Text Mining and Natural Language Processing

2022-2023

[Guirges Beshoy (502456), Totis Elisa (506950), Peppoloni Vittoria (502861)]

*[Es <-> It]*

Introduction

Our project is about Machine Translation, which is a subfield of natural language processing (NLP) that focuses on developing algorithms and models to automatically translate text or speech from one language to another. The goal of machine translation is to enable communication and understanding across different languages by leveraging computational methods.

We take as datasets two files containing the same sentences respectively in Italian and Spanish. The idea is to train a model for the translation of sentences and then provide an input cell to the user to translate a custom input.

Thanks to a classification task, the program recognises itself the language of the given sentence, and provides the user three different messages according to the situation:

* If the input language is Spanish, the program tells the user that it is going to run the translation from Spanish to Italian
* If the input language is italian, the program tells the user that it is going to run the translation from Italian to Spanish
* If the input language is neither Italian nor Spanish, the program informs the user that it is not able to translate

Once this classification is correctly computed, it starts the real translation: the sentence is preprocessed, so the program applies subwording and tokenization; then, thanks to the trained model, the translation is performed and provided to the user.

More precisely, in order to make the program more useful and also to make the translation “bidirectional”, we provide two different solutions: the first one is to build two separate models in which we modify the source and the target files accordingly; the second one is to create an implemented model that automatically makes the translation possible in both directions, by tagging the training sentences, however this approach had really poor performances.

Data

We download the ‘wikimatrix’ dataset from Opus at this [link](https://opus.nlpl.eu/download.php?f=WikiMatrix/v1/tmx/es-it.tmx.gz). The dataset is composed of 2.2M sentences, more precisely with 298.2M Spanish tokens and 381.7M Italian tokens.

The wikimatrix is a project implemented by facebook company, in which they managed the collection of data from all the wikipedia pages combining the same sentences in different languages.

The dataset contains different pairs of sentences translated in both languages in this form: [Sus treinta artículos debían ser aprobados uno por uno.] [I suoi trenta articoli dovettero essere approvati uno per uno.]

Methodology

Preprocessing

The first step of our project involves preprocessing. Here we search for what we can apply to our dataset to make it more manageable and easier to work with.

At the very beginning we perform tokenization. It is the process of breaking down a text into smaller units called tokens. In natural language processing, tokens are considered to be words, subwords, or even characters, depending on the choices of the tokenization approach. It helps convert unstructured text into a format that can be processed.

In our project we perform the subword tokenization. Instead of treating each word as a single token, this approach breaks down words into meaningful subunits, which can be characters or character sequences. This helps the model to handle rare or unseen words more effectively and capture morphological variations.

The specific algorithm we use for dividing words into subword units is Byte Pair Encoding (BPE). The idea is that the algorithm iteratively merges the most frequent character unions (starting from single characters), allowing to discover recurring subword units and creating a vocabulary that captures both individual characters and commonly occurring subword units.

BPE can be divided in two parts: it starts with a token learner which takes a raw training corpus and induces a vocabulary that is the one that the tokenizer will try to map things into; then, after the token learner, we have the token segmenter which takes a test sentence and it tokenizes it according to that vocabulary.

Another approach that we try for subword tokenization is the unigram model: it is a statistical approach that assigns probabilities to subword units based on their occurrence frequency in the training data. It aims to minimize the total number of tokens in the vocabulary while maximizing the coverage of training data. However, after measuring the performances of both techniques, the result is that BPE works better for our dataset. In particular, the BPE is optimal for handling the Out-of-vocabulary (OOV) problem, which occurs when the model encounters a word or subword unit that it has not seen during training.

The other important technique that we apply is filtering: it refers to the process of removing certain types of data or elements from the dataset based on specific criteria or rules.

Later on, thanks to a few python lines of code, we perform the preprocessing. An example of steps that are usually computed for dataset of text type are:

* Case normalization: upper and lower cases needs to be normalized before being processed
* Punctuation removal: if the input sentence contains punctuation, the code returns the text without it
* Lemmatization: reduction of words to their base or canonical form, known as the lemma. The lemma represents the dictionary form or the root of a word, which captures its core meaning
* Handling contractions: if input sentences are written using contraction, the output will be written in an extended version
* Handling special characters: characters that are not numbers or letters, are transformed in a standardized format

The goal of all these preprocessing steps is to ensure that the text is in a standardized format that can be effectively processed by the machine translation system, without losing important linguistic information or introducing inconsistencies.

Data splitting

The splitting of the dataset usually generates three subsets:

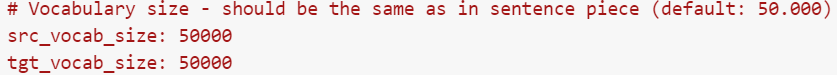
* The training set: it is the largest portion of the dataset. It is used for training the model in order to reach the best translation performance. More precisely, starting from parallel sentences in the source and target languages, the model can learn and understand the relationship between them.
* The development dataset: used to run regular validations during the training to help improve the model parameters.
* The testing dataset: a holdout dataset used after the model finishes training to finally evaluate the model on unseen data. It helps in assessing how well the model generalizes to new examples.

Model configuration

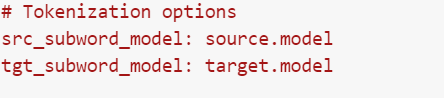
We create a single model both for the translation from spanish to italian and from italian to spanish. However, after checking performances, we realized that is better to keep 2 distinct models, since colab doesn’t allow us to perform more than 15.000 training steps due to time limitations on the GPU. So, we decide to provide a less optimized solution, but with much higher accuracy.

In order to define the model, we create a configuration YAML file, in which all hyperparameters are set:

* Data: in this first section we define the paths to the training and validation datasets. The source (input) and target (output) files are specified for both data.
* Vocabulary: in this section it is specified the path of the vocabularies created. The vocabulary contains the mapping between tokens and their numerical representations.



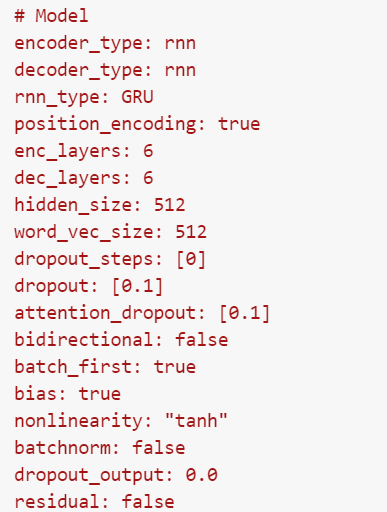
* Tokenization options: it specifies the path of source and target subword models generated by tools like SentencePiece.



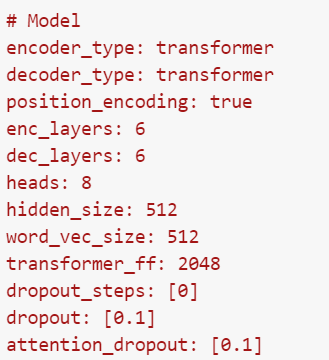
* Stop training: it decides to stop the iterations after n validations without improvement.
* Training and optimization parameters: this section defines various parameters related to training and optimization. It includes settings like early stopping criteria, steps for saving checkpoints, random seed, training steps, validation steps, warm-up steps, reporting frequency, number of GPUs, batching options, and optimization settings such as learning rate, decay method, Adam optimizer parameters.

Finally, we define our model by introducing the type of encoder and decoder.

Our first model utilizes a recurrent neural network (RNN), specifically the GRU (Gated Recurrent Unit), a neural architecture designed to capture long-term dependencies in sequential data.



However, we want to evaluate also a different kind of model to compare them. We try to use a transformer, with the following hyperparameters:

****

* Encoder\_type and decoder\_type: they specify the type of encoder and decoder architecture used. In our case it is set to ‘transformer’
* Position\_encoding: it determines whether positional encoding is applied to the input sequences. It helps the model capture the relative positions of words in the input sequence
* Enc\_layers and dec\_layers: they define the number of layers in the transformer encoder and decoder
* Heads: it determines the number of attention heads in the multi-head attention mechanism used in the transformer
* Hidden\_size: it specifies the size of the hidden state in the transformer model
* Word\_vec\_size: it defines the size of the word vector (word embedding) used in the model
* Transformer\_ff: this parameter determines the size of the feed-forward layer in the transformer
* Dropout\_steps: it specifies the dropout rate applied to the output of each transformer layer during training
* Dropout: it determines the dropout rate applied to the output of the fully connected layers in the transformer
* Attention\_dropout: it specifies the dropout rate applied to the attention weights in the transformer

What we find out is that it is better to use the transformer for several reasons. The first one is that, by measuring the performances of the two different models, the one with the transformer turns out to have the highest scores. But the decision is also based on some theoretical assumptions:

* Architecture: unlike RNNs, the transformer architecture takes the input as a single embedding, rather than processing the words sequentially. This allows the model to include more attention mechanism, since the model can compute the dependencies between all the words simultaneously. As a result, this will increase the highlighting of the dependencies between words.
* Parallelization: even if RNNs are very efficient and useful in processing sequential data, they are not able to compute them in a parallel way; the transformer, instead, will parallelize all the computations making the whole learning process faster.
* Long-term dependencies: thanks to its architecture, the transformer is better at capturing long-term dependencies, while an RNN, after processing a certain amount of inputs, will lose the older information.

Training

One important thing to mention is that we train the model using the OpenNMT library, an open-source toolkit for neural machine translation.

The actual training of the model is an expensive task in terms of time and resources. In fact, as explained in the next section, it is one of the most difficult aspects of the design of our project.

User prompt

The final step of our project is to furnish the actual possibility for a user to insert a sentence to be translated. Thanks to a Python script we manage to detect the language of the inserted sentence. We rely on the FastText library which is very efficient in the detection of a sentence language.

The if statement allows the program to run the translation from Spanish to Italian or vice versa.

Result

We rely our evaluation on word-based overlap metric called BLEU (BiLingual Evaluation Understudy) that measures the similarity between the translation generated by the machine and a human reference translation, and analyzes the overlap of n-grams between them.

It focuses on two components: the n-gram precision and the brevity penalty. The first one, calculates the precision of the n-grams thanks to a comparison done between the number of n-grams in the machine translation that also appear in the reference translation. In essence, precision is calculated as the ratio of matching n-grams over the total number of n-grams in the machine translation. The second component includes a brevity penalty that tries to handle the length difference between the machine translation and the reference translation.

Another good idea is to plot the accuracy variation during training, and we manually managed it, by collecting the accuracies at each validation step.

Finally, we used the Tatoeba challenge corpus. Its sets are used as benchmark in machine translation, and it has more than 400 languages, it is written and maintained by a community of volunteers through a model of open collaboration. Individual contributors are known as Tatoebans. It is run by Association Tatoeba, a French non-profit organization funded through donations.

We have a BLEU on the Tatoeba that is 41, which is relatively good according to what said in the [google documentation](https://cloud.google.com/translate/automl/docs/evaluate?hl=it).

On our test set we obtained a BLEU of about 39, which is still good.

Conclusion

To sum up, we build a project that can be useful and can have real world applications: If we need to know how to say something in spanish to have a conversation with a friend of us, or if we need to understand what the mexican chef is telling us about its meal. It will recognize the input language, tokenize it and provide an appropriate translation with a certain accuracy, thanks to the previously trained model.

We figured out that writing a project is a dynamical process, hence we decided that everyone needed to be involved in each step of the program. We wanted to be sure that every line of code was clear to everyone in the group. This allowed us to interact with each other and give suggestions for finding the best possible solutions to arised problems.

Writing the report was useful to recap all the steps done and to check if everything worked properly.

As mentioned above, the main difficulties that we had to face were time and complexity: the running time of our whole project was really high. The provided GPU in Google Colab was not sufficient to support all the steps in the training that we needed to achieve. In fact, we gave as hyperparameters the number of iterations according to the size of our dataset, which is pretty large, but the running stopped at a very early stage, for the colab time limitations. To try to solve this problem we decided to run everything on a desktop pc with a dedicated NVIDIA GPU which was slower than the colab one, but at the end it was too slow that to achive a good model it would have required weeks of no stop running.

The time that it takes to run is a huge limitation, because for computing the evaluation of the performances we always needed to wait for the model to reach a certain training. So modifying and manipulating the program required a long period of time, and we were always limitated from that.