La traducción automática neuronal

1. Introducción

En los últimos años, la calidad alcanzada por los sistemas de traducción automática, especialmente desde la aparición de la traducción automática neuronal, ha mejorado espectacularmente y no solo para pares de idiomas cercanos (como el español-catalán, por ejemplo, que ya lograba buenos resultados con la traducción automática basada en reglas), ni para pares de idiomas relativamente similares (como el español-inglés, por ejemplo, ya alcanzaba niveles de calidad significativos con la traducción estadística), si no incluso para pares de lenguas muy alejados (como el español-chino, por ejemplo). Tanto es la ventaja competitiva de los sistemas de traducción automática neuronales, que la práctica totalidad de nuevos sistemas se desarrollan bajo este paradigma.

La mejora de la calidad de los sistemas de traducción automática neuronal es tan notable que incluso algunos investigadores se han atrevido a reclamar la paridad en calidad con los traductores humanos (Hassan et al., 2018), a pesar de que enseguida muchos investigadores (Toral et al., 2018; Läubli et al., 2020) han refutado o matizado esta idea.

Lo que sí que es cierto es que los sistemas de traducción automática neuronal ofrecen una traducción con una gran fluidez y que es correcta en la mayoría de los casos, pero que todavía esconde, de vez en cuando, errores muy importantes y difíciles de detectar. Por este motivo es muy importante que los traductores conozcan muy a fondo esta tecnología para poder decidir en qué situaciones es recomendable usarla y como poder afrontar una tarea de posedición con éxito.

En este punto es importante recordar los dos grandes grupos en que se pueden clasificar los sistemas de traducción automática.

- **Sistemas basados en reglas**. Son sistemas que se desarrollan, es decir, un equipo de programadores y lingüistas escriben programas, reglas y diccionarios que son capaces de traducir oraciones en una determinada lengua de partida a una determinada lengua de llegada. A este grupo pertenecen los sistemas de traducción directa y los sistemas de transferencia sintáctica.
- **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 estas oraciones no estén en el corpus de entrenamiento. A este grupo pertenecen los sistemas estadísticos y los sistemas neuronales.

También es interesante recordar en esta introducción la línea temporal de los grandes hitos en la historia de la traducción automática neuronal. John Hutchins (1995) nos proporciona un detallado recorrido sobre la historia de la traducción, que merece la pena recordar en esta introducción, pero que hay que completar con los acontecimientos de los años posteriores. Según Hutchins, la historia de la traducción automática se puede dividir en las siguientes épocas:

- **Precursores y pioneros (1933-1958)**, cuando aparecieron las primeras patentes relacionadas con la traducción mecánica, donde se pueden destacar las propuestas de George Artsrouni y Petr Smirnov-Troyanskii. Durante esta época también se celebró la primera conferencia que tenía como tema central la traducción automática. El concepto de traducción automática también se le puede atribuir a Weaver (1949), que además de definir el concepto general incluso propuso diversos métodos para llevarla a cabo. La primera demostración pública de un sistema de traducción automática tuvo lugar el enero de 1954. Se trataba de un sistema muy limitado que era capaz de traducir 49 oraciones del ruso al inglés usando un diccionario de 250 palabras y un total de 6 reglas gramaticales.

- **La década de las grandes expectativas y desilusiones (1956-1966)**, cuando aparecieron los primeros sistemas de traducción automática, todavía muy limitados, capaces de traducir conjuntos reducidos de oraciones. A mediados de los años 60 había grupos de investigación dedicados a la traducción automática en los Estados Unidos, Unión Soviética, en varios países europeos (Hungría, Checoslovaquia, Bulgaria, Bélgica, Alemania, Francia, etc.), China, México y Japón.

- **El informe ALPAC y sus consecuencias (1966)**. Este informe del *Automatic Language Processing Advisory Committee*, publicado el noviembre del 1966 tuvo una gran influencia y su publicación, según Hutchins (1996) es el acontecimiento más conocido de la historia de la traducción automática. Este informe llegaba a la conclusión que la traducción automática era más lenta, menos precisa y el doble de cara que la traducción humana y que no había sistemas de traducción automática útiles en aquellos momentos ni se preveía que los hubieran en un futuro próximo. Por lo tanto, el informe recomendaba no invertir más en traducción automática. Pero el informe también recomendaba el desarrollo de herramientas de ayuda a la traducción, como por ejemplo diccionarios automáticos, y hacer una apuesta por la investigación básica en lingüística computacional.

- **La década silenciosa (1967-1976)**. El foco de la investigación en traducción automática cambió, a consecuencia del informe ALPAC, de los Estados Unidos a Europa y Canadá. Las necesidades de traducción de la Comunidad Económica Europea fue un incentivo para el desarrollo de sistemas en Europa. El 1970 en el Canadá se inició el proyecto TAUM (*Traduction Automatique del Université de Montréal*), que desarrolló el famoso sistema Météo para la traducción de previsiones meteorológicas del inglés al francés.

- **Sistemas operativos y comerciales (1976-1989)**. Durante esta década hubo una gran expansión en el número de sistemas comerciales disponibles y la investigación en traducción automática se diversificó en muchas direcciones. Entre los sistemas disponibles en aquella época se pueden destacar Systran, Logos y METAL. Hutchins (1995) hace un resumen interesante de la investigación que hubo en traducción automática desde 1976 hasta 1989, que no resumiremos en esta introducción.

- **La traducción automática basada en corpus: aparición de la traducción automática estadística (1988 - 2014)**. A partir del 1988 aparecen nuevos métodos basados en corpus, con los resultados publicados por IBM de los experimentos en un sistema basado puramente en métodos estadísticos. El hito más notable de este periodo es la aparición del sistema Moses[1] (Koehn et al., 2007), hacia el año 2005. Moses es un *toolkit* completo para el entrenamiento de sistemas de traducción automática estadística que se distribuye bajo una licencia libre. Este sistema ha sido la base de muchísimo sistemas estadísticos, tanto comerciales como de investigación. A pesar de que a causa de la predominancia actual de los sistemas neuronales este *toolkit* prácticamente ya no se actualiza, todavía es plenamente funcional. Durante este periodo también se investiga y se desarrollan sistemas usando otras metodologías. Hay que destacar en este periodo el desarrollo del sistema de transferencia sintáctica superficial Apertium[2] (Corbí-Bellot et al., 2005; Forcada et al., 2011), que se empieza a desarrollar también hacia el 2005.

- **La traducción automática neuronal (2014-)**. A pesar de que la idea de traducción automática neuronal era conocida desde hacía unos años, no es hasta aproximadamente el 2014 cuando aparecen los primeros sistemas funcionales (Cho et al.,2014b; Sutskever et al., 2014; Bahdanau et al., 2015). A partir de este momento aparecen un gran número de *toolkits* para el entrenamiento de sistemas neuronales y la traducción con los sistemas entrenados. Más adelante en este capítulo mencionaremos algunos de estos *toolkits*.

- **Grandes modelos de lenguaje (*Large Language Modelos, LLMs*)**: hacia el año 2022 aparecen los grandes modelos de lenguaje generativos, que usan también tecnología neuronal. Estos modelos son capaces de generar textos en varias lenguas y también de traducir textos. En el momento de escribir este capítulo todavía no está clara la influencia de estos modelos del lenguaje sobre la investigación y el desarrollo en traducción automática.

[1] http://www2.statmt.org/moses/
[2] https://apertium.org/

1930	1940	1950	1960	1970	1980	1990	2000	2010	2020	2022
Precursores y pioneros			Grandes expectativas	ALPAC	Década silenciosa	Sistemas operativos y comerciales		T.A. Estadística	T.A. Neuronal	LLMs

2. Redes neuronales

Una red neuronal es una técnica de aprendizaje automático que es capaces de tomar una serie de entradas y predecir una salida. Las redes neuronales están formadas por una serie de unidades interconexionadas entre ellas, llamadas neuronas artificiales. En esta sección veremos todos estos conceptos.

2.1. Neurona artificial

Las neuronas artificiales son los elementos principales de las redes neuronales y se pueden definir como una función matemática que recibe una o más entradas y que cada una de estas entradas tiene un valor de ponderación. La neurona tiene asociada una determinada función de activación que será la encargada de calcular el valor de salida. Para explicar todos estos conceptos, consideremos una red neuronal muy sencilla compuesta por 4 neuronas artificiales, como se muestra a la figura 1.

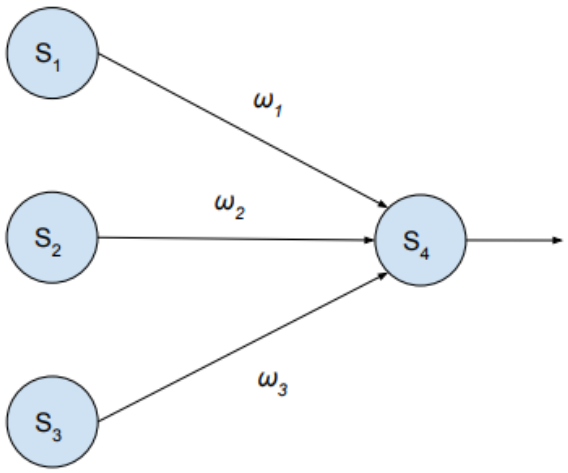

Figura 1. Red neuronal artificial

En esta red neuronal el valor de salida de la neurona artificial S_4 toma el siguiente valor:

$S_4 = F\ (\omega_1 \times \#S_1 + \omega_2 \times S_2 + \omega_3 \times S_3)$ [Fórmula 1]

Si nos fijamos en esta expresión el valor de la salida es una función de las salidas de las neuronas de entrada multiplicadas por unos pesos (ω_1 , ω_2 , ω_3). Es decir que el valor de la salida de la neurona S_4 será un valor dado por una función que recibe el siguiente estímulo x:

$x = \omega_1 \times S_1 + \omega_2 \times S_2 + \omega_3 \times S_3$ [Fórmula 2]

La salida, pues, de la neurona S4, vendrá dada por una determinada función, que recibe el nombre de función de activación. En redes neuronales se acostumbran a usar alguna de las funciones indicadas a la tabla 1 y mostradas a las figuras 2, 3 y 4. En el repositorio asociado con estos materiales[3] encontrarás unos programas en Python que implementan estas funciones y crean las figuras que m0stramos a continuación.

Función	**Fórmula**	**Rango**
Tangente hiperbólica	$tanh(x) = \frac{sin(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	de -1 a +1
Sigmoide	$sigmoid(x) = \frac{1}{1 + e^{-x}}$	de 0 a +1
Rectificador o ReLU	$relu(x) = max(0, x)$	de 0 a ∞

Tabla 1. Funciones de activación

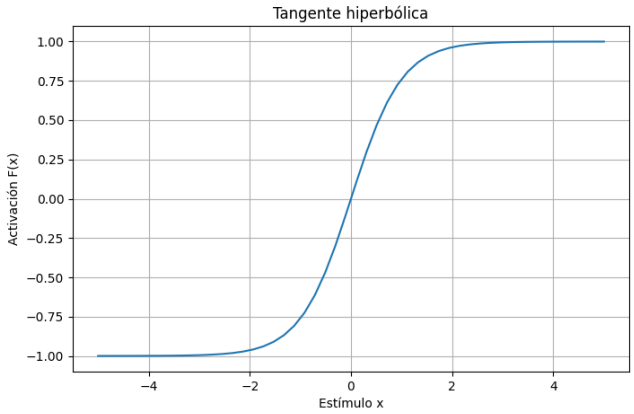

Figura 2. Función de activación tangente hiperbólica

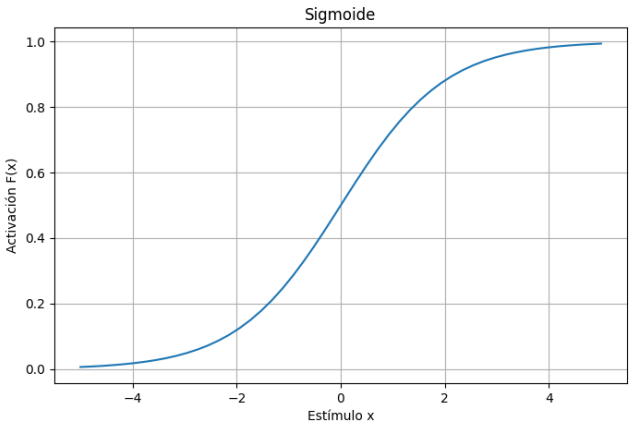

Figura 3. Función de activación sigmoide

[3] https://github.com/aoliverg/materiales-TAN

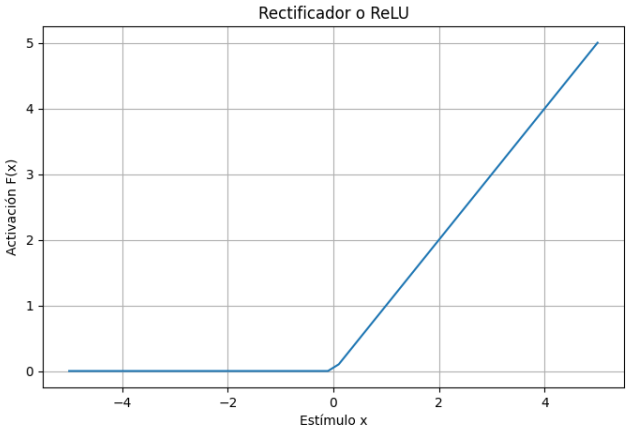

Figura 4. Función de activación ReLU

Una característica importante de estas funciones de activación es que son no lineales. La combinación de los valores ponderados expresada por la fórmula 2 es una función lineal, que pasa a ser una función no lineal aplicando la función de activación. El uso de funciones no lineales permite definir relaciones más complejas a partir de los rasgos de entrada.

En el repositorio asociado a estos materiales encontrarás una demo que entrena una neurona artificial para resolver un problema concreto.

2.2. Redes neuronales

Las neuronas artificiales presentadas a la sección anterior se pueden conectar para formar redes neuronales. Estas redes neuronales serán capaces de llevar a cabo una tarea computacional concreta, como podría ser solucionar un problema específico, como por ejemplo, traducir una oración de entrada en una lengua a una oración de salida en otra lengua.

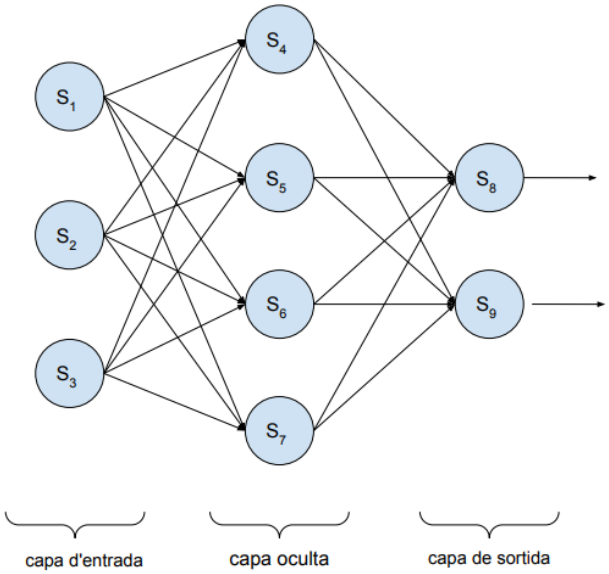

Figura 5. Red neuronal

En la figura 5 podemos observar una red neuronal formada por un total de 9 neuronas artificiales. Esta red neurona tiene tres capas: una capa de entrada, una capa oculta y una capa de salida. Las neuronas de la capa de entrada emiten directamente los valores de la entrada externa y no calculan ningún estímulo ni reaccionan al estímulo a través de la función de activación. La capa oculta recibe este nombre porque no tenemos acceso ni en al estado de las neuronas ni a sus salidas. Las neuronas de la capa de salida emiten los valores que serán accesibles en la salida de la red neuronal.

2.3. Aprendizaje profundo

Los sistemas de aprendizaje profundo (*deep learning* en inglés) son sistemas de aprendizaje automático formados por muchas capas. La mayoría de estos sistemas funcionan con redes neuronales. Estas redes neuronales de aprendizaje profundo, en vez de tener una única capa oculta, tienen dos o más.

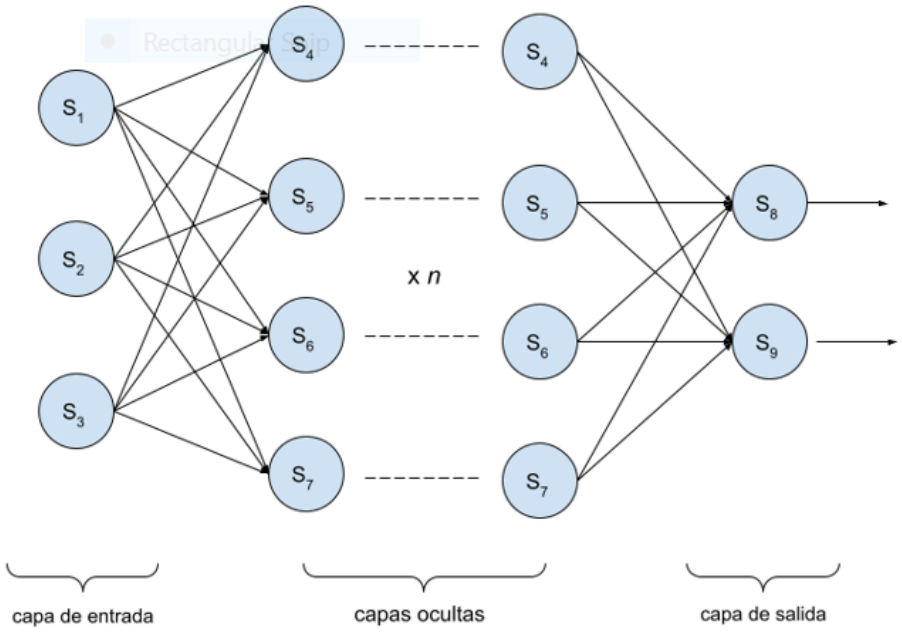

Figura 6. Red neuronal profunda

En el repositorio asociado con estos materiales podrás encontrar un programa de demostración de una red neuronal en funcionamiento.

3. Word Embeddings

Si hemos visto los ejemplos en funcionamiento de una neurona artificial y una red neuronal en las demo de la sección anterior, habremos observado que las entradas en las redes neuronales son valores numéricos. En el caso que nos interesa, la traducción automática, las unidades de entrada serán palabras pero estas palabras se tendrán que representar mediante valores numéricos. Esta representación numérica de las palabras se hace mediante los llamados *word embeddings*[4]. Estos word embeddings toman la forma de vectores numéricos con muchas dimensiones (usualmente 300 o más dimensiones).

3.1. Cálculo de word embeddings

Pero, ¿como podemos convertir una palabra en un vector numérico? En esta sección presentaremos algunas de las técnicas empleadas para calcular word embeddings.

3.1.a One-hot word embeddings

La primera idea que se nos ocurre para representar una palabra mediante un vector es usar vectores con una dimensión igual al número total de palabras que queramos representar, es decir, el diccionario. Entonces la posición de la palabra en el diccionario determinará en qué posición del vector ponemos un 1, y el resto de los valores del vector serán un 0. Por ejemplo, si tenemos un diccionario con 150.000 palabras y la palabra que queremos representar ocupa la posición 3.270, el one-hot vector tendrá ceros en las 150.000 posiciones del vector, excepto en la posición 3.270, que tendrá un 1.

Esta técnica es muy fácil de implementar y rápida de ejecutar, pero los vectores resultantes no tienen ninguna relación especial con las palabras más allá de una posición concreta en una lista.

En el repositorio asociado con estos materiales encontraréis un programa en Python que implementa el cálculo de los one-hot word embeddings.

3.1.b. Word embeddings contextuales

Si queremos disponer de representaciones vectoriales de las palabras que tengan alguna relación con el significado de la palabra representada, será necesario que nos fijemos en el contexto en el que aparece la palabra en un corpus. La relación entre el contexto de una palabra y su significado es un hecho que han destacado muchos lingüistas desde hace muchos años, como por ejemplo en las siguientes famosas afirmaciones:

[4] En español los *words embeddings* a menudo reciben la denominación de incrustación de palabras. Aun así, este término no está totalmente fijado y todavía suena extraño en muchos ámbitos. Por este motivo, en este capítulo mantendremos la denominación inglesa.

Never ask for the meaning of a word in isolation, but only in the context of a sentence (Frege, 1884)

For a large class of cases... the meaning of a word is its use in the language (Wittgenstein, 1953)

You shall know a word by the company it keeps (Firth, 1957)

Words that occur in similar contexts tend to have similar meaning (Harris, 1954)

Existen diversos algoritmos para el cálculo de word embeddings que se basan en esta idea y que tienen en cuenta una ventana contextual, definida por una cadena de palabras antes y después de la palabra central para la que queremos calcular el embedding. Por ejemplo, si consideramos la palabra central coche y la siguiente oración:

La matrícula de mi coche era fácilmente identificable en una ciudad pequeña como la nuestra[5].

La ventana contextual estaría definida por el ancho de la ventana (c) definida por el número de palabras antes y después de la palabra central. En la siguiente tabla podemos observar la palabra centrar en azul y las palabras de la ventana contextual para diferentes valores de c.

c=0	La matrícula de mi coche era fácilmente identificable en una ciudad pequeña como la nuestra.
c=1	La matrícula de mi coche era fácilmente identificable en una ciudad pequeña como la nuestra.
c=2	La matrícula de mi coche era fácilmente identificable en una ciudad pequeña como la nuestra.
c=3	La matrícula de mi coche era fácilmente identificable en una ciudad pequeña como la nuestra.

Siguiendo esta idea, podemos distinguir dos maneras de calcular los word embeddings:

- CBOW (*Continuous Bag of Words*), que lee las palabras del contexto e intenta predecir la palabra central más probable.
- Modelo Skip-Gram, que predice las palabras del contexto a partir de la palabra central.

[5] Oración extraída de Carme Riera, Jo pos per testimoni les gavines, 1977.

En la figura 7 podemos observar gráficamente esta idea.

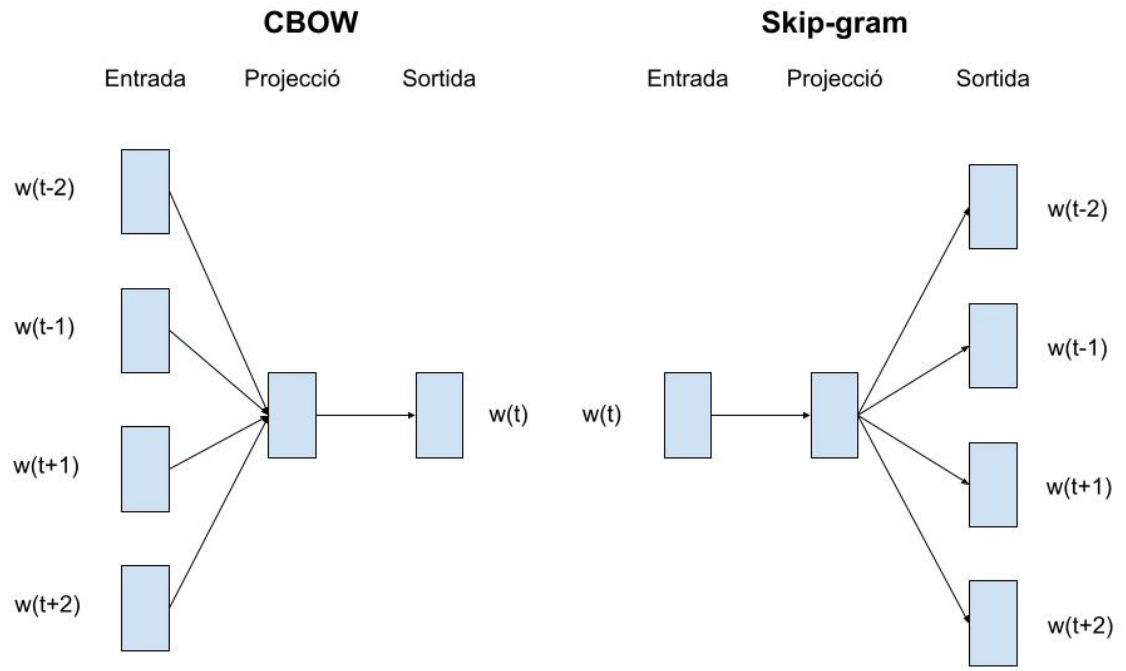

Figura 7. Arquitectura CBOW (que predice la palabra actual a partir del contexto) y Skip-gram (que predice las palabras del alrededor a partir de la palabra actual)

Estos dos modelos se entrenan a partir de corpus de oraciones de la lengua donde cada una de las palabras aparecerá varias veces en contextos diferentes. En el repositorio asociado a este material podéis encontrar programas que calculan los word embeddings usando estas dos metodologías.

Si nos fijamos ahora en el vector asociado a la palabra "coche" calculado con CBOW y usando un corpus de 100.000 segmentos del español obtenemos algo del estilo (el cálculo se ha hecho con 512 dimensiones y aquí lo mostramos recortado).

```
coche 0.26469058 0.11690102 0.064389735 0.0999779 -0.4663968 -0.039075375 -
0.24742055 -0.3318912 0.13531172 0.25524357… 0.4009233 0.4096302
0.015230008 0.23592739 -0.256997 0.8789512 -0.5697038 -0.044996012 -
0.3101102 -0.55564547
```

Y si observamos el vector correspondiente calculado con Skip Gram veríamos:

```
coche -0.014861197 -0.06669786 0.09336726 0.029614344 -0.095167145
0.13100427 -0.21605827 -0.08450274 0.032672953 0.24099794… 0.21151796
0.32695696 0.03717789 0.22560515 -0.04501914 0.28448212 -0.2495678 -
0.1408974 -0.005146907 -0.06010072
```

Además de observar que son cifras diferentes, poca cosa nos quedaría clara. Ahora bien, los vectores calculados de este modo guardan cierta relación con el significado de la palabra. Recuerda que el contexto de una palabra tiene que ver con su significado y que estos vectores se han calculado a partir de los contextos de las palabras.

Para comprobar que los word embeddings tienen que ver con las palabras, podemos hacer una serie de operaciones con los vectores. La primera de ella sería el cálculo de la similitud entre dos vectores que representen palabras con significados parecidos, por ejemplo perro y gato, que tendría que ser más grande que la similitud entre los vectores de palabras que tienen significados diferentes, como perro y grifo. En el repositorio hay un programa que calcula esta similitud y que ofrece los siguientes resultados:

similitud entre perro y gato: 0.81765985
similitud entre perro y grifo: 0.14417386

Otra posibilidad es hacer operaciones con los vectores y obtener resultados. En el repositorio disponéis de un programa que toma el vector de rey, le resta el vector de hombre y le suma el de mujer y busca las palabras con vectores más similares al vector resultante de esta operación. ¿Que da como resultado?

reina 0.5088128447532654
zar 0.5039949417114258
conde 0.5027994513511658
príncipe 0.5018768906593323
monarca 0.49871590733528137
trono 0.495973140001297
corte 0.4910587966442108
Napoleón 0.4826952815055847
duque 0.4769935607910156
papa 0.4703159034252167

La palabra cuyo vector es más similar al resultado de la operación es reina. En cierto modo estas operaciones hacen: cogen rey y le sacan las características de hombre y le suman las de mujer. ¿Y si a Madrid le restamos España y le sumamos Francia?

París 0.6241433024406433
Nápoles 0.5885446071624756
Suecia 0.5847496390342712
Bélgica 0.5675634145736694
Polonia 0.549278736114502
Alemania 0.5441193580627441

Dinamarca 0.5427573919296265
Italia 0.5388016104698181
Londres 0.5377974510192871
Suiza 0.5317127108573914

Estas pueden ser algunas demostraciones del hecho que los word embeddings tienen relación con el significado de las palabras.

3.1.c. Aplicaciones de los word embeddings

Los word embeddings son un componente fundamental en traducción automática neuronal, pero también se usan en otras muchas áreas del Procesamiento del Lenguaje Natural. Algunos otros ámbitos de aplicación son:

- Análisis de sentimientos (*sentiment analysis*). Esta tarea consiste a determinar si una determinada oración tiene un sentimiento positivo o negativo asociado. Se usa habitualmente para analizar las opiniones de los clientes hacia ciertos productos o servicios. Normalmente estos sistemas se entrenan de forma supervisada usando un corpus de oraciones que están anotadas según el grado de positividad o negatividad del sentimiento que transmite. Por ejemplo podría ser del 0 (muy negativo) al 1 (muy positivo). Con este corpus se entrena una red neuronal bastante sencilla que es capaz de asignar un grado de positividad o negatividad a una nueva oración de entrada. Si el resultado es, por ejemplo, 0.97 quiere decir que la oración expresa un sentimiento muy positivo, si es 0.18, muy negativo, y si es de 0.52 expresaría una opinión neutra.
- Reconocimiento de entidades nombradas (*Named Entity Recognition, NER*). Estos sistemas pretenden reconocer y clasificar nombres de persona, empresas, ciudades, países, etc. El uso de word embeddings puede mejorar la precisión de estos sistemas.
- Sistemas de pregunta-respuesta (*Question Answering*). Estos sistemas son capaces de responder a preguntas expresadas en lenguaje natural y dar la respuesta también en lenguaje natural. Los word embeddings pueden usar tanto para analizar la pregunta, como para procesar el corpus de donde se obtendrán las respuestas.

3.1.d. Modelos de word embeddings pre-entrenados

En esta sección hemos visto cómo podemos calcular los word embeddings a partir de un corpus. Para que los word embeddings sean significativos y funcionen bien en diferentes tareas es importante que se calculen a partir de corpus de gran tamaño. El uso de corpus muy grandes miedo hacer que el cálculo de los word embeddings sea muy lento y que sea necesario disponer de ordenadores muy potentes. Afortunadamente, hay disponible una gran cantidad de colecciones de word-embeddings pre-entrenados que se pueden descargar y usar libremente. Entre estas colecciones se pueden destacar:

- GloVe (*Global Vectors for Word Representation*)[6] (Pennington, Socher i Manning, 2014). Es un algoritmo no supervisado para el cálculo de word embeddings que se basa en estadísticas de coocurrencias entre palabras. También ofrece una serie de modelos pre-entrenados.
- Word2Vec[7] (Mikolov et al., 2013). Es una herramienta que proporciona una implementación eficiente de las arquitecturas CBOW y Skip Gram para el cálculo de word embeddings.
- fastText[8] (Bojanowski et al., 2017). Es una librería que permite entrenar representaciones y clasificaciones de texto. Permite, además, descargar un modelo monolingüe para el inglés y un modelo multilingüe compatible con 157 lenguas.

Incluso se pueden encontrar buscadores específicos para encontrar word embeddings pre-entrenados, como el *NLPL word embedding repository*.[9]

[6] https://nlp.stanford.edu/projects/glove/
[7] https://code.google.com/archive/p/word2vec/
[8] https://fasttext.cc/
[9] http://vectors.nlpl.eu/repository/

4. Sentence embeddings

El concepto de *sentence embedding* es similar al de *word embedding*, pero en lugar de representar numéricamente en forma de vector una única palabra, se representa toda una oración como un vector que también tendrá relación con su significado.

No entraremos en detalles de cómo se pueden calcular los *sentence embeddings*, pero mencionaremos algunos métodos (Aponyi, 2021):

- Un método considerado como *baseline* consistiría al usar los word embeddings de cada palabra de la oración y calcular la media de todos estos vectores.
- SkipThought (Kiros et al., 2015): es una adaptación del algoritmo Word2Vec para el cálculo de word embeddings. El sistema entrena un modelo codificador-decodificador que intenta reconstruir las oraciones que rodean a la oración codificada.
- SentenceBERT[10] (Reimers y Gurevych, 2019): se basa en el modelo BERT (*Bidireccional Encoder Representations from Transformers*) y es capaz de codificar oraciones en más de 100 lenguas.
- InferSent (Conneau et al., 2017): produce sentence embeddings para el inglés entrenando redes neuronales para identificar relaciones semánticas entre oraciones de una manera supervisada.
- Universal Sentence Encoder (USE) (Cer et al., 2018): se trata de dos modelos que aprovechan el aprendizaje multitarea para codificar frases en sentence embeddings muy genéricos que son fácilmente adaptables a una amplia gama de tareas de Procesamiento del Lenguaje Natural.

La mayoría de modelos de sentence embeddings mencionados anteriormente (excepto SenteceBERT) son monolingües. Pero se han desarrollado una serie de modelos que permiten representar oraciones en muchos idiomas en un mismo espacio vectorial. Algunos de estos modelos son:

- LASER (*Language-Agnostic SEntence Representations*)[11] (Schwenk i Douze, 2017). Es una librería para calcular y usar sentence embeddings multilingües y que ofrece modelos que proporcionan soporte para más de 200 lenguas.
- LaBSE (*Language-agnostic BERT Sentence Embedding*)[12] (Feng et al., 2022). Es una adaptación de BERT para producir sentence embeddings independiente de la lengua para 109 lenguas.

Del mismo modo que podemos encontrar palabras con significado similar representándolas con sus word embeddings y buscando las palabras con word embeddings más próximos en el espacio multidimensional, podemos usar la misma idea para buscar oraciones que sean equivalentes de traducción en un corpus no paralelo. En el repositorio asociado a estos

[10] https://www.sbert.net/
[11] https://github.com/facebookresearch/LASER
[12] https://github.com/bojone/labse

materiales encontraréis los archivos y programas necesarios para llevar a cabo estos experimentos.

5. Modelos de lenguaje neuronales

Un modelo de lenguaje es un modelo capaz de predecir la palabra que sigue a un cierto número de palabras precedentes.

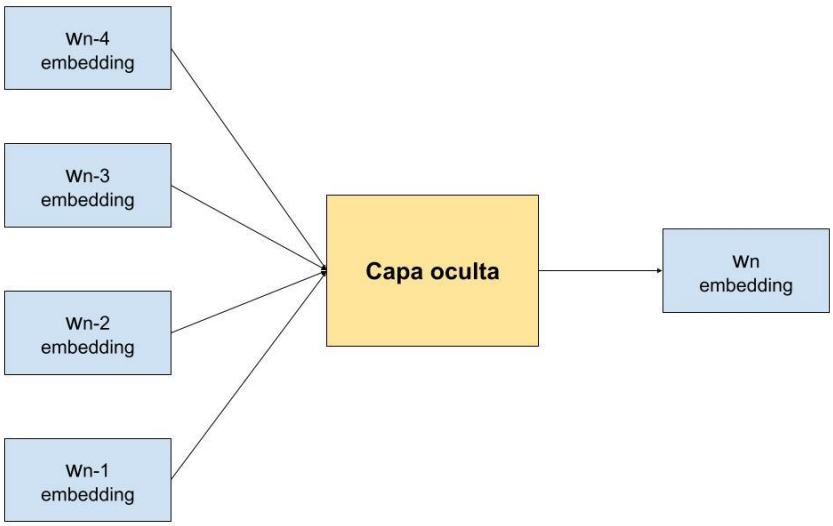

Figura 8. Esquema de un modelo de lenguaje neuronal

En la figura 8 podemos observar un esquema de modelo de lenguaje neuronal. Cada una de las palabras de entrada se representan con sus word embeddings y la red neuronal es capaz de dar a la salida un word embedding correspondiente a la siguiente palabra de la secuencia.

Para entrenar esta red neuronal necesitamos de un corpus de gran tamaño. Con este corpus se calculan los modelos de word embeddings y también todos los parámetros de la red neuronal. Para hacer esto se procesan todos los n-gramas del corpus y se van modificando los parámetros de la red para que sea capaz de predecir la palabra siguiente.

La arquitectura descrita hasta aquí se denomina *feed-forward neural network* y usa una ventana contextual de palabras de tamaño fijo (4 en la figura 8)

Las llamadas redes neuronales recurrentes (*recurrent neural networks*) pueden tener en cuenta secuencias de palabras de cualquier tamaño. Para hacer esto, reutiliza la capa oculta usada por pedir la palabra w_n como entrada adicional para predecir la palabra w_{n+1}

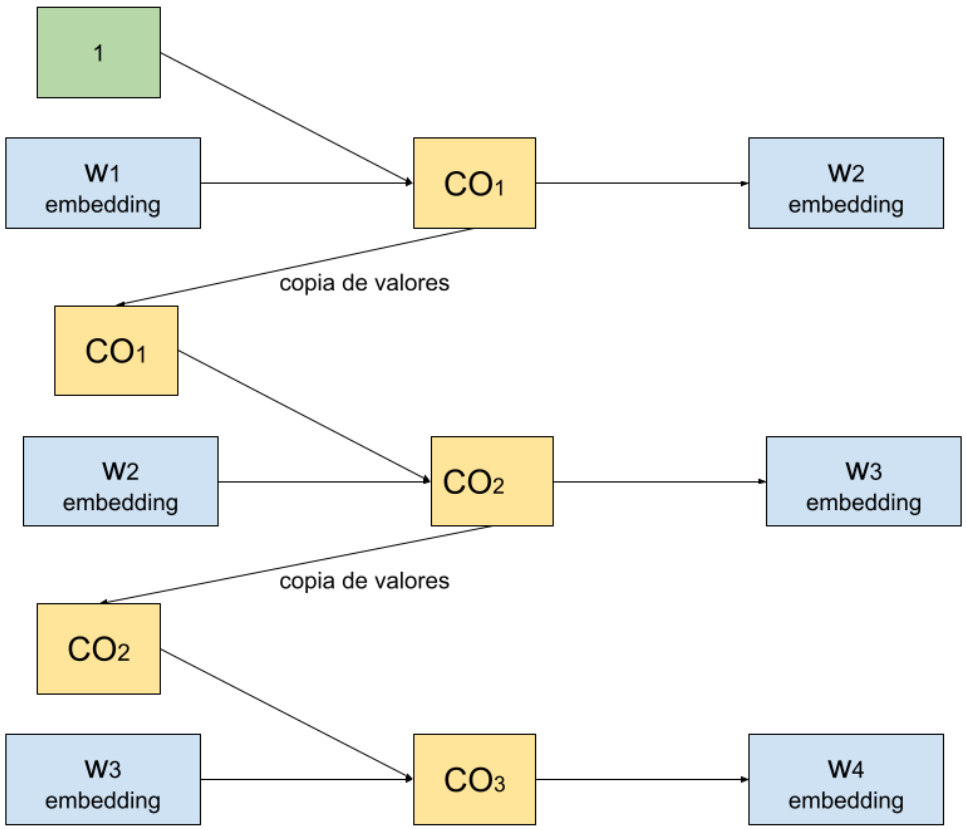

Figura 9. Esquema de un modelo de lenguaje neuronal recurrente (*recurrent neural language model*).

En la figura 9 podemos observar un esquema de un modelo de lenguaje neuronal recurrente. Después de predecir la palabra 2 en el contexto que sigue la palabra 1, reutiliza la capa oculta 1 (CO 1) y la usa, junto con el word embedding de la palabra 2 para predecir la palabra 3. A su vez, la capa oculta 2 (CO_2) se usa con el word embedding de la palabra 3 para predecir la palabra 4.

Para predecir la palabra 2 usamos el word embedding de la palabra 1, pero no tenemos ninguna capa oculta anterior y lo que se usa en su lugar es el vector asociado al símbolo de inicio de oración.

Es muy importante tener en cuenta que las capas ocultas de los pasos precedentes contienen información sobre todo el contexto anterior, no solo información sobre la palabra anterior. Por lo tanto, incluso la palabra predicha como última de la secuencia está condicionada en cierto modo por la primera palabra,

Los modelos neuronales recurrentes permiten mantener información de todo el contexto anterior, pero da más importancia a los contextos próximos. En muchas ocasiones las

palabras que proporcionan más información son las próximas, como en el siguiente ejemplo:[13]

After much economic progress over the years, the country → has

Donde se predice la correctamente la siguiente palabra (*has*). Pero en otros casos, las palabras más distantes son las más importantes, como por ejemplo en:

The country which has made economic progress over the years → has

En este ejemplo la conjugación del verbo *have* depende del sujeto *country*, que está separado por una oración subordinada bastante larga.

Para solucionar este problema se puede usar una arquitectura llamada *long short-term memory (LSTM)* o una alternativa más simple, denominada *gated recurrente units (GRU)*. En este capítulo no entraremos en detalles sobre estas arquitecturas. Los lectores interesados pueden consultar Koehn (2017): 13.4.5 y 13.4.6.

[13] Aquest exemple està reproduït de Koehn (2017).

6. Modelos de traducción neuronal

Las arquitecturas más habituales de traducción automática neuronal son una extensión directa de los modelos de lenguaje neuronales, con un añadido adicional, un modelo de alineación.

6.1. Arquitectura codificador-decodificador

Esta arquitectura es una extensión directa de los modelos de lenguaje neuronales. Los modelos de lenguaje recurrentes (figura 9) predicen la siguiente palabra a partir de las palabras anteriores. Utilizando exactamente esta misma estructura, podemos entrenar un sistema de traducción concatenando las oraciones correspondientes a la lengua de partida y a la lengua de llegada de un corpus paralelo, añadiendo a la oración correspondiente a la lengua de partida una marca de inicio de oración (por ejemplo <s>) y una marca de final de oración (por ejemplo </s>). Si por ejemplo en nuestro corpus paralelo tenemos el par de oraciones:

The house is big. La casa es grande.

Añadiríamos las siguientes marcas:

<s> The house is big . </s> La casa es grande .

Y esto mismo lo haríamos para todos los pares de oraciones del corpus paralelo de entrenamiento y procederíamos a entrenar la red neuronal.

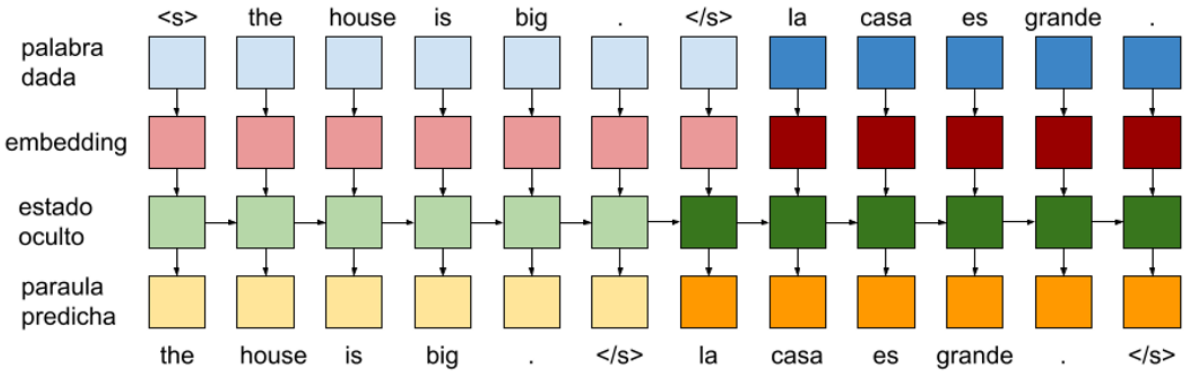

Figura 10. Modelo de traducción *sequence-to-sequence encoder*-decoder

En el momento de traducir con el modelo entrenado, introduciríamos en el sistema toda la oración de partida y continuaríamos con el proceso de obtención de predicciones hasta obtener el símbolo de final de oración.

A esta arquitectura de codificador-decodificador se le acostumbra a denominar *sequence-to-sequence encoder-decoder model*. La tarea del codificador (*encoder*) consiste en proporcionar una representación de la oración de entrada. El decodificador (*decoder*), por su parte, genera las predicciones de las palabras de salida.

Esta arquitectura funciona razonablemente bien para oraciones cortas (hasta 10 u 11 palabras), pero no funciona demasiado bien para oraciones largas. En general estos sistemas tienen que usar representaciones vectoriales de las oraciones de partida que tienen una longitud fija y por este motivo tienen la mencionada dificultad para traducir oraciones largas.

Para saber más sobre esta arquitectura podéis consultar Cho et al. (2014a), Cho et al. (2014b) y Sutskever et al. (2014).

6.2. Atención o modelo de alineación

El primer modelo de traducción automática neuronal que obtuvo unos resultados aceptables tenía una arquitectura *sequence-to-sequence encoder-decoder* con atención. Esta atención consiste en un mecanismo de alineación. Los términos atención y alineación son intercambiables en este contexto.

Como ya hemos comentado, el codificador (*encoder*) se encarga de proporcionar una representación vectorial de la oración de entrada. Dado que la oración de entrada es una secuencia de palabras, el codificador consulta una matriz de embeddings para construir una representación vectorial de la oración de entrada. El codificador procesa las palabras de entrada desde el principio al final, con una red neuronal recurrente, que resulta en estados ocultos que codifican cada palabra teniendo en cuenta su contexto anterior. Ahora bien, muchas arquitecturas, para obtener una representación más fiel al contexto, procesan también las oraciones de entrada desde el final hacia el principio, para obtener estados ocultos que representen también el contexto posterior a cada palabra. Las redes neuronales recurrentes que funcionan en las dos direcciones, es decir, tanto del principio de la oración hacia el final, como desde el final de la oración hacia el principio, se denominan redes neuronales recurrentes bidireccionales (*bidirectional recurrent neural network*).

A partir de estos estados ocultos se predicen las palabras de salida. Estas predicciones toman la forma de distribuciones de probabilidades sobre todo el vocabulario de salida.

Durante el entrenamiento del sistema las palabras de salida son conocidas, puesto que entrenamos con un corpus paralelo, y el objetivo del entrenamiento consiste a dar la máxima masa de probabilidad a las palabras de salida correctas.

El mecanismo de atención calcula asociaciones entre el último estado oculto del decodificador y las representaciones de cada palabra, es decir, los estados del codificador.

Estas asociaciones se usan para calcular una suma ponderada de los estados del codificador. La salida de este cálculo es un número que indica como de importante es una determinada palabra de entrada para producir una determinada palabra de salida.

6.3. Entrenamiento

El proceso de entrenamiento de un modelo de traducción automática neuronal consiste en los siguientes pasos (algunos conceptos que aparecen se explican un poco más adelante):

- Desordenar aleatoriamente (*shuffle*) el corpus de entrenamiento. Este paso es importante para evitar posibles efectos negativos en el supuesto de que el texto tenga algún tipo de orden temporal o temático. No es necesario llevar a cabo este proceso si ha hemos desordenado el corpus aleatoriamente antes del entrenamiento.
- Dividir el corpus de entrenamiento en *maxi-batches*.
- Dividir cada *maxi-batch* *mine-batches*.
- Procesar cada *mini-batch* para obtener los gradientes.

Antes de continuar hay que aclarar los conceptos de *batch*, *maxi-batch* y *mini-batch* (que en castellano se podrían traducir por lote, maxi-lote y mini-lote). Hay que recordar que el entrenamiento de sistemas neuronales requiere el cálculo de miles de millones de operaciones y que para poder llevar a cabo este entrenamiento en un tiempo razonable es imprescindible paralelizar estos cálculos (es decir, llevarlos a cabo en paralelo, al mismo tiempo). Por este motivo se utilizan unidades GPU (*Graphical Processing Units*) que disponen de centenares o miles de procesadores, lo que permite hacer esta paralelización). Para incrementar todavía más esta paralelización se procesan varios pares de segmentos al mismo tiempo (por ejemplo 100).

A la totalidad del corpus de entrenamiento a menudo se le denomina *batch* (en plural *batches*). Los *maxi-batches* son subconjuntos del corpus de entrenamiento (a pesar de que en algunos textos encontraremos que en estos subconjuntos se le denominan *batches*). A su vez, los *maxi-batches* se dividen en una serie de subconjuntos que reciben el nombre de *mini-batches*. Estos *mini-batches* se crean con segmentos que tienen una longitud similar. Esto se hace de este modo porque para poder hacer la paralelización es necesario que los segmentos que se procesan simultáneamente tengan la misma longitud, es decir, el mismo número de palabras. Los segmentos de cada *mini-batch* tienen longitudes similares y se convierten en segmentos de la misma longitud considerando la longitud del segmento más largo del *mini-batch* y añadiendo elementos vacíos o *no-palabra* a los segmentos más cortos hasta completar la longitud del más largo. El sistema tiene en cuenta hasta donde llegan los datos válidos de cada segmento. El hecho de agrupar en *mini-batches* segmentos con longitudes similares minimiza el número de elementos vacíos o *no-palabra* introducidos y optimiza el proceso de entrenamiento.

A un paso completo sobre todo el corpus de entrenamiento se le denomina época (*epoch*) y típicamente un entrenamiento dura entre 5 y 15 épocas. Pero es muy importante tener en cuenta que hay que establecer un criterio de parada del entrenamiento, puesto que si no, el entrenamiento no pararía nunca y llegaría un momento donde la calidad del sistema no mejoraría, e incluso empeoraría debido al fenómeno de *overfitting* (que se podría traducir al español como sobreajuste). El concepto de *overfitting* significa que el sistema funciona muy bien para los datos de entrenamiento, pero que no es capaz de generalizar bien sobre nuevos datos.

Así pues, es necesario establecer un criterio de parada o *early stopping*. Para hacer esto se reserva una parte de corpus paralelo para la validación (este fragmento recibe el nombre de corpus de validación). Cada ciertos pasos de entrenamiento se traduce la parte del corpus de validación correspondiente a la lengua de partida y el resultado se compara con la parte correspondiente a la lengua de llegada del corpus de validación. A partir de esta comparación se calculan los valores de una o más métricas de validación (métricas como la entropía cruzada (*cross-entropy*), BLEU u otras, o incluso más de una métrica al mismo tiempo). Cuando el valor de esta métrica de validación no mejora en un cierto número de pasos de validación (por ejemplo 5), el entrenamiento se detiene y se conserva el modelo que ha proporcionado el mejor valor. A este número de pasos de validación antes de parar el entrenamiento se le denomina paciencia (*patience*)

A la figura 11 podemos observar un gráfico con los valores del BLEU-detok (el BLEU calculado una vez se ha hecho la detokenización de sentencepiece de la traducción) en un entrenamiento real de un sistema de traducción automática neuronal.

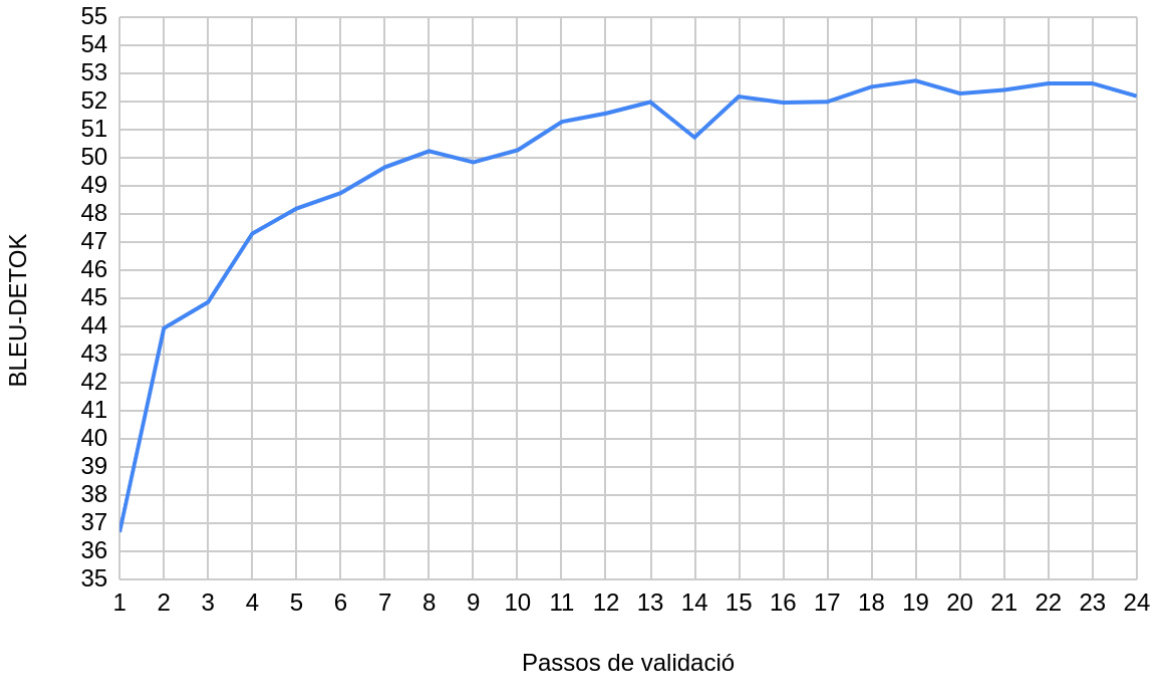

Figura 11. Valores de BLEU en la validación

Cómo se puede observar, el valor de BLEU-DETOK aumenta muy rápidamente durante las primeras validaciones, pero se va estabilizando con el tiempo. El proceso de entrenamiento se ha parado al llegar a las 5 validaciones sin obtener mejoras, y el modelo final es el correspondiente a la validación con el valor más alto. En el gráfico de la figura 11 sería el modelo correspondiente al paso de validación 19, que logra un valor de BLEU-DETOK de 52.74.

7. Conclusiones

En estos materiales hemos visto los principios básicos de funcionamiento de los sistemas de traducción automática neuronal. Quien quiera profundizar sobre la traducción automática neuronal puede leer el siguiente libro:

Koehn, P. (2020). Neural machine translation. Cambridge University Press.

Para aprender a entrenar sistemas de traducción automática neuronal, y también estadísticos puedes consultar el Curso práctico de Traducción Automática[14] de Antoni Oliver.

Bibliografía

ALPAC (1966) LANGUAGE AND MACHINES COMPUTERS IN TRANSLATION AND LINGUISTICS https://web.archive.org/web/20110409070141/http://www.mt-archive.info/ALPAC-1966.pdf

Aponyi, Andras (2021). What are sentence embeddings and their appllications? TAUS (https://www.taus.net/resources/blog/what-are-sentence-embeddings-and-their-applications) (accedido 21/08/2021)

Bahdanau, D., Cho, K. H., & Bengio, Y. (2015, January). Neural machine translation by jointly learning to align and translate. In 3rd International Conference on Learning Representations, ICLR 2015.

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. Transactions of the association for computational linguistics, 5, 135-146.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Lyn Untalan Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Céspedes, Steve Yuan, Chris Tar, Yun-hsuan Sung, Brian Strope, & Ray Kurzweil (2018). Universal Sentence Encoder. In *In submission to: EMNLP demonstration*.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014a). Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (p. 1724). Association for Computational Linguistics.

Cho, K., van Merrienboer, B., Bahdanau, D., & Bengio, Y. (2014b). On the properties of neural machine translation: Encoder–decoder approaches. InProceedings of SSST-8,Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, pp.103–111, Doha, Qatar. Association for Computational Linguistics

Conneau, A., Kiela, D., Schwenk, H., Barrault, L., & Bordes, A. (2017). Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing* (pp. 670–680). Association for Computational Linguistics.

Antonio M. Corbí-Bellot, Mikel L. Forcada, Sergio Ortiz-Rojas, Juan Antonio Pérez-Ortiz, Gema Sánchez-Ramírez, Felipe Sánchez-Martínez, Iñaki Alegria, Aingeru Mayor, Kepa Sarasola (2005) "An open-source

[14] https://github.com/aoliverg/a_practical_course_on_machine_translation

shallow-transfer machine translation engine for the romance languages of Spain", in Proceedings of the European Associtation for Machine Translation, 10th Annual Conference (Budapest, Hungary, 30-31.05.2005), p. 79--86

Feng, F., Yang, Y., Cer, D., Arivazhagan, N., & Wang, W. (2022, May). Language-agnostic BERT Sentence Embedding. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (pp. 878-891).

Mikel L. Forcada, Mireia Ginestí-Rosell, Jacob Nordfalk, Jim O'Regan, Sergio Ortiz-Rojas, Juan Antonio Pérez-Ortiz, Felipe Sánchez-Martínez, Gema Ramírez-Sánchez & Francis M. Tyers (2011). Apertium: a free/open-source platform for rule-based machine translation. Machine translation, 25, 127-144.

Hany Hassan, Anthony Aue, Chang Chen, Vishal Chowdhary, Jonathan Clark, Christian Federmann, Xuedong Huang, Marcin Junczys-Dowmunt, Will Lewis, Mu Li, Shujie Liu, Tie-Yan Liu, Renqian Luo, Arul Menezes, Tao Qin, Frank Seide, Xu Tan, Fei Tian, Lijun Wu, Shuangzhi Wu, Yingce Xia, Dongdong Zhang, Zhirui Zhang, and Ming Zhou. (2018). Achieving human parity on automatic chinese to english news translation. *arXiv preprint arXiv:1803.05567*.

John Hutchins (1995) Machine Translation: a brief history. Concise history of the language sciences: from the Sumerians to the cognitivists. Edited by E.F.K.Koerner and R.E.Asher. Oxford: Pergamon Press, 1995. Pages 431-445 (https://www.infoamerica.org/documentos_pdf/bar05.pdf)

John Hutchins (1996) ALPAC: the (in)famous report. [from: MT News International, no. 14, June 1996, pp. 9-12] https://aclanthology.org/www.mt-archive.info/90/MTNI-1996-Hutchins.pdf

Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, & Sanja Fidler. (2015). Skip-Thought Vectors. https://arxiv.org/abs/1506.06726

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, Evan Herbst. (2007, June). 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).

Koehn, P. (2020). *Neural Machine Translation*. Cambridge: Cambridge University Press. doi:10.1017/9781108608480

Läubli, S., Castilho, S., Neubig, G., Sennrich, R., Shen, Q., & Toral, A. (2020). A set of recommendations for assessing human–machine parity in language translation. *Journal of Artificial Intelligence Research*, *67*, 653-672.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. In Proceedings of NIPS, 2013.

Jeffrey Pennington, Richard Socher, & Christopher D. Manning (2014). GloVe: Global Vectors for Word Representation. In Empirical Methods in Natural Language Processing (EMNLP) (pp. 1532–1543).

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., & Weinberger,K. Q. (Eds.),Advances in Neural Information Processing Systems 27, pp. 3104–3112.Curran Associates, Incl

Holger Schwenk and Matthijs Douze, Learning Joint Multilingual Sentence Representations with Neural Machine Translation, ACL workshop on Representation Learning for NLP, 2017

Toral, A., Castilho, S., Hu, K., & Way, A. 2018 Attaining the Unattainable? Reassessing Claims of Human Parity in Neural Machine Translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers* (pp. 113-123).

Weaver W 1949 Translation. In: Locke & Booth (1955): 15-23 (https://sites.cs.ucsb.edu/~lilei/course/dl4mt21fa/Weaver_1949_Translation.pdf)