UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Generación de resúmenes de textos

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Jorge Verdeguer Gómez
Tutor: Jon Ander Gómez Adrián

Curso 2018-2019

Resum

En el nostre treball, proposem un nou model per a la generació de resums de forma abstractiva. El nostre model consistix en juntar una tècnica emprada actualment en la generació de resums amb una nova representació del text mitjançant un vector tridimensional. També hem emprat una tècnica pionera per a crear una representació individual de cada paraula sense recurrir als mètodes emprats actualment de word2vec, a causa de diversos problemes que la representació per contexte posseeix. En este document, discutirem tant l'enfocament que hem seguit com els resultats del nostre treball.

Paraules clau: abstractiva, vector, word2vec, representació per contexte

Resumen

En nuestro trabajo, proponemos un nuevo modelo para la generación de resúmenes de forma abstractiva. Nuestro modelo consiste en juntar una técnica usada actualmente en la generación de resúmenes con una nueva representación del texto mediante un vector tridimensional. Tambien hemos usado una técnica pionera para crear una representación individual de cada palabra sin recurrir a los métodos usados actualmente de word2vec, debido a varios problemas que la representación por contexto posee. En este documento, discutiremos tanto el enfoque que hemos seguido como los resultados de nuestro trabajo.

Palabras clave: abstractiva, vector, word2vec, representación por contexto

Abstract

In our work, we propose a new model for abstractive text summarization. Our model consists in putting together a current technique in abstractive summarization with a new representation of the text through a tridimensional vector. We have also employed a pioneer technique for creating an individual word embedding for each word without recurring to the currently used method of word2vec, since representation by context has multiple inherent problems. In this document, we will discuss both the approach we have followed and the results of our work.

Key words: abstractive summarization, vector, word2vec, word embedding

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

La generación de resúmenes a partir de un texto tiene como objetivo que, a partir de un texto, crear una representación reducida del mismo que consiga transmitir el mensaje principal del texto y sus ideas clave. Actualmente, en este ámbito existen dos tipos de enfoque para crear resúmenes a partir de un texto.

El primero de ellos es llamado *extractive summarization*. La mayoría de los generadores de resúmenes que funcionan actualmente hacen uso de este enfoque. Esta metodología consiste en detectar y recopilar las oraciones que contienen información importante sobre el texto. Luego, se juntan las oraciones mas importantes para generar el resúmen del texto.

El segundo de los enfoques es llamado *abstractive summarization*. Esta metodología consiste en crear un resumen del texto desde cero, posiblemente conteniendo características, construcciones y vocabulario no presente en el texto a resumir. Este tipo de enfoque tiene la ventaja potencial de ser capaz de comprimir el texto en mayor medida, al no estar limitado a las oraciones que presenta el texto, y pudiendo englobar las más importantes con otro tipo de construcción linguística. En nuestro trabajo, perseguimos crear un sistema basado en *abstractive summarization*.

1.1 Motivación

La razón de centrarse en la generación de resúmenes de textos está basada en no solo la utilidad de los propios resúmenes, si no en enseñar a nuestros sistemas a captar y entender las ideas principales del texto, para así proceder a su posterior análisis y uso en sistemas más complejos que generen resúmenes más sofisticados. Este enfoque es necesario para ser capaces de comprimir aun más la información en el mundo actual, no solo a través de herramientas propias del analísis de lenguaje natural, si no también a partir de las herramientas que nos ofrece el propio lenguaje.

En nuestro caso, hemos elegido basarnos en *abstractive summarization* por dos motivos principales. El primero de ellos es que ofrece mayor potencial que *extractive summarization*, ya que permite usar construcciones no presentes en el texto, posiblemente capaces de otorgarnos una version todavia mas reducida del resumen. El segundo motivo consiste en que mientras que la *extractive summarization* ya ha sido explorada en gran medida y ha conseguido unos buenos resultados, la *abstractive summarization* aún es un campo mucho más inexplorado, y creemos que sería de mayor ayuda para el campo explorar varias de las oportunidades que ofrece este segundo enfoque.

1.2 Objetivos

Nuestro objetivo principal consiste en crear un sistema que sea capaz de recibir un texto y generar su resumen correspondiente, conteniendo las ideas clave y los detalles importantes del mismo. Para evaluar nuestro sistema usaremos la métrica *ROUGE*. Además de nuestro objetivo principal también contemplamos varios objetivos secundarios. En primer lugar esperamos que este experimento nos ayude a comprender como funcionan las redes neuronales recurrentes, como las *LSTM*, y aprender a utilizarlas. En segundo lugar, probar si las redes convolucionales pueden ser de ayuda en alguno de nuestros modelos. Por último, aprenderemos a usar las métricas estándar para evaluar los resultados obtenidos en nuestro experimento.

1.3 Estructura de la memoria

La estructura de la memoria consistirá en cinco partes principales. Primero, se realizará un resumen sobre el estado del arte referente a nuestro tópico, asi como comentar varios de los acercamientos que se usan actualmente y sus resultados. En segundo lugar, comentaremos las herramientas que hemos usado para realizar este proyecto. En este apartado incluiremos tanto los *backends* usados para el *deep learning* como las diferentes librerías usadas específicamente para la creación de los resúmenes. En tercer lugar, explicaremos extensamente nuestro enfoque en el proyecto. Esta parte incluye la descripción del modelo creado, nuestro preprocesado de los textos para su posterior entrenamiento y análisis, y varias de las estructuras o metodologías que se han usado. La cuarta parte de nuestro proyecto consiste en una descripción detallada de las pautas que hemos seguido a la hora de experimentar con los datos y de los resultados obtenidos según las métricas estándar actuales. Por último, haremos una breve revisión de nuestros objetivos y analizaremos nuestros logros con el proyecto. En este apartado también analizaremos el enfoque seguido y si son beneficiosos los distintos cambios que hemos usado respecto a los sistemas actuales.

CAPÍTULO 2
Estado del arte

En este apartado procederemos a comentar diferentes modelos usados actualmente. En esta selección de modelos hemos incluido principalmente modelos basados en *abstractive summarization*, ya que están más relacionados con nuestro trabajo.

2.1 GIGAWORD

Gigaword[2] es un conjunto de datos muy usado actualmente a la hora de entrenar diversos modelos de generadores de resúmenes. Casi todos los generadores aquí comentados hacen uso de este conjunto de datos. Gigaword consiste en varios millones de diferentes textos y noticias con sus respectivos resúmenes, y su popularidad ha sido bastante alta desde que fue creado. No hemos hecho uso de este conjunto de datos al ser de pago, pero creíamos importante mencionar su existencia debido a su popularidad y su frecuente uso en el campo.

2.2 A Neural Attention Model for Abstractive Sentence Summarization

En [9], se discute la creación de un modelo el cual usa un sistema de atención para codificar las oraciones, en vez de el tradicional *encoder* de *bag-of-words* o del más reciente *convolutional encoder*. Este *encoder* gana eficacia debido a que el *convolutinal encoder* proporciona una capacidad mayor para analizar el input que el *bag-of-words*, sigue necesitando que se le proporcione una representación para el texto de entrada. Por ello, optan por un *attention-based contextual encoder* que construye una representación del mismo basado en la generación de contexto. Ademas, en la generación de los resúmenes, este modelo usa el algoritmo *Beam search*, el cual es uno de los algoritmos que hemos usado en la generación de resúmenes de nuestro propio modelo.

2.3 Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond

En [10] se propone un nuevo modelo de *abstractive summarization* usando *autoencoders* de redes neuronales recurrentes con mecanismos de atención. En el, proponen diferentes modelos para afrontar los problemas críticos que no son evitados por las herramientas básicas actuales. Ademas, proponen un nuevo conjunto de datos que incluye resúmenes

de múltiples frases para establecer un punto de referencia para futuras investigaciones en este ámbito.

Algunas de las herramientas que presentan sus modelos son *large-vocabulary-trick*, explicada en [4], captura de palabras clave y la inclusión de palabras poco comunes en el resumen, gracias a un sistema que permite extraerlas del propio texto.

2.4 Convolutional Sequence to Sequence Learning

En [12] se explora la posibilidad de usar redes convolucionales en vez de las redes recurrentes, que forman el *state-of-the-art*. Aunque este artículo contiene pruebas de *Machine Translation* y no de *abstractive summarization*, nos ha parecido relevante para el experimento por su uso de las redes convolucionales para analizar el lenguaje natural. En este trabajo, se explica que las redes convolucionales actuales están mejor preparadas para aprovechar las herramientas *hardware* actuales y también facilitan el aprendizaje. Las ventajas que presentaba este artículo nos incitaron a probar las redes convolucionales en nuestros modelos.

2.5 Abstractive text summarization with Attention-based Mechanism

En [17] se busca una forma de abordar el problema de *abstractive summarization* de una forma más humana. Su premisa consiste en que si aspiramos a hacer sistemas que generen resúmenes como los que hacemos los humanos, deberíamos incluir en ese sistema las herramientas y metodologías que usan los humanos a la hora de crear resúmenes. Por ello, exploran la evolución de los modelos secuenciales, desde las primeras redes neuronales recurrentes y sus resultados. Ademas, también ponen a prueba el *transformer*, el sistema que representa el *state-of-the-art* en el ámbito de la traducción automática. Para ello, usarán dos sets de datos diferentes entre sí y los compararán con los modelos de redes neuronales recurrentes usados hasta ahora.

CAPÍTULO 3
Descripción de las herramientas utilizadas

En este apartado describiremos las herramientas que hemos utilizado para llevar a cabo nuestra experimentación.

3.1 Tensorflow

TensorFlow[7] es una librería de computación numérica. Esta herramienta fue creada por Google y, desde 2015, se trata de una software *open source* disponible en su propio repositorio de GitHub. Aunque conocemos que existen varios backends más para Machine Learning, como MXNet, Pytorch, Chainer y Caffee, nos decantamos por Tensorflow por diversas razones. La primera de ellas es que su gran flexibilidad y escalabilidad nos ha permitido hacer muchas pruebas de concepto en poco tiempo sobre los modelos que hemos ido diseñando. También cabe destacar que Tensorflow va más allá del Machine Learning, y aunque no ha sido necesario en nuestro caso, se puede usar para facilitar la computación numérica o el aprendizaje por refuerzo.

En nuestro caso, hemos aprovechado la flexibilidad que nos ofrece TensorFlow para definir los modelos usando las partes de más alto nivel. De esta manera, nos ha sido posible diseñar, mantener y probar varios modelos de una manera muy conveniente. No obstante, para aprovechar aún más el potencial de TensorFlow, hemos usado otra librería, llamada **Keras**, que nos permite aumentar incluso más el nivel de abstracción. Aunque nuestro desarrollo es sobre la librería **Keras**, es importante comentar las funcionalidades y posibilidades de TensorFlow debido a que **Keras** puede usar TensorFlow como backend. En nuestro caso, hemos elegido TensorFlow como backend de Keras.

3.2 Keras

Keras[6] es una librería de alto nivel usada para el *Machine Learning*. Es una libreria que ha cobrado mucha fama estos últimos años debido a su facilidad de uso y a su suave curva de aprendizaje. Es por esta razón que actualmente es muy usada en los proyectos de *Machine Learning*, ya que te permite definir modelos de aprendizaje en unos pocos minutos. En nuestro caso, hemos decidido usar **Keras** como librería principal en nuestro desarrollo tanto por la abundancia de recursos que podemos encontrar sobre ella como por su facilidad de aprendizaje y uso. Con **Keras** definiremos todos nuestros modelos usando TensorFlow como backend.

3.3 Anaconda

Anaconda [15] es un proyecto de código abierto que se usa para instalar múltiples versiones de paquetes de software y sus dependencias, y cambiar fácilmente entre paquetes de software. Es multiplataforma y principalmente se usa para paquetes de Python, pero se puede usar con cualquier software

En nuestro caso, lo hemos usado para establecer nuestro entorno de trabajo, instalar las librerías de Keras, Tensorflow, CUDA y los controladores de las gráficas que hemos usado en el entrenamiento. Además, al usar varios entornos de trabajo, nos ha facilitado tener la misma configuración para todos ellos.

3.4 CUDA toolkit

Cuda toolkit es una librería de parte de NVIDIA que nos proporciona un entorno de desarrollo para crear aplicaciones de alto rendimiento aceleradas por la tarjeta gráfica. Aunque nosotros no hemos trabajado directamente con el CUDA toolkit, es una dependencia imprescindible para poder entrenar los modelos creados con Keras en la GPU. El uso de esta librería nos permite aumentar en gran medida el rendimiento de nuestro entrenamiento, haciendo posible unos resultados que serían muy costosos de conseguir si nos tuviesemos que basar solo en la CPU para entrenar.

3.5 py-rouge

Py-rouge[13] es una implementación completa de la métrica de ROUGE, la cual es el estándar actual tanto para las tareas de resumir textos como de traducción de textos. En nuestro caso, hemos usado esta librería para facilitar la obtención de resultados con los que poder compararnos con algunos de los modelos explicados anteriormente y entre nuestros propios resultados.

3.6 Sets de datos

Para entrenar las redes neuronales, hemos usado conjuntos de datos diferentes, los cuales procedemos a definir a continuación

3.6.1. Amazon Fine Food Reviews

Este conjunto de datos[11][3] consiste en una serie de valoraciones de los productos de Amazon por parte de los usuarios. Es un conjunto de datos muy extenso, en el cual una muestra consiste en una valoración extendida del producto y una versión muy resumida de la propia valoración. Este es el conjunto de datos que más hemos tenido en cuenta a la hora de entrenar y de probar nuestros modelos. En el anexo, describimos unas pocas valoraciones de muestra de este conjunto de datos.

Este conjunto de datos se puede encontrar en la dirección de la bibliografía de [11].

3.6.2. News

Este conjunto de datos consiste en una serie de noticias de actualidad de varios periódicos. Es un conjunto de datos mucho menos extenso que el anterior. Una muestra consiste en el cuerpo de la noticia y su cabecera. Para conseguir este conjunto de datos, hemos tenido que usar la herramienta de News API[16] para poder conseguir las url de las noticias en cuestión. Luego, con un web scrapper, conseguimos extraer el cuerpo de la noticia para así poder usarlo en nuestro entrenamiento, ya que News API solo nos proporciona las primeras 250 palabras del cuerpo.

CAPÍTULO 4
Descripción del sistema

En este capítulo vamos a describir tanto las capas que hemos usado como los enfoques que hemos seguido para crear nuestro sistema. Primero definiremos las capas que hemos utilizado en nuestros modelos. En segundo lugar describiremos nuestro proceso de obtención de los conjuntos de datos que hemos usado para entrenar nuestros modelos. En tercer lugar describiremos los diferentes enfoques que hemos tomado para preprocesar los datos antes de suministrarlos a los modelos. En cuarto lugar explicaremos la estructura simplificada que siguen nuestros modelos. En quinto lugar comentaremos como hemos realizado nuestros entrenamientos. En sexto lugar definiremos nuestros métodos de creación de resúmenes. Finalmente, comentaremos las métricas estándar usadas actualmente para medir la eficacia de los generadores de resúmenes.

4.1 Capas

4.1.1. Convolucional

Las capas convolucionales se encargan de aplicar un número definido de filtros convolucionales a una imagen. Por cada subregión, de dimensiones también definidas, la capa realiza un conjunto de operaciones matemáticas para producir un solo valor en el mapa de output de la capa. Las capas convolucionales normalmente aplican una funcion de activación ReLU al output para introducir no liniaridades en el modelo.

En nuestro caso, aunque no tratamos exactamente con imagenes, sí que tratamos con vectores tridimensionales, sobre los cuales las capas convolucionales son capaces de aplicar sus filtros como si de una imagen se tratara.

4.1.2. Pooling layers

Las *Pooling layers* son unas capas que se encargan de reducir la dimensionalidad de los datos extraidos por las capas convolucionales, para así reducir el tiempo de procesamiento. El algoritmo de agrupamiento más comunmente usado por estas capas, que es el que usamos en nuestros modelos, es el *Max pooling* que extrae subregiones del mapa de datos de unas dimensiones definidas, se queda con el valor máximo de la subregión y descarta los demas valores.

4.1.3. Fully connected

Para definir una capa *Fully connected*, antes debemos definir una *Fully connected network*. Esta red consiste en varias capas *Fully connected*, donde cada valor en la capa está conectado mediante un peso a todos los valores en la siguiente capa. Por ello, reconocemos la capa *Fully connected* como cada conjunto de valores de esta red. Este tipo de capas son muy usadas para proporcionar los datos de salida en el campo de reconocimiento de imagen.

En nuestro sistema, hemos usado una *Fully connected network* para producir el output a partir de los datos preprocesados con las capas anteriores.

4.1.4. LSTM

Para describir correctamente las redes *LSTM*, hemos de hacer incapié en las redes recurrentes. Las redes recurrentes surgen de la necesidad de recordar lo que se ha procesado previamente, una necesidad muy común cuando trabajamos con lenguaje natural. Por ello, las redes recurrentes contienen ciclos dentro de ellas, que se encargan de hacer que la información persista. Aunque el hecho de una red con bucles internos puede resultar lioso, propongo la siguiente figura, que simplifica el diseño.

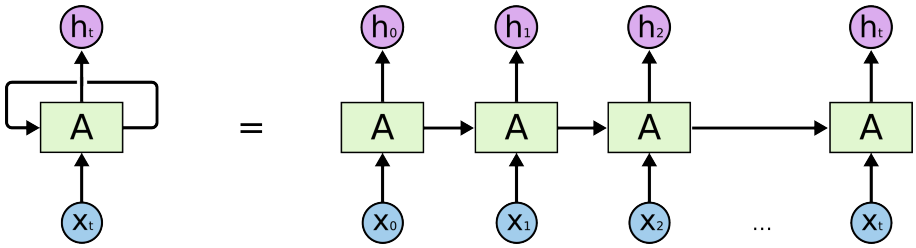

Figura 4.1: Representación simplificada de las redes recurrentes

No obstante, las redes recurrentes poseen un problema, y es que cuando hablamos de grandes volumenes de datos, es fácil que esa información procesada anteriormente se acabe perdiendo si es muy lejana al punto de procesamiento actual, impidiendo que las redes recurrentes cumplan su función. Por este motivo y como solución a este problema, surgen las redes *LSTM*. Las redes *LSTM* son un tipo especial de redes que son capaces de aprender este tipo de dependencias a largo plazo. En [8] se puede encontrar información más detallada sobre este tipo de redes recurrentes.

En nuestro caso, nos hemos decantado por las redes *LSTM* porque son las más usadas actualmente en lenguaje natural, en contraposición al resto de las redes recurrentes.

4.1.5. Softmax

Aunque la *Softmax* no es técnicamente una capa en sí, es una denominación muy común en este ámbito. Cuando hablamos de una capa *Softmax*, nos referimos a una capa *Fully connected* que posee una función de activación *Softmax*, en lugar de la comunmente usada *ReLU*. Esto se debe a que la función de activación *Softmax* nos devuelve un vector de probabilidades cuya suma da 1. Por ello, es una función de activación muy usada cuando el objetivo del modelo es clasificar el input en una clase definida.

En nuestro caso, usaremos este tipo de capa para generar la salida en algunos modelos. En lugar de clases, lo usaremos para devolver las probabilidades de la siguiente palabra a generar.

4.1.6. Concatenate

La función principal de una capa *Concatenate* consiste en juntar los resultados de dos tensores a lo largo de una sola dimensión. En nuestro caso, lo usaremos para juntar los valores que surgen del análisis de la representación del texto como el análisis de las palabras precedentes a la palabra que queremos generar.

4.2 Recopilación de datos

En nuestro sistema, hemos dispuesto de tres conjuntos de datos en total, explicados anteriormente en la descripción de las herramientas.

4.2.1. Recopilación: Amazon Fine Food Reviews

Este conjunto de datos lo hemos conseguido en la página web de Kaggle, de mano de Stanford Network Analysis Proyect bajo una licencia pública de Creative Commons. Para mas información, se puede encontrar en el *snap* de *Amazon Fine Food Reviews* [11].

4.2.2. Recopilación: News

Este conjunto de datos lo hemos recopilado nosotros mismos a lo largo de una serie de pasos que procedemos a comentar.

En primer lugar, hemos hecho uso de la *API* ofrecida por News API [16]. Esta *API* nos ofrece múltiples posiblidades para buscar noticias de diferentes periódicos online. Entre estas búsquedas se pueden recuperar gran cantidad de datos aunque, principalmente, nosotros nos hemos centrado en tres de ellos en cada noticia: el título, el cuerpo de la noticia y la cabecera. En nuestro caso la entrada sería el cuerpo de la noticia, y el objetivo a crear sería la cabecera.

No obstante, nos encontramos con un primer problema debido a las varias limitaciones de esta API. La mas importante de ellas fue que el cuerpo de cada noticia estaba limitado a tan solo las primeras 250 palabras de esta. Este problema lo hemos conseguido solucionar usando otro de los datos que nos proporciona la *API*: el enlace URL a la noticia. Teniendo el enlace, hemos creado un *Web Scrapper* para cada uno de los tres periódicos que hemos elegido. Aunque intentamos conseguir noticias gracias a más periódicos, fue imposible debido a las limitaciones que presentaban las estructuras de las paginas *web* de dichos periódicos.

Una vez conseguido el cuerpo íntegro, el título y la cabecera, nos encontramos el segundo problema que tenía el conjunto de datos. En estas tres partes, existían diferentes tipos de carácteres que no pertenecian a los 128 primeros símbolos de *Unicode*. Esto no solo daba problemas con nuestro programa por errores de compatibilidad con *Unicode*, sino que ademas extendía el vocabulario y reducía la frecuencia de algunas palabras, ya que se creaban con otra entrada por ese carácter diferente.

La mejor solución que conseguimos consistió en buscar individualmente estos carácteres especiales y substituirlos por su equivalente en los primeros 128 caracteres de *Unicode* . Una vez teníamos los códigos *Unicode* de cada uno, los reemplazamos usando el método **re.sub** de *python*. Como la librería Pickle no nos permitía guardar estos carácteres fue relativamente fácil saber cuando los habiamos eliminado todos.

Una vez solucionado este problema, el conjunto de datos ya está listo para su preprocesado.

4.3 Preprocesamiento de los datos

Antes de suministrar cualquiera de los conjuntos de datos a nuestro modelo, hemos tenido que hacer varias operaciones a los datos despues de cargarlos en memoria, para así poder usarlos correctamente.

4.3.1. Word Embeddings

Para crear una representación individual de cada palabra que suministrar a nuestro modelo, hemos seguido tres tipos de enfoques.

word2vec

El primero de ellos fue Word2Vec. Para ello, usamos representaciones pregeneradas que forman parte de GloVe [5]. Este Word2Vec preentrenado viene con representaciones de distintos tamaños. En nuestro caso, hemos optado por la representación de dimensionalidad R^{200}.

Super Characters

En segundo lugar, optamos por un método cercano a este usando *Super Characters*. Este enfoque consiste en crear una representación de cada palabra a partir de una imagen en la que está escrita esta misma palabra. Para ello, esta imagen se convierte en un vector de representación compacta con un autoencoder hecho con capas convolucionales y se entrena para que la pérdida sea mínima. Una vez logrado esto, solo queda pasar la palabra que queremos codificar por la parte del encoder y obtenemos la representación requerida.

One-Hot-Vectors

Por último, hemos usado *one-hot-vectors* para codificar las palabras. En este caso, la representación consiste en un vector de un tamaño fijo,en nuestro caso R^{10000} , con todas las posiciones situadas a 0, excepto una posición que se encuentra con valor 1. Por tanto, la posición que tiene valor 1 representa la palabra que queremos codificar. Para crear esta representación, hemos tenido que analizar la frecuencia de todas las palabras que queriamos utilizar. Las características de la representación son las siguientes:

1. Las primeras 9997 posiciones corresponden a las 9997 palabras mas frecuentes en los resumenes de las noticias.

2. La posición 9998 pertenece a un token que simboliza el principio del resumen.

3. La posición 9999 pertenece a un token que simboliza el final del resumen.

4. Por último, la posición 10000 corresponde al token de **unk**, que simboliza que la palabra que intentabamos codificar no se encuentra dentro de las 9997 primeras. Por tanto, si estamos codificando una palabra y no tiene una entrada en las 9997 palabras más frecuentes, la codificaremos como **unk**.

4.3.2. Representación del texto

En el caso de nuestra representación del texto, hemos seguido varios enfoques a lo largo de nuestro experimento

Autoencoder

Nuestra primera representación consiste en procesar el texto con un autoencoder de secuencias. Este autoencoder esta formado por una red *LSTM* con una red *Fully connected* al final. Esta estructura se puede apreciar en la figura 4.2.

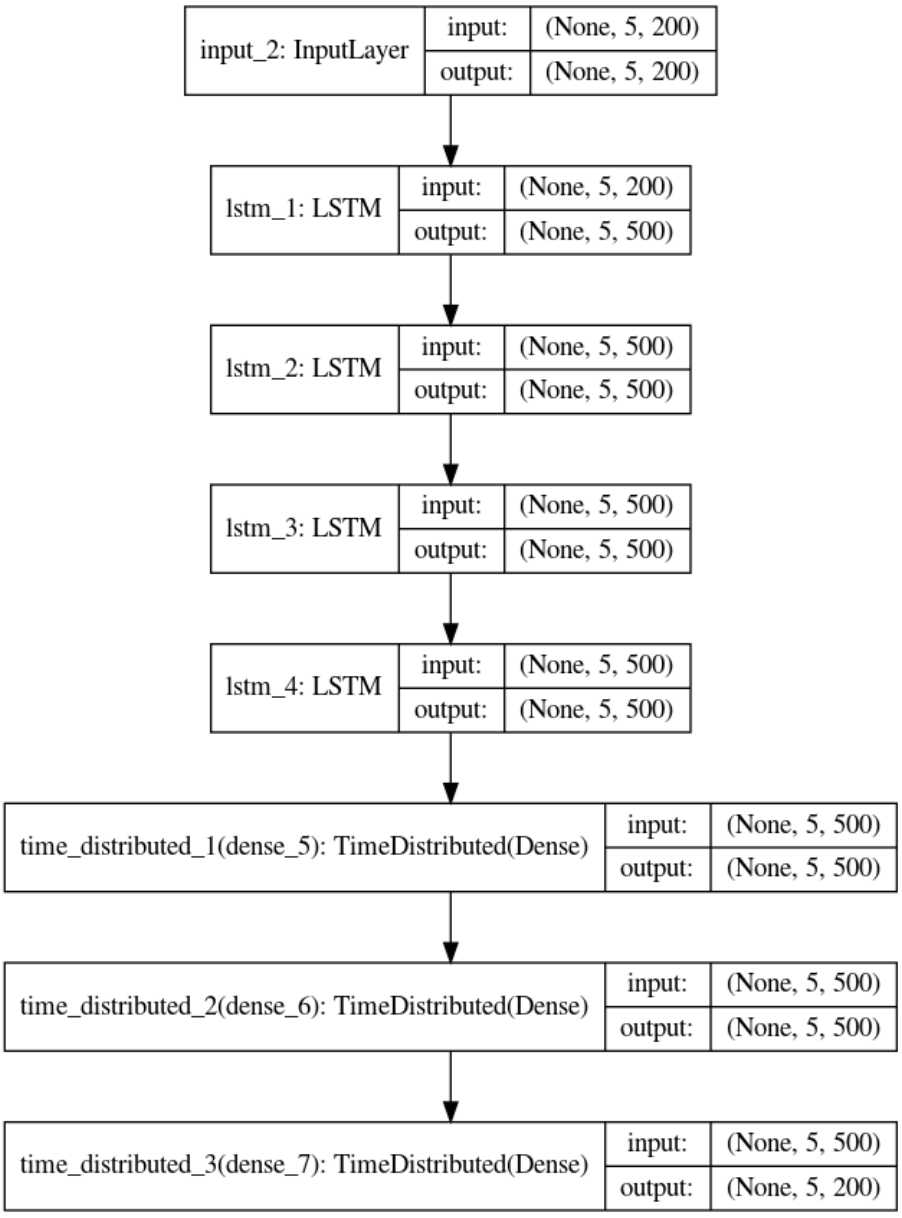

Figura 4.2: Estructura interna del autoencoder de secuencias

Los pasos son los siguientes para generar la codificación de una secuencia:

1. Elegimos el tamaño de ventana que queremos para nuestra secuencia. En el caso de nuestro autoencoder, hemos elegido un tamaño **Timesteps = 5**.
2. Se coloca la ventana al principio del texto.

3. Introducimos la secuencia dentro de la ventana al autoencoder, y nos guardamos la representación de esa secuencia
4. Movemos la ventana una palabra hacia el final del texto
5. Repetimos los pasos 3 y 4 hasta realizarlos con la última palabra del texto.

Esta és la idea fundamental detrás de este tipo de codificación. El como usaremos esas representaciones en los distintos modelos que hemos generado será explicado en el propio modelo.

Voxels

Para nuestro experimento hemos decidido usar un nuevo enfoque. Nuestro método consiste en convertir el texto a un equivalente en un vector tridimensional, como si de una imagen 3D se tratara. Para ello, seguimos los siguientes pasos.

En primer lugar, hemos seleccionado las 54872 palabras mas frecuentes en los textos. Esto se debe a que hemos adjudicado a nuestro vector una dimensionalidad de $R^{38x38x38}$. Hemos elegido esta dimensionalidad debido a que posee un buen equilibrio entre capacidad de palabras y tamaño para que los batches sean lo más reducidos posibles. Tras elegir estas 54872, las hemos ordenado alfabéticamente, y les hemos asignado una posición en el vector tridimensional, con los tres índices correspondientes.

Cuando ya tenemos un diccionario donde la clave es la palabra y su valor es el índice que representa, comienza la conversión del texto. y procedemos a llenarlo de la siguiente manera:

1. Inicializamos un vector $R^{38x38x38}$ a ceros .
2. Marcamos la primera palabra del texto. Marcar la palabra consiste en aumentar el valor de la posición del vector en la que se encuentra esa palabra en nuestro vector tridimensional.
3. Seleccionamos la palabra siguiente a la última palabra que hemos marcado y marcamos el valor de la posición de esa palabra en el vector.
4. Dibujamos una línea a traves del vector tridimensional desde la penúltima palabra que hemos añadido hasta la última que hemos añadido. Esta línea que dibujamos la vamos a crear haciendo uso del algoritmo de Bresenham[1], y aumentaremos las celdas que contienen la linea en un cierto valor.
5. Repetimos el tercer y el cuarto paso hasta que hayamos hecho el proceso con todas las palabras del texto. Entonces, el vector estará preparado.

La razón de ordenar las palabras alfabéticamente es para tener una distribución de las palabras en el vector en la cual las palabras más frecuentes no se encuentren en un espacio reducido del vector. Esto podría generar mucho ruido y, consecuentemente, degenerar los resultados. Por ello, el orden alfabético es un buen orden para asignar los indices a las palabras.

Bag-Of-Words

Nuestra versión de *Bag-Of-Words* consiste en un vector R^{10000} donde una posicion **0 <= i <= 9999** contiene el número de ocurrencias de la palabra con indice **i** en el texto.

En nuestro caso, solo contabilizaremos las palabras que se encuentran dentro de nuestro vocabulario elegido de tamaño R^{10000}, descartando las demas. En este vocabulario se encuentran las 10000 palabras mas frecuentes en los textos, las cuales creemos que contendrán la mayor cantidad de información a extraer.

4.4 Modelo

Para explicar nuestro modelo, hemos explicar primero tanto los datos de entrada como los datos esperados. Los datos de entrada consisten en, por un lado, la representación que hemos elegido para el texto o cuerpo de la noticia y, por otro lado, un número *Timesteps* de palabras previas en el resumen que vamos a generar. De esta manera, para una muestra con estos datos de entrada, queremos generar la siguiente palabra del resumen a partir de las *Timesteps* anteriores y de la representación del texto.

Una vez explicado en qué consiste una muestra, pasamos a los optimizadores. En nuestro entrenamiento, hemos usado **RMSprop**, **AdaGrad** y **AdaDelta**. Para la pérdida, hemos usado **Categorical crossentropy** cuando el output se encuentra en formato *one-hot-vector*. En nuestros primeros modelos, como este no era el caso, hemos aplicado **mean_squared_error** para representar la pérdida. A continuación, comentaremos las diferentes capas de nuestro modelo:

1. En primer lugar, hemos preparado una serie de capas LSTM para recibir a las **Timestep** palabras previas a la palabra que vamos a generar. Hemos elegido el tipo de capa *LSTM* dado que son capaces de tener en cuenta la secuencialidad en la que se le presentan a la entrada las palabras previas.

2. En segundo lugar, hemos utilizado varias capas convolucionales para extraer una representación de los voxels que pueda concatenarse con la representación de la entrada. Hemos elegido capas convolucionales ya que son un tipo de capa que ha avanzado mucho en estos ultimos años y son perfectas para analizar vectores multidimensionales.

3. En tercer lugar, hemos preparado una capa que concatena el output de estas dos primeras partes de la red, para poder tratarlas conjuntamente.

4. Por último, hemos preparado una serie de capas *Fully connected* para producir el output en los distintos formatos que hemos ido probando a lo largo de nuestros experimentos.

4.5 Entrenamiento

Para hacer el entrenamiento, hemos hecho uso de una estructura de datos llamada *Data Generator*. Esta estructura, compatible con Keras, nos permite implementar varios métodos que nos facilitan el trabajo. Además, nos proporciona una función fundamental para nuestro trabajo, que consiste en ser capaces de preparar cada batch de entrenamiento manualmente. De esta forma, podemos crear la representación del texto de cada muestra del batch antes de usarlo directamente en el entrenamiento. Si tuviesemos que mantener en memoria todas las representaciones de cada texto, nos sería imposible debido a la falta de espacio. Es por ello que usamos esta estructura para hacer posible la carga de los datos.

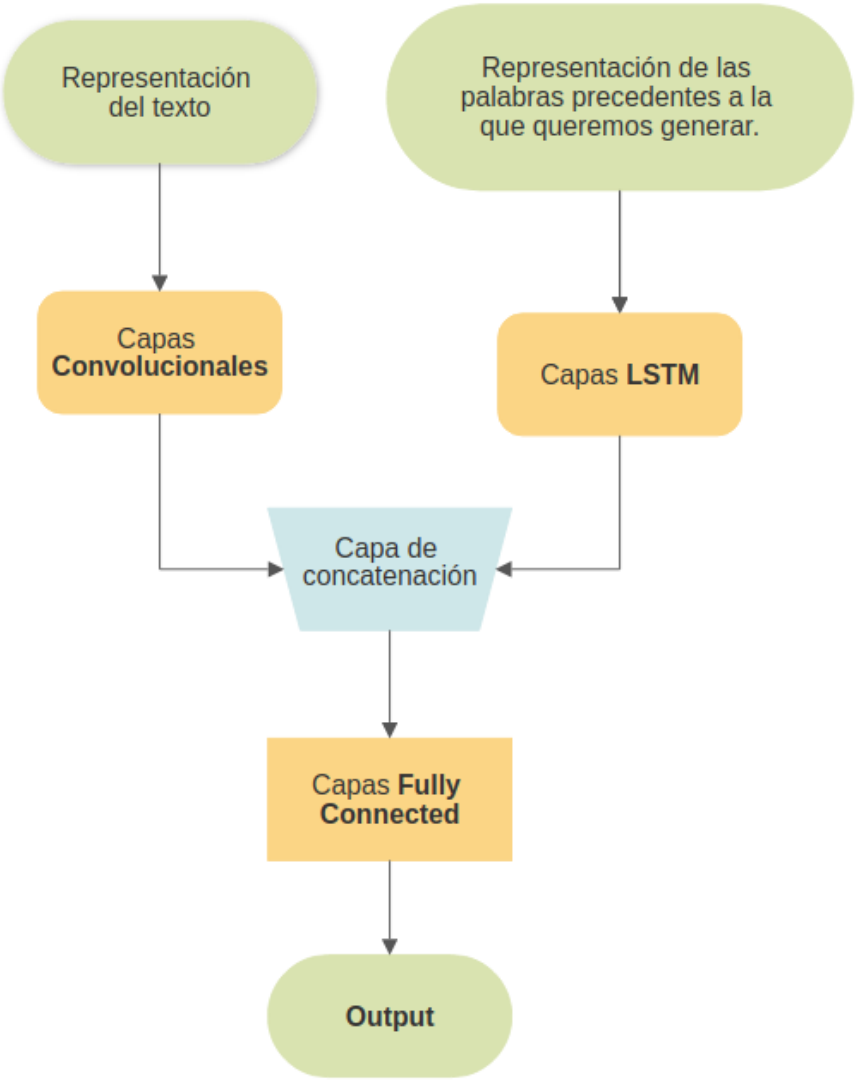

Figura 4.3: Representación simplificada y generalizada para todos los modelos que vamos a describir.

El entrenamiento lo hemos efectuado de la siguiente manera. Una vez tenemos cargados los datos de las noticias, tanto el cuerpo como el resumen objetivo, procedemos a generar un vector por cada história.

En este vector, queremos almacenar tanto las palabras del resumen con su correspondiente representación como la representación de los tokens de inicio y final de resumen. En concreto, generaremos un vector de tamaño $R^{\textbf{Timesteps}+\textbf{Resumen.length}+1}$. Las primeras **Timesteps** posiciones seran ocupadas por la representación del token de inicio de resumen. Las consecuentes posiciones contendrán la representación de cada palabra del resumen en el orden en el que aparecen en el resumen. Por último, la última posición corresponderá al token de final del resumen.

Una vez el vector de cada noticia está preparado, procedemos a generar las muestras que le suministraremos a nuestro sistema para poder entrenarlo. Para la muestra, no introduciremos la representación del texto. En su lugar, introduciremos el texto como palabras dentro de la muestra. De esta forma, cuando el batch esté apunto de entrenar, sustituirá este texto por su correspondiente representación en nuestro vector tridimensional. En el caso de las palabras previas al resumen, nuestro enfoque depende de la

metodología seguida para representar las palabras individuales. Si hemos usado *Super caracters*, guardaremos la representación directamente en la muestra. Si por el contrario hemos usado *one-hot-vectors*, introduciremos en la muestra el indice del vector que hay que activar. De esta manera, ahorraremos mucho espacio en memoria, sustituyendo un vector de tamaño R^{10000} por un solo entero. El procedimiento para elegir la palabra resultante es el mismo que para la representación en *one-hot-vector*.

Por cada noticia, generaremos **Resumen.length** muestras. Esto se debe a que usaremos una ventana en el vector generado para cada noticia de tamaño $R^{\mathbf{Timesteps}+1}$. Con esta ventana, recorreremos el vector moviendo la ventana hacia la derecha de una en una posición. Antes de mover la ventana, generaremos una muestra con las **Timesteps** primeras palabras como palabras previas y la última palabra de la ventana como palabra objetivo.

Una vez hemos creado todas estas muestras, procedemos a mezclarlas al principio del entrenamiento y al final de cada *Epoch*. Cuando empieza el entrenamiento, debemos preparar cada batch antes de que entre al entrenamiento. Para ello, crearemos la representación en forma de vector tridimensional del texto, y sustituiremos los índices de *one-hot-vectors* por sus vectores correspondientes, con la posición indicada activada y el resto a cero.

El modelo de Keras, al llamar a la función de *fit*, automáticamente nos proporciona tanto la pérdida como el porcentaje de acierto del sistema en cada *epoch*. Al final de este, también nos enseña el porcentaje de acierto y la pérdida pasando el set de validación por el modelo.

4.6 Generación del resumen

Para la generación del resumen, definiremos dos técnicas diferentes para mantener y desarrollar hipótesis de forma controlada, para que el tiempo de ejecución sea aceptable.

Para evitar confusiones, definiremos como funcionan las hipótesis en los dos algoritmos: Las hipótesis serán una lista que contendrá dos objetos. Uno es una lista de tres palabras o más. El otro objeto será la *log probabilidad* de que esa hipótesis sea cierta. Cuando queramos calcular sus hipótesis hijas, las palabras que meteremos en el modelo serán las tres últimas en el orden en el que estan dispuestas en la lista, junto al voxel ya pregenerado. El método de elección para las palabras siguientes está definido en cada algoritmo.

Usaremos a *log probabilidad* para denotar la probabilidad de una hipótesis ya que dejar la probabilidad normal podría llegar a dar fallos de precision, y si normalizaramos la probabilidad se perdería información.

4.6.1. Viterbi

En primer lugar, hemos adaptado el algoritmo de Viterbi para nuestro problema.

Al algoritmo también se le suministra un parametro **BEAM**. La función de este parámetro es determinar como de probables han de ser las hipótesis respecto de la hipótesis más probable de la lista **current** para que la exploremos o, en caso contrario, la descartemos si no llega a este umbral.

A continuación, procedemos a explicar como funciona el algoritmo:

1. Generamos la representación del texto una única vez.
2. Preparamos una hipótesis inicial que contiene el token de inicio de resumen tres veces.

3. Creamos una cola de hipótesis actuales, desde ahora **current**, y otra cola con las siguientes hipótesis, desde ahora **next**.
4. Creamos una cola de hipótesis completadas, desde ahora **completes**
5. Metemos la hipótesis inicial en la cola **current**.
6. Mientras que **current** contenga hipótesis sin procesar, se ejecutan los siguientes pasos:
   a) Se saca la siguiente hipótesis **h** de la lista **current**.
   b) Si la última palabra de **h** es el token de final de resumen, la añadimos a **completes** y volvemos al principio de este bucle.
   c) Si la *log probabilidad* de **h** es menor que la *log probabilidad* de la hipótesis más probable menos el **BEAM**, **h** se descarta, y se pasa a explorar la siguiente hipótesis en la lista **current**.
   d) Generamos el *one hot vector* resultante de pasar **h** por el modelo entrenado, como hemos explicado anteriormente.
   e) Seleccionamos las palabras más probables y las vamos insertando en una nueva lista, que llamaremos **posibles**, hasta que la suma de las probabilidades de las palabras dentro de **posibles** alcance un 95 % o mayor.
   f) Por cada palabra en **posibles**, generamos una nueva hipótesis, añadiendo a una copia de h dicha palabra e insertándola en **next**.
   g) Repetir estos pasos con la siguiente hipótesis
7. Ahora que la lista **current** esta vacía, convertiremos la lista **next** en la lista **current**.
8. Repetiremos los pasos 6 y 7 un número de veces igual al máximo tamaño de resumen que queremos obtener.
9. Obtendremos la hipótesis más probable en la lista **completes** y la devolveremos como resultado de la generación del resumen.

4.6.2. Beam Search

En segundo lugar, queremos describir el algoritmo de *Beam Search*, presente en el artículo [9]. Para este algoritmo, necesitaremos suministrarle, además del cuerpo de la noticia, un parametro llamado **Beam Width**. Este parámetro se encarga de elegir cuantas hipótesis mantenemos al mismo tiempo. El algoritmo funciona de la siguiente manera:

1. Generamos el voxel perteneciente al texto una única vez.
2. Preparamos una hipótesis inicial que contiene el token de inicio de resumen tres veces.
3. Creamos una cola de hipótesis actuales, desde ahora **current**, y otra cola con las siguientes hipótesis, desde ahora **next**.
4. Creamos una lista para guardar las hipótesis terminadas, desde ahora **completes**.
5. Generamos el *one hot vector* resultante de pasar h por el modelo entrenado, como hemos explicado anteriormente.
6. Elegimos las **Beam Width** palabras más probables y, añadimos cada una por separado a una copia de la hipótesis inicial. Estas hipótesis las guardaremos en **current**.

7. Vaciamos la lista **next**
8. Mientras current contenga alguna hipótesis:
    a) Sacamos la primera hipótesis de la lista, desde ahora h.
    b) En el caso de que sea una hipótesis cuya última palabra sea el token de final de resumen, lo añadimos a **completes** y reducimos el **Beam Width** en 1. En este caso, continuariamos con la siguiente hipótesis.
    c) Calculamos las **Beam Width** palabras siguientes más probables de la hipótesis h, creamos su correspondiente hipótesis y las añadimos a la lista **next**, junto a su **log probabilidad** acumulada con la de la hipótesis madre.
9. Una vez current esta vacía, seleccionamos las **Beam Width** hipótesis mas probables y las insertamos en la lista **current**
10. Si **Beam Width** es mayor que cero, volvemos al paso 7.
11. Finalmente, seleccionamos la hipótesis más probable entre las que se encuentran dentro de la lista **completes**.

Al final del experimento decidimos no usar este algoritmo en nuestra generación de los resúmenes, ya que creemos que *Viterbi* contempla más hipótesis que este algoritmo.

4.7 Métricas

En la generación de resúmenes no existe una forma segura de medir la eficacia real de un generador. Esto se debe a que aunque un sistema no otorgue el resumen esperado, dicho resumen no ha de ser necesariamente erróneo. Por ello, a lo largo de los últimos años se han definido dos tipos de métricas complementarias, *ROUGE* y *BLEU*. En nuestro caso, también es necesario poseer unas métricas para medir la eficacia real de nuestro modelo. Aunque la propia libreria de *Keras* nos otorga una medida de acierto en el entrenamiento y en la validación, esta medida solo nos dice en cuantos casos la palabra generada era la palabra esperada. El problema de esta probabilidad de acierto proviene de que no diferencia si se trata de la primera palabra de un resumen que has de generar, o de una palabra para la cual ya tienes las tres precedentes. Hay una diferencia substancial en la cantidad de información que posee cada muestra. Además, las primeras palabras siempre son más importantes que las consecuentes, ya que las primeras palabras generadas también participan en la generación de las palabras siguientes.

4.7.1. Precision y Recall

Para hablar de las diferentes métricas usadas actualmente para medir la calidad de un resumen, primero hemos de definir dos conceptos basicos, la precision y el *Recall*. Ambas medidas son complementarias, y nos permiten evitar los resumenes excesivamente largos o excesivamente cortos.

En primer lugar, la precisión se encarga de medir la cantidad de palabras relevantes dentro del resumen generado respecto del resumen esperado. Es decir, qué porcentaje de las palabras del resumen generado se encuentran dentro del resumen esperado. Esto nos puede provocar varios problemas, como repetir varias veces una palabra muy común nos dara unos porcentajes muy altos de precisión. Aquí es donde entra en juego el *recall*

En segundo lugar, el *recall* se encarga de medir la cantidad de palabras del resumen esperado que se encuentran dentro del resumen generado. Esto nos puede provocar un

problema, y es un resumen excesivamente largo, con muchos tipos de palabras diferentes. El tener muchas palabras nos aumentará las posibilidades de que las palabras del resumen esperado se encuentren en nuestor resumen generado.

Estas dos medidas son complementarias, y aunque cuando se usan juntas no evitan todos los problemas, si que se consideran los mas comunes.

4.7.2. ROUGE

La métrica *ROUGE* se encarga de medir cual es la precisión y el *recall* de un sistema. En sí, la propia métrica se encarga de ofrecer diferentes variantes de la misma para tener una medición más acertada de las características del resumen. En concreto, explicaremos las tres más comunes, que son las que usaremos.

1. La primera de ellas es *ROUGE-1*. Esta variante de la métrica calcula la precisión y el *recall* como hemos explicado previamente, a partir de los unigramas del resumen generado y del resumen esperado.
2. La segunda de ellas es *ROUGE-2*. Esta variante es similar a *ROUGE-1*, pero usa bigramas en vez de unigramas.
3. La tercera de ellas es *ROUGE-L*. Esta variante es bastante diferente de las anteriores. Lo que hace es buscar la cadena común mas larga contenida dentro de ambos resúmenes.

Existen más tipos de métricas, y la *ROUGE-1* y *ROUGE-2* se pueden agrupar en la denominada *ROUGE-N*, que incluye todos los tipos de n-gramas.

4.7.3. BLEU

La métrica *BLEU* consiste en una modificación del cálculo de la precisión para evitar diversos problemas. Es una métrica usada comunmente en traduccion de lenguaje natural entre idiomas, aunque posee ciertos precedentes a la hora de calcular la precisión en un resumen generado por una máquina. No la usaremos en nuestro problema, pero nos parece importante comentarla.

El cálculo de la precisión según la métrica *BLEU* consiste en que, de las palabras en el resumen generado, solo contarán como que se encuentran dentro del resumen esperado si aun no se ha llegado al máximo de ocurrencias en esa palabra dentro del resumen esperado. Su utilidad es mas clara en la figura 4.4:

Example of poor machine translation output with high precision

Candidate	the	the	the	the	the	the	the
Reference 1	the	cat	is	on	the	mat	
Reference 2	there	is	a	cat	on	the	mat

Figura 4.4: En esta figura se puede observar un resumen generado y dos resumenes esperados. En precisión normal, el resultado sería 1, ya que todas las palabras del resumen generado se encuentran en el resumen esperado. En la métrica de *BLEU*, la precisión sería de 2/7, debido a que la palabra **the** aparece un máximo de dos veces en el resumen esperado 1. Por tanto, las cinco siguientes palabras **the** son clasificadas como erróneas. Fuente: [18].

CAPÍTULO 5
Experimentación y resultados

En este capítulo, presentaremos los modelos que hemos entrenado a lo largo del experimento, mostraremos los resultados obtenidos y los compararemos con un artículo que ha publicado sus resultados para el mismo conjunto de datos.

5.1 Evolución del experimento

En esta sección, contaremos nuestra progresión a traves de los modelos, los cambios que hemos ido haciendo y los problemas que hemos ido solucionando.

5.1.1. Primer modelo

Nuestro primer modelo usa el autoencoder de secuencias como representación del texto siguiendo estos pasos:

1. Mediante el encoder ya entrenado, consigue la representación de todas las secuencias posibles en el texto.
2. De estas secuencias, selecciona la codificación con los valores máximos, desde ahora **max**.
3. Selecciona la codificación con los valores mínimos, desde ahora **min**.
4. Crea una codificación artificial, que contiene en cada valor la media de todas las codificaciones del texto, desde ahora **mean**.

Una vez realizados estos pasos, los tres vectores, **max**, **min** y **mean**, contendrán nuestra representación del texto.

Para nuestra representación de las palabras precedentes en el texto, lo que hemos hecho ha sido usar **timesteps** vectores con las representaciones en glove de dimensionalidad R^{200} de estas palabras. En este caso, hemos usado **timesteps=3**. A la hora de procesar esta información, hemos usado una red de capas *LSTM*.

Cuando hemos procesado ambas partes por separado, las juntamos con una capa *Concatenate* y las pasamos por una red de capas *Fully connected*, hasta que nos da como output una representación de la palabra resultante en el mismo formato que las palabras precedentes, en este caso un vector de dimensionalidad R^{200}. El sistema en conjunto se puede apreciar en la figura 5.1. Sabemos que la visualización es muy pequeña para el documento, pero queremos enseñar el modelo en la totalidad de su conjunto.

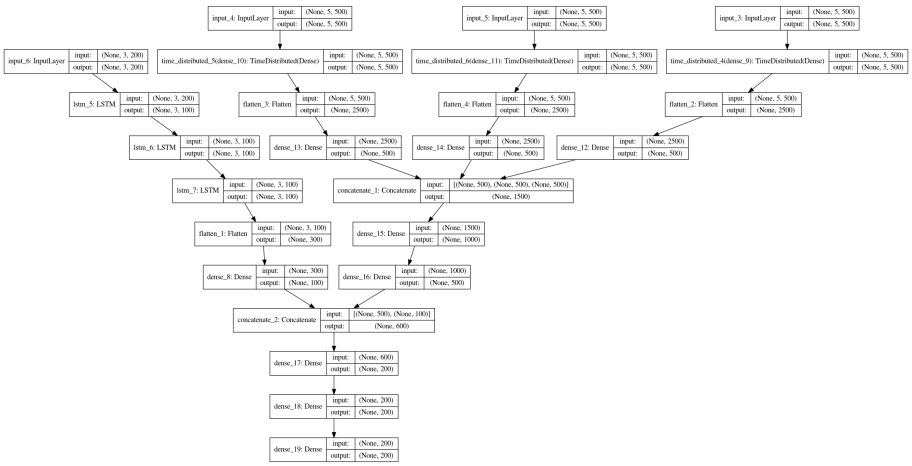

Figura 5.1: Estructura del primer modelo

Para elegir la palabra real generada por el sistema, planeamos usar el algoritmo *KD-Trees*, pero este modelo no logró conseguir una *loss* que nos pareciese aceptable para este tipo de output. Por ello, diseñamos un nuevo modelo y nos apartamos de este.

5.1.2. Segundo modelo

Nuestro segundo modelo parte de la idea de que la representación que estamos usando del texto no es la correcta. Esto se debe a que estamos dándole mucha importancia a los límites, los cuales no tienen por que ser una representación fiable del texto. Como alternativa, proponemos una representación que sigue estando basada en el autoencoder, pero de forma diferente. Lo que haremos esta vez para representar el texto será pasarlo enteramente por el encoder y quedarnos únicamente con la representación generada para la última ventana del texto. Segun la naturaleza de las *LSTM*, el último estado generado debería contener suficiente información del texto como para ser relevante a la hora de generar el resumen.

Para la representación de las palabras precedentes usaremos la misma estructura que en el primer modelo. No obstante, cambiaremos el calculo del output en los siguientes puntos:

- En vez de una capa *Concatenate* usaremos una capa *Add* Esta capa recibe dos vectores de la misma dimensionalidad y suma sus contenidos.
- Tras pasar por la capa *Add*, pasará a una capa *Repeat Vector*. Su función es convertir el vector que recibe en tres instancias del mismo vector.
- En vez de una red de capas *Fully connected*, usaremos una red de capas *LSTM*.

La red resultante puede ser apreciada en la figura 5.2

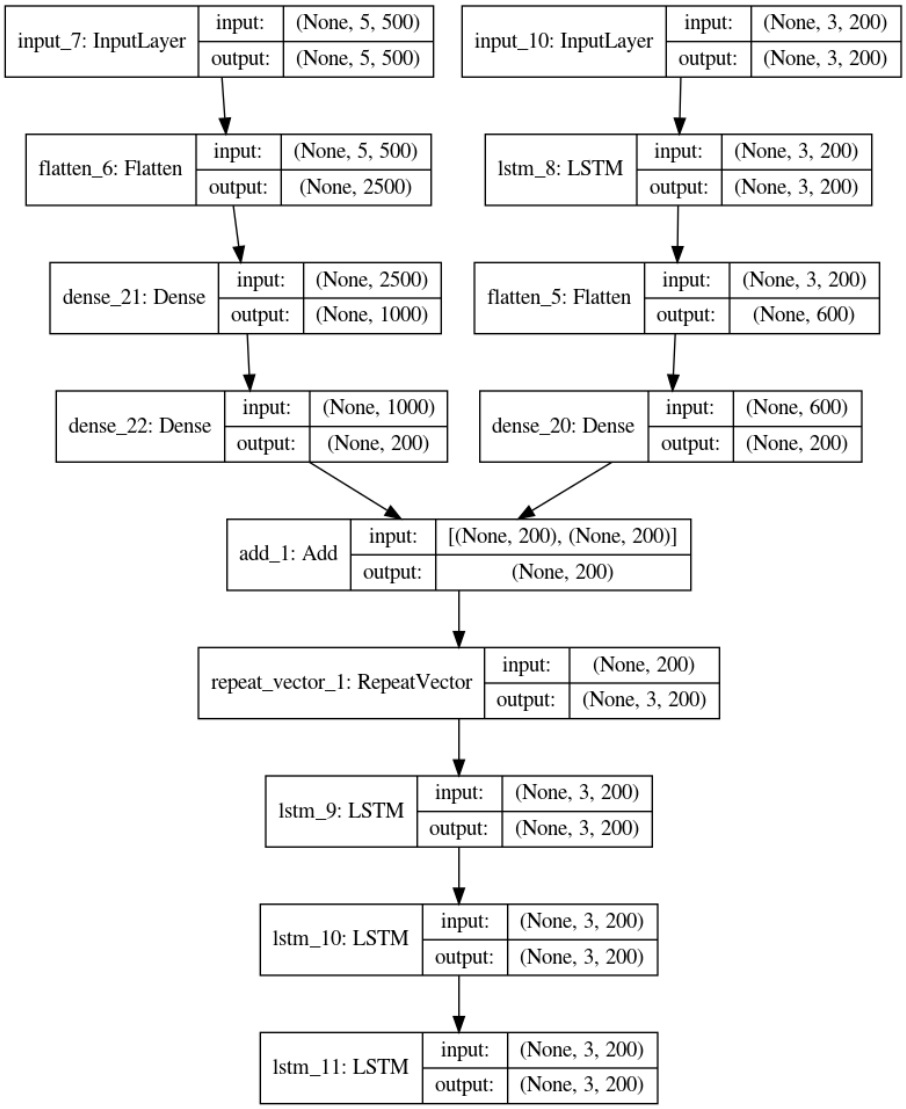

Figura 5.2: Estructura del segundo modelo

Los resultados fueron similares a los ofrecidos en el primer modelo. El modelo no parecía aprender, así que también abandonamos este tipo de modelo y pasamos al siguiente. Los resultados del entrenamiento se pueden apreciar en las figuras 5.3 y 5.4.

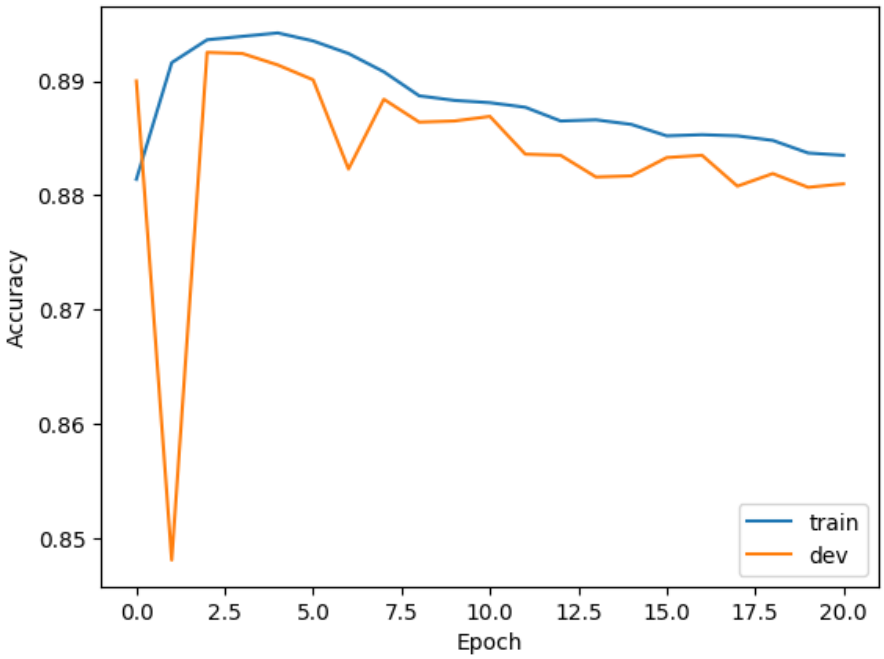

Figura 5.3: Gráfica representando la evolución del *accuracy* del segundo modelo

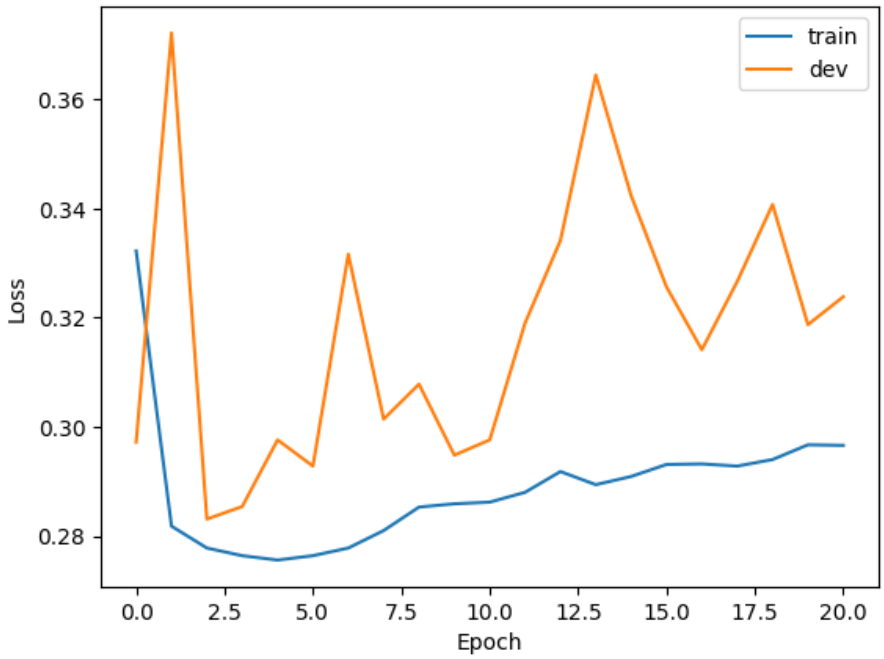

Figura 5.4: Gráfica representando la evolución del *loss* del segundo modelo

5.1.3. Tercer modelo

El tercer modelo posee varios cambios respecto al segundo modelo:

- Para la representación del texto, pasamos a coger el último de los 5 vectores generados por el autoencoder en la ventana final del texto.
- Para la representación de las palabras precedentes, pasamos a usar *Supercharacters*, con el sistema que hemos entrenado previamente.
- Volvemos a usar la capa *Concatenate* para juntar ambas representaciones
- También cambiaremos la red de capas *LSTM* por una red de capas *Fully connected*.

El modelo completo se puede apreciar en la figura 5.5. Los resultados fueron similares al primer y segundo modelo. Se puede observar la evolución del entrenamiento en las figuras 5.6 y 5.7.

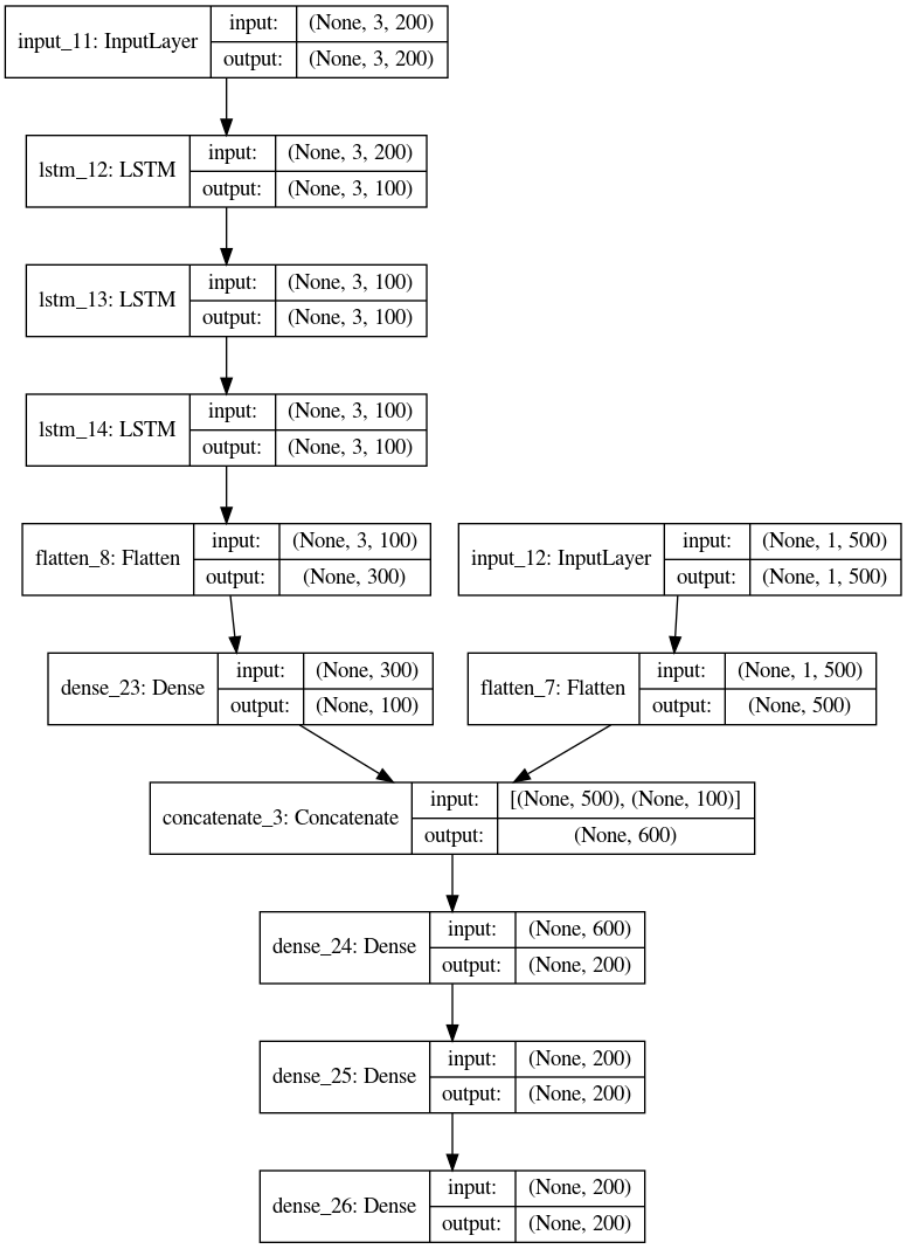

Figura 5.5: Estructura del tercer modelo.

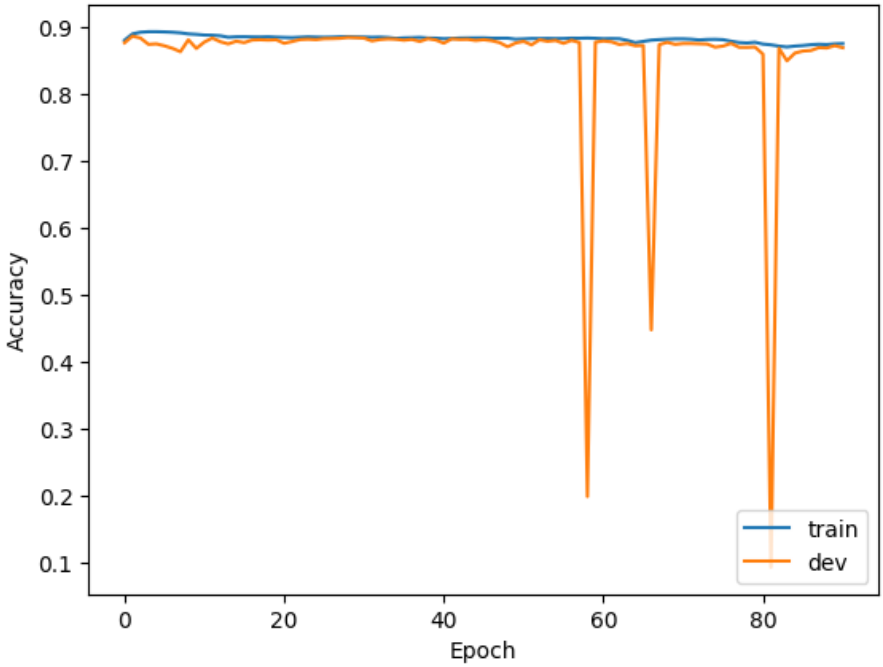

Figura 5.6: Gráfica representando la evolución del *accuracy* del tercer modelo.

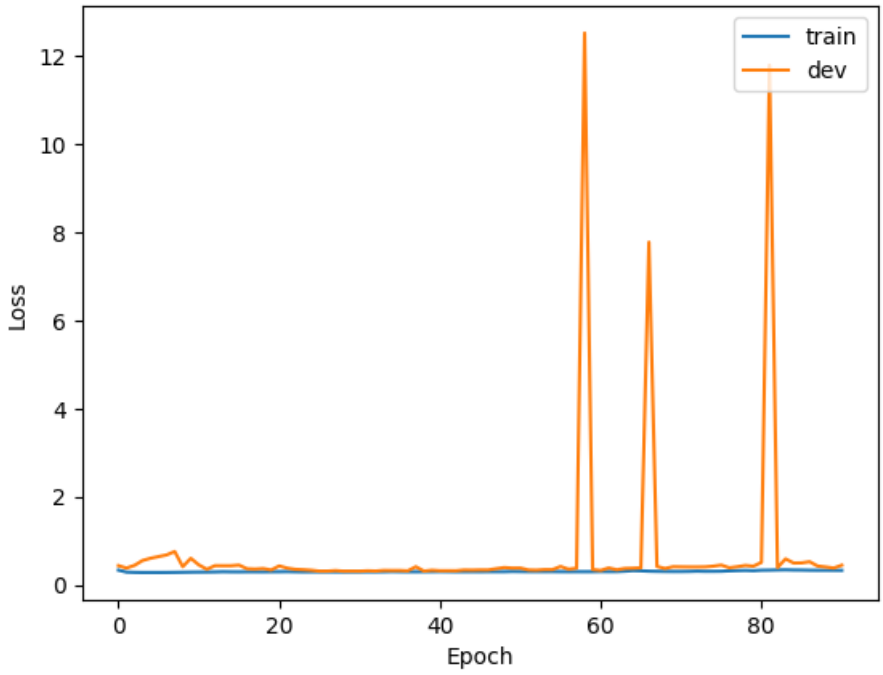

Figura 5.7: Gráfica representando la evolución del *loss* del tercer modelo.

5.1.4. Quinto modelo

No comentaremos el cuarto modelo por su similaritud con los modelos anteriores. Para el quinto modelo decidimos hacer unos cambios importantes en la estructura de los modelos:

- Pasamos a usar los *Voxels* como representación del texto. Esta representación la analizamos mediante una red de capas *Convolucionales*.

- Para la representación de las palabras precedentes, usamos una codificación de *One-hot-vectors*, explicado en la sección 4.3.1.

- En vez de usar una capa *Fully Connected* con una función de activación *ReLU*, usaremos una función de activación *Softmax* en la última capa.

Para codificar la palabra resultante, usaremos el método de *One-hot-vectors*, explicado en la sección 4.3.1. También hicimos algunos cambios en la red convolucional que analiza la representación del texto. El modelo resultante se puede apreciar en la figura 5.8. Además, los resultados del entrenamiento se pueden apreciar en las figuras 5.9 y 5.10. Como los resultados fueron demasiado bajos en *accuracy*, decidimos pasar a otro tipo de modelo.

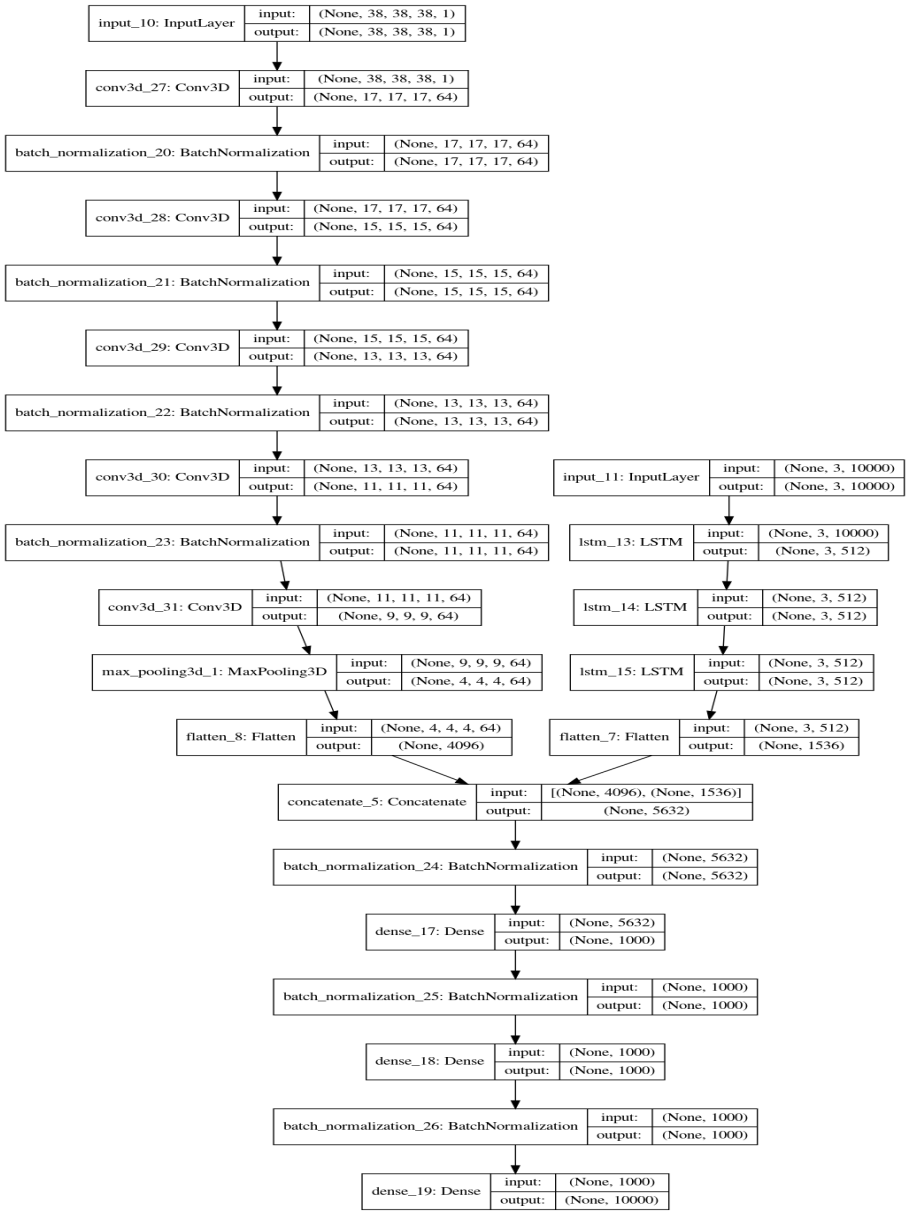

Figura 5.8: Estructura del quinto modelo.

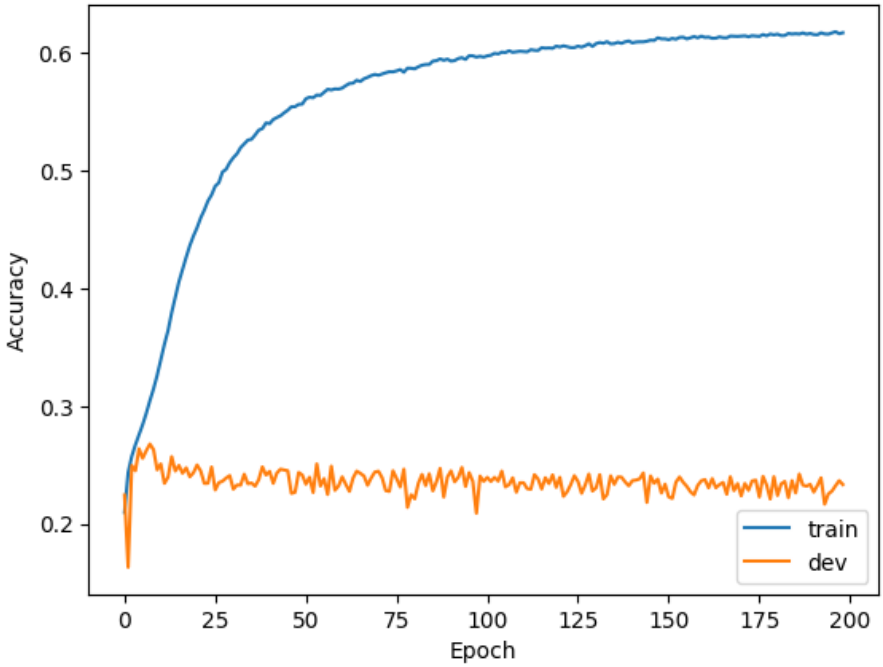

Figura 5.9: Gráfica representando la evolución del *accuracy* del quinto modelo.

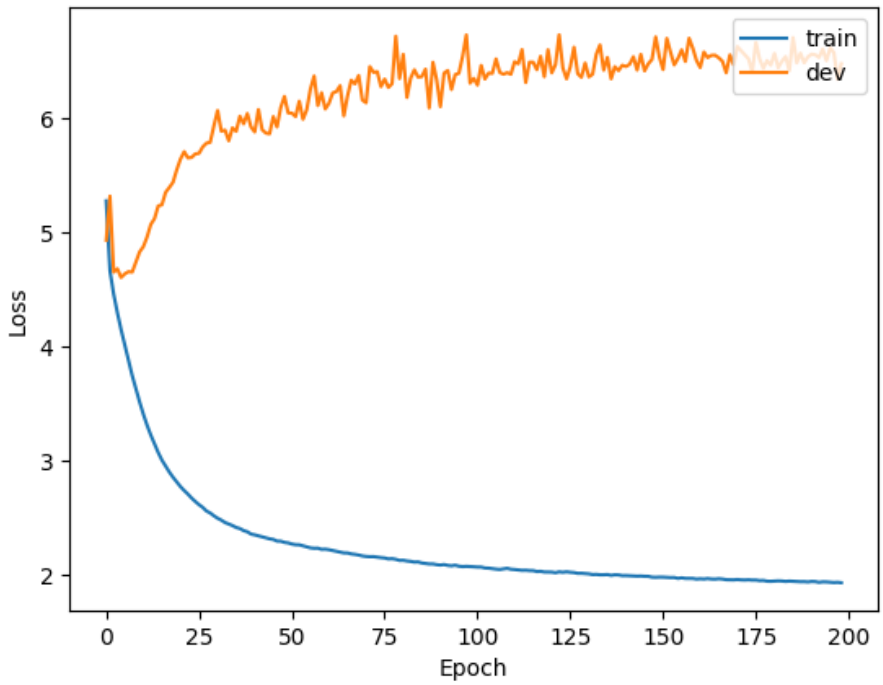

Figura 5.10: Gráfica representando la evolución del *loss* del quinto modelo.

5.1.5. Octavo y noveno modelo

Para el octavo y noveno modelo, hemos hecho cambios significativos en el modelo.

- Hemos usado para la codificación de las palabras precedentes una *Embedding layer*.
- Hemos cambiado la representación del texto por una *Bag-Of-Words*.

Comentamos conjuntamente ambos modelos porque son muy parecidos. Sus diferencias se pueden apreciar en las siguientes figuras. La figura 5.11 pertenece al octavo modelo y la figura 5.12 pertenece al noveno modelo.

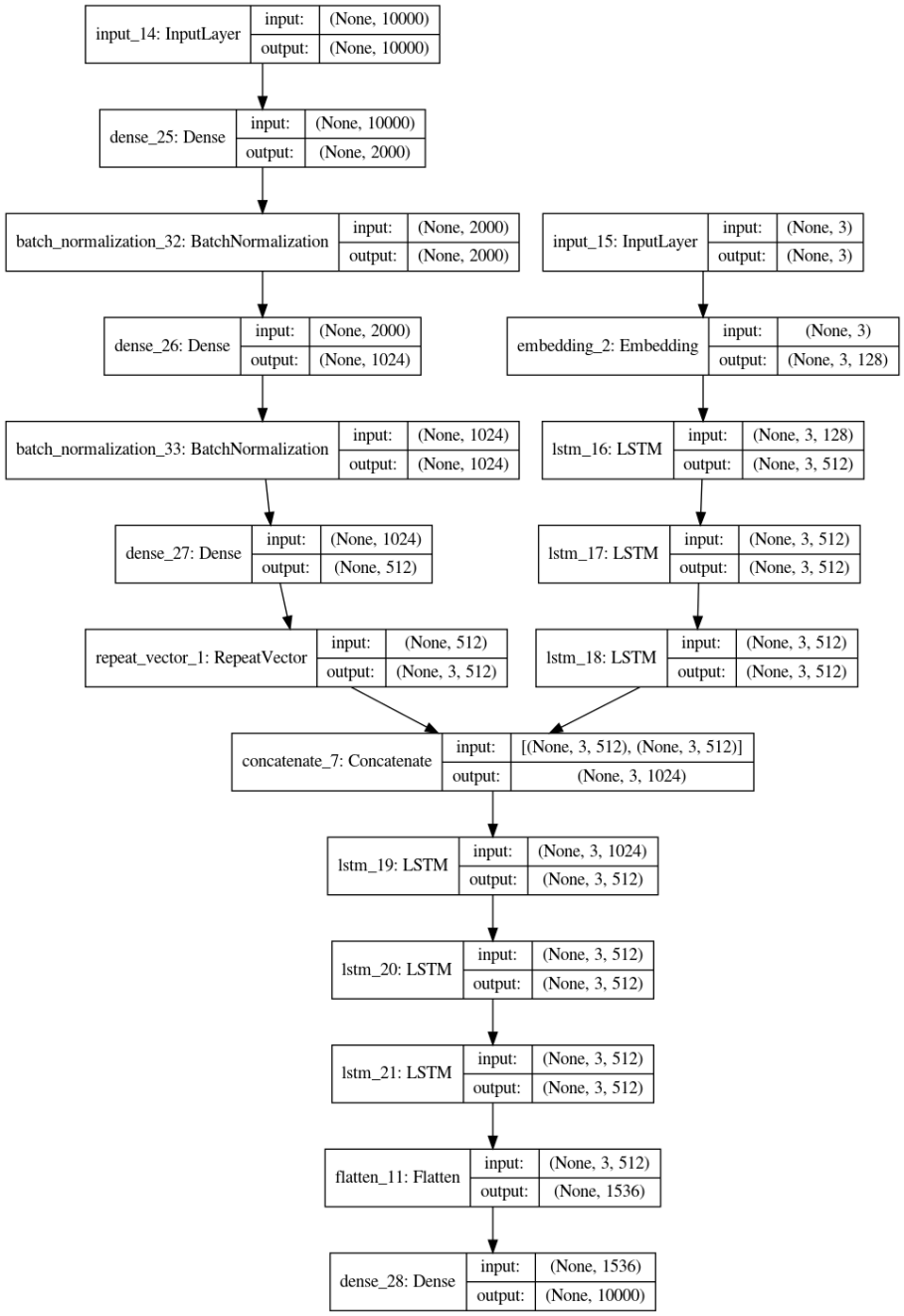

Figura 5.11: Estructura del octavo modelo.

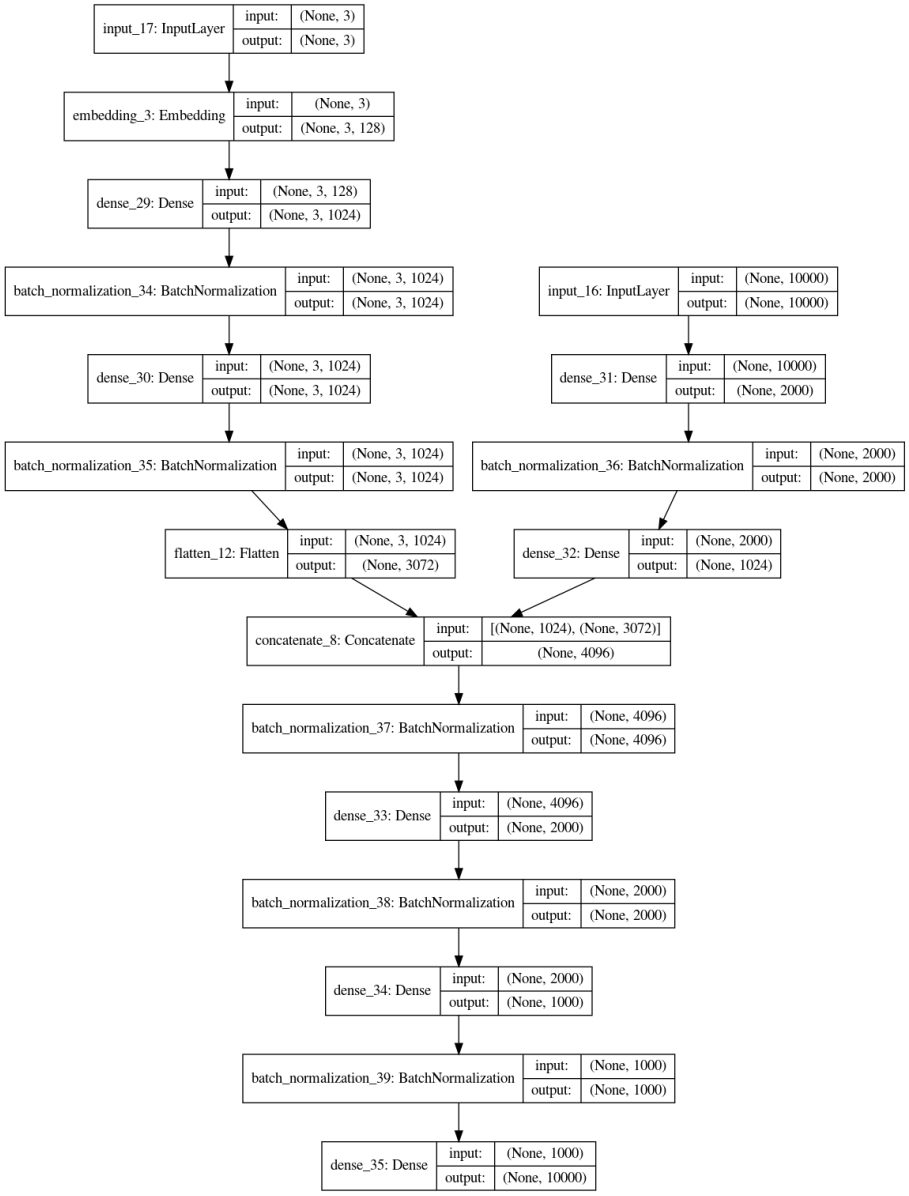

Figura 5.12: Estructura del noveno modelo.

Estos modelos si que nos han reportado unos buenos valores en el entrenamiento, por lo que los usaremos para generar los resultados con las métricas comentadas anteriormente. Los valores de *accuracy* y *loss* de estos modelos se pueden apreciar en las figuras 5.13, 5.14, 5.15 y 5.16.

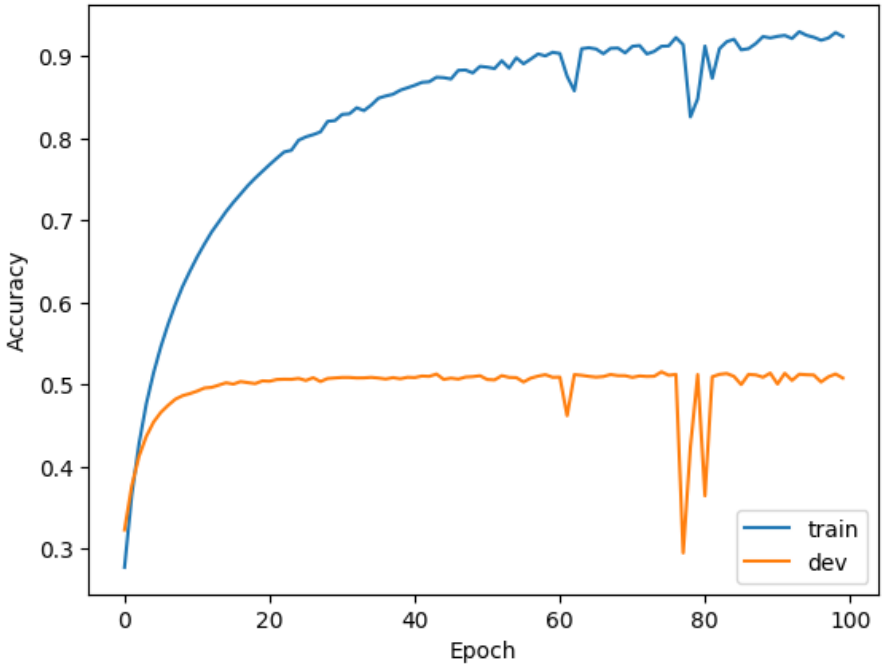

Figura 5.13: Gráfica representando la evolución del *accuracy* del octavo modelo.

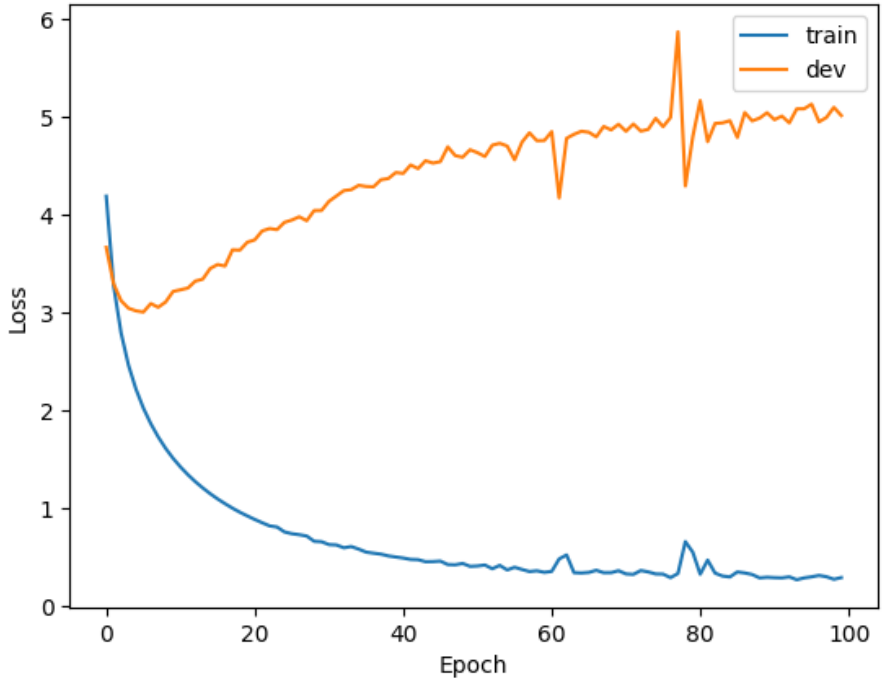

Figura 5.14: Gráfica representando la evolución del *loss* del octavo modelo.

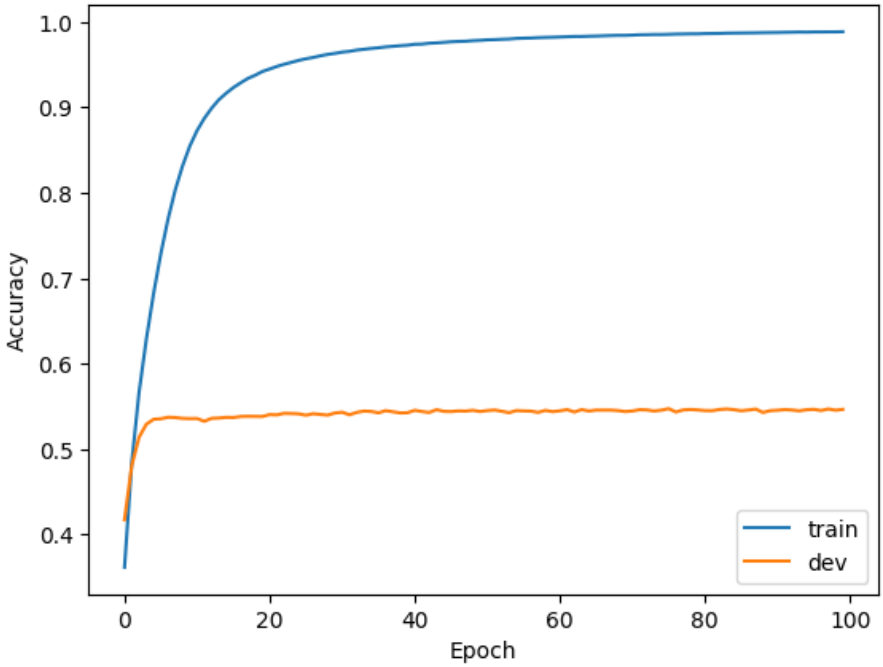

Figura 5.15: Gráfica representando la evolución del *accuracy* del noveno modelo.

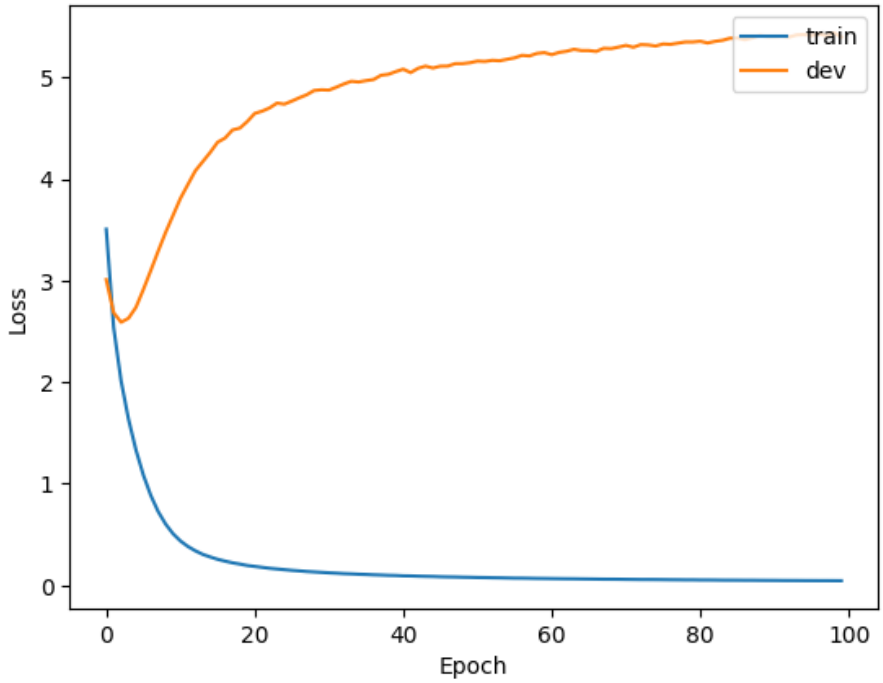

Figura 5.16: Gráfica representando la evolución del *loss* del noveno modelo.

5.2 Resultados de los modelos

Una vez entrenados los modelos 8 y 9, decidimos que los valores de *accuracy* y *loss* fueron suficientemente buenos como para empezar a generar resumenes. En nuestro caso, hemos usado el algoritmo de Viterbi explicado en 4.6.1 con nuestros modelos para gene-

rar los resúmenes. Como los resultados de ambos modelos son diferentes, presentamos diferentes posibles casos que se han dado en los siguientes ejemplos:

1. **Ambos modelos aciertan el resumen**
   - **Cuerpo de la valoración:** i have been a loyal user of the cool citrus flavor for years , but recently tried the fruit punch in the new formula . the old formula cool citrus always had a bit of a soapy taste , but the flavoring was good , much better than gross accelerade . i do not notice any soapiness with the new formula fruit punch , nor do i find it to have any kind of off taste , which i normally notice with artificial sweeteners . i have always been a fan of cytomax due to its lack of sugar in favor or more complex carbohydrates , plus the easy mixing . many people swear by accelerade , but the foul taste , pour mixing , and crazy amount of foaming have always turned me off . looks like i will stay a cytomax user with the new formula .
   - **Resumen original:** still tastes fine
   - **Resumen generado por el modelo 8:** still tastes fine
   - **Resumen generado por el modelo 9:** still tastes fine
2. **Ambos modelos fallan el resumen**
   - **Cuerpo de la valoración:** i ordered this by mistake , as i have no grinder . however , i was still able to use it all up by beating the beans in a thick envelope with a rubber mallet each morning , good thing i do not have any neighbors ! the coffee was delicious .
   - **Resumen original:** whole bean coffee
   - **Resumen generado por el modelo 8:** good
   - **Resumen generado por el modelo 9:** no complaints
3. **Modelo 8 acierta, pero modelo 9 falla**
   - **Cuerpo de la valoración:** solixir is my new favorite beverage ! each can of solixir is packed with >1,400mg of botanical extracts designed to serve a functional purpose . the light botanical blend is the perfect balance between juice and sparkling water . there is no added sugar . it is all natural . the orange mate , also called awaken , is the ultimate beverage for maintaining good hydration and a healthy balance without consuming added sugar and artificial ingredients . it is not a traditional energy drink . it has less caffeine than a cup of decaf coffee . the natural caffeine is from the botanical extracts . no jittery feelings and post buzz crash as with the traditional energy drink . just smooth , steady natural energy . also check out the blackberry chamomile relax and the pomegranate ginger awaken for a well rounded selection of functional , healthy beverages .
   - **Resumen original:** crisp , cool , and totally hydrating
   - **Resumen generado por el modelo 8:** crisp , cool , and totally hydrating
   - **Resumen generado por el modelo 9:** affordable , natural tasting drinking tea to go

4. **Modelo 9 acierta, pero modelo 8 falla**
   - **Cuerpo de la valoración:** before trying this tea i would been drinking the packets my husband brought home from japan . the stash had the same nutty flavor of the good japanese green tea but the bags were fragile and leaked tea leaf into the cup . just a personal pet peeve . i bought the stash green tea based on a review i read in men s health magazine that rated stash as having the highest level of anti oxidents and caffeine in those tested .
   - **Resumen original:** very tasty but bags leak the very finely crushed leaves
   - **Resumen generado por el modelo 8:** very tasty but bags leak the good
   - **Resumen generado por el modelo 9:** very tasty but bags leak the very finely crushed leaves

Como podemos observar en los ejemplos expuestos, existen casos donde un modelo es más exacto que otro. Además, a la hora de comprobar si un resumen acierta o falla, solo estamos contando si es exactamente el mismo resumen el que genera el modelo y el que tenemos de referencia. Pero esta medida no es suficiente para medir la eficacia de los modelos. Por ello, hemos decidido usar la métrica de *ROUGE* para poder medir la eficacia de los modelos formalmente. Las tablas 5.1, 5.2 y 5.3 muestran los resultados sobre los datos que se usaron para entrenar los modelos. Las tablas 5.4, 5.5 y 5.6 muestran los resultados sobre los datos de test. Como se puede observar en estas tablas, los resultados sobre el set de entrenamiento son mucho mejores que los de test. Por tanto, podemos concluir que el sistema sobreentrena.

Tabla 5.1: Resultados set de entrenamiento con *ROUGE-1*

ROUGE-1	Precisión	Recall	F1-score
modelo 8	83.16	81.79	81.98
modelo 9	89.18	88.52	88.70

Tabla 5.2: Resultados set de entrenamiento con *ROUGE-2*

ROUGE-2	Precisión	Recall	F1-score
modelo 8	70.27	69.22	69.35
modelo 9	78.12	77.56	77.74

Tabla 5.3: Resultados set de entrenamiento con *ROUGE-L*

ROUGE-L	Precisión	Recall	F1-score
modelo 8	83.09	81.73	81.92
modelo 9	89.14	88.49	88.67

Tabla 5.4: Resultados set de test con *ROUGE-1*

ROUGE-1	Precisión	Recall	F1-score
modelo 8	38.06	36.85	36.85
modelo 9	40.13	39.03	38.95

Como se observa en estas tablas, el modelo 9 obtiene una mejora en el rendimiento respecto del modelo 8. Aunque hemos visto casos anteriores en los cuales el modelo 8 puede ser mas preciso, el modelo 9 es capaz de dar mejores resultados mas probablemente.

Tabla 5.5: Resultados set de test con *ROUGE-2*

ROUGE-2	Precisión	Recall	F1-score
modelo 8	29.03	28.54	28.59
modelo 9	31.01	30.63	30.67

Tabla 5.6: Resultados set de test con *ROUGE-L*

ROUGE-L	Precisión	Recall	F1-score
modelo 8	37.96	36.78	36.77
modelo 9	39.97	38.89	38.81

5.3 Comparativa con el estado del arte

A la hora de comparar nuestros resultados, hemos encontrado un artículo que posee resultados de *abstract summarization* sobre nuestro set de datos 3.6.1. Los datos de comparación con nuestros modelos se pueden ver en la tabla 5.7

Tabla 5.7: Comparación con *State-of-the-art*

Model	Hyper-parameters and setting	Rouge-F1
modelo 8	default	36.85
modelo 9	default	38.95
Baseline	default	17.87
Transformer	2 layer 128- dropout 0.2 - no truncation - no stopwords	12.95
Transformer	2 layer 128- dropout 0.4 - no truncation - no stopwords	13.91
Transformer	2 layer 128- dropout 0.4 - truncated to 20 - no stopwords	15.34
Transformer	2 layer 128- dropout 0.4 - truncated to 40 - no stopwords	14.76
Transformer	2 layer 128- dropout 0.4 - no truncation - with stopwords	13.21

Como podemos observar en la comparativa con el artículo [17], nuestros resultados son superiores a los resultados actuales sobre el set de datos de 3.6.1.

5.4 Resultados set de datos de News

Después de toda nuestra experimentación llevada a cabo, hemos conseguido dos modelos funcionales para generar resúmenes a partir de un texto. Una vez comprobada su eficacia comparándolos con un artículo que usa el mismo conjunto de datos, vamos a comentar su rendimiento con el conjunto de datos de *News*.

En primer lugar, seguimos los mismos pasos que con el conjunto de *Amazon Fine Food Reviews* y entrenamos con pesos diferentes los modelos 8 y 9, por ofrecer los mejores resultados. Los modelos no han tenido ninguna modificación por lo que la estructura sigue siendo la de las figuras 5.11 y 5.12. En cuanto a los resultados del entrenamiento de estos modelos con el nuevo set de datos, se pueden ver en las figuras 5.17, 5.18, 5.19 y 5.20.

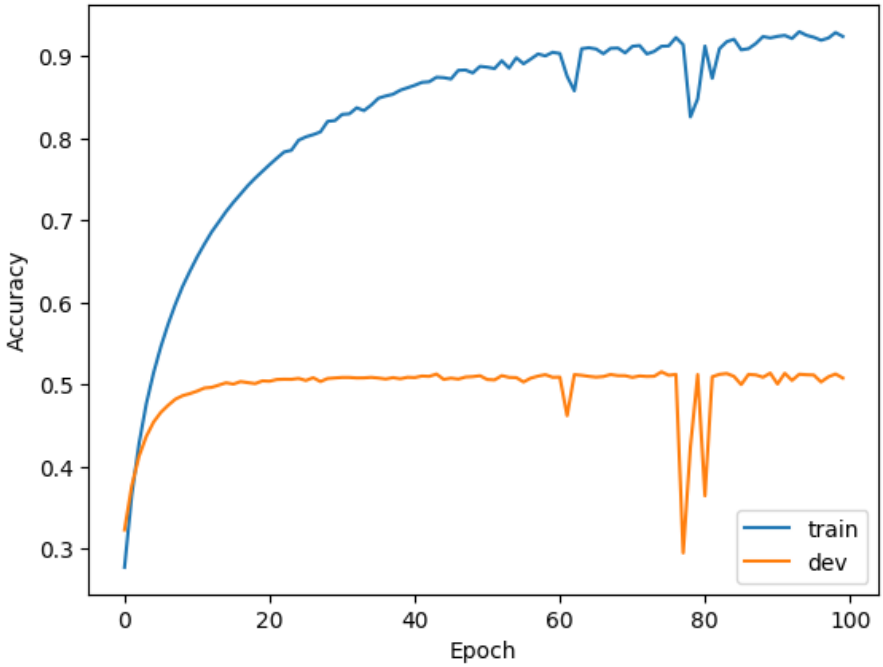

Figura 5.17: Gráfica representando la evolución del *accuracy* del octavo modelo con el conjunto de datos *News*.

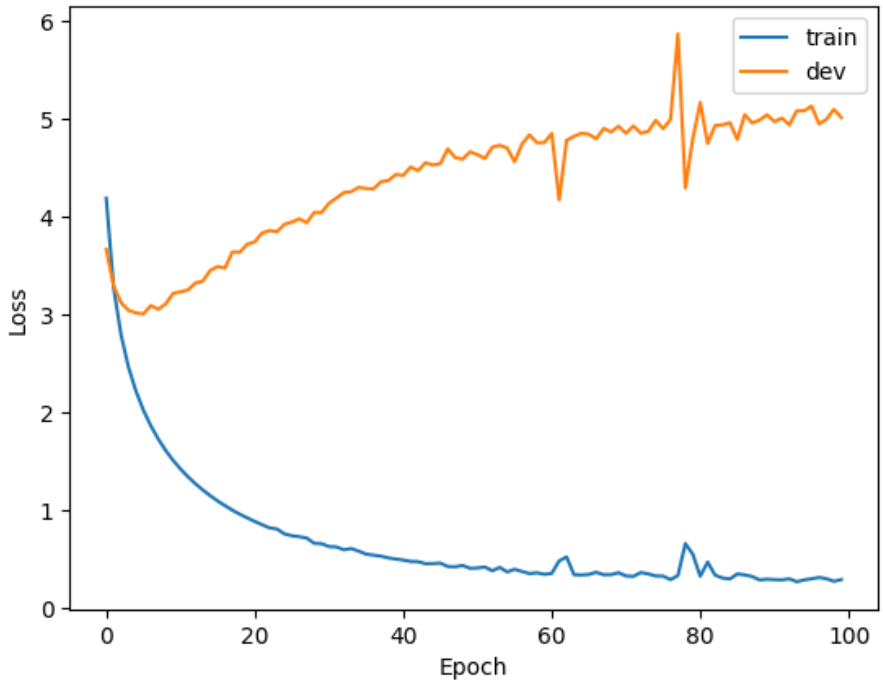

Figura 5.18: Gráfica representando la evolución del *loss* del octavo modelo con el conjunto de datos *News*.

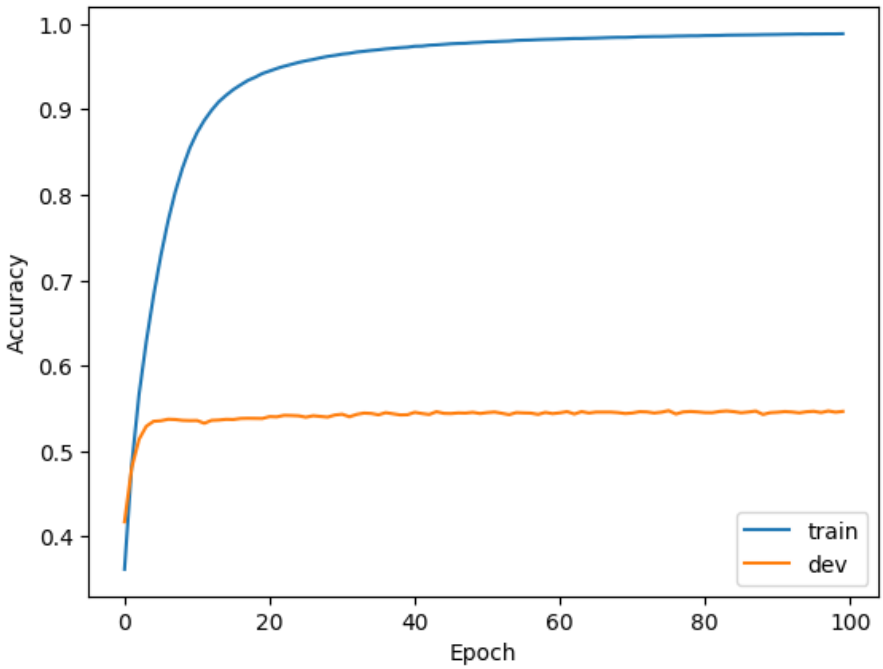

Figura 5.19: Gráfica representando la evolución del *accuracy* del noveno modelo con el conjunto de datos *News*.

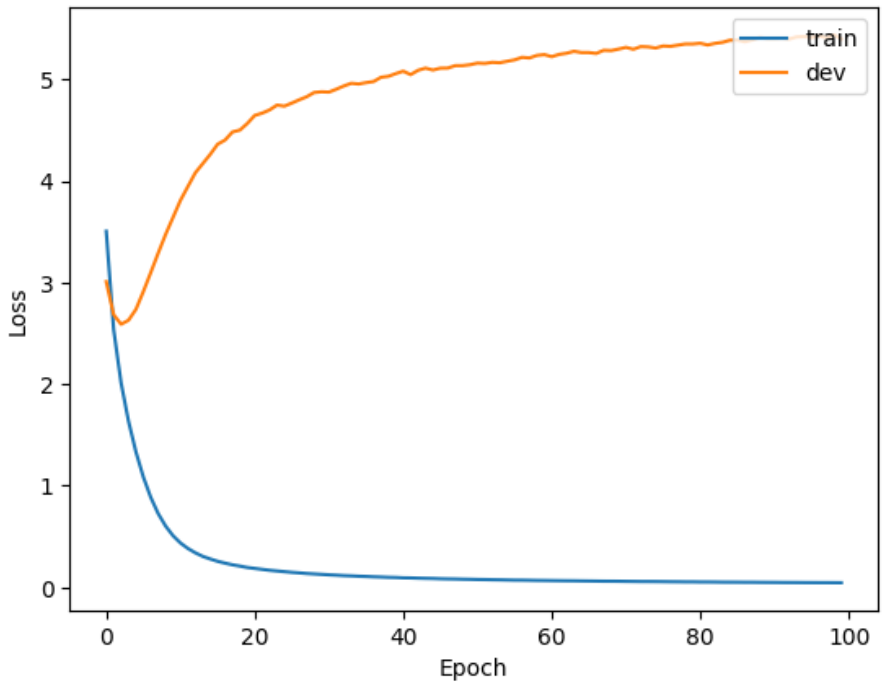

Figura 5.20: Gráfica representando la evolución del *loss* del noveno modelo con el conjunto de datos *News*.

Los resultados del entrenamiento no fueron tan buenos como con el set de datos anterior, pero son suficientemente buenos como para probar su eficacia real.Una vez los nuevos modelos estaban entrenados, nos preparamos a hacer las pruebas y a calcular su puntuación según la métrica de ROUGE. Los resultados de estas pruebas se pueden apreciar en las tablas 5.8, 5.9, 5.10, 5.11, 5.12 y 5.13.

Tabla 5.8: Resultados set de entrenamiento con *ROUGE-1*, con el set de datos de *News*.

ROUGE-1	Precisión	Recall	F1-score
modelo 8	00.03	00.03	00.03
modelo 9	93.06	92.41	92.56

Tabla 5.9: Resultados set de entrenamiento con *ROUGE-2*, con el set de datos de *News*.

ROUGE-1	Precisión	Recall	F1-score
modelo 8	00.00	00.00	00.00
modelo 9	88.35	87.71	87.91

Tabla 5.10: Resultados set de entrenamiento con *ROUGE-l*, con el set de datos de *News*.

ROUGE-1	Precisión	Recall	F1-score
modelo 8	00.03	00.03	00.03
modelo 9	93.06	92.41	92.56

Tabla 5.11: Resultados set de test con *ROUGE-1*, con el set de datos de *News*.

ROUGE-1	Precisión	Recall	F1-score
modelo 8	00.55	00.42	00.40
modelo 9	37.53	33.17	33.53

Tabla 5.12: Resultados set de test con *ROUGE-2*, con el set de datos de *News*.

ROUGE-1	Precisión	Recall	F1-score
modelo 8	00.00	00.00	00.00
modelo 9	22.64	22.32	21.98

Tabla 5.13: Resultados set de test con *ROUGE-l*, con el set de datos de *News*.

ROUGE-1	Precisión	Recall	F1-score
modelo 8	00.54	00.39	00.39
modelo 9	33.62	30.12	30.33

El modelo 8 ha tenido un mal desarrollo con el conjunto de datos de *news*. Sospechamos que podría ser a causa de un error, pero no hemos conseguido encontrarlo. El modelo 9 ha dado unos resultados que podrían verificar su utilidad para resumir textos más extensos, generando, a su vez, resúmenes más extensos. Debido a la falta de modelos con los que compararse por ser un conjunto de datos que hemos conseguido nosotros mismos, no podemos asumir su eficacia real. Aún así, compararemos la eficacia obtenida por el noveno modelo para ambos conjuntos de datos. En este caso, usaremos solo *ROUGE-1*, debido a que es la única métrica de *ROUGE* que hemos podido verificar con otro artículo. Esta comparación se puede apreciar en la tabla 5.14.

Tabla 5.14: Comparativa de la eficacia del noveno modelo para los dos conjuntos de datos en test.

ROUGE-1	Precisión	Recall	F1-score
modelo 9 *reviews*	40.13	39.03	38.95
modelo 9 *news*	37.53	33.17	33.53

El modelo 9 funciona mejor en el conjunto de datos de *Amazon Fine Food Reviews*. Esto se podría deber a varias razones, como que ese conjunto de datos posee muchos más ejemplos para entrenar o que funciona mejor en resúmenes y textos cortos que en algunos más largos. En cualquier caso, los resultados obtenidos con la métrica *ROUGE* para ambos conjuntos de datos (Reviews i news) no distan mucho, considerando que en el caso de news, tanto los textos como los resúmenes son más largos, podemos decir que la diferencia entre los valores de *F1-score*, 38.95 % para Reviews frente a 33.53 % para news, no es relevante y, por tanto, aún sin poder comparar con otros trabajos, los resultados obtenidos con el conjunto de datos news son, a priori, satisfactorios.

CAPÍTULO 6
Conclusiones

Después de toda la experimentación llevada a cabo, hemos llegado a las siguientes conclusiones:

- Nuestra representación para el texto basada en Voxels 4.3.2 no ha sido útil a la hora de representar el texto. Esto se debe a que los modelos que han usado este tipo de representación no han conseguido un rendimiento aceptable.
- El algoritmo de Viterbi, adaptado para la generación de resúmenes, ha resultado ser de mucha utilidad, otorgando unos altos resultados.
- La mejor combinación que hemos encontrado para los modelos consiste en una representación de las palabras basada en vectores preentrenados de word2vec, una representación del texto basada en *bag-of-words* y una salida activada por una función *Softmax*, dentro de un vocabulario preseleccionado.
- El modelo 9, a pesar de no utilizar redes recurrentes, ha demostrado obtener mejores resultados, es lo más parecido a la aproximación transformer. En nuestro caso, ha dado buenos resultados en los dos conjuntos de datos que hemos probado.

6.1 Objetivos logrados

A lo largo de este trabajo, hemos conseguido lograr nuestro objetivo de generar un resumen a partir de un texto, como se puede ver en el apartado 5.2. Aunque el sistema no es perfecto, la precisión y la capacidad de acierto que nos ofrece es capaz de competir con los modelos publicados en el *state-of-the-art* para el mismo dataset.

Bibliografía

[1] J. E. Bresenham. "Algorithm for computer control of a digital plotter". En: *IBM Systems Journal* 4.1 (1965), págs. 25-30. ISSN: 0018-8670. DOI: 10.1147/sj.41.0025.

[2] Courtney Napoles, Matthew Gormley y Benjamin Van Durme. "Annotated gigaword". En: *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*. Association for Computational Linguistics. 2012, págs. 95-100.

[3] Julian J. McAuley y Jure Leskovec. "From Amateurs to Connoisseurs: Modeling the Evolution of User Expertise through Online Reviews". En: *CoRR* abs/1303.4402 (2013). arXiv: 1303.4402. URL: http://arxiv.org/abs/1303.4402.

[4] Sébastien Jean y col. "On using very large target vocabulary for neural machine translation". En: *arXiv preprint arXiv:1412.2007* (2014).

[5] Jeffrey Pennington, Richard Socher y Christopher D. Manning. "Glove: Global vectors for word representation". En: *In EMNLP*. 2014.

[6] François Chollet. *keras*. https://github.com/fchollet/keras. 2015.

[7] Martín Abadi y col. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[8] Christopher Olah. "Understanding lstm networks". En: (2015). URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[9] Alexander M. Rush, Sumit Chopra y Jason Weston. "A Neural Attention Model for Abstractive Sentence Summarization". En: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, sep. de 2015, págs. 379-389. DOI: 10.18653/v1/D15-1044. URL: https://www.aclweb.org/anthology/D15-1044.

[10] Ramesh Nallapati, Bing Xiang y Bowen Zhou. "Sequence-to-Sequence RNNs for Text Summarization". En: *CoRR* abs/1602.06023 (2016). arXiv: 1602.06023. URL: http://arxiv.org/abs/1602.06023.

[11] Stanford Network Analysis Proyect. *Amazon Fine Food Reviews*. https://www.kaggle.com/snap/amazon-fine-food-reviews. 2016.

[12] Jonas Gehring y col. "Convolutional Sequence to Sequence Learning". En: *CoRR* abs/1705.03122 (2017). arXiv: 1705.03122. URL: http://arxiv.org/abs/1705.03122.

[13] Diego Antognini. *py-rouge*. https://github.com/Diego999/py-rouge. 2018.

[14] Ashish Vaswani y col. "Tensor2Tensor for Neural Machine Translation". En: *CoRR* abs/1803.07416 (2018). arXiv: 1803.07416. URL: http://arxiv.org/abs/1803.07416.

[15] *Anaconda Software Distribution*. Ver. 2-24.0 Anaconda. URL: https://anaconda.com.

[16] *News API*. https://newsapi.org/.

[17] Nima Sanjabi. *Abstractive Text Summarization with Attention-based Mechanism*. URL: https://upcommons.upc.edu/bitstream/handle/2117/119051/131670.pdf.

[18] *Wikipedia/BLEU*. URL: https://en.wikipedia.org/wiki/BLEU.