Lyric Generation with Bidirectional Long Short-Term Memory

Dylan Bowman
Department of Mathematics
University of Illinois
Urbana, Illinois
dcbowma2@illinois.edu

Junseok Yang
Department of Statistics
University of Illinois
Urbana, Illinois
jyang247@illinois.edu

***Abstract*—This project explored a widely used neural network architecture for text generation, which is Long short-term memory (LSTM), using song lyrics data to generate new lyrics. The data was subset based on genres and further cleaned to be suitable for our LSTM model. Songs with country genre were trained into the model and successfully generated new lyrics.**

***Index Terms*—RNN, LSTM, text generation**

I. Introduction

For our project, we used a Long Short-Term Memory (LSTM) architecture to generate song lyrics given user input for genre and a base phrase to start off with. It is a specific instance of next-token prediction, which is a well-documented task, and the project is interesting because it involves using a widely used and effective deep learning architecture to tackle a fun problem. We are interested in how the lyrics generated from the model be similar or dissimilar and analyze any noticeable features based on different genres.

We will use a dataset that contains approximately 5 million songs' lyrics and other features such as genre from Kaggle[5]. For the task of lyric prediction, an analogous project has been done using LSTMs in the domain of rap lyric generation[6], but we plan to span across a broader array of genres and use a larger corpus.

II. Related Work

Constructing a background about natural language process (NLP) and text data is required, and the book written by Bird et al introduces about these materials well. Some data preprocessing steps will be referred to the book[2]. Connected to this, it is also necessary to have a solid understanding about deep learning and text generation, and diverse information from basic guidelines to experiments about recurrent neural network and long short-term memory (LSTM) can be referred from the work by Graves[4]. For constructing a LSTM model, we chose to use a simple Long short-term memory structure introduced by Bitvinskas, and further modify and develop into our own model with PyTorch[3].

Similar project was done and published by Potash et al, which is about ghostwriting and creating a LSTM model to generate rap lyrics. By referring to the previous work, we can get some useful information and limitation of their project. Based on this, our project can not only be able to set and navigate to the right direction, but also build more sophisticated LSTM model[6]. Lastly, Bergstra and Bengio introduce and compare few different search methods for hyperparameter tuning process for neural networks, and it is crucial for our project to find the best optimization values to generate more accurate and less grammatic error lyrics[1].

III. Data

The data was sourced from a Kaggle repository containing a dump of all the lyrics on Genius. It contains approximately 5 million songs with 8 variables such as 'artist', 'tag', and 'lyrics' in .csv format. A sample can be seen below:

Title	Tag	Artist	Year	Lyrics
Killa Cam	Rap	Cam'ron	2004	[Chorus: Opera Steve ...
Can I Live	Rap	Jay-Z	1996	[Produced by Irv Got...
Forgive M...	Rap	Fabolous	2003	Maybe cause I'm eatin...
Down an...	Rap	Cam'ron	2004	[Produced by Kanye W...
Fly In	Rap	Lil Wayne	2005	[Intro] So they ask m...

A. Pre-Processing

To prepare a suitable dataset for our model, few pre-processing steps are necessary. Due to high and complex computational costs, it was inevitable to subset the dataset into a smaller size. We first created several csv files based on 6 different genres ('tag') and tested cleaning process with one of the subset datasets.

B. Cleaning

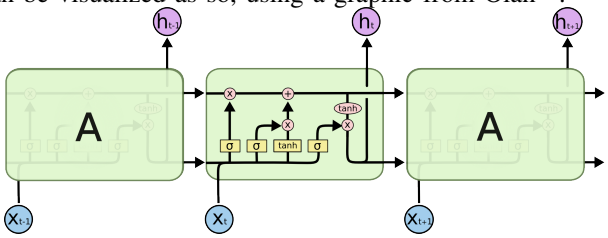
Using one of the subset datasets, regular expressions ('regex') were used to tidy the 'lyrics' variable data (Bird et al.). Some unnecessary patterns were detected in the lyrics such as 'verse 1' or 'chorus' which indicate different parts in a song. These patterns are one of the obstacles that would hinder the LSTM model to generate interpretable lyrics. After the detaching step, two more stages of removing punctuation and separating sentences by a new line were processed.

IV. Preliminary Technical Details

A recurrent neural network (RNN) is a neural network that takes its previous outputs as parameters in order to process sequential data. Long short-term memory (LSTM) is a type of recurrent neural network (RNN) that is able to

learn and remember long-term dependencies in data. Unlike traditional RNNs, which can have difficulty learning long-term dependencies, LSTMs have a special structure that allows them to remember information for longer periods of time. This makes them well-suited for tasks such as language modeling, machine translation, and speech recognition, where the input data may have long-term dependencies. LSTMs are a type of artificial neural network that is widely used in deep learning applications.

Specifically, LSTM uses a "forget" gate to select which data to hold onto over long periods of time. The idea is that the module takes in three inputs, the previous output h_{t-1}, the previous state vector C_{t-1}, and the new sequential value X_t. The LSTM module then outputs the next value in the output sequence h_t and the state vector C_t, as well as feeding h_t into the output of the whole model. The internals of the modules can be visualized as so, using a graphic from Olah[7]:



Most importantly, X_t is concatenated with X_{t-1} and then combined with the long-term state vector C_{t-1}. This allows the architecture to incorporate both long and short-term information into its output h_t.

For our preliminary work, we used the default PyTorch settings for the hyperparameters, except for dropout of 0.2 and 3 layers. Our model was a unidirectional LSTM with one layer, using bias weights. For more information, see the PyTorch documentation[8]. We used cross-entropy loss as our loss function since the task is fundamentally one of next-word classification. The way that it works is that given the long-term state and short-term state, the model outputs a softmax distribution over all words in the corpus, with each word assigned the probability that it is the next word. For the generative task, we just sample from this softmax distribution. A past example of cross-entropy loss for LSTMs can be found in Mardanirad et al.[9].

V. Training Details

To train our LSTM, we accessed the HAL cluster from the NCSA at the University of Illinois. We ran our training script directly from the browser interface. Our train time improved significantly on the HAL GPUs, going from a number of hours to a number of minutes.

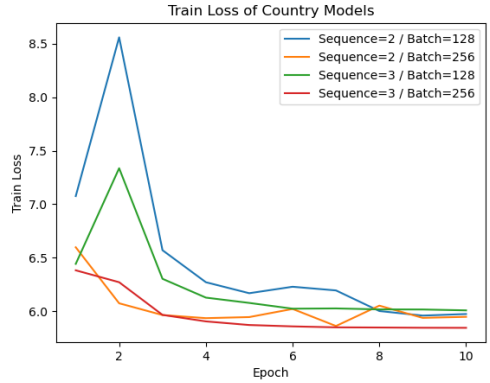
After verifying that the trained models functioned properly, we trained our LSTM model on 4 different genres (rap, pop, R&B, country) using different values for the sequence length and batch size hyperparameters. Sequence length is the number of time steps that the network is trained on. This is an important parameter to set when training an LSTM, as it determines how much past information the network can use to make predictions. If the sequence length is too short, the LSTM may not have enough information to accurately model the dependencies in the data. If the sequence length is too long, the LSTM may have difficulty learning and generalizing to new data. Batch size is the size of batches used in the Adam optimizer we employed. We used values of 2 and 3 for our sequence length, and we used batch sizes of 128 and 256. We trained 16 models in all.

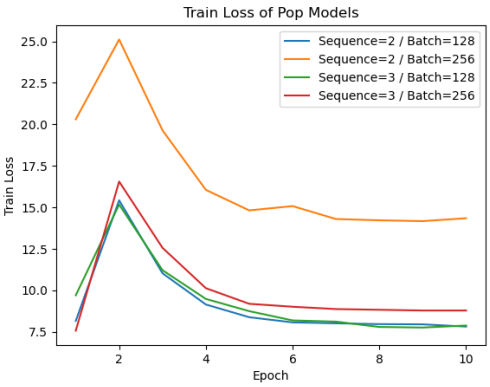
VI. Results

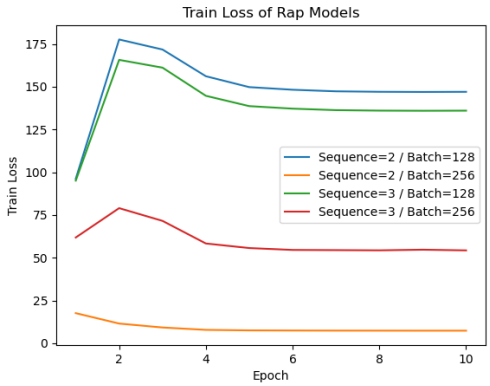
Our results can be divided into two parts: the training/testing empirical results, and the lyrics that we were able to generate.

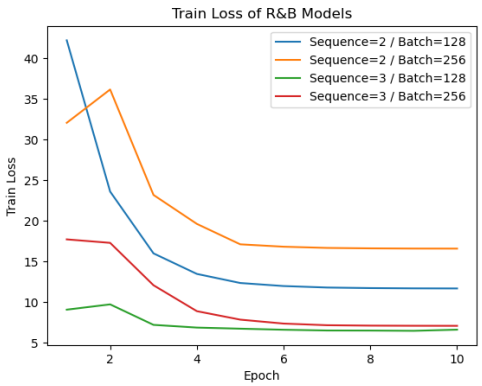
A. Training and Testing Results

The loss for each of our models can be viewed here:









In addition to tracking the training loss, we also measured the next-word prediction of the models for R&B and country (unfortunately, the pop and rap datasets were too large and unwieldy to perform this analysis on our local devices). The results from this test can be viewed:

R&B Accuracy	Batch Size = 128	Batch Size = 256
Seq. Length = 2	1.03%	11.93%
Seq. Length = 3	1.81%	5.96%

Ctry. Accuracy	Batch Size = 128	Batch Size = 256
Seq. Length = 2	0.55%	0.20%
Seq. Length = 3	0.22%	6.56%

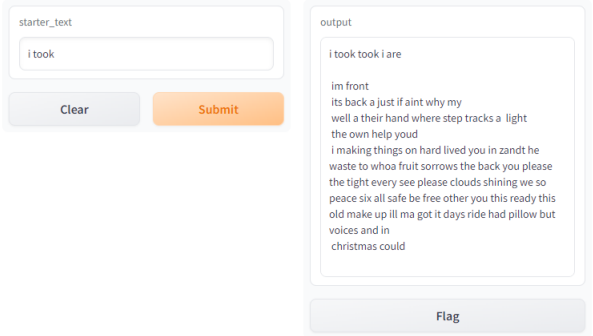
Given that there were thousands of words in the corpus, we are pleased with the accuracy of these models.

B. Generated Work

With the model that we were able to train, we were able to generate songs, such as the following:

> aware couldnt find the lost that fear
> in night daylight though i long under you
> wishing the halfhearted man im coming at love with me
> when her head are remain in the wind man
> its a addict town in your hand i can never leave being
> cause the beautiful trip i was born at me
> and you got starin is going to got a glimpse
> and a janitor sound to alabama
> what mamas twenty black true
> as our rhyme one jesus stone to blue scene
> its parton country busier train in

For users to input their own sample text and have it completed by our songwriter, we also built a gradio interface:



VII. Other Possible Approaches

While LSTMs proved adequate for this task, they are quite old and other approaches may have been more effective. We will review alternate approaches here.

The main alternative we could have used would have been a transformer model like OpenAI's GPT, which has been making waves in pop culture. One advantage of transformer architectures over LSTMs for text generation is that they are able to capture long-range dependencies in the input data more effectively. This is because transformers use self-attention mechanisms, which allow the model to weigh the importance of different input tokens at each time step. This makes it possible for the model to attend to relevant information from earlier in the input sequence, even if that information is far away in terms of the number of time steps. LSTMs, on the other hand, use fixed-sized memory cells, which can make it difficult for them to remember information from long ago in the input sequence.

Another advantage of transformers is that they are generally more efficient to train and use than LSTMs. This is because transformers are parallelizable, meaning that different parts of the model can be trained and evaluated simultaneously on different CPUs or GPUs. This makes it possible to train very large transformer models, which can achieve state-of-the-art performance on many tasks. In contrast, LSTMs are typically trained and evaluated sequentially, which can make it more difficult to scale them to large datasets.

Another architecture we could have used is the diffusion model. A diffusion model introduces noise to the data so that the data distribution can be learned without overfitting. Diffusion models are well-suited for generative tasks like text generation because they are able to learn the relationships between different words since they represent each word as a node in the model.

Had we more time, we would have trained our models on the entire dataset rather than the small slice that we took. Models like ChatGPT owe their strength to the size of their training corpus, and it would have been interesting to see how far our model could have gone with a larger training set.

VIII. Appendix

A. Code

Our code repository can be accessed at https://github.com/dylanbowman314/lstm-lyrics-generation.

IX. Contributions

Junseok - 50 - Wrote up most of report, found and read past literature, planned architecture and hyperparameters, logged into HAL and trained models, created graphics.

Dylan - 50 - Set up GitHub repository, cleaned data, ran model with stronger computer, ran tests, wrote majority of final report.

References

[1] Bergstra, James, and Yoshua Bengio. "Random Search for Hyper-Parameter Optimization." Journal of Machine Learning Research, 12 Feb. 2012, https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf.

[2] Bird, Steven, et al. Natural Language Processing with Python. O'Reilly Media, 2009.

[3] Bitvinskas, Domas. Pytorch LSTM: Text Generation Tutorial, 15 June 2020, https://closeheat.com/blog/pytorch-lstm-text-generation-tutorial.

[4] Graves, Alex. "Generating Sequences with Recurrent Neural Networks." ArXiv.org, 5 June 2014, https://arxiv.org/abs/1308.0850.

[5] Nayak, Nikhil. "5 Million Song Lyrics Dataset." Kaggle, 22 Apr. 2022, https://www.kaggle.com/datasets/nikhilnayak123/5-million-song-lyrics-dataset

[6] Potash, Peter, et al. "Ghostwriter: Using an LSTM for Automatic Rap Lyric Generation." Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, 2015, https://doi.org/10.18653/v1/d15-1221.

[7] Olah, Chris. "Understanding LSTM Networks." Understanding LSTM Networks - Colah's Blog, https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[8] "LSTM PyTorch 1.13 Documentation". pytorch.org/docs/stable/generated/torch.nn.LSTM.html.

[9] Mardanirad, S., Wood, D.A. Zakeri, H. The application of deep learning algorithms to classify subsurface drilling lost circulation severity in large oil field datasets. SN Appl. Sci. 3, 785 (2021). https://doi.org/10.1007/s42452-021-04769-0