Lyrics Generation using BERT

David Buckman **Apurba Sengupta** **Abhishek Tiwary** **Yash Vardhan**
University of Pennsylvania
{dbuckman,aseng,amannitr,yashv}@seas.upenn.edu

Abstract - *Text generation is an interesting research problem that has been of interest to the Computational Linguistics community for a long time. With huge Transformer networks replacing RNN- and LSTM-based architectures for various language modeling tasks, the push towards using these Transformer-based architectures for achieving state-of-the-art results is fascinating. In this project, we try to explore one such model developed by Google for the task of lyrics generation. We first use the pre-trained model to predict masked out words for a given lyric. The rhyming procedure is based on effective searching of all possible rhyming words. The pre-trained model is then used to predict the most probable lyric sequence.*

1 Introduction

A recent focus of Natural Language Processing has been on using semi-supervised and unsupervised approaches to train large Transformer-based networks. These networks can then be used to generate coherent text sequences given some context. An example of text generation by such a language model (LM) developed by OpenAI[1] is shown below:

Context - "*In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English."*

Generated Text - "*The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.*

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow."

We can see that the above model is able to generate meaningful and coherent text sequences. We would like to explore such Transformer-based models for the task of song lyrics generation. Songs, as we know, tend to tell a story. We can hence condition these models on some context lyrics and generate the next sequence of lyrics. Specifically, we try to explore the English Rock genre.

In this project, we work with the BERT language model[5] developed by Google. BERT is pre-trained to predict missing or masked out words in a sequence. For example, given a sentence like "*My dog is* `[MASK]`", where `[MASK]` represents the masked-out word, BERT can predict it to be "*cute*", "*hairy*", etc. BERT can also predict if a given pair of sentences occur in that particular sequence or not. For example, a sentence pair like ("*The man went to the store*", "*He bought a gallon of milk*") are predicted to be contiguous sentences, while a sentence pair like ("*The man went to the store*", "*Penguins are flightless birds*") are predicted to be non-contiguous sentences. We utilize the above two properties of the BERT model for song lyric generation.

2 Related Work

The problem of text generation using Transformer networks has gained attention over the past year since the introduction of Transformers in 2017 by Vaswani et al[2]. Radford et al[3] from OpenAI explored the use of Transformer networks for text generation and language understanding at large by incorporating generative pre-training (GPT) approaches. They used a left-to-right (unidirectional) LM using a 12-layer decoder-only Transformer with masked self-attention heads for this purpose. The GPT model was pre-trained on the BooksCorpus dataset[4] by unsupervised means, i.e., predicting the likelihood of a word given a *k*-sized context window.

Devlin et al[5] from Google AI argued that text generation (and language understanding in general) is a bidirectional problem instead of a unidirectional procedure as in GPT. They used a multi-layered bidirectional Transformer encoder called BERT as the LM. They developed 2 models - $\text{BERT}_{\text{Base}}$ with 12 Transformer blocks (just as in GPT) and $\text{BERT}_{\text{Large}}$ with 24 blocks. BERT is pre-trained using unsupervised training procedures based on masked language modeling (MLM)[6] techniques, wherein the model is trained to predict masked-out words in a sentence, and whether a pair of sentence sequences are truly a sequence or not. The BERT model was pre-trained on the BooksCorpus dataset and English Wikipedia.

Radford et al[7] argued that LM is a multitask learning procedure and glorified the GPT model by increasing the number of Transformer blocks in steps of 12 from 12 to 48. All these LMs are trained using the same unsupervised procedure as in GPT. The largest model, called GPT-2, achieves state-of-the-art results on text generation and other language understanding tasks. A more recent study by Wang et al[8] proposes text generation methods for BERT. They argue that BERT is a Markov Random Field (MRF) LM and that it learns a distribution over sentences of given lengths. The sampling process is then based on Gibbs sampling. They use a sequential sampling technique and a masked sampling technique for determining the masked-out words in a sentence.

Non-Transformer based methods for text generation have also been explored earlier. Greene et al[9] employed statistical procedures for English poetry generation. They utilized Finite State Transducer (FST)[10] and weighted FST[11] based methods for mapping word sequences to

stressed and unstressed syllable sequences. They then chose the most probable syllable alignments using Expectation Maximization (EM)[12] algorithm. Ghazvininejad et al[13] devised a model for poetry generation called Hafez, wherein they first chose rhyme pairs based on input words provided by the user, and then developed a Finite State Acceptor (FSA)[14] for every possible sequence with the rhyme words in place. Finally, they used a Recurrent Neural Network (RNN)[15] model to select the best sequence among the sequences generated by the FSA. Ghazvininejad et al[16] further tried to improve the accuracy of Hafez by building a web interface to get feedback from humans on the quality of the generated and the model then used these feedbacks to adjust its weights accordingly to generate better quality poems.

3 Experiments

3.1 Data and Evaluation Metrics

We use the *380,000+ lyrics from MetroLyrics* dataset available at Kaggle[18] for this project. The dataset consists of songs from various genres and artists in different languages. But our primary focus is on English Rock genre. We therefore filter out the rest of the songs and only keep English Rock songs for our project. We achieve this by converting the entire data into a `pandas DataFrame` object and selecting the Rock songs out of it. We then use `langdetect` Python library[20] to filter out Rock songs in languages other than English. Finally, we remove redundant songs by removing duplicate lyrics.

The filtered data consists of ~92k English Rock songs between 1968 and 2016 (both inclusive), from over 3400 artists, including Bob Dylan, Elton John, David Bowie, America, Elvis Presley, Beatles, Coldplay and many others. The data is available in comma-separated value (CSV) format in the file *english_rock.csv*. A typical row from the CSV file looks as shown below. It consists of the index of the song, the song title, the year the song was produced, the artist name, the genre (which is Rock for all data points), the song lyrics and the language of the song (which is all '*en*', representing English songs):

50732,desolation-row,1990,bob-dylan,Rock,"They're selling postcards of the hanging, they're painting the passports brown\nThe beauty parlor is filled with sailors, the circus is in town\n...",en

For the purpose of this project, we split the data into training, validation and test sets in the ratio of 8:1:1. We first randomly shuffle the data and then perform the above split. The training, validation and test data are then available in CSV format in the files *train_rock.csv*, *val_rock.csv* and *test_rock.csv* respectively. The `make_data.py` script performs the above split and creates the training, validation and test data files.

We use two evaluation metrics for this project, Accuracy and BLEU score[19]. These metrics are explained below:

Accuracy: We check the prediction of next lyric given the first lyric. If the predicted lyric matches exactly to the correct lyric, accuracy is 1 else it is 0. We compute the average accuracy over all the lyrics pairs in the ground truth and prediction files. Higher accuracy score is better.

BLEU score: We also use the BLEU score as a measure of how closely the predicted lyric matches the correct lyric. The BLEU score is obtained by comparing the *n*-gram overlap of candidate text with reference text(s). This metric is commonly is used to assess the quality of machine translation but we believe that it suits well for our task as well. We use NLTK's implementation for calculating BLEU score. We assign equal weights for unigram, bigram and trigram features and assign zero weight for 4-gram ones. Similar to accuracy, higher BLEU score is better.

The score calculation for given ground truth and predicted lyrics is implemented in the Python script `scoring.py`. It takes as inputs the ground truth lyric pairs (say, in *goldfile*) and the first and next predicted lyric (say, in *predfile*). We can then use the script as below:

```
python scoring.py --goldfile goldfile --predfile predfile
```

3.2 Simple Baseline

We first implement a simple baseline. In this, we first randomly select 100 pairs of lyric sequences. For each of the first lyric in the sequence, we also select 9 other random lyrics. The ground truth therefore contains 100 pairs of true lyric sequences. The prediction involves selecting the second lyric randomly from the 10 available lyrics for each of the 100 first lyrics.

The ground truth sequences are stored in a file named *goldfile*, while the prediction sequences are stored in a file named *predfile*. The baseline is implemented in the Python script `simple-baseline.py`. It takes as an input the training/validation/test file in CSV format, depending on which dataset we are trying to evaluate. We can use the script as below:

```
python simple-baseline.py --datafile data/<operation_type>_rock.csv
```

For the test data in *test_rock.csv*, we get the *goldfile* and *predfile* files. A few sample predictions are shown below. The first column is the first lyric, the second column is the true next lyric and the last column is the predicted next lyric:

First Lyric	**Second Lyric (Truth)**	**Second Lyric (Predicted)**
Been screamin' out your name,	I wake up in a sweat,	In fact I think I wish you would

Make my dad pay the bill	Yeah man, that's heaven to me	I deserve this!
Now I'm the invisible man	My head is spinning round faster and faster	'Cause I got voices down inside me,

Running the evaluation script `scoring.py` with these ground truth and prediction files as inputs, we get an Accuracy of 0.11 and a BLEU score of 0.11.

3.3 BERT Baselines

We experiment with the BERT model for our task of song lyrics generation. We use pre-trained PyTorch BERT models[17] for experimenting with single and multiple masked-word prediction. We use the data we obtained after extracting English Rock songs from the Kaggle Lyrics dataset. We first choose a pair of contiguous lyrics from our data and check the prediction for a single masked-out word from the second lyric using the `BertForMaskedLM` LM. For example, for the two contiguous lyrics from *Desolation Row* by Bob Dylan[7],

And the riot squad they're restless, they need somewhere to go
*As **Lady** and I look out tonight, from Desolation Row.*

if we mask-out the word "*Lady"* with the `[MASK]` tag and then use the `BertForMaskedLM` LM to predict the masked-out word, the model predicts "*Dad"* as the masked-out word, which is different from the original word "*Lady"* but makes perfect sense as well. The baseline is implemented in the Jupyter notebook `bert-baselines.ipynb`.

We also check the performance of the BERT model for next sentence prediction using the `BertForNextSentencePrediction` LM. Similar to the simple baseline prediction presented earlier, we first randomly select 100 pairs of lyric sequences. For each of the first lyric in the sequence, we also select 2 other random lyrics. The ground truth therefore contains 100 pairs of true lyric sequences. The prediction then involves using the `BertForNextSentencePrediction` LM to predict which among the 3 available lyrics for each of the 100 first lyrics is most likely to occur next. Again, the ground truth sequences are stored in a file named *goldfile*, while the prediction sequences are stored in a file named *predfile*. This baseline is also implemented in the Jupyter notebook `bert-baselines.ipynb`. It takes as an input the training/validation/test file in CSV format and generates the ground truth and predicted lyrics files.

A few sample predictions from the generated *goldfile* and *predfile* files are shown below for the test data in *test_rock.csv*. As before, the first column is the first lyric, the second column is the true next lyric and the last column is the predicted next lyric:

First Lyric	**Second Lyric (Truth)**	**Second Lyric (Predicted)**
Loneliness found me, looks like it's here to stay	I know that I oughta find someone new	I can give you songs to straighten your face
Registered in Pennsylvania	Do you even know what a wawa is, girl?	Since,
A white girl in a dashiki says you're all the rage	My friends and I think you're quite the sage	My friends and I think you're quite the sage

Running an evaluation script with these ground truth and prediction files as inputs, we get an Accuracy of 0.47 and a BLEU score of 0.50.

Lastly, we choose a pair of contiguous lyrics from our data and mask out all the words in the second lyric except the last one, which we keep as it is in order to maintain the rhyming of the two lyrics and to give the second lyric some context from its right hand side. We use the `BertForMaskedLM` LM for predicting all the masked-out words for the second lyric. The predictions are chosen according to maximum probability, with the additional possibility of choosing top-*k* predictions. As an example, for the two contiguous lyrics taken from *Dream On* by Aerosmith[7],

Sing with me, sing for the years
Sing for the laughter, sing for the *tears.*

if we mask out all words in the second lyric with the `[MASK]` tag except the last word "*tears*", and then use the `BertForMaskedLM` LM to predict all the masked-out words, we end up with:

Sing with me, sing for the years
Sing with me now, sing for the *tears.*

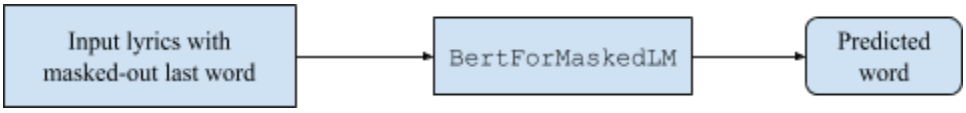
We see that this prediction is different from the original one, but it makes perfect sense too. This baseline is implemented in the Jupyter notebook `bert-baselines.ipynb` as well.

3.4 BERT for Lyrics Generation

We explore three strategies to get the desired lyrics using the above described pre-trained BERT models. First, we try to predict the last word of the target lyric using the `BertForMaskedLM` LM keeping its left and right context lyrics as it is and check the Accuracy and BLEU scores. Second, we search for all possible last words for the target lyric based on a proposed rhyming technique and predict the remaining words for the target lyric using the `BertForMaskedLM` LM. We then check which of the predicted lyrics are a continuation of the seed lyrics using the `BertForNextSentencePrediction` LM and keep just that lyric and discard the rest. We focus on the test data available in *test_rock.csv*. For this test data, we choose continuous chunks of 5 lines for each song. For each chunk, we try to predict the 4th line of lyrics. Hence, the first 3 lines and the 5th line become the left and right context lyrics for the BERT model. Third,

3.4.1 Last word prediction

In the target lyric, we replace just the last word by the `[MASK]` tag. We tokenize the 5 lines of lyrics using `BertTokenizer` and feed the tokens as an input to the `BertForMaskedLM` LM, which then gives out predictions for the masked-out last word in the target lyric. A schematic of the procedure is shown below:



We obtain the predictions for all the chunks of the test data and compare the results with the ground truth lyrics. The implementation can be found in the Jupyter notebook `last-word-bert.ipynb`. The ground truths and the predicted lyrics are available in the files *test_gold_file.txt* and *test_pred_file.txt* in the `nextword` subdirectory of the `output` directory. Some examples are shown below:

Lyric	Predicted word	True word
I've gotta empty out the inside of my head *I'd like to turn this place into my* <To be predicted>	*own*	*home*
Down here where everything's crazy *The whole world's falling*	*apart*	*apart*

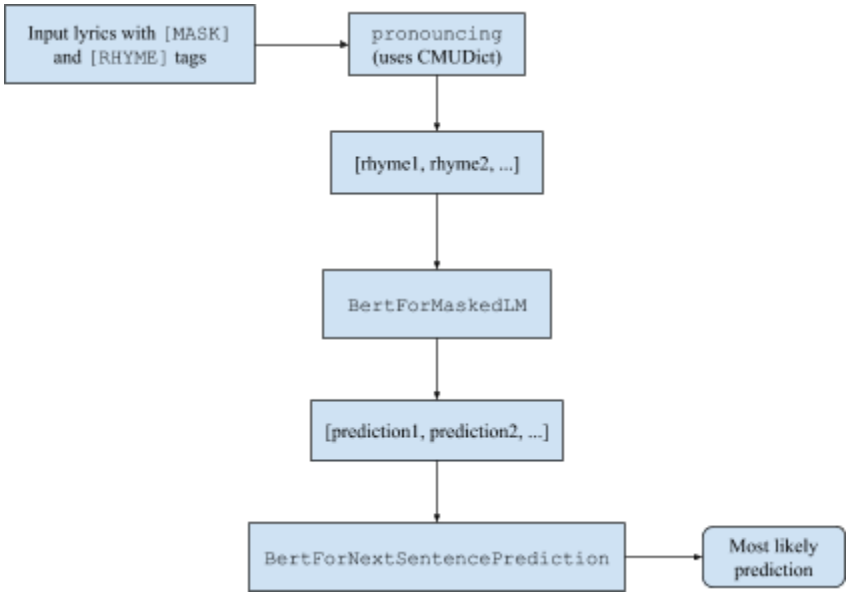
`<To be predicted>`		
The world was that way *when I got hear* *We all link hands when the crowd cheers* `<To be predicted>`	*again*	*loudly*

Running an evaluation script with these ground truth and prediction files as inputs, we get an Accuracy of 0.31 and a BLEU score of 0.80. However, we do not use the BLEU score here as the prediction here involves only a single word than the entire lyrics.

3.4.2 Whole lyric prediction

For the entire target lyric prediction, we presume an AABB rhyming scheme, i.e., a given line of lyrics rhymes with the preceding line of lyrics. In the target lyric, we replace all the words except the last word by the `[MASK]` tag. The last word is replaced by a special `[RHYME]` tag. We use the `pronouncing` Python library[21], which is a simple interface for the CMU Pronouncing Dictionary[22], to search for phones, number of syllables and rhyme words for the last word (a.k.a focus) of the preceding line of lyrics to the target lyric. We consider all words that contain the same number of syllables as the focus and also have same phonetic ending as the focus, to be the possible last word for the target lyric. In case we do not find any appropriate rhyming word based on these criteria, we simply choose the focus to be the last word.

For each possible rhyming word we end up with, we replace the `[RHYME]` tag with this rhyming word and feed the tokenized lyrics to the `BertForMaskedLM` LM, which then gives predictions for the masked-out words in the target lyric. Thus for each chunk of 5 line lyrics, we get a set of possible 4^{th} line lyrics. We feed each of the predicted lyric into the `BertForNextSentencePrediction` LM along with the context lyrics and find the most probable 4^{th} lyric. In implementation, we try to keep a record of top-3 predictions. But we take the most probable lyric to be our final predicted lyric. A schematic of the procedure is shown below:



The implementation can be found in the Jupyter notebook `rhymings.ipynb`. The ground truths and the predicted lyrics are available in the files *test_gold_file.txt* and *test_pred_file.txt* in the `rhymings` subdirectory of the `output` directory. An example of the context lyrics, the ground truth and the predicted lyrics is shown below:

Context	Predicted Lyric	True Lyric
It's tearing me apart *It's ruining every day* *For me* `<To be predicted>` *And fellow, so did you*	*I just wanted to ... flee*	*I swore I would be true*

For this example, the top-3 predicted lyrics were found to be:

1. *I just wanted to ... flee*
2. *"I was never really free*
3. *I heard the same desperate plea*

Some more examples are shown below:

Context	Predicted Lyric	True Lyric	Top-3 predictions
Better to write than to have read	*Better to have read*	*Easier done that said*	1. *Better to have read*

Better to cope than to complain *Better in your hands that in your head* <To be predicted> *Like a dying flower that just don't give a damn*			2. *You can still Reade* 3. *Or in your bed*
Im not high on life *Im not drunk on love* *Im broken down , not feeling right* <To be predicted> *And theres nothing on the inside*	*Im not drunk on my spite*	*Im happy as Im gonna be*	1. *Im not drunk on my spite* 2. *Im not a good love spight* 3. *Im not feeling so damn tight*
Sorry, that's the last stop *I'll see you* *I'll see you* <To be predicted> *I'll see you*	*Thanks, Miss Crewe*	*On the way down*	1.*Thanks, Miss Crewe* 2. *Bye, Miss Joo* 3. *I will, yoo*

Running an evaluation script with these ground truth and prediction files as inputs, we get a BLEU score of 0.49. However, we cannot really use either Accuracy or BLEU score here as there can be multiple possible predictions for the required lyrics. Since it is difficult to quantify the predicted lyrics as correct using Accuracy or BLEU score, we would ideally have to rely on human judgements for understanding the accuracy of lyrics prediction.

3.5 Song Reconstruction

Another task we thought would be an interesting way to assess the performance of the model was trying to reconstruct a song from its lines, i.e., consider all of the lines in the songs and choose an order to arrange them in. In this way, we can measure how close to the original song our reconstruction is. Then, if our model is good, we can generate original songs by considering arbitrary sets of lines instead of only lines that come from a particular song.

3.5.1 Evaluation

The model is primarily concerned with using one line context to predict the next lyric of the song. Thus, our goal with an evaluation metric is to see how well the model can select the next line of a song, without being concerned as to how those two lines fit into the broader idea of the song. We opted to evaluate the model by considering each line in the song, and trying to

predict the next line, with the options being all of the lines in the original song. In this way, we have *n*-1 (where *n* is the number of lines in the song) instances of correct or incorrect predictions, and can calculate the accuracy of our model. Additionally, we can calculate the BLEU score using the function available in the NLTK library. Note that in general this strategy does not necessarily directly correlate with the model's ability to construct an original song - when putting the model to use, "errors" will compound, as the third line will be predicted from the erroneous second line. What we are measuring here is the model's ability to predict the third line based on the *actual* second line. However, we believe that this evaluation metric will provide valuable insight into the model's ability to model one line of context for predicting a subsequent line, and so models that would be good at creating original songs will in fact score higher in our testing.

3.5.2 Random Baseline

For this task, the simple baseline that we compared the model to was randomly selecting the next lyric from a set of all lyrics in the song. This worked well with the evaluation metric we chose because there is no interaction between the selections. In theory, this baseline on average scores an accuracy of 1/*n*, since for each line there is a 1/*n* chance that the randomly selected line is actually the one that follows it in the original song. In practice however, we found that the baseline ended up performing slightly better than that, due to the repetition of lines in many songs.

3.5.3 BERT Baseline

Next, we tried using BERT to help make more intelligent predictions. Our hypothesis was that on average, any given line in a song would tend to be a fitting, logical, continuation of the line that came before it. We used the `BertForNextSentencePrediction` LM to perform the procedure described in the evaluation section, but instead of choosing the next lyric randomly, we chose the lyric that BERT thought was most likely the next lyric.

3.5.4 BERT + Rhyming

By using just BERT, we were essentially just doing next sentence prediction on text that happened to be song lyrics, which we knew could be improved by including features that songs have that general text does not. We used the `pronouncing` library, which meant that we could tell when words rhymed. Thus, we added another layer to the model - when considering a set of lines and trying to decide which one followed some previous line, we applied an upwards adjustment in probability for lines whose last word rhymed with the last word of the previous line. In this way, we would be better able to capture the additional signal provided by song lyrics over other forms of text, as well as make it so that the lines that the model generates when creating original works will rhyme with each other.

3.5.5 Results

The implementation is done in the Jupyter notebook `reconstruction.ipynb`. For this task, there was not really any training to be done, nor hyperparameters to vary. The only hyperparameter that we considered varying was how heavily to value the fact that two lines rhymed at points where BERT disagreed - the best results were achieved when rhyming was believe completely over BERT, and BERT used as a tiebreaker or second step if there were still more potential lines (multiple rhyming lines or no rhyming lines). In order to obtain the results reported below, 10 songs were randomly selected (from among songs with no more than 20 lines), and the correct and incorrect prediction counts were combined for this final result. Although we would have rather done every song, or at least more songs, this model takes a long time to run, especially when there are so many choices for each line pair, so we were time constrained in how many songs we were able to run this on. The ground truths are available in the file *gold_results.txt*, random baseline results are available in the file *random_results.txt*, the BERT baseline results are available in the file *bert_results.txt*, and the BERT + Rhyming results are available in the file *rhyme_results.txt*. All these files can be found in the `reconstruction` subdirectory of the `output` directory.

	Accuracy	**BLEU score**
Random Baseline	0.05	0.18
BERT Baseline	0.10	0.22
BERT + Rhyming	0.14	0.24

For the BERT + Rhyming scheme, we try to generate songs by choosing a lyric randomly, giving 20 choices for each lyric and then 50 choices for each lyric. The results are described in **Appendix 1**.

4 Conclusions

One area that we believe could significantly improve the performance of our extension is the ability to detect and enforce rhyming schemes. Essentially, our model is assuming an AABB rhyming scheme, which is not the rhyming scheme used by the vast majority of songs. This weakness comes partially as a result of only analyzing songs with the context of the immediately preceding line - given this, AABB is really the only rhyming scheme that we can employ. Assuming it is better than not assuming it, because it doesn't usually hurt as there is a relatively low chance that two arbitrary lines rhyme - the chances of a false positive is much lower than the chances of a true positive. However, if we were able to include rhyme schemes like ABCB, we believe that the performance of our model would greatly improve.

For the time being, this just means that our model performs far better on songs that actually employ an AABB rhyme scheme - in fact, just using the rhyming information and no BERT

performs better than just BERT, since the rhyming sound tends to be relatively unique for each pair (AABBCC as opposed to AABBAA). An example of this is the song *Dead Heat* by The Dickies[7] - the lyrics and results of running the baselines and our model on this song is shown in **Appendix 2**.

Another extension that could have helped our results was the ability to detect near-rhymes. There were many cases when looking manually through results that we saw lines that would have been marked as rhyming by a human or a more sophisticated rhyme model, but not the simple rhyming model that we were using.

A final improvement that we would be interested in is having subsequent choices interact with each other by disallowing the model from predicting the same line to show up in two different places.

The code is available in the GitHub repo https://github.com/abhishek7t/Lyrics-Generation. The presentation slides are available at the Google Drive link.

Appendix 1

Generated Lyrics (Random) :

I did it justice I just did it for us all
I rise up to the street
With a love that I know pure as new fallen snow
I couldn't help it
I got so used to her somehow
Of every history revisionist
I'm looking to the sun
They come again in the morning!
Oh what a shame (x 2)
If they were giving it up to you

Generated Lyrics (20 choices) :

You made me happy
Jimmy Carl Black (drums)
You tell me you don't love me over a cup of coffee
it's hot it's hot tonight
I prayed for a light
The saddest thing's to see him venerate that ball and chain
Yes the wonder of the Tundra
Time comes to a halt
he's fighting until he dies
Oh no, we won't give in, let's go living in the past.
And it's a strange infatuation taking off across the nations
Beat is getting stronger
You apologize but I don't hear

You'll be sayin', you'll be saying
Sometimes big wave coming when I'm down

Generated Lyrics (50 choices) :

To condemn the man who dies
I need someone
Somewhere there's a place a soul can never go,
Can't get no you know
We're doin' a nu thang
Is there life after talk 'cause there's talk on the road
Changing rearranging every single day
Care what these people say
Won't you walk away
fuck what the others have to say

Appendix 2

Song: dead-heat

Lyrics:

put them up against the wall
pull the trigger watch them fall
they can only feel the pain
stand them up and start again
tell the sargeant what you saw
fear the long arm of the law
even though it's hanging there
drugs and bad guys you'd better beware
of dead heat they're dead heat
if you shoot 'em down they'll be back on thier feet
they're dead heat they're dead heat
if you shoot 'em down they'll be back on the street
take 'em down contempt divine (?)
a cat that looks like frankenstein
he's holding up a jewellery store
listen to his bullets roar
their job is done they're all alone
they work their fingers to the bone
they're weary as they walk their beat
all day long they're dead on their feet
they're dead heat they're dead heat
certified zombies from their head to their feet
they're dead heat they're dead heat
if you shoot 'em down they'll be back on thier feet

	Accuracy	**BLEU score**
Random Baseline	0.10	0.22
BERT Baseline	0.05	0.12
BERT + Rhyming	0.38	0.56

References

[1] "*Better Language Models and Their Implications*", OpenAI Blog, 2019, URL - https://openai.com/blog/better-language-models/

[2] A.Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, I. Polosukhin, 2017, "*Attention Is All You Need*", URL - https://arxiv.org/pdf/1706.03762.pdf

[3] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, 2018, "I*mproving Language Understanding by Generative Pre-Training*", URL - https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf

[4] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, S. Fidler, 2015, "*Aligning books and movies: Towards story-like visual explanations by watching movies and reading books*", In Proceedings of the IEEE international conference on computer vision, pages 19–27

[5] J. Devlin, M. Chang, K. Lee, K. Toutanova, 2018, "*BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*", URL - https://arxiv.org/pdf/1810.04805.pdf

[6] W. Taylor, 1953, "*Cloze procedure: A new tool for measuring readability*", Journalism Bulletin, 30(4):415–433

[7] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, 2019, "*Language Models are Unsupervised Multitask Learners*", URL - https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

[8] A. Wang, K. Cho, 2019, "*BERT has a Mouth, and It Must Speak: BERT as a Markov Random Field Language Model*", URL - https://arxiv.org/pdf/1902.04094.pdf

[9] E. Greene, T. Bodrumlu, K. Knight, 2010, "*Automatic Analysis of Rhythmic Poetry with Applications to Generation and Translation*", URL - https://www.aclweb.org/anthology/D10-1051

[10] M. Mohri, 1997, "*Finite-state transducers in language and speech processing*", URL - https://aclweb.org/anthology/J97-2003

[11] M. Mohri, F. Pereira, M. Riley, 2002, "*Weighted finite-state transducers in speech recognition*", URL - https://repository.upenn.edu/cgi/viewcontent.cgi?article=1010&context=cis_papers

[12] A. Dempster, N. Laird, D. Rubin, 1977, "*Maximum Likelihood from Incomplete Data Via the EM Algorithm*", URL - http://web.mit.edu/6.435/www/Dempster77.pdf

[13] M. Ghazvininejad, X. Shi, Y. Choi, and K. Knight, 2016, "*Generating Topical Poetry*", URL - https://www.aclweb.org/anthology/D16-1126

[14] "*Finite State Machines - Acceptors (recognizers)*", Wikipedia, URL - https://en.wikipedia.org/wiki/Finite-state_machine#Acceptors_(recognizers)

[15] D. Rumelhart, G. Hinton, R. Williams, 1986, "*Learning representations by back-propagating errors*", Nature, **323** (6088): 533–536

[16] M. Ghazvininejad, X. Shi, J. Priyadarshi, K. Knight, 2017, "*Hafez: an Interactive Poetry Generation System*", URL - https://aclweb.org/anthology/P17-4008

[17] "*PyTorch Pretrained BERT: The Big & Extending Repository of pretrained Transformers*", GitHub, URL - https://github.com/huggingface/pytorch-pretrained-BERT

[18] "*380,000+ lyrics from MetroLyrics*", Kaggle, URL - https://www.kaggle.com/gyani95/380000-lyrics-from-metrolyrics/data

[19] Papineni et al, 2002*, "BLEU: a Method for Automatic Evaluation of Machine Translation"*, URL - https://www.aclweb.org/anthology/P02-1040.pdf

[20] "*langdetect 1.0.7 - Language detection library ported from Google's language-detection*", PyPI, URL - https://pypi.org/project/langdetect/

[21] "*pronouncing 0.2.0 - A simple interface for the CMU pronouncing dictionary*", PyPI, URL - https://pypi.org/project/pronouncing/

[22] "*The CMU Pronouncing Dictionary*", URL - http://www.speech.cs.cmu.edu/cgi-bin/cmudict