# Deep Learning Assignment 3

Yonatan Amaru – 316365998  
Noam Munz – 207042292

## Introduction

In this project, we create a Lyrics Generator—a DL-driven system designed to craft original song lyrics by learning from a dataset of songs and their corresponding MIDI files. Through the integration of LSTM neural networks and MIDI autoencoder, the model aims to blend the lingual content with the rhythm and mood of music.

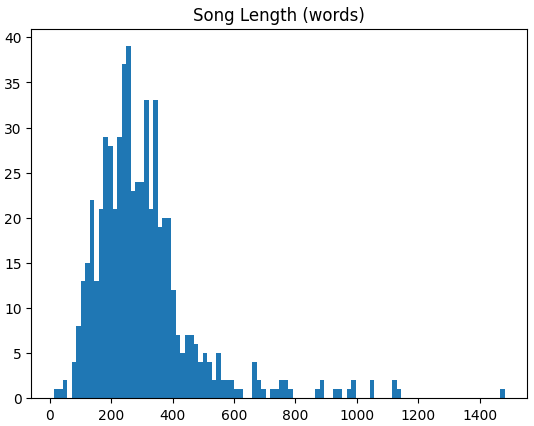
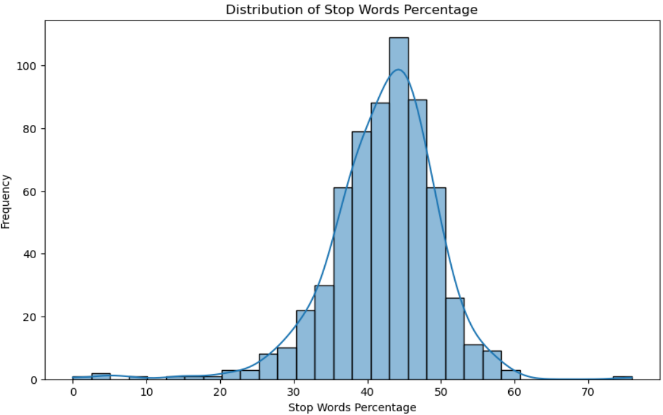
The project repository, and detailed information about running the code, can be found at <https://github.com/AmaruCrunch/lyric-generation/>.

## Data Analysis

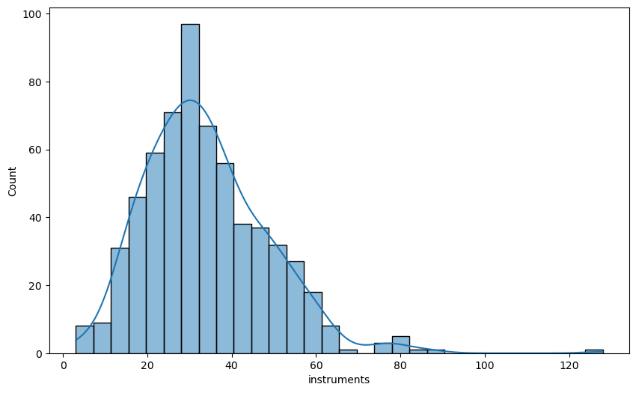
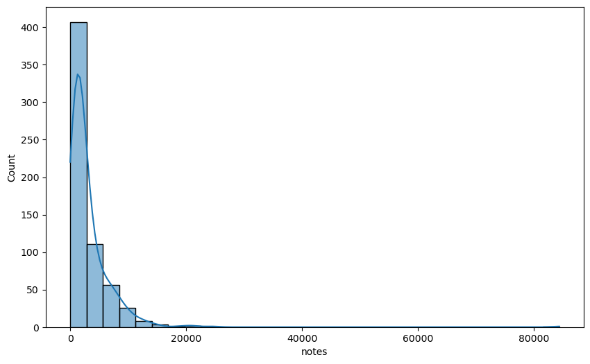
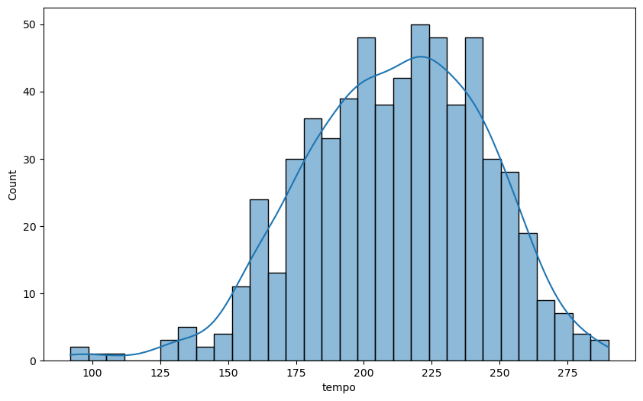
**Overview of the Dataset**

* The dataset comprises 599 songs, each associated with lyrics and corresponding MIDI files.
* The data is split into 594 training samples and 5 test samples.

**Lyrics Analysis**

* Song Length Distribution:
  + The length of songs (measured in words) follows a normal distribution, with the majority of songs containing fewer than 600 words.
* Stop Word Analysis:
  + Approximately 43% of the words in the lyrics are stop words, indicating a normal distribution of common versus unique words across the dataset.
* Vocabulary and Word2Vec Coverage:
  + The Word2Vec model utilized has a vocabulary of 3 million words.
  + Roughly 2% of the words in the lyrics dataset are out-of-vocabulary (OOV) for the Word2Vec model, indicating a good coverage but also highlighting areas for potential enhancement in understanding song lyrics. We will delve into this in the preprocessing.

**MIDI Analysis**

* **Instrumentation:**
  + Most songs feature between 10 and 60 different instruments, showcasing a wide range of musical complexity and styles.
* **Note Distribution:**
  + The distribution of notes per song has a heavy-tailed distribution, with most songs having fewer than 10,000 notes but a few outliers having significantly more.
* **Tempo Analysis:**
  + Tempos of songs in the dataset range between 150 and 275 beats per minute (BPM), suggesting a variety of song paces from moderate to fast.

**Summary**

The analysis of lyrics reveals common patterns in song length and word usage, while the MIDI analysis uncovers the instrumental diversity and note distribution among songs.

## Preprocessing

**Lyrics preprocessing:**

* For the lyrics preprocessing we used GoogleNews word2vec model with vocabulary size of 3,000,000.
* We decided to remove both stop words and punctuation after some experimentation detailed in the table below:

Percentage of out-of-vocabulary Words by Filtering Method

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Quantile | 25% | 50% | 75% | 90% | 95% |
| No filtering | 7.69% | 9.96% | 12.8% | 16.01% | 17.51% |
| Removing Punctuation | 4.73% | 6.58% | 8.68% | 10.78% | 11.98% |
| Removing Stop Words | 2.61% | 5.77% | 9.67% | 14.29% | 17.35% |
| Removing Both | **0%** | **0%** | **0.52%** | **1.76%** | **4.0%** |

* We used three tokens (existed in the word2vec vocabulary):
  + BOS – token that represents beginning of song. Will be used in inference as the first word.
  + EOS – end of sentence. Replaced ‘&’ in the data so represent end of sentence
  + EOF – end of file. Represents end of song, appended at the end. In inference stage we will stop generating after this token.
* Each song was divided into sequences (dynamic size) and each word in sequence was converted into a vector.
* For targets, each sequence word was designated the class of its following word according to the vocabulary index.

**MIDI Preprocessing:**

* Feature Extraction from MIDI Files:
  + normalized pitches
  + normalized velocities
  + mean chroma values.
* Normalization:
  + Pitches and Velocities: These are normalized by dividing by 127, the maximum MIDI value for pitches and velocities, ensuring these features are scaled between 0 and 1.
  + Mean Chroma: This feature represents the average intensity of each of the 12 chromatic pitch classes throughout the song, providing insight into its harmonic content.
* Feature Vector Construction:
  + The normalized pitches, velocities, and mean chroma values are concatenated into a single feature vector for each song.
  + If the number of pitches or velocities exceeds the max\_length (2048 by default), the data is truncated to fit; otherwise, it's zero-padded to ensure uniform vector lengths.

Train data was divided into 534 train samples and 60 validation samples to fine tune training parameters.

## Architecture

The architecture of our lyrics generation model consists of two main components: an LSTM network and a MIDI autoencoder. The LSTM network processes the textual information, while the autoencoder handles the musical context provided by MIDI files.

For this task we will test two approaches, the first will include simply feeding the each word of the song. For the second, we will first train an autoencoder that learns a smaller number of features present in each melody.

A diagram of a code

Description automatically generated

We hope that our encoded melody vectors will represent the important features and will be less noisy which will lead to greater performance. Even an identical level of performance will suffice as the computational resources required for the encoded vectors are lesser.

## Autoencoder

Architecture (Layer sizes are Dynamic):

Encoder:

* + Linear layer (4096)
  + Relu layer
  + Linear layer (2048)
  + Relu layer
  + Linear layer (1024)

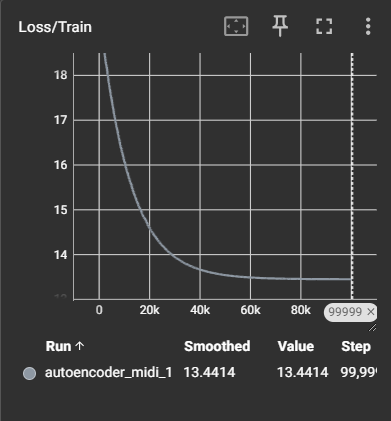
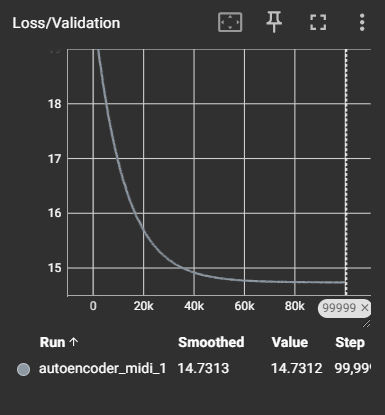
Decoder:

* + Linear layer (1024)
  + Relu layer
  + Linear layer (2048)
  + Relu layer
  + Linear layer (4096)
  + Sigmoid

We trained the autoencoder with the following hyperparameters:

* Epochs = 100000
* Batch size = 128
* We tested the following dropout values: [0, 0.1]

We used the same train and validation test as the original data to train and evaluate the performance of the autoencoder. The dropout didn’t seem to affect the validation loss a lot so we removed it.



## model

LSTM

* Input: concatenated Text vector + embedding midi (1324)
* LSTM: hidden\_size=128, number of layers=2
* Output: word2vec model vocabulary size logits (3,000,000)

We tested multiple approaches, and in consideration of GPU memory allocation and training time we chose a basic LSTM architecture, focusing more on the input creation.

## training

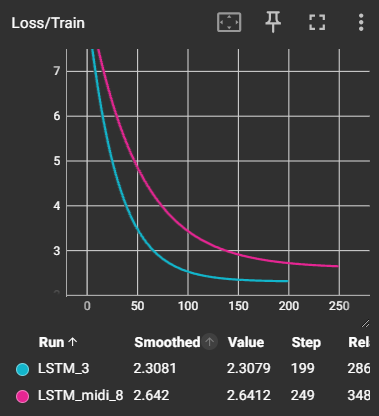
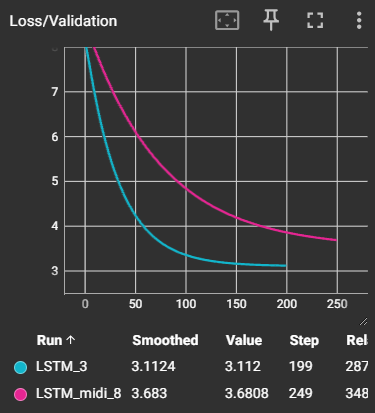
For training, initially we ran the model with different hyperparameters using the first approach (no MIDI autoencoder), this is how we set the model hyperparameters and optimized the learning rate.

We test the following hyperparameters:

* Epochs: [30,50, 100, 200]
* Learning rate: [0.0005, 0.001]
* Dropout: [0,0.1]
* LSTM hidden layers: [1,2]

Our testing focused on checking in which the lost converged the best. This is a tricky test, as we don’t want to over fit the wards to the data to much and we are working with a very limited dataset.

After finding the optimal hyperparameters, we integrated the Midi embedding feature vector, and retrained the model, the model obtained its ability to learn, but produced a slightly higher loss.



## Lyrics generator

The Lyrics Generator class uses our trained LSTM model and Autoencoder model, as well with the word2vec model to generate songs given a Midi file.

* Word Sampling: uses the output logits from the LSTM to sample the next word, with an adjustable **temperature** parameter to control randomness.
* Lyrics Generation:
  + Accepts a song\_key to utilize the corresponding MIDI embedding for musical context.
  + Begins with a seed\_text, typically 'BOS' (Beginning of Song), to start the lyrics generation.
  + Generates words one by one up to a max\_length, concatenating each new word to the growing lyrics.
  + Ends generation upon reaching 'EOF' (End of File) or the maximum length.
* Output Processing: The final generated text is cleaned to produce coherent lyrics, stripping special tokens like 'EOF' and 'EOS' (End of Sentence).

The Lyrics Generator streamlines the synthesis of lyrics aligned with the mood and style of specific songs.

## experiment

**Setup**

We compare the two LSTM-based lyric generators that use MIDI files as input. The key difference is:

* Model 1: Employs an autoencoder to create embeddings (representations) of the MIDI features.
* Model 2: Does not use an autoencoder for MIDI embeddings.

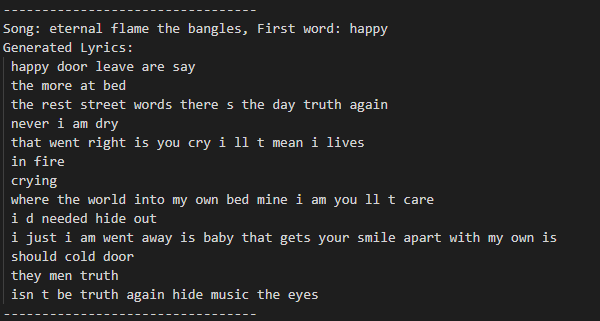
We used an unsupervised test set, in which the model wasn’t trained on either the lyrics or the midi files, and tested the model performance on two tasks:

1. Generating lyrics given the music file and the first word of the song
2. Generating lyrics given the music file and a word of choice

For the first test, we use the original lyrics as a ground truth, to see how similar each of the models performs. For the second test we check to see if the sentiment is more reflective of the word or music.

We set the song limit length to 100, and the temperature to 1.

## Evaluation

****

**We documented the results in the outputs/ folder of the project.**

We note that although we used an average temperature (1), both models produced lyrics with a significantly high variation. This points to the limited training.

Both models succeeded and producing (somewhat) lyrical texts. In terms of structure we did not identify a significant difference.

**Musical Feedback**

* Model 1 (With Autoencoder): Produced songs slightly more in line with the original lyrics in term of mode.
* Model 2 (Without Autoencoder): Generates less structured and often nonsensical lyric, without a clear motif.

**Sentiment Understanding**

We tested both models by providing three emotion words ("happy", "sad", "angry") as the first word and a MIDI file.

* Model 1 (With Autoencoder): Produces more coherent and thematically consistent lyrics. The generated songs, while imperfect, tend to reflect the specified emotion.
* Model 2 (Without Autoencoder): Generates less structured and often nonsensical lyrics. The connection to both the input MIDI and given emotion word is weaker.

**Key Takeaways**

Autoencoders Help: The use of an autoencoder to handle MIDI data improves the ability of the lyric generator to learn meaningful patterns and produce lyrics that are more aligned with the musical input and the desired emotion.

Room for Improvement: While the results are encouraging, both models have significant room for improvement in terms of grammatical correctness, thematic consistency, and overall lyric quality.

## Conclusion

The Lyrics Generator shows promise but highlights key areas for development:

* **Functionality**: It generates lyrics with musical context, yet there's room for improvement in consistency and relevance.
* **Pretrained Autoencoder**: Pretraining the MIDI autoencoder has been advantageous for better musical alignment.
* **Evaluation Complexity**: Success is challenging to quantify due to the creative nature of lyrics generation.
* **Longer Context**: A wider context window could improve the model's lyrical coherence.
* **Dataset Expansion**: A larger dataset, or fine-tuning on pre-existing models, may enhance performance and overcome current dataset limitations.