Finetuning transformer models to generate lyrics

Hassan Mustafa[1]

Abstract
this project looks into the process of using transformer models to generate music lyrics. More specifically the project describes the process of choosing the correct transformer model aswell as the correct model head to be able to specialize the model to certain tasks. The projects also describes the process of finetuning a larger generalized model to perform well on a specific NLP task by using a dataset that the model is trained on.
Source code: https://github.com/Siala-94/musicGenerator

Author[1] *Media Technology Student at Linköping University, hasmu396@student.liu.se*

Keywords: NLP — Transformer models — bart language model

Contents

1. Introduction

Ever since the introduction of the transformer models in 2017[1], the way we handle NLP tasks is forever different. In contrast to the traditional sequential way of solving NLP problems using Neural Networks, transformer models are different in their structure and allows for parallel processing. This has resulted in some giant leaps in some previously difficult NLP tasks such as translation, text-generation and more. By using the parallel processing properties we can reduce the training time by a large amount and produce much better results than we previously were able to[2]. This project aims to apply some of the fundemental techniques of the transformer models to showcase how we can generate content based on some input from the user. In this case we will try to generate some lyrics based on a few inputs such as artist name, genre and song name.

2. Theory

The transformer model is characterized by its complex architecture, which requires some understanding of its components to fully leverage its capabilities. To comprehend how the model can be tailored for particular applications, we will explore the nuances of its structure, with a special emphasis on the sequence-to-sequence (seq2seq) head. Moreover, we will examine the process of fine-tuning a model to solve specific tasks. By dissecting the model's structure and examining the unique features of the transformer seq2seq head, we will gain insights into the fine-tuning process, enabling us to configure the model effectively for targeted challenges in natural language processing[2].

2.1 Structure of the transformer model

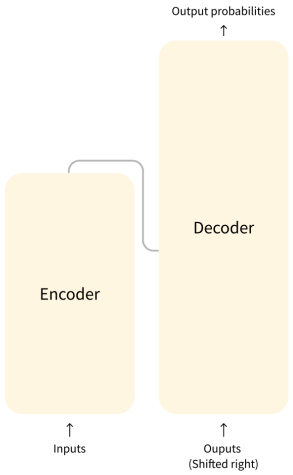

Figure 1. The Encoder-Decoder Architecture of Transformer Models

The transformer model differs from traditional sequential processes like neural networks by accounting for textual context, thanks to its encoder and decoder architecture. The encoder interprets the input text, establishing an understanding of it, while the decoder generates features to create a target output. They can operate independently or together to form various models. Utilizing only the encoder is good for tasks like sentence classification that require text understanding. Contrarily, employing just the decoder is ideal for text generation, producing coherent output texts. Combining both is well suited in scenarios like text summarization where both input understanding and output generation are crucial, these models are called sequence2sequence models[2].

2.2 Using the transformer model

The understanding and execution of a transformer-based model encompasses three integral phases, namely, the preprocessing phase, the model core, and the model head. Let's delve into each of these phases to unravel how they contribute to the functioning of the model, with a special emphasis on the role of the model head in fine-tuning the model to achieve desired outcomes[2].

2.2.1 Preprocessing

The journey of text data through the transformer pipeline begins with the preprocessing phase, where the raw textual input is prepared to be sent to the model. A crucial step in this phase is tokenization, where the input string is dissected into tokens. Tokens embody the lowest contextual representation of words or characters, encapsulating the inherent meaning. These tokens are mapped to corresponding numerical values or integers, rendering them comprehensible to the model. This conversion enables the transition of textual data into the model in a format that the model can comprehend[2].

2.2.2 Modeling

The tokenized input is passed onto the model. The models responsibility is to generate a tensor that correctly represents the data. It executes a series of operations on the input to produce an output tensor which is a numerical representation of the desired output, encapsulating the relationships and information present in the input data[2].

2.2.3 Model Head

The model head plays a crucial role in adapting the model to specialize in diverse tasks. It acts as an additional layer to the transformer model that tailors the output generated by the model to suit specific tasks. The model head can be seen as a customization tool. For instance, if the task is about classification, a model head with a softmax activation function might be used to output probabilities for different classes. On the other hand, for a regression task, a model head that outputs continuous values would be employed. Moreover, the model head can also include additional components like attention mechanisms or other custom layers, which can further refine the output to meet the specific requirements of a task. By manipulating the model head, it is possible to leverage the power and generalizability of transformer models, tuning them to perform in a wide array of applications ranging from text classification, sentiment analysis, to more complex tasks like translation or question answering[2].

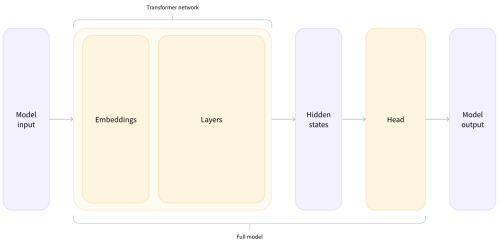

Figure 2. The transformer pipeline stages

2.3 finetuning a model

Numerous pre-trained models are available for utilization, offering some advantages especially when tackling NLP tasks. Training a transformer model from scratch requires substantial amount of data, which can be challenging to produce. However, by employing pre-trained models, one can effectively leverage previously trained models and fine-tune them to align with specific needs. Typically, larger models are trained to exhibit a broad generalization capacity. Fine-tuning these models allows for the utilization of the generalized features and honing them on a particular dataset to specialize the model for specific tasks. Consequently, the model can produce impressive performance even with a small amount of training data[2].

3. Method

3.1 Pretrained model selection

In deciding on a pretrained model suited for the task at hand, there are several criteria to consider. To generate a coherent structure of lyrics, as well as to generate some contextual coherence of the lyrics, there needs to be an special emphasis on a model that can handle both the context and structure of the lyrics. For this project, Facebook's BART (Bidirectional Auto-Regressive Transformers), which is essentially a seq2seq model with an encoder-decoder architecture, was chosen. This model is suited for tasks that center around generating text based on user input by using a language model head[2].

3.2 developing environment and deployment

Since transformers are capable of parallel processing, it is more advantageous to operate on a GPU. Google colab is thus used in this project to leverage its free of use GPU processing capabilities. This sets some constraints on the accessibility. The data set used for the training model must be relatively small in size due to the runtime restrictions. As for the front-end, dash is chosen primarily for its simplicity. The project has a relatively simplistic structure by displaying some input and output. Dash allows for the project to be run completely

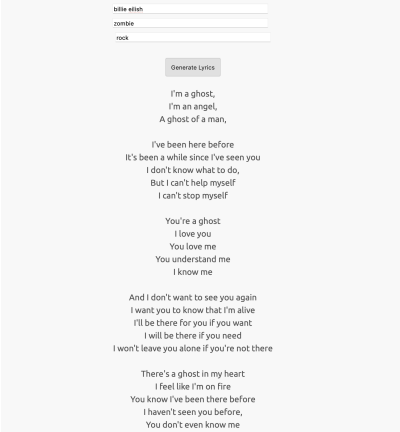

Figure 3. lyrics generated based on the input ”Billie eilish zombie rock”

in python and eliminating the need of setting up APIs between different languages.

3.3 Data preperation

In any machinelearning problem the solution is usually mostly based on the quality of the data at hand. To ensure that the model can produce good result an effort in ensuring that the model can train on quality data is of upmost importance. For this project a data set uploaded on kaggle was used. The dataset has a variety of lyrics from 79 music genres with over 250 000 data points. As previously described this was reduced to a subsample of 5000 data points that showcased the artist name, song name, genre and the lyrics[3].

3.4 finetuning

In the process of fine-tuning the model, we partition the dataset into two segments: the input segment and the target segment. The input segment is structured in a specific format comprising three key elements: the name of the lyrics, the genre, and the artist's name, arranged in the following pattern: ” $< lyricsname >< genre >< artistname >$ ”. This structured input serves as a prompt to the model during the training phase.

The objective of the training is to enable the model to generate the corresponding lyrics when provided with a string structured as described previously. This is achieved by training the model to map the input to the target lyrics in the dataset.

In essence, through this method of fine-tuning, we are training the BART model to recognize the relationship between specific key inputs and the corresponding lyrics. This enables the model to produce a particular song's lyrics based on a defined input format, thereby fine-tuning it to generate lyrics, deviating from the generalized structure of the model prior to fine-tuning.

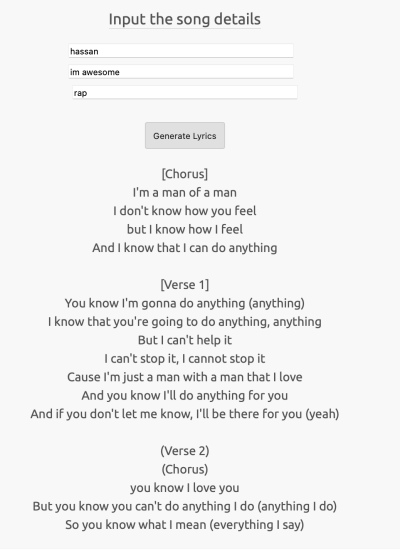

Figure 4. lyrics generated based on the input ”hassan im awesome rap”

Step	Training Loss	Validation Loss
500	2.060600	1.738168
1000	1.846600	1.678777
1500	1.719500	1.660916

Figure 5. lyrics generated based on the input ”hassan im awesome rap”

4. Result

When inputing a artist name, song name and a genre that the model has trained on we see that the songs generated have more substance and structure to them, see figure below that generates a song by billie eilish in the style of rock with the name of the song being zombie. we can see that there are some similarities with the song zombies by the cranberries. When using inputs that we have not trained on we can see that the model actually is able to create the structure of the lyrics but is struggling with building a meaningful context in the wording and we see a lot of repetition.

when looking at the loss validation in the training and validation stages we see that we are performing quite well for the size of the model.

5. Discussion

The results indicate that the model successfully replicates structural elements found in lyrics, such as understanding of choruses and verses. Even if the lyrics lack substantial coherence and value in its phrasing there seems to be some rhythm in the generated lyrics. This issue is likely due to the limited data sample. It is possible that there is a lack of training on each of the artists to learn the overall message in the lyrics.

Another reason could be the selection of input parameters. In this project, inputs like artist name, song name and genre was chosen with the expectation that the model would create songs mirroring specific artists and genre style. However it seems that we would require more data on each artist and genre to create something really valuable.

An improved methodology for a project of this scale could be to let the users provide emotional keywords along with the song title and genre. This adjustment might enable the model to produce enhanced outcomes. To be able to achieve this, it would be necessary to pre process the training data to include these keywords.

6. Conclusion

Although some improvements can be made due to some restrictions this project had to take upon itself, the project can be considered successful in choosing a good transformer model. The model is trained to be able to generate lyrics based on the inputs of artist name, song name and genre as proposed in the introduction.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[2] Hugging Face. Nlp course, 2023.

[3] ANDERSON NEISSE. Song lyrics from 79 musical genres, 2021.