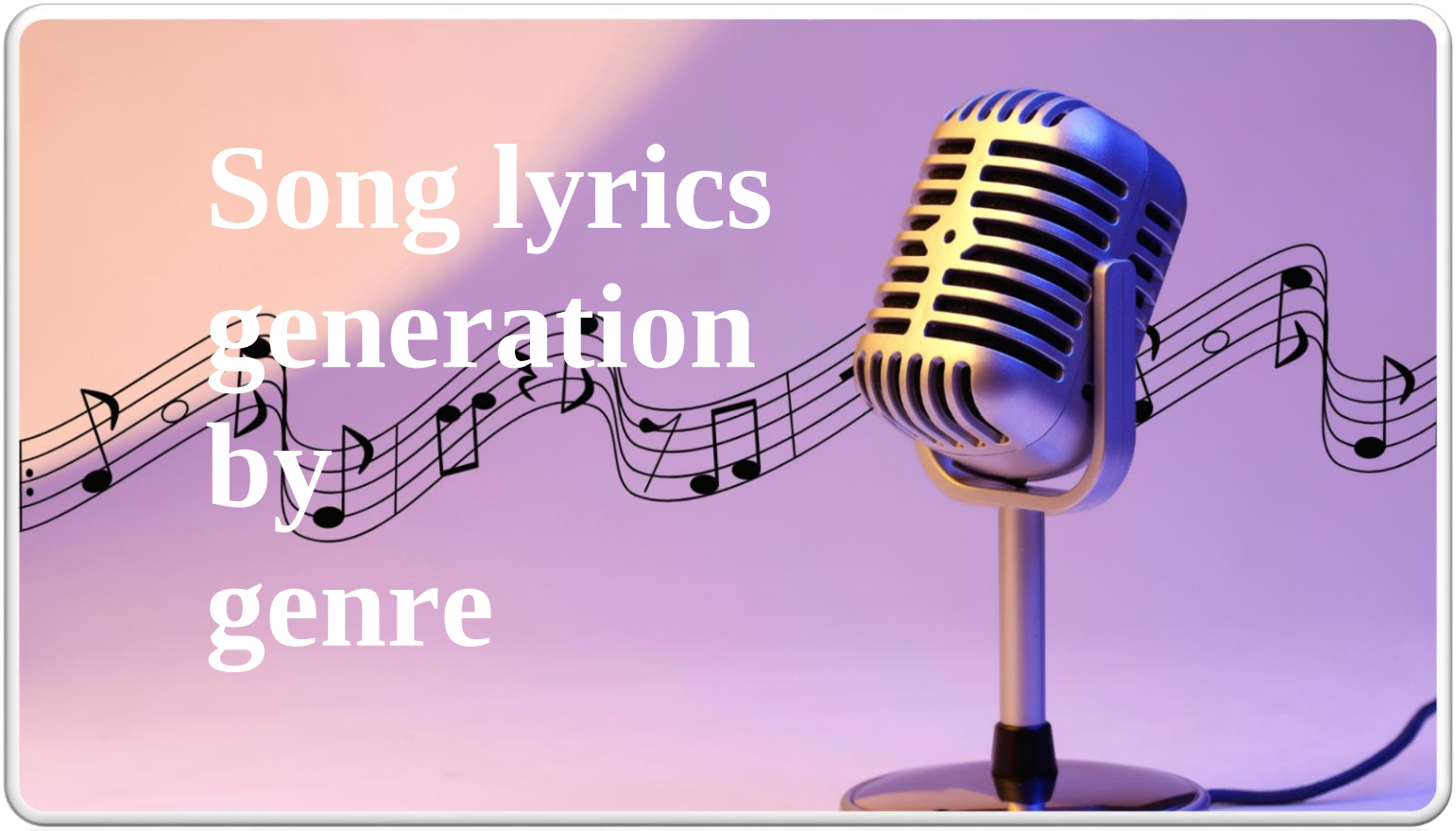


Deep learning project

Moran Danino

Introduction:

I chose the project “Song Lyrics Generation by Genre” with the approach of using GPT-2 fine-tuning. The purpose of the project is to build models that are able to generate song lyrics depending on genre (specifically, rap songs and pop songs). In this report, I will first describe and explain each code file that I wrote for the project and elaborate the many decisions I made over the course of the project as well. Lastly, I will provide full instructions on how to run all the code for the project.
All code for my project was written using the Python language in the Google Collab environment. In all the files I needed to use the function "High ram" in the Google Collab and in some I used GPU.

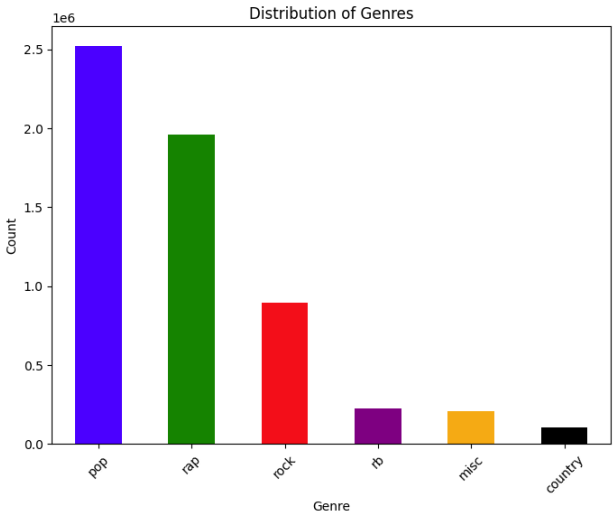
The Dataset:

I downloaded the original dataset from the Kaggle link in the projects list file provided to us. Dataset file name is “ds.csv”.

The original dataset contains 5,913,411 elements and 8 features: title, tag (genre of the songs), artist, year, views, feature (additional artists), lyrics (full lyrics of the entire song), id. The full size of the dataset is 9.21 GB.

Of the eight features, the relevant features for my project were only the tag (the song genre) and the lyrics.

Of the songs, 43% are pop songs, and 33% are rap songs. The remaining 24% is made up of other genres like rock, country, etc.:



I decided not to use the other dataset that was provided to us in the projects list file for a couple of reasons:

Firstly, the second dataset contains a negligible amount of data (only 60,000 samples, which is only about 1% of the data in the first dataset).

Secondly, the second dataset does not contain full songs – The elements in it are sentences that were taken from songs. I considered splitting the first dataset into sentences to match it, but doing so would erase valuable information from the model

and the resulting model would be significantly worse at generating consistent songs. Unlike other sequence generating models (like an LSTM), GPT models are capable of learning long-term word contexts. Because the model I chose is a GPT-2 model, I felt the model would be able to learn from the full song lyrics, and there was no reason to separate into sentences.

Preprocessing:

The dataset contained some empty ‘nan’ values in the Lyrics feature, so I filtered those out. Then, I deleted all songs that are in a different genre than pop or rap, and kept only the Genre and Lyrics features from the dataset.

Many of the song lyrics contained invalid comments inside of brackets. For example, in some songs, the artist’s name appeared in the lyrics as “[name]”, and other examples, so I deleted all words inside of brackets, curly brackets and parenthesis. Some songs contained lyrics and other kinds of comments, so I cleaned those from the songs as well. Some songs contained only "Loading…"/ "Save Cancel" so I delete those songs.

There were some songs in other languages in the dataset. Since I want a model that generates English lyrics only, I filtered out all songs that contained non-English letters.
Even after this filtering, I realized that there are still songs that are not in the English language, which are still in my data set because the letters are English letters. I tried to use language recognition libraries but since the data set is very large, the running time was very long.

Lastly, since I are interested in one model – I added the genre itself at the beginning of each song's lyrics: [Genre: Rap]/ [Genre: Rap] as an additional (new) token. I did this so I could create songs conditioned by genre later.

Dataset Splitting:

Tokenizing each song at training-runtime would take far too long and would be a waste of computation power, especially if I want to perform multiple training epochs, and thus would have to re-tokenize each song multiple times.

To deal with these issues, I decided to tokenize all the songs before training the models. This would not only save time with loading each batch, but would also make it easier to load the data into the memory, as token indexes take up less space than the string-typed lyrics themselves.

The resulting dataset was still massive even after all the pre-processing. While it was possible to load them into the memory, I found that tokenizing the datasets at once is an impossible task with the computation power I had at my disposal.

To deal with this issue, I decided to split the dataset into 40 equally sized dataset chunks, and save them in a fitting subfolder. I made sure that each part included at

least as many pop songs as rap songs because I planned to use down sampling later from the pop data to fit the number of rap songs.

Tokenization:

I load each of the 40 chunks to the memory one-at-a-time.

Before performing the tokenization, I performed down sampling. That is, I randomly selected from each chunk a number of pop songs according to the number of rap songs in the chunk, and I did this for each chunk out of the 40 chunks. I did this after filtering the amount of words in the song, which I will expand on later.

I then preformed tokenization using the pre-trained tokenizer of the GPT-2 model I are fine-tuning on each chunk.

The tokenizer returns a list of input ids and a list of attention masks for each song and the tokenizer folder I used.

I made sure that the tokenizer preserves the beginning of any song lyrics that indicate the genre [genre: pop/rap]. I did this by checking if it was in the tokenizer's vocabulary and if not then I added it.

I noticed at this time a few issues. Firstly, many of the songs in the dataset were not valid for song generation – some were only instrumental and contained little lyrics, some were incomplete and overall many songs contained only few words. Secondly, to load songs in batches at training I had to pad all the songs to a consistent length, however if I padded songs that were short too much I might cause damage to the resulting model's understanding. Lastly, if the resulting tokenized songs were too long, I would not be able to load batches of songs to train, and training time would end up taking far too long.

To deal with all those issues I decided that I would keep only tokenized songs with more than 20 tokens and less than 400. I then padded them to 300 tokens as well. The resulting data contained over 600,000 tokenized songs (total for pop and rap).

After tokenizing all the songs in each chunk, there was no longer a reason to keep the data in chunks. I saved a file that contained all the input id lists and a file that contained all the matching attention mask lists. (2 files)

Model Training:

I trained one model on the appropriate input id and attention mask files. Since the model has an enormous amount of data, I split the data into train-validation sets such that the validation set contained 0.005% of the data. This is because if I used the usual 90%-10% split I would end up with a training time of entire years, since validation is run many times over the entire validation set for each training epoch. To reduce training time spent on validation even further, I decided to run a validation loop once every 1000 training batches (and also on the first training batch for reference as well).

Then, I saved the data into a dataset class, and initialized a Dataloader over it. I used a batch size of 10 songs. This was the maximum batch size that could be loaded at once to the GPU I had without crashing according to my testing.

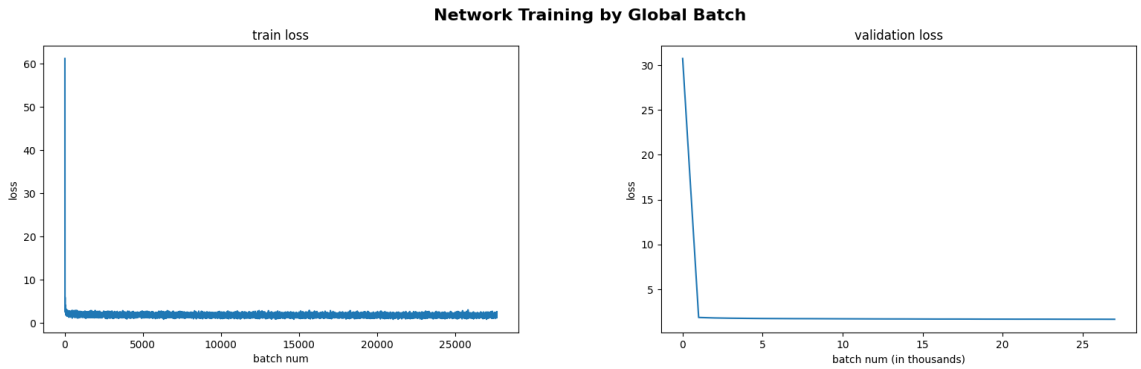
I loaded the pre-trained GPT-2 model from the Transformer AutoModelForCausalLM library.

I ended up with 27,685 training batches for the model.
The model took around 10 hours to fit.

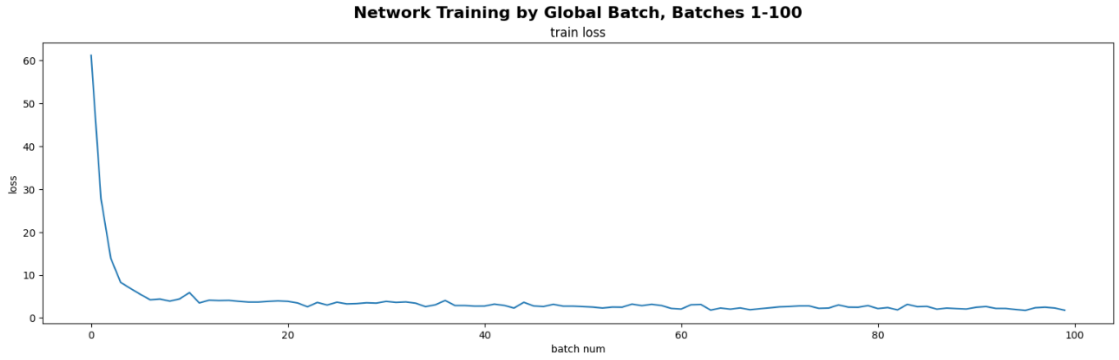
In training, I used the ADAM optimizer.

After training, I saved the model as a triplet that included a configuration json file, a generation-configuration json file and the model's state file itself in accordance with what the Transformer library needed to correctly load the model later.

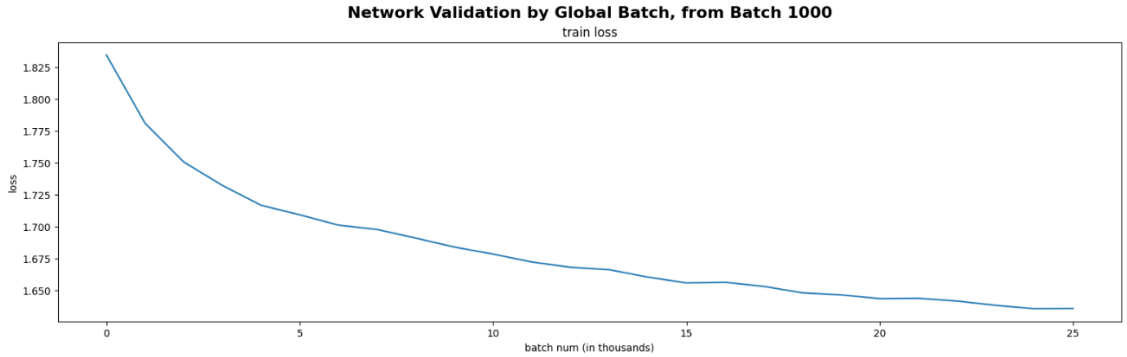
The calculated metrics that resulted from training were the train loss (per batch) and validation loss (per 1000 batches):



I noticed training loss converges to a small range that it keeps at in the first 100 batches. To show the learning effect, I generated an additional visualization of only the first 100 batches:



In addition to this, since the first point of the validation graph is the validation of the model before any training (in the first batch), I generated an additional visualization of the validation after the first 1000 batches have occurred to show the learning of the model in a closer viewpoint:



Song Generation:

I loaded the model, set the hyperparameters I want, and generated the tokens. I then pass the result through a short post-processing function to delete special characters (such as padding, start-of-sentence, end-of-sentence etc.). In addition, I added '\n' characters after commas, periods etc. to make the generated text fit the structure of a song. Finally, I print the results.

The parameters and hyperparameters of generation:

- **genre:** Can be "rap", "pop" or "both". If "rap" will generate only rap songs with the rap model, if "pop" will generate only pop songs with the pop models. If "both" will output both rap and pop songs using both the models.
- **num_songs:** Number of songs to generate for each genre.
  value: 2
- **max_length:** The maximum length the generated song will be. The song can still be shorter than this (if padding characters are outputted by the model itself).
  value: 150
- **temperature:** This hyperparameter controls randomness and creativity. Higher values cause the model to be more creative.
  value: 2.5
- **top_p:** The model will sort the probabilities in descending order, and keeps the most relevant words probabilities according to p. Higher values keep words that are more common.
  value: 0.8
- **top_k:** restricts the selection of tokens to the "k" most likely options based on their probability in accordance to top_p. Lower values will result in less variance in words.
  value: 50.
- **repetition_penalty:** Penalizes repetition. Higher values encourage the model to generate more diverse and non-repetitive output. Songs do have repetition in them, so important not to keep too high.
  value: 0.8

Results:

I printed two songs from each genre to get a general picture of the quality of the songs**.**

rap song generation:
song 1:

Lately,
You have kept away some friends who want

A different,
but you stayed away from her
In some distant and sometimes
In some way they are gone
That girl is your favorite boo,
she's my everything
The way you act
Just be careful to put your emotions away
No you'll need to look a

A couple who be
I could spend a life of me,
in a couple of us

song 2:

I been grinding my time is
i been working hard I been taking all
i know I been trying
i dont feel alone
i know its taking all
i know its taking on taking my pain away
i know my problems never stop
I guess it is hard getting to see the light yeah so so yeah
I wanna ride the world oh girl
i never would never doubt oh
i said no no no no yes me so me on ride
Hood yeah my diamonds green it glo
i got that
i ain never stop so it be the pain yeah im on on to ride yeah yeah
i just want some time away yeah my yeah
I been grinding yeah
I been waiting wait all week now
I got it yeah yeah yeah yeah and
im working yeah yeah

pop song generation:
song 1:

Hey,
can anybody care,
hey who wanna feel like who's so cold like them,

and who wants to feel just like nobody feels.
what happened in America this,
yeah the most crazy,
how crazy can't anybody really give,
and who'd have believed in the best in humanity to have ever tried to feel good,
so that's the most that can do it with nobody's so crazy about.
you really feel so cold.
who knows how big the big hole you live in just don't think about how many other people who've come to this it don't give that like it can't give to anyone that wants to feel great.
that is,
oh,

song 2:

This song can save me

We can sing but just because we won't know

Well if we go
So would you?
We won't make this hurt a pain
For I've got one of a sudden?
So we may just fall apart

I can see what is yours tonight
And you see what is mine tonight

My eyes can heal
My heart could see everything's still
Just another wasted breath inside
My head hurts.
I still think this shit out of the mind
So let's start to let's just sit down

Don't want this
Just want this You can start the fire.

And make your fire come to life in just one shot.

This song Can save

Evaluation:

I used perplexity as an evaluation measure.
Perplexity is a measure used in natural language processing and machine learning to evaluate the performance of language models. It measures how well the model predicts the next word or character based on the context provided by the previous words or characters.

I calculated the perplexity value between the validation set and my model and compared the result to the value I got between the validation set and the gpt-2 model without fine-tuning.

The results I got were clear, the perplexity value was smaller and significantly better in my model in which I performed fine tuning than the model without fine tuning. I did expect to get this result.

In addition to this, the value I received for my model is relatively low and shows a consistent and good model.

```
Average perplexity of untuned model over validation dataet: 5.676198831197441e+51
Average perplexity of tuned model over validation dataet: 8.476248169339895
```

Conclusion:

In conclusion, since the pop genre had more correct and reliable information, it can be seen that the model was indeed able to create good pop songs.

In pop, for example, I sometimes see rhymes between the lines, and in rap I see the use of words that do repeat themselves in rap songs.

I are overall satisfied with the result.

Running the model:

Instructions for running the model:

My project contains the following files:

- “preprocessing.ipynb” - code file that reads the original data and does preprocessing.
  input: “ds.csv”- The original dataset
  output: "ds_clean.csv"- dataset file after preprocessing.

- "SplitData.ipynb"- splits the dataset into 40 chunks.
  input: "ds_clean.csv"
  output: 40 chunks of the dataset.
  path file: chunks → "chunk_i" (i from 1 to 40).

- "Tokenizer.ipynb": iterates over all the 40 chunks files, does tokenization after down sampling.
  input: 40 chunks of the datadet.
  output: two files- "all_input_ids.npy", "all_attention_masks.npy", saving tokenizer.
  path file: splits → "all_input_ids.npy", "all_attention_masks.npy" / tokenizer folder.

- "Training.ipynb" –
  input: "all_input_ids.npy", "all_attention_masks.npy", tokenizer folder.
  output: three files: "model.safetensors", "generation_config.json", "config.json".
  path file: **model**→"model.safetensors", "generation_config.json", "config.json".

- "Generarion.ipynb":
  input: three files: "model.safetensors", "generation_config.json", "config.json".
  output: Prints song lyrics according to the genre (rap/pop/both).

- "PerplexityComparison.ipynb":
  input: model folder: "model.safetensors", "generation_config.json", "config.json", tokenizer folder, and: "all_input_ids.npy", "all_attention_masks.npy".
  output: comparison of perplexity values between my model and the untuned model