Traducción automática: Uso métodos estadísticos y neuronales para la traducción español a inglés de Europarl

Toni Blasco Calafat
Universitat Politècnica de València
MIARFID

13 de febrero de 2023

1. Introduction

En el marco de la asignatura de Traducción automática se propone la traducción del inglés al español para el corpus de Europarl. El corpus paralelo Europarl se extrae de las actas del Parlamento Europeo. Incluye versiones en 21 lenguas europeas: Románico (francés, italiano, español, portugués, rumano), germánico (inglés, neerlandés, alemán, danés, sueco), eslavo (búlgaro, checo, polaco, eslovaco, esloveno), finougrio (finés, húngaro, estonio), báltico (letón, lituano) y griego. Para ello se hace uso de las mismas técnicas que en las utilizadas en las prácticas, tanto de la predicción estadística basada en moses[1], como en el uso de las redes neuronales[4] para la obtención de la mejor traducción. Este corpus está formado por 50.000 frases de entrenamiento y otras 1.000 de test, donde el objetivo principal será el de encontrar un traductor que realice la mejor traducción posible del inglés al español.

2. Predicción basada estadística: MOSES

Dado que el problema a resolver es una extensión del problema resuelto en prácticas, se ha optado por tener gran influencia de los parámetros y valores allí obtenidos para la realización del presente trabajo. A continuación, se presentarán cuáles han sido las diferentes variaciones y que resultados se han obtenido con ellas.

2.1. Preproceso de datos

La primera tarea realizada ha sido la manipulación del corpus de forma correcta para poder realizar diversos experimentos sobre el mismo. Dado que la descarga del corpus viene sin ningún tipo de procesado, se ha realizado primeramente una tokenización del corpus de entrenamiento y el de test para ambos idiomas. Posteriormente se ha limpiado el corpus de entrenamiento con la sentencia correspondiente con la diferencia respecto a la práctica de que se ha variado el parámetro de valor máximo de limpieza probando como valores además del 60, el 80 y el 100. Una vez tokenizado y limpiado el corpus se ha dividido el corpus de entrenamiento en dos partes una de entrenamiento para el cálculo de la traducción y otro de development para el cálculo de los pesos del modelo de la traducción estadística, en esta división del corpus se ha probado con un número de 2.000, 3.000 y 5.000 frases.

2.2. Entrenamiento del modelo de lenguaje

Primeramente con la parte de entrenamiento del corpus con un limpiado de valor máximo de 60 y 5.000 frases se ha optado por probar diferentes variaciones en el modelo de lenguaje viendo así que diferencias aporta en el BLEU para discriminar cual de los traductores obtiene mejor resultado. El software utilizado para la obtención de los modelos de lenguaje es el de SRLIM[2].Siguiendo los resultados de las prácticas se ha optado por una variación de trigramas y 5-gramas además de diferentes tipos de descuento, tanto el KneserKney como el WrittenBell que eran los que mejores resultados obtenían empíricamente de los testados.

2.3. Entrenamiento del modelo de traducción

El siguiente paso es el de la obtención del modelo de traducción a partir del modelo de lenguaje, se ha hecho uso del software GIZA++[3] para construir las tablas de segmentos y los modelos de reordenamientos que correspondan a los modelos escogidos dado el conjunto de entrenamiento de ambos corpus.

2.4. Entrenamiento de los pesos del modelo

El último paso del entrenamiento es el ajuste de los pesos del modelo de traducción anterior. Para ello se hace uso de MERT, un software que aplica técnicas de optimización y ajuste de pesos del modelo. El ajuste de pesos se ha realizado con el conjunto de development comentado anteriormente. Se ha probado la obtención de los pesos con 5 iteraciones como en la práctica para discriminar el modelo de lenguaje y finalmente con 7 y 10 iteraciones. Recalcar que el proceso de ajuste de pesos es tardío ya que necesita del corpus por cada iteración.

2.5. Proceso de traducción del traductor

Una vez ajustados los pesos del modelo, el siguiente paso consiste en realizar la traducción del conjunto no visto en entrenamiento, es decir, del conjunto de test en inglés al español. Para ello a diferencia que en el caso de entrenamiento, no han sido limpiados simplemente tokenizados para la obtención de los resultados.

La herramienta que se ha utilizado para el proceso de traducción del traductor entrenado ha sido el descodificador de MOSES, el resultado ha sido guardado para compararlo posteriormente con el conjunto de test en español viendo así cual era la calidad del resultado. A continuación en la siguiente sección se exponen que resultados se han obtenido para las diversas pruebas realizadas.

2.6. Análisis de resultados

En esta sección se pretenden mostrar cuáles han sido los resultados obtenidos anteriormente con las variaciones de los parámetros mostrados además de una discusión acerca de porque se pueden dar esos resultados.

2.6.1. Resultados con la variación del modelo de lenguaje

Como bien se ha remarcado en la sección del modelo de lenguaje, se han probado diversos modelos entre 3 y 5-gramas con diferentes métodos de variación en el descuento. Todos estos resultados son con un clear del corpus de 60 y con 5 iteraciones en el modelo de ajustes de los pesos en el traductor. En esta tabla se muestran los resultados:

N-gramas	**Suavizado**	**Descuento**	**Develop**	**BLEU**
3-gramas	Interpolate	KneserKney	5.000	27.83
5-gramas	Interpolate	KneserKney	5.000	27.93
3-gramas	Interpolate	WrittenBell	5.000	27.62
5-gramas	Interpolate	WrittenBell	5.000	27.89

Cuadro 1: Caption

Como se puede observar el modelo de 5-gramas obtiene los mejores resultados, a diferencia de en la práctica donde el modelo de trigramas obtenia mejores debido posiblemente a que el conjunto de entrenamiento era menor y los 5-gramas serían un modelo demasiado grande para el conjunto de datos obtenido. Sin embargo, al aumentar el tamaño del corpus se observa como 5-gramas obtiene resultados más altos. Es por ello que se hace uso de ese modelo con descuento KneserNey y suavizado interpolado para los siguientes experimentos.

2.6.2. Variación de parámetros

La estrategia seguida posterior a la de saber que modelo de lenguaje utilizar para la obtención del traductor ha sido la de variar el limpiado del corpus, es por ello que se hace uso de los valores de 80 y 100, para el mostrado de resultados. Cabe añadir que no se ha vuelto a hacer uso de un corpus para el development de 5.000, debido a que la mayor parte del tiempo de cómputo para encontrar una solución esta en el ajuste de pesos, cuando mayor sea el corpus mayor es el tiempo, como en las experimentaciones que se han hecho se ha optado por aumentar el número de iteraciones se ha considerado trabajar con un corpus de entre 3.000 y 5.000 pares de frases para el ajuste de pesos.

Otras de las variaciones que se han realizado han sido la del tipo de clean del corpus, primeramente para un clean de 80, se ha probado con 7 iteraciones y 2.000 de talla de test obteniendo:

Clean	develop	iteraciones	BLEU
80	3000	10	27.01
80	2000	7	27.40

Cuadro 2: Resultado traductor con un clean de 80

Como se puede observar en este caso el corpus con un clean de 80 y menos datos de entrenamiento para el ajuste de peos además de menos iteraciones, obtiene un mejor resultado aunque no muy significativo que el caso extendido, esto puede deberse a que en el caso de 3.000 frases de entrenamiento el modelo necesite de más elementos para entrenarse ya que con el uso de 5-gramas se pueda quedar pequeño. Por otra parte en la tabla 3, se muestran los resultados obtenidos de aplicar todavía un clean más alto.

Clean	develop	iteraciones	BLEU
100	2000	10	28.11

Cuadro 3: Resultado traductor con un clean de 80

En este caso en el que el clean ha obtenido el mejor resultado presentado hasta el momento, se ha decidido realizar todavía más variaciones con el objetivo de intentar mejorarlo con ese clean es por ello que se ha cambiado el modelo de lenguaje viendo lo que se puede observar en la tabla 4.

Clean	N-gramas	Descuento	develop	iteraciones	BLEU
100	3-gramas	WrittenBell	2000	10	27.80
100	5-gramas	WrittenBell	2000	10	27.82

Cuadro 4: Resultado traductor con un clean de 80

Esta tabla muestra como para un clean de 100, no se ha podido mejorar el BLEU obtenido previamente, ya sea tanto aplicando el modelo WrittenBell como el de añadir un modelo más pequeño para el conjunto de entrenamiento.

2.6.3. Conclusiones del análisis para SMT

Mostrados todas las diferentes experimentaciones sobre los modelos estadísticos que se han aplicado, se concluye con que el modelo obtenido con un clean de 100, 2000 iteraciones para el ajuste de pesos y 10 iteraciones para ello junto con un modelo de lenguaje que utiliza interpolado y un descuento de KneserKney, obtiene el mejor resultado. Se ha observado además de que aplicar diferentes técnicas de descuento sobre el modelo afecta en el resultado de forma notoria dada la gran diferencia entre el 27.82 obtenido con un descuento WrittenBell y el definitivo. Además es importante remarcar el hecho del ajuste del set de entrenamiento y development, ya que con modelos con gran dimensionalidad en el ajuste de pesos puede hacer que el modelo no se entrene bien y viceversa. Por ello, la estrategia que se ha seguido es la de escoger menor valor en el ajuste de pesos pero con más iteraciones.

3. Predicción basada en redes neuronales:NMT

Siguiendo la misma estructura que en el caso anterior, se pretende abordar el problema de traducción del corpus Europarl, pero con el uso de redes neuronales, más concretamente con el uso de un transformer que se hace uso de él en la práctica Open-NMT[5].

La estrategia abordada en este caso ha sido la configuración primeramente del transformer con la ayuda de la API y la documentación, posteriormente se mostrarán los resultados y se realizará un breve comentario al respecto, para posteriormente ver que otros parámetros se han modificado, siguiendo los resultados de la práctica, viendo así que conclusiones se pueden extraer y cuáles son los mejores resultados que se han obtenido.

3.1. Tratamiento del corpus

A diferencia que en el caso anterior, en este caso no ha sido necesaria la limpieza del corpus simplemente se ha tokenizado el corpus con la misma herramienta que la práctica. Se recalca el hecho de que no es necesario para el caso del transformer de Open-NMT la subdivisión del corpus de entrenamiento en uno de validación por lo que primero se han hecho pruebas simplemente con el corpus de entrenamiento tokenizado. Posteriormente se ha añadido el corpus de validación con diferentes tamaños.

3.2. Ajuste de parámetros

Al igual que en la práctica, se ha hecho uso del archivo de configuración de hiperparámetros, para variar diversos de sus valores, es por ello que se han provado diferentes tamaños de batch size: 50, 64, 128 y 500. Además se han provado diferentes optimizadores, siguiendo los resultados de las prácticas donde se ha probado SGD, ADAM y Adagrad, con diferentes valores de *Learning-Rate* para cada uno según necesidades. Además se han probado diferentes valores de Word-embedding y del tamaño de la red neuronal del transformer probando 64 y 128. Finalmente también se ha variado tanto el número de steps de entrenamiento como el de validation. Cabe recalcar que todas estas variaciones se han hecho con la herramienta de Google Colab, y con el uso de la GPU que proporciona ya que los tiempos de cómputo pueden ser elevados.

3.3. Planteamiento inicial

Primeramente se ha hecho uso de un corpus con las 50.000 frases para el entrenamiento sin el uso del corpus de validación debido a que el transformer no lo requiere, viendo así que resultados obtiene. Los parámetros que se han hecho uso para el entrenamiento son de un tamaño de Word-Embedding de 64, como se ha comentado anteriormente se ha optado por hacer uso de la API para valores recomendados, teniendo un optimizador ADAM dado los buenos resultados obtenidos en las prácticas, con unos valores de beta 1 y 2 de 0.9 y 0.999 respectivamente junto con un learning-rate de 0.001 obteniendo un resultado de 1.6 de BLEU. El tamaño utilizado para realizar el entrenamiento ha sido de 10.000. Posteriormente dado el resultado obtenido que es muy bajo, se ha optado por poner un corpus de validación de 4.000 frases y uno de entrenamiento de 46.000, con los mismos parámetros que los anteriores, obteniendo un resultado de 1.9 en el mejor caso. A continuación en la tabla 5, se muestran los resultados con los diferentes parámetros obtenidos de forma más clara.

Entrenamiento	Validación	Word-Embedding	Optimizador	LR	BLEU
50.000	0	64	Adam	0.001	1.6
46.000	4.000	64	Adam	0.001	1.9
46.000	4.000	128	SGD	1	0.4

Cuadro 5: Resultados OpenNMT con valores de API

3.4. Experimentaciones de otras técnicas

Viendo que la calidad de los resultados no han sido óptimas en el caso anterior, se ha seguido una estrategia parecida a la de la práctica, en el que el valor de su LR, han sido elevados además de

mantener un BATCH SIZE de 64 debido a que seguramente hayan pocos datos y el modelo sea muy grande y necesite de muchas iteraciones y de más datos para obtener resultados mejores. Es por ello que se han lanzado diversos experimentos obteniendo los siguientes resultados todos ellos un validation de 4.500 y entrenamiento de 45.000 frases con un word-embedding de 128 y un optimizador Adam.

Experimento	Batch-size	LR	Train Steps	Valid Steps	BLEU
A	64	2.0	10.000	1.000	16.1
B	64	3.0	50.000	2.000	17.6
C	64	5	45.000	2.000	17.4
D	128	10	40.000	3.000	16.5

Cuadro 6: Resultados OpenNMT con corpus de validación grande

Tras estos resultados bastante variados en sus parámetros podemos extraer una serie de reflexiones para realizar más experimentaciones en base a las conclusiones extraídas. Por una parte, se observa como a cuantas más iteraciones para entrenar el modelo el resultado mejora en todos los casos, como en caso del experimento B, donde se ha obtenido el mejor BLEU donde se tiene una gran cantidad de datos para entrenar y menos de validación. En cambio, cuando encontramos gran parte de iteraciones en la validación el resultado ha empeorado como vemos en el caso del experimento D, donde es el que más iteraciones realiza en el set de validación. Por otra parte, se observa como el aumento del batch size va relacionado con el número de iteraciones propuesto en entrenamiento, a mayor batch size se necesita de un consistente número de pasos en entrenamiento. Finalmente, se observa que el learning-rate para valores similares si se observan diferencias aunque no muy notorias. Es por ello que en las experimentaciones lanzadas a posterior a este análisis se ha tenido muy en cuenta el hecho de los pasos de entrenamiento y de validación, dado que se dispone de una sesión de Google Colab, y los datos muestran que a mayores datos de entrenamiento mayor BLEU, se ha probado a realizar unos experimentos limitando el tamaño de validación a 2.000 frases aumentando así el tamaño a 48.000 frases de entrenamiento junto con las mismas iteraciones que en el caso anterior. A continuación se muestran los resultados obtenidos en el caso de aumentar el corpus de entrenamiento.

Experimento	Batch-size	Optimizador	LR	Word-embedding	Train Steps	Valid Steps	BLEU
A	64	Adagrad	100	128	50.000	2.000	23.1
B	64	Adam	5.0	128	50.000	1.000	18.3
C	64	Adagrad	100	256	50.000	2.000	24.5
D	64	Adagrad	100	128	51.000	1.000	23.2
E	64	Adagrad	50	128	45.000	2.000	23.4

Cuadro 7: Resultados OpenNMT con corpus de validación pequeño

Sobre los resultados mostrados se pueden extraer varios puntos. Por una parte los LR respecto los experimentos anteriores no realizan grandes cambios en el resultado final, como en el experimento E y D donde para el mismo optimizador no hay una variación significativa en el BLEU, sin embargo si se remarca que un LR que no cumpla un cierto umbral si afecta negativamente al BLEU. Por otra parte, se han aumentado al máximo el número de iteraciones de entrenamiento respecto la sesión de Google Colab, se observa como darle un valor bajo al set de validación proporciona malos resultados como el caso del experimento B junto con el optimizador. Finalmente el mejor resultado se ha obtenido con un aumento del Word-embedding y un optimizador Adagrad que si ha influenciado potencialmente en el BLEU final obteniendo un 24.5

4. Uso de RNN en el Transformer

En esta sección se plantea el uso de redes neuronales recurrentes tanto en el encoder como decoder en el transformer. Para ello se ha tomado de referencia como a lo largo de la memoria diferentes parámetros tanto obtenidos anteriormente como inicialmente los de la práctica. Las redes neuronales

recurrentes utilizadas en la experimentación ha sido tanto las LSTM como las GRU. A continuación en esta tabla se presentan unos primeros resultados.

RNN	Word-embedding	Optimizador	Layers	Train	BLEU
LSTM	128	Adagrad	2	16.000	15.4
LSTM	256	Adagrad	4	13.000	9.2
GRU	128	Adagrad	4	x	
GRU	256	Adagrad	2	18.000	6.6

Cuadro 8: BLEU para diferentes redes recurrentes.

16.000, 13.000, x, 18.000 Cabe remarcar que los resultados mostrados son con las iteraciones máximas que la sesión de google colab ha permitido, los hiperparámetros utilizados a parte de los que se muestran en la tabla se han usado aquellos que proporcionaban resultados buenos en las otras experimentaciones, teniendo un LR de 100 en todos los casos además de un batch size de 64 para cada uno de los modelos, se ha variado el tamaño de word-embedding viendo que resultados producía en el BLEU además de las diferentes layers para escalar el problema. Principalmente se observa que el BLEU no mejora el resultado obtenido anteriormente, donde el número de iteraciones de entrenamiento es mucho menor a los anteriores con el mismo entorno de ejecución. Finalmente si se observa que aumentando el número de layers mejora el BLEU

4.1. Uso de Fairseq para la traducción

Finalmente se ha hecho uso de otro transformer Fairseq[7] para la traducción, Fairseq, es un conjunto de herramientas de modelado de secuencias que permite a investigadores y desarrolladores entrenar modelos personalizados para traducción, resumen, modelado lingüístico y otras tareas de generación de textos. Para la implementación de este transformer se ha hecho uso de la herramienta de Google Colab utilizando las máquinas y características que esta proporciona.

El primer paso realizado ha sido la instalación de Fairseq, se ha importado el modelo de transformer de [], donde el primer paso ha sido la instalación de las dependencias mediante **pip install -r requirements.txt**, dado que se necesitan algunas dependencias más se han instalado mediante la sentencia **pip install fastBPE sacremoses subword_nmt** y **!python setup.py build develop**, posteriormente se han aplicado los diferentes módulos del entrenamiento

1. **preprocess.py**: Para el preprocesamiento de los datos y construcción del vocabulario.
2. **train.py**: Para el entrenamiento de nuevos modelos en una o más GPU.
3. **generate.py**: Traducción de los datos del preprocesados con el modelo entrenado.
4. **interactive.py**: Traducción del texto sin procesar con el modelo entrenado.
5. **score.py**: Puntuación BLEU de las traducciones generadas frente a las traducciones de referencia.
6. **eval_lm.py**: Evaluación del modelo de lenguaje.

El primer paso realizado ha sido la de la inclusión del data Europarl, para ello se ha añadido una carpeta nueva donde se han situado tanto el corpus de entrenamiento, como el de validación y el de test, posteriormente se ha hecho uso de **preprocess.py**. Este script de generación produce cuatro tipos de salidas: una línea con el prefijo S muestra la frase fuente suministrada tras aplicar el vocabulario; O es una copia de la frase fuente original; H es la hipótesis junto con una log-verosimilitud media; y A son los máximos de atención para cada palabra de la hipótesis, incluido el marcador de fin de frase que se omite en el texto. Posteriormente se ha entrenado el modelo con 10 iteraciones usando un LR de 0.25, clip norm de 0.1 y un dropout de 0.2. El resto de parámetros son los encontrados por default en [https://fairseq.readthedocs.io/en/latest/command_line_tools.html]. Finalmente se ha hecho uso del **generate.py** para generar las traducciones y de **score.py** para la generación del BLEU, obteniendo finalmente un BLEU de 15.6.

A modo de conclusión se remarca los diferentes problemas que han surgido para la implementación de Fairseq con Google Colab por las diferentes dependencias. Por otra parte, el entrenamiento del modelo ha sido más rápido que en los otros casos, por lo que se puede extraer que harían falta muchos más epochs como en el caso de OpenNMT.

4.2. Conclusión NMT

Con la implementación de métodos que hacen uso de redes neuronales se concluye que en general hacen falta de muchos más datos que los modelos estadísticos para su entrenamiento y es por ello que a medida que mayores valores de data de entrenamiento han sido proporcionados mayores han sido los valores de BLEU obtenidos. Por otra parte, se remarca el hecho de que las variaciones en los hiperparámetros en los modelos neuronales marcan la diferencia en el entrenamiento, mientras que en el caso de los modelos estadísticos las variaciones no producían grandes cambios en el BLEU resultante.

Referencias

[1] Philipp Koehn. MOSES: Statistical Machine Translation System. User Manual and Code Guide. University of Edinburgh. 2014. http://www.statmt.org/moses/manual/manual.pdf

[2] SRILM - The SRI Language Modeling Toolkit http://www.speech.sri.com/projects/srilm/

[3] http://web.archive.org/web/20100221051856/http://code.google.com/p/giza-pp/

[4] Philipp Koehn. Neural Machine Translation. arXiv:1709.07809v1. 2017.

[5] https://opennmt.net

[6] https://opennmt.net/OpenNMT-py/options/train.html

[7] https://github.com/deeplanguageclass/fairseq.git