CS 6120: Song Lyric Generator

Rajat Keshri, keshri.r@northeastern.edu
Samanvya Tripathi, Tripathi.sam@northeastern.edu
Tannishtha Mandal, mandal.t@northeastern.edu
GitHub - https://github.com/rajatkeshri/NLP-Project-Lyric-Gen

Abstract

Lyrics generation is an interesting and challenging Natural Language Processing (NLP) task. The generation of lyrics requires the model to understand the language's context, meaning, and structure. In this paper, we propose and compare multiple NLP models for generating song lyrics: Naïve Bayes Language N-Gram model, Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM) and Transformers. The N-Gram Generator model is a probabilistic algorithm that is used for generating sequences of words based on their frequency of co-occurrence in the training corpus, while the RNN LSTM is a deep learning model that is widely used for sequence generation tasks, and transformers are encoder decoder structure which learn to generate text using two feed forward networks. To evaluate the performance of the models, we use Taylor' swifts song dataset and train them on different models. We use the perplexity metric and BLEU score to measure the model's ability to predict the next word in the sequence. Furthermore, we perform a qualitative analysis of the generated lyrics to evaluate the creativity and coherence of the generated lyrics (based on rhyming scheme "aabb" and "abab". Overall, this study demonstrates the potential of NLP techniques in generating lyrics and highlights the superiority of deep learning models such as RNN LSTM and transformers for this task. The results of this study could be useful for developing lyric generation systems in the music industry, helping songwriters and artists to generate lyrics more efficiently and effectively.

1.1. Introduction

Lyrics generation is an important task in the field of natural language processing (NLP) and has gained significant attention in recent years due to its potential applications in the music industry. The ability to generate creative and meaningful lyrics can be a valuable tool for songwriters and artists who are looking to create new music content. However, the generation of lyrics is a complex task that requires the model to understand the language's context, meaning, and structure.

The ability to generate lyrics for a song using a machine learning model has numerous potential applications in the music industry, from helping songwriters come up with new ideas to providing inspiration for musicians who are struggling with writer's block. However, generating lyrics that not only sound good but also follow the standard structure and rhyme patterns of a song can be a challenging task for an AI model.

To address this challenge, our project aims to build a text generation model that can produce song lyrics that follow the fundamental composition of a song, including an introduction, verse, pre-chorus, chorus, bridge, and outro. We plan to achieve this by using a combination of NLP techniques, such as N-Gram models and RNN LSTMs, to capture the statistical patterns of language and generate sequences of words that are both grammatically correct and semantically coherent.

Moreover, we aim to incorporate the constraint of rhyming patterns into our model to generate lyrics that are musically appealing. In songwriting, rhyming is a crucial aspect that gives a song its unique character and helps to establish a sense of rhythm and flow. We plan to implement a rhyming algorithm that ensures that the end of every sentence or verse in the generated lyrics has words that rhyme with the next verse, following "aabb" or "abab" type of rhyme sequence.

Overall, our project has the potential to provide songwriters and musicians with a tool that can generate high-quality and creative lyrics while also following the standard structure and rhyme patterns of a song. We believe that this project can contribute to the field of natural language processing and music composition and lead to exciting new possibilities for AI-generated music.

1.2. Background/Related Work

Several studies have been conducted in the area of lyrics generation using N-Gram models and RNN LSTMs. For instance, Harrison Gill, Daniel Lee, and Nick Marwell proposed a model that generates lyrics by combining Markov chains and RNN LSTMs [1]. The study demonstrated that the proposed model can generate lyrics that are comparable in quality to those written by human songwriters.

Another study Zihao Wang proposed an N-Gram language model for lyrics generation and evaluated its performance using perplexity and human evaluation metrics [2]. The study showed that the proposed model can generate lyrics that are grammatically correct and semantically coherent.

In addition, a study in Transformers: State-of-the-Art Natural Language Processing [3] proposed a system for generating song lyrics using a transformer encoder decoder language model. The system uses the transformers for the lyric generation with a technique called "top-k sampling," which involves selecting the k most probable words at each step in the generation process and randomly choosing one of them to be the next word in the sequence to generate new lyrics. The study showed that the proposed system can generate lyrics that are comparable in quality to those written by human songwriters.

Despite these efforts, there is still much room for improvement in the field of lyrics generation using N-Gram models and RNN LSTMs. In particular, there is a need for more research on the use of different feature extraction techniques and model architectures to generate high-quality and creative lyrics. The current study addresses this gap by comparing the performance of different N-Gram models and RNN LSTM architectures for English lyrics generation.

2. Methodology

2.1. Data Description:

2.1.1. Kaggle Dataset:

For this project, we used a dataset of Taylor Swift's song lyrics obtained from Kaggle. The dataset consists of a CSV file with two columns: one containing the song's title and the other containing the lyrics. Figure 1 shows the overall distribution of the dataset. The dataset contains lyrics for all of Taylor Swift's albums, including her earliest work and her most recent releases. The structure of the data can be seen in Figure 3.

The dataset contains 93 songs, a total of 35250 words (tokens), and the vocab size is 33568. The songs are from Taylor Swift's six studio albums: Taylor Swift (2006), Fearless (2008), Speak Now (2010), Red (2012), 1989 (2014), and Reputation (2017).

The dataset is provided in a CSV file format and contains four columns: "album", "track_title", "track_n", and "lyrics". The "album" column indicates the album name, the "track_title" column specifies the song title, the "track_n" column lists the order of the song in the album and the "lyrics" column contains the complete lyrics of the song. The lyrics in the dataset follow the standard structure of a song with a verse, chorus, bridge, and sometimes a pre-chorus or outro.

2.1.2. HuggingFaceArtistTaylorSwift:

GPT-2 is a large-scale language model with over 1.5 billion parameters. Larger models have a higher capacity to capture more complex patterns and dependencies in the data. Thus, they need more data to learn the patterns and relationships in the language it is being trained on. To fully utilize their capacity and achieve the best possible performance, GPT-2 model was trained on the HuggingFaceArtistTaylorSwift dataset.

The HuggingFaceArtistTaylorSwift dataset is a collection of lyrics from the popular American singer-songwriter Taylor Swift. The dataset contains a total of 762 songs, spanning her career from her debut album in 2006 to her most recent album release in 2020. The lyrics are available in plain text format, with one song per file. Each file contains the title of the song, the album it was released on, the year of release, and the lyrics themselves. The dataset is well-organized, with consistent naming conventions and file structures. It provides a great opportunity for natural languages processing tasks such as sentiment analysis, text generation, and language modeling. Additionally, the popularity of Taylor Swift and her music makes this dataset an excellent resource for studying the evolution of her style and themes over time.

2.2. Data -preprocessing Steps:

Before we can train our models to generate lyrics, we need to preprocess the data to make it suitable for machine learning. The following steps were taken to preprocess the data:

2.2.1. Tokenization:

The first step is to break down the lyrics into individual words or tokens.

2.2.2. Text Cleaning:

This step involves converting all text to lowercase to avoid the model treating the same word in different cases as different tokens.

2.2.3. Lowercase:

This step converts all text to lowercase to avoid the model treating the same word in different cases as different tokens.

2.2.2. Stop words Removal:

The next step is to tokenize the lyrics into individual words. This involved splitting the lyrics into a sequence of words, which we could then use as input for our models.

2.2.3. Stemming/Lemmatization:

This step converts words to their base form to reduce the dimensionality of the feature space.

2.2.4. Padding/Truncation:

This ensures that all input sequences have the same length by padding or truncating them to a fixed length.

2.2.5. Encoding:

Finally, the text data needs to be encoded into a numerical format, such as one-hot encoding, word embedding, or character embedding, depending on the model being used.

Unknown words and words with frequency less than 3 are treated as <UNK> characters in the project for N-Gram Language Model. Figure 2 shows the words and their frequency, where we can see the diversity in words on the dataset.

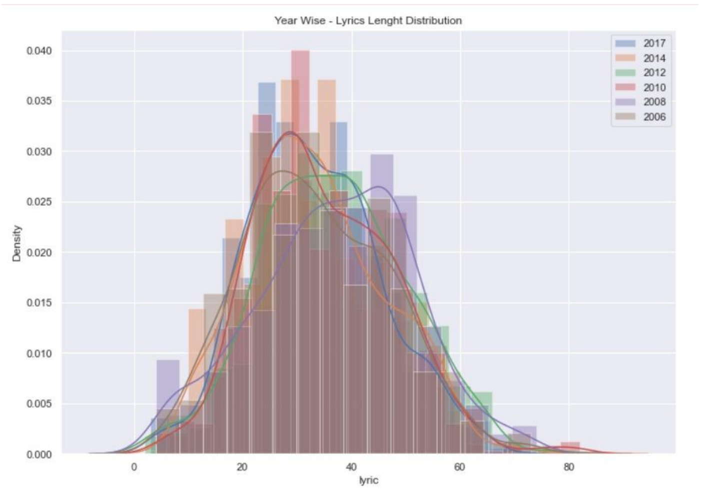


Figure 1: This graph shows the year-wise distribution of the lyric length of Taylor Swift songs.

2.3. Models:

The goal of this project is to train models to generate lyrics from various artists. In order to achieve this goal, we used a variety of generative machine learning approaches, including Naive Bayes N-gram, RNN with LSTM, and Transformers (GPT-2).

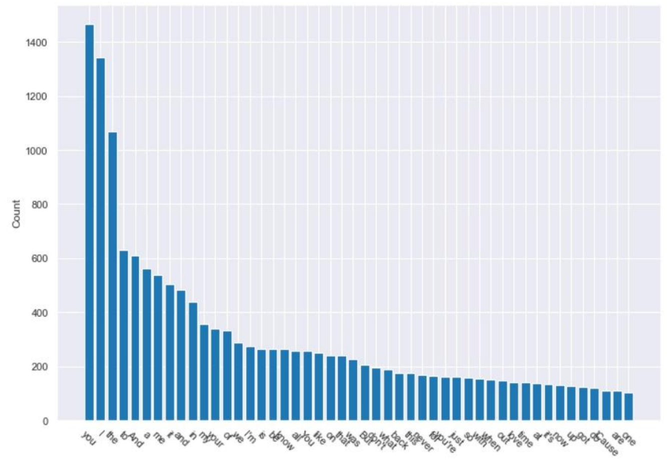
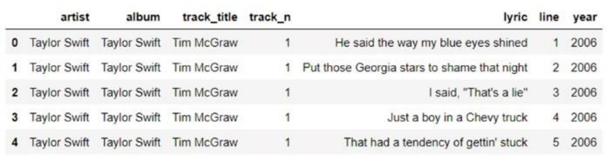


Figure 2: This graph shows the words that are used the most in Taylor Swift's songs, words with less than a 100 count are not shown in the graph for brevity.



	artist	album	track_title	track_n	lyric	line	year
0	Taylor Swift	Taylor Swift	Tim McGraw	1	He said the way my blue eyes shined	1	2006
1	Taylor Swift	Taylor Swift	Tim McGraw	1	Put those Georgia stars to shame that night	2	2006
2	Taylor Swift	Taylor Swift	Tim McGraw	1	I said, "That's a lie"	3	2006
3	Taylor Swift	Taylor Swift	Tim McGraw	1	Just a boy in a Chevy truck	4	2006
4	Taylor Swift	Taylor Swift	Tim McGraw	1	That had a tendency of gettin' stuck	5	2006

Figure 3: This table depicts the structure of the dataset in hand, a compilation of Taylor Swift's discography.

2.3.1. Naive Bayes N-Gram:

For text classification problems, the Naive Bayes algorithm is frequently employed since it is straightforward and quick. Naive Bayes is typically used for text classification tasks, but it can also be used for text generation by taking advantage of n-grams. An n-gram is a contiguous sequence of n items from a given sample of text, where n is a positive integer. For example, in the sentence "The quick brown fox jumps over the lazy dog," some possible 2-grams (also called bigrams) are "The quick", "quick brown", "brown fox", "fox jumps", etc. [4]

For our use case, Naïve Bayes with N-Gram language modeling with the help of Shannon's method is applied. First, all possible n-grams are extracted from the tokenized text, the frequency of each n-gram token is calculated, and then for each n-gram, the conditional probability of each possible next word given the n-gram is calculated. This is done using Bayes' rule and assuming conditional independence between the n-gram and the next word. The final step is text generation, where we start with a seed phrase (which can be any n-gram), and iteratively choose the next word based on the probabilities calculated. The probability is calculated based on Baye's formula given as (trigram) –

$$P(w_n \mid w_{n-2}w_{n-1}) = \frac{count(w_{n-2}, w_{n-1}, w_n)}{count(w_{n-2}, w_{n-1})}$$

2.3.2. Recurrent Neural Networks (RNN):

Among the several types of neural networks, RNNs excel at tasks requiring sequential data, like text or speech. RNNs can process variable-length data sequences, in contrast to typical feedforward neural networks, which can only handle fixed-size inputs and outputs. They function by keeping an internal state that is modified by each input in the sequence, which enables them to simulate dependencies between various sequence pieces. RNNs can be used to model the structure of the lyrics in the instance of lyric generation because lyrics frequently have a repeating structure, such as verses and choruses [5].

2.3.3. Long Short-Term Memory (LSTM):

LSTMs are a type of RNN that are able to learn long-term dependencies between elements in a sequence. They function by using a number of "gates" that control the information flow through the network. These gates are particularly well-suited for jobs where the network must preserve a memory of past inputs because they enable the network to selectively remember or forget information from earlier time steps. LSTMs can be used to model the structure of the lyrics in the context of lyric generation while also enabling the network to retain crucial information from earlier lines or verses. [6]

2.3.4. Transformers:

The neural network architecture known as "Transformers" was first developed in 2017, is a powerful deep learning model architecture, introduced in a 2017 paper by Vaswani et al that has revolutionized the field of natural language processing.

Transformers are based on self-attention, which allows the model to focus on different parts of the input sequence at different processing stages. This enables the model to capture long-range dependencies in the input sequence. Thus, they have the capacity to process complete input sequences simultaneously, in contrast to RNNs which process input sequences one element at a time [7].

One of its key advantages is its ability to be pre-trained on large amounts of text data using unsupervised learning. This pre-training allows the model to learn a great deal about the structure of natural language, which can then be fine-tuned on smaller datasets for specific NLP tasks. This has made it a valuable tool for a wide range of NLP tasks, like translating or creating text. OpenAI's GPT2 (Generative Pretrained Transformer 2) is a specific implementation of the Transformer architecture that was released in 2020. GPT2 is an advanced language model trained on a massive amount of text data, allowing it to generate high-quality, human-like text in various contexts. GPT2 offers several key features and benefits that make it a powerful tool for NLP tasks, including:
Large Training Data: GPT2 has been trained on a significantly larger amount of text data than previous implementations of Transformers, which has allowed it to achieve state-of-the-art performance on a wide range of NLP tasks. It is better suited for applications that require high accuracy and broad coverage of the language, such as language translation.
High Scalability: GPT2 has been designed to be highly scalable, allowing it to process large amounts of text data quickly and efficiently. This makes it well-suited for applications that require real-time text processing, such as chatbots and voice assistants.
Generative capabilities: GPT2 is a generative model, which means it can be used to generate new text based on a given input. This makes it particularly useful for tasks like text generation, where it can be used to produce high-quality, coherent text that resembles human writing.
High accuracy: GPT2 has been trained on a massive corpus of text data, allowing it to accurately model the structure and nuances of natural language. This makes it well-suited for a wide range of NLP tasks, from language translation to text summarization.

GPT2 volumes are a versatile and cost-effective option for general-purpose workloads that require consistent performance and moderate I/O throughput. With their baseline and burst performance capabilities, elasticity, and pay-as-you-go pricing model, GPT2 volumes provide a great balance between performance and cost for a wide range of applications.

In summary, each of these methods has its own strengths and weaknesses, and their suitability for the lyric generation task will depend on factors such as the size and complexity of the dataset, as well as the specific requirements of the task. By using a combination of these methods, we can develop more sophisticated models that are better able to generate high-quality lyrics.

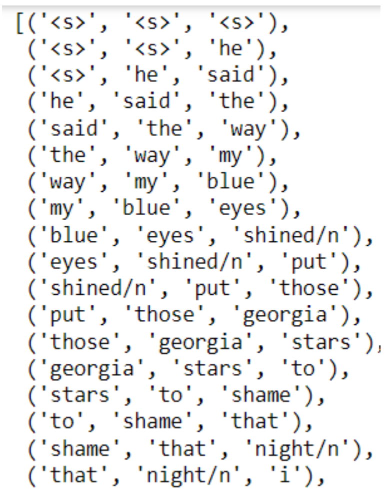
3. Experiments

We have conducted our experiments on 3 different machine learning models – Naïve Bayes N-Gram, RNN LSTM, and Transformer.

3.1. Naïve Bayes N-Gram:

From the dataset, we see that each sentence in the song is approximately 5-8 words long. After every 5-8 words, a new phrase or sentence is present in the song. Usually, the last word of each sentence rhyme with the next sentence's last word. We differentiate each of the ending words of each sentence by adding "/n" to the word. Suppose the following are two of the sentences in the dataset –
"Had to listen to all this drama/n" and "Me and drama always go on/n"
In the above sentences the words "drama/n" and "drama" will be treated as different words by the tokenizer. This is done because we want to capture ending words separately as they tend to rhyme with the next sentence's last word.

Next, we define the N for N-gram. In our case, we have used N = 3 (trigram). N-gram tokens are created as depicted in Figure 4.

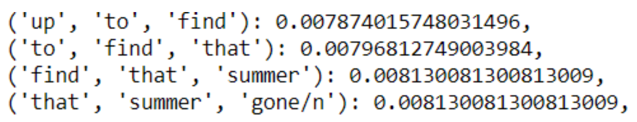


```
[('<s>', '<s>', '<s>'),
 ('<s>', '<s>', 'he'),
 ('<s>', 'he', 'said'),
 ('he', 'said', 'the'),
 ('said', 'the', 'way'),
 ('the', 'way', 'my'),
 ('way', 'my', 'blue'),
 ('my', 'blue', 'eyes'),
 ('blue', 'eyes', 'shined/n'),
 ('eyes', 'shined/n', 'put'),
 ('shined/n', 'put', 'those'),
 ('put', 'those', 'georgia'),
 ('those', 'georgia', 'stars'),
 ('georgia', 'stars', 'to'),
 ('stars', 'to', 'shame'),
 ('to', 'shame', 'that'),
 ('shame', 'that', 'night/n'),
 ('that', 'night/n', 'i'),
```

Figure 4: This figure depicts the trigram

Once we have the N-gram tokens, we calculate the probability of each token appearing, using Baye's rule. This is depicted in Figure 5

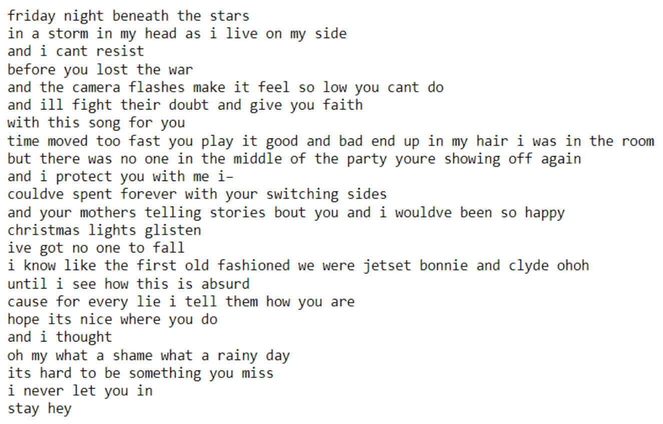
.



```
('up', 'to', 'find'): 0.007874015748031496,
('to', 'find', 'that'): 0.00796812749003984,
('find', 'that', 'summer'): 0.008130081300813009,
('that', 'summer', 'gone/n'): 0.008130081300813009,
```

Figure 5: Tokens and their probability

The probability matrix is used to then generate lyrics. Figure 6 displays sample lyrics generated. We see that the model produces good lyrics and does look like Taylor Swift's song.



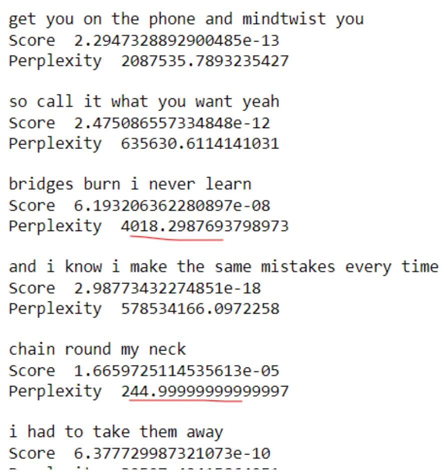
```
friday night beneath the stars
in a storm in my head as i live on my side
and i cant resist
before you lost the war
and the camera flashes make it feel so low you cant do
and ill fight their doubt and give you faith
with this song for you
time moved too fast you play it good and bad end up in my hair i was in the room
but there was no one in the middle of the party youre showing off again
and i protect you with me i-
couldve spent forever with your switching sides
and your mothers telling stories bout you and i wouldve been so happy
christmas lights glisten
ive got no one to fall
i know like the first old fashioned we were jetset bonnie and clyde ohoh
until i see how this is absurd
cause for every lie i tell them how you are
hope its nice where you do
and i thought
oh my what a shame what a rainy day
its hard to be something you miss
i never let you in
stay hey
```

Figure 6: Sample generated lyrics using N-Grams

We have further calculated the score and perplexity. It measures how well the model is able to predict the next word in a given sequence of words. A higher perplexity indicates that the model is less accurate in predicting the next word, while a lower perplexity indicates a more accurate model. We see from figure 7 that this model does perform well in certain cases and its perplexity is low, but in most cases, we see the perplexity is very high, which shows the model produces inaccurate results. This is major because of the size of the dataset. The dataset has 35250 words, which is very less to learn and master the art of writing lyrics. Another way to reduce perplexity and improve model performance is by applying Good-Turing smoothing, or Kneser-Ney smoothing to adjust the probabilities of rare n-grams and reduce the impact of out-of-vocabulary words.

3.2. RNN - LSTM

For the preprocessing, we split the dataset into sequences of length 7. This acts as training data for our RNN-LSTM



```
get you on the phone and mindtwist you
Score  2.2947328892900485e-13
Perplexity  2087535.7893235427

so call it what you want yeah
Score  2.475086557334848e-12
Perplexity  635630.6114141031

bridges burn i never learn
Score  6.193206362280897e-08
Perplexity  4018.2987693798973

and i know i make the same mistakes every time
Score  2.98773432274851e-18
Perplexity  578534166.0972258

chain round my neck
Score  1.6659725114535613e-05
Perplexity  244.99999999999997

i had to take them away
Score  6.377729987321073e-10
```

Figure 7: Perplexity and Score of random input sentences

model, a small part of the data has been set aside for validation and testing. We obtain a total of 25,694 valid sequences of size 7 from the Taylor Swift dataset that we have. The model consists of an initial Embedding layer which learns the word embeddings and is tuned according to the dataset. Then comes the bidirectional LSTM layer which learns words and their context from both left and right contexts. And finally, we have dense layer which points us to the generated word. We have trained the model for 30 epochs since we reached a desirable training accuracy of 96%, and used this trained model for generating our lyrics. The figure 8 displays the general Bidirectional RNN structure and figure 11 displays the architecture used in this paper.

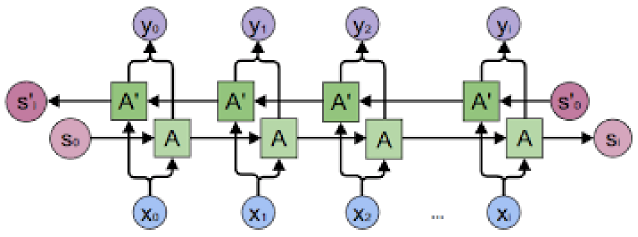
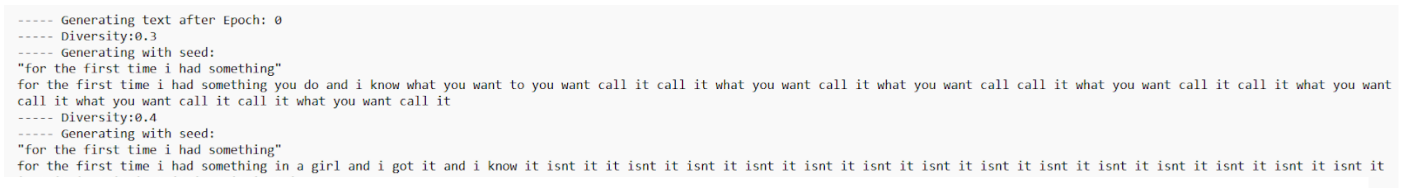


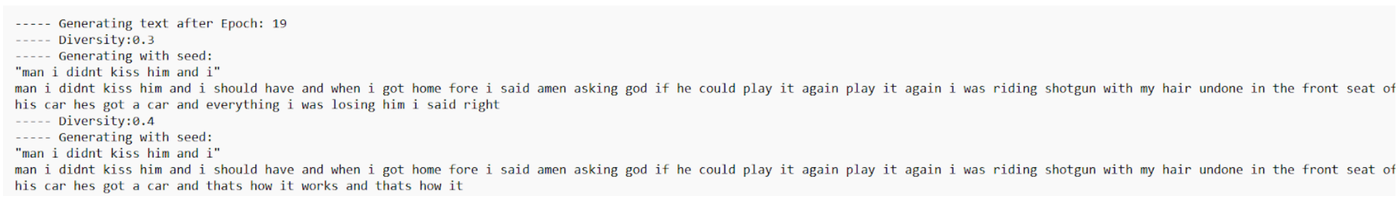
Figure 8: Bidirectional LSTM architecture

The trained model was used to generate sentences and we experimented with the temperature, which controls the randomness of the generation. Higher temperature generates very unique and fresh lyrics, whereas lower temperature generates a standard set of lyrics which may also result in a lot of repeated words/sentences. We empirically used 0.6 as the temperature and were able to



```
----- Generating text after Epoch: 0
----- Diversity:0.3
----- Generating with seed:
"for the first time i had something"
for the first time i had something you do and i know what you want to you want call it call it what you want call it what you want call call it what you want call it call it what you want
call it what you want call it call it what you want call it
----- Diversity:0.4
----- Generating with seed:
"for the first time i had something"
for the first time i had something in a girl and i got it and i know it isnt it it isnt it isnt it isnt it isnt it isnt it isnt it isnt it isnt it isnt it isnt it isnt it isnt it isnt it
```

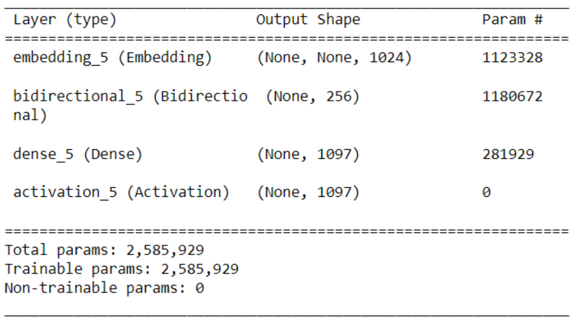
Figure 9: Performance of the model at epoch 0 with different diversity/temperatures



```
----- Generating text after Epoch: 19
----- Diversity:0.3
----- Generating with seed:
"man i didnt kiss him and i"
man i didnt kiss him and i should have and when i got home fore i said amen asking god if he could play it again play it again i was riding shotgun with my hair undone in the front seat of
his car hes got a car and everything i was losing him i said right
----- Diversity:0.4
----- Generating with seed:
"man i didnt kiss him and i"
man i didnt kiss him and i should have and when i got home fore i said amen asking god if he could play it again play it again i was riding shotgun with my hair undone in the front seat of
his car hes got a car and thats how it works and thats how it
```

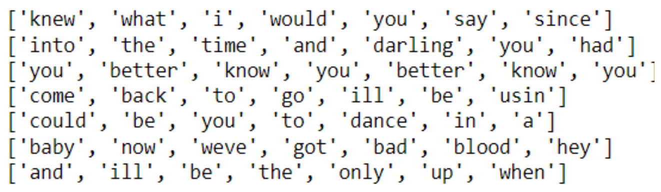
Figure 10 Performance of the model at epoch 19 with different diversity/temperatures.

generate very good sentences that sound like right out of a Taylor Swift song.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, None, 1024)        1123328

 bidirectional_5 (Bidirectio  (None, 256)              1180672
 nal)

 dense_5 (Dense)             (None, 1097)              281929

 activation_5 (Activation)   (None, 1097)              0

=================================================================
Total params: 2,585,929
Trainable params: 2,585,929
Non-trainable params: 0
_________________________________________________________________
```

Figure 11: Our bidirectional LSTM model architecture



```
['knew', 'what', 'i', 'would', 'you', 'say', 'since']
['into', 'the', 'time', 'and', 'darling', 'you', 'had']
['you', 'better', 'know', 'you', 'better', 'know', 'you']
['come', 'back', 'to', 'go', 'ill', 'be', 'usin']
['could', 'be', 'you', 'to', 'dance', 'in', 'a']
['baby', 'now', 'weve', 'got', 'bad', 'blood', 'hey']
['and', 'ill', 'be', 'the', 'only', 'up', 'when']
```

Figure 12: Sentences generated by trained RNN-LSTM with a temperature of 0.6.

Now we see that this model performs better than Naïve Bayes subjectively, as the generated sentences in the case of RNN-LSTM make more sense as compared to Naïve Bayes generations. This is explainable because the model understands the context of the words and where/how they are used. This helps it generate more meaningful lyrics. Figure 9 and 10 displays the results of the training at 1 epoch and 19th epoch respectively. The figure 12 displays an example of the final generated lyrics.

3.3. Transformers

The GPT-2 language model was trained on a massive dataset called WebText, which was crawled from the internet by OpenAI researchers. The dataset was 40GB in size, which is significantly larger than the storage space occupied by common mobile applications. The GPT-2 architecture is based on transformer decoder blocks, which use a stack of transformer blocks to generate text. After each token is produced, that token is added to the sequence of inputs, which becomes the input to the model in its next step. This idea is called "auto-regression" and has made RNNs very effective. Figure 13 shows the GPT-2 architecture.

The GPT-2 architecture consists of a stack of transformer blocks, each of which consists of two sub-layers: a multi-head self-attention layer and a feedforward neural network layer. The multi-head self-attention layer allows the model to attend to different parts of the input sequence simultaneously and learn complex relationships between words. The feedforward neural network layer processes the output of the self-attention layer to generate the final output. GPT-2 also includes positional encodings, which give the model information about the position of each token in the input sequence.

One of the main improvements in GPT-2 over the original GPT architecture is the use of a larger number of transformer blocks and parameters. The original GPT used only 12 transformer blocks, while GPT-2 uses up to 1,550. The number of parameters in GPT-2 is up to 1.5 billion, compared to 117 million in the original GPT. This allows GPT-2 to generate longer and more complex text passages with higher quality and coherence.

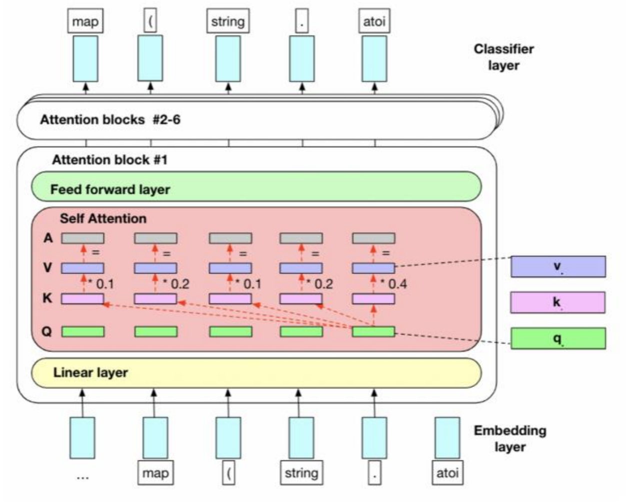
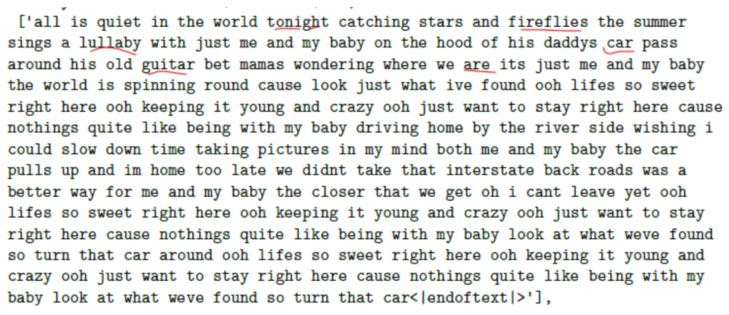


Figure 13: Architecture of GPT-2 Transformer Model

GPT-2 introduced a new training objective called "unsupervised multi-task learning." This objective involves training the model on multiple tasks at the same time, such as language modeling, text classification, and question-answering. This helps the model to learn more general and versatile representations of natural language. As a result, GPT-2 is a powerful and flexible language model that has been used in a wide range of natural languages processing applications, such as text generation, language translation, and question-answering.

The figure 14 represents the lyrics generated by the GPT-2. We can see that the generated lyrics follows rhyming sequence "aabb" scheme. The model learnt how to generate almost perfect lyrics, very similar to the original songs. In the figure, the rhyming sequence is underlined.



```
 ['all is quiet in the world tonight catching stars and fireflies the summer
sings a lullaby with just me and my baby on the hood of his daddys car pass
around his old guitar bet mamas wondering where we are its just me and my baby
the world is spinning round cause look just what ive found ooh lifes so sweet
right here ooh keeping it young and crazy ooh just want to stay right here cause
nothings quite like being with my baby driving home by the river side wishing i
could slow down time taking pictures in my mind both me and my baby the car
pulls up and im home too late we didnt take that interstate back roads was a
better way for me and my baby the closer that we get oh i cant leave yet ooh
lifes so sweet right here ooh keeping it young and crazy ooh just want to stay
right here cause nothings quite like being with my baby look at what weve found
so turn that car around ooh lifes so sweet right here ooh keeping it young and
crazy ooh just want to stay right here cause nothings quite like being with my
baby look at what weve found so turn that car<|endoftext|>'],
```

Figure 14: Generated Lyrics by GPT-2 model

3.4 Results

From the above experiments, we see that the transformer model performed better than that of the RNN – LSTM model and Naïve Bayes Language Model. We have used a single LSTM layer for the RNN model and we achieved an accuracy of approximately 96.16 % on 20 epochs, whereas, the Naïve Bayes Trigram model produced a perplexity score of 200 on average. The ideal perplexity score on unseen data should be as low as possible. However, as the dataset isn't as large in terms of the corpus, the perplexity for unseen data is higher. However, the Naïve Bayes trigram model was able to produce a rhyming sequence well due to the differentiation of end words from the other words.

The RNN model on the other hand has the scope for improvement. Currently, we are training the RNN model with a single Bidirectional LSTM layer of 128 neurons. With increasing the number of layers, it is expected for the model to learn the parameters better, but not to a great extent. We can also see the model performance in figure 15.

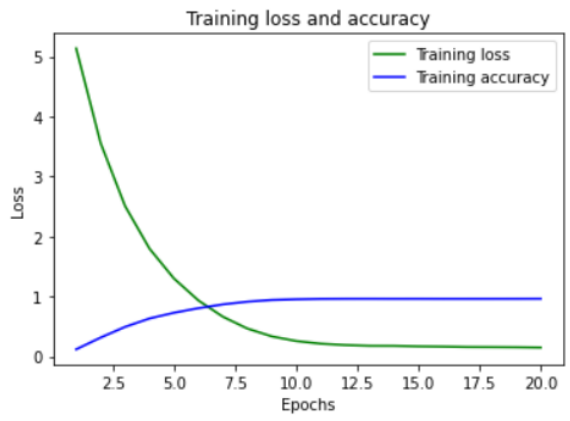


Figure 15: RNN LSTM loss vs accuracy graph

The transformer GPT-2 model learns to generate Taylor swift's lyrics with accurate rhyming sequence. It produces lyrics with "aabb" and "abab" scheme.

Model	Accuracy/Perplexity
Naïve Bayes Trigram Model	Average score of 200 on 10 unseen samples
RNN LSTM	96.16%

Table 1: Comparing model accuracies

Traditional evaluation metrics for language models such as accuracy and perplexity are not always suitable for GPT-2 because its output is not binary or categorical, but rather probabilistic and continuous.

Rogue metrics are a set of evaluation metrics that are specifically designed to evaluate the quality of text generated by language models like GPT-2. These metrics consider various aspects of generated text such as coherence, fluency, diversity, and semantic similarity, which are not captured by traditional metrics like accuracy and perplexity.

ROUGE METRIC	VALUE
ROUGE PRECISION	0.26482994094491
ROUGE RECALL	0.3878714295849204
ROUGE F-MEASURE	0.313116654753685

Table 2: Comparing rogue score for Transformer

Comparing the text generated, we can see that the Naïve Bayes NGram model generated close results to the initial dataset, because it generates text from the dataset itself. It also kept the rhyming sequence intact. Compared to the RNN LSTM-generated lyrics, the RNN model generates unique sequences. It finds it hard to maintain the rhyming sequence. The transformer model performs the best, due to its deep architecture and pre-trained weights from a larger vocab.

3.3. Future Experiments

Future experiments for n-gram language models could include incorporating semantic information, developing context-aware models and exploring hybrid models that combine different types of language models. As natural language processing continues to evolve, new and innovative approaches to language modeling are likely to emerge.

Future experiments for RNN-LSTM language models could include exploring different architectures (with GRU for example), and multi-layer RNNs, which can capture more complex patterns in the data.

The Transformer architecture is a powerful type of language model that has shown significant improvements in natural language processing tasks. Future experiments for Transformer language models could include exploring different variations of the model architecture, such as incorporating different types of attention mechanisms or experimenting with different types of positional encodings. This can also include using GPT-3 or newer GPT model architecture or even the BERT model.

4. Conclusions

In conclusion, N-gram, RNN, and transformer models are all viable options for training a lyrics generator. N-gram models are simple and computationally efficient but may struggle with capturing long-term dependencies and generating coherent and meaningful lyrics. RNNs can handle variable-length sequences and capture complex patterns in the text but may suffer from vanishing or exploding gradients and require significant computational resources. Transformers, on the other hand, are currently state-of-the-art models for natural language processing tasks, including lyrics generation. They can capture long-term dependencies and generate coherent and meaningful text but may require extensive training data and computational resources.

References

[1] Harrison Gill, Daniel Lee, Nick Marwell, *Recurrent neural networks for melody harmonization Deep Learning in Musical Lyric Generation: An LSTM-Based,* The Yale Undergraduate Research Journal., Volume 1, Issue 1 Fall 2020

[2] Zihao Wang. *N-gram and LSTM based Language Models.* Research School of Computer Science, the Australian National University.

[3] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, et al. 2020. *Transformers: State-of-the-Art Natural Language Processing*. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 38–45, Online. Association for Computational Linguistics.

[4] A. K. Yadav and S. K. Borgohain, "Sentence generation from a bag of words using N-gram model," *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies*, Ramanathapuram, India, 2014, pp. 1771-1776, doi: 10.1109/ICACCCT.2014.7019414.

[5] Matthew E. Peters, Waleed Ammar, Chandra Bhagavatula, Russell Power, "*Semi-supervised sequence tagging with bidirectional language models*", arXiv:1705.00108 [cs.CL], Apr. 2017.

[6] Mandar Joshiy, Eunsol Choiy, Daniel S. Weldy, Luke Zettlemoyeryz, "*TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension*", arXiv:1705.03551 [cs.CL], May 2017.

[7] Gareth Ari Aye, Seohyun Kim, Hongyu Li, "Learning Autocompletion from Real-World Datasets" *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*