NLP Lyrics Generation

Hazem Reda 49-0785 , Omar Ashraf 49-5964

May 19, 2024

1 Introduction

In recent years, the intersection of artificial intelligence and creative expression has led to groundbreaking advancements in various fields, including music composition. Among these innovations, the use of Natural Language Processing (NLP) techniques for generating lyrics has garnered significant attention. In this paper, we explore the application of NLP methods to the task of automatically generating lyrics, presenting both the current state-of-the-art techniques and potential future directions in this evolving field.

Crafting compelling lyrics is a time-consuming and intricate process, demanding a deep understanding of language, rhythm, and thematic coherence. Moreover, the sheer volume of lyrical content across various musical genres poses a challenge for songwriters seeking fresh, original material.

The emergence of Natural Language Processing (NLP)-powered lyric generation offers a promising solution. By analyzing vast textual data and generating human-like language, these algorithms provide inspiration for new compositions, aiding in overcoming writer's block and streamlining the music-making process.

In this Research, we provide a comprehensive overview of lyric generation using NLP, exploring methodologies, challenges, and opportunities. Through empirical analysis and critical synthesis, we aim to contribute to the ongoing dialogue on computational creativity, inspiring further innovation in music composition and cultural enrichment.

2 Literature Review

2.1 NLP

Natural language processing (NLP) is a sub-field of artificial intelligence that focuses on enabling computers to understand and process human languages. It highlights the interactions between humans and computers through natural language, studying fundamental technologies for interpreting word meanings, syntactic and semantic processing, and developing applications such as machine translation, question-answering, information retrieval, dialog systems, text generation, and recommendation systems. NLP is dissected into three fundamental perspectives: modeling, learning, and reasoning, each playing a crucial role in advancing NLP technology.[ZDLS19]

2.1.1 Modeling

This perspective focuses on designing appropriate network structures for different NLP tasks. It involves encoding natural language sentences in neural networks and generating sequences of labels or sentences for tasks such as classification, sequence labeling, question-answering, dialog systems, natural language generation, and machine translation.

2.1.2 Learning

The learning perspective aims at optimizing model parameters using various learning methods. These include supervised learning (for tasks with abundant labeled data), semi-supervised and unsupervised learning (for low-resource tasks with limited or no labeled data), multitask learning, transfer learning, and active learning.

2.1.3 Reasoning

Reasoning is highlighted as a crucial aspect of NLP to generate answers to unseen questions by manipulating existing knowledge with inference techniques. It emphasizes the importance of building interpretable and knowledge-driven neural NLP models to handle complex tasks effectively.

2.2 RNN

Recurrent Neural Networks (RNNs) are a class of artificial neural networks designed to process data sequentially. In contrast to conventional feedforward neural networks, which handle each input separately, RNNs maintain an internal state or memory that allows them to capture information from previous inputs and use it to influence future predictions or outputs.[LBE15]

The key characteristic of RNNs is their ability to handle input sequences of variable lengths and model temporal dependencies between elements in the sequence. This makes RNNs appropriate for tasks like machine translation, speech recognition, natural language processing, handwriting recognition, and time series analysis where the context and sequence of the data are important.[1]

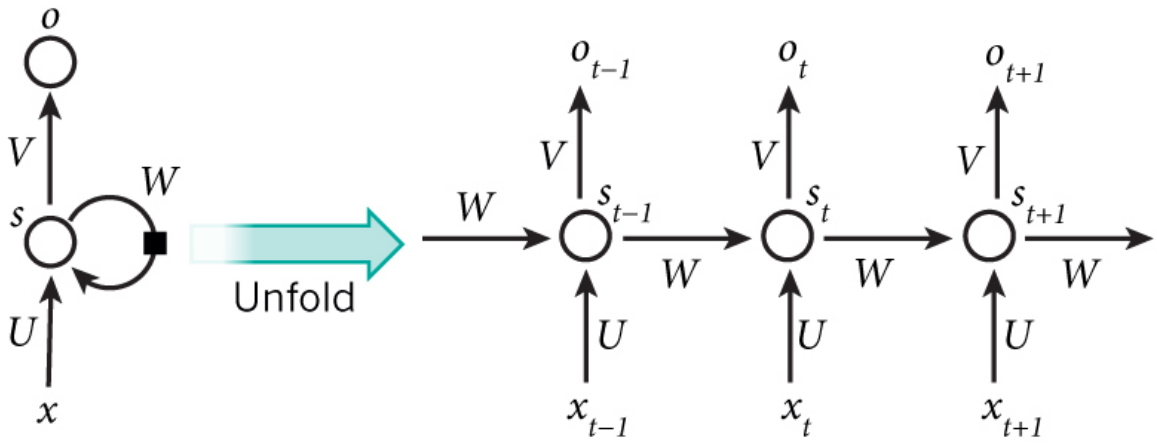

Figure 1: RNN Architectures.

The basic building block of an RNN is a recurrent unit, which typically takes an input vector and the previous state as inputs and produces an output and a new state. The output can be used for prediction or passed as input to the next step in the sequence. By propagating information through time, RNNs can capture long-term dependencies and learn to generate sequences or make predictions based on contextual information.

2.3 LSTM

A recurrent neural network (RNN) model that is particularly good at identifying long-term dependencies in sequential data is the Long Short-Term Memory (LSTM) architecture. The key component of LSTM is the existence of a memory cell that maintains its state throughout time, as well as non-linear gating units that regulate the information entering and leaving the cell. LSTMs can be used for a variety of tasks involving sequential data because of their ability to learn and retain patterns in sequences according to their design.

Long Short-Term Memory (LSTM) networks work by utilizing a specialized architecture that allows them to effectively capture long-term dependencies in sequential data. Here is a simplified explanation of how LSTM works:[2]

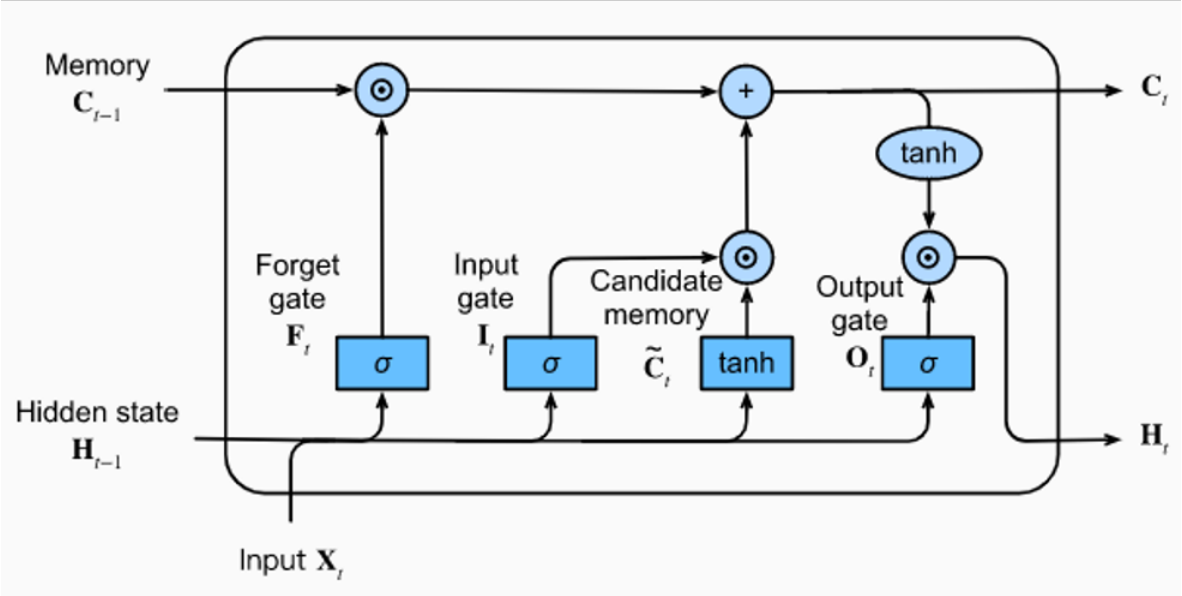

Figure 2: LSTM Architectures.

Memory Cells: The memory cell, which has the ability to retain information over time, is the main part of an LSTM. This cell state runs along the entire chain of the LSTM, and information can be added or removed from it through interactions with gates.

Gates: LSTMs have three types of gates that regulate the flow of information:

- Forget Gate: Decides what information to discard from the cell state.
- Input Gate: Determines what new information to store in the cell state.
- Output Gate: Selects what information to output based on the cell state.

Activation Functions: LSTMs use activation functions to control the flow of information. These functions include the sigmoid function to regulate the gate outputs and the hyperbolic tangent function to regulate the cell state.

Information Flow: The forget gate decides which information from the previous cell state to forget. The input gate decides what new information to store in the cell state. The cell state is updated based on the forget gate, input gate, and the previous cell state. The output gate determines the output based on the updated cell state. Training: During training, LSTMs use backpropagation through time to update the weights and biases of the network. The network learns to adjust the gates and cell state to optimize performance on the given task.[GSK+15]

One of the methods LSTM are used involved collecting a dataset of lyrics from various composers and musicians. The Lyrics of a particular artist were collected and standardized. The string was then broken into a word and rhyme index. A multilayer LSTM-based model is then trained with bidirectional neurons and BERT integration. Bidirectional neurons are a type of neural network architecture that processes input data in two directions: from past to future and from future to past. In the context of lyric generation, bidirectional neurons allow the model to consider information from both preceding and succeeding words in a sequence. By capturing dependencies in both directions, the model gains a more comprehensive understanding of context and relationships within the lyrical content. This bidirectional processing aids in improving the coherence and relevance of the generated lyrics by incorporating

a broader context of the input data. BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based model developed by Google researchers for natural language processing tasks. BERT excels in capturing contextual relationships between words by leveraging bidirectional attention mechanisms. In the study, BERT integration into the model architecture enhances the understanding of language context and semantics. By pre-training on a large corpus of text data, BERT provides the model with rich language representations that can be fine-tuned for specific tasks like lyric generation. This integration allows the model to generate more nuanced and human-like lyrics by leveraging the advanced language understanding capabilities of BERT.

This model architecture enabled the capturing of long-term dependencies and contextual relationships in lyrical content. The training process involved feeding the dataset into the model to learn patterns and structures, aiming to generate lyrics that align with the designated musical genre. During the training process, the model iteratively learned from the input lyrics, adjusting its internal parameters to minimize the loss function and improve its ability to generate coherent and relevant lyrics.[NMB23]

In addition, another way LSTM can be used to Generate Lyrics for Hip-Hop and Gospel Songs, the data preparation process involved collecting a diverse range of song lyrics from various artists in these genres. The collected lyrics underwent cleaning procedures to remove punctuation, non-alphabetic tokens, and standardize the text by converting all words to lowercase. Subsequently, the lyrics were tokenized into individual words, and the dataset was split into training and testing sets to facilitate model training and evaluation. Word-to-index mappings were created to represent words numerically for model input. Data analysis was conducted to understand the distribution of words and thematic elements in the lyrics. Through these steps, the data was preprocessed to ensure consistency and accuracy, providing a structured dataset for training language models to generate coherent and meaningful lyrics for Hip-Hop and Gospel songs.

N-gram models, such as Bi-gram and Tri-gram models were utilized, as baseline approaches to understand the basic patterns and structures present in the song lyrics dataset. These n-gram models provided a foundation for exploring more advanced techniques in lyric generation.

The Bi-gram model focuses on the current word and the former word in the text. It considers the combination with the highest probability as the output, without taking into account sentence structure or syntactic relationships. This model may produce results that lack meaningful context or syntactic structure.

The Tri-gram model extends the Bi-gram model by considering the current word along with the two preceding words. This allows for capturing longer relationships between prediction words, potentially leading to more meaningful and coherent sentences. The Tri-gram model takes into account more words in the prediction process, enhancing the model's ability to generate text with better syntactic structure.

When compared together (Bi-gram and Tri-gram models) it was found that the Tri-gram model generated better sentences compared to the Bi-gram model. Both models exhibited valid syntactic structure and meaningful sentences, but the Tri-gram model showed improvements in sentence quality and coherence.

Two different advanced methods were considered for structuring the data as input to the LSTM models. Each song was added to a single, sizable text file, with a newline at the end of each line, in order to prepare the data. After gathering all of the dataset's lines, they were divided into three groups: 80 percent for the train, 20 percent for the test, and 10 percent for the train that was further divided into a validation set. In order to tokenize these sets, word-to-index and index-to-word dictionaries were made, and each line ended with a unique token. We began by selecting a word at random from the lexicon and generated x words to create a new song. Every time the unique token was extracted from the model, a newline character was used in its place.

In version 2, the song will remain in its entirety rather than having each line separated and rearranged. Every song in the dataset will still have a newline character at the end of each line, but a single line with the character "-" will separate each song. Rather than splitting and shuffle the dataset by line, it is done so per song. A "eov" token will be tokenized at the conclusion of songs, and a "eol" token will substitute newlines when they are tokenized. The model is instructed to generate words until the "eov" appears, replacing any "eol" with a newline character in order to create a new song. Version 2 showed better results.[KMWY23]

The main goal of the model is to create song lyrics that are contextually appropriate in the Korean language. The focus is on generating lyrics based on predicates, which are crucial contextual factors in Korean, an agglutinative language.

The learning model utilizes LSTM cells for generating Korean song lyrics. The model operates at a character unit level, predicting the next character in the lyrics based on the input text. Training data includes space and new line characters treated as separate characters. The model structure involves predicting characters sequentially to generate coherent lyrics.

The training starts by calculating the state values of the LSTM cells. These state values capture the memory and information flow within the network. The total cost is computed based on the difference between the embedding values of the predicted characters and the correct characters.

The calculated total cost is then plugged into a partial differential equation. The results of this equation are organized into a vector, which helps in understanding the impact of the cost on the model's parameters.

Gradient clipping is a technique used to keep the gradients from growing too large and creating instability during training. In order to provide steady and efficient learning, this procedure entails modifying the maximum value of the gradients.

The learning sessions are started to update the model's weights after the gradients have been adjusted. By iteratively changing the parameters in accordance with the computed gradients and cost function, the goal is to minimize the losses.

The state values in the LSTM cells are computed and updated continuously during the learning process. As training goes on, improved text creation and prediction are made possible by the state values being updated, which aids in the model's ability to remember and update information over time.[SNLK23]

3 Data Analysis

3.1 Introduction

In this section, we present the results of our data analysis over the dataset we used. We explore various aspects of the data and derive insights that

contribute to a better understanding of the dataset and the operations that were done on the data.

3.2 Operations

On the given data we conducted a series of operations starting with text preprocessing by converting everything to lower case to make it easier to track and not mistake a word starting with an uppercase letter as being a different word from the same word starting in a lowercase letter. Then we started tokenization which is splitting the input into different tokens to facilitate further operations on the words. Then we opted to perform lemmatozation but not stemming since, while we wanted to be able to link similar words better and have a more accurate representation of the vocabulary, we did not want to lose the semantic meaning of the words or have words that are not real being generated which stemming would have done. Then we proceeded to clean the data by removing stop words and punctuation from the tokens to be able to better understand the vocab and remove unneccessary values which would negatively affect accuracy.

3.3 Insights

The dataset consists of 57650 rows of data and has columns that represent the following: The artist name, The song name, The link to the song and finally the text/lyrics of the song. Through Analyzing the dataset we could see that it was very diverse in terms of the different artists that were in it with some being repeated almost 200 times and some only having a few songs in the dataset. By examining the length of the song, we could identify that there was a significant variation in the song length across different artists. The typical length of songs for the majority was around 200 words or slightly under the 200 word mark. We also checked for the word frequency to see which words were repeated most to get an idea of the prevalent theme or topic around the songs. Most common words in the dataset:

3.4 Limitations

3.4.1 Data Quality

The data was found not to contain any null values which was good since it meant we didn't have to do any operations to handle the nulls. However, the text did contain a lot of strings or substrings which are more common in spoken language and that were not detected and dealt with correctly by the lemmatization or the removal of stop words and punctuation. Words containing substrings such as "n't" or " 'm " were still taken as words of their own which negatively affects the quality of the data.

4 Training a Model

4.1 Methodology

4.1.1 Data Collection

The dataset used in this project consists of song lyrics. The data is stored in a CSV file, "Dataset m2 csv2 snip.csv," which contains columns with text data and word tokens (as well as the artist name, song title, link to the song which were ignored in the model training process)

4.1.2 Data Preprocessing

Preprocessing steps included: - Loading Data: Reading the CSV file into a pandas DataFrame. - Tokenization: Converting words into integer sequences. - Sequence Creation: Generating input-output pairs for training the model. - Padding Sequences: Ensuring all sequences have the same length for batch processing.

Steps: 1. Loading Data: "'python

data = pd.read$_csv$("/$content/Datasetm2csv2snip.csv$")

"'

2. Converting String Representations of Lists to Actual Lists: "'python

data["word$_t okens$"] = $data$["$word_t okens$"].$apply(lambdax : literal_eval(str(x))ifisi$

3. Extracting Relevant Columns: "'python

text$_d ata$ = $data$["$text$"].$tolist()$

word$_tokens = data[$"$word_tokens$"$].$tolist()$‘‘‘

4. Converting Float Values to Strings in text$_data$: ‘‘‘$python$

text$_data = [str(item) for item in text_data]$‘‘‘

5. Concatenating All Lyrics into a Single String for Vocabulary Creation: ‘‘‘python

all$_lyrics = ' '.join(text_data)$

‘‘‘

6. Tokenizing the Text and Creating Vocabulary: ‘‘‘python

tokenizer = Tokenizer()

tokenizer.fit$_on_texts([all_lyrics])$

‘‘‘

7. Converting Words to Integers Based on the Vocabulary: ‘‘‘python

sequences = tokenizer.texts$_to_sequences(text_data)$

‘‘‘

8. Creating Input-Output Sequences: ‘‘‘python

input$_sequences = []$

output$_sequences = []$

for sequence in sequences:

for i in range(1, len(sequence)):

input$_sequence = sequence[: i]$

output$_sequence = sequence[i]$

input$_sequences.append(input_sequence)$

output$_sequences.append(output_sequence)$‘‘‘

9. Padding Sequences to Ensure Uniform Length: ‘‘‘python

max$_sequence_length = max([len(seq) for seq in input_sequences])$

input$_sequences = pad_sequences(input_sequences, maxlen = max_sequence_length, pa$
$pre')$

output$_sequences = to_categorical(output_sequences, num_classes = len(tokenizer.wo$
1)

‘‘‘

Model Architecture The LSTM model consists of the following layers:

- Embedding Layer: Maps input sequences to dense vectors of fixed size.
- LSTM Layers: Two LSTM layers to capture temporal dependencies in the data.
- Dense Layer: Outputs probabilities for the next word in the sequence.

“‘python

model = Sequential([

Embedding(input$_dim = len(tokenizer.word_index) + 1, output_dim =$ 100,

input$_length = max_sequence_length$),

LSTM(128, return$_sequences = True$),

LSTM(128),

Dense(len(tokenizer.word$_index) + 1, activation =' softmax'$)

]) “‘

Training Process

- Data Splitting: Splitting the dataset into training and validation sets.
- Data Generator: Using a data generator to feed batches to the model.

- Training Configuration: Training the model using ‘fit$_generator$‘.

Splitting the Data: “‘python

X$_train, X_val, y_train, y_val = train_test_split(input_sequences, output_sequences, test_siz$ 0.2,

random$_state = 42$)“‘

Data Generator: “‘python

def data$_generator(input_sequences, output_sequences, batch_size)$:

total$_sequences = len(input_sequences)$

while True:

for start in range(0, total$_sequences, batch_size)$:

end = min(start + batch$_size, total_sequences$)

yield input$_sequences[start : end], output_sequences[start : end]$‘‘‘

Training the Model: ‘‘‘python

history = model.fit(data$_generator(X_train, y_train, batch_size)$,

steps$_per_epoch = len(X_train)//batch_size$,

epochs=2,

validation$_data = data_generator(X_val, y_val, batch_size)$,

validation$_steps = len(X_val)//batch_size)$‘‘‘

4.2 Results

Training Performance - Training and validation loss over epochs. - Training and validation accuracy over epochs.

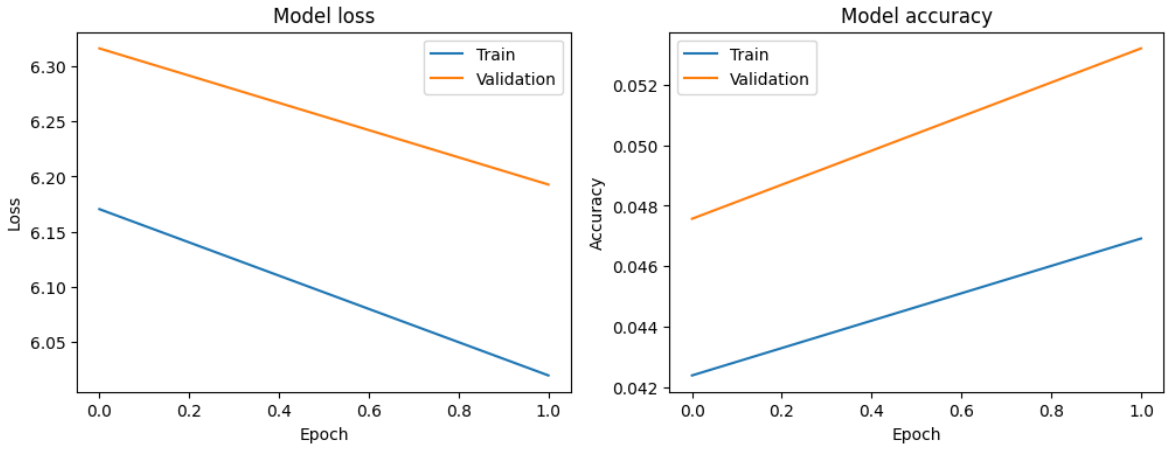

Figure 3: Training Performance

‘‘‘python

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history[’loss’])

plt.plot(history.history[’val$_loss’])

plt.title(’Model loss’)

plt.ylabel(’Loss’)

plt.xlabel(’Epoch’)

plt.legend(['Train', 'Validation'], loc='upper right')
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val$_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show() "‘
Evaluation Validation Results:

- Loss: 6.353457927703857
- Accuracy: 0.04410751163959503

"‘python evaluation = model.evaluate(X$_val, y_val, verbose = 0$)
loss, accuracy = evaluation
print(f"Validation Loss: loss")
print(f"Validation Accuracy: accuracy") "‘

4.3 Discussion

4.3.1 Model Performance

The model demonstrated reasonable learning ability, but the validation accuracy indicates that there is significant room for improvement. The low accuracy suggests the model might be overfitting or that the dataset is not large enough to generalize well.

4.3.2 Challenges

- Memory Issues: Handling large datasets caused memory overflow, which was mitigated by using data generators.

- Overfitting: The model may have overfitted the training data, as suggested by the low validation accuracy.

4.3.3 Improvements

- Increase Dataset Size: Collect more lyrics data to improve generalization.

- Regularization Techniques: Apply dropout layers to prevent overfitting.

- Hyperparameter Tuning: Experiment with different model architectures and training configurations.

4.4 Conclusion

4.4.1 Summary

This project successfully implemented an LSTM-based model to generate song lyrics. The preprocessing steps, model architecture, and training process were thoroughly detailed, and the results showed initial promise despite the challenges encountered.

4.4.2 Future Work

Future efforts could focus on enhancing the model's performance through more extensive data collection, advanced regularization techniques, and further hyperparameter tuning.

5 Pre-Trained Model

The Song Lyrics Generator is a Recurrent Neural Network (RNN) model designed to generate song lyrics based on an input string. This approach leverages the capabilities of RNNs to handle sequence-based tasks by predicting the next word iteratively, starting from the input and continuing until a fixed-length text is produced.

5.1 Data

The dataset for this project is sourced from the Poetry dataset on Kaggle. It includes lyrics from various artists such as Kanye West, Lil Wayne,

Adele, and the Beatles, among others. Each artist has a dedicated file containing their song lyrics, with a total of 49 artists represented.

5.2 Model Architecture

The model used for generating song lyrics is an RNN, specifically utilizing Gated Recurrent Units (GRUs), which are a type of RNN similar to Long Short-Term Memory (LSTM) networks but with fewer parameters. The architecture of the Song Lyrics Generator can be broken down as follows:

- Embedding Layer: Converts the input text into dense vectors of fixed size, creating an embedding of the input text.
- GRU Layer: Processes the embedded input, maintaining a hidden state that serves as the memory of the network, influenced by both the current input and the hidden state from the previous time step.
- Dense Output Layer: Generates the output, which is then transformed into the next word in the sequence.

The parameters of the network are:

- Vocabulary Size (vocab size): The total number of unique characters in the dataset.
- Embedding Dimension (embedding dim): 256
- Number of RNN Units (rnn units): 1024

5.3 Mathematical Formulation

The RNN's computations at each time step t are described by the following equations:

- x(t): Input at time t
- h(t): Hidden state at time t, representing memory
- o(t): Output at time t
- y(t): Softmax of o(t)

- W, V, U: Network weights

These components work together to generate coherent lyrics by predicting the next word in a sequence, influenced by the preceding words and the learned context from the dataset.

5.4 Results

Epoch	Training Loss	Training Accuracy
1	6.3181	0.0754
2	5.7102	0.1037
3	5.3516	0.1306
4	5.0679	0.1522
5	4.8254	0.1697
6	4.6161	0.1841
7	4.4340	0.1982
8	4.2644	0.2139
9	4.1122	0.2276
10	3.9726	0.2416
11	3.8460	0.2560
12	3.7260	0.2702
13	3.6128	0.2833
14	3.5075	0.2971
15	3.4101	0.3100

Table 1: Training and Validation Results over 15 Epochs

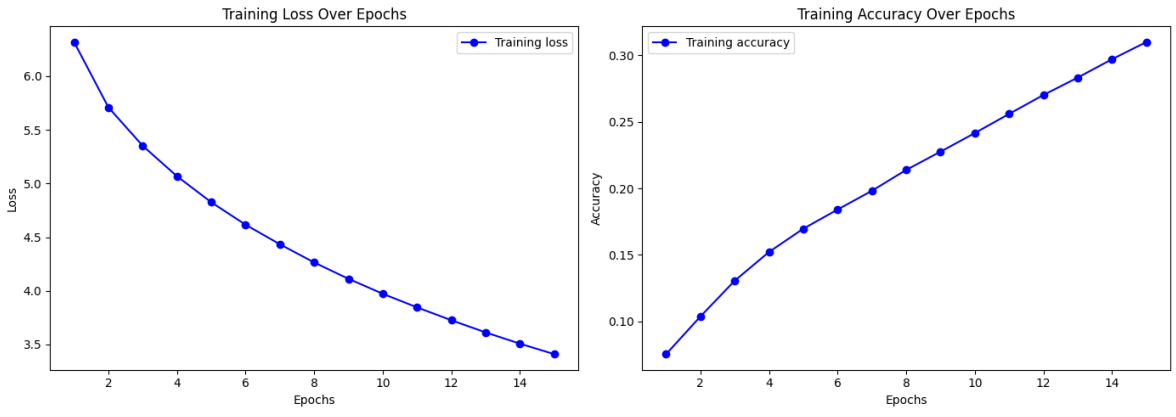

Figure 4: Training Performance

5.5 Conclusions

5.5.1 Model Learning

The model successfully learned from the training data, as evidenced by the decreasing loss and increasing accuracy.

Overfitting: Without validation data, it's challenging to assess overfitting. Ideally, validation loss and accuracy should be monitored to ensure the model generalizes well to unseen data.

5.5.2 Further Improvements

Future work could involve fine-tuning the hyperparameters, using more epochs, or employing regularization techniques to improve performance. Incorporating validation data will provide better insights into the model's generalization ability.

References

[GSK+15] Klaus Greff, Rupesh K. Srivastava, Jan Koutník, Bas R. Steunebrink, and Jurgen Schmidhuber. Lstm: A search space odyssey. *arXiv preprint arXiv:1503.04069*, March 2015. Version 2: 4 Oct 2017.

[KMWY23] Bryon Kucharski, Damain Moquin, Juexing Wang, and Weiqiu You. Language modeling to generate lyrics for hip-hop and gospel songs. *University of Massachusetts Amherst*, 2023. Emails: bkucharski@umass.edu, dmoquin@umass.edu, juexingwang@umass.edu, wyou@umass.edu.

[LBE15] Zachary C. Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. June 2015. Working Paper.

[NMB23] Ilakiyaselvan Nagappan, Satyaki Mandal, and Sandipta Bhadra. *Lyrics Generation Using LSTM and RNN*. June 2023. Chapter.

[SNLK23] Sung-Hwan Son, Gyu-Hyeon Nam, Hyun-Young Lee, and Seung-Shik Kang. Korean song-lyrics generation by deep learning. *Kookmin University, Dept. of Computer Science*, 2023. Emails: zldejagkcm@naver.com, ngh3053@gmail.com, le32146@gmail.com, sskang@kookmin.ac.kr.

[ZDLS19] Ming Zhou, Nan Duan, Shujie Liu, and Heung-Yeung Shum. Progress in neural nlp: Modeling, learning, and reasoning. *Microsoft Research Asia*, 100080, April 2019. Received 30 April 2019, Revised 30 August 2019, Accepted 13 October 2019, Available online 7 January 2020.