Chorus Generation Using Language Models

Krutika Ingale, Amangeet Samra
UC Berkeley MIDS

Abstract:

Machine Learning has revolutionized the music industry significantly over the past decade. For instance, artists and producers now have access to technology that allows them to easily put together instruments and lyrics to produce beautiful melodies. In this project, we explore the idea of a type of lyric generation using multiple iterations of masked language models, where we aim to predict the chorus of any song given the rest of the verses in that song. We start off with a T5 model for our baseline, and then later incorporate a BART model as our final model. We are exploring whether it is possible to create one model to generate lyrics across several genres, because the majority of previous work has been genre specific.

1. Introduction:

In this project, we aim to predict the chorus of any given song by reading in the rest of verses and masking out the chorus. This project is important because it has numerous applications across the music industry, allowing artists to have a baseline for their lyrics. This will specifically help with writer's block and provide artistic direction to the song writer. While we are focusing solely on chorus generation, this idea can easily be applied to any portion of a song, and when taken more broadly, it could be applied to any piece of text (books, magazine, blogs, etc.). This project is challenging, because bulk chorus generation has not been done before. The problem is also difficult, because the models not only have to learn to interpret song lyrics, but also output meaningful lyrics for the chorus. Evaluation is especially difficult, because there currently is not a gold-standard metric to qualitatively measure whether text to text generation is accurate. In our case, as we already have our target values, we will be using the Rouge metric to score the similarity between the original target chorus and the model's generated chorus. In the future, we can build off of our research and develop a tool that will allow a user to input an entire song of verses and indicate where the choruses should be, and our model will generate the choruses for that song in order.

2. Background

A plethora of research has already been conducted on lyric generation models, and the LSTM model has proven to be among those most commonly used. In their paper *Deep Learning in Musical Lyric Generation: An LSTM-Based Approach*, researchers designed an LSTM model that would read in lyrics to songs of a certain genre and produce the next line of the song. Their research focused on genre-specific lyric generation. However, our focus was to see if it is possible to predict choruses well, without specifying the genre. Additionally, because the LSTM model has been so thoroughly explored for applications such as this, we chose not to use this model because we wanted to go in a different direction than the established works.

We also looked into using a Recurrent Neural Network, which is another model that is frequently used for lyric generation. We looked at *Generation of Hip-Hop Lyrics with Hierarchical Modeling and Conditional Templates*, where researchers developed a Recurrent Neural Network model to predict hip hop lyrics. And again, this research was limited in that it was genre specific and our goal was to broaden these genre specific models to see if training a model across different genres and artists was a possibility. We decided against this model because our aim was to do chorus generation and we wanted to feed in multiple lines of lyrics to the model rather than read them in one at a time, which is something an RNN model is not optimized for.

In order to achieve generalized lyric generation, we decided to use the base T5 model (T5-base) as our baseline model. This decision was made based on research conducted by the authors of *Say What? Collaborative Pop Lyric Generation Using Multitask Transfer Learning*. In their project, they introduced the idea of using a T5 transformer model for lyric generation, which up until their research, had not been used for that purpose. While their project focused solely on generating lyrics for pop songs, with the help of artists in the music industry, researchers were able to create a T5 model that generated lyrics with the nuances of human song writing. We saw the potential to take this one step further in order to see how removing the genre aspect of the generation would affect the model's performance.

In 2019, there was a new text generation model introduced by researchers in their paper *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. The authors described this transformer-based model as a model that was "implemented as a sequence-to-sequence model with a bidirectional encoder over corrupted text and a left-to-right autoregressive decoder." While still relatively a new model in machine learning, the authors were able to show that it performed very well for text infilling tasks, which perfectly fit the criteria for our project.

Given the amount of research previously done on lyric generation, we were able to see the relevance of our project to the music industry in general. Rather than focus on genre specific lyric generation, however, we want to broaden the individual genre models to see if it's possible to train a model across different genres and artists.

3. Methods

3.1 About the Data

Our dataset, pulled from Kaggle, contains song lyrics for our text generation model. The data was extracted via the Genius API. In total, there are 40,000 songs in the dataset. There are only two initial columns in the dataset: song (a combined string containing the artist and song name) and the lyrics (string) for each song.

3.2 Cleaning the Dataset

We first utilized a language detector tool (langdetect) to determine the language of each song, and filtered out any songs that were not in English, and then set all the lyrics to be in lowercase. Because the dataset was generated via the Genius API, the verses in each song were clearly labeled, and in our case, we ensured that each song contained a [chorus] label or [hook] label in the text. We included the [hook] label based on our research into song structure, since we found choruses and hooks are synonymous when only one or the other is present. Consequently, we also removed songs that had both a [hook] label and a [chorus] label present because they did not fit that criterion. For the purposes of our research, anything that is not labeled as a [hook] or [chorus], for example [intro], [bridge], [refrain], etc. is considered a verse. In order to actually execute this data processing, we created functions to extract the indices for choruses and verses in each song. From these functions, we filtered out songs that did not contain a chorus, leaving us with 28,732 songs in our dataset.

3.2 *Finalizing the Dataset*

Once we cleaned the dataset, our next step was to decide how we wanted to feed song lyrics into our models. Because we wanted our model to generate choruses in order, we decided to pass in a song's sections in lyrical order. This process would also allow for evaluation to be done at the chorus prediction level, rather than the entire song level, since we want to specifically focus on the model's ability to predict the chorus.

At first, we used individual mask tokens for each word in a verse or chorus while creating our final dataset. We quickly realized this approach was incorrect as there was no way to limit each mask to only represent one word, so individual masks for each word were not an option. Additionally, the fact that we were going to be using BART as our final model solidified this because BART only accepts one mask token per input. While creating our final dataset we processed each song into smaller verse-chorus pairs and sent those to the model with a single mask token per input and target. As an example, say we have Song A with the following structure.

Song A: [verse_1] [chorus_1] [verse_2] [chorus_2] [verse_3] [chorus_3] [verse_4]

While training the model, we would pass in all verses before a chorus, with a mask token for the chorus in order to predict that chorus. Table 1. below shows the inference process for Song A:

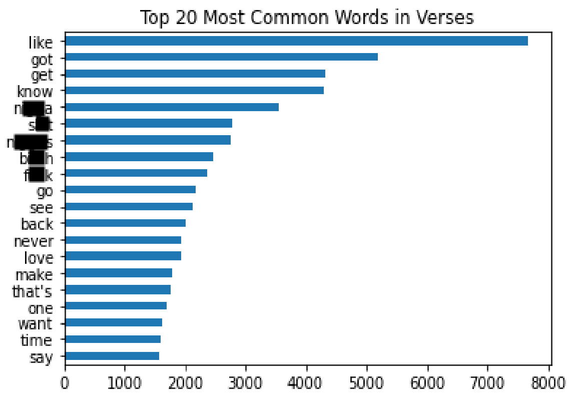
Table 1. Verse Chorus Chunking Method

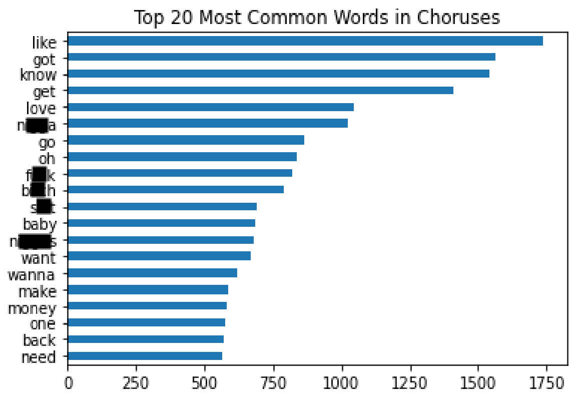
Model Input	Model Target	Model Output
[verse_1 + <chorus_1 mask>]	[<verse_1 mask> + chorus_1]	chorus_1
[verse_1 + verse_2 + <chorus_2 mask>]	[<verse_1 + verse_2 mask> + chorus_2]	chorus_2
[verse_1 + verse_2 + verse_3 + <chorus_3 mask>]	[<verse_1 + verse_2 +verse_3 Mask> + chorus_3]	chorus_3

In order to achieve these verse chorus pairs and finalize our dataset, we randomly sampled 4,000 songs from the initial dataset and created 13,176 verse-chorus pairs. Our final dataset had three columns: song (a combined string containing the artist and song name), inputs (verse-chorus pairs with the chorus masked out), and targets (verse-chorus pairs with the verses masked out).

3.3 Exploratory Data Analysis

Upon finalizing our dataset, our next step was to begin analyzing the data. Because we are doing text-to-text generation, we wanted to look at word distributions. In order to get a meaningful analysis, we filtered out stop words and found the following distributions.





From our EDA, we can see that there are similar many words repeated across verses and choruses. Furthermore, there are lots of rap songs in our dataset, leading to any explicit words that we have blacked out.

3.4 Baseline Model

Our baseline model was the T5-base Conditional Generation Model, utilizing a T5-base tokenizer (T5-base-sm). Our goal was to fine-tune the T5-base model to perform a fill-in-the-blank task, where the "blank" was the chorus of a song. As mentioned above in the *Finalizing the Dataset* section, we tokenized the input string containing the verse(s) and masked sentinel token of the chorus, and tokenized the target string containing the real chorus string and the masked non-sentinel token of the verses. We did not shuffle the inputs, but rather kept the inputs in sequential order. We set the max sequence length of the tokenizer to be 256, with padding for shorter inputs and truncation for longer inputs. We also included an AdamW Optimizer, with learning rate of 3e-4 and adam epsilon of 1e-8. The batch size was 8 and we ran the model for 2 epochs.

Initially, we just tracked the Cross-Entropy loss and accuracy metrics. We calculated the accuracy of each sample by comparing the indices of the target words and the indices of the generated words. However, we found the accuracy to be very low. This was due to the fact that target texts in T5 utilized the <PAD> token at the end of the string, whereas the model-outputted strings utilized empty string tokens ‘’ at the end. The model outputted empty string tokens at the end, since we had set the max token length to be 256. Therefore, during training, we manually replaced empty string tokens at the end of each model output to pad tokens, which allowed for a more standardized comparison of the words and caused accuracy to increase to a more reasonable range.

Masked language modeling problems usually compare the similarity of content between the target text and generated text, so we use Rouge and Bleu score metrics as measures of model performance. Below, we have two tables that show the metrics for our T5-base model.

Table 2. Model Performance (13,176 Verse Chorus pairs from 4,000 songs)

	Model	Rouge1	Rouge2	RougeL	Accuracy	Bleu	Loss
Validation	T5-base	0.5308	0.2941	0.4555	0.5416		2.4367
Test	T5-base	0.1502	0.0194	0.1014	0.0175	0.0065	

Transfer learning occurs during the validation step of the model training process. Because this is a sequence-to-sequence model, it is trained using teacher forcing, meaning that we pass in both the input tokens (verse tokens) and the expected labels (chorus tokens) which is why the generated text is more accurate on the validation set, because the decoder always gets the ground truth token regardless of what the previous predicted token was. For the test set, we call the generate function, which is autoregressive, meaning it takes in the previously predicted token to generate the next token. We notice that the T5-base-sm model overfit on the validation data, and was not able to generalize well on the test data. The model had trouble generating text by itself, without target text fed in.

3.5 Model Experiments

We decided to run a few experiments to see if we could improve the scores we got on our test set. From the choruses generated by our T5-base-sm model, we saw that it is unable to accurately fill in long variable length masks of text. Therefore, we ran two experiments to check if we needed to change our chunking method to shorten our input verse text and target chorus text. As such, we trained a T5-base model (T5-base-10-sm) to learn the first 10 words of a verse and predict the first 10 words of a chorus and trained another T5-base model (T5-base-20-sm) to learn the first 20 words of a verse and predict the first 20 words of a chorus. Through this approach, we hoped T5 would be able to better understand and predict shorter snippets of songs. However, as noted in Table 3. below, T5-base-20-sm performed worse than T5-base-sm and T5-base-10-sm performed even worse than those two models. These experiments revealed that T5 requires more verse text to learn better and we would need to keep our full verse chorus chunking method as is.

Table 3. Model Test Performance (13,176 Verse Chorus pairs from 4,000 songs)

Model	Rouge1	Rouge2	RougeL	Accuracy	Bleu
T5-base-sm (baseline)	0.1502	0.0194	0.1014	0.0175	0.0065
T5-base-10-sm	0.0968	0.0146	0.0737	0.0241	0.0057
T5-base-20-sm	0.1327	0.0186	0.0942	0.0183	0.0070
BART-base-sm	0.1625	0.0234	0.1086	0.0189	0.0090

Table 4. Model performance (33,025 Verse Chorus pairs from 10,000 songs)

Model	Rouge1	Rouge2	RougeL	Accuracy	Bleu
T5-base-lg	0.1437	0.0201	0.0993	0.0148	0.0050
BART-base-lg	0.1701	0.0260	0.1133	0.0091	0.0096

3.6 Final Model

Based on our findings with our T5-base-sm, T5-10-base-sm, and T5-20-base-sm models, we decided to utilize another language model, BART for Conditional Generation. We chose

BART architecture because the model was pre-trained on text infilling tasks, and had mask-filling capabilities. For the sake of consistency and to be able to accurately compare our results from the BART model with the results from our baseline T5 model, we chose to use the BART base model as well.

While training the BART-base-sm model on our dataset of 4,000 songs (13,176 verse chorus pairs), we noticed it was not performing as well as the initial T5-base had, so we trained the model instead on a larger dataset. We created a completely separate dataset of 10,000 individual songs (33,025 verse-chorus pairs) and reran a BART model, BART-base-lg. For a more standardized comparison, we retroactively ran our T5-base-lg model, where we trained our T5-base model on the larger dataset as well.

In order to train the BART models, we also had to go back and modify our dataset because the model only accepted <mask> as the tokenized input. Additionally, removed new line characters in the inputs and targets, since the BART model was not able to properly generate spaces between words in the output. When we actually ran the BART-base-lg model, we used the same parameters and setup as the T5-base model: a max sequence length of 256 for the tokenizer, with padding for shorter inputs and truncation for longer inputs, an AdamW Optimizer with a learning rate of 3e-4 and an adam episol of 1e-8, and a batch size of 8. We ran this model for 2 epochs as well.

We noticed that BART models in general had a few limitations. Firstly, the BART model wasn't padding the end of predicted text as the targets or the T5 model had. It also kept generating more words than were in the predicted text. BART also had trouble generating spaces in between words, which we fixed by removing new line characters in the text. We also tried implementing the BART-large model instead of the BART-base, but unfortunately, even with the removal of new line characters, this model did not work as expected as it was unable to predict spaces between the words properly.

4 Results

From Table 4. above, we can see that the BART-base-lg model trained on the larger data of 10,000 songs performed better than the T5-base-lg model trained on the larger dataset. BART-base-lg scored 0.0046 Bleu points higher, 0.0264 Rouge 1 points higher, 0.0059 Rouge 2 points higher, 0.014 Rouge L points higher than T5-base-lg model did. The BART-base-lg accuracy, however, is 0.0057 points lower than T5-base model's. Accuracy was calculated by comparing the model's generated word indices, and since the BART model did not generate enough pad tokens and kept generating more word tokens than the T5-base-lg model did, the lower accuracy for BART is reasonable.

It is surprising to see that the T5-base-lg model trained on the larger dataset performs slightly worse than the T5-base-sm model trained on the smaller dataset. This could be due to the fact that the T5 model, despite having more training data, does not learn song patterns that well. We could have improved the larger T5 model's performance by increasing the batch size, and increasing the number of training epochs.

The larger BART-base-lg model performs slightly better than the smaller BART-base-sm model on Rouge and Bleu scores, but not by a huge margin (at most 0.014). From these results, we can also see that the BART-base-lg model scored better than our original T5-base-sm baseline model. However, it is important to note that our BART-base-lg model performed better by a fairly small margin. These small margins for both the BART and T5 models demonstrate that just throwing more training data at the large language models won't give you that great a jump in performance.

5 Conclusion

Based on our results, we were able to conclude that the BART-base-lg, which was trained on the larger dataset of 33,025 verse chorus pairs, performed better than the T5-base-sm model, which was trained on the smaller dataset of 13,176 verse chorus pairs. We decided to train our models on a larger dataset because the BART-base-sm model was performing poorly compared to our T5-base-sm, and we wanted to rule out an issue with the dataset.

Future work would consist of exploring whether or not it is possible to train the T5 and BART models to recognize lyric patterns and rhymes. We would also like to try a different model architecture, where we add some linear and dropout layers on top of the word embeddings outputted from BART and process them to create more comprehensible and cleaner choruses. As stated previously, we are hoping to develop a tool that will allow a user to input an entire song of verses and indicate where the choruses should be, and our model will generate the choruses for that song.

6 References

1. Gill, H., Lee, D., & Marwell, N. (2020, September). *Deep Learning in Musical Lyric Generation: An LSTM-Based Approach*. https://yurj.yale.edu/sites/default/files/deep_learning_in_musical_lyric_generation_an_ltsm_based_approach.pdf.
2. Ram, N., Gummadi, T., Bhethanabotla, R., Weinberg, G., & Savery, R. J. (2021, November 15). *Say What? Collaborative Pop Lyric Generation Using Multitask Transfer Learning*. https://arxiv.org/pdf/2111.07592.pdf.
3. Manjavacas, E., Karsdorp, F., & Kestemont, M. (2019). *Generation of Hip-Hop Lyrics with Hierarchical Modeling and Conditional Templates*. https://www.inlg2019.com/assets/papers/46_Paper.pdf.
4. Lewis, M., Liu, Y., Goyal, N., Zettlemoyer, L., Stoyanov, V., Levy, O., Mohamed, A., & Ghazvininejad, M. (2019). *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. https://aclanthology.org/2020.acl-main.703.pdf.
5. *tf_fine-tuning*. https://colab.research.google.com/github/patil-suraj/exploring-T5/blob/master/t5_fine_tuning.ipynb#scrollTo=RKNr7fgzcKpZ. (2019).
6. Sinclair, N. (2020, October 18). *Teaching BART to Rap: Fine-tuning Hugging Face's BART Model*. tf_fine-tuning. https://colab.research.google.com/github/patil-suraj/exploring-T5/blob/master/t5_fine_tuning.ipynb#scrollTo=RKNr7fgzcKpZ. (2019).