|  |
| --- |
| **Neural Language Model Song Generation** |
|  |
| **Mason Heaman Danielle Phillips** |
|  |
|  |
|  |
|  |

Abstract

In this study, the neural language model GPT-2 and bi-gram baseline are utilized as methods of song lyric generation. The neural language model is trained with the goal of generating language representing the complex nature of songs. This model takes on an even more challenging task as it is trained to replicate a certain genre of music. As expected, lyrics generated with the neural language model drastically outperform those of the baseline in terms of complexity. In the generated samples, bigram generations obtain an average perplexity *P* = 975.23 compared to an average *P* = 61.27 in GPT-2 generation. The authors focus their work on a generated corpus of USA's Top Classic Rock Songs, a bi-gram baseline, a trained GPT-2 model, and generated lyrics.

Introduction

Song lyrics and poetry provide a complexity that is arguably higher than other forms of conversational or informative literature. This complexity is due to the common use of metaphors, clever rhyming, and repetition. This study aims to discover the extent to which a neural language model can understand these characteristics. Using this understanding this model will attempt to generate language that falls into the song lyric format. Furthermore, the classification of songs into genres provides an interesting question for our study. Can a model be trained to not only replicate the structure and complexity of a song, but also to generate language that represents a specific genre? If so, song writers may be provided an empirical measure of the structure and ideals behind a genre of music. This may also present a method of analysis for studies investigating the societal biases supported within a given genre of music. Therefore, this study will benefit both academia and lyricists. At a high level, the approach will be broken into four key processes. Creating a corpus, establishing a baseline, training a neural language model, and lyric generation and analysis. This structure proved to be successful and should be followed with further extensions to this study. ­­

Related Work­

In order to understand the proper use of our model for this project, we did some research and came across a GitHub page from OpenAI that provided information we used for a large majority of this project. Not only did it include the needed files to download the model, but incorporated reading to learn about GPT-2. Solaiman (2019) and other authors who contributed to OpenAI’s final blog post, *GPT-2: 1.5B Release*, gave a brief description of their findings when working with the model after a nine-month period. This language model was made to serve the purpose of obtaining a better understanding of behaviors, capabilities, biases, and constraints of large-scale generative language models (Clark, 2019). The GitHub page assisted us to have a deeper understanding behind the use of the GPT-2 model and applying it to our song generator.

As for our libraries, we pulled song lyrics from the Genius API Python libraries to play a part in retrieving the lyrics for each song and sending the data to a generated corpus. To assist us in properly using this library, we read Raizel Bernstein’s (2020) *How To Access and Use the Spotify and LyricGenius API* blog to walk us through the steps. Throughout her article, Bernstein provided many examples and graphs to properly cite song lyrics to implement into the project.

Approach

The approach follows a structure that may be broken into four distinct sections. Creating a corpus, establishing a baseline, training a neural language model, and lyric generation and analysis.

In order to train both the baseline and neural language model, a large corpus of song lyrics must be created. This study chose to focus on a singular genre to allow the model to learn its characteristics over an effective period of time. The genre chosen was American Classic Rock (ACR). Song titles and artists were collected from a source of the 1000 most popular ACR songs[1]. Using the collected song identifiers, the Genius API was then utilized to efficiently collect lyrics for each available song. The available song lyrics (*N*=932) were written to our corpus and separated by the text delimiter “<|end of text|>”. Collection resulted in respective line and token counts (*L*=44,571; *T*=381,700).

To establish a baseline from which to compare the state-of-the-art neural language model, a bigram model was produced from the lyric corpus. In order for bi-grams to be calculated correctly, preprocessing the text by eliminating the text delimiter “<|end of text|>” and punctuation (! , . ? etc.) was a necessity. All bigrams were collected from the dataset. The conditional probability of each bigram was then calculated.

Experiments

Using the ACR lyric corpus, language generations were made with both baseline and state-of-the-art language models. Different methods were taken for each model to receive a similar format of generation.

Four attributes were adjusted to ensure that bigram generations followed the structure of song lyrics: *stanza number, stanza length, line length,* and *top\_k*. While the first three attributes control the physical structure of the generated lyrics, *top\_k* will control diversity. This attribute will determine how many words will be considered when determining the next word to generate. As an example, with a *top\_k­*=10, the next word to be generated will be randomly selected from the top 10 most probable words that follow the previous. Attributes were randomized or set in the following matter to provide a varying structure (*SN*=2-5*; SL*=2-10*; LL*=8) while *top\_k* was set at 25, 50, and 75 to find the optimal diversity in respect to perplexity. Generations (*N*=20) were then made with the noted parameters using the following method:

1. Begin sentence with a random token from corpus
2. Use bigram probability distribution to find top\_k most likely tokens to follow
3. Select a random index of the most likely tokens and append to the sentence
4. Return to step 2. until target length

GPT-2 proved to be capable of learning the physical structure of songs, separating output into intuitive lines and stanzas. Therefore, two attributes were experimented with to provide varying outputs from the model: *temperature* and *top\_k*. Temperaturecontrolling randomness in the Boltzmann distribution, and *top\_k* controlling diversity similar to the bigram model. To briefly describe the *temperature* attribute, a lower value will result in a more confident model while a larger value will produce a smoother probability distribution[2]. There are pros and cons to be seen with a high or low temperature. For a low temperature, the model will be more confident on which tokens are likely to follow the previous, however, this will result in less diversity as the model is less likely to sample from unlikely candidates. Conversely, a high temperature will result in a much smoother probability distribution that will allow for a more diverse set of possible candidates, but this heightens the probability of a word being chosen that does not logically follow the previous. The *top\_k* will be similarly set at 25, 50, and 75 in hopes of establishing a similar diversity to that of the baseline model. GPT-2’s default *temperature*=1 was chosen with the same goal.

Results and Discussions

As mentioned, *top\_k* values were varied on both models at values of 25, 50, and 75. This variation was done in order to find the *top\_k* that minimized perplexity in generated samples. Perplexity was calculated using \*\*\*\*\*\*\*\*\*\*\*\* The results of the experiment are documented below, with the inner-most values being perplexity:

Chart, treemap chart

Description automatically generated

It can be concluded that an increase in *top\_k* generally results in a higher perplexity, especially in the bigram model. These results are to be expected, as a higher *top\_k* results in a larger set of possible tokens to follow the previous. Therefore, there is a higher probability of choosing a token that is not a likely successor.

Conclusion

If there was more time available, spending time on fine-tuning the GPT-2 model would be the next challenge. Before fully analyzing and calculating probabilities/perplexities for ACR, the authors were attempting to use a corpus that contains top N songs of different decades using *SpotiPy*. Revisiting the original idea and having more time to dive deep into that project is something to consider!

Future Work

If there was more time available, spending time on fine-tuning the GPT-2 model would be the next challenge. Fine-tuning consists of making small adjustments to the model to improve the results. With the GPT-2 model, there are numerous ways to fine-tune it as there are methods available to call in code. OpenAI released an article where they fine-tuned the GPT-2 model based on human preferences by asking human labelers for the best and accurate result. This would be great to consider for the project when making the comparison between the GPT-2 model and bi-gram baseline.

Before fully analyzing and calculating probabilities/perplexities for ACR, the authors were attempting to use a corpus that contains top N songs of different decades using *SpotiPy*. Revisiting the original idea and having more time to dive deep into that project is something to consider! \*different genres, provide name and artist\*

Alfred. V. Aho and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation and Compiling, volume 1*. Prentice-Hall, Englewood Cliffs, NJ.

American Psychological Association. 1983. *Publications Manual.* American Psychological Association, Washington, DC.

Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. 1981. [Alternation](http://dl.acm.org/citation.cfm?doid=322234.322243). *Journal of the Association for Computing Machinery*, 28(1):114-133. https://doi.org/10.1145/322234.32224.

Association for Computing Machinery. 1983. *Computing Reviews*, 24(11):503-512.

James Goodman, Andreas Vlachos, and Jason Naradowsky. 2016. [Noise reduction and targeted exploration in imitation learning for abstract meaning representation parsing](http://aclweb.org/anthology/P16-1001). In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 1–11. <https://doi.org/10.18653/v1/P16-1001>.

Dan Gusfield. 1997. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, Cambridge, UK.

Mary Harper. 2014. [Learning from 26 languages: Pro- gram management and science in the babel program](http://aclweb.org/anthology/C14-1001). In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*. Dublin City University and Association for Computational Linguistics, page 1. <http://aclweb.org/anthology/C14-1001>.

Alexander V. Mamishev and Murray Sargent. 2013. *Creating Research and Scientific Documents Using Microsoft Word*. Microsoft Press, Redmond, WA.

Alexander V. Mamishev and Sean D. Williams. 2010. *Technical Writing for Teams: The STREAM Tools Handbook*. Wiley-IEEE Press, Hoboken, NJ.

not be asked to review the supplementary material.