Deep Comedy

Deep Learning - project 4

a.a. 2019/2020

Generating original verses in Divine Comedy-style with Transformers

Cozzi Riccardo
riccardo.cozzi@studio.unibo.it

Liscio Alessandro
alessandro.liscio@studio.unibo.it

Abstract The artificial generation of poetries has been proposed several times as a way to demonstrate the performance of sequence-to-sequence models such as RNNs and LSTMs used in transfer learning tasks oriented to capture the intrinsic characteristics of a set of textual documents and producing a novel text with the same style. The standard solutions proposed so far copied the writing styles and the structure of the text but in some cases it is too difficult to catch some more hidden and complex characteristics, like rhymes, and keep the references to the context for a long period. With this work we want to demonstrate how good transformers are in performing those tasks, even on the relatively small dataset of the Divine Comedy, by reproducing the Dante's style of writing and vocabulary, in some case imitating the Tuscan involved language, by building new nonexistent but convincing words in a sort of metasemantic poems, perfectly respecting the rules of *terza rima* and hendecasyllables, without any type of external heuristic intervention.

1 Introduction

The aim of the work is to build a deep learning model able to generate a text with the style of the Divine Comedy, that means respecting constraints such as “terza rima” (verses organized in tercets with the specific enchained rhyme structure ABA BCB...) and hendecasyllables (concept which has been oversimplified in this instance in order to reach a solution for the problem, as we will see and discuss later). As we discovered during our preliminary research, examples of existing works with the same purpose are very sparse, which are in the majority oriented through the generation of poems and sonnets [1, 2]. The only work we found in literature which was specifically oriented in our direction is described in the paper “Neural Poetry” [3]. As they show, according to evaluations given by experts in that domain, their results appear to be quite good, respecting terza rima and hendecasyllables. Despite this, the approach that they chose was to generate tercets and keep only the good ones, thus not generating a full canto. We could not find any long enough text generated by their model, and also they did not share a repository containing the source code. For this reason we are not sure about their model being actually capable of generating a full canto or just one tercet per time. However we have found extremely interesting and useful the approach they used and we have taken inspiration from it when implementing the text tokenization procedure.

Despite the initial idea of using RNNs, as well as the majority of the works we have encountered exploit them [2, 8], we quickly changed our mind and decided to adopt transformers, a relatively new attention-based encoder-decoder architecture, due to the need of capturing the constraints in the text which compose the rhymes [4]. The intuition was right, indeed, because doing so we obtained such good results and built the model we are going to present, which is able to generate a potentially infinite number of verses, always respecting the conditions mentioned above. It follows a sample of three tercets generated by our model.

E io vidi con le genti in novelle
più alto poco dir per la vista corte
rivolge a la cagion che non favelle.

ma dimmi quei che pace a te forte,
se troppa luce, che la gente nota,
rimembrar dal fatto hai mano inforte!.

ed ella: io di onde qui rinota,
rispuos'io lui, ti giova prìa ch'i' tolsi
come mano a costui fu sì commota.

1.1 Problem analysis

There is an important discussion to be done about how the problem is defined and how we had to reformulate it in order to make it solvable. The problem, as already stated, is to generate hendecasyllables in terza rima. In order to do that, we do not need to make the model know what an hendecasyllable or a rhyme is, but we just need it to notice the patterns appearing in the text (i.e. syllables sequences), since it is a poem. First, let us define what an hendecasyllable is and what makes two words rhyme each other:

1. in the Italian metric, a verse is hendecasyllable when its tenth syllable is stressed, so if it contains a tonic vowel. This means, given that in the majority of Italian words the tonic accent falls on the penultimate syllable, almost 70% of the verses are composed of 11 syllables, but there exists hendecasyllable verses with 10, 12, or even more syllables.
2. two words rhyme each other if their letters, starting from the tonic vowel, are equal. For example the words “oscura” (o-scù-ra) and “paura” (pa-ù-ra) make a perfect rhyme. On the other hand “vedere” (ve-dé-re) and “andare” (an-dà-re) does not make a rhyme.

Synalepha and dialefa have an impact on the counting of the syllables in a verse. The first is the act of merging the last syllable of a word with the first syllable of the next one, in order to pronunciate them with a single sound when those syllables are not stressed. The latter, in contrast, consists of keeping the sounds separated. E.g. “selva oscura” becomes “sél-vao-scù-ra” but “selva aspra” remains “sél-va-à-spra” because the initial “a” of “aspra” is stressed.

As far as the hyphenation is concerned, the main difficulty one has to deal with is the fact that hyphenating italian words, with particular regard to those used by Dante, who often used archaic or self coined idioms, is not so immediate. We managed to hyphenate most of the terms by using the website sillabare.net and making it hyphenate the whole set of words of the productions. Doing so we obtained a dictionary to draw hyphenations from anytime we needed (during the tokenization of the text and the posterior evaluation). This made the computation considerably faster, with respect to the initial approach consisting of using an hyphenation algorithm. Also, we used a python API [5] to know the position of the tonic accent in all the common italian words (so excluding the complex or uncommon terms), which are almost the 10% of Dante's vocabulary. However, the problem is that this method, as well as other APIs, are not precise enough to ensure that the resulting hyphenated verse is hendecasyllable. In fact, the only way to correctly hyphenate a word for which the

hyphenation is unknown apriori, is to know which syllable is the stressed one. In practise, it all orbits around the understanding of where the tonic accent falls inside of each word. Unfortunately, we did not have any information about tonic accents from the text, so it appeared almost impossible to trace these features. For this reason we decided to simplify the task, redefining the concepts of hendecasyllable and rhyme.

More precisely, we assumed that:

1. a hendecasyllable is a verse of eleven syllables
2. two words rhyme each other if their last syllables are equal.

This allows us to make the problem treatable with the information we had available, but it follows that, since the original texts are almost perfect (in terms of rhymes and hendecasyllables), not always the correct patterns have been captured. However we are confident that, having a perfect hyphenated dictionary of the set of words used by Dante, the hendecasyllables will be perfect, but even with this solution the rhymes problem remains, as one should find another way to tokenize the text in order to preserve hyphenation and, at the same time, using as a token the part of the word which makes the rhyme. E.g. the words “vedere” and “andare”, if both tokenized by syllables, do not make a rhyme. However the pattern catched by the model is that the 1st and the 3rd verses of a tercet (as well as the 2nd and the 1st of the next tercet) must end with the same token, so it may be confident that placing in those position these two words should be a good choice. On the other hand, the solution of tokenizing by words or by characters is not applicable, because doing so the notion of the quantity of syllables in the verse (without knowing anything about the tonic accent, of course) is not preserved. We are confident that a solution to this problem could really make the production perfect and thus should be further investigated.

2 The data

2.1 Productions

The Divine Comedy is a long Italian narrative poem by Dante Alighieri completed in the 14th century. It is widely considered to be the pre-eminent work in Italian literature and one of the greatest poems of the world, it helped establish the archaic Tuscan language, in which it was written, as the standardized Italian language. Around the nature of the language used in Dante's compositions, it orbits the majority of the problems we encountered during our work, as we will further discuss later. The Divine Comedy is divided in three cantiche: Inferno, Purgatorio, and Paradiso, each consisting of 33 cantos, plus an initial introductive canto, for a total number of 100 cantos and 14,233 verses. The correlated productions are sometimes a combination of prose and verses. For the sake of coherence, we have chosen to use only the parts organized in verses.

2.2 Preprocessing

The files have been retrieved from the website liberliber.it, which provides a plenty of free texts also in .txt format. The productions we used are “Divina Commedia”, “Il Convivio”, “Il Fiore”, “Vita Nuova” and “Detto d’amore”. All the files have been cleaned from unuseful information, such as chapter numbers (both in arabic and roman format) titles, headings, notes and so on. Also, we stripped all the non-punctuation characters and accents, which have been replaced with the corresponding accented letter. Normally, the known related works also ensured to remove punctuation and accented characters, but instead we decided to preserve them, in order to test (and demonstrate) the strength of the model, even in a more complex situation. We finally converted the text in lowercase, in order not to duplicate words and subwords (e.g. “Nel” vs “nel” would be represented as two different tokens and probably obstruct the task) and saved them all in utf-8 so to both uniform the encoding and prevent reading errors.
What we obtained after the preprocessing are text files containing lines (verses) of text, divided by blank lines where necessary (in case of Divine Comedy we needed to keep the tercets split).

2.3 Tokenization

The normal approach in sequence-to-sequence models, in tasks as text understanding and text generation, and more in general in NLP domain, is to treat the text as a sequence of tokens, which generally are characters, subwords or words. Since the fundamental constraint is the creation of rhyming hendecasyllables, we needed to consider the single syllable of the text as an atom, as explained in the problem definition section.

The tokenization process starts by placing tags inspired from [3]: <S> for spaces, <V> and </V> at the beginning and ending of verses and <T> at the beginning of each tercet and ~ for the synalepha. We then transformed each word in its hyphenated version, by splitting each syllable by a space. The punctuation symbols have been separated from the words by a space, so to be treated as independent tokens. At the end we kept attention on synalepha (and automatically dialefa) by reprocessing the just tokenized text and merging the candidate syllables (as previously described). Since we knew the tonic stress of a small part of the vocabulary, we could exploit it as much as possible: in fact, the synalepha comes when the syllables are not stressed, otherwise we considered it as a case of dialefa and so without merging the vowels. This was easy enough when we knew the accent of the first syllable of the second of the two words. Of course, it is still an approximation, but this method generally showed a good performance, even if the new token composed of two (or more) syllables joined by synalepha went to make a total new token, which clearly becomes quite rare in the production.

```
<t> <v> nél <s> mèz zo <s> dél <s> cam min <s> di <s> no stra <s> vì ta </v>
<v> mi <s> ri tro va i <s> per <s> ù na <s> sél va~o scu ra </v>
<v> che' <s> la <s> di rìt ta <s> vì a <s> è ra <s> smar ri ta . </v>
<t> <v> à hi <s> quàn to~a <s> dir <s> qual <s> è ra <s> è <s> cò sa <s> dù ra </v>
<v> é sta <s> sél va <s> sel vag gia~e <s> à spra~e <s> fòr te </v>
<v> ché <s> nél <s> pen sier <s> ri no va <s> la <s> pau ra ! </v>
```

Figure 1: Example of the final tokenization of the first 6 verses of the Divine Comedy.

3 Models

3.1 Former models

The first approach was to use RNNs, in particular LSTMs, due to the need of keeping in memory the previous context while generating text. We soon realised that, in order to make the model understand rhymes, we needed to keep in memory the tokens of the two previous verses (which can generally be composed of 30-40 tokens). As we know, the longer the sequence, the harder becomes holding attention on specific parts of the text. We also tried to use a bidirectional LSTM but without good initial results, also because these models solve other problems, different from ours. LSTMs (and bidirectional LSTMs) are very good in text generation and there exist plenty of works in literature which obtain very good results in this task. In fact, what we generated with LSTMs was not bad in terms of structure of the text (we usually obtained tercets and some hendecasyllables) and goodness of words, but the rhymes problem was still there. Soon we realised that we needed a model able to enhance the focus of highly specific parts of the text and, more important, to find patterns and schemas, such as the terza rima. It is fundamental to underline the importance of this step. As shown by the related works mentioned above, an LSTM can easily find a rhyme in a tercet just by learning the n-th token, suppose the 33rd, to be equal to the 11th. This works for a single isolated tercet, but not when producing a general verse. We made some experiments with GPT-2, knowing it as the former state-of-the-art architecture (now ousted by GPT-3) in transfer learning for text generation. It was good in part: the structure was almost optimal and words were faithful enough, but there was not any trace of rhymes. This is because GPT-2 is only composed of decoders, so it is extremely performant in generating text but not so good at capturing the schemas (as terza rima) we were looking for. Once pointed this, we decided to implement a full Transformer architecture.

3.2 Transformer

Until recently, the dominant sequence transduction models were based on complex recurrent or convolutional neural networks that include an encoder-decoder structure, where the best performing models also connect the encoder and decoder components through an attention mechanism.

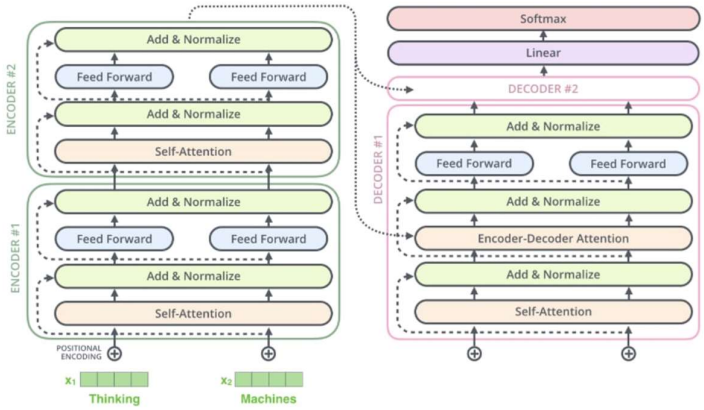

Figure 2: Transformer composed of 2 stacked encoders and decoders

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. Even though the Transformer was initially developed for sequence transduction tasks, it has been found to be a state-of-the-art model also for other tasks, such as sequence classification, sequence modeling and text generation [4].

4 Experiments

We tried to understand how to obtain the better result from the transformer, substantially by tuning the model's hyperparameters and trying many ways of tokenization and parsing of the text.

Tokenization: First, we tried to tokenize the text by characters and by words, without acceptable results. In fact, as previously explained, the only way to ensure the goodness of the hendecasyllables is to tokenize by syllables, in order to make the model understand the right number of syllables in each verse.

Tags: We tried different sets of tags. Initially we had the characters of start and end for both tercets and verses. We then noticed that even though just a begin (or end) tercet symbol is enough, it is better tagging either the beginning than the ending of a verse, because it seems easier for the model for generating equally long verses. Of course the definition of a tag (e.g. <T> rather than *tercet*) is not relevant, as all of them will then be translated into numerical codes.

Using productions: Ideally, doing a pre-training on other Dante's productions expands the vocabulary (more known syllables) and the relations among them, which means more data to be used as a base for the choice of which syllable follows another one, so to obtain more realistic words and less invented terms [3, 6]. However it should be taken into account that the style of the Divine Comedy is different and very peculiar, so that it could be cause of confusion if the model is not perfectly tuned. In our opinion, it is highly probable that with a more extensive exploration we would come up with a more efficient way to exploit the productions.

Using synalepha: From a pragmatic point of view the synalepha consists of merging two or more tokens into a single one. This means that when we meet the tokens A and B on which the synalepha can be applied, we obtain a new token A~B. It is trivial to understand how this operation enlarges the vocabulary, and how the new produced tokens becomes more rare in text with respect to the original couple. In fact, if we suppose the tokens A and B appearing in the training text several times and the synalepha to be applicable between them just one time, we would still obtain many occurrences of A and B and just one of the new token A~B. This can clearly influence the probability of A~B to be chosen during the generation, but we noticed that, despite so, the model is able to correctly place these tokens

in the right place. We also considered and applied the multiple synalepha, which is however even more rare.

Sequence length: Due to the structure of the Divine Comedy (tercets and terza rima), the only way to let the model learn the rhyming scheme was to choose a sliding window of at least four verses. More precisely we decided to split the Divine Comedy in couples of input-target where the input is composed by three successive verses and the target is composed by the input plus the successive verse. This way we train the model to generate new verses and to be auto-regressive in order to be able to take as input its own output and generate a whole new canto.

Epochs of training: As usual we started testing different numbers of epochs, thus discovering the most interesting results to be obtained with a number of epochs falling in the range from 70 to 150 epochs. We then tried to implement an interesting novel technique [7] consisting of using just one epoch of training while augmenting the dataset by repeating N times the dataset instead of using N epochs, each time with a different shuffling. Quite surprisingly, this improved our results beyond our expectations, drastically reducing the overfitting, accelerating the loss precipitation and thus decreasing by a lot the required time of training.

Model parameters: The tunable parameters are: number of encoder and decoder layers, *dff* (the number of neurons in the first dense layer of each feed-forward neural network), *d_model* (the dimension of the output of each layer), number of attention heads and epochs of training (one value for the training on the Divine Comedy and one for the pre-training on the other productions). Unfortunately, running an exhaustive gridsearch on Colab resulted in being infeasible (it would last a week of training or more, due to the imposed GPU usage limits), so we decided to try as many intuitive configurations as possible and then compare the results.

Generation procedure: The generation of a canto starts with giving as the initial input sequence the tokenized first tercet of the Divine Comedy. The model then proceeds with generating the new verse, token by token, until it produces the end-of-verse tag </V>. Once the new verse has been generated, it is appended to the previous input, while the first verse is dropped, so that the generation task proceeds in the same manner as the training process: the model is asked to predict the whole input (composed by three verses) plus the new verse.

Generation temperature: The temperature is a peculiar parameter in text generation models and more in general in neural networks, aimed to reduce the determinism of the production. In practice, it is a divisor of the probability distribution of the tokens, so that at higher temperatures the text is more unrealistic and unpredictable, while at lower temperatures the outcome is more faithful to the original text. For each model configuration we generated several productions, one for each different temperature in range [0.5, 1.5].

5 Results

5.1 Training procedure

After running several configurations of the model, we found the best performing ones to have *dff* = 256, *d_model* = 512. These first tests on the model also let us find out, quite surprisingly, that the pre-training on the other productions did not really lead to better results, thus showing that the best ones were obtained by training just on the Divine Comedy with 150 repetitions.
Keeping these considerations in mind, we decided to fix these values and train other models with different combinations of encoders and decoders in order to find the best ones.

5.2 Metrics

To evaluate the goodness of a generated canto we decided to use the following metrics:

1. **rhymes**: the percentage of correct rhymes in the canto.
2. **hendec_ratio**: the percentage of correct hendecasyllables in the canto.
3. **hendec_correctness**: differently from hendec_ratio, this metric keeps into account the distance between the number of syllables of the verse and the correct amount.
4. **word_correctness**: the percentage of correct words in the canto.

As already mentioned, the first three metrics are influenced by the lack of information about the tonic vowels.

5.3 Best models

top of word_correctness:

id	model	temperature	word_correctness
44	3_1_256_512_4_0_150	0.5	0.78
202	7_1_256_512_4_0_150	0.9	0.77
221	7_5_256_512_4_0_150	0.6	0.77
220	7_5_256_512_4_0_150	0.5	0.76
224	7_5_256_512_4_0_150	0.9	0.76

top of rhymes:

id	model	temperature	rhymes
36	1_7_256_512_4_0_150	0.8	0.984
143	5_5_256_512_4_0_70	0.5	0.969
25	1_5_256_512_4_0_150	0.8	0.969
114	5_3_256_512_4_0_150	0.9	0.969
169	5_5_256_512_4_70_70	0.9	0.969

top of hendec_correctness:

id	model	temperature	hendec_correctness
219	7_3_256_512_4_0_150	1.5	0.94
212	7_3_256_512_4_0_150	0.8	0.88
54	3_1_256_512_4_0_150	1.5	0.82
241	7_7_256_512_4_0_150	1.5	0.81
52	3_1_256_512_4_0_150	1.3	0.81

top of hendec_ratio:

id	model	temperature	hendec_ratio
44	3_1_256_512_4_0_150	0.5	1.00
83	3_5_256_512_4_0_150	1.1	0.97
68	3_3_256_512_4_150_150	0.7	0.97
37	1_7_256_512_4_0_150	0.9	0.97
154	5_5_256_512_4_0_150	0.5	0.96

Figure 3: Table of the best models for each metric.

In order to be able to draw useful conclusions about the different models, we inspected the best models according to each metric. Figure 3 shows the top five models for each metric and the name of each model is given by the list of its parameter separated by an underscore, more precisely:

1. *number of encoders*
2. *number of decoders*
3. *dff*
4. *d_model*
5. *epochs_production*
6. *epochs_comedy*

Finally, to establish the best models we simply sum the computed metrics, but without taking into account the hendec_correctness, in order to avoid giving too much weight to the hendecasyllables. Figure 4 shows the top five models.

id	model	temperature	struct	rhymes	hendec_ratio	word_correctness
154	5_5_256_512_4_0_150	0.5	1.0	0.922	0.96	0.65
143	5_5_256_512_4_0_70	0.5	1.0	0.969	0.90	0.66
155	5_5_256_512_4_0_150	0.6	1.0	0.922	0.96	0.63
169	5_5_256_512_4_70_70	0.9	1.0	0.969	0.93	0.61
25	1_5_256_512_4_0_150	0.8	1.0	0.969	0.93	0.58

Figure 4: Table of the best models.

6 Conclusions

Basing on the results we obtained, we have come to the following conclusions:

- number of encoders and decoders:
    - As far as the word correctness and hendecasyllables correctness are concerned, we noticed that a greater number of encoders allow the model to better understand the relations among tokens in general, finding patterns and schemas and thus improving the correctness of the words and the number of syllables between each verse token <V> and </V>.
    - A better rhyme schema is usually achieved with a balanced combination of encoders and decoders, not showing any particular relations between the two components and the obtained rhymes.
    - The worst results were achieved by using any highly unbalanced number of encoders and decoders (e.g. 7-1 appeared on top of rhyming scores but in reality the text generated is really repetitive and the structure of the tercets has been completely lost).
    - Our results do not show any particularly good configuration for the number of decoders.
    - In general we can finally state that a good model should be composed of an even number of encoders and decoders, or at least a quite balanced combination of both.
- number of epochs:
    - The number of epochs does not seem to be relevant as long as it is chosen between an interval from 70 and 150.
- temperature:
    - The results do not show any particularly well performing temperature value. By looking at Figure 3, we can clearly see that a high temperature only improves the hendec_correctness metric (and yet, it is difficult to say why). Intuitively, a good solution might be to remain strictly below the value 1.0 (as shown in Figure 4).

7 Future Works

Clearly the metrics we provided have been thought to merely evaluate our results basing on the other types of evaluations used by other students. A deeper and more accurate study of which factors are relevant is necessary in order to better determine differences between various models.

At the moment the pretraining on the other Dante's production does not seem to have led to any particularly good result, but further tests definitely need to be done. This consideration is also supported by several literatures, which generally state that a bigger variety of data usually improves the model capability of generalization and thus improving the text generation when the model is fed with its own generations and not the training data.

Another clarification to be done regards the beginning of the generated text and a small overfitting when forecasting the initial verse, which corresponds to the 4th verse, given the first tercet of the Divine Comedy as prompt. Our assumption is that the model might be overfit, and thus probably leading to a worse general performance.

One of the most important aspects to be improved is the pre-processing step on the original text, so the hyphenation procedure. Given a perfect hyphenated text (as well as right information about tonic accents) we are confident enough the model could produce an even better text.

Last but not least, testing other *dff* and *d_model* parameters could definitely further improve the general quality of the generations. To this purpose an exhaustive grid search cross validation process could be run in order to achieve the best possible model.

8 References

[1] Xie et al. (2017). Deep Poetry: Word-Level and Character-Level Language Models for Shakespearean Sonnet Generation - https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1174/reports/2762063.pdf

[2] Lau et al. (2018). Deep-speare: A joint neural model of poetic language, meter and rhyme - arXiv:1807.03491v1

[3] Zugarini et al. (2019). Neural Poetry: Learning to Generate Poems using Syllables - arXiv:1908.08861v2

[4] Vaswani et al. (2017). Attention is all you need - arXiv:1706.03762v5

[5] italian-dictionary API - https://pypi.org/project/italian-dictionary/

[6] Gero et al. (2018). Transfer Learning for Style-Specific Text Generation.

[7] omatsuzaki (2019). One epoch is all you need - rXiv:1906.06669v1

[8] Santhanam (2020). Context based text-generation using lstm networks. - arXiv:2005.00048v1