Machine Learning for Creative Means: Using Natural Language Generation to Create Poetry

Naima Karzouz

INTRODUCTION

Using computer science for creative means is nothing new- since the creation of computers, humans have used them to express themselves. One of the earliest forms of computer speech synthesis, the IBM 704 singing the song 'Daisy Bell', was done in 1961, taking one hour to play seventeen seconds of melody, then using an audiotape to speed the song up. This was done simply for the sake of 'converting analog information into digital forms', and laid the groundwork for most music and sound software (O'Dell, 2009). This example is one of many which demonstrates how a creative pursuit using computer science can lead to a breakthrough in what technology is capable of. By seeing how machine learning constructs poetry, we can also gain insight into humanity, and how human creativity is 'generated' organically. As such, my project follows in the tradition of using natural language generation for creative means, processing data from the Poetry Foundation organization to try and create new, coherent poems. I have based my approach off of Tim Van de Cruys' "Automatic Poetry Generation from Prosaic Text", which uses RNNs in an encoder-decoder configuration(Van de Cruys, 2020).

There are three critical challenges when creating generated poetry, the first being that creativity is fundamentally difficult to replicate. Unique combinations of text with meaning are complex for even humans to create, and what makes a work 'creative' is subjective. Following from this, what is and isn't considered poetry is subjective - not all people agree on what makes a poem. For instance, some only qualify poetry with a rhyming scheme to be poetry, or consider poetry in stanzas to be 'correct'. Therefore, what typically qualifies a poem is that its poet considers it to be poetry- when creating poems synthetically, from a computer program, this distinction becomes ambiguous, as the program (the poet) does not classify, it only generates. Lastly, the ambiguity of poetry lends itself to making it difficult for a machine learning model to learn what a poem is. Not all poetry has a confined structure and, therefore, the commonalities of poetry are harder to define. The main subsection of data I used was poetry created by William Shakespeare, as he has a uniform style for his poetry, however I have included other experimental results from other subsections of the dataset to demonstrate how broad poetry generation can become.

In this paper, I will first summarize the previous approaches to generating poetry using natural language generation. Then, I will demonstrate my algorithm, and elaborate on how it works, as well as its implementation. Lastly, I will discuss a few examples of the generated text, as well as the results of using different subsections of the data.

BACKGROUND

The two main papers used as literature review for this paper are "Automatic Poetry Generation from Prosaic Text" by Tim Van de Cruys and "Deep-speare: A joint neural model of poetic language, meter, and rhyme" by Jey Han Lau, Trevor Cohn, Timothy Baldwin, Julian Brooke, and Adam Hammond.

"Automatic Poetry Generation from Prosaic Text", published in July 2020, proposes two models, Charles and Sylvia, which take in prosaic text (commonplace text without poetic intent, such as news articles) and feeds it through recurrent neural networks containing an encoder-decoder configuration. The result is both French and English poetry that follows a strict rhyming scheme, and is nearly indistinguishable from human-written poetry. This is done by adding 'poetic constraints' to the output of the neural network: a rhyme constraint and a topical constraint. The rhyme constraint was done by first generating the concluding rhyming word, then generating the rest of the line- for instance, generating the word 'embrace' to rhyme with the previous word 'suitcase', and then generating the rest of the sentence based off of the word 'embrace' to make sure that the word would make sense in context(Van de Cruys, 2020). This was done by grouping words together by their phonetic representation, such that the rhyme sound was determined by the last phonetic sound in a word. The topical constraint was used to combat the generated prosaic results, which wouldn't be considered poetic. Van de Cruys accomplished this by creating a matrix of topical dimensions, such that the network's output is constrained to a distribution with low entropy, such that the neural network is confident that the next word fits with the topic. If the entropy is high, however, the distribution is modified in order to 'force' the desired topic. The data used is pulled from the CommonCrawl corpus, which was preprocessed to make sure it was constrained to either French or English (for the respective model), limited to 20 words per sentence, and each sentence must occur within a broader context (following another sentence). The poems were evaluated using human annotators, which would assign a point value (one to five) on four characteristics: fluency (grammatical and syntactical fluency), coherence (thematically structured), meaningfulness, and poeticness ('does the text display the features of a poem') (Van de Cruys, 2020).

"Deep-speare: A joint neural model of poetic language, meter, and rhyme" (henceforth referred to as 'Deep-speare') was proposed in 2018, amid an increase in interest in deep learning, and proposed a model which generates sonnets and quatrains (stanzas of four lines, with alternating rhymes) in iambic pentameter (a rhythm constraint of unstressed then stressed syllables) (Lau et al., 2018). Deep-speare, the language model, is composed of three separate neural models. The first, a language model, is used to predict the next word, which helps to train the architecture into what words are 'acceptable' in a Shakespearan sonnet. The second model is a pentameter model, which learns the alternating stress patterns (the aforementioned unstressed then stressed syllables). The last model is the rhyme model, which figures out what portions of the quatrain rhyme. 'Deep-speare' generates poetry one word at a time in order to maintain iambic pentameter, and then enforces the rhyme scheme after the sentence has been created. The poems were then evaluated by literature experts.

Of these two proposed methods, Van de Cruys's model greatly outperforms Deep-speare, as it is not confined to Shakespearan sonnets, and makes use of encoders/decoders. Van de Cruys' model is able to generate more 'authentic' poetry, in that it takes in everyday concepts and then generates poetry, where Deep-speare is limited to its learned Shakespearean parameters. Where both have a high computation cost, Van de Cruys' recurrent neural encoder-decoder architecture is particularly taxing due to the constraint system, however, based on human evaluation, over half of the poems were deemed 'written by a human' when they were generated by the model. Deep-speare, by contrast, lacked emotional impact and readability, and its rigorous conformity to iambic pentameter meant that it failed human evaluation. Based on Van de Cruys's model, I researched LSTM natural language generation models, and based my method on it. While I do not factor in the constraints that either paper proposes, I did attempt to keep topical coherence using a 'query word', which is passed to the decoder and is used to start generating the line to try and force context. Additionally, a few of my experiments used the Shakespearean subset of the Poetry Foundation, to which I was interested in seeing if the encoder-decoder architecture could mimic 'emotional impact' without strictly adhering to the iambic pentameter.

METHODS

In order to realize a poetry generator, I heavily relied on the Keras and Tensorflow libraries. I used two main subsections of the Poetry Foundation dataset- poetry authored by William Shakespeare and poetry authored by anonymous sources. I first pre-process the data by choosing which source I'd like to pull from (anonymous or Shakespearean), then I combine all of the distinct poems by that author into one large array. Then, the data is processed to include <sos> and <eos> tags, "start of sentence" and "end of sentence", respectively. From there, I experimented with different amounts of epochs, batch_sizes, and validation splits.

The core of my program is the LSTM (Long Short-Term Memory) network, which processes sequences of words relative to one another. I first create the Keras embedding layer, passing in the number of words used for the input_dimension based off of the SequenceGenerator() class (a class which essentially streamlines word embeddings and tokenizing sentences, using GloVe's word-embeddings), the dimension of the dense embedding, and the weights. The two internal states, the hidden state and cell state, are then initialized, such that they can be used for the encoder. The embeddings are then passed to the LSTM, which is then fed to a dense layer. I've used the softmax activation function in order to gain the probabilities of each word in the vocabulary. Finally, this is composed into the Encoder model. Then, the layers for the decoder are initialized, using the same embedding layer and LSTM layer. The Encoder is then compiled using the categorical cross-entropy loss function, and, of the other optimizers I tested, the Adam optimizer works best. The model is then trained on the input sequences and state vectors.

To generate the poem from the model, one line of poetry is generated at a time. First, the <sos> (start of sentence) token is passed in, then the model predicts the next word until it either reaches the <eos> (end of sentence) token or it reaches the maximum sequence length. Until it does so, it appends the predicted word to the output, and then returns the sentence. This is looped for as many times specified - when testing, I typically generated five lines. Because of this structure, the lines may greatly vary in subject, and there is no definitive rhyme scheme. Despite this, because of word2vec's capability for finding similar sentences, it is possible to have a somewhat 'themed' poem by inputting a query_word- the success of which is dependent on if the query word appears in the text. In creating this natural language generation model, I've learned a great deal about LSTMs and encoders/decoders and, were I to further this project, I would look into replicating Van de Cruys's rhyming scheme model and adjusting the context-based query word.

EXPERIMENTS

The majority of this code was run in Google Colab, with mixed success, as I had to greatly shorten the number of epochs and the batch size in order to keep from running out of RAM. As such, I shortened the parameters from an ideal 250-500 epochs to a mere 4-10 epochs, and lowered the batch size from 64 to 8. Upon further reflection, I should've figured out how to save the encoder-decoder model, but, instead, I would individually run them to re-train when the runtime disconnected, and run the last bracket of code to re-generate a new poem. For my presentation, I'd trained my model with 4 epochs and a batch size of 8 with the Shakespearean data, which resulted in semi-coherent fragmented poems, such as "she's man would, men- note wound their by burning not lov'd" and "the tongue. / be bloody / the forlorn / she's thousand buried".

Trained on the 'Anonymous' authored data, it appears that a majority of it is in old English, and, due to the sheer size of the corpus, in order to get it to run, I had experimented with splitting it into 10ths, 50ths, and 100ths and using the first (index position 0) bracket of data. The parameters for this trained model included 20 epochs with a batch size of 8, but the results, being in old English, were nearly incomprehensible.

Lastly, I trained my model on poetry authored by "Alfred, Lord Tennyson", who has the third most entries in the dataset. For this, my training parameters were 19 epochs with a batch size of 8, and it resulted in the most 'modern' set of poems, included below:

> "king twined that "this with my craven seeks the rose, once more
> to many-tower'd and thou, to deep peace
> and that by and mingle now? bud the turrets of the world
> and blurr'd and iii tristram, and to many-tower'd and thou, over and
> and blurr'd sweet that oft the turrets"
>
> "what comfort thyself: what comfort thyself: what comfort thyself: what comfort my
> 'the from on my heart with flowers faith, a joust and a
> shrill, puppet belted furnace sorrow's things, my mission which i see, in
> shrill, puppet belted furnace somewhere goat in mine." want what comfort in
> bridal, to aftertime, coldness and sundered.
> bridal, to aftertime, coldness and sundered. and will not in wet we
> contemplate ere"

While I did not make any true breakthrough or conclusion, I do believe that, in working through this problem, it demonstrates a need for consistent poetic data, or, otherwise, rigorous model creation like Van de Cruys is needed. With consistent data, such as Shakespeare's poems, one can mimic the fragmented sentences, but they lack comprehensive meaning. Upon review of the data, the Poetry Foundation's data is highly skewed towards older English, meaning that it's harder to assess whether or not the generated poems are truly readable, or simply read as poetic because of their archaic language. The generated poems, no matter how long the epochs are, tend to remain grammatically incorrect from a modern perspective.

CONCLUSIONS

Overall, the greatest benefit I have gained from this project is how to proceed with Natural Language Generation. I believe that, while the results of my project are not nominal, they are interesting, and do prove my point in that it is difficult to distinguish what is and isn't poetry. While my model does not always generate coherent poems, it does occasionally generate interesting sentences, which is a good foothold for future research. Regarding future work, given more time, I would like to pursue data from other sets, such as web-scraped quotes from other authors or song lyrics, which would be pulled from more modern poets, and I'd like to refine my model to include a rhyme scheme, such as the ones included in Deep-speare and Van de Cruys' work. Additionally, I would like to look into how to save an encoder-decoder model, such that I would not have to retrain the model every time the runtime ran out, or I had a RAM issue.

CITATIONS

Lau, J. H., Cohn, T., Baldwin, T., Brooke, J., & Hammond, A. (2018). Deep-speare: A joint neural model of poetic language, meter and rhyme. *arXiv preprint arXiv:1807.03491*.

O'Dell, C. (2009). Daisy Bell (bicycle built for two). Library of Congress. Retrieved December 15, 2021, from https://www.loc.gov/static/programs/national-recording-preservation-board/documents/DaisyBell.pdf

Van de Cruys, T. (2020). Automatic Poetry Generation from Prosaic Text. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.acl-main.223