DEEP LEARNING APLICADO AL RESUMEN DE TEXTOS

Ángel Javier Alonso Hernández

Trabajo de fin de grado
Doble grado en Ingeniería Informática y Matemáticas
Universidad Complutense de Madrid

16 de junio de 2017

Director:
Alberto Díaz Esteban

Índice general

Agradecimientos

A mi director por toda la ayuda prestada.
A NVIDIA por la donación de una tarjeta gráfica.
A mi padre.

Deep learning aplicado al resumen de textos

Resumen

Estudiamos la aplicación del deep learning a la tarea del resumen automático y abstractivo de textos. Para ello, hemos realizado un extenso estudio de la bibliografía y, siguiendo las últimas líneas de investigación, diseñamos nuestro propio modelo, que cuenta con una arquitectura encoder-decoder y un mecanismo de atención. Este modelo lo entrenamos sobre una tarjeta gráfica donada por NVIDIA. Finalmente, evaluamos los resultados obtenidos: aunque muchos de los resúmenes son buenos, identificamos algunos problemas como la repetición de frases y la falta de vocabulario.

Palabras clave: deep learning, procesamiento de lenguaje natural, resumen automático.

Deep learning applied to text summarization

Abstract

We study the application of deep learning techniques to the task of automatic abstractive summarization. For this purpose, we have done a broad study of the literature and, following in the line of recent research, we design our own model, with an encoder-decoder architecture and a mechanism of attention. We train this model using a GPU that was granted to us by NVIDIA. Lastly, we evaluate the results we obtained: although plenty of summaries are good, we identify some problems like the repetition of phrases and the lack of vocabulary.

Keywords: deep learning, natural language processing, automatic summarization.

Capítulo 1

Introducción

La materia prima del siglo XXI es la información. El vehículo por excelencia de la información es el texto. Tiene sentido, entonces, el desarrollo de métodos para procesar el texto. En particular, nos interesa el resumen. Sin embargo, este es difícil de modelizar; pues hace uso del lenguaje, formando conceptos y componiéndolos para formar ideas para representar un mundo complejo y abierto a la interpretación. Esto es, precisamente, lo que hace tan interesante la tarea del resumen. No tratamos con una secuencia de símbolos, sino con una serie de conceptos que apelan a todas las facetas del ser humano. El resumen nos plantea dos grandes desafíos: el uso del lenguaje y el entendimiento del contexto en el que vivimos.

Formalizar el lenguaje a partir de una colección de reglas, tal como prentendía Chomsky, no ha dado grandes resultados en la tarea del resumen. Definir estas reglas es complicado. Nuestra aproximación va a ser inferir las reglas de los datos. Pero no sólo las reglas: también los conceptos que representan las palabras y al relación entre ellos, que al fin y al cabo son más importantes para producir un buen resumen. Para ello utilizaremos el *deep learning*.

El *deep learning* es una rama del *machine learning*, o aprendizaje automático. Mientras que el *machine learning* se puede caracterizar por hacer predicciones basadas en observaciones pasadas, el *deep learning* se fundamenta en predecir formando representaciones adecuadas de la entrada. Dados un gran número de pares entrada-salida deseados, los sistemas *deep learning* funcionan aplicando a la entrada una serie de transformaciones aprendidas hasta poder predecir la salida deseada para una entrada nueva. Una de las claves del *deep learning* es diseñar un sistema relativamente simple que sea capaz de aprender de los ejemplos, más que un sistema excesivamente complicado con reglas definidas a mano. Las representaciones intermedias de la entrada que producimos en el proceso se pueden considerar equivalentes a

las ideas mentales que formamos los humanos en el cerebro al leer un texto. Esto está relacionado con la gran motivación del trabajo: dilucidar los principios mediante los cuales el deep learning ha sido capaz de resolver tareas antes consideradas sólo propias de los humanos. En efecto, el deep learning ha obtenido grandes resultados en el reconocimiento de imágenes [Szegedy et al., 2015], la transcripción del habla [Xiong et al., 2016] y la traducción automática [Wu et al., 2016], situándose en muchas de estas tareas al nivel humano.

1.1. Objetivos

Cuando le propuse a mi director el trabajo tenía dos objetivos en mente: hacer deep learning y hacer resúmenes. En un principio la tarea parecía difícil. Para entrenar los modelos se usan tarjetas gráficas caras y potentes y en la facultad no contabamos con ese equipamiento. Pedimos a NVIDIA una tarjeta gráfica y afortunadamente nos fue concedida. Estamos muy agradecidos a NVIDIA y estamos seguros que este nuevo hardware impulsará varios proyectos deep learning en la facultad.

En septiembre del año pasado, cuando comenzamos a trabajar, la tarea del resumen mediante deep learning estaba empezando a despegar. [Nallapati et al., 2016b] empezó a abrir el camino. Este artículo adapta un modelo orientado a la traducción automática y obtiene resultados en aquel momento del estado del arte. Además, introducía un dataset (ya usado por otros para otras tareas [Hermann et al., 2015]) para resumir textos largos en varias frases, mientras que los anteriores trabajos aplicando deep learning sólo resumían una frase en otra más corta [Rush et al., 2015, Chopra et al., 2016]. Este dataset, a diferencia del Gigaword, que era el que venía siendo usado, no requería de licencia. Según ibamos progresando en el estudio, prácticamente cada mes se publicaban nuevos artículos [Ramachandran et al., 2016, Kikuchi et al., 2016, See et al., 2017, Nallapati et al., 2016a]. Finalmente, el objetivo se definió como el siguiente: estudiar las técnicas y la bibliografía, intentar replicar los últimos resultados y evaluar de primera mano los resúmenes producidos.

1.2. Procesamiento del lenguaje natural

Nuestro trabajo se enmarca en el campo del procesamiento del lenguaje natural (o NLP, de sus siglas en inglés), que es el encargado de estudiar la comunicación hombre-máquina. Este campo goza de una larga tradición

científica. Existen numerosas introducciones al campo, por ejemplo, [Nadkarni et al., 2011], y la overview de Stanford[1]. Aquí nos limitamos a situar nuestro trabajo en el campo.

Desde los inicios del NLP existen dos vertientes: una basada en reglas lógicas sobre las que contruir el lenguaje y otra probabilística. La primera se fundamenta en las ideas de Chomsky y la teoría de lenguajes formales. Para Noam Chomsky existe una gramática universal que genera el lenguaje natural.

Nuestro trabajo se encuentra dentro la corriente probabilística, que en las últimas décadas ha obtenido mejores resultados empíricos. Al igual que en [Cho, 2015], adquirimos un punto de vista funcional del lenguaje. El lenguaje, en lo referente a nuestro estudio, sólo está determinado por su uso entre emisor y receptor en un contexto determinado. Esto es diferente a las ideas de Chomsky en las cuales el lenguaje es una entidad aislada que no requiere del mundo real. Bajo este punto de vista podemos definir el resumen como una función que toma un texto de entrada, T, y un contexto del mundo, C, y produce un texto T' más corto que T pero con su información más relevante. Modelar esta función sobre unas reglas y una base de conocimiento sería sumamente complicado. Sin embargo, podemos aproximarla haciendo uso del aprendizaje automático, que, como veremos más adelante, es en esencia un método de aproximación de funciones.

1.3. Estructura del documento

Tras esta introducción, el capítulo 2 describe la tarea del resumen y sitúa el trabajo en él. Además, se estudia como se puede evaluar automáticamente un resumen. El capítulo 3 expone las técnicas deep learning que utilizamos y su metodología. El hilo conductor del capítulo será nuestro modelo, pues primero lo planteamos y después vamos describiendo los elementos que lo componen, explicando la motivación detrás de las decisiones tomadas. Procedemos, de esta manera, de arriba a abajo, lo cual esperamos que facilite la comprensión. El capítulo 4 está dedicado al entorno de trabajo; esto es, las herramientas que hemos utilizado, desde la GPU a las librerías. Llegado al capítulo 5 describimos nuestros experimentos y discutimos los resultados. Finalmente, el capítulo 6 consiste en las conclusiones y un planteamiento del trabajo futuro.

[1] `http://cs.stanford.edu/people/eroberts/courses/soco/projects/2004-05/nlp/overview_history.html`

Capítulo 2

Resumen de textos

2.1. La tarea del resumen

[Mani et al., 1999] definen la tarea del resumen como:

> The process of distilling the most important information from a source (or sources) to produce an abridged version for a particular user (or user) and task (or tasks).

Existen dos corrientes en la tarea del resumen: una *extractiva* y otra *abstractiva*. La corriente extractiva genera resúmenes copiando fragmentos (generalmente frases) del texto original. Los métodos abstractivos generan palabras y frases desde cero, que pueden no encontrarse en el texto, tal como hacemos las personas. La aproximación extractiva es más facil porque garantiza una mínima coherencia linguística. Una posibilidad es hacer un estudio estadístico de las palabras más prominentes y copiar las frases en las que se encuentran; de esta manera podemos producir un resumen sin verdadera comprensión del lenguaje. Por otro lado, los métodos abstractivos deben poseer un dominio más profundo del lenguaje, comúnmente usan la paráfrasis para formar frases más cortas y son capaces de hilar las frases. Nuestro trabajo se enmarca en la corriente abstractiva.

Por otra parte, podemos resumir un documento o varios. Tradicionalmente los sistemas de resumen se han orientado al resumen de un documento. El resumen de múltiples textos encuentra su motivación en la red, y se centra en extraer información de varios textos escritos sobre el mismo tema.

El contenido del resumen puede ser indicativo, que consiste en una serie de datos aislados (longitud, palabras clave, estilo, etc) con el objetivo de proporcionar una idea al lector, o informativos, que se puede leer en lugar del documento original.

Por otra parte, podemos producir un párrafo, palabras clave o bien una sola frase (lo que sería el titular). La mayor parte de la tradición investigadora está enfocada sobre la producción de párrafos. Generar titulares tiene la gran ventaja de que existen millones de pares de referencia artículo-titular en la web, con los cuales es fácil construir un dataset para entrenar modelos deep learning. Existe una tarea, relacionada con el resumen, que consiste en la compresión de frases (*sentence compression* o *sentence summarization*). El deep learning ha obtenido buenos resultados en esta tarea [Rush et al., 2015, Chopra et al., 2016] y ha sido el germen de los modelos de resumen de textos más largos [Nallapati et al., 2016b].

2.2. Evaluación de resúmenes

Una de las dificultades de la tarea del resumen es evaluar los resúmenes producidos, ya que es complicado especificar formalmente lo que un buen resumen debe capturar. [Hovy, 2003] define dos pautas generales para un buen resumen:

- Debe ser más corto que el texto original.
- Debe contener la información más importante del texto original y no otra que no aparezca.

Cuando tenemos resúmenes producidos por una persona (resumen de referencia) para comparar con los resúmenes producidos por el sistema podemos usar las métricas *recall* y *precision*. Para ello dividimos los resúmenes de referencia y del sistema en unidades, que pueden ser frases o n-gramas.

Un n-grama es una subcadena de n elementos. Por ejemplo, los 1-gramas son las palabras, mientras que los 2-gramas (o bigramas) son todos los pares de palabras contiguas. En la frase "/police killed the gunman/" hay tres 2-gramas: "police killed", "killed the" y "the gunman".

Calculamos las métricas *precision* and *recall* de la siguiente manera:

$$\text{precision} = \frac{\text{correct}}{\text{correct} + \text{wrong}}, \tag{2.1}$$

$$\text{recall} = \frac{\text{correct}}{\text{correct} + \text{missed}}, \tag{2.2}$$

donde *correct* es el número de unidades presentes tanto en el resumen de referencia como en el del sistema, *wrong* el número de unidades sólo presentes en el del sistema y *missed* el número de unidades en el de referencia pero no

en el del sistema. Conocidos el *precision* y el *recall*, es habitual combinarlos para producir la métrica F_1, que es dos veces la media harmónica de ambos:

$$F_1 = 2\frac{\text{precision recall}}{\text{precision} + \text{recall}}. \tag{2.3}$$

Otra métrica importante es la subsecuencia común más larga (*longest common subsequence*, LCS). Una secuencia $Z = z_1, \ldots, z_n$ es una subsecuencia de otra secuencia $X = x_1, \ldots, x_n$ si existe una secuencia creciente de índices $i_1, \ldots, i_k$ tal que para todo $j = 1, 2, \ldots, k$ tenemos $x_{ij} = z_j$. La métrica LCS calcula la longitud de la subsecuencia común a ambos resúmenes más larga. Para facilitar la exposición introducimos las siguientes frases extraídas de [Lin, 2004]:

- S_1: "police killed the gunman".
- S_2: "police kill the gunman".
- S_3: "the gunman kill police".

Consideramos S_1 como el resumen de referencia de un texto y S_2 y S_3 como dos resúmenes generados por dos sistemas diferentes y los dividimos en bigramas. Vemos que S_2 y S_3 tienen la misma métrica *precision* y *recall*, porque sólo contienen el bigrama "the gunman" común con S_1. Sin embargo, el significado de S_2 y S_3 es contrario. La métrica LCS nos proporciona una visión más cercana a la realidad, ya que es consciente del orden de las palabras. La subsecuencia más larga común a S_1 y a S_2 es "police the gunman". Para el par S_1 y S_3, la subsecuencia común más larga es "the gunman". Entonces la métrica LCS para S_2 es 3 y para S_3 es 2.

Para calcular estás métricas se utiliza el paquete ROUGE [Lin, 2004]. Principalmente se usan tres, aunque ROUGE es capaz de calcular más. Estas son:

- ROUGE-1: que se corresponde con las métricas *precision* y *recall* dividiendo los textos en 1-gramas.
- ROUGE-2: que se corresponde con las métricas *precision* y *recall* dividiendo los textos en 2-gramas.
- ROUGE-L: que se corresponde con la métrica LCS.

Además, [Lin, 2004] muestra que existe correlación entre estas métricas y la valoración de los resúmenes por personas.

Capítulo 3

Deep learning y el resumen

En este capítulo exponemos las técnicas deep learning que utilizamos. Situamos el foco sobre los detalles particulares al resumen, explicando el porqué de nuestras decisiones.

En todo sistema deep learning hay dos actores principales: el modelo y el dataset. El dataset con el que vamos a trabajar, que se llama *CNN-DailyMail* [Hermann et al., 2015], consiste en aproximadamente 300 000 pares de artículos periodísticos y su resumen. Veremos como el modelo es capaz de inferir el lenguaje y sus reglas de los datos. Usualmente, el dataset se dividie en tres conjuntos:

- Training set: son los pares sobre los que se entrena el modelo. En nuestro caso 287 227 artículos.
- Validation set: se usa para ajustar los parámetros que definen el proceso de entrenamiento, también llamados hiperparámetros. Lo conforman 13 368 pares.
- Test set: el conjunto sobre el que se hace el análisis final de los resultados. Es importante que no se tome ninguna decisión en el modelado en base a resultados obtenidos en el test set; el test set juega un papel de testigo imparcial. Esto se debe a que el objetivo del aprendizaje automático es generalizar: ajustamos los parámetros sobre el training set pero la evaluación de los resultados se hace sobre el test set, cuyos elementos el modelo nunca ha visto en el proceso de entrenamiento. Este conjunto está compuesto por 11 490 pares.

La división que usamos es la misma que [Hermann et al., 2015], y que comparten los artículos ciéntificos que discutimos [Nallapati et al., 2016b, See et al., 2017]. Más detalles sobre el dataset se encuentran en la sección 5.1. A continuación exponemos nuestro modelo.

3.1. El modelo

La definición de la tarea del resumen proporcionada en la introducción no nos ayuda a la hora de construir un modelo. Necesitamos una definición matemática para empezar a aproximar la tarea. Para ello, como en [Cho, 2015], vamos a considerar un punto de vista funcional. Consideramos una persona que toma un texto A y produce un resumen B, para ello utilizará un contexto del mundo C, que incluye las reglas de su lenguaje, su vocabuluario e información sobre historia, geografía y demás, esto se traduce en:

$$f(A; C) = B, \tag{3.1}$$

que es una función abstracta y sólo nos sirve conceptualmente. Sin embargo, podemos reunir una colección de pares $\{(A_i, B_i)\}$ texto-resumen con los que aproximar f. Entonces, nuestro objetivo es obtener una función g de la forma:

$$g(A; \theta) = B, \tag{3.2}$$

donde A es el texto a resumir, B el resumen y θ un conjunto de parámetros. Está función es el grafo computacional del modelo deep learning que construiremos y θ los parámetros de tal modelo. g debe estar bien definida y todavía quedan muchos elementos por precisar: ¿cuál es el formato de los textos A y B? ¿qué forma tiene g? ¿cómo se obtienen los parámetros? Estas y otras preguntas son las que responderemos en el resto de la memoria.

Los parámetros θ son una representación del contexto del mundo C de (3.1). Este hecho resalta la dificultad de la tarea del resumen, pues debe ser capaz de sintetizar las reglas del lenguaje, el vocabulario, que India es un país, que la capital de Alemania es Berlín... Formalizar estos conceptos manualmente es imposible, pero descubriremos que los modelos deep learning son capaces de aprenderlos y usarlos satisfactoriamente.

3.2. El texto

Vamos a comenzar definiendo los textos A y B de (3.2). Consideraremos un texto como una secuencia de elementos: ya sean palabras, signos de puntación u otros elementos semánticos.

$$A = (x_1, \ldots, x_n), \tag{3.3}$$

$$B = (y_1, \ldots, y_m). \tag{3.4}$$

A cada uno de los elementos de la secuencia los denominamos *tokens* [Manning et al., 2008].

Para poder computar con ellos, necesitamos una representación de los tokens. Vamos a considerar un token como un elemento de un conjunto ordenado finito, llamado *vocabulario*. Entonces podemos asignar a cada token un número, correspondiente a su posición en el vocabulario. Si queremos ser capaces de representar cualquier elemento semánticos de cualquier texto el tamaño de este vocabulario sería intratable. Por esto, se suele trabajar con vocabularios de cardinalidad reducida, en general del orden de los 100000 elementos.

3.3. Word embeddings

En la anterior sección hemos definido un token como un elemento de una colección, el vocabulario, y le hemos asignado un número correspondiente a su posición en el vocabulario. En la práctica en vez de un número se usa un vector de K elementos cero o uno, donde K es el tamaño del vocabulario, con todos sus elementos a cero salvo el correspondiente a la posición de la palabra en el vocabulario. Esta representación se llama *1-of-K coding*. Sin embargo, no nos ofrece ninguna información relevante para el resumen. Todas las palabras están igual de relacionadas, no podemos obtener significado de la representación. Nos interesa obtener una representación que preserve las relaciones, que "madre" esté cerca de "padre", "España" cerca de "país", etc. Como es común en el deep learning, vamos a dejar que la representación sea aprendida a partir de los datos.

Dado un vector *1-of-K coding* lo proyectamos en un espacio vectorial denso mediente una matriz E, cuyos elementos son parámetros a aprender. Entonces para la palabra i con vector *1-of-K coding* v_i obtenemos:

$$w_i = Ex_i, \tag{3.5}$$

donde w_i es un vector de dimensión prefijada (generalmente entre 128 y 512, dependiendo de la cantidad y profundidad de las relaciones que queramos modelar y la capacidad computacional y de memoria). La matriz E tendrá tantas columnas como palabras en el vocabulario y tantas como filas como la dimensión del vector w_i. En la figura 3.1 se muestra un ejemplo de este proceso.

Mientras que en el *1-of-K coding* las palabras eran elementos equidistantes y estaban dispersas, la representación obtenida es densa; hemos pasado de vectores de dimensión desde 50000 hasta 100000 (dependiendo del tamaño del vocabulario) a dimensión desde 128 a 512. [Cho et al., 2014a] muestran como el entrenamiento de esta representación produce vectores que preservan

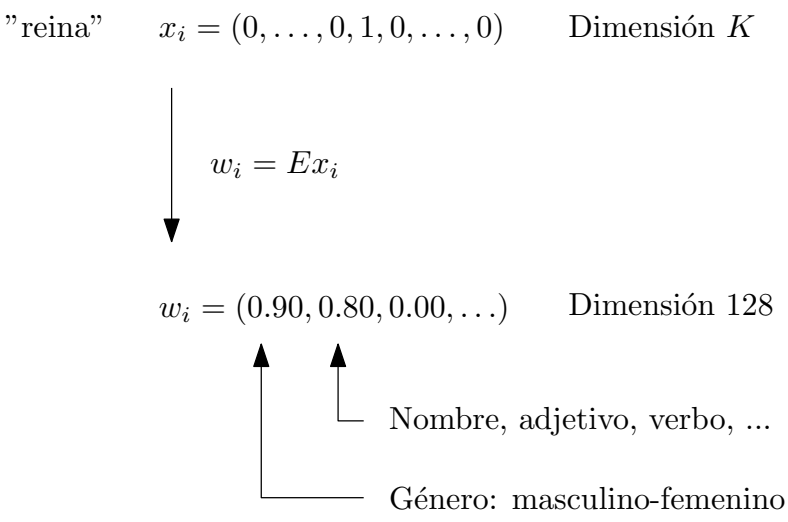

Figura 3.1: Ejemplo del proceso de inmersión para la palabra "reina". Se ha comprobado que tras el entrenamiento el sistema ha aprendido a destilar propiedades de las palabras como elementos del vector [Cho et al., 2014a]. Aquí mostramos el género y la categoría sintáctica a modo de ejemplo muy simplificado. En la práctica es díficil asociar a cada elemento una característica.

las relaciones entre las palabras; es decir, palabras similares se encuentran relacionadas en el espacio vectorial denso.

El entrenamiento de representaciones densas es una de las claves del deep learning. Más adelante veremos una representación densa de todo un texto, que va a ser fundamental para producir resúmenes. De la misma manera que para las palabras, haremos una inmersión los textos (de longitud variable) en un espacio vectorial de dimensión fija. En el caso de las palabras es más facíl porque contamos con un vector *1-of-K coding* de dimensión fija, con lo cual podemos usar una transformación lineal. El texto requerirá de transformaciones más complejas. Es importante hacer notar que aunque en el paso del vector *1-of-K coding* al vector denso la tranformación es lineal, los parámetros de la transformación se obtienen a través de un proceso no lineal; i.e. estos parámetros se entrenan para maximizar una función objetivo generalmente no lineal.

Intuitivamente, estos vectores densos se corresponden con los conceptos mentales que la mente forma al leer una palabra o un texto. O, en otras palabras, estos vectores pretenden ser análogos al conjunto de señales neuronales que se producen en el cerebro al leer una palabra específica. Cuanto mejor sea la equivelencia entre conceptos mentales y vectores mejores serán los resultados. Esta equivalencia se infiere del conjunto de datos de entrenamiento.

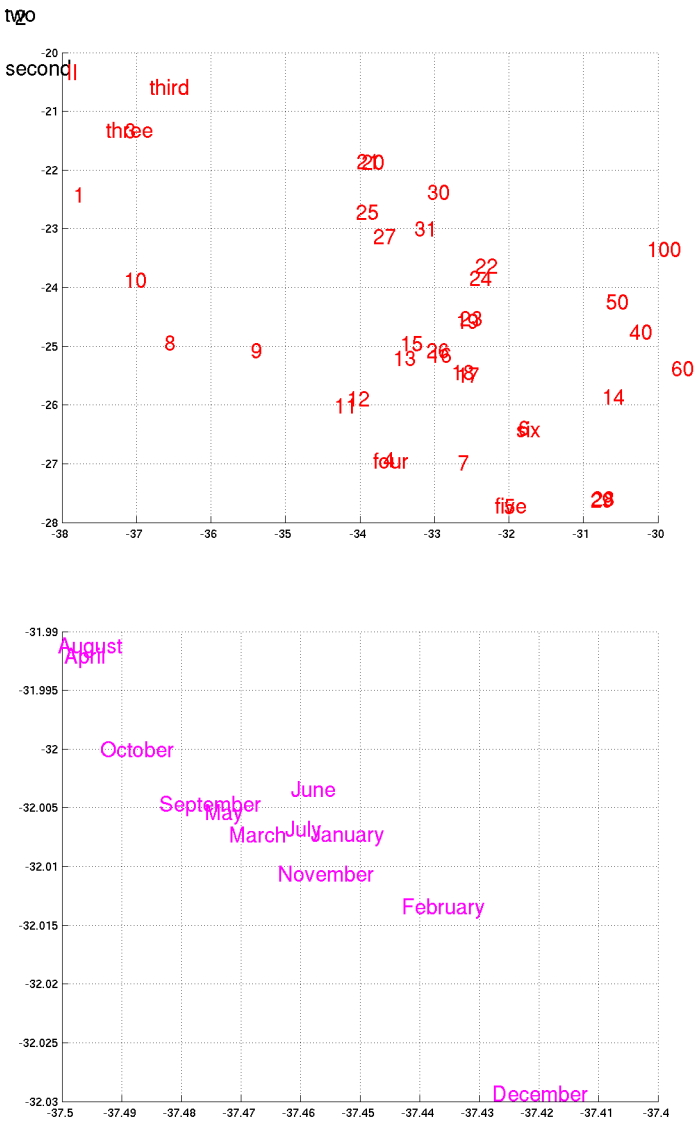

Figura 3.2: Proyección de algunos word embeddings en dos dimensiones. Estos gráficos pertenecen a [Cho et al., 2014a] y se reproducen aquí con permiso de sus autores. Son el resultado de entrenar un modelo de traducción automática, pero la parte referente a los word embeddings es la misma que en nuestro caso y tras el entrenamiento de nuestro modelo se obtienen vectores que guardan relaciones parecidas.

3.4. Relación entre las secuencias

Una vez definidos A y B estudiamos las relaciones entre ellos como base para más tarde definir g. Con este objetivo, definimos un modelo probabilístico. Esto es, podemos enfocar la tarea del resumen como encontrar el texto B que maximice:

$$p(B \mid A), \tag{3.6}$$

o, escrito en término de las secuencias de tokens:

$$p(y_1, \ldots, y_m \mid x_1, \ldots, x_n), \tag{3.7}$$

donde $B = (y_1, \ldots, y_m)$ y $A = (x_1, \ldots, x_n)$. Es más, aplicando la regla de Bayes obtenemos que:

$$p(B \mid A) \propto p(A \mid B)p(B), \tag{3.8}$$

donde llamamos a $p(B)$ el modelo del lenguaje (la probabilidad de que B esté en el lenguaje, i.e. la coherencia semántica y gramátical de B) y a $p(A \mid B)$ el modelo de resumen.

Esta es la misma relación que guardan una frase en un idioma y la misma frase en otro idioma, pues es una relación entre dos secuencia cualesquiera; las tareas se diferenciaran en las propias distribuciones de $p(B)$ y $p(A \mid B)$. De hecho, este modelo tiene sus origines en la traducción [Koehn et al., 2003].

Tanto la traducción como el resumen se enmarcan en un conjunto de tareas denominado *sequence to sequence learning*, comúnmente abreviado *seq2seq*, precisamente por la relación de (3.7). Más adelante expondremos que una herramienta potente para hacer frente a este tipo de tareas son las redes neuronales recurrentes [Sutskever et al., 2014].

[Nallapati et al., 2016b] se dan cuenta de la similitud entre ambas tareas y obtienen buenos resultados adaptando mínimamente (*off-the-shelf*) sistemas de traducción basados en el modelo probabilístico aquí descrito. En este trabajo nosotros seguimos el mismo camino.

3.5. Sequence to sequence

Estamos más cerca de definir g por completo, es aquella que calcula $p(B \mid A)$. Nótese como gran parte de la explicación anterior se refiere sólo a la relación entre dos secuencias: en este caso, el texto, A, y su resumen, B. Esta carácteristica es compartida con muchas otras tareas, como la traducción, en la que se relaciona una frase en un idioma con su traducción, y la transcripción

del habla, en la que se relaciona una serie de sonidos con una secuencia de palabras. Estamos, entonces, en una situación más general, conocida en la literatura como modelado *sequence to sequence* (o *seq2seq*) [Sutskever et al., 2014]. En las sucesivas describiremos cómo usamos las diferentes técnicas *sequence to sequence* para producir nuestros resúmenes. En particular, hacemos uso de redes neuronales.

3.5.1. Red neuronal feedforward

Las redes neuronales se enmarcan en el campo del machine learning y tienen como objetivo aproximar una función $f*$, definiendo una aplicación $f = (x; \theta)$ donde θ son los parámetros que mejor aproximan $h*$. Su forma básica se conoce como feedforward, pues la información fluye desde la entrada x, pasando por f y directamente a la salida; no existe ninguna retroalimentación (con la cual un valor intermedio se le volvería a aplicar h hasta que se de un criterio de parada), que carecteriza precisamente a las redes neuronales recurrentes.

Se llaman redes porque f es en realidad una composición de funciones. Esta composición se describe con un grafo dirigido acíclico, llamado grafo computacional, donde cada nodo es una función y las aristas representan la composición entre dos funciones.

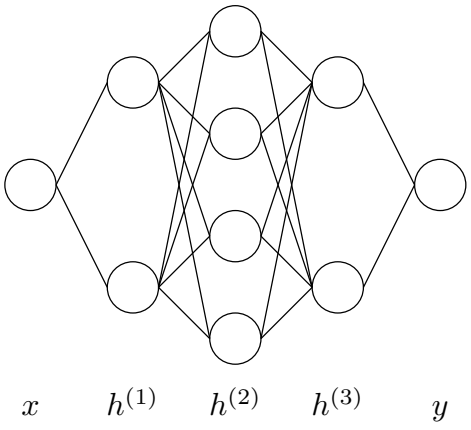

Figura 3.3: Ejemplo del grafo computacional de una red neuronal feedforward.

3.5.2. Redes neuronales recurrentes

En el caso del resumen estamos modelando relaciones entre secuencias $A = (a_1, \ldots, a_n)$ y $B = (b_1, \ldots, b_m)$ de longitud variable. En esta situación necesitamos usar una red neuronal recurrente (RNN). Si la longitud fuese fija podríamos usar una red neuronal feedforward con n entradas y m salidas.

Una RNN adquiere una representación de la entrada por medio de la recursión. Nuestro objetivo va a ser producir un vector h de dimensión fija que represente la secuencia $(a_1, \ldots, a_n)$. En cada paso de la recursión consumimos un elemento de la secuencia y producimos un vector representativo de los elementos consumidos hasta ese paso. Formalmente, en el paso t consumimos a_t y producimos h_t, resumen de $(a_1, \ldots, a_{t-1})$, aplicando la función que define la recursión, f:

$$h_t = f(x_t, h_{t-1}; \theta), \tag{3.9}$$

donde θ son los parámetros a aprender. La función f se denomina función de activación. La elección de una función de activación adecuada es fundamental para el éxito del modelo. Una posibilidad es usar una transformación afín de la entrada, pero en general, en la práctica, se usa la composición de una transformación lineal y una función no lineal. Esta función no lineal puede tomar muchas formas: desde las más simples, como la tangente hipérbolica y la función sigmoide, descrita en la figura 3.4, a otras más complejas, como lo que se conoce como unidades GRU, *gated recurrent units*, [Cho, 2015], que estudiaremos a continuación.

3.5.3. Unidades GRU

Las unidades GRU nacen con un objetivo en mente: dotar a la recurrencia de capacidad para recordar información. Más detalladamente: supongamos que nuestra función de activación es una tranformación lineal, entonces, en cada iteración t de la recurrencia, el estado h_t se encuentra predominadamente influenciado por los últimos elementos de la entrada. Sin embargo, puede suceder, por ejemplo, que el último elemento guarde gran relación con el primero. Luego, nuestra función de activación debe ser capaz de preservar dependencias por largo tiempo, lo que se conoce en la literatura como *long term dependencies*. A estos efectos, [Hochreiter and Schmidhuber, 1997] introducen las unidades *long short-term memory* (LSTM), las cuales aprenden a preservar o actualizar su estado en cada iteración de la recurrencia. Las unidades GRU [Cho et al., 2014a] tienen el mismo propósito que las LSTM pero son más simples y fáciles de entrenar. En la práctica, las unidades LSTM y GRU se consideran equivalentes [Chung et al., 2014]. Por estas razones hemos decidido usar unidades GRU y a continuación las describimos.

Sea $a_1, \ldots, a_n$ una secuencia de word embeddings, descritos en la sección 3.3 y sea k el tamaño prefijado de los vectores de estado h_t, es decir, $h_t \in \mathbb{R}^k$. Como nuestro objetivo es preservar la información, vamos a utilzar dos vectores $r, z \in [0, 1]^k$: r indicará en que grado queremos preservar el estado y

z el grado en el que queremos actualizarlo. Estas nociones cobrarán sentido pronto. En cada paso t de la recurrencia, estos valores los calculamos de la siguiente manera:

$$z = \sigma(W_z a_t + U_z h_{t-1} + b_z), \tag{3.10}$$

$$r = \sigma(W_r a_t + U_r h_{t-1} + b_r), \tag{3.11}$$

donde σ es la función sigmoide (aplicada elemento a elemento), W_z, U_z, W_r y U_r son matrices de parámetros a aprender y b_z, b_r son vectores. Las matrices W_z, U_z y el vector b_z definen una transformación afín.

La función sigmoide (figura 3.4) es muy usada en el deep learning. Su rango es $(0, 1)$ y esto nos permite interpretar la salida como el grado de confianza que tenemos en una afirmación (el parámetro de una distribución de Bernoulli [Goodfellow et al., 2016]). La función se satura en las colas y esto es congruente con la idea intuitiva de grado de confianza: cuando estamos muy seguros de una afirmación, obtener más información que sustente nuestra hipótesis no incrementa mucho nuestro grado de confianza. Sin embargo, cuando no estamos seguros, información nueva se traduce en un mayor incremento del grado de confianza en un sentido u otro.

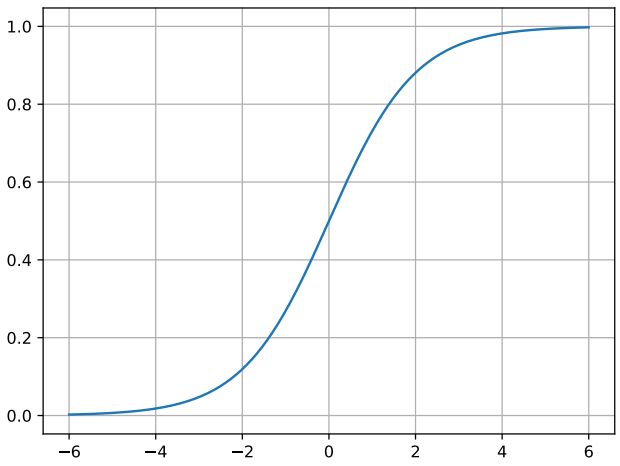

Figura 3.4: Función sigmoide. $\sigma(x) = \frac{1}{1+e^{-x}}$.

Ahora bien, como hemos dicho, r representa cuánto queremos preservar el estado. En el paso t de la recurrencia contamos con el estado h_{t-1} y queremos producir uno nuevo, h_t. Para ello primero calculamos un vector estado candidato, $\widetilde{h}_t$:

$$\widetilde{h}_t = \tanh(Wa_t + U(r \odot h_{t-1})), \tag{3.12}$$

donde W, U son matrices de parámetros a aprender, tanh la tangente hipérbolica (ver figura 3.5) y $\odot$ la multipliación elemento a elemento. Nótese que cuánto más cercano a 1 esté cada elemento de r, más queremos preservar el estado actual, h_{t-1}.

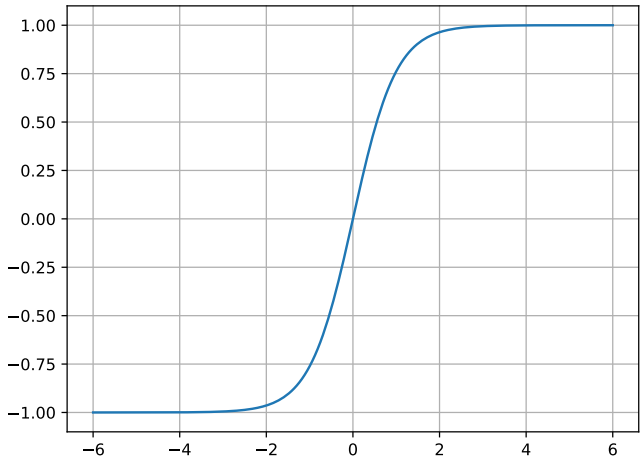

Figura 3.5: Gráfico de la tangente hipérbolica, tanh. El rango es $(-1, 1)$. Nótese que, como σ, satura en las colas.

Obtenido el vector estado candidato, calculamos el estado final, h_t:

$$h_t = z \odot h_{t-1} + (1 - z) \odot \widetilde{h}_t. \tag{3.13}$$

Cuánto más cercano a 1 esté cada elemento de z, más preservamos del estado anterior. Cuando $z = 0$ el paso t de la recurrencia no cambiará el estado.

3.6. Modelando probabilidades con RNNs

Hemos visto como una RNN adquiere una representación de la entrada mediante una recurrencia. Con el objetivo del resumen y a la vista de las ecuaciones de nuestro modelo, nos interesa que esta representación sea una probabilidad. Esto es, dada una secuencia de entrada $X = (x_1, \ldots, x_n)$ queremos escribir $p(X)$ como una recurrencia. Para ello factorizamos $p(X)$ usando la probabilidad condicional:

$$p(X) = p(x_1, \ldots, x_n) = p(x_1)p(x_2 \mid p_x 1) \cdots p(x_n \mid x_1, \ldots, x_{n-1}), \tag{3.14}$$

ahora en el paso t de la recurrencia modelamos $p(x_t \mid x_1, \ldots, x_{t-1})$:

$$p(x_t \mid x_1, \ldots, x_n) = g(h_{t-1}; \theta), \tag{3.15}$$

$$h_t = f(x_t, h_{t-1}; \theta), \tag{3.16}$$

donde g produce una distribución de probabilidad condicionada a h_{t-1} y f es la función de activación de la RNN. En nuestro caso, la distribución de probabilidad se obtiene mediante la función *softmax*:

$$p(x_t \mid x_1, \ldots, x_n) = g(h_{t-1}; \theta) = \frac{\exp(W_j h_{t-1})}{\sum_{j'=1}^{K} \exp(W_{j'} h_{t-1})}, \tag{3.17}$$

donde W es una matriz de parámetros a aprender, W_j indica la fila j y K es el número de elementos posibles. En el contexo de nuestro modelo tenemos una lista de vocabulario de K palabras, y dado un texto lo recorremos palabra a palabra intentado predecir la siguiente y cambiando los parámetros según acertamos o fallamos. Para producir un texto podemos ir generando la palabra más probable de acuerdo al contexto actual (un vector h_t) utilizando las mismas ecuaciones aquí descritas.

3.7. Encoder-decoder

En la sección anterior hemos trabajado con la probabilidad de una sola secuencia, $p(X)$. Sin embargo, para el resumen nos interesa modelar la probabilidad del resumen, B, condicionada al texto, A, $p(B \mid A)$.

Generalmente, cuando una persona va a resumir un texto primero adquiere una idea de él: cuál es el tema del texto, las personas que aparecen, los lugares, etc. A partir de este concepto del texto se escribe el resumen. Para modelizar $p(B \mid A)$ mediante RNNs vamos a usar un procedemiento análogo: tal como representamos las palabras (sus conceptos) como vectores, vamos a representar el concepto del texto como otro vector. Dado el texto $A = (a_1, \ldots, a_n)$ usamos una RNN como la descrita por la ecuación (3.9). Tras consumir el elemento a_n obtenemos un vector h_n que identificaremos como el resumen del texto, $c = h_n$.

Obtenido c, usamos otra RNN para producir desde cero otra secuencia $B = (b_1, \ldots, b_m)$, calculando cuál es la palabra más probable de acuerdo a la probabilidad esta vez condicionada al contexto c:

$$p(b_t \mid b_{t-1}, \ldots, b_1, c) = g(h_t, b_{t-1}, c; \theta), \tag{3.18}$$

donde g es la función softmax descrita en (3.17). El estado interno de la RNN lo actualizamos según:

$$h_t = f(h_{t-1}, b_{t-1}, c; \theta), \tag{3.19}$$

que esta vez depende de c. f es una unidad GRU, tal como se describe en la sección 3.5.3.

Esta arquitectura se llama *encoder-decoder*. La RNN que toma el texto A es el encoder (figura 3.9) y la segunda que produce una secuencia nueva, B, decoder (figura 3.7).

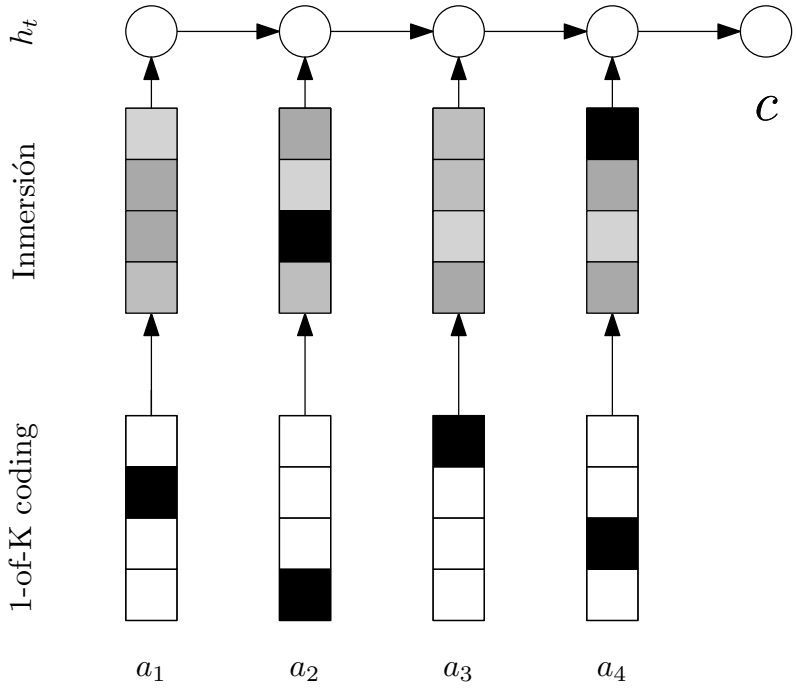

Figura 3.6: Esquema del encoder. Comenzamos con el 1-of-K coding y lo inmergimos en el espacio vectorial denso. h_t indica el estado de la recurrencia. En cada paso consumimos un elemento de la entrada y lo actualizamos.

3.8. RNN bidireccional

Hasta ahora, en la recurrencia del encoder consideramos cada estado h_t como representación del texto hasta el elemento a_t. Sin embargo, cuando las

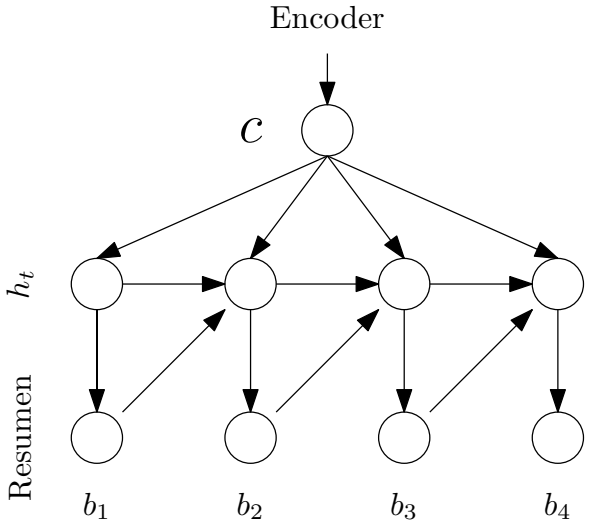

Figura 3.7: Esquema del decoder. La flechas de a a b indican que b depende de a. Obtenido el vector contextual, c, del encoder, comenzamos a generar palabras. La palabra escogida será aquella que maximice la ecuación (3.17). En este caso el estado de la recurrencia depende también de la palabra anteriormente generada.

personas leemos un texto tenemos conciencia del contexto de las palabras. No leemos todo de seguido, a veces nos paramos y volvemos a mirar atrás. Además, por la naturaleza secuencial del encoder el estado h_t tiende a estar dominado por las palabras consumidad más recientemente.

Para intentar dotar de contexto al estado de la recurrencia, usamos una RNN bidireccional [Schuster and Paliwal, 1997], introducida en el campo de la traducción por [Bahdanau et al., 2014]. Una RNN bidireccional consiste en duplicar nuestra RNN anterior: ahora una RNN consumirá las palabras en un sentido y la otra en el contrario. Ambas definidas por la misma recurrencia. Si consideramos una secuancia $A = (a_1, \ldots, a_n)$, esto se traduce a:

$$\overrightarrow{h}_t = f(a_t, \overrightarrow{h}_{t-1}; \theta), \tag{3.20}$$

$$\overleftarrow{h}_t = f(a_{n-t}, \overleftarrow{h}_{t-1}; \theta), \tag{3.21}$$

donde $\overrightarrow{h}_t$ y $\overleftarrow{h}_t$ son los estados de la primera y segunda RNN, respectivamente. Un diagrama se muestra en la figura 3.8. Antes teníamos el vector estado h_t asociado al elemento a_t, ahora asociamos al elemento a_t la concatenación de $\overrightarrow{h}_t$ y $\overleftarrow{h}_t$.

El decoder se mantiene igual, porque, intuitivamente (avalado por la práctica), para escribir un resumen sólo necesitamos la *idea* (el vector de

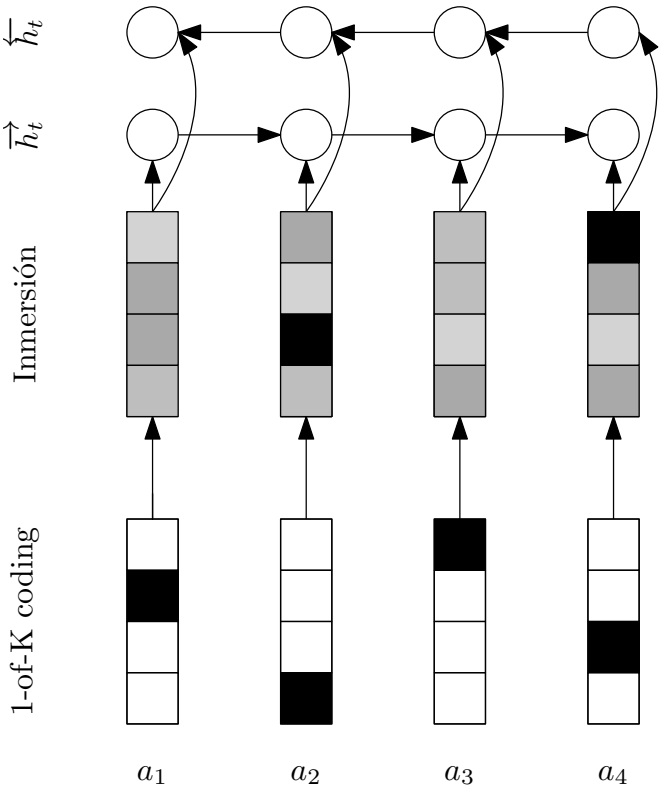

Figura 3.8: Esquema del encoder bidireccional.

contexto, c) para ir generando palabras.

3.9. Mecanismo de atención

Aunque la idea de resumir un texto en un vector de longitud fija, c, es intuitiva, esta no da buenos resultados. El problema radica en que, aunque la longitud del texto puede variar, la longitud del vector contexto es fija. Un vector contexto de esta forma puede carecer de capacidad representativa para frase más largas [Cho et al., 2014b]. Junto al uso de las RNN bidireccionales, [Bahdanau et al., 2014] introduce un mecanismo de atención con el objetivo de salvar este obstáculo. Ahora en cada paso de la recurrencia del decoder vamos a prestar atención a un subconjunto del texto original. Esta noción imita el comportamiento humano: según vamos generando partes del resumen nos fijamos en distintas partes del texto.

Procedemos a formalizar el mecanismo de atención. Ahora, en vez de tener un mismo vector c para todo la recurrencia del decoder vamos a usar uno diferente para cada paso, es decir, en el paso t usamos un vector c_t en

lugar de c en la ecuación (3.19). Este vector c_t representamos la parte del texto a la que estamos prestando atención. Falta, entonces, cómo calcular c_t. Sea $a_1, \ldots, a_n$ la secuencia del texto de entrada y sea $h_t = \overrightarrow{h}_{t-1}, \overleftarrow{h}_{t-1}$, tal como se describe en la sección 3.8. En el paso t de la recurrencia del decoder, la idea ahora es asignar a cada elemento $a_i, i = 1, \ldots, n$ del texto una puntuación que indicará cuánto vamos a atender esa palabra; calculamos para cada $i = 1, \ldots, n$:

$$e_{ti} = v_e^\top \tanh(U_e s_{t-1} + W_e h_i + b_e), \tag{3.22}$$

donde s_{t-1} es el anterior estado de la recurrencia del decoder y U_e, W_e, b_e y v_e parámetros a aprender. U_e y W_e (de una transformación afín) son matrices y b_e y v_e son véctores, de esta forma e_{ti} es un escalar.

Acto seguido, normalizamos estás puntuaciones para que sumen 1, de esta manera podemos interpretar esto valores como probabilidades. Una normalización de estás características se puede conseguir aplicando *softmax*:

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_{j=1}^{n} \exp(e_{tj})}. \tag{3.23}$$

Finalmente, calculamos el vector c_t:

$$c_t = \sum_{i=1}^{n} \alpha_{ti} h_i. \tag{3.24}$$

Un diagrama del modelo resultante se encuentra en la figura **??**.

3.10. Entrenamiento y estimación

Llegados aquí ya tenemos nuestro modelo listo. Sin embargo, tenemos un conjunto de parámetros, θ, todavía desconocido. Durante la parte anterior de la memoria hemos hablado de parámetros a estimar mediante un entrenamiento. ¿Cómo hacemos este entrenamiento?

Para estimar nuestros parámetros θ recurrimos a la estadística. El principio de máxima verosimilitud (*maximum likelihood principle*) es un método habitual para producir estimadores. Supongamos que tenemos un conjunto de pares $\mathbb{X} = \{(A_1, B_1), \ldots, (A_n, B_n)\}$, donde A_i es un texto y B_i su resumen. Como hemos visto, nuestro modelo es capaz de producir la probabilidad de B_i dado A_i, $p(B_i \mid A_i)$. Entonces, aplicando el principio de máxima verosimilitud, el estimador de máxima verosimilitud de θ es:

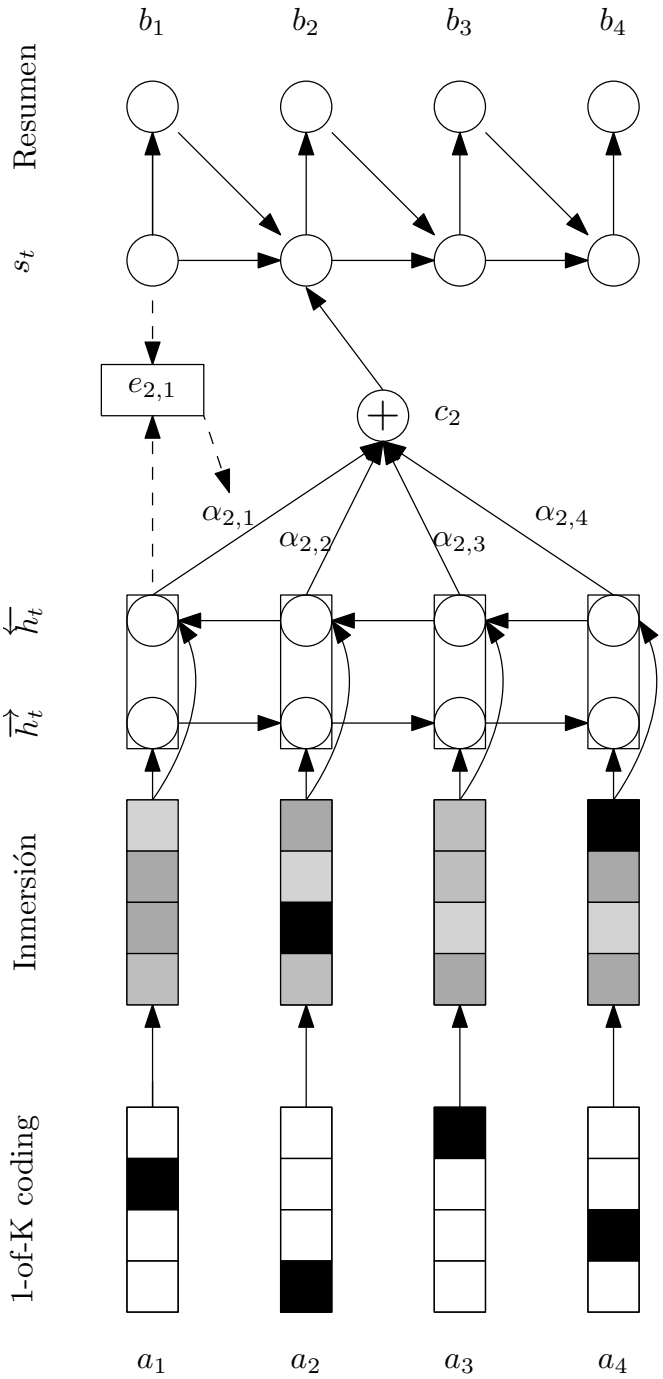

Figura 3.9: Esquema del modelo completo. s_t es el estado del decoder en cada iteración. Se muestra el mecanismo de atención para la segunda iteración, en el resto de pasos es igual.

$$\hat{\theta} = \arg\max_{\theta} \prod_{i=1}^{n} p(B_i \mid A_i), \tag{3.25}$$

es decir, el estimador de máxima verosimilitud es aquel que maximiza el producto de la probabilidades de los datos de entrenamiento.

La ecuación (3.25) anterior tiene el problema de que probabilidades cercanas a cero pueden provocar errores de redondeo. Para solucionar esto, nos damos cuenta que maximizar el producto es equivalente a maximizar la suma de los logaritmos. Esto es:

$$\hat{\theta} = \arg\max_{\theta} \sum_{i=1}^{n} \log p(B_i \mid A_i). \tag{3.26}$$

Esta es la forma más utilizada ya que presenta mayor facilidad de entrenamiento.

Contamos ahora con una función objetivo, a saber, $\sum_{i=1}^{n} \log p(B_i \mid A_i)$. Nuestro objetivo es ahora maximizarla. Como gran parte de la literatura está escrita en el contexto del problema de minimización, vamos a considerar equivalentemente que queremos minimizar $-\sum_{i=1}^{n} \log p(B_i \mid A_i)$.

3.10.1. Gradient descent

Consideremos una función diferenciable $f(x)$ a minimizar. En nuestro caso será el estimador de máxima verosimilitud definido en (3.26), dependiente de los parámetros θ.

Existen numerosas maneras para minimizar una función diferenciable. Por ejemplo, si la función es sencilla podemos estudiar los puntos tales que $\nabla_x f(x) = 0$, pues sabemos que es una condición necesaria de mínimo local.

Sin embargo, en el caso del deep learning la función con la que trabajamos es compleja y no es facíl resolver $\nabla_x f(x) = 0$. La norma general en el campo es utilizar *gradient descent*. Este método consiste en moverse por el espacio de parámetros en la dirección que más decrementa la función objetivo. Está dirección será aquella, indicada por el vector unitario u, que minimiza:

$$\frac{\partial}{\partial \alpha} f(x + \alpha u), \tag{3.27}$$

evaluada en $\alpha = 0$. $\frac{\partial}{\partial \alpha} f(x + \alpha u)$ es la derivada direccional de $f(x)$ en la dirección u. Por la regla de la cadena, $\frac{\partial}{\partial \alpha} f(x + \alpha u)\big|_{\alpha=0} = u^\top \nabla_x f(x)$. Y si consideramos que η es el ángulo entre el vector u y el gradiente obtenemos que:

$$\min_u = u^\top \nabla_x f(x) = \min_u \|\nabla_x f(x)\| \cos\eta, \tag{3.28}$$

con lo que concluimos que $\eta = \pi$ y que por lo tanto u es la dirección opuesta al gradiente. Conocida la dirección en la que decrementamos la función objetivo, a saber, $-\nabla_x f(x)$, nos desplazamos siguiéndola:

$$x' = x - \varepsilon \nabla_x f(x), \tag{3.29}$$

donde ε es un escalar que se conoce como *learning rate*. ε determina el tamaño del salto. Un ε pequeño reduce la velocidad de entrenamiento pero lo hace más preciso, mientras que un ε grande puede converger más rápido pero es más inestable. El *learning rate* se puede fijar a una constante o se puede variar a lo largo del proceso. En muchas ocasiones el valor de ε decidirá el éxito o el fracaso del entrenamiento de un modelo.

Nótese como el método del *gradient descent* no garantiza la llegada a un mínimo global. En efecto, el proceso convergerá a entornos de mínimos locales, pues en tal caso tendremos $\nabla_x f(x) \approx 0$. Cuando la función objetivo no es convexa, como en nuestro caso, esto puede suponer un problema. Sin embargo, en general en el deep learning, estos mínimos locales son casi tan buenos como el mínimo global. ¿Por qué? Simplemente porque se corrobora en la práctica. La teoría matemática sobre la optimización no convexa no está tan avanzada como para poder dilucidar la situación. Sólo recientemente ha empezado a haber avances en el campo, como dan fe los workshops en optimización convexa de conferencias recientes[1, 2]. Por último, nos vamos a limitar a citar una presentación de Yann LeCunn [Lecun, 2007], que trata sobre este problema:

> Let's not be afraid of methods for which we have no theoretical guarantee, particularly if they have been shown to work well. [..] We should use our theoretical understanding to expand our creativity, not to restrict it.

3.10.2. Backpropagation

Dado el papel fundamental que juega el gradiente, nos interesa poder calcularlo de manera eficiente.

Consideramos un grafo computacional (un pequeño ejemplo, ilustrativo de parte del encoder, se encuentra en la figura 3.10) y numeramos sus vértices de

[1] `https://sites.google.com/site/nips2015nonconvexoptimization/papers`
[2] `https://sites.google.com/site/noncvxicml16/`

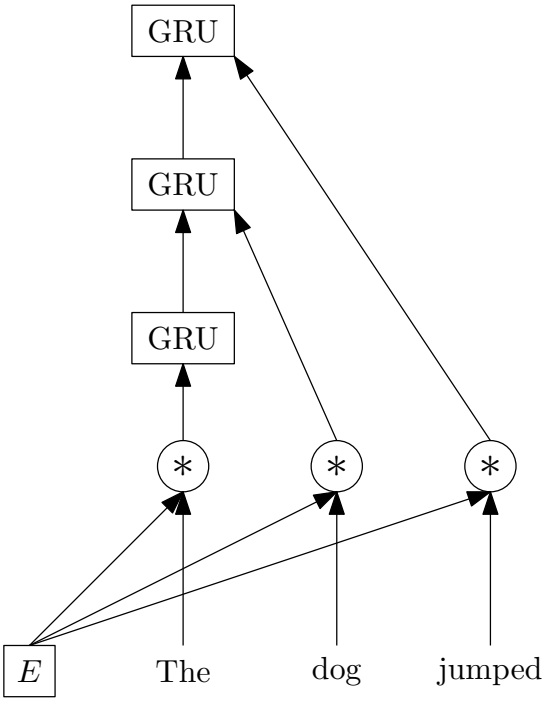

Figura 3.10: Ejemplo de grafo computacional. Se representa parte del encoder desarrollando la recurrencia durante tres iteraciones. Las palabras estarían en *1-of-K coding* y E sería la matriz de inmersión.

1 a N según un orden topológico, de forma que N sea el nodo salida. Sea $\pi(i)$ el conjunto de sucesores del nodo i. Haciendo un pequeño abuso de notación, denotamos por $\frac{\partial i}{\partial j}$ como la derivada de la salida del nodo i respecto de la entrada proveniente del nodo $j \in \pi(i)$. Nuestro objetivo es obtener la derivada de N respecto a todos los nodos de entrada. Una posibilidad es proceder de abajo a arriba, comenzando en un nodo de entrada y recorriendo todo el grafo hasta llegar a N. El método de backpropagation procede de arriba a abajo, recorriendo todo el grafo sólo una vez. Para ello, vemos que para todo nodo i:

$$\frac{\partial N}{\partial i} = \sum_{j \in \pi(i)} \frac{\partial N}{\partial j} \frac{\partial j}{\partial i}, \tag{3.30}$$

por la regla de la cadena. Luego, sólo queda recorrer los nodos $i = N-1, \ldots, 1$ (recordamos que están ordenados topologícamente) e ir calculando $\frac{\partial N}{\partial i}$ según (3.30).

3.10.3. Stochastic gradient descent

Volvamos por un momento a nuestra función objetivo, descrita en la ecuación (3.26). Tal función depende del rendimiento del modelo sobre todos los datos de entrenamiento, y, por lo tanto, el gradiente también depende de todos las datos de entrenamiento. Nuestros datos de entrenamiento van a estar compuestos por 300000 pares. En muchos otros modelos deep learning el tamaño puede ser mucho mayor, del orden de millones. Luego, resulta aparente que es prohibitivamente costoso calcular el gradiente en función de tantos ejemplos.

Para solucionar este problema se introduce el *stochastic gradient descent* (SGD), que es una extensión del *gradient descent*. La clave del SGD es aproximar el gradiente con un subconjunto de los datos de entrenamiento. Cada subconjunto se llama *minibatch*, generalmente de un tamaño m fijo. Los pares que conforman el minibatch se eligen al azar uniformemente. De esta manera, la esperanza del gradiente obtenido mediante SGD será el gradiente de los datos totales.

3.10.4. Gradient clipping

Las funciones que resultan de las RNNs a menudo tienen regiones de gradientes extremos [Pascanu et al., 2013]. Estas zonas se llaman barrancos, porque el grafo de la función objetivo en ellas es empinado. Esta situación también se conoce como *exploding gradients* [Bengio et al., 1993]. El problema radica en que en estas regiones, aunque el *learning rate* sea pequeño, la etapa de actualización de los parámetros del *gradient descent* puede alejarnos del mínimo local.

Una técnica, fundamentanda en los resultados obtenidos en la práctica, es el *gradient clipping* [Mikolov, 2012] [Pascanu et al., 2013], que consiste en establecer un máximo a la normal del gradiente. Esto es, cuando $\|\nabla_x f(x)\| > g$, g fijo, usamos como gradiente $\frac{g}{\|\nabla_x f(x)\|}\nabla_x f(x)$.

Capítulo 4

Entorno de trabajo

Entrenar un modelo deep learning como el descrito requiere una gran capacidad computacional: contamos con 300 000 parejas artículo-resumen y debemos iterar sobre ellas aproximadamente 30 veces, calculando probabilidades y ajustando los parámetros repetidas veces. Para acelerar el proceso se paraleliza la computación del grafo computacional; se puede aplicar las funciones asociadas a las neuronas de una capa en paralelo, con el mismo flujo de instrucciones pero distintos datos. Este modelo de computación, llamado *Single Instruction Multiple Data*, es el usado por las tarjetas gráficas (GPUs) y NVIDIA lo llama *Single Instrucion Multiple Thread*. Luego, es necesario contar con una GPU.

4.1. GPU

No toda GPU es adecuada. Requerimos de velocidades altas y una gran cantidad de memoria para guardar los parámetros y los datos. Para satisfacer estas necesidades se ha desarrollado un mercado de tarjetas gráficas orientadas al deep learning y tareas similares.

Para la realización de este trabajo y el posterior desarrollo de nuevos proyectos deep learning en la facultad se solicitó a NVIDIA la donación de una GPU a través del programa *GPU Grant Program* [1]. La solicitud fue aceptada y recibimos una NVIDIA Titan X con la arquitectura Pascal. En el momento de la realización del trabajo esta tarjeta es una de las más rápidas del mercado y ofrece 12 GB de memoria. Creemos que esta tarjeta será de gran utilidad en la realización de próximos proyectos en la facultad, ya que es la primera y antes no era posible realizar este tipo de trabajos.

[1] NVIDIA GPU Grant Program: `https://developer.nvidia.com/academic_gpu_seeding`

4.2. CUDA

Originalmente las tarjetas gráficas sólo se programaban mediante APIs de gráficos, como DirectX o OpenGL, que permiten definir las rutinas de tratamiento de los vértices y polígonos involucrados en el renderizado de las escenas. Estas rutinas son de naturaleza paralela y se adaptan al modelo SIMD, por ejemplo una proyección es aplicar la misma transformación lineal a una colección de vértices. Gracias a la predominancia del algebra lineal y las transformaciones lineales en las tareas gráficas, las GPUs están especialmente optimizadas para estas tareas. El deep learning y muchos otros algoritmos del machine learning tienen como componente fundamental el algebra lineal; otra razón más para el uso de GPUs.

Sin embargo, las funciones involucradas en los modelos deep learning difícilmente son describibles como rutinas de tratamiento de vértices y polígonos. Para la programación de estas funciones sobre GPUs NVIDIA ha desarrollado un toolkit, CUDA, que ofrece una serie de librerias y un compilador, que permiten describir en C las rutinas que ejecutarán un conjunto de threads y los datos que reciben.

CUDA es tecnología propietaria de NVIDIA. Existe OpenCL que es un estándar abierto y que funciona sobre más entornos que CUDA, en particular sobre las GPUs de AMD. A pesar de esto, CUDA es la tecnología predominante en el deep learning y la gran mayoría de herramientas están pensadas para el uso de CUDA.

4.3. Framework: Theano

El API de CUDA es de muy bajo nivel. Nuestro modelo se compone de un grafo computacional que es de muy alto nivel. Implementar este grafo computacional, de tan alto nivel, con herramientas de tan bajo nivel es laborioso. Además, necesitamos del cálculo del gradiente del grafo computacional del modelo para el entrenamiento. También nos gustaría describir el grafo computacional y poder ejecutarlo indistintamente sobre una CPU o GPU. A estos efectos, existen diversos frameworks orientados al deep learning, que permiten describir el grafo computacional en alto nivel, calcular su gradiente, cargar los datos de entrada y compilar el grafo para su ejecución sobre una GPU o una CPU.

Entre todos los frameworks disponibles podemos destacar tres: Theano [Theano Development Team, 2016], ideado en la Universidad de Montreal, Tensorflow [Abadi et al., 2015], de Google, y Torch [Collobert et al., 2011], de Facebook. Generalmente la gran mayoría de trabajos con redes neuronales

recurrentes usan Theano o Tensorflow; Torch está menos capacitado en estos casos por la falta de utilidades en sus librerías. Theano y Tensorflow son capaces de hacer cálculos simbólicos para calcular el gradiente; Torch no. Aquí nos centraremos en Theano, que es el que utilizamos, pero la mayoría de los principios se aplican a Tensorflow.

4.3.1. Funcionamiento

El tensor es el objeto base de Theano. Los tensores tienen larga tradición en las matemáticas y la física. Sin embargo, tanto aquí como en Theano y Tensorflow se limitan a abstraer el concepto de array multidimensional. Así, un tensor de dimensión 0 es un escalar, uno de dimensión 1 un vector y uno de dimensión 2 una matriz.

Para construir el grafo computacional en Theano trabajamos describiendo operaciones sobre tensores. Una vez descrito, se compila a un objeto `function`, que es la interfaz al cálculo en sí. Ilustramos esto con un ejemplo:

```
import theano
import theano.tensor as T
x = T.dmatrix('x')
y = T.dmatrix('y')
```

Aquí acabamos de declarar dos tensores matrices. Ahora podemos obtener otro tensor, producto de x e y:

```
z = x * y
```

Como se aprecia, los operadores arítmeticos para los tensores están sobrecargados. Nótese que ni x ni y ni z representan matrices concretas, tán sólo variable abstractas con las que se define el grafo. Para poder hacer cálculos, compilamos el grafo a una `function`:

```
f = function([x, y], z)
```

Ahora `f` es una función con la que podemos calcular con entradas concretas:

```
>>> f([[1, 0], [2, 2]], [[3, 3], [2, 1]])
    array([[ 3.,  0.],
           [ 4.,  2.]])
```

Theano es capaz de compilar `f` para ser ejecutada en una GPU o en una CPU, indistintamente. Además, permite calcular el gradiente del grafo computacional automáticamente, tan sólo hay que usar la función `tensor.grad`. Como la comunicación entre CPU y GPU es lenta, Theano permite declarar los tensores como `shared`, que indica que se almacenaran en la memoria de la GPU. Se declarán shared los parámetros del modelo, y sólo se volverán a acceder cuando necesitemos guardarlos en disco. Se ha proporcionado una pequeña introducción a los conceptos fundamentales de Theano. Para más detalles se recomienda consultar los tutoriales[2] de *deeplearning.net* o simplemente consultar el código del trabajo.

[2]Disponibles en: `http://deeplearning.net/tutorial/`

Capítulo 5

Experimentos

Tal como hemos dicho al principio, [Nallapati et al., 2016b] tomaba un sistema orientado a la traducción a la tarea del resumen. Más concretamente, adaptó el tutorial[1] que Kyunghyun Cho publicó en relación al DL4MT Winter School 2015. [Nallapati et al., 2016b] no publica ningún código fuente. Tuvimos que adaptar el modelo a la tarea del resumen y preparar el dataset para su uso. A continuación relatamos los detalles de nuestros experimentos.

5.1. Dataset

El dataset clásico en la tarea del resumen ha sido el Gigaword, que sólo consiste en un artículo y su titular y que cuenta con 4 millones de pares. La tarea de resumir un artículo a una frase es menos interesante y más simple que el resumen general, en el cual la longitud del resumen depende de la información del artículo.

[Nallapati et al., 2016b] son los primeros en proponer un dataset de resúmenes de varias frases. En su trabajo usan el mismo dataset que [Hermann et al., 2015] introduce para entrenar un sistema que responde a preguntas sobre el texto. Este dataset recopila 312085 artículos de los períodicos *CNN* y *DailyMail*, así como el titular y las entradillas (que vamos a llamar *highlights*). La concatenación de los highlights forma el resumen objetivo.

El dataset *CNN-DailyMail* tiene significamente menos pares que el Gigaword, sin embargo, esto no es impedimento para obtener resúmenes aceptables, reduciendo el tiempo de entrenamiento. [Nallapati et al., 2016b] destacan que la métrica ROUGE de este dataset se encuentra en el mismo rango que la del Gigaword.

[1] Disponible en `https://github.com/nyu-dl/dl4mt-tutorial`.

[Hermann et al., 2015], en vez del propio dataset, distribuyen una serie de scripts que descargan los artículos visitando las páginas de la CNN y el DailyMail. Sin embargo, con gran dificultad, pues algunas páginas ya no existen. Afortunadamente, Kyunghyun Cho ha subido el dataset ya recopilado[2]. Antes de usarlo debemos preprocesarlo.

5.2. Preprocesamiento

Para usar el dataset *CNN-Dailymail* primero hace falta convertirlo al formato adecuado. Originalmente se encuentra formado por archivos de texto con el cuerpo de la noticia al principio y el titular y las entradillas añadidos al final, separados por la palabra clave `@highligth`. Nuestro objetivo es obtener un cuerpo de textos paralelos: un archivo con el cuerpo de un artículo en cada línea y otro con los resúmenes, donde cada resumen se encuentra en el mismo número de línea que su artículo correspondiente.

Primero comenzamos separando el cuerpo del artículo de los highlights. Algunas frases, generalmente las que forman parte del pie de una imagen, que el dataset también incluye (pero no la imagen), carecen de punto final. Esto lo corregimos, como también hace [See et al., 2017]. Posteriormente tokenizamos las frases con la librería CoreNLP [Manning et al., 2014]. Una vez hecho esto sólo queda formar los archivos de textos paralelos. Todo esto se ha programado en Python.

5.3. Embeddings

[Nallapati et al., 2016b] usan word2vec [Mikolov, 2012] para preentrenar los embeddings, aunque dejan que estos sean actualizados en el entrenamiento. Word2vec estima los embeddings sin supervisión: palabras que comparten contexto en los textos se sitúan cercanas en el espacio vectorial denso. Nosotros, como [See et al., 2017], no usamos preentrenamiento. Aparte de [Nallapati et al., 2016b], hay estudios [Ramachandran et al., 2016] con resultados que muestran un incremento de hasta 4 puntos en las métricas ROUGE. Sin embargo, obtiene [See et al., 2017] resultados en el estado del arte sin preentrenamiento. Además, es más facíl de entrenar. [See et al., 2017] no usan preentremaniento. Nosotros tampoco usamos preentrenamiento.

La dimensión de los vectores word embedding (los w_i de la ecuación (3.5)) se ha fijado a 128, tal como hace [See et al., 2017].

[2]Disponible en: `http://cs.nyu.edu/%7Ekcho/DMQA/`

5.4. Hiperparámetros

Como método para ajustar el learning rate usamos Adadelta [Zeiler, 2012]. La dimesión de del vocabulario la hemos fijado a 50 000 palabras, tanto para el encoder como para el decoder. En general, otros estudios utilizan más. Por ejemplo, [Nallapati et al., 2016b] utiliza 150 000 en el encoder y 60 000 en el decoder. Sin embargo, más no necesariamente implica mejor, pues [See et al., 2017] ha obtenido mejores resultados ROUGE con 50 000 que con 150 000. Además, tras una evaluación cualitativa de nuestros resultados, la falta de vocabulario no parece un problema significativo. Los resúmenes producidos con el test set consisten en un 2.5 % en tokens *UNK*, que el sistema produce para las palabras fuera de vocabulario.

La dimensión de los vectores internos, es decir, los estados h_t de la recurrencia y los relativos a la atención, que se conocen grupalmente como *hidden states*, se ha fijado a 256. Por otra parte, tanto [Nallapati et al., 2016b] como [See et al., 2017] usan gradient clipping (ver sección 3.10.4) y nosotros también, a un valor de 2,0.

5.5. Entrenamiento

Para el entranamiento se ha usado una NVIDIA Titan X con 12 Gb de memoria. El tiempo total de entrenamiento ha sido aproximadamente 4 días. Hemos entrenado el modelo 33 epochs. Una *epoch* consiste en iterar sobre todos los pares de entrenamiento. Estos tiempos se corresponden con [Nallapati et al., 2016b] y [See et al., 2017].

5.6. Análisis cuantitativo

Comenzamos la evaluación de nuestro sistema con el análisis cuantitativo. Este análisis consiste en las métricas ROUGE, que se detallan en el cuadro , donde se encuentran nuestro resultados juntos a los de [Nallapati et al., 2016b] y [See et al., 2017]. Usualmente se usan las métricas ROUGE-1, que consiste en la métrica F_1 asociada a la división en 1-gramas, ROUGE-2, que es la métrica F_1 asociada a la división en 2-gramas, y ROUGE-L, que es la métrica LCS. El paquete ROUGE muestra las métricas en el intervalo $0, 1$, aquí se exponen en tanto por ciento como es la costumbre en la bibliografía.

Se puede observar que los sistemas más avanzados de [Nallapati et al., 2016b, See et al., 2017] tienen, especialmente, mayores resultados ROUGE-1. Esto sistemas implementan un mecanismo, llamado *pointer-generator networks* [Gülçehre et al., 2016], que permite al decoder incorporar vocabulario que

se encuentra en el texto fuente, lo cual reduce la limitación, que nosotros tenemos, de un vocabulario de tamaño fijo. Sin embargo, el problema de la falta de vocabulario no es prominente en el análisis cualitativo: sobre el total de tokens generados en el test set, sólo el 2.5 % se corresponde al token *UNK* (que representa que el sistema quería emitir no está en el vocabulario). Como la diferencia en la métrica ROUGE-1 es mayor que 2.5 puntos, es posible que la falta de vocabulario este interfiriendo en el aprendizaje.

Nuestro modelo es equiparable (con las mismas características) al modelo baseline de [See et al., 2017]. En relación a este, obtenemos prácticamente los mismos resultados ROUGE. Esto proporciona validación exterior a nuestro trabajo.

Cuadro 5.1: Métricas ROUGE. Entre paréntesis se indica el tamaño del vocabulario cuando procede.

	ROUGE-1	ROUGE-2	ROUGE-L
[Nallapati et al., 2016b]	35.46	13.30	32.65
baseline (50k) [See et al., 2017]	30.49	11.17	28.08
baseline (150k) [See et al., 2017]	31.33	11.81	28.83
nuestro trabajo	30.93	11.87	28.42
[See et al., 2017] state-of-the-art	39.53	17.28	36.38

5.7. Análisis cualitativo

El sistema produce, en efecto, resúmenes abstractivos y desmuestra un cierto dominio del lenguaje, siendo capaz de hilar frases y escribir pasajes que no aparecen en el texto fuente.

Por ejemplo, el resumen de la figura A.1 es bueno. Pocas de las palabras del resumen se utilizan en el mismo contexto que en el artículo: en él se menciona que "now scientists involved in the women 's health initiative study have warned" pero el modelo usa el estudio como sujeto de la frase, conjugando los verbos adecuadamente, "women 's health initiative study has warned". También usa "hrt" en lugar de "hormone replacement therapy" para hacerlo más corto. Aunque en general tanto los resúmenes de referencia como los generados están compuestos por más de una frase, en este caso el sistema sólo ha generado una. Sin embargo, esta frase resume sin duda lo más relevante del texto. Existe en este caso un error factual en el resumen, pues ha omitido el estudio "the million women" que el artículo menciona. Quizás esto se debe a que el nombre del estudio no incluye palabras que lo puedan identificar

como tal: “the million women study” puede referirse a que el estudio se llama a million women o que un millón de mujers participaron en él.

Otro ejemplo de la capacidad abstractiva es el resumen de la figura A.2. En este ejemplo el sistema ha sido capaz de resumir satisfactoriamente. Se puede observar como copia algunas partes de frases del texto, modificandolas y acortandolas para extraer lo más relevante: por ejemplo, el texto dice “geoff haigh, 61, from oldham, greater manchester” pero el resumen omite “greater manchester” y también “wanted to marry his long-term partner” pero el resumen omite “long-term”. Hace uso de los pronombres, pues el texto dice “the pair settled” y el resumen emite “they settled”. Este ejemplo también es muestra de la ocasional falta de vocabulario; ha producido `UNK` porque la palabra “bungles” no se encuentra en el vocabulario. Entre los casi 300 000 artículos que conforman el training set, la palabra “bungles” sólo aparece 13 veces.

El resumen A.3 evidencia la dificultad del resumen de los artículos periodísticos. Este artículo tiene una estructura muy particular: es una crónica de un partido de fútbol pero el periodista comienza escribiendo sobre el entrenador y empieza a describir el resultado del partido, lo que debería extraer el resumen, tarde en el texto. Aún así, el sistema es capaz de identificar el tipo de texto y de emitir algunos datos sobre el partido. El resumen dice “real sociedad beat atletico in la liga”, que es un fallo factual, pues en realidad fue el Atlético quién ganó. Esto es posible que se deba al estilo tan peculiar del artículo. Es interesante comprobar como, tras identificar que es un artículo de fútbol, el sistema intenta proporcionar detalles sobre el partido: cuándo se produjo, los goles y el resultado.

Los artículos de sucesos, con poco contenido de opinión, en general el sistema los traduce satisfactoriamente. Probablemente esto se deba a que los artículos de sucesos cuentan con una estructura fija, más facíl de inferir de los datos. Los artículos de opinión son más complejos y por esto quizás requieran de más ejemplos de entrenamiento —300 000 ejemplos es una cantidad relativamente pequeña comparada con la norma en el deep learning—. El resumen A.5 es de un artículo de sucesos y resulta satisfactorio. La primera frase, “dr UNK UNK , 52 , is thought to have left the country after a teacher died .” es combinación de las dos primeras frases del texto. El patrón “nombre, edad” es muy repetido a lo largo de todos los resúmenes generados y con casi total seguridad se debe a la también abundante presencia de este patrón en los resúmenes de referencia. El único problema de este resumen en particular es la falta de vocabulario, pero por lo menos se limita al nombre del doctor. El resumen A.6 es también de sucesos y vuelve a ser satisfactorio.

Acaso el mayor problema que hemos identificado es la repetición. Un ejemplo se encuentra en el resumen A.4. En él, el sistema repite una frase con

el mismo sentido tres veces. Esto muy probablemente se debe a que el modelo en ningún momento penaliza la atención repetida sobre las mismas partes del texto. Por otra parte, es interesante destacar que el sistema se refiere al mismo sujeto de diversar maneras: “elizabeth ‘ elle ’ edmunds”, “she” y “ms edmunds”, esto es muestra de su capacidad abstractiva. También parafrasea para simplificar las frases: de “used a falsified doctor ’s certificate stating she had terminal cancer to raise about $ 2500 from her page” pasa a “ms edmunds claimed she had cancer to raise about $ 2500 from her page”.

Capítulo 6

Conclusiones y trabajo futuro

El trabajo comienza como una investigación de la literatura existente. He estudiado todos los ámbitos del deep learning: cómo modelar problemas, cómo entrenar modelos y qué metodología seguir. También he analizado las incipientes técnicas deep learning enfocadas al procesamiento de lenguaje natural: las redes neuronales recurrentes y los modelos encoder-decoder, entre otras.

Después de la revisión de la bibliografía, he sido capaz de replicar algunos resultados de los artículos que describen el estado del arte en la tarea del resumen. Esto ha incluido el aprendizaje de las herramientas establecidas en el campo y su modificación y programación para conseguir nuestros objetivos. También hemos obtenido hardware especializado sin el que hubiera sido imposible realizar el trabajo. Este hardware es una aportación a toda la facultad. En definitiva, he adquirido una visión del proceso que hay detrás de los últimos artículos.

He comprobado como las técnicas estudiadas son capaces de tener cierto éxito en tareas con alto grado de inteligencia. Aún así, como se presenta en esta memoria, los resúmenes producidos no son excelentes. Existen problemas como la repetición de frases, las palabras fuera de vocabulario y, en pocas ocasiones, la invención de información. Sin embargo, siguen siendo sorprendentes por su capacidad abstractiva y de uso del lenguaje.

Las direcciones de trabajo futuro son múltiples. Por un lado, se puede trabajar en la resolución de los problemas anteriormente mencionados. Una posibildad es, dado que las tareas de la traducción y del resumen están relacionadas, como se ha visto, la aplicación al resumen de técnicas orientadas a la traducción. Esta es de hecho una línea de investigación prometedora que ya ha cosechado éxitos [See et al., 2017]. Por otro lado, considero interesante el estudio del resumen en español. El modelo aquí descrito valdría a tales efectos, pues no incorpora ninguna información del lenguaje —aprende el

lenguaje de los datos—. Sería necesario la recopilación de un dataset con pares artículo-resumen, parecido al CNN-DailyMail aquí descrito.

Bibliografía

[Abadi et al., 2015] 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., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[Bahdanau et al., 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

[Bengio et al., 1993] Bengio, Y., Frasconi, P., and Simard, P. (1993). The problem of learning long-term dependencies in recurrent networks. In *IEEE International Conference on Neural Networks*, pages 1183–1188 vol.3.

[Cho, 2015] Cho, K. (2015). Natural language understanding with distributed representation. , New York University. Lecture Note for DS-GA 3001.

[Cho et al., 2014a] Cho, K., Merrienboer, B. v., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014a). Learning phrase representations using rnn encoder-decoder for statistical machine translation.

[Cho et al., 2014b] Cho, K., van Merrienboer, B., Bahdanau, D., and Bengio, Y. (2014b). *On the properties of neural machine translation: Encoder-decoder approaches.*

[Chopra et al., 2016] Chopra, S., Auli, M., and Rush, A. M. (2016). Abstractive sentence summarization with attentive recurrent neural networks. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 93–98.

[Chung et al., 2014] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). *Empirical evaluation of gated recurrent neural networks on sequence modeling.*

[Collobert et al., 2011] Collobert, R., Kavukcuoglu, K., and Farabet, C. (2011). Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[Gülçehre et al., 2016] Gülçehre, Ç., Ahn, S., Nallapati, R., Zhou, B., and Bengio, Y. (2016). Pointing the unknown words. *CoRR*, abs/1603.08148.

[Hermann et al., 2015] Hermann, K. M., Kociský, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching machines to read and comprehend. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *NIPS*, pages 1693–1701.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

[Hovy, 2003] Hovy, E. (2003). Text summarization. In Mitkov, R., editor, *The Oxford Handbook of Computational Linguistics*, Oxford Handbooks in Linguistics, chapter 32, pages 583–598. Oxford University Press, Oxford.

[Kikuchi et al., 2016] Kikuchi, Y., Neubig, G., Sasano, R., Takamura, H., and Okumura, M. (2016). Controlling output length in neural encoder-decoders.

[Koehn et al., 2003] Koehn, P., Och, F. J., and Marcu, D. (2003). Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 48–54, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Lecun, 2007] Lecun, Y. (2007). Who is afraid of nonconvex loss functions? NIPS Workshop on Efficient Machine Learning, Whistler 2007.

[Lin, 2004] Lin, C.-Y. (2004). Rouge: A package for automatic evaluation of summaries. In *Proc. ACL workshop on Text Summarization Branches Out*, page 10.

[Mani et al., 1999] Mani, I., House, D., Klein, G., Hirschman, L., Firmin, T., and Sundheim, B. (1999). The tipster summac text summarization evaluation.

[Manning et al., 2008] Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

[Manning et al., 2014] Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

[Mikolov, 2012] Mikolov, T. (2012). *Statistical language models based on neural networks*. PhD thesis, Brno University of Technology.

[Nadkarni et al., 2011] Nadkarni, P. M., Ohno-Machado, L., and Chapman, W. W. (2011). Natural language processing: an introduction. *Journal of the American Medical Informatics Association*, 18(5):544–551.

[Nallapati et al., 2016a] Nallapati, R., Zhai, F., and Zhou, B. (2016a). Summarunner: A recurrent neural network based sequence model for extractive summarization of documents.

[Nallapati et al., 2016b] Nallapati, R., Zhou, B., santos, C. N. d., Gulcehre, C., and Xiang, B. (2016b). Abstractive text summarization using sequence-to-sequence rnns and beyond.

[Pascanu et al., 2013] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1310–III–1318. JMLR.org.

[Ramachandran et al., 2016] Ramachandran, P., Liu, P. J., and Le, Q. V. (2016). Unsupervised pretraining for sequence to sequence learning.

[Rush et al., 2015] Rush, A. M., Chopra, S., and Weston, J. (2015). A neural attention model for abstractive sentence summarization. In Màrquez, L., Callison-Burch, C., Su, J., Pighin, D., and Marton, Y., editors, *EMNLP*, pages 379–389. The Association for Computational Linguistics.

[Schuster and Paliwal, 1997] Schuster, M. and Paliwal, K. (1997). Bidirectional recurrent neural networks. *Trans. Sig. Proc.*, 45(11):2673–2681.

[See et al., 2017] See, A., Liu, P. J., and Manning, C. D. (2017). Get to the point: Summarization with pointer-generator networks. *CoRR*, abs/1704.04368.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, NIPS'14, pages 3104–3112, Cambridge, MA, USA. MIT Press.

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*.

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

[Wu et al., 2016] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

[Xiong et al., 2016] Xiong, W., Droppo, J., Huang, X., Seide, F., Seltzer, M., Stolcke, A., Yu, D., and Zweig, G. (2016). Achieving human parity in conversational speech recognition. *arXiv preprint arXiv:1610.05256*.

[Zeiler, 2012] Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701.

Apéndice A

Ejemplos de resúmenes

Artículo (truncado): a new study has shown women still have an increased risk of breast cancer years after stopping taking hormone replacement therapy . hormone replacement therapy increases a woman 's risk of breast cancer up to eight years after she has stopped treatment , scientists have warned . the new findings come from experts who conducted one of the world 's largest studies into the effects of hrt - used to ease the effects of menopause . the women 's health initiative in the us and the million women study in the uk have both revealed strong evidence that hrt increases a woman 's risk of breast , womb and ovarian cancers . the million women study , a collaboration between cancer research uk and the nhs , results suggested that women who are using combined hrt have double the risk of breast cancer compared to non-users . and researchers found if women use hrt for more than 10 years , their risks are even higher . now scientists involved in the women 's health initiative study have warneyd , after further follow-ups , that women who take the combined oestrogen and progesterone hrt therapy continue to have a high risk of breast cancer , up to eight years after they stop treatment . after going through the menopause a woman 's ovaries stop producing the hormone oestrogen . it triggers menopausal symptoms including hot flashes and mood swings . hrt treats the body by introducing female sex hormones . there are three types , oestrogen-only hrt , cyclical hrt - where oestrogen is taken continuously but progesterone is given either monthly or every three months - and continuous combined hrt , where both female sex hormones are taken at the same time . in the early 1990s , some smaller studies suggested the therapy would reduce a woman 's risk of coronary heart disease and osteoporosis . to examine whether that was the case , the women 's health initiative in the us was tasked with investigating . they enrolled more than 26,000 women aged 50 to 79 in the study . meanwhile , in the uk another key study , the million women study , explored the effects of hrt on women aged 50 or over . both concluded that women prescribed hrt were at a higher risk of breast cancer , than those not receiving the treatment .

Referencia: hrt increase risk of disease up to eight years after stopping treatment . applies to those taking combined hrt - oestrogen and progesterone . discovery made by scientists from one of largest ever studies into hrt . but women taking oestrogen-only hrt may actually have a reduced risk .

Modelo: women 's health initiative study has warned that hrt increases woman 's risk of breast , womb and ovarian cancers .

Figura A.1

Artículo (truncado): a terminally-ill landlord whose dying wish was to marry his partner of 26 years saw their special day descend into farce after a series of blunders . geoff haigh , 61 , from oldham , greater manchester , wanted to marry his long-term partner heather after being diagnosed with terminal pancreatic cancer and the pair settled on a venue close to their home . but , instead of having the magical day they had planned , the couple were left reeling after a number of bungles thwarted the ceremony from running smoothly . guests were forced to listen to the wedding song 10 times while it played over and over , and the couple were left with three marriage certificates after officials got crucial details wrong on the first two . geoff haigh , 61 , from oldham , greater manchester , wanted to marry his long-term partner heather , 54 , after being diagnosed with terminal pancreatic cancer but the couple 's special day -lrb- pictured -rrb- was full of blunders . guests at the couple 's -lrb- pictured -rrb- wedding were forced to listen to the song ‘ wonderful life ’ 10 times in a row due to a blunder and their marriage certificate was recorded with the wrong date on , rendering it invalid . mr haigh , who died weeks after the ceremony in november last year , was left so devastated by the day that he told his wife that he felt like he 'd let her down . in fact , the day was so full of blunders that all 11 guests wrote a separate letter of complaint to manchester register office , which is based at heron house in the city centre , where the pair said their vows . mrs haigh , 54 , said : ‘ geoff was heartbroken by it all . he kept apologising to me and saying he 'd wished we 'd got married years earlier and done it properly . ‘ he was devastated , as the way he saw it , this was his final gesture for me and it had been ruined by their thoughtlessness . ’ the couple had initially dreamed of getting married abroad once mr haigh discovered his advanced pancreatic cancer was terminal . however , the landlord of the ram 's head pub , in oldham , was distraught to discover

---

Referencia: geoff haigh , 61 , wanted to marry partner heather after cancer diagnosis . couple married at manchester register office but day was full of blunders . guests forced to listen to wedding song 10 times in a row and venue dirty . marriage certificate had spelling errors and wrong date and names .

Modelo: geoff haigh , 61 , from oldham , wanted to marry partner heather . they settled on venue close to their home but instead of having the magical day they had planned , the couple were left reeling after a number of UNK thwarted the ceremony .

Figura A.2

Artículo (truncado): for all the problems that david moyes had during his year at manchester united , you would n't associate his teams with defensive frailties and a lack of organisation at set pieces . and since taking over at real sociedad moyes has enhanced his reputation as a manager who brings , if nothing else , solidity . in his 21 la liga games to date sociedad have conceded just 29 times , climbing from the relegation when moyes took over , to an extremely comfortable tenth place . mikel gonzalez diverts a corner past his own goalkeeper geronimo rulli who fails to punch it away . atletico madrid celebrate after going two goals ahead after just 10 minutes , killing the match as a contest . antoine griezmann -lrb- left -rrb- is congratulated by fernando torres , but did n't celebrate against his former club . atletico madrid : oblak ; gimenez , miranda , siqueira , gamez ; suarez , tiago , koke , turan -lrb- niguez 81 mins -rrb- ; griezmann -lrb- gabi 90 -rrb- , torres -lrb- jimenez 61 -rrb- subs not used : ansaldi , bernabe , cani , hernandez . goals : gonzalez -lrb- og -rrb- 2 , griezmann 10 . booked : turan . real sociedad : rulli ; gonzalez , elustondo , yuri , zadldua ; bergara , ruben pardo ; zurutuza -lrb- finnbogason 74 -rrb- , canales -lrb- agirretxe 74 -rrb- , xabi prieto ; castro -lrb- granero 81 -rrb- subs not used : martinez , hervias , estrada , zubikarai . yet , on tuesday night , sociedad showed none of the qualities usually associated with their manager , capitulating inside 10 minutes against the spanish champions . by the time antoine griezmann struck the hosts ' second goal , lifting the ball over the hapless geronimo rulli after the keeper had parried koke 's shot straight to the frenchman , atletico could have been four up , and the game was all but over . arda turan had already seen his volley hit the post after being left totally unmarked from a free kick , before mikel gonzalez diverted the resulting corner past his own keeper after little more than 90 seconds . soon after griezmann narrowly failed to connect with a header after yet more shocking set-piece defending , and the former sociedad man also came close to tricking

---

Referencia: mikel gonzalez own goal puts atletico madrid ahead inside two minutes . antoine griezmann adds second after goalkeeping error . champions could have scored more as real socieda offer very little .

Modelo: real sociedad beat real sociedad in la liga on tuesday night . david moyes had a goal disallowed for a corner for the first time . real sociedad beat atletico in la liga .

Figura A.3

Artículo (truncado): police have charged a woman with fraud after she allegedly faked a terminal illness to raise thousands of dollars from sympathetic strangers . in august 2014 elizabeth ‘ elle ’ edmunds established an online fundraising page claiming she had ovarian cancer and seeking money for medical treatment . police alleged the 31-year-old mother from belmont , near newcastle in nsw , used a falsified doctor ’s certificate stating she had terminal cancer to raise about $ 2500 from her page . in february , ms edmunds told a belmont local court she was put up to the act by her violent partner john heagney , and did it because she feared for her life . elizabeth ‘ elle ’ edmunds , 31 , was issued a court notice this week for the offence of two counts of obtaining benefits by deception , after she faked having terminal cancer to raise money in 2014 . in october , a report was made to police regarding ms edmund ’s activities , and detectives from the lake macquarie local area command soon after launched an investigation . she was issued a court notice this week for the offence of two counts of obtaining benefits by deception , and will appear at belmont local court in june . police inquiries into the matter are continuing and investigators are appealing for anyone with information to come forward . ms edmunds went into hiding after revelations of her alleged fraud came to light last year . ‘ i was in fear for my life , ’ she told belmont local court , near lake macquarie in the hunter valley north of sydney , at a hearing in february . she said mr heagney had tried to kill her and had severely beaten her on two occasions last year . heagney claims ms edmunds faked the violence and was a self-harmer who punched herself , and that the two dates on which the alleged violence took place , october 20 and november 17 last year , related to the faking of the cancer . in april last year , ms edmunds sat her family down and told them she had cancer and doctors had given her just three years to live .

Referencia: elizabeth ‘ elle ’ edmunds was charged with two counts of fraud . the 31-year-old mother will appear at belmont local court in june . the lake macquarie woman faked having stage six ovarian cancer . she said she believed her mind tricked her into thinking she had illness . she also said that her former partner forced her to fake the disease . police inquiries are continuing and investigators have appealed for anyone with information to come forward .

Modelo: elizabeth ‘ elle ’ edmunds claimed she had ovarian cancer and sought money for medical treatment . she claimed she had ovarian cancer to raise money for medical treatment . ms edmunds claimed she had cancer to raise about $ 2500 from her page .

Figura A.4

Artículo (truncado): dr nadeem azeez , 52 , -lrb- pictured -rrb- is thought to be in pakistan but was yesterday charged with manslaughter by gross negligence . an international arrest warrant has been issued for an nhs doctor who is believed to have left the country after a teacher died following a caesarean . dr nadeem azeez , 52 , is thought to be in pakistan but was yesterday charged with manslaughter by gross negligence , along with fellow anaesthetist dr errol cornish , 67 . maidstone and tunbridge wells nhs trust is also accused of corporate manslaughter – the first time a health trust has been charged with the offence since its introduction in 2008 . frances cappuccini died within hours of her second son 's birth after suffering major complications that resulted in the loss of half her blood . at an earlier inquest hearing , the family 's lawyer said her death might have been avoided if doctors had not persuaded her to change her birth plan . the 30-year-old – a primary school teacher – had intended to have an elective caesarean at tunbridge wells hospital . but she and her husband tom , 35 , reluctantly abandoned their plans after doctors said they should allow labour to progress naturally overnight . their son giacomo was eventually born by emergency caesarean – but mrs cappuccini suffered massive bleeding and died soon afterwards on october 9 , 2012 . her husband was left to bring up the little boy – who has the middle name frankie in memory of his mother – and the couple 's other son , luca , five , by himself . the family 's lawyer neil sheldon told mrs cappuccini 's inquest that she was left struggling to breathe for 90 minutes without being given sufficient oxygen . he also questioned why the teacher did not receive a blood transfusion earlier and suggested there could be an issue with maternity ward staff overruling women 's wishes for their birth plans . the inquest was halted in february last year so evidence could be referred to prosecutors to allow them to consider bringing criminal charges . mr sheldon said at the time : ‘ the family has not entered into the process for any other reason than to have questions answered . they are not trying to have anyone convicted , but they do want

---

Referencia: frances cappuccini died after giving birth to son giacomo in october 2012 . two doctors and kent hospital accused of gross negligence manslaughter . international arrest warrant issued for dr nadeem azeez , 52 , from pakistan . teacher and husband wanted caesarian but allegedly persuaded not to . mrs cappuccini required emergency c-section and died hours later .

Modelo: dr UNK UNK , 52 , is thought to have left the country after a teacher died . she died within hours of her second son 's birth after suffering major complications that resulted in the loss of half her blood . dr UNK UNK , 52 , is also accused of manslaughter by gross negligence .

Figura A.5

Artículo (truncado): new york city police have released the image of a tattoo found on the body of a woman in hopes of identifying her nearly a month after she died on a brooklyn street . the unnamed woman , believed to be between 25 and 45 years old , was discovered lying unconscious march 22 in the area near shore parkway and 26th avenue in the gravesend section of brooklyn . paramedics who responded to the scene were unable to revive her and she was pronounced dead that evening . distinctive : nypd has released images of a heart tattoo with the name ‘ monique ’ found on an unnamed woman who died in brooklyn last month . to the left is the body art as it appears on her leg . to the right is a tattoo artist ’s rendering of how the actual tattoo may have appeared originally . the deceased woman had no identification on her , but she sported a distinctive tattoo on her right leg . investigators believe the faded body art was the name ‘ monique ’ etched across a tattooed ribbon within a heart topped with a rose . nbc new york reported that police are still awaiting a medical examiner ’s report to find out the cause of the woman ’s death . anyone with information about the unidentified woman is being asked to contact nypd crime stoppers at 800-577-tips . death in the street : the unnamed woman , believed to be 25 to 45 years old , was discovered lying unconscious march 22 in the area near shore parkway and 26th avenue in the gravesend section of brooklyn .

Referencia: woman believed to be 25 to 45 years old was found march 22 near shore parkway and 26th avenue in gravesend . had name ‘ monique ’ tattooed on right leg within a heart and rose .

Modelo: the unnamed woman , believed to be between 25 and 45 , was discovered in march 22 . paramedics who responded to the scene were unable to revive her and she was pronounced dead .

Figura A.6

Apéndice B

Uso y entrenamiento del modelo

Todo el trabajo se ha realizado en un sistema GNU/Linux, con la distribución Archlinux. Las instrucciones aquí detalladas son para GNU/Linux. Las librerías necesarias para el uso de la GPU son las siguientes:

- CUDA v8 - el paquete CUDA de NVIDIA.
- CUDNN v6 - librerías adicionales de NVIDIA para CUDA para programar redes neuronales.

Su instalación será dependiente de la distribución utilizada. En Archlinux se encuentran disponibles en los repositorios oficiales. Es recomendable que el ordenador en el cual se instale la tarjeta gráfica cuenta con una fuente de alimentación de al menos 600W de potencia.

La opción más recomendable para gestionar Python y las librerías relaciones es usar la herramienta Anaconda[1]. Anaconda nos permite definir entornos virtuales e instalar allí la distribución de Python, incluyendo Theano. Una vez instalado, creamos un nuevo entorno:

```
conda create --name py2_theano python=2
```

En el directorio actual se habrá creado una carpeta llamada `py2_theano`, raíz del entorno. Siempre que queramos usar el entorno debemos activarlo antes:

```
source activate py2_theano
```

Una vez hecho esto, instalamos las librerías:

[1]Disponible en: `https://www.continuum.io/downloads`.

```
conda install numpy scipy mkl
conda install theano
```

La configuración de Theano se encuentra en el archivo `~/.theanorc`. Nostros hemos usado la siguiente configuración:

```
[global]
device = gpu
floatX = float32

[cuda]
root = /opt/cuda
```

Para usar la CPU en vez de la GPU basta cambiar `device = gpu` a `device = cpu`. Estamos en disposición de empezar a entrenar el modelo. Primero tenemos que preparar el dataset[2]. Suponemos que este se encuentra descomprimido en un directorio llamado `stories`. Ejecutamos en consola los scripts del trabajo:

```
mkdir sanitized
mkdir tokenized
mkdir bitexts
python sanitize.py stories sanitized
python tokenize.py sanitized tokenized
python generate_bitexts.py all_train.txt tokenized bitexts/train
python generate_bitexts.py all_val.txt tokenized bitexts/val
python generate_bitexts.py all_val.txt tokenized bitexts/test
python build_dictionary.py bitexts/train.articles
    bitexts/train.highlights
python shuffle.py bitexts/train.articles
    bitexts/train.highlights
```

Ya podemos entrenar el modelo. Para ello primero hay que cambiar las rutas de los archivos de `train_articles.py` y ejecutamos:

```
python train_articles.py
```

En `train_articles.py` se puede configurar cada cuantas iteraciones queremos guardar los parámetros del modelo. El proceso de entrenamiento es largo, alrededor de cuatro días. Durante el entrenamiento se puede ver el estado de la GPU con el comando `nvidia-smi`, incluido en CUDA. Esta herramienta nos mostrara la cantidad de memoria utiliza, el consumo y la

temperatura, entre otras cosas. Durante nuestros entrenamientos la GPU se mantuvo a una temperatura de entre 80 y 83 grados. Son temperaturas muy altas, pero 83 grados es la temperatura objetivo que NVIDIA ha configurado para la tarjeta Titan X, es decir, se ajustará automáticamente la velocidad del ventilador para nunca sobrepasarla.

Para generar resúmenes se ejecuta el siguiente comando:

```
python summarize.py model1.npz bitexts/train.articles.pkl
    bitexts/train.highlights.pkl bitexts/test.articles
    test_gen
```

La generación de resúmenes se puede realizar sobre la CPU. Se produce un resumen en cuestión de segundos, usando la CPU.