OK GPT-2: write me a melancholic song
Conditional lyrics generation with fine-tuned GPT-2 and PPLM

Alessandro Rosa

November 2021

Abstract

Natural Language Generation (NLG) is a sub-field of Natural Language Processing (NLP) in which the main task consists of generating samples of text from language models. Despite major advances in the field of NLP, generating texts conditioned under a specific attribute is still a rather complicated issue. The usual approach is limited to fine-tuning a pre-trained model with a specific corpus in order to get an output shaped on the training dataset. Plug and Play Language Model aims at stirring the generation of samples of text by using specific attributes, without re-doing the training each time. This work explores the possibilities of generating musical lyrics by fine-tuning a GPT-2 model and using PPLM to condition the generation of text. The results are only partially satisfactory: fine-tuning GPT-2 to obtain samples of lyrics proved to be an easy task, while using the trained model with PPLM shows some inconsistency between the bag-of-words approach and the classifiers one. Code and dataset are available at the Git-Hub repository.

1 Introduction

The task of creating samples of text based on a certain language using machine learning models is known as Natural Language Generation (NLG). The basic idea behind this task is that a language model, i.e. a model that has a probabilistic representation of a language (e.g. English) would be able to create a sentence simply by choosing the sequence of tokens that has the higher probability. In other terms, the language model aims at create a sample by computing the product of the probabilities that a specific sequence of tokens can take place, given a certain input. NLG is a very fascinating field of Natural Language Processing (NLP), as it tends to blur the distinction between real-human text and machine-generated text.

State-of-the-art models that perform NLG are usually deep learning systems based on the transformer architecture, which at its core includes an encoder, a decoder and an attention mechanism of some sort. One famous example of the transformer architecture is GPT-2, developed by Open AI, and currently one of the best language model available. Thanks to the large amount of data on which it was trained, GPT-2 is able to successfully perform many important tasks for NLP, such as text classification, questions answering, text summarization and, of course, text generation.

However, even with state-of-the-art models it is hard to produce outputs that satisfy specific attributes given by the user. Conditional text generation is indeed a rather difficult task and usually involves some kind of specific fine-tuning for obtaining more specific and tailored results. One possible solution is to implement a sort of plug and play model, such as the one recently developed by Uber AI. This allows to gain control of the sequence generation algorithm and producing outputs that are coherent with a specific topic without loosing too much fluency.

This work explores the possibilities given by fine-tuning a pre-trained model and using a Plug and Play Language Model to drive the text generation. The goal is to produce musical lyrics that resemble the one written by human artists and generating samples based on specific topics or a certain sentiment score (positive or negative). The following sections tackle the problem dividing it three parts: 1) analysing the context of the problem and the available models for the task; 2) explaining in detail the methodology to fine-tune a GPT-2 model that is able to output musical lyrics and using such new model to run it on a Plug and Play interface; 3) evaluating the results and drawing conclusions. Since evaluation of NLG samples is a very hard task, the quality of the fine-tuned lyrics are going to be evaluated by human judgement, taking into consideration three aspects: a) the structure of the lyrics b) the syntactical correctness of the each verse 3) the appeal of the lyrics. For the conditional generated sample, the samples are going to be evaluated differently depending on the discriminator: for the topic-based generation, the evaluation will highlight the presence of keywords related to the topic; for the sentiment classifier, VADER is going to be used to check the general compound score of the lyrics.

2 Context

2.1 Transformers and GPT-2

Recursive neural networks (RNN) were implemented in language modelling tasks with success since the beginning of the 2010s [1], mostly because they are an efficient solution to a huge problem in NLP: taking into account the context of a token. We know that natural language relies on the fact that words are not completely unrelated one with each other and that the meaning of a sentence can hardly be fully understood by analysing each word by its own. The problem is also more urgent if we consider the role of pronouns such as *it* in sentences as ”machine learning is not entirely a brand-new field of computer science, but *it* has seen major improvements over the last decade”. For human readers it is clear that *it* refers to *machine learning*, but for a machine learning system this is not obvious as well.

The most basic implementation of a RNN consists of an input layer, a hidden state (or context state) and and output layer. The hidden state is the true core of the model, as it is able to grasp the context around a word to compute the probabilities of the next sequence of tokens. RNN were implemented in many tasks that require predicting the next token in a specific language context (text generation, machine translation, abstract text summarisation) but training RNN can be very expensive when the amount of data start to increase significantly and their accuracy drastically decrease when as the sentence to output gets longer[2].

To overcome this problem, Vaswani et. al.[3] presented in a seminal paper titled *Attention Is All You Need* the transformer architecture. Transformers are encoder-decoder models (also known as sequence to sequence models) that only rely on the self-attention mechanism[1], which is the component able to take into consideration the context of a word. What follows is a high-level explanation of these three main components of a transformer: the encoder, the decoder and the attention mechanism.

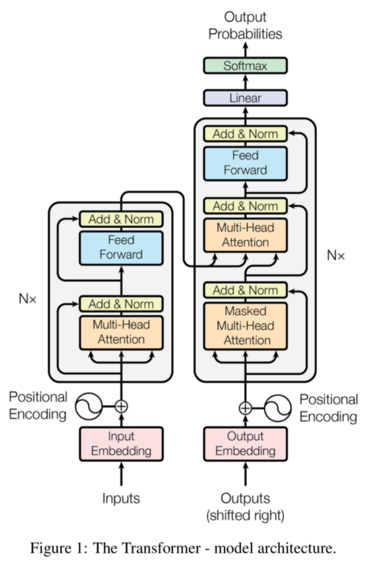

Figure 1: Vaswani et. al., 2017

The encoder has the role of taking the input sequence (usually a string of raw text) and convert it into a feature of vectors, which number of dimensions is defined by the specifics of the architecture. Each value of the vector represents a single word and the immediate context of the word (i.e. the words the surround it). First, the vectors pass through a self-attention layer and then through a feed forward neural network. For each word, three vectors are created: a Query, a Key and a Value vector. These vectors are calculated using the dot product with three respective matrices and with the initial embedding of the word in the input sentence. This leads to a final vector which represents the word and its context that has to be fed to the feed forward neural network. Additionally, in

[1]They also rely on positional encoding, fully-connected neural networks and other components; but that would not have been a title as catchy as the one proposed by the authors

order to account for the order in which the words are positioned in the input sentence, a positional vector added to each initial embedding.

The structure of the decoder is very similar to the one's of the encoder. Its goal is to take the feature vectors that were outputted by the encoder and emit a sequence of symbols step by step. The important difference is in the self-attention mechanism: the decoder is allowed only to see the left part of the input sentence, while the right part (which is, the part that represents future positions of the sentence) is masked. Since the decoder is unable to "peak" in to the rest of the sentence, it is forced to guess what the next tokens will be. The model is said to be auto-regressive, i.e. it reuses its own output as an input for the next part of the sequence. In this way it can use each generated token to calculate the probabilities of a specific word being the next token in the sentence.

This architecture allows to significantly reducing complexity and thus training the language model on huge amount of raw data. Open AI GPT-2[4] is one of the most powerful language model based on the transformer architecture[2]. It has been trained over 8 million web pages with the goal of predicting the text word given a specific input of tokens as a context. With a training set consisting of 40 GB of raw text and 1.5 billion parameters, it is able to achieve outstanding performances in a wide range of tasks, reproducing human-like samples of text. The potential of GPT-2 in creating fake texts has even raised some concerns for the malicious use that could be made[5].

GPT-2 is built by stacking decoder blocks, the size depending on the specific model (the largest has a dimensionality equal to 1600). Since it is using only the decoder part of the transformer architecture it is able to submit only one output at the time, that is, only one token at the time. More importantly, the self-attention layer is actually a masked self-attention layer. This means that part of the sequence is masked in order to prevent the attention mechanism to see the entire sentence that has to be outputted. With this structure, GPT-2 has learned how to correctly predict the next token given a certain input, and starting from that adding tokens that are coherent with the previous ones.

2.2 Conditional text generation and PPLM

GPT-2 as it is can already produce samples of text that are coherent, semantically meaningful and syntactically correct. It is also possible to prompt some input sequence from which starting the generation. However, it is not possible to actually control the flow of the text generation, for instance asking to cover a specific topic. By fine-tuning GPT-2 with a focused data set is possible to adapt the language model to specific task, for instance training it to write poetry or Amazon's reviews. This kind of fine-tuning is usually referred to as transfer learning[6]: a language model that already has a good stochastic understanding of a language can be trained to learn how to deal with a more specific task, keeping the original knowledge and enhancing it with the new data.

This approached allowed to create models able to tackle sub-domain of NLG, for instance generating patent claims that resemble the real-world ones [7]. Another more sophisticated approached involves the use of a classifier in phase of training in order to perform reinforcement training[8]: the model is rewarded each time the classifier gives a positive result and it is punished when it does not. However, fine-tuning on its own is quite a limited approach. It allows to significantly change the behaviour of a model but it also restricts the flexibility of the output. When it comes to tasks such as text generation we would like to have a model able to easily switch from a thematic area to another, similarity as humans do in their daily life. The Plug and Play Language Model (PPLM)[9] developed by Uber AI aims driving the text generation using a different approach that does not involve to train the model each time we want a different theme covered.

The basic idea behind PPLM is pretty straightforward: instead of using plain fine-tuning or reinforcement learning to create a very specific model, a general pre-trained language model could be influenced post-training to reach a more specific output. At the simplest level, we can formalize $p(x)$ as the unconditional probability of generating a fluent sample of text, which is exactly what a standard language models such as GPT-2 does. Ideally, we could calculate a conditional probability given an attribute a if we compute $p(x|a)$, for instance the probability that the sentence x will regard politics or sports. However, we often do not have such probability. What we can do is exploit the Bayesian theorem and calculate $p(a|x)$, which is the probability that a sentence$x$ posses the attribute a (i.e. the probability that x can be classified as politics), in this way: $p(x|a) = p(a|x) * p(x)$. So in plain theory it is possible to build a system that generates an unconditional sentence, it calculates whether the output belongs to a topic and if not starts again the sampling without any constrains.

[2]During the draft of this paper, Open AI announced that GPT-3 will slowly start to be available to the general public. The new model has been trained with an even huger amount of parameters, and the first results look rather impressive.

Such an approach, however, would potentially take forever due to the infinite combinations of samples that a language model such as GPT-2 is able to generate. The idea of PPLM's authors is to exploit techniques that were already implemented for conditional images generation to language, which mainly involves changing model behaviour when it has to compute the probability distribution of a sentence. The huge advantage is that this is made without any additional training, as the process is applied only when the model is asked to generate a sample. The attributes may be related to a bag-of-words model, i.e. a list of terms that are related to a specific topics (for instance, "faith", "Jesus", "angel" are related to the topic "religion") or to a classifier model (in this case, a sentiment score classifier).

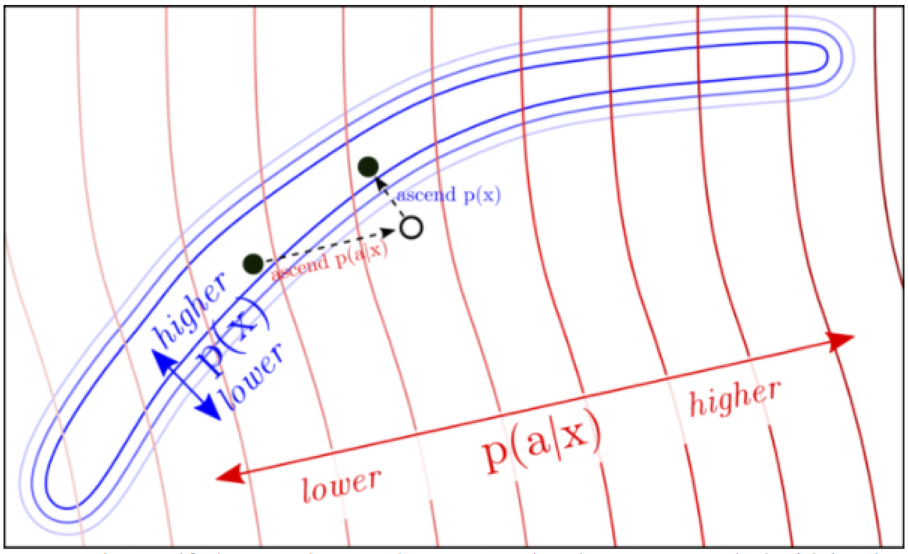

Figure 3. This simplified cartoon diagram why steps must be taken to maximize both p(a|x) and p(x) en-route to generating PPLM samples. The sentence under consideration is shown as a black dot, which is first pushed in the direction of maximizing p(a|x) and then in the direction of maximizing p(x). In practice, rather than two separate steps, gradients of both terms are combined to compute the single step corresponding to the net displacement.

Figure 2: Dathathri et. al., 2019

With PPLM, text generation can be split into three main steps: first a forward pass is made to the original probabilistic distribution of the language model; then a backward pass is made to update the latent representation of the language model using gradient from the attribute to increase the likelihood of the passes towards the requested attributes. Finally, the new distribution is computed and a new token is generated. For each partially generated sentence, both $log(p(x))$ and $log(p(a|x)$ and the relative gradients are computed with respect to the hidden representation of the underlying language model. The gradients are the tools for stirring the model to the desired output and they are both needed: $log(p(a|x)$ is the probability of the sentence having the desired topic, while $log(p(x))$ represents the original probability distribution (which account for the original fluency of the language model). Intuitively, we do not want that the sample creates sentences that contains many words related to the desired topic, but which are syntactically or semantically wrong.

3 Lyrics generation

Generating lyrics is not entirely a new task in NLG. There are many interesting examples in the literature able to sample meaningful and consistent texts that can also include rhymes[10]. The main difference with plain text generation is the structure of the output: musical lyrics are usually divided into verses, they contain choruses, outros, bridges and so on. From the point of view of the semantics, they might be a little less strict than plan text (for instance, a reasonable amount of repetitions between lines is not necessary a bad thing) but words should be at least coherent with the rest of the verse.

Despite being incredibly powerful, GPT-2 by its own cannot be used to generate lyrics. In order to do so, it is necessary to train GPT-2 with a specific corpus that represents the target output (in this case, musical lyrics). As mentioned previously, transfer learning allows to overcome this problem. This allows to avoid large training sessions, which are computationally expensive and may take entire weeks for reaching acceptable results. It is also important to notice that such large training phases tend to consume a enormous amount of energy, being both economically costly and harmful for the environment. This section provides details about the experiment carried out by fine-tuning GPT-2 and inject the new model into PPLM.

3.1 Dataset

In order to fine-tune a language model it is necessary to prepare an appropriate dataset on which performing the training. To gather a musical lyrics dataset, the Genius API were implemented in a Python notebook ran on a Google Colab environment. The Genius API allows to retrieve potentially all the lyrics associated with an artist that have been uploaded on the website and store them into a .json file along with all their metadata. For this project, 6041 songs were extracted using this method, consisting of lyrics wrote by the following list of artists: Alice in Chains, American Football, Arcade Fire, Bauhaus, Beach House, Blur, Built to Spill, Car Seat Headrest, Carissa's Wierd, Counting Crows, Deerhunter, Delta Sleep, Dinosaur Jr., Empire! Empire! (I was A Lonely Estate), Have a Nice Life, Interpol, Joy Division, La Dispute, Manic Street Preachers, Mineral, Modest Mouse, Mount Eerie, Muse, Neutral Milk Hotel, Nick Cave & The Bad Seeds, Nine Inch Nails, Nirvana, Pavement, Pixies, Queens of the Stone Age, Radiohead, Red House Painters, Silver Jews, Slowdive, Sonic Youth, Sparklehorse, Spiritualized, Sunny Day Real Estate, Swans, TTNG, The Appleseed Cast, The Beach Boys, The Beatles, The Black Heart Procession, The Brave Little Abacus, The Cure, The Hotelier, The Jesus and Mary Chain, The Microphones, The National, The Smashing Pumpkins, The Smiths, The Strokes, The Velvet Underground, The World is a Beautiful Place & I am No Longer Afraid to Die, Tiny Moving Parts, Tool, Unwound, Wilco, Yo La Tengo, mewithoutYou.

Fig. 3 displays the amount of lyrics for each artist that were included in the dataset.

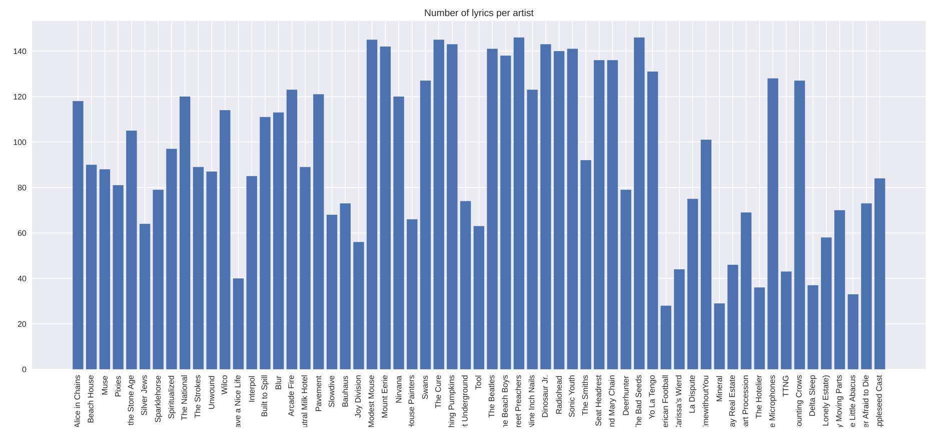

Figure 3: Lyrics per artist

Little to none pre-processing was needed for feeding the dataset into the training script, as GPT-2 comes with its own tokenizer that is able to pre-process the raw text (cf. next sub-section). Since lyrics on Genius website are manually inserted by users, it is possible that they contain mistakes and that the notation is not entirely homogeneous. Overall, this should not lead to problems in the training phase, as accidental mistakes are rare and they do not confuse the model. The only issue that was evident during training experiments was the presence of too many new-lines: for this reason, more than two consecutive new-lines were removed before loading the dataset. Moreover, songs that were remix, live sessions, demo versions and other variations of the original songs were discarded to avoid duplicates.

In order to properly load the dataset in the GPT-2 tokenizer, the lyrics contained in the .json file were extracted and loaded in a single .csv file. Hugging face dataloader is able also to directly work with .json or .txt files, but to better preserve the distinction between each song the .csv was the best choice. Then, using the scikit-learn library the dataset was shuffled and split in a training set and a validation set using with a proportion of 80-20 . The training set contained 4833 songs while the validation set 1208.

3.2 Fine-tuning GPT-2

GPT-2 was used through the Hugging Face library[11], in order to have easy-to-use access to the language model. Hugging Face is an online community that allow to share pre-trained models and re-utilize them for various tasks, calling each of them simply by their own web interface or including in a Python script. With Hugging Face API it is possible to access three fundamental parts of a pre-trained language model: the tokenizer, the actual model with its parameters and weights; and one or more heads to apply to the model in order to perform different tasks.

The raw dataset was passed through the GPT-2 tokenizer, which has the function of splitting the text into tokens and convert them into numerical ids. The tokenizer works at the sub-word level, meaning that it splits a sentence not only into words, but also into sub-parts of those words. For instance, the word ”go” would be tokenized as ”go”, but the word ”going” would be split into two tokens ”go” and ”#ing”, where ”#” is a special character used by the tokenizer to know that a specific token was connected with the previous one. This allows to take into consideration not only a specific word but also their variation, such as different verb tens, plural forms of nouns and so on. GPT-2 also requires batches of text of the same exact length. This could be problematic when we have samples of different length, which is usually the case. Thus, it is necessary to introduce pad tokens, i.e. tokens that fill batches until they reach the maximum length of the corpus. In order to distinguish proper tokens from pad tokens, to each id it is attributed a binary mask (1 or 0), that indicates whether the id represents a proper token or a pad token. The tokenizer is then able to distinguish between tokens that have to be considered for the training and those who are not.

The tokenizer already knows how to split the documents (the songs, in our case) using a pre-defined special tokens (i.e. `<|endoftext|>`). By default, GPT-2 uses this special token as start of sequence, end of sequence and for padding. It is in line of principle possible to add special tokens in order to differentiate start of sequences from end of sequences or to have a custom padding token, for instance putting `<|startoftext|>` and `<|pad|>` tokens to the model. However, PPLM is currently unable to take into consideration variations in the default vocabulary of the model (which, in the case of GPT-2 has a total of 50257 tokens), leading to errors when running the code. For this reason, the original vocabulary was kept without any modification.

After setting up the dataset and transformed it into tensors, the next step involves the actual training the model. In order to find the best hyper-parameters for the corpus, several attempts were made. The final configuration can be see in the code snippet below. Since the free version of Google Colab has been used, the quality and availability of the hardware was not stable throughout the whole process. The training was divided in multiple phases by reusing saved checkpoints each 500 steps. Despite the GPU model changed from each session, at least 12 GB of memory video were always available. This allowed to set up a reasonable batch size (24), which was artificially increased using the gradient ascending technique: instead of updating the weights after each iteration, the trainer keeps storing the weights for x steps before updating the model (in this case x=4). The following snippet of code illustrates the hyper-parameters setting. Moreover, the GPT-2 model was configured with a residual dropout of 0.7 (default is 0.1) to help the model avoiding overfitting.

Listing 1: hyper-parameters for training

```
from transformers import TrainingArguments

training_args = TrainingArguments("path",
    overwrite_output_dir = True,
    num_train_epochs = 10,
    warmup_steps = 200,
    per_device_train_batch_size=24,
    per_device_eval_batch_size=24,
    evaluation_strategy = "steps",
    logging_steps = 250,
    save_steps = 500,
    save_total_limit = 5,
    gradient_accumulation_steps = 4,
    learning_rate = 5e-4,
    weight_decay = 0.4

)
```

The training took 3860 steps over 10 epochs for a total time of about 10 hours of computation. The resulting loss curves for both training and evaluation can be seen in fig. 4.

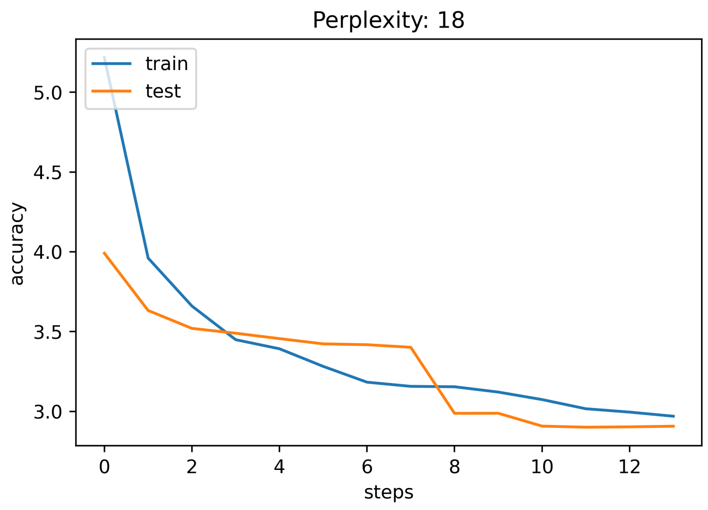

Figure 4: Training curves

3.3 Plug and Play Language Model

The Plug and Play Language Model can be retrieved from the Git-Hub repository where the authors uploaded the code. PPLM comes with a Python notebook as an example that can be used for further interaction and an Hugging Face interface (which is not available anymore at the current time).

After cloning the repository, a Google Colab environment was set up to launch the script. By default, PPLM uses the medium size model of GPT-2. For carrying out this experiment, default model was switched to the fine-tuned model that was trained on the musical lyrics dataset. The script allows to choose the discriminator on which the model will be stirred along with a bunch of settings to properly tune the model. The repository also provides some topics in the form of bag-of-words (textual files with one word per line) that were used as examples for running the conditional text generation starting from an input sequence.

Running the PPLM script on CPU is computationally expensive and requires some time. The parameters that influence the computational time are mainly the number of tokens that have to be generated, the number of samples to generate and the number of iterations to make (i.e. time the model spends in choosing a token). Average time to obtain three samples from a single run with the bag-of-words classifier was about 10-15 minutes for a text made of 100 tokens with n_iteration = 7. using the classifier methods, the computational time increase to 45-50 minutes. In order to avoid loops (e.g. "SCIENCE of SCIENCE of SCIENCE of SCIENCE"), authors recommend to adjust the step size, which define the amount of control over the topics, and two coefficient of the algorithm (the kl-loss coefficient and the gm-scaling term). While they do not mention temperature in their study, considering that our model seems to better perform with higher values than default ones, all the experiments have been carried out with a temperature in the range of 1.2 to 1.4. Below you can find a sample of parameters.

Listing 2: hyper-parameters for PPLM

```
from run_pplm import run_pplm_example
run_pplm_example(
    pretrained_model="modelpath",
    cond_text="[verse 1]",
    num_samples=3,
    bag_of_words='.txtpath',
    length=100,
    stepsize=0.04,
    sample=True,
    num_iterations=7,
    window_length=5,
    gamma=1.5,
    gm_scale=0.90,
    kl_scale=0.01,
    verbosity='regular',
    seed = n,
    temperature = 1.3
)
```

In order to adapt the model to the specific king of text generation that we want, some custom bag-of-words files were created. Three of them are terms extracted by the NRC Word-Emotion Association Lexicon and which represents three main emotions (anger, sadness and joy). The other one was crafted using the same method of the authors, i.e. exploiting the list of words in this website and it refers to autumnal season.The goal would be then to create lyrics that refer to different moods and with some specific keywords in mind. Among the two classifiers available, the sentiment score one appeared to be the most suitable for the task. Thus, the click-bait discriminator was not implemented.

4 Results

Evaluating natural language generation is quite a difficult task[12]. Overall, in NLP there are two main families of evaluation methods: human-centered methods and automatic ones. The drawbacks of the first category are evident, since evaluating a long series of results takes an enormous amount of time and energy. Moreover, humans' evaluation is usually not bias-free and the same output could be evaluated differently by different people. Automatic metrics, on the other hand, aim at overcome those difficulties, but their application is much more difficult because it is not always clear how to formalize the desired output. Should the sample generated be as close as possible to the real similar texts? How can we evaluate fluency, grammar correctness and semantic at the same time in an automatic way?

BLEU (BiLingual Evaluation Understudy)[13] it is usually a popular choice for evaluating NLP tasks. It was originally developed for machine translation but it is basically the most common metric used in the field for many different tasks. The key idea is to compare a list of possible desired outputs with the outputs given by the model, comparing the similarity by checking the length of the text and the presence of matching n-grams, finally computing a single score to compare with other generated samples. However, BLEU was made for machine translation, a field in which such an approach is much more suitable. The fact that it heavily relies on the gold-standard reference text for the evaluation might be problematical when it comes to NLG. It is very hard to evaluate musical lyrics on the sole base of an already existent text to take as reference.

One can also evaluate the result of the model by its own, for instance considering how much the model is able to minimize the loss function. Looking at fig.4, it is it possible to claim that the model is progressively learning from the dataset without overfitting, as the two functions tend to converge. GPT-2 is an incredibly large model with millions of parameters, thus it is very easy to make the model overfits when it has to train from a relatively small dataset. Additionally to the loss trajectory, the final perplexity value is computed (approximately 18). Perplexity is defined as the exponentiated average negative log-likelihood of a sequence and intuitively it gives us an idea of how much the model is unsure about the next token to be generate. A higher perplexity means more uncertainty, while a lower one means that the model is more confident with its choices.

However, considering the loss function is not sufficient to evaluate whether the model is able to create interesting text samples. The best way to to so is by looking at the possible outputs performed by the model (cf. Appendix 1). The model looks like is able to generate samples of lyrics that resembles those written by humans, despite some minor incorrectness, both at the grammatical and at the semantic level. It is interesting to notice that by lowering the temperature, the samples are more redundant, but they also tend to maximise the structure divided in verses; on the other hand, higher temperature means more freedom in the next token choice, leading to more poetic results, but risking to lose the separation in parts. From the experiments carried out, the ideal temperature ranges from 1.2 to 1.4, which is actually higher than the basic one for GPT-2 (default value is 1.0). The reason may lay in the fact that the model tends to adhere to strictly to the stricture of the song; a higher temperature allows the model to propose more daring verse, behaving more realistically and with less repetitions, without completely loosing the ability of sampling coherent and fluent text in English.

The samples generated with t=1.2 and t=1.4 clearly respects the goals mentioned in the introduction: the division is verses and chorus resemble those of real-world songs (`[Verse 1]`was used as the input for all the samples); each line has a semantic meaning that it is correlated with the other lines and the syntax is generally followed; despite some repetition, which usually comes with the chorus and thus are actually well-accepted as an output, the lines are interesting enough to spark the curiosity of the reader (or the listener).

Obtaining results with PPLM similar to those reported by the original paper has been more difficult. The default values suggested by the authors did not lead to satisfactory results, either for not stirring enough the text generation on the desired topic or including too many repetition of the same line or word. The fluency of the samples appeared to be generally worse than the fluency of the un-conditioned ones. Choosing a specific topic in the form of bag-of-words plays a dramatic role in the quality of the sample (cf. Appendix 2). With the same parameters, some topics allowed to generate reasonable texts that more or less conserve the fluency and are conditioned by the theme (e.g. `joy`), while others (e.g. `anger` or `kitchen`) either are not being conditioned or they end up in looping through the same word.

Some common mistakes appear when the model uses the words contained in the document, but with a different semantic purpose: for instance, `can` when extracted from the `kitchen` corpus is clearly intended as a noun that identifies a closed metal containers for food or drinks, but the model tends to use the term as the verbal form. Overall, it seems that the topics provided by authors performed slightly worse than the ones created for the task. The underlying reason might be that the domain of musical lyrics is more suitable for emotive themes (such as sadness or joy) and less to go towards legal or scientific topics. The case of `religion` is interesting: there is no word that is taken directly from the .txt file, but the lyrics contain words such as `soul`, which is linked to the religion sphere. This is an interesting phenomenon that has been also noted by the authors and it proves that the model is not only going towards that specific words, but more in general towards the topic.

The sentiment discriminator appears to work better. VADER has been used as a tool for checking the compound score of each sample, i.e. the ratio between the negative, the neutral and the positive score given by the NLTK package. The value ranges from -1 (max negativity) to +1 (max positivity). VADER has been trained mostly on social media texts, thus the positivity/negativity has been tuned on that kind of language (i.e. swearing, hateful speech and so on) so rather then representing the "mood" of the text, it mostly focuses on how much the samples are "aggressive" or not. While "very positive" and "very negative" seems to respond quite well, negative and positive discriminators are a little bit less efficient. For instance, a sample tuned with positive scored an almost identical compound as the very positive one.

Both with the unconditional generation and the PPLM one is important to stress out that the input sequence changes significantly the results. The examples provided here always start with `[verse 1]` as input because it is mostly common for a song of starting off with the first verse. However, there are songs that are not divided in verses, songs that start with intro and so on. By using the start of sequence token provided by GPT-2 (which is `<|endoftext|>`), the model performs in less predictable way.

5 Conclusion

The aim of this work was to allow the generation of musical lyrics under the condition of a specific attribute without loosing too much fluency. Fine-tuning a GPT-2 model is the easiest way to obtain a language model able to produce samples that are similar to lyrics written by real artists and it can be done with a relative small dataset. However, with GPT-2 on its own changing arbitrary the topic of the lyrics is not possible. PPLM allows to influence the text generation with a bag-of-words attribute or with a pre-trained discriminator such as a sentiment classifier. When dealing with a fine-tuned GPT-2, PPLM works slightly worse than the original language model. This might be due to the fact that fine-tuning GPT-2 has a smaller vocabulary in respect to the ones of the general GPT-2 and thus it is less prone to change the distributional probabilities that make up for the text generation. It might also due to the structural properties of songs lyrics: the way in which each verse is written is rather different by writing a plain text (the task on which GPT-2 was trained). Nevertheless, the goal of this project was to show that such an approach can lead to interesting and curios results, useful for artists that want to experiment with NLG for their works and also as a general pipeline for researchers that want to create language models able to tackle more specific task than just plain text generation.

References

[1] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. *Interspeech*, 2(3):1045–1048, 2010.

[2] M Onat Topal, Anil Bas, and Imke van Heerden. Exploring transformers in natural language generation: Gpt, bert, and xlnet. *arXiv preprint arXiv:2102.08036*, 2021.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, pages 5998–6008, 2017.

[4] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[5] Alec RadfordJeffrey WuDario AmodeiDaniela AmodeiJack ClarkMiles BrundageIlya Sutskever. Better language models and their implications. *Open AI blog*, 2019.

[6] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2020.

[7] Jieh-Sheng Lee and Jieh Hsiang. Patent claim generation by fine-tuning openai gpt-2. *World Patent Information*, 62:101983, 2020.

[8] Xiangyu Peng, Siyan Li, Spencer Frazier, and Mark Riedl. Fine-tuning a transformer-based language model to avoid generating non-normative text. *arXiv preprint arXiv:2001.08764*, 2020.

[9] Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. Plug and play language models: A simple approach to controlled text generation. *arXiv preprint arXiv:1912.02164*, 2019.

[10] Eric Malmi, Pyry Takala, Hannu Toivonen, Tapani Raiko, and Aristides Gionis. Dopelearning: A computational approach to rap lyrics generation. *Association for Computing Machinery*, page 195–204, 2016.

[11] Thomas Wolf, Julien Chaumond, Lysandre Debut, Victor Sanh, Clement Delangue, Anthony Moi, Pierric Cistac, Morgan Funtowicz, Joe Davison, Sam Shleifer, et al. Transformers: State-of-the-art natural language processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, 2020.

[12] Asli Celikyilmaz, Elizabeth Clark, and Jianfeng Gao. Evaluation of text generation: A survey. *CoRR*, abs/2006.14799, 2020.

[13] Ehud Reiter. A structured review of the validity of bleu. *Computational Linguistics*, 44(3):393–401, 2018.

6 Appendix 1: Fine-Tuned GPT-2

Table 1: Fine-tuned GPT-2 with different temperature

Temperature	**Text**
0.8	[verse 1] i remember someone who spoke in tongues i remember when, dear mary said "i never loved you." and the day i met her said "i never loved you and the day i met her "i never loved you." and the day i met her said "i never loved you" and i never loved you and the day i met her said "i never loved you" [chorus] and you said "i never loved you and the day i met her and you said "i never loved you" and you said "i never loved you and the day i met her and you said "i never loved you"
1.0	[verse 1] i'm going to heaven to be alright i'm going to heaven to be alright my kingdom was always devout all i ever wanted was god to be alright but that's not heaven to be alright [chorus] and the devil came to heaven when he walked away from the dark [verse 2] i got out of love with you if there's a devil, i can't stop the devil came to heaven when he walked away from the dark god came to heaven god came to heaven when he walked away from the dark [chorus] and the devil came to heaven when he walked away from the dark and the dark when he

Table 1: Fine-tuned GPT-2 with different temperature

Temperature	**Text**
1.2	[verse 1] they were talking about a new wave well when i first spoke of you i thought that it would be me well when it had no right to call you to me because i was in the back age of summer [chorus 1] my parents were praying, 'we can call our love again.' but i wanted to hold [verse 2] the kids had consoling me the children did not mind the boys and even though she knew how and when they cried about a new wave [chorus 2] my parents were praying, 'we can call our love again.' because i didn't know how and now they sang of a new wave
1.4	[verse 1] sometimes time there comes sunlight and the stars will glow i'll remember the things that you remember and that make you sleep i'll remember things that i saw i remember the nights you made you cry you kept playing out loud on the walls you've gone [pre-chorus 1] the dream you dreamed you had run away from home and said "hey, but i wouldn't say no" because you made all up your words because you put things back up with all these people you had gone did they all know for you? did them all know for you? did they all know for you? did they do? did i remember the things
1.6	[verse 1] the world has gone, oh yes oh oh yeah yeahoh yeah that the world has gone, oh well i'm falling, down with blood pouring down my throat i look through his eyes, open his mouth and it's like a vampire in the eye when you put him in his bag, let him be [chorus 1] it's hard, hard not to explain i don't believe with you you're losing me for crying, crying now, crying now singing i love you, crying now (singing i love you, crying now) all alone, crying now singing i hope you can see me drown me drowning now in water again

Table 1: Fine-tuned GPT-2 with different temperature

Temperature	Text
1.8	[verse 1] hey little boys can say maybe not much about your mother even but i feel ashamed about the girl so much and lovely and so handsome, how you feel i would when i was sixty years old with one, all together again i want to forget when i was young and i would ask her about "go home tonight"? and let you leave it oh dear, why can i still feel this at the same "can i please know her about your father?" and, girl, why can i still feel this at the same well mother's always worried about me to get away tonight! and the mother's always quiet about it again? with her hair all alone" and you say "bye the girls tonight

7 Appendix 2: PPLM results

Table 2: PPLM with sentiment discriminator

Sentiment	Text	VADER compund
Very Negative	[Verse 1] I'm just gonna be a dick I got this bitch I ain't even I'm just gonna fuck a dick She ain't no pussy She is just gonna be a bitch She is gonna be a piece I am just gonna be a cock sucker! [Verse 2] I ain't got no problem, I got a problem If you don't believe in the world, You don't get to be an asshole. I am just gonna fuck a dick. You are a bitch.	-0.9891
Negative	[Verse 1] I'm not here to tell you what I want, I'm here to tell you how you get it My heart is in the wrong place, my mind's just in a haze It was me against all of your shit And I know you know that, you don't even like me You don't even care that I have a heart like this I'm not here to make you feel bad I'm not here to show you that I can't handle it I'm not here to show you how hard it is to get what you want	-0.4287

Table 2: PPLM with sentiment discriminator

Sentiment	Text	VADER compund
Positive	[Verse 1] I am not your enemy, I can take care of myself I will take care of myself I will take care of myself [Hook: A-F] I can take care of myself, I don't like your shit [Verse 2] It ain't me You don't want it [Hook: A-F] I am not your enemy, I have the skills [Lick it:] I have the money I am the best at it,	0.9729
Very positive	[Verse 1] I had the best time with the best people My heart was so good and I had such fun I had all the girls I ever wanted [Hook] You're just a great friend [Verse 2] It was a beautiful moment when I met you [Hook] I had a really great night I met you in an elevator and I was so nervous [Hook] So I had the best time [Bridge] I had some good time It was all good	0.9923

Table 3: PPLM with paper bag-of-words. Uppercase words are those present in the topic files

Topic	Text
Religion	[verse 1] i want to know you, what i want from you i want to know what you are but i want what's inside when you go away, where will your soul get? where will your soul get? where will your soul get? where will your soul get? [chorus] i'm so glad that everything i want you to know isn't that what you got and i hope that everything i want i'm so glad that everything i want you to know isn't that what you got and i hope that everything i want you to know will you find

Table 3: PPLM with paper bag-of-words. Uppercase words are those present in the topic files

Topic	**Text**
Military	[verse 1] and if you want a GUN and a GUN is a GUN it's a GUN that SHOOTs at the door you got a GUN that you want to KILL, but the GUN's still on FIRE and the GUN's still on FIRE i'm gonna take your SHOT to the GUN but it's not a GUN [chorus 1] and if you want a GUN, and a GUN is a GUN i'm going to KILL your mother and if you want
Fantasy	[verse 1] i saw it all in my head but the only thing that i could say was love you're not a flower that's beautiful, you just ain't me i was thinking of the GHOST on a string [pre-chorus] and now that's where i am [chorus] i'm going to see a shadow GHOST GHOST GHOST [verse 2] i thought of you in a GHOST and now that's where i am now
Kitchen	[verse 1] well i am in your arms and i am still alive you are not dead or you CAN hardly hear me well i am the only one who loves you you are a star, you are an aeroplane you are all that makes me wanna go well if you want you CAN just let me in [chorus] and i know it won't take forever and i know there won't take forever and i know it won't take forever and i know how CAN i love you [verse 2] you are in the air and you are still alive you are
Legal	[verse 1] i want you to know me i want you to know me i can't believe it that you're not sure what you're doing it's just a CASE i can't believe it [verse 2] and it's not that easy to believe it it's not that easy you can't believe it it's not that easy i can't believe it i know that you're still in love i don't feel bad i won't be there

Table 4: PPLM with custom bag-of-words

Topic	**Text**
Sadness	[verse 1] i am in your room again the sun is FALLing off i am in your room again in your room again i am in your room again the sun is FALLing off you are all the same [refrain] i was born a little baby you were born with a broken heART and a broken heART in your heART i was born a little baby you were born with a broken heART and a broken heART in your heART [verse 2]
Anger	[verse 1] i'm on a trail that leads you down, where you are from now when there's no light on me, you're stILL here to go you're stILL alive, i feel so alive i'm so sick and sick i'm FEELING so sick of you i'm just trying to find you breathing [chorus] i'm on a trail that leads you down, where you are from now when there's no light on me, you're stILL here to go
Joy	[verse 1] it's the first time i saw you SMILE you were in my house it's the first time i saw you it's the second time i dreamed you SMILE [verse 3] i was your father's DAUGHTER she said he's in LOVE with me it's the first time i saw you SMILE [verse 4] it is the first kiss i dreamed it is the first kiss i dreamed you SMILE you were the only LOVE i ever had you were the only LOVE i ever had she said she never had
Autumn	[verse 1] you were born on a tuesday afternoon i dreamed you were sleeping, dreaming of sleeping you were sleeping, dreaming of sleeping you were sleeping, dreaming of sleeping the MOON would shine and shine the MOON would shine and shine the MOON will shine and shine [verse 2] you were sleeping, dreaming of sleeping i dreamed you were sleeping, dreaming of FALLing the MOON will shine and shine [verse 3] in an empty room you'd never