

Deep Comedy

Syllabification and Generation of tercets in Divine Comedy's style

Valentina Borianio
valentina.borianio@studio.unibo.it

Pietro Fanti
pietro.fanti@studio.unibo.it

September 2021

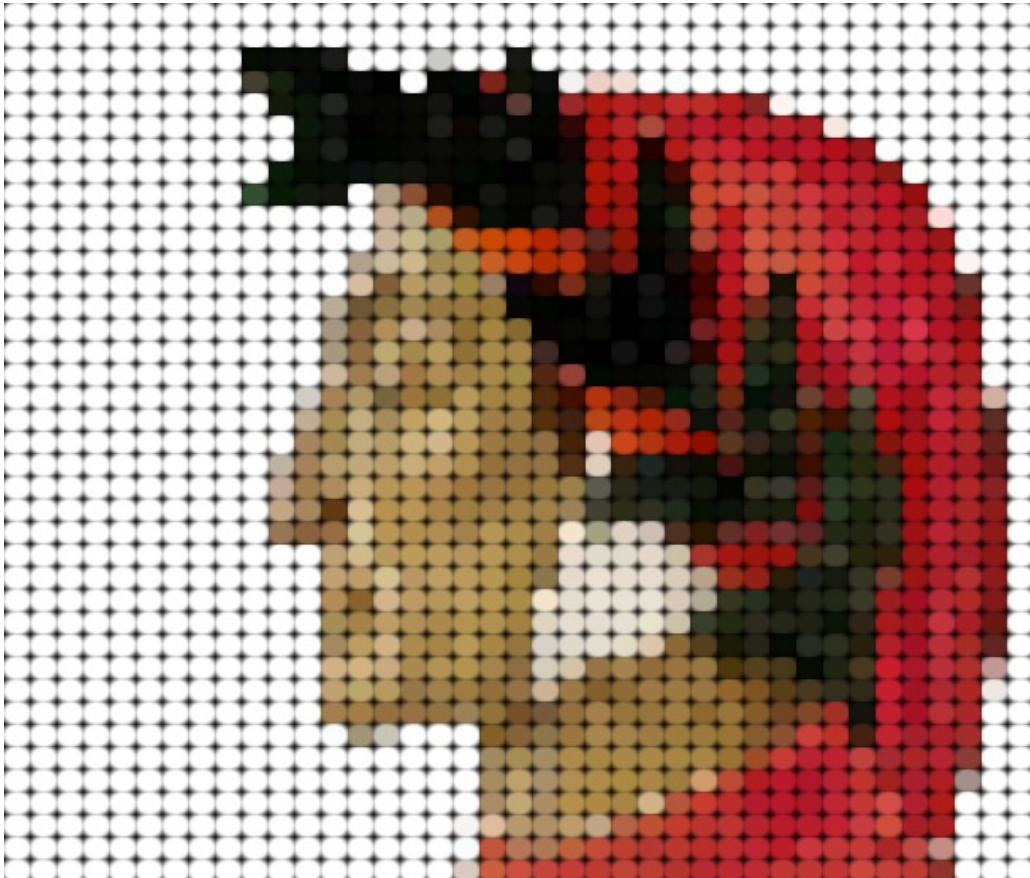
Abstract

The aim of this project is to preprocess Dante's Divine Comedy in order to design and train two Deep Learning models able to:

- syllabify and insert caesura in hendecasyllable verses;
- generate syllabified text with caesura following the style of Dante's Divine Comedy.

We achieve it using transformers, whose high performances in the field of natural language processing are well-known. Regarding the generation task, exploiting beam search algorithm to perform the prediction step improved text's quality and fluency, but the same does not happen with top-k sampling and other techniques that generate a non-deterministic output, confirming that greedy and beam search apparent tendency to generate repetitive text is caused by a wrong model's training rather than the decoding and prediction method.

The code can be found on the GitHub repository of the project [1].



Contents

| | | |
|----------|-----------------------------|-----------|
| 1 | Introduction | 3 |
| 2 | Transformer | 3 |
| 2.1 | Loss Function | 4 |
| 2.2 | Accuracy Function | 4 |
| 2.3 | Optimizer | 4 |
| 3 | The Data | 5 |
| 3.1 | Preprocessing | 5 |
| 3.2 | Tokenization | 6 |
| 4 | Syllabification | 7 |
| 5 | Generation | 9 |
| 5.1 | Learning Phase | 9 |
| 5.2 | Greedy Search | 10 |
| 5.3 | Beam Search | 12 |
| 6 | Conclusions | 13 |

1 Introduction

On September exactly 700 years ago, Dante Alighieri died in Ravenna [2]. Today he is considered the father of Italian language and he is world-wide famous for his *Commedia*, later renamed *Divina Commedia*.

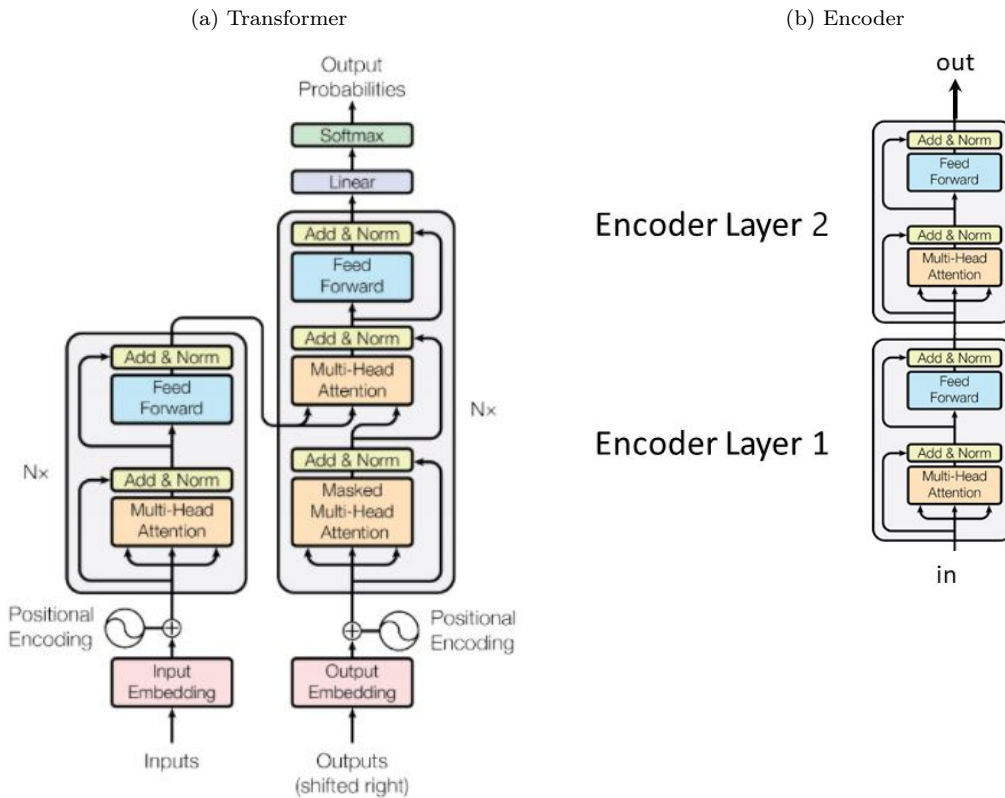
On the occasion of this anniversary, we have designed two artificial neural networks and trained them on *Divina Commedia* [3]. The first model, as explained in section 4 takes in input a line and return its syllabification taking in account metrics figures like *synalepha* and *dyalepha* and also pointing out *caesura*, which in a *hendecasyllable* is the metric pause that divides the first and the second *hemistich*. Regarding the second model, as explained in section 5, it takes in input a *tercet* (a group of three hendecasyllables) and return a brand new tercet, generated by the model itself. For both models we choose to rely on transformers, which are state of the art technologies in natural language processing tasks [4], to see if we can obtain better results than those obtained using recurrent neural network, as in other projects on the same topic.

The results of the first task are impressive, reaching very high accuracy on both training and validation set, and performing quite well also on hendecasyllables from other authors and from other historical periods, which strongly influence the language. The second task was more complicated, but results are still quite satisfactory.

2 Transformer

The Transformer in NLP is a novel architecture that aims to solve sequence-to-sequence tasks. It relies entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution [4]. Transformers are manly divided in two part: Encoder and Decoder.

Figure 1



The first step of the transformer is the *input embedding* that provides a vector of d_{model} size and update its values each batch, to optimize the space.

A particular part of the transformer is the *positional encoding* to address the problem of processing an input sentence sequentially, one word at a time. The positional encoding makes it possible for the transformer to use information about word order, in order to process all words in the input simultaneously.

Now it's time for the encoder to process our data.

In the figure, we can see that the encoder is made by layers (the number of the layers is defined by the hyperparameter `num_layers`). Each encoder layer contains two sub-layers:

- the first sub-layer is Multi-Head Attention.

- the second sub-layer is Feed Forward Neural Network

The Decoder is composed by 3 sub-layers:

- the first sub-layer is Multi-Head Attention.
- the second sub-layer is Feed Forward Neural Network
- the third sub-layer is another Multi-Head Attention layer.

To implement transformers we used Tensorflow and we mostly stayed true to tutorial’s implementation [5].

2.1 Loss Function

The loss function that we have chosen is the *Sparse Categorical Crossentropy*, which computes the crossentropy loss between the labels and the prediction when, as in our cases, true labels are integer encoded, and not one-hot encoded [6].

2.2 Accuracy Function

Our accuracy function is trivial but effective: simply we calculate the number of correct predictions (greedy choosing every time the token with the highest probability) over the number of total predictions. The prediction of the initial token is excluded from the calculation.

2.3 Optimizer

We use the Adam optimizer with a custom learning rate scheduler, according to the formula

$$lrate = d_{model}^{-0.5} * \min(step_num^{-0.5}, step_num \cdot warmup_steps^{-1.5}) \quad (1)$$

Which is proposed in *Attention is all you need* paper [4] and plots a learning rate’s progress as the one in figure 2.

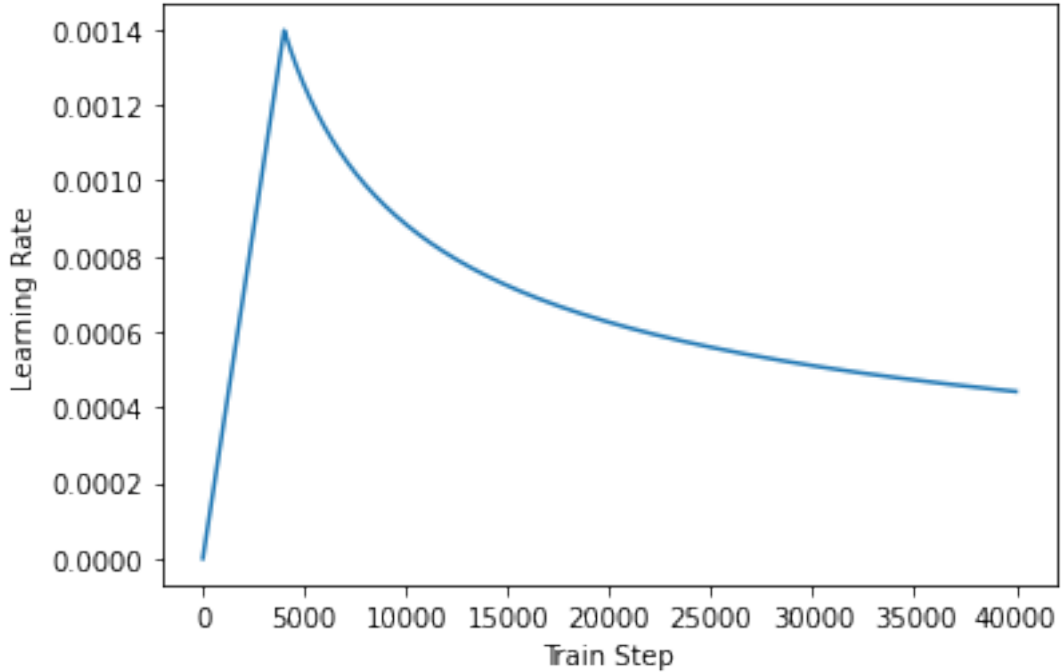


Figure 2: Learning rate progress over train steps

3 The Data

We used the following data:

- a syllabified version of Petrocchi’s critical edition of the Comedy [3] obtained by Asperti and Dal Bianco’s work [7];
- a vocabulary containing all the words of Divine Comedy and their stressed syllable position [7];
- a list of Italian *proclitic* words, namely monosyllabic words without an autonomous accent [8].

3.1 Preprocessing

The Divine Comedy file that we have utilized, has been cleaned from all unuseful information, such as:

- all the capital letters have been converted in lowercase;
- all the punctuation (, . ; : \— ? ! " () [] « » ” ” ”) symbols have been removed, but ticks;
- all the headings "Canto" have been removed;
- all the number of the lines have been removed;
- all the types of ticks has been replaced in only one type.

Accents and umlauts, as well as ticks, are preserved. Also, we add some special tokens:

- spaces are replaces with `S`;
- in the `X.csv` file syllable divisors `|` are removed, while in the `y.csv` file they are replaced with `Y`

The more complicated step was to detect caesura, which is a metric pause preceded by a stress on either the fourth or sixth syllable. In the first case the hendecasyllable is called *a minore*, in the latter *a maggiore*. The caesura is called *maschile* when the stressed syllable belongs to an oxytone or *femminile* when it belongs to a paroxytone. More rarely it can belong to a proparoxytone. Since the caesura is situated after the word that host the stressed syllable, if the word is oxytone, paroxytone or proparoxytone then caesura’s position changes. Furthermore, there is also another kind of caesura, called *lirica*, which happens when we have a stress on the third and sixth syllables.

Also there some very rare cases: caesura can divide two words joint by sinalefe, or it can divide a word if it ends with *-mente* or *-zial*. Combining all this variety, we have 12 possible cases, which cover almost all the lines of the poem. However, 7 verses do not have caesura, since are non-canonical hendecasyllables [7].

Moreover, in Italian language some monosyllabic worlds count as stressed, while others not. The latter ones are called *proclitic*, and they must be taken in account in order to properly detect caesura’s position. To deal with this, we have used a listed of proclitic words. However, the list is partial, since exclude some words that can be proclitic or not, depending on the use.

For this reason, but also because sometimes caesura cannot be just deducted from metric structure of the line, but needs also a semantic analysis, our algorithm is far from being perfect.

The only way to have caesura perfectly annotated in each line is with the intervention of a human expert, but unfortunately nowadays does not exist a public database containing information like this.

Anyway, once detected the caesura is marked with `C` and lines are written in `y_cesura`. So from the first line of Divine Comedy

```
1 |Nel |mez|zo |del |cam|min |di |no|stra |vi|ta
```

we obtain an input version in `X.csv`

```
nel S mezzo S del S cammin S di S nostra S vita
```

and a target version in `y_cesura.csv`

```
Y nel S Y mez Y zo S Y del S Y cam Y min S C di S Y no Y stra S Y vi Y ta
```

3.2 Tokenization

In order to be passed as an input to transformers, text must be tokenized, i.e. split in units and then converted in integers. Tokenization is an important step since a good or a bad choice of the tokenizer can influence the quality of results. Since we need to deal with syllables in both syllabification and generation task (in the latter syllables are important to respect the metric scheme) we must rely on a sub-syllabic tokenization or on a character-wise tokenization. The second one is of course simpler to implement, but produced worse results, so we opted for a sub-syllabic tokenization.

The first step is to build a vocabulary of sub-syllabic units. We have used TensorFlow’s `bert_vocab.bert_vocab_from_dataset` from `tensorflow_text` package, which follows the top-down implementation of WordPiece algorithm from BERT [9, 10]. Actually WordPiece produces sub-words tokens, but instead of passing words to it, we passed syllables, so we obtain exactly what we need. The hyperparameters of the algorithm are the reserved tokens and the length of the vocabulary. As reserved tokens we selected [START] and [END] as start and end tokens, whose values are respectively 0 and 1. Also S, Y and C for space, syllable divisor and caesura. Furthermore, as shown in section 5, one more token is added: N for the new line. Regarding vocabulary length, we have chosen 200 and so we obtained `vocab.txt` and `vocab_gen.txt` which only differ for the new line token.

Given the vocabulary, the tokenization is performed by TensorFlow’s `BERTTokenizer` [11], that on Divine Comedy’s first line works as follows:

VERSE INPUT:

Nel mezzo del cammin di nostra vita

TOKENIZED INPUT:

[0, 198, 71, 171, 57, 120, 97, 69, 69, 50, 42, 44, 66, 72, 60, 98, 83, 60, 47, 1]

4 Syllabification

The aim of this first task is to syllabify each hendecasyllable of Italian poetry. The tokenization process is the same explained in section 3.2.

Model's inputs are tokenized verses without the special tokens that have to be inserted by the model:

- Y that correspond to the syllable, the numerical value of the token is 2;
- C that correspond to the caesura, the numerical value of the token is 3.

We divided the set of 14233 lines in a training set of 13808 lines, a validation set of 282 lines and a testing set of 143 lines. We used the following hyperparameters :

- **num_layers** = 4
- **d_model** = 128
- **dff** = 512
- **num_heads** = 8
- **dropout_rate** = 0.1
- **epochs** = 20

The following graphs show the accuracy and the loss in each epoch of training and validation set:

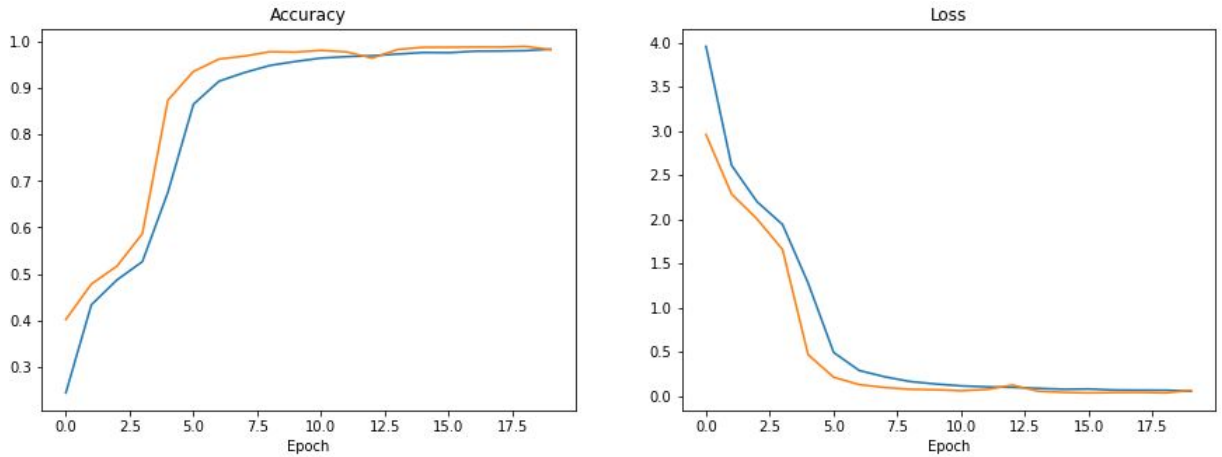


Figure 3: Accuracy and Loss of training and validation set

We can see that accuracy keep growing both in training and validation set, meaning that no overfitting appeared.

We have tested our model not only in Divine Comedy but also with other Italian works (e.g. *L'infinito* by Leopardi, *Cinque Maggio* by Manzoni). To select the predicted token we have simply implemented a greedy search that picks the token with the highest probability. No need to implement more sophisticated methods, like instead happened in generation task (section 5).

To have a more readable output, the token Y is replaced with the symbol | to show the syllables' division, while the token S is replaced with space and the token C with the symbol \$ to show the caesura.

Figure 4 shows the results we had for the syllabification in the Divine Comedy.

| | |
|------------|---|
| Original: | in voce assai più che la nostra viva |
| Predicted: | in vo ce as sai più che \$la no stra vi va |
| True: | in vo ce as sai più che \$la no stra vi va |
| Original: | ché per etterna legge è stabilito |
| Predicted: | ché per et ter na \$leg ge è stab bi li to |
| True: | ché per et ter na leg ge è \$sta bi li to |
| Original: | che son quinc' entro se l' unghia ti basti |
| Predicted: | che son quin c' en tro se \$l' un ghia ti ba sti |
| True: | che son quin c' en tro \$se l' un ghia ti ba sti |
| Original: | saver fu messo che se 'l vero è vero |
| Predicted: | sa ver fu mes so che \$se 'l ve ro è ve ro |
| True: | sa ver fu mes so che \$se 'l ve ro è ve ro |
| Original: | e sempre di mirar faceasi accesa |
| Predicted: | e sem pre di mi rar \$fa cea sa si ac ce sa |
| True: | e sem pre di mi rar \$fa cea si ac ce sa |
| Original: | e bēatrice forse maggior cura |
| Predicted: | e bē a tri ce for se \$mag gior cru ra |
| True: | e bē a tri ce for se \$mag gior cu ra |

Figure 4: *Divine Comedy* syllabified

The following figure shows the results we had for the syllabification in Leopardi's *L'infinito* and in Manzoni's *Cinque Maggio*. The first is a poem in hendecasyllables, as *Divina Commedia*, and indeed results are quite good. Regarding the latter, results are of course worse, and this is caused by the training phase. Indeed, the model is only trained on hendecasyllables and so fails to syllabify other types of verses, like *Cinque Maggio*'s septenaries.

Figure 5: Syllabification on other Italian poems

| (a) Leopardi's <i>L'infinito</i> | | (b) Manzoni's <i>Cinque Maggio</i> | |
|----------------------------------|---|------------------------------------|--|
| Original: | sempre caro mi fu quest'ermo colle | Original: | ei fu siccome immobile |
| Predicted: | sem pre ca ro mi fu \$que st' er mo col le | Predicted: | ei fu sic co me \$im mo bi bi le |
| Original: | e questa siepe che da tanta parte | Original: | dato il mortal sospiro |
| Predicted: | e que sta sie pe che \$da tan ta par te | Predicted: | da to il mor tal \$so spi pro ro |
| Original: | dell'ultimo orizzonte il guardo esclude | Original: | stette la spoglia immemore |
| Predicted: | del l'ul ti mo o riz zon te il \$guar do e sclul de | Predicted: | stet te la spo glia \$im me me mo re |
| Original: | ma sedendo e mirando interminati | Original: | orba di tanto spiro |
| Predicted: | ma se den do e \$mi ran do in ter mir ni ta ti | Predicted: | or ba di tan to \$spi ro ro |
| Original: | spazi di là da quella e sovrumani | Original: | così percossa attònta |
| Predicted: | spa zi di là da quel la e \$sov rur mu ni | Predicted: | co sì per cos sa \$at tò ni ti ta |
| Original: | silenzi e profondissima quiete | Original: | la terra al nunzio sta |
| Predicted: | si len zi e pro fon dis si ma \$qu uī e te | Predicted: | la ter ra al nun zio sta |
| Original: | io nel pensier mi fingo ove per poco | Original: | muta pensando all'ultima |
| Predicted: | io nel pen sier \$mi fin go o ve per po co | Predicted: | mu ta pen san do \$al l' ll ti ma ma |

5 Generation

The aim of this second task is to generate a well-formed tercet of hendecasyllable verses with the correct rhyme scheme, that is a three-line stanza using chain rhyme pattern ABA BCB CDC and so on, which is called *terza rima* or *rima dantesca* [12]. Our idea was to start from the previous model changing input and target sequences in order to generate new text.

The difference between the other model is that we added one new token, N, to separate a line from another. So the input sentences are a tercet, while the target is the syllabified version with caesura of the subsequent stanza, which can be a triple or a single line, indeed the last line of each *canto* is a stanza itself, and is not grouped with other lines. A single line can not be an input, since being the last stanza of a *canto* it does not have a subsequent stanza to use as target.

So we have a set of 4811 couples of stanzas (triplets or single lines) which is divided in a training set of 4666 couples, a validation set of 96 couples and testing set of 49 couples.

5.1 Learning Phase

We utilized the following hyperparameters:

- `num_layers` = 4
- `d_model` = 256
- `dff` = 1024
- `num_heads` = 8
- `dropout_rate` = 0.1

We trained this model up to 200 epochs and in figure 6 we can see loss and accuracy's progress over epochs' advancement. We can notice that the model start overfitting around the twenty-fifth epoch. So at first we tried with just 20 epochs, and results was quite good, but we obtained better results with 50 epochs, while the overfitting causes bad results starting from 100 epochs. Indeed, it learns a single triplet as optimal and it starts to repeat it over and over.

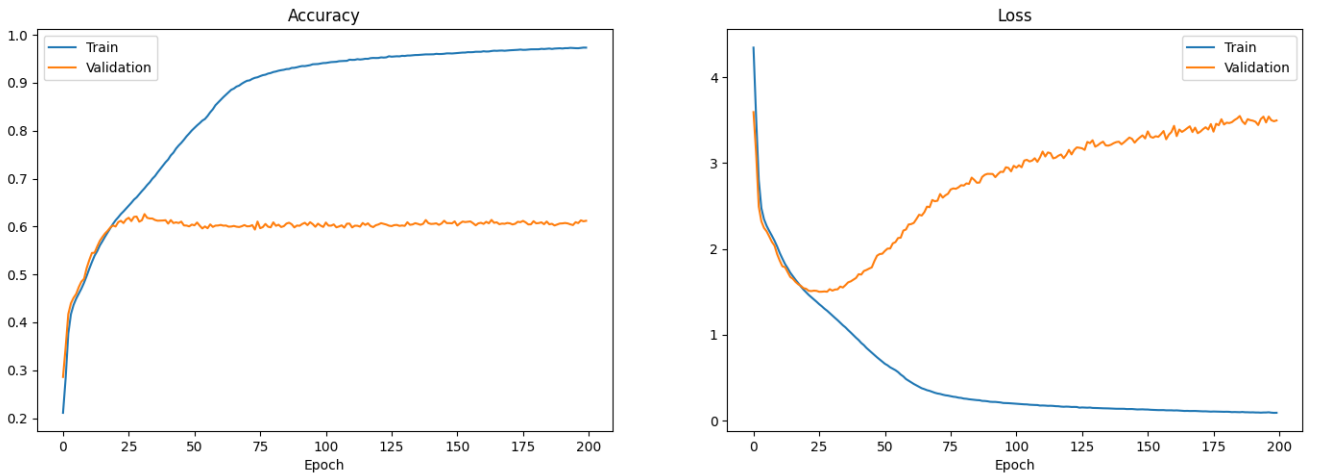


Figure 6: Loss and Accuracy over epochs

The input taken is one triplet from the testing set (SEED), the output is generated by the model tercet by tercet.

We adopted different methods to predict the output: standard greedy search, greedy search with top-k sampling and beam search. Results obtained are overall good, indeed the metric scheme is quite respected: almost all the lines are hendecasyllables, and the few verses that are not, just miss one syllable to be correct. The caesura is present in all the lines and usually it is in a correct position and emerges the trend of having mainly hendecasyllables *a maggiore*, namely with the stress before the caesura on the sixth syllable.

Rhymes' scheme is worse, but still quite good, indeed the model learned that the last tokens of the first line of each tercet are usually equal to those of the last line of the same tercet. But it fails in chaining stanzas together, in fact there is no correlation between the last tokens of the second line of a tercet and those of the

first and the third lines of the following tercet. So, the resultant rhyme scheme is something like AXA BYB CZC, instead of ABA BCB CDC. Also note that not all the rhymes are fully correct, because even if the last characters are the same, a rhyme to be correct must have character identity starting from the last tonic stress of the line [13]. However, since our transformer has no indication about where the last tonic stress is, it sometimes fails to make an identity long enough, but it just makes equal the last two or three characters.

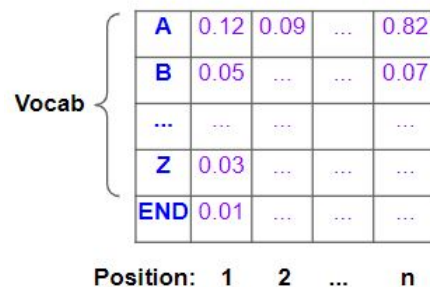
Even if all the three methods are generally good, still there are some differences between them with beam search that particularly stands out, as explained in the following subsections.

5.2 Greedy Search

As seen before, the NLP model constructs a vocabulary consisting of the entire set of words. The model takes the source sentence as its input and passes it through the embedding layer and the encoder.

Then the output of the encoder passes through the decoder that generates its own output, that is an encoded representation of the sentence in the target language.

After that, it passes through an output layer, which is composed by some Linear layers and a softmax. The output of the Linear Layers is the likelihood of occurrence of each word in the vocabulary, at each position in the output sequence. Then the softmax is responsible to convert the scores into probabilities.



| | | | | | |
|-------|-----------|------|------|-----|------|
| Vocab | A | 0.12 | 0.09 | ... | 0.82 |
| | B | 0.05 | ... | ... | 0.07 |
| | ... | ... | ... | | ... |
| | Z | 0.03 | ... | ... | ... |
| | END | 0.01 | ... | ... | ... |
| | Position: | 1 | 2 | ... | n |

Figure 7: Vocabulary probabilities

To get the final sentence, the model decide which word it should predict for each position. The *Greedy Search* choose the word that has the higher probability at each position:

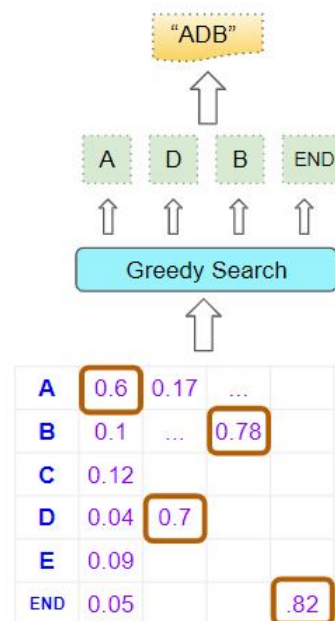


Figure 8: Greedy Search

Greedy search performs very bad when the model presents a high overfitting (+100 epochs) because the generated text is very repetitive. Regarding a well trained model, instead, it performs quite good, but a bit of repetition is still presents.

Repetition can be eliminated with k-sampling, which consists in filtering the k most likely next tokens, redistributing the probability mass among only these k tokens and randomly picking one of them according to the redistributed condition probability distribution [14].

Using this technique highly improve the text fluency when the model is overfitted, but actually does not affect too much the model trained with only 50 epochs, as you can see in following output:

Listing 1: Standard greedy search

```
|se |ben |ti |ri|cor|de|re e $nel |dol|ce
|do|ve |le |co|se |mie $ri|mi|ran|do in|te|so
|ma |io |veg|gio e |l' ac|qua e $l' al|ta |fol|le

|e |io |che |son |ri|vol|si al $pri|mo |dè|ro
|s' io |non |sa|rei |sa|lir $mi |sco|sce|se
|a |li al|tri |due $che |mi|se|ri |si|cu|ro

|e |io |che |son |l' af|fo|co $mi |fui |tol|se
|non |vol|le a |me $co|me |ve|nir |le |vi|te
|s' io |non |ri|vol|si a $l' a|ni|ma |tol|se

|e |io |che |son |ri|vol|si a $quei |che |mor|de
|s' io |ri|vol|go |stret|to $son |spi|ri|ti
|non |vi|de |ma|dre a $l' ul|ti|ma |sor|cor|ce

|e |l' al|tro |se|gui|tar $la |ri|spo|sta |chia
|che |fé |tar|di |di |quei $che |di |di|scen|de
|per|ché |l' a|spet|to |vi|di $di|spo|ria
```

Listing 2: Greedy search with top-k sampling where k is 3

```
|ben |ma|e|stro |mio $che |di|sce|se an|co|ra
|se |mi |sve|mi al |mio $et|ter|no |le |fo|co
|co|sì |ri|co|min|cia' io $mi |ri|vol|si a|ra

|che |l' u|no e |l' al|tro |più $non |si |per|de
|per |la |via |per|cu|ta $mol|to e|ra |bru|scia
|non |per |cos|sa |dì $a |lei |a|scol|le

|e |io |sen|ti' |già |mai $chi |sia|ma |gue|se
|l' a|ni|ma |tri|sta e |non $per|de|ri|sto a |fo|co
|ma |quel |che |to|sto an|cor $ti |ri|ste|se

|e |quel|la |so|le on|de $non |pa|dre |vo
|che |lì |ti |fia |se|col|la in $là|si |ti|ra
|io |veg|gio in|nan|zi |ch' è $guar|da e 'l |so|le

|co|sì |al |so|le e 'l $piè |du|ca a |vòl|to
|s' a|vria |m' ha |fat|to a |me $de|gne |mie
|dis|s' io |ma |non |fui $mag|gior |né |mal|to
```

Both outputs 1 and 2 are generated from the seed:

```
i' fui colui che la ghisolabella
condussi a far la voglia del marchese
come che suoni la sconcia novella
```

5.3 Beam Search

The proceeding in the Beam Search is more complicated. It keeps the most likely w sequences of tokens at each step and at the end it chooses the hypothesis which has the overall highest probability [15].

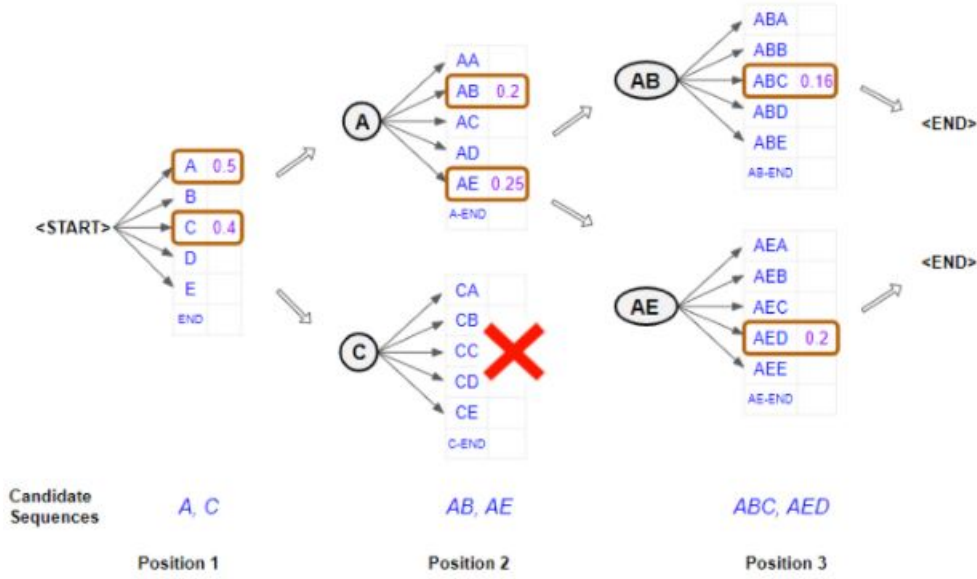


Figure 9: Beam Search

Beam search gives the best results. Repetition is present but very low, and the metric structure is more accurate than the outputs generated by the greedy search. Also from a semantic point of view the lines sound better. We have also implemented a non deterministic version of beam search, but if it improves outputs of overfitted models, it has a bad impact on the model trained on 50 epochs.

Using the same seed used in section 5.1 the first four tercets we obtain are the ones in listing 3.

Listing 3: Beam search with beam width 5

```
|ben |m' ac|cor|si |ch' el|li e|ra $da |ciel |mes|so
|ch' io |vol|gea i |cad|di e $ri|ma|se e 'l |vol|to
|non |per |ve|der |non |ei $com' |io |ti |mes|so

|co|sì |l' ae|re |vi|cin $qui|vi |si |mi|ra
|qui|vi è il |gran |d' o|gne $par|te u|dir |quin|ta
|co|sì |ri|spuo|se a |me $che |ti |ri|mi|ra

|e |l' al|tro |dis|se |quel $che |tu |hai |guar|do
|ri|tro|ve|rai |co|me a $quel|l' uom |ti|ra
|co|me an|cor |ti |sa|reb|be in $ma|ra|vi|glia

|ben |ma|e|stro |mio $dis|s' io |ch' io |vi|di
|ed |el |s' ac|cor|s' io |e|ra $sì |com' |el|l' ac|cor|to
|ch' io |di|co |co|lui $che |fui |tut|ti |ti|ri
```

6 Conclusions

Transformers confirmed to be very strong and versatile tools for the natural language processing field, indeed the aim is to obtain good results in two very different applications, like syllabification and generation of new text.

The syllabification task gave of course better results because of the easier nature of the problem. The model is able of properly syllabifying *Divina Commedia*'s lines from the test set and also lines from other Italian poems in hendecasyllables. Poems with other metric schemes could be of course approached easily training the model with a more variegated database, containing also septenaries and other metrics.

The generation task gave solid results, too. Almost every line is a well-formed hendecasyllable, the caesura is always present and usually it is in the correct position. Regarding the rhymes they are present but far to be perfect. In this direction, we could obtain a better rhyme scheme redefining the loss function in order to penalize the model when it does not respect the scheme. The lines generated respect also basic grammar rules, like gender concordance between articles, nouns and adjectives. Not all the words actually exist in Italian, but they "sound" like they were. This is not necessarily a disvalue, since Dante is particularly famous to introduce neologisms in his poems, mainly in *Divina Commedia* [16]. The principal issue of this task is that most of the verses have not meaning, in order to solve it, we should train our model with a bigger database, for example including also other poems.

One of the most interesting thing emerged from this project is that when the model is properly trained and not overfitted, sampling techniques have almost no impact on the quality of the generated text confirming state of the art theories. In fact, recent studies suggest that greedy search and beam search's apparent flaws are only determined by the model and its training and not by the decoding algorithm [17].

References

- [1] Deep Poets Society, *Deep Comedy Project* on GitHub, <https://github.com/Deep-Poets-Society/Deep-Comedy>, URL consulted on 9 Jul 2021.
- [2] *Dante Alighieri*, in *Treccani.it – Enciclopedie on line*, Istituto dell’Enciclopedia Italiana, URL consulted on 6 Jul 2021.
- [3] Giorgio Petrocchi (editor). *La Commedia secondo l’antica vulgata*, 4 voll. A. Mondadori, 1966-67
- [4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin (2017), *Attention Is All You Need*, <https://arxiv.org/abs/1706.03762>.
- [5] *Transformer model for language understanding* (2021), Tensorflow, <https://www.tensorflow.org/text/tutorials/transformer>, consulted on 6 Jul 2021.
- [6] *tf.keras.losses.SparseCategoricalCrossentropy* (2021), Tensorflow, https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy, consulted on 7 Jul 2021.
- [7] Andrea Asperti, Stefano Dal Bianco (2020), *Syllabification of the Divine Comedy*, <https://arxiv.org/abs/2010.13515>.
- [8] Livio Gaeta (2011), *Parole proclitiche*, in *Treccani.it – Enciclopedia dell’Italiano*, Istituto dell’Enciclopedia Italiana, URL consulted on 7 Jul 2021.
- [9] *Subword tokenizers* (2021), Tensorflow, https://www.tensorflow.org/text/guide/subwords_tokenizer, consulted on 7 Jul 2021.
- [10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova (2019), *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, <https://arxiv.org/abs/1810.04805>.
- [11] *text.BertTokenizer* (2021), Tensorflow, https://www.tensorflow.org/text/api_docs/python/text/BertTokenizer, consulted on 7 Jul 2021.
- [12] Claudio Ciociola (2011), *Terza rima*, in *Treccani.it – Enciclopedia dell’Italiano*, Istituto dell’Enciclopedia Italiana, URL consulted on 7 Jul 2021.
- [13] *Rima*, in *Treccani.it – Enciclopedie on line*, Istituto dell’Enciclopedia Italiana, URL consulted on 8 Jul 2021.
- [14] Angela Fan, Mike Lewis, Yann Dauphin (2018), *Hierarchical Neural Story Generation*, <https://arxiv.org/abs/1805.04833>.
- [15] Patrick von Platen (2020), *How to generate text: using different decoding methods for language generation with Transformers* on <https://huggingface.co/blog/how-to-generate>, URL consulted on 8 Jul 2021.
- [16] Ghino Ghinassi (1970), *Neologismi*, in *Treccani.it - Enciclopedia dantesca*, Istituto dell’Enciclopedia Italiana, URL consulted on 9 Jul 2021.
- [17] Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, Jason Weston (2019), *Neural Text Generation with Unlikelihood Training*, <https://arxiv.org/abs/1908.04319>.

Figures 7, 8 and 9 are taken from <https://towardsdatascience.com/foundations-of-nlp-explained-visually-beam-search-how-it-works-1586b9849a24>, while figure 1 is taken from <https://www.tensorflow.org/text/tutorials/transformer>.