NLP – MACHINE TRANSLATION

LUCÍA POBLADOR ÁLVAREZ

ALEJANDRO POLO MOLINA

Tabla de contenido

INTRODUCCIÓN

Desde la aparición de *Google Translator* y otros traductores automáticos, el interés por conseguir un nivel de naturalidad aceptable en el idioma objetivo no ha parado de crecer.

Durante los primeros años estas traducciones estaban dotadas de excesiva literalidad. Se limitaban a traducir palabra por palabra, sin tener en cuenta aspectos como el contexto, las frases hechas o los dobles sentidos. El desafío tecnológico en este contexto se encontraba – y se encuentra – en conseguir traducciones naturales y coherentes evitando la traducción literal de cada una de las palabras.

La mayor capacidad de procesamiento y de búsqueda automática de patrones con la que cuentan las máquinas actualmente ha conseguido mejorar significativamente la performance de estos traductores.

Esta práctica pretende conseguir una primera inmersión en el mundo de la traducción de texto de manera automática mediante la traducción de un texto escrito en español a inglés.

PREPROCESADO

Lectura y limpieza del texto:

El primer paso antes de comenzar con el preprocesado del texto es la lectura de los datos y división de este por frases. En este caso, el fichero con el que se va a trabajar contiene texto escrito tanto en inglés como en español que, tras eliminar el texto que viene por defecto en cada una de las líneas y ser convertido a *array*, presenta la siguiente forma:

```
spa_eng[0:5]
```

```
array([['Go.', 'Ve.'],
       ['Go.', 'Vete.'],
       ['Go.', 'Vaya.'],
       ['Go.', 'Váyase.'],
       ['Hi.', 'Hola.']], dtype='<U332')
```

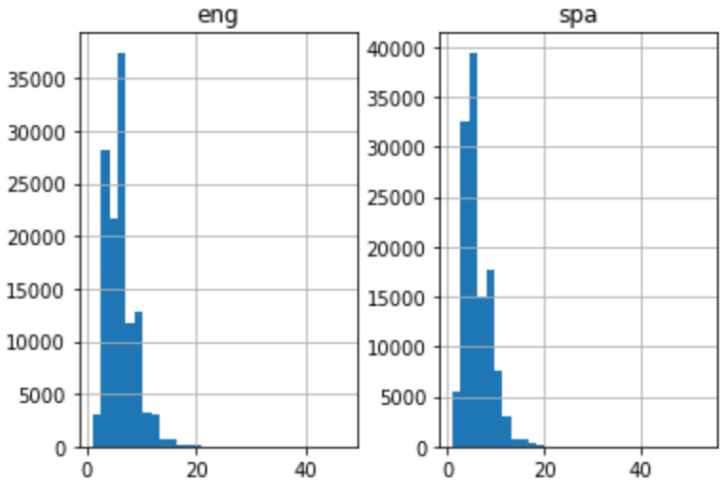
Como es natural, existen en las frases de ambos idiomas caracteres no alfanuméricos que conviene eliminar. Del mismo modo, para que nuestro modelo no sea *case sensitive*, es preciso homogeneizar el texto eliminando las mayúsculas. Finalmente, el *dataset* que se va a utilizar es:

```
spa_eng
```

```
array([['go', 've'],
       ['go', 'vete'],
       ['go', 'vaya'],
       ...,
       ['since there are usually multiple websites on any given topic i usually just click the back button when i arrive on
any webpage that has popup advertising i just go to the next page found by google and hope for something less irritating',
        'como suele haber varias páginas web sobre cualquier tema normalmente sólo le doy al botón de retroceso cuando entro
en una página web que tiene anuncios en ventanas emergentes simplemente voy a la siguiente página encontrada por google y es
pero encontrar algo menos irritante'],
       ['if you want to sound like a native speaker you must be willing to practice saying the same sentence over and over i
n the same way that banjo players practice the same phrase over and over until they can play it correctly and at the desired
tempo',
        'si quieres sonar como un hablante nativo debes estar dispuesto a practicar diciendo la misma frase una y otra vez d
e la misma manera en que un músico de banjo practica el mismo fraseo una y otra vez hasta que lo puedan tocar correctamente
y en el tiempo esperado'],
       ['it may be impossible to get a completely errorfree corpus due to the nature of this kind of collaborative effort ho
wever if we encourage members to contribute sentences in their own languages rather than experiment in languages they are le
arning we might be able to minimize errors',
        'puede que sea imposible obtener un corpus completamente libre de errores debido a la naturaleza de este tipo de esf
uerzo de colaboración sin embargo si animamos a los miembros a contribuir frases en sus propios idiomas en lugar de experime
ntar con los idiomas que están aprendiendo podríamos ser capaces de minimizar los errores']],
      dtype='<U332')
```

En el que, como se puede apreciar en la figura anterior, se encuentran desde registros con una única palabra, hasta oraciones extensas y complejas.

A nivel ilustrativo se muestra a continuación los histogramas que muestran la frecuencia con la que aparecen frases de diferente longitud:



Siendo las palabras que aparecen con mayor frecuencia:

```
[('go', 2936),
 ('hi', 20),
 ('run', 219),
 ('who', 1453),
 ('wow', 11),
 ('fire', 239),
 ('help', 1347),
 ('jump', 28),
 ('stop', 535),
 ('wait', 413)]
```

```
[('ve', 229),
 ('vete', 39),
 ('vaya', 170),
 ('váyase', 8),
 ('hola', 22),
 ('icorre', 1),
 ('icorran', 1),
 ('icorra', 1),
 ('icorred', 1),
 ('corred', 3)]
```

Codificación:

Con el *dataset* de oraciones en el formato deseado es momento de codificar el texto convirtiéndolo en vectores de igual tamaño. No obstante, como es natural, no todas las frases presentan el mismo tamaño. Para solucionar este problema se va a aplicar *padding* posterior (dejando la frase a la izquierda y los 0's a la derecha) sobre aquellas oraciones con una longitud menor que la máxima en cada uno de los idiomas.

Se ha observado que la oración más extensa en español tiene una longitud de 53 palabras, mientras que en inglés presenta una longitud de 47 palabras.

Esto, tras la codificación de las frases en secuencias de enteros y la adición de 0's para normalizar el tamaño, nos deja con unas muestras de entrenamiento del siguiente tamaño:

- Español: (98.348, 53)
- Inglés: (98.348, 47)

Y unas muestras de *test*:

- Español: (24.588, 53)
- Inglés: (24.588, 47)

Con estas 4 muestras ya estamos en disposición de comenzar a construir el modelo.

SEQUENCE TO SEQUENCE MODEL

Como sabemos, las redes neuronales sirven para atajar una gran cantidad de problemas en los cuales podemos codificar la entrada como una serie de vectores con una longitud fijada a priori. Sin embargo, esto supone una gran limitación en algunos casos, como en el análisis de secuencias de texto, la longitud de los vectores de entrada puede cambiar. Es en este punto en el que podemos entender porqué no podemos usar una *RNN* simplemente para poder generar el algoritmo de *machine translation*. Sólo podríamos usar *RNN* en el caso en el que supiésemos la longitud de las secuencias de entrada y de salida a priori y esto fuese así de forma fija. [1]

¿Cómo podemos solventar el problema de la dimensionalidad, entonces?

Pues bien, el objetivo va a ser mapear la secuencia de entrada a un vector de una longitud fija usando una *RNN* y luego mapear dicho vector a la secuencia *target* usando otra *RNN*. En particular, nosotros vamos a considerar el caso de emplear una red neuronal profunda de tipo *LSTM*. El motivo por el que privilegiamos estas redes por encima de las *simpleRNN* es que, como sabemos, estas últimas sufren el principio del *vanishing gradient* que provoca que las *RNN* no sean capaces de retener la información en el largo plazo.

Volviendo a nuestro modelo, lo primero que debemos comprender a la hora de hablar de modelos de tipo *Sequence to Sequence* es nuestro *input* será una secuencia de texto y nuestro *output* también, pero la longitud de ambas no tiene porqué ser las mismas. Esto es algo obvio y natural en el caso que nos ocupa, al traducir una frase puede que la longitud de la misma cambie entre idiomas.

En cuanto al modelo, *seq2seq* es el resultado de la combinación de un *encoder* con un *decoder*. Veamos qué significa esto en más detalle. En primer lugar, plantearemos la estructura desde un punto de vista más matemático, para comprender el *core* de las redes, y luego pasaremos a su estructura e implementación propiamente.

Siguiendo con lo comentado anteriormente, tenemos que ser capaces de convertir la entrada en un vector de una longitud determinada. El modelo *encoder*, que se trata de

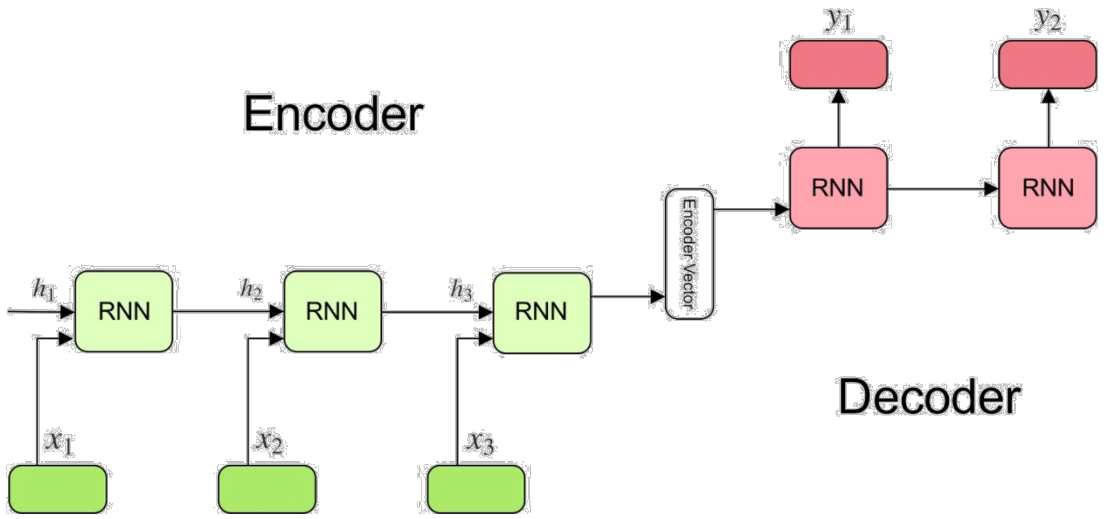
una *RNN/LSTM*, leerá cada elemento de la secuencia en el orden que se procesa. Recordemos que una *RNN* no es más que una *feed forward neural network* con un bucle *for* de modo que el estado oculto h_t es el resultado de aplicar una función de activación $f(h_{t-1}, x_t)$ de manera recursiva. Como vemos al estado $h_t = f(h_{t-1}, x_t)$ entran como *inputs* al mismo tiempo el estado anterior y el *input* del estado t. Una vez que ha procesado toda la entrada, el último estado oculto, que de aquí en adelante llamaremos *encoder vector*, es una representación de toda la información de la secuencia.

Por su parte, el *decoder*, debe ser capaz de predecir el siguiente elemento de la secuencia y_t dado el estado oculto h_t condicionado a y_{t-1} (dado que se trata de una RNN). Por lo tanto, el estado oculto del decoder a tiempo t se calcula como

$$h'_t = f(h'_{t-1}, y_{t-1}, c) \text{ [2]}$$

dónde c es el *encoder vector*.

Una vez comprendido el *core* de ambas redes, pasamos a ver su estructura. El objetivo de un modelo de tipo *Sequence to Sequence* es mapear un *input* con una entrada fija a un *output* con un tamaño fijo, aunque el tamaño de entrada y de salida pueden ser distintos. Es claro el porqué de la necesidad de este tipo de modelos dado que no es suficiente usar una *RNN* para convertir las palabras una a una.



Encoder-decoder sequence to sequence model

Observando la arquitectura del *encoder* vemos que se compone de una serie de capas *RNN* apiladas. En el caso en que nos centramos, cada palabra será una de las entradas que tendremos como *input*. Como ya sabemos, las redes neuronales de tipo *RNN* pasan el *output* de cada celda a la siguiente junto con el *input* de esa propia celda.

Tras el paso por el *encoder*, se genera el *encoder vector*. Este es el último paso dentro de la red *encoder*. La idea es que este vector encapsule la máxima información posible de modo que permita al *decoder* hacer predicciones fiables.

Por último, presentamos el *decoder*. Esta parte de la arquitectura consiste en una serie de capas recurrentes donde en cada estado no pasará solo el estado anterior de la red y el *input* si no también el *encoder vector*. De esta forma, conseguimos solventar el problema del cambio de dimensiones y mapear dos secuencias por medio de la combinación de dos redes recurrentes.

EXPLICACIÓN DEL MODELO UTILIZADO

En esta parte nos vamos a centrar en presentar el modelo concreto que vamos a utilizar. Como hemos comentado en la sección anterior, los modelos seq2seq se componen por dos redes neuronales recurrentes que en este caso van a ser de tipo LSTM ambas. El motivo es el ya conocido, queremos que sea capaz de captar información a largo plazo. Esta característica consideramos que es absolutamente clave cuando hablamos de problemas de tipo machine translation porque lo que diferencia la traducción literal de las palabras de un modelo que sea capaz de traducir una frase es entre otras cosas ser capaz de entender el contexto y el contexto, en buena medida, viene dado por la información global que nos provee la secuencia.

La primera capa de nuestro modelo viene dada por una capa de tipo embedding para que sea capaz de generar el embedding para el corpus que suministramos en train. El útimo paso de nuestra red es una capa densa con una activación softmax para que decida, en vista a la probabilidades asignadas, que elemento es el que el elegido para formar parte de la secuencia output.

```
model = Sequential()

model.add(Embedding(spa_vocab_size, batch_size, input_length = spa_length, mask_z
ero = True))

model.add(LSTM(batch_size))

model.add(RepeatVector(eng_length))

model.add(LSTM(batch_size, return_sequences = True))

model.add(Dense(eng_vocab_size, activation = 'softmax'))


model.compile(optimizer = 'adam',

loss = 'sparse_categorical_crossentropy',

metrics=['accuracy'])
```

De cara a entrenar el modelo hemos usado un optimizador Adam dado que este tipo de algoritmos de descenso del gradiente adaptativos parecen dominar al resto de algoritmos elegibles.

ANÁLISIS DE LOS RESULTADOS OBTENIDOS

Pasamos por último a comentar los resultados que hemos obtenido. Debido a la falta de capacidad de cómputo, hemos decidido entrenar nuestro modelo durante solo 6 épocas dado que el tiempo medio por época era de 2000 s ($\approx$ 33 $minutos$) lo que hacía absolutamente imposible ejecutar el modelo por más tiempo. Para poder usar los recursos gpu de nuestra máquina en local hemos usado el backend de keras optimizado con plaidml. Igualmente tratamos de ejecutar nuestro programa en colab haciendo uso de los recursos de TPU ofrecidos por las máquinas de google pero tampoco mejora de forma excesiva la capacidad de cómputo.

Observando las curvas de error entre train y validation, se observa como el modelo parece ser capaz de obtener un muy buen resultado en ambos conjuntos desde un primer momento. El accuracy máximo obtenido en training es de 0.93 mientras que se ha obtenido un 0.92 en validación. A continuación, mostramos los resultados obtenidos tal y como se pueden ver en Tensorboard dado que hemos añadido una callback en keras para poder visualizar las curvas.

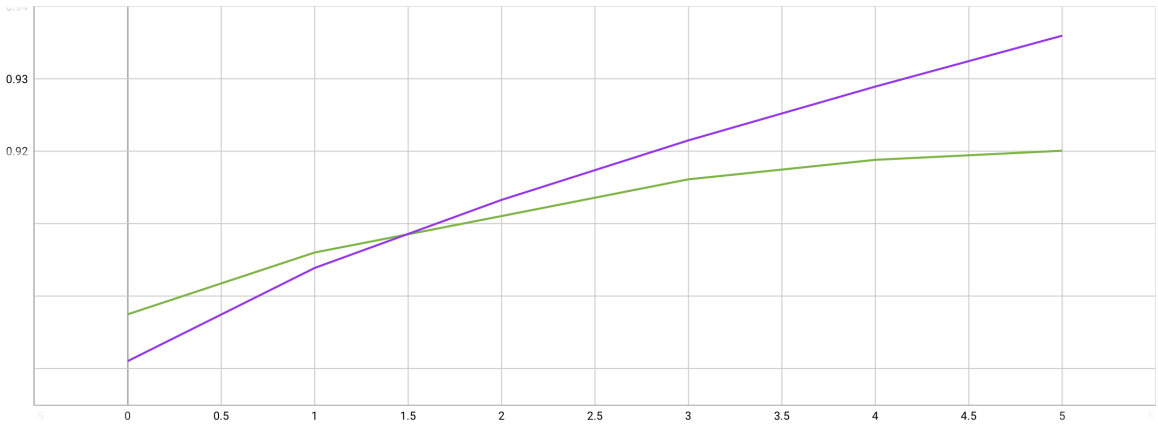


Ilustración 1: Curvas de accuracy training (morado) y validación (verde) usando Tensorboard

```
Epoch 1/10
1230/1230 [==============================] - 1976s 2s/step - loss: 0.6944 - accuracy: 0.8910 - val_loss: 0.6498 - val_accuracy: 0.8975

Epoch 00001: val_accuracy improved from 0.87521 to 0.89749, saving model to best_model.h5
Epoch 2/10
1230/1230 [==============================] - 2063s 2s/step - loss: 0.5663 - accuracy: 0.9039 - val_loss: 0.5533 - val_accuracy: 0.9060

Epoch 00002: val_accuracy improved from 0.89749 to 0.90601, saving model to best_model.h5
Epoch 3/10
1230/1230 [==============================] - 2064s 2s/step - loss: 0.4630 - accuracy: 0.9133 - val_loss: 0.4969 - val_accuracy: 0.9110

Epoch 00003: val_accuracy improved from 0.90601 to 0.91104, saving model to best_model.h5
Epoch 4/10
1230/1230 [==============================] - 2053s 2s/step - loss: 0.3839 - accuracy: 0.9215 - val_loss: 0.4621 - val_accuracy: 0.9161

Epoch 00004: val_accuracy improved from 0.91104 to 0.91612, saving model to best_model.h5
Epoch 5/10
1230/1230 [==============================] - 1974s 2s/step - loss: 0.3228 - accuracy: 0.9289 - val_loss: 0.4416 - val_accuracy: 0.9188

Epoch 00005: val_accuracy improved from 0.91612 to 0.91881, saving model to best_model.h5
Epoch 6/10
1230/1230 [==============================] - 1935s 2s/step - loss: 0.2736 - accuracy: 0.9359 - val_loss: 0.4353 - val_accuracy: 0.9201
```

Ilustración 2: Accuracy en training y validation

BIBLIOGRAFÍA

[1] O. V. a. Q. V. L. Ilya Sutskever, Sequence to Sequence Learning with Neural Networks.

[2] K. Cho, Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation.