Facultad de Matemática, Atronomía, Física y Computación

Universidad Nacional de Córdoba

Estudio de Simplificación de Oraciones con Modelos *Actor-Critic*

Tesis

para obtener el título de

Licenciado en Ciencias de la Computación

Autor

Mauricio Diego Mazuecos Perez

Directora: Milagro Teruel

Córdoba, Argentina 2019

DEDICATORIA

Dedicado a mis padres Diego y María Eugenia por su apoyo y cariño durante todos estos años.

Agradecimientos

A mi directora de tesis Milagro por la paciencia y las enseñanzas que me dio en el desarrollo de este trabajo.

Clasificación (ACM CCS 2012):

- Computing methodologies Natural language processing
- Computing methodologies Reinforcement learning

Palabras Clave:

- Reinforcement Learning
- Sentence Simplification
- Deep Learning
- Artificial Intelligence
- Natural Language Processing

Resumen

La simplificación de oraciones es una tarea de Procesamiento del Lenguaje Natural que se centra en transformar escritos para que su gramática, estructura y palabras sean más sencillas de comprender, sin perder la semántica de la oración original. Como tal, no es una tarea simple de abordar y requiere métodos sofisticados que permitan definir las características que hacer a una oración simple. Al mismo tiempo, estos modelos deben tener una representación adecuada del significado de la oración, que no debe ser alterado durante el proceso de simplificación.

En este trabajo de tesis se explora el uso de aprendizaje por refuerzos para la tarea de simplificación de oraciones. Se utiliza un entrenamiento en dos etapas, construyendo primero un sistema de traducción automática clásico que luego es ajustado durante un segundo entrenamiento con un algoritmo *actor-critic*. Se muestran resultados de la primera etapa de entrenamiento y su comparación con trabajo previo y se hace una exposición de las dificultades y problemas de la segunda etapa de entrenamiento.

Overview

Sentence Simplification is a Natural Language Processing task which focuses on transforming writings so its grammar, structure and words are easier to understand, without loosing the semantics of the original sentence. As such, its not a simple task to face and it requires sophisticated methods which allow to define the features which make a sentence simple. At the same time, these models should be provided with an accurate representation of the meaning of the sentence, which must not be changed in the simplification process.

In this thesis, the use of reinforcement learning for the sentence simplification task is explored. A two-steps training is used, first building a classic machine translation system that is later adjusted in a second step of training with an *actor-critic* algorithm. Results for the first step of training and it's comparison with previous work and an exhibition of the difficulties and problems of the second step of training are both shown in this thesis.

Índice general

Capítulo 1

Introducción y Motivación

En este trabajo se aborda el problema de simplificación de oraciones a través de una aproximación empleando métodos neuronales de traducción automática y aprendizaje por refuerzos. El modelado del problema y la implementación se inspira en el éxito del uso de aprendizaje por refuerzos para el mismo problema empleando el algoritmo REINFORCE (Zhang and Lapata 2017).

El problema de simplificación de oraciones es la tarea de tomar una oración sintácticamente compleja y transformarla en otra oración, más simple en cuanto a sintaxis y vocabulario, pero que conserve la semántica de la original. Esta tarea no sólo es un fin en sí misma, al permitir el acceso a información escrita a personas con baja comprensión lectora en un idioma particular, sino que también puede ser utilizada como parte de sistemas de compresión y procesamiento de texto automático.

Por la naturaleza del problema, es fácil ver la simplificación como una tarea de transformación secuencia a secuencia, es decir, se recibe una secuencia de palabras (oración) original y se devuelve otra secuencia de palabras simplificada. Este tipo de problema es tratado ampliamente en el campo del procesamiento del lenguaje natural, y particularmente en el área de traducción automática.

Por las similitudes de la tarea de simplificación con la de traducción, se emplea un modelo neuronal de traducción automática donde el idioma de origen y el idioma de salida es el mismo, sólo que uno utiliza estructuras más simples.

Los sistemas de traducción neuronal, es decir que utilizan técnicas del aprendizaje profundo, son altamente exitosos en esta tarea (Goodfellow, Bengio, and Courville 2016). Las redes neuronales profundas combinan muchas características que las vuel-

ven ideales para el modelado del lenguaje natural. Por un lado, un tipo particular de redes neuronales, llamadas recurrentes, permiten procesar secuencias de tamaño variable e identificar relaciones o patrones en dentro de ellas aunque los componentes se encuentren muy separados entre sí dentro de la secuencia (dependencias de largo alcance). Con las redes recurrentes se pueden construir modelos de lenguaje precisos que generan oraciones verosímiles (Karpathy 2015). Por otra parte, todos las arquitecturas profundas se caracterizan por crear representaciones internas de los datos de entrada con sucesivas *capas ocultas*. Estas representaciones son lo suficientemente poderosas como para poder capturar la semántica no sólo de las palabras, sino también de las oraciones. Utilizando ambas propiedades, los sistemas de traducción automática pueden modelar el significado de la oración de entrada y generar una oración de salida en el idioma deseado que respete la misma semántica original.

Sin embargo, la traducción automática no consigue modelar adecuadamente algunos aspectos de las oraciones más simples y tiende a replicar la oración original. Por este motivo, en este trabajo se propone emplear un esquema de entrenamiento en dos etapas. Una primera etapa de entrenamiento emplea un método de aprendizaje supervisado clásico para la tarea de traducción automática, y una segunda etapa ajusta el modelo con un esquema de aprendizaje por refuerzos.

El aprendizaje por refuerzos, a diferencia del aprendizaje supervisado comúnmente utilizado en tareas de traducción, permite al modelo aprender simplificaciones que no sean explícitas en el conjunto de datos de entrenamiento, utilizando el concepto de recompensas en lugar de funciones de pérdida tradicionales. Las recompensas, que pueden ser funciones complejas, estiman qué tan simple es la oración final y permiten al agente explorar distintas técnicas de simplificación sin la necesidad de contar con un ejemplos anotados.

Durante el resto del trabajo, se explicarán en detalle los modelos de traducción automática, los conceptos del aprendizaje por refuerzos y el algoritmo *Advantage Actor-Critic* (A2C), empleado en este trabajo. Además se aborda más profundamente el motivo detrás del uso de aprendizaje por refuerzos y la necesidad de emplear un entrenamiento en dos etapas.

Se exponen resultados de los experimentos de la primera etapa de entrenamiento sobre los mismo conjuntos de datos empleados en trabajos previos que tratan al tema de simplificación de oraciones y se lo compara con los resultados obtenidos con el modelo que aplica REINFORCE, otro algoritmo de aprendizaje por refuerzos.

Luego de eso se expone las complicaciones que se encontraron adaptando un modelo neuronal de traducción automática para aplicar el algoritmo de A2C a la tarea. Se explica cuáles fueron los fallos o cuáles son las hipótesis de las divergencias presentadas en el entrenamiento de estos modelos. Además se reflexiona sobre el trueque del costo computacional de aplicar aprendizaje por refuerzos y el esfuerzo de realizar dicha implementación por el potencial aumento en el desempeño del modelo en simplificación.

Este trabajo final está organizado de la siguiente manera: en el capítulo 2 se hablará acerca de la tarea de simplificación de oraciones y la delimitación del problema. En el capítulo 3 se hará un análisis del trabajo previo en el tema de simplificación y aplicaciones con éxito de aprendizaje por refuerzos en otras áreas de procesamiento de lenguaje natural. A continuación, en el capítulo 4 se da una explicación técnica de los métodos a usar y se explican las bases de aprendizaje por refuerzos, para terminar con una explicación del algoritmo A2C y una descripción de la arquitectura propuesta para la tarea. En el capítulo 5, se dará una explicación de los conjuntos de datos con los que se trabaja, de las métricas que se reportan y los experimentos que se llevan a cabo, junto con un reporte de los problemas vistos en la segunda etapa de entrenamiento empleando aprendizaje por refuerzos y las hipótesis de por qué estos suceden. Finalmente, en el capítulo 6, se cierra con un resumen del aporte de esta tesis y se enumeran posibilidades de trabajo a futuro sobre la misma temática.

Capítulo 2

Descripción del Problema

La simplificación de oraciones es un proceso en el cual se toma una oración original y se genera una oración más simple y más legible, preservando la semántica del contenido de la oración original. Típicamente, una oración simplificada difiere de una más compleja en que la primera incluye palabras más cortas y simples ("*use*" en lugar de "*exploit*"); construcciones sintácticas más simples; y menos modificadores ("He slept" en lugar de "He also slept").

La tarea de simplificación es un tópico popular dada su relevancia en varias aplicaciones e impacto en la sociedad. Por ejemplo, la simplificación automática puede beneficiar a la comprensión de texto y acceso a la información de personas con bajo nivel de alfabetización (Watanabe et al. 2009), personas con afasia (John Carroll 1999) y a hablantes no nativos (Siddharthan 2006). Se ha analizado la utilidad de la simplificación en el paso del pre procesamiento para mejorar el desempeño de los *parsers* (Chandrasekar et al. 1996), resúmenes (Beata Beigman Klebanov and Marcu 2004) y etiquetadores de roles semánticos (Vickrey and Koller 2008).

Un sistema de Simplificación de Oraciones toma oraciones en texto plano y, luego de procesarlas, retorna oraciones simplificadas. Ejemplos de estas oraciones, tomados del dataset Wikismall, son presentados a continuación:

Ejemplo 1.

In 1558 he married Regina Wäckinger, the daughter of a maid of honor of the Duchess; they had two sons, both of whom became composers.

es simplificada a:

In 1558 he married the daughter of a maid of honour of the Duchess; they had two sons who both became composers.

En este ejemplo podemos ver que la simplificación conserva la mayor parte de la semántica original, pero se reemplazan construcciones sintácticas complejas como "both of whom".

Ejemplo 2.

This movie was presented at the Cannes Film Festival in France, and was shown at theaters few years after.

es simplificada a:

This movie was shown at the Cannes Film Festival in France.

En este ejemplo, por otra parte, se obtiene una oración más simple, pero se omite parte de la semántica de la oración original.

Sin embargo, existen también muchos ejemplos en donde la simplificación fuera de su contexto original pierde interpretabilidad. Entre ellos:

Ejemplo 3.

The greater part of the islands are found in the southwest in the Archipelago Sea, part of the archipelago of the Åland Islands, and along the southern coast in the Gulf of Finland.

es simplificada a:

Thousands of islands are part of the Åland archipelago.

Ejemplo 4.

Former Presidents Jimmy Carter, George H. W. Bush, and Bill Clinton appeared to celebrate with Graham.

es simplificada a:

Former Presidents Jimmy Carter, George H. W. Bush, and Bill Clinton came.

Otro problema en la falta de contexto son oraciones donde los sujetos cambian de un nombre propio a uno genérico, o viceversa. Por ejemplo:

Ejemplo 5.

Naturalmente, la gente del pueblo deseaba que las vías pasaran más cerca o atraves de él.

es simplificada a:

La gente en Yass quería que las vías más cerca o incluso a traves del pueblo.

Ejemplo 6.

Warner's performance earned him a $1 million dollar bonus for the year, and he fell just short of attaining a 90.0+ passer rating, which would have given him an extra $500,000.

es simplificada a:

Because he played well, he was given a $1 million bonus for the year.

Estos ejemplos dejan en claro que el criterio de qué constituye una oración simplificada correcta no está fijo. Esto es una complicación al momento de aprender a simplificar porque distintas oraciones, si bien parecidas en sintaxis, podrían tener simplificaciones que no conserven el total de la semántica de la original. Sumado a esto, una misma oración podría tener varias simplificaciones válidas, sin un consenso claro sobre cuál es la mejor de ellas.

Sin embargo, hay criterios que sí influyen en la simplicidad de una oración, como ser la complejidad de las palabras, estando ésta ligada al significado de las mismas y especialmente a la pluralidad de significados que puedan tener. Además, la estructura y longitud de la oración juegan un rol importante en la constitución de una oración simple. La cantidad de palabras y la relación de los componentes de la oración determinan la facilidad de compresión de la oración, y por lo tanto la accesibilidad a la información presentada en la misma.

2.1. Aprendizaje profundo para tareas semánticas

En la tarea de simplificación debe conservarse la semántica de la oración, y en consecuencia es necesario que los modelos propuestos para el problema puedan generar una representación de la semántica de una oración. Por ello, en este trabajo se utilizan métodos de aprendizaje profundo, comúnmente conocidos como Redes Neuronales Profundas (DNN por sus siglas en inglés). Estos métodos se caracterizan

por tener capas ocultas de neuronas que procesan la entrada al mismo tiempo que generan una representación interna de los datos.

La importancia de estas capas está en su capacidad para modelar causas profundas: como no tenemos a priori conocimiento de cuál es la mejor representación para la semántica de una oración, se modelan los fenómenos observados, es decir, el texto, y a través de sucesivas abstracciones se llega a modelar las causas latentes, en este caso el significado. Cada capa oculta genera una nueva representación utilizando una combinación no lineal de los elementos de la representación generada por la capa anterior. Además, estas activaciones no lineales de las neuronas de cada capa ayudan a que estos problemas, cuya no linealidad hace que no puedan ser modelados con métodos más simples, sean tratables.

Como cada capa crea una representación intermedia, las redes neuronales son muy útiles para tareas con causas latentes no observables. El significado de las palabras y la relación entre ellas o estructuras sintácticas son ejemplos de causas latentes no observables que pueden ser representadas por las capas de una DNN.

En Simplificación de oraciones una secuencia de palabras (una oración) es transformada en otra secuencia (una oración más simple). Esto es un problema del tipo *sequence to sequence*. En la Figura 2.1[1] se muestra la arquitectura general de este tipo de sistemas *sequence to sequence*. Las secuencias de palabras pueden ser arbitrariamente largas y ser convertidas a secuencias de un largo que en principio podría no coincidir con el original. Podemos observar que el modelo procesa primero toda la secuencia de entrada y genera un vector que representa la semántica de la oración de origen (c). Luego, utilizando esta representación se genera la secuencia de salida.

Una vez que se define la arquitectura del modelo, es necesario encontrar los parámetros óptimos para cada tarea en particular. Esto se logra con un proceso de entrenamiento del modelo, en el cual se optimizan iterativamente los parámetros minimizando una función de pérdida. En el caso de tareas de traducción, por ejemplo, esta función de pérdida está basada en la similitud de las oraciones generadas con las traducciones correctas anotadas. Notar que, para poder entrenar el modelo, se necesita un conjunto de ejemplos correctamente traducidos o simplificados. Esta técnica de entrenar un modelo en base a ejemplos etiquetados es conocida como Aprendizaje Supervisado (*Supervised Learning*).

Las Redes Neuronales en particular son entrenadas utilizando el algoritmo de

[1] Créditos a Zhang Rong, `https://zr9558.com/`

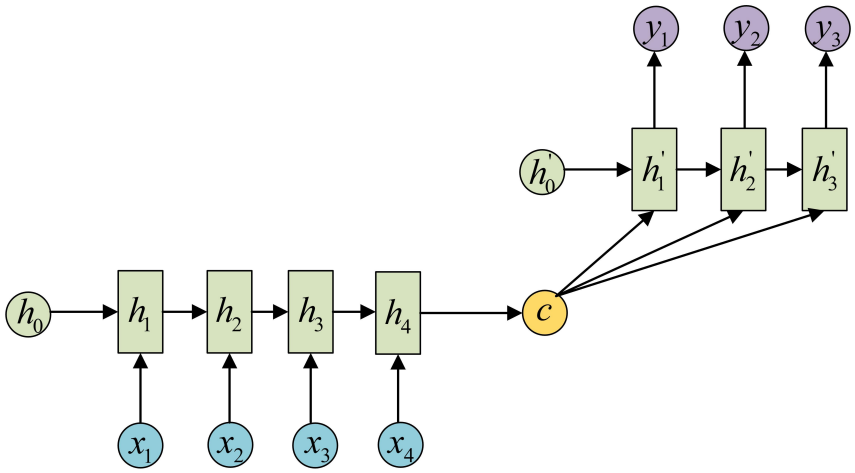

Figura 2.1: Esquema de un modelo de aprendizaje automático *Sequence to Sequence*. x_1, x_2, x_3 y x_4 son la secuencia de origen, mientras que y_1, y_2 y y_3 son la secuencia de destino.

Backpropagation, que permite propagar los posibles errores generados por cada capa a la anterior de una manera eficiente.

Es importante recalcar en este punto que las funciones de pérdida se limitan a comparar las simplificaciones anotadas con las simplificaciones generadas por el sistema. Como resultado, el sistema está diseñado para aprender los ejemplos anotados, y será penalizado por cualquier variación que pueda introducir en las simplificaciones generadas. Su desempeño final estará altamente ligado a la calidad y la cobertura de los ejemplos etiquetados. En el capítulo 4 se profundizará más este concepto.

2.1.1. Arquitecturas recurrentes

Dada la necesidad de procesar secuencias de largo arbitrario, las Redes Neuronales Recurrentes (RNN por sus siglas en inglés) son las que mejor se amoldan al problema de simplificación. A diferencia de las DNN clásicas, en las que se asume que toda entrada es independiente de las otras, las RNNs operan sobre secuencias de datos y mantienen una "memoria" (estado oculto) de los elementos previamente vistos en la secuencia. Como se puede ver en la Figura 2.1.1[2], estas arquitecturas

[2]Créditos a Christopher Olah, `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

procesan de una palabra a la vez. Al recibir cada palabra, actualizan su estado oculto con información de la nueva entrada, pero también con la información que se encontraba previamente en el estado oculto. De esta manera, acumulan la información relevante de toda la secuencia procesada hasta el momento. Además, pueden ser utilizadas para generar secuencias de salida de distintos tamaños.

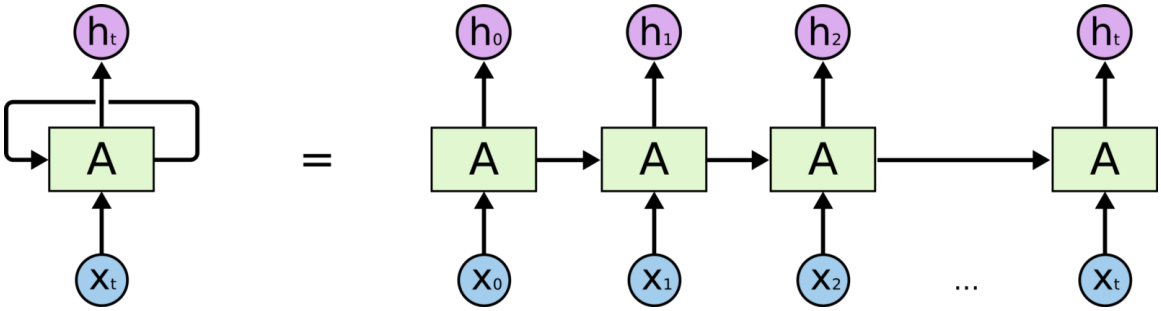

Figura 2.2: Diagrama de una Red Neuronal Recurrente.

Históricamente las RNNs han sido las que mejor desempeño han tenido en tareas de procesamiento de lenguaje natural, gracias a esta capacidad de mantener en su estado oculto información desde el principio de la secuencia.

Simplificación, al igual que otras áreas del procesamiento de lenguaje natural, es una tarea que tienen dependencias de largo alcance. Un caso contrario es el etiquetado gramatical o *PoS Tagging*, que es la tarea de asignar categorías morfosintácticas a cada palabra de acuerdo a su rol en el texto. La mayoría de las palabras tiene sólo una categoría gramatical posible, por ejemplo, *dog* es siempre un sustantivo. Si una palabra tiene más de una categoría posible, por ejemplo la palabra *light* que puede ser un verbo o un sustantivo, sólo es necesario mirar las palabras en su contexto cercano para determinar la etiqueta correcta. Es decir, las dependencias en esta tarea generalmente no son de largo alcance, ya que una palabra influye muy poco en la categoría gramatical de las palabras en el otro extremo de la oración.

Por otra parte, en la tarea de desambiguación de sentidos sí encontramos dependencias de largo alcance. Esta tarea consiste en asignar un significado a cada palabras de un texto con respecto a su definición y contexto. Dado que en los lenguajes naturales varias palabras pueden tener más de un significado, se debe mantener información de todas las palabras en la oración para determinar la correcta interpretación de estas. Finalmente, en Simplificación, las dependencias temporales de

largo alcance están relacionadas con la relevancia de una palabra en el contexto de una simplificación. Es decir, se decide si incluir, reemplazar o eliminar una palabra en la oración simplificada en base a todas las palabras restantes para asegurar que el resultado final tiene sentido.

Las LSTM (Memoria de Corto y Largo Plazo, por sus siglas en inglés) (Hochreiter and Schmidhuber 1997) son un tipo de RNNs diseñadas para mantener dependencias temporales a largo plazo, capacidad de la que carecen las RNNs clásicas. Con esta característica, sumada a las capacidades ya presentes por ser una RNN, las LSTMs han demostrado ser el estado del arte en tareas de procesamiento de secuencias con causas latentes complejas. Trabajando con ellas es se puede obtener un modelo *end-to-end* con las capacidades requeridas para la tarea de simplificación de oraciones.

2.1.2. Sistemas de traducción automática

Con los requisitos de mantener semántica, generar una representación de la misma, operar sobre secuencia y mantener dependencias sobre elementos de la secuencia, los modelos de Neuronales de Traducción Automática (NMT por sus siglas en inglés) son una buena opción para tratar el problema. Las variantes neuronales de estos modelos están compuestas por una o más LSTMs que transforman la oración original a una representación interna y otro grupo de LSTMs que transforman esa representación en una oración objetivo. Este tipo de arquitectura es llamada *encoder-decoder*, y ya ha sido presentada en la Figura 2.1.

Para mantener una noción de significado de las palabras, se emplean vectores de palabras o *word embeddings*, que son representaciones de la semántica de una palabra en forma de un vector denso de baja dimensionalidad. Los *word embeddings* se obtienen con un proceso previo no supervisado, es decir, que no requiere datos etiquetados. Por lo tanto, pueden ser entrenados con las grandes cantidades de texto disponibles en internet y alcanzan representaciones del significado de las palabras de alta calidad.

Para aplicar los métodos de traducción automática al problema de simplificación, se modela el problema como una traducción de un idioma (por ejemplo, inglés) a una versión simplificada del mismo idioma, con un vocabulario más simple. Un modelo NMT es agnóstico de los idiomas origen y destino. Teniendo los *word embeddings* correspondientes para cada uno de los idiomas y un conjunto de datos de

traducciones anotadas, el modelo NMT puede comenzar a generar traducciones (en este caso, simplificaciones).

Para este trabajo se utilizará una implementación libre de un modelo de traducción neuronal descripto anteriormente. Reutilizar esta arquitectura no sólo reduce los costos de desarrollo, sino que permite aplicar métodos del estado del arte. Aún más, evitamos la introducción de errores inherentes en el proceso de desarrollo.

2.1.3. Intuición detrás de Aprendizaje por Refuerzos

Los modelos NMT pueden aprender a replicar la semántica de la entrada en la oración de salida, pero no tienen noción de si las oraciones generadas son más simple o de qué estructuras son más valiosas de obtener. Al utilizar Aprendizaje por Refuerzos, se le asigna al modelo una recompensa por sus simplificaciones, dependiendo de su desempeño.

El Aprendizaje por Refuerzos es un paradigma de aprendizaje automático orientado a agentes que interactúan con un entorno. Difiere de los paradigmas supervisado y no supervisado en que no requiere pares anotados entrada-salida y en que las acciones sub óptimas no son explícitamente corregidas.

Un modelo de aprendizaje por refuerzos modela a un agente que debe realizar una tarea con la motivación de maximizar una noción de recompensa acumulativa. Al no contar con datos anotados, el sistema de recompensas permite dar una intuición al agente de qué acciones o estados son deseables realizar o alcanzar. De esta forma un modelo puede ser entrenado con información no etiquetada y se puede mesurar qué tan adecuada es una solución en base a la recompensa obtenida. Ejemplos clásicos de la aplicación de este paradigma son agentes que aprenden a jugar videojuegos (Mnih et al. 2013) y robots que aprenden a moverse (Jiang et al. 2018).

En el caso de simplificación de oraciones, no se cuenta con un conjunto de datos etiquetado con las estructuras gramaticales que resultan más simples ni las operaciones que generan la oración objetivo. Más aún, para cada oración de entrada, múltiples oraciones simplificadas son correctas al mismo tiempo, y no hay garantía de que la "mejor" simplificación sea la presente en los datos etiquetados. Por lo tanto, esta tarea puede beneficiarse ampliamente de un entrenamiento empleando aprendizaje por refuerzos. El sistema genera oraciones que son medidas no sólo por el solapamiento de sus palabras con la oración objetivo, sino que se pueden declarar

recompensas específicas para cada caso.

Estas recompensas, a diferencia de las funciones de pérdida utilizadas para la optimización de los modelos NMT, pueden modelar propiedades mucho más complejas de las oraciones. Modificando el sistema de recompensas se puede llevar al modelo a aprender cosas más allá de replicar las referencias. Por la naturaleza de los modelos de aprendizaje por refuerzo, el modelo tendrá libertad de explorar el espacio de simplificaciones, pudiendo así generar oraciones no presentes en el conjunto de datos de entrenamiento.

Para simplificación de oraciones, tres medidas de recompensa fueron propuestas: "Simplicidad" que mide cuán simplificada está una oración con respecto a su oración original, "Relevancia" que asegura que la información expuesta en la oración generada coincida con la de la oración original y "Fluidez" que mide qué tan bien formada está una oración. Durante el entrenamiento con aprendizaje por refuerzos, el modelo tratará de maximizar su recompensa y, por consiguiente, generará estructuras sintácticas que maximicen estos tres principios. Estas recompensas serán explicadas en el capítulo 4.

Capítulo 3

Aproximaciones al problema

Simplificación de oraciones es un área del Procesamiento de Lenguaje Natural que ya a sido tratada con anterioridad. Muchos métodos han sido propuestos, desde emplear modelos de traducción basados en árboles hasta el uso de *Neural Machine Translation*.

La simplificación usualmente es modelada usando 4 operaciones (Zhu et al. 2010; Woodsend and Lapata 2011; Wubben et al. 2012): *separar* oraciones complejas en oraciones más simples, *eliminar* o *reordenar* frases o palabras y *substituir* palabras o frases con otras más simples. Estas operaciones son aprendidas a partir del corpus de entrenamiento. Este corpus está compuesto de instancias de todas estas operaciones de forma no explícita, es decir, están ejemplificadas en pares de oraciones alineados con el formato (oración original, oración simplificada).

El área de simplificación de oraciones ha sido un área de investigación activa por mucho tiempo. Antes del auge de los métodos neuronales en el año 2012 con las redes neuronales en Visión por Computadora, simplificación de oraciones era tratada con métodos no neuronales, principalmente basados en modelos de traducción estadística.

(Zhu, Bernhard, and Gurevych 2010) crearon el conjunto de datos PWKP (*Wikismall*) que se usa en este trabajo y además trataron el tema de simplificación con traducción estadística empleando un modelo monolingüe basado en árboles, entrenado empleando el algoritmo de *Expectation-Maximization* (EM).

(Woodsend and Lapata 2011) trabajaron en simplificación de texto utilizando Gramáticas Quasi-síncronas. Se aprenden reglas simplificación sintáctica (elimina-

ción o reordenamiento de componentes); reglas de simplificación léxica y reglas de separación de oraciones complejas en oraciones más simples. Luego, moldean el problema de encontrar una simplificación adecuada como un problema de Programación Lineal Entera.

(Wubben, van den Bosch, and Krahmer 2012) emplean un modelo de PBMT (*Phrase Based Machine Translation*). Utilizan el paquete de alineamiento estadístico GIZA++ para entrenar un modelo de lenguaje de n-gramas en el dataset PWKP (Zhu et al. 2010) para aprender las probabilidades de los distintos n-gramas. Luego utilizan el decodificador Moses para generar n (en su caso, $n = 10$), simplificaciones, ordenadas por su puntaje de Moses. Estas oraciones son reordenadas con respecto a la distancia de Levenshtein a nivel palabra, contando la mínima cantidad de ediciones necesarias para llegar de una oración origen a una oración simplificada, donde las ediciones disponibles son inserción, eliminación y substitución de una palabra.

(Narayan and Gardent) utilizan Procesamiento Lingüístico Profundo y un *pipeline* para simplificación basado en tres componentes: un modelo probabilístico para realizar *separación* y *eliminación* (DRS-SM); un modelo de traducción basado en frases para *reordenamiento* y *substitución* (PBMT); finalmente un modelo de lenguaje para fluidez y gramaticalidad (LM). Se convienten las oraciones en identificadores empleando el *Stanford CoreNLP toolkit* y se pasan esas oraciones por el DRS-SM para generar una (o más) oraciones simples para luego simplificarlas aún más enviando la/s oración/es al PBMT-LM.

Las redes neuronales eventualmente tomaron mayor importancia en áreas como PLN. Modelos del tipo *Sequence to Sequence* (Seq2Seq) comenzaron a tener un buen desempeño en áreas como traducción automática. Con la capacidad de procesar secuencias y generar representaciones de los datos, estos modelos se mostraban idóneos para la tarea. Además, el surgimiento de los vectores de palabras (word embeddings), que permite tener una representación de la semántica de cada palabra, planteó un panorama prometedor para emplear modelos de este tipo para Simplificación de Oraciones. (Zhang and Lapata 2017) encontraron empíricamente que los en los conjuntos de datos basados en Wikipedia, un 83 % de la palabras en las oraciones originales son copiadas a la salida. Debido a esto, aproximaciones basadas en modelos Seq2Seq aprenden a copiar la entrada bastante bien a expensas de posibles sustituciones y algunas veces haciendo unos pocos cambios triviales.

Para incentivar una mayor cantidad de acciones de sustitución (Zhang and La-

pata 2017) usan una estructura de Aprendizaje por Refuerzos Profundo para simplificar oraciones. Los autores modelaron el problema siguiendo a un modelo encoder-decoder como un agente que lee una oración X y toma una acción $\hat{y}_t \in V$ (donde V en un vocabulario de salida). Las recompensas son dadas con métricas de simplicidad, relevancia y fluidez. Con esto entrenan un modelo usando el algoritmo de REINFORCE (Williams 1992). Trabajaron sobre conjuntos de datos extraídos de Wikipedia y de artículos de noticias escritos por humanos.

Este trabajo toma inspiración en el trabajo realizado por (Zhang and Lapata 2017), cambiando el algoritmo de Aprendizaje por Refuerzos REINFORCE por el algoritmo A2C, una versión síncrona del algoritmo A3C (Mnih et al. 2016). Este algoritmo, además de estar preparado para trabajar sobre lotes de experiencias (en este caso, oraciones), permite evitar la convergencia prematura a mínimos locales sub óptimos al mismo tiempo de favorecer la exploración del espacio de simplificaciones.

La motivación de emplear Aprendizaje por Refuerzos parte del trabajo en otras áreas de PNL donde Aprendizaje por refuerzos fué aplicado éxito. Ejemplos de estas áreas son procesamiento de texto y generación de diálogo. Estos trabajos se describen a continuación.

(Li, Monroe, Ritter, Galley, Gao, and Jurafsky 2016) trabajaron sobre las falencias de los modelos Seq2Seq para generación de diálogo y usaron Aprendizaje por Refuerzos profundos para generarlo. Cada acción a es una 'utterance' y los estados son los últimos dos turnos de diálogo $[p_i, q_i]$. El historial del diálogo es representado como un vector formado a partir de la concatenación de p_i y q_i que luego era introducido en una LSTM. Luego la política es representada con una representación estocástica. Las recompensas asignadas al entrenamiento venían de métricas de facilidad de respuesta, flujo de información y coherencia semántica. Entrenaron sobre el dataset de OpenSubtitles (Tiedemann 2009).

(Yu, Lee, and Le 2017) desarrollaron un método para PLN sin tener que procesar todo el texto disponible. Usaron una LSTM con saltos cuyos parámetros son N saltos permitidos, R cantidad de tokens leidos entre saltos y K máximo tamaño del salto. Entrenaron una LSTM y un word embedding θ_m al mismo tiempo que unos parámetros de salto en el texto θ_a. Usaron el algoritmo REINFORCE (Williams 1992) para aproximar el gradiente y entrenar el modelo.

(Lascarides, Lemon, Guhe, Keizer, Cuayáhuitl, Efstathiou, Engelbrecht, and Dobre 2017) entrenan bots para jugar Settlers of Catan. El trabajo se centra en la

estrategia de negociación, para la cual usa Aprendizaje por Refuerzos Profundo. Entrenan sobre un corpus de conversaciones entre humanos jugando el mismo juego (Afantenos et al. 2012).

(Peng, Li, Li, Gao, Çelikyilmaz, Lee, and Wong 2017) usan Aprendizaje Jerárquico de Política, separando el problema de resolver una tarea (travel planning, en el caso del paper) en subtareas resolubles a través de acciones primitivas. Cuentan con dos políticas: π_g que se encarga de seleccionar una subtarea g de un conjunto de tareas dado el estado del entorno y $\pi_{a,g}$ que, dado una tarea g y el estado, selecciona una tarea a de un conjunto de acciones primitivas. Trabajan sobre un corpus de interacción humano-humano llamado Frames (Asri et al. 2017).

Capítulo 4

Diseño y Arquitectura

En esta sección se presenta los preliminares y arquitectura del los modelos empleados en este trabajo. Se profundiza el concepto de Modelo Neuronal Encoder-Decoder, junto con una explicación del paradigma de Aprendizaje por Refuerzos. A continuación, se explica el pipeline de Simplificación de oraciones y se introducen los algoritmos A3C y A2C. Finalmente se describe la arquitectura de simplificación propuesta para este trabajo.

4.1. Modelo Neuronal Encoder-Decoder

Un modelo *Encoder-Decoder* es un modelo de redes neuronales compuesto de dos partes: un *encoder* que transforma una secuencia X en una secuencia de estados ocultos $(h_1^E, h_2^E, ..., h_{|X|}^E)$ con una LSTM (Hochreiter and Schmidhuber 1997) y un *decoder* que usa otra LSTM para generar una oración objetivo Y a partir de estos estados ocultos.

Los estados ocultos $\mathbf{h}_t^T$ contienen una representación de la información procesada de la secuencia S por la LSTM T hasta el instante de tiempo t. Estos estados son calculados de la siguiente forma:

$$\mathbf{h}_t^T = LSTM(s_t, \mathbf{h}_{t-1}^T) \tag{4.1}$$

donde $s_1, s_2, ..., s_{|S|}$ es una secuencia de datos.

Para el *encoder* E, los estados ocultos $\mathbf{h}_t^E$ pueden entenderse como el estado de la semántica de la sub secuencia $x_{1:t}$ de la oración de entrada X hasta el momento t.

Del mismo modo, los estados $\mathbf{h}_t^D$ generados por el *decoder* D mantienen información de la secuencia de palabras $y_{1:t}$ generadas por el modelo hasta el momento t.

Para ayudar al proceso de generación de la secuencia de salida se utiliza un mecanismo de atención. La atención permite al *decoder* utilizar todos los estados ocultos generados por el *encoder*, en lugar de utilizar sólo el último. Esto se logra al asignar más relevancia a los estados ocultos más relacionados con la porción de la oración que se está generando en ese momento. En la tarea de traducción esto resulta muy útil, ya que las palabras están quasi-alineadas entre la oración de origen y la de destino.

El mecanismo de atención mantiene un vector dinámico de contexto $\mathbf{c}_t$, que condensa información del contexto en el que se encuentra el modelo y es la suma pesada de los estados ocultos de la oración original:

$$\mathbf{c}_t = \sum_{i=1}^{|X|} \alpha_{ti} \mathbf{h}_i^E \tag{4.2}$$

donde α_{ti} son los puntajes de atención calculados de la siguiente manera:

$$\alpha_{ti} = \frac{\exp(score(\mathbf{h}_t^D, \mathbf{h}_i^D))}{\sum_i \exp(score(\mathbf{h}_t^D, \mathbf{h}_i^D))} \tag{4.3}$$

donde *score* es una función basada en el contenido de los vectores $\mathbf{h}_t^T$ y $\mathbf{h}_i^T$ (Luong et al. 2015). Los mecanismos de atención han mostrado mejorar el desempeño de modelos de traducción de secuencias de largo variable.

Finalmente, la generación de las oraciones está condicionada por cada palabra previamente generada $y_{1:t}$ y del vector de contexto $\mathbf{c}_t$, como se describe en la fórmula 4.4:

$$P(Y|X) = \prod_{t=1}^{|Y|} P(y_t|y_{1:t-1}, X) \tag{4.4}$$

$$P(y_t|y_{1:t-1}, X) = softmax(g(h_t^D, c_t)) \tag{4.5}$$

donde $g(\cdot)$ es una red neuronal recurrente con una capa oculta con la siguiente parametrización:

$$g(\mathbf{h}_t^D, \mathbf{c}_t) = \mathbf{W}_o \tanh(\mathbf{U}_h \mathbf{h}_t^D + \mathbf{W}_h \mathbf{c}_t) \tag{4.6}$$

donde $\mathbf{W}_o \in \mathbb{R}^{|V|\times d}$, $\mathbf{U}_h \in \mathbb{R}^{d\times d}$ y $\mathbf{W}_h \in \mathbb{R}^{d\times d}$; $|V|$ es el tamaño del vocabulario de salida y d el tamaño de la unidad oculta.

4.2. Preliminares de Aprendizaje por Refuerzos

Un escenario estándar de Aprendizaje por Refuerzos consiste en un agente que interactúa con un entorno $\mathcal{E}$ a lo largo de pasos de tiempo discretos siguiendo una política π. Sea $\mathcal{A}$ el espacio de acciones que el agente puede realizar y S el conjunto de estados, una política π es una distribución de probabilidad sobre el espacio de acciones definida como:

$$\pi : S \times \mathcal{A} \to [0, 1]$$
$$\pi(a|s) = P(a_t = a|s_t = s)$$

Un episodio en un marco de Aprendizaje por Refuerzos se desarrolla de la siguiente manera: A cada paso de tiempo t, el agente recibe un estado s_t y selecciona una acción $a_t \in \mathcal{A}$ de acuerdo a la política π. El agente luego recibe un estado s_{t+1} y una recompensa r_t. El proceso continúa hasta que el agente recibe un estado terminal, luego del cual el proceso se reinicia. Se define la recompensa en el paso de tiempo t como $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$, donde $\gamma \in (0, 1]$ es un factor de descuento. Luego, el objetivo del agente es maximizar la recompensa esperada para cada estado s_t.

Clásicamente los problemas de aprendizaje por refuerzos son planteados sobre un Proceso de Decisión Markoviano (MDP, por sus siglas en inglés). Un MDP está definido como una 4-upla (S, A, P_a, R_a) donde S es un conjunto de estados, A es un conjunto de acciones, P_a es la probabilidad de transición del estado s al estado s' tomando la acción a y está definida como:

$$P_a(s, s') = Pr(s_{t+1} = s'|s_t = s, a_t = a)$$

Finalmente $r_a(s, s')$ es la recompensa inmediata para dicha transición. Los estados deben ser tal que contengan en todo momento toda la información necesaria del entorno para que el agente pueda actuar. En caso de MDP, el estado debe contener una descripción completa del entorno, mientras que en otras variantes donde no es posible definir una probabilidad P_a con respecto a los estados (usualmente siendo

continuo), este debe contener en sí mismo información del agente durante ese episodio. En el caso de este trabajo final, los estados ocultos de una LSTM codifican esta información.

Dos funciones relevantes en un marco de Aprendizaje por Refuerzos son la *value function* V^π y la función de *action-value* Q^π. Por un lado la *value function* que se define como $V^\pi(s) = \mathbb{E}[R_t|s_t = s]$ que es la recompensa esperada a largo plazo con el factor de descuento para un estado s bajo la política π. Por otro lado se tiene la función de *action-value* definida como $Q^\pi(s, a) = \mathbb{E}[R_t|s_t = s, a]$, similar a la *value function* pero con un parámetro adicional que es la acción actual a. Esta se refiere a la recompensa a largo plazo para el estado s tomando la acción a bajo la política π.

4.2.1. Clasificación de algoritmos de Aprendizaje por Refuerzos

Los algoritmos de Aprendizaje por Refuerzos pueden ser clasificados en dos categorías: algoritmos *Model-based* o *Model-free*.

Los algoritmos *Model-based* tratan de generar una representación de la dinámica del entorno. Para esto, el modelo aprende la probabilidad de transición $T(s_{t+1}|(a_t, s_t))$ para cada par de estados s_t y acción a a siguiente estado s_{t+1}. Si esta transición es aprendida con éxito, el agente sabrá cuál es el mejor curso de acción para cada estado, pero estos modelos se vuelven imprácticos cuando el espacio de estados o de acciones son grandes (dado que $T : S \to S \times \mathcal{A}$).

Por otro lado, los algoritmos *Model-free* no tienen la necesidad de almacenar todas las combinaciones de estado-acción, sino que mantiene una representación parcial aprendida a partir de prueba y error en los estados encontrados. La tarea de Simplificación, dado que el espacio de acciones para el problema de simplificación de oraciones es el vocabulario del lenguaje objetivo y que el espacio de estados es continuo, está limitada a emplear algoritmos *Model-free*.

El aprendizaje del agente también puede ser clasificado en dos categorías: *On-policy* y *Off-policy*. En el aprendizaje *On-policy*, el agente aprende a partir de las acciones muestreadas de la misma política que se entrena, mientras que en el aprendizaje *Off-policy* el agente aprende a partir de acciones muestreadas de otra política distinta a la que se entrena (ϵ-greedy, por ejemplo). En este trabajo, se aplicará

aprendizaje *On-policy*.

4.3. Pipeline de Simplificación de oraciones

A continuación se presenta la arquitectura planteada por (Zhang and Lapata 2017) para simplificación de oraciones. Antes de entrar en detalles de la arquitectura, se da una breve explicación del algoritmo REINFORCE (Williams 1992) que es el empleado para entrenar el modelo *encoder-decoder*.

4.3.1. REINFORCE

Una forma de aplicar Aprendizaje por Refuerzos es parametrizar una política $\pi(a|s;\theta)$ y, dado que el objetivo es maximizar la recompensa, actualizar los parámetros de θ utilizando ascenso de gradiente en $\mathbb{E}[R_t]$. Dado el costo computacional de calcular esa esperanza, el algoritmo REINFORCE, que es usado en el algoritmo base de simplificación de texto, actualiza los parámetros de θ en la dirección de $\nabla_\theta \log \pi(a_t|s_t;\theta)R_t$, que es un estimador sin sesgo de $\nabla_\theta \mathbb{E}[R_t]$.

Para reducir la varianza y mantener el estimador insesgado, se puede sustraer una función aprendida de los estados $b(s_t)$ llamada *baseline* (Williams 1992). El gradiente resultante queda como:

$$\nabla_\theta \log \pi(a_t|s_t;\theta)(R_t - b(s_t)) \tag{4.7}$$

En este trabajo se utilizan redes recurrentes con celdas LSTM tanto para el *encoder* como para el *decoder*.

El uso de Aprendizaje por Refuerzos viene de modelar al *encoder-decoder* como un agente que a cada instante de tiempo t toma una acción $\hat{y}_t$, que es una palabra perteneciente al vocabulario de salida. El agente continúa tomando acciones hasta que encuentra un Final de Oración (EOS por sus siglas en inglés), terminando con una secuencia $\hat{Y} = (\hat{y}_1, \hat{y}_2, ..., \hat{y}_{|\hat{Y}|})$. Luego, los pesos del *encoder-decoder* son ajustados usando el algoritmo REINFORCE para maximizar al recompensa. Siendo $P_{LR}(\cdot|X)$ la distribución descrita en 4.4, la función de pérdida del sistema queda definida por:

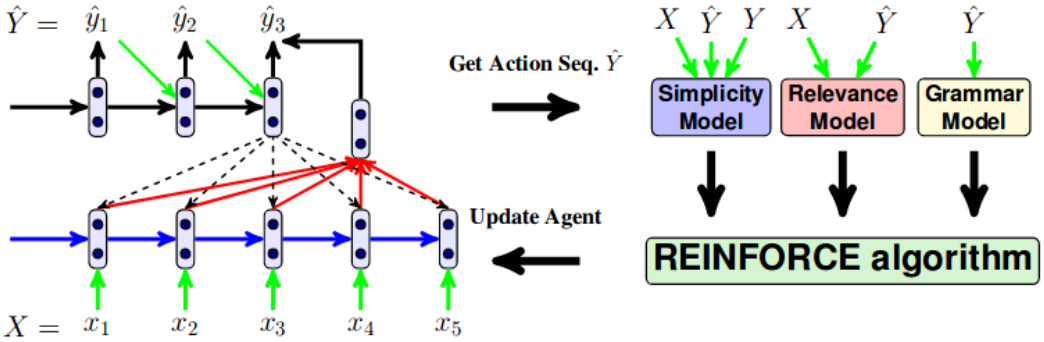

Figura 4.1: Esquema de la arquitectura original del pipeline para Simplificación de Oraciones usando REINFORCE.

$$\mathcal{L}(\theta) = -\mathbb{E}_{(\hat{y}_1,...,\hat{y}_{|\hat{Y}|})...}[r(\hat{y}_1, \hat{y}_2, ..., \hat{y}_{|\hat{Y}|})] \tag{4.8}$$

Las recompensa que recibe el agente $r(\hat{Y})$ es la suma pesada de tres componentes:

$$r(\hat{Y}) = \lambda^S r^S + \lambda^R r^R + \lambda^F r^F \tag{4.9}$$

donde $\lambda^S, \lambda^S, \lambda^S \in [0, 1]$. $\lambda^S r^S$ es un término de recompensa correspondiente a la Simplicidad (qué tan simple es la oración). r^S es calculado empleando la métrica SARI (descripta en el capítulo 5). $\lambda^R r^R$ un término correspondiente a la Relevancia (qué tanta información con respecto a la original se mantiene). r^R es calculado empleando una LSTM entrenada con un *autoencoder* que transforma las oraciones X e $\hat{Y}$ en los vectores q_X y $q_{\hat{Y}}$ para luego tomar su distancia coseno. Finalmente $\lambda^F r^F$ correspondiente a la Fluidez (bien formada está la oración) y es calculada empleando la exponencial de la *perplexity* de la oración $\hat{Y}$ con un modelo de lenguaje, que es una distribución de probabilidad sobre las secuencias de palabras modelado (en este caso) con una LSTM, empleando oraciones simples. En este trabajo se utilizarán solo recompensas por Simplicidad.

4.4. Asynchronous Advantage Actor-Critic

Asynchronous Advantage Actor-Critic (Mnih et al. 2016) (A3C, para abreviar), es un algoritmo de Aprendizaje por Refuerzos del tipo *Actor-Critic*. A3C usa el

cálculo de la *Advantage* para entrenar el modelo. La *Advantage* de la acción a_t en el estado s_t se define como

$$A(a_t, s_t) = Q(a_t, s_t) - V(s_t)$$

Un estimador de $A(a_t, s_t) = Q(a_t, s_t) - V(s_t)$ es $R_t - b_t$, inspirados en el gradiente del algoritmo REINFORCE, que substrae el *baseline* 4.7. Tomamos la función de valor como un estimador de la función de baseline $b_t(s_t) \approx V^{\pi}(s_t)$, $R_t - b_t$, y R_t es estimador de $Q^{\pi}(a_t, s_t)$. Esta forma de modelar el problema es conocida como una arquitectura *Actor-Critic*.

En A3C, en lugar de tener un agente que interactúa con un entorno, se tienen múltiples instancias del mismo esquema. Se define una red global y múltiples agentes con sus propios parámetros ejecutándose en paralelo. Además del aumento de velocidad en el entrenamiento, la experiencia de cada agente ejecutando en paralelo es independiente de la de los otros, lo cual hace que el aprendizaje sea más variado. El algoritmo mantiene una política $\pi(a_t|s_t;\theta)$ (el *actor*) y un estimador de la función de valor $V(s_t;\theta_v)$ (el *critic*). La política y la función de valor son actualizados cada t_{max} acciones o cuando un estado terminal es alcanzado. Luego, los parámetros son actualizados con ascenso de gradiente en la dirección de:

$$\nabla_{\theta'} log\pi(a_t|s_t;\theta')A(s_t, a_t;\theta,\theta_v) \tag{4.10}$$

donde $A(s_t, a_t;\theta,\theta_v)$ un estimador de la función de ventaja dado por $\sum_{i=0}^{k-1}\gamma^i r_{t+i} + \gamma^k V(s_{t+k};\theta_v) - V(s_t;\theta_v)$ donde k puede variar de estado a estado y está acotado superiormente por t_{max}.

$$A(s_t, a_t;\theta,\theta_v) = \sum_{i=0}^{k-1}\gamma^i r_{t+i} + \gamma^k V(s_{t+k};\theta_v) - V(s_t;\theta_v) \tag{4.11}$$

Esta ecuación corresponde a sumar la recompensa acumulada desde el instante de tiempo t hasta el instante de tiempo $t + k - 1$ y se emplea el valor de la *value function* para estimar el valor de la recompensa desde el paso $t + k$ hasta el final del episodio. Luego el término de la sumatoria corresponde con una aproximación de R_t a la cual, como el cálculo de la *Advantage* lo indica, se le substrae la *Value Function* del estado s_t.

El uso del *Advantage* es muy importante en la optimización del modelo. Los agentes son utilizados para explorar el ambiente con distintos episodios. En cada

episodio, el agente experimenta una secuencia de cambios de estado y genera las correspondientes acciones. Como podemos ver, en el primer término de la ecuación 4.10, se está maximizando la verosimilitud o *likelihood* de las acciones producidas por la política π. Si este término no estuviera multiplicado por la función de ventaja, entonces se aumentaría la verosimilitud de todas las secuencias de acciones experimentadas por el agente, independientemente de si obtienen una buena recompensa o no. La función de ventaja nos permite penalizar las trayectorias que obtienen una recompensa menor de la esperada por el modelo actual (estimada a través de la *value-function* $V(s_t;\theta_v)$), y aumentar la verosimilitud de las trayectorias con una recompensa mayor a la esperada.

En el caso del problema de Simplificación, todas las recompensas son cero excepto por el último paso del episodio en el cual se completa la oración. Como resultado, todas las actualizaciones son llevadas a cabo en un estado terminal, y la fórmula 4.11 se reescribe como:

$$A(s_t, a_t; \theta, \theta_v) = \sum_{i=0}^{T-t} \gamma^i r_{t+i} - V(s_t; \theta_v)$$
$$A(s_t, a_t; \theta, \theta_v) = \gamma^{T-t} r_T - V(s_t; \theta_v)$$

donde T es el largo de la secuencia

Mientras los parámetros θ de la política y θ_v de la función de valor son mostrados separadamente, en la práctica se comparten muchos de esos parámetros. Una opción es emplear una DNN y utilizar dos capas de salida distintas para el *actor* y el *critic* (Mnih et al. 2016).

Otra técnica usada para favorecer la exploración y evitar la convergencia prematura a mínimos locales subóptimos es sumar la entropía de la política π a la función de recompensa. El gradiente de la función objetivo completa, incluyendo el término de regularización de la entropía con respecto a la política π queda:

$$\nabla_{\theta'} log\pi(a_t|s_t;\theta')(R_t - V(s_t;\theta_v)) + \beta\nabla_{\theta'} H(\pi(s_t;\theta')) \tag{4.12}$$

donde H es la entropía. El hiperparámetro β controla la fuerza del término de regularización.

4.4.1. Advantage Actor-Critic

Una variante de A3C (y la que se usa en la arquitectura final) es *Advantage Actor-Critic* (A2C para abreviar). A2C es una versión síncrona y determinística que espera por cada agente termine su ejecución antes de actualizar los parámetros, promediando entre todos los actores. Esta implementación tiene mejor desempeño que A3C y es más *cost-efficient* en arquitecturas de cómputo con una GPU (OpenAI 2017).

4.4.2. Arquitectura propuesta

Para este trabajo proponemos utilizar un modelo neuronal *Encoder-Decoder*, descripto en la sección 4.1, implementado por Google para Traduccion Automática y disponible libremente en `https://github.com/tensorflow/nmt`. El *Encoder-Decoder* utiliza capas LSTM junto con un mecanismo de atención basado en (Luong et al. 2015).

Este modelo será optimizado utilizando un entrenamiento en dos etapas, primero usando el paradigma de aprendizaje supervisado clásico y luego con un paradigma de aprendizaje por refuerzos con un algoritmo A2C. Toda la implementación fue realizada con la librería Tensorflow (`https://www.tensorflow.org/`), en la versión 1.10.1.

En la primera etapa de entrenamiento se optimiza el modelo NMT utilizando el algoritmo ADAM para ajustar los pesos del modelo. La función de pérdida utilizada es la entropía cruzada o *cross-entropy* entre la simplificación anotada Y y la producida por el modelo $\hat{Y}$.

$$loss(\hat{Y}, Y) = \sum_{w \in V} P_Y(y) \log P_{\hat{Y}}(y|X, \theta) \tag{4.13}$$

donde X es la secuencia de entrada y θ corresponde a los parámetros del modelo, P_Y corresponde a la distribución de los ejemplos anotados y $P_{\hat{Y}}$ corresponde a la distribución generada por el modelo.

En la segunda etapa de entrenamiento se modifica la arquitectura anterior como se muestra en la figura 4.4.2. Se agrega una capa densa a la salida del *decoder* para calcular la función de valor $V(s_t|\theta_v)$ para los s_t estados ocultos de la LSTM del

decoder. Se cambia la función a optimizar a 4.8 y los gradientes para el *actor* se calculan como se indica en 4.12.

Para calcular la recompensa se emplea la función SARI como estimador de "simplicidad", que será descripta en detalle en la sección 5.4.3.

La pérdida para el *critic* queda definida como $\mathcal{L}(\theta'_v) = \sum_t \delta(R_t - V(s_t; \theta'_v))^2 / \delta\theta'_v$ y se agrega a la función de pérdida del *actor*. Se mantienen todos los pesos de la etapa anterior y se inicializan los pesos de la capa densa. Luego se optimiza el modelo empleando el algoritmo RMSProp.

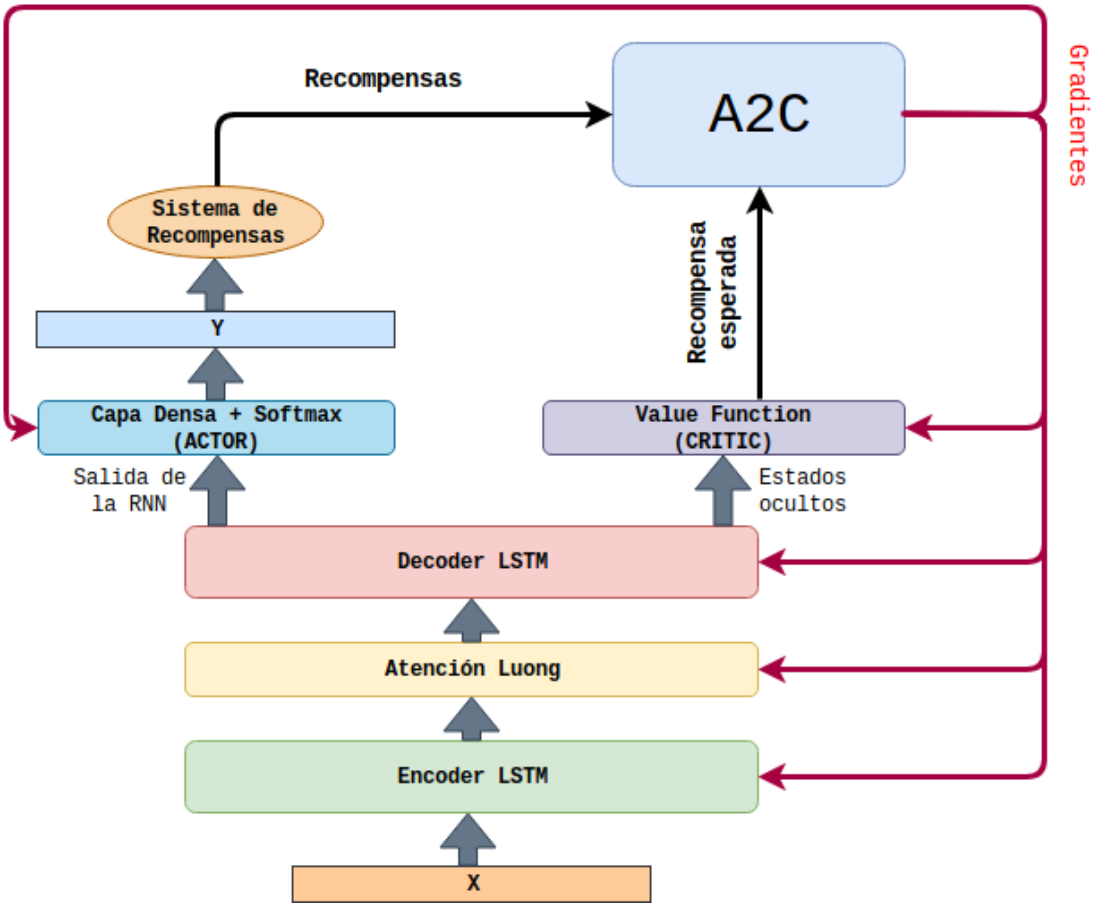

Figura 4.2: Esquema de la arquitectura propuesta.

Para simular los distintos agentes que requiere el modelo de A2C, se utiliza el entrenamiento por *batches* del modelo neuronal *Encoder-Decoder*. Esto significa que múltiples oraciones son procesadas en paralelo por el mismo modelo.

En el paradigma A2C, distintos agentes son instanciados a partir del modelo original y se utilizan para jugar distintos episodios. Estos agentes coleccionan experiencias distintas en forma de recompensas obtenidas de la secuencia de estados y acciones generadas. Cada cierta cantidad de pasos, el algoritmo optimiza los agentes en base a la experiencia de cada uno y combina todos los gradientes parciales de cada agente para optimizar el modelo original. Sin embargo, en el escenario de simplificación de texto sólo existe una señal de recompensa, al finalizar el episodio. Es decir, cuando la oración está completa. Por lo tanto los gradientes parciales siempre son nulos y los agentes nunca divergirían del modelo original. Luego no existen diferencias empíricas entre utilizar el entrenamiento por *batches* y los múltiple agentes del algoritmo A2C. La arquitectura se esquematiza en la figura 4.4.2

Capítulo 5

Experimentos

En esta sección se describirán los distintos experimentos llevados a cabo para evaluar el desempeño del modelo bajo las distintas arquitecturas posibles. Se incluye una descripción del conjunto de datos, las métricas utilizadas en la evaluación y las distintas configuraciones de entrenamiento. Finalmente, se presentan los resultados obtenidos y se analizan las distintas dificultades experimentadas durante la optimización con el algoritmo A2C.

5.1. Conjunto de datos

Los conjuntos de datos *Wikismall* (Zhu et al. 2010) y *Wikilarge* (Zhang and Lapata 2017) son conjuntos de oraciones extraídas de la Wikipedia para simplificación de texto. *Wikismall* fue formado con oraciones complejas alineadas automáticamente a oraciones simples de la Wikipedia ordinaria y la Wikipedia simplificada (en inglés). *Wikilarge* está compuesto de distintos corpus para simplificación extraídos también de Wikipedia. Las oraciones están separadas entre oraciones originales (*source*) y oraciones simplificadas (*destiny*).

Las oraciones, tanto de wikilarge y wikismall son oraciones de distinta longitud, que contiene desde una sola palabra hasta más de 80. Los signos de puntuación son considerados palabras en sí mismos. Wikilarge contiene un total de casi 7.5M de palabras (7.492.446) en sus oraciones originales y más de 5.5M de palabras (5.512.275) en sus oraciones simplificadas. Por su lado, wikismall cuenta con más de 2.1M palabras (2.163.047) en sus oraciones originales y más de 1.8M (1.812.852) palabras en

sus oraciones simplificadas.

La longitud promedio de una oración original de wikilarge es de (aproximadamente) 26 palabras, mientras que las oraciones simplificadas es de (aproximadamente) 19 palabras. En wikismall, las longitudes promedio de oraciones originales y simplificadas es de (aproximadamente) 25 y 21 palabras, respectivamente. En la figura 5.1 se presentan una comparación entre la distribución de la longitud de oraciones originales y simplificadas para ambos conjuntos de datos.

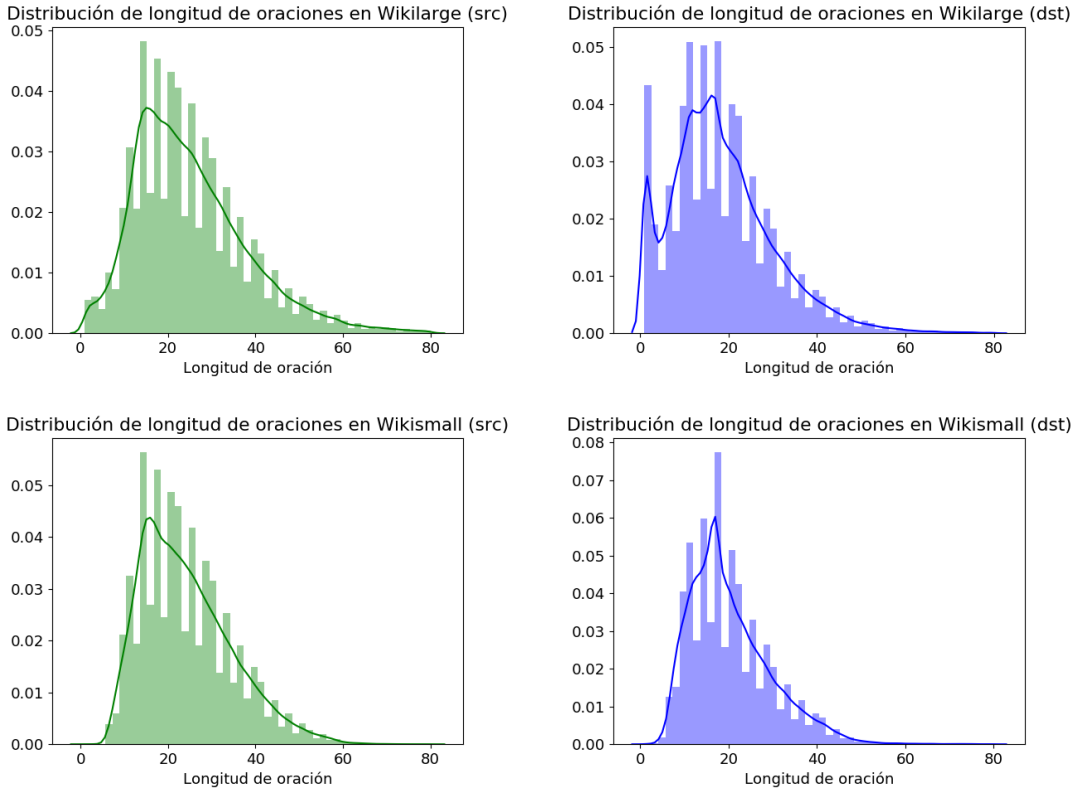

Figura 5.1: Distribución de longitud de oraciones en los conjuntos de datos *wikilarge* y *wikismall*. En verde la distribución sobre las oraciones *source* y en azul las oraciones *destiny*.

Ambos conjuntos de datos fueron separados en particiones aleatorias como se lo describe en la tabla 5.1. El conjunto de entrenamiento es utilizado para optimizar los parámetros del modelo. El conjunto de validación, por otra parte, nunca es expuesto al algoritmo de entrenamiento y se utiliza para comparar el desempeño de clasificadores con distintos hiperparámetros. Finalmente, el conjunto de evaluación

es utilizado para estimar el desempeño del modelo en un conjunto de datos nuevo, y por lo tanto evaluar su capacidad de generalización.

	wikismall	wikilarge
Entrenamiento	88837	296402
Validación	205	992
Evaluación	100	359
Total	89142	297753

Cuadro 5.1: Cantidad de oraciones por conjunto de datos y subconjunto de entrenamiento, validación y evaluación.

Como en la mayoría de los problemas del traducción automática, hay un vocabulario para las secuencias de origen y otro para las secuencia de destino. Los vocabularios, tanto de wikilarge y wikismall, fueron obtenidos de un trabajo previo por los mismos (Zhang and Lapata 2017). El vocabulario de origen de wikilarge está compuesto por 39028 palabras, mientras que el vocabulario de destino cuenta con 30342. En wikismall el vocabulario de origen y destino están compuestos por 20080 y 16598 palabras, respectivamente.

5.2. Word Embeddings

Para poder ingresar los ejemplos de texto a un modelo neuronal, primero es necesario representar las palabras de forma numérica. Tradicionalmente, cada palabras es reemplazada por un vector *one hot*, es decir, un vector de ceros del tamaño del vocabulario que tiene un uno en el lugar correspondiente a la palabra dada. Este tipo de vectores no sólo tiene una dimensionalidad demasiado alta para modelos neuronales, sino que también elimina todas las relaciones semánticas entre palabras, ya que el producto punto entre dos vectores *one hot* siempre es nulo.

Para evitar estos problemas, se utilizan *word embeddings* densos y de baja dimensionalidad, optimizados a partir de grandes cantidades de texto extraído de diversas fuentes. La idea detrás de estos vectores es tener una representación que permita que palabras de semántica similar estén cercanas entre ellas en un espacio vectorial. Para este trabajo se utilizan los vectores preentrenados de GloVe (Pennington, Socher, and Manning 2014), los cuales tienen la propiedad de que los *word embeddings* simi-

lares están cerca con respecto a una distancia Euclidea entre ellos. Otras relaciones semánticas también están representadas en forma de subestructuras lineales en el espacio. Un ejemplo conocido es que el vector correspondiente a REINA se encuentra cerca del vector resultante de (REY - HOMBRE + MUJER). En la figura 5.2 se observa otro ejemplo de cómo las relaciones semánticas tienen interpretaciones geométricas en este espacio de *embeddings*.

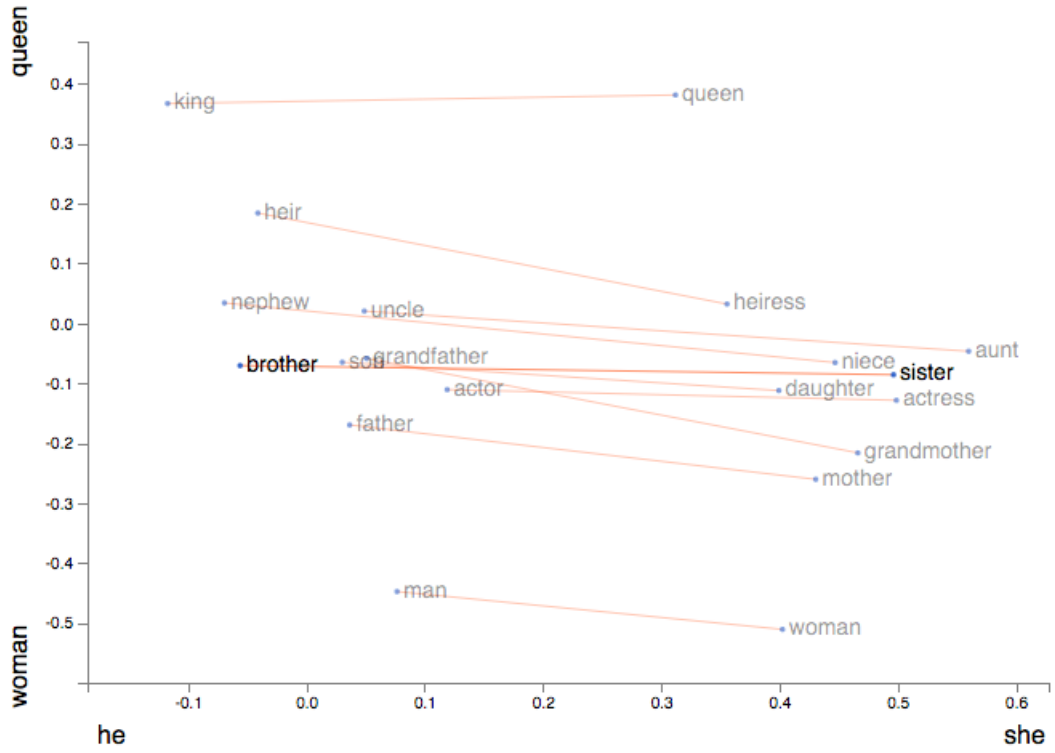

Figura 5.2: Visualización de las subestructuras lineales en GloVe. `https://p.migdal.pl/2017/01/06/king-man-woman-queen-why.html`

Utilizando estos word embeddings proveemos al modelo con una representación de las palabras que ya captura su semántica, en lugar de utilizar una representación plana y forzar al modelo a crear otra representación interna solamente en base a los datos de entrenamiento.

Los word embeddings han demostrado un impacto altamente positivo en diversas tareas del procesamiento del lenguaje natural. Sin embargo, cabe destacar que la

representación elegida no contempla palabras del lenguaje que no fueron vistas durante el entrenamiento o aparecieron muy pocas veces, las cuales son reemplazadas por un token <unk>y asignadas a un vector aleatorio.

5.3. Hiperparámetros

Los hiperparámetros son parámetros que rigen el comportamiento del modelo, pero son definidos antes de comenzar en entrenamiento. En el caso de la arquitectura propuesta, los hiperparámetros necesarios para la primera etapa de entrenamiento del modelo son:

- *Batch Size*: la cantidad de oraciones procesadas al mismo tiempo. La importancia del tamaño del *batch size* es relevante ya que puede mejorar el uso de los recursos disponible y acelerar el entrenamiento. Sin embargo, un *batch size* demasiado grande puede llevar sabotear la capacidad de generalización del modelo (Keskar et al. 2016).

- Cantidad de épocas que entrena el modelo. Con cada época de entrenamiento, el modelo recorre todos los ejemplos del conjunto de datos de entrenamiento. Se espera que el desempeño mejore luego de cada actualización, pero a medida que el clasificador se acerca al mínimo local la velocidad de descenso disminuye y ya no conviene seguir invirtiendo recursos en la optimización del modelo. Continuar el entrenamiento del modelo por demasiadas épocas puede tener incluso efectos adversos, cuando se produce un sobreajuste u *overfitting* a los datos de entrenamiento y se pierde generalización sobre los datos de validación.

- *Dropout*: es la probabilidad de apagar una compuerta de la celda LSTM. El dropout no solo sirve para agregar no determinismo a los modelos, sino también para mejorar la generalización y evitar el sobreajuste a un único conjunto de datos.

- Cantidad de unidades o neuronas de cada capa LSTM. Este hiperparámetro determina la capacidad de representación del modelo. Con muy pocas neuronas, el modelo no podrá representar suficientes causas latentes que explican los datos. Con demasiadas neuronas, el modelo puede sobreajustar los datos

e incluso memorizarlo. El tamaño de la capa también influye en el tiempo de entrenamiento y los recursos computacionales necesarios.

- *Learning Rate*: coeficiente de la agresividad de las actualizaciones de los gradientes en cada paso de entrenamiento. Si el *learning rate* es demasiado alto, el modelo tiene posibilidades de diverger del mínimo local. Si es bajo, las actualizaciones serán demasiado conservadoras y el entrenamiento llevará más tiempo. Optimizadores como Adam sólo necesitan una estimación inicial de este parámetro y varían el *learning rate* de acuerdo al progreso del aprendizaje.

- Optimizador: la función de optimización que se usa para entrenar. Para la primera etapa del entrenamiento se emplea ADAM. Para la segunda etapa de entrenamiento con A2C se utiliza el optimizador RMSProp. Se hace este cambio de optimizador, dado que RMSProp es el estado del arte en tareas de aprendizaje por refuerzos (Mnih et al. 2015).

En la segunda etapa de entrenamiento, al emplear el algoritmo de A2C, a los hiperparámetros anteriores se les suman un nuevo conjunto de hiperparámetros, que son:

- Factor de Descuento: factor de descuento $\gamma \in (0, 1]$ para el cómputo de la recompensa R_t. Este factor determina el impacto de la recompensa a largo plazo en las acciones tomadas. Con un factor de descuento alto, las recompensas posteriores tienen un peso similar al las recompensas inmediatas.

- VF Coef: coeficiente de relevancia de la pérdida de la *value function* en el cómputo de la función de pérdida total.

- Ent Coef: coeficiente β del término de la entropía en la función de pérdida de A2C.

- PG Coef: coeficiente de relevancia de la pérdida del actor en el cómputo de la función de pérdida total.

5.4. Métricas de evaluación

5.4.1. BLEU

BLEU (Bilingual Evaluation Understudy) (Papineni, Roukos, Ward, and Zhu 2002) es una métrica de la *calidad* de un texto traducido automáticamente de un lenguaje a otro. La *calidad* es considerada como la correspondencia entre un texto generado por la máquina y uno generado por un humano.

BLEU es una modificación de la precisión de n-gramas. Para computar precisión (de unigramas), se cuenta la cantidad de palabras (unigramas) de la oración candidata que ocurren en alguna oración de referencia y divide por la cantidad de palabras de la oración candidata. Por desgracia, los sistemas de traducción automática pueden generar oraciones indadecuadas pero con un gran puntaje en precisión. En el ejemplo 1 se ve un ejemplo de una oración candidata que consigue un puntaje de 7/7 en la precisión de unigramas. BLEU ataca ese problema usando una modificación de la precisión de unigramas que primero computa la cantidad máxima de unigramas presentes en cualquier oración de referencia. Luego, por cada unigrama g de la oración candidata se cuenta la cantidad de ocurrencias y se las acota por la cantidad máxima de ocurrencias en las referencias, es decir, $Count_g = \text{mín}(Count, Max\_ref\_Count)$. Luego suma estas cuentas y las divide por la cantidad de palabras en la oración candidata. En el ejemplo 1, la oración candidata consigue un puntaje 2/7 en la presición modificada de n-gramas.

Ejemplo 1.
Candidate: the the the the the the the.
Reference1: The cat is on the mat.
Reference2: There is a cat on the mat.

$$Mod\_unigram\_precision = \frac{\sum_{g \in Gen\_Sentence} Count_g}{\#Gen\_Sentence} \tag{5.1}$$

La precisión modificada de n-gramas funciona igual para cualquier n. Primero se cuentan todos los n-gramas de la oración candidata y las oraciones de referencia. Luego se acotan por la cantidad de ocurrencias en las referencias, y se suman y se dividen por la cantidad de n-gramas de la oración candidata. Esta modificación de la precisión de n-gramas permite capturar dos aspectos de la traducción: adecuación

y fluidez. Una oración usando las mismas palabras (unigramas) que las referencias tiende a satisfacer adecuación, mientras que correspondencias de n-gramas más grandes cuentan para la fluidez.

Dado que la precisión de los n-gramas decae casi exponencial a medida que n es más grande, BLEU usa el promedio logarítmico con pesos uniformes, que es equivalente a usar la media geométrica de las modificaciones de precisión de n-gramas.

Las oraciones más largas que la referencia ya son penalizadas por la presición modificada de n-gramas. Para lidiar con oraciones más cortas que las referencias se usa un factor de penalización a la brevedad. Con este factor, las oraciones candidatas deben coincidir en largo, elección de palabras y orden de las palabras.

Finalmente, para computar BLEU primero se computa el largo efectivo de las referencias en el corpus de test, r, sumando las mejores coincidencias de largo (largo de oraciones que tienen el mismo largo que alguna oración de referencia) para cada oración candidata del corpus. Se computan el promedio geométrico de las precisiones modificadas de n-gramas, usando n-gramas hasta el largo N y los pesos positivos w_n, tales que sumen a 1.

Luego, siendo c el largo de la oración candidata y r el el largo efectivo de las referencias del corpus, se computa la penalidad de brevedad.

$$BP = \begin{cases} 1 & \text{si } c{>}r \\ e^{(1-r/c)} & \text{si } c \leq r \end{cases} \tag{5.2}$$

Finalmente,

$$BLEU = BP * \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{5.3}$$

5.4.2. Perplexity

La *Perplexity* es una medida de qué tan bien una distribución de probabilidad o un modelo probabilístico predice una muestra. Una baja *perplexity* indica que la distribución de probabilidad o modelo probabilístico es bueno prediciendo la muestra.

Se puede suponer que una muestra de entrenamiento fue extraída de un modelo de distribución de probabilidad desconocida p. Dado un modelo probabilísti-

co propuesto q, se puede evaluar qué tan bien q predice una muestra de prueba $x_1, x_2, ..., x_N$, también extraída de p.

$$b^{-\frac{1}{N}\sum_{i=1}^{N} log_b q(x_i)} \tag{5.4}$$

donde b es usualmente 2. Un buen modelo q de la distribución desconocida p asignará mayor probabilidad $q(x_i)$ a los eventos de prueba, por lo tanto teniendo menos *perplexity*. El exponente puede ser pensado como la *cross-entropy*,

$$H(\tilde{p}, q) = -\sum_{x} \tilde{p}(x) \log_2 q(x) \tag{5.5}$$

donde $\tilde{p}$ denota la distribución empírica de la muestra de prueba (es decir, $\tilde{p}(x) = n/N$ si x apareció n veces en la muestra de prueba de tamaño N.

5.4.3. SARI

SARI es una métrica específica de para simplificación de texto que compara la salida del sistema con las oraciones de referencia y con la oración original (**S**ystem output **A**gainst **R**eferences and against **I**nput) (Xu, Napoles, Pavlick, Chen, and Callison-Burch 2016). Mide explícitamente qué tan bueno es agregar, eliminar o mantener una palabra. Dado un n-grama O generado por el modelo, la entrada I y una lista de referencias R, por cada acción (agregar, eliminar o mantener) se definen la precisión y *recall* de la operación agregar para el n-grama n de la siguiente forma:

$$p_{add}(n) = \frac{\sum_{g \in O} \text{mín}\left(\#_g(O \cap \bar{I}), \#_g(R)\right)}{\sum_{g \in O} \#_g(O \cap \bar{I})} \tag{5.6}$$

$$r_{add}(n) = \frac{\sum_{g \in O} \text{mín}\left(\#_g(O \cap \bar{I}), \#_g(R)\right)}{\sum_{g \in O} \#_g(R \cap \bar{I})} \tag{5.7}$$

Donde $\#_g$ es la indicadora de si la palabra g está en el conjunto dado y

$$\#_g(O \cap \bar{I}) = \text{máx}\left(\#_g(O) - \#_g(I), 0\right)$$

$$\#_g(R \cap \bar{I}) = \text{máx}\left(\#_g(R) - \#_g(I), 0\right)$$

Cuando se trabaja con múltiples referencias, es importante saber cuándo una palabra ha estado presente en más de una referencia. Sea R' el conjunto de las

palabras pesadas por ocurrencia, es decir, si la palabra p se encuentra en k oraciones sobre un total de r, luego su cuenta es pesada por k/r. Con esto en cuenta, se computa la precisión y *recall* de la siguiente forma:

$$p_{keep}(n) = \frac{\sum_{g \in I} \text{mín}\left(\#_g(I \cap O), \#_g(I \cap R')\right)}{\sum_{g \in I} \#_g(I \cap O)} \tag{5.8}$$

$$r_{keep}(n) = \frac{\sum_{g \in I} \text{mín}\left(\#_g(I \cap O), \#_g(I \cap R')\right)}{\sum_{g \in I} \#_g(I \cap R')} \tag{5.9}$$

Donde

$$\#_g(I \cap O) = \text{mín}\left(\#_g(I), \#_g(O)\right)$$

$$\#_g(I \cap R') = \text{mín}\left(\#_g(I), \#_g(R)/r\right)$$

Finalmente para la operación de eliminar se usa solo la precisión, dado que eliminar una palabra afecta mucho a la lectura. La precisión es usada para denotar la suficiencia de las eliminaciones. También se tiene en cuenta la ocurrencia de las distintas referencias, para denotar que existen *eliminaciones no requeridas*, es decir, que podrían o no realizarse:

$$p_{del}(n) = \frac{\sum_{g \in I} \text{mín}\left(\#_g(I \cap \bar{O}), \#_g(I \cap \bar{R}')\right)}{\sum_{g \in I} \#_g(I \cap \bar{O})} \tag{5.10}$$

Donde

$$\#_g(I \cap \bar{O}) = \text{máx}(\#_g(I) - \#_g(O), 0)$$

$$\#_g(I \cap \bar{R}') = \text{máx}(\#_g(I) - \#_g(R)/r, 0)$$

Finalmente, el puntaje de SARI para el n-grama n se calcula de la siguiente forma:

$$SARI = d_1 F_{add} + d_2 F_{keep} + d_3 P_{del} \tag{5.11}$$

Donde $d_1 = d_2 = d_3 = 1/3$ y

$$P_{operation} = \frac{1}{k} \sum_{n=[1,\ldots,k]} p_{operation}(n)$$

$$R_{operation} = \frac{1}{k} \sum_{n=[1,\ldots,k]} r_{operation}(n)$$

$$F_{operation} = \frac{2 \times P_{operation} \times R_{operation}}{P_{operation} + R_{operation}}$$
$$operation \in del, keep, add$$

donde k es la longitud del n-grama más largo (en caso de este trabajo, te emplean hasta tetra-gramas).

5.4.4. FKGL

Flesch-Kincaid grade level (FKGL) es una métrica de simplicidad de texto que, dado un texto, devuelve un número que corresponde al nivel del sistema educativo de EEUU necesario para complenderlo. Sin embargo, este número devuelto es relevante cuando la fórmula devuelve un valor mayor a 10. FKGL se calcula de la siguiente forma:

$$0{,}39 \left(\frac{\text{total de palabras}}{\text{total de oraciones}} \right) + 11{,}8 \left(\frac{\text{total de sílabas}}{\text{total de palabras}} \right) - 15{,}59 \tag{5.12}$$

La fórmula hace énfasis en el largo de la oración por sobre el largo y, por su construcción, no tiene cota superior.

Cada una de estas métricas tienen un rol en el proceso de entrenamiento y diagnóstico del modelo. Mientras que la *Perplexity* del modelo provee información de qué tan bien se están prediciendo las referencias, BLEU da una métrica de calidad de texto con respecto a la misma. FKGL se emplea para tener una intuición de la simplicidad del texto generado. SARI, por su lado, es la métrica que se usa para el cómputo de las recompensas de las simplificaciones y aporta una visión de la calidad de las simplificaciones. Finalmente se reportan las cuatro métricas para evaluar desempeño en los conjuntos de validación y evaluación. Cabe aclarar que si bien BLEU y SARI están definidas en el intervalo $[0, 1]$, son reportadas multiplicadas por 100 para facilitar su lectura.

5.5. Baseline

Como punto de partida, se entrenaron modelos NMT con los datasets Wikilarge y Wikismall. Se replica la configuración base empleada en (Zhang and Lapata 2017);

LSTMs de 256 unidades, *dropout* de 0,2, *batch size* de 128 y optimizador ADAM con *learning rate* de $1e^{-3}$. El resto de los hiperparámetros fueron utilizados con los valores por defecto de la librería *nmt*. Se emplearon dos variantes: con atención y sin atención. Se entrenaron los modelos durante 18 épocas sobre ambos Wikilarge y Wikismall.

Se probó entrenar estos mismos modelos con un optimizador más simple, *Stochastic Gradient Descent*, pero estos experimentos no lograron converger.

Para interpretar las métricas de evaluación automática, debe tenerse en cuenta que, a mayor puntaje de BLEU y SARI, se tiene mejor desempeño, mientras que a mayor puntaje de FKGL, menor es el desempeño.

Los modelos * son los presentados en (Zhang and Lapata 2017) y las métricas reportadas fueron extraídas del mismo paper. Se llamará a los distintos modelos con el formato <modelo>-<dataset>, donde WL corresponde a Wikilarge y WS a wikismall.

Resultados Wikismall	BLEU	FKGL	SARI	PPL
EncDecA*	47,93	11,35	13,71	-
DRESS (sin LS)*	34,53	7,48	27,48	-
NMT-WS	7,337	9,4	24,250	36,47
NMT-WS + Atención	41,38	12,05	37,13	7,989

Cuadro 5.2: Evaluación de los sistemas *baseline* en el corpus Wikismall.

El cuadro 5.2 muestra que, en el dataset Wikismall, el modelo NMT-WS tiene pésimos puntajes en todas las métricas excepto por FKGL. La medida de *Perplexity* da una intuición de que el modelo está fallando en aprender a predecir la distribución de las oraciones de referencia. Por otro lado, la evaluación del modelo NMT-WS + Atención dio resultados bastante cercanos al modelo base y llegando a superar a todos los modelos en la métrica de SARI.

SARI da una intuición del porcentaje de información que se conserva en la simplificación con respecto a la oración de entrada, y que está validada por la referencia. Se recompensan las adiciones de palabras que no estén en la entrada que se encuentre en la referencia, la conservación de palabras que se encuentren tanto en la entrada y la referencia y la suficiencia de las eliminaciones (eliminar palabras de más es altamente penalizado).

Por otro lado, la métrica de FKGL mide la simplicidad del texto con respecto a su cantidad de oraciones y sílabas por palabra.

Resultados Wikilarge	BLEU	FKGL	SARI	PPL
EncDecA*	88,85	8,41	35,66	-
DRESS (sin LS)*	77,18	6,58	37,08	-
NMT-WL	9,749	9,45	21,78	24,73
NMT-WL + Atención	52,21	11,87	34,98	5,657

Cuadro 5.3: Evaluación automática en Wikilarge.

Por otro lado, en el cuadro 5.3 se puede apreciar que el problema adquiere complejidad sobre el dataset de Wikilarge. El modelo NMT-WL vuelve a fallar en todas las métricas por lo que podemos deducir que el mecanismo de atención es necesario para la tarea. El modelo NMT-WL + Atención adquiere mucho mejor desempeño en todas las evaluaciones, pero cae por debajo en todas las métricas que el modelo base EncDecA y DRESS por amplio margen.

En los siguientes ejemplos se muestra cómo se comportan los modelos baseline entrenados. En el siguiente ejemplo se puede observar un ejemplo de un alto puntaje en SARI, el cuál está dando un puntaje alto para una oración similar a la de entrada.

Ejemplo 1. Puntaje SARI: $57,931$. Modelo: NMT-WS + Attention.
Original: His father's posting lasted six years, so PERSON@1's early childhood was spent in a flat in LOCATION@1, overlooking LOCATION@2, where he often played.
Referencia: PERSON@1's early childhood was spent in a flat in LOCATION@1. It overlooked the LOCATION@3 where he often played.
NMT: His father's <unk> lasted six years, so PERSON@1's early childhood was spent in a flat in LOCATION@1, with LOCATION@2, where he often played.

Por su lado FKGL, dada una oración similar a la original da un puntaje alto, dada que las oraciones originales ya eran complejas desde un principio.

Ejemplo 2. Puntaje FKGL: $20,9$. Modelo: NMT+WS + Attention.
Original: Unlike the rest of the violin family, the double bass still reflects influence and can be considered partly derived from the viol family of instruments, in particular the violone, the bass member of the viol family.

Referencia: The double bass also has influences from the viol family.
NMT: Unlike the rest of the violin family, the double bass still reflects influence and can be considered partly derived from the <unk> family of instruments, in particular the <unk>, the bass member of the viol family.

Si bien no se pudo medir tiempos de cómputo en un entorno controlado, el promedio de tiempo de entrenamiento de estos modelos ronda los 35 minutos para modelos entrenados en Wikismall y 215 minutos para modelos entrenados sobre Wikilarge.

5.5.1. Ejemplos

Durante el proceso de entrenamiento es importante ver la calidad de oraciones generadas por el modelo. Además de la evaluación automática, es importante corroborar que las oraciones mantengan una estructura sintáctica correcta y que, efectivamente, son simplificaciones de la oración original.

Ejemplo 3. BLEU: 87,01; FKGL:13,4; SARI: 25,84. Modelo: NMT-WL + Attention.
Original: He was born in LOCATION@1 and studied at the école Normale Supérieure in LOCATION@2, where he eventually became Professor of Philosophy.
Referencia: He was born in LOCATION@1 and studied at the école Normale Supérieure in LOCATION@2. Eventually, he became Professor of Philosophy there.
NMT: He was born in LOCATION@1 and studied at the école <unk> Supérieure in LOCATION@2, where he eventually became Professor of Philosophy.

Estos son ejemplos del comportamiento de los modelos neuronales de traducción automática clásicos en el campo de simplificación. La oración generada por el modelo es una copia de la oración original con leves diferencias. Si bien, la oración de referencia es bastante similar, no son pocos los ejemplos de este tipo de oraciones.

Ejemplo 4. BLEU: 65,77; FKGL:14,2; SARI: 32,29. Modelo: NMT-WL + Attention.
Original: As recently as NUMBER@1, the ORGANIZATION@1 -LRB- WHO -RRB- estimated that NUMBER@2 million people contracted the disease and that two million died in that year.

Referencia: Even in NUMBER@1, about NUMBER@2 million people caught the disease, and about two million people died of it, according to the ORGANIZATION@1 -LRB- WHO -RRB-.
NMT: As recently as NUMBER@1, the ORGANIZATION@1 -LRB- <unk> -RRB- estimated that NUMBER@2 million people signed the disease and that two million died in that year.

Ejemplo 5. BLEU: 66,24; FKGL:13; SARI: 57,93. Modelo: NMT-WS + Attention.
Original: His father's posting lasted six years, so PERSON@1's early childhood was spent in a flat in LOCATION@1, overlooking LOCATION@2, where he often played.
Referencia: PERSON@1's early childhood was spent in a flat in LOCATION@1. It overlooked the LOCATION@3 where he often played.
NMT: His father's <unk> lasted six years, so PERSON@1's early childhood was spent in a flat in LOCATION@1, with LOCATION@2, where he often played.

Otro problema observable en los ejemplos es la gran cantidad de "<unk>". El siguiente es un ejemplo de dicho comportamiento.

Ejemplo 6. BLEU: 83,09; FKGL:10,3; SARI: 53,46. Modelo: NMT-WS + Attention.
Original: Colorless or light-colored marbles are a very pure source of calcium carbonate, which is used in a wide variety of industries.
Referencia: Colorless marbles are a very pure source of calcium carbonate, which is used in a wide variety of industries.
NMT: <unk> or <unk> <unk> are a very pure source of calcium carbonate , which is used in a wide variety of industries.

El modelo falla en capturar el significado de ciertas palabras y en consecuencia completa con estos marcadores. Una hipótesis es que este es un punto en el cual un esquema de aprendizaje por refuerzos puede ayudar al entrenamiento: siendo las palabras las acciones que toma el agente se puede penalizar aún más el uso de "<unk>" en las oraciones generadas y aproximar alguna otra palabra en el vocabulario en su lugar.

5.6. Entrenamiento del modelo

La segunda etapa del entrenamiento parte de un modelo NMT previamente entrenado para realizar las simplificaciones a través del aprendizaje supervisado. Como fue descripto en la sección 5.3, se utiliza un modelo basado en A2C que cumple que tanto el agente como la *value function* comparten todos los parámetros de la red excepto por las capas de salida (Mnih et al. 2016). Dado los resultados expuestos en 5.5, no emplear atención lleva a que el modelo sea incapaz de generar oraciones. Por lo tanto, todos los entrenamientos empleando A2C fueron llevados a cabo con los modelos NMT-WS + Atención y NMT-WL + Atención.

Este nuevo modelo es entrenado usando una nueva función de pérdida, el optimizador RMSProp y agregando los hiperparámetros para A2C con los valores: PG Coef= 1, VF coef= 0,5, Ent Coef= 0,01 y el Factor de Descuento= 0,99. Los parámetros de la red son inicializados con los pesos entrenados en la etapa anterior.

En esta etapa de experimentación, por cuestiones de tiempo de cómputo se probaron cambios y corrieron experimentos usando los modelos entrenados sobre Wikismall. Los tiempos de cómputo de 9 épocas que se corrieron empleando el modelo de A2C rondaron los 70 minutos, sin tener en cuenta el tiempo de entrenamiento en la etapa anterior. El modelo completo ronda los 105 minutos de tiempo de cómputo.

A continuación se presentan problemas que se afrontaron en la implementación y entrenamiento del modelo utilizando A2C.

5.6.1. Divergencia del modelo

El primer entrenamiento empleando A2C se llevó a cabo con el modelo NMT-WS + Atención. El modelo divergía en esta etapa. La función de pérdida tendía a $-\infty$ debido a que la función de pérdida del actor tendía a $-\infty$. La función de pérdida del actor es calculada en base a la *Advantage*, es decir, $A(s_t, a_t; \theta, \theta_v) = R_t - V(s_t; \theta_v)$. Inspeccionando las recompensas se encontró que las mismas estaban oscilando entre 0,33 y 0,41, pero la *value function* estaba tendiendo a ∞.

Este comportamiento se explica con el hecho de que el optimizador, tratando de minimizar la función de pérdida, hace crecer a ∞ la *value function*, de forma la *Advantage* tienda a $-\infty$ y así minimizar la función de pérdida del crític.

Se probó aumentar el el valor de VF Coef y reducir el de PG Coef, pero esto solo atrasaba el punto en el que la divergencia era notable, sin poder efectivamente

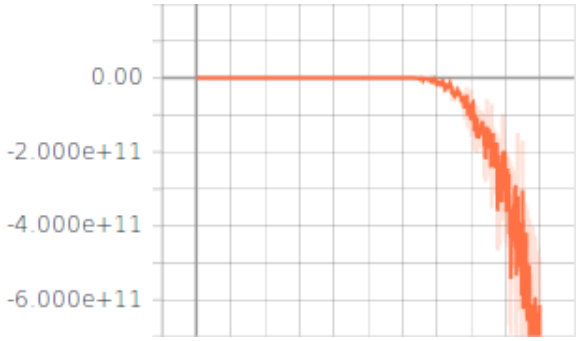

Figura 5.3: Función de pérdida del modelo empleando A2C. El primer tramo recto corresponde a la función de pérdida empleando el modelo NMT, la curva que desciende a $-\infty$ corresponde a luego de comenzar a entrenar con A2C.

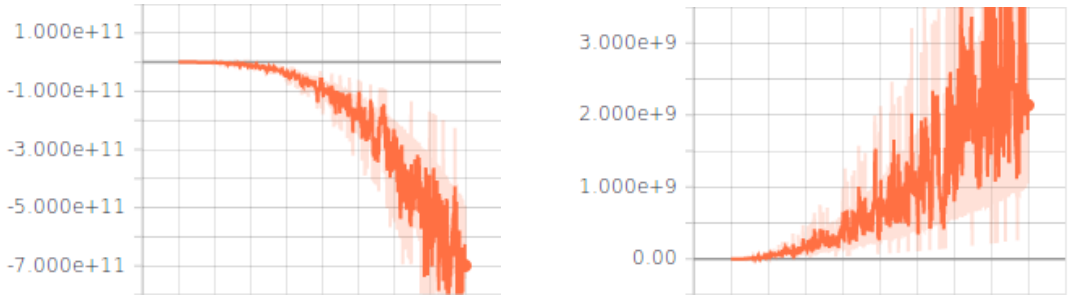

Figura 5.4: Gráficas de las funciones de pérdida del actor (izqueirda) y el critic (derecha) en el primer experimento.

solucionar el problema.

La causa de esto es que el algoritmo de *backpropagation* estaba ajustando los pesos de los *labels* (es decir, las palabras muestreadas de la distribución de probabilidad saliente del *decoder*), las recompensas y la *advantaje*. Quitando esos tensores del cálculo de los gradientes el modelo comenzó a tener un comportamiento más acercado a lo esperable.

5.6.2. El modelo genera palabras aleatorias

Luego de corregir el cálculo de los gradientes hubo un nuevo problema: el modelo comenzó a generar palabras aleatorias luego de la primera época de entrenamiento

empleando A2C, a pesar de que las funciones de pérdida estaban siendo minimizadas. La *value function* pudo converger a predecir los valores provenientes de las recompensas (los cuales al momento oscilaban entre $0,34$ y $0,42$), pero el actor estaba estaba divergiendo, como se puede apreciar de las gráficas 5.5.

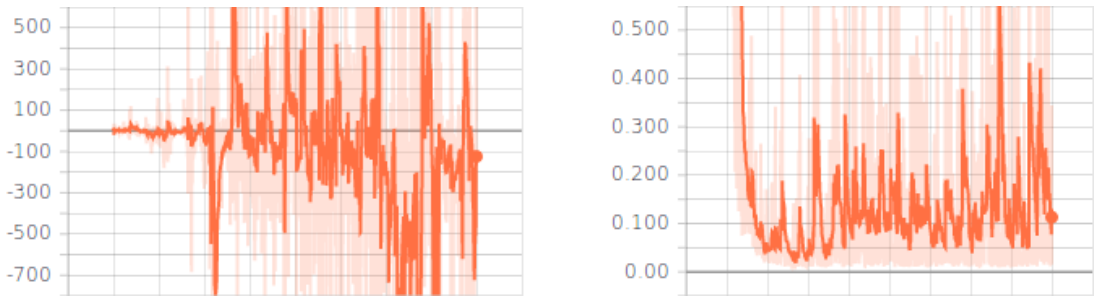

Figura 5.5: Gráficas de las funciones de pérdida del actor (izquierda) y el critic (derecha) en el segundo experimento.

A partir de las gráficas de las distribuciones de la *advantage* y la *value function* expuestas en la figura 5.6, se puede ver que estas no estaban generando la explosión en los gradientes que llevaban al modelo a divergir. Por lo tanto, el cálculo de $\log \pi(a_t|s_t;\theta)$ es lo que está llevando la función de pérdida a divergir.

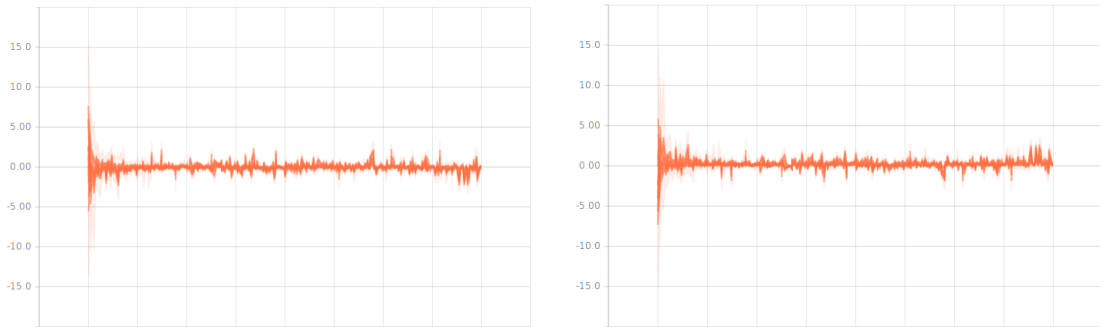

Figura 5.6: Distribucion de *Advantage* (izquierda) y *Value Fuction* (derecha) en el experimento 2.

El cálculo de $\log \pi(a_t|s_t;\theta)$ es llevado a cabo usando los logits salientes del *decoder*. Estos son la salida del *decoder* luego de la capa densa de proyección. Para diagnosticar si el problema estaba en la capa densa que generaron los logits o en la

salida de la RNN de la LSTM del decoder, se graficaron los histogramas de estos tensores.

Observando las gráficas de las salidas de la RNN de la LSTM y los logits en la figura 5.7 se puede notar que, luego de cada nueva época de entrenamiento, la gráfica de la salida del *decoder* se aplasta cada ves más. Esto se corresponde con una pérdida de las representaciones internas generadas en la etapa previa del entrenamiento.

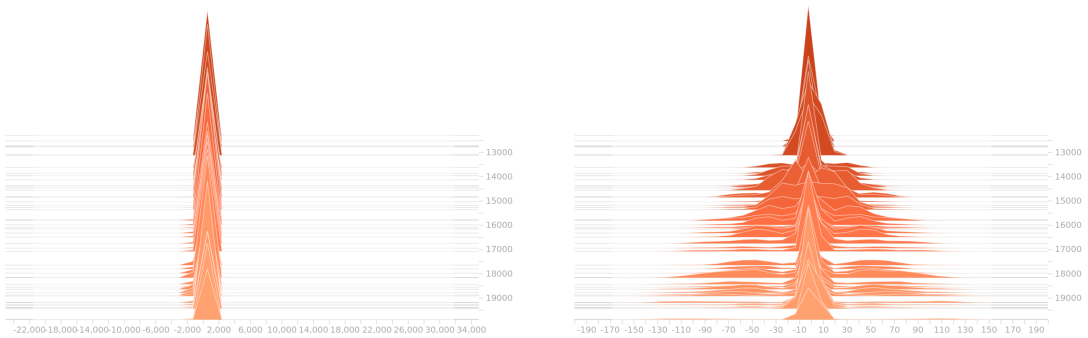

Figura 5.7: Histograma de logits (izquierda) y de la salida de la RNN del *decoder* (derecha), graficadas por su distribución a través de las épocas de entrenamiento.

Se intentó aplicar regularización en ambas capas densas: la proyección de salida del NMT y en la capa densa de la *value function*. Esto llevó a que la *value function* convergiera y pudiera predecir los valores de las recompensas con poco error cuadrático, pero aún así divergiendo en la función de pérdida del *actor*.

Una hipótesis que explica este comportamiento es el sistema de recompensas. Si se mira las métricas del modelo luego de diverger, se puede notar que BLEU tiene un valor de 0 para el conjunto de validación, mientras que SARI tiene un valor de $17,84$. En las gráficas 5.8 se puede ver cómo luego de la primera etapa de entrenamiento, los puntajes de BLEU y SARI caen en picada, pero SARI sigue conservando un puntaje mayor a 0.

Esto podría estar desorientando al modelo en el aprendizaje dado que una oración sinsentido y claramente errónea no implicaba una recompensa nula. En consecuencia, el modelo no pudo converger ni generar oraciones remotamente parecidas a una válida.

Recordando la fórmula 5.11, $SARI = d_1 F_{add} + d_2 F_{keep} + d_3 P_{del}$. Observando las recompensas individuales del cálculo de SARI en en el ejemplo 1 se puede observar

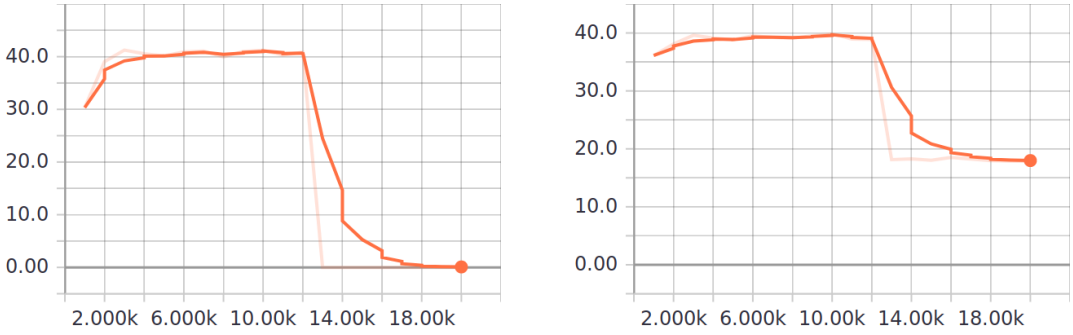

Figura 5.8: Gráfica de BLEU (izquierda) y SARI (derecha) a lo largo de las épocas de entrenamiento de ambas etapas de entrenamiento.

la razón del puntaje tan alto para una oración tan mala:

$$F_{add} = 0; F_{keep} = 0; P_{del} = 82{,}25$$

mientras que para el ejemplo 4 de la sección de ejemplos los puntajes del mismo son:

$$F_{add} = 0; F_{keep} = 36{,}47; P_{del} = 60{,}41$$

lo cual muestra que SARI está dando mucha recompensa al modelo por generar oraciones sinsentido dado que el puntaje de P_{del} es alto. Incluso con los pesos $d1, d2, d3$, la relevancia del puntaje de P_{del} es suficiente para equiparar la recompensa de una oración bien formada. Esto explica la divergencia del modelo, ya que guiado por estas recompensas el desempeño del modelo que diverge es correcto.

Ejemplo 1. Puntaje SARI: 30,71. Modelo: A2C NMT-WS + Attention.
Original: Beginning as a chorister at the collegial church of LOCATION@1, then at the cathedral of LOCATION@2, where he pursued his music studies, PERSON@1 was named chorus master at the cathedral of LOCATION@3.
Referencia: PERSON@1 became chorus master at the cathedral of LOCATION@3.
NMT: Dr Dr Dr Dr Dr Dr Dr Dr Dr Dr Dr Dr Dr Dr Dr Dr Dr Dr Dr Dr Dr Dr Dr Dr Dr Dr Dr Concerto Concerto percentage <unk> percentage Dr Concerto Concerto percentage percentage percentage percentage percentage pope pope pope pope pope Concerto Concerto Concerto down pope <unk> percentage Concerto

Concerto Concerto Concerto percentage percentage down <unk> percentage Concerto Concerto pope <unk> percentage down down pope <unk> percentage

Se espera que agregar más componentes al sistema de recompensas puede orientar mejor el aprendizaje del modelo. Medidas para “Fluidez” y “Relevancia” pueden mejorar la calidad de recompensas y empujar la relevancia de la recompensa de SARI más bajo en oraciones donde el modelo diverge.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Aportes

En este trabaja se evaluó la implementación de un modelo neuronal de traducción automática en el campo de simplificación de oraciones y su desempeño empleando Aprendizaje por Refuerzos para la misma tarea. En cuanto al modelo base, se logró obtener resultados comparables a trabajos previos en algunas métricas pero disímiles en otras. Esto indica que las métricas para este problema no están bien definidas.

Además, se provee una implementación de un modelo neuronal de traducción automática empleando el algoritmo de aprendizaje por refuerzos A2C. Cabe destacar que el esfuerzo de implementación es no despreciable para un (potencial) incremento marginal en el desempeño.

También se da un reporte de los problemas encontrados en el proceso de implementación y entrenamiento del modelo y se explican y expone anomalías encontradas en los cálculos. Se incluye, además, un reporte del impacto computacional del modelo empleando aprendizaje por refuerzos.

6.2. Conclusiones

Se ha comprobado que las técnicas de traducción automática pueden ser usadas para modelar problemas de simplificación. En particular, los modelos neuronales conservan muy bien la semántica de las oraciones. Sin embargo, los resultados obtenidos tienden a replicar la oración de entrada sin modificaciones, ya que no se

recompensa ni penaliza directamente al modelo por la simplicidad de las oraciones generadas. Como consecuencia, es necesario desarrollar técnicas nuevas para adaptar estos sistemas a la tarea de Simplificación.

Aprendizaje por Refuerzos es un paradigma de aprendizaje automático que permite mucha libertad y creatividad en el modelado de los problemas, abordando el problema desde el punto de vista de un agente que intenta maximizar una recompensa y dando libertad a que el mismo explore el espacio de soluciones posibles.

Para un problema como simplificación de oraciones, no es complicado ver la posibilidad de utilizar aprendizaje por refuerzos para explorar el espacio de simplificaciones válidas. Pero para poder aplicar satisfactoriamente un modelo de este tipo, se debe tener un sistema de recompensas sólido y que efectivamente esté recompensando el buen desempeño y penalizando el malo. Como consecuencia, se debe tener un consenso de las propiedades que constituyen las oraciones simples, antes de poder formalizarlas en un sistema de recompensas.

Un buen sistema de recompensas es el punto clave en la aplicación con éxito de aprendizaje por refuerzos a algún problema y por esto el diseño y estudio de los sistemas de recompensas sigue siendo un área de investigación activa.

6.3. Trabajo futuro

Como se ha mencionado anteriormente, existen diversas avenidas por las cuales se puede continuar esta investigación.

El avance más importante que puede hacerse es agregar otros sistemas de Recompensa, como proponen (Zhang and Lapata 2017). Esto permitirá evaluar si existe una mejora en utilizar el algoritmo de A2C, como se propone en entre trabajo, por sobre el de REINFORCE. La métrica SARI ya recompensa la "Simplicidad" de una oración con respecto a su entrada y sus referencias. Por un lado recompensas para la "Relevancia" basados en codificar las oraciones de entrada y salida con una LSTM entrenada con un *autoencoder* y calculando la distancia coseno de ayuda a mantener recompensas por mantener la semántica de la oración. Mientras que para "Fluidez" se puede aplicar la exponencial de la *perplexity* de una oración generada contra un modelo de lenguaje entrenado sobre oraciones simples usando una LSTM.

Desde un punto de vista de implementación, es necesario crear una versión vectorizada de la función SARI para reducir el impacto de calcular recompensas en GPU.

Esto ayudará a recortar tiempos de cómputo para futuros trabajos en el tema.

Si bien los conjuntos de datos utilizados son extensos y de referencia para el área, estos sistemas podrían ser entrenados con otros conjuntos de datos para simplificación que provean múltiples simplificaciones de referencia para las misma oraciones. Parte del problema que tienen los modelos neuronales de traducción automática y, en consecuencia, las aproximaciones empleando aprendizaje por refuerzos son que muchas de las simplificaciones de referencia no conservan el total de la información contenida en las entradas. Esto se solucionaría con ejemplos de simplificación dados en un contexto más amplio que una única oración.

Finalmente, lograr que el modelo converja. Una vez encontrada una función de recompensa razonable y robusta, se puede comenzar a explorar las combinaciones de hiperparámetros del modelo entrenado con A2C para encontrar la mejor configuración para el problema.

Desde otra perspectiva, estos métodos pueden ser empleados para simplificación de textos en dominios particulares. Esto trae una serie de desafíos nuevos, como el vocabulario específico, pero también simplifica el problema si el lenguaje utilizado es altamente formal. Por ejemplo, la simplificación los textos legales o técnicos, asumiendo que se puedan obtener dichos conjuntos de datos para el tema, puede tener un gran impacto al facilitar el acceso a la información a más personas.

Bibliografía

Afantenos, S., N. Asher, F. Benamara, A. Cadilhac, C. Dégremont, P. Denis, M. Guhe, S. Keizer, A. Lascarides, O. Lemon, P. Muller, S. Paul, V. Rieser, and L. Vieu (2012, September). Developing a corpus of strategic conversation in The Settlers of Catan. In *SeineDial 2012 - The 16th WORKSHOP ON THE SEMANTICS AND PRAGMATICS OF DIALOGUE*, Paris, France.

Asri, L. E., H. Schulz, S. Sharma, J. Zumer, J. Harris, E. Fine, R. Mehrotra, and K. Suleman (2017). Frames: A corpus for adding memory to goal-oriented dialogue systems. *CoRR abs/1704.00057*.

Bakker, B. (2002). Reinforcement learning with long short-term memory. In T. G. Dietterich, S. Becker, and Z. Ghahramani (Eds.), Advances in Neural Information Processing Systems 14*, pp. 1475–1482. MIT Press.*

Beata Beigman Klebanov, K. K. and D. Marcu (2004). Text simplification for information-seeking applications. In Proceedings of ODBASE, volume 3290 of Lecture Notes in Computer Science*, pp. 735–747. Agia Napa, Cyprus. Springer.*

Chandrasekar, R., C. Doran, and B. Srinivas (1996). Motivations and methods for text simplification. In Proceedings of the 16th Conference on Computational Linguistics - Volume 2*, COLING '96, Stroudsburg, PA, USA, pp. 1041–1044. Association for Computational Linguistics.*

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

Hochreiter, S. and J. Schmidhuber (1997). Long short-term memory. Neural computation 9, 1735–1780.

Jiang, Y., F. Yang, S. Zhang, and P. Stone (2018). Integrating task-motion planning with reinforcement learning for robust decision making in mobile robots. *CoRR abs/1811.08955.*

John Carroll, Guido Minnen, D. P. Y. C. S. D. J. T. (1999). Simplifying text for language-impared readers. In Proceedings of the 16th Conference of European Chapter of the ACL*, pp. 269–270.*

Karpathy, A. (2015). The unreasonable effectiveness of recurrent neural networks. `https://karpathy.github.io/2015/05/21/rnn-effectiveness/`.

Keskar, N. S., D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang (2016). On large-batch training for deep learning: Generalization gap and sharp minima. CoRR abs/1609.04836.

Lascarides, A., O. Lemon, M. Guhe, S. Keizer, H. Cuayáhuitl, I. Efstathiou, K. Engelbrecht, and M. S. Dobre (2017). Evaluating persuasion strategies and deep reinforcement learning methods for negotiation dialogue agents. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 2: Short Papers*, pp. 480–484.

Li, J., W. Monroe, A. Ritter, M. Galley, J. Gao, and D. Jurafsky (2016). Deep reinforcement learning for dialogue generation. Volume abs/1606.01541.

Luong, M., H. Pham, and C. D. Manning (2015). Effective approaches to attention-based neural machine translation. *CoRR abs/1508.04025.*

Mnih, V., A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu (2016). Asynchronous methods for deep reinforcement learning. CoRR abs/1602.01783.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller (2013). Playing atari with deep reinforcement learning. *CoRR abs/1312.5602.*

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie,

A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis (2015, February). Human-level control through deep reinforcement learning. Volume 518, pp. 529–533. Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved.

Narayan, S. and C. Gardent. Hybrid simplification using deep semantics and machine translation. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pp. 435–445.*

OpenAI (2017). Openai baselines: Acktr & a2c. `https://blog.openai.com/baselines-acktr-a2c/`.

Papineni, K., S. Roukos, T. Ward, and W.-J. Zhu (2002). Bleu: A method for automatic evaluation of machine translation. In Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, Stroudsburg, PA, USA, pp. 311–318. Association for Computational Linguistics.*

Pathak, D., P. Agrawal, A. A. Efros, and T. Darrell (2017). Curiosity-driven exploration by self-supervised prediction.

Peng, B., X. Li, L. Li, J. Gao, A. Çelikyilmaz, S. Lee, and K. Wong (2017). Composite task-completion dialogue system via hierarchical deep reinforcement learning. CoRR abs/1704.03084.

Pennington, J., R. Socher, and C. D. Manning (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543.

Siddharthan, A. (2006, Jun). Syntactic simplification and text cohesion. *Research on Language and Computation*@(1), 77–109.

Tiedemann, J. (2009). News from OPUS - A collection of multilingual parallel corpora with tools and interfaces. In N. Nicolov, K. Bontcheva, G. Angelova, and R. Mitkov (Eds.), *Recent Advances in Natural Language Processing*, Volume V, pp. 237–248. Borovets, Bulgaria: John Benjamins, Amsterdam/Philadelphia.

Vickrey, D. and D. Koller (2008). Sentence simplification for semantic role labeling. In *Proceedings of ACL-08: HLT*, pp. 344–352.

Watanabe, W. M., A. C. Junior, V. R. Uzêda, R. P. d. M. Fortes, T. A. S. Pardo, and S. M. Aluísio (2009). Facilita: Reading assistance for low-literacy readers. In *Proceedings of the 27th ACM International Conference on Design of Communication*, SIGDOC '09, New York, NY, USA, pp. 29–36. ACM.

Williams, R. J. (1992, May). Simple statistical gradient-following algorithms for connectionist reinforcement learning. Volume 8, pp. 229–256.

Woodsend, K. and M. Lapata (2011). Learning to simplify sentences with quasi-synchronous grammar and integer programming. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, Stroudsburg, PA, USA, pp. 409–420. Association for Computational Linguistics.

Wubben, S., A. van den Bosch, and E. Krahmer (2012). Sentence simplification by monolingual machine translation. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ACL '12, Stroudsburg, PA, USA, pp. 1015–1024. Association for Computational Linguistics.

Xu, W., C. Napoles, E. Pavlick, Q. Chen, and C. Callison-Burch (2016). Optimizing statistical machine translation for text simplification. *TACL 4, 401–415.*

Yu, A. W., H. Lee, and Q. V. Le (2017). Learning to skim text. Volume abs/1704.06877.

Zhang, X. and M. Lapata (2017). Sentence simplification with deep reinforcement learning. CoRR abs/1703.10931.

Zhu, Z., D. Bernhard, and I. Gurevych (2010). A monolingual tree-based translation model for sentence simplification. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pp. 1353–1361. Coling 2010 Organizing Committee.