Genre-Based Song Lyrics Generation

Gharam Walid, Yousef Hassan

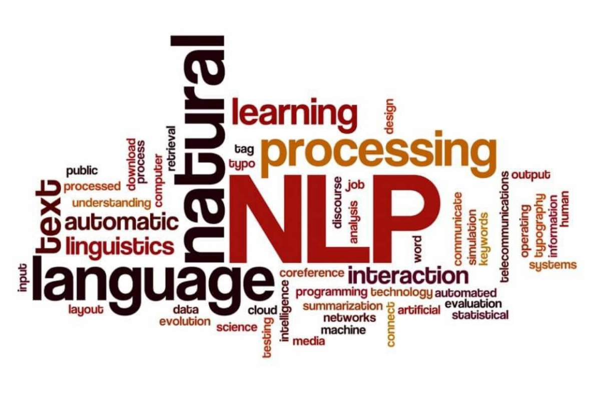
May 25, 2024

1 Introduction

In the era of digital streaming services and personalized playlists, the demand for innovative approaches to music generation and recommendation has surged. Among these, genre-based song lyrics generation stands out as a fascinating avenue, offering a blend of creativity, artificial intelligence, and linguistic analysis. With the advent of advanced machine learning techniques and the abundance of digital music data, researchers and enthusiasts alike are exploring novel methods to automatically generate song lyrics tailored to specific genres. This paper delves into the burgeoning field of genre-based song lyrics generation, discussing its significance, challenges, and recent advancements. By leveraging natural language processing (NLP) algorithms and large-scale datasets, genre-based song lyrics generation holds the potential to revolutionize music creation, empower artists, and enhance listener experiences. Through an exploration of existing methodologies and emerging trends, this paper aims to shed light on the evolving landscape of AI-driven music generation and its implications for the future of music composition and consumption.

2 Literature Review

Neural network-driven language models have demonstrated potential in generating original lengthy written content using minimal initial text, with recent efforts extending to the realm of musical lyric creation. However, crafting lyrics presents a distinct set of challenges not encountered in typical prose. Ensuring appropriate line breaks, mastering stylistic elements such as flow, rhyming, and repetition, and adhering to established lyrical structures like the verse-chorus format are all crucial factors in producing compelling songs. Despite varying degrees of success in these endeavors, it's important to recognize the broad spectrum encompassed by the term "lyrics." Different music genres exhibit unique lyrical styles, characterized by variations in line length, word repetition, semantic nuances, and narrative perspectives. Our research endeavors to explore the capacity of neural networks to generate genre-specific song lyrics while preserving the intrinsic linguistic features characteristic of each musical genre.



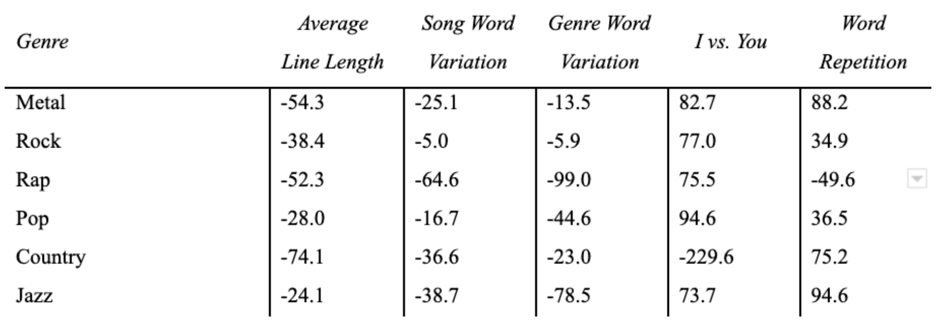
LSTM-based lyrical generation has been investigated within defined music genres. In their work "GhostWriter: Using an LSTM for Automatic Rap Lyric Generation," Potash et al. (2015) [Pot15] endeavored to produce lyrics akin to those of a "ghostwriter," crafting content tailored to a particular artist. Nonetheless, their model's scope was constrained to generating lyrics within a singular genre due to its training on a specific artist. Moreover, Potash et al. implemented rhyme schemes by training their model on datasets containing rhyming word pairs.

In [LFM19], the authors introduce iComposer, a system designed to simplify music production by automatically generating melodies and lyrics based on text input. The system is built on a sequence-to-sequence model architecture, with a focus on predicting melody, rhythm, and lyrics. Specifically, the authors utilize LSTM (long short-term memory) networks in their model, which are well-suited for sequence-based tasks due to their ability to capture long-range dependencies in data.

By leveraging LSTM networks, iComposer effectively learns the structure of Chinese popular music and generates melodies and lyrics that align harmoniously. The experimental results demonstrate that iComposer can compose compelling songs at a level comparable to human composers, showcasing the effectiveness of LSTM in capturing the intricate relationships between lyrics and melody. Overall, the integration of LSTM in iComposer enables the system to create music that resonates with listeners, highlighting the potential of LSTM in enhancing automated music composition systems.

In [GLM20] , the paper "Deep Learning in Musical Lyric Generation" explores the application of neural network-based language models, specifically LSTM, in generating original song lyrics across different genres. The study focuses on key linguistic characteristics such as average line length, word variation, point-of-view analysis, and word repetition to capture the essence of various music genres. By training models on specific genres like jazz, country, metal, pop, rap, and rock, the researchers generated lyrics based on 16-word prompts, showcasing the ability of the models to mimic genre-specific lyrical styles.
Despite challenges in capturing certain metrics like point-of-view in rock lyrics, the models demonstrated a high degree of semantic relevance and stylistic accuracy. The research not only highlights the potential of deep learning in music composition but also opens avenues for personalized text generation tailored to specific audience preferences within distinct writing styles.

The following figure is from [GLM20] which shows the percentage changes from original to generated lyrics for each metric and genre. This figure provides a detailed comparison of how the generated lyrics differ from the original ones in terms of various linguistic metrics, offering insights into the model's performance and the extent to which it captures the nuances of different music genres.



Genre	*Average Line Length*	*Song Word Variation*	*Genre Word Variation*	*I vs. You*	*Word Repetition*
Metal	-54.3	-25.1	-13.5	82.7	88.2
Rock	-38.4	-5.0	-5.9	77.0	34.9
Rap	-52.3	-64.6	-99.0	75.5	-49.6
Pop	-28.0	-16.7	-44.6	94.6	36.5
Country	-74.1	-36.6	-23.0	-229.6	75.2
Jazz	-24.1	-38.7	-78.5	73.7	94.6

Figure 1: Percentage Changes from original to generated lyrics for each metric and genre

In [FS14] , the paper presents a novel approach for analyzing and classifying song lyrics. The authors experiment with n-gram models and more sophisticated features to model different dimensions of song texts, such as vocabulary, style, semantics, orientation towards the world, and song structure. They demonstrate the effectiveness of these features in three classification tasks: genre detection, distinguishing the best and worst songs, and determining the approximate publication time of a song. The study draws on earlier work on text classification, poetry analysis, and lyrics-based music classification.
The authors collected their own dataset of English song lyrics and used metadata such as genre information, quality ratings, and publication years for their experiments. They designed 13 feature classes grouped into five abstract sets, and conducted three experiments to classify songs by genre, quality, and publication time. The results show that lyrics-based statistical models can be employed to perform different music classification tasks, and the extended feature set consistently improves classification performance. The study provides insights into the potential benefits of lyrics-based music classification for music retrieval, recommendation systems, and musicology research.

In [GSB], the paper investigates the effectiveness of different RNN-based language modeling architectures for music lyrics classification and generation. The custom dataset contains 80k song lyrics of the genres Rock, Pop, Rap, and Country. The classification performance of Vanilla RNN, GRU, and various LSTM variations is evaluated, with the GRU achieving the best performance. For lyrics generation, LSTM consistently produced higher quality lyrics. The paper also explores the use of initial cell-state and hidden-state to generate lyrics in specific genres. The results show that LSTM and GRU architectures are well-suited for classification and generation tasks. Additionally, the paper discusses the challenges in genre classification and the potential for future work using transformer-based models.

In [SKAO22] the paper explores the application of Bidirectional Long Short-Term Memory (BLSTM) networks for sentiment classification of Covid-19 related online articles. The study evaluates how changing various parameter values impacts the performance of LSTM and BLSTM models. The dataset consists of over 10,000 articles sourced from prominent news websites, including CNN and Fox News, covering the period from May 2020 to September 2020. The research highlights the significance of parameter tuning in achieving optimal model performance, with experimental results demonstrating that the BLSTM model generally outperforms the unidirectional LSTM in terms of classification accuracy. The findings underscore the robustness of BLSTM models in handling textual data with complex linguistic structures and varying sentiment polarities, providing valuable insights for sentiment analysis tasks in NLP.

In [Kab21], the author proposes a novel sentiment analysis model combining convolutional neural networks (CNN) and bidirectional long short-term memory (BiLSTM) networks for analyzing Turkish tweets about COVID-19. The study emphasizes the inclusion of emojis and emoticons in sentiment analysis and employs a lexicon-based tweet annotation approach. The model, trained on 15,000 Turkish tweets, achieved an impressive accuracy of 97.895%, outperforming state-of-the-art baseline models.This paper's approach to combining CNN and BiLSTM architectures and its focus on handling non-textual elements like emojis provide valuable insights for our project. The high accuracy achieved by this hybrid model underscores the potential benefits of integrating different neural network architectures for complex text analysis tasks.

In the master's thesis by Fredrik Pettersson [Pet19], the focus is on optimizing deep neural networks for the classification of short texts. The research explores how state-of-the-art neural network models can be adapted for specific natural language processing (NLP) datasets and investigates the impact of different dimensionality reduction techniques on commonly used pre-trained word embeddings. The thesis employs convolutional neural networks (CNNs), long short-term memory (LSTM) networks, and bidirectional LSTMs (Bi-LSTMs) combined with various word embeddings like GoogleNews, GloVe, Paragram, and Wiki-news. The research involves three main experiments: creating a top-performing text classification model for a Kaggle competition, reducing the dimensionality of word embeddings using principal component analysis (PCA) and random projection (RP), and benchmarking the model on a secondary dataset (Sentiment140) to evaluate generalization. The findings show that PCA can reduce embedding dimensionality by 66% with minimal accuracy loss, improving execution time by 13%. The Bi-LSTM model achieved an F1 score of 71% in the competition and 86% accuracy on the Sentiment140 dataset, highlighting its generalizability and efficiency.

Finally , In [Mah20] , the author explores an advanced approach to text summarization using deep learning techniques. The study focuses on implementing a Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM) units, enhanced by an attention mechanism, to generate abstractive summaries of text documents. This method aims to produce summaries that not only capture the essential information but also maintain the semantic and grammatical integrity of the original text. The research involves training the model on a large dataset and evaluating its performance using the ROUGE metric, demonstrating the effectiveness of attention-based mechanisms in improving the quality of abstractive summaries. The findings indicate that incorporating attention significantly enhances the model's ability to understand and summarize long documents, making this approach a valuable contribution to the field of natural language processing.

3 Dataset

In this dataset manipulation process, we first loaded a CSV dataset file using the pandas library in Python. We attempted to read the dataset into a DataFrame, ensuring proper encoding and handling potential parsing errors. Subsequently, we displayed the first few rows of the dataset and retrieved information about its structure, including data types and missing values. After handling missing values and removing duplicate rows, we proceeded to tokenize the text data using the NLTK library, breaking down each artist's name into individual words. Next, we cleaned the tokenized text by converting words to lowercase and removing punctuation and stopwords. Finally, we saved the modified DataFrame to a new CSV file and provided a link for downloading the processed dataset. This comprehensive process allowed us to prepare the dataset for further analysis or modeling tasks.

3.1 Data Analysis

In the given dataset, the analysis focused on determining the most frequent word present in the 'artist' column. After processing the text data by joining the 'artist' column entries into a single string and then splitting them into individual words, a dictionary was created to store the frequency count of each word. Through iteration over the words, their frequencies were tallied in the dictionary. Subsequently, the word with the highest frequency was identified using the max() function, which returned the word 'John'. The frequency of occurrence of this word was found to be 968 times within the 'artist' column. This outcome suggests that 'John' appears most frequently among all the artist names recorded in the dataset. Such analysis provides valuable insights into the distribution of artist names and the prominence of specific terms within the dataset.

The analysis focused on determining the most frequent word present in the 'song' column. The text data from the 'song' column was combined into a single string, and then split into individual words. A dictionary was created to store the frequency count of each word. Iterating over the words, their frequencies were tallied in the dictionary. Subsequently, the word with the highest frequency was identified using the max() function. In this case, the word 'The' was found to be the most frequent, appearing 7933 times within the 'song' column. This outcome suggests that 'The' is the most commonly occurring word among all the song titles recorded in the dataset. Such analysis provides valuable insights into the distribution of song titles and the prominence of specific terms within the dataset.

In this analysis of the dataset, the objective was to identify the most frequent word in the 'text' column. Initially, the textual data from the 'text' column was concatenated into a single string. This string was then split into individual words. A dictionary was employed to track the frequency count of each word encountered. Through iteration over the words, their frequencies were recorded in the dictionary. Subsequently, utilizing the max() function, the word with the highest frequency was determined. In this instance, the word 'the' emerged as the most frequently occurring word in the 'text' column, with a frequency count of 446,872. This finding underscores the prevalence of the word 'the' within the textual content of the dataset, offering valuable insights into common linguistic patterns present in the dataset.
We aimed to identify the largest elements within each column of our dataset. By iterating through the columns, the code computed the maximum length of each element within a column and stored the largest element along with its size. Upon execution, the output revealed the largest elements along with their corresponding sizes for each column in the dataset. For instance, in the 'artist' column, the largest element identified was 'Joseph And The Amazing Technicolor Dreamcoat', with a size of 44 characters. Similarly, the 'song' column featured the largest element 'A Simple Desultory Philippic (Or How I Was Robert McNamara'd Into Submission)' with a size of 77 characters. Notably, the 'text' column exhibited the largest element, a lengthy lyrical excerpt, spanning 3997 characters. These findings underscore the considerable variability in element sizes across different columns, reflecting the diverse nature of the textual data contained within the dataset.

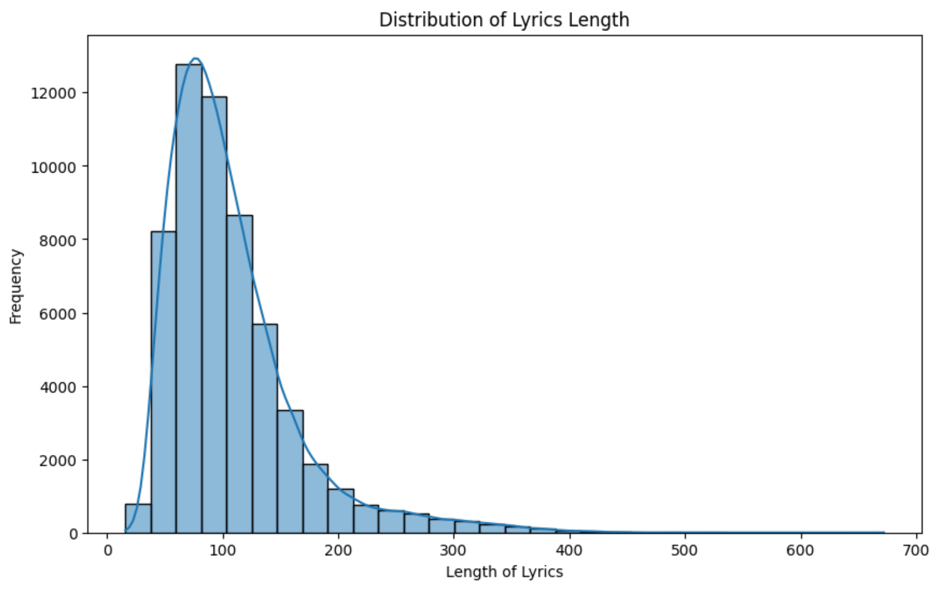


Figure 2: Distribution of Lyrics Length

3.1.1 Distribution of Lyrics Length

The histogram above shows the distribution of the length of lyrics in the dataset. The x-axis represents the length of the lyrics (number of words), and the y-axis represents the frequency (number of songs). We observe that the majority of the lyrics fall within the range of 50 to 150 words. This indicates that most songs in the dataset have relatively short lyrics, with a decreasing number of songs as the lyrics length increases. This distribution helps us understand the typical length of song lyrics and can guide the preprocessing steps in terms of padding and truncation during model training.

3.1.2 Top 10 Most Common Artists

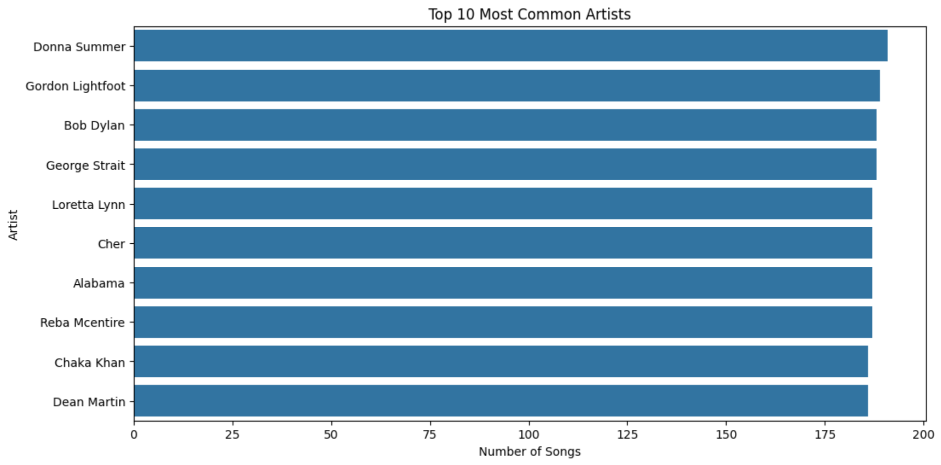


Figure 3: Top 10 Most Common Artists

The bar chart above shows the top 10 most common artists in the dataset, based on the number of songs attributed to each artist. Donna Summer, Gordon Lightfoot, and Bob Dylan are among the most frequently occurring artists, each with nearly 200 songs. This analysis provides insight into the distribution of songs across different artists and highlights the prevalence of certain artists in the dataset. Understanding which artists are most represented can help in tailoring genre-specific models or focusing on artist-specific lyric generation tasks.

3.1.3 Top 20 Words by TF-IDF Score

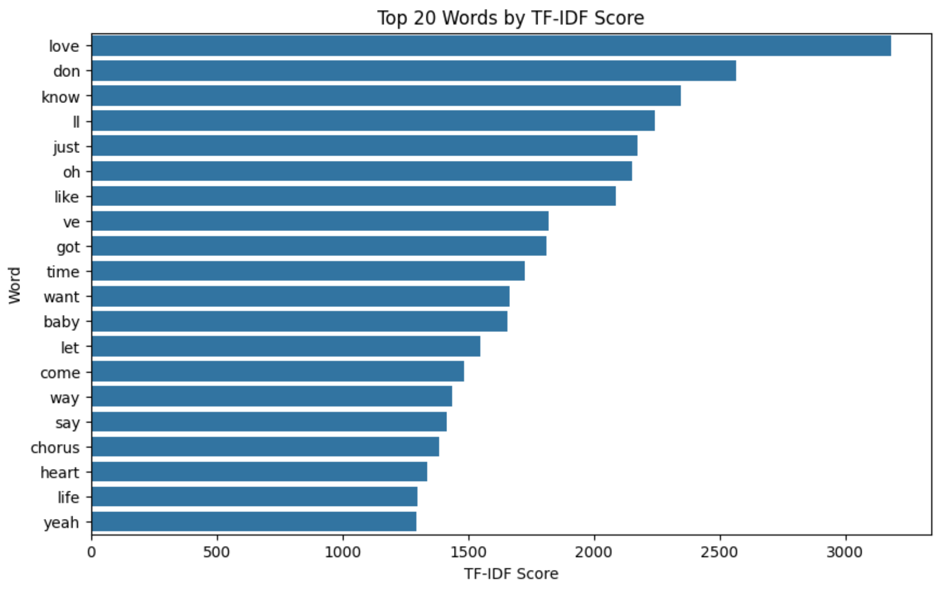


Figure 4: Top 20 Words by TF-IDF Score

The bar chart above illustrates the top 20 words in the lyrics dataset based on their TF-IDF scores. TF-IDF (Term Frequency-Inverse Document Frequency) is a metric used to evaluate the importance of a word in a document relative to a collection of documents (corpus). Words like ”love”, ”don”, ”know”, and ”ll” have the highest TF-IDF scores, indicating their significance in the lyrics. This analysis helps identify the most impactful words in the dataset, which can be crucial for understanding the thematic elements of the songs.

3.1.4 Top 20 Most Common Words

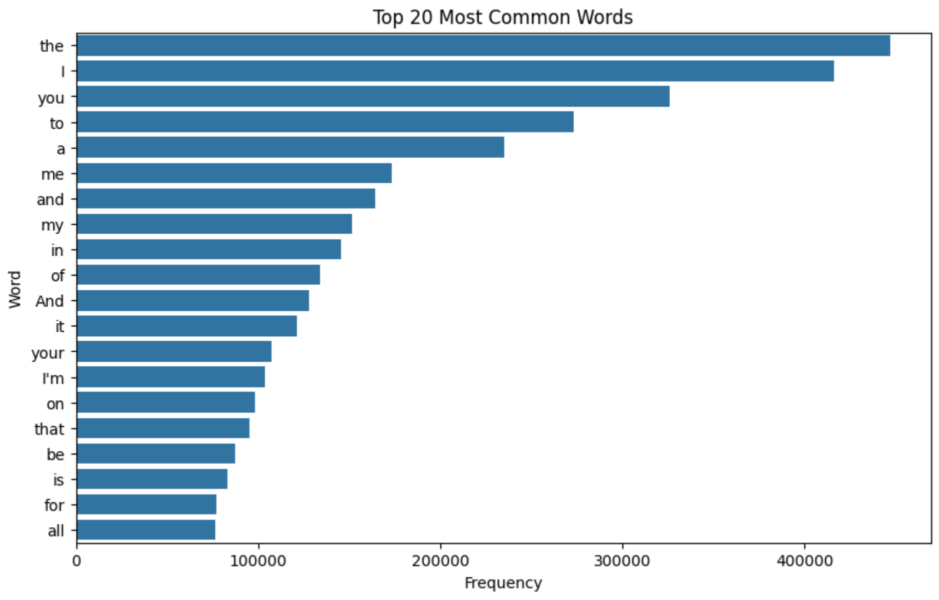


Figure 5: Top 20 Most Common Words

The bar chart above shows the top 20 most common words in the lyrics dataset. The x-axis represents the frequency of each word, while the y-axis lists the words. Common stopwords like "the", "I", "you", and "to" appear most frequently, which is typical in text datasets. Despite their high frequency, these words often carry less semantic weight in the context of the lyrics, as they are used for grammatical purposes. Understanding the frequency of common words helps in preprocessing steps such as stopword removal for better model performance.

3.1.5 Distribution of Words per Document

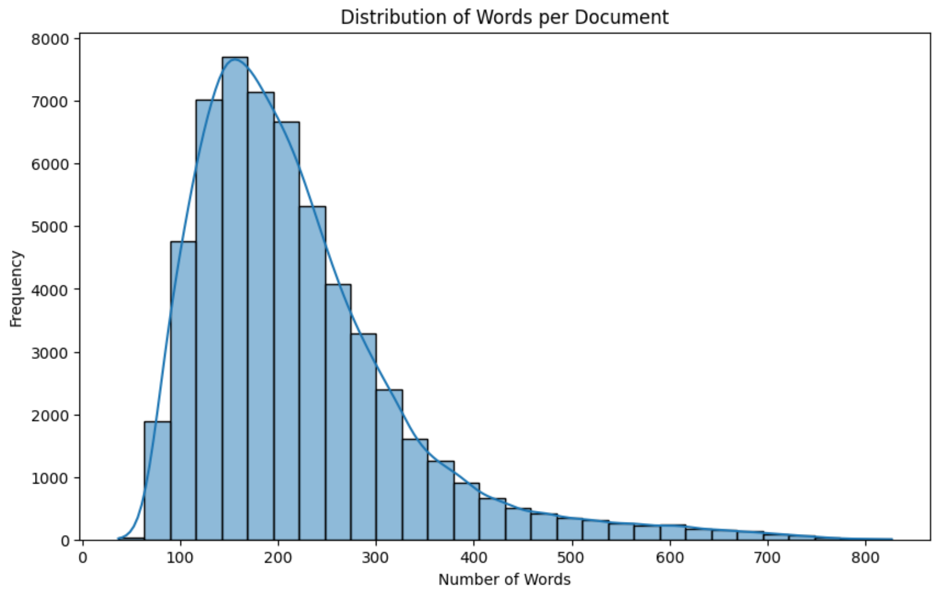


Figure 6: Distribution of Words per Document

The histogram above displays the distribution of the number of words per document (song) in the dataset. The x-axis represents the number of words, and the y-axis represents the frequency of documents with that word count. Most documents have between 100 to 300 words, with a peak around 200 words. This distribution helps in setting the sequence length for models, ensuring that the majority of documents are appropriately represented without excessive padding or truncation.

3.2 Dataset Limitations

In spite of the richness of the dataset, several limitations warrant consideration. One notable limitation is the absence of null values or duplicate entries within the dataset. While the absence of null values and duplicates ensures data integrity and reliability, it also contributes to the dataset's substantial size. As a consequence, managing and processing such a voluminous dataset may pose challenges in terms of computational resources and time efficiency.

We obtained an empty dataframe after performing correlation analysis, it typically means that there is no linear relationship between the variables in the dataset.

It's essential to acknowledge another limitation observed within the dataset: the presence of identical song names attributed to different artists. This phenomenon introduces ambiguity and potential confusion, as song titles may lack the specificity required to uniquely identify a musical composition.

4 Methodology

4.1 Milestone 2: Building a Neural Network Model

In this milestone, our objective was to design and implement a neural network model entirely from scratch to generate song lyrics using a specific dataset. This section provides a comprehensive overview of the methodology we employed, delves into the intricate architecture of the neural network, explains the training regimen, and presents the evaluation outcomes. Our focus was to understand the fundamental aspects of neural network design and how they can be applied to natural language processing tasks without relying on pre-trained models.

The methodology adopted for this project encompassed several critical steps, each essential to ensure the robustness and efficacy of the neural network model:

4.1.1 Data Loading and Preprocessing

- We started by importing the dataset, which comprised song lyrics from various artists. Using the pandas library, we read the dataset into a DataFrame for easy manipulation and analysis. This dataset was analyzed earlier, showing a distribution of lyrics lengths and identifying the most common artists.
- To maintain consistency and focus, we identified the artist with the highest number of songs in the dataset, as shown in our previous analysis where Donna Summer was the most frequent artist. This step involved using the `value_counts` method to count the number of songs per artist and then filtering the dataset to include only the songs of the most prolific artist. This filtering process ensured that the training data was homogeneous in terms of style and language.
- Tokenization was performed using the `Tokenizer` class from TensorFlow Keras, which converts the text data into sequences of integers. Each word in the lyrics was assigned a unique integer, allowing the model to process the text in a numerical format. The earlier analysis of the most common words and their TF-IDF scores provided insights into the vocabulary used in the lyrics.
- We created input sequences and their corresponding target words. For each lyric, we generated n-gram sequences where the input was a sequence of words and the target was the next word in the sequence. This setup is crucial for training the model to predict subsequent words based on preceding context.
- Padding was applied to ensure that all input sequences had the same length. This involved adding zeros to the beginning of shorter sequences, making them equal in length to the longest sequence in the dataset. Consistent input lengths are necessary for the neural network to process the data effectively. The distribution of words per document, analyzed earlier, helped determine appropriate sequence lengths.

4.1.2 Model Building

- The architecture of the model was designed using the Sequential API from TensorFlow Keras. This straightforward approach allowed us to stack layers in a linear fashion, defining the flow of data through the network.
- The first layer was an Embedding layer, which converts the integer-encoded words into dense vectors of fixed size. This layer is essential for capturing the semantic meaning of words and reducing the dimensionality of the input space. The earlier analysis of TF-IDF scores helped in understanding the importance of embedding layers to capture word meanings effectively.
- Two Long Short-Term Memory (LSTM) layers were added next. LSTMs are a type of recurrent neural network (RNN) capable of learning long-term dependencies in sequential data. The first LSTM layer was configured to return sequences, allowing the subsequent LSTM layer to process the entire sequence of hidden states.

- A Dropout layer was incorporated between the LSTM layers to mitigate overfitting. Dropout works by randomly setting a fraction of the input units to zero during training, which helps the network generalize better to unseen data.
- The final layer was a Dense output layer with a softmax activation function. This layer provided a probability distribution over the vocabulary, enabling the model to predict the likelihood of each word being the next word in the sequence.

4.1.3 Training Process

- **Data Splitting:** The dataset was divided into training and testing sets to allow for model validation and evaluation. The training set comprised 90% of the data, while the testing set comprised 10%. This split ensured that the model could be evaluated on a separate dataset, providing a realistic measure of its generalization performance.
- **Callbacks:**
  - **Early Stopping:** Early stopping monitored the validation loss and halted training if the loss did not improve for a specified number of epochs (patience). This helped prevent overfitting by stopping training when the model was no longer improving.
  - **Model Checkpointing:** Model checkpointing saved the best model based on validation loss during training. This ensured that the best-performing model was retained, even if subsequent epochs resulted in worse performance.
  - **Learning Rate Reduction:** Learning rate reduction dynamically adjusted the learning rate when the validation loss plateaued. Lowering the learning rate helped fine-tune the model and led to better convergence.
- **Training:** The model was trained for five epochs with a batch size of 128. During each epoch, the model learned to predict the next word in a sequence by minimizing the prediction error. The validation set was used to monitor the model's performance and apply the configured callbacks. The use of early stopping and model checkpointing ensured that the best model was saved, and the training process was halted once the model's performance no longer improved.

4.1.4 Model Evaluation and Text Generation

After training the model, we implemented a function to generate text based on a given seed text. This function demonstrates the model's practical application in generating song lyrics by predicting the next word in a sequence. The function operates as follows:

- **Initialization:** The function starts with a seed text, which serves as the initial context for generating new words. This seed text is crucial as it provides the initial context for the model to begin making predictions.
- **Tokenization and Padding:** The seed text is first tokenized into a sequence of integers using the trained tokenizer. These integers correspond to the words in the tokenizer's vocabulary. Since neural networks require fixed-length input, the tokenized sequence is padded to ensure it matches the maximum sequence length expected by the model. Padding involves adding zeros to the beginning of the sequence if it is shorter than the maximum length.
- **Prediction:** The padded sequence is then fed into the trained model, which generates a probability distribution over the entire vocabulary. This distribution indicates the likelihood of each word in the vocabulary being the next word in the sequence. The model uses its learned patterns to make this prediction.
- **Selecting the Next Word:** The word with the highest probability is selected as the predicted next word. This is done by identifying the index with the highest probability value from the model's output. The tokenizer's index-to-word mapping is used to convert this index back into the corresponding word.

- **Updating the Text:** The predicted word is appended to the current text, extending the sequence. The function then repeats the process: tokenizing the updated text, padding it, and predicting the next word. This iterative process continues until the desired number of new words has been generated.
- **Handling Invalid Predictions:** If the predicted word is invalid (e.g., it is `None` or an empty string), the function skips it and continues with the next iteration. This ensures that only meaningful words are added to the generated text.
- **Final Output:** Once the specified number of words has been added, the function returns the complete text, which now includes the seed text followed by the newly generated words. This method effectively demonstrates the model's ability to generate coherent and contextually appropriate lyrics by leveraging its learned knowledge.

After completing the training process, the model was evaluated using a test dataset to measure its performance. The evaluation metrics obtained were:

- **Test Loss:** 5.2321
- **Test Accuracy:** 0.1230

Interpretation of Test Loss and Accuracy

The test loss is a measure of how well the model's predictions match the true labels on the test dataset. A lower loss indicates a better fit of the model to the data. In this case, the test loss is 5.2321, which is relatively high. This suggests that the model has not achieved a good fit and that there may be significant errors in its predictions. Several factors could contribute to this high test loss:

- Model Complexity: The model might be too complex for the given data, leading to overfitting during training but poor generalization to unseen data.
- Data Quality: The quality and diversity of the training data play a crucial role. If the training data does not adequately represent the test data, the model may struggle to perform well.
- Training Duration: The model might require more epochs to learn from the data effectively. Conversely, it might also indicate that the model has been overfitted, necessitating early stopping or regularization techniques.

Test accuracy measures the proportion of correct predictions made by the model on the test dataset. An accuracy of 0.1230 means that the model correctly predicts only 12.30% of the time, which is significantly below the expected performance for a robust model.Several reasons could explain this low accuracy:

- Imbalanced Dataset: If the dataset is imbalanced (e.g., more negative samples than positive), the model might struggle to learn the minority class, leading to poor performance.
- Inadequate Training: The model might not have been trained for a sufficient number of epochs, or the learning rate might be too high or too low, preventing the model from converging to an optimal solution.
- Model Architecture: The chosen model architecture might not be suitable for the complexity of the task. Adjustments to the architecture, such as adding more layers, changing activation functions, or using different types of layers (e.g., combining CNN with LSTM), could improve performance.

The implemented text generation function highlights the model's practical application in generating song lyrics. By predicting and appending words iteratively, the function showcases the model's capability to create text that maintains the style and structure of the training data. This function underscores the potential of neural networks in natural language processing tasks, particularly in creative text generation.

This comprehensive approach, integrating data analysis and preprocessing, ensured that the model was trained on a well-structured and representative subset of the dataset. The architecture and training strategies were chosen based on the nature of the data, aiming to optimize the model's ability to generate coherent and contextually appropriate song lyrics.

4.2 Milestone 3: Fine-Tuning a Pre-Trained Model

In this milestone, we aim to enhance our song lyrics generation system by fine-tuning a pre-trained large language model (LLM). Leveraging the powerful capabilities of this state-of-the-art model, which has been extensively trained on vast amounts of text data, we will adapt it to our specific task of generating lyrics by artist name and genre. This process involves refining the model's knowledge using a dataset tailored to our needs, enabling it to produce more relevant and contextually accurate lyrics. By fine-tuning the pre-trained LLM, we expect to achieve significant improvements in the quality and coherence of the generated lyrics, ultimately enhancing the overall performance and applicability of our lyrics generation system.

4.2.1 Selection of the Pre-trained Model

For this milestone, we chose GPT-2 as our pre-trained large language model (LLM) to fine-tune for generating song lyrics. GPT-2, developed by OpenAI, is renowned for its remarkable ability to generate coherent and contextually relevant text, making it an ideal candidate for our project.

Some additional points that makes GPT-2 an ideal model for our task are the following:

1. **State-of-the-Art Performance:** GPT-2 has demonstrated outstanding performance across a variety of natural language processing tasks. Its capacity to generate human-like text with high fluency and contextual accuracy makes it well-suited for creative tasks like lyrics generation.

2. **Robust Pre-Training:** GPT-2 has been pre-trained on a diverse and extensive dataset comprising billions of words from the internet. This broad exposure equips the model with a rich understanding of language patterns, idioms, and nuances, which is essential for generating diverse and stylistically varied song lyrics.

3. **Ease of Fine-Tuning:** The architecture of GPT-2, based on the Transformer model, is designed to be fine-tuned easily with specific datasets. This adaptability allows us to tailor the model's capabilities to focus on generating lyrics that are relevant to specific artists and genres, thereby improving the specificity and relevance of the generated content.

4.2.2 GPT-2 Architecture

The architecture of GPT-2 is based on the Transformer model, which has become a cornerstone in natural language processing due to its efficiency and scalability. We will break down the key components of GPT-2's architecture and how they contribute to its impressive capabilities. At a high level, GPT-2 consists of the following main components:

- Embedding Layer
- Transformer Blocks
- Self-Attention Mechanism
- Layer Normalization
- Feed-Forward Networks
- Activation Function
- Dropout Layer

- Output Layer

Embedding Layer

The embedding layer is the first step in the model. It converts input tokens (words) into dense vector representations. There are two types of embeddings:

- **Token Embeddings**: Map each word or token in the input to a high-dimensional vector.
- **Position Embeddings**: Add information about the position of each token in the sequence, which is crucial for understanding the order of words.

Transformer Blocks

GPT-2 is composed of multiple Transformer blocks, each containing several sub-layers. The blocks are identical and stacked on top of each other, allowing the model to capture complex patterns in the data.

Self-Attention Mechanism

At the heart of each Transformer block is the self-attention mechanism. This allows the model to weigh the importance of different words in a sequence relative to each other, regardless of their distance apart. Here's a simplified explanation of how it works:

- **Query, Key, Value Vectors**: Each token in the sequence is transformed into three vectors: query, key, and value.
- **Attention Scores**: The query vectors are matched against the key vectors to calculate attention scores, which determine how much focus to give to each word.
- **Weighted Sum**: These scores are used to compute a weighted sum of the value vectors, which represents the context of each word considering all other words in the sequence.

Layer Normalization

Each sub-layer (attention and feed-forward) is followed by layer normalization. This normalizes the inputs to each layer, stabilizing and speeding up training by preventing drastic changes in the distribution of layer inputs.

Feed-Forward Networks

After the attention mechanism, each Transformer block contains a feed-forward neural network. This consists of two linear transformations with a ReLU (Rectified Linear Unit) activation in between. It further processes the attended information and allows the model to capture complex features.

Activation Function

GPT-2 uses the Gaussian Error Linear Unit (GELU) activation function in its feed-forward networks. GELU is known for its performance in deep learning models due to its ability to combine the properties of both ReLU and dropout, providing a smooth and non-linear transformation that helps the model learn more effectively.

Dropout Layer

The dropout layer is used as a regularization technique to prevent overfitting. Dropout randomly sets a fraction of the input units to zero during training, which helps to ensure that the model generalizes well to unseen data. GPT-2 employs dropout layers after the attention and feed-forward sub-layers within each Transformer block.

Output Layer

The final output of the Transformer blocks is fed into a linear layer (output layer), which maps the processed information to the vocabulary size. This layer predicts the next word in the sequence by providing a probability distribution over all possible words.

4.2.3 Data Preparation

In this section, we discuss the steps taken to prepare the data for fine-tuning our GPT-2 model for lyrics generation. Data preparation is a crucial step to ensure that the model learns effectively from the input text and performs well on the task.

Data Cleaning

The first step in our data preparation was cleaning the dataset. This involved several sub-steps to ensure the text data was in a consistent and usable format. Specifically, we:

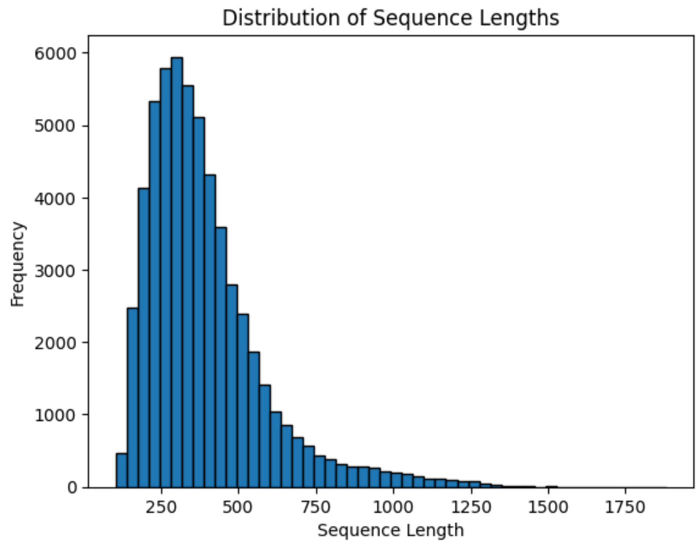
1. Removed any punctuation marks from the text, which helps in standardizing the input.
2. Converted all characters to lowercase helps in reducing the complexity by treating 'The' and 'the' as the same word.
3. Eiminated any extra spaces to avoid inconsistencies in the data.

Combining Artist Name and Lyrics

To provide the model with better context, we combined the lyrics with the artist's name. This helps the model understand the style associated with each artist. We added the artist's name and the lyrics in a single column called input_text, separated by a colon ":" .

Sequence Length Analysis

To determine the appropriate max_length value for tokenization, we conducted a sequence length analysis. This involved tokenizing the text without truncation and analyzing the distribution of sequence lengths.The following figure visualizes the distribution of sequence lengths and their frequency.



Based on the analysis, we chose max_length = 450 to ensure that most sequences are appropriately captured without truncating too much relevant information.

Tokenization and Padding

Next, we tokenized the combined text using the GPT-2 tokenizer. Tokenization is the process of converting text into numerical tokens that the model can understand. We also added an end-of-text padding to mark the end of each sequence.

Filtering Empty Sequences

After tokenization, we filtered out any empty sequences to ensure that the data fed into the model was meaningful.

Data Splitting

We split the dataset into training and test sets with a ratio of 90% for training and 10% for testing. This split ensures that we have enough data for the model to learn from while keeping some data aside to evaluate its performance on unseen text.

Data Collation

We used the "DataCollatorForLanguageModeling" from Hugging Face to create batches of data for training. This collator dynamically pads sequences to the maximum length within the batch, which helps in efficient training.

4.2.4 Training

In this section, we describe the training process for fine-tuning our GPT-2 model for lyrics generation. The training process involves configuring various parameters and arguments to ensure efficient learning, monitoring progress, and saving the model at appropriate intervals.

To manage the training process, we used the TrainingArguments class from the Hugging Face Transformers library. Here are the key attributes we configured and the rationale behind each choice:

1. Number of epochs: 6
2. "per_device_train_batch_size" : 8, which specifies the batch size for training on each device (GPU). This batch size was chosen based on the available GPU memory to ensure efficient training without running into memory issues.
3. "gradient_accumulation_steps" : 4, effectively increasing the batch size to 32 . This technique allows for more stable updates by accumulating gradients over multiple steps before performing a backward pass.
4. "weight_decay": 0.01 to prevent overfitting by penalizing large weights. This regularization technique helps in maintaining the generalizability of the model.
5. "save_steps": 10,000 to ensure that we have intermediate checkpoints,meaning the model's state is saved every 10,000 steps.
6. "logging_steps" : 500, logging training information every 500 steps to monitor progress and debug issues.
7. "fp16" : True , which speeds up training and reduces memory usage by using 16-bit floating point numbers.
8. Optimizer: "adamw_torch" ,This optimizer combines the benefits of Adam and L2 regularization, making it effective for training large models.

9. ”report_to” : wandb , for reporting and monitoring, This integration provides an interactive interface to track progress, visualize metrics, and make informed decisions during training.

Callbacks

We employed the Early Stopping Callback with ”early_stopping_patience” set to 1. This callback stops the training early if there is no improvement in the validation loss for one consecutive epoch. Early stopping helps in preventing overfitting and saves computational resources by terminating training when further improvement is unlikely.

Function to Compute Metrics

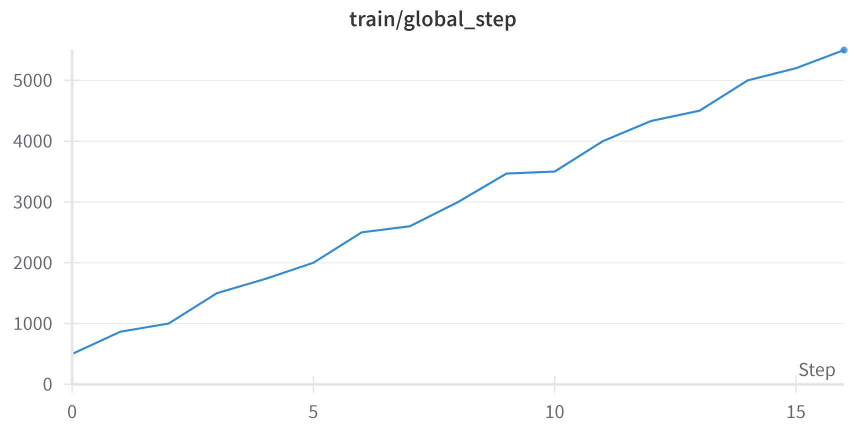
We defined a custom function to compute metrics during training. This function calculates accuracy and perplexity, providing a more comprehensive view of the model’s performance. The function computes the predictions and labels, calculates the perplexity, and measures the accuracy by comparing the predicted labels with the actual labels.

4.2.5 Results and Evaluation

In this section, we present and discuss the training and evaluation results of our fine-tuned GPT-2 model for lyrics generation. The following graphs illustrate the key metrics and their progression throughout the training process.

Training Progress

The graph below above shows the global step progression during training. It indicates the number of training steps completed over time. The consistent increase demonstrates that the training process proceeded smoothly without interruptions, leading to effective learning by the model.

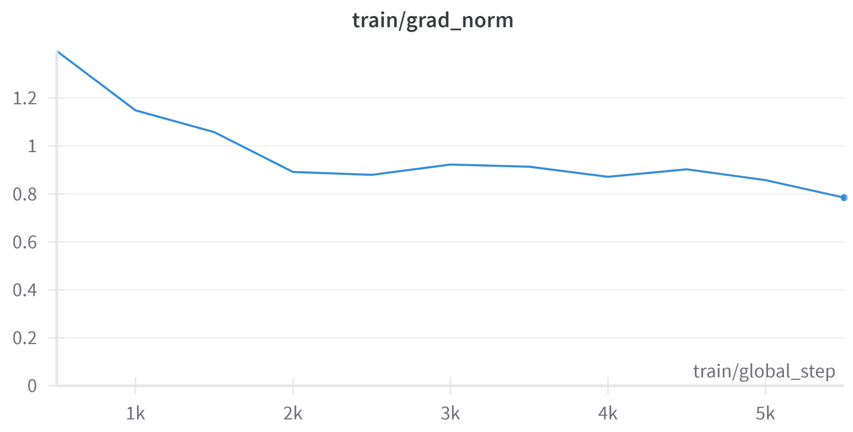


Gradient Norm

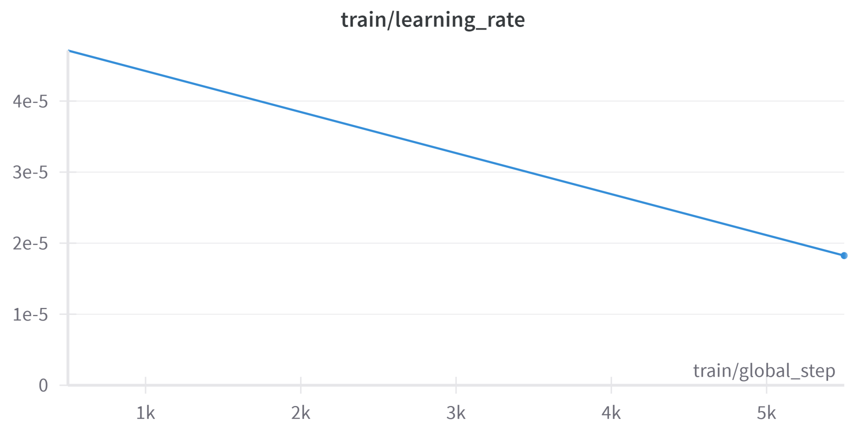
This graph below illustrates the gradient norm over the course of training. The gradient norm is a measure of the magnitude of gradients during backpropagation. A decreasing trend in the gradient norm is a good sign, indicating that the model’s parameters are stabilizing and the learning process is becoming more refined as training progresses.

Learning Rate

The graph below shows the learning rate schedule used during training. The learning rate decreases linearly from the initial value to ensure that the model’s updates become more precise over time. This



helps in fine-tuning the model effectively, avoiding large updates that could destabilize the training process.



Training Loss

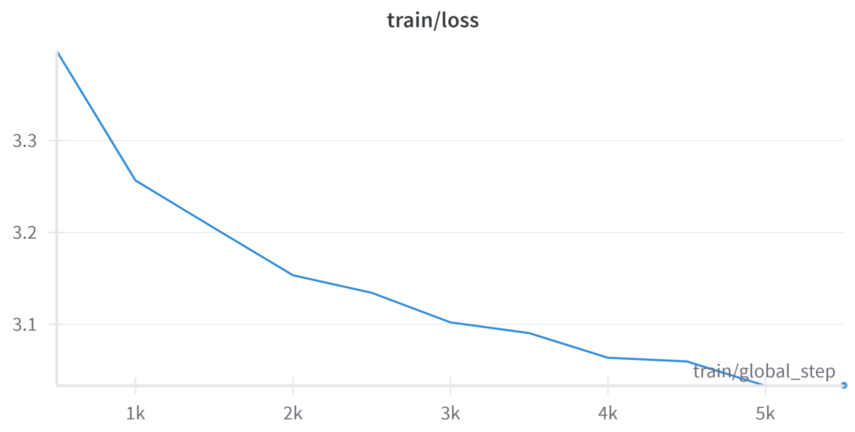
The graph below shows the training loss over the global steps. Training loss represents the model's performance on the training dataset. A downward trend in the training loss indicates that the model is learning and improving its ability to generate relevant lyrics. The final training loss achieved was 3.03.

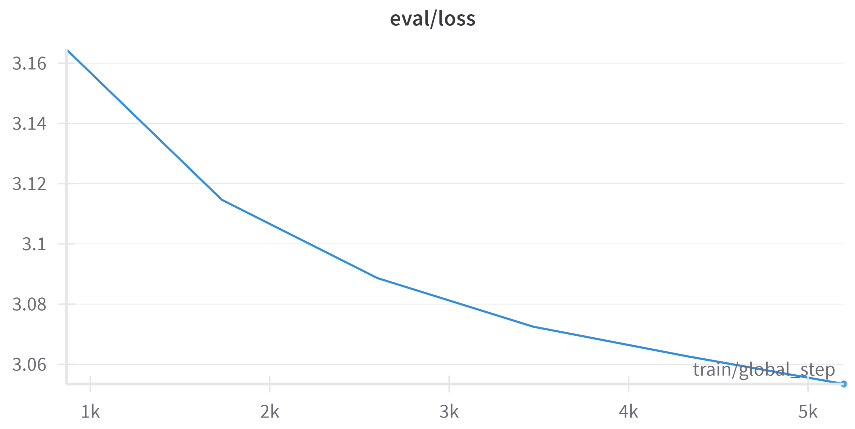
Evaluation Loss

The graph above shows the evaluation loss over the global steps. Evaluation loss is calculated on the validation dataset and is a good indicator of the model's generalization performance. A downward trend in the evaluation loss suggests that the model is not only fitting the training data but also generalizing well to unseen data. The final validation loss achieved was 3.05.

Finally , we can deduce the following:

- Final training loss: 3.03
- Final validation loss : 3.05





4.2.6 Post-processing and Lyrics Generation Function

In this section, we discuss the implementation of the function used to generate lyrics with our fine-tuned GPT-2 model. The function is designed to take an initial prompt, such as the artist's name followed by some lyrics, and generate a continuation based on the model's learned patterns. Let's break down each step and explain the rationale behind the choices made.

Input Preparation

The function begins by encoding the input prompt using the GPT-2 tokenizer. The tokenizer converts the text into a sequence of numerical tokens that the model can process. This is essential because the model operates on numerical data, not raw text.

Attention Mask

Next, we create an attention mask, which is a tensor of ones with the same shape as the inputs. The attention mask tells the model which tokens it should focus on during the generation process. Since we are providing a complete sequence of tokens without padding, the attention mask is simply a tensor of ones.

Model Generation Parameters

- max_length: Specifies the maximum number of tokens in the generated sequence. This controls how long the output text will be.

- num_return_sequences: Determines how many different sequences the model should generate. In this case, we set it to 1 to generate a single sequence.
- no_repeat_ngram_size: Ensures that no n-grams (sequences of n tokens) are repeated within the generated text. Setting this to 2 helps avoid repetitive and redundant text, enhancing the diversity and coherence of the output.
- top_p (nucleus sampling): This parameter controls the sampling strategy by keeping only the top p fraction of probability mass. For instance, top_p=0.95 means the model will consider the top 95% of the most likely next tokens, ensuring that the generated text is both coherent and diverse.
- temperature: Adjusts the randomness of the predictions by scaling the logits before applying softmax. A lower temperature (closer to 0) makes the output more deterministic, while a higher temperature (closer to 1) makes it more random. We use temperature=0.7 to balance creativity and coherence in the generated lyrics.
- pad_token_id: Specifies the token used for padding sequences. We set this to the end-of-sequence token to ensure proper termination of the text.

Post-Processing the Output

Once the model generates the output tokens, these tokens need to be converted back into human-readable text. This is done using the tokenizer's decode method, which translates the numerical tokens back into a string. We use the skip_special_tokens=True parameter to remove any special tokens that were added during the encoding process.

Post-processing of the generated text involves ensuring that the output is clean and readable. This includes removing any extraneous whitespace, fixing punctuation, and ensuring that the text flows naturally. Since our function skips special tokens during decoding, the main post-processing step involves reviewing the output for any formatting issues and making minor adjustments as needed.

4.2.7 Discussion

Advantages

The fine-tuned GPT-2 model for lyrics generation has demonstrated several notable advantages. Firstly, the model leverages the powerful Transformer architecture, which excels at capturing complex patterns in text data. This architecture enables the model to understand and generate coherent, contextually relevant lyrics. The use of self-attention mechanisms allows GPT-2 to weigh the importance of different words in a sequence, making it particularly effective for generating text that maintains a logical flow and adheres to the style of the input prompt.

Moreover, the model's ability to handle large amounts of data and learn intricate language patterns is a significant advantage. By training on extensive datasets, the model can generate lyrics that are not only contextually appropriate but also stylistically consistent with the given artist. This results in a more authentic and engaging output that closely mimics human creativity.

Complexity

The complexity of the GPT-2 model is one of its defining characteristics. With millions of parameters and multiple layers of Transformer blocks, the model can capture and generate highly detailed and nuanced text. Each layer in the Transformer model performs complex operations, including self-attention and feed-forward neural networks, which enable the model to learn intricate dependencies within the data. This depth of learning allows GPT-2 to generate text that is remarkably coherent and contextually rich.

However, this complexity also presents challenges. The computational resources required to train and fine-tune such a model are substantial. Training GPT-2 efficiently necessitates the use of high-performance GPUs, such as the T4 or V100, which are not always readily accessible. Additionally, the large number of parameters increases the demand for memory and processing power, making the training process resource-intensive and time-consuming.

Interpretability

Despite its complexity, GPT-2 offers a certain level of interpretability that can be valuable for understanding and refining its outputs. The self-attention mechanism, a core component of the Transformer architecture, provides insights into how the model makes its predictions. By examining the attention weights, we can identify which words or phrases the model focuses on when generating text. This can help us understand why the model produces certain outputs and how it captures the context of the input prompt.

For example, in the context of lyrics generation, attention weights can reveal how the model associates certain lyrical themes or phrases with specific artists. This interpretability can be particularly useful for debugging and improving the model, as it allows us to see which parts of the input have the most influence on the generated text. By adjusting the model's parameters or training data based on these insights, we can enhance its performance and ensure that the generated lyrics better reflect the desired style and content.

Key Findings

- The GPT-2 model effectively generated coherent and contextually relevant lyrics by leveraging the powerful Transformer architecture and its self-attention mechanism, which maintained the logical flow and style of the input prompt. Despite its complexity, the model's interpretability through attention weights provided valuable insights into its prediction process, aiding in refining and improving the outputs.
- The training process was resource-intensive, requiring high-performance GPUs like the T4 to manage the model's computational demands. We trained the model for six epochs, balancing adequate learning and overfitting, despite initially planning for ten epochs due to Google Colab's T4 GPU runtime limitations. The training configuration included a batch size of 8, gradient accumulation steps, weight decay, and an AdamW optimizer, all contributing to the model's effective learning process.
- The training and evaluation losses showed a consistent downward trend, indicating the model's improving ability to generate relevant lyrics, with final training and validation losses of 3.03 and 3.05, respectively. Metrics such as accuracy and perplexity, computed using Weights & Biases (wandb), provided a comprehensive view of the model's performance.
- The generation function implemented parameters such as n-gram repeat prevention, nucleus sampling (top-p), and temperature scaling to ensure output quality. It successfully generated lyrics that were contextually appropriate and stylistically consistent with the provided prompts, despite not producing the exact lyrics.

Limitations

While the model exhibits many strengths, it also has some limitations. One notable issue is that the generated lyrics, although coherent and contextually appropriate, are not the correct lyrics. The words generated make sense and fit the context provided by the input prompt, but they may not align with the actual lyrics of the song. This discrepancy could be due to several factors, including the model's training data and the number of training epochs.

Training on more epochs might help the model learn more intricate patterns and produce outputs that are closer to the actual lyrics. However, due to the runtime limits of the T4 GPU processor on

Google Colab, we were only able to run the training for six epochs instead of the intended ten. This limitation affected the model's ability to fully converge and achieve optimal performance. Unfortunately, once the T4 GPU runtime limit was reached, we could not reconnect to the T4 GPU due to Colab's usage restrictions.

Another limitation is the computational resource requirement. GPT-2's complexity necessitates powerful hardware for efficient training. The reliance on high-performance GPUs like the T4 means that training on standard CPUs is not feasible within a reasonable timeframe. This constraint limits the ability to iterate on the model and experiment with different configurations or additional epochs.

5 Comparison Between Milestone 2 and Milestone 3

In this section, we discuss the key differences between the approaches and outcomes of Milestone 2 and Milestone 3, focusing on model architectures and data preprocessing and postprocessing steps.

5.1 Model Architectures

5.1.1 Milestone 2: Custom Neural Network

In Milestone 2, we built a neural network model from scratch. The architecture consisted of an Embedding layer followed by two LSTM (Long Short-Term Memory) layers. The LSTM layers were crucial for capturing temporal dependencies in the lyrics, allowing the model to generate coherent sequences of text. A Dropout layer was included between the LSTM layers to prevent overfitting by randomly setting a fraction of the input units to zero during training. Finally, a Dense output layer with a softmax activation function was used to predict the next word in the sequence based on the learned patterns.

5.1.2 Milestone 3: Fine-Tuning GPT-2

In Milestone 3, we leveraged a pre-trained GPT-2 model, which is based on the Transformer architecture. GPT-2 is composed of multiple Transformer blocks, each containing self-attention mechanisms and feed-forward neural networks. This architecture enables GPT-2 to capture complex patterns and long-range dependencies in text data more effectively than LSTM-based models. The self-attention mechanism allows the model to weigh the importance of different words in a sequence, enhancing its ability to generate contextually relevant and coherent lyrics. Additionally, the use of Layer Normalization and the GELU activation function in GPT-2 contributes to the model's stability and performance.

5.2 Data Preprocessing and Postprocessing

5.2.1 Milestone 2: Custom Data Preprocessing

In Milestone 2, the data preprocessing involved tokenizing the lyrics, creating input sequences, and padding these sequences to ensure consistent input lengths. The lyrics were filtered to include only those by the most frequent artist to maintain a homogeneous training dataset. This approach required significant manual intervention to prepare the data for training the custom neural network.

5.2.2 Milestone 3: Advanced Data Preparation for GPT-2

Milestone 3 involved a more sophisticated data preparation process tailored to fine-tuning GPT-2. The steps included:

- **Data Cleaning**: Removing punctuation, converting text to lowercase, and eliminating extra spaces to standardize the input.
- **Combining Artist and Lyrics**: Merging the artist's name and lyrics into a single text field to provide context to the model.

- **Tokenization and Padding**: Using the GPT-2 tokenizer to convert text into tokens and adding end-of-text padding.
- **Sequence Length Analysis**: Conducting a sequence length analysis to determine an appropriate maximum length, ensuring most sequences are captured without excessive truncation.
- **Filtering Empty Sequences**: Removing sequences that resulted in no tokens after preprocessing.

5.2.3 Postprocessing

In Milestone 2, postprocessing primarily involved generating text by predicting the next word and appending it to the input sequence iteratively. The generated text was directly produced without additional refinement.

In Milestone 3, the postprocessing steps included using advanced generation parameters such as attention masks, no-repeat n-gram constraints, nucleus sampling (top-p), and temperature adjustments to balance creativity and coherence. The generated output was decoded and cleaned to ensure readability and adherence to the desired format.

Overall, the transition from a custom neural network in Milestone 2 to a fine-tuned GPT-2 model in Milestone 3 significantly enhanced the model's ability to generate coherent and contextually relevant lyrics. The sophisticated architecture and comprehensive data preparation methods used in Milestone 3 contributed to improved performance and more accurate generation capabilities.

6 Conclusion

In this project, we embarked on an exploration of genre-based song lyrics generation using advanced neural network architectures. Our journey through the various milestones provided valuable insights into the complexities and nuances of natural language processing in the context of musical lyrics.

6.1 Milestone 1: Data Analysis and Preparation

We began by meticulously analyzing the dataset, which included identifying the most common words, artists, and the distribution of lyrics length. This preliminary analysis was crucial in understanding the structure and characteristics of the data. We implemented various preprocessing techniques such as tokenization, padding, and the inclusion of artist information alongside lyrics, which significantly enhanced the quality of the input data for subsequent modeling tasks.

6.2 Milestone 2: Building a Neural Network Model from Scratch

In Milestone 2, we designed and implemented a neural network model from scratch to generate song lyrics. The architecture included embedding layers, LSTM layers, and dropout layers, with a final dense layer using a softmax activation function. Despite the challenges, the model demonstrated the ability to predict the next word in a sequence, providing a foundational understanding of neural network-based text generation. The training and evaluation results highlighted areas for improvement, particularly in terms of data diversity and model complexity.

6.3 Milestone 3: Fine-Tuning a Pre-Trained Model

In this milestone, we significantly enhanced our song lyrics generation system by fine-tuning a pre-trained GPT-2 model. We began by meticulously preparing our dataset, which involved cleaning the text data, merging the artist's name with the lyrics for better context, and performing sequence length analysis to determine optimal tokenization parameters. The training process was carefully managed using advanced training arguments and monitoring tools, including early stopping and gradient accumulation, to ensure efficient and effective learning. We leveraged the powerful architecture of GPT-2, which utilizes self-attention mechanisms and Transformer blocks, to generate coherent and contextually accurate lyrics. Post-training, we implemented a text generation function with fine-tuned parameters to produce high-quality lyrics based on input prompts.

6.4 Future Directions

Looking ahead, several exciting avenues can enhance our lyrics generation model. Increasing training epochs and securing more powerful computational resources will improve model accuracy and contextual relevance. Moreover, Expanding the dataset to include more artists and to include genres will enable the model to generate diverse and varied lyrics.

Integrating feedback loops where generated lyrics are evaluated and refined based on human feedback could further enhance the quality and relevance of the outputs. This iterative process can help fine-tune the model to better meet the nuanced expectations of both artists and listeners.

Finally , exploring other pre-trained models, such as transformer-XL or GPT-3, could offer improvements in capturing long-range dependencies.

References

[FS14] Michael Fell and Caroline Sporleder. Lyrics-based analysis and classification of music. pages 620–631, 2014.

[GLM20] Harrison Gill, Daniel Lee, and Nick Marwell. Deep learning in musical lyric generation: an lstm-based approach. *The Yale Undergraduate Research Journal*, 1(1):1, 2020.

[GSB] Daniel Gordin, Arnab Sen Sharma, and Jaydeep Borkar. Music lyrics classification & generation using rnns.

[Kab21] Abdullah Talha Kabakus. A novel covid-19 sentiment analysis in turkish based on the combination of cnn and bilstm on twitter. *Department of Computer Engineering, Faculty of Engineering, Duzce University, Duzce, Turkey*, 2021.

[LFM19] Hsin-Pei Lee, Jhih-Sheng Fang, and Wei-Yun Ma. icomposer: An automatic songwriting system for chinese popular music. pages 84–88, 2019.

[Mah20] Rojina Maharjan. Abstractive text summarization using recurrent neural network with attention based mechanism. 2020.

[Pet19] Fredrik O. Pettersson. Optimizing deep neural networks for classification of short texts. 2019.

[Pot15] Romanov A. Rumshisky A. Potash, P. Ghostwriter: Using an lstm for automatic rap lyric generation. *In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing.*, 2015.

[SKAO22] Xiaoyu Li Samuel Kofi Akpatsa, Hang Lei and Victor-Hillary Kofi Setornyo Obeng. Sentiment classification of covid-19 online articles: The impact of changing parameter values on bidirectional lstm. *International Journal of Engineering Research & Technology (IJERT)*, 11(1):526–532, 2022.