Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

# Traducción Automática

curso 2023-2024

Francisco Casacuberta, Miguel Domingo
Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

## 1. Traducción estadística basada en frases: MOSES

### 1.1. Introducción

El objetivo de esta práctica es construir sistemas de traducción a partir de conjuntos de pares de frases bilingües. Para ello utilizaremos la herramienta *Moses* que se puede descargar en:

`http://www.statmt.org/moses/`

*Moses* es un sistema que permite entrenar modelos de traducción automática para cualquier par de lenguas. Todo lo que se necesita es una colección de textos traducidos (corpus paralelo). Una vez el modelo entrenado, *Moses* proporciona un algoritmo de búsqueda eficiente que le permite encontrar eficientemente la traducción de mayor probabilidad.

La instalación de *Moses* tiene su dificultad porque depende de librerías que no están instaladas por defecto. Una guía se puede encontrar en:

`http://www.statmt.org/moses/manual/manual.pdf`

o en la página web:

`http://www.statmt.org/moses/?n=Development.GetStarted`

Otra guía resumida en:

`http://www.prhlt.upv.es/~fcn/Students/ta/Guide-JesusG-MOSES-13.pdf`

Esta guía de prácticas está en:

`https://www.prhlt.upv.es/~fcn/Students/ta/p1mt.pdf`

Un software extra que hay que instalar es GIZA cuyo enlace en los dos últimos está roto. Alternativamente se puede bajar de:

`https://github.com/moses-smt/giza-pp`

o instalar mgiza de:

`https://github.com/moses-smt/mgiza`

Otra alternativa para la instalación de *Moses* es el uso de Docker. Esta opción es significativamente más fácil de instalar, pero cambia un poco la forma de usarlo (se recomienda tener unas nociones básicas de Docker). Para ello, es necesario tener instalado Docker:

```
curl -fsSL https://get.docker.com -o get-docker.sh && chmod +x \
get-docker.sh && sudo sh get-docker.sh
sudo usermod -aG docker $username
```

siendo `$username` el nombre de usuario. Es necesario reinicar.

Una vez configurado Docker, en el siguiente repositorio se puede econtrar información acerca de cómo ejecutar *Moses*:

`https://github.com/midobal/dockerfiles/tree/master/moses`

En el aula informática ya está instalado el software *Moses* aunque se recomienda su instalación en los ordenadores personales con sistema operativo Linux.

## 1.2. Definición de las variables de entorno

Para empezar definiremos unas variables y actualizaremos otras para facilitar el accesos a las herramientas de Moses y auxiliares. Asumimos que hemos creado un directorio `TA` donde realizaremos las prácticas.

```
~/TA> export PATH=$PATH:/opt/moses-4/bin/
~/TA> export PATH=$PATH:/opt/moses-4/scripts/training/
~/TA> export PATH=$PATH:/opt/srilm/bin/i686-m64
~/TA> export SCRIPTS_ROOTDIR=/opt/moses-4/scripts/
~/TA> export GIZA=/opt/mgiza/
~/TA> export MOSES=/opt/moses-4/
```

## 1.3. Prueba de la instalación

En esta primera parte se pretende probar que la instalación es correcta por lo que se va a traducir una frase utilizando unos modelos ya entrenados.

```
~/TA> mkdir Practica1; cd Practica1
~/TA/Practica1> mkdir Prueba; cd Prueba
~/TA/Practica1/Prueba> wget http://www.statmt.org/moses/download/sample-models.tgz
~/TA/Practica1/Prueba> tar xzvf sample-models.tgz
~/TA/Practica1/Prueba> cd sample-models/
~/TA/Practica1/Prueba/sample-models> moses -f phrase-model/moses.ini \
< phrase-model/in > out
~/TA/Practica1/Prueba/sample-models> cat out
```

El fichero de entrada phrase-model/in contiene la oración "das ist ein kleines o haus" dos veces, por lo que el fichero de salida out debería contener la traducción "this is a small house" dos veces.

## 1.4. Preparación de los datos

Utilizaremos un conjunto de datos de la tarea EuTrans de traducción castellano-inglés en un escenario de un turista ante el mostrador de un hotel. En primer lugar volver al directorio Practica1.

```
~/TA/Practica1/Prueba/sample-models> cd ../../; mkdir Tarea; cd Tarea
~/TA/Practica1/Tarea> wget --no-check-certificate \
     http://www.prhlt.upv.es/~fcn/Students/ta/train.tgz
~/TA/Practica1/Tarea> tar xzvf train.tgz
```

El primer paso a realizar es la tokenización. En esta ocasión, el corpus ya está tokenizado y no es necesario hacer nada. En el caso de que no lo estuviera, se podría utilizar la herramienta:

```
$MOSES/scripts/tokenizer/tokenizer.perl -l en   < corpus.en > corpus.tok.en
```

A continuación, será necesario limpiar los datos:

```
~/TA/Practica1/Tarea> clean-corpus-n.perl  train/training es en train/training.clean 1 60
```

Se han creado `train.clean.en` (parte inglesa) y `train.clean.es` (parte española) en en el directorio `train`. Las frases en los dos ficheros están alineados uno a uno.

```
~/TA/Practica1/Tarea> wc train/training.clean.*
  9500  94439 420353 train/training.clean.en
  9500  92361 499168 train/training.clean.es
 19000 186800 919521 total
~/TA/Practica1/Tarea> head -1 train/training.clean.es
'? le importar'ia darnos las llaves de la habitaci'on , por favor ?
~/TA/Practica1/Tarea> head -1 train/training.clean.en
would you mind giving us the keys to the room , please ?
```

## 1.5. Entrenamiento de los modelos de lenguage de salida (inglés)

Para entrenar los modelos de lenguage de inglés utilizaremos `train.clean.en` y el software SRILM:

```
~/TA/Practica1/Tarea> cd train; mkdir lm
~/TA/Practica1/Tarea/train> ngram-count -order 3 -unk -interpolate \
    -kndiscount -text training.clean.en -lm lm/turista.lm
```

En el directorio `lm` se ha creado el modelo de trigrama `turista.lm`. Más adelante es necesario disponer del camino absoluto del modelo de lenguaje.

```
~/TA/Practica1/Tarea/train> export LM=$PWD/lm/turista.lm
```

A continuación se presentan las primeras líneas de `turista.lm`

```
~/TA/Practica1/Tarea/train> less lm/turista.lm

\data\
ngram 1=516
ngram 2=2389
ngram 3=3244

\1-grams:
-2.246388       !       -1.196656
-2.562205       'Alvarez        -0.6283603
-3.022378       'Angel  -0.1209041
-3.184932       'Oscar  -0.1209041
-1.375849       ,       -1.377377
-0.9880823      .       -2.441463
-2.972268       </s>
-99     <s>     -2.356152
-3.363441       <unk>
-1.511654       ?       -1.91976
-2.972268       Aguilera        -0.5276494
-2.828413       Alicia  -0.1209041
-2.972268       Alted   -0.5276494
-2.972268       Amador  -0.1209041
...
```

## 1.6. Entrenamiento del modelo de traducción

En este apartado vamos a construir la tabla de segmentos (phrases) y los modelos de reordenamiento con los datos de entrenamiento obtenidos anteriormente: `training.clean.en` y `training.clean.es`. Para construir la tabla de segmentos, primero hay que alinear a nivel de palabra las dos partes del conjunto de entrenamiento y para ello utilizaremos el software GIZA++.

```
~/TA/Practica1/Tarea/train> export CPU=1
~/TA/Practica1/Tarea/train> $SCRIPTS_ROOTDIR/training/train-model.perl -root-dir work \
 -mgiza -mgiza-cpus  $CPU \
 -corpus training.clean -f es -e en \
 -alignment grow-diag-final-and -reordering msd-bidirectional-fe \
 -lm 0:3:$LM  -external-bin-dir $GIZA>& training.out &
```

Tarda unos minutos, podemos ver la evolución del proceso haciendo:

```
~/TA/Practica1/Tarea/train> tail -f training.out
```

Por otra parte, el directorio `work/model/` contiene los modelos de traducción:

```
~/TA/Practica1/Tarea/train>  ls -l work/model/

-rw-r--r-- 1 fcn elirf 461928 jul 24 15:38 aligned.grow-diag-final-and
```

```
-rw-r--r-- 1 fcn elirf 420878 jul 24 15:38 extract.inv.sorted.gz
-rw-r--r-- 1 fcn elirf 387406 jul 24 15:38 extract.o.sorted.gz
-rw-r--r-- 1 fcn elirf 438542 jul 24 15:38 extract.sorted.gz
-rw-r--r-- 1 fcn elirf  37053 jul 24 15:38 lex.e2f
-rw-r--r-- 1 fcn elirf  37053 jul 24 15:38 lex.f2e
-rw-r--r-- 1 fcn elirf    981 jul 24 15:38 moses.ini
-rw-r--r-- 1 fcn elirf 890537 jul 24 15:38 phrase-table.gz
-rw-r--r-- 1 fcn elirf 397807 jul 24 15:38 reordering-table.wbe-msd-bidirectional-fe.gz
```

## 1.7. Entrenamiento de los pesos del modelo log-lineal

Ahora obtendremos el conjunto de desarrollo y lo limpiaremos.

```
~/TA/Practica1/Tarea/train> wget --no-check-certificate \
     http://www.prhlt.upv.es/~fcn/Students/ta/dev.tgz
~/TA/Practica1/Tarea/train> tar xzvf dev.tgz
~/TA/Practica1/Tarea/train> mv dev/development.e* .
~/TA/Practica1/Tarea/train> clean-corpus-n.perl  development es en development.clean 1 60
```

A continuación obtendremos los pesos del modelo log-lineal mediante MERT.

```
~/TA/Practica1/Tarea/train> cd ..
~/TA/Practica1/Tarea> $MOSES/scripts/training/mert-moses.pl \
 train/development.clean.es train/development.clean.en \
 $MOSES/bin/moses train/work/model/moses.ini \
 --maximum-iterations=5 \
 --mertdir $MOSES/bin/
~/TA/Practica1/Tarea> mv mert-work/moses.ini  mert-work/moses.turista.ini
```

Tarda unos minutos (se ha limitado a 5 el número máximo de iteraciones). El modelo está en `mert-work/moses.turista.ini`.

## 1.8. Proceso de traducción

Utilizando un conjunto de test, utilizaremos el decodificador de Moses con los modelos obtenidos en las secciones anteriores. Los datos a traducir están en `test.es` y los que produce el descodificador estarán en `test.hyp`:

```
~/TA/Practica1/Tarea> wget --no-check-certificate \
     http://www.prhlt.upv.es/~fcn/Students/ta/test.tgz
~/TA/Practica1/Tarea> tar zxvf test.tgz
~/TA/Practica1/Tarea> cd test
~/TA/Practica1/Tarea/test> $MOSES/bin/moses \
 -f ../mert-work/moses.turista.ini < test.es > test.hyp
```

Si miramos alguna frase de entrada, generada por el traductor y alguna referencia:

```
~/TA/Practica1/Tarea/test> head -3 test.es

 por favor , desearía reservar una habitación hasta mañana .
```

```
 por favor , despiértenos mañana a las siete y cuarto .
 voy a marcharme hoy por la tarde .

~/TA/Practica1/Tarea/test> head -3 test.hyp

please I would like to book a room until tomorrow .
please wake us up tomorrow at a quarter past seven .
I am leaving today in the afternoon .

~/TA/Practica1/Tarea/test> head -3 test.en

 I would like to book a room until tomorrow , please .
 please wake us up tomorrow at a quarter past seven .
 I am leaving today in the afternoon .
```

## 1.9. Evaluación del proceso de traducción

Ahora calcularemos el BLEU del resultado comparando `test.hyp` con los datos de referencia `test.en`:

```
~/TA/Practica1/Tarea/test> $MOSES/scripts/generic/multi-bleu.perl -lc test.en < test.hyp
```

Se obtiene

```
BLEU = 92.12, 97.6/93.2/91.3/89.3 (BP=0.993, ratio=0.993, hyp_len=11700, ref_len=11786)
```

Es probable que el resultado de BLEU (92.12 en el ejemplo) que se obtengan pueda diferir en algún punto debido a ciertas componentes aleatorias de MERT.

## 1.10. Ejercicios

1. Probar el modelo obtenido en el apartado 6 sin ajuste de pesos.
2. Probar para valores más altos del número máximo de iteraciones del MERT (apartado 7).
3. Probar distintos valores de n-gramas (apartado 5).
4. Opcional: Probar MIRA para el entrenamiento de los pesos del modelo log-lineal (ver manual de Moses).
5. Opcional: Probar otras técnicas de suavizado (ver manual de SRILM).
6. Opcional: Probar moses monótono (ver manual de Moses).

# 2. Traducción basada en redes neuronales dinámicas

## 2.1. Introducción

El objetivo de esta práctica es construir sistemas de traducción basado en redes neuronales a partir de conjuntos de pares de frases bilingües. Concretamente estos sistemas están basado en el modelo neuronal Transformer. Para ello utilizaremos una herramienta (*opennmt-py*[1]) basada en *PyTorch*.

Al igual que *Moses*, *opennmt-py* permite entrenar modelos de traducción automática para cualquier par de lenguas, pero en este caso los modelos son redes neuronales Transformer. Por lo tanto, se necesita un corpus paralelo y una vez el modelo entrenado, *opennmt-py* proporciona un algoritmo de búsqueda eficiente que le permite encontrar eficientemente la traducción de mayor probabilidad.

### 2.1.1. Instalación

Para instalar *opennmt-py*, se asume que se tiene creado un directorio `TA` donde realizaremos las prácticas. Por simplicidad, asumimos que este directorio se encuentra ubicado en el home. De no ser así, bastará con editar `$TA`. De forma similar, asumimos que se desea instalar *opennmt-py* en el directorio `TA`. De no ser así, bastará con modificar `$INSTALLATION_PATH`.

```
~/TA$ wget https://raw.githubusercontent.com/PRHLT/OpenNMT-py\
/lab_sessions/full_installation.sh
~/TA$ chmod +x full_installation.sh
~/TA$ export INSTALLATION_PATH=~/TA
~/TA$ ./full_installation.sh ${INSTALLATION_PATH}
```

**Docker**

Alternativamente, *opennmt-py* puede ejecutarse desde Docker. Para ello, consultar la información disponible en:

```
https://github.com/PRHLT/OpenNMT-py/tree/lab_sessions/docker
```

**Google Colab**

*opennmt-py* también puede ejecutarse desde Google Colab. Para ello, consultar la información disponible en:

```
https://github.com/PRHLT/nmt-practical-session
```

### 2.1.2. Definición de variables

Para el correcto uso de *opennmt-py*, es necesario configurar las siguientes variables:

```
~/TA$ export INSTALLATION_PATH=/opt/opennmt-py
~/TA$ export NMT=${INSTALLATION_PATH}/NMT_TA
~/TA$ export PATH=${NMT}/miniconda/bin/:${PATH}
```

**En el caso de estar usando una instalación propia**, `$INSTALLATION_PATH` será la variable definida en el apartado 2.1.1.

[1] `https://github.com/PRHLT/OpenNMT-py/tree/lab_sessions`.

### 2.1.3. Descripción de la red

En el fichero `${NMT}/OpenNMT-py/config.yaml` está detallada la red que se va a utilizar y que se compone de:

- Tanto codificador como decodificador son Transformer de 64 neuronas.
- El tamaño del vector para codificar las palabras fuente es de 64.
- El tamaño del vector para codificar las palabras destino es de 64.
- La red tiene 2 capas.
- Hay una capa oculta *Transformer feed-forward* de tamaño 64.
- 2 cabezas de auto-atención.
- Otros parámetros de la red se encuentran en `config.yaml`.

## 2.2. Datos

Utilizaremos los mismos datos de entrenamiento, desarrollo y test de la práctica primera, pero en este caso no hace falta aplicar la herramienta de filtrado de *Moses*.

```
~/TA> mkdir Practica2; cd Practica2
~/TA/Practica2> mkdir data; mkdir data/EuTrans
~/TA/Practica2> cp ../Practica1/Tarea/train/training.e? data/EuTrans/
~/TA/Practica2> cp ../Practica1/Tarea/train/development.e? data/EuTrans/
~/TA/Practica2> cp ../Practica1/Tarea/test/test.e? data/EuTrans/
```

## 2.3. Entrenamiento del modelo de traducción neuronal

Antes de comenzar con el entrenamiento, será necesario construir el vocabulario:

```
~/TA/Practica2$  onmt_build_vocab -config ${NMT}/OpenNMT-py/config.yaml
```

Tras esto, el entrenamiento se inicia de la forma siguiente:

```
~/TA/Practica2$ onmt_train -config ${NMT}/OpenNMT-py/config.yaml
```

## 2.4. Proceso de traducción

Una vez entrenada la red, traduciremos de la forma siguiente:

```
~/TA/Practica2$ onmt_translate -model data/models/EuTrans_step_5000.pt \
-src ${NMT}/OpenNMT-py/dataset/EuTrans/test.es -output hyp.test.en \
-verbose -replace_unk
```

## 2.5. Evaluación de las traducciones

La evaluación se realizará del siguiente modo:

```
~/TA/Practica2$ sacrebleu --force -f text data/EuTrans/test.en < hyp.test.en
```

El BLEU obtenido es de 96. Como en la práctica 1, el resultado que se obtenga pueda diferir en algún punto.

## 2.6. Ajuste de parámetros

Para poder modificar los parámetros de la red, es necesario realizar una copia local del fichero `config.py`:

```
~/TA/Practica2> cp ${NMT}/OpenNMT-py/config.yaml .
```

Tras esto, se procederá a modificar los parametros deseados que están definidos en la copia local que acabamos de crear. Una vez definidos los parámetros deseados, bastará con cambiar la ruta del fichero de configuración tanto a la hora de construir el vocabulario como de entrenar el modelo.

## 2.7. Ejercicios

1. Probar otros tamaños de representación de las palabras fuentes y destino (word embeddings).
2. Opcional: Probar a variar el número de capas del codificador y decodificador.
3. Opcional: Probar otros algoritmos de aprendizaje: *SGD*, *Adagrad*, *Adadelta*, etc.
4. Opcional: Probar redes recurrentes.

# 3. Memoria

La memoria debe estar formada por una introducción, una descripción del trabajo experimental realizado, resultados obtenidos, conclusiones y bibliografía. Los resultados deben presentarse en forma de tablas o gráficas y solo del BLEU obtenido. La notación matemática debe estar correctamente expresada (sumatorios con límites bien establecidos, subíndices y supraíndices correctamente expresados, etc). El nombre del documento debe estar formado por los apellidos y el nombre del alumno/a sin acentos ni espacios (e.g., *CasacubertaNollaFrancisco.pdf*). El trabajo mínimo (hasta 8 puntos) **debe incluir una descripción del trabajo realizado en las sesiones de prácticas y en los ejercicios** no opcionales. Mediante la realización correcta de ejercicios opcionales se puede obtener hasta 2 puntos.

# 4. Bibliografía


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. Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press. 2010.

3. Philipp Koehn. *Neural Machine Translation*. arXiv:1709.07809v1. 2017.

4. Philipp Koehn. *Neural Machine Translation*. Cambridge University Press. 2020.

5. Kishore Papineni, Salim Roukos, Todd Ward, Wei-Jing Zhu. *BLEU: a method for automatic evaluation of machine translation*. 40th Annual meeting of the Association for Computational Linguistics. 2002.

6. Guillaume Klein, Yoon Kim, Yuntian Deng, Vincent Nguyen, Jean Senellart, Alexander M. Rush. OpenNMT: Neural Machine Translation Toolkit. Proceedings of 13th Conference of the Association for Machine Translation in the Americas (AMTA), 2018,

7. Andreas Stolcke. *SRILM - An Extensible Language Modeling Toolkit*. 7th International Conference on Spoken Language Processing, 2002.

8. Andreas Stolcke, Jing Zheng, Wen Wang, Victor Abrash, *SRILM at Sixteen: Update and Outlook*. IEEE Automatic Speech Recognition and Understanding Workshop, 2011.

9. *SRILM - The SRI Language Modeling Toolkit*
   `http://www.speech.sri.com/projects/srilm/`