Text Mining and Natural Language Processing

2022-2023

Manuel Dellabona 503086

Alberto Roggero 509588

EuroTranslation

# 

# 

# Introduction

We built an array of NMT models to translate across multiple European languages and, in the process, we compared the results with the ones obtained from our source paper [2], which used the SMT approach.

A neural machine translation (NMT) system is a type of language processing technology that uses neural networks to translate text.

The neural translator is trained on a large amount of bilingual text data, known as parallel corpora, which consists of source sentences and their corresponding translations.

Neural translators achieved significant improvements in translation quality compared to traditional statistical machine translation (SMT) approaches.

NMT models require far more data to be trained but when they have it they achieve greater accuracy and a better handling of context because of the way information is represented.

Our task was composed of two main goals:

* Build a fine-tuned model capable of good translations from Italian to English.
* Build a standard architecture to utilize in both directions between all the pairs formed by our 4 chosen languages (Italian, English, German and Spanish) and thus be able to consistently compare the obtained results.

Every model was then evaluated through BLEU: a standard evaluation tool for translations based on n-grams. We used the same evaluation method as the source paper to keep everything consistent.

Differences emerged between scores inside a specific pair of languages (back and forth) and, of course, between different languages. This led to some interesting hypotheses about the structure and richness of different languages.

# Data

The dataset used is a subset of a corpus of parallel text in 21 languages extracted from the proceedings of the European Parliament, which are publicly available on the official website. This corpus has found widespread use in the NLP community.

The corpus, originally composed of 11 languages, was retrieved from the web by the authors of our source paper “Europarl: A Parallel Corpus for Statistical Machine Translation”, *Philipp Koehn*, MT Summit 2005 who used it for building SMT translators several years ago. The corpus has since been constantly updated and languages have been added.

The dataset chosen was [version8](https://opus.nlpl.eu/Europarl.php) [1] of this corpus and was optimal for the task for multiple reasons.

First of all the magnitude of the set was perfect for training an NMT.

Moreover it represented a fair, standard starting point for the training of parallel models thanks to its content and structure.

The content being the exact same for every official language in the EU prevents possible interference given from diversified corpuses for different languages while permitting a vast range of language choices.

The coherent and tightly-organized structure ensures very good sentence mapping, as a matter of fact sentence alignment is usually a hard problem, but in this case it is simplified by the native division in paragraph-aligned format, as the authors stated.

The data we used comes from a dataset of almost 100M total words and 2M paired sentences for each couple of languages

| **EN-IT** | 1,907,207  sentence pairs | 99.41M words |
| --- | --- | --- |
| **EN-DE** | 1,961,119  sentence pairs | 94.61M words |
| **EN-ES** | 1,962,064  sentence pairs | 102.51M words |
| **IT-DE** | 1,832,989  sentence pairs | 90.95M words |
| **IT-ES** | 1,880,982  sentence pairs | 99.34M words |
| **DE-ES** | 1,848,294  sentence pairs | 93.67M words |

The corpus of each language is then divided in training, development, and test sets during the processing step, in which 8000 sentences are set apart. Some are used in the development set to calculate the validation accuracy and perplexity at the end of each 500 training steps. The remaining part is used in the testing set to calculate the BLUE score on unseen data.

# Methodology

Regarding the methodology used in this project we shall identify two main steps, the processing of the data and the training of the models.

The first part of the process is composed of the following sub-steps:

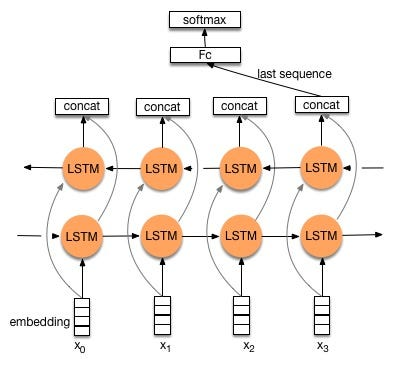
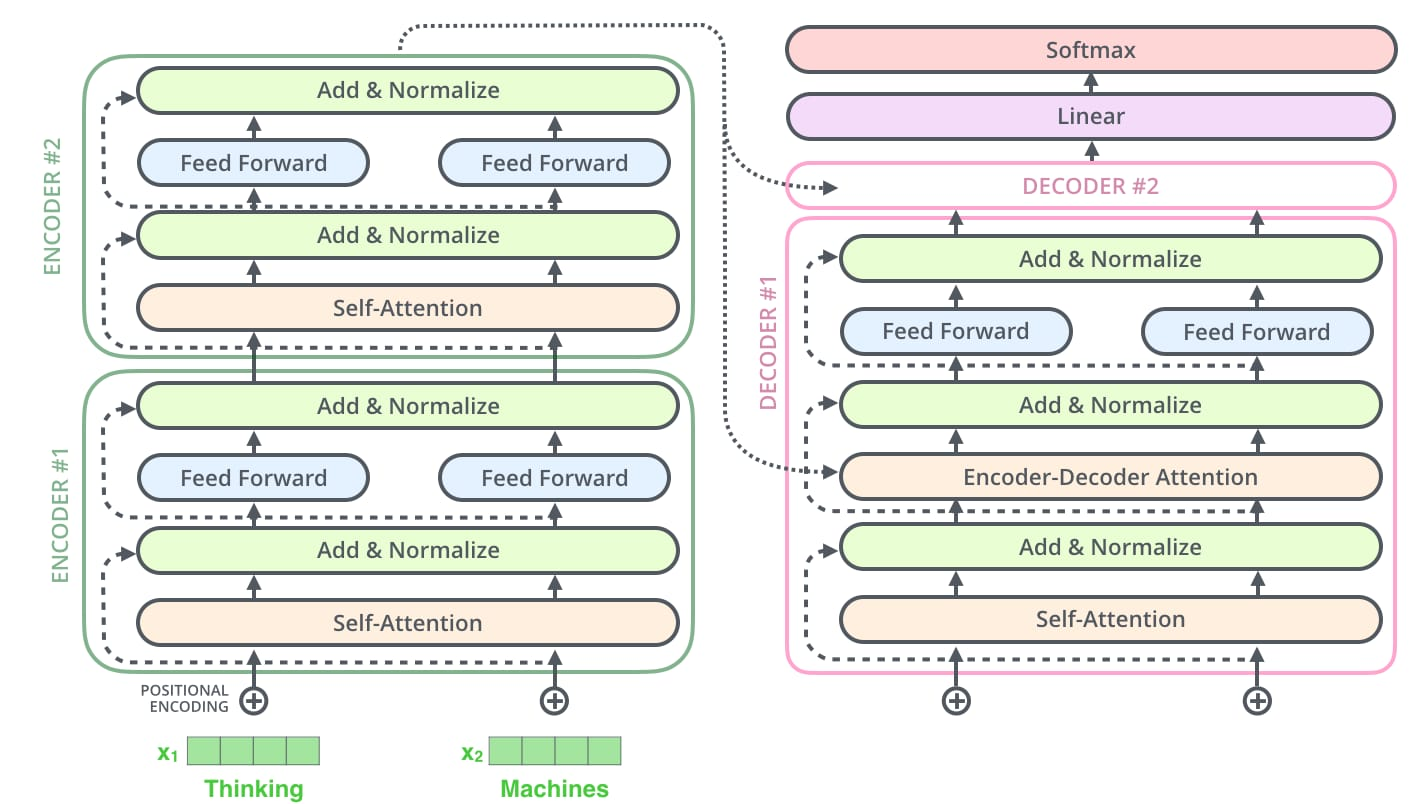
1. After unzipping the parallel data of the pair of languages that has to be processed, we apply a filter in order to delete: empty rows, duplicates, source-copied rows, too long source or target rows and HTML formatting.
2. We train a unigram model in order to subword each file.
3. We apply the model to the source and target text.
4. We split the resulting file in three segments, representing training (main corpus), development (4000 sentences) and test (4000 sentences) sets

The training of the model is composed of the following sequence:

1. We run the script for creating the configuration file which specifies the type of model and its parameters, later used by the library openNMT to build the model.
2. Then we build the vocabulary of all the words in the training dataset.
3. At this point we can start the training of the NMT model.

In particular, since in this project we would like to describe two different tasks we need to discuss how we proceeded in each of those.

3.1 Translator model

Starting with the goal of building a neural translator from Italian to English, we tried what we consider to be a simple model to become conscious of the practical difficulty of the task. So we set-up a model composed by the RNNEncoder that is responsible for encoding the input sequence.It utilizes an embedding layer to map each token in the input sequence to a continuous vector representation. The embeddings are then fed into a bidirectional LSTM, which enables the model to capture both forward and backward context information. The output of the RNNEncoder is a context vector that summarizes the input sequence. The model is also composed by the InputFeedRNNDecoder that takes the context vector from the RNNEncoder as input and generates the output sequence step by step. Similar to the RNNEncoder, the InputFeedRNNDecoder employs an embedding layer to represent the tokens in the output sequence. The decoder incorporates stacked LSTM layers. Additionally, the decoder utilizes an attention mechanism, known as GlobalAttention, to focus on relevant parts of the input sequence during each decoding step. Lastly the generator is a linear layer that takes the final hidden state of the decoder and produces the probability distribution over the vocabulary. It maps the hidden state to the output vocabulary size, allowing the model to generate diverse sequences. 

The results of our experiments indicate that the proposed architecture sequence-to-sequence deep learning model, described earlier, underfits the data. The model's limited capacity to capture complex patterns in the input and output sequences contribute to this underfitting behavior. To mitigate underfitting, we should focus on increasing the model's complexity.

Metrics:

*Step 5000  
Validation perplexity: 152.792  
Validation accuracy: 24.1202  
BLEU: 2.643207991873375*

As said before, we selected a more complex model, in particular we chose to design an encoder-decoder structure with transformers, a structure that leverages self-attention and position-wise feed-forward networks to capture global and local dependencies in the input and output sequences.

The encoder consists of an embedding layer, followed by a Transformer encoder layer. The latter employs multi-headed self-attention mechanisms and position-wise feed-forward networks to process the input sequence and capture its contextual information.

This is followed by the decoder that also comprises an embedding layer with positional encodings and a stack of transformer decoder layers. Similar to the encoder, the decoder utilizes multi-headed self-attention. Additionally, the decoder employs a context attention mechanism to attend over the encoder's output during the decoding process. This allows the decoder to access relevant information from the input sequence while generating the output to be passed to the generator, which produces a probability distribution over the target vocabulary.

Metrics:

*5000 step  
Validation perplexity: 16.925  
Validation accuracy: 66.7646*

The first parameter to focus on in building an efficient model is actually identifying the complexity needed to represent the problem, so in this case we have decided to increment the layers of the decoder and the encoder to 2 and 3, noting some significant improvements.

Then, as further advancement we tried to optimize the trade-off between the complexity of the model and the training time and resources involved. In fact by halving the hidden size of the layers after 5000 training steps we could notice there was not a noticeable loss in the accuracy and in the perplexity of the model but with the increasing of the training steps the model could not compete with the ones with hidden size 512.

Metrics:

*5000 step  
2 layer  
Validation perplexity: 16.925  
Validation accuracy: 66.7646  
3 Layer  
Validation perplexity: 17.4307  
Validation accuracy: 66.1826  
4 layer  
Validation perplexity: 17.1091  
Validation accuracy: 66.3933*

This led us to try with architectures with hidden size 1024 but the complexity reached was too high to make experiments with our resources.

Metrics:

*2500 step (extremely time consuming)  
Validation perplexity: 21.4756  
Validation accuracy: 62.774*

Continuing with the optimization we tried a model with a bigger learning rate (increased from 1 to 2) but with more warm-up steps (increased from 1000 to 2000), this can lead to a faster convergence but can also cause the model to converge too quickly to a suboptimal solution.

In this case it seemed not to have an impact in our favor, probably due to the optimizer “adam” and the variable learning rate.

So we came up with the decision of using the architecture with 3 layers as decoder and encoder and 512 as hidden size, this will be implemented as a starting point for the comparison between all the language pairs.

Metrics:

*5000 step  
Validation perplexity: 16.7742  
Validation accuracy: 66.8346  
BLEU: 35.05880515364017  
20000 step  
Validation perplexity: 13.6078  
Validation accuracy: 70.0763  
BLEU: 37.692925963718515*

3.2 Standard model for comparison task

To pursue the comparison task between the translator we selected a standard architecture to build all the NMTs and calculate the BLUE scores, in particular this architecture will be the one proposed before with transformers, 3 layer for each decoder and encoder, 512 for the hidden size, the learning rate set as 2, the adam as the optimizer and 2000 warm-up steps.

# Results

The reference we used to evaluate our results were the BLEU scores obtained by the author of the paper.

|  | **EN** | **IT** | **DE** | **ES** |
| --- | --- | --- | --- | --- |
| **EN** |  | 25,3 | 17,6 | 30,1 |
| **IT** | 27,8 |  | 16,9 | 34,0 |
| **DE** | 25,3 | 21,3 |  | 25,4 |
| **ES** | 30,5 | 32,3 | 18,2 |  |

Europarl’s BLEU’s

We compared their scores both with our best scores,

|  | **EN** | **IT** | **DE** | **ES** |
| --- | --- | --- | --- | --- |
| **EN** |  | 34,2 | 30,0 | 44,1 |
| **IT** | 37,7 |  | 24,6 | 36,6 |
| **DE** | 36,7 | 28,1 |  | 34,0 |
| **ES** | 44,1 | 32,0 | 27,0 |  |

Our best BLEU’s (roughly 20000 steps)

immediately noticing a significant improvement (with the exception of the Italian-Spanish pair whose case will be later discussed)

|  | **EN** | **IT** | **DE** | **ES** |
| --- | --- | --- | --- | --- |
| **EN** |  | 8,9 | 12,4 | 14,0 |
| **IT** | 9,9 |  | 7,7 | 2,6 |
| **DE** | 11,4 | 6,8 |  | 8,6 |
| **ES** | 13,6 | -0,3 | 8,8 |  |

Best BLEU’s improvement

as well as with our scores from early stages of the training, which also revealed a clear improvement compared to the use of STM.

The poor results of the Italian-Spanish reflect poorly trained models. This is because we faced some technical difficulties during the use of Google Colaboratory which often stopped our training mid-execution thus leading to early-stopped, unconverged models (5000 steps scores are instead fully reliable).

We managed to rerun most of the models until they reached convergence but this wasn’t the case with IT-ES.

Setting aside faulty results, the rest of them immediately outlined how NMT models have far outperformed old SMT techniques even from early stages of training.

|  | **EN** | **IT** | **DE** | **ES** |
| --- | --- | --- | --- | --- |
| **EN** |  | 31,3 | 27,2 | 41,6 |
| **IT** | 35,1 |  | 21,9 | 35,0 |
| **DE** | 34,4 | 25,5 |  | 31,3 |
| **ES** | 41,2 | 30,5 | 24,4 |  |

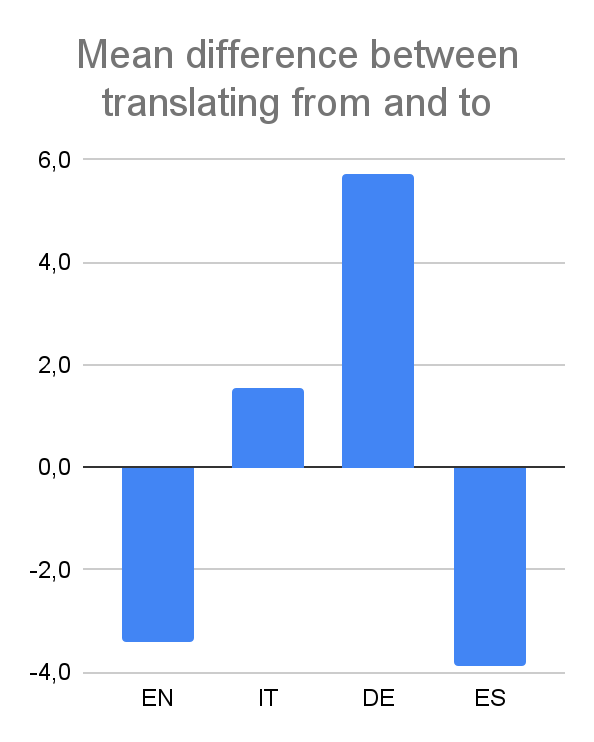
Our initial BLEU’s (5000 steps)

One thing that caught our eyes when looking back at early-stage improvements was that even there, on an even field, the couple IT-ES seemed to perform poorly (ES → IT direction performed even worse than STM).

|  | **EN** | **IT** | **DE** | **ES** |
| --- | --- | --- | --- | --- |
| **EN** |  | 6,0 | 9,6 | 11,5 |
| **IT** | 7,3 |  | 5,0 | 1,0 |
| **DE** | 9,1 | 4,2 |  | 5,9 |
| **ES** | 10,7 | -1,8 | 6,2 |  |

Initial BLEU’s improvement

What we think this is due to is the structural and lexical similarity between Italian and Spanish that permitted the original statistical machine translator to perform quite well, thus making it harder to improve the score (especially if you don’t achieve full training).

From the results we extracted some more statistics and the one we found most interesting was the average difference between the scores a specific language got while translating to it and from it. This produced a metric showing how easy it is to translate from a specific language into other languages. 

(the higher the easier it is to translate from the language as opposed to into it)

Following what already pointed out in the paper this metric can be correlated with the language’s morphological richness (German is a very rich language whereas English and Spanish are less so).

We found, sometimes significant, differences between BLEU scores, even going back and forth between the same two languages.

While inter-pair scoring differences can be quite obviously explained through language families and similarities, intra-pair delta is less straightforward to justify.  
This again probably depends on language informational richness (it is easier to translate from a rich language into a poor one)

# Conclusion

# Recap

In this project, we built a set of Neural Machine Translation (NMT) models to translate between multiple European languages and compared the results with a source paper that used Statistical Machine Translation (SMT) approaches. Our goal was to build a fine-tuned model for Italian to English translation and establish a standard architecture for comparing translations among four chosen languages.

For the Italian to English translation task, we initially experimented with a simple model consisting of an RNNEncoder and InputFeedRNNDecoder. However, this model underfit the data due to its limited capacity to capture complex patterns. We then moved to a more complex model using an encoder-decoder structure with Transformers, which proved to be more effective.

To optimize the trade-off between model complexity and training resources, we conducted various experiments, including reducing the hidden size of layers and adjusting learning rates and warm-up steps. These experiments helped us find a balance between model performance and training efficiency.

The results of our experiments showed significant improvements in translation quality compared to the SMT approach used in the source paper.

The findings highlight the power of neural translators and the importance of considering language characteristics when developing and evaluating translation models.

* 1. Who did what:

Manuel Dellabona: Conducted data preprocessing, implemented the initial model architecture, trained models and performed experiments for hyperparameter tuning.

Alberto Roggero: Trained models, analyzed the results, compared the scores with the source paper, and discussed the linguistic considerations and implications of the findings.

* 1. Most Difficult Steps:

The most challenging steps of the project included understanding and implementing the complex model architecture with Transformers, fine-tuning the hyperparameters to optimize the models, and analyzing the linguistic factors influencing translation quality.

Working with the limitations of Google Colab posed challenges during the project. The restricted computational resources, limited session duration, and frequent runtime disconnections made training and fine-tuning the NMT models more difficult. Longer training times, difficulties in experimenting with hyperparameters and model architectures, and the need for manual intervention to resume interrupted sessions were some of the issues faced. We also tried to locally train models but lacked the graphic memory to get any advantage over online training.  
Despite the limitations, the project successfully achieved its goals with additional effort and adaptability within the constraints of Google Colab.

* 1. Future improvements

The majority of the improvements can be done working on the parameter tuning of the models, in particular without the conflict between restrictions imposed by google colab and the complexity of the openNMT-py calculations, should be applied an automatic parameter tuning algorithm, that optimizes the search space.

One fascinating solution could also be implementing genetic algorithms.

References

[1] “Europarl.” OPUS corpus.   
<https://opus.nlpl.eu/Europarl.php>.

[2] Koehn, Philipp. “Europarl: a parallel corpus for statistical machine translation.” ACL Anthology.   
<https://aclanthology.org/2005.mtsummit-papers.11.pdf>.