UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Generación de datos sintéticos para traducción automática entre español y guaraní

Informe de Proyecto de Grado presentado por

Alexis Baladón, Agustín Lucas, Victoria Pardiñas

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisores

Luis Chiruzzo
Santiago Góngora

Montevideo, 8 de enero de 2024

Agradecimientos

Ante todo, queremos agradecer profundamente a nuestros seres queridos, quienes nos han brindado su incondicional apoyo a lo largo de este viaje académico.

A Marvin Agüero-Torales, por ayudarnos a evaluar la calidad de uno de nuestros corpus.

Un agradecimiento especial a Luis y a Santiago, nuestros mentores en este proyecto, por siempre alentarnos y ayudarnos a enfocar de la mejor manera nuestro trabajo, y acompañarnos a lo largo de este año de forma tan constante y con tanta voluntad.

Finalmente, a todos aquellos que de una forma u otra nos ayudaron a superar los obstáculos y avanzar en nuestro trayecto, fuera demostrando interés en una conversación, ofreciendo un consejo oportuno, brindando una palabra de aliento en los momentos difíciles, o simplemente estando presentes cuando más los necesitábamos.

Gracias.

Resumen

Este proyecto trata el problema de la traducción automática entre español y guaraní, como un caso particular de traducción automática en una lengua de escasos recursos, investigando un enfoque de aumentado de datos como posible alternativa a la escasez de texto guaraní-español.

En este contexto, nos enfocamos en la construcción de dos nuevos conjuntos de oraciones paralelas guaraní-español, obtenidos mediante el uso de gramáticas formales y la aplicación de técnicas de traducción basadas en reglas a un corpus monolingüe generado automáticamente y otro ya existente. Luego, experimentamos preentrenando modelos de traducción automática sobre estos nuevos datos, con el fin de determinar si los corpus generados mejoran el desempeño de los modelos, y evaluar la viabilidad y efectividad de esta metodología en el contexto de lenguas de escasos recursos. Hasta el momento son pocos los trabajos realizados de procesamiento de lenguaje natural para el guaraní, por lo que a su vez se busca expandir este repositorio con los nuevos conjuntos de datos y modelos entrenados.

Para esto, creamos una gramática de rasgos en español a partir de datos etiquetados sintácticamente, con la que generamos más de 200.000 frases gramaticalmente correctas en español, junto a sus árboles sintácticos. Posteriormente, implementamos un mecanismo de traducción automática basada en reglas haciendo uso de técnicas de transferencia sintáctica desde español a guaraní, generando con este un corpus paralelo a partir de las oraciones obtenidas en español, y otro al aplicar estas técnicas al corpus monolingüe de Ancora.

En cuanto a los modelos entrenados, utilizamos modelos de traducción automática neuronal, los cuales son a día de hoy el estado del arte en esta área. En particular, las arquitecturas utilizadas fueron transformer y seq2seq, las cuales fueron tratadas tanto con sus hiperparámetros por defecto como ajustados a través de métodos de búsqueda aleatoria y de grilla. Además, utilizamos métodos para evitar el sobreajuste, el problema de desvanecimiento y explosión de gradientes, y para aumentar la eficiencia computacional del entrenamiento.

Como resultado, los modelos que obtuvieron un mejor desempeño lo hicieron preentrenando con un conjunto de datos formado por la concatenación de todos los corpus que generamos además de la Biblia, lo que sugiere la viabilidad de la metodología utilizada. Nuestro mejor modelo sigue una arquitectura seq2seq multicapa con celdas GRU sobre texto tokenizado con el método de unigramas. Además, siendo preentrenado logró superar resultados del traductor de Google de español al guaraní en el subconjunto de test del corpus de Jojajovai, al igual que obtener resultados competitivos desde guaraní a español.

Palabras clave: Español, Guaraní, Aprendizaje automático, Procesamiento de lenguaje natural, Traducción automática neuronal, Transferencia sintáctica, Gramática de rasgos, Aumentado de datos, Seq2seq, Transformer, Google

Índice general

Capítulo 1

Introducción

El idioma guaraní, una de las lenguas más habladas de América del Sur, no solo es reconocido por su singularidad lingüística, sino también por su estatus como lengua oficial del Mercosur; además, sobrevive como la única lengua indígena del continente que es hablada por la mayoría no indígena de un país, Paraguay (Estigarribia, 2015). Su larga historia de contacto con el español, que data de los primeros encuentros con los colonizadores europeos del siglo XVI, ha dado lugar a una profunda influencia de este idioma sobre la lengua guaraní, derivando en variantes como el Jehe'a y, el más relevante para nuestro caso, el Jopará; este último consiste en una mezcla entre guaraní y español, muy común en el habla paraguaya. A pesar de esta larga historia compartida, hay una grave escasez de texto bilingüe y recursos de traducción entre ambos idiomas, especialmente de recursos digitales.

En este contexto, la creación de traductores automáticos efectivos se ve obstaculizada por la escasez de datos, especialmente en lenguas menos documentadas como el guaraní. Esto dificulta la aplicación de algunos métodos de traducción automática que actualmente constituyen el estado del arte, los cuales requieren de grandes volúmenes de información, particularmente para el entrenamiento de modelos neuronales, limitando su capacidad de obtener resultados precisos y contextualmente relevantes.

Por esta razón, en este proyecto nos hemos propuesto abordar esta limitación propia de las lenguas de escasos recursos mediante técnicas innovadoras de aumentado de datos. Nuestra metodología se centra en la construcción de un corpus paralelo completamente nuevo, con oraciones sintéticas basadas en información gramatical pura de ambos idiomas. Este enfoque nos ha permitido superar la escasez de datos, generando un conjunto de entrenamiento robusto que proporciona una base para el preentrenamiento de modelos de traducción automática.

Además, hemos ampliado nuestros conjuntos de datos con información obtenida de un extenso corpus existente en español, aplicando nuestro método de traducción basada en reglas sobre este, para generar un nuevo corpus bilingüe. Esta combinación nos ha proporcionado una base sólida y diversa para entrenar

y mejorar significativamente los sistemas de traducción neuronal. Nuestros experimentos muestran mejoras en la precisión y calidad de la traducción, igualando o incluso superando a los sistemas actuales de referencia en el campo.

En resumen, este proyecto presenta un enfoque innovador para superar el desafío que representa la falta de datos en la construcción de traductores automáticos para lenguas de escasos recursos. Nos enfocaremos en la realización de técnicas de aumentado de datos junto con la construcción estratégica de corpus paralelos para impulsar el desarrollo de sistemas de traducción automática más precisos y efectivos.

1.1. Motivación

El guaraní es una lengua hablada por más de 10 millones de personas en países sudamericanos como Brasil, Argentina, Bolivia y principalmente Paraguay, representando una de las lenguas indígenas más habladas del continente. De hecho, según la Encuesta Permanente de Hogares Continua 2022, aproximadamente un 70 % de los habitantes de Paraguay encuestados manifiestan hablar guaraní regularmente en el hogar, ya sea exclusivamente o en conjunción con el español. Además, el guaraní y el español son lenguas oficiales en Paraguay y conviven extensamente, por lo que puede resultar de interés desarrollar herramientas de traducción entre estos dos idiomas.

Por otro lado, el guaraní es considerado un lenguaje poco formalizado y de *escasos recursos*. De hecho, Joshi et al. (2020) lo ubican en la última categoría (*los rezagados*) de su taxonomía de lenguajes, indicando una escasez, rozando la nulidad, de recursos lingüísticos, e incluso de texto sin anotar. Esto dificulta cualquier tarea del área de procesamiento de lenguaje natural (PLN), particularmente lo referente a traducción automática entre español y guaraní. Sin embargo, también presenta la oportunidad de aplicar métodos alternativos y explorar nuevas posibilidades que se ajusten a esa realidad, donde los grandes modelos que requieren millones de oraciones para ser entrenados no resultan adecuados.

1.2. Objetivos

El proyecto tiene por objetivo construir conjuntos de datos sintéticos, en la forma de corpus paralelos generados a través del uso de gramáticas formales y un proceso de traducción basado en reglas (transferencia sintáctica), y evaluar los resultados de preentrenar modelos neuronales de traducción automática sobre estos corpus. Buscamos estudiar la viabilidad y desempeño de esta clase de métodos respecto a la traducción entre español y guaraní, y su potencial como herramienta a la hora de tratar con lenguas de escasos recursos. A su vez, obtuvimos como resultado dos nuevos corpus bilingües de diferentes características, junto a una evaluación de su calidad para ser usados como *silver standard* a la hora de entrenar un modelo de aprendizaje automático; y modelos neuronales

de traducción automática español-guaraní con métricas de su desempeño.

1.3. Estructura

En esta sección describimos la estructura que sigue nuestro documento y las temáticas tratadas en cada capítulo.

En el capítulo 2 presentamos los conceptos requeridos para comprender los siguientes capítulos. Entre ellos mencionamos conceptos fundamentales de sintaxis guaraní, procesamiento de lenguaje natural, aprendizaje automático, gramáticas formales y técnicas de traducción automática.

En el capítulo 3 explicamos la solución que diseñamos e implementamos para generar nuevos recursos para la traducción entre español y guaraní, en la forma de corpus bilingües generados automáticamente mediante un enfoque basado en reglas.

En el capítulo 4 presentamos los experimentos que realizamos sobre estos nuevos recursos y los resultados de esta evaluación.

Finalmente, en el capítulo 5 abordamos las conclusiones que surgen del proyecto al igual que áreas a mejorar y de trabajo futuro.

Capítulo 2

Marco teórico

2.1. Gramática del guaraní

El guaraní es un lenguaje especial con sus propias características. A continuación, especificaremos las más significativas para el desarrollo del proyecto.

La estructura básica gramatical del guaraní es Sujeto + Verbo + Complemento. Puede ser alterado, generalmente con la forma Complemento + Verbo + Sujeto, pero a fines de lograr una gramática sencilla nos concentramos en la primer forma.

Como ejemplo, observemos las siguientes oraciones.

- **Ore romomarandu** - Nosotros participamos
- **Juan ojogua pe kamisa** - Juan compra esa camisa

En el primer ejemplo vemos el formato sujeto, "Ore" ("nosotros") + verbo, "romomarandu" ("participamos"). En el segundo, tenemos el "Juan" como el sujeto, "ojogua" ("compra") es el verbo, y "pe kamisa" ("esa camisa") corresponde al complemento.

El alfabeto guaraní, denominado "achegety", consta de 33 letras, donde 12 son vocales (la letra "Y" se considera vocal gutural), 15 son consonantes simples y 6 consonantes digramas (es decir, compuestas por dos consonantes, como por ejemplo "ch", "mb", entre otros)[1]. Entre las consonantes, cabe destacar la consonante "puso" ('), la cual causa un efecto de interrupción en el ritmo de la palabra

2.1.1. Nasalidad

Dependiendo de las tildes y sus consonantes, las palabras se pueden clasificar en nasales u orales, siendo las nasales aquellas que contengan al menos uno de los siguientes grafemas: ã, ẽ, ĩ, õ, ũ, ỹ, g̃, m, n, ñ, mb, nd, ng, nt.

[1] https://logos.it/corso_gn/01.htm

Esta característica es importante, debido a que se debe tener en cuenta para definir las posposiciones a utilizar en caso de conjugaciones de verbos, los determinantes que preceden al sustantivo, entre otros; existen variantes diferentes para muchas de estas partículas, dependiendo de la nasalidad del vocablo al que acompañan.

2.1.2. Pronombres

Los pronombres personales en guaraní son ocho (Academia de la Lengua Guaraní, 2018), cuenta con los mismos que en español, pero con una variación para los plurales de la primera y tercera persona.

En cuanto a la primera persona del plural, hay dos formas de hablar de "nosotros"; una corresponde a un nosotros inclusivo, "ñandé", es decir, que el oyente se incluye en la oración, y la otra, "oré", corresponde a un nosotros exclusivo, donde el que habla cuenta un hecho en el que los receptores no están incluidos. Esta diferencia influye, por ejemplo, en la conjugación de verbos, ya que las posposiciones varían para cada pronombre.

El otro caso, para la tercera persona del plural, son los pronombres "ha'ekuéra" e "hikúai", donde la diferencia entre ambos es que el primero se utiliza de forma anterior al verbo y el otro siempre es posterior.

2.1.3. Verbos

Al igual que en español, los verbos se conjugan dependiendo del número y persona correspondiente, con la diferencia de que en guaraní tienen la siguiente estructura: persona – raíz (verbo sin conjugar) – tiempo.

En el caso de los verbos regulares, se agrega una preposición al comienzo del verbo, y luego una posposición indicando el tiempo (ninguna en caso de ser en presente). Por ejemplo, el verbo "karu" es comer, la preposición "a" indica primera persona del singular (yo), por lo que "akaru" es "yo como"; la posposición "ta" indica tiempo futuro, por lo tanto, "akaruta" corresponde a "comeré".

Tiempo	**Posposición**
Presente	-
Pretérito imperfecto	Mi
Futuro	Ta
Pretérito simple	Kuri

Tabla 2.1: Tiempos de verbos

A su vez, en el guaraní existen los verbos irregulares, donde cambia ligeramente la conjugación de persona y número, y los verbos defectivos, que son verbos colectivos, es decir, no existe conjugación para el número singular (Academia de la Lengua Guaraní, 2018). La cantidad de estos podría dar la ilusión de que no tienen gran importancia (en total son 6 y 3 respectivamente), pero sus sig-

nificados indican lo contrario, ya que son verbos de uso muy frecuente, como se puede observar en la tabla 2.2. (Olivera, 2022)

Verbo Irregular	Significado	Verbo Defectivo	Significado
a	caer	ko'i	concurrir
u	comer	je'ói	ir juntos
y'u	beber	hua'ĩ	concurrir en masa
ju	venir	-	-
ha	ir	-	-
e	decir	-	-

Tabla 2.2: Verbos irregulares y defectivos

Otro tipo de verbos con conjugación diferente son los verbos aireales. Estos se comportan de forma similar a los regulares, con la diferencia de que al ser conjugados interponen la partícula "i" entre la de número y persona y el verbo.

Por último, por más que no los utilizamos para este proyecto debido a su complejidad, cabe mencionar otro tipo de verbos. Estos son los llamados verbos chendales o pronominales, los cuales son el correspondiente a los verbos ser y estar en español. La particularidad de estos es su composición, no consisten en un palabra como en español, sino que existen vocablos que cumplen doble funcionalidad, que dependiendo del tipo de vocablo que los rodea es la función que cumplen.

Un ejemplo de estos es la palabra "porã". Si esta sigue a un sustantivo, oficia de adjetivo calificativo, por ejemplo si decimos "oga porã" estamos diciendo la casa bella. En cambio, si a este vocablo le anteponemos un determinante posesivo-atributivo como puede ser "che", correspondiente a la primera persona del singular, obtenemos la palabra "cheporã", la cual traduce como "soy bello". De esta forma, los verbos chendales se componen de determinantes posesivo-atributivos seguidos de un adjetivo.

2.1.4. Negación

A diferencia del español, en guaraní el verbo negado es una sola palabra. Es decir, se toma el verbo conjugado y se le anexa el circunfijo correspondiente. En su forma más genérica, este circunfijo tiene la forma nd + ... + i, con variaciones dependiendo de si la palabra es nasal, comienza con consonante, o finaliza con i. Para negar el verbo, se inserta la primera parte ("nd") del circunfijo de forma anterior al verbo, y la segunda parte ("i") antes de la posposición correspondiente al tiempo. Retomando el ejemplo anterior donde se explica el verbo *akarumi*, su forma negada, no comeré, se representa de la siguiente manera: **nd** + **a** + **karu** + **i** + **mi**, *ndakaruimi*.

2.1.5. Sustantivos y adjetivos

Tanto los adjetivos como los sustantivos cuentan con sus formas singular y plural, donde el plural se conforma anexando la posposición -kuera o -nguera según su nasalidad.

En guaraní no existe el género gramatical, como sí sucede en español, donde los sustantivos se clasifican como masculinos o femeninos, y existen versiones masculinas y femeninas de los determinantes y adjetivos. En su lugar, en guaraní hay otras formas de especificar el género de un ser vivo cuando es relevante, como en el caso de "kavara" ("cabra"), que añadiendo el vocablo "kuña" obtenemos la versión en femenino, "kavara kuña" ("cabra hembra"). Sin embargo, luego de realizar investigaciones comparando el corpus de Jojajovai, y contactando hablantes nativos, se llegó a la conclusión de que esto no es lo más usual, sino que en la mayoría de los casos se habla sin género específico, por lo que decidimos no integrar estas formas en nuestro proyecto.

2.1.6. Determinantes

En guaraní no existe la categoría gramatical "determinante", sino que esta se compone de otros tipos de pronombres, diferentes del pronombre personal, como por ejemplo pronombres demostrativos ("aquel", "este", "ese"), y pronombres posesivos ("mi", "sus", "nuestros"). A lo largo del proyecto, a estos se los denominará determinantes, de forma de no confundirlos con los pronombres personales directos ("yo", "tú", etc.).

En cuanto a los determinantes posesivos, es decir, aquellos que indican apropiación del sustantivo o nombre ("mi", "su", etc.), en guaraní son iguales a los pronombres personales, en primera y segunda persona, y tercera del plural.

Sin embargo, cuenta con algunas variaciones dependiendo de la palabra en el caso de la tercera persona del singular. En estos casos se usan "índices" de posesión.

- En caso de que el sustantivo comience con la letra "T" o la letra "O" (llamados sustantivos triformes), este será sustituido por la letra "H". Por ejemplo, si queremos decir "su cara", como la traducción de "cara" es "tova", la frase toma la forma "**h**ova".

- Si el sustantivo es oral, e inicia con vocal y termina con vocal tónica, se antepone el vocablo "ij". En el caso del sustantivo "apyka" ("asiento"), la traducción de "su asiento" es "**ij**apyka".

- Sustantivo nasal que inicia con vocal y termina con vocal tónica, se antepone el vocablo "iñ". Si queremos decir "su cabeza", siendo que "cabeza" se traduce al guaraní como "akã", obtenemos la traducción "**iñ**akã".

- Con sustantivo que inicia con consonantes que no sean sustantivos triformes, se antepone el vocablo "i". Por ejemplo, teniendo el sustantivo "kuatia" ("papel"), obtenemos la frase "**i**kuatia".

- Con sustantivos que inician con vocal tónica y terminan con vocal átona, se antepone el vocablo "hi", incluyendo el apóstrofe de forma anterior al sustantivo. Si tenemos el sustantivo "áva" ("cabello"), y queremos decir "su cabello", la traducción resulta en "**hi**áva".

2.1.7. Posposiciones

En español se cuenta con las preposiciones, que son palabras invariables que sirven de nexo entre las diferentes partes de una oración y presentan complementos. Sin embargo, en guaraní se utilizan posposiciones. Las hay de dos tipos: hay posposiciones monosilábicas, como "e" (a, en) y "gua" (de), que van unidas a la raíz; y hay posposiciones polisilábicas, como por ejemplo "guive" (desde), que van separadas de la raíz.

Ejemplificando, si se quiere decir "voy **a** casa", la traducción de la preposición "a" al guaraní, resulta en la posposición "pe", la cual va unida al sustantivo. Por lo tanto, obtenemos una frase con la forma "Ahátama óga**pe**". Si tenemos la frase "vino con su madre", la preposición en español "con", se traduce al guaraní como "ndive", el cual es polisilábico (va separado de la ráiz), por lo que obtenemos como resultado de la traducción completa, la frase "Ou isy **ndive**".

2.2. Aprendizaje automático

Nos referimos a aprendizaje automático como una subdisciplina de la inteligencia artificial que estudia la construcción de sistemas que aprenden a realizar una tarea de forma automática a partir de un conjunto de datos (Mitchell, 1997) (Jurafsky y Martin, 2009).

2.2.1. Aprendizaje supervisado

Dentro de las distintas categorías de aprendizaje automático, una de las más comunes es el aprendizaje supervisado, donde el algoritmo aprende a realizar una tarea a partir de un conjunto de datos etiquetados, o sea, un conjunto que contenga diversas entradas x asociadas a su correspondiente salida esperada y (Jurafsky y Martin, 2009).

En este caso, las entradas y salidas de la tarea a realizar pueden representar cualquier tipo de elemento mientras sea interpretable por una máquina. Por ejemplo, una tarea podría basarse en generar respuestas (salida) a preguntas de un usuario (entrada), o dada una imagen de una persona (entrada) aprender a determinar su edad (salida). En ambos casos es necesario el entrenamiento de un modelo sobre datos etiquetados para ser considerado supervisado.

Formalmente, una tarea de aprendizaje supervisado puede verse como la tarea de aprender una función $h : X \to Y$ que aproxime una función $f : X \to Y$ dado un conjunto de datos D de la forma $\{(x_i, f(x_i)) \mid x_i \in X\}_{i \in 1...|D|}$. De aquí en adelante llamaremos a $h(x)$ como modelo.

2.2.2. Sobreajuste y desajuste

Una vez definida la tarea a realizar y obtenidos los datos necesarios para que el sistema aprenda de ellos, será necesario evaluar si este es capaz de hacer uso de lo aprendido más allá de los datos utilizados para su entrenamiento. Para ello, es común dividir el conjunto D en un conjunto de entrenamiento D_{train} y uno de evaluación (o test) D_{test}, donde se entrenará al modelo únicamente en D_{train} y luego se evaluará su desempeño en el conjunto hasta ahora no utilizado D_{test} (Mitchell, 1997) (Abu-Mostafa et al., 2012).

Dado este esquema de entrenamiento y evaluación, el objetivo principal para lograr que el modelo aprenda será aumentar su desempeño en el conjunto de entrenamiento D_{train} y en simultáneo fuera de él, lo cual implica que el modelo puede aprender patrones generalizables más allá de los datos de entrenamiento.

Se define entonces sobreajuste, u *overfitting* en inglés, como el fenómeno que ocurre cuando el proceso de entrenamiento sobre D_{train} ya no supone una mejora de desempeño fuera de él, y desajuste, o *underfitting*, a aquel donde el modelo no logra obtener un alto desempeño en D_{train} (Mitchell, 1997) (Abu-Mostafa et al., 2012) (Goodfellow et al., 2016).

Es posible observar una representación de ambos fenómenos en la figura 2.1.

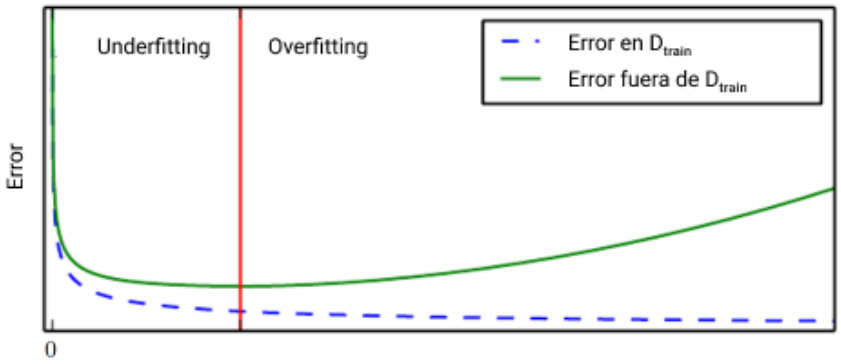

Figura 2.1: Representación del fenómeno de sobreajuste y desajuste adaptada de Goodfellow et al. (2016), donde se observa cómo aumentar el desempeño inicial del modelo en el conjunto de entrenamiento permite una mejora fuera de él hasta que el modelo comienza a sobreajustar.

2.2.3. Metodología para aprendizaje

Usualmente existen distintos métodos para resolver un mismo problema de aprendizaje automático. Definiremos entonces hiperparámetro, que es un conjunto de parámetros que modifican el funcionamiento de un modelo (Jurafsky y Martin, 2009) (Goodfellow et al., 2016). Formalmente, es posible ver a los hiper-

parámetros como parámetros p_i que alteran la salida de la anterior función $h(x)$, que ahora podría expresarse como $h(x; p_1, \ldots, p_n)$. Como ejemplo, un hiperparámetro podría ser el tiempo en segundos que se entrenará un modelo (asumiendo que un mismo tiempo equivale a un mismo desempeño), o el algoritmo utilizado para aprender $h(x)$. Es importante diferenciar a los hiperparámetros de los parámetros aprendidos por un modelo, ya que nos referimos a hiperparámetro solo a aquellos parámetros que modifica el diseñador del algoritmo.

Supongamos ahora que queremos determinar el valor óptimo de un hiperparámetro para mejorar el desempeño del modelo. Ya vimos que basarnos únicamente en el error del modelo en D_{train} para determinar $h(x)$ podría causar sobreajuste, sin embargo, determinar una mejor estrategia basándonos en el error en D_{test} también. Esto se debe a que en primer lugar el propósito de un conjunto de evaluación es simular datos nunca antes vistos, y comparar resultados en base a ellos supone verlos. Es por esto que no es considerada buena práctica ajustar hiperparámetros en D_{test}.

Debido a la necesidad de ajustar hiperparámetros sobre un conjunto de datos no vistos sin intersección con D_{test} es que suele dividirse el conjunto D_{train} una vez más, teniendo ahora un conjunto $D'_{train} \subset D_{train}$ más pequeño y un nuevo conjunto $D_{val} \subset D_{train}$ llamado conjunto de validación o desarrollo, y el ya creado D_{test} (Mitchell, 1997).

Cuando se separa el conjunto de datos en tres únicas partes distinguidas D_{train}, D_{val} y D_{test}; una para entrenar, otra para tomar decisiones y otra para evaluar; se dice que D_{test} está siendo utilizada como conjunto *held-out*. Existen también métodos más avanzados y robustos como la validación cruzada, que crea n combinaciones distintas de conjuntos D^i_{train} y D^i_{val} y promedia los resultados obtenidos en validación (Mitchell, 1997).

2.2.4. Mecanismos de ajuste de hiperparámetros

Aún solucionado el problema del sobreajuste al seleccionar hiperparámetros, de esta solución surgen dos nuevos problemas. El primero es que al ajustar dos hiperparámetros distintos, asumir independencia entre ellos no siempre es válido. Por ejemplo, si un hiperparámetro es el algoritmo utilizado, y el otro el tiempo de entrenamiento, el tiempo óptimo para un algoritmo no necesariamente lo será para el otro. El segundo problema es que, siguiendo este ejemplo, el tiempo es una medida continua, lo cual causa que haya infinitas combinaciones posibles de hiperparámetros, y aún siendo finita, el número de combinaciones de valores y por lo tanto pruebas a realizar crecerán exponencialmente al número de hiperparámetros a probar. Debido a estos problemas es que existen métodos específicos de ajuste de hiperparámetros.

El método de ajuste más sencillo es el conocido en inglés como *grid search*, o búsqueda en grilla (Goodfellow et al., 2016). El método de *grid search* soluciona solo el problema de la continuidad de los hiperparámetros primero discretizando el espacio de búsqueda del modo que el diseñador de los experimentos desee. Luego, en él se procede a validar al modelo sobre el producto cartesiano hiperparámetros posibles. Es importante resaltar que este recorrido puede realizarse

en el orden que se desee ya que cada evaluación es independiente.

Por otro lado existe el método de búsqueda aleatoria o *random search*, el cual permite solucionar ambos problemas recorriendo el espacio de búsqueda de forma estocástica según una distribución de probabilidad elegida (Goodfellow et al., 2016). Esto tiene la desventaja de quitar el control de saber de antemano los valores exactos que serán recorridos.

Tanto *grid search* como *random search* tienen la ventaja de ser fácilmente paralelizables dado que la elección de hiperparámetros en el paso i no afecta la elección en el paso $i + k$ para ningún k positivo. Sin embargo, en caso de conocerse regiones del espacio de búsqueda donde el desempeño del modelo suela ser alto o bajo, podría ajustarse la búsqueda para evitar o preferir determinadas regiones. Debido esto es que existen métodos como la búsqueda bayesiana que se basa en los resultados de pasos anteriores para guiar la búsqueda (Goodfellow et al., 2016). No obstante, dada su naturaleza secuencial el método pierde la posibilidad de ser paralelizable. Pueden observarse las diferencias mencionadas entre métodos de ajuste en la tabla 2.3.

Propiedad	**Búsqueda en grilla**	**Búsqueda aleatoria**	**Búsqueda bayesiana**
Paralelizable	Si	Si	No
Usa resultados previos	No	No	Si
Se elige número de iteraciones	No	Si	Si

Tabla 2.3: Comparación de mecanismos de ajuste de hiperparámetros

2.3. Redes neuronales

Una familia de funciones muy utilizada en la actualidad para el aprendizaje automático son las redes neuronales, que son la base de modelos más complejos que se tratarán más adelante. Una red neuronal es un modelo definido como una red de unidades, donde cada una tiene la responsabilidad de propagar entre ellas la entrada x de la red a su salida, realizando cálculos intermedios sobre x hasta transformarla en la salida $\hat{y}$ (Mitchell, 1997) (Jurafsky y Martin, 2009). El objetivo principal de una red neuronal es aproximar una función $f : \mathbb{R}^n \to \mathbb{R}^m$ mediante una función $h : \mathbb{R}^n \to \mathbb{R}^m$.

Dada la complejidad en cuanto a operaciones y número de parámetros, las redes neuronales son consideradas algoritmos difíciles de interpretar. Además, ellas son propensas al sobreajuste dada la alta complejidad que puede adquirir un modelo de muchos parámetros. Por último, las redes neuronales suelen tener un tiempo de entrenamiento prolongado, ya sea debido al tamaño de la red o porque se utilizan en problemas donde se involucran grandes volúmenes de datos.

2.3.1. Redes feedforward

Una red feedforward es una red neuronal conformada por capas, donde cada capa está compuesta por un conjunto de unidades que solo propagan datos de una capa i a una capa $i+1$ (Mitchell, 1997) (Jurafsky y Martin, 2009). Puede verse entonces la función aprendida $h(x)$ de una red feedforward como una aplicación secuencial de funciones $h_i : \mathbb{R}^{n_i} \to \mathbb{R}^{n_{i+1}}$ sobre la salida de la capa anterior, donde n_i es el tamaño de la entrada de la capa i, y n_{i+1} de la salida de la capa i (ver figura 2.2).

En redes feedforward, las capas distintas a la de entrada y salida son llamadas capas ocultas, y llamaremos a d la profundidad de la red, que equivale al número de capas ocultas. Notar que en una red neuronal el número de capas d es un hiperparámetro del algoritmo, al igual que lo son las distintas n_i excepto para la primera y la última, ya que corresponden al tamaño de la entrada y salida de la red.

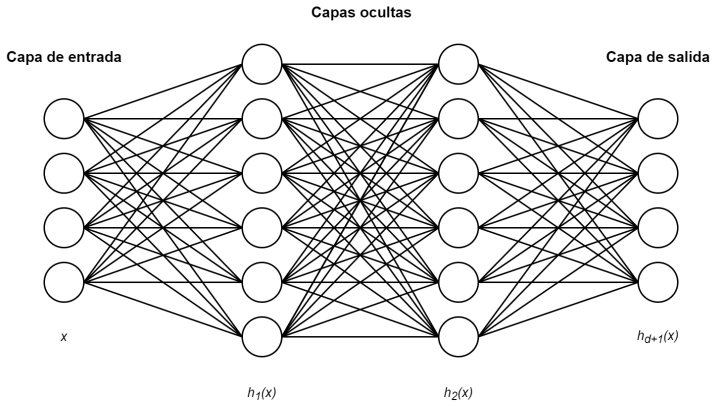

Figura 2.2: Ejemplo de red neuronal de dos capas ocultas.

2.3.2. Propagación hacia adelante

Como ya fue mencionado, para obtener el valor de $h(x)$ es necesario propagar la entrada x por las distintas funciones $h_{i \in \{1,\ldots,d+1\}}$, proceso que se llama propagación hacia adelante o *forward propagation* en inglés (Goodfellow et al., 2016).

En cada función $h_i(x)$, la entrada numérica es sometida a operaciones matriciales y posteriormente funciones lineales o no lineales antes de llegar a la capa $i+1$. Se define entonces la función $h(x)$ aprendida como:

$$\begin{cases} h(x) &= h_{d+1} \circ h_d \circ \ldots \circ h_1(x) \\ h_i(x) &= g_i(W_i x + b_i) \end{cases}$$

Donde:

- Denotaremos matriz de pesos de la capa a $W_i \in \mathbb{R}^{n_i \times n_{i-1}}$.
- Denotaremos vector de sesgo a $b_i \in \mathbb{R}^{n_i}$.
- Denotaremos función de activación de la capa a $g_i : \mathbb{R}^{n_i} \to \mathbb{R}^{n_i}$.

2.3.3. Funciones de activación

Fue mencionada en la sección anterior la existencia de ciertas funciones $g_i : \mathbb{R}^{n_i} \to \mathbb{R}^{n_i}$ llamadas funciones de activación (Jurafsky y Martin, 2009). Ellas son también hiperparámetros de una red neuronal, y su elección puede ser crucial para que el modelo logre un buen desempeño.

En casos de uso como problemas de clasificación, es necesario saber si una entrada pertenece (1) o no pertenece (0) a determinada clase. Sin embargo, en casos donde un error del modelo puede ser crucial, por ejemplo al querer diagnosticar o no una enfermedad, es a veces necesario más que ceros y unos. Es por esto que existen funciones de activación como las funciones $softmax$ y $sigmoide$, que tienen la facultad de retornar valores reales entre 0 y 1. Esto es primordial desde un punto de vista de interpretabilidad, ya que ahora al utilizar estas funciones en la última capa de la red podremos conocer el nivel de confianza del modelo al dar un resultado, interpretando la salida de $h(x_k)$ como una función de probabilidad discreta $p(h(x_k) = 1)$ ($sigmoide$), o un vector de probabilidades en problemas multiclase ($softmax$) (Jurafsky y Martin, 2009). Formalmente, ellas pueden definirse como:

$$softmax(x_1, \ldots, x_n)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

$$sigmoide(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

Por otro lado, para las capas ocultas pueden ser utilizadas funciones como la sigmoide, $tanh : \mathbb{R} \to [-1, 1]$ o $Relu : \mathbb{R} \to [0, \infty]$ (Jurafsky y Martin, 2009), que son funciones que van de $\mathbb{R}$ a $\mathbb{R}$ aunque pueden ser generalizadas a funciones de $\mathbb{R}^n$ a $\mathbb{R}^n$ simplemente aplicando la misma función unidimensional sobre cada coordenada de la entrada. Ambas pueden visualizarse en la figura 2.3 y se encuentran definidas a continuación en su versión unidimensional:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$ReLU(x) = max(0, x)$$

2.3.4. Descenso por gradiente

Se definió anteriormente la función aprendida de una red neuronal, la cual se ayuda de un conjunto de matrices y vectores W y b, sin embargo, no se habló de cómo obtener sus valores.

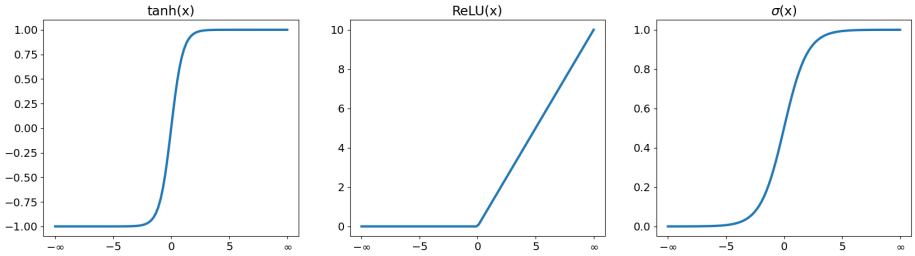

Figura 2.3: Funciones de activación *tanh*, *ReLU* y sigmoide

En este caso, la red aprenderá esos parámetros a partir de un algoritmo de optimización iterativo y de primer orden llamado descenso por gradiente (Mitchell, 1997) (Jurafsky y Martin, 2009). El descenso por gradiente es un algoritmo que ajusta los parámetros de pesos y sesgos de forma iterativa a modo de minimizar una función $J : \mathbb{R}^m \to \mathbb{R}$ inversamente proporcional al desempeño del modelo, lo cual es realizado a una velocidad α. Llamaremos a J función de pérdida o costo, donde m es la dimensión de todos los posibles pesos y sesgos del algoritmo. Además, llamaremos tasa de aprendizaje o *learning rate* a esa velocidad α.

Este método se basa en modificar los parámetros a optimizar en la misma dirección que la opuesta al gradiente $\nabla_\theta(J)$, donde el valor en la posición i corresponde a $\frac{\partial J}{\partial \theta_i}$ y puede interpretarse como la medida en la que variar el parámetro θ_i localmente en esa dirección aumenta el valor de J, siendo 0 en un punto crítico.

Formalmente, dada una tasa de aprendizaje α y una función de costo $J : \mathbb{R}^m \to \mathbb{R}$, el método de optimización de descenso por gradiente en su forma vectorial es calculado iterativamente como:

$$\theta_i = \theta_{i-1} - \alpha \nabla_\theta J(\theta_{i-1})$$

Donde:

- $\theta_i \in \mathbb{R}^n$ es el vector de parámetros a ajustar en la iteración i.
- $\alpha \in \mathbb{R}^+$ es la tasa de aprendizaje.

Se destaca que la elección de α como hiperparámetro del modelo es crucial para que los parámetros a optimizar no se ajusten muy lento, ni tan grande para que el algoritmo nunca converja. Más aún, autores afirman que a la hora de ajustar hiperparámetros en una red neuronal, el principal hiperparámetro a tener en cuenta es la tasa de aprendizaje (Goodfellow et al., 2016).

Uno de los problemas de este método es que en muchos casos el cálculo del gradiente puede ser costoso. Debido a esto es que suele ser conveniente aplicarlo calculando el gradiente por lotes del conjunto de entrenamiento, y no sobre todo

el conjunto. En este caso, se dice que se está utilizando descenso por gradiente de mini-lote o mini-batch. Se dice además que la red recorre una *epoch* cada vez que el algoritmo se ejecuta sobre todos los ejemplos de la red desde la *epoch* anterior, o sea cada vez que entrena sobre todos los lotes. Otro problema del método es que la optimización avanza menos en el espacio cuanto menor es el gradiente, lo cual causa que no siempre halle un óptimo global al poder converger en óptimos locales.

Finalmente, a partir del método de descenso por gradiente surge la necesidad de hallar el gradiente de la función de costo. El algoritmo que en una red neuronal realiza el cálculo de gradientes es conocido como propagación hacia atrás o *backpropagation* en inglés (Mitchell, 1997) (Jurafsky y Martin, 2009) (Goodfellow et al., 2016).

2.3.5. *Vanishing* y *exploding gradients*

Un problema que suele ocurrir con ciertas funciones de activación al utilizar métodos derivados del descenso por gradiente es que, como puede observarse en la fórmula en 2.3.4, cuando el gradiente de la entrada de la función es cercano a cero, o muy grande, la neurona parará de aprender o lo hará tan rápido que no llegue a un resultado consistente. Cuando el gradiente en una neurona no logra dejar de ser nulo, se está frente al problema de desvanecimiento de gradientes o *vanishing gradients* en inglés. Cuando el gradiente se mantiene en valores muy altos, se le llama explosión de gradientes o *exploding gradients* (Jurafsky y Martin, 2009) (Goodfellow et al., 2016).

De las funciones mencionadas, la función *tanh* y σ son las mayores causantes del fenómeno de *vanishing gradients*, ya que cuando su entrada se acerca al infinito o menos infinito su derivada tiende a 0. Por otro lado, la función *ReLU* es una buena solución a este problema, ya que mientras su entrada sea positiva su pendiente se mantendrá en 1 .

2.3.6. Preentrenamiento

Algo realmente útil de las redes neuronales es que, dada su naturaleza iterativa, es posible guardar los valores de sus parámetros en una iteración deseada y luego cargarlos ya sea para reutilizar el modelo o para reanudar el entrenamiento en esa iteración. A raíz de esto es que se ha descubierto que es posible además reutilizar los parámetros aprendidos por el modelo para llevar a cabo una tarea t_1 sobre un conjunto de datos D_1, y reanudar el entrenamiento sobre una tarea t_2 y un conjunto D_2 de datos.

Se le llama entonces preentrenamiento al acto de entrenar a un modelo sobre un conjunto y tarea con el fin de ser reutilizado para la realización de otra, y *fine-tuning* a reanudar el ajuste de parámetros desde la última iteración del preentrenamiento sobre otro conjunto de datos (Jurafsky y Martin, 2009) (Goodfellow et al., 2016). El *fine-tuning* es una técnica muy común en tareas resueltas con redes neuronales, y suele causar un aumento en el desempeño del modelo en la nueva tarea comparado a no haberlo entrenado.

2.3.7. Métodos para evitar el sobreajuste

Los métodos para evitar el sobreajuste usualmente dependen de la metodología, modelo y datos utilizados. Por un lado, presentamos en una sección previa el método de validación con un conjunto de datos *held-out* como técnica general para evitar el sobreajuste. Por otro lado vimos en una sección posterior que el sobreajuste es un problema importante en las redes neuronales, principalmente cuando la red utilizada cuenta con muchos parámetros. En esta sección se presentarán por lo tanto algunas técnicas utilizadas frecuentemente en redes neuronales, aunque no se apliquen solamente a ellas.

Aumentar datos

Existen casos donde la muestra disponible de datos de entrenamiento es poco representativa de la población de datos no disponibles. Aumentar datos se refiere a crear datos nuevos a partir de datos disponibles (Jurafsky y Martin, 2009) (Goodfellow et al., 2016). Por ejemplo, al tratar con imágenes es posible diversificar los datos rotando o ajustando el contraste de imágenes del conjunto de entrenamiento. Al tratar con texto también es posible sustituir palabras de una frase por sus sinónimos o alterar el orden de las palabras. En este trabajo utilizamos una técnica alternativa de aumentado de datos, no necesariamente para reducir el sobreajuste sino para mejorar la *performance* del modelo: generar un gran conjunto de datos sintéticos a partir de conocimiento lingüístico.

Reducir la cantidad de parámetros

Nos referimos con esta sección a ajustar cualquier hiperparámetro que reduzca la cantidad de parámetros del modelo, y por ende su complejidad (Abu-Mostafa et al., 2012). Algunos ejemplos son reducir la cantidad de capas de una red neuronal o compartir parámetros entre capas como se muestra en Inan et al. (2017) y Press y Wolf (2017).

Early stopping

El método de *early stopping* tiene como requisitos ser utilizado sobre métodos de aprendizaje iterativos como lo es una red neuronal, y utilizar un conjunto de validación. Este método se basa en, dado un hiperparámetro k y una medida de desempeño P, entrenar un modelo sobre el conjunto de entrenamiento hasta que la medida P no mejore durante k iteraciones en el conjunto de validación (Goodfellow et al., 2016).

Otra utilidad del método es que permite reducir el tiempo de entrenamiento frente a otros criterios de parada como utilizar una cantidad máxima de iteraciones, ya que un número constante de iteraciones requiere ser conservador para no parar las iteraciones de forma prematura, y esta técnica se ajusta automáticamente en tiempo de entrenamiento.

Dropout

El método de dropout (Srivastava et al., 2014) se basa en, para cada unidad de una red neuronal, desactivar tanto sus entradas como salidas con probabilidad p.

La técnica de dropout es utilizada durante el entrenamiento de una red neuronal para evitar que el éxito del modelo dependa solo de un conjunto reducido de neuronas, lo cual logra inhabilitando unidades aleatoriamente para exigir a que el resto mejore sin ellas.

Label smoothing

Label smoothing (Szegedy et al., 2015) es una técnica utilizada para tareas donde la salida esperada y es un vector one-hot, y busca entrenar al modelo sobre un conjunto de etiquetas y' modificado según un hiperparámetro α, el cual aumenta los valores nulos y reduce el valor más alto del vector de salida y real. Este método puede utilizarse en tareas de modelado de lenguaje ya que se suelen utilizar salidas como vectores one-hot a partir de una función de activación softmax, y de hecho es uno de los métodos para evitar el sobreajuste utilizado al introducir el modelo transformer (Vaswani et al., 2017).

Formalmente, dada una tarea con salida codificada utilizando one-hot, la salida real $y \in [0,1]^n$ de una tarea, la salida $\hat{y} \in [0,1]^n$ de un modelo, una función de costo $J : [0,1]^n \times [0,1]^n \to [-\infty, \infty]$ y un hiperparámetro $\alpha \in [0,1]$, entonces el método de label smoothing plantea calcular el costo del modelo a partir de $J(y', \hat{y})$ en lugar de $J(y, \hat{y})$, donde y' está dada por:

$$y' = y * (1 - \alpha) + \frac{\alpha}{n}$$

2.4. Procesamiento de lenguaje natural

Nos referimos al procesamiento de lenguaje (PLN) como un área de conocimiento dedicada a estudiar el procesamiento de lenguaje humano, ya sea texto o audio. A día de hoy son utilizadas técnicas de PLN para resolver múltiples tareas de automatización, entre ellas la clasificación de documentos, sistemas de recomendación, extracción de información y traducción automática; las cuales pueden y suelen ser resueltas utilizando técnicas de aprendizaje automático (Jurafsky y Martin, 2009).

2.4.1. Corpus

Para generar soluciones aplicando técnicas de PLN, es fundamental poder evaluar su efectividad. Para ello, es usualmente necesario el uso de datos en formato de texto (Jurafsky y Martin, 2009). Se define entonces el término corpus, que es un conjunto de datos lingüísticos, ya sea en formato de texto o audio.

En tareas como traducción a veces es necesario que corpus de dos idiomas estén alineados, o sea, que se indique ya sea a nivel de frase o documento qué

parte de una lengua corresponda a la otra. En este caso decimos que se está frente a un corpus paralelo (Jurafsky y Martin, 2009).

Por último, definimos a un *token* como la mínima unidad que compone un corpus, siendo un token usualmente una palabra (Jurafsky y Martin, 2009).

2.4.2. n-gramas

Dado un corpus, un n-grama es una secuencia de n tokens subsecuentes (Jurafsky y Martin, 2009). Este concepto es útil por ejemplo para tareas de análisis de datos para comprender un corpus según sus n-gramas más frecuentes, o si se quiere determinar qué token es más probable observar luego de un n-grama. Un modelo de n-gramas con $n = 1$ es llamado de unigramas, cuando $n = 2$ se le llama de bigramas y así sucesivamente. Además, cuando todos los tokens son caracteres, se dice que se se está tratando con n-gramas de caracteres.

2.4.3. Tokenización

Dada la naturaleza semi-estructurada del lenguaje, suele requerirse realizar tareas de preprocesado antes de poder un texto ser interpretado por una máquina. Una tarea frecuente en el área de PLN es la de tokenización (Jurafsky y Martin, 2009), la cual se basa en dividir un texto en una secuencia de tokens. Por ejemplo, un tokenizador de palabras puede ser representado como una función $t(w)$ donde t(“La reunión es a las 15:00”) = (“La”, “reunión”, “es”, “a”, “las”, “15:00”). Notar que existen casos donde este proceso se hace difícil de automatizar solo mediante reglas, por ejemplo, queda poco claro si ‘15:00’ debería ser un solo token o varios, ya que de hecho la respuesta ideal dependerá de la tarea donde se utilice.

Los diversos problemas a tener en cuenta a la hora de elegir un tokenizador llevan a que existan también diferentes técnicas distinguidas. Dentro de aquellos métodos existentes son de particular interés la tokenización a nivel de palabra y subpalabra.

La forma más simple de dividir un texto en palabras es delimitando los tokens en espacios. Luego, existen técnicas que aplican métodos basados en reglas, probabilidades o hasta métodos de aprendizaje automático.

Por otro lado, los métodos más comúnes de tokenización de subpalabra utilizan algoritmos que aprenden una función $t(w)$ dado un corpus de entrenamiento ya tokenizado por palabras, y un tamaño de vocabulario k de subpalabras a generar. Notar que requerir de un corpus pre-tokenizado es una desventaja, ya que esto agrega una capa de complejidad. Un ejemplo de métodos de subpalabra son WordPiece (Wu et al., 2016) utilizado para preentrenar a BERT, Byte Pair Encoding (BPE) (Sennrich et al., 2016b) utilizado en modelos como GPT2 (Radford et al., 2019) o RoBERTa (Liu et al., 2019) y la tokenización de unigramas (Kudo, 2018) utilizada en T5 (Raffel et al., 2023). De aquí en adelante profundizaremos en los últimos dos modelos de tokenización.

BPE es un algoritmo surgido en el área de compresión de datos, y plantea construir una función de tokenización $t(w)$ dado un corpus de entrenamiento,

a partir del cual unirá los pares de caracteres (bytes) más frecuentes en un solo caracter de forma iterativa hasta obtener un vocabulario de caracteres de tamaño k, almacenando el orden de operaciones realizadas en un diccionario. Luego, en tiempo de tokenización se aplican las mismas reglas en orden en que fueron aprendidas sobre la frase a tokenizar.

Por otro lado, la tokenización de unigrama es un método probabilístico que modela cada secuencia como una distribución de probabilidad, asumiendo independencia entre tokens en una misma secuencia. Dado un corpus de entrenamiento, el método almacena inicialmente en su vocabulario toda posible subpalabra del corpus pre-tokenizado almacenando su probabilidad empírica de ocurrencia. Luego, el método reduce el vocabulario iterativamente hasta tener tamaño k, descartando en cada paso el token que de no existir se aumentaría más la verosimilitud del modelo de tokenización que quitar cualquier otro token. Notar que esta técnica requiere de estructuras de datos y algoritmos eficientes dado que trabaja con todas las combinaciones de subpalabras a la vez.

2.4.4. Vectores de palabras

Luego de tokenizar, es común asignarle valores numéricos a cada token, en particular, asignarle un vector a cada uno. Esto es útil ya que como vimos en la sección de redes neuronales, permite utilizar texto tanto como entrada como salida de un modelo de aprendizaje automático. Además, de quererse utilizar sobre una secuencia y no solo un token, es posible inmediatamente representar frases como vectores, por ejemplo asignándole el centroide (promedio entre vectores) de los vectores de cada token que la componen (Jurafsky y Martin, 2009).

Un vector de palabra es una forma de representar una palabra de forma vectorial y numérica. La creación de estos vectores puede llevarse a cabo de diversas formas según la tarea. Una común clasificación para ellos es la de vectores dispersos (con muchos ceros) o densos (Jurafsky y Martin, 2009).

En el área de PLN, para los vectores dispersos predominan los vectores one-hot (Jurafsky y Martin, 2009), que asumen que existe un vocabulario V de tokens t_i donde $i \in \{1, .., |V|\}$, y asignan a cada token un vector v_i que cumple que valor vector v_{ij} vale 1 sii $i = j$, y 0 si no. Una ventaja de los vectores one-hot es que dado un vector es inmediato saber cuál es el elemento original del que provino, dada la inyectividad de la asignación de vectores. Como ejemplo del método de codificación one-hot, dado el vocabulario ordenado ("Lunes", "Martes", "Miércoles"), la palabra "Miércoles" tendrá asignado el vector $(0, 0, 1)$.

Por otro lado, para la creación de vectores densos pueden utilizarse métodos como Word2Vec (Mikolov et al., 2013), Glove (Pennington et al., 2014), o FastText (Bojanowski et al., 2017) que, a diferencia de los vectores one-hot, asignan posiciones cercanas en el espacio a palabras con semántica similar. A los vectores densos que mantienen esta información semántica se les llama *word embeddings*.

2.5. Gramáticas formales

Las gramáticas formales son conjuntos de reglas que determinan cómo construir enunciados que sean válidos según la sintaxis de un lenguaje (Sag y Wasow, 1999). Se puede decir que estas reglas, llamadas producciones, describen sistemáticamente la sintaxis del lenguaje en cuestión.

En este sentido, podemos definir el análisis sintáctico o *parsing* de un string o una oración como un proceso que resulta en una representación estructural de la oración en un formalismo gramatical (Sag y Wasow, 1999). Particularmente, un parser tendría por salida la sucesión de reglas que debieron ser aplicadas para construir una oración, partiendo desde un cierto símbolo inicial. A una representación jerárquica se le conoce como árbol sintáctico o *parse tree*.

2.5.1. Gramáticas libres de contexto

Las gramáticas más simples son las gramáticas libres de contexto o GLC (Sag y Wasow, 1999). Denominamos símbolos no terminales a aquellos que pueden derivar en una secuencia de símbolos según alguna producción, y símbolos terminales a aquellos que no pueden ser sustituidos por otros; estos últimos serían las palabras del lenguaje. Las producciones entonces definen las sustituciones posibles de símbolos por secuencias de símbolos.

Un ejemplo sencillo de GLC sería el siguiente:

```
# S símbolo inicial
S -> NP V
NP -> D N
D -> 'la' | 'los'
N -> 'niños' | 'niña'
V -> 'duermen' | 'vive'
```

Este permite derivar oraciones como "los niños duermen" o "la niña vive", pero también admite oraciones como "la niños vive" o "los niña duermen", que no cumplen con la gramática del español. Para solucionar este problema, podríamos definir diferentes símbolos para el caso donde el sustantivo es masculino o femenino, y si es plural o singular, pero a medida que incluimos más matices y aplicamos la misma lógica a otros símbolos (determinantes, verbos), el conjunto de reglas crece exponencialmente.

2.5.2. Gramáticas de rasgos

El problema recién planteado es el de *concordancia*, específicamente concordancia en español entre determinante y sustantivo, o entre sujeto y verbo. Una solución a este problema es considerar gramáticas que trabajan con *estructuras de rasgos* (*feature structures*) (Sag y Wasow, 1999). Estas estructuras pueden contener rasgos o *features*, que representan distintos atributos del símbolo o palabra, como características morfológicas; o a otras estructuras, siguiendo un

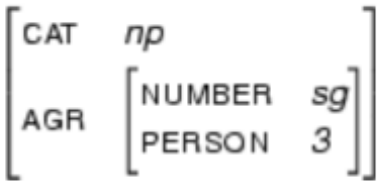

Figura 2.4: Ejemplo de estructura de rasgos para el inglés. Esta representa un sintagma nominal, con otra estructura anidada señalando el número y persona del mismo.

esquema de anidación. Un ejemplo de estas estructuras de rasgos puede observarse en la figura 2.4.

En estas gramáticas de rasgos, las reglas de producción pueden hacer referencia a los rasgos para exigir concordancia entre distintos componentes de la oración. Un ejemplo de una gramática de rasgos sencilla sería la siguiente:

```
# S símbolo inicial
S -> NP[AGR=?a] V[AGR=?a]
NP[AGR=?a] -> D[AGR=?a] N[AGR=?a]
D[AGR=[GEN=m, NUM=pl]]  -> 'los'
D[AGR=[GEN=f, NUM=sg]]  -> 'la'
N[AGR=[GEN=m, NUM=pl, PER=ter]] -> 'niños'
N[AGR=[GEN=f, NUM=sg, PER=ter]] -> 'niña'
V[AGR=[NUM=pl, PER=ter]] -> 'duermen'
V[AGR=[NUM=sg, PER=ter]] -> 'vive'
```

En este caso, ya no es posible derivar oraciones que sean incorrectas en español, como "la niños vive"; las reglas exigen que el determinante y sustantivo posean el mismo valor para los rasgos de género y número, y que el sujeto y verbo coincidan en persona y número. Estas concordancias se implementan a través del proceso de *unificación*.

Unificación

Definimos unificación como una operación sobre dos estructuras de rasgos tal que su resultado incluye toda la información contenida en esas estructuras, y nada más (Sag y Wasow, 1999). En el caso anterior, con una regla como NP[AGR=?a] → D[AGR=?a] N[AGR=?a], estamos exigiendo que NP herede en el rasgo AGR el resultado de unificar esa estructura en D y en N. Si algún rasgo de esa estructura no concuerda (ej. el género es masculino en el determinante y femenino en el sustantivo), la unificación falla, y por eso no es posible derivar sintagmas como "los niña" según esta gramática.

2.5.3. Head-driven Phrase Structure Grammar

Las gramáticas HPSG (de *Head-driven Phrase Structure Grammars*, o gramáticas sintagmáticas nucleares) son un tipo de gramática fuertemente lexicalizada que utiliza estructuras de rasgos tipificadas para representar tanto palabras como árboles (Pollard y Sag, 1994). Estas estructuras se organizan en una jerarquía de tipos, donde cada tipo tiene rasgos específicos para los que define posibles valores.

En una HPSG, la mayor parte de la información necesaria para la combinación de palabras y la formación de estructuras gramaticales está contenida en las entradas léxicas (es decir, las producciones que derivan en símbolos terminales), lo que significa que las HPSG contienen relativamente pocas reglas para la construcción de oraciones. Estas reglas están inspiradas en la teoría X', incluyendo aquellas para combinar un núcleo con su especificador, sus complementos o sus modificadores.

$$\begin{bmatrix} \textit{word} & & \\ \text{PHON} & \langle \text{'she'} \rangle & \\ \text{SYNSEM} & \begin{bmatrix} \textit{synsem} & \\ \text{CAT} & \begin{bmatrix} \text{HEAD} & \textit{noun} \end{bmatrix} \\ \text{CONT} & \begin{bmatrix} \textit{ref-index} & \\ \text{PER} & \textit{3rd} \\ \text{NUM} & \textit{sing} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figura 2.5: Representación de la palabra *she* como una matriz de HPSG.

En estas gramáticas, el concepto de núcleo es fundamental, y guía el proceso de parsing. En un sintagma nominal, por ejemplo, el núcleo sería el sustantivo; en un sintagma verbal, el verbo. El núcleo define, entre otras cosas, con qué signos puede combinarse ese símbolo, y qué rasgos heredará la frase que compone.

Las gramáticas HPSG no solo contienen información sintáctica, sino también información semántica y fonológica, como se observa en la figura 2.5. La figura 2.6 muestra un ejemplo más complejo que involucra símbolos no terminales[2].

2.6. Traducción automática

Un traductor automático es, en esencia, un sistema capaz de convertir oraciones de un idioma origen a un idioma destino. Este concepto representa uno de los primeros desafíos que se abordaron en el ámbito del PLN, marcando así el inicio de una rica historia que vio evolucionar los métodos utilizados a la par del avance tecnológico y la profundización del entendimiento del lenguaje humano.

[2]Figuras tomadas de https://en.wikipedia.org/wiki/Head-driven_phrase_structure_grammar

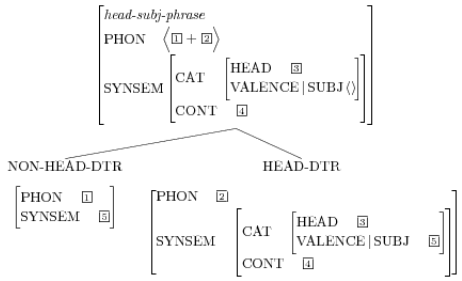

Figura 2.6: Árbol para una frase núcleo-sujeto, que espera un verbo y un sujeto. Si bien HPSG consiste exclusivamente en estructuras tipificadas, la representación como árbol suele resultar más práctica.

A continuación presentamos las principales familias de métodos de traducción automática.

2.6.1. Traducción basada en reglas

Los primeros sistemas de traducción automática fueron creados utilizando reglas de traducción (Nirenburg et al., 2003). Estos métodos son los más básicos y fáciles de interpretar, ya que se basan en heurísticas simples aunque altamente propensas a fallar; un ejemplo es traducir cada palabra utilizando un diccionario. Un método que más adelante tomará gran relevancia es la traducción mediante transferencia sintáctica, la cual se basa en interpretar cada frase según su árbol sintáctico y realizar transformaciones del árbol hasta llevarlo a la estructura y vocabulario correspondiente en su lenguaje destino.

Transferencia sintáctica

Un ejemplo avanzado de traducción basada en reglas es la *transferencia sintáctica*, que corresponde al proceso de parsear el texto de entrada y aplicar reglas para transformar el árbol sintáctico del idioma origen a un árbol sintáctico del idioma destino, para finalmente a partir de este árbol generar una oración en el idioma de destino, que sea una traducción del texto original (Jurafsky y Martin, 2009). Los autores lo distinguen de otros enfoques, como pueden ser la traducción directa o la interlingua, como se observa en la figura 2.7.

Para llevar adelante un proceso de traducción mediante transferencia sintáctica, es necesario contar con una gramática formal para el idioma de origen y otra para el idioma de destino, y definir reglas de transferencia que asocien producciones del idioma de origen a producciones del idioma de destino. Esto último incluye las producciones léxicas, es decir, aquellas que derivan en terminales; por ende, es necesario contar con alguna clase de diccionario bilingüe o tabla de traducción entre términos de ambos idiomas.

Las reglas de transferencia permiten representar las diferencias sintácticas entre dos idiomas, especialmente entre lenguas con estructuras gramaticales no-

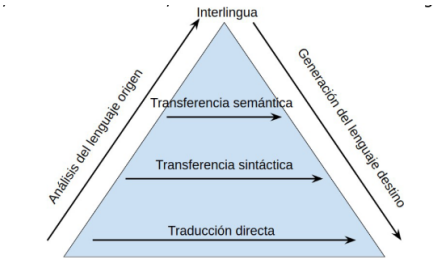

Figura 2.7: Triángulo de Vauquois.

toriamente distintas, como es el caso del español y el guaraní. En la figura 2.8 se observa un ejemplo de traducción mediante transferencia sintáctica entre estos dos idiomas.

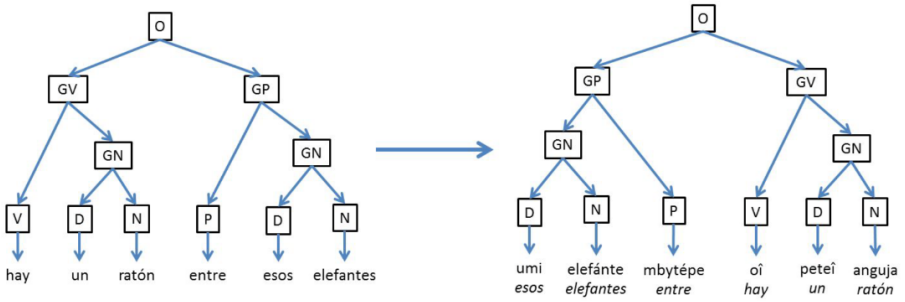

Figura 2.8: Ejemplo de transferencia sintáctica entre español y guaraní[3].

La primer etapa en este proceso consiste en construir un árbol sintáctico en el idioma de origen. Este es una representación de las reglas sintácticas utilizadas, en un orden específico, derivando en un vocabulario específico en sus hojas. Para obtener un árbol sintáctico en el idioma de destino, se toman dos pasos.

El primero es transformar el árbol original a uno del idioma objetivo mediante la aplicación de reglas de transferencia, que asocian una regla de generación con su equivalente en otra lengua.

El segundo es tomar las hojas del árbol original, y hacer uso del diccionario bilingüe para traducir directamente cada palabra del idioma de origen al idioma de destino.

Así, como resultado final se obtiene un par de árboles sintácticos equivalentes, de los cuales se puede deducir entonces un par de oraciones equivalentes; es decir, se logra el objetivo de traducir un texto del idioma origen al idioma

[3]Tomado del curso de Introducción al Procesamiento de Lenguaje Natural, Facultad de Ingeniería. https://eva.fing.edu.uy/course/view.php?id=211

destino.

2.6.2. Traducción estadística

A partir de los modelos de traducción basado en reglas es que surgen sistemas de traducción estadística que buscan hallar el resultado más probable y dada una frase de origen x (Brown et al., 1993). La ecuación fundamental de la traducción estadística parte del teorema de Bayes y puede expresarse como:

$$h(x) = \arg\max_{y} p(x|y)p(y)$$

Donde la finalidad de un modelo de traducción estadística es modelar tanto $p(x|y)$ (modelo de traducción) como $p(y)$ (modelo de lenguaje), y buscar candidatos de traducción óptimos mediante un algoritmo eficiente (decodificador).

Estos modelos fueron los que superaron a aquellos basados en reglas ya que ofrecían un método más fácil de automatizar y mantener, además de tener gran potencial de descubrir patrones difíciles de reconocer por humanos de tenerse suficientes datos.

2.6.3. Traducción automática neuronal

La traducción automática neuronal (Goodfellow et al., 2016) es el tipo de traducción que hoy en día ofrece resultados de mejor desempeño en condiciones óptimas. Esta metodología permite no solo descubrir patrones ocultos en los datos de forma automática, sino que logra reducir significativamente la memoria utilizada y complejidad alcanzable por un modelo de traducción.

Presentaremos a continuación arquitecturas de redes neuronales utilizadas en el área de traducción, comenzando por las redes recurrentes, que comparten conceptos con las redes neuronales feedforward y son la base de modelos de traducción más complejos y actualmente usados.

Redes neuronales recurrentes

Las redes neuronales recurrentes, a diferencia de las feedforward son útiles cuando los datos de entrada y/o salida de nuestro problema son secuenciales (Jurafsky y Martin, 2009) (Goodfellow et al., 2016). En este caso, ser secuencial implica tener un orden. Por ejemplo, estos datos podrían ser tanto las ventas de una empresa desde enero a diciembre, como una secuencia de palabras de un texto.

Aquello que hace que estas redes puedan hacer uso de datos secuenciales es que procesan cada elemento de la secuencia $(x^{(1)}, \ldots, x^{(n)})$ de forma ordenada, donde al procesar el elemento $x^{(k)}$ se generará información que será utilizada subsecuentemente al procesar $x^{(k+1)}$, información a la cual se le llamará estado $s^{(k)}$.

Dependiendo del número de entradas y salidas de la red, una red recurrente puede ser formalizada de varias formas. A fines de comprender el uso de las

redes recurrentes en la tarea de traducción automática nos concentraremos en una red secuencial many-to-many, o sea, redes de tamaño de más de una entrada y salida. En dicho caso, la secuencia de entrada estará conformada por la frase origen a traducir, y la de salida por la frase traducida.

De forma similar a las redes neuronales feedforward, una red recurrente calcula, dada una entrada $x^{(t)}$, la salida $h(x^{(t)})$ a partir de operaciones matriciales y aplicación de funciones de activación. Sin embargo, aunque las dimensiones de $x^{(t)}$ no tienen por qué coincidir con las de $h(x^{(t)})$, es importante resaltar que en una red recurrente secuencial clásica el largo de la secuencia de entrada es estríctamente idéntica a la de salida (ver figura 2.9). Esto hace que esta arquitectura, aunque buena para modelar datos secuenciales, requiera de modificaciones para ser utilizable en tareas de traducción automática, dado que una frase en un idioma no tiene por qué tener el mismo largo que su traducción en otra lengua.

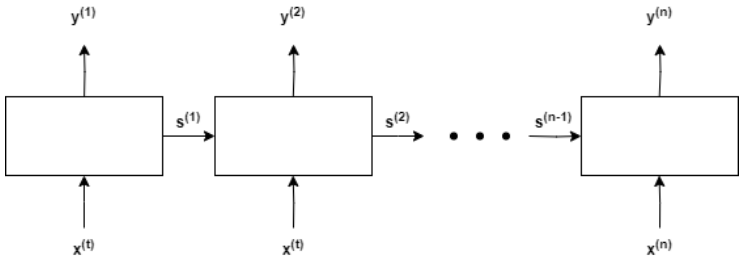

Figura 2.9: Ejemplo de red recurrente secuencial con mismo tamaño de entrada que de salida.

Vanishing y *exploding gradients* en RNNs

Uno de los problemas que de las redes recurrentes es un problema ya mencionado en secciones anteriores, el problema de *vanishing* y *exploding gradients*, además de la dificultad de modelar dependencias entre elementos lejanos de la secuencia.

Algunos de los métodos utilizados para resolver estos problemas son:

- Utilizar funciones de activación que no sufran de este problema, como las mencionadas en secciones anteriores.
- Forzar al gradiente a mantenerse en tamaños pequeños (solo evita el problema de explosión) utilizando la técnica de recorte de gradiente o *gradient clipping* en inglés (Pascanu et al., 2013).
- Omitir operaciones de salidas calculadas de forma temprana, utilizando por ejemplo *skip connections* (Goodfellow et al., 2016).
- Adaptar la arquitectura de la red recurrente para que esto no ocurra, lo cual se tratará en la próxima sección.

LSTMs y GRUs

Como fue introducido en la sección anterior, una de las formas de solucionar el problema de *vanishing gradients* en redes neuronales es modificando la arquitectura de la red. Existen dos propuestas que hoy en día pertenecen al estado del arte en el área de modelado de secuencias, y han logrado mejorar el desempeño de las redes recurrentes introducidas previamente: las redes LSTM (Hochreiter y Schmidhuber, 1997) y GRU (Chung et al., 2014). Aunque ambas son variaciones de redes recurrentes, a partir de ahora pasaremos a llamar a la primer red recurrente mencionada como red recurrente clásica, y a ellas como LSTM y GRU.

Las redes LSTM (long short-term memory) resuelven el problema de las RNN reemplazando la idea original de utilizar un único estado a corto plazo $s^{(t)}$ a tanto un estado a corto como largo plazo, dando la posibilidad a la red de aprender a qué tantos pasos atrás en el tiempo es conveniente tomar en cuenta en el tiempo t. Por otro lado, las redes GRU (gated recurrent unit) mantienen la idea de las LSTM de regular la importancia del corto y largo plazo, aunque haciendo uso de un solo estado, lo cual permite reducir la cantidad de parámetros a aprender aunque nuevamente con el problema de delegar ambas tareas a un estado. Además, aún con esta modificación las redes LSTM y GRU siguen sin ser capaces de generar secuencias de salida de largo independiente al número de secuencias de entrada. Una representación de los tres tipos de redes recurrentes mencionados puede observarse en la figura 2.10.

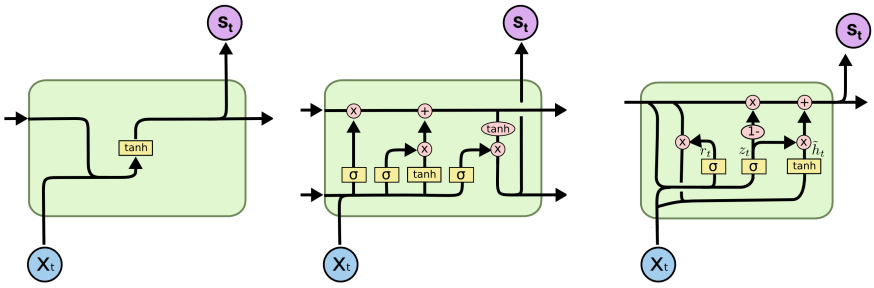

Figura 2.10: Comparación esquemática de estructura de redes RNN, LSTM y GRU adaptada de Olah (2015).

seq2seq

La arquitectura seq2seq (también conocida como encoder-decoder) (Sutskever et al., 2014) surge para resolver el problema de las redes recurrentes de no poder producir una salida de distinto tamaño que el de la secuencia de entrada. La red seq2seq soluciona este problema concatenando dos redes recurrentes distintas:

un codificador y un decodificador. En este caso, la responsabilidad del codificador es generar un estado $s_{enc}^{(n)}$ a partir de los elementos de entrada $(x_{enc}^{(1)}, \ldots, x_{enc}^{(n)})$ (sin tener por qué utilizar su salida), mientras que la del del decodificador es generar la salida esperada $(y_{dec}^{(1)}, \ldots, y_{dec}^{(m)})$ del modelo a partir de $s_{enc}^{(n)}$ (ver figura 2.11). De aquí en adelante llamaremos *celda* a la unidad que procesa cada entrada y estado en el tiempo t.

En cuanto al uso de esta arquitectura en tareas de traducción automática, es necesario tener varias consideraciones. En primer lugar, se mencionó que el decodificador es una red recurrente que recibe un estado $s_{enc}^{(n)}$ del codificador, sin embargo, no se menciona cuál es su entrada. En este caso, la entrada $x_{dec}^{(t)}$ del decodificador suele corresponder a la misma salida $y_{dec}^{(t-1)}$ que produjo en el paso anterior. Por otro lado, para el caso de $x_{dec}^{(1)}$ suele ser utilizado $y_{enc}^{(n)}$ para determinar el inicio de la frase. Además, dado que una red recurrente no tiene límite en cuanto a número de salidas generadas, suele utilizarse un token <EOS> para determinarse el final de una secuencia de salida. Adicionalmente, existen técnicas más avanzadas como apilar redes seq2seq (Sutskever et al., 2014), redes recurrentes bidireccionales (Schuster y Paliwal, 1997) o mecanismos de atención (Bahdanau et al., 2016) que suelen superar el rendimiento del modelo original.

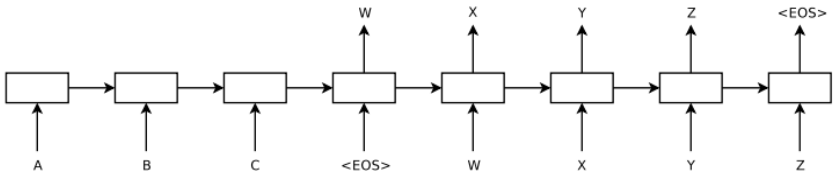

Figura 2.11: Red seq2seq haciendo uso de tokens de fin de oración (Sutskever et al., 2014).

Por otra parte, el primer sistema de traducción neuronal propuesto (Kalchbrenner y Blunsom, 2013) fue hecho utilizando redes recurrentes clásicas como decodificador y otro tipo de redes más comúnes en el área de procesamiento de imágenes llamadas redes convolucionales (Lecun et al., 1998) como codificador. No fue sino un año después (Sutskever et al., 2014) que fue planteado el uso de redes LSTM alcanzando resultados cercanos al estado del arte haciendo uso de considerablemente menos memoria. Además, ese mismo año aparecen modelos que son comparables con el estado del arte en traducción estadística (Bahdanau et al., 2016), y más adelante algunos que la superan (Luong et al., 2015). Desde entonces el estado del arte de la traducción neuronal ha avanzado y superado los otros mecanismos de traducción, por lo que en el año 2016 Google sustituye el sistema detrás de su servicio de traducción por uno basado en traducción neuronal (Wu et al., 2016).

Transformers

Uno de los problemas de las redes recurrentes (y por ende de las redes seq2seq) es que dada su naturaleza secuencial suelen tener tiempos de entrenamiento extensos, ya que son difíciles de paralelizar. Por esto es que fue propuesta la arquitectura de transformers (Vaswani et al., 2017), la cual es entrenada de forma más veloz que arquitecturas basadas en RNNs, e impone un nuevo estado del arte en tareas de traducción como la WMT 2014 English-to-French translation task, aunque no reducido a esta área del PLN.

Al igual que las primeras redes seq2seq, los transformers cuentan con dos componentes principales: Un codificador y un decodificador, ambos compuestos por N bloques de codificadores y decodificadores apilados, donde N es un hiperparámetro del modelo. Una representación de esta arquitectura aparece en la figura 2.12 según (Vaswani et al., 2017), y como puede verse está formada por algunos elementos ya introducidos anteriormente y otros que valdrá la pena mencionar a continuación.

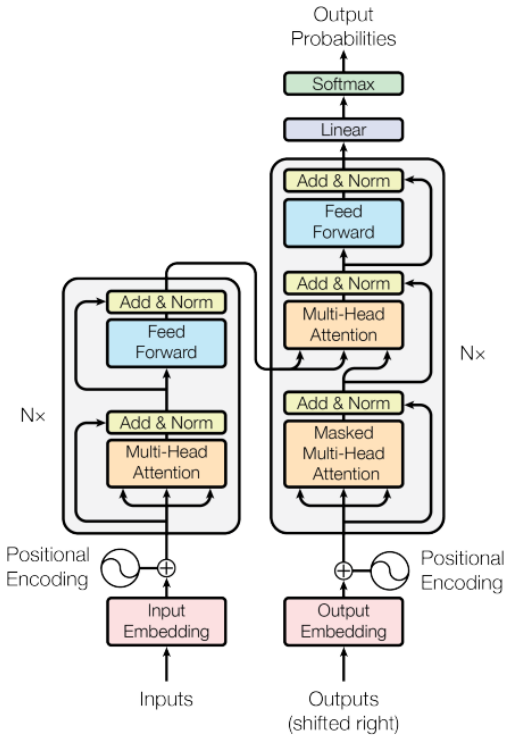

Figura 2.12: Esquema de arquitectura de transformer (Vaswani et al., 2017).

En primer lugar, se observa que en la entrada del modelo los transformers convierten la entrada, que se asume ya está transformada de forma vectorial, a

embeddings que preservan información de la posición de cada secuencia, permitiendo modelar el orden de los tokens de una secuencia al igual que los modelos seq2seq, aunque de forma paralelizable.

Posteriormente, tanto en codificador como decodificador los embeddings posicionales pasan a un mecanismo de atención, donde el transformer en tiempo de entrenamiento busca aprender la relación entre elementos en distintas posiciones de la secuencia. Por ejemplo, en la secuencia de texto 'Ayer leí ese libro, aunque me pareció muy aburrido', es importante que el modelo aprenda que el elemento 'aburrido' está haciendo referencia a 'libro'. Este mecanismo es implementado comparando todos los tokens de una secuencia en simultáneo y determinando con qué nivel afecta un token a otro, y es uno de los avances más importantes que plantea el transformer.

Otro elemento no visto con anterioridad es la etapa de adicionar y normalizar. Con adicionar se está refiriendo a un tipo de *skip connection* que hace que salidas de etapas previas se salteen etapas sumándolas (Huang et al., 2018). Luego, normalizar un valor se realiza para acelerar el entrenamiento (Ba et al., 2016). Además, el modelo transformer retorna en el decodificador, mediante la función de activación $softmax$, la probabilidad de que cada token sea el próximo de la secuencia.

2.6.4. Medidas de evaluación

Para la evaluación de modelos de traducción automática suelen utilizarse medidas que comparan un corpus de referencia y (traducción real) y una traducción candidata $\hat{y}$. Una función de evaluación puede verse entonces como una función $E(y, \hat{y})$ que retorna un valor real.

Para la evaluación de un corpus traducido existen múltiples métricas de evaluación. A fines de comprender este proyecto vamos a dividirlas en métricas que evalúan coincidencia exacta de resultados y aquellas basadas en similitud semántica. Nos referimos las primeras cuando hablamos de métricas como BLEU (Papineni et al., 2002), chrF (Popović, 2015) y TER (Snover et al., 2006), que son métricas que evalúan la fidelidad de una traducción en base a la diferencia entre secuencias de caracteres o palabras. Por ejemplo, para una referencia y ="¿Cómo estás?", y una traducción $\hat{y}$ ="¿Qué tal?", métricas como BLEU otorgan $E(y, \hat{y}) = 0$. Por otro lado, las medidas como BLEURT (Sellam et al., 2020) o COMET (Rei et al., 2020) otorgarían un valor más cercano a 1, ya que utilizan criterios de similitud semántica que evalúa su significado más allá de las palabras y caracteres de las frases comparadas.

En este proyecto utilizamos principalmente las medidas BLEU y chrF. Como vimos, ambas son medidas que se basan en coincidencia de n-gramas de tokens o caracteres, y tienen desventaja de no dar tanta importancia al significado de las frases evaluadas, aunque también por ello son sencillas de calcular y no requieren del costo computacional ni del trabajo de obtener un modelo que determine la información semántica de una frase (lo cual es particularmente difícil en lenguas de bajos recursos como el guaraní). Por otro lado, en cuanto a sus valores, ambas se encuentran entre 0 y 1, y suelen ser presentadas también con porcentajes.

Sin embargo, es difícil para ambas obtener valores cercanos al máximo, dado que eso significaría que la traducción esperada y sea exactamente idéntica al texto traducido $\hat{y}$. Además, aunque como veremos tanto BLEU como chrF son métricas que se comportan de forma similar, la medida BLEU es más sensible a cambios en la traducción $\hat{y}$, dado que cambiar por ejemplo la conjugación de la palabra $\hat{y}_1$ = "dirigirá" a $\hat{y}_2$ = "dirigiré" produciría un leve aumento en chrF y un gran aumento en BLEU.

BLEU

La métrica BLEU busca evaluar el desempeño de una traducción y una referencia según el número de coincidencias de n-gramas de palabras.

La fórmula para calcular la medida BLEU se presenta entonces a continuación:

$$BLEU(y, \hat{y}) = BP \times \exp\left(\sum_{n=1}^{N} w_n \cdot \log \frac{p_n(\hat{y}, y)}{|\hat{y}|_n}\right)$$

Donde:

- N es el número máximo de n-gramas a considerar. El mínimo siempre es 1, y N suele ser 4.
- $|y|_n$ se refiere a la suma de la cantidad de n-gramas sobre cada frase del corpus. Notar que $|y|_1$ representa el número de tokens en el corpus.
- BP (brevity penalty) es una medida que penaliza traducciones breves. Esto se debe a que el la precisión toma en cuenta casos en los que un n-grama en la traducción coincide con la referencia, aunque no n-gramas en la referencia que no están en la traducción. Se define BP entonces como

$$BP = min(1, e^{(1-|y|_1/|\hat{y}|_1)}).$$

- $w_n \in [0, 1]$, con $\sum_{n=1}^{N} w_n = 1$, es el peso otorgado a cada n-grama, y usualmente coincide con la distribución uniforme $\frac{1}{N}$.
- $p_n(\hat{y}, y)$ equivale a la suma de veces donde un n-grama en $\hat{y}$ también pertenece en y. En casos que haya más ocurrencias de ese n-grama en $\hat{y}$ que en y, ese sumando se trunca al número de ocurrencias de ese n-grama en y.

Por otro lado, BLEU tiene varias desventajas. Una de ellas es que necesita de un tokenizador para determinar qué es un n-grama en una frase, lo cual no solo supone una capa extra de complejidad sino que dificulta la comparación de medidas de distintos experimentos (Post, 2018). La otra es que no tiene en cuenta semántica, y no coincidir exactamente con una palabra o n-grama tiene una gran penalización. Esto último es muy relevante ya que dada la naturaleza del lenguaje suele existir más de una forma de escribir una misma frase, y un claro ejemplo de esto son los sinónimos.

chrF

La medida chr$F$ logra compensar la desventaja de BLEU de penalizar ampliamente coincidencias a nivel de token ya que chr$F$ se evalúa a nivel de caracter. Esto es útil ya que amortigua la penalización en a casos donde el traductor tiene errores morfológicos, como faltas ortográficas o errores de conjugación, caso el cual BLEU puntua de forma nula para ese n-grama, y chr$F$ de forma no tan alta.

Esta medida es útil para lenguajes donde la morfología de las palabras es más rica, ya que aquí es de interés valorar el uso correcto de morfemas aunque así lo sea de forma parcial. Un ejemplo de esto es la evaluación de la secuencia "Estaré" contra "Estoy".

La fórmula de chrF_β está basada en la de la medida F_β para aprendizaje supervisado, y coincide con:

$$\text{chrF}_\beta(\hat{y}, y) = \frac{(1+\beta^2) \times \text{chrP}(\hat{y}, y) \times \text{chrR}(\hat{y}, y)}{\beta^2 \times \text{chrP}(\hat{y}, y) + \text{chrR}(\hat{y}, y)}$$

Donde:

- $\beta \in \mathbb{N}^+$ es un parámetro que modifica el comportamiento de la métrica. Mientras más alto β más se tiene en cuenta el *recall*.
- $\text{chrP}(\hat{y}, y) = \sum_{n=1}^{N} p_n^{char}(y, \hat{y})/N$. Esta métrica es una medida de coincidencias entre n-gramas de caracteres en $\hat{y}$ que también están en y.
- $p_n^{char}(\hat{y}, y)$ equivale a la suma de veces donde un n-grama de caracteres en $\hat{y}$ también pertenece en y.
- $\text{chrR}(\hat{y}, y) = \sum_{n=1}^{N} r_n^{char}(y, \hat{y})/N$. Similarmente, esta métrica es una medida de cantidad de coincidencias entre n-gramas de caracteres en y que también están en $\hat{y}$.
- $r_n^{char}(\hat{y}, y)$ equivale a la suma de veces donde un n-grama de caracteres en y también pertenece a $\hat{y}$. Notar que esta métrica es simétrica a $p_n^{char}(\hat{y}, y)$ en cuanto a y e $\hat{y}$.

2.7. Antecedentes en lenguajes de bajos recursos

Durante los últimos años ha habido un gran esfuerzo por parte de algunos autores de mejorar tanto la cantidad de datos de lenguas de bajos recursos como los modelos existentes de traducción automática. En particular, el guaraní es una de las lenguas donde se ha hecho más énfasis en los últimos años, y es de aquellas con más recursos disponibles dentro de esta categoría.

En cuanto al trabajo realizado en la generación de recursos, se ha hecho un gran avance colectivo para logar mejoras significativas. En cuanto a corpus paralelos, existen a día de hoy conjuntos de datos de la Biblia para varios idiomas (McCarthy et al., 2020) o el corpus paralelo guaraní-español de Jojajovai

(Chiruzzo, Góngora, et al., 2022) que comprende un conjunto de ocho corpus. En cuanto a corpus monolingües, también existen corpus tomados de Wikipedia (Tiedemann, 2020). Además, existen más recursos de los aquí mencionados que fueron principalmente creados en los últimos años (Borges et al., 2021-03).

En cuanto a modelos de traducción, existen comunidades como AmericasNLP que realizan eventos periódicamente con el fin de superar resultados de años previos en traducción automática (Mager et al., 2021) (Ebrahimi et al., 2023), donde se trabaja en varias lenguas de bajos recursos incluyendo el guaraní. En estas tareas, los candidatos que han obtenido mejores resultados en guaraní suelen ser el grupo Helsinki (Vázquez et al., 2021) (De Gibert et al., 2023) utilizando modelos de transformers multicapa aplicando técnicas de regularización y tokenización de unigrama. Además, logran estos resultados haciendo uso de una cantidad de datos masivos los cuales fueron sometidos a un proceso de limpieza que concluyen puede ser fundamental para un mejor rendimiento del modelo. Por otro lado, también utilizan técnicas para compartir embeddings entre lenguajes, y utilizan modelos preentrenados en corpus en otras lenguas de bastos recursos. En particular, en el año 2021 sus modelos obtuvieron las mejores 3 posiciones de la tarea, y en 2023 volvieron a ganar utilizando el mismo modelo[4] aunque con más datos.

Aún observando el gran desempeño de arquitecturas transformer, autores discuten que no solo el buen ajuste de hiperparámetros y técnicas para reducir el sobreajuste son fundamentales para el desempeño de los transformers en lenguajes de bajos recursos, sino que cuando la cantidad de datos es realmente escasa los modelos basados en RNNs suelen dar mejores resultados (Araabi y Monz, 2020). Esto parece indicar que a partir de cierto umbral en la cantidad de datos disponibles, utilizar modelos seq2seq puede ser una mejor opción. Aún así, los modelos de traducción neuronal siguen siendo superiores a los de traducción estadística en condiciones de bajos recursos (Sennrich y Zhang, 2019).

Por otro lado, en cuanto a técnicas de tokenización, se habló ya que grupos como Helsinki lograron los mejores resultados en varios eventos haciendo uso de la tokenización de unigrama, particularmente mediante el método de regularización de subpalabras (Kudo, 2018). Algunos autores enfatizan la importancia del tokenizador, en particular del hiperparámetro del tamaño de vocabulario generado (Sennrich y Zhang, 2019), principalmente para arquitecturas basadas en RNNs. Por otra parte, se habló de que es común el uso de técnicas de preentrenamiento y *fine-tuning*, y de hecho, existe evidencia de que al preentrenar un modelo, tokenizar utilizando BPE es subóptimo en comparación a la tokenización de unigrama, la cual además parece adaptarse mejor a representaciones morfológicas (Bostrom y Durrett, 2020), que son fundamentales en lenguas de rica morfología como el guaraní.

Como ya se vio, trabajar en estas lenguas suele ser difícil, en primer lugar debido a que como su nombre lo indica, los recursos existentes son insuficientes. También, existen problemas que dificultan aún más trabajar en este campo como la cantidad de datos ruidosos, morfología compleja o hasta variaciones de

[4] https://github.com/AmericasNLP/americasnlp2021

dialectos y ortografía debido a la falta de formalización de la lengua (Mager et al., 2023). Asimismo, tampoco es sencillo determinar la calidad de un resultado dada la baja cantidad de mediciones disponibles (Guzmán et al., 2019). Sin embargo, más allá de la dificultad que trabajar en esta área pueda conllevar, los resultados obtenidos cada año superan los del año anterior para las múltiples lenguas de bajos recursos existentes (De Gibert et al., 2023), lo cual parece indicar que aún hay mucho trabajo por realizar.

2.8. Conjuntos de datos

En esta sección presentamos los conjuntos de datos ya existentes que utilizamos durante el proyecto.

2.8.1. Jojajovai

El primer corpus que mencionaremos será el corpus de Jojajovai. Ya introdujimos este corpus, y es el que utilizaremos en el proyecto como conjunto de entrenamiento en todos nuestros experimentos. Este conjunto tiene una cantidad total de 30.855 frases alineadas en guaraní y español, y ya está dividido en conjuntos de entrenamiento, validación y evaluación de tamaños 20.207, 5.314 y 5.334 respectivamente. Además, está compuesto por 8 subconjuntos de datos de diversas fuentes, principalmente periodísticas, por lo que será conveniente que los analicemos por separado en la sección de resultados y será algo a tener en cuenta en el análisis.

2.8.2. Biblia

Como ya fue presentado, existe un corpus paralelo que presenta frases alineadas de la lengua español a guaraní. De hecho, este corpus presenta alineación para más de 1600 lenguas y contiene más de 20.000 líneas de texto. En nuestro trabajo utilizamos este corpus como uno de los conjuntos de datos de preentrenamiento.

La Biblia es un ejemplo de corpus paralelo interesante, debido a que está traducida a diversos idiomas. Sin embargo, presenta la desventaja de que es un texto muy antiguo, escrito en un registro que no es habitual en la actualidad, por lo que los datos son bastante ruidosos. De todas maneras, es ampliamente utilizada en escenarios de lenguajes de bajos recursos.

2.8.3. Ancora

La suite de AnCora es un conjunto de recursos para el español y el catalán creados por el grupo CLIC de la Universidad de Barcelona[5]. De estos recursos utilizamos el corpus AnCora, y el corpus AnCora-Verb.

[5] https://clic.ub.edu/

El primero es un corpus para el español y el catalán con aproximadamente medio millón de palabras, compuesto de texto periodístico, donde se analiza el rol de cada vocablo de diferentes oraciones[6]. De este, nos interesan particularmente las oraciones utilizadas, debido a que corresponden a diferentes fragmentos de artículos periodísticos, es decir, que son semánticamente correctos además de sintácticamente, aportando mayor variedad a nuestro proyecto.

El corpus AnCora-Verb resulta del análisis del anterior. Es es un corpus de verbos en español con varias de sus características, entre ellas la transitividad[7]. Este cuenta con diferentes archivos para cada verbo presentado, con las etiquetas morfológicas correspondientes de él, acompañado de ejemplos para lograr una mejor comprensión del caso.

2.8.4. Freeling

Freeling[8] es una biblioteca de procesamiento del lenguaje natural de código abierto desarrollada para el análisis lingüístico de texto en varios idiomas, entre ellos el español. Esta herramienta ofrece una amplia cantidad de funcionalidades que incluyen análisis morfológico, etiquetado gramatical, análisis sintáctico y desambiguación semántica, entre otros. Esta fue desarrollada por el grupo TALP (Tratamiento Automático del Lenguaje y la Parla) de la Universidad Politécnica de Cataluña, y se destaca por su capacidad para descomponer las palabras en sus formas básicas, identificar la estructura gramatical de las oraciones y asignar etiquetas a las palabras según su función en el texto.

Particularmente, para nuestro proyecto, utilizamos el tagset de categorías gramaticales, o *parts of speech*, para palabras en español proporcionado por la biblioteca. En este se asignan etiquetas gramaticales precisas a cada palabra, permitiendo una identificación clara de su función gramatical dentro de una oración[9]. Este conocimiento sobre cada vocablo resulta fundamental para lograr una traducción sintácticamente correcta. Entre estas etiquetas gramaticales podemos destacar, por ejemplo, tiempo, número y persona para verbos, género en pronombres, sustantivos y adjetivos, etc. Ejemplificando, el tagset correspondiente a verbos cuenta con líneas como la presentada a continuación.

```
sonará,sonar,V,M,I,F,3,S,0
```

En esta encontramos "sonará" que es el verbo que se está presentado, "sonar" corresponde a su raíz, y luego le siguen las etiquetas gramaticales. En este caso corresponden a que es un verbo (V), principal (M), en modo indicativo (I), en futuro (F), de tercera persona (3) del singular (S), al cual no le corresponde género (0). Estas etiquetas tienen un orden especificado en los documentos de la biblioteca para cada tipo de vocablo, el cual siempre se presentará de esa forma y sin ninguna faltante, es decir, si alguna no corresponde se utiliza una

[6] https://clic.ub.edu/corpus/es
[7] https://clic.ub.edu/corpus/es/ancoraverb_es
[8] https://github.com/TALP-UPC/FreeLing
[9] https://www.sketchengine.eu/spanish-freeling-part-of-speech-tagset/

tag indicando neutro o indiferencia, como se puede ver en el caso del género en el ejemplo anterior.

2.8.5. Diccionarios Guaraní - Español

Al dar inicio al proyecto, contábamos con acceso a tres fuentes principales de referencia para la traducción entre español y guaraní. Estos recursos lingüísticos proporcionaron una base fundamental para la comprensión y el análisis de ambos idiomas al comenzar este proyecto. En primer lugar, el Wiktionary alberga un diccionario español-guaraní y viceversa, que proporciona una recopilación colaborativa de términos y sus traducciones correspondientes entre ambos idiomas[10]. Sin embargo, este está basado en aportes de usuarios, por lo que su calidad y exhaustividad pueden variar.

Luego tenemos el diccionario Ávalos (Ávalos, 2011), otra fuente de referencia importante, que ofrece una recopilación de términos y expresiones en español y guaraní. En este caso, al contar no solo con términos sino también con expresiones, una gran cantidad de su contenido no nos resulta útil a la hora de traducir oraciones. Por ejemplo, es común encontrar en él líneas de la forma `v gn:akyve'ỹ es:recuperar_el_valor`, donde el correspondiente en español constituye un grupo de palabras, lo cual dificulta la transferencia sintáctica.

Por último, está el diccionario proveniente de Descubrir Corrientes[11], y procesado en un proyecto anterior (Chiruzzo et al., 2023), el cual ofrece otra perspectiva y una colección adicional de palabras y sus traducciones correspondientes entre español y guaraní. Este fue finalmente el que escogimos, debido a su amplia variedad de vocablos con traducciones simples.

2.9. Herramientas

En esta sección presentamos las herramientas más relevantes utilizadas durante la realización del proyecto.

2.9.1. Código

Para el desarrollo de la gramática sintáctica junto a su algoritmo de traducción[12] fue utilizado el lenguaje de programación Python.

Por otro lado, para la realización de experimentos[13] también fue utilizado Python en su versión 3.8.10 en conjunto con scripts de bash y slurm.

2.9.2. Gramáticas

Para llevar a cabo el análisis sintáctico en nuestro proyecto, utilizamos la biblioteca NLTK (Natural Language Toolkit). NLTK es una herramienta amplia-

[10] https://www.wiktionary.org/
[11] https://descubrircorrientes.com.ar/2012/index.php/diccionario-guarani/
[12] https://github.com/baladon-lucas-pardinas/SyntaxGrammar-es-gn
[13] https://github.com/baladon-lucas-pardinas/NMT-Translation-gn-es

mente reconocida en el campo del procesamiento del lenguaje natural, ofreciendo una amplia gama de funciones y algoritmos para el análisis lingüístico. Entre sus capacidades, NLTK proporciona módulos para tokenizar, etiquetar palabras, realizar análisis morfológico, y construir estructuras gramaticales, permitiendo así la comprensión y manipulación de datos textuales en diferentes idiomas[14].

Una característica que cabe destacar son las Feature Grammar[15] brindadas por la biblioteca, las cuales son una forma de gramática en la que se pueden asociar características o rasgos (*features*) a las estructuras gramaticales de un lenguaje, como las definidas en la sección Sección 2.5. Estas características pueden representar información sintáctica, semántica o cualquier otro tipo de información relevante para el análisis lingüístico, permitiendo de esta forma crear reglas gramaticales con más detalles y más realistas.

Por ejemplo, al trabajar con una Feature Grammar en NLTK para el análisis sintáctico del idioma español, se pueden asignar atributos como género, número, persona, entre otros, a los elementos gramaticales, logrando generar oraciones con sentido sintáctico. Se podría asociar a un sustantivo la información sobre su número (singular/plural), género (masculino/femenino), y garantizar mediante reglas que este asocie con adjetivos que se ajusten a las restricciones definidas.

2.9.3. Software de traducción

Hoy en día existen múltiples alternativas en software para realizar traducción automática. A fines prácticos vamos a dividirlas en dos grupos: open-source y pagos. En cuanto a su calidad, las investigaciones al respecto son escasas, especialmente para idiomas como el guaraní. Aún así, existen estudios realizados[16] que parecen indicar que no hay una diferencia significativa entre frameworks pagos y no pagos en cuanto a calidad de traducción para el caso general, por lo que decidimos no considerar soluciones pagas. Asimismo, otros artículos (Birch et al., 2018) indican algo similar entre algunos de los frameworks open-source más populares, como OpenNMT y MarianNMT.

Por otro lado, identificamos que OpenNMT es de las bibliotecas con más amplia comunidad, mientras que MarianNMT parecía resaltar por su eficiencia computacional a la hora de entrenar, dado que es un software implementado en C++. Debido a que ambos permiten hacer uso de modelos y técnicas consideradas estado del arte, decidimos utilizar MarianNMT, dado que valoramos eficiencia sobre comunidad.

2.9.4. Métricas

Para la extracción de métricas decidimos utilizar la biblioteca Sacrebleu (Post, 2018), ya que provee funciones que implementan el cálculo de métricas como BLEU, chrF y TER tanto a nivel de corpus como de oración. En este proyecto utilizamos principalmente las medidas a nivel de corpus, y solo las

[14] https://www.nltk.org/
[15] https://www.nltk.org/book/ch09.html
[16] https://github.com/Optum/nmt

utilizamos a nivel de oración para visualizar dado un corpus la calidad de traducción de cada frase individual.

Además, Sacrebleu se encarga tanto de la implementación como estandarización de la etapa de tokenización. Gracias a esto es posible que los resultados sean reproducibles al aplicar otra persona esta función sobre un mismo corpus, ya que quita la responsabilidad del usuario de elegir cómo tokenizar.

2.9.5. Tokenización

Con respecto al cálculo de métricas, fue mencionado que la biblioteca Sacrebleu se encarga de la tokenización. Por otro lado, también utilizamos un tokenizador en los experimentos de traducción para crear el vocabulario (lista de palabras únicas) de cada lengua. Para ello utilizamos inicialmente un tokenizador a nivel de palabra de Spacy (Honnibal y Montani, 2017) entrenado sobre su conjunto de `es_core_news_md` en su versión 3.7.0, aunque luego hicimos uso principalmente de la biblioteca Sentencepiece (Kudo y Richardson, 2018) para tokenizar a nivel de subpalabra.

Sentencepiece es una biblioteca muy utilizada en el área de traducción automática, e implementa los métodos de tokenización BPE y de unigrama utilizando algoritmos y estructuras de datos eficientes. Una de las razones es que, como fue mencionado con anterioridad, los métodos de tokenización de subpalabra requieren del uso de un pretokenizador, lo cual dificulta la reproducibilidad de los resultados. Entonces, al igual que Sacrebleu, Sentencepiece provee un pretokenizador estándar para evitar estos problemas. Además, provee métodos de normalización de codificaciones y diacríticos, implementa métodos para evitar el sobreajuste, y es compatible con el software MarianNMT.

Capítulo 3

Generación de los corpus

En esta sección presentamos el proceso por el cual generamos los nuevos conjuntos de datos para la traducción entre español y guaraní.

Parte de nuestra motivación para crear un corpus sintético es poner el foco en la clase de oraciones que no solemos encontrar en los corpus existentes. Por ejemplo, gran parte del Jojajovai consiste en texto periodístico, el cual por su naturaleza cuenta con un cierto lenguaje, mayor formalidad, y favorece enunciados más largos. Oraciones más cotidianas, breves, tal vez más propias de la oralidad, o aquellas que refieren a la primer o segunda persona gramatical, resultan notoriamente escasas en un texto de tales características. Esto se ve reflejado en los resultados que presentan Chiruzzo, Góngora, et al. (2022) para los distintos subconjuntos de Jojajovai, donde la traducción muestra peores resultados para los subconjuntos con más oraciones cortas, más diálogo, y menos nombres de entidades.

Por este motivo consideramos interesante proveer una gran cantidad de ejemplos de esta índole al sistema, buscando darle una visión más completa y más versátil de la lengua al modelo entrenado. Tener el control sobre el vocabulario y la sintaxis utilizados para sintetizar oraciones facilita este fin, a la vez que nos permite enfocar otras particularidades de la lengua guaraní como pueden ser las posposiciones o las conjugaciones verbales.

Nuestro proceso consiste, a grandes rasgos, en construir un lexicón bilingüe con etiquetas morfológicas, construir una gramática de rasgos para el español, generar y parsear oraciones en español usando estos recursos, y aplicar un algoritmo de transferencia sintáctica para obtener árboles y oraciones en guaraní. Así, finalmente obtenemos pares de oraciones guaraní-español que constituyen nuestro nuevo corpus.

3.1. Lexicón

El primer paso para generar las oraciones requeridas es contar con un diccionario español-guaraní con cada tipo de palabra necesaria, su traducción, y

sus respectivas características morfológicas que se consideren pertinentes. Para lograr esto, se cuenta con un diccionario existente, obtenido del portal Descubrir Corrientes (ver Subsección 2.8.5), donde se encuentran vocablos en guaraní y su correspondiente traducción al idioma español, específicamente verbos, adjetivos y sustantivos. Al ser un diccionario bilingüe tradicional, se cuenta con la forma "base" de cada palabra; es decir, sin variantes de género ni número en el caso de los sustantivos y adjetivos, y solo en infinitivo en el caso de los verbos.

Por otro lado, para categorías que este diccionario no incluye, como los determinantes, pronombres, adposiciones y conjunciones, definimos el listado bilingüe de forma manual, basados en la gramática presentada por la Academia de la Lengua Guaraní (2018)

Definiremos un lexicón compuesto de varios archivos, uno para cada tipo de vocablo, con el siguiente formato:

$$\text{Palabra\_en\_guaraní}, e_1, \ldots, e_n, \text{palabra\_en\_español}, f_1, \ldots, f_m$$

Donde $e_1, \ldots, e_n$ corresponde a las etiquetas definidas para la palabra en guaraní, como se explicará en la Sección 3.1.2, y $f_1, \ldots, f_m$ a las etiquetas en español, abordadas en la Sección 3.1.1. Cabe destacar que, para cada archivo, la cantidad de elementos en cada línea debe ser la misma, por lo que, si a una palabra no le corresponde alguna de las etiquetas definidas, contiene aquel que corresponda a indefinido, neutro, etc.

3.1.1. Generación de lexicón en español

Para construir un lexicón bilingüe como el que necesitamos para el proceso de transferencia sintáctica, que incluya las variantes anteriormente mencionadas de género, número, persona y tiempo, entre otras, y los rasgos correspondientes a cada uno de esos atributos, hicimos uso del diccionario interno en español de Freeling, presentado en la Subsección 2.8.4. A través de su código *open-source* logramos conseguir un listado amplio de palabras en español para cada categoría gramatical, con sus respectivas etiquetas morfológicas como, por ejemplo, número, persona o género.

Etiquetas del español

En cuanto a los verbos, se cuenta con etiquetas indicadoras de verbo, tipo, modo, tiempo, persona, número, género y transitividad. Por ejemplo, para el verbo "prosperan" tenemos la siguiente fracción.

```
prosperan,prosperar,V,M,I,P,3,P,0,0,0,intr.
```

En esta fracción, podemos observar el verbo conjugado en español y su raíz (verbo en infinitivo), seguidos de las etiquetas indicadas en la tabla 3.1, que este caso son el indicador de verbo (V), principal (M), modo indicativo (I), presente

(P), tercera persona (3), plural (P), sin género (0), verbo no transitivo (0), verbo no ditransitivo (0), verbo intransitivo (intr).

Cabe destacar, que al obtener los verbos de la biblioteca de Freeling, entre ellos se encontraban verbos en todos los tiempos y modos, incluyendo participios; motivo por el cual se incluye la etiqueta correspondiente a género.

Español
Indicador de verbo: V
Tipo, M: principal, A: auxiliar; S: semi auxiliar
Modo, I: indicativo, S: subjuntivo, M: imperativo, P: participio, G: gerundio, N: infinitivo
Tiempo, P: presente, I: pretérito imperfecto, F: futuro, S: pretérito perfecto, C: condicional
Persona, 1: primera, 2: segunda, 3: tercera
Número, S: singular, P: plural
Género, M: masculino, F: femenino, C: común
Indicador de verbo transitivo (tr o 0)
Indicador de verbo ditransitivo (di o 0)
Indicador de verbo intransitivo (intr o 0)

Tabla 3.1: Conjunto de todas las etiquetas utilizadas por los verbos para el idioma español

Para los determinantes contamos con indicador de determinante, tipo de determinante, persona, género, número y número del posesor, con las variedades de etiquetas especificadas en la tabla 3.2. Por ejemplo, para el pronombre "nuestra" contamos con la siguiente definición.

```
nuestra,nuestro,D,P,1,F,S,P
```

Podemos observar que se define que es un determinante (D), posesivo (P), en primera persona (1), femenino (F) y singular (S), donde el posesor tiene número plural (P).

Español
Indicador de determinante: D
Tipo, A: artículo, D: demostrativo, I: indefinido, P: posesivo, T: interrogativo, E: exclamativo
Persona, 1: primera, 2: segunda, 3: tercera
Género, M: masculino, F: femenino, C: común
Número, P: plural, S: singular, N: invariable
Número del posesor, S: singular, P: plural, N: Invariable

Tabla 3.2: Conjunto de todas las etiquetas utilizadas por los determinantes para el idioma español

Los sustantivos, por su parte, tienen etiquetas simplemente de identificador, tipo de sustantivo, género y número, como se observa en la tabla 3.3.

Español
Indicador de sustantivo: N
Tipo, C: común, P: propio
Género, M: masculino, F: femenino, C: común
Número, P: plural, S: singular, N: invariable

Tabla 3.3: Conjunto de todas las etiquetas utilizadas por los sustantivos para el idioma español

Por otro lado, los pronombres cuentan con indicador de pronombre, tipo, persona, género, número, caso y formalidad (es decir, usted vs tú o vos), presentadas en la tabla 3.4. A continuación, presentamos un ejemplo de un pronombre en español, "tú", con sus características. Este corresponde a pronombre (P), personal (P), en segunda persona (2), sin género (C), del singular (S), nominativo (N), y que no indica formalidad (0).

```
tú,tú,P,P,2,C,S,N,0
```

Español
Indicador de pronombre: P
Tipo, D: demostrativo, I: indefinido, P: personal, T: interrogativo, E: exclamativo, R: Relativo
Persona, 1: primera, 2: segunda, 3: tercera
Género, M: masculino, F: femenino, C: común
Número, P: plural, S: singular, N: invariable
Caso, N: Nominativo, A: acusativo, D: dativo, O: oblicuo
Formalidad, P: formal, 0: no formal

Tabla 3.4: Conjunto de todas las etiquetas utilizadas por los pronombres para el idioma español

En cuanto a los adjetivos, se cuenta con las etiquetas morfológicas correspondientes a indicador de adjetivo, tipo de adjetivo, grado, género, número, persona del posesor y número del posesor (tabla 3.5).

Por último, contamos con diccionarios para adposiciones y conectores, donde los primeros simplemente tienen indicador de preposición (P), mientras que los conectores no cuentan con ninguna etiqueta morfológica. Esto se debe a que estos últimos fueron realizados manualmente, y no requieren de ninguna característica particular para formar las oraciones.

Transitividad

Entre las características extraídas de Freeling no se encuentran las correspondientes a la transitividad de cada verbo. Por lo tanto, para definirla, usamos por

Español
Indicador de adjetivo: A
Tipo, Q: calificativo, P: posesivo, O: ordinal
Grado, S: superlativo, V: evaluativo, 0: no corresponde
Género, M: masculino, F: femenino, C: común
Número, P: plural, S: singular, N: invariable
Persona del posesor, 1: primera, 2: segunda, 3: tercera
Número del posesor, S: singular, P: plural, N: Invariable

Tabla 3.5: Conjunto de todas las etiquetas utilizadas por los adjetivos para el idioma español

un lado el diccionario de la Real Academia Española[1], y por el otro, el lexicón de Ancora-Verbs (Subsección 2.8.3). De esta forma obtuvimos, para cada verbo, información de si eran transitivos, intransitivos o ditransitivos. Construimos un algoritmo en Python que recorre los archivos provistos por el lexicón de verbos de Ancora, y verifica las características morfológicas de cada uno. El siguiente es un ejemplo de representación XML de Ancora-Verbs para el verbo "adorar".

```
<lexentry lemma="adorar" lng="es" type="verb">
  <sense id="1">
    <frame default="yes" lss="A21.transitive-agentive-
       patient" type="default">
      <argument argument="arg0" function="suj"
         thematicrole="agt"/>
      <argument argument="arg1" function="cd"
         thematicrole="pat"/>
      <examples>
        <example>
Adoro y admiro a San Juan de la Cruz
        </example>
      </examples>
    </frame>
    <frame lss="B22.unaccusative-passive-transitive"
       type="passive">
      <argument argument="arg1" function="suj"
         thematicrole="pat"/>
      <argument argument="argM" function="cc"
         thematicrole="adv"/>
      <examples>
        <example>
el recuerdo de su adorado presidente
        </example>
      </examples>
```

[1] https://dle.rae.es/

```
20      </frame>
21    </sense>
22  </lexentry>
```

De estos archivos obtenemos los diferentes tipos de objetos que puede tener el verbo, indicados mediante `function="cd"` para el caso de objetos directos y `function="ci"` para los indirectos. En ejemplos donde el verbo tuviera objeto directo e indirecto dentro de la misma sección (indicada mediante *frame*), lo tomamos como ditransitivo; si tiene objeto directo, pero no indirecto, lo tomamos como transitivo; y en el caso de que exista algún ejemplo donde el verbo no cuente con objeto directo ni indirecto, se interpreta como intransitivo. En el ejemplo provisto, podemos observar que el verbo "adorar" cuenta con objeto directo en la primera sección, resultando transitivo, mientras que en la segunda cumple una función de verbo pasivo, por lo que no se tiene en cuenta para este proyecto.

3.1.2. Generación del lexicón en guaraní

Los vocablos en guaraní no cuentan con las mismas características morfológicas que aquellos en español, por lo tanto, definimos las más relevantes para cada uno de forma de facilitar el proceso de traducción posteriormente.

Etiquetas del guaraní

En primer lugar, para los verbos se comparten todas las etiquetas con excepción de las correspondientes a tipo de verbo y género. A su vez anexamos algunas específicas al idioma, como la inclusividad para la primer persona del plural, y la ubicación con respecto al pronombre, presentadas en la Subsección 2.1.3. Las distintas variaciones de cada etiqueta se pueden encontrar en la tabla 3.6.

Para los determinantes (presentados en la Subsección 2.1.6) resolvimos anexar las etiquetas correspondientes a inclusividad, nasalidad, conjugación en tercera persona, cuyo uso es exclusivo de la traducción del pronombre "sus" o "su", y por último la presencia o ausencia del sujeto. Observando la tabla 3.7 y el siguiente ejemplo, podemos deducir que la traducción del pronombre en español "nuestra", cuenta en guaraní con las características morfológicas correspondientes a determinante (D), demostrativo (D), en primera persona (1), común para cualquier género (C), número invariable (N, porque en guaraní no hay distinción en pluralidad en determinantes), posesor plural (P), inclusivo (I), oral (O), donde no corresponde conjugación de tercera persona (0), y no corresponde presencia o ausencia de sujeto (0).

```
ñande,ñande,D,P,1,C,N,P,I,O,0,0
```

Guaraní
Indicador de verbo: V
Modo, I: indicativo, S: subjuntivo, M: imperativo, P: participio, G: gerundio, N: infinitivo
Tiempo, P: presente, I: pretérito imperfecto, F: futuro, S: pretérito perfecto, C: condicional
Persona, 1: primera, 2: segunda, 3: tercera
Número, S: singular, P: plural
Inclusión (solo primera del plural), I: Incluyente, E: Excluyente, 0: Ninguno. Esta solo aplica a verbos primera del plural, los demás tienen 0.
Ubicación con respecto al pronombre (solo tercera persona del plural), B: Anterior al verbo, P: Posterior, 0: No corresponde
Indicador de verbo transitivo
Indicador de verbo ditransitivo
Indicador de verbo intransitivo

Tabla 3.6: Conjunto de todas las etiquetas utilizadas por los verbos para el idioma guaraní

Guaraní
Indicador de determinante: D
Tipo, A: artículo, D: demostrativo, I: indefinido, P: posesivo, T: interrogativo, E: exclamativo
Persona, 1: primera, 2: segunda, 3: tercera
Género, M: masculino, F: femenino, C: común
Número, P: plural, S: singular, N: invariable
Número del posesor, S: singular, P: plural, N: Invariable
Inclusión, I: Incluyente, E: Excluyente, 0: Ninguno
Nasalidad, O: Oral, N: Nasal, 0: Ninguno
Conjugación de tercera persona
Presencia, P: Presente, A: Ausente, 0: Ninguno

Tabla 3.7: Conjunto de todas las etiquetas utilizadas por los determinantes para el idioma guaraní

En cuanto a los sustantivos, simplemente se anexa indicador de tercera persona y nasalidad de este, como se muestra en la tabla Tabla 3.8. Contamos, además, con un pequeña variación en el indicador de género, donde en lugar de utilizar la etiqueta "C", correspondiente a "común", utilizamos la etiqueta "0", debido a que no tienen género en guaraní.

Como podemos observar en la tabla 3.9, para los pronombres no se utiliza la etiqueta indicadora de formalidad, y se agregan las correspondientes a inclusividad, posición respecto al verbo (como se explica en la Subsección 2.1.2), y nasalidad.

Para los adjetivos, solamente tomamos el indicador de adjetivo y el tipo de

Guaraní
Indicador de sustantivo: N
Tipo, C: común, P: propio
Género, 0: No tienen género en guaraní
Número, P: plural, S: singular, N: invariable
Indicador para determinante (solo para tercera persona)
Nasalidad, N: Nasal, O: Oral

Tabla 3.8: Conjunto de todas las etiquetas utilizadas por los sustantivos para el idioma guaraní

Guaraní
Indicador de pronombre: P
Tipo, D: demostrativo, I: indefinido, P: personal, T: interrogativo, E: exclamativo, R: Relativo
Persona, 1: primera, 2: segunda, 3: tercera
Género, M: masculino, F: femenino, C: común
Número, P: plural, S: singular, N: invariable
Caso, N: Nominativo, A: acusativo, D: dativo, O: oblicuo
Inclusividad, I: Incluyente, E: Excluyente, 0: Ninguno
Posición respecto al verbo, B: Ambos, P: Posterior, 0: Ninguno
Nasalidad, O: Oral, N: Nasal, 0: Ninguno

Tabla 3.9: Conjunto de todas las etiquetas utilizadas por los pronombres para el idioma guaraní

adjetivo de las etiquetas en español, siendo estas las únicas aplicadas del lado guaraní.

En las adposiciones (posposiciones en guaraní), agregamos las etiquetas morfológicas correspondientes a posición (es decir, si es una posposición que va anexada al vocablo anterior, o se escribe de forma separada), el tipo de uso (si es para verbos, sustantivos o ambos), y la nasalidad correspondiente, como se puede observar en la tabla 3.10.

Guaraní
Indicador de posposición / preposición, P
Posición, U: Unido, S: Separado
Uso, 0: Verbos y Sustantivos, V: Solo para verbos, S: Solo para sustantivos
Nasalidad, 0: Neutro, N: Nasal, O: Oral

Tabla 3.10: Conjunto de todas las etiquetas utilizadas por las adposiciones para el idioma guaraní

Finalmente, al igual que en español, no utilizamos etiquetas para los conectores. Simplemente definimos la correspondencia entre los términos de cada idioma, sin especificar ninguna etiqueta. Por ejemplo, para el conector en espa-

ñol “y”, que en guaraní le corresponde la palabra “ha”, simplemente especificamos la línea `ha,y`.

Algoritmos generadores

Una vez obtenido cada lexicón en español, procedimos a agregar las traducciones al guaraní con sus respectivas etiquetas morfológicas. Para esto, debido a que la cantidad de términos en español era mucho más amplia que la del idioma guaraní, realizamos una intersección entre ambas, contando finalmente con un diccionario de palabras con su traducción en ambos idiomas, y las características morfológicas correspondientes al idioma español.

Siendo que el resultado final de verbos, sustantivos y adjetivos es un listado extenso, decidimos crear procedimientos y algoritmos, de forma de definir automáticamente las características morfológicas de cada una. Por otro lado, los pronombres, adposiciones, determinantes y conjunciones los realizamos de forma manual, siguiendo las reglas definidas en la guía del guaraní (Academia de la Lengua Guaraní, 2018).

Los verbos provistos por el diccionario de Descubrir Corrientes, presentado en la Subsección 2.8.5 se encuentran en el modo infinitivo, por lo que para lograr un corpus con mayor variedad se decidió conjugar estos verbos según las reglas presentadas en la Subsección 2.1.3. Aunque en la intersección de los diccionarios contamos con verbos en distintos modos, y una amplia variedad de tiempos, decidimos utilizar aquellos en modo indicativo, y tiempos presente, pretérito simple, pretérito imperfecto y futuro. Esta decisión se tomó debido a que estas son las conjugaciones fácilmente asimilables al español, con el que tendremos que establecer una morfología paralela, comparado con demás modos y tiempos, además de que el objetivo se trata de generar oraciones sencillas. A medida que se conjuga cada verbo, se procesan y se guardan también las características morfológicas necesarias, según lo definido en anteriormente.

En cuanto a sustantivos y adjetivos, procedimos de igual manera que con los verbos. Es decir, primero realizamos la intersección entre los diccionarios obtenidos de las diferentes bibliotecas y luego procesamos cada línea a fines de definir las etiquetas pertinentes de cada palabra en guaraní. Para los sustantivos, en particular, también decidimos obtener la forma plural de cada uno, anexando el vocablo “kuéra”, o “nguéra” en caso de tener final nasal. Por ejemplo, contábamos con el sustantivo “ma’ẽha” (en español “mirador”), por lo que generamos a su vez el sustantivo “ma’ẽhakuéra”, que corresponde a la palabra “miradores” en español.

3.2. Gramática de rasgos

Para poder generar oraciones y construir sus árboles sintácticos en español, concebimos una gramática simplificada del español con oraciones cortas. Estas surgen de reglas que siguen la estructura de sujeto y predicado, donde el sujeto

puede ser o bien un pronombre personal o un sintagma nominal no pronominal, y el predicado es un sintagma verbal que puede ser o no seguido por un sintagma preposicional. Además, ese sintagma verbal puede ser o no negado, y su estructura varía en función de la transitividad del verbo en cuestión. Por otro lado, los sintagmas nominales pueden o no incluir adjetivos.

Estas reglas, a las que denominamos producciones gramaticales, se combinan con las producciones léxicas (que surgen del lexicón presentado en la sección anterior) para formar una gramática completa.

Dada la riqueza del idioma español en conjugaciones verbales, personas y géneros gramaticales, y la fuerte concordancia que se exige entre distintos componentes de una oración, decidimos implementar una gramática de rasgos que permita tomar tales aspectos en cuenta. Si bien cuenta con algunas diferencias importantes, nuestra gramática se inspira en las HPSG, tanto en su funcionamiento como en los rasgos que la componen y en su concepto de unificación.. Por otro lado, nuestra gramática sigue el formato de una *feature grammar* de NLTK, incluyendo el uso de variables como $?a$ para explicitar la concordancia en un cierto rasgo entre elementos del lado derecho de la regla. Seguir este formato es importante para nuestro uso de NLTK a la hora de construir árboles sintácticos usando nuestra gramática.

En la especificación de las reglas hacemos uso de la noción de concordancia sintáctica (agreement), forzando por ejemplo que la persona gramatical del sujeto y del núcleo verbal del predicado coincidan. También hacemos referencia a los rasgos para filtrar nuestra selección de vocabulario a partir del lexicón que produjimos anteriormente, por ejemplo limitando los pronombres que admitimos como sujeto a aquellos con caso nominativo.

Específicamente, las producciones gramaticales expanden los siguientes símbolos:

- S: símbolo inicial
- NP: sintagma nominal
- VP: sintagma verbal
- PP: sintagma preposicional
- PPA: caso especial del sintagma preposicional, que introduce el objeto indirecto de un verbo ditransitivo

Por otro lado, hacemos uso en las producciones gramaticales de los siguientes rasgos:

- AGR: Agreement, es el principal rasgo en las reglas y se utiliza para exigir concordancia sintáctica entre distintos componentes (ej. Entre determinante y sustantivo). Este será compuesto por otros rasgos a nivel de producciones léxicas.
- CASE: determina el caso de un pronombre, ej. nominativo (él, tú), acusativo (lo, la), dativo (le, les), oblicuo (mí, ti).

- MOOD, SUBCAT: determinan el modo y transitividad de un verbo. Particularmente, la transitividad del verbo define la presencia de los objetos directo e indirecto en el predicado, mientras que el modo se usa para remitirnos al modo indicativo, debido a las complejidades que presenta la comparativa de modos entre español y guaraní.

Por otro lado, como mencionamos anteriormente, las producciones léxicas son aquellas que derivan en terminales; en este caso, en palabras comprendidas en nuestro lexicón. Los símbolos que esta clase de reglas pueden expandir se observan en la tabla 3.11, junto a sus posibles rasgos. La mayoría de los símbolos corresponden a categorías gramaticales o *parts of speech*, con algunas excepciones; particularmente el adverbio "no" y la partícula "a" cuando introduce un objeto indirecto. A su vez, los rasgos se corresponden en gran medida con los de nuestro lexicón, presentados en la sección 3.1. A la hora de efectivamente construir la gramática, generamos las producciones léxicas en función del contenido de nuestro lexicón, construyendo para cada entrada una regla equivalente.

Símbolo	**Rasgos**	**Descripción/componentes**
V: verbo	AGR: agreement, compuesto	NUM: Número
		PER: Persona
		TENSE: Tiempo verbal
	MOOD	Modo
	SUBCAT	Transitividad
N: sustantivo	AGR	NUM: Número
		PER: Persona
		GEN: Género
D: determinante	AGR	NUM: Número
		GEN: Género
	TYPE	Tipo
	POSSPER	Persona del posesor
	POSSNUM	Número del posesor
P: pronombre	AGR	NUM: Número
		PER: Persona
		GEN: Género
	TYPE	Tipo
	CASE	Caso
	POLITE	Formalidad
A: adjetivo	AGR	NUM: Número
		GEN: Género
PR	Preposición	
NEG	"no", adverbio de negación	
AA	"a", preposición de objeto indirecto	

Tabla 3.11: Símbolos que pueden ser expandidos por producciones léxicas, junto con sus rasgos

A modo ilustrativo, se presenta en el cuadro 3.1 un pequeño extracto de la gramática desarrollada, correspondiente a las reglas más básicas de un enunciado

con sujeto pronominal, con y sin sintagma preposicional en el predicado. A su vez, en el cuadro 3.2 se observan algunos ejemplos de nuestras producciones léxicas, para un pronombre y un verbo. En el anexo B.1 se incluye una versión más completa de la gramática utilizada.

```
# S expansion productions
S[AGR=?a] -> P[AGR=?a, CASE=n] VP[AGR=?a, MOOD=i]
S[AGR=?a] -> P[AGR=?a, CASE=n] VP[AGR=?a, MOOD=i] PP
```

Listing 3.1: Ejemplo de producciones gramaticales. Estas dos producciones indican que una oración se puede formar de un pronombre (P) de caso nominativo, seguido de un sintagma verbal (VP) de modo indicativo, y opcionalmente puede haber un sintagma preposicional (PP) como se indica en la segunda línea. A su vez, se asigna la misma variable a los rasgos AGR del pronombre y el sintagma verbal, por lo que estas estructuras deben concordar.

```
P[AGR=[ GEN=f, NUM=s, PER=3], TYPE=p, CASE=n, POLITE=0]
    -> 'ella'
V[AGR=[NUM=s, PER=3, TENSE=s], MOOD=i, SUBCAT=intr] -> '
    reflexionó'
```

Listing 3.2: Ejemplo de producciones léxicas. La primer producción refiere al pronombre *vos*, con género femenino, número singular y tercera persona, tipo personal, caso nominativo y sin formalidad (los pronombres formales son *usted* y *ustedes*). La segunda producción corresponde al verbo *reflexionó*, conjugado en tercera persona del singular, pretérito simple, modo indicativo, y se especifica que es intransitivo.

En base a todo lo anteriormente presentado en esta sección, desarrollamos un programa en Python que, dados los archivos que contienen el lexicón a utilizar, genera un archivo de texto con una representación en texto plano de la gramática producida. Esta sigue un formato que puede ser directamente interpretado por NLTK como una *featgram* (gramática de rasgos), de forma que se pueda usar NLTK para parsear oraciones en árboles sintácticos más adelante.

Varias diferencias importantes distinguen a nuestra gramática de una HPSG. En primer lugar, la nuestra es una gramática generativa en vez de basada en restricciones; particularmente siguiendo la noción de *feature grammar* implementada por NLTK que extiende a las gramáticas libres de contexto con estructuras de rasgos. En segundo lugar, los rasgos no se condicen uno a uno con los típicos de una HPSG (si bien se inspiran en ellos); por ejemplo, nuestra gramática carece de rasgos correspondientes a la semántica. Finalmente, nuestra gramática carece del concepto de "núcleo" (*head*), que es fundamental para una HPSG; en su lugar, por ejemplo, la naturaleza combinatoria de la valencia se ve reemplazada por lo que las propias producciones estipulan, combinando símbolos para formar otros, y la herencia de características morfológicas se da según lo que especifican las variables del lado izquierdo de la regla en lugar de depender del núcleo del sintagma.

Decidimos no modelar una gramática completa HPSG, ni utilizar alguna de las existentes para el español, porque nuestro objetivo es generar un conjunto de oraciones relativamente simples. La gramática que construimos puede usarse como generadora de oraciones simples al convertirla a una gramática libre de contexto, algo en lo que profundizaremos en la siguiente sección.

3.3. Generación en español

En esta sección tratamos el problema de la generación de un corpus sintético en español en base a la gramática desarrollada, de forma que sea posible traducirlo más adelante para obtener un corpus bilingüe. No es posible generar oraciones directamente a partir de la gramática de rasgos definida en la sección anterior, por lo que diseñamos una estrategia para poder construir una gran cantidad de oraciones candidatas y luego filtrarlas usando la gramática. Nuestro proceso consiste en:

- Transformar la gramática de rasgos en una gramática libre de contexto.
- Generar una gran cantidad de oraciones en español en base a esta.
- Filtrar aquellas que no cumplen con todas las restricciones impuestas por la gramática de rasgos.
- Construir al menos un árbol sintáctico para cada una, preservando estos para la siguiente etapa de transferencia sintáctica al guaraní.

A continuación detallaremos cada paso de este proceso.

3.3.1. Transformación a gramática libre de contexto

Antes de poder avanzar con el proceso de generación, es necesario contar una gramática libre de contexto. Para esto, escribimos un script que toma la gramática de rasgos, en su formato de string, y remueve todas las referencias a los rasgos, resultando de esa forma en una gramática libre de contexto. Por ejemplo, las siguientes reglas de producción con rasgos:

```
S[AGR=?a] -> NP[AGR=?a] VP[AGR=?a, MOOD=i]
V[AGR=[NUM=s, PER=3, TENSE=p], MOOD=i, SUBCAT=tr] -> '
    abandona'
```

Se convierten en las siguientes reglas libres de contexto simplificadas:

```
S -> NP VP
V -> 'abandona'
```

Naturalmente, la gramática libre de contexto resultante genera un superconjunto de todas las oraciones posibles de la gramática de rasgos, incluyendo oraciones que no son gramaticales según esta última. Por esa razón la usamos para una generación primaria, procediendo luego a filtrar según la gramática más estricta.

3.3.2. Asignación de pesos al vocabulario

Para generar oraciones con un vocabulario que se asemeje a aquel usado a lo largo del corpus Jojajovai, asignamos pesos a las producciones léxicas en la gramática libre de contexto, basados en las ocurrencias de los elementos de nuestro lexicón en la partición de entrenamiento del corpus. Estos determinarán la probabilidad de que el algoritmo aplique una regla por sobre las demás.

Con este fin, desarrollamos un script que itera sobre el corpus y el lexicón, resultando en una tabla con las columnas: categoría gramatical, palabra, ocurrencias exactas de la palabra, *lema*, y *ocurrencias totales*. *Lema* refiere a la forma que nos encontraríamos como entrada en un diccionario tradicional, como ser masculino singular para sustantivos, o infinitivo para verbos; *ocurrencias totales* representa la suma de las ocurrencias de todas las variantes de ese lema, como pueden ser todas las conjugaciones de un mismo verbo.

Finalmente, la fórmula de los pesos asignados a cada palabra consiste en $\log_2(1{,}5 + ocurrencias\_totales)$. Esta surge de un análisis experimental, donde mediante la inspección manual de los resultados de pequeñas tandas generadas se fue ajustando los parámetros para regularizar la aparición de palabras muy poco probables en el español paraguayo, sin que derive en una falta de variedad ni en términos que dominaran significativamente a lo largo de las oraciones generadas.

3.3.3. Algoritmo de generación

La biblioteca NLTK no ofrece una forma de generar oraciones al azar a partir de una gramática dada. Frente a esta necesidad, decidimos implementar un algoritmo propio. En este sentido, primero generamos las oraciones en función a la gramática libre de contexto, previo a realizar un proceso de filtrado.

Para el paso de generación de oraciones nos inspiramos en el algoritmo propuesto por Eli Bendersky[2], desarrollando una clase CFG que representa la gramática. En función del archivo previamente generado con la gramática libre de contexto y los pesos de cada regla, se van agregando reglas de producción con pesos asignados a la clase CFG. Esta clase implementa un método de generación de oraciones que consiste en, partiendo de un símbolo inicial, aplicar en cada paso para el símbolo actual una de las reglas posibles según las probabilidades que sus pesos definen.

Como un detalle de implementación, debido al gran número de combinaciones posibles de tokens al que da lugar la gramática libre de contexto, resulta conveniente dividir esta gramática en varias más pequeñas que generan subconjuntos menores de lo que generaría su unión (por ejemplo, separar la gramática que genera oraciones con sujeto pronominal de aquellas cuyo sujeto es un sintagma nominal genérico). Con esto en mente, el programa que escribimos para generar gramáticas (descrito en la sección 3.2) toma como argumento, además de los lexicones a utilizar, una bandera indicando el tipo de gramática que se

[2] https://eli.thegreenplace.net/2010/01/28/generating-random-sentences-from-a-context-free-grammar

desee. Los ejes de variación son: sujeto pronominal o no pronominal, presencia o no de adjetivos en los sintagmas nominales, presencia o no de sintagmas preposicionales. Esto permite eliminar elementos innecesarios de la gramática; por ejemplo, para el caso de la gramática generadora de oraciones con sujeto no pronominal, se omiten todas las formas verbales correspondientes a la primer y segunda persona gramatical, ya que el sujeto siempre estará en tercera persona (ej. "el amor", "una casa grande"). Esto se traduce en una mejora sustancial de la eficiencia en la etapa de generación, y a nivel de uso del sistema implica generar oraciones de forma independiente con cada gramática para unir los resultados en un corpus final. Por ende, en gran parte de esta sección, cuando hablamos de "la gramática" estamos haciendo referencia a una subgramática de las obtenidas anteriormente, donde los pasos descritos se aplican de forma independiente para cada una de ellas.

3.3.4. Filtrado y construcción de los árboles sintácticos

Para filtrar las oraciones generadas según el proceso anteriormente descrito, que no tienen en cuenta las restricciones de concordancia que surgen de la gramática de rasgos, decidimos usar las clases *FeatureGrammar* y *FeatureChartParser* de NLTK. La primera admite nuestra representación de la gramática en forma de string para crear un objeto que representa la gramática de rasgos, mientras que la segunda provee la funcionalidad de parsear oraciones en función de la gramática de rasgos provista.

Para cada oración que generamos en el paso anterior, intentamos entonces parsearla según nuestra gramática de rasgos, descartando aquellas que no demuestren concordancia total. La inmensa mayoría de las oraciones son descartadas en este paso, que representa el cuello de botella en cuanto a la generación de un corpus sintético en español. Sin embargo, de esta forma es que eliminamos enunciados con errores gramaticales como "el perras duerme" en lugar de "la perra duerme", o "yo pensábamos" en lugar de "yo pensaba".

El método de parsing provisto por FeatureChartParser ofrece la posibilidad de retornar uno o varios árboles sintácticos para la oración provista; en nuestro caso, decidimos generar todos los árboles posibles, y registrarlos como distintas interpretaciones de la misma oración, que derivarían en distintas traducciones posibles al guaraní. El principal artefacto resultante de la etapa de generación es entonces un archivo de texto con representaciones de árboles sintácticos de oraciones en español, con sus *parts of speech* y rasgos pertinentes. El siguiente es un ejemplo de árbol sintáctico construido mediante nuestro proceso, correspondiente a la oración "tus participantes no existían":

```
(S[AGR=[GEN=c, NUM=p, PER=3, TENSE=i]]
  (NP[AGR=[GEN=c, NUM=p, PER=3]]
    (D[AGR=[GEN=c, NUM=p], POSSNUM=s, POSSPER=2, TYPE=p] '
        tus')
    (N[AGR=[GEN=c, NUM=p, PER=3]] 'participantes'))
  (VP[AGR=[NUM=p, PER=3, TENSE=i], MOOD=i]
    (NEG[] 'no')
```

```
7       (V[AGR=[NUM=p, PER=3, TENSE=i], MOOD=i, SUBCAT=intr]
8         'existían')))
```

De estos árboles es posible extraer la oración final, pero sobre todo la información sintáctica que contienen es crucial para la etapa de transferencia sintáctica al guaraní.

3.4. Traducción al guaraní

Para lograr la traducción de oraciones en español a guaraní, decidimos implementar un programa en Python, el cual procesa los árboles sintácticos presentado en la sección anterior, el lexicón bilingüe desarrollado en la sección 3.1, y las reglas que definiremos para la transferencia sintáctica entre español y guaraní. Este programa aplica un algoritmo de transferencia sintáctica donde recorre los árboles desde sus hojas (las palabras) hacia la raíz, intentando en cada paso transformar los subárboles al guaraní según las reglas indicadas, y unificar los rasgos donde sea necesario, para obtener finalmente un árbol sintáctico en guaraní. Finalmente, obtenemos un corpus con oraciones en español junto con sus traducciones correspondientes al guaraní.

3.4.1. Rasgos del guaraní

Con el objetivo de traducir las oraciones de manera coherente entre ambos idiomas, intentando no perder información, definimos rasgos característicos del guaraní para sus reglas.

- **POS:** Indica la posición del pronombre con respecto al verbo.
- **NF:** Indica la nasalidad del vocablo, específicamente, la nasalidad de la última sílaba de este.
- **NEG:** Indica si un verbo se encuentra en su forma negativa (Subsección 2.1.4).
- **U:** Indica la posición de una posposición con respecto al verbo (unida o separada).
- **S:** Indica si una posposición se usa para verbos, sustantivos o ambos.

A su vez, también se definieron otros rasgos cuyo uso es exclusivo dentro del grupo AGR.

- **NAS:** Indica la nasalidad del vocablo completo.
- **INC:** Indica si el vocablo corresponde a la forma inclusiva o exclusiva de la primera persona del plural (Subsección 2.1.3).
- **TER:** Indica el tipo de conjugación de tercera persona del plural.
- **PRES:** Indica presencia o ausencia en determinantes (Subsección 2.1.3).

3.4.2. Reglas de transferencia

Para llevar adelante el proceso de transferencia sintáctica, diseñamos un conjunto de reglas de transferencia que facilitan la traducción entre guaraní y español. Definimos un diccionario clave-valor, donde cada clave representa una producción sintáctica en español y su valor es una lista de posibles transformaciones a una producción equivalente en el idioma guaraní. Las reglas siguen el siguiente formato:

Producción de español → [*Producción de guaraní 1*, *Producción de guaraní 2*, ...]

Como se puede ver, a cada clave (regla del español) le puede corresponder una o varias reglas del guaraní, debido a que puede haber varias formas de reestructurar la oración manteniendo su significado. Por ejemplo, en la regla mostrada a continuación se muestran diferentes traducciones posibles para las oraciones de forma "S → P VP" en español.

```
S -> P VP: [
    S[AGR='?i'] -> P[AGR='?i', POS='B'] VP[AGR='?i', POS='B'],
    S[AGR='?i'] -> VP[AGR='?i', POS='P'] P[AGR='?i', POS='P']
]
```

En este caso, esto se debe al hecho de que, como se explica en la Subsección 2.1.2, en guaraní los pronombres pueden ser anteriores o posteriores al verbo. Como se puede observar en las reglas, la diferencia entre las opciones de traducción está en la posición del pronombre, indicando a su vez que, si el pronombre tiene la característica de posición igual a "B", este es anterior, y si es igual a "P" es posterior. Además, indicamos que el verbo debe estar en su conjugación correspondiente al pronombre de la regla (anterior o posterior). Cada regla especifica los rasgos sobre los atributos gramaticales, como AGR, POS y otros, para garantizar una conversión precisa. Al igual que sucede con la gramática presentada en la Sección 3.2, la presencia de variables como "`?i`" indica qué rasgos deben coincidir entre determinados vocablos y si estos serán heredados por el símbolo del lado izquierdo de la regla.

En el anexo B.2 se incluyen las reglas de transferencia utilizadas.

3.4.3. Traducción mediante transferencia sintáctica

El proceso de traducción comprende principalmente de dos pasos, donde el primero se encarga de procesar los archivos de entrada, particularmente los árboles, y el segundo traduce cada árbol al guaraní. Entre estos logramos llamar a los archivos con árboles de oraciones en español y reglas de transferencia sintáctica entre español y guaraní, procesar los datos para traducir los árboles, y

devolver los pares de oraciones finales.

En primer lugar se procesan los archivos definidos anteriormente, es decir, el conjunto de árboles sintácticos generados en español, el lexicón bilingüe, y el conjunto de reglas para la transferencia sintáctica entre ambos idiomas. Particularmente, al procesar los árboles, desarrollamos una implementación propia del concepto de árbol sintáctico, con nociones de *símbolo*, *rasgos*, *hijos*, y *palabra* (en el caso de las hojas), que nos permitiera llevar a cabo los pasos de transferencia entre un árbol y otro de forma directa.

Sobre cada uno de estos árboles, realizamos la transferencia sintáctica del árbol en español a los posibles árboles en guaraní con las reglas definidas. El algoritmo en cuestión recibe árbol gramatical correspondiente a una oración, las reglas de transferencia sintáctica, y nuestro lexicón bilingüe, y procede a recorrer el árbol de forma recursiva. En cada paso, se llama recursivamente sobre los hijos del nodo actual, almacenando las traducciones obtenidas para cada hijo. Por ende, los primeros nodos en ser traducidos serán las hojas, que se corresponden a las categorías gramaticales simples como lo son los sustantivos, los verbos o los determinantes.

Por ejemplo, supongamos que tenemos el sintagma nominal en español, "Nuestra amistad". Un árbol correspondiente a este puede ser el siguiente.

```
(NP[AGR=[GEN=f, NUM=s, PER=3]]
    (D[AGR=[GEN=f, NUM=s], POSSNUM=p, POSSPER=1, TYPE=p]
      'nuestra')
    (N[AGR=[GEN=f, NUM=s, PER=3]] 'amistad'))
```

Allí observamos que el determinante tiene los rasgos indicadores de número y persona del posesor, y tipo de determinante, junto con género y número dentro del agreement. El sustantivo por su parte cuenta con los rasgos distintivos de género, número y persona, también dentro del agreement.

Como mencionamos anteriormente, las primeras traducciones se hacen sobre las hojas de la oración, en este caso "amistad" y "nuestra". Para esto, nuestro programa busca estas palabras en nuestro lexicón, filtrando no solo por el string de la palabra sino por su categoría gramatical y sus rasgos (por ejemplo, para distinguir homófonos). Continuando nuestro ejemplo, al buscar en el diccionario correspondiente obtenemos las siguientes traducciones:

Nuestra

- Ñande (inclusivo, oral) `[AGR:[‘PER’:1,’NUM’:N,‘POSS’:P,‘INC’:I, ‘NAS’:O,‘PRES’:0] ]`
- Ñane (inclusivo, nasal) `[AGR:[‘PER’:1,‘NUM’:N,‘POSS’:P,‘INC’:I, ‘NAS’:N,‘PRES’:0] ]`
- Ore (exclusivo, nasalidad indiferente) `[AGR:[‘PER’:1,‘NUM’:N,‘POSS’: P,‘INC’:E,‘NAS’:0,‘PRES’:0] ]`

Amistad

- Joayhu (oral) `[AGR:[‘NAS’:O, ‘TER’:4], ‘NF’:O]`
- Ñoirũ (nasal) `[AGR:[‘NAS’:N, ‘TER’:4], ‘NF’:N]`

Para cada hoja, entonces, el programa devuelve una lista de pares, compuestos por la palabra y su estructura de rasgos, que representan las posibles traducciones.

Una vez procesadas todas las hojas, la recursión eventualmente vuelve a un *part of speech* compuesto de más de una categoría gramatical; es decir, a un nodo con hijos, como sería `NP` en nuestro ejemplo. En este punto, contamos con las posibles traducciones para cada hijo. Por otro lado, a partir del nodo y sus hijos es posible determinar la producción gramatical en español que fue utilizada en la construcción del árbol, para luego consultar en las reglas de transferencia qué posibles producciones gramaticales del guaraní podrían aplicarse.

En nuestro ejemplo, la producción utilizada fue `NP` $\rightarrow$ `D N`, y la regla de transferencia establece que su equivalente en guaraní es `NP[AGR=‘?a’, NF=’?n’]` $\rightarrow$ `D[AGR=‘?a’] N[AGR=‘?a’, NF=‘?n’]`. En esta podemos ver que tanto el determinante como el sustantivo deben coincidir en los rasgos que tengan en el agreement, y que el sintagma nominal final heredará tanto este como la nasalidad de la última sílaba del sustantivo. En este caso, solo hay una regla equivalente en guaraní, y el orden de los símbolos se mantiene, pero esto no siempre es así. Por ejemplo, para las oraciones con forma `S` $\rightarrow$ `P VP`, existen dos versiones en guaraní: una donde el pronombre precede al sintagma verbal, y otra donde lo sucede.

A continuación, y ya contando con las traducciones de cada hijo del nodo actual, el algoritmo considera cada regla equivalente en guaraní, analizando las distintas posibles traducciones que resultarían de aplicar cada una. Es decir, para cada regla, que potencialmente define un orden diferente de los símbolos, se toma el producto cartesiano de las posibles traducciones del primer símbolo, con las posibles traducciones del segundo símbolo, y así sucesivamente. Así, se obtiene una lista de potenciales traducciones para el nodo actual, que luego filtraremos según si sus rasgos concuerdan con lo establecido por la regla sintáctica en guaraní.

En nuestro ejemplo se generan entonces las siguientes posibilidades:

- “ñande joayhu”
- “ñande ñoirũ”
- “ñane joayhu”
- “ñane ñoirũ”
- “ore joayhu”
- “ore ñoirũ”

Luego, para cada potencial traducción se comprueba que cumpla con lo establecido en la regla gramatical correspondiente; es decir, que las estructuras de rasgos de cada componente sean compatibles entre sí. Desarrollamos nuestra propia implementación de un algoritmo de unificación (concepto definido en la subsección 2.5.2), que basado en los rasgos y variables presentes en la regla gramatical guaraní, determina si las estructuras de rasgos son compatibles, y

retorna la estructura que heredará el *part of speech* del lado izquierdo de la regla.

Siguiendo nuestro ejemplo, determinaremos cuáles traducciones son sintácticamente correctas en función de la concordancia de sus rasgos según la regla gramatical en guaraní que presentamos anteriormente. En este caso, ambas palabras deben coincidir únicamente en el rasgo agreement, por lo que se aplicará el proceso de unificación sobre ese rasgo, y no otros. Particularmente, descartaremos aquellas traducciones donde se intente utilizar un determinante oral con un sustantivo nasal, y viceversa. Además, construimos la estructura de rasgos que heredará la traducción en cada caso.

Una vez filtradas las traducciones, obtenemos cuatro posibilidades para traducir el sintagma nominal en nuestro ejemplo: "ñande joayhu", "ñane ñoirũ", "ore joayhu" y "ore ñoirũ", cada una con sus rasgos correspondientes según la regla, es decir, con agreement (en este caso solo nasalidad, dado que es el único compartido) e indicador de nasalidad en la última sílaba del sustantivo (heredado de este). Para la primer traducción del ejemplo anterior ("ñande joayhu"), obtenemos el siguiente árbol sintáctico.

```
(NP[AGR=[NAS='O'], NF='O']
    (D[AGR=[PER='1', NUM='N', POSS='P', INC='I', NAS='O',
        PRES='O']]
      ñande)
    (N[AGR=[NAS='O'], TER='4', NF='O'] joayhu))
```

De esta forma, vamos recorriendo el árbol en español de las hojas hacia arriba, obteniendo la traducción de cada sintagma necesario en guaraní hasta llegar a formar la oración final.

Finalmente, generamos un corpus, donde cada línea consiste de la oración original en español, seguida de su traducción al guaraní. Cabe destacar, que es posible que una oración en español genere varias traducciones en guaraní; en este caso es posible definir antes de ejecutar el programa la cantidad máxima de traducciones deseadas, las cuales son seleccionadas de forma aleatoria. A continuación presentamos un ejemplo de cuatro líneas, con los resultados previos a realizarse el postprocesamiento correspondiente, detallado en la Sección 3.5.

- yo no volvía,che ndajevyimi *
- tu adolescente creció,ne mitãrusu okakuaakuri *
- nuestra ironía no encoge de esta pieza,ore ñembohory nomocha'ĩri * ko kotypy _gua
- ellas pasarán,*hikuái ojehuta *

3.5. Post procesamiento del corpus generado

Una vez definidas las oraciones en español y sus respectivas traducciones, procedimos a realizar un post procesamiento del corpus. Esto es con el obje-

tivo de “limpiar” lo generado, y lograr representaciones más exactas de cada idioma. Entre las actividades realizadas podemos encontrar la unificación de “a el” como “al”, por ejemplo, se cambia la oración “sube a el avión” por “sube al avión”; además agregamos puntos al final de cada una, mayúsculas al comienzo, y eliminamos los espacios extras que pudieran haberse generado.

Al traducir las oraciones, algunas características del guaraní no son implementadas directamente con el traductor, sino que este utiliza una serie de *marcas* (caracteres con un significado especial) que son interpretadas y removidas posteriormente. Un caso para el cual es necesario se debe a que, en guaraní, las posposiciones pueden aparecer tanto unidas como separadas del verbo, dependiendo del caso. Para lograr esto, al definir el diccionario se anexó una barra baja (_) al comienzo de los vocablos que se unen a los verbos, para poder efectivamente distinguir esa clase de posposiciones y unirlas al verbo que las precede durante el post procesamiento. Por ejemplo, en el caso de la posposición “pe”, que corresponde con “a” en español, la registramos como “ _pe”. La unión de esa partícula con el vocablo que la precede la realizamos en el post procesado de oraciones.

Otro caso es con los pronombres de la tercera persona del plural. Como se especifica en la sección 2.1.2, en guaraní se dan dos posibles casos, uno con posición anterior al verbo y otro posterior al verbo. Resolvimos que todos se escribieran de forma anterior, pero dejando marcas en caso de que fueran posteriores. Estas marcas (asteriscos), permiten mover los pronombres según corresponda al procesar las traducciones. Al procesar las hojas, le anexamos un asterisco a todos los verbos, y luego el pronombre “hikuái” lo registramos de la forma “*hikuái” en el diccionario. De esta forma, cuando el postprocesador se encuentra con una oración de la forma “*hikuái verbo * . . .”, la transforma en “Verbo hikuái . . .”.

Volviendo a los ejemplos de la sección anterior, una vez aplicado el postprocesamiento, obtenemos el siguiente resultado.

- Yo no volvía.,Che ndajevyimi.
- Tu adolescente creció.,Ne mitãrusu okakuaakuri.
- Nuestra ironía no encoge de esta pieza.,Ore ñembohory nomocha’ĩri ko kotypygua.
- Ellas pasarán.,Ojehuta hikuái.

3.6. Construcción del corpus bilingüe basado en Ancora

Nuestro método de generación se preocupa de generar oraciones sintácticamente válidas, pero no necesariamente las oraciones tendrán sentido en el mundo real; si bien oraciones como “ellas presentaban aquella lluvia” o “él no apretará nuestras sopas” cumplen con la gramática del español, son oraciones poco probables en un uso real (humano) del idioma. Para obtener oraciones que sean más

fácticas, una posible alternativa sería entonces partir de un corpus existente y tomado de textos reales en español. Si bien perdemos el control sobre el largo de las oraciones (donde favorecíamos oraciones cortas, que era donde los modelos anteriores flaqueaban) y la diversidad de formas sintácticas (oraciones en primera y segunda persona, etc) y vocabulario utilizados, ganamos en el ámbito semántico al comparar con un corpus completamente sintético.

En vista de que es posible reutilizar el sistema de transferencia sintáctica que desarrollamos, con pequeñas modificaciones, para textos que no hayamos generado sintéticamente, es que decidimos tomar un corpus existente y desarrollar un proceso que involucre nuestro sistema de transferencia sintáctica.

En este sentido, tomamos el corpus de Ancora (Subsección 2.8.3) como punto de partida. Este es un corpus de textos anotados para el idioma español, que contiene alrededor de 500,000 palabras, constituido fundamentalmente por textos periodísticos. Por ende, es de esperar que la forma de las oraciones se asemeje más a aquella que se encuentra en el Jojajovai, que también cuenta con mucho texto de origen periodístico. Si bien este corpus está anotado con información sintáctica y semántica, decidimos extraer y tratar directamente con las oraciones, dado que nuestro sistema se ajusta bien a trabajar con texto sin anotar.

A grandes rasgos, modificamos ligeramente la gramática de rasgos que usamos en la sección anterior para que se ajuste a nuestro nuevo objetivo; extrajimos las oraciones del corpus de Ancora, sin información sintáctica ni semántica; parseamos todas las suboraciones no-superpuestas que fuera posible, priorizando árboles sintácticos de mayor tamaño por sobre los de menor tamaño; aplicamos nuestro proceso de transferencia sintáctica a estos árboles para traducir las suboraciones obtenidas; y finalmente incrustamos estas traducciones en la oración original, reemplazando el texto en español por texto guaraní donde fuera posible.

El resultado de este proceso es un corpus bilingüe donde las traducciones alternan vocabulario español y guaraní, aplicando conceptos de gramática guaraní como pueden ser las posposiciones. De esta manera el resultado son oraciones con un frecuente cambio de código (*code-switching*) entre ambos idiomas, que se asemeja en cierta medida al jopará que podemos encontrar en el habla usual en Paraguay, y en particular en el corpus Jojajovai.

3.6.1. Gramática ajustada

La gramática utilizada está basada en aquella que construimos para la generación del corpus sintético, con cuatro diferencias.

En primer lugar, el lexicón que referenciamos para crear las producciones léxicas incluye palabras cuya traducción al guaraní no es conocida por nuestro sistema. Recordemos que anteriormente considerábamos como vocabulario español la intersección de nuestro lexicón en español, basado en la información obtenida de Freeling, el Ancora Lex y el diccionario de la RAE, con el vocabulario del diccionario bilingüe de Descubrir Corrientes. En este nuevo escenario, consideramos directamente el lexicón en español sin intersectar, puesto que si

una palabra no tiene traducción conocida simplemente mantendremos su forma en español.

La segunda diferencia es que no dividimos la gramática en subgramáticas para trabajar con ella. Esta era una decisión que habíamos tomado para el proceso de generación del corpus sintético, debido al requerimiento de eficiencia que implicaba la necesidad de parsear millones de oraciones autogeneradas, y que nuestro control sobre las oraciones que serían parseadas nos permitía particionar la gramática de esta forma. En este caso sin embargo se desea usar la gramática en su totalidad para parsear oraciones de la mayor variedad posible.

En tercer lugar, agregamos un nuevo *part of speech*: los conectores, tales como *y*, *o*, *pero* y *porque*. Esto nos permitirá reemplazarlos por sus equivalentes en guaraní durante el proceso de traducción.

La cuarta y última diferencia es que agregamos producciones a partir del símbolo inicial S con el fin de flexibilizar la gramática. Esto nos permitirá más adelante parsear segmentos de oraciones correspondientes, por ejemplo, a sintagmas verbales o nominales, sin la necesidad de parsear la oración completa. Es decir, pasamos a tener las siguientes producciones iniciales:

```
# Original S expansion productions
S[AGR=?a] -> NP[AGR=?a] VP[AGR=?a, MOOD=i]
S[AGR=?a] -> NP[AGR=?a] VP[AGR=?a, MOOD=i] PP
S[AGR=?a] -> P[AGR=?a, CASE=n] VP[AGR=?a, MOOD=i]
S[AGR=?a] -> P[AGR=?a, CASE=n] VP[AGR=?a, MOOD=i] PP
# Productions added for extra Ancora matching
S -> VP
S -> VP PP
S -> NP
S -> N
S -> A
S -> PP
S -> CON
```

La gramática en cuestión se incluye en el anexo B.1.

3.6.2. Extracción y preprocesamiento de las oraciones

Para extraer las oraciones del corpus de Ancora, desarrollamos un script que toma solo los enunciados, ignorando las anotaciones sintácticas y semánticas. Este mismo script implementa un preprocesado de las oraciones, principalmente descomponiendo contracciones ("del" → "de el", "al" → "a el"), y lidiando con cuestiones de puntuación. En cuanto a esto último, es necesario separar las palabras de los signos de puntuación, pero teniendo en cuenta casos borde como la puntuación numérica (ej. un millón como "1.000.000"), y agregando marcas que permitan recomponer el texto al finalizar el proceso de traducción (ej. distinguiendo comillas iniciales de finales, si bien el signo es el mismo).

3.6.3. Parsing de suboraciones y construcción de árboles sintácticos

Desarrollamos un programa que toma como entrada las oraciones preprocesadas y la gramática de rasgos a utilizar, y cuya salida es una lista de árboles sintácticos, y una lista de índices que asocian cada árbol a una oración y a una posición inicial y final dentro de esa oración; el substring delimitado por tales índices corresponde al segmento de oración que fue parseado y cuyo árbol sintáctico fue construido.

El foco fue obtener todos los árboles sintácticos que fuera posible construir, correspondientes a substrings no superpuestos, con el criterio de priorizar el parsing de los substrings de mayor largo posible. En este sentido, el algoritmo que diseñamos consta de dos partes.

Primero, divide cada oración en suboraciones, separando según palabras que no pertenezcan al "índice léxico" (*lexical index*) de la gramática. Este es un atributo de la FeatureGrammar de NLTK que creamos a partir de nuestra representación en texto plano de la gramática, y consiste en un dict de Python con el conjunto de todas las palabras que es posible generar a partir de la gramática. Es decir, que estamos separando la oración original en suboraciones partiendo por tokens desconocidos. Esto también implica separar por puntuación, dado que la misma no forma parte del lexicón; por ejemplo, si un texto tiene dos partes separadas por una coma, el algoritmo considerará al menos dos suboraciones.

Segundo, definimos una función recursiva que busca cumplir con nuestro cometido de construir cuantos árboles sintácticos sea posible para segmentos no superpuestos de una oración, priorizando parsear segmentos de mayor largo, y aplicamos esta función a cada suboración obtenida en el paso anterior. Esta, además, lleva registro de los índices dentro de la suboración para cada árbol generado para poder embeber las traducciones más adelante. A continuación presentamos un pseudocódigo para esta función:

```
1:  function MAX_TREE(sub_sentence, feature_parser)
2:      words ← divide sub_sentence en palabras
3:      substrings ← [ ]
4:      result ← [ ]
            ▷ Generar todos los posibles segmentos de oración, comenzando por
        los más largos, ej. un perro ladra = [un perro ladra, un perro, perro ladra,
        un, perro, ladra]
5:      for j ← longitud de words downto 1 do
6:          for i ← 0 to longitud de words − j do
7:              substring ← substring de words desde i con longitud j
8:              agregar substring a substrings
9:          end for
10:     end for
                                          ▷ Intentar parsear cada substring, de mayor
```

a menor, hasta obtener un árbol sintáctico, y repetir el proceso en los lados de la oración que no pudimos parsear

```
    for all substring en substrings do
        tree ← intentar parsear substring con feature_parser
        if tree ≠ null then
            encontrar índices de inicio y fin de substring en sub_sentence
            if hay parte de sub_sentence antes de substring then
                left ← max_tree(parte izquierda, feature_parser)
                agregar left a result
            end if
            agregar (tree, índices) a result
            if hay parte de sub_sentence después de substring then
                right ← max_tree(parte derecha, feature_parser)
                ajustar índices de right y agregar a result
            end if
            return result
        end if
    end for
    return [ ]                          ▷ Lista vacía si no hay substring parseable
end function
```

3.6.4. Transferencia sintáctica sobre los árboles obtenidos

Para traducir los árboles extraídos por el algoritmo anterior, reutilizamos el sistema de transferencia sintáctica que desarrollamos para el corpus sintético, con ligeras modificaciones.

En primer lugar, esta versión tiene en cuenta la posibilidad de que al procesar una hoja del árbol sintáctico no se conozca ninguna traducción para la palabra en cuestión. En ese caso, se mantiene la forma en español en la hoja del árbol guaraní, y se le asignan rasgos que permitan continuar con el proceso de unificación de forma satisfactoria. Particularmente, se asignan valores de nasalidad en función de los sonidos nasales presentes en la palabra en español, y se niegan los verbos según la lógica del español ("no" + verbo) en lugar del guaraní. Por otro lado, donde no se conocen rasgos se omiten los mismos o se explicitan todos los casos posibles, de forma que la palabra unifique de la forma más amplia posible.

En segundo lugar, tuvimos el cuidado de llevar registro de los índices de los segmentos traducidos, en caso de que un árbol fallara el proceso de transferencia.

Por último, cambió el input del sistema, puesto que estamos proveyendo una nueva gramática adaptada a nuestro nuevo propósito, y por ende también modificamos ligeramente las reglas de transferencia, creando un Ancora-rules.json. Las nuevas reglas de transferencia consideran las nuevas producciones que agregamos a partir del símbolo inicial anteriormente, para dar mayor flexibilidad a la gramática a la hora de parsear segmentos de oración.

El resultado de este proceso es entonces pares español-guaraní de segmentos de oración y sus respectivos índices en las oraciones originales, donde la tra-

ducción guaraní usa términos en español donde no conoce un término guaraní equivalente, pero mantiene conceptos fundamentales de la gramática guaraní, de alguna manera construyendo una versión artificial del jopará típico de los corpus bilingües con los que trabajamos.

3.6.5. Incrustación de las traducciones en las oraciones originales

Por último, desarrollamos un script que incrustara las traducciones en las oraciones originales, reemplazando los segmentos de oración correspondientes. Este hace primero un preprocesado ligero de algunas (no todas) de las marcas presentes en la traducción al guaraní.

Luego, hace uso de los índices que registramos correspondientes a cada traducción, para sustituir segmentos de las oraciones originales por sus traducciones al guaraní. Esto nuevamente resulta en una intercalación del español original con nuestras traducciones, que ya de por sí representaban un *code-switching* entre ambos idiomas. El resultado son oraciones similares a las que se encuentra en el Jojajovai, donde partes de las oraciones se mantienen en el idioma original, e incluso las partes que se traducen al guaraní hacen uso de términos en español.

Finalmente, hacemos un posprocesado de los pares de oraciones. En este se procesan las marcas restantes del guaraní, fundamentalmente uniendo prefijos y sufijos con sus términos correspondientes; luego, tanto para la oración original como para su traducción, se revierten cambios a la capitalización, puntuación y contracciones realizados durante el primer preprocesado del texto extraído de Ancora. Así se reconstituyen los textos originales, y se cuenta con una traducción al guaraní para cada uno.

3.6.6. Ejemplos

Presentamos algunos ejemplos de oraciones tomadas de Ancora y su traducción al guaraní según nuestro proceso en la tabla 3.12.

3.7. Evaluación

Con el fin de estimar de forma primaria la calidad y características de los recursos generados, tomamos dos medidas. Por un lado, extrajimos ciertas estadísticas de los corpus generados, para obtener una visión a alto nivel de ciertos rasgos de los conjuntos de datos y cómo se comparan entre sí. Por el otro, solicitamos a un experto en la lengua guaraní la evaluación de una muestra de pares de oraciones español-guaraní, tomada de nuestro corpus sintético.

3.7.1. Estadísticas

Presentamos en la tabla 3.13 algunos datos de interés sobre el corpus sintético y aquel basado en Ancora.

Español	**Guaraní**
La reestructuración de los otros bancos checos se está acompañando por la reducción del personal.	Reestructuración de los otros apykakuéra checos se oĩ acompañando ñemomichĩ rupi personalgui.
Por parte de los militares, los negociadores de Binimarama pondrán sobre la mesa el tema de los rehenes y cuándo tendrá lugar su liberación.	Por oho militaresgua, negociadores de Binimarama omoĩt sobre la mesa ñe'ẽmbyrã rehenesgua ha cuándo oguerekot mamo iñemosãso.
Por ello, González pidió una reforma "urgente" del sistema de contratación de extranjeros que permita un sistema "ágil y rápido" para insertar a los inmigrantes en el mercado laboral.	Por ello, González ojerurekuri una ñembopyahu "pojava" sistemagui de contratación de pytagua que permita un sistema "rari ha kyre'ỹ " omombyta insertar inmigrantespe ñemuha laboralpe.

Tabla 3.12: Pares español-guaraní basados en Ancora

Métrica	**Corpus Sintético**	**Corpus Ancora**
Número total de pares de oraciones	277.842	14.120
Número total de tokens en español	1.215.305	392.921
Número total de tokens en guaraní	999.398	334.976
Promedio de tokens por oración en español	4,37	27,83
Promedio de tokens por oración en guaraní	3,60	23,72
Correlación en la longitud de pares de oraciones	0,9129	0,9868
Tamaño del vocabulario en español	32.758	34.701
Tamaño del vocabulario en guaraní	41.202	44.869

Tabla 3.13: Estadísticas para los corpus bilingües sintético y basado en Ancora

Es interesante notar la diferencia en el promedio de tokens por oración entre ambos corpus; dado que el corpus sintético priorizaba oraciones cortas y más propias quizás de la oralidad, su promedio de 4,4 tokens en español y 3,6 tokens en guaraní resulta minúsculo comparado a los 27,8 y 23,7 del corpus de Ancora, el cual consistía principalmente de texto periodístico.

Por otro lado, también vale resaltar que en ambos casos el promedio de tokens en guaraní resultó menor que aquel en español. Esto se debe a la ausencia

de algunos artículos y, sobre todo, al carácter aglutinante de la lengua guaraní, donde fenómenos como la negación y las aposiciones no resultan en tokens extra, sino que se adhieren a la palabra que modifican. Estos mismos factores podrían explicar también el mayor tamaño de vocabulario encontrado en el lado guaraní para ambos corpus; las variantes que se generan al aglutinar negaciones o posposiciones derivan en una mayor diversidad en el vocabulario al tokenizar separando por espacios, respecto a la que encontramos para el español.

En el anexo B.3 se incluyen algunas métricas de eficiencia del proceso, como el tiempo de ejecución y el porcentaje promedio de oraciones autogeneradas que supera el filtro de la gramática de rasgos.

3.7.2. Evaluación por hablante de la lengua

Para determinar la calidad de las traducciones de nuestro sistema, previo al entrenamiento de un modelo neuronal basado en los corpus paralelos construidos, seleccionamos un pequeño conjunto de pares de oraciones de nuestro corpus sintético y se lo presentamos a un experto en la lengua guaraní para que evaluara las mismas.

En la tabla 3.14 se observan las oraciones evaluadas. Las mismas fueron seleccionadas de tal forma que combinaran distintos largos y diferentes estructuras, buscando la mayor representatividad posible.

Español	**Guaraní**
Ellos no jurarán aquel mundo a aquellos hermanos.	Ha'ekuéra noñe'ẽme'ẽit amo yvy umi joyke'ykuérape.
Nuestra tutela parará.	Ñande poguy omombytat.
Nuestros fines no importunaron estos preceptos.	Ñande pahakuéra nomoangekóirikuri ko'ã tembiapoukapykuéra.
Él no trabajaba.	Ha'e nomba'apoimi.
Esa serie no valió.	Ku tysýi ndovaleikuri.
Yo no quebraré vuestro razonamiento.	Che ndajokait pene ñe'ẽmongeta.
Vos no sacas esos fusiles.	Nde napenohẽi umi mbokakuéra.
Nuestro desgarro fijará vuestra figura a aquella formación.	Ñane mondoro omombytat pene ta'anga aipo tesape'ape.

Tabla 3.14: Pares evaluados de oraciones en español y guaraní

La evaluación que hace el experto acerca de las traducciones es positiva; las oraciones en guaraní resultan mayormente correctas en cuanto a estructura y sintaxis, con pequeños fallos particularmente en algunas terminaciones, como *omombytat* en lugar de *ombotýta*.

Por otro lado, reconoce que se debe tener cuidado con el aspecto semántico. En algunos casos, había diferencias entre el significado de algunas palabras

en español y nuestra traducción al guaraní: por ejemplo, al traducir "fines" como *pahakuéra*, toma una semántica de "término", en lugar de ser sinónimo de "propósito" (*rembipotakuéra*). Esto es esperable, puesto que nuestro sistema de traducción por reglas carece de una componente semántica que permita diferenciar esa clase de sutilezas en el significado.

El experto concluye que, a pesar de los detalles mencionados, en general "las oraciones están generadas coherentemente tanto en guaraní como en español, y parece que el generador está haciendo su trabajo en estructura sintáctica y uso de palabras".

Capítulo 4

Experimentos de traducción neuronal

En esta sección describimos los experimentos donde utilizamos métodos de traducción automática neuronal con el objetivo de evaluar la viabilidad de nuestra técnica de aumentado de datos mediante texto sintético, tanto para la tarea de traducción de guaraní a español como español a guaraní. Para ello, entrenamos modelos de arquitectura seq2seq y transformer tanto utilizando sus hiperparámetros por defecto en la biblioteca MarianNMT como ajustando sus hiperparámetros para poder entrenar también modelos de mejor desempeño. Luego, evaluamos a cada modelo en escenarios con y sin preentrenamiento.

En cuanto a las métricas utilizadas, si bien evaluamos todas las pruebas utilizando medidas de chrF y BLEU extraídas cada 10 *epochs*, consideramos la métrica chrF como la más relevante dadas las características morfológicamente ricas del guaraní (además de las ventajas sobre BLEU que mencionamos en Sección 2.6.4).

Dividimos los experimentos a realizar en cinco etapas, cuatro de validación y una de evaluación. En todas las etapas utilizamos como conjunto de datos de entrenamiento al conjunto de entrenamiento de Jojajovai, y extrajimos métricas de su conjunto de desarrollo en las pruebas de validación, y de su conjunto de test para la etapa de evaluación.

Para la primera etapa, en vías de obtener una línea base, realizamos una prueba inicial sobre el conjunto de validación utilizando los modelos seq2seq y transformer con sus hiperparámetros por defecto. Luego, realizamos para la segunda un pequeño conjunto de pruebas preliminares para examinar el efecto de distintos hiperparámetros sobre los modelos, y para la tercera un ajuste aleatorio de hiperparámetros con el fin de obtener modelos con mejor desempeño.

Posteriormente, en la cuarta etapa de validación preentrenamos los modelos obtenidos en etapas previas utilizando por separado el conjunto de datos generado por la gramática (al cual llamaremos "Gramática"), el conjunto de Ancora, la Biblia, y la concatenación de estos últimos tres (al cual llamaremos "Todos").

En ella, ajustamos el número de *epochs* de preentrenamiento utilizados por cada modelo. Finalmente, en la etapa de evaluación realizamos pruebas con los mejores modelos de las etapas anteriores con los mismos datos de la fase anterior, aunque ahora extrayendo métricas del conjunto de evaluación. Pueden observarse estadísticas de estos conjuntos de datos en la tabla 4.1.

Conjunto de datos	**Pares de oraciones**	**Tokens en español**	**Tokens en guaraní**
Jojajovai - Train	20.213	459.629	309.920
Jojajovai - Dev	5.315	108.604	73.414
Jojajovai - Test	5.335	109.806	73.111
Gramática	277.842	1.215.305	999.398
Ancora	14.120	392.921	334.976
Biblia	21.979	497.815	372.166
Todos (sintético + Ancora + Biblia)	312.690	2.098.350	1.699.886

Tabla 4.1: Tamaño de cada conjunto de datos utilizados.

La división lógica de experimentos es fundamental en el contexto en que realizamos el proyecto, ya que generamos los corpus de preentrenamiento mientras que en simultáneo implementamos un sistema de traducción y extracción de métricas. Esto nos permitió generar mejoras a ambos sistemas de forma iterativa y poder, una vez disponibles los conjuntos de datos, realizar pruebas de preentrenamiento.

Como expandiremos en posteriores secciones, durante el proyecto nos fue de utilidad apodar a cada prueba de validación con nombres de niveles: *nivel 1*, *nivel 2*, *nivel 3* y *nivel 4*; donde en cada nivel se buscará mejorar los resultados del nivel anterior, y solo los mejores modelos del nivel 4 pasarán a la etapa de evaluación.

Puede hallarse una descripción formal de los experimentos en A.1.

4.1. Diseño de los experimentos y construcción de los modelos

En esta sección detallamos las distintas etapas de validación que seguimos para obtener modelos con hiperparámetros ajustados y por defecto.

4.1.1. Nivel 1: Prueba con modelos por defecto

En esta etapa buscamos, con el fin de obtener una línea base, probar modelos de distintas arquitecturas de MarianNMT utilizando sus hiperparámetros por defecto. Realizamos estas pruebas en ambas direcciones (español-guaraní y guaraní-español), y utilizamos las arquitecturas de transformer y seq2seq, sumando un total de 4 experimentos. El uso de estas arquitecturas se debe a que,

aunque MarianNMT provee la posibilidad de utilizar una mayor variedad de arquitecturas, algunas permiten ajustar pocos hiperparámetros, y otras son casos particulares de otras arquitecturas.

Para obtener el vocabulario requerido por MarianNMT, en este primer experimento lo generamos utilizando un tokenizador a nivel de palabra de Spacy entrenado sobre el corpus de `es_core_news_md` y aplicándolo en el corpus de entrenamiento de Jojajovai. Aunque el tokenizador es entrenado en español y utilizado en ambas lenguas, lo utilizamos para obtener una primera aproximación con un método menos simple que tokenizar por espacios, y aprovechando similitudes entre idiomas como signos de puntuación[1] y que el corpus en guaraní contiene fragmentos en español. Además, activamos el hiperparámetro max-length-crop descrito en la tabla 4.2 ya que observamos las oraciones de entrada de Jojajovai eran realmente largas, y ejecutamos los modelos por un máximo de 200 *epochs* dado que los modelos por defecto son pequeños y no suelen converger en más de 100.

Los resultados pueden observarse en la tabla 4.3, y muestran un BLEU y chrF realmente bajos en comparación a los obtenidos con el modelo por defecto utilizando la herramienta OpenNMT (Chiruzzo, Góngora, et al., 2022), que son tomadas sobre el conjunto de test y no el de desarrollo de Jojajovai, por lo que no son directamente comparables aunque son una primera aproximación ya que ambas son medidas de desempeño tomadas sobre datos no vistos con anterioridad. Además, aunque se observa un BLEU muy lejano, la distancia relativa entre las medidas de chrF es considerablemente distinta. Esto generó inicialmente una incógnita ya que en la medida de BLEU se observan resultados realmente menores a los obtenidos en el estudio mencionado, que obtuvo 16,10 BLEU en la dirección español-guaraní y 19,06 BLEU en la dirección opuesta, mientras que el nuestro solo 5,51 y 5,58 para los modelos seq2seq.

Otra observación de los resultados es que para ambas direcciones de traducción, la arquitectura seq2seq parece ser superior a la arquitectura transformer, al menos con hiperparámetros por defecto.

Pueden observarse gráficas de los resultados en el anexo A.3.

[1] Los tokenizadores ya implementados que proveen la mayoría de librerías usualmente fueron creados para el idioma inglés, que carece de símbolos como "¿" o "¡", y al ser aplicados sobre el español suelen unir de forma indeseada símbolos y palabras en un mismo token.

$P_{Vocabulario}$		
Parámetro	**Valor**	**Descripción**
max-length-crop	true	Este hiperparámetro se asegura de que, al tener una frase del corpus más tokens que el número máximo, ella se trunque en vez de descartarse. Lo consideramos fundamental ya que observamos que las secuencias del corpus de entrenamiento de Jojajovai tiene secuencias con múltiples oraciones juntas.
dim-vocabs$_{source}$	2^{14}	La dimensión de los vocabularios se refiere al tamaño del vocabulario máximo que genera Sentencepiece. Tomamos un número cercano a la mitad de la cantidad máxima permitida en nuestro corpus, la cual es aproximadamente 25000. Asumimos este como el valor por defecto ya que MarianNMT requiere de uno para funcionar con Sentencepiece.
dim-vocabs$_{target}$	2^{14}	Utilizamos el mismo tamaño en el origen que en el destino para simplificar la elección de hiperparámetros.

Tabla 4.2: Conjunto de hiperparámetros y valores de MarianNMT relacionados con el vocabulario de los lenguajes

Origen	**Destino**	**Arquitectura**	***Epoch***	**BLEU**	**chrF**
es	gn	seq2seq	80	5,51	25.7
es	gn	transformer	10	3,01	18,7
gn	es	seq2seq	120	5,83	30,31
gn	es	transformer	60	5,58	26,24

Tabla 4.3: Resultados de pruebas de nivel 1.

4.1.2. Nivel 2: Ajuste preliminar de hiperparámetros

Esta etapa sirve como fase previa a una búsqueda de hiperparámetros exhaustiva que realizamos en el próximo nivel. Aquí experimentamos con distintos hiperparámetros de Marian con el fin de observar la sensibilidad del modelo por defecto a cambios en hiperparámetros específicos. La búsqueda tiene el objetivo de poder interpretar cómo se relaciona el cambio de un hiperparámetro con el desempeño del modelo dejando otros constantes, por lo que utilizamos optimización coordenada a coordenada para observar la correlación entre cada hiperparámetro y las métricas de evaluación.

En esta etapa solo experimentamos con el modelo de mejor rendimiento en la fase previa (que fue el modelo seq2seq) y en la dirección guaraní-español, para que la salida de la validación en español permita la detección de errores en

caso haberlos y reducir el número de experimentos. Está claro que como mencionamos en la sección 2.2.4, los mejores hiperparámetros para la arquitectura seq2seq difícilmente coincidan con los mejores para la arquitectura transformer, o con los mejores para el modelo en la dirección español-guaraní, sin embargo, consideramos que no valía la pena realizar cuatro veces más experimentos solo para ajustar un poco más el espacio de búsqueda del próximo nivel. El número total de experimentos debido a esto y una descripción de los hiperparámetros ajustados puede observarse en la tabla 4.4.

Parámetro	**Valores**	**Descripción**	n
-	default	Los resultados serán los mismos para los experimentos con parámetros por defecto de este nivel.	1
(enc-depth, dec-depth)	[**(1, 1)**, (4, 4), (6, 6), (8, 8)]	Se refieren estos hiperparámetros a la profundidad del codificador y decodificador de la arquitectura, ya sea apilando redes seq2seq (Sutskever et al., 2014) o aumentando la cantidad de bloques N del transformer (Vaswani et al., 2017). Este hiperparámetro es fundamental para que el modelo no desajuste o sobreajuste, ya que afecta directamente la cantidad de parámetros del modelo utilizado.	3
enc-cell	["tanh", "lstm", **"gru"**]	Vimos que las redes seq2seq se conforman por un codificador y decodificador, ambos implementados con redes recurrentes clásicas (tanh), LSTM y GRU. La celda del codificador tanto como la del decodificador es fundamental para evitar el problema de *exploding* o *vanishing gradients*, lo cual es más probable que ocurra en modelos recurrentes.	2
dim-vocabs	[2^{13}, $\mathbf{2^{14}}$, 2^{15}]	La dimensión del vocabulario de Sentencepiece define qué tan grande es el vocabulario interpretable por el modelo. Un valor alto significa que habrá pocos tokens que el modelo no reconozca, y uno bajo que el modelo reconocerá menos tokens que sean poco frecuentes en el corpus de entrenamiento.	2
max-length	[**50**, 100, 200, 250]	Este parámetro ajusta el tamaño máximo de tokens permitido en una secuencia. En caso de superarse la secuencia es descartad a no ser que se active el hiperparámetro max-length-crop.	3
Total de experimentos			11

Tabla 4.4: Hiperparámetros ajustados en las pruebas de nivel 2 con parámetros por defecto resaltados, y número de pruebas n.

En cuanto a los hiperparámetros que dejamos fijos, utilizamos nuevamente los hiperparámetros en la tabla 4.2, y dado que quisimos realizar la cantidad ahora mayor de experimentos con mejor eficiencia computacional, también utilizamos los hiperparámetros en la tabla 4.5 aún sin layer-normalization ni maxi-batch. Además, de este experimento en adelante, el vocabulario utilizado es creado utilizando la tokenización de unigrama (que como vimos es un método de subpalabra proveído por la librería Sentencepiece), y el número de *epochs* máximo es 1000.

$P_{eficiencia}$		
Parámetro	**Valor**	**Descripción**
early-stopping	7	Pacienia del método de *early stopping* introducido anteriormente.
layer-normalization	true	Técnica de *layer normalization* utilizada estabilizar y acelerar el entrenamiento.
devices	"0"	ID's de GPUs utilizadas por experimento. "0" es una, "0 1" son dos.
fp16	Si	Se almacenan números de punto flotante con solo 16 bits. Esto hace el entrenamiento menos estable aunque más veloz y con poca pérdida de desempeño (Micikevicius et al., 2018).
workspace	6500	Cantidad de MB asignados al programa al iniciarse. Utilizamos el máximo que el software nos permitió en nuestro entorno ya que es lo que recomiendan los desarrolladores de MarianNMT junto con mini-batch-fit.
mini-batch-fit	Si	Ajusta el tamaño del mini-batch al tamaño disponible en memoria.
maxi-batch	1000	Precarga 1000 mini-batches.
quiet-translation	Si	No escribir *logs* en el buffer *stdout*.
overwrite	Si	Sobreescribir archivos de ejecuciones anteriores.

Tabla 4.5: Conjunto de hiperparámetros y valores de MarianNMT relacionados con la eficiencia computacional.

De los resultados mostrados en la tabla 4.6, descatamos que modificar algunos hiperparámetros mejoró sustancialmente al modelo, aunque en otros casos tuvo un efecto leve. Los hiperparámetros max-length, enc-depth y dim-vocabs mostraron la mayor correlación con el desempeño del modelo en ambas BLEU y chrF. Por otro lado, enc-cell no tanto.

En particular, el tamaño máximo de tokens en una secuencia (max-length) logró casi duplicar el chrF del modelo por defecto. Esto parece indicar que el modelo es capaz de adquirir más información aún teniendo las frases una cantidad considerable de tokens (4 veces más que la cantidad por defecto).

Por otro lado, la profundidad del codificador y decodificador mostró tam-

Parámetro	**Valor**	***Epoch***	**BLEU**	**chrF**
-	default	250	3,71	19,86
max-length	100	320	6,09	24,16
max-length	200	330	15,78	**37.28**
max-length	250	630	10,25	30
(enc-depth, dec-depth)	(4, 4)	270	5,45	23,04
(enc-depth, dec-depth)	(6. 6)	160	7,89	**26.14**
(enc-depth, dec-depth)	(8, 8)	170	7,24	25,93
enc-cell	lstm	290	3.7	**20.64**
enc-cell	tanh	200	2,97	18,19
dim-vocabs	25000	290	3,66	20,31
dim-vocabs	8192	660	7,57	**25.71**

Tabla 4.6: Resultados de pruebas de nivel 2 según hiperparámetros de MarianNMT realizados en la dirección guaraní-español y utilizando la arquitectura seq2seq.

bién un aumento importante en el desempeño, maximizando las medidas en la profundidad (enc-depth y dec-depth) 6, posiblemente indicando que la profundidad por defecto (1) no era suficiente para aprender suficiente del corpus de entrenamiento.

Luego, el tamaño del vocabulario de tokens mostró nuevamente un aumento importante al reducirlo. Un vocabulario de alrededor de 8000 tokens indica que el modelo descarta a los tokens menos importantes, reduciendo el número de parámetros a aprender y el posible impacto de ruido en el conjunto de entrenamiento.

Por último, dado que la celda de GRU ya soluciona el problema de *vanishing gradients* al igual que la LSTM, GRU es el valor por defecto en MarianNMT y no consideramos significativa la diferencia en desempeño entre LSTM y GRU, dejaremos el valor por defecto (GRU) de aquí a próximas pruebas.

Pueden observarse gráficas de los resultados en el anexo A.4.

4.1.3. Nivel 3: Ajuste con búsqueda aleatoria

En esta etapa, dada la retroalimentación obtenida de la etapa anterior, realizamos una búsqueda de hiperparámetros óptimos en el conjunto de desarrollo de Jojajovai. Con el objetivo de recorrer el espacio de búsqueda de hiperparámetros de forma completa, teniendo control sobre el número de experimentos, y realizando varias ejecuciones en simultáneo, utilizamos el método de búsqueda *random search*. Como usamos redes seq2seq e identificamos que el conjunto de datos contiene secuencias largas, también utilizamos hiperparámetros que reducen el problema de *vanishing* y *exploding gradients*. Configuramos también más hiperparámetros para evitar el sobreajuste, dada la cantidad de datos disponibles en una lengua de bajos recursos. Todos los hiperparámetros fijos que aquí utilizamos pueden observarse en las tablas 4.2, 4.5, 4.7 y 4.8.

$P_{sobreajuste}$		
Parámetro	**Valor**	**Descripción**
transformer-dropout	0,1	Técnica de dropout para capas de modelos transformer (no disponible para modelos seq2seq) con el mismo valor fijo de Vaswani et al. (2017).
dropout-rnn	0,2	Técnica de dropout entre capas de redes recurrentes (no disponible para transformers). Cada valor de dropout que fijamos está basado en el modelo de Sennrich et al. (2016a).
dropout-src	0,1	Técnica de dropout para capa de entrada (no disponible para transformers).
dropout-trg	0,1	Técnica de dropout para capa de salida (no disponible para transformers).
label-smoothing	0,1	Técnica de label smoothing. 0,1 es de los valores con mejores resultados en Micikevicius et al. (2018).
exponential-smoothing	true	Técnica de suavizado exponencial aplicada a parámetros de la red. Es uno de los métodos que provee MarianNMT para evitar el sobreajuste.
tied-embeddings-all	Si	Se comparten matrices de pesos entre capa de embeddings de entrada y capa de embeddings de salida de la red.

Tabla 4.7: Conjunto de parámetros y valores de MarianNMT relacionados con técnicas para evitar el sobreajuste.

En cuanto al total de experimentos, realizamos una cantidad de iteraciones de *random search* de $n = 20$ para cada para cada arquitectura y dirección de traducción, conformando un total de pruebas mostrado a continuación:

$$2 * 2 * 20 = 80$$

pruebas. Nuevamente esto es un balance entre buscar cubrir el espacio de búsqueda de forma extensiva y no realizar demasiados experimentos.

A continuación describimos las distribuciones de probabilidad utilizadas en la búsqueda aleatoria, también detalladas en profundidad en A.2. Para la tasa de aprendizaje nos basamos en la distribución propuesta por (Bergstra y Bengio, 2012): una distribución que da probabilidad exponencialmente mayor a valores más bajos que a altos. Para el largo máximo utilizamos una distribución normal centrada en el valor de mayor desempeño en la etapa de nivel 2, truncada a 3 desviaciones estándar (donde se encuentra el 99.7 % del volumen de la distribución), con el fin de evitar el $overflow$, similar a la profundidad del modelo. Para la dimensión del vocabulario utilizamos una distribución categórica centradas

$P_{gradientes}$		
Parámetro	**Valor**	**Descripción**
clip-norm	true	Se refiere a la técnica de *gradient clipping*. En MarianNMT este parámetro está activado por defecto.
enc-cell	GRU	Vimos que el problema de *vanishing* y *exploding gradients* ocurría principalmente en RNNs clásicas. Tanto GRU como LSTM. En particular, GRU es el parámetro por defecto en MarianNMT. Este hiperparámetro está solo disponible para redes seq2seq.
dec-cell	GRU	Misma descripción que para las celdas del codificador, pero para el decodificador. Este hiperparámetro está solo disponible para redes seq2seq.
skip	true	Técnica de *skip connections*. Este hiperparámetro es únicamente modificable para redes seq2seq.

Tabla 4.8: Conjunto de parámetros y valores de MarianNMT relacionados con técnicas para evitar el problema de *vanishing* o *exploding gradients*.

en el mejor valor de la fase 2. Esto se justifica por razones de implementación, ya que para cada tamaño de vocabulario debería crearse un vocabulario nuevo.

Adicionalmente, dado que extrajimos los resultados en los que nos basamos solo de redes seq2seq, suavizamos las distribuciones del modelo transformer utilizando distribuciones de mayor desviación, y utilizamos la distribución uniforme para la profundidad, ya que no consideramos a la retroalimentación de ese hiperparámetro directamente transferible. Sin embargo, debimos además reducir su profundidad y *learning rate* por inconvenientes que comentamos a continuación, y eso causó que descartáramos algunos resultados y realizaramos experimentos hasta obtener el número deseado de pruebas satisfactorias.

Origen	**Destino**	**Arquitectura**	***Epoch***	**Semilla**	**learn-rate**	**max-length**	**depth**	**dim-vocabs**	**BLEU**	**chrF**
gn	es	seq2seq	240	1234	0,0016	187	6	6000	25,37	47,32
		transformer	920	60438	$3{,}95e-05$	182	3	2000	15,88	39,72
es	gn	seq2seq	190	38006	0,00013	153	3	6000	23,32	46,2
		transformer	800	30883	$5{,}77e-05$	198	2	2000	15,73	40,35

Tabla 4.9: Resultados en conjunto de desarrollo de Jojajovai de mejores experimentos de nivel 3 para cada dirección de traducción y arquitectura.

Como puede observarse en la tabla 4.9 y la figura 4.1, los mejores resultados se dieron para arquitecturas del tipo seq2seq para ambas direcciones. Esto podría indicar que dicha arquitectura se comporta mejor para la tarea de traducción para los datos del corpus de Jojajovai. Sin embargo, hay que resaltar que vimos la necesidad de modificar los modelos con la arquitectura transformer para utilizar una menor tasa de aprendizaje α. Esto se debe a problemas que tuvimos con el software MarianNMT, que durante la validación se detenía en etapas tempranas del entrenamiento por errores inesperados. De hecho, modificamos el espacio de

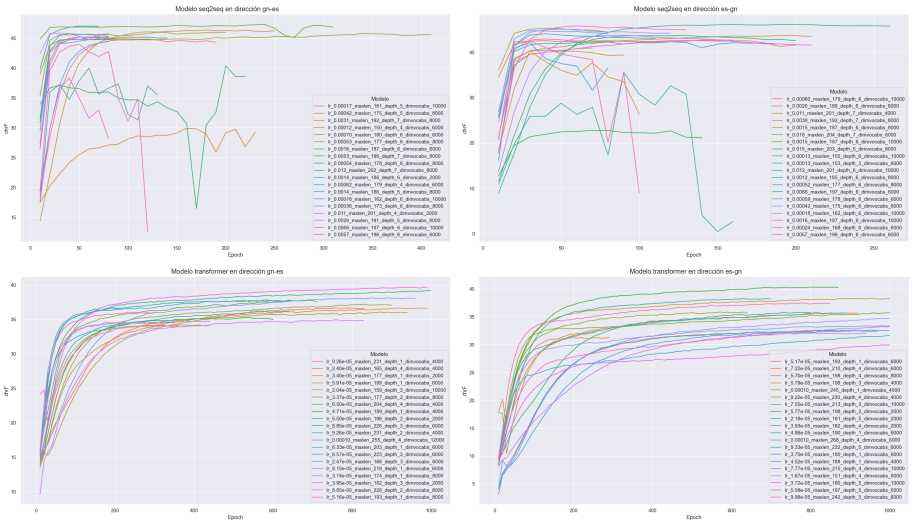

Figura 4.1: Avance de desempeño por *epoch* de modelos en pruebas de nivel 3.

búsqueda de modo que se mantuviera por debajo de la tasa de aprendizaje por defecto ya que hasta en valores cercanos al valor por defecto se daban muchas veces errores para ambas arquitecturas (principalmente para transformers), y observamos también que modelos de alta profundidad tendían a no concretar su entrenamiento sin errores. Esto puede observarse al ver la diferencia de suavidad entre las gráficas de arquitecturas seq2seq y transformers (figura 4.1). Intuimos que este hecho puede explicarse por fallas del software de MarianNMT o por falta de memoria en la plataforma utilizada. Por otro lado, debido a que el principal causante es la tasa de aprendizaje, podría también deberse al efecto comentado por Goodfellow et al. (2016), donde se comenta con ayuda de la gráfica 4.2 la existencia de un umbral a partir del cual el valor α hace al error del modelo crecer al infinito.

Asimismo, algo importante a resaltar es el hecho de que para el caso del modelo seq2seq en la dirección guaraní-español los mejores modelos tuvieron profundidad 6, superando a modelos más pequeños. Sin embargo, en la dirección español-guaraní aparece un modelo de profundidad 3 (evento con un 2 % de probabilidad de ocurrir en cada iteración de la búsqueda aleatoria) que supera al resto. Esto podría explicarse por dos factores, uno es que el modelo obtenga buenos resultados a causa de su profundidad, y otro es que los obtenga a pesar de ella. Una hipótesis que proponemos es que exista un óptimo local que se da cuando la profundidad es 6, y otro cuando es 3, y que en cada caso el modelo aproveche las ventajas de cada profundidad, como almacenar más información en el modelo más profundo o comprimir más datos en pocos parámetros. Por otro lado, podría también deberse a que el resto de hiperparámetros simplemente

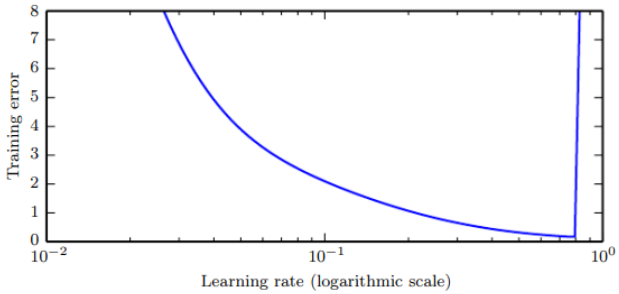

Figura 4.2: Error en conjunto de entrenamiento en función a tasa de aprendizaje (Goodfellow et al., 2016).

sea tan bueno en el modelo de profundidad 3 que logre superar al resto aún con baja profundidad.

Se destaca además que las mejores dimensiones para el vocabulario variaron en ambas arquitecturas, obteniendo mejores resultados con vocabularios de tamaño 6000 para modelos seq2seq y 2000 para modelos transformer. Es importante resaltar que un vocabulario menor implica que la tokenización de unigrama solo mantuvo a conjuntos frecuentes de tokens, posiblemente dejando afuera tokens ruidosos y favoreciendo hallar subpalabras frecuentes y con información generalizable como morfemas, que pueden encontrarse en palabras no tan comunes que ahora serán descartadas.

Los resultados completos del nivel 3 pueden entonctrarse en el anexo A.5.

4.1.4. Nivel 4: Selección de *epochs* de preentrenamiento

Hasta este nivel en ningún momento utilizamos un conjunto de preentrenamiento. En esta etapa preentrenamos, para cada arquitectura y dirección, los mejores modelos de la etapa anterior al igual que modelos por defecto (para tener resultados con modelos de características variadas) utilizando en distintas pruebas los conjuntos de datos de "Gramática", Ancora, la Biblia, y "Todos", y finalmente realizando un *fine-tuning* sobre el corpus de Jojajovai. El único antecedente del que tenemos conocimiento que ha realizado un experimento similar proviene de Chiruzzo, McGill, et al. (2022), donde preentrenaron modelos con un gran corpus paralelo de datos sintéticos para la tarea de traducción de español hablado a lengua de señas española.

Dado que no sabemos de antemano el número de *epochs* de preentrenamiento que obtendrá un mejor resultado, realizamos una búsqueda en grilla sobre la cantidad de *epochs* y los distintos corpus de preentrenamiento. En este caso el número de *epochs* de preentrenamiento se encontrará dentro del conjunto $\{0, 1, 4, 16, 64\}$, el cual definimos mediante pruebas preliminares preentrenando sobre el conjunto "Gramática" donde observamos que obtuvimos mejores resultados preentrenando para valores bajos de $e_{pretrain}$ (ver tabla 4.10). Se hace

énfasis en que cuando el número de *epochs* es 0, los resultados corresponden a los del modelo sin preentrenar (datos ya obtenidos en la fase anterior) y son iguales sin importar el corpus.

			gn-es		**es-gn**	
Ajuste	**Arquitectura**	$e_{pretrain}$	**BLEU**	**chrF**	**BLEU**	**chrF**
Default	seq2seq	0	4,97	22,77	2,93	18,49
		1	**12,61**	**32,77**	10,27	29,45
		4	10,94	30,26	**11,23**	**31,62**
		16	9,89	28,16	9,25	28,08
		64	7,25	24,66	4,94	20,79
Ajustado	seq2seq	0	25,37	47,32	**23,35**	**46,20**
		1	**25,66**	**47,38**	22,03	43,64
		4	24,81	46,63	23,21	45,79
		16	21,70	43,16	23,34	45,64
		64	20,00	41,39	23,00	44,98
Default	transformer	0	**7,53**	25,40	5,64	22,22
		1	7,40	**25,95**	**5,71**	**22,40**
		4	7,27	24,91	5,22	21,43
		16	6,13	22,59	4,66	19,64
		64	4,59	20,18	3,44	17,72
Ajustado	transformer	0	**15,88**	**39,72**	**15,85**	**40,35**
		1	12,40	35,54	10,57	34,01
		4	13,19	36,90	11,25	34,99
		16	14,33	38,47	13,95	38,41
		64	14,40	38,43	14,52	39,02

Tabla 4.10: Resultados en el conjunto de desarrollo de Jojajovai de pruebas de nivel 4 en el corpus "Gramática" para distintas *epochs* de preentrenamiento $e_{pretrain}$ para cada método de ajuste, arquitectura y dirección de traducción.

Una decisión importante que tomamos con respecto a los experimentos donde preentrenamos modelos es que decidimos crear el vocabulario requerido por MarianNMT a partir del conjunto de entrenamiento de Jojajovai. Decidimos hacer esto ya que consideramos que el efecto de crear el vocabulario a partir de conjuntos de datos sintéticos podría ser nocivo para el desempeño del modelo en conjuntos distintos, debido a que los métodos de tokenización dentro del estado del arte crean su vocabulario en base a frecuencias de tokens, que podrían no corresponderse en nuestros datos sintéticos con las del corpus de Jojajovai.

En este experimento, los hiperparámetros utilizados coinciden con los mismos de la etapa anterior, que pueden observarse en las tablas 4.2, 4.5, 4.7 y 4.8. Adicionalmente, dado que hay 2 formas de ajuste distintas (default o con parámetros ajustados), 2 arquitecturas, 2 direcciones y 4 corpus de preentrenamiento (de las cuales los resultados de cantidad de *epochs* de preentrenamiento $e_{pretrain} = 0$ dan los mismos resultados para todas las pruebas de misma ar-

quitectura y dirección), este experimento comprende una cantidad de pruebas de:

$$2 * 2 * 2 * (4 * 4 + 1) = 136$$

			gn-es		**es-gn**	
Ajuste	**Arquitectura**	$e_{pretrain}$	**BLEU**	**chrF**	**BLEU**	**chrF**
Default	seq2seq	0	4,97	22,77	2,93	18,49
		1	4,02	21,00	3,01	18,17
		4	4,05	21,15	3,19	18,53
		16	6,45	23,78	10,39	29,35
		64	**15,78**	**36,70**	**14,56**	**35,01**
Ajustado	seq2seq	0	25,37	47,32	23,35	46,20
		1	23,90	45,92	21,28	43,83
		4	24,86	47,03	21,85	44,52
		16	25,19	47,78	23,11	46,44
		64	**25,70**	**48,14**	**24,46**	**47,77**
Default	transformer	0	7,53	25,40	5,64	22,22
		1	7,77	26,30	5,50	21,77
		4	**8,15**	27,14	**5,68**	22,37
		16	7,99	26,66	5,64	**22,42**
		64	8,14	**27,37**	5,29	21,71
Ajustado	transformer	0	15,88	39,72	15,85	40,35
		1	11,91	35,39	10,50	34,34
		4	11,87	35,38	10,70	34,88
		16	12,73	36,58	13,08	37,69
		64	**16,05**	**40,88**	**16,71**	**41,69**

Tabla 4.11: Resultados en el conjunto de desarrollo de Jojajovai de pruebas de nivel 4 en el corpus de Ancora para distintas *epochs* de preentrenamiento $e_{pretrain}$ para cada método de ajuste, arquitectura y dirección de traducción.

Las tablas 4.10, 4.11, 4.12 y 4.13 muestran los resultados de estos experimentos variando los distintos corpus de preentrenamiento: "Gramática", Ancora, la Biblia, y "Todos". Además, la tabla 4.14 muestra los mejores cinco resultados de cada dirección sobre todos los conjuntos de preentrenamiento. Como puede observarse, al igual que vimos inicialmente que para el corpus "Gramática" el desempeño mejoraba preentrenando por pocas *epochs*, esta tendencia se mantuvo para el corpus de la Biblia, y también para la concatenación de todos los corpus. Sin embargo, para el corpus de Ancora todos los mejores resultados medidos en chrF en la dirección guaraní-español se dieron para $e_{pretrain} = 64$ (máximo valor probado), y tres de cada cuatro en la dirección contraria. Es importante notar que el corpus de Ancora es más pequeño que el resto, por lo que es posible que usar el número de *epochs* en lugar de número de otra medida independiente al tamaño del corpus como actualizaciones de parámetros en este caso no haya

sido favorable para su interpretación.

			gn-es		es-gn	
Ajuste	**Arquitectura**	$e_{pretrain}$	**BLEU**	**chrF**	**BLEU**	**chrF**
Default	seq2seq	0	4,97	22,77	2,93	18,49
		1	**5,24**	**22,80**	2,98	18,44
		4	3,83	20,86	3,00	**18,95**
		16	4,09	21,38	**3,10**	18,77
		64	3,71	20,09	2,87	17,46
Ajustado	seq2seq	0	25,37	47,32	**23,35**	**46,20**
		1	24,02	45,17	21,29	42,75
		4	24,92	46,38	21,59	43,28
		16	**25,45**	**47,82**	22,70	44,96
		64	24,19	46,25	23,06	45,56
Default	transformer	0	7,53	25,40	**5,64**	**22,22**
		1	7,39	**25,58**	5,16	20,96
		4	**7,58**	25,39	5,15	20,87
		16	6,31	23,29	4,22	18,71
		64	6,03	22,19	3,92	18,38
Ajustado	transformer	0	**15,88**	**39,72**	**15,85**	**40,35**
		1	12,56	35,61	10,80	34,55
		4	12,53	36,05	10,61	34,30
		16	12,89	36,37	11,43	35,32
		64	13,57	36,99	11,88	35,73

Tabla 4.12: Resultados en el conjunto de desarrollo de Jojajovai de pruebas de nivel 4 en el corpus de la Biblia para distintas *epochs* de preentrenamiento $e_{pretrain}$ para cada método de ajuste, arquitectura y dirección de traducción.

			gn-es		es-gn	
Ajuste	**Arquitectura**	$e_{pretrain}$	**BLEU**	**chrF**	**BLEU**	**chrF**
seq2seq	Default	0	4,97	22,77	2,93	18,49
		1	9,83	28,42	11,04	30,38
		4	12,73	32,48	12,53	33,88
		16	**12,96**	**33,15**	**13,55**	**34,78**
		64	9,46	27,05	11,01	30,24
seq2seq	Ajustado	0	25,37	47,32	23,35	46,20
		1	25,44	47,14	22,20	43,93
		4	**25,95**	**48,33**	23,29	46,09
		16	25,12	47,28	24,95	48,16
		64	23,6	44,94	**25,84**	**49,07**
transformer	Default	0	7,53	25,40	5,64	22,22
		1	8,00	26,96	5,78	22,44
		4	**8,65**	**27,66**	**6,49**	**23,75**
		16	7,52	25,31	4,97	21,00
		64	5,97	22,91	3,93	18,61
transformer	Ajustado	0	15,88	39,72	15,85	40,35
		1	12,57	36,10	10,53	34,32
		4	13,54	37,27	11,77	35,61
		16	15,34	39,80	15,53	40,14
		64	**18,75**	**43,49**	**19,06**	**43,80**

Tabla 4.13: Resultados en el conjunto de desarrollo de Jojajovai de pruebas de nivel 4 en el corpus "Todos" para distintas *epochs* de preentrenamiento $e_{pretrain}$ para cada método de ajuste, arquitectura y dirección de traducción.

Origen	**Destino**	**Ajustado**	**Modelo**	**Corpus**	***Epochs***	**BLEU**	**chrF**
gn	es	Si	seq2seq	Todos	4	25,95	48,33
		Si	seq2seq	Ancora	64	25,7	48,14
		Si	seq2seq	Biblia	16	25,45	47,82
		Si	seq2seq	Ancora	16	25,19	47,78
		Si	seq2seq	Gramatica	1	25,66	47,38
es	gn	Si	seq2seq	Todos	64	25,84	49,07
		Si	seq2seq	Todos	16	24,95	48,16
		Si	seq2seq	Ancora	64	24,46	47,77
		Si	seq2seq	Ancora	16	23,11	46,44
		Si	seq2seq	-	0	23,35	46,2

Tabla 4.14: Mejores 5 resultados de pruebas de nivel 4 realizadas en el conjunto de desarrollo de Jojajovai para cada dirección de traducción.

Analizando los resultados, según la métrica chrF, 25 de los 32 conjuntos diferenciados de hiperparámetros, corpus y direcciones de traducción obtuvieron

mejores resultados al ser preentrenados. De ellos, el corpus que obtuvo mejores resultados en ambas direcciones fue la concatenación de todos, con 48,33 de chrF en el par guaraní-español y 49,07 en español-guaraní. Este resultado es sumamente relevante ya que parece indicar que todos los conjuntos de preentrenamiento son (ya sea en menor o mayor medida) útiles, y que más aún, su unión también logra mejores resultados. Aún así, resaltamos que estos resultados son solo preliminares ya que son realizados en el conjunto de validación.

También enfatizamos que el corpus de Ancora fue, sin contar al corpus "Todos", el que obtuvo el máximo desempeño medido en chrF posible, obteniendo 48,14 en la dirección guaraní-español y 47,77 en español-guaraní. Hallamos una posible explicación a estos resultados, la cual es que el buen desempeño preentrenando en el corpus de Ancora es a causa de la naturaleza de gran parte del corpus Jojajovai, que contiene frases periodísticas donde en muchos casos las traducciones eran realizadas de español a guaraní, donde frecuentemente se optaba por mantener palabras en español, lo cual coincide con el procedimiento que realizamos de traducción con la gramática.

En cualquier caso, parece relevante como trabajo a futuro probar el efecto de distintos tamaños de corpus de preentrenamiento al igual que en cuanto a número de *epochs* , donde también recomendamos utilizar como medida actualizaciones de parámetros en lugar de número de *epochs* para que el tamaño de los corpus vuelva los resultados comparables.

4.2. Evaluación de los modelos construidos

En esta etapa evaluamos en el conjunto de test de Jojajovai los modelos con las *epochs* de preentrenamiento que en el nivel 4 obtuvieron un mejor desempeño para cada epoch de preentrenamiento según la métrica chrF, además de los mismos modelos sin preentrenar. Notar que esto implica que los hiperparámetros utilizados en ambos casos son los mismos, con la diferencia de que las métricas serán ahora extraídas del conjunto de evaluación de Jojajovai. Buscaremos además realizar un análisis de los resultados tanto a nivel del corpus de test de Jojajovai como de los subconjuntos individuales que lo conforman.

Además de los resultados de nuestros modelos, incluimos métricas provenientes del traductor de Google, y de modelos entrenados en el mismo artículo que presenta al corpus de Jojajovai (Chiruzzo, Góngora, et al., 2022), ambos resultados tomados sobre el conjunto de test de Jojajovai. En el caso del traductor de Google, obtuvimos las medidas accediendo al traductor mediante la API de Google Cloud Platform (GCP) el día 15 de octubre de 2023. Para el modelo del artículo, tomamos métricas extraídas y reportadas en él, las cuales fueron calculadas a partir de dos modelos basados en arquitecturas LSTM con mecanismos de atención, que fueron entrenados con la librería OpenNMT. Ambos modelos de OpenNMT fueron entrenados con sus hiperparámetros por defecto, aunque lo fue entrenado sobre el conjunto de entrenamiento de Jojajovai y el otro sobre la concatenación de ese conjunto y el corpus de La Biblia. De aquí en adelante llamamos a estos últimos modelos como `jojajovai_base` y `jojajovai_biblia`

respectivamente.

Dado que ahora solo utilizamos una *epoch* de preentrenamiento por modelo, el número de evaluaciones a realizar será, sin contar el experimento de Google ni las métricas del artículo de Jojajovai, el calculado a continuación:

$$2 * 2 * 2 * (4 + 1) = 40$$

.

4.2.1. Resultados sobre el corpus de evaluación completo

Modelo	**Corpus**	**Ajustado**	**BLEU**	**chrF**
seq2seq	-	No	2,81	18,64
		Si	24,40	47,96
	Gramática	No	11,83	33,05
		Si	23,48	46,71
	Biblia	No	3,42	19,39
		Si	24,12	47,40
	Ancora	No	14,69	35,46
		Si	26,17	49,65
	Todos	No	14,44	35,78
		Si	**26,64**	**50,34**
transformer	-	No	4,59	20,44
		Si	15,96	40,65
	Gramática	No	4,90	20,54
		Si	15,00	40,07
	Biblia	No	4,68	20,89
		Si	12,08	36,83
	Ancora	No	5,84	23,15
		Si	17,70	43,28
	Todos	No	6,10	23,62
		Si	19,81	45,26
Google	-	-	19,31	48,92
jojajovai_base	-	No	16,10	29,41
jojajovai_bible	-	No	20,77	35,28

Tabla 4.15: Resultados de pruebas sobre el conjunto de test de Jojajovai ordenadas por chrF en la dirección español-guaraní para mejores modelos del nivel 4 variando en cuanto a *epochs* de preentrenamiento y los mismos modelos sin preentrenar. Adicionalmente se incluyen los resultados de la traducción de Google y del artículo de Jojajovai (Chiruzzo, Góngora, et al., 2022).

Lo primero que se resalta de la fase de evaluación es que los resultados en las tablas 4.15 y 4.16 se mantienen en condiciones prácticamente idénticas a los vistos en la fase de validación en las tablas 4.10, 4.11, 4.12 y 4.13; lo cual parece indicar un correcto diseño e implementación de la metodología. En cuanto a

Modelo	Corpus	Ajustado	BLEU	chrF
seq2seq	-	No	3,73	20,83
		Si	25,53	47,24
	Gramática	No	11,15	30,57
		Si	25,63	47,44
	Biblia	No	11,33	31,40
		Si	25,97	47,87
	Ancora	No	15,25	37,11
		Si	26,83	49,11
	Todos	No	12,85	33,04
		Si	26,20	49,09
transformer	-	No	6,62	24,01
		Si	4,08	18,50
	Gramática	No	7,38	25,78
		Si	14,78	39,36
	Biblia	No	5,46	21,30
		Si	14,11	37,96
	Ancora	No	7,73	26,75
		Si	17,24	42,44
	Todos	No	7,80	26,59
		Si	20,37	45,38
Google	-	-	**26,96**	**50,95**
jojajovai_base	-	No	19,06	31,84
jojajovai_bible	-	No	19,98	33,31

Tabla 4.16: Resultados de pruebas sobre el conjunto de test de Jojajovai ordenadas por chrF en la dirección guaraní-español para mejores modelos del nivel 4 variando en cuanto a *epochs* de preentrenamiento y los mismos modelos sin preentrenar. Adicionalmente se incluyen los resultados de la traducción de Google y del artículo de Jojajovai.

orden de resultados, en ambos casos los modelos preentrenados por la concatenación de todos los corpus y con el de Ancora logran un mejor desempeño (ver figura 4.3). Más aún, los valores de los resultados son casi idénticos a los de validación, y hasta en la dirección español-guaraní se supera la marca de 50 chrF que no fue alcanzada en etapas el conjunto de desarrollo de Jojajovai. Por esto consideramos que la metodología de validación que utilizamos fue satisfactoria en cuanto a los objetivos de no sobreajustar y de obtener modelos de alto desempeño.

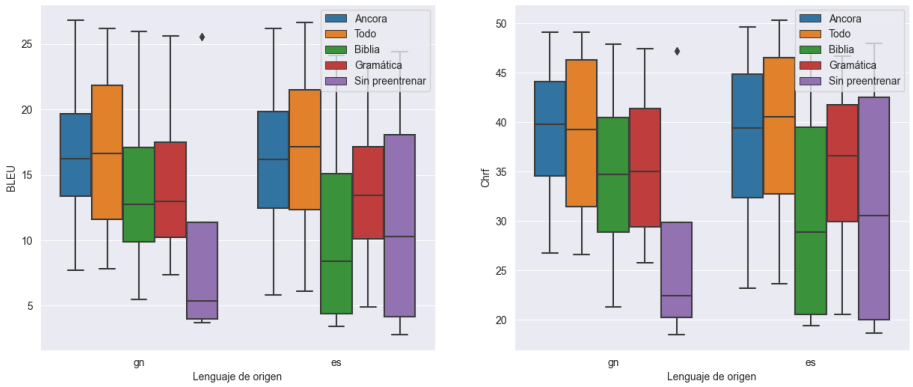

Figura 4.3: Desempeño en la epoch de mayor chrF de todos los experimentos del conjunto de test de Jojajovai distinguidos por el corpus utilizado para preentrenar.

El segundo hecho relevante es que, como puede observarse en la tabla 4.17, se logra en la dirección español-guaraní superar la traducción de Google por 1,42 de chrF con el mejor modelo en evaluación (que fue también el mejor en validación). Además, en la dirección guaraní-español obtuvimos resultados menores aunque cercanos en cuanto a chrF, y aún más con la medida de BLEU, donde hay una diferencia absoluta de tan solo 0,13 para nuestro mejor modelo. Consideramos esto realmente positivo ya que creemos que en un proyecto de mayor alcance donde pudiéramos usar técnicas más avanzadas (como ensamblajes de modelos o intentar mejorar el desempeño de transformers) los resultados alcanzados serían considerablemente mejores a los logrados en este proyecto para ambas direcciones. Esto da a entender que aún sin la tecnología de punta con la que trabaja una empresa de tan alto renombre es posible lograr resultados competitivos en cuanto al estado del arte en traducción de lenguajes de bajos recursos como el guaraní. Se resalta además, que dado que Google ha trabajado durante años con el lenguaje español, es razonable que el desempeño de sus modelos generando frases en ese idioma sea superior a generarlas en la lengua opuesta, debido a que la cantidad de datos y la calidad del modelo de lenguaje en español que utilizan debe ser muy superior a la nuestra.

Adicionalmente, observando las medidas de BLEU en la dirección español-guaraní, el modelo de Google ni siquiera ingresa a los mejores 5 modelos utilizados (ver tabla 4.15). Esto es sorprendente ya que es la primera vez que se observa en uno de nuestros experimentos un modelo con tan bajo BLEU y tan alto chrF. Además, observamos que los modelos del artículo de Jojajovai no alcanzan a ninguno de nuestros modelos con hiperparámetros ajustados, y hasta pierden frente a algunos que entrenamos en MarianNMT tampoco lo fueron aunque preentrenamos con el corpus de Ancora o de “Todos”.

Subconjunto	Origen	Destino	Modelo	Ajustado	Corpus	BLEU	chrF
abc	es	gn	s2s	Si	Todos	31,45	**58,76**
			Google	-	-	23,56	56,61
			jojajovai_base	No	-	18,24	37,44
			jojajovai_bible	No	-	24,48	46,14
	gn	es	s2s	Si	Ancora	30,83	56,31
			Google	-	-	30,81	**56,73**
			jojajovai_base	No	-	20,84	40,25
			jojajovai_bible	No	-	22,14	42,03
anlp	es	gn	s2s	Si	Todos	3,01	24,58
			Google	-	-	6,01	**37,05**
			jojajovai_base	No	-	0,75	14,10
			jojajovai_bible	No	-	1,76	18,67
	gn	es	s2s	Si	Ancora	4,04	21,17
			Google	-	-	19,8	**42,04**
			jojajovai_base	No	-	1,55	14,77
			jojajovai_bible	No	-	2,52	17,19
blogs	es	gn	s2s	Si	Todos	16,1	32,3
			Google	-	-	16,27	**39,38**
			jojajovai_base	No	-	7,73	21,35
			jojajovai_bible	No	-	11,26	25,45
	gn	es	s2s	Si	Ancora	18,13	33,37
			Google	-	-	24,45	**45,25**
			jojajovai_base	No	-	11,89	24,71
			jojajovai_bible	No	-	12,50	25,40
hackathon	es	gn	s2s	Si	Todos	5,47	34,69
			Google	-	-	5,75	**41,71**
			jojajovai_base	No	-	3,09	20,02
			jojajovai_bible	No	-	3,06	23,39
	gn	es	s2s	Si	Ancora	10,86	30,34
			Google	-	-	18,44	**46,32**
			jojajovai_base	No	-	6,45	19,35
			jojajovai_bible	No	-	6,48	23,58
libro_gn	es	gn	s2s	Si	Todos	7,72	**30,16**
			Google	-	-	8,3	28,82
			jojajovai_base	No	-	3,44	16,98
			jojajovai_bible	No	-	7,46	19,15
	gn	es	s2s	Si	Ancora	10,5	30,36
			Google	-	-	11,29	**31,88**
			jojajovai_base	No	-	5,40	17,15
			jojajovai_bible	No	-	7,80	19,08
libro_td	es	gn	s2s	Si	Todos	10,49	**39,38**
			Google	-	-	3,09	28,15
			jojajovai_base	No	-	5,15	24,10
			jojajovai_bible	No	-	3,38	28,25
	gn	es	s2s	Si	Ancora	18,41	**37,69**
			Google	-	-	9,02	29,62
			jojajovai_base	No	-	10,25	24,02
			jojajovai_bible	No	-	8,56	26,45
seminario	es	gn	s2s	Si	Todos	7,78	28,88
			Google	-	-	9	**35,94**
			jojajovai_base	No	-	3,02	19,83
			jojajovai_bible	No	-	5,15	22,32
	gn	es	s2s	Si	Ancora	10,1	30,64
			Google	-	-	13,16	**36,73**
			jojajovai_base	No	-	6,80	23,05
			jojajovai_bible	No	-	6,37	23,15
spl	es	gn	s2s	Si	Todos	29,58	**48,5**
			Google	-	-	16,74	39,66
			jojajovai_base	No	-	20,73	37,49
			jojajovai_bible	No	-	23,51	39,63
	gn	es	s2s	Si	Ancora	31,21	**48,58**
			Google	-	-	23,58	44,49
			jojajovai_base	No	-	25,93	41,68
			jojajovai_bible	No	-	25,83	41,24

Tabla 4.17: Resultados de modelos de mayor desempeño sobre el conjunto de test de Jojajovai según cada uno de sus subconjuntos, donde se incluyen modelos del artículo de Jojajovai y del traductor de Google sobre el mismo conjunto.

4.2.2. Resultados sobre el corpus de evaluación separado por subconjuntos

En la tabla 4.17 pueden observarse los resultados de nuestros modelos de mejor desempeño para cada subconjunto de Jojajovai y dirección de traducción, al igual que los resultados del traductor de Google y los modelos del artículo de Jojajovai sobre el conjunto de test de Jojajovai. De esta tabla se observa, evaluando cada subconjunto individualmente, que nuestros modelos superan en la dirección español-guaraní en 4 de los 8 corpus en cuanto a chrF: abc, `libro_gn` y `libro_td`. La diferencia tan alta a nivel global se da porque el corpus abc compone aproximadamente la mitad del corpus global. Por otro lado, en la dirección guaraní-español solo se logra en 2 de los 8: `libro_td` y en spl, todos por modelos seq2seq con hiperparámetros ajustados. Puede observarse en la figura 4.4 los resultados de evaluar en la dirección español-guaraní, donde nuestro modelo superó de forma global a Google, al desempeño del traductor de Google y nuestro mejor modelo sobre cada subconjunto del corpus de test de Jojajovai.

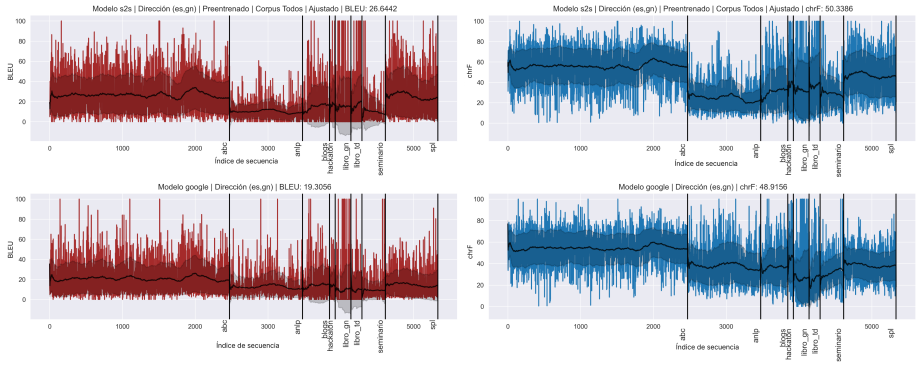

Figura 4.4: Comparación de mejor modelo del conjunto de test de Jojajovai en la dirección español-guaraní preentrenado en el conjunto "Todos" y comparado con el resultado obtenido por el traductor de google.

Se observa que los conjuntos del artículo de Jojajovai no logran los mejores resultados en ningún subconjunto, ni siquiera preentrenando con el conjunto de La Biblia. Esto tiene sentido ya que no tienen sus hiperparámetros ajustados.

Por otro lado, puede contemplarse en la tabla que mientras que muchas veces Google obtenía un desempeño similar a nuestro mejor modelo, existían casos donde la diferencia de chrF era mucho mayor, particularmente favoreciendo al modelo de Google, como en el caso de los subconjuntos "anlp", "blogs", "hackathon", o el de "seminario". Una teoría que consideramos es que es posible que el modelo de Google haya sido entrenado en los subconjuntos donde obtuvo resultados considerablemente más altos a los nuestros. En cualquier caso, consideramos el hecho de superar a Google en cualquier subconjunto como un

logro relevante, y una latente oportunidad para lograr mejoras en el área de traducción entre español y guaraní.

4.3. Detalles de implementación y ejecución

En esta sección describimos detalles relacionados con la implementación y ejecución de experimentos realizados durante el proyecto. En primer lugar se presenta el entorno y sistema implementado para

4.3.1. Entorno de ejecución

Inicialmente utilizamos un cuaderno de jupyter para probar el correcto funcionamiento de MarianNMT. Para ello utilizamos un cuaderno creado con las instalaciones necesarias para ser luego utilizado[2] el cual hallamos en un foro[3].

Para el entorno de ejecución utilizamos la plataforma de cómputo más potente del Uruguay, ClusterUY (Nesmachnow y Iturriaga, 2019), la cual concede a alumnos de la facultad de ingeniería la posibilidad de utilizar nodos junto con múltiples procesadores y GPUs mediante el protocolo SSH, aunque con un límite de ejecuciones de trabajos en simultáneo por usuario. Además, dado que un usuario estudiante no tiene permisos de administrador, utilizamos un contenedor de docker creado por un colaborador del proyecto de MarianNMT para llevar a cabo los experimentos[4].

4.3.2. Implementación

Para la realización de experimentos fue implementado un sistema capaz de automatizar la ejecución de MarianNMT en la plataforma ClusterUY. El sistema sigue una arquitectura de capas de scripts y programas capaces de funcionar independientemente de la capa superior, dado que cada capa fue creada para automatizar una tarea sobre la anterior. Definimos a continuación cada componente del sistema.

Software de traducción

Un componente fundamental del sistema implementado para realizar los experimentos es el software MarianNMT, el cual es ejecutado por consola pasando hiperparámetros y direcciones de archivos mediante banderas. Este requiere una estructura determinada de archivos, como un archivo para cada idioma de vocabulario tokenizado, frases alineadas para el entrenamiento y opcionalmente para validación, o simplemente frases para inferencia y un modelo.

[2] https://colab.research.google.com/drive/1n1_SsXUoktJllZPLpgKW4TA1zyq8waUM?usp=sharing

[3] https://groups.google.com/g/marian-nmt/c/uQC6q1PIf3k

[4] https://hub.docker.com/r/lefterav/marian-nmt

Pipeline de entrenamiento

Dado que MarianNMT requiere de los archivos antes mencionados, implementamos un software con arquitectura de pipeline con pasos desactivables que tiene soporte a multiprocesos. Este provee las siguientes funcionalidades:

1. Convertir el conjunto de datos en los artefactos esperados por MarianNMT (conjunto de entrenamiento, validación, vocabularios, y/o test y modelos) dado el conjunto de datos de Jojajovai.

2. Recibir un conjunto de configuraciones en forma de grilla (para realizar *grid search*), espacio de búsqueda aleatorio con distribuciones de probabilidad (para *random search*), o configuraciones individuales, y transformarlas en un conjunto ordenado de configuraciones.

3. En caso de querer preentrenar un modelo, se creará un vocabulario de tokens de Sentencepiece a partir del conjunto de entrenamiento. Luego, se preentrenará el modelo guardándolo en caché en caso de requerirse para futuros preentrenamientos.

4. Entrenar al modelo eventualmente preentrenado extrayendo opcionalmente métricas cada determinado número de *epochs*.

Ejecuciones reproducibles

Para que las ejecuciones sean reproducibles, fueron guardadas en scripts de bash que son capaces de ejecutar el pipeline con los mismos parámetros que fueron ejecutados para los experimentos.

Contenedor

Dado que, como ya fue mencionado, la plataforma del cluster no permite la instalación de software externo, es necesario utilizar un contenedor para la ejecución de MarianNMT, por ejemplo mediante el script mencionado en la capa anterior.

Ejecución en Cluster

Luego, para realizar tareas en la infraestructura de ClusterUY es necesario utilizar una serie de comandos con un formato específico, para lo cual es posible crear scripts de slurm con el fin de automatizar la especificación de configuración de hardware del nodo a utilizar, como cantidad de memoria y tiempo de solicitud. Esto permitirá además ejecutar tareas en múltiples nodos en simultáneo.

Asignación automática de trabajos

La capa anterior permite realizar tareas individualmente, aunque tiene el problema de que para cada tarea a realizar, debe crearse un nuevo script de slurm para el nodo y bash para las ejecuciones reproducibles, y luego ejecutar la tarea

de cada nodo uno por uno manualmente. A causa de esto fue creado un script el cual dado un número de nodos a utilizar y un conjunto de configuraciones a ejecutar, las reparte entre estos nodos. Otra funcionalidad relevante es que, en caso de una falla en la plataforma, el script es capaz de detectar trabajos cancelados, y volverlo a ejecutar con la misma configuración.

Análisis de Resultados

Además de la implementación en software, realizamos el análisis de resultados mediante notebooks de jupyter que, dada una planilla de resultados (obtenida del pipeline mencionado en la sección anterior), generaba un análisis gráfico de las distintas secciones de experimentos. Dada la similitud entre experimentos, los métodos de graficado eran compartidos entre cuadernos.

Pruebas automáticas

Dado que fueron identificadas áreas críticas en el sistema, y lo difícil que es identificar y solucionar un fallo en el entorno del cluster (además del tiempo que conlleva un error en este entorno) fueron implementadas pruebas unitarias a modo de reducir fallas leves y mitigar las críticas. Las funcionalidades evaluadas de forma automática son:

- Creación de conjuntos de entrenamiento, validación y evaluación, además de vocabularios.
- Entrenamiento del modelo en un contexto de preentrenamiento y ajuste de hiperparámetros.
- Correcto funcionamiento de extracción de métricas comparando con datos de prueba y métricas oficiales[5].

4.3.3. Ejecución

A continuación presentamos algunos detalles acerca de la ejecución de los distintos experimentos.

Los experimentos de traducción fueron realizados con un número de 4 tareas de 9 CPUs cada una, y 60GB de memoria RAM, con una duración máxima de 5 días.

Dado que la versión de MarianNMT del contenedor no era compatible con la versión de las GPU A100 de ClusterUY, se agregó la restricción de solo utilizar GPUs P100 para los experimentos. Además, por restricciones de disponibilidad, todos los experimentos fueron realizados con solo una GPU.

Por último, dado que el almacenamiento del clúster es limitado, los modelos creados eran eliminados automáticamente luego del cálculo de todas las métricas.

[5] https://huggingface.co/spaces/evaluate-metric/sacrebleu

Capítulo 5

Conclusiones

5.1. Conclusiones del proyecto

En este estudio buscamos evaluar la viabilidad de técnicas de aumentado de datos, mediante gramáticas formales y transferencia sintáctica, para preentrenar modelos neuronales de traducción automática, en el marco de una lengua de escasos recursos. A su vez, buscamos generar recursos de PLN para el idioma guaraní que puedan principalmente ser usados para tareas de traducción automática de guaraní-español, y evaluar su efectividad utilizando modelos de aprendizaje automático.

Como resultado de este proyecto generamos una gramática de rasgos en español y reglas de transferencia sintáctica de español a guaraní. A través de ellas produjimos además dos corpus paralelos, uno creado a raíz de generar frases sintéticas en español a partir de nuestra gramática y traducirlas utilizando técnicas de transferencia sintáctica, y otra traduciendo el corpus de Ancora en español utilizando la gramática y las reglas de transferencia que creamos.

Por otro lado, entrenamos modelos de aprendizaje automático en el corpus paralelo de Jojajovai y preentrenando sobre distintas combinaciones de corpus paralelos disponibles (los que construimos, y uno basado en la traducción al guaraní de la Biblia), donde obtuvimos los mejores resultados utilizando modelos seq2seq preentrenados sobre la concatenación de todos los conjuntos. Aún más, estos resultados fueron superiores a los extraídos de la traducción de Google tanto en la métrica BLEU como chrF en la dirección español-guaraní, además de ser competitivos en la dirección contraria. Consideramos que esto indica un gran potencial para aplicar técnicas alternativas de aumentado de datos en el preentrenamiento de modelos de traducción automática, en un contexto de escasez de recursos.

5.1.1. Trabajo futuro

Consideramos que los resultados que obtuvimos tienen lugar a la mejora. En el ámbito de las gramáticas, agregar una componente semántica, por mí-

nima que sea, podría ayudar a desambiguar la traducción mediante reglas; sin embargo, la carencia de recursos para el guaraní puede dificultar enormemente esta tarea. Por otro lado, se pueden construir gramáticas paralelas en español y guaraní mucho más ricas, particularmente si se cuenta con mayor conocimiento de la lengua y su sintaxis; distintos tiempos y modos no fueron considerados por nuestra gramática, así como tampoco lo fueron los adverbios, distintos tipos de pronombres, o los verbos compuestos. Una gramática más rica podría permitir traducir una mayor variedad de textos y proveer durante la fase de preentrenamiento una mayor exposición a distintas estructuras sintácticas y morfológicas.

En cuanto al modelo neuronal obtenido, es posible que se pueda mejorar el desempeño de nuestro mejor modelo sin mucho esfuerzo. En primer lugar, no utilizamos técnicas de limpieza y normalización del conjunto de datos, los cuales parecen ser fundamentales según hallazgos en experimentos previos. Por otro lado, la cantidad de parámetros ajustados (contando el número de *epochs* de preentrenamiento) de cada modelo fue bastante reducida debido al alcance del proyecto, y hasta hallamos que aunque los modelos con profundidad 6 daban buenos resultados en comparación a modelos de profundidad 4 y 5, utilizar profundidad 3 podría otorgar mejores resultados. Además, tuvimos que reducir el espacio de búsqueda para la arquitectura de transformers dado a problemas con el software MarianNMT, lo cual probablemente haya causado que nos perdamos de modelos que ofrezcan mayores resultados. Tampoco probamos variaciones del tamaño del conjunto de preentrenamiento, por ejemplo experimentar si el corpus generado por la gramática daría mejores resultados con la mitad o el doble de frases. Por último, creemos que hacer uso de técnicas más costosas como ensamblajes de modelos podría favorecer casi inmediatamente la solución que planteamos.

En cuanto a la metodología seguida, se podría considerar en un futuro hacer uso de más semillas para la generación de pesos iniciales con el fin de generar métricas más precisas. Por otro lado, utilizar actualizaciones en lugar de *epochs* como medida de frecuencia de extracción de métricas podría ser mejor metodológicamente para casos donde se quieren comparar distintos corpus de preentrenamiento como en nuestra etapa de validación. Por último, hacer uso de métricas de evaluación basadas en embeddings como BLEURT o Meteor podría mostrar resultados interesantes dado que ninguna de las métricas utilizadas utiliza de forma directa información semántica.

Referencias

Abu-Mostafa, Y. S., Magdon-Ismail, M., y Lin, H.-T. (2012). *Learning from data*. AMLBook.

Academia de la Lengua Guaraní. (2018, agosto). Gramática guaraní. Descargado de https://guaraniayvu.com/pdf/GramaticaGuarani.pdf

Araabi, A., y Monz, C. (2020, diciembre). Optimizing transformer for low-resource neural machine translation. En D. Scott, N. Bel, y C. Zong (Eds.), *Proceedings of the 28th international conference on computational linguistics* (pp. 3429–3435). Barcelona, Spain (Online): International Committee on Computational Linguistics. Descargado de https://aclanthology.org/2020.coling-main.304 doi: 10.18653/v1/2020.coling-main.304

Ba, J. L., Kiros, J. R., y Hinton, G. E. (2016). *Layer normalization.*

Bahdanau, D., Cho, K., y Bengio, Y. (2016). *Neural machine translation by jointly learning to align and translate.*

Bergstra, J., y Bengio, Y. (2012). Random search for hyper-parameter optimization [misc]. *Journal of Machine Learning Research*, *13*(Feb), 281-305. Descargado de http://www.jmlr.org/papers/v13/bergstra12a.html

Birch, A., Finch, A., Luong, M.-T., Neubig, G., y Oda, Y. (2018, julio). Findings of the second workshop on neural machine translation and generation. En A. Birch, A. Finch, T. Luong, G. Neubig, y Y. Oda (Eds.), *Proceedings of the 2nd workshop on neural machine translation and generation* (pp. 1–10). Melbourne, Australia: Association for Computational Linguistics. Descargado de https://aclanthology.org/W18-2701 doi: 10.18653/v1/W18-2701

Bojanowski, P., Grave, E., Joulin, A., y Mikolov, T. (2017). *Enriching word vectors with subword information.*

Borges, Y., Mercant, F., y Chiruzzo, L. (2021-03). *Using guarani verbal morphology on guarani-spanish machine translation experiments.*

Bostrom, K., y Durrett, G. (2020, noviembre). Byte pair encoding is suboptimal for language model pretraining. En T. Cohn, Y. He, y Y. Liu (Eds.), *Findings of the association for computational linguistics: Emnlp 2020* (pp. 4617–4624). Online: Association for Computational Linguistics. Descargado de https://aclanthology.org/2020.findings-emnlp.414 doi: 10.18653/v1/2020.findings-emnlp.414

Brown, P. F., Della Pietra, S. A., Della Pietra, V. J., y Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estima-

tion. *Computational Linguistics*, *19*(2), 263–311. Descargado de https://aclanthology.org/J93-2003

Chiruzzo, L., Agüero-Torales, M., Alvarez, A., y Rodríguez, Y. (2023). Initial experiments for building a guarani wordnet. En (p. 197-204).

Chiruzzo, L., Góngora, S., Alvarez, A., Giménez-Lugo, G., Agüero-Torales, M., y Rodríguez, Y. (2022, junio). Jojajovai: A parallel Guarani-Spanish corpus for MT benchmarking. En N. Calzolari et al. (Eds.), *Proceedings of the thirteenth language resources and evaluation conference* (pp. 2098–2107). Marseille, France: European Language Resources Association. Descargado de https://aclanthology.org/2022.lrec-1.226

Chiruzzo, L., McGill, E., Egea-Gómez, S., y Saggion, H. (2022, octubre). Translating Spanish into Spanish Sign Language: Combining rules and data-driven approaches. En A. K. Ojha et al. (Eds.), *Proceedings of the fifth workshop on technologies for machine translation of low-resource languages (loresmt 2022)* (pp. 75–83). Gyeongju, Republic of Korea: Association for Computational Linguistics. Descargado de https://aclanthology.org/2022.loresmt-1.10

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

De Gibert, O., Vázquez, R., Aulamo, M., Scherrer, Y., Virpioja, S., y Tiedemann, J. (2023, julio). Four approaches to low-resource multilingual NMT: The Helsinki submission to the AmericasNLP 2023 shared task. En M. Mager et al. (Eds.), *Proceedings of the workshop on natural language processing for indigenous languages of the americas (americasnlp)* (pp. 177–191). Toronto, Canada: Association for Computational Linguistics. Descargado de https://aclanthology.org/2023.americasnlp-1.20 doi: 10.18653/v1/2023.americasnlp-1.20

Ebrahimi, A., Mager, M., Rijhwani, S., Rice, E., Oncevay, A., Baltazar, C., ... Kann, K. (2023, julio). Findings of the AmericasNLP 2023 shared task on machine translation into indigenous languages. En M. Mager et al. (Eds.), *Proceedings of the workshop on natural language processing for indigenous languages of the americas (americasnlp)* (pp. 206–219). Toronto, Canada: Association for Computational Linguistics. Descargado de https://aclanthology.org/2023.americasnlp-1.23 doi: 10.18653/v1/2023.americasnlp-1.23

Estigarribia, B. (2015). Guaraní-spanish jopara mixing in a paraguayan novel: Does it reflect a third language, a language variety, or true codeswitching? *Journal of Language Contact*, *8*(2), 183–222.

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

Guzmán, F., Chen, P.-J., Ott, M., Pino, J., Lample, G., Koehn, P., ... Ranzato, M. (2019, noviembre). The FLORES evaluation datasets for low-resource machine translation: Nepali–English and Sinhala–English. En K. Inui, J. Jiang, V. Ng, y X. Wan (Eds.), *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (emnlp-ijcnlp)* (pp. 6098–

6111). Hong Kong, China: Association for Computational Linguistics. Descargado de https://aclanthology.org/D19-1632 doi: 10.18653/v1/D19-1632

Hochreiter, S., y Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Honnibal, M., y Montani, I. (2017). *spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing.* (To appear)

Huang, G., Liu, Z., van der Maaten, L., y Weinberger, K. Q. (2018). *Densely connected convolutional networks.*

Inan, H., Khosravi, K., y Socher, R. (2017). *Tying word vectors and word classifiers: A loss framework for language modeling.*

Joshi, P., Santy, S., Budhiraja, A., Bali, K., y Choudhury, M. (2020, julio). The state and fate of linguistic diversity and inclusion in the NLP world. En D. Jurafsky, J. Chai, N. Schluter, y J. Tetreault (Eds.), *Proceedings of the 58th annual meeting of the association for computational linguistics* (pp. 6282–6293). Online: Association for Computational Linguistics. Descargado de https://aclanthology.org/2020.acl-main.560 doi: 10.18653/v1/2020.acl-main.560

Jurafsky, D., y Martin, J. H. (2009). *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition.* Upper Saddle River, N.J.: Pearson Prentice Hall. Descargado de http://www.amazon.com/Speech-Language-Processing-2nd-Edition/dp/0131873210/ref=pd_bxgy_b_img_y

Kalchbrenner, N., y Blunsom, P. (2013). Recurrent continuous translation models. En *Conference on empirical methods in natural language processing.* Descargado de https://api.semanticscholar.org/CorpusID:12639289

Kudo, T. (2018). *Subword regularization: Improving neural network translation models with multiple subword candidates.*

Kudo, T., y Richardson, J. (2018, noviembre). SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. En E. Blanco y W. Lu (Eds.), *Proceedings of the 2018 conference on empirical methods in natural language processing: System demonstrations* (pp. 66–71). Brussels, Belgium: Association for Computational Linguistics. Descargado de https://aclanthology.org/D18-2012 doi: 10.18653/v1/D18-2012

Lecun, Y., Bottou, L., Bengio, Y., y Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324. doi: 10.1109/5.726791

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... Stoyanov, V. (2019). *Roberta: A robustly optimized bert pretraining approach.*

Luong, T., Sutskever, I., Le, Q., Vinyals, O., y Zaremba, W. (2015, julio). Addressing the rare word problem in neural machine translation. En C. Zong y M. Strube (Eds.), *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint confe-*

rence on natural language processing (volume 1: Long papers) (pp. 11–19). Beijing, China: Association for Computational Linguistics. Descargado de https://aclanthology.org/P15-1002 doi: 10.3115/v1/P15-1002

Mager, M., Bhatnagar, R., Neubig, G., Vu, N. T., y Kann, K. (2023). *Neural machine translation for the indigenous languages of the americas: An introduction.*

Mager, M., Oncevay, A., Ebrahimi, A., Ortega, J., Rios, A., Fan, A., ... Kann, K. (2021, junio). Findings of the AmericasNLP 2021 shared task on open machine translation for indigenous languages of the Americas. En M. Mager et al. (Eds.), *Proceedings of the first workshop on natural language processing for indigenous languages of the americas* (pp. 202–217). Online: Association for Computational Linguistics. Descargado de https://aclanthology.org/2021.americasnlp-1.23 doi: 10.18653/v1/2021.americasnlp-1.23

McCarthy, A. D., Wicks, R., Lewis, D., Mueller, A., Wu, W., Adams, O., ... Yarowsky, D. (2020, mayo). The Johns Hopkins University Bible corpus: 1600+ tongues for typological exploration. En N. Calzolari et al. (Eds.), *Proceedings of the twelfth language resources and evaluation conference* (pp. 2884–2892). Marseille, France: European Language Resources Association. Descargado de https://aclanthology.org/2020.lrec-1.352

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., ... Wu, H. (2018). *Mixed precision training.*

Mikolov, T., Chen, K., Corrado, G., y Dean, J. (2013). *Efficient estimation of word representations in vector space.*

Mitchell, T. M. (1997). *Machine learning* (Vol. 1) (n.º 9). McGraw-hill New York.

Nesmachnow, S., y Iturriaga, S. (2019). Cluster-uy: Collaborative scientific high performance computing in uruguay. En M. Torres y J. Klapp (Eds.), *Supercomputing* (pp. 188–202). Cham: Springer International Publishing.

Nirenburg, S., Somers, H. L., y Wilks, Y. A. (2003). *Readings in Machine Translation.* The MIT Press. Descargado de https://doi.org/10.7551/mitpress/5779.001.0001 doi: 10.7551/mitpress/5779.001.0001

Olah, C. (2015). *Understanding lstm networks.* https://colah.github.io/posts/2015-08-Understanding-LSTMs/. (Accessed: 2023-12-08)

Olivera, D. G. (2022). Verbos irregulares y aireales de la lengua guaraní. *Revista Multilingüe de Lengua, Sociedad y Educación*, *4*(1), 47–52.

Papineni, K., Roukos, S., Ward, T., y Zhu, W.-J. (2002, julio). Bleu: a method for automatic evaluation of machine translation. En P. Isabelle, E. Charniak, y D. Lin (Eds.), *Proceedings of the 40th annual meeting of the association for computational linguistics* (pp. 311–318). Philadelphia, Pennsylvania, USA: Association for Computational Linguistics. Descargado de https://aclanthology.org/P02-1040 doi: 10.3115/1073083.1073135

Pascanu, R., Mikolov, T., y Bengio, Y. (2013). *On the difficulty of training recurrent neural networks.*

Pennington, J., Socher, R., y Manning, C. (2014, octubre). GloVe: Global vectors for word representation. En A. Moschitti, B. Pang, y W. Dae-

lemans (Eds.), *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532–1543). Doha, Qatar: Association for Computational Linguistics. Descargado de https://aclanthology.org/D14-1162 doi: 10.3115/v1/D14-1162

Pollard, C., y Sag, I. A. (1994). *Head-Driven Phrase Structure Grammar*. Chicago: The University of Chicago Press.

Popović, M. (2015, septiembre). chrF: character n-gram F-score for automatic MT evaluation. En O. Bojar et al. (Eds.), *Proceedings of the tenth workshop on statistical machine translation* (pp. 392–395). Lisbon, Portugal: Association for Computational Linguistics. Descargado de https://aclanthology.org/W15-3049 doi: 10.18653/v1/W15-3049

Post, M. (2018, octubre). A call for clarity in reporting BLEU scores. En O. Bojar et al. (Eds.), *Proceedings of the third conference on machine translation: Research papers* (pp. 186–191). Brussels, Belgium: Association for Computational Linguistics. Descargado de https://aclanthology.org/W18-6319 doi: 10.18653/v1/W18-6319

Press, O., y Wolf, L. (2017). *Using the output embedding to improve language models.*

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., y Sutskever, I. (2019). Language models are unsupervised multitask learners.. Descargado de https://api.semanticscholar.org/CorpusID:160025533

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... Liu, P. J. (2023). *Exploring the limits of transfer learning with a unified text-to-text transformer.*

Rei, R., Stewart, C., Farinha, A. C., y Lavie, A. (2020). *Comet: A neural framework for mt evaluation.*

Sag, I. A., y Wasow, T. (Eds.). (1999). *Syntactic theory: A formal introduction.* Stanford, CA: CSLI Publications.

Schuster, M., y Paliwal, K. K. (1997, November). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, *45*, 2673–2681.

Sellam, T., Das, D., y Parikh, A. (2020, julio). BLEURT: Learning robust metrics for text generation. En D. Jurafsky, J. Chai, N. Schluter, y J. Tetreault (Eds.), *Proceedings of the 58th annual meeting of the association for computational linguistics* (pp. 7881–7892). Online: Association for Computational Linguistics. Descargado de https://aclanthology.org/2020.acl-main.704 doi: 10.18653/v1/2020.acl-main.704

Sennrich, R., Haddow, B., y Birch, A. (2016a, agosto). Edinburgh neural machine translation systems for WMT 16. En O. Bojar et al. (Eds.), *Proceedings of the first conference on machine translation: Volume 2, shared task papers* (pp. 371–376). Berlin, Germany: Association for Computational Linguistics. Descargado de https://aclanthology.org/W16-2323 doi: 10.18653/v1/W16-2323

Sennrich, R., Haddow, B., y Birch, A. (2016b). *Neural machine translation of rare words with subword units.*

Sennrich, R., y Zhang, B. (2019, julio). Revisiting low-resource neural machine translation: A case study. En A. Korhonen, D. Traum, y L. Màr-

quez (Eds.), *Proceedings of the 57th annual meeting of the association for computational linguistics* (pp. 211–221). Florence, Italy: Association for Computational Linguistics. Descargado de https://aclanthology.org/P19-1021 doi: 10.18653/v1/P19-1021

Snover, M., Dorr, B., Schwartz, R., Micciulla, L., y Makhoul, J. (2006, agosto 8-12). A study of translation edit rate with targeted human annotation. En *Proceedings of the 7th conference of the association for machine translation in the americas: Technical papers* (pp. 223–231). Cambridge, Massachusetts, USA: Association for Machine Translation in the Americas. Descargado de https://aclanthology.org/2006.amta-papers.25

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., y Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(56), 1929–1958. Descargado de http://jmlr.org/papers/v15/srivastava14a.html

Sutskever, I., Vinyals, O., y Le, Q. V. (2014). *Sequence to sequence learning with neural networks.*

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., y Wojna, Z. (2015). *Rethinking the inception architecture for computer vision.*

Tiedemann, J. (2020, noviembre). The tatoeba translation challenge – realistic data sets for low resource and multilingual MT. En L. Barrault et al. (Eds.), *Proceedings of the fifth conference on machine translation* (pp. 1174–1182). Online: Association for Computational Linguistics. Descargado de https://aclanthology.org/2020.wmt-1.139

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. En I. Guyon et al. (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc. Descargado de https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Vázquez, R., Scherrer, Y., Virpioja, S., y Tiedemann, J. (2021, junio). The Helsinki submission to the AmericasNLP shared task. En M. Mager et al. (Eds.), *Proceedings of the first workshop on natural language processing for indigenous languages of the americas* (pp. 255–264). Online: Association for Computational Linguistics. Descargado de https://aclanthology.org/2021.americasnlp-1.29 doi: 10.18653/v1/2021.americasnlp-1.29

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., ... Dean, J. (2016). *Google's neural machine translation system: Bridging the gap between human and machine translation.*

Ávalos, C. (2011). *Ñe'eryguasu (gran diccionario) ˜ guaraní-español, español–guaraní.*

Anexo A

Anexo 1

A.1. Formalización de un experimento de traducción

Formalizamos a continuación las entradas y salidas de un experimento.

Entrada:
Definimos formalmente el espacio de parámetros de entrada de un experimento E_{in} como:

$$E_{in} = \{(D_{train}, D_{metrics}, V, P, e_{max}, e_{interval}, e_{pretrain}, M, D, A, s, params)\}$$

Donde:

- D_{train} es el conjunto de datos de entrenamiento.
- $D_{metrics}$ es el conjunto de datos de validación o test donde serán extraidas las métricas.
- $V = \{V_{source}, V_{target}\}$ son los vocabularios de cada lengua.
- P es el conjunto de preentrenamiento.
- $e_{interval}$ es el número de *epochs* entre cada extracción de métricas.
- e_{max} es la cantidad máxima de *epochs* del experimento.
- $e_{pretrain}$ es la cantidad de *epochs* de preentrenamiento realizadas sobre el conjunto de datos de preentrenamiento P. Cuando $e_{pretrain} = 0$ no se realiza preentrenamiento.
- $M \subseteq \{BLEU, chrF\}$ es el conjunto de métricas utilizadas para evaluar al modelo cada $e_{interval}$ *epochs* hasta e_{max}.

- $d \in \{(gn, es), (es, gn)\}$ es un conjunto de pares formados por los lenguajes de origen y destino del experimento.
- $a \in \{\text{transformer}, \text{seq2seq}\}$ es el conjunto de arquitecturas utilizadas en el experimento.
- $s \in \mathbb{N}$ es la semilla utilizada para el software de traducción.
- *params* son los parámetros del software de traducción.

Salida:

Se define formalmente el espacio de salidas de un experimento E_{out} como:

$$E_{out} = \{(S, \hat{y})\}$$

Donde:

- S es la secuencia de métricas extraidas cada $e_{interval}$ *epochs* para ese experimento.
- La traducción $\hat{y}$ del experimento que pueden encontrarse para el conjunto de evaluación en el repositorio del proyecto de traducción.

Definimos entonces a un experimento como la transformación de un conjunto E_{in} a otro conjunto E_{out}.

A.2. Distribuciones de búsqueda aleatoria de pruebas de nivel 3

Los parámetros utilizados se muestran a continuación:

Modelo seq2seq:

- **learn-rate**: Valores generados por la v.a. $L \sim \text{loguniform}(0{,}0001, 1{,}0)$
- **max-length**: Valores generados por la v.a. $M \sim \mathcal{N}_{\text{trunc}}(200, 25, -3, 3)$. O sea, aquí $\mu = 200$, $\sigma = 25$, y la distribución se encuentra truncada a un radio de 3σ de μ.
- **enc-depth y dec-depth**: Valores compartidos generados por la v.a. $\mathcal{N}_{\text{trunc}}(7, 1{,}5, -3, 3)$ evitando generar valores mayores a 6 y convirtiéndolos en enteros.

- **dim-vocabs**: Sea V una v.a. con dominio $D_{dim\_vocabs} = \{(2000, 2000), (4000, 4000), (6000, 6000), (8000, 8000), (10000, 10000), (12000, 12000)\}$, donde $V \sim V(x)$, entonces v está definida por la función de densidad $v : D_{dim\_vocabs} \to [0, 1]$. La distribución de probabilidad $v(x)$ se define como:

$$v(x) = \begin{cases} 0{,}05 & \text{si } x = (2000, 2000) \\ 0{,}1 & \text{si } x = (4000, 4000) \\ 0{,}3 & \text{si } x = (6000, 6000) \\ 0{,}3 & \text{si } x = (8000, 8000) \\ 0{,}1 & \text{si } x = (10000, 10000) \\ 0{,}05 & \text{si } x = (12000, 12000) \\ 0 & \text{en otros casos} \end{cases}$$

Modelo Transformer:

- **learn-rate**: Valores generados por la v.a. $L \sim \text{U}(0{,}00001, 0{,}0001)$
- **max-length**: Valores generados por la v.a. $M \sim \mathcal{N}_{\text{trunc}}(200, 33, -3, 3)$
- **enc-depth y dec-depth**: Valores compartidos generados por la v.a. $D \sim \text{U}(1, 6)$.
- **dim-vocabs**: Sea V una v.a. con dominio $D_{dim\_vocabs} = \{(2000, 2000), (4000, 4000), (6000, 6000), (8000, 8000), (10000, 10000), (12000, 12000)\}$, donde $V \sim v(x)$, entonces v está definida por la función de densidad $v : D_{dim\_vocabs} \to [0, 1]$. La distribución de probabilidad $v(x)$ se define como:

$$v(x) = \begin{cases} 0{,}1 & \text{si } x = (2000, 2000) \\ 0{,}15 & \text{si } x = (4000, 4000) \\ 0{,}25 & \text{si } x = (6000, 6000) \\ 0{,}25 & \text{si } x = (8000, 8000) \\ 0{,}15 & \text{si } x = (10000, 10000) \\ 0{,}1 & \text{si } x = (12000, 12000) \\ 0 & \text{en otros casos} \end{cases}$$

A.3. Resultados de experimentos de nivel 1

Presentamos en este anexo las figuras A.1 y A.2 de los resultados de los experimentos de nivel 1.

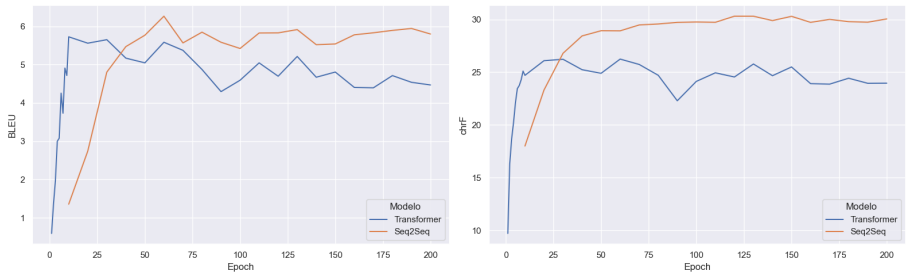

Figura A.1: Resultados de pruebas de nivel 1 en dirección guaraní-español sobre conjunto de desarrollo de Jojajovai.

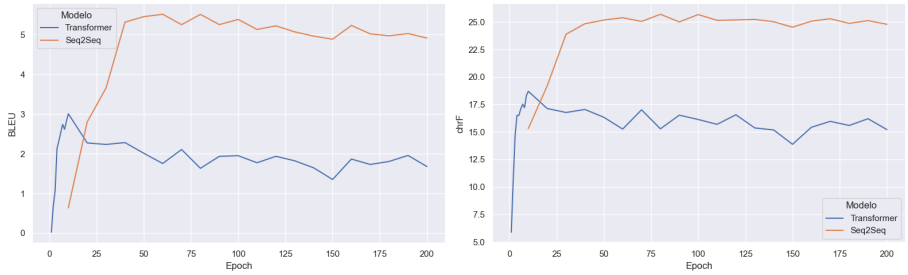

Figura A.2: Resultados de pruebas de nivel 1 en dirección español-guaraní sobre conjunto de desarrollo de Jojajovai.

A.4. Resultados de experimentos de nivel 2

Presentamos en este anexo las figuras A.3, A.4, A.5, A.6 y A.7 de los resultados de los experimentos de nivel 2.

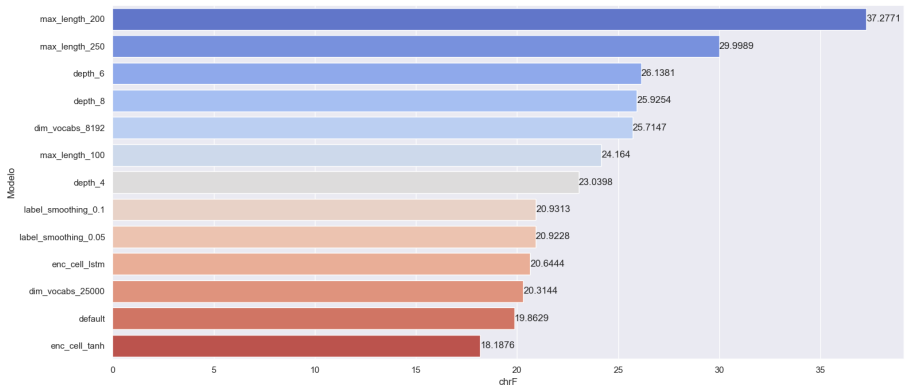

Figura A.3: Resultados de pruebas de nivel 2 en dirección guaraní-español realizadas con la arquitectura seq2seq sobre conjunto de desarrollo de Jojajovai.

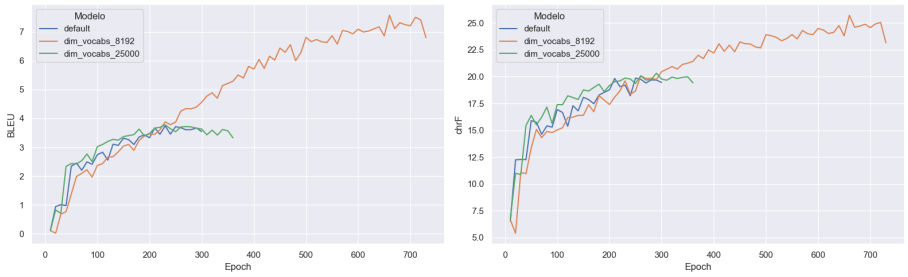

Figura A.4: Resultados de pruebas de nivel 2 para el hiperparámetro dim-vocabs en dirección guaraní-español realizadas con la arquitectura seq2seq sobre conjunto de desarrollo de Jojajovai.

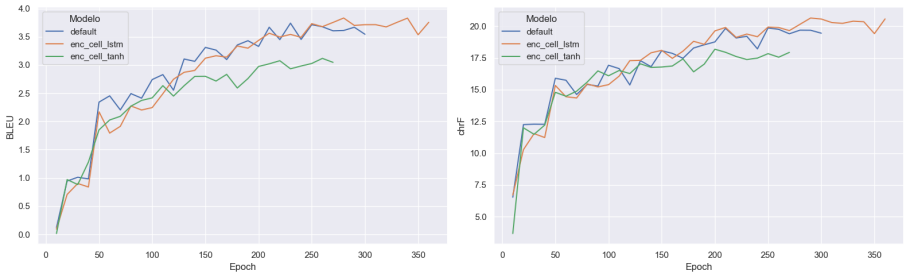

Figura A.5: Resultados de pruebas de nivel 2 para el hiperparámetro enc-cell en dirección guaraní-español realizadas con la arquitectura seq2seq sobre conjunto de desarrollo de Jojajovai.

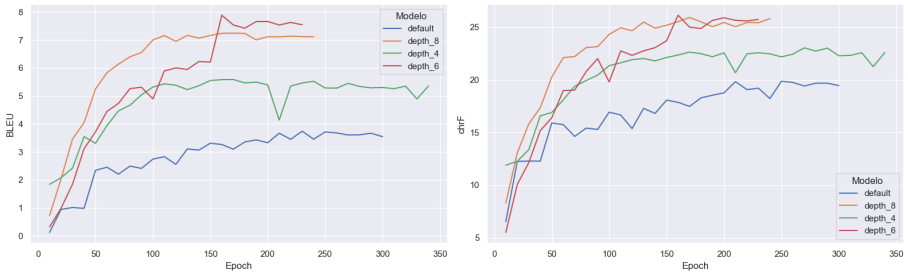

Figura A.6: Resultados de pruebas de nivel 2 para el hiperparámetro enc-depth y dec-depth en dirección guaraní-español realizadas con la arquitectura seq2seq sobre conjunto de desarrollo de Jojajovai.

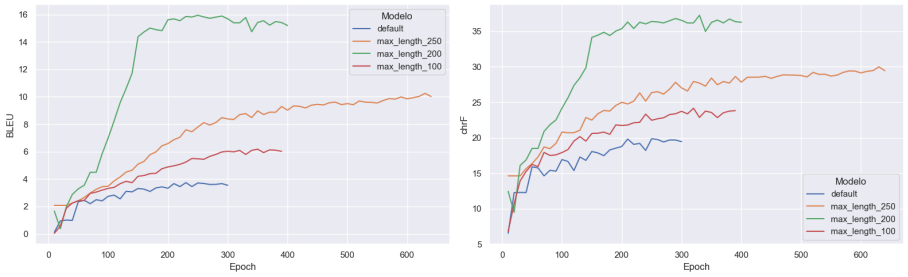

Figura A.7: Resultados de pruebas de nivel 2 para el hiperparámetro max-length en dirección guaraní-español realizadas con la arquitectura seq2seq sobre conjunto de desarrollo de Jojajovai.

A.5. Resultados completos de experimentos de nivel 3

En esta sección se encuentran los resultados completos de los experimentos de nivel 3, contenidos en las tablas A.1 y A.2.

Arquitectura	learn-rate	max-length	depth	dim-vocabs	*Epoch*	BLEU	chrF
s2s	0.0016	187	6	6000	240	25.37	47.32
s2s	0.0014	186	6	2000	50	25.06	47.06
s2s	0.00012	150	4	6000	180	24.34	46.39
s2s	0.00070	180	6	6000	130	24.33	46.02
s2s	0.011	201	4	2000	30	23.99	45.89
s2s	0.00053	177	6	8000	340	24.33	45.85
s2s	0.00062	179	4	6000	40	23.95	45.8
s2s	0.00042	175	5	8000	50	23.34	45.35
s2s	0.0014	186	5	8000	80	23.72	45.34
s2s	0.0029	191	5	8000	70	23.9	45.3
s2s	0.00054	178	6	8000	120	23.77	45.22
s2s	0.00036	173	6	8000	70	23.57	45.21
s2s	0.00017	161	5	10000	120	23.05	44.77
s2s	0.0053	196	7	8000	90	23.12	44.59
s2s	0.00018	162	6	10000	80	22.97	44.47
s2s	0.0057	196	6	6000	50	22.85	44.03
s2s	0.012	202	7	8000	60	19.17	40.01
s2s	0.0066	197	6	10000	40	18.32	38.33
s2s	0.0031	192	7	8000	160	12.17	29.93
transformer	3.95e-05	182	3	2000	920	15.88	39.72
transformer	5.50e-05	196	2	2000	990	15.75	39.18
transformer	2.47e-05	166	3	6000	890	15.11	38.15
transformer	8.85e-05	226	3	6000	580	14.9	37.86
transformer	6.50e-05	204	4	4000	640	14.84	37.83
transformer	8.57e-05	223	3	6000	570	14.86	37.78
transformer	3.37e-05	177	2	8000	830	14.57	37.23
transformer	3.40e-05	177	1	2000	920	14.12	36.68
transformer	2.40e-05	165	4	4000	760	14.11	36.67
transformer	9.26e-05	231	2	4000	150	13.89	36.54
transformer	0.00010	255	4	12000	570	14.3	36.23
transformer	2.04e-05	159	3	10000	870	14.14	36.07
transformer	8.85e-05	226	2	8000	230	13.99	36.05
transformer	4.71e-05	189	1	4000	530	12.97	35.29
transformer	3.14e-05	174	1	8000	760	13.27	34.94
transformer	8.15e-05	219	1	6000	370	13.04	34.83
transformer	6.33e-05	203	1	6000	280	12.77	34.64
transformer	9.26e-05	231	1	4000	250	12.34	34.62
transformer	5.91e-05	199	1	6000	370	12.77	34.22
transformer	5.16e-05	193	1	8000	290	12.62	34.16

Tabla A.1: Resultados para pruebas de nivel 3 en la dirección guaraní-español para cada arquitectura

Arquitectura	learn-rate	max-length	depth	dim-vocabs	*Epoch*	BLEU	chrF
s2s	0.00013	153	3	6000	190	23.32	46.2
s2s	0.00024	168	6	6000	60	23.25	45.84
s2s	0.018	204	7	6000	40	22.53	45.29
s2s	0.00042	175	6	6000	50	22.66	45.11
s2s	0.00059	178	6	6000	30	22.46	45.03
s2s	0.00052	177	6	8000	40	21.76	44.1
s2s	0.0020	189	6	6000	40	21.7	44.03
s2s	0.00018	162	6	10000	90	21.44	43.8
s2s	0.00013	155	6	10000	130	21.37	43.78
s2s	0.0015	187	6	8000	190	21.71	43.72
s2s	0.011	201	7	4000	20	4.38	43.43
s2s	0.00060	179	6	10000	40	20.76	43.01
s2s	0.0012	185	6	8000	80	20.89	42.76
s2s	0.0016	187	6	10000	140	20.87	42.74
s2s	0.0066	197	6	6000	30	20.39	42.5
s2s	0.0057	196	6	6000	30	20.69	42.5
s2s	0.0030	192	7	8000	30	19.67	41.53
s2s	0.0015	187	6	10000	70	19.61	40.96
s2s	0.012	201	6	10000	90	15.46	35.62
s2s	0.015	203	5	8000	70	6.52	22.94
transformer	5.77e-05	198	2	2000	800	15.73	40.35
transformer	5.79e-05	198	3	4000	950	13.84	38.32
transformer	3.93e-05	182	4	2000	630	13.55	38.27
transformer	9.98e-05	242	3	6000	780	13.77	37.49
transformer	9.22e-05	230	4	4000	570	12.44	35.86
transformer	2.18e-05	161	5	2000	730	11.95	35.81
transformer	0.00010	268	4	6000	820	12.47	35.74
transformer	7.55e-05	213	3	10000	960	12.53	35.72
transformer	7.22e-05	210	4	6000	850	12.33	35.7
transformer	7.77e-05	215	4	10000	980	12.16	34.75
transformer	4.52e-05	188	1	4000	980	11.15	33.39
transformer	5.58e-05	197	5	6000	1000	11.43	33.26
transformer	3.70e-05	180	1	6000	960	10.86	32.6
transformer	4.86e-05	190	1	6000	900	10.88	32.58
transformer	5.17e-05	193	1	6000	810	10.96	32.56
transformer	0.00010	245	1	4000	260	10.56	32.32
transformer	1.67e-05	151	4	8000	870	10.74	32.1
transformer	9.33e-05	232	5	6000	1000	10.64	31.66
transformer	5.70e-05	198	4	8000	220	10.39	31.29
transformer	3.72e-05	180	5	10000	1000	9.89	29.96

Tabla A.2: Resultados para pruebas de nivel 3 en la dirección español-guaraní para cada arquitectura

Anexo B

Anexo 2

B.1. Reglas gramaticales para el español

Anexamos a continuación las reglas de la gramática de rasgos desarrollada para la generación en español. Solo dos producciones léxicas se incluyen, que corresponden a aquellas que son fijas y no se desprenden de los lexicones construidos. Recordamos que la gramática utilizada en distintas etapas de nuestro proceso varía según nuestras necesidades, para evitar sobregenerar oraciones; las reglas aquí presentadas corresponden a la unión de estas subgramáticas.

```
# Grammar Productions
# S expansion productions
S[AGR=?a] -> NP[AGR=?a] VP[AGR=?a, MOOD=i]
S[AGR=?a] -> NP[AGR=?a] VP[AGR=?a, MOOD=i] PP
S[AGR=?a] -> P[AGR=?a, CASE=n] VP[AGR=?a, MOOD=i]
S[AGR=?a] -> P[AGR=?a, CASE=n] VP[AGR=?a, MOOD=i] PP
# VP expansion productions
VP[AGR=?a, MOOD=?m] -> V[AGR=?a, MOOD=?m, SUBCAT=intr]
VP[AGR=?a, MOOD=?m] -> V[AGR=?a, MOOD=?m, SUBCAT=tr] NP
VP[AGR=?a, MOOD=?m] -> V[AGR=?a, MOOD=?m, SUBCAT=di] NP PPA
VP[AGR=?a, MOOD=?m] -> NEG V[AGR=?a, MOOD=?m, SUBCAT=intr]
VP[AGR=?a, MOOD=?m] -> NEG V[AGR=?a, MOOD=?m, SUBCAT=tr] NP
VP[AGR=?a, MOOD=?m] -> NEG V[AGR=?a, MOOD=?m, SUBCAT=di] NP
    PPA
# PP expansion productions
PP -> PR NP
# PPA expansion productions
PPA[AGR=?a] -> AA NP[AGR=?a]
# NP expansion productions
NP[AGR=?a] -> D[AGR=?a] N[AGR=?a]
NP[AGR=?a] -> D[AGR=?a] N[AGR=?a] A[AGR=?a]
# Lexical Productions
AA -> 'a'
NEG -> 'no'
```

Por otro lado, para la construcción de árboles sintácticos a partir de las oraciones de Ancora, no solo utilizamos un lexicón más amplio que no requiriera conocer la traducción al guaraní (lo cual derivó en un mayor número de producciones léxicas), sino que modificamos también las reglas gramaticales para ser más flexibles. A continuación se presenta esta gramática modificada, sin incluir producciones léxicas.

```
# Grammar Productions
# S expansion productions
S[AGR=?a] -> NP[AGR=?a] VP[AGR=?a, MOOD=i]
S[AGR=?a] -> NP[AGR=?a] VP[AGR=?a, MOOD=i] PP
S[AGR=?a] -> P[AGR=?a, CASE=n] VP[AGR=?a, MOOD=i]
S[AGR=?a] -> P[AGR=?a, CASE=n] VP[AGR=?a, MOOD=i] PP
# For extra Ancora matching
S -> VP
S -> VP PP
S -> NP
S -> N
S -> A
S -> PP
S -> CON
# VP expansion productions (no restrictions on transitivity)
VP[AGR=?a, MOOD=?m] -> V[AGR=?a, MOOD=?m]
VP[AGR=?a, MOOD=?m] -> V[AGR=?a, MOOD=?m] NP
VP[AGR=?a, MOOD=?m] -> V[AGR=?a, MOOD=?m] NP PPA
VP[AGR=?a, MOOD=?m] -> NEG V[AGR=?a, MOOD=?m]
VP[AGR=?a, MOOD=?m] -> NEG V[AGR=?a, MOOD=?m] NP
VP[AGR=?a, MOOD=?m] -> NEG V[AGR=?a, MOOD=?m] NP PPA
# PP expansion productions
PP -> PR NP
# PPA expansion productions
PPA[AGR=?a] -> AA NP[AGR=?a]
# NP expansion productions (added the third one)
NP[AGR=?a] -> D[AGR=?a] N[AGR=?a]
NP[AGR=?a] -> D[AGR=?a] N[AGR=?a] A[AGR=?a]
NP[AGR=?a] -> D[AGR=?a] A[AGR=?a] N[AGR=?a]
```

B.2. Reglas de transferencia entre español y guaraní

A continuación incluimos las reglas de transferencia entre español y guaraní, en formato JSON. Las aquí presentadas son la versión extendida que incluye tanto las reglas que fueron utilizadas para la generación del corpus sintético, como aquellas desarrolladas específicamente para la traducción del corpus de Ancora.

```
"S -> NP VP PP": [
```

```
    "S[AGR=?a] -> NP[AGR=?a] VP[AGR=?a] PP"
],
"S -> NP VP": [
    "S[AGR=?a] -> NP[AGR=?a] VP[AGR=?a]"
],
"S -> P VP PP": [
    "S[AGR=?i] -> P[AGR=?i, POS=B] VP[AGR=?i, POS=B] PP",
    "S[AGR=?i] -> P[AGR=?i, POS=0] VP[AGR=?i, POS=0] PP",
    "S[AGR=?i] -> VP[AGR=?i, POS=0] P[AGR=?i, POS=0] PP",
    "S[AGR=?i] -> VP[AGR=?i, POS=P] P[AGR=?i, POS=P] PP"
],
"S -> P VP": [
    "S[AGR=?i] -> P[AGR=?i, POS=B] VP[AGR=?i, POS=B]",
    "S[AGR=?i] -> P[AGR=?i, POS=0] VP[AGR=?i, POS=0]",
    "S[AGR=?i] -> VP[AGR=?i, POS=0] P[AGR=?i, POS=0]",
    "S[AGR=?i] -> VP[AGR=?i, POS=P] P[AGR=?i, POS=P]"
],
"S -> VP": [
    "S[AGR=?a] -> VP[AGR=?a]"
],
"S -> VP PP" : [
    "S[AGR=?a] -> VP[AGR=?a] PP"
],
"S -> NP": [
    "S[AGR=?a] -> NP[AGR=?a]"
],
"S -> PP": [
    "S -> PP"
],
"S -> N": [
    "S -> N"
],
"S -> CON": [
    "S -> CON"
],
"S -> A": [
    "S -> A"
],
"VP -> V": [
    "VP[AGR=?a, POS=?p] -> V[AGR=?a, NEG=0, POS=?p]"
],
"VP -> V NP": [
    "VP[AGR=?a, POS=?p] -> V[AGR=?a, NEG=0, POS=?p] NP"
],
"VP -> V NP PPA": [
    "VP[AGR=?a, POS=?p] -> V[AGR=?a, NEG=0, POS=?p] NP PPA"
],
"VP -> NEG V": [
    "VP[AGR=?a, POS=?p] -> V[AGR=?a, NEG=1, POS=?p]"
],
```

```
"VP -> NEG V NP": [
    "VP[AGR=?a, POS=?p] -> V[AGR=?a, NEG=1, POS=?p] NP"
],
"VP -> NEG V NP PPA": [
    "VP[AGR=?a, POS=?p] -> V[AGR=?a, NEG=1, POS=?p] NP PPA"
],
"PPA -> AA NP": [
    "PPA[AGR=?a] -> NP[AGR=?a, NF=?n] AA[NF=?n]"
],
"NP -> D N": [
    "NP[AGR=?a, NF=?n] -> D[AGR=?a] N[AGR=?a, NF=?n]"
],
"NP -> D N A": [
    "NP[AGR=?a, NF=?n] -> D[AGR=?a] N[AGR=?a, NF=?n] A[AGR=?
        a]"
],
"NP -> D A N": [
    "NP[AGR=?a, NF=?n] -> D[AGR=?a] N[AGR=?a, NF=?n] A[AGR=?
        a]"
],
"PP -> PR NP": [
    "PP -> NP[AGR=?a, NF=?n] PR[AGR=?a, U=U, S=S, NF=?n]",
    "PP -> NP[AGR=?a, NF=?n] PR[AGR=?a, U=U, S=0, NF=?n]",
    "PP -> NP[AGR=?a] PR[AGR=?a, U=S, S=S]",
    "PP -> NP[AGR=?a] PR[AGR=?a, U=S, S=0]"
]
```

B.3. Eficiencia del proceso de generación de corpus

Para la generación del corpus sintético, tomamos algunas métricas de eficiencia.

En promedio, generar unas 2.500.000 oraciones y parsearlas, resultó en 22.000 oraciones aceptadas y 23.300 árboles sintácticos diferentes; la diferencia entre estos dos últimos números responde a oraciones con más de un árbol posible. Esto quiere decir que en torno al 0,9 % de las oraciones que generamos con la gramática libre de contexto superan el filtro impuesto por la gramática de rasgos.

Por otro lado, la generación de esas mismas 22.000 oraciones y 23.300 árboles tomaba unas 7 horas y 50 minutos, corriendo un solo proceso con un único hilo, en un procesador Intel Core i5-8365U. En la práctica, se corríeron siempre entre 4 y 6 procesos a la vez, uniendo los resultados finales. Vale aclarar que el cuello de botella se encuentra en el paso de filtrado: intentar construir el árbol sintáctico para una oración toma alrededor de 12 veces más que generar esa oración.

En cuanto a la traducción, para 42.000 árboles, tomó 20 minutos en promedio corriendo un solo proceso con un único hilo, usando el mismo hardware, y

resultando en 41.500 pares español-guaraní.

Para el corpus de Ancora, constituido por 14.287 oraciones, nuestro sistema tomó 2 horas en parsear las oraciones y construir los árboles sintácticos para cada segmento posible; 13 minutos en aplicar el proceso de transferencia sintáctica a los árboles para obtener traducciones de los segmentos al guaraní; y menos de 5 segundos en incrustar estas traducciones en las oraciones originales y posprocesando el corpus. Esto es un total de menos de 2 horas y 15 minutos, corriendo un solo proceso con un único hilo, en un procesador Intel Core i5-8365U.

De las 14.287 oraciones originales, obtuvimos un total de 14.120 pares español-guaraní.