CS230

Creative Lyrics with Transformer-Based Reinforcement Learning

Chinmaya Andukuri
Department of Computer Science
Stanford University
andukuri@stanford.edu

Niral Patel
Department of Computer Science
Stanford University
patnir41@stanford.edu

1 Introduction

Lyric generation is a subproblem with large opportunity for exploration within the generative AI space. Existing frameworks [1] [2], while impressive, on occasion will produce uninteresting lyrics which seem grammatically correct. Human artists might develop writers' block and search for direction or guidance as they produce poetry, song lyrics and other creative text. A model tailored to generating captivating, authentic and distinct content can serve as a springboard for emerging artists and individuals seeking creative inspiration.

We propose a transformer-based reinforcement learning approach to generate context-aware results and encourage creative output. Specifically, we create a reward mechanism which rewards more creative outputs as measured by metrics of creativity and coherence as described in section 2. We hope this approach will help us produce more semantically rich, unique lyrics. In this report we describe our method and results for our baseline model, which is a distilGPT2 model finetuned on a dataset of over 3 million song lyrics, and our novel model which incorporates RL techniques to encourage lower perplexity and higher creativity metrics.

2 Novelty

There are a few key differences in our approach from previous lyric generation models:

1. Previous approaches to this problem [2] condition on artist and build individual small models with differing weights for each artist. They do not make full use of shared idiosyncrasies between lyric language across artists in the same genre, or across genres, which we attempt to do by including genre and title information in the input vector and building one model.
2. Previous approaches also do not attempt to hand-construct reward or loss functions based on multiple language quality metrics - some may try to use a single metric BLEU score [6]. We explore the use of "balancing" two particular metrics that should respectively encourage creativity and coherence in our loss function.
3. Previous transformer-based approaches to the song lyric generation problem specifically rely almost exclusively on finetuning pretrained models, using standard cross-entropy loss and adding no custom layers. [2] [5] Our reward-based creative head attempts to step outside of the finetuning box and evaluate the potential for adding a small number of extra layers on top of a transformer for this purpose.
4. Previous models also do not take into consideration the notion of an "ideal" level of creativity and coherence for a given genre and title and train the model to approach this level. Our

CS230: Deep Learning, Spring 2023, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

approach attempts to incorporate this idea to not reach some arbitrarily high level of quality in these areas, but to approach the level of quality exhibited by *real* lyrics.

3 Dataset

The dataset consists of lyrics and associated data (title, genre, artist, release data, etc.) for over 3 million songs through 2022 from Genius, a widely-used and widely-verified source of song lyrics. First, we filtered for English songs (about 70% of dataset) to maintain a narrow scope for our work and avoid problems regarding the translation of creative non-English lyrics. We then chose a 99%, 0.5%, 0.5% train, test, dev split (3374200, 16871, 16871) given the large amount of data available.

Next, we performed some preprocessing steps (detailed in training/train_text_file.py) in order to reformat all input strings to concatenate genre, title, and full lyrics along with beginning and end tokens such that the model could 1) learn information about song length and 2) avoid beginning to generate lyrics for a new song midway through an existing prompt. We used regular expressions to remove unwanted song structure tokens like '[Chorus:]', '(Verse)' and 'x2'. While these tokens might be useful in an ML task to learn structure of songs, we focused on generating overall semantically interesting output and leave the structure task for another project.

For the baseline model, after preprocessing lyrics for each song and tokenizing each word to an ID 1-50257 - using the standard GPT-2 tokenizer - we dumped all concatenated tokens in order into a text file, which was then read into a custom HuggingFace Dataset class with block size 512 tokens. In baseline model training, each "block" serves as one training example. For the novel model, we used each song as one example on its own rather than a sub-block of a song.

4 Baseline Model Approach

For our baseline model, we finetuned HuggingFace's distilGPT2 model on size 512 blocks of song lyrics. This is a model intended to be a lighter, faster version of GPT-2 with 82 million parameters (as opposed to the true model's 1.5 billion). We used a standard cross-entropy loss function and AdamW optimizer, which is the default optimizer for HuggingFace's GPT-2 family. Since our baseline model used an existing model architecture many hyperparameters were selected for us.

Important hyperparameters that were selected ourselves were learning rate α = 1e-3 and batch_size = 4. Most crucially, the model trained for one epoch vastly outperformed the initial model trained for 5 epochs with learning rate $\alpha = 5$e-5 at first glance.

We did not include explicit metrics for this initial model, but note that any sample input x' which was identical to some dev-set example input x produced exactly the corresponding output y, while any x' which was not identical to any example input produced gibberish, clearly indicating a dangerously overfitted model. As such, we chose to use the training_args default learning rate and train for only one epoch, which resulted in outputs that were coherent and clearly resembled lyrics that were different from the original lyrics, indicating the generalizability of the model.

Here are the first eight lines of 3 sample output songs given prompts from the test set.

	Prompt	formatted_generated_lyrics
4	Genre: rock Title: Little of Your Love Lyrics:	Ive been waiting for you For a long time But now Im here to stay And Ive got to find The way to love you again Little of your love Is there a way
64	Genre: pop Title: Compromise Lyrics:	Im not a fan of the way Im feeling But I know that I cant stop And Ill be fine Youre the one I need To make it through Its not the same The same thing Ive been through
99	Genre: pop Title: Bar Close And The West Bank Bridge Lyrics:	Bar close and the West bank bridge The Westbank bridge is a beautiful bridge that you cant walk on You can walk the bridge youre not going to walk And youll never walk down the street But you will walk up the hill Ill walk in the west bank Theres a bridge in front of you

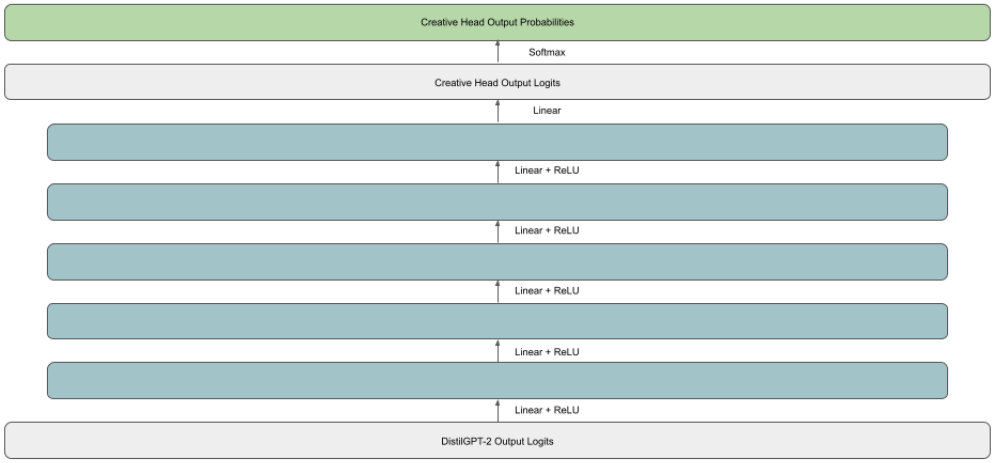
5 Background Knowledge and Metrics

Type-token ratio measures the degree of lexical variation in some text (by calculating the total number of unique tokens in a sequence by the total number of tokens). This means a higher type-token ratio closer to 1 indicates a more varied and diverse text while a lower type-token ratio indicates a less varied and diverse text. These properties allowing us to use it as a proxy for creativity in some sense. Perplexity can be used as a proxy for the coherence of some text in some sense - the formula is given below, but for simplicity it can be interpreted as the inverse of the predictability of the overall sequence $w_1, \ldots, w_N$. In other words, the more predictable or coherent a sequence is, the lower the perplexity. Note that perplexity is calculated over a group of all N-grams in a sequence (where a 2-gram is a pair of sequential words, a 3-gram is a group of 3 sequential words). A specific N must be chosen for the task at hand; a larger N makes a low perplexity value more difficult to achieve (it's harder to predict 10 words in a row correctly than 2, for example).

$$
\begin{aligned}
PP(W) &= 2^{-\frac{1}{N}\log_2 P(w_1, w_2, \ldots, w_N)} \\
&= \left(2^{\log_2 P(w_1, w_2, \ldots, w_N)}\right)^{-\frac{1}{N}} \\
&= P(w_1, w_2, \ldots, w_N)^{-\frac{1}{N}} \\
&= \sqrt[N]{\frac{1}{P(w_1, w_2, \ldots, w_N)}}
\end{aligned}
$$

6 Novel Model

The novel model consists of an RL-based head on top of the final baseline model in order to produce more creative output. This "creative head" takes an input containing the output logits from the baseline model and will output a probability distribution of the same shape after 5 hidden linear layers + ReLU blocks of size 768 and a final linear layer + softmax to produce and normalize the output layer of size $|\textbf{Vocabulary}|$. Note that the parameters from the finetuned distilGPT2 model were frozen - we only trained the parameters in the new layers to produce more creative output. The model tended to degenerate quickly with the default learning rate of $1\text{e}{-3}$, repeatedly generating the same token to minimize the loss. While this could have been a symptom of a suboptimal choice of loss function, we ultimately began to see better results on smaller learning rates and settled on a more conservative $5\text{e}{-8}$.



After autoregressively sampling new tokens and appending to a sequence S until 100 tokens are generated, we:

1. Calculate reward $R_S = typetokenratio_S - perplexity_S$
2. Calculate ground truth reward $R_T = typetokenratio_T - perplexity_T$

Perplexity was calculated based on predictability of bigrams (groups of 2). We then calculated the negative squared error

$$R_{example} = -(R_T - R_S)^2$$

The goal was to minimize the difference in the reward between the ground truth and the generated output sequence. (We scale by -1 due to the structure of our code, which seeks to *maximize* the reward; maximizing the negative of the squared difference corresponds to minimizing the squared difference). This assumes the ground truth lyrics display an "ideal" level of coherence and creativity, which we accounted for in our reward calculation. In order to train the model, we used a variation on the REINFORCE algorithm. This decision was motivated by several factors. First, note that the reward we describe above is not differentiable with respect to the model parameters; therefore these semantic metrics should not directly be used as losses with a gradient-based backpropagation technique. The REINFORCE algorithm [7] is as follows:

$$Q^{\pi_\theta}(s_t, a_t) = v_t$$
$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

function REINFORCE
  Initialise θ arbitrarily
  for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
    for $t = 1$ to $T - 1$ **do**
      $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
    end for
  end for
  return θ
end function

The algorithm works by sampling a trajectory of decisions from probability distributions until a "full" sequence is generated. It is commonly presented as a stochastic gradient method, meaning rewards are backpropagated for each single trajectory or "example". Then, the the products between each decision's standalone reward and its log probability are summed. In our implementation, since our reward is calculated on a sequence level, rather than the token level, we cannot truly sum the appropriate products for each decision. However, we instead sum the log probabilities for each decision and distribute the total sequence reward over this sum, as follows:

$$Loss_{example} = R_{example} \sum_{i=1}^{len(sequence)} \log p(token_i | token_1, ..., token_{i-1})$$

Finally, using the REINFORCE method involving sums of the products of log probabilities and rewards allows the use of gradient optimization with respect to the model weights, which only using the computed reward as a loss would not have allowed. As a result, we chose to use the standard Adam optimizer for training.

The reward was designed in order to encourage the model to produce text sequences that correspond to type-token ratio and perplexity as closely as possible to true lyrics, assuming these lyrics displayed an ideal level for these metrics. Originally, assuming we may have tried some other reward based model that try to maximize the reward, we chose to reward *higher* type-token ratio and *lower* perplexity for generated sequences (see metric explanations in section 5), explaining the subtraction order of the reward calculation. However, after deciding to minimize the squared difference between the ground truth and generated sequences, this order was no longer relevant and just a remnant.

Finally, this model took much longer for a single forward pass for each example than the baseline. In addition to this, given the stochastic gradient descent framework most compatible with our approach, we chose to train on a randomly selected 20000 song examples from the training set rather than the full dataset.

7 Model Evaluation

After prompting the baseline model and the novel model each with the same 100 randomly sampled prompts from the test set, each metric here was calculated as the average of 100 respective outputs each produced. The values for respective ground truth lyrics for those 100 prompts were also averaged for comparison.

	Type-Token Ratio	**Perplexity**
Truth	0.47294	1.86978
Baseline	0.56488	2.41210
Novel Model	0.99931	1.00099

For the baseline model, the relative differences here between the metrics are reasonable given this is our baseline model. We see a higher type-token ratio in the finetuned baseline model than in the ground truth lyrics - we later use this metric as a proxy for creativity, this could also indicate a tendency towards less likely or less sensible tokens than the truth. While this is not an optimal outcome, it is expected for now that the model produces seemly "random" text on occasion. We also see a higher perplexity in the baseline model, signaling more complexities and incoherence in this model than the ground truth lyrics. This is also reasonable for a baseline model.

By the end of training, the novel model had on average produced type-token ratio and perplexity of about 1 each. Recall the reward calcualation for R_S on page 4; this indicates that the reward for novel generated sequences was on average about 0. Interestingly, this does not correspond with the objective; we hope to make the reward for generated sequences close to the reward for true lyrics, *not* to make the components of the reward in generated sequence equal to each other. The model, however, still achieved the lower perplexity (coherence) we intended and higher type-token ratio (diversity in sequence) discussed in section 6.

There are some possible explanations for this result:

1. type-token ratio, which is the positive element in the reward calculation, is capped at a maximum value of 1. The model may have learned to increase the TTR of generated sequence on its way to numerically balancing R_S and R_T, and ultimately was forced to decrease perplexity toward 1 in order to *attempt* to achieve the appropriate balance late in training.
2. The true average reward among the training set's ground truth lyrics may have been 0; the model was ultimately successful in achieving the appropriate balance, but the distribution of language style between the test set and the training set was quite different.

Regardless, it should be mentioned that the text produced by the novel model was unappealing and could not reasonably be used as inspiration for lyrics. In particular, although the perplexity decreased, the human-eye coherence of the text was quite low. This may indicate that a larger N-gram size should have been chosen in our perplexity calculation in order to encourage more sensible output; subsequent pairs of words were easily predicted but larger chunks were unpredictable.

Our biggest takeaway, confirmed by the state of NLP today, is that the finetuning with cross-entropy loss, should continue to be a trusted paradigm for text generation tasks like ours as it is vetted and quite likely to work well, especially given enough compute and a large dataset. However, reinforcement learning and reward-based approaches show promise by offering a kind of control over style and preference that the original paradigm may not. Based on results, some changes I would make in a future iteration are the following:

1. Spend time to construct a reward very carefully. Two metrics are likely not enough to capture the variety of language phenomena we see in creative text. Even if two metrics are sufficient, the operations used in the construction should be well-tested and well-informed by approaches in other subfields of AI. While this approach was a good first attempt, this is a difficult problem that will take a lot of care to solve.
2. Use a true reinforcement, policy-learning approach. It is well-known that RL-natural language problems can suffer from the "sparse reward" problem (meaning rewards are given only at the end of a sequence rather than on a single -decision or -token basis, making good behavior harder to learn). Explore methods that allow dense rewards for text generation, or more carefully constructed approaches [8] to address the sparsity, since distributing the sequence reward over all the individual token probabilities was not effective in our case.

8 Submission

Here is the GitHub Repository for the project. The important training loops are in training/dev_baseline.py and training/train_novel.py. The preprocessing of the data and the train/test/dev split happen in train_text_file.py and exploration/data_split.ipynb. More exploration is in other notebooks in the exploration/ directory.

The model weights and additional metadata after finetuning are altogether too large to fit into a normal GitHub repository. We store them instead in this shared Google Drive folder. The final traiing weights for both baseline and novel model are in the directory. Finally, here is the link to the dataset - the full data is too large to include in our repository or Google Drive.

9 Contributions

All members have contributed equally to the project's progress.

Chinmaya: Experimented with model architectures/hyperparameters to find an appropriate baseline model and explored possible techniques for the novel model head. Adjusted and debugged the training process to train on the training set, producing the final baseline + novel model.

Niral: Experimented with model architectures/hyperparameters to find an appropriate baseline model and explored possible techniques for the novel model head. Ultimately produced the selected baseline model and trained/evaluated it on the dev set.

10 References

[1] Dhariwal, P., Jun, H., Payne, C., Kim, J. W., Radford, A., Sutskever, I. (2020). Jukebox: A Generative Model for Music. arXiv preprint arXiv:2005.00341.

[2] Korshuk, A. (2021). HuggingArtists [Software]. GitHub. https://github.com/AlekseyKorshuk/huggingartists

[3] Jiang, Yifan, Shiyu Chang, and Zhangyang Wang. "TransGAN: Two Pure Transformers Can Make One Strong GAN, and That Can Scale Up." arXiv, 2021, https://arxiv.org/abs/2102.07074.

[4] Saeed, Asir, Suzana Ilić, and Eva Zangerle. "Creative GANs for generating poems, lyrics, and metaphors." arXiv, 2019, https://arxiv.org/abs/1909.09534.

[5] sfs0126. "SFS0126/Lyric-Generator-Fine-Tuned-GPT-2: This Project Uses Huggingface Transformers GPT-2 to Fine-Tune Text Generation Models Based on Lyric Data to Specific Music Genres." GitHub, 7 Dec. 2021, github.com/sfs0126/Lyric-Generator-fine-tuned-GPT-2.

[6] Fedus, William, et al. "Maskgan: Better Text Generation via Filling in the ___." arXiv Vanity, 2018, www.arxiv-vanity.com/papers/1801.07736/.

[7] "Papers with Code - Reinforce Explained." Explained | Papers With Code, paperswithcode.com/method/reinforce. Accessed 7 June 2023.

[8] Ramamurthy, Rajkumar, et al. "Is Reinforcement Learning (Not) for Natural Language Processing: Benchmarks, Baselines, and Building Blocks for Natural Language Policy Optimization." arXiv.Org, 1 Mar. 2023, arxiv.org/abs/2210.01241.

11 Appendix: Our Favorite Lyrics

The following lyrics were generated from our baseline finetuned model. While this project was difficult at times, these outputs beautifully showcase the power of AI to provide creative inspiration for us as humans.

Undo the unknown
The secret smiles the night over
Over and over again
For me it all comes down down down down down down down
In a world without the sun now
Back around down
Now when the dark comes out brings out the light
We turn the world around All the moods in motion
Out of the darkness into the light
We turn the world around now
And dream about the future
Tonight's the night we're fine
Goin' out goin' out to the world now
So let's go
Blindfolded in darkness I'm making my heart
Not to see right in the beginning
Got the morning
So I feel the night right
Still I've come
Feel as I'm dreaming
Still make this feel it to the feeling till it in
Well be strong yet with the night
Dream again
Blindfolded be felt life

I know tomorrow you will be there
As always with me
All alone in your cold sky
Watching all your angels cry
But then I'll wonder you where do we go
We don't have to find ourselves alone again
I know tomorrow you'll be here
Holding someone as close as I can
Holding someone as close as I can
Holding someone as close as I can
Foolish memories and fading smoke
Grace follows me on her to the howling wind
But our love is gone now
And now it falls
How long will it be
Before me and where we go
We kiss until the morning broken low
When good will it seems
Wrap is gone and though
We love are pale from now
When we laugh and bitter love
Has broken heart
I can tear becomes how long
And crying light
Back our love
And our arms abandoned
Our moments take it drags
Never sober be a hurried love