede adquirir otro significado.

La técnica del suavizado de etiquetas [25] tiene como objetivo mejorar la capacidad de generalización del modelo al reducir la confianza que este debe asignar a las predicciones de entrenamiento. Esta técnica consiste en suavizar la distribución *one-hot* de una etiqueta. Esto funciona al reservar una cierta cantidad de masa de probabilidad para las etiquetas incorrectas y redistribuir esta probabilidad de alguna forma.

Hasta ahora, la distribución de probabilidad de una etiqueta se calculaba como:

$$p_{etiq}(y = c|x) = \delta_{y,c} \tag{2.45}$$

donde $\delta_{y,c}$ es la función delta de Dirac, en la que su valor es 1 si $y = c$ y 0 en caso contrario.

El suavizado de etiquetas introduce la siguiente modificación, donde se descuenta una masa de probabilidad ϵ y se distribuye entre todos los valores de etiqueta posibles C mediante la distribución de probabilidad uniforme:

$$p'_{etiq}(y = c|x) = (1 - \epsilon)\delta_{y,c} + \frac{\epsilon}{C} \tag{2.46}$$

Además de ayudar al modelo a confiarse demasiado en sus predicciones debido a que está sobreajustado a los datos de entrenamiento, esto puede tener efectos beneficiosos adicionales durante la decodificación. Se cree que la distribución de suavizado ayuda al modelo durante la decodificación al realizar una poda menos estricta de hipótesis parciales, lo que puede resultar en una mejor traducción general, una vez que el proceso de decodificación haya terminado.

2.5 Conclusiones

De cara al sistema de traducción automática que se realizará para este trabajo, y habiendo visto las diferentes aproximaciones que se han explicado en el presente capítulo, nuestros sistemas propuesto están basados en la arquitectura de encoder-decoder con mecanismo de atención propuesta en [19] y el modelo de Transformer propuesto en [20], los cuales nos han sido proporcionado por el toolkit NMT-Keras [58]. Más adelante, en los capítulos 4 y 5, entraremos en detalles sobre las configuraciones que hemos utilizado.

CAPÍTULO 3
Comprensión de texto para tareas de búsqueda de respuestas

La búsqueda de respuestas es un tipo de recuperación de información en el que, dada una determinada cantidad de documentos (o contextos, como se suele expresar de manera más formal), el sistema ha de ser capaz de responder a preguntas planteadas sobre dichos documentos.

La realización de un sistema de búsqueda de respuestas completamente funcional es un problema que ha sido bastante popular entre los investigadores del campo del procesamiento del lenguaje natural. Los nuevos algoritmos, especialmente aquellos basados en el aprendizaje profundo, han logrado un progreso decente en la clasificación de texto e imágenes. A pesar de ello, estos sistemas aún no han logrado resolver las tareas que involucran el razonamiento lógico. El problema de respuesta a preguntas dado un contexto es un buen ejemplo de estas tareas.

Una implementación muy común de un sistema de búsqueda de respuestas es un chatbot, ya que su principal función es justamente construir una respuesta cuando se le realiza una consulta. El bot ALICE [26] es un ejemplo de esta implmentación, que fue desarrollado usando AIML [27], un lenguaje de programación basado en XML y diseñado específicamente para ayudar en la creación de chatbots.

Por aquella época, se lanzaron muchas aplicaciones similares, pero el mayor avance fue cuando se utilizó el procesamiento del lenguaje natural para resolver la tarea de búsqueda de respuestas [28]. Esta solución demostró ser una mejora importante en la precisión de los sistemas y, desde entonces, todos los modelos han intentado basarse únicamente en el procesamiento del lenguaje natural. En los últimos años, este tipo de soluciones se han visto eclipsadas por redes neuronales profundas, ya que tienden a producir mejores resultados.

Los sistemas de preguntas y respuestas son muy útiles, ya que permiten a los usuarios ingresar una consulta basada en algunos hechos o historias y el sistema intenta usar el contexto existente en ellos para responder las preguntas de forma efectiva. Además, la mayoría de los problemas en el aprendizaje automático y el procesamiento del lenguaje natural se pueden modelar como un problema de respuesta a una pregunta. Por ejemplo, la tarea de resumir textos puede modelarse como una tarea de respuesta a preguntas en el sentido de que si el usuario preguntase '¿Cual es el resumen de este texto?', el sistema puede responder proporcionando el resumen apropiado.

3.1 Arquitectura de un sistema de búsqueda de respuestas

En los sistemas de búsqueda de respuestas, encontrar una respuesta a una pregunta devolviendo un pequeño fragmento de un texto es diferente de la tarea de recuperación de información (*Information Retrieval*, abreviado como IR) o de la tarea de extracción de información (*Information Extraction*, abreviado como IE). Los sistemas de IR permiten localizar documentos completos que pueden contener la información pertinente, dejando que el usuario extraiga la respuesta de una lista ordenada de textos. Por el contrario, los sistemas de IE extraen la información de interés, siempre que se haya presentado en una representación objetivo predefinida, es decir, una plantilla. La solución inmediata de combinar técnicas de IR e IE para búsqueda de respuestas no es práctica, ya que los sistemas de IE dependen en gran medida del conocimiento del dominio y, además, la generación de plantillas no se realiza automáticamente.

En los sistemas de búsqueda de respuestas, generalmente, primero se realiza un procesamiento de la pregunta combinando información sintáctica, resultante de un análisis poco profundo, con información semántica que caracteriza la pregunta. Después, la búsqueda de la respuesta se basa en una nueva forma de indexación, denominada indexación de párrafos y en nuevos métodos de recuperación relacionados. Finalmente, para extraer las respuestas y evaluar su corrección, se usan técnicas basadas en métodos empíricos y en información léxico-semántica.

De esta forma, y gracias al análisis de algunas aproximaciones relevantes en este campo [29] [30] [31], los principales componentes de un sistema de búsqueda de respuestas se pueden identificar de la siguiente forma:

- Procesamiento y análisis de la pregunta.
- Selección de documentos o pasajes.
- Extracción de respuestas.

Estos componentes se relacionan entre sí procesando preguntas y documentos en diferentes niveles hasta obtener la respuesta. En la figura 3.1 se puede apreciar la secuencia de ejecución de estos procesos.

Las preguntas formuladas al sistema son procesadas inicialmente por el módulo de análisis de la pregunta. Este realiza dos tareas principales: Detectar el tipo de información que la pregunta espera como respuesta (fecha, lugar, etc.) y seleccionar aquellos elementos de la pregunta que van a permitir la localización de los documentos que puedan contener la respuesta. Esto es de vital importancia ya que de la calidad de la información extraída depende en gran medida el rendimiento de los otros módulos y, por consiguiente, también el del sistema.

Una parte de la información que se obtiene como resultado del análisis de la pregunta es utilizada por el módulo de recuperación de documentos para realizar una primera selección de textos. Dado el gran volumen de documentos a tratar por estos sistemas y las limitaciones de tiempo de respuesta con las que trabajan, esta tarea se realiza empleando sistemas de recuperación de información, generalmente orientada a la detección de extractos de texto más reducidos que el documento completo. El resultado obtenido es un subconjunto muy reducido de la base de datos documental sobre el que se afrontará la extracción de la respuesta.

Finalmente, el módulo de extracción de respuestas se encarga de realizar un análisis más detallado del subconjunto de textos relevantes resultado del proceso anterior, con la finalidad de localizar y extraer la respuesta buscada.

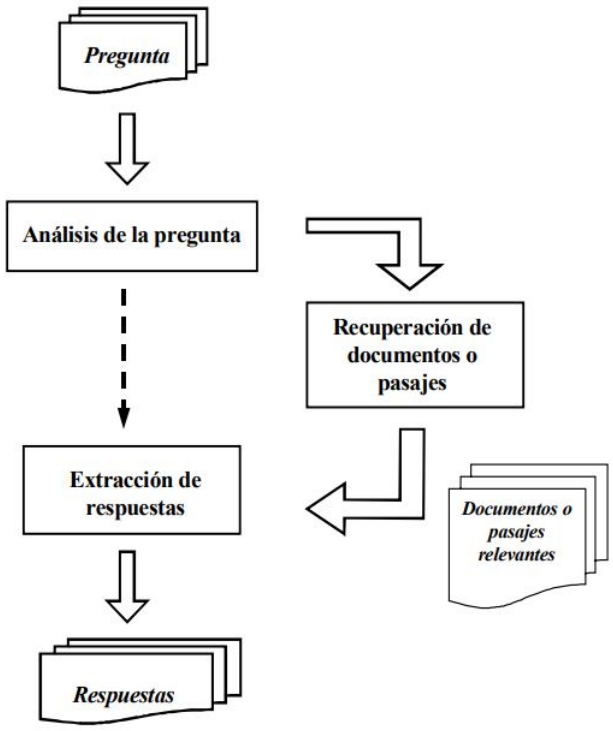

Figura 3.1: Arquitectura simple de un sistema de Búsqueda de Respuestas.

3.2 Dimensiones del problema

Para responder una determinada pregunta, un sistema debe analizarla en un contexto correspondiente, ha de consultar una serie de recursos para localizar la respuesta y tiene que presentarla de manera adecuada al usuario.

El análisis de cada uno de los siguientes procesos da lugar a una serie de aspectos o dimensiones del problema que resulta conveniente abordar de forma previa a su resolución: Usuarios, preguntas, respuestas y nivel de conocimiento necesario o requerido.

3.2.1. Usuarios

Probablemente, el grado de satisfacción de diferentes usuarios ante el mejor sistema de búsqueda de respuestas será totalmente variable en función de las expectativas de cada uno de ellos.

Podemos encontrar un amplio espectro de usuarios que requieren diferentes capacidades del sistema para satisfacer sus necesidades de información. Estas necesidades pueden variar entre las solicitadas por un usuario casual, que interroga al sistema para la obtención de datos puntuales, y las que puede necesitar un analista profesional. Estos tipos representan los extremos de ese amplio abanico de usuarios potenciales de un sistema de búsqueda de respuestas.

Se pueden clasificar los diferentes usuarios de un sistema de búsqueda de respuestas en cuatro tipos generales en función de la complejidad de sus requerimientos:

- **Usuario casual**. Este tipo de usuario necesita información puntual acerca de hechos concretos. Realiza preguntas cuya contestación se puede hallar en un docu-

mento expresada, generalmente, de forma simple: '¿En qué año nació Roosevelt?' o '¿Cuántos habitantes tiene Rusia?'.

- **El recopilador de información**. Este usuario realiza preguntas cuya respuesta necesita de un proceso de recopilación de varias instancias de información indicadas en la pregunta. Por ejemplo: '¿Qué países componen la actual Unión Europea?' o 'Dime los principales datos biográficos de Winston Churchill'. Como se puede apreciar, este tipo de preguntas requiere la localización de varias informaciones (probablemente en diferentes documentos) y su posterior combinación como respuesta definitiva.

- **El periodista**. Supongamos un periodista al que se le ha encargado la redacción de un artículo relacionado con un evento determinado, por ejemplo, un terremoto en la zona de Carolina del Sur. Para ello, el reportero necesitaría recabar, por un lado, datos concretos del suceso (intensidad del terremoto, lugar del epicentro, daños materiales, ...) así como informaciones anteriores más o menos relacionadas que permitan enmarcar el suceso en un contexto adecuado (terremotos anteriores en la zona, estudios sismológicos previos, predicciones, ...). En ambos casos, el sistema necesitaría tener en cuenta el contexto de la serie de preguntas que el usuario le interpondrá. Este contexto permitiría al sistema determinar la amplitud de la búsqueda y la necesidad de profundizar en determinados aspectos del mismo.

- **El analista profesional**. El perfil de este usuario corresponde con el de un consumidor profesional de información experto en temas concretos. Por ejemplo, analistas financieros, personal de organismos estatales especializados en política internacional, tráfico de drogas, etc. Un sistema de que trabaje a este nivel debe poder aceptar preguntas muy complejas cuyas respuestas pueden basarse en conclusiones y decisiones realizadas por el propio sistema. Estas respuestas necesitaría de la recopilación y síntesis de información obtenida en diferentes fuentes y debería ser presentada al usuario de una manera adecuada a su forma de trabajo. Además, este sistema debería de disponer de potentes herramientas de navegación multimedia que permitieran no sólo revisar la respuesta propuesta por el sistema a través de todo el proceso de su obtención (revisión de la información de soporte, interpretaciones, conclusiones y decisiones realizadas) sino también facilitar la interacción con el usuario en cada uno de esos procesos. Esta interactividad daría como resultado una respuesta conjunta entre el sistema y el analista.

3.2.2. Preguntas

La experiencia demuestra que resulta difícil determinar cuales son las características que hacen que unas preguntas resulten más difíciles de contestar que otras. Esta circunstancia hace inviable una clasificación general de las mismas desde este punto de vista. Es por eso que la clasificación aceptada de forma general se basa en el tipo de respuesta que han de obtener:

- **De hechos concretos**. Requieren como respuesta uno o varios datos muy específicos como fechas, nombre de entidades, cantidades, etc.

- **De resumen**. Necesitan localizar instancias de información relacionadas con la pregunta y resumirlas para devolverlas al usuario.

- **De opinión**. Corresponden a preguntas muy complejas que requieren de la recopilación de datos y de la aplicación de técnicas de deducción en base a ellos. Un

ejemplo sería '¿Qué pasaría si mañana se demostrase que la vacuna que Rusia propone contra el COVID-19 funciona?'.

Esta clasificación resulta muy importante desde el punto de vista de la efectividad del proceso de búsqueda de respuestas. Esto se debe a que un correcto análisis de la pregunta permitiría reducir considerablemente el espacio de respuestas posibles a considerar por el sistema en el proceso de localización de la misma [32].

3.2.3. Respuestas

La forma de las respuestas suministradas por un sistema de búsqueda de respuestas puede ser variada y está íntimamente relacionada tanto con el tipo de pregunta como con el tipo de usuario del sistema.

Desde el punto de vista de su extensión, éstas pueden ser cortas, como respuestas a preguntas de hechos concretos (el nombre de un país o de una entidad) o largas, si son respuestas a preguntas de opinión y/o resumen.

Desde la perspectiva del usuario, estas respuestas pueden ir acompañadas de los documentos y de los criterios de selección empleados que justifiquen la respuesta. De esta forma, el usuario puede validar la corrección de las respuestas suministradas por el sistema.

Por otra parte, existen diferentes técnicas relacionadas con la forma de construcción de la respuesta: la extracción y la generación. La extracción consiste en seleccionar uno o varios extractos de los documentos analizados y presentarlos como respuesta tal y como aparecen en los documentos originales. Por otra parte, la generación consiste en elaborar una presentación coherente de la respuesta a partir de la información original localizada en los documentos. En este caso, necesitaríamos la aplicación de técnicas de generación de lenguaje.

3.2.4. Conocimiento necesario

El poder contemplar con éxito el desarrollo de sistemas de búsqueda de respuestas que soporten los diferentes niveles de la tipología de usuarios explicada previamente, necesita de un incremento progresivo del nivel de conocimiento utilizado por estos sistemas.

Se puede estructurar este conocimiento en cuatro niveles en función de la necesidad de su participación para afrontar preguntas de creciente complejidad. Cada nivel incluiría el conocimiento de los niveles anteriores:

- **De hechos concretos**. Corresponde al nivel mínimo exigido en un sistema de búsqueda de respuestas. Este conocimiento permite la contestación de preguntas cuya respuesta es un hecho concreto que bien puede ser el nombre de una persona u organización, una cantidad, un lugar o una fecha. Las bases de conocimiento utilizadas pueden estar formadas por diccionarios o enciclopedias.

- **Explicativo**. Este nivel de conocimiento ha de permitir que el sistema responda a preguntas más complejas en las que la respuesta constituye la explicación, justificación o causa de un suceso. En este caso, las bases de conocimiento utilizadas pueden estar formadas por ontologías y bases de conocimiento léxico-semánticas como WordNet.

- **Modal**. Se necesita un mayor nivel de conocimiento para que un sistema pueda afrontar preguntas de opinión y resumen: '¿Qué pasaría si se demostrase que la vacuna que Rusia propone contra el COVID-19 funciona?' La respuesta a esta pregunta se obtendría dentro de un dominio específico, por ejemplo, la evaluación de las posibles consecuencias en el campo de la medicina o las repercusiones políticas y económicas de ese suceso. El tipo de conocimiento requerido para realizar este análisis vendría representado por lo que se conoce como *bases de conocimiento de alto rendimiento*. Estas bases estarían formadas por ontologías restringidas al dominio de la pregunta junto con axiomas particulares y estrategias genéricas de solución de problemas asociados a dicho dominio.
- **General del mundo**. Un amplio conocimiento general del mundo permitiría al sistema procesar preguntas del tipo anterior pero sin limitar el dominio de aplicación. De hecho, el sistema podría ser capaz de 'descubrir' nuevo conocimiento relacionado con la pregunta, 'aconsejar' y 'justificar' los motivos de dicha relación e incluso facilitar al usuario la posibilidad de interactuar con el sistema para dirigir el proceso de generación de la respuesta en función del descubrimiento de información relacionada.

3.3 Clasificación de sistemas de búsqueda de respuestas

La mayoría de sistemas de búsqueda de respuestas afrontan la tarea desde la perspectiva del usuario casual. Es decir, un usuario que realiza preguntas simples que requieren un hecho, situación o dato concreto como contestación. Estos sistemas utilizan como fuente de información una base de datos textual compuesta por documentos escritos en un único lenguaje. El conocimiento utilizado en estos sistemas corresponde también con el nivel mínimo detallado anteriormente (de hechos concretos). En algunos casos se ha avanzado un poco más mediante el uso de bases de datos léxico-semánticas (principalmente WordNet) y la integración de algún tipo particular de ontología como SENSUS [33] o Mikrokosmos [34].

Clasificar los sistemas existentes resulta una tarea bastante complicada. Esta dificultad radica principalmente en la selección de la perspectiva desde la que se desea realizar dicha clasificación y en la gran variedad de aproximaciones existentes.

En primer lugar, se presenta una clasificación que tiene en cuenta la situación actual de los sistemas de búsqueda de respuestas en el ámbito de una perspectiva general. En segundo lugar, y con la intención de profundizar en las diferentes aproximaciones existentes, se hace una clasificación en función del nivel de análisis del lenguaje natural que estos sistemas utilizan.

3.3.1. Perspectiva general

La taxonomía presentada en [35] clasifica los sistemas de búsqueda de respuestas en función de tres criterios:

- Las bases de conocimiento empleadas.
- El nivel de razonamiento requerido.
- Las técnicas de indexación y de procesamiento del lenguaje natural utilizadas.

Las bases de conocimiento y los sistemas de razonamiento proporcionan el medio que facilita la construcción del contexto de la pregunta y la búsqueda de la respuesta en los

documentos. Por otro lado, las técnicas de indexación permiten localizar aquellos extractos de documentos en los que pueden aparecer las respuestas. Por último, las técnicas de procesamiento del lenguaje natural proporcionan el entorno general que permite la localización y extracción de dichas respuestas de forma precisa.

3.3.2. Sistemas que no utilizan técnicas de procesamiento del lenguaje natural

Estos sistemas emplean técnicas de recuperación de información (RI) adaptadas a la tarea de búsqueda de respuestas. La forma general de proceder de estos sistemas se basa en la recuperación de extractos de texto relativamente pequeños con la suposición de que dichos extractos contendrán la respuesta esperada.

Generalmente, el análisis de la pregunta consiste en seleccionar aquellos términos de la pregunta que deben aparecer cerca de la respuesta. Para ello, se eliminan las palabras de parada y se seleccionan aquellos términos con mayor 'valor discriminatorio' (palabras clave). Estos términos se utilizan para recuperar directamente fragmentos relevantes de texto que se presentan directamente como respuestas [36] o bien, para recuperar documentos que posteriormente serán analizados. Este análisis consiste en dividir el texto relevante en ventanas de un tamaño inferior o igual a la longitud máxima permitida como cadena respuesta. Cada una de estas ventanas se valora en función de determinadas heurísticas para finalmente presentar como respuestas aquellas ventanas que consiguen la mejor puntuación. Esta valoración suele tener en cuenta aspectos como el valor de discriminación de las palabras clave contenidas en la ventana, el orden de aparición de dichas palabras en comparación con el orden establecido en la pregunta, etc.

Se pueden incluir en este grupo los sistemas utilizados por la universidad de Massachusetts [37] y los laboratorios RMIT/CSIRO [38].

El rendimiento alcanzado por este tipo de sistemas es relativamente bueno cuando la longitud permitida como respuesta es grande (del orden de 250 caracteres o más), sin embargo, decrece mucho cuando se requiere una respuesta corta y precisa.

3.3.3. Sistemas que emplean técnicas de análisis superficial

Estos sistemas se caracterizan, en primer lugar, por la realización de un análisis detallado de la pregunta que permite extraer y representar aquella información que será de utilidad en las sucesivas fases del proceso. De forma general, este proceso permite obtener la siguiente información:

- El tipo de entidad que la pregunta espera como respuesta (el nombre de alguien, un lugar, etc).
- Restricciones y características adicionales relacionadas con el tipo de respuesta esperada. Por un lado, están los términos de la pregunta que permiten la recuperación de aquellas partes del texto sospechosas de contener la respuesta. Por otro lado, están las relaciones (sintácticas o semánticas) que deben aparecer entre las entidades de la pregunta y la respuesta a encontrar.

Para obtener el tipo de respuesta es necesario que estas entidades se organicen en clases semánticas, del estilo de 'persona', 'organización', 'tiempo', 'lugar', etc. Para identificar el tipo de respuesta esperada se suele realizar un análisis de los términos interrogativos de la pregunta. Por ejemplo, el término 'who' indica que la pregunta está buscando como respuesta una persona. Sin embargo, en otros casos, se necesita analizar algunas estructuras sintácticas de la pregunta para obtener la clase semántica de la respuesta. Por

ejemplo, 'Who is the tallest man ...?' es el término 'man' (núcleo del sintagma nominal 'tallest man') el que indica el tipo de respuesta esperado, en este caso, el nombre de una persona.

Del análisis de la pregunta también se obtiene aquella información que permite la generación de consultas que facilitan la selección de los extractos de texto de los documentos sospechosos de contener la respuesta.

Existen dos tendencias que se siguen para obtener estas consultas. Por un lado, está la selección de palabras clave, que consiste en seleccionar aquellos términos de la pregunta cuya aparición en un texto es de por sí indicativa de la posibilidad de existencia en sus alrededores de la respuesta buscada. Para la pregunta '¿Qué país limita al sur con Brasil?', el conjunto de palabras clave estaría formado por los términos 'limita', 'sur' y 'Brasil'. Por otro lado, está el proceso de patrones de respuesta, en el que las consultas estarán formadas por una o varias combinaciones de los términos de la pregunta en forma de expresiones en las que podría encontrarse la respuesta. Posibles consultas derivadas del ejemplo anterior serían: 'X limita al sur con Brasil', 'X, país que limita al sur con Brasil', 'La frontera sur de Brasil es X', etc. donde X es una referencia a la respuesta a encontrar. En este caso, el sistema de recuperación de información se encarga de localizar extractos de texto que contengan posibles expresiones de respuesta asociadas a cada tipo de pregunta [39] [40]. Ambas estrategias no son excluyentes la una de la otra, ya que existen sistemas que combinan ambas aproximaciones [41].

Por lo que respecta al proceso final de extracción de la respuesta, se suelen emplear, o bien técnicas de recuperación de información, o bien patrones de respuesta en combinación con el uso de clasificadores de entidades. Estas herramientas permiten localizar aquellas entidades cuya clase semántica corresponde con aquella que la pregunta espera como respuesta. De esta forma, el sistema extraerá la respuesta de aquellos extractos de texto que contienen alguna entidad del tipo semántico requerido, de forma combinada con la aparición de términos clave en sus cercanías y/o la validación de patrones de respuesta. Finalmente, el sistema ha de elegir de entre las entidades que pueden ser respuesta a la pregunta. Este proceso se lleva a cabo mediante la aplicación de medidas que permitan valorar de alguna forma el grado de corrección de cada posible respuesta. Esta valoración se suele realizar aplicando funciones que miden, por una parte, el grado de cumplimiento de aquellas características que tiene en cuenta el sistema en el proceso de búsqueda de la respuesta (cercanía de palabras clave en el texto, fiabilidad de los patrones validados, etc.) y, por otra parte, circunstancias generalmente relacionadas con la redundancia de aparición de cada respuesta posible en diferentes documentos.

Entrando en los sistemas que utilizan patrones como base para las tareas de búsqueda de respuestas, estos se fundamentan en la identificación y construcción de una serie de patrones indicativos que dependen del tipo de pregunta a tratar y cuya validación está relacionada con la posibilidad de encontrar la respuesta correcta. Un patrón indicativo es una secuencia o combinación determinada de caracteres, signos de puntuación, espacios, dígitos o palabras. Estos patrones se obtienen de forma totalmente manual mediante el estudio de expresiones que son respuestas a determinados tipos de preguntas. Por ejemplo, la cadena J. R. R. Tolkien (1892-1973) contiene la respuesta a preguntas relacionadas con los años en que Tolkien nació y falleció. A partir de aquí, se puede construir el siguiente patrón: '*[palabra con la primera letra en mayúsculas; paréntesis; cuatro dígitos; guión; cuatro dígitos; paréntesis]*'. Dicho patrón permite detectar respuestas a preguntas acerca del periodo de existencia de una persona. A cada uno de estos patrones se le asigna un valor de forma que el sistema pueda elegir entre varias posibles respuestas a una pregunta en función del grado de fiabilidad de cada patrón con respecto a la pregunta.

Hay sistemas, como el utilizado por IBM [42], que basan su aproximación en el concepto de anotación predictiva. Esto consiste en utilizar un etiquetador de entidades para anotar, en todos los documentos de la colección, la clase semántica de aquellas entidades que detecta. Dicha clase semántica se indexa junto con el resto de términos de los documentos, facilitando así la recuperación de los extractos de documentos que contienen entidades cuya clase semántica coincide con la esperada como respuesta. Estos sistemas tienen en cuenta la similitud entre las estructuras sintácticas de las preguntas y posibles respuestas como factor importante en el proceso de extracción de la respuesta final [43] [44].

Por último, cabe destacar que algunos sistemas se caracterizan principalmente por la aplicación de técnicas de aprendizaje, basadas en modelos de máxima entropía, a los procesos de análisis de la pregunta y de extracción final de la respuesta [45] [46]. En ambos casos, estas técnicas se aplican en un módulo que valida la corrección de las respuestas suministradas por el sistema mediante la estimación de la probabilidad de que una respuesta sea correcta.

3.3.4. Sistemas que utilizan técnicas de análisis profundo

De forma general, estos sistemas obtienen la representación semántica de la pregunta y de aquellas sentencias que son relevantes a dicha pregunta. La extracción de la respuesta se realiza mediante procesos de comparación y/o unificación entre las representaciones de la pregunta y las frases relevantes.

Los sistemas de la universidad de Pisa [47] y CLR [48] utilizan el concepto de tripletas semánticas para representar dicha información. Una tripleta semántica está formada por una entidad del discurso, el rol semántico que dicha entidad desempeña y el término con el que dicha entidad mantiene la relación. Con esta notación se representan las preguntas y las frases que contienen respuestas del tipo esperado para proceder a la extracción de la respuesta comparando y puntuando el nivel de relación existente entre las estructuras semánticas obtenidas en preguntas y frases objetivo.

El sistema de la Universidad de Sheffield [49] utiliza fórmulas lógicas para representar las preguntas y los pasajes candidatos a contener la respuesta y los incorpora en un modelo de discurso. Este modelo codifica el conocimiento general del mundo y se enriquece con el conocimiento específico codificado en las fórmulas lógicas de la pregunta y los pasajes candidatos. La selección de la respuesta final se realiza mediante la aplicación de sistemas de puntuación que valoran, principalmente, la redundancia observada en cada una de las respuestas posibles.

3.4 BERT

BERT, cuyas siglas significan *Bidirectional Encoder Representations from Transformers*, es un modelo de representación del lenguaje introducido entre finales de 2018 y principios de 2019 [50].

Este modelo consta de dos pasos: preentrenamiento y afinación. Durante el preentrenamiento, el modelo es entrenado con datos sin etiquetar sobre disferentes tareas. Para la afinación, se inicializa el modelo con los parámetros preentrenados, y todos estos se afinan usando datos etiquetados para tareas posteriores. Cada una de estas tareas tiene modelos de afinación distintos, aunque se inicialicen con los mismos parámetros entrenados previamente. Un ejemplo para la tarea de búsqueda de respuestas se puede apreciar en la figura 3.2, donde [CLS] es un símbolo especial añadido al principio de todos los

ejemplos de entrada, y [SEP] es un token separador que, en el caso de la búsqueda de respuestas, sirve para separar las preguntas de las respuestas.

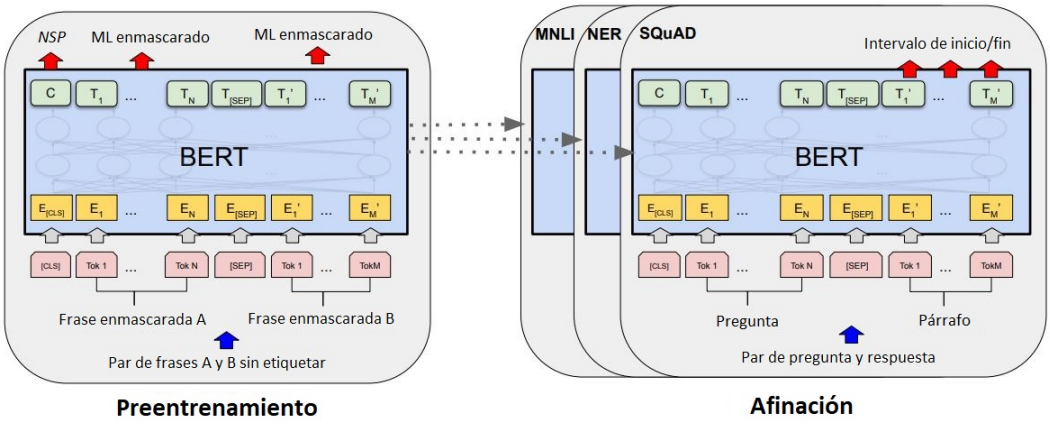

Figura 3.2: Procedimientos generales de preentrenamiento y afinación de BERT.

3.4.1. Arquitectura

La arquitectura del modelo de BERT consiste en un codificador de Transformer bidireccional multicapa basado en la implementación original [20]. A efectos prácticos, es como si se tomase el modelo del Transformer y suprimiéramos el Decoder, quedándonos con el Encoder.

El número de capas, es decir, los bloques del Transformer, se denotan como L, el tamaño de las capas ocultas como H y el número de *heads* como A. De esta forma, BERT presenta dos tamaños de modelo. Por un lado, está el BERT Base, con $L = 12$, $H = 768$ y $A = 12$, por lo que el número total de parámetros asciende a 110 millones. Por otro lado, está el BERT grande (*BERT Large* en inglés), con $L = 24$, $H = 1024$ y $A = 16$, de manera que el número total de parámetros es de 340 millones.

3.4.2. Representaciones de la entrada y salida

Para hacer que BERT maneje una variedad de tareas posteriores, la representación de entrada ha de ser capaz de representar en una secuencia de tokens tanto una sola oración como un par de oraciones, este último caso es de vital importancia en la tarea de búsqueda de respuestas.

Como se ha explicado anteriormente, el primer token de cada secuencia es siempre un token de clasificación especial, denominado [CLS]. El estado oculto final correspondiente a este token se utiliza como representación de secuencia agregada para tareas de clasificación. Los pares de oraciones se agrupan en una sola secuencia, diferenciándolas de dos formas. Primero, se separan con un token especial ([SEP]), y luego se agrega un *embedding* aprendido a cada token que indica si pertenece a la oración 1 o a la oración 2. Como se puede observar en la figura 3.2, el *embedding* de entrada es representado por una E, el vector oculto final para el token especial [CLS] por una $C \in \mathbb{R}^H$, y el vector oculto final para el i-ésimo token por $T_i \in \mathbb{R}^H$.

Dado un token, su representación de entrada es construida sumando el token correspondiente con el *embedding* de segmento y posición. Esta construcción se puede apreciar en la figura 3.3.

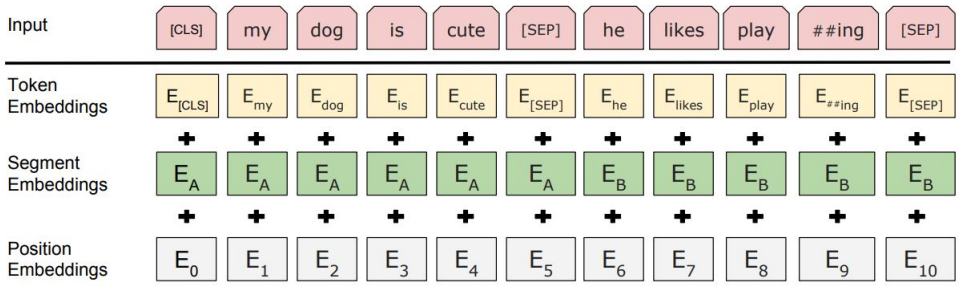

Figura 3.3: Representación de la entrada para BERT.

3.4.3. Preentrenamiento

A pesar de los métodos tradicionales de aprendizaje de modelos de lenguaje de izquierda a derecha o derecha a izquierda [51] [52], BERT utiliza un procedimiento distinto para su preentrenamiento. Este proceso consiste en dos tareas no supervisadas, y que se pueden apreciar en la figura 3.2.

Por un lado, está la tarea del modelado de lenguaje enmascarado (**ML enmascarado**). Intuitivamente, es razonable pensar que un modelo bidireccional profundo es estrictamente más potente que un modelo de izquierda a derecha o que la concatenación superficial de un modelo de izquierda a derecha con uno de derecha a izquierda. Desafortunadamente, los modelos de lenguaje condicionales estándar solo se pueden entrenar de izquierda a derecha o de derecha a izquierda, ya que el condicionamiento bidireccional permitiría que cada palabra 'se vea a si misma' indirectamente, por lo que el modelo podría predecir trivialmente la palabra de destino en un contexto de múltiples capas.

Para entrenar una representación bidireccional profunda, simplemente enmascaramos un porcentaje de los tokens de entrada al azar y luego predecimos esos tokens enmascarados. Este procedimiento es conocido como modelado de lenguaje enmascarado, aunque a menudo se le denomina tarea *Cloze* [53]. En este caso, los vectores ocultos finales correspondientes a los tokens enmascarados se introducen en una *softmax* de salida sobre el vocabulario, como en un modelado de lenguaje estándar.

Aunque esto permite obtener un modelo preentrenado bidireccional, una desventaja es que se está creando un desajuste entre el preentrenamiento y la afinación, ya que el token [MASK] no aparece durante este último. Para mitigar esto, no siempre se reemplazan las palabras 'enmascaradas' con el token [MASK]. El generador de datos de entrenamiento elige el 15 % de las posiciones de los tokens al azar para la predicción. Si se elige el token i-ésimo, reemplazamos el token i-ésimo:

- El 80 % de las veces, por el token [MASK].
- El 10 % de las veces, por un token aleatorio.
- El 10 % de las veces, permanece igual.

Luego, T_i se usará para predecir el token original con pérdida de entropía cruzada.

Por otro lado, está la tarea de predicción de la siguiente frase, en inglés ***Next Sentence Prediction*** **(NSP)**. Muchas tareas importantes posteriores, entre las que se encuentra la búsqueda de respuestas, se basan en la comprensión de la relación entre dos oraciones, que no se captura directamente en el modelado del lenguaje. Con el fin de entrenar un modelo que comprenda las relaciones entre oraciones, se realiza un entrenamiento

previo para una tarea binarizada de predicción de la siguiente oración que se puede generar trivialmente a partir de cualquier corpus monolingüe. Específicamente, al elegir las oraciones *A* y *B* para cada ejemplo de entrenamiento previo, el 50 % del tiempo *B* es la siguiente oración real que sigue a *A* (etiquetada como *IsNext*), y el 50 % de las veces es una oración aleatoria del corpus (etiquetada como *NotNext*). Como se muestra en la figura 3.2, *C* se usa para la predicción de la siguiente frase.

La tarea de NSP está estrechamente relacionada con los objetivos de aprendizaje de representación utilizados en [54] y [55]. Sin embargo, en trabajos previos, solo los *embeddings* de oraciones se transfieren a tareas posteriores, mientras que BERT transfiere todos los parámetros para inicializar los parámetros del modelo de la tarea final.

3.4.4. Afinación del modelo

La afinación es sencilla, ya que el mecanismo de atención en el Transformer permite que BERT modele muchas tareas posteriores intercambiando las entradas y salidas apropiadas, independientemente de que involucren texto único o pares de texto. Para aplicaciones que involucren pares de texto, un patrón común es codificar pares de texto de forma independiente antes de aplicar atención cruzada bidireccional [56] [57]. BERT, en cambio, utiliza el mecanismo de atención para unificar estas dos etapas, ya que la codificación de un par de texto concatenado mediante atención incluye atención cruzada bidireccional entre dos oraciones.

Para cada tarea, simplemente se conectan las entradas y salidas específicas de la tarea en BERT y se ajustan todos los parámetros de un extremo a otro. En la entrada, la oración A y la oración B del entrenamiento previo son análogas a los pares de preguntas y pasajes en la respuesta a preguntas. En la salida, las representaciones de tokens se introducen en una capa de salida para tareas a nivel de token, y la representación [CLS] es alimentada a una capa de salida para su clasificación.

3.5 Conclusiones

Habiendo explicado en qué consiste un sistema de búsqueda de respuestas y el modelo de BERT, nuestro enfoque para abordar esta tarea se basará en la versión base de este modelo, cuya configuración, implementación y demás se tratará con más detalle en los capítulos 4 y 5.

CAPÍTULO 4
Implementación y experimentación

En este capítulo se expondrán y se compararán las implementaciones de traducción neuronal y comprensión de texto para tareas de búsqueda de respuestas, basadas en los conceptos explicados previamente.

4.1 Detalles de implementación

Para la implementación de los sistemas de traducción automática neuronal se ha usado Keras[1], una API de alto nivel escrita en Python sobre Tensorflow, una librería de Python para entrenar y evaluar redes neuronales de manera eficiente. Para trabajar con Keras, se empleó el *toolkit* NMT-Keras [58]. Para los sistemas de búsqueda de respuestas se utilizó Google Colab, la herramienta de Google en línea para ejecutar código Python y crear modelos de aprendizaje automático a través de la nube de Google, con la posibilidad de hacer uso de sus GPU. Los modelos de traducción fueron entrenados en una GeForce RTX 2080, proporcionada por el grupo de investigación PRHLT[2], mientras que los modelos de búsqueda de respuestas fueron entrenados en una Tesla P100-PCIe, proporcionada por Google.

4.2 Métricas empleadas

Para la evaluación de los sistemas implementados se emplean unas determinadas métricas que midan la calidad de los modelos obtenidos. En el caso de traducción automática, hemos utilizado la métrica BLEU, mientras que para la búsqueda de respuestas se ha hecho uso de la métrica F1. Para entender esta última métrica, es necesario explicar también las métricas de precisión y *recall*.

4.2.1. BLEU

El BLEU (*BiLingual Evaluation Understudy*) [59] es una métrica para evaluar el texto traducido automáticamente. La puntuación BLEU es un número entre cero y uno que mide la similitud del texto traducido de manera automática con un conjunto de traducciones de referencia de alta calidad. Un valor de 0 significa que la traducción automática de salida no se superpone con la traducción de referencia (calidad baja), mientras que un valor de 1 significa que se superpone perfectamente con las traducciones de referencia (calidad alta).

[1] https://keras.io
[2] https://www.prhlt.upv.es/wp/es/

Matemáticamente, la puntuación BLEU se define como:

$$BLEU = min(1, e^{1-\frac{r}{c}})e^{\sum_{n=1}^{N} w_n log P_n} \tag{4.1}$$

Donde r es la longitud de la frase de referencia, c es la longitud de la frase candidata, N los ngramas que se vayan a utilizar (en nuestro caso, 4 como máximo), w_n el peso que se le asigna a cada ngrama, tal que $\sum_{n=1}^{N} w_n = 1$ (típicamente $w_n = \frac{1}{N}$) y P_n es la precisión modificada para el ngrama n.

El cálculo de dicha predicción modificada es la siguiente:

$$P_n = \frac{ngramascomunes}{ngramascandidata} \tag{4.2}$$

De esta forma, si la frase candidata es 'A cat is on the mat' y la frase de referencia es 'The cat is on the table', la precisión en 1-gramas sería de $\frac{4}{6}$, mientras que en 2-gramas sería de $\frac{3}{5}$ y así sucesivamente.

4.2.2. Precisión

La precisión es el porcentaje de muestras recuperadas que son relevantes. De una manera más formal se puede definir como:

$$Precision = \frac{C \cap P}{R} \tag{4.3}$$

Donde C es el conjunto de muestras correctas, P es el conjunto de muestras predichas y R es el conjunto de muestras recuperadas.

4.2.3. *Recall*

El *recall* o exhaustividad es el porcentaje de muestras relevantes que han sido recuperadas. De una manera más formal se puede definir como:

$$Recall = \frac{C \cap P}{C} \tag{4.4}$$

Donde C y P tienen el mismo significado que en el apartado anterior.

4.2.4. F1

La puntuación F1 [60] es una métrica que se define como una media armónica entre la *Precision* y el *Recall*:

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{4.5}$$

4.3 Sistemas de traducción neuronal

Cada modelo está basado en la arquitectura de encoder-decoder con mecanismo de atención propuesta en [19]. Gráficamente, se puede representar el modelo del cuál se ha

partido en la figura 4.1, proporcionado por NMT-Keras. Para el encoder-decoder se escogió LSTM como función de activación para las redes recurrentes, tanto en el encoder como en el decoder. El tamaño del estado oculto de cada LSTM fue de 128 y el tamaño de los *word embeddings*, tanto para el idioma fuente como el de destino, fue de 64. Los modelos han sido entrenado usando el algoritmo Adam [61] con un factor de aprendizaje de 2^{-4}. La selección de dicho algoritmo y factor de aprendizaje fue el resultado de un proceso de experimentación, donde se probaron otros valores para el factor de aprendizaje y otros algoritmos de optimización. Se utilizó un tamaño de *batch* de 16 y un tamaño de *beam* de 6 para todos los modelos. El entrenamiento se detuvo después de 50 *epochs* para cada modelo, debido a que el BLEU no mejoraba, y este se calculaba al final de cada *epoch* sobre el conjunto de validación.

NMT-Keras también proporciona un modelo de Transformer, el cuál se puede apreciar en la figura 4.2. Sin embargo, los resultados obtenidos tras el entrenamiento y evaluación con él fueron peores que los obtenidos con el modelo de atención de red neuronal recurrente, por lo que fue descartado en favor del modelo de atención encoder-decoder con redes neuronales recurrentes. Para este modelo de Transformer se probaron distintas configuraciones, que consistieron en modificar los hiperparámetros del tamaño de salida del Transformer, el tamaño de las capas *feed-forward* y el número de capas de atención paralelas del modelo.

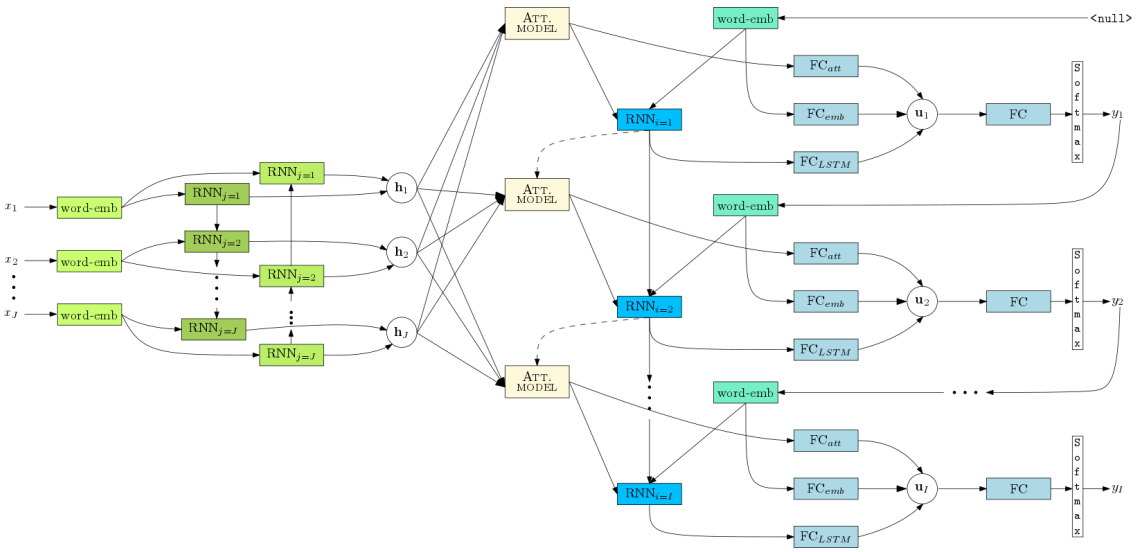

Figura 4.1: Modelo de atención de red neuronal recurrente proporcionado por NMT-Keras.

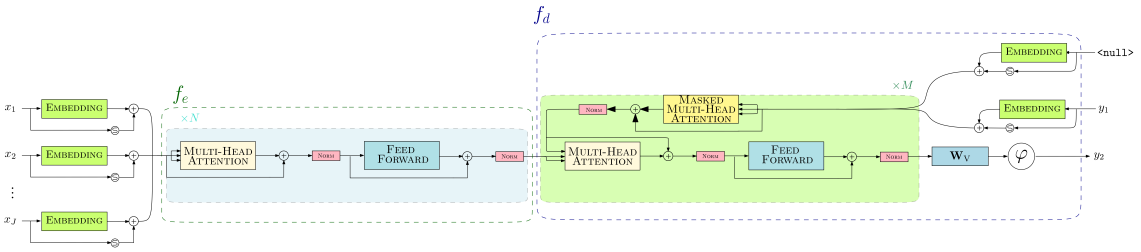

Figura 4.2: Modelo de Transformer proporcionado por NMT-Keras.

4.3.1. Datasets

Para probar el proceso de experimentación, los sistemas de traducción automática se evaluaron frente a varios conjuntos de datos extraídos de la web de OPUS[3], un proyecto en el que se intenta alinear datos en línea, agregar notificaciones lingüísticas y proporcionar a la comunidad científica un corpus paralelo disponible de manera pública. A partir de estos corpus se extrajeron particiones seleccionadas al azar para la fase de entrenamiento, validación y evaluación. Debido a la escasa cantidad de datos paralelos entre el inglés y el catalán y el inglés y el euskera, se decidió que el idioma fuente para realizar las traducciones de esos dos idiomas fuese el español, debido a que había una mayor cantidad de datos emparejados entre este último y los otros dos, quedándose el inglés como el idioma fuente a la hora de traducir al castellano.

A continuación serán presentados los conjuntos de datos escogidos y sus estadísticas. A partir de esta información se puede inferir lo difícil que puede llegar a resultar una tarea, ya que intervienen diversos factores como pueden ser la longitud de la oración o los datos disponibles para entrenar. Es necesario recalcar que los tres conjuntos seleccionados para entrenamiento, validación y evaluación se componen de la unión de muestras entre distintos corpus. Esto se debe a que el conjunto de datos de la tarea de búsqueda de respuestas que se quiere traducir, el SQuAD-v2.0, contiene oraciones correspondientes a distintos contextos, por lo que el traductor ha de ser capaz de comprender bien dichos contextos a la hora de traducirlos, o al menos estar bien orientado para que las traducciones estén tan ajustadas al idioma original como sea posible.

En lo que respecta al tamaño de las muestras, se decidió que cada frase tuviese una longitud máxima de 20 palabras, quedándonos con aquellas frases de cada dataset que fuesen de longitud menor o igual a dicho tamaño. Esto se debe a que, a la hora de entrenar el sistema, la diferenciación automática es demasiado costosa computacionalmente si la matriz de *embeddings* que representa las muestras tiene un tamaño intratable.

Para la tarea de traducción del inglés al español, los datasets escogidos, junto con el número de muestras de cada uno de ellos y cuantas de ellas se dedicaron a entrenamiento, validación y evaluación se pueden observar en la tabla 4.1. Del mismo modo, para la tarea de español a catalán se puede apreciar en la tabla 4.2 y para la tarea del español al euskera en la tabla 4.3.

4.4 Sistemas de búsqueda de respuestas

Los tres modelos empleados (uno para el español, otro para el catalán y otro para el euskera), han sido entrenados utilizando BERT en su versión base. El tamaño de *batch* utilizado fue de 12, mientras que el número de *epochs* durante el entrenamiento fue de 2, y el factor de aprendizaje se fijó en 3^{-5}. La longitud máxima por secuencia fue variando para comprobar si esta repercutía en el resultado final, por los que los valores que se tuvieron en cuenta fueron desde 32 hasta 384.

4.4.1. Datasets

Para el proceso de experimentación, los sistemas de búsqueda de respuestas se evaluaron frente a los conjuntos de datos SQuAD (en su versión 2.0) y XQuAD.

A diferencia de la traducción automática, para la búsqueda de respuestas no se utilizaron los conjuntos de validación durante el entrenamiento, a pesar de que el mismo

[3] http://opus.nlpl.eu/

Dataset	Entrenamiento	Validación	Evaluación	Total
GNOME	845	422	422	1.689
KDEdoc	1.126	563	565	2.254
EUconst	1.720	860	863	3.443
Ubuntu	3.379	1.690	1.690	6.759
PHP	3.620	1.810	1.814	7.244
WMT-News	4.080	2.040	2.039	8.159
OpenOffice	5.390	2.695	2.694	10.779
UN	47.734	6.547	6.842	61.100
Bible	48.046	6.590	6.825	61.500
Books	72.499	9.943	10.347	92.800
ECB	78.124	10.715	27.274	100.000
Tanzil	78.124	10.715	44.896	100.000
KDE4	84.507	11.590	12.073	108.170
News-Commentary	84.507	11.590	12.073	108.170
TED2013	84.507	11.590	12.073	108.170
EMEA	84.507	11.590	12.073	108.170
SciELO	84.507	11.590	12.073	108.170
JRC-Acquis	84.507	11.590	12.073	108.170
GlobalVoices	84.507	11.590	12.073	108.170
Wikipedia	84.507	11.590	12.073	108.170
Europarl v7	84.507	11.590	12.073	108.170
DGT	84.507	11.590	12.073	108.170
TildeMODEL	84.507	11.590	12.073	108.170
EUbookshop	84.507	11.590	12.073	108.170
MultiUN	84.507	11.590	12.073	108.170
UNPC	84.507	11.590	12.073	108.170
ParaCrawl	84.507	11.590	12.073	108.170
OpenSubtitles	84.507	11.590	12.073	108.170
Total	**1.696.799**	**240.030**	**299.439**	**2.236.268**

Tabla 4.1: Datasets y cantidad de frases paralelas tomadas de cada uno de ellos para la traducción del inglés al español.

Dataset	Entrenamiento	Validación	Evaluación	Total
EUbookshop	892	446	446	1.784
Ubuntu	3.425	1.712	1.713	6.850
GNOME	3.375	1.688	1.687	6.750
GlobalVoices	6.038	3.020	3.018	12.076
KDE4	75.940	37.970	37.973	151.883
OpenSubtitles	363.435	60.572	60.573	484.580
DOGC	981.063	163.510	163.466	1.308.039
Total	**1.434.168**	**268.918**	**268.876**	**1.971.962**

Tabla 4.2: Datasets y cantidad de frases paralelas tomadas de cada uno de ellos para la traducción del español al catalán.

conjunto de datos de SQuAD tiene su propio archivo dedicado a ello. Esto se debe a que, en las primeras fases de experimentación, se observó que la diferencia de entrenar con o sin estos conjuntos era mínima en tanto que el resultado obtenido era muy similar.

Dataset	Entrenamiento	Validación	Evaluación	Total
GNOME	1.230	616	615	2.461
Ubuntu	1.495	749	745	2.989
MaSS	3.197	1.600	1.597	6.394
KDE4	41.953	20.970	20.982	83.905
EhuHac	450.000	75.000	61.839	586.839
Elhuyar	450.000	75.000	85.796	610.796
OpenSubtitles	525.000	87.500	95.462	707.962
Total	**1.472.875**	**261.435**	**267.036**	**2.001.346**

Tabla 4.3: Datasets y cantidad de frases paralelas tomadas de cada uno de ellos para la traducción del español al euskera.

Para el caso de SQuAD, se escogió la versión pequeña de entrenamiento, que incluía 18.000 párrafos/contextos y 69.000 preguntas con sus correspondientes respuestas.

En lo que respecta a XQuAD, este dataset presenta algunas diferencias con el anterior. Para empezar, SQuAD tiene, en algunos párrafos, preguntas cuya respuesta exacta no se encuentra en su correspondiente párrafo y para las cuales se da una respuesta plausible. En XQuAD este tipo de respuestas no se encuentran. Además de eso, XQuAD tiene su propia versión en español, por lo que únicamente fue necesario traducir este conjunto de datos al catalán y al euskera. De esta forma, la versión del dataset que se empleó para evaluación contiene 240 párrafos y 1.190 preguntas con sus correspondientes respuestas.

CAPÍTULO 5
Resultados

En este capítulo se describirán los resultados obtenidos por cada una de las experimentaciones realizadas. Los resultados se expresan como porcentaje en BLEU, en el caso de traducción automática, y como porcentaje en F1, para el caso de búsqueda de respuestas.

5.0.1. Traducción automática

Los mejores resultados obtenidos para las tres tareas de traducción automática se consiguieron utilizando el modelo de atención de red neuronal recurrente proporcionado por NMT-Keras, empleando los parámetros que se han expuesto en la sección 4.3, es decir, mediante LSTM de tamaño 128, con *word-embeddings* de tamaño 64 en el idioma fuente y destino, algoritmo de optimización Adam con factor de aprendizaje 2^{-4}, estos dos últimos hiperparámetros debido a que, tras probar con distintos valores, los mejores resultados se obtuvieron con esos. Finalmente, el tamaño de *batch* fue de 16, el tamaño de *beam* de 6 y el entrenamiento duró 50 *epochs*. Los resultados con esta configuración, y que resultaron ser los mejores que obtuvimos, se pueden apreciar en la tabla 5.1.

	BLEU
Del inglés al español	40,7
Del español al catalán	47,9
Del español al euskera	33,6

Tabla 5.1: Resultados para el conjunto de test usando el modelo de atención de red neuronal recurrente de NMT-Keras.

La tarea de traducción de español a catalán obtiene los mejores resultados, probablemente debido a que es el par de idiomas que más similares son entre si. De forma parecida, la traducción de inglés a español obtiene un mejor resultado que la del español al euskera, lo que puede deberse a que este último no tiene relación o conexión lingüística alguna con otros idiomas conocidos, a pesar de haber adquirido bastante léxico del español.

También se realizaron experimentos con el modelo de Transformer que proporciona NMT-Keras, cuya mejor combinación de hiperparámetros fue el de utilizar un tamaño de modelo de 64 (es decir, que produce salidas de tamaño 64), un tamaño de capas *feed-forward* de 256 y 16 capas de atención paralelas. Los resultados con esta configuración, aunque no resultaron ser los mejores que obtuvimos, se pueden apreciar en la tabla 5.2.

	BLEU
Del inglés al español	34,6
Del español al catalán	41,3
Del español al euskera	29,8

Tabla 5.2: Resultados para el conjunto de test usando el modelo de Transformer de NMT-Keras.

5.0.2. Búsqueda de respuestas

Para la búsqueda de respuestas, además de emplear la F1 que se ha explicado anteriormente, se utiliza la métrica de *Exact Match* (EM) que simplemente comprueba que las palabras de las respuestas de referencia y las de las candidatas coinciden exactamente. Es una métrica que sirve de referencia y de apoyo para la F1 más que como una métrica independiente, debido a que si, por ejemplo, la respuesta candidata es 'Me llamo Juan' y la de referencia era 'Mi nombre es Juan', el EM tiene en cuenta a la candidata como respuesta incorrecta.

De esta forma, empleando el modelo de BERT base, con un tamaño de *batch* de 12, 2 *epochs*, factor de aprendizaje 3^{-5} y variando la longitud máxima por secuencia tal y como se había explicado en la sección 4.4, los resultados obtenidos se pueden observar en la tabla 5.3.

Idioma	Longitud máxima por secuencia	EM	F1
Español	32	42,6	61,3
	64	44,8	63,9
	192	56,7	74,7
	256	56,1	74,5
	384	**56,8**	**75,9**
Catalán	32	41,9	56,4
	64	44	60,6
	192	56,1	72,2
	256	55,7	71,8
	384	**56,2**	**72,9**
Euskera	32	17,2	31,1
	64	20,8	36,5
	192	32,3	48,4
	256	31,9	48,4
	384	**32,3**	**48,9**

Tabla 5.3: Resultados para el conjunto de test XQuAD usando BERT en Google Colab.

A la vista de estos resultados se puede afirmar que la mejor configuración para la tarea de búsqueda de respuestas es la asignar un valor alto a la longitud máxima por secuencia que se tiene en cuenta de las muestras, ya que para los tres idiomas, por lo general, cuanto mayor es este valor, mejores resultados se obtienen. Además, en español y catalán se obtienen unos resultados competentes, ya que si los comparamos con los resultados obtenidos para la misma tarea en inglés [64], podemos observar que en esa el resultado en F1 era de **77,8**. Esto probablemente se deba a que la construcción de sus respectivos datasets en español y catalán ha sido satisfactoria partiendo de una buena traducción. Por otro lado, los resultados obtenidos para la misma tarea en euskera han sido más pobres, lo que puede deberse al problema que se ha comentado anteriormente con la traducción (la poca similitud entre el euskera y el idioma fuente), por lo que es altamente probable que las traducciones de las respuestas no se correspondan con su

aparición en su respectivo párrafo, sino que se traten de sinónimos que el traductor ha dado por válidos.

5.1 Conclusiones

A la vista de los resultados obtenidos para las dos tareas que se han propuesto, se puede afirmar que el peor de los resultados, en ambos casos, es el que involucra al idioma euskera. En el caso de búsqueda de respuestas, puede tener su origen en la generación del conjunto de datos SQuAD y en las traducciones que se han realizado, lo cuál puede haber dado lugar a que el F1 haya sido más bajo que en el caso de sus correspondientes versiones en español y catalán. Si seguimos tirando del hilo y nos planteamos el por qué de estas traducciones tan poco íntegras, una causa puede ser el poco parecido entre el idioma español y el euskera, lo que ocasiona que el traductor neuronal construido no tenga en cuenta una parte importante del contexto del idioma de entrada.

CAPÍTULO 6
Conclusiones finales y trabajos futuros

6.1 Conclusiones finales

En este trabajo de fin de Máster se ha propuesto un sistema de traducción neuronal y un modelo de búsqueda de respuestas. El objetivo de esto era tratar de traducir el conjunto de datos dedicado a la búsqueda de respuestas para poder obtener datasets en aquellos idiomas de los que no se disponen de datos para esta tarea.

Para hacer esto, se propuso un sistema de atención de redes neuronales recurrentes proporcionado por el *toolkit* de traducción NMT-Keras. Una vez traducidos los párrafos, las preguntas y las respuestas de los conjuntos de datos SQuAD y XQuAD al español, catalán y euskera, se propuso el modelo base de BERT para la obtención de respuestas.

Los resultados obtenidos, tanto en traducción como en búsqueda de respuestas, resultan aceptables. En el caso de la traducción, el BLEU obtenido para la tarea de traducir al euskera es más reducido debido a la poca similitud entre el idioma fuente y el destino. Por otra parte, en el caso de la búsqueda de respuestas, se ha arrastrado dicho problema en la traducción y se ha obtenido un resultado más pobre que su homónimo en español y catalán.

Con el fin de poder replicar experimentos realizados, se alojará en Github[1] el código desarrollado, junto con los datasets empleados para la traducción automática, así como los conjuntos de datos SQuAD y XQuAD, en sus distintas versiones en español, catalán y euskera.

Por último, a nivel profesional este trabajo ha permitido obtener conocimientos más sólidos desde un punto de vista teórico y práctico, este último en especial gracias al uso de *toolkits* como NMT-Keras para el desarrollo de modelos predictivos, dedicado en este caso a la traducción. Estos conocimientos son de vital importancia para el mundo productivo, ya que empresas como Google o Amazon emplean librerías como Tensorflow o Keras para el análisis, la extracción y la comprensión de información a partir de grandes cantidades de datos.

[1] https://github.com/jualora/MIARFID/TFM

6.2 Trabajos Futuros

Anteriormente se han presentado diferentes enfoques de traducción neuronal para lidiar con las tareas que aborda este trabajo. Sin embargo, todavía quedan líneas de trabajo abiertas que podrían ampliar este trabajo de fin de Máster.

6.2.1. Nuevos datasets para la búsqueda de respuestas

A pesar de que se ha escogido SQuAD y XQuAD como datasets para entrenamiento y evaluación de sistemas de búsqueda de respuestas, existen otros conjuntos de datos que podrían haber sido abordados para esta tarea, como es el caso de CoQA[2] o MLQA [63], este último es de especial interés para la siguiente línea de trabajo futuro plausible.

6.2.2. Enfoque multilingüe en la búsqueda de respuestas

A pesar de que para este trabajo se ha considerado un idioma por tarea, en la búsqueda de respuestas también suelen ser habituales métodos multilingües, como puede ser el entrenamiento de un sistema en un idioma y su validación en otro.

MLQA es un conjunto de datos indicado para este enfoque, ya que contiene, para un mismo párrafo, diferentes preguntas (con sus respectivas respuestas) planteadas en múltiples idiomas, como se puede apreciar en la figura 6.1.

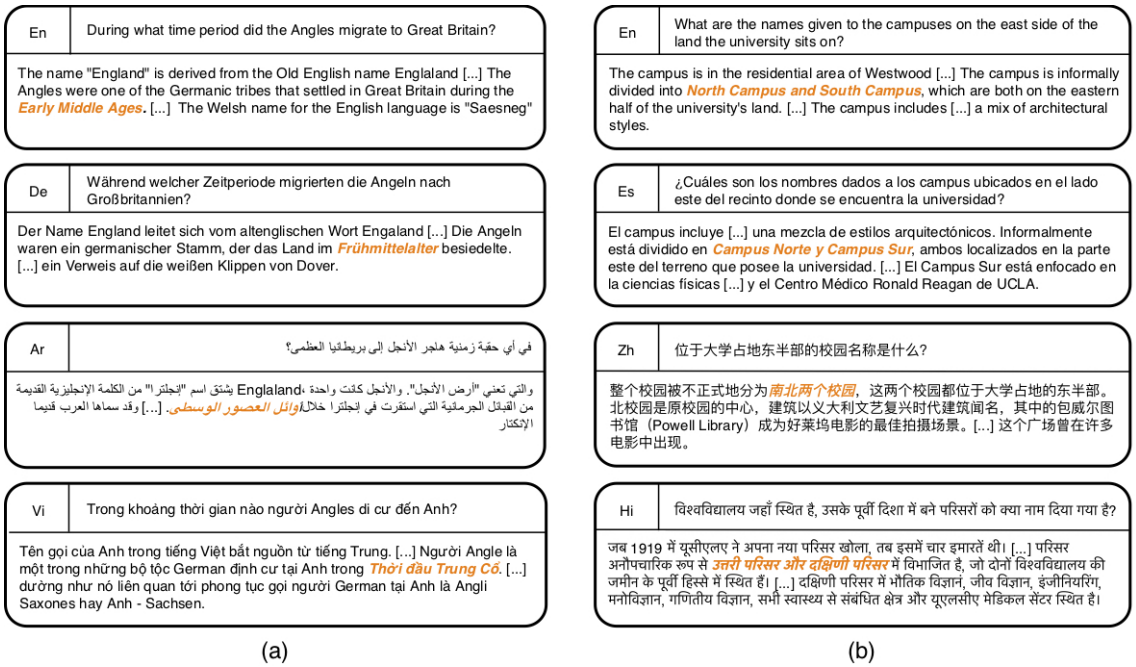

Figura 6.1: Dos instancias de MLQA.

6.2.3. Mejora de resultados para euskera

Debido a los mejorables resultados obtenidos para el euskera, tanto en la traducción automática como en la búsqueda de respuestas, sería interesante investigar acerca de este suceso.

[2] https://stanfordnlp.github.io/coqa/

Un primer enfoque podría ser el análisis de los datasets de SQuAD y XQuAD generados en el presente trabajo, debido a que una buena cantidad de muestras no tengan las respuestas íntegras para su correspondiente pregunta y contexto de referencia, bien porque la respuesta se encuentra repetida en dicho contexto y la generación del dataset no ha seleccionado la adecuada, o bien porque el traductor al euskera ha tomado una respuesta al español y la ha traducido por una palabra que, aunque puede ser correcta, no se encuentre en el contexto y se trate de un sinónimo de la respuesta apropiada.

CAPÍTULO 7
Agradecimientos

A Francisco Casacuberta, por dirigir este Trabajo de Fin de Máster.
A Miguel Domingo, por todo el apoyo prestado.
A mi familia y a mis amigos, que siempre están ahí.
Gracias.

Bibliografía

[1] Rajpurkar, P., Jia, R. y Liang, P. (2018). Know What You Don't Know: Unanswerable Questions for SQuAD. *Computing Research Repository-arXiv*, abs/1806.03822.

[2] Chomsky, N. (1957). Syntactic Structures. *Mouton & Co.*

[3] Castano, M. A., Casacuberta, F. y Vidal, E. (1997) Machine translation using neural networks and finite-state models. *Theoretical and Methodological Issues in Machine Translation*, pp. 160-167.

[4] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. y Zheng, X. (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *Computing Research Repository-arXiv*, abs/1603.04467.

[5] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. y Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Computing Research Repository-arXiv*, abs/1912.01703.

[6] Theano Development Team. (2016). Theano: A Python framework for fast computation of mathematical expressions. *Computing Research Repository-arXiv*, abs/1605.02688.

[7] Bengio, Y., Ducharme, R., Vincent, P. y Janvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137-1155.

[8] Mikolov, T., Chen, K., Corrado, G. y Dean, J. (2013). Efficient estimation of word representations in vector space. *Computing Research Repository-arXiv*, abs/1301.3781.

[9] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. y Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, pp. 3111-3119.

[10] Jordan, M. I. (1990). Artificial neural networks. chapter *Attractor Dynamics and Parallelism in a Connectionist Sequential Machine*, pages 112–127. IEEE Press, Piscataway, NJ, USA.

[11] Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179-211.

[12] Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550-1560.

[13] Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673-2681.

[14] Bengio, Y., Simard, P. Y. y Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *Institute of Electrical and Electronics Engineers Transactions on Neural Networks*, 5(2):157-166.

[15] Hochreiter, S. y Schmidhuber, J. (1996). LSTM can solve hard long time lag problems. In *Advances in Neural Information Processing Systems 9*, pp. 473-479.

[16] Gers, F. A., Schmidhuber, J. y Cummins, F. A. (2000). Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):2451-2471.

[17] Sutskever, I., Vinyals, O. y Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104-3112.

[18] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. y Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *Computing Research Repository-arXiv*, abs/1406.1078.

[19] Bahdanau, D., Cho, K. y Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *Computing Research Repository-arXiv*, abs/1409.0473.

[20] Vaswani, A., Shazeer, N., Parmar N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. y Polosukhin, I. (2017). Attention is all you need. *Computing Research Repository-arXiv*, abs/1706.03762.

[21] He, K., Zhang, X., Ren, S. y Sun, J. (2016). Deep residual learning for image recognition. In *CVPR*, pp. 770-778. IEEE Computer Society.

[22] Ba, L. J., Kiros, R. y Hinton, G. E. (2016). Layer normalization. *Computing Research Repository-arXiv*, abs/1607.06450.

[23] Press, O. y Wolf, L. (2017). Using the output embedding to improve language models. In *EACL (2)*, pp. 157-163. Association for Computational Linguistics.

[24] Qi, Y., Sachan, D., Felix, M., Padmanabhan, S. y Neubig, G. (2018). When and why are pre-trained word embeddings useful for neural machine translation? In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 529-535. Association for Computational Linguistics.

[25] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. y Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *CVPR*, pp. 2818-2826. IEEE Computer Society.

[26] AbuShawar, B., Atwell, E. (2015). ALICE Chatbot: Trials and Outputs. *Computación y Sistemas*, vol. 19, nº 4.

[27] das Graças Bruno Marietto, M., Varago de Aguiar, R., de Oliveira Barbosa, G., Tanaka Botelho, W., Pimentel, E., dos Santos França, R., da Silva V. L. (2013). Artificial Intelligence MArkup Language: A Brief Tutorial. *Computing Research Repository-arXiv*, abs/1307.3091.

[28] Udani, M., Shrivas, A., Shukla, V., Rao, A. (2013). Question Answering System Based on Artificial Intelligence for Restricted Domain. *International Journal of Engineering Research & Technology*, vol. 2, Issue 12.

[29] Moldovan, D., Harabagiu, S., Gîrju, R., Morarescu, P., Lacatusu, F., Novischi, A., Badulescu, A., Bolohan, O. (2002). LCC Tools for Question Answering. In *Eleventh Text REtrieval Conference*. NIST Special Publication, vol. 500-251.

[30] Yang, H., Chua, T. (2002). The Integration of Lexical Knowledge and External Resources for Question Answering. In *Eleventh Text REtrieval Conference*. NIST Special Publication, vol. 500-251.

[31] Magnini, B., Negri, M., Prevete, R., Tanev, H. (2002). Mining Knowledge from Repeated Co-Occurrences: DIOGENE at TREC 2002. In *Eleventh Text REtrieval Conference*. NIST Special Publication, vol. 500-251.

[32] Light, M., Mann, G. S., Riloff, E., Breck, E. (2001). Analyses for Elucidating Current Question Answering Technology. *Journal of Natural Language Engineering*, 7(4), 325-342.

[33] Hovy, E., Gerber, L., Hermjacob, U., Junk, M., Lin, C. (2000). Question Answering in Webclopedia. *Ninth Text REtrieval Conference*. NIST Special Publication, vol. 500-249, pp. 655-664.

[34] Odgen, B., Cowie, J., Ludovik, E., Molina-Salgado, H., Nirenburg, S., Sharples, N., Sheremtyeva, S. (1999). CRL's TREC-8 Systems Cross-Lingual IR and Q&A. *Eigth Text REtrieval Conference*. NIST Special Publication, vol. 500-246, pp. 513-522.

[35] Moldovan, D., Harabagiu, S., Pasca, M., Mihalcea, R., Goodrum, R., Gîrju, R., Rus, V. (1999). LASSO: A Tool for Surfing the Answer Net. *Eigth Text REtrieval Conference*. NIST Special Publication, vol. 500-246, pp. 175-184.

[36] Cormack, G. V., Clarke, C. L. A., Palmer, C. R., Kisman, D. I. E. (1999). Fast Automatic Passage Ranking (MultiText Experiments for TREC-8). *Eigth Text REtrieval Conference*. NIST Special Publication, vol. 500-246, pp. 735-742.

[37] Allan, J., Connel, M., Croft, W., Feng, F., Fisher, D., Li, X. (2000). INQUERY and TREC-9. *Ninth Text REtrieval Conference*. NIST Special Publication, vol. 500-249, pp. 551-562.

[38] Fuller, M., Kaszkiel, M., Kimberley, S., Zobel, J., Wilkinson, R., Wu, M. (1999). The RMIT/CSIRO Ad Hoc, Q&A, Web, Interactive, and Speech Experiments at TREC-8. *Eigth Text REtrieval Conference*. NIST Special Publication, vol. 500-246, pp. 549-564.

[39] Hermjacob, U., Echinabi, A., Marcu, D. (2002). Natural Language Based Reformulation Resource and Wide Exploitation for Question Answering. In *Eleventh Text REtrieval Conference*. NIST Special Publication, vol. 500-251.

[40] Soubbotin, M., Soubbotin, S. (2002). Use of Patterns for Detection of Likely Answer Strings: A Systematic Approach. In *Eleventh Text REtrieval Conference*. NIST Special Publication, vol. 500-251.

[41] Lin, J., Fernandes, A., Katz, B., Marton, G., Tellex, S. (2002). Extracting Answers from the Web Using Data Annotation and Knowledge Mining Techniques. In *Eleventh Text REtrieval Conference*. NIST Special Publication, vol. 500-251.

[42] Chu-Carrol, J., Prager, J., Welty, C., Czuba, K., Ferrucci, D. (2002). A Multi-Strategy and Multi-Source Approach to Question Answering. In *Eleventh Text REtrieval Conference*. NIST Special Publication, vol. 500-251.

[43] Lee, G., Seo, J., Lee, S., Jung, H., Cho, B., Lee, C., Kwak, B., Cha, J., Kim, D., An, J., Kim, H., Kim, K. (2001). SiteQ: Engineering High Performance QA system Using Lexico-Semantic Pattern Matching and Shallow NLP. In *Tenth Text REtrieval Conference*. NIST Special Publication, vol. 500-250.

[44] Oard, D. W., Wang, J., Lin, D., Soboroff, I. (1999). TREC-8 Experiments at Maryland: CLIR, QA and Routing. *Eigth Text REtrieval Conference*. NIST Special Publication, vol. 500-246, pp. 623-636.

[45] Ittycheriah, A., Roukos, S. (2002). IBM's Statistical Question Answering System-TREC 11. In *Eleventh Text REtrieval Conference*. NIST Special Publication, vol. 500-251.

[46] Xu, J., Licuanan, A., May, J., Miller, S., Weischedel, R. (2002). TREC 2002 QA at BBN: Answer Selection and Confidence Estimation. In *Eleventh Text REtrieval Conference*. NIST Special Publication, vol. 500-251.

[47] Attardi, G., Cisternino, A., Formica, F., Simi, M., Tommasi, A. (2002). PIQASso 2002. In *Eleventh Text REtrieval Conference*. NIST Special Publication, vol. 500-251.

[48] Litkowski, K. C. (2002). Question Answering Using XML-Tagged Documents. In *Eleventh Text REtrieval Conference*. NIST Special Publication, vol. 500-251.

[49] Greenwood, M. A., Roberts, I., Gaizauskas, R. (2002). The University of Sheffield TREC 2002 Q&A System. In *Eleventh Text REtrieval Conference*. NIST Special Publication, vol. 500-251.

[50] Devlin, J., Chang, M., Lee, K., Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Computing Research Repository-arXiv*, abs/1810.04805.

[51] Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., Zettlemoyer, L. (2018). Deep contextualized word representations. *Computing Research Repository-arXiv*, abs/1802.05365.

[52] Radford, A., Narasimhan, K., Salimans, T., Sutskever, I. (2018). Improving language understanding with unsupervised learning. Technical report, OpenAI.

[53] Taylor, W. L. (1953). Cloze procedure: A new tool for measuring readibility. *Journalism Bulletin*, 30(4):415-433.

[54] Jernite, Y., Bowman, S. R., Sontag, D. (2017). Discourse-based objectives for fast unsupervised sentence representation learning. *Computing Research Repository-arXiv*, abs/1705.00557.

[55] Logeswaran, L., Lee, H. (2018). An efficient framework for learning sentence representations. In *International COnference on Learning Representations*.

[56] Parikh, A. P., Täckström, O., Das, D., Uszkoreit, J. (2016). A decomposable attention model for natural language inference. In *EMNLP*.

[57] Seo, M., Kembhavi, A., Farhadi, A., Hajishirzi, H. (2017). Bidirectional attention flow for machine comprehension. In *ICLR*.

[58] Peris, A., Casacuberta, F. (2018). NMT-Keras: a Very Flexible Toolkit with a Focus on Interactive NMT and Online Learning. *Computing Research Repository-arXiv*, abs/1807.03096.

[59] Papineni, K., Roukos, S., Ward, T., Zhu, W. J. (2002). BLEU: a method for automatic evaluation of machine translation. In *ACL-2002: 40th Annual meeting of the Association for Computational Linguistics*, pp. 311-318.

[60] Sasaki, Y. (2007). The truth of the F-measure. In *Teach Tutor Mater*.

[61] Kingma, D., Ba, J. (2014). Adam: A method for stochastic optimization. *Computing Research Repository-arXiv*, abs/1412.6980.

[62] Artetxe, M., Ruder, S., Yogatama, D. (2019). On the cross-lingual transferability of monolingual representations. *Computing Research Repository-arXiv*, abs/1910.11856.

[63] Lewis, P., Oguz, B., Rinott, R., Riedel, S., Schwenk, H. (2019). MLQA: Evaluating Cross-lingual Extractive Question Answering *Computing Research Repository-arXiv*, abs/1910.07475.

[64] Zhang, Y., Xu, Z. (2019). BERT for Question Answering on SQuAD 2.0.