Text Summarization

José Jorge Rodríguez Salgado C-511
Alberto González Rosales C-511
David Castillo López C-511

Curso 2019-2020

Resumen

El resumen automático de textos se plantea como una tarea útil en los tiempos que transcurren. Poder procesar grandes cantidades de información en poco tiempo y ser capaces de reducirla a volúmenes más manejables sin que pierda su significado es objeto del presente trabajo. En este se propondrá un modelo de solución a este problema mediante el uso de redes neuronales y se hará un esbozo de toda la línea de pensamiento seguida para la resolución del problema planteado.

1. Resumen Automatizado de Textos

El resumen automatizado de textos es la tarea de producir un resumen conciso y fluido que mantenga la información clave y el significado general del texto original.

Existen dos formas de abordar este problema:

- Extractive Summarization
- Abstractive Summarization

1.1. Extractive Summarization

Como su nombre indica se basa en la extracción de partes del texto. Se identifican oraciones, frases o palabras importantes y se extraen solamente esas para conformar el resumen. **El resumen consiste solamente de las partes extraídas del texto**.

1.2. Abstractive Summarization

Este enfoque es más interesante y es el que se estará utilizando para resolver el problema que nos ocupa. La idea principal es generar nuevas oraciones a partir del texto original. Por tanto las oraciones que conformarán el resumen pueden estar presentes o no en el texto original.

2. Problema

Dado una reseña en inglés sobre un producto alimentario, devolver un resumen de la misma, una secuencia corta de palabras que contenga la esencia de esta. Por ejemplo si la reseña fuera "great chai tea aromatic flavorful overpowering strong like", un posible resumen puede ser "best tea ever".

3. Modelo Sequence to Sequence(*Seq2Seq*)

El modelo *Sequence to Sequence* se puede utilizar en problemas que involucren información secuencial. Ejemplos de estos son *Clasificación de Sentimientos, Traducción y Reconocimiento de Entidades Nombradas*. En el caso de la traducción la entrada es un texto en un idioma y la salida es el mismo texto en otro idioma. En el problema de las **entidades nombradas** la entrada es una secuencia de palabras y la salida una secuencia de etiquetas asociadas a las palabras de entrada.

El problema que se intenta resolver tiene como entrada una gran secuencia de palabras(el texto original) y su salida es una secuencia(más corta) que resuma el contenido del texto de entrada. Por tanto, cae dentro de la clasificación de problema con información secuencial y se puede aplicar el modelo *Seq2Seq*.

Existen dos componente principales en el modelo *Seq2Seq*:

- **Encoder**
- **Decoder**

La arquitectura *Encoder-Decoder* es mayormente utilizada para resolver problemas de tipo *Seq2Seq* cuando las longitudes de la entrada y la salida difieren. Existen varias variantes de *Redes Neuronales Recurrentes(RNN)*; para las componentes *Encoder* y *Decoder* las preferibles son *Gated Recurrent Neural Network(GRU)* o las *Long Short Term Memory(LSTM)*. Estas últimas serán las utilizadas para resolver el problema.

3.1. Encoder

Un *Encoder LSTM* lee la secuencia completa de entrada, recibiendo una palabra a la vez. Procesa la información recibida hasta el momento y captura la información contextual presente en la secuencia de entrada. Después de cada palabra recibida y procesada la información mencionada previamente queda en unos estados h_i y c_i a los cuales llamaremos *hidden state* y *cell state*, respectivamente. Los estados que se produzcan después de analizar la última palabra se utilizan para inicializar el *Decoder*.

3.2. Decoder

El *Decoder* es una red neuronal *LSTM* que se entrena para predecir la siguiente palabra en función de la palabra anterior. Recibe como entrada la salida del *Encoder* y un *token* especial denominado *Start*, que indica la primera palabra del *output* para así poder predecir la siguiente hasta que se produzca el *token End* o se llegue a la longitud máxima establecida.

4. Limitaciones de la Arquitectura *Encoder-Decoder*

El *Encoder* convierte la secuencia de entrada en un vector de longitud fija y luego el *Decoder* predice la secuencia de salida. Esto funciona mejor cuando la secuencia de entrada es corta ya que el *Decoder* tiene que mirar la entrada completa para hacer la predicción. La dificultad para el *Encoder* radica en memorizar secuencias muy largas en vectores de tamaño fijo.

La forma de lidiar con este problema se llama *mecanismo de atención*, el cual consiste en intentar predecir una palabra teniendo en cuenta solo algunas partes de la entrada y no su totalidad.

5. Mecanismo de Atención

Como se mencionó anteriormente, en modelos tradicionales de *Sequence to Sequence* se descartan todos los vectores de estado intermedios del *Encoder* y solo se utiliza el último vector de estado para inicializar el *Decoder*. Esto funciona bien para textos pequeños, pero para textos grandes tener solo en cuenta el último vector como salida del *Encoder* se convierte en una limitante. Esta observación se ha comprobado en la práctica al ver como el rendimiento de estos modelos disminuye drásticamente a medida que el tamaño del texto de *input* aumenta. Este problema se intenta resolver con algo conocido como *mecanismo de atención*, que sí tiene en cuenta los estados intermedios del

Encoder construyendo un vector de contexto que va a utilizar el *Decoder* en cada paso para generar su salida.

La *capa de atención* va a hacer que la salida del *Decoder* en cada paso sea dependiente de la entrada (que es el token *start* o la salida anterior del *Decoder*), del estado interno de la *LSTM* (que inicialmente es la salida del *Encoder*) y del vector de contexto que es un vector igual a la suma de todos los vectores que dió como salida el *Encoder*, multiplicados cada uno por un peso que regula que tan importante es dicho vector para la predicción del paso actual del *Decoder*. La capa de atención es una red neuronal tradicional que, utilizando propagación hacia atrás y descenso de gradiente, va a ser entrenada para dar valores correctos a los pesos que multiplican a cada uno de los vectores de estado que generó el *Encoder*.

Ahora cada vez que se vaya a evaluar el *Decoder* se va a crear el vector de contexto, que es la combinación lineal de los vectores $h_1, h_2, h_3, \ldots, h_n$, donde h_i es la salida del *Encoder* en el paso i, multiplicados por los escalares $e_1, e_2, e_3, \ldots, e_n$, donde e_i es el peso que la red neuronal de la capa de atención ya entrenada predice le corresponde a h_i. Tener en cuenta que a $e_1, e_2, e_3, \ldots, e_n$ se les aplica la función *softmax* lo cual hace que $0 \leq e_i \leq 1$ para todo i, y que $e_1+e_2+e_3+\cdots+e_n = 1$, dándole un concepto probabilístico a e_i. Luego el vector de contexto sería igual a $h_1 * e_1 + h_2 * e_2 + \cdots + h_n * e_n$. Ahora el vector de contexto se concatena al *input* que recibe el *Decoder*, permitiendo que este utilice toda la información que generó el *Encoder*, de acuerdo a un concepto de atención que se le debe prestar a cada uno, de una forma elegante y eficiente.

6. Preprocesamiento

La realización de un preprocesamiento es muy importante antes de pasar a la construcción del modelo. El uso de datos sin procesar puede resultar en un desastre. En esta fase dejaremos fuera todos los símbolos no deseados que no afectan el objetivo de nuestro problema.

Vamos a definir dos partes de preprocesado, una para las entradas y otra para los resúmenes, ya que difieren ligeramente.

6.1. Preprocesamiento del Texto

- Convertir todos los caracteres a minúsculas.
- Realizar el mapeo de las contracciones.
- Eliminar los apóstrofos.

- Eliminar todo texto entre paréntesis.
- Eliminar signos de puntuación y caracteres especiales.
- Eliminar *stopwords*
- Eliminar palabras cortas.

6.2. Eliminación de Datos de Entrada

Mediante una inspección a los datos de entrada se observó que la longitud de la mayoría de los textos era no mayor de 30 palabras y la de los resúmenes rondaba las 8 palabras. Razón por lo cual, para tener un conjunto de datos más uniforme con los que trabajar, se decidió desechar aquellos textos cuya longitud fuese mayor que 30 o cuyo resumen tuviese una longitud mayor de 8 palabras.

6.3. Eliminación de palabras poco frecuentes

Se decidió que una palabra aportaba poca información en un texto si no era usada frecuentemente. Como umbral se escogió el valor 4, así, toda palabra que ocurra menos de 4 veces en los textos es desechada. En nuestro conjunto de datos la cantidad de palabras poco frecuentes estaba cerca del 65 porciento del total. Para los resúmenes el valor escogido fue 6, lo que representaba un 78 porciento del total.

7. Modelo de Entrenamiento

Primeramente tenemos la capa de *Input* del *Encoder* llamada *encoder-inputs*, luego una capa de *embedding* llamada *enc-emb* conectada a *encoder-inputs*. Después de la capa de *embedding* siguen tres capas *LSTM*, cada una produce como *output* su respectivo vector de salida, su *hidden state* y su *cell state*. La primera de estas capas está conectada a la capa de *embedding* y las otras dos a la capa *LSTM* anterior.

Luego tenemos la capa de *Input* del *Decoder*, una capa de *embedding* conectada a esta y una capa *LSTM* conectada a esta última y a la salida que produce el *Encoder*. Luego viene la capa de atención, la cual está conectada a la salida del *Encoder* y del *Decoder*. Seguida a esta viene una capa de concatenación unida a la salida del *Decoder* y de la *capa de atención*. Por último, una capa densa conectada a la capa de concatenación, produciendo la salida del *Decoder*.

Como optimizador se utilizó *rmsprop* y como función de pérdida la "sparse-categorical-crossentropy". Se prefirió esta a la çategorical-crossentropy"debido a su mejor uso de la memoria, esta última utilizaba vectores *one-hot*, los cuales tienen dimensión igual a la longitud del vocabulario.

El modelo se entrenó en 50 *epochs*, con una condición de *early stopping* que detenía el entrenamiento si el valor de pérdida aumentaba dos veces seguidas.

8. Predicción

Nuestra entrada(la secuencia que corresponde al texto original) se pasa al *Encoder* y este produce como salida los vectores *e-out*, *e-H* y *e-C*. Tendremos en todo momento una palabra objetivo que llamaremos *target-seq*; al principio esta palabra es el *token* especial que indica el inicio del resumen y en cada paso excepto el primero tendrá la palabra generada en el paso anterior.

Mientras no se cumpla la condición de parada de la predicción pasamos como entrada al *Decoder* la *target-seq* y los vectores producidos por el *Encoder* y este nos devuelve los vectores *output-tokens*, H y C. Se escoge de *output-tokens* el que mayor probabilidad tiene de ser el siguiente en la secuencia. Si este es el *token* especial de fin del resumen o si el resumen alcanzó su longitud máxima establecida, entonces se detiene el proceso de generación del resumen. Sino, se concatena este últimos *token* a nuestra respuesta parcial y se actualizan la *target-seq* y los vectores *e-H* y *e-C*, los cuales se convierten en H y C, respectivamente.

9. Resultados

Se seleccionó una muestra de 500 reseñas para que constituyeran el conjunto de prueba. El modelo evaluado en este conjunto tuvo una pérdida de 1,9879. Este valor, a priori, parece bueno; sin embargo, se consideró que este problema específico está mal planteado y, por tanto, no está bien definido lo que significa que un resumen sea mejor que otro. Esto dificulta un poco el proceso de evaluación y sería una buena tarea plantearse de que forma analizar superioridad entre un resumen generado y otro, de forma automática.

Si bien es cierto que al ojo humano le es fácil determinar la calidad del resumen, diseñar una función comparativa parece ser una tarea complicada donde, quizás, también haya que hacer uso de redes neuronales que nos

ayuden en esta tarea.

10. Trabajos Relacionados

Debido a su éxito reciente, los modelos de redes neuronales han sido utilizados con resultados muy positivos en *abstractive summarization*. Este problema fue abordado primeramente con un modelo de atención neuronal, logrando resultados relevantes en múltiples *datasets*. Este modelo ha sido extendido usando una red neuronal recurrente(RNN) como decodificador. La mejora inmediata fue utilizar un *framework* donde tanto el codificador como el decodificador eran RNN, también empezaron a tener más relevancia las características léxicas y estadísticas. Uno de los mayores rendimientos en la actualidad se alcanza mediante la codificación selectiva de palabras como un proceso de destilación de información relevante.

Los modelos neuronales de *abstractive summarization* han sido utilizados para resumir cada vez textos más largos. Primero se utilizaron los modelos de extracción de palabras, los cuales mostraron peores resultados que los modelos de extracción de oraciones. Redes neuronales jerárquicas basadas en atención también han sido aplicadas en este problema. Finalmente los modelos basados en distracción se propusieron para permitir la opción de transitar por el contenido del texto y obtener una idea general de su significado. Los mejores resultados se han alcanzado utilizando redes neuronales *atencionales*, considerando los factores claves en resúmenes de texto como son fluidez, novedad y prominencia.

Los resúmenes que utilizan extracción tratan de identificar las oraciones más relevantes del texto y después concatenarlas. En varios trabajos recientes ((Cheng and Lapata, 2016; Nallapati et al., 2017; Narayan et al., 2017; Yasunaga et al., 2017) enfocan el problema como etiquetar cada oración en relevante o no para agregarla al resumen respuesta. Los modelos existentes utilizan redes neuronales recurrentes para llegar a una representación general del documento que después es utilizada para etiquetar cada oración en relevante o no, utilizando las oraciones etiquetadas anteriormente.