First Steps Towards Collaborative Poetry Generation

David Uthus, Maria Voitovich, R.J. Mical, and Ray Kurzweil
Google Research
{duthus,mvoitovich,gameman,raykurzweil}@google.com

Abstract

We present our initial research on creating an application that gives users the opportunity to compose poetry in collaboration with an AI. Our goal is to allow for a user to have full control of the flow of the poem. When composing the poem, the application will provide multiple suggestions for the next possible verse given the previous verse, which the user may then use or write one of their own. This differs from many other generative approaches that have been more focused on generating poems with little to no user interaction. In this paper, we will explain how the verses are generated and how the application determines which generated verses to suggest as next possible verses.

1 Introduction

We are interested in creating a tool that empowers those interested in poetry to compose a poem in collaboration with an AI. When composing a poem, we want to offer suggestions to a user of possible next verses, and then allow the user to choose which verse they feel like best continues the poem or to write their own. Even when writing their own verse, they may be inspired by the suggested verses.

Many past approaches have focused on generating a poem in full with minimal user interaction [3]. Hafez [1, 2] has been one approach though that has allowed for some human interaction. While it did not allow for users to pick or write verses, it did offer a variety of inputs for users to dictate how a poem was generated (e.g., topic; desired words; control for sentiment, alliteration, etc.). Users could then further tweak the controls until a poem was generated to their liking. Also relevant is DopeLearning [6], an interactive approach for generating rap lyrics. It allowed finer interaction for composing a rap song – for each verse, a user could either enter their own input or pick from a list of existing rap lyrics that are ranked given the previous verse.

In this paper, we will give a brief overview of our system – how the verses are generated, how we determine which verses to suggest to the user, and discussions on future work.

2 Approach

As a high-level overview, a user begins to compose a poem with the AI by writing the first line of verse. Following this, we then allow a user to either write the next verse, choose from a list of AI suggestions, or edit one of the AI suggestions. The suggestions are influenced by the previous verse and the structure of the poem that the user is making (e.g., number of syllables, rhyming schema).

2.1 Generation

For generation, verses are generated offline and stored for later serving. Offline generation allows us to suggest verses to a user quickly, with more detail in Section 2.2.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

Cats playing in the light of the moon, *(Human)*	I see ships in the water, *(Human)*
The sound of my life, my soul in tune, *(AI)*	Roots in the path of the sea, *(AI)*
As music flows through the dying night, *(Human)*	Batman riding a dolphin up high, *(Human)*
There is a song to the morning light. *(AI)*	Mad in the path of the scene of life. *(AI)*

Figure 1: Two example poems composed with our application. The first uses an AABB rhyme schema with 9 syllable count while the second is without restriction on rhyme or syllable count.

To begin with, for source material we use the poems of classical American poets (e.g., Walt Whitman, Emily Dickinson), with most material found on Gutenberg[1]. These original verses are broken into trigrams and grouped into three sets – *starting trigrams* marking the start of a verse; *ending trigrams* marking the end of a verse; and *middle trigrams* composed of trigrams that come in between. We then create all possible permutations for a given poet – we start with the *starting trigrams*, find all possible *middle trigrams* that overlap by two tokens, and iterate until we find *ending trigrams* that overlap with the partially-constructed verse. All generated verses are novel – we check that no generated verses are found in the original corpus.

As we are using trigrams for generation, many generated verses suffer from poor grammar or are nonsensical. To overcome this, following the generation of all feasible verses, we train classifiers to remove many of these poorly-constructed verses. The classifiers are trained per-poet (to help capture their writing style), using a poet's original verses as "good" verses and a random sampling of generated verses as "poor" verses. This helps filter out a substantial number of poorly-generated verses.

2.2 Recommendation

As mentioned earlier, we want to suggest to the user multiple verses given the previous verse. To do so, we use a dual encoder model to recommend the next possible suggestions. This is similar to the model used in Gmail's Smart Reply [5]. In our dual encoder model, one encoder is used to encode a parent (previous) verse and the other encoder is used to encode a child (next) verse. The model is then trained to optimize the dot product similarity between the current verse and all possible next verse within a batch. For training data, we use a variety of sources including English poems found in Gutenberg, modern song lyrics, and Reddit.

After all verses have been generated, they are then encoded with the child encoder of the dual encoder model, annotated with metadata (such as the rhyming phonemes), and finally indexed. For serving, we encode the previous verse with the parent encoder and then retrieve the best possible next verses using a hierarchical quantization approach to search through all possible verses [4, 7].

Our motivation for this approach is to allow us to train the dual encoder model on a large corpus of data beyond the original poems used for generation. This allows the model to gain a semantic understanding of a wider variety of subjects.

3 Discussion and Future Work

Figure 1 gives two example of recent poems that we composed in collaboration with our AI. As can be seen, our initial approach can make poetic verses that respect rhyme and syllable count. More importantly, it showed some semantic understanding of what verses to suggest next, for example associating "music" with "song", "night" with "light", and "water" with "sea".

In the future, we plan to explore replacing the trigram concatenation generation with a generative model. The trigram concatenation generation results in too many poor verses, and has found to be restrictive when working with poets whose writing style offers few overlapping trigrams. We would still generate and index offline to allow for faster serving, but we hope that by adopting a generative model approach that it will lead to better quality and diversity of verses generated.

We also plan to start doing human evaluations. As seen in many recent works on poetry generation, human evaluation is often needed due to the subjective nature of poetry. We plan to use these evaluations to both just the quality of the verses themselves along with the ranking of suggestions.

[1] http://www.gutenberg.org/

References

[1] Marjan Ghazvininejad, Xing Shi, Yejin Choi, and Kevin Knight. Generating topical poetry. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1183–1191. Association for Computational Linguistics, 2016.

[2] Marjan Ghazvininejad, Xing Shi, Jay Priyadarshi, and Kevin Knight. Hafez: an interactive poetry generation system. In *Proceedings of ACL 2017, System Demonstrations*, pages 43–48. Association for Computational Linguistics, 2017.

[3] Hugo Gonçalo Oliveira. A survey on intelligent poetry generation: Languages, features, techniques, reutilisation and evaluation. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 11–20. Association for Computational Linguistics, 2017.

[4] Ruiqi Guo, Sanjiv Kumar, Krzysztof Choromanski, and David Simcha. Quantization based fast inner product search. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51, pages 482–490, Cadiz, Spain, 09–11 May 2016.

[5] Matthew Henderson, Rami Al-Rfou, Brian Strope, Yun-Hsuan Sung, László Lukács, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. Efficient natural language response suggestion for smart reply. *CoRR*, abs/1705.00652, 2017. URL `http://arxiv.org/abs/1705.00652`.

[6] Eric Malmi, Pyry Takala, Hannu Toivonen, Tapani Raiko, and Aristides Gionis. DopeLearning: A computational approach to rap lyrics generation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 195–204, 2016.

[7] Xiang Wu, Ruiqi Guo, Ananda Theertha Suresh, Sanjiv Kumar, Daniel N Holtmann-Rice, David Simcha, and Felix Yu. Multiscale quantization for fast similarity search. In *Advances in Neural Information Processing Systems 30*, pages 5745–5755. 2017.