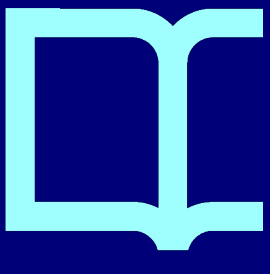

Entrenamiento de motores de traducción automática

1. INTRODUCCIÓN

En este capítulo vamos a presentar el proceso de entrenamiento de sistemas de traducción automática, tanto estadísticos como neuronales. Los sistemas de traducción automática se pueden dividir en dos grandes grupos:

- Sistemas basados en reglas, que son sistemas que se desarrollan, es decir, un equipo de lingüistas y programadores escriben reglas, diccionarios y programas que son capaces de traducir oraciones en una determinada lengua de partida a una determinada lengua de llegada.
- Sistemas basados en corpus, que son sistemas que se entrenan, es decir, que "aprenden" a traducir a partir de un conjunto de oraciones en una determinada lengua de partida y sus correspondientes traducciones a una determinada lengua de llegada, lo que se conoce como corpus paralelo. Una vez entrenado, el sistema resultante es capaz de traducir nuevas oraciones, aunque no estén en el corpus de entrenamiento.

En este capítulo nos vamos a centrar en los sistemas basados en corpus: los sistemas de traducción automática estadística y neuronal.

Los sistemas de traducción automática estadística son capaces de traducir a partir de cálculo de dos probabilidades: la probabilidad de que una determinada oración en la lengua de llegada sea la traducción de una determinada oración en la lengua de partida y la probabilidad de que una determinada oración en la lengua de llegada sea una oración correcta en esa lengua. La primera de estas probabilidades se calcula a partir de un modelo de traducción. Este modelo de traducción se construye a partir de un corpus paralelo de la lengua de partida y de llegada. La segunda probabilidad se calcula a partir de un modelo de lengua de la lengua de llegada. Este modelo de lengua se construye a partir de un corpus monolingüe de la lengua de llegada, o bien de la parte del corpus paralelo correspondiente a la lengua de llegada.

Los sistemas de traducción automática neuronal se basan en el entrenamiento de redes neuronales. Estas redes neuronales están compuestas por diversas capas de neuronas artificiales interconectadas entre ellas. Una neurona artificial es una función matemática que recibe una o más entradas. Cada una de estas entradas reciben un valor de ponderación. La neurona tiene asociada una determinada función de activación que será la que calcule el valor de salida. El entrenamiento de la red neuronal consiste en encontrar los valores óptimos de estas ponderaciones para que realice la acción deseada, en nuestro caso, traducir. Para entrenar las redes neuronales necesitaremos representar las oraciones de entrada numéricamente. Para ello se representan las palabras de la lengua mediante vectores con muchas dimensiones, los denominados *word embeddings*. No nos extenderemos aquí en el funcionamiento de las redes neuronales. El lector interesado puede leer la interesante introducción de Forcada (2017).

En general, los sistemas neuronales se entrenan utilizando corpus paralelos. A este tipo de entrenamiento se le denomina entrenamiento supervisado. Dado que para algunos pares de lenguas no

existen corpus paralelos ni tampoco suficientes textos originales y traducidos para crearlos, se han desarrollado técnicas para el entrenamiento de sistemas de traducción utilizando corpus comparables (Artetxe et al., 2019). En este capítulo únicamente explicaremos cómo llevar a cabo un entrenamiento supervisado, es decir, utilizando corpus paralelos.

2. OBTENCIÓN Y CREACIÓN DE CORPUS PARALELOS

2.1. Obtención de corpus paralelos

Existen diversos repositorios en Internet donde se pueden descargar corpus paralelos. De entre estos repositorios, sin duda, destaca Opus Corpus[1] (Tiedemann, 2012). En esta colección encontraremos corpus paralelos para muchísimos pares de lenguas. En estos corpus por regla general no sabemos cuál de las dos lenguas es la original y cuál la traducida, o incluso si ambas lenguas son la traducción de originales en una tercera lengua. Desde Opus Corpus se pueden descargar corpus paralelos en dos formatos: Moses y TMX.

El formato Moses consiste en dos archivos de texto. Uno de ellos contiene los segmentos en la lengua de partida y el otro los segmentos en la lengua de llegada. Estos dos archivos están alineados línea a línea, es decir, la oración de la primera línea de uno de los archivos se corresponden con el de la misma línea en el otro archivo. El formato Moses es el que más se utiliza para entrenar sistemas de traducción automática. Durante el preprocesamiento del corpus es muy importante mantener la relación entre los segmentos. Así, es frecuente convertir el formato Moses en un archivo de texto tabulado (tsv), donde en cada línea tenemos el segmento en la lengua de partida, un tabulador y el segmento en la lengua de llegada.

El formato TMX (*Translation Memory eXchange*) es un formato XML estándar para el intercambio de memorias de traducción. Los conceptos de memoria de traducción y de corpus paralelo son muy similares. Si disponemos de memorias de traducción, ya sea en formato TMX como en algún formato propietario de algún sistema de traducción asistida por ordenador (como SDLTM, por ejemplo, correspondiente a Trados Studio), las podremos convertir a formato Moses o a formato tsv y utilizarlas para el entrenamiento de sistemas.

2.2. Creación de corpus paralelos

Si disponemos de una serie de documentos en una lengua y su traducción a otra lengua, podemos crear un corpus paralelo utilizando algún programa de alineación automática de documento, como por ejemplo Hunalign[2] (Varga et al., 2005). El proceso de alineación automática consta de los siguientes pasos:

1. Conversión de los documentos a formato texto. Cuando hablamos de documentos nos referimos tanto a textos, como a páginas web multilingües que hayamos descargado o a archivos en diferentes formatos. Será necesario convertir estos archivos a un formato de texto. A menudo esta conversión no es perfecta, como cuando tratamos con archivos PDF, y requieren algo de esfuerzo manual en arreglarlos.

[1] https://opus.nlpl.eu/
[2] https://github.com/danielvarga/hunalign

2. Segmentación de los archivos de texto. El proceso de segmentación consiste en dividir el texto de los documentos, que probablemente esté organizado en párrafos, en segmentos. Un segmento es una unidad que se asemeja a una oración, aunque no en todos los casos coincide con una oración gramatical. Habitualmente el proceso de segmentación se realiza utilizando un conjunto de reglas que indican posiciones del texto donde debe haber un salto de segmento, o por el contrario, donde no se debe producir un salto de segmento. Estas reglas se pueden encontrar en el formato estándar SRX (*Segmentation Rules eXchange*). A menudo, en el momento de la segmentación se añade una marca de párrafo, que será útil para el algoritmo de alineación. A continuación podemos ver un ejemplo de texto antes y después de la segmentación[3].

Texto antes de la segmentación:

```
The front line shifts on a daily basis, and both sides take and lose territory regularly. The
human cost remains extremely high, with dozens Ukrainian and Russian soldiers, as well as
Ukrainian civilians, dying on an almost daily basis.

The daily threats to Ukrainian security have also affected its society and institutions. Key
national debates are now centered on the need to accelerate reforms, the role of women, media
plurality, the approach to Russian culture, a new identity defined by the war effort, and the
even more crucial role of digitization.
```

Texto después de la segmentación:

```
The front line shifts on a daily basis, and both sides take and lose territory regularly.
The human cost remains extremely high, with dozens Ukrainian and Russian soldiers, as well as
Ukrainian civilians, dying on an almost daily basis.
<p>
The daily threats to Ukrainian security have also affected its society and institutions.
Key national debates are now centered on the need to accelerate reforms, the role of women,
media plurality, the approach to Russian culture, a new identity defined by the war effort, and
the even more crucial role of digitization.
```

3. Alineación automática. Algunos programas, como Hunalign, admiten opcionalmente un diccionario bilingüe entre la lengua de partida y de llegada. Los algoritmos de alineación acostumbran a proporcionar los segmentos alineados junto algún índice que nos indica la fiabilidad de cada alineación, como en el siguiente ejemplo.

```
The front line shifts on a daily basis, and both sides take and lose territory regularly.
        La línea del frente cambia a diario, y ambos bandos toman y pierden territorio con
regularidad.    1.125
The human cost remains extremely high, with dozens Ukrainian and Russian soldiers, as well as
Ukrainian civilians, dying on an almost daily basis.        El coste humano sigue siendo
extremadamente alto, ya que docenas de soldados ucranianos y rusos, así como civiles ucranianos,
mueren casi a diario.  1,231
<p>     <p>     0
The daily threats to Ukrainian security have also affected its society and institutions.  Las
amenazas diarias a la seguridad de Ucrania también han afectado a su sociedad e instituciones.
        1,078
Key national debates are now centered on the need to accelerate reforms, the role of women,
media plurality, the approach to Russian culture, a new identity defined by the war effort, and
the even more crucial role of digitization.  Los principales debates nacionales se centran
ahora en la necesidad de acelerar las reformas, el papel de las mujeres, la pluralidad de los
medios de comunicación, el acercamiento a la cultura rusa, una nueva identidad definida por el
esfuerzo bélico y el papel aún más crucial de la digitalización.    1,123
```

[3] Texto obtenido de: https://globalvoices.org/special/russia-invades-ukraine/

El índice de fiabilidad nos permite seleccionar los segmentos alineados que tengan una mayor probabilidad de estar correctamente alineados.

Recientemente han aparecido técnicas que permiten detectar oraciones traducidas en corpus no paralelos (sean comparables o no). Estas técnicas se basan en representaciones vectoriales de las oraciones mediante modelos independientes de la lengua (o al menos válidos para un gran número de lenguas), como por ejemplo SBERT[4] (Reimers y Gurevych, 2019). Con esta representación vectorial de las oraciones, las oraciones con significados similares, aunque estén en lenguas distintas, estarán cercanas en el espacio multidimensional. Algunos de los corpus paralelos disponibles más extensos, como por ejemplo CCMatrix (Schwensk et al., 2021) se han recopilado utilizando estas técnicas.

2.3. Combinación de corpus paralelos

Una situación muy habitual cuando queremos entrenar un sistema de traducción automática, es que dispongamos de un corpus pequeño o mediano de especialidad y de un corpus general de gran tamaño. Por ejemplo, queremos entrenar un sistema inglés-español especializado en ingeniería aeronáutica. De nuestras memorias de traducción podemos obtener un corpus de 100.000 segmentos, muy lejos de los 10 millones de segmentos que necesitaríamos para entrenar un sistema neuronal. No hay ningún corpus paralelo para este par de lenguas y especialidad en Opus Corpus. Pero sí que hay disponibles algunos corpus generales de muy gran tamaño, como CCMatrix, con más de 400 millones de segmentos para este par de lenguas. En este punto, lo que nos interesaría sería seleccionar un conjunto de segmentos (por ejemplo 10 millones) que sean "parecidos" a los segmentos de nuestro pequeño corpus paralelo. Para realizar esta selección podemos calcular un modelo de lengua de la lengua de partida a partir de nuestro pequeño corpus paralelo y calcular la perplejidad de cada uno de los segmentos en la lengua de partida del corpus de gran tamaño. Acabado el cálculo, podremos seleccionar los 10 millones de segmentos paralelos cuya parte correspondiente a la lengua de partida tenga una menor perplejidad respecto al modelo de lengua.

Para entender un poco más este procedimiento vamos a explicar qué es un modelo de lengua y cómo se calcula la perplejidad. Un modelo de lengua es un archivo que contiene estadísticas sobre la aparición de las palabras y n-gramas (combinación de n palabras) que se calcula a partir de un corpus monolingüe de la lengua en cuestión. A continuación podemos observar algunos fragmentos de un modelo de lengua para el inglés.

```
1-grams:
-1.285941       a       -0.69897
-1.687872       also    -0.30103
-1.687872       beyond  -0.30103
….
\2-grams:
-0.09132547     a little        -0.69897
-0.2922095      also call
-0.2922095      beyond immediate
-0.2705918      biarritz .
-0.2922095      call for
-0.2922095      concerns in
…
3-grams:
```

[4] https://www.sbert.net/

```
-0.0283603     on a little    -0.4771212
-0.0283603     screening a little     -0.4771212
-0.01660496    a little more  -0.09409451
…
\4-grams:
-0.009249173   looking on a little    -0.4771212
-0.005464747   on a little more       -0.4771212
-0.005464747   screening a little more
-0.1453306     a little more loin
…
\5-grams:
-0.001813953   looking on a little more
-0.0432557     on a little more loin
-5      also would consider higher looking
```

A partir de este modelo de lengua se puede calcular la perplejidad de una determinada oración. Cuanto más baja sea la perplejidad, más se parece la oración a las oraciones definidas por el modelo de lengua, es decir, más se "parece" esta oración a las oraciones del corpus a partir del cual se ha calculado el modelo. Por ejemplo, si calculamos un modelo de lengua utilizando la parte inglesa del corpus ECB (European Central Bank, y calculamos la perplejidad para una oración relacionada con la banca, y la perplejidad para una oración no relacionada, el valor de perplejidad será mayor para la segunda oración. Podemos observar estos resultados en la tabla 1.

Oración	Perplejidad
The Central Bank has modified the interest rate .	119.835
I like to read books in the evening .	474.689

Tabla 1. Valores de perplejidad de dos oraciones respecto a un modelo de lengua correspondiente al corpus ECB en inglés.

Esta técnica para combinar un corpus paralelo pequeño con uno grande utiliza únicamente la parte correspondiente a la lengua de partida del corpus pequeño. Se puede utilizar esta misma técnica para crear un corpus paralelo a partir de un corpus monolingüe de la lengua de partida y un corpus paralelo muy grande entre la lengua de partida y la lengua de llegada.

3. PREPROCESAMIENTO DE CORPUS

3.1. Eliminación de segmentos repetidos

Uno de los primeros pasos de preprocesamiento suele ser la eliminación de los segmentos repetidos. Habitualmente se trabaja con corpus paralelos en formato Moses, que son dos archivos que contienen los segmentos en la lengua de partida (por ejemplo inglés, en) y los archivos en la lengua de llegada (por ejemplo español, es), alineados linea a linea. La eliminación de segmentos repetidos no se puede llevar a cabo separadamente para cada archivo, ya que probablemente perderíamos la alineación. Por ello, antes de eliminar los segmentos repetidos se suele convertir el corpus de formato Moses a formato

de texto tabulado. Esta operación, en un terminal de Linux, es muy sencilla. Si los archivos del corpus se llaman corpus.en y corpus.es, podemos hacer

```
paste corpus.en corpus.es > corpus-en-es.txt
```

Ahora, para eliminar los archivos podemos concatenar con *pipes* las instrucciones *sort* (que ordena el archivo alfabéticamente), *uniq* (que elimina los repetidos si el archivo está ordenado alfabéticament) y *shuf* (que desordena el archivo de manera aleatoria, paso muy importante para evitar que el corpus resultante esté ordenado alfabéticamente, lo que perjudicaría al entrenamiento). Podemos escribir en un terminal de Linux:

```
cat corpus-en-es.txt | sort | uniq | shuf > corpus-uniq-en-es.txt
```

Las dos operaciones anteriores se pueden llevar a cabo en una sola instrucción escribiendo:

```
paste corpus.en corpus.es | sort | uniq | shuf > corpus-uniq-en-es.txt
```

En el archivo corpus-uniq-en-es.txt tendremos el corpus sin los archivos repetidos.

3.2. Limpieza de corpus

Una vez tenemos el corpus paralelo en formato texto separado por tabuladores, es aconsejable realizar un proceso de limpieza antes de empezar los pasos de entrenamiento. Los corpus paralelos pueden presentar diversos problemas:

- Presencia de caracteres representados por su entidad HTML/XML. Por ejemplo, nos podemos encontrar apariciones de la a acentuada como "á" o bien como "á". Conviene representar estos caracteres siempre de la misma manera, y preferentemente evitando las entidades.
- Caracteres con errores de codificación, como por ejemplo la "á" representada como "  " o bien como "Ã¡".
- Segmentos vacíos en alguna de las lenguas.
- Segmentos en lenguas diferentes de las requeridas.
- Segmentos con un porcentaje alto de expresiones numéricas.
- Segmentos con etiquetas XML. Normalmente se entrenan los sistemas con corpus donde se han eliminado estas etiquetas y en el momento de traducir se recuperan automáticamente.

Existen diversos programas que permiten realizar estas verificaciones y otras adicionales.[5]

3.3. Tokenización

La tokenización consiste en separar las unidades léxicas de un segmento (palabras, expresiones numéricas, signos de puntuación, etc.) por un determinado carácter, habitualmente un espacio. Por ejemplo, si tenemos la oración inglesa:

[5] Como por ejemplo MTUOC-clean-parallel-corpus: https://github.com/aoliverg/MTUOC-clean-parallel-corpus o bifixer https://github.com/bitextor/bifixer

```
About 8% of the world's annual CO2 emissions are related to the global tourism industry.
```

Su versión tokenizada sería:

```
About 8 % of the world 's annual CO2 emissions are related to the global tourism industry .
```

La operación inversa a tokenizar se llama detokenizar, y consiste en recuperar la oración tal y como tiene que ser por las normas ortotipográficas. Es decir, volver a obtener:

```
About 8% of the world's annual CO2 emissions are related to the global tourism industry.
```

Algunas lenguas, como el chino o el japonés, no separan las palabras con espacios en blanco, si no que se escriben todos los caracteres seguidos. Por ejemplo en la oración:

毛澤東思想、馬克思主義可戰 勝新冠肺炎？中共狂灌輸中小學生

También es importante tener en cuenta que una palabra puede estar formada por uno o más caracteres. Los tokenizadores para el chino son más complejos que los del inglés, castellano y otras lenguas alfabéticas que separan las palabras por espacios. La frase tokenizada automáticamente (al tratarse de un proceso automático se pueden producir errores) sería:

毛澤東思想 、 馬克思 主 義可戰 勝新冠 肺炎 ？ 中共 狂 灌輸 中小學生

Para lenguas como el chino o el japonés, a menudo se recurre a una pseudotokenización, que consiste a separar todos los caracteres por un espacio. Aunque desde el punto de vista lingüístico este proceso no tiene ningún sentido, los motores de traducción neuronales se pueden entrenar con estos corpus pseudotokenizados.

毛 澤 東 思 想 、 馬 克 思 主 義 可 戰 勝 新 冠 肺 炎 ？ 中 共 狂 灌 輸 中 小 學 生

3.4. Truecasing

El *truecasing* consiste en asignar a cada token las mayúsculas y minúsculas como le corresponde, independientemente de la posición en la oración o del estilo de capitalización de la oración. Para realizar esta operación primero tenemos que disponer de un modelo de truecasing que se entrena a partir de diccionarios de formas de palabras y de los propios corpus.

Es importante recordar que el *truecasing* solo tiene sentido para aquellas lenguas que disponen de mayúsculas y minúsculas. Únicamente las lenguas que utilizan los siguientes alfabetos tienen letras mayúsculas y minúsculas: armenio, cirílico, griego y latín. Por lo tanto, las siguientes lenguas[6] no hacen distinción de mayúsculas y minúsculas y no necesitan el proceso de *truecasing*: amárico, árabe, asamés,

[6] Lista traducida de https://www.quora.com/Which-languages-have-no-capitalized-letter

azerbaiyano, brahui, balinés, baluchi, batak (toba), baybayin, bengalí, bilén, birmano, chino, georgiano, guyaratí, gurmují, hebreo, hindi, japonés, canarés o kannada, cachemir, jémer, coreano, kurdo, laosiano, lontara, malayalam, brahmi medio, odia, pastún, persa, punjabi, sindhi, cingalés, sundanés, sylheti, tamil, telugu, tailandés, tibetano, tigré, tigriña, tirhuta, urdu, uigur y yídish.

Por ejemplo, si tenemos una oración como:

```
La Primavera llegará pronto a Barcelona.
```

(donde por motivos estilísticos se ha puesto primavera con la primera letra en mayúsculas). Su correspondiente oración truecased es:

```
la primavera llegará pronto a Barcelona.
```

El proceso contrario al truecasing es el detruecasing, donde habitualmente solo se pone en mayúscula la primera palabra de la oración (si las normas ortotipográficas de la lengua en cuestión así lo requieren).

El proceso de truecasing es muy habitual cuando entrenamos sistemas estadísticos. Para los sistemas neuronales donde se utilice SentencePiece (algoritmo que veremos más adelante), el truecasing es opcional, ya que el propio algoritmo de SentencePiece es capaz de tratar muchos de los casos de mayúsculas/minúsculas.

Es importante que cuando aplicamos truecasing, se aplica tanto a los corpus de entrenamiento antes de entrenar el sistema, como una vez en funcionamiento el sistema entrenado para hacer truecasing de la oración de entrada al sistema y detruecasing en la oración de salida traducida.

3.5. Tratamiento de expresiones numéricas

Las expresiones numéricas requieren un tratamiento especial tanto en los sistemas estadísticos como en los neuronales. Hay que recordar que existe un número infinito de expresiones numéricas diferentes, por lo que aprender la traducción de cada una de ellas sería imposible. La estrategia de tratamiento de expresiones numéricas será diferente para los sistemas estadísticos y neuronales. Como ejemplo, veremos cómo se tratará una oración como la siguiente:

```
Transport-related carbon dioxide emissions from tourism grew at least 60% between
2005 – 2016.
```

- Sistemas estadísticos: las expresiones numéricas se sustituirán por un código (por ejemplo @NUM@) en los corpus de entrenamiento y en las oraciones a traducir por el sistema y se restaurarán en las oraciones traducidas. La oración de ejemplo, aplicando esta estrategia quedaría:

```
Transport-related carbon dioxide emissions from tourism grew at least @NUM@ between
@NUM@ – @NUM@.
```

- Sistemas neuronales: se separan las cifras de las expresiones numéricas con espacios en blanco en el corpus de entrenamiento y en las oraciones a traducir. Se eliminan los espacios en blanco entre cifras cuando se recibe una traducción. La oración de ejemplo, aplicando esta estrategia quedaría:

```
Transport-related carbon dioxide emissions from tourism grew at least 6 0 % between
2 0 0 5 – 2 0 1 6 .
```

3.6. Tratamiento de emails y URLs

Estos dos elementos, emails y URLs, en general no son traducibles y requieren un tratamiento especial. Normalmente se reemplazan por unos códigos tanto en el corpus de entrenamiento como en la oración a traducir, y se restauran una vez traducida la oración por los correspondientes originales:

- Reemplazo de los emails por un código (por ejemplo @EMAIL@).
- Reemplazo de las URLs por un código (por ejemplo@URL@).

3.7. Subpalabras

Los primeros sistemas neuronales tenían un problema importante en la extensión del vocabulario, es decir, en el número de palabras diferentes (en realidad, de formas, es decir, de palabras flexionadas) capaz de traducir. Dado que cada palabra (en realidad forma) se convierte en un vector y el sistema trabaja con estos vectores, el número de palabras era finito. Si el número de palabras posible era muy grande, el sistema requería mucha memoria para trabajar, y si era pequeño, el número de palabras desconocidas era demasiado elevada.

Con estos sistemas, ocurría a menudo que una oración como "I spent my holidays in Spain" en un sistema que no tenía "Spain" en su diccionario pero sí France, por ejemplo, la oración se traducía por "He pasado mis vacaciones en Francia". Una estrategia utilizada era sustituir las palabras desconocidas por una etiqueta como por ejemplo <unk> , de manera que la oración de partida quedaba "I spent my holidays in <unk>", que se traducía por "He pasado mis vacaciones en <unk>". Una vez obtenida la traducción se recuperaba el original de la palabra desconocida y se ponía en la traducción, o bien directamente, quedando "He pasado mis vacaciones en Spain.", o bien consultando un diccionario, que si contenía el par Spain-España, resultaba en la traducción correcta: "He pasado mis vacaciones en España".

Posteriormente la estrategia más extendida para solucionar el problema de la limitación de vocabulario ha pasado a ser trabajar con subpalabras (subwords), es decir, con fragmentos de palabras. De esta manera, las palabras más frecuentes (en realidad formas) se utilizan enteras, pero las menos frecuentes se dividen en fragmentos que sean frecuentes. De esta manera se pueden representar todas las palabras, ya sean enteras o por fragmentos. El sistema traduce palabras y subpalabras dando como resultado una traducción con palabras y subpalabras. Al final del proceso, se unen los trozos de las subpalabras traducidas.

En esta sección presentamos dos de los algoritmos utilizados para el cálculo y uso de subpalabras: subword-nmt y SentencePiece. Como veremos un poco más adelante, SentencePiece va más allá del cálculo y uso de subpalabras y pretende ser un tokenizador y detokenizador universal no supervisado.

3.7.a. subword-nmt

El algoritmo subword-nmt[7] (Sennrich, Haddow and Birch, 2015) implementa la idea de dividir las palabras menos frecuentes por fragmentos de palabras que sí que sean frecuentes, con el objetivo de superar el problema del límite del vocabulario en los sistemas neuronales. Los autores se basan en la intuición de que diversas clases de palabras se pueden traducir mediantes unidades más pequeñas que palabras. Los autores mencionan algunos ejemplos: los nombre propios, mediante la copia de caracteres o la transliteración; las palabras compuestas, mediante la traducción de cada componente y los cognados y préstamos, mediante transformaciones fonológicas y morfológicas. Los autores implementan y analizan diversas técnicas de segmentación de palabras, desde una segmentación basada en modelos de n-gramas de palabras, hasta una basada en el algoritmo de compresión byte pair encoding (BPE).

El proceso de utilización de subword-nmt consta de una paso de entrenamiento del modelo de segmentación y en un paso de segmentación del corpus de entrenamiento utilizando el modelo entrenado. Una vez entrenado el sistema, el mismo modelo de segmentación se utilizará para segmentar la oración de entrada al sistema. La traducción ofrecida por el sistema tendrá también segmentación en subpalabras, que posteriormente serán convertidos en palabras uniendo los diferentes componentes.

Los autores hacen la siguiente recomendación en el momento de entrenar y utilizar el modelo de segmentación:

- Para lenguas que utilizan el mismo alfabeto: entrenar un único modelo de segmentación a la vez, es decir, concatenando el corpus de la lengua de partida y el de la lengua de llegada. Así, por ejemplo, utilizaremos esta opción para sistemas inglés-español, ya que ambas lenguas utilizan el alfabeto latino, aunque algunos caracteres aparezcan solo en una de las lenguas (como la ñ, á, é, etc.)
- Para lenguas que no utilizan el mismo alfabeto: entrenar dos modelos separados, uno para cada lengua. Esta seria la opción para entrenar un sistema chino-español, por ejemplo.

Si tenemos un modelo calculado para un par inglés-español, por ejemplo, y tenemos la siguiente oración inglesa:

```
Countless local communities are forced from their neighborhoods as gentrification and rising prices make life unaffordable.
```

Si aplicamos subword-nmt obtendremos el siguiente resultado (recordemos que la oración previamente estará tokenizada y truecased):

```
countless local communities are forced from their neighbor@@ hoods as g@@ ent@@ rific@@ ation and rising prices make life una@@ f@@ ford@@ able .
```

[7] https://github.com/rsennrich/subword-nmt

Donde las palabras poco frecuentes se han dividido en subpalabras: neighborhoods en neighbor@@ hoods; gentrification en g@@ ent@@ rific@@ ation y unaffordable en una@@ f@@ ford@@ able.

El sistema de traducción automática recibirá la oración con las subpalabras y devolverá una traducción que contendrá también subpalabras (y que además estará tokenizada y truecased):

```
innumerables comunidades locales se ven obligadas a abandonar sus vecind@@  arios a medida que la g@@ ent@@ rific@@ ación y el aumento de los precios hacen que la vida sea in@@ ase@@ qui@@ ble .
```

Donde también las palabras poco frecuentes están divididas en subpalabras: vecindarios en vecind@@ arios; gentrificación en g@@ ent@@ rific@@ ación e inasquible en in@@ ase@@ qui@@ ble.

Para deshacer la segmentación es suficiente aplicar una expresión regular:

```
sed -r 's/(@@ )|(@@ ?\$)//g'
```

Que nos devolverá:

```
Innumerables comunidades locales se ven obligadas a abandonar sus vecindarios a medida que la gentrificación y el aumento de los precios hacen que la vida sea inasequible .
```

que ahora solo tendremos que de detokenizar y aplicar detruecase, obteniendo la oración final traducida:

```
Innumerables comunidades locales se ven obligadas a abandonar sus vecindarios a medida que la gentrificación y el aumento de los precios hacen que la vida sea inasequible.
```

Cuando utilizamos subword-nmt es imprescindible también utilizar tokenización y truecasing.

3.7.b. SentencePiece

SentencePiece[8] (Kudo and Richardson, 2018) es una algoritmo para el cálculo de subpalabras, que pretende ser también un tokenizador y detokenizador no supervisado que además no requiere el proceso truecasing.

Esto significa que SentencePiece se puede utilizar como algoritmo de preprocesamiento único, sin requerir ningún otro procesamiento específico para una lengua determinada. Así pues, quedara a nuestro criterio optar por:

- Llevar a cabo un proceso de tokenización y truecasing previamente a la aplicación de SentencePiece.
- Utilizar únicamente SentencePiece.

[8] https://github.com/google/sentencepiece

Aunque no haya una regla única para decidir entre una opción u otra, aquí dejo unos consejos:

- Si los textos a traducir son muy libres en el uso de oraciones en mayúsculas (por ejemplo en literatura donde los títulos de los capítulos pueden ir en mayúculas), es mejor optar por tokenización, truecasing y SentencePiece.
- En los casos en que el uso de palabras u oraciones en mayúsculas están limitados a pocos casos repetitivos (por ejemplo textos de diarios oficiales de gobiernos) es mejor utilizar solo SentencePiece.

La aplicación de SentencePiece requiere un paso de entrenamiento, donde aplica también el criterio se subword-nmt respecto a la unión de las lenguas para las que comparten alfabeto y el cálculo de modelos separados para las lenguas que no comparten vocabulario.

A continuación podemos observar la misma oración del ejemplo de subword-nmt, pero aplicando SentencePiece. La oración original es (que trataremos sin ningún preprocesamiento previo):

Aplicando SentencePiece obtenemos:

```
▁Coun t less ▁local ▁communities ▁are ▁forced ▁from ▁their ▁neighbor hoods ▁as ▁g ent rific ation ▁and ▁rising ▁prices ▁make ▁life ▁una f ford able .
```

Vemos en este caso que los espacios en blanco se representan por un carácter especial (▁) y que las subpalabras se dividen por espacios. La división és prácticamente igual que en el caso de subword-nmt, pero ahora Countless (que no estaba truecased) se divide en ▁Coun t less. Si enviamos a traducir esta oración obtendremos la oración traducida también con SentencePiece:

```
▁In numer ables ▁comunidades ▁locales ▁se ▁ven ▁obligadas ▁a ▁abandonar ▁sus ▁vecind arios ▁a ▁medida ▁que ▁la ▁g ent rific ación ▁y ▁el ▁aumento ▁de ▁los ▁precios ▁hacen ▁que ▁la ▁vida ▁sea ▁in ase qui ble .
```

Ahora podemos recuperar la oración en español eliminando los espacios en blanco y después substituyendo los caracteres especiales "▁" por espacios, resultando en:

```
Innumerables comunidades locales se ven obligadas a abandonar sus vecindarios a medida que la gentrificación y el aumento de los precios hacen que la vida sea inasequible.
```

3.8. División del corpus en train, val y eval

Antes de entrenar el sistema, el corpus paralelo de entrenamiento se divide en tres fragmentos. Los nombres de estos fragmentos pueden variar dependiendo del tipo de sistema a entrenar

- fragmento de entrenamiento (train): será el fragmento que sirva para entrenar el sistema, ya sea estadístico o neuronal.
- fragmento de validación (val) o de optimización. En los sistemas neuronales, cada cierto número de iteraciones se realiza un proceso de validación, para ver si se está mejorando en calidad o no. En los sistemas estadísticos, una vez entrenado el sistema, se procede a optimizar los parámetros utilizando este fragmento del corpus.

- fragmento de evaluación (eval): será el fragmento que se utilice para evaluar el sistema una vez entrenado.

Es importante que los segmentos del corpus de validación y de evaluación no estén también en el corpus de entrenamiento. Si durante el procesamiento hemos eliminado los segmentos repetidos, nos aseguramos de que esto no pase.

Los fragmentos de validación y de evaluación acostumbran a tener entre 1000 y 5000 segmentos, y el resto de segmentos disponibles se dedican al fragmento de entrenamiento.

4. ENTRENAMIENTO DEL SISTEMA

4.1. Toolkits de traducción automática

Todos los programas necesarios para poder entrenar sistemas de traducción automática, ya sea estadística como neuronal, se pueden conseguir descargando alguno de los *toolkits* disponibles. Estos *toolkits* proporcionan todas las herramientas necesarias para poder entrenar sistemas de traducción automática y traducir con los sistemas entrenados. La mayoría de estos *toolkits* se distribuyen bajo licencias libres muy persmisivas, por lo que disponer de estos programas no supone de ninguna inversión en software. A continuación enumero algunos de los toolkits más utilizados:

- Moses[9] (Koehn et al., 2007): es un *toolkit* para el entrenamiento de motores estadísticos. Algunas de sus herramientas de preprocesamiento (como tokenizadores y truecasers) todavía se utilizan para el entrenamiento de motores neuronales.
- Marian[10] (Junczys-Dowmunt et al., 2018): es un *toolkit* para el entrenamiento de motores neuronales. Está desarrollado en C++ por lo que es muy rápido y eficiente.
- OpenNMT[11] (Klein, 2017): también es un *toolkit* para el entrenamiento de motores neuronales. Se distribuyen dos implementaciones: una basada en Python y PyTorch (OpenNMT-py) y otra basada en TensorFlow (OpenNMT-tf). Incluye algunos subproyectos entre los que cabe destacar CTranslate2[12], un motor de inferencia rápido para los modelos entrenados con OpenNMT y Tokenizer[13], una librería para la tokenización que dispone de APIs para C++ y Python.
- ModernMT[14] (Bertoldi, Carroselli and Federico, 2018): hasta la versión 2.5 proporcionaba la posibilidad de entrenar tanto motores estadísticos como neuronales, pero las versiones posteriores únicamente permiten entrenar motores neuronales. Destaca por su facilidad de entrenamiento y por proporcionar traducción automática adaptativa. De esta manera, la traducción proporcionada dependerá del contexto en que se encuentre la oración a traducir.

Dos cosas muy importantes también a tener en cuenta son:

[9] http://www.statmt.org/moses
[10]https://marian-nmt.github.io/
[11] https://opennmt.net/
[12] https://github.com/OpenNMT/CTranslate2
[13] https://github.com/OpenNMT/Tokenizer
[14] https://github.com/modernmt/modernmt

- En general, estos *toolkits* funcionan bien bajo Linux, aunque algunos de ellos también funcionen bajo otros sistemas operativos.
- Para entrenar sistemas neuronales es imprescindible disponer de unidades GPU (*Graphical Processing Unit*), ya que sin estas unidades el entrenamiento resultaría larguísimo y a la práctica no se podría llevar a cabo. En cambio, para entrenar sistemas estadísticos no son necesarias estas unidades, pero conviene disponer de bastante memoria RAM en el sistema.

4.2. Entrenamiento de sistemas estadísticos con Moses

Antes de iniciar el entrenamiento, es necesario tener un corpus paralelo preprocesado convenientemente. Los pasos de preprocesamiento del corpus para sistemas estadísticos habitualmente consiste en:

- Tokenización
- Entrenamiento del truecaser
- Truecasing del corpus
- Sustitución de expresiones numéricas, URLs y emails por códigos
- División del corpus en entrenamiento, optimización y evaluación. Los corpus de optimización y evaluación acostumbran a tener un tamaño de entre 1000 y 5000 segmentos.

El entrenamiento de un sistema estadístico con Moses suele dividirse en tres etapas:

- Entrenamiento
- Optimización
- Binarización

Y se suele complementar con una etapa de evaluación, que explicaremos más adelante, ya que se realiza de la misma manera para los sistemas estadísticos y neuronales.

4.2.a. Entrenamiento

Durante el entrenamiento se calculan dos modelos: el modelo de lengua de la lengua de llegada y el modelo de traducción (que en realidad estará compuesto por una tabla de traducción y un modelo de reordenamiento).

Como ya vimos en la sección 2.3, el modelo de lengua es una estadística de apariciones de palabras y n-gramas (combinaciones de palabras) de un determinado corpus.

Una tabla de traducción es un archivo que contiene n-gramas en la lengua de partida con posibles n-gramas que pueden ser su traducción a la lengua de llegada, junto a una serie de cifras que indican diferentes probabilidades. A continuación podemos observar un fragmento de una tabla de traducción:

...

```
any international agreement that has already been ||| ningún acuerdo internacional
que se haya ||| 1 1.5782e-06 0.5 9.1287e-05 ||| 0-0 2-1 1-2 3-3 5-4 4-5 6-5 ||| 1 2
1 ||| |||
any international agreement that ||| ningún acuerdo internacional que ||| 1 0.0866426
1 0.018654 ||| 0-0 2-1 1-2 3-3 ||| 1 1 1 ||| |||
any international agreement ||| ningún acuerdo internacional ||| 1 0.308584 1 0.021812
||| 0-0 2-1 1-2 ||| 1 1 1 ||| |||
by international agreement ; ||| por acuerdo internacional ; ||| 1 0.234749 1 0.265537
||| 0-0 2-1 1-2 3-3 ||| 2 2 2 ||| |||
by international agreement ||| por acuerdo internacional ||| 1 0.236314 1 0.266187
||| 0-0 2-1 1-2 ||| 2 2 2 ||| |||
…
```

El modelo de reordenamiento también es archivo que contiene n-gramas en las dos lenguas y que intenta modelar los cambios de posición de las palabras. Podemos ver un ejemplo a continuación:

```
…
any international agreement that has already been ||| ningún acuerdo internacional
que se haya celebrado ||| 0.6 0.2 0.2 0.6 0.2 0.2
any international agreement that has already been ||| ningún acuerdo internacional
que se haya ||| 0.6 0.2 0.2 0.2 0.2 0.6
…
```

Cuando finaliza el entrenamiento también se crea un archivo de configuración que contiene una serie de pesos para cada uno de los modelos y parámetros adicionales. Estos pesos, contiene una serie de valores que se han fijado a unos valores predeterminados. Por ejemplo:

```
UnknownWordPenalty0= 1
WordPenalty0= -1
PhrasePenalty0= 0.2
TranslationModel0= 0.2 0.2 0.2 0.2
LexicalReordering0= 0.3 0.3 0.3 0.3 0.3 0.3
Distortion0= 0.3
LM0= 0.5
```

En el siguiente paso, denominado optimización, se ajustarán estos valores.

4.2.b.Optimización

En este paso se ajustan los parámetros del archivo de configuración. Para ello el programa traduce la parte correspondiente a la lengua de partida del corpus de optimización y compara la traducción con la parte correspondiente a la lengua de llegada. En cada iteración va realizando cambios a los parámetros del archivo de configuración hasta obtener unos parámetros óptimos (de allí el nombre de optimización de este proceso y del corpus de optimización). Una vez finalizado este proceso, los pesos de los diferentes parámetros han cambiado, como podemos observar a continuación:

```
LexicalReordering0= 0.0911963 0.0424064 0.101578 0.0499331 0.0642619 0.0828595
Distortion0= 0.0214228
LM0= 0.122748
WordPenalty0= -0.154592
PhrasePenalty0= 0.0686402
TranslationModel0= 0.103175 0.00431867 0.0725423 0.0203253
UnknownWordPenalty0= 1
```

4.2.d. Binarización

Una vez realizados los pasos anteriores ya podríamos traducir con Moses o poner en marcha un servidor de Moses, pero los archivos de los modelos de lengua y traducción son archivos de muy gran tamaño guardados en un formato texto. Esto supone que la carga de estos archivos cada vez que ponemos en marcha el sistema sea muy lenta. Por este motivo, los archivos que contienen los modelos normalmente se binarizan, es decir, se convierten en un formato binario que resulta en una carga de los modelos mucho más rápida.

4.3. Entrenamiento de sistemas neuronales con Marian

En este apartado vamos a ver cómo se entrena un sistema neuronal con Marian. Para poder iniciar el entrenamiento necesitaremos disponer del corpus preprocesado convenientemente, de marian compilado para nuestro ordenador y de una o más unidades GPU potentes.

El preprocesamiento del corpus debe incluir el uso de subpalabras (ya sea con subword-nmt, SentencePiece u otros algoritmos disponibles). Los pasos de tokenización y truecasing son opcionales si se utiliza SentencePiece, pero sí que se suele hacer el tratamiento específico de emails y URLs. El tratamiento de expresiones numéricas se suele llevar a cabo separando las cifras por espacios en blanco.

4.3.a. Entrenamiento básico

Si nuestro corpus de entrenamiento se llama train.sp.en y train.sp.es, un entrenamiento muy básico se puede llevar a cabo mediante la siguiente instrucción:

```
./marian --train-sets train.sp.en train.sp.es --vocabs vocab-en.yml vocab-es.yml --
model model.npz
```

El entrenamiento generará los vocabularios vocab-en.yml y vocab-es.yml y un model de traducción neuronal (model.npz). Una vez entrenado, este modelo se puede binarizar para conseguir una carga más rápida.

4.3.b. Validación y early stopping

Durante el preprocesamiento, el corpus se ha dividido en un fragmento de entrenamiento (train), uno de validación (val) y uno de evaluación (eval). Durante el entrenamiento, cada ciertas iteraciones, se puede llevar a cabo un proceso de validación. Este proceso consiste en traducir la parte correspondiente a la lengua de partida del corpus de validación y calcular una o más medidas de validación comparando esta traducción con la parte correspondiente a la lengua de llegada del corpus de validación. Este cálculo permite establecer un criterio de early stopping, es decir, indicar que el entrenamiento se detenga en el momento que las medidas de validación no mejoren durante un número concreto de iteraciones. Esto se puede indicar de la siguiente manera:

```
./marian --train-sets train.sp.en train.sp.es --valid-sets val.sp.en val.sp.es --vocabs vocab-en.yml vocab-es.yml --valid-metrics cross-entropy bleu-detok --model model.npz
```

Si nos fijamos, hemos indicado que queremos validar mediante dos medidas: cross-entropy y bleu-detok. Ahora también podremos indicar cuántas iteraciones sin mejoras en las medidas tienen que pasar para detener el entrenamiento, que nos guarde un archivo de validación con las indicaciones de las medidas obtenidas en las diferentes iteraciones, etc. Además podemos indicar muchísimas más opciones, que hacen que la instrucción final que tenemos que escribir sea muy larga y que resulte muy fácil equivocarse. Veremos a continuación que es mucho más práctico y sencillo utilizar archivos de configuración para llevar a cabo nuestros entrenamientos.

4.3.c. Entrenamiento mediante archivos de configuración

En lugar de indicar todas las opciones en el comando de entrenamiento podemos utilizar un archivo de configuración de tipo yaml, que especifique las opciones. Por ejemplo, en lugar del entrenamiento básico que realizamos con el comando:

```
./marian --train-sets train.sp.en train.sp.es --vocabs vocab-en.yml vocab-es.yml --model model.npz
```

Podemos tener un archivo que se llame por ejemplo conf-train-basic.yml y que contenga:

```
train-sets:
  - train.sp.en
  - train.sp.es
vocabs:
  - vocab-en.yml
  - vocab-es.yml
model: model.npz
```

Y entrenar el sistema escriiendo:

```
./marian -c conf-train-basic.yml
```

Para un caso tan sencillo quizás no se vea la utilidad real, pero hay que pensar que en una situación real se especifican decenas de parámetros. Algunos de estos parámetros no los variaremos habitualmente, pero otros, como el nombre de los archivos que contienen los corpus, los códigos de lengua, etc. se variarán a menudo. Por este motivo, en los archivos de configuración se acostumbran a poner las opciones que más se cambian en la parte superior. A continuación podemos ver un ejemplo de las primeras líneas de un archivo de configuración real.

```
train-sets:
  - train.sp.en
  - train.sp.es
valid-sets:
  - val.sp.en
  - val.sp.es
vocabs:
  - vocab-en.yml
```

```
  - vocab-es.yml
guided-alignment: train.sp.en.es.align
valid-metrics:
  - cross-entropy
  - bleu-detok
early-stopping: 5
early-stopping-on: any
keep-best: true
overwrite: true
valid-log: valid.log
...
```

En este archivo indicamos que queremos hacer *guided alignment* (concepto que explicamos en el apartado siguiente), las métricas de evaluación, que hacemos un *early stopping* de 5 con cualquiera de las métricas (es decir la primera métrica que no mejore en cinco iteraciones detendrá el entrenamiento), que nos quedaremos con el mejor modelo y que se borrarán todos los modelos intermedios. También indicamos que nos genere un archivo de log de la validación, que contendrá información sobre los valores obtenidos por cada métrica en cada paso de validación. Podemos observar a continuación fragmentos del contenido de un archivo de validación de ejemplo.

```
[2022-09-23 00:35:28] [valid] Ep. 1 : Up. 10000 : cross-entropy : 84.7741 : new best
…
[2022-09-23 00:38:01] [valid] Ep. 1 : Up. 10000 : bleu-detok : 36.6797 : new best
[2022-09-23 01:32:09] [valid] Ep. 1 : Up. 20000 : cross-entropy : 65.9883 : new best
[2022-09-23 01:34:38] [valid] Ep. 1 : Up. 20000 : bleu-detok : 43.9389 : new best
[2022-09-23 02:28:59] [valid] Ep. 1 : Up. 30000 : cross-entropy : 60.8309 : new best
[2022-09-23 02:31:06] [valid] Ep. 1 : Up. 30000 : bleu-detok : 44.8641 : new best
…
[2022-09-23 21:09:31] [valid] Ep. 2 : Up. 230000 : cross-entropy : 37.1511 : new best
[2022-09-23 21:11:30] [valid] Ep. 2 : Up. 230000 : bleu-detok : 52.6491 : stalled 4 times (last
best: 52.7422)
[2022-09-23 22:03:43] [valid] Ep. 2 : Up. 240000 : cross-entropy : 37.5659 : stalled 1 times
(last best: 37.1511)
[2022-09-23 22:05:42] [valid] Ep. 2 : Up. 240000 : bleu-detok : 52.1931 : stalled 5 times (last
best: 52.7422)
```

Esta información nos permite generar gráficos que nos muestran la evolución de las medidas de validación, como por ejemplo el de la figura 1, que muestra la evolución del BLEU-DETOK en este entrenamiento:

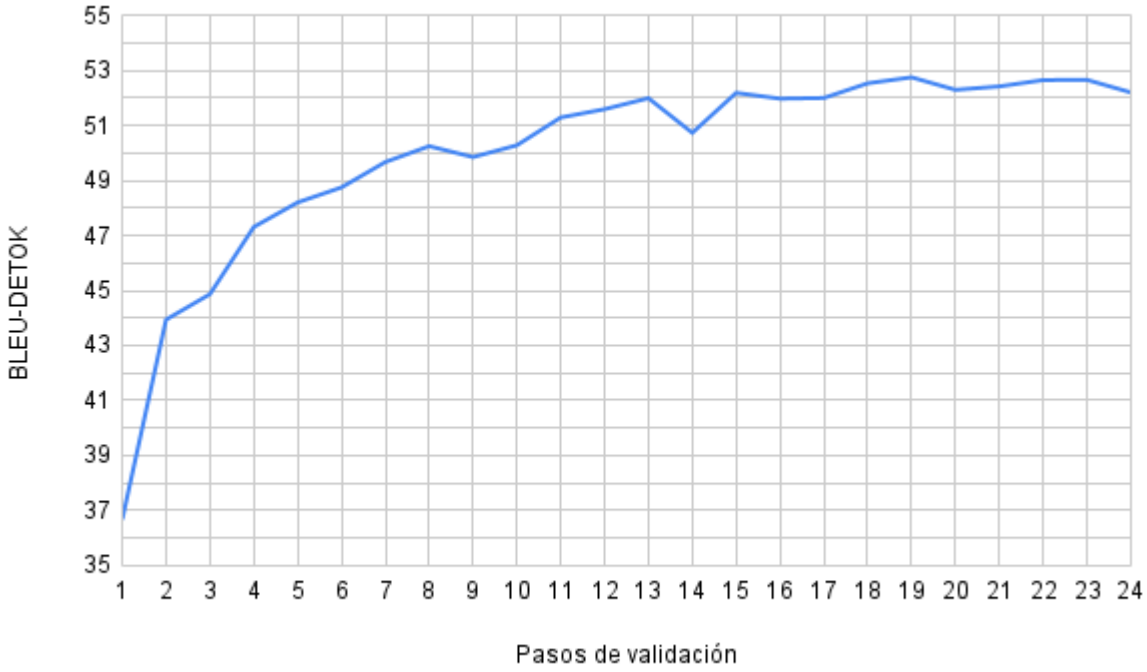

Figura 1. Evolución de la métrica BLEU-DETOK en los diferentes pasos de validación.

Como se puede observar, el valor de BLEU-DETOK aumenta rápidamente durante las primeras validaciones, pero se va estabilizando en el tiempo. El proceso de entrenamiento se ha detenido al llegar a las 5 validaciones sin obtener mejora, y el modelo final es el correspondiente a la validación con el valor más alto. En el gráfico de la figura 1 sería el modelo correspondiente al paso de validación 19, que consigue un valor de BLEU-DETOK de 52.74.

4.4. Guided alignment

De un sistema de traducción automática nos interesa que nos devuelva una buena traducción del segmento original. Para muchas tareas adicionales también nos va a interesar que nos devuelva la alineación a nivel de palabra, es decir, la relación entre las palabras del segmento original y las del segmento traducido. Por ejemplo, si enviamos la siguiente oración a un motor de traducción automática:

```
This is a simple example.
```

Que en realidad enviaremos tokenizada y posiblemente con unas etiquetas de inicio y final de oración:

```
<s> This is a simple example . </s>
```

El sistema de traducción automática nos devolverá la traducción tokenizada y con las etiquetas de inicio y final de oración.

```
<s> Este es un ejemplo sencillo . </s>
```

Que una vez detokenizada y eliminando las marcas de principio y final de oración nos quedará la traducción:

```
Este es un ejemplo sencillo.
```

Los sistemas de traducción automática también son capaces de devolvernos la alineación a nivel de palabra (de token en realidad y de la versión con etiquetas de inicio y final de oración) que tendría un aspecto como el siguiente:

```
0-0 1-1 2-2 3-3 4-5 5-4 6-6 7-7
```

Si nos fijamos en la tabla 2, vemos que esta relación es correcta:

Alineación	Token SL	Token TL
0-0	<s>	<s>
1-1	This	Este
2-2	is	es
3-3	a	un
4-5	simple	sencillo
5-4	example	ejemplo
6-6	.	.
7-7	</s>	</s>

Tabla 2. Alineación a nivel de palabra entre el original inglés y la traducción automática al español.

Esta alineación puede ser útil para diversas tareas de postprocesado de la traducción. Una tarea habitual es la recuperación automática de etiquetas XML o HTML del original. Por ejemplo, si nuestro segmento original contiene etiquetas, como por ejemplo:

```
This is a <b>simple</b> example.
```

Habitualmente al sistema de traducción automática se le envía el segmento a traducir sin etiquetas y un algoritmo de postprocesado puede determinar que las etiquetas están delante y detrás del token número 4 del original (recordad que frecuentemente se trabaja internamente con las marcas de inicio y final de oración), y que mirando la alineación, estas etiquetas tienen que estar delante y detrás del token 5 de la traducción (observa la tabla 2), resultando en:

```
Este es un ejemplo <b>sencillo</b>.
```

Los sistemas estadísticos y los neuronales basados en RNN (*Recurrent Neural Networks*) son capaces de generar alineaciones con precisión. En cambio, los sistemas neuronales basados en *Transformers* no

pueden generar las alineaciones con precisión si no entrenamos alguna de sus capas específicamente para esta tarea. Este entrenamiento específico se denomina *guided alignment*.

Para entrenar un sistema con *guided alignment* necesitamos proporcionar las alineaciones a nivel de palabra del corpus de entrenamiento y, opcionalmente, del corpus de validación. Por ejemplo, tenemos un corpus de entrenamiento inglés - español (fijaos que el corpus está procesado con subpalabras utilizando SentencePiece, y que por lo tanto las alineaciones serán en realidad a nivel de subpalabra..

```
<s> ▁Order ly ▁international ▁migration ▁can ▁have ▁positive ▁impacts ▁on ▁both ▁the ▁communities ▁of ▁origin ▁and ▁the ▁communities ▁of ▁destination , ▁providing ▁the ▁former ▁with ▁remittances ▁and ▁the ▁latter ▁with ▁needed ▁human ▁resources . </s>
<s> ▁ 4 7 . ▁The ▁unutilized ▁balance ▁was ▁attributable ▁to ▁lower ▁requirements ▁in ▁the ▁hazardous ▁duty ▁station ▁allowance ▁as ▁a ▁result ▁of ▁the ▁suspension ▁of ▁the ▁allowance ▁for ▁staff ▁deployed ▁or ▁travelling ▁to ▁Abidjan , ▁Yamoussoukro , ▁Daloa , ▁Bouaké ▁and ▁San ▁Pedro , ▁effective ▁from ▁ 1 6 ▁May ▁ 2 0 0 7 . </s>
<s> ▁Paragraph ▁ 1 2 . 8 </s>
…
<s> ▁Una ▁migración ▁internacional ▁ordenada ▁puede ▁tener ▁repercusiones ▁positivas ▁tanto ▁para ▁las ▁comunidades ▁de ▁origen ▁como ▁para ▁las ▁de ▁destino , ▁al ▁constituir ▁para ▁las ▁primeras ▁una ▁fuente ▁de ▁remesas ▁y ▁proporcionar ▁a ▁las ▁segundas ▁los ▁recursos ▁humanos ▁necesarios . </s>
<s> ▁El ▁saldo ▁no ▁utilizado ▁obedeció ▁al ▁descenso ▁de ▁las ▁necesidades ▁para ▁sufragar ▁prestaciones ▁por ▁lugar ▁de ▁destino ▁peligroso ▁ya ▁que , ▁a ▁partir ▁del ▁ 1 6 ▁de ▁mayo ▁de ▁ 2 0 0 7 , ▁se ▁suspendió ▁esa ▁prestación ▁para ▁el ▁personal ▁desplegado ▁en ▁Abidján , ▁Yamoussoukro , ▁Daloa , ▁Bouaké ▁y ▁San ▁Pedro ▁o ▁que ▁viaj ara ▁a ▁esas ▁localidades . </s>
<s> ▁Párrafo ▁ 1 2 . 8 </s>
…
```

Tendremos también que proporcionar las alineaciones a nivel de palabra para todo el corpus:

```
0-0 2-1 4-2 3-3 2-4 5-5 6-6 8-7 7-8 10-9 9-10 11-11 12-12 13-13 14-14 15-15 16-16 17-17 18-18 19-19 20-20 21-22 22-24 23-25 24-26 25-29 26-30 21-31 27-33 28-34 31-35 32-36 31-37 30-38 33-39 34-40
0-0 5-1 6-2 6-3 7-4 9-5 10-6 11-7 12-10 13-11 14-12 18-13 16-15 16-17 15-18 19-19 20-20 44-21 45-23 46-24 47-25 48-26 49-27 50-29 51-31 52-32 53-33 54-34 55-35 22-36 23-37 24-38 26-39 27-40 28-41 29-43 30-44 34-46 35-47 36-48 37-49 38-50 39-51 40-52 41-53 42-54 43-55 31-56 32-58 33-60 34-62 56-63 57-64
0-0 1-1 2-2 3-3 4-4 5-5 6-6 7-7
```

Estas alineaciones del corpus se pueden generar automáticamente utilizando diversos algoritmos:

- fast-align[15] (Dyer, Chahuneau & Smith, 2013)
- eflomal[16] (Robert Östling and Jörg Tiedemann, 2016)
- Simalign[17] (Sabet et al., 2020)
- Awesome[18] (Dou & Neubig, 2021)

El cálculo de las alineaciones a nivel de palabra del corpus de entrenamiento y opcionalmente el de validación se realiza como un paso más del preprocesamiento del corpus.

[15] https://github.com/clab/fast_align
[16] https://github.com/robertostling/eflomal
[17] https://github.com/cisnlp/simalign
[18] https://github.com/neulab/awesome-align

5. EVALUACIÓN DE SISTEMAS DE TRADUCCIÓN AUTOMÁTICA MEDIANTE MÉTRICAS AUTOMÁTICAS

Una vez ya hemos entrenado un sistema de traducción automática nos interesa evaluarlo para saber si es mejor que algún otro sistema alternativo para el mismo par de lenguas, o si ha mejorado respecto a entrenamientos anteriores. Esta evaluación conviene realizarla de manera rápida y sin dedicarle esfuerzo humano excesivo. Para conseguir este objetivos existen un gran número de métricas automáticas de evaluación de sistemas de traducción automática.

Todas estas métricas se basan en disponer de una serie de segmentos en la lengua de partida con su traducción (o diversas traducciones diferentes) a la lengua de llegada. Si recordáis, uno de los pasos de preprocesamiento de corpus era la división del corpus de entrenamiento en tres fragmentos: entrenamiento, validación y evaluación. Precisamente el corpus de evaluación será el utilizado par evaluar el sistema. Los segmentos en la lengua de partida del corpus de evaluación se traducen con el sistema que queremos evaluar y las traducciones automáticas obtenidas (que denominaremos hipótesis) se compararan con las traducciones del corpus de evaluación (que denominaremos referencias). Mediante esta comparación se obteniena alguna de las métricas. Alguna de las métricas más utilizadas son: BLEU (Papineni et al., 2002), NIST (Doddington, 2002), WER (*Word Error Rate*) (Nießen et al., 2000), % de distancia de edición y TER (*Translation Error Rate*) (Klakow and Peters, 2002).

6. INTEGRACIÓN DE TRADUCCIÓN AUTOMÁTICA EN ENTORNOS PROFESIONALES

Una vez hemos entrenado nuestro sistema de traducción automática ya podremos utilizarlo con los programas que ofrecen los toolkits. Será posible, por ejemplo, traducir una sola oración desde nuestro terminal. Si hemos entrenado un sistema estadístico con Moses podríamos escribir:

```
echo "this is a simple example ." | ./moses -f moses.ini
```

Y el sistema respondería con una serie de mensajes y con la traducción:

```
BEST TRANSLATION: este es un sencillo ejemplo . [111111]  [total=-5.531]
core=(0.000,-6.000,5.000,-2.159,-4.385,-3.135,-5.113,-1.311,0.000,0.000,-
1.979,0.000,0.000,0.000,-48.961)
```

Lo mismo podríamos hacer con un sistema neuronal entrenado con Marian:

```
echo "<s> ▁This ▁is ▁a ▁simple ▁example . </s>" | ./marian-decoder -m model.bin -v
vocab-en.yml vocab-es.yml
```

Que nos respondería con:

```
[2022-10-13 12:37:53] Best translation 0 : <s> ▁Este ▁es ▁un ▁ejemplo ▁sencillo .
</s>
<s> ▁Este ▁es ▁un ▁ejemplo ▁sencillo . </s>
[2022-10-13 12:37:53] Total time: 0.03278s wall
```

Si nos fijamos, para utilizar los sitemas entrenados hemos tenido que introducir los segmentos a traducir preprocesados. En el caso de Moses, el segmento está tokenizado y truecased; y en el caso de Marian, con SentencePiece. Además los sistemas nos devuelven las traducciones también preprocesadas de igual manera que los originales, por lo que sería necesario un paso posterior para de-preprocesar los segmentos.

Como que no resulta práctico traducir segmento a segmento en el terminal, es posible poner en marcha un servidor, que pueda responder a las peticiones de traducción de un programa cliente. En Moses podemos poner en marcha un servidor escribiendo:

```
./moses -f moses.ini --server
```

Que nos responde con el mensaje:

```
[moses/server/Server.cpp:49] Listening on port 8080
```

En Marian podemos hacer lo mismo escribiendo:

```
./marian-server-GPU -m model.bin -v vocab-en.yml vocab-es.yml
```

Una vez cargados los modelos aparecerá en pantalla:

```
[2022-10-13 12:46:35] Server is listening on port 8080
```

Pero aunque pongamos en marcha un servidor, los segmentos que envíe el programa cliente tendrán que estar preprocesados y las traducciones que reciba tendrán que ser tratadas para de-preprocesarlas.

Para solventar este problema se suele recurrir a un programa servidor intermedio que puede realizar las siguientes funciones:

- Poner en marcha el servidor real de traducción (por ejemplo un Marian)
- Recibir del programa cliente (que puede ser, por ejemplo, una herramienta de traducción asistida por ordenador) el segmento a traducir, por ejemplo "This is a simple example." El servidor intermedio es capaz de comunicarse con el cliente mediante diversos protocolos, de manera que es complatible con muchos más programas clientes.
- Preprocesar el segmeno a traducir de la manera que necesite el servidor real de traducción que se utilizará. En nuestro ejemplo, aplicar SentencePiece, quedando el segmento de la siguiente manera: <s> ▁This ▁is ▁a ▁simple ▁example . </s>
- Enviar el segmento a traducir al servidor real.
- Recibir la traducción, que estará preprocesada también, en este ejemplo con SentencePiece: <s> ▁Este ▁es ▁un ▁ejemplo ▁sencillo . </s>
- De-prepocesar el segmento traducido para que quede una oración en la lengua de llegada, en nuestro ejemplo: "Este es un ejemplo sencillo."
- Enviar la traducción al programa cliente utilizando el protocolo requerido.

Este funcionamiento se puede observar en la figura 2.

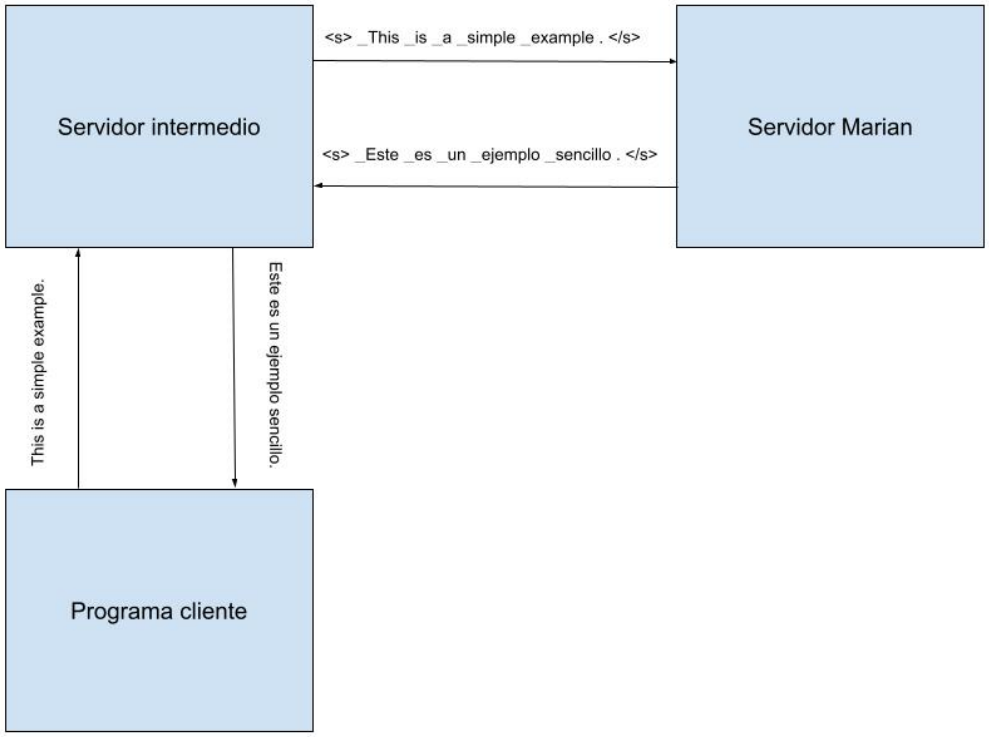

Figura 2. Funcionamiento de un servidor intermedio de traducción automática.

Siguiendo el esquema de la figura 2, los sistemas estadísticos y neuronales que entrenemos podrán conectarse con diversas herramientas de traducción asistida, como por ejemplo, Trados Studio u OmegaT.

7. CONCLUSIONES

En este capítulo hemos presentado el procedimiento básico para poder entrenar nuestros sistemas de traducción automática tanto estadísticos como neuronales. Hemos visto que hay corpus paralelos disponibles para muchos pares de lenguas y que podemos crear nuestros propios corpus paralelos a partir de memorias de traducción o bien de textos en una determinada lengua con sus correspondientes traducciones a otras lenguas.

La calidad de los sistemas de traducción automática ha mejorado mucho durante los últimos años, especialmente desde la aparición de la traducción automática neuronal. Pero aún queda mucho camino por recorrer y quedan diversos retos por resolver y que son objeto de investigación:

- Pares de lenguas con pocos recursos: para entrenar sistemas de traducción automática se necesitan diversos millones de segmentos paralelos. Muchos pares de lenguas no disponen de

estos recursos, por lo que se están desarrollando diversas técnicas para poder entrenar sistemas sin disponer de corpus paralelos de gran tamaño. Algunas de las técnicas se basan en aprovechar el conocimiento que se pueda aprender a partir de corpus paralelos de gran tamaños en pares de lenguas relacionadas. También se está investigando en el entrenamiento de sistemas a partir de corpus monolingües de las dos lenguas implicadas, sin la necesidad de disponer de corpus paralelos.

- Adaptación a dominio. Incluso para pares de lenguas con corpus paralelos de gran tamaño, estos corpus pueden no estar disponibles para el dominio que necesitemos. Por ejemplo, queremos entrenar un sistema de traducción automática inglés-español para textos de aeronáutica, de los dispongo de pocos pares de segmentos paralelos. ¿Puedo adaptar un sistema de traducción automática inglés-español de ámbito general a un ámbito especializado?
- Uso de bases de datos terminológicas. A menudo disponemos de una base de datos terminológica y queremos que el resultado de la traducción automática respete los equivalentes de traducción de esta base de datos, cuando quizás los términos en cuestión no aparecían en el corpus de entrenamiento o bien los equivalentes de traducción utilizados en el corpus son diferentes de los de la base de datos terminológica. El uso de esta información terminológica plantea diversas dificultades y se han sugerido diversas soluciones, no siempre totalmente eficientes, por lo que este aspecto es también objeto de investigación.

Durante los próximos años la investigación en traducción automática continuará siendo muy intensa y se esperan mejoras en la calidad de los sistemas resultantes. Pero esto no debe preocupar al traductor profesional, ya que para conseguir unos niveles de calidad profesional, la posedición humana continuará siendo imprescindible durante muchos años.

Bibliografía

Artetxe, M., Labaka, G., & Agirre, E. (2019). An Effective Approach to Unsupervised Machine Translation. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (pp. 194-203).

Bertoldi, N., Caroselli, D., & Federico, M. (2018). The ModernMT Project. In *Proceedings of the 21st Annual Conference of the European Association for Machine Translation*, p. 345

Doddington, G. 2002. Automatic Evaluation of Machine Translation Using n-gram Cooccurrence Statistics. *Proceedings of the Second International Conference on Human Language Technology Research.* pp. 138-145

Dou, Z.Y., & Neubig, G. (2021). Word Alignment by Fine-tuning Embeddings on Parallel Corpora. In *Conference of the European Chapter of the Association for Computational Linguistics* (EACL).

Dyer, C., Chahuneau, V., & Smith, N. A. (2013). A simple, fast, and effective reparameterization of ibm model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 644-648).

Forcada, M. L. (2017). Making sense of neural machine translation. *Translation spaces*, *6*(2), 291-309. Preprint disponible en https://www.dlsi.ua.es/~mlf/docum/forcada17j2.pdf

Junczys-Dowmunt, M., Grundkiewicz, R., Dwojak, T., Hoang, H., Heafield, K., Neckermann, T., Seide, F., Germann, U., Fikri Aji, A., Bogoychev, N., Martins, A., & Birch, A. (2018). Marian: Fast Neural Machine

Translation in C++. In *Proceedings of ACL 2018, System Demonstrations* (pp. 116–121). Association for Computational Linguistics.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, & Alexander M. Rush (2017). OpenNMT: Open-Source Toolkit for Neural Machine Translation. In *Proceedings of ACL 2017, System Demonstrations, pages 67–72, Vancouver, Canada. Association for Computational Linguistics*.

Klakow, D. & Peters J. 2002. Testing the correlation of word error rate and perplexity. *Speech Communication*, 38 (1-2). pp. 19-28.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., & others (2007). Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the association for computational linguistics companion volume proceedings of the demo and poster sessions* (pp. 177–180).

Kudo, T., & Richardson, J. (2018). SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 66–71, Brussels, Belgium. Association for Computational Linguistics.

Nießen, S., Och, F.J., Leusch, G. & Ney, G. (2000). An evaluation tool for machine translation: fast evaluation for MT research. In *Proceedings of the Second International Conference on Language Resources and Evaluation* (LREC'2000), Athens, Greece. European Language Resources Association (ELRA).

Östling R. & Tiedemann, J.. (2016). Efficient word alignment with Markov Chain Monte Carlo. *The Prague Bulletin of Mathematical Linguistics*, 106(1).

Papineni, K., Roukos, S., Ward, T. & Zhu, W. 2002. BLEU: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pp. 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Maja Popović. 2015. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal. Association for Computational Linguistics.

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. https://arxiv.org/abs/1908.10084

Jalili Sabet, M., Dufter, P., Yvon, F., & Schutze, H. (2020). SimAlign: High Quality Word Alignments without Parallel Training Data using Static and Contextualized Embeddings. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings* (pp. 1627–1643). Association for Computational Linguistics.

Sennrich, R., Haddow, B., & Birch, A. (2015). Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Schwenk, H., Wenzek, G., Edunov, S., Grave, É., Joulin, A., & Fan, A. (2021, August). CCMatrix: Mining Billions of High-Quality Parallel Sentences on the Web. In *Proceedings of the 59th Annual Meeting of the*

Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pp. 6490-6500.

Matthew Snover, Bonnie Dorr, Rich Schwartz, Linnea Micciulla, and John Makhoul. 2006. A Study of Translation Edit Rate with Targeted Human Annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*, pages 223–231, Cambridge, Massachusetts, USA. Association for Machine Translation in the Americas.

Tiedemann, J. (2012, May). Parallel data, tools and interfaces in OPUS. In *Proceedings of the 8th International Conference on Language Resources and Evaluation* (LREC'2012) pp. 2214-2218

D. Varga, L. Németh, P. Halácsy, A. Kornai, V. Trón, V. Nagy (2005). Parallel corpora for medium density languages In *Proceedings of the RANLP 2005*, pages 590-596.