Deep Comedy Generator

Riccardo Cozzi

Alessandro Liscio

Pietro Fanti

Novembre 2021[[1]](#footnote-0)

# Introduction

In September exactly 700 years ago, Dante Alighieri died in Ravenna. Today he is considered the father of Italian language and he is world-wide famous for his *Commedia*, later renamed *Divina Commedia*.

On the occasion of this anniversary we have tried to design an artificial neural network able to generate lines in Dante’s style. Since previous attempts struggled reaching satisfying results, for this project we have tried to simplify the task and we focused on the last stressed syllable of each line, which in an *hendecasyllable* is the tenth.

With this simplified task we have been able to obtain good results with an implementation of a transformer very similar to the *vanilla* one. However an *ad-hoc* customized architecture is probably needed to make the model able to generate more complex metric structures in a solid way.

# Problem description

*Divina Commedia* is an Italian narrative poem composed of 14233 lines grouped in *triplets*, also called *tercets*, which are groups of three lines. Each line is an *hendecasyllable*, i.e. a line, usually of 11 syllables, which satisfies the following criteria:

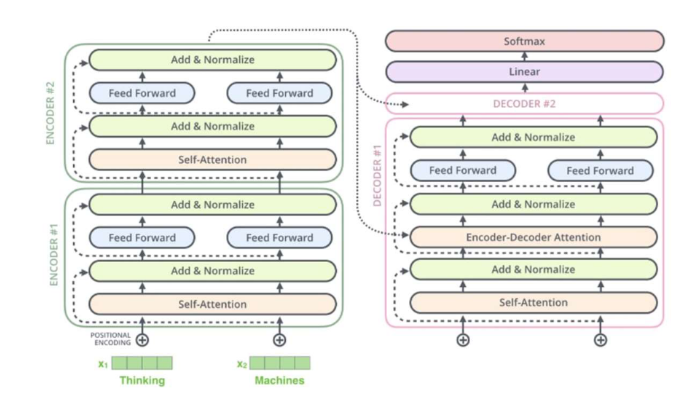
* It has the last tonic accent on the 10th syllable;
* It has a *secondary* tonic accent on the 4th (*hendecasyllable a minore*) or on the 6th (*hendecasyllable a maiore*) syllable;

Since previous works on this project failed in obtaining a good ratio of well-formed hendecasyllables, this time we have tried to simplify the problem considering as a hendecasyllable each line that satisfies just the first criterion, which is the most important.

To further simplify the problem, we have also decided to not consider other metrical structures, like the *cesura* and the rhyme structure.

Beside maximizing the ratio of lines which satisfy the first criterion, which from now on is called *hendecasyllables correctness*, we also took in account other metrics important for a basic but still acceptable generation. Indeed we have been interested in keeping the *plagiarism* as low as possible, while maximising the number of different words generated by the model and the *words correctness* (the ratio of words generated by the model which are present in the original corpus). All these metrics are better defined further in the report.

# Transformer



The Transformer is a model proposed by Google which is solely based on attention mechanisms, a decision that granted him not only a superior quality of the results, but also a highly parallelizable system and thus requiring significantly less time to train.

Recurrent neural networks, long short-term memory and gated recurrent neural networks have previously been established to be state of the art approaches in sequence modeling problems, but the problem with these models is that, due to the fundamental constraint of sequential computation, they all perform worse once the sequence length increases and the model’s computation are not parallelizable.

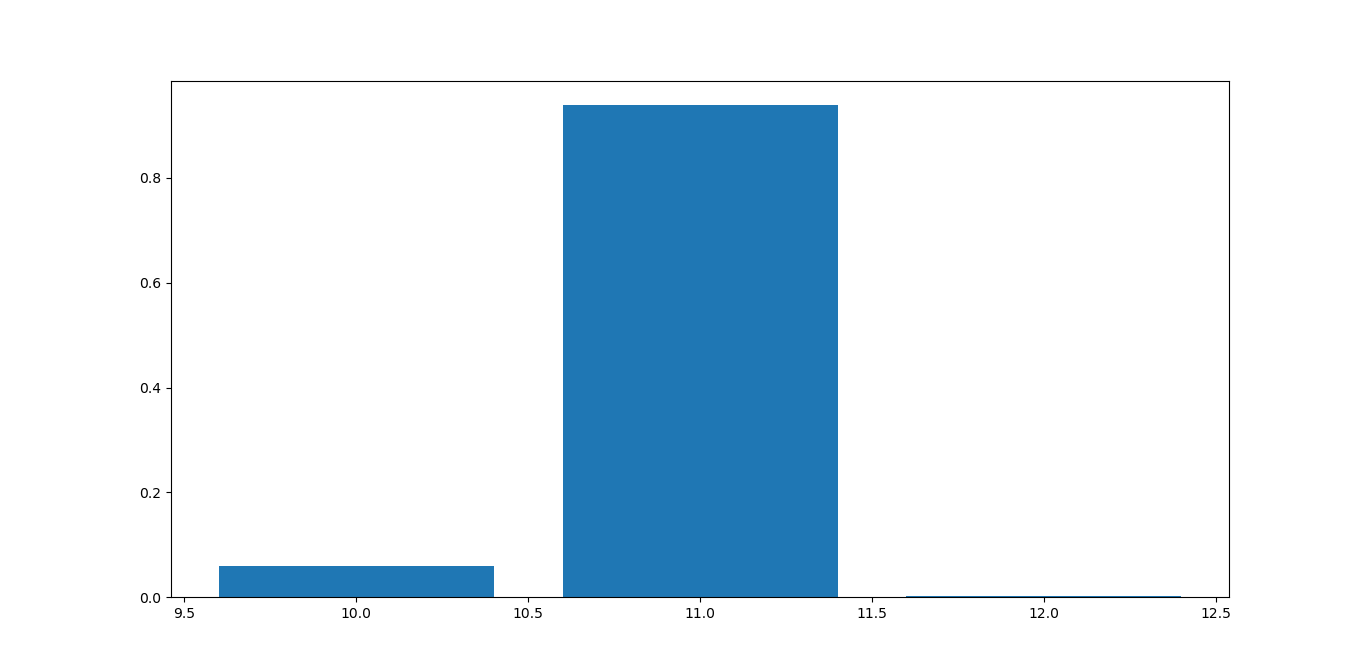
Attention mechanisms allow the modelling of dependencies between different positions of the sequence without regard to their distance in the input or output sequence in order to draw global dependencies and patterns between input and output.

In this project we have implemented the vanilla transformer architecture, which is composed by a stack of encoder layers followed by a stack of decoder layers. Both encoder and decoder layers exploit the self-attention mechanism (masked self-attention for the decoder layers) followed by normalization and point-wise feed-forward neural networks. More precisely the model has been instantiated with the following parameters:

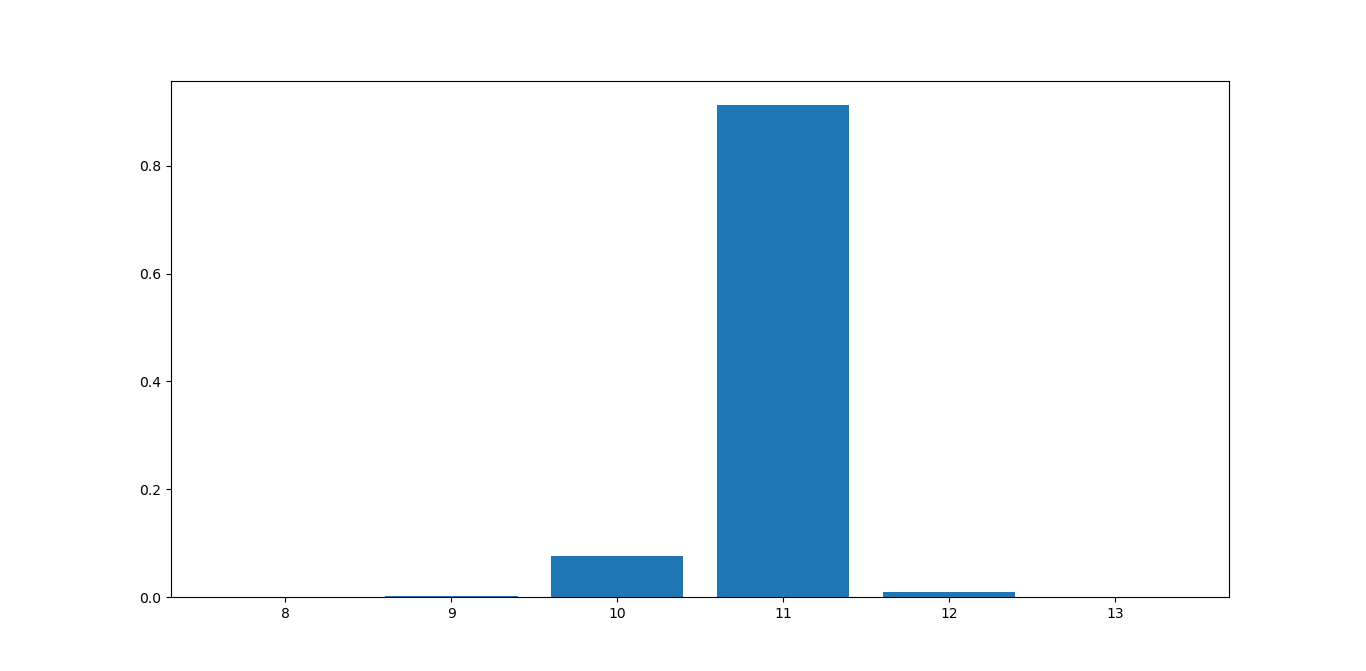
* 5 encoder layers
* 5 decoder layers
* 4 attention heads
* 512 d\_model (size of the tokens embeddings)
* 512 dff (number of units in the feed-forward layers)
* 0.2 dropout rate

# CNN-based syllabification

One of the main concerns of this work is the syllabification of the corpus. Since our aim was to generate hendecasyllables, we needed to consider the text as sequences of syllables. Although we already disposed of a syllabified dataset, in order to improve the generalizability of our pipeline, we firstly tried to use a pretrained cnn-based model, provided by Lapenna and Pacielli in their previous work, reproducing their experiments and using their baseline model.

It must be underlined that the syllabification produced with such a model was not perfect if compared to the ground truth. It was clearly visible by counting the length of the verses - in terms of number of syllables - obtained with those of the ground truth. The following diagrams show the distribution of the lengths of the verses in both the datasets. 

| Length | Frequence |
| --- | --- |
| 10 | 5.979 % |
| 11 | 93.866 % |
| 12 | 0.155 % |



| Length | Frequence |
| --- | --- |
| 8 | 0.014 % |
| 9 | 0.176 % |
| 10 | 7.574 % |
| 11 | 91.218 % |
| 12 | 1.012 % |
| 13 | 0.007 % |

Although the usage of both the datasets led to good results, since our objective was strict to produce accurate hendecasyllabic verses, we opted for the exploitation of the original dataset rather than the one generated by the model. Nevertheless it is still possible with our pipeline to process any poetic corpus, providing as input a plain non-hyphenated text.

# Dataset preparation

### Text cleaning

The Divine Comedy originally comes with some special characters, such as double quotes, brackets and so on. The first preprocessing step has consisted in removing all the punctuation characters, except for the apostrophe < **’** >. Since the apostrophe has the aim of replacing a truncated letter, we decided to maintain it in order to preserve the readability and as a consequence, the correct phonetic syllable separation.

### Tokenization

As the generation of hendecasyllable verses was the goal of the project, we tried to use different types of text tokenization techniques, which differ in the way in which tercets and spaces (and thus, sinalefa) are treated. The aim of testing these different types of tokenization was to study the relation between the variance of the number of tokens in each verse and the ability of the model to generate an hendecasyllabic structure.

# Experiments

### Tokenization

Here we present the different kinds of tercets and spaces tokenization techniques that have been tested.

The tercets tokenization techniques are the following:

* sov: each verse starts with the “start-of-verse” token <v> and the concept of tercet is not kept into account;
* sov-sot: each verse starts with the token <v> and if the verse is the first of the tercet, the “start-of-tercet” token <t> is also appended to the start of the sequence;
* sov-sot-together: each verse starts with the token <v> and if the verse is the first of the tercet, the token <v> is replaced with <t><v> (using a single token instead of two).

We then defined three different space tokenization techniques:

* base: the space character is treated as a token by itself;
* es: the space character is integrated inside the last syllable token of a word;
* is-es: the space character is integrated inside both the first and last syllable tokens of a word.

Finally, in all the tokenization techniques, the “end-of-verse” token </v> is appended at the end of each verse, in order for the model to learn where to stop the single verse generation step. Following is an example of sov-sot-together tercet tokenization with is-es spaces tokenization.

****

### Training

As far as training is concerned, the tokenized text is fed to the transformer model by means of a single sequence as input and a single sequence as output, but in reality a single sequence contains a fixed number of concatenated tokenized verses.

Keeping in mind that the transformer is an autoregressive generation model, we decided to test two different training strategies:

* The input sequence is composed of three consecutive verses and the output sequence is composed of the same three verses plus the fourth consecutive verse.
* The input sequence is composed of just once verse and the output sequence is composed of the same verse plus the second consecutive verse.

### Loss weights

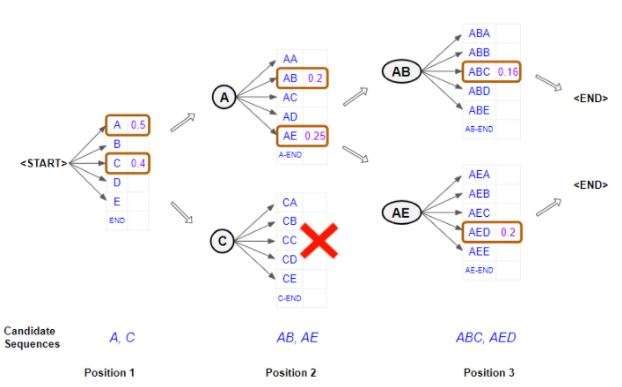
As explained in the problem discussion, we are particularly interested in obtaining lines that satisfy the first criterion of a well-formed hendecasyllables, i.e. generating lines having the last tonic accent on the tenth syllable. To enforce this aspect, we have tried a customized loss which gives more importance to the prediction of the end-of-verse token.

In particular, the loss increment caused by a wrong prediction of the end-of-verse token is multiplied by a fixed weight. Experimental results have shown that this does not affect at all the model for small values of the weight (e.g 2 or 3), while some differences appear with a weight equal to 10, which is the one used in the results shown in this report.

Another good idea could be having a step decaying weight that decreases epoch after epoch, but we were not sure how modifying the loss function during the training would have affect the model, so we did not try this approach, but it could be a further development, as explained at the end of this report.

### Beam Search

The beneficials of more sophisticated decoding techniques which explore more in depth the search space are well-known in the natural language processing field. Since a full search is too expensive to be suitable for almost any application, one of the most popular techniques is the beam search which explores the search space by expanding the most promising *w* sequences of tokens at each step.



Also, we hypothesized that using beam search would have been even more essential in a poem generation task, since constructing the sequence token by token without exploring the search state could lead, in theory, to situations where it is impossible to properly complete the line respecting the metric structure.

We decided to implement this generation technique in opposition to the procedure of single token sampling with temperatures, where each token is sampled from the probability distribution of all the vocabulary, where the probability of each token is divided by the temperature float value, thus meaning that generating with a high temperature score (i.e. greater than one) leads to a more original text.

# Results analysis

### Plagiarism

In order to compare and measure the goodness of the generated cantica we have taken into account the cases of plagiarism, meaning the percentage of sequences of words which have been fully copied from the original text. A generated cantica with a higher plagiarism score should be considered as slightly penalized with respect to a more original one.

The score has been computed by summing all the lengths of the ngrams (from 3-grams to n-grams with n equal to the length of the sequence of all the words in the generation) and averaging the result by the number of generated words. In order to better understand this metric, let us suppose A to be the generated text and B to be the reference text (i.e. the Divine Comedy) and consider a letter as a word.

A = "a b c d x y d e f x y a b"

B = "a b c d e f g h i j k l m n o"

If fixing a lower threshold of 3-grams (meaning that 1-grams and 2-grams are not considered) the sequences “a b c d” and “d e f” result to be copied, for a total of 7 words out of 13 total words in the generation. The score is finally computed as 7/13 = 0.54. To conclude, the plagiarism score obtained is useful to describe situations in which the generated text has many short ngrams copied as well as less but larger ngrams.

### Word correctness

We checked the number of words which have been totally made up by the model. Since each token is a syllable, it often occurs that new words are created. Interestingly, not all of them were meaningless. Here are some examples of meaningless and meaningful words which have been created by the model but are not present in the original Divine Comedy.

| **meaningless** | **meaningful** |
| --- | --- |
| supesce | avanzava |
| assiderio | decida |
| conveloce | vestire |
| diserrame | ritorto |

It is interesting to see how the model, while assembling new words based on syllables, managed to produce words which were unknown for it but that are indeed meaningful in Italian language.

### Hendecasyllables correctness

The evaluation of hendecasyllables is the central criteria used to evaluate our generations. The score has been computed by dividing the number of correct hendecasyllabic veses by the total number of verses in the cantica.

A verse has been considered correct if its 10th syllable was stressed.

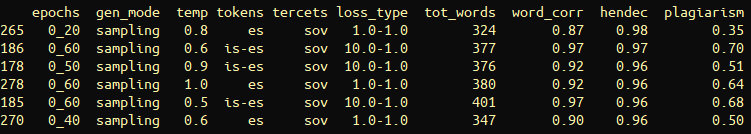
Example:

| nel | mez|zo | del | cam|min | di | no|stra | **vi**|ta

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | **10**|11

In this verse the accent in the last word “vita” falls on the first syllable that is also the 10th of the verse.

In order to retrieve the position of the stress in the words we exploit the dictionary provided from the work of Lapenna and Pacielli, which contains all the information regarding the syllabification and the stressed syllable/character inside the word.



As we can see in the table above, the overall best generation (with id 265) shows a **98%** of correct hendecasyllables, with a very low plagiarism score, which nevertheless caused the word correctness to be not the best one in the proposed rank. However it does not represent a relevant issue since the task was not about optimizing the composition of words. As an example of which kind of evaluation has to be done, we can note that the second entry of the table (id 186) has a hendecasyllable score of just 1 percent lower than the first one, but showing a higher correctness and richness (tot\_words) of the text. Despite this, it also has a higher plagiarism score from which we should not consider it as a valuable result, since we are not interested in generations which are syntactically and semantically perfect but that contain too many plagiarized verses.

# 

# Conclusions

From the obtained results we can state that the idea of including the space character inside the syllables tokens has clearly helped the model in improving the generation of hendecasyllables, as the best performing models have resulted to be the ones implementing this strategy (i.e. those with es and is-es space tokenization). A possible future implementation for the tokenization process could involve the insertion of a syllables separator character as a token combined with a sub-word tokenization technique, so as to improve the overall words correctness while keeping the goodness of the hendecasyllables.

As far as tercet tokenization is concerned, the best models for the hendecasyllables task resulted to be the ones that do not take into account the concept of tercets (i.e. those with sov tercet tokenization), perhaps suggesting that the addition of the start-of-tercet tokens draws some attention away from the hendecasyllables task. We hypothesize that a fine hyperparameters tuning could fix this throwback and allow the generation of good hendecasyllables while preserving the tercets structure.

We stated that the addition of the loss weight parameter for the end-of-verse token did not improve the generation of good hendecasyllables, but additional tests with different values for this parameter should be run for a better understanding.

Finally, the current implementation of the beam search generation technique did not lead to any improvement on overall goodness of the generations, but we think that by mixing the temperatures-based sampling process with the beam search technique would lead to better results. Also, the beam search could lead to better results if the single generation step, which is currently done verse by verse, would instead be done by keeping a longer sequence, perhaps a whole tercet.

1. https://github.com/AlessandroLiscio/DeepComedyGenerator [↑](#footnote-ref-0)