Beginner's Guide to Retrain GPT-2 (117M) to Generate Custom Text Content

Ng Wai Foong
May 13 · 16 min read ★

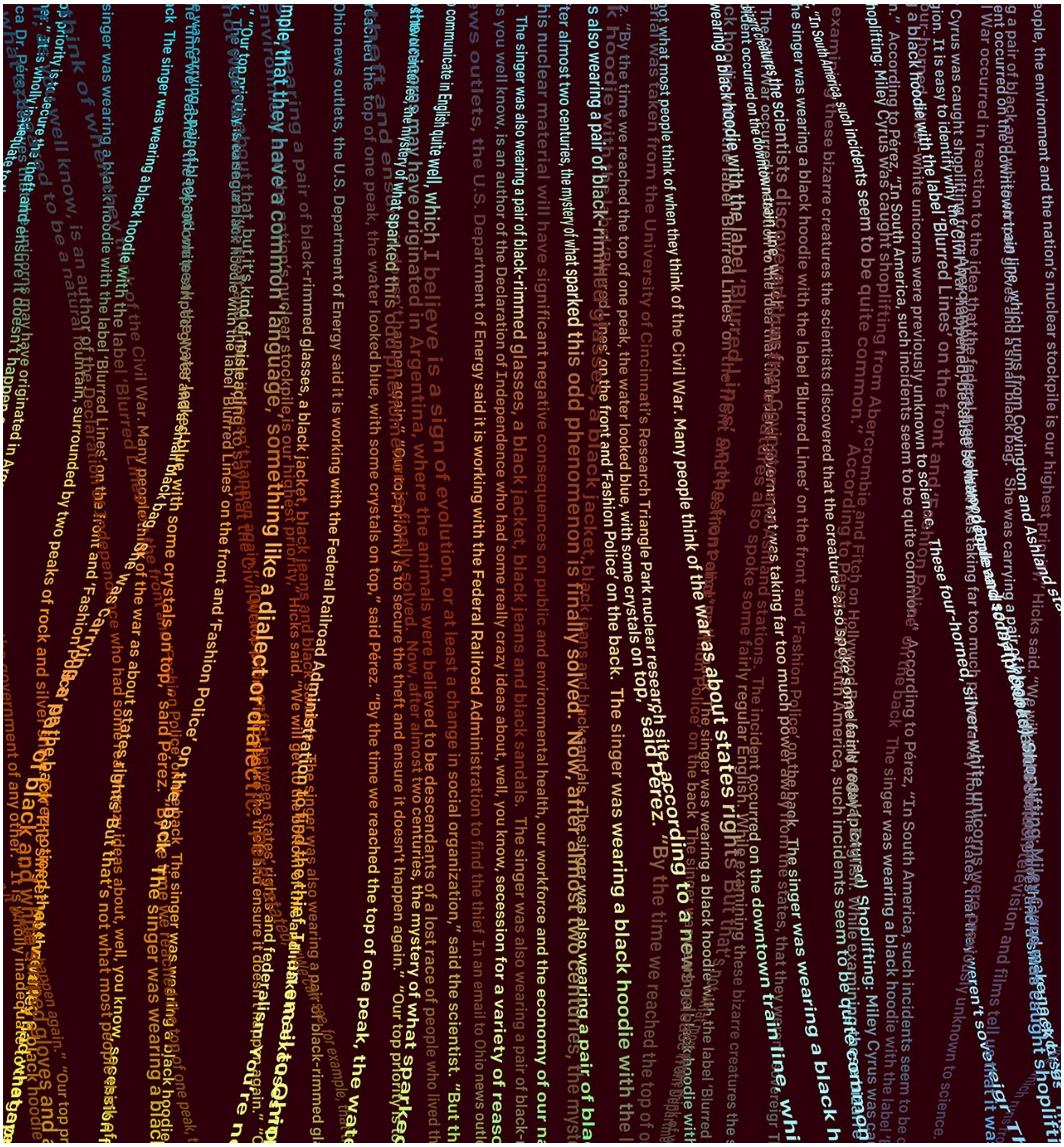


Image taken from https://openai.com/blog/better-language-models/

In this article, we will be exploring the steps required to retrain GPT-2 (117M) using custom text dataset on Windows. For start, GPT-2 is the advanced version of a transformer-based model that was trained to generates synthetic text samples from a variety of user-prompts as input. Check out the official blog post to find out more about GPT-2:

The original version has 1.5GB parameters but the creator, OpenAI team did not released the pre-trained model due to their concerns about malicious applications of the technology. Having said that, they did released a smaller version which has 117MB parameters that can be retrained on custom text dataset.

There are altogether 5 sections in this article.

1. Setup and installation
2. Preparing custom text dataset
3. Training GPT-2
4. Generate samples
5. Conclusion

[Section 1] Setup and installation

As I have mentioned in the introduction, I will be using Windows in this tutorial. However, it should works for any other operating system. I am training it on Windows 10 Pro with the following specifications:

1. **Processor**: Intel(R) Core(TM) i7–8550U CPU @ 1.80GHz 1.99 GHz
2. **Installed memory (RAM)**: 16.0 GB (15.9 GB usable)

3. **System type**: 64-bit Operating System, x64-based processor

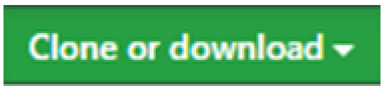
Question: *What do you call a computer without GPU?*

Answer: *CPU!*

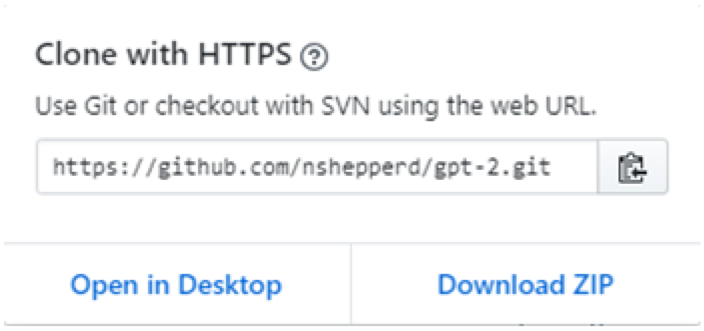
I am not joking as I will be using just CPU to do the training simply because I don't have any GPU. If you are in the same boat as me, do expect that the training time will be exceptionally long in order to achieve good results.

Download required files

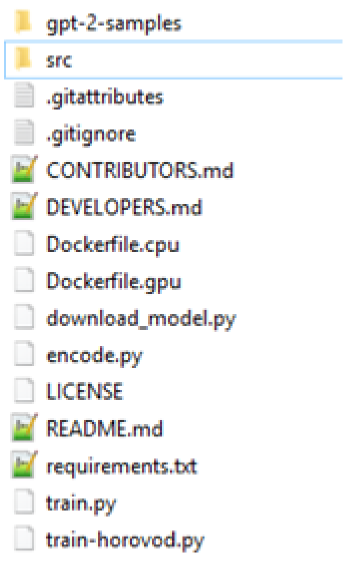
Go to the following Github link and click on the "Clone or download" button



After that, you will see the following popup.


Clone with HTTPS
Use Git or checkout with SVN using the web URL.
https://github.com/nshepperd/gpt-2.git
Open in Desktop
Download ZIP

Click on "Download ZIP" to get the files. For advanced users, feel free to use any other methods available. Once you have extracted the files, you should have the following files:


gpt-2-samples
src
.gitattributes
.gitignore
CONTRIBUTORS.md
DEVELOPERS.md
Dockerfile.cpu
Dockerfile.gpu
download_model.py
encode.py
LICENSE
README.md
requirements.txt
train.py
train-horovod.py

Moving python files to resolve path and directory issues (optional)

There are times where path and directories can make us insane especially when you are a Windows user dealing with command prompt. To solve this, I will be using a *stupid but it works* method. Copy *encode.py* and *train.py* and paste them into the **src** folder. For those that wish to to do distribution on multiple GPUs or machines using Horovod, kindly copy *train-horovod.py* as well. This step is just to make our life easier without the need to modify the PYTHONPATH. It is definitely not the best way and differs from the steps provided in the official Github. For beginners like me, if it works, why not!

Installing Python

You should already have Python installed in your system. In case you have accidentally uninstalled it, feel free to download and install it from the following link.

I am using Python 3.7.1 installed in a virtual environment via Anaconda. You can check the version by typing python in the command prompt using the following command:

```
python --version
```

If you encountered error that reads *'python' is not recognized as an internal or external command, operable program or batch file*, it means that python is not installed. Try to resolve this on your own before you proceed to the next part.

Installing Python modules (Method 1)

The first method is to install via the *requirement.txt* file provided. Requirement files contains the necessary modules (including the version) that are required to run the py files. Check the path to the *requirements.txt*. It is one of the file that you have downloaded in the previous section. We will install the required modules by using the following command in command prompt. The -r in the command is a shortcut for --requirement which means "install from a requirements file".

```
pip install -r path/to/requirements.txt
```

Installing Python modules (Method 2)

The second method is to manually install it one by one. Run each of the following command one at a time.

```
pip install fire>=0.1.3

pip install regex==2017.4.5

pip install requests==2.21.0

pip install tqdm==4.31.1
```

Check installed modules

Kindly check if you have installed the following modules:

1. fire
2. regex
3. requests
4. tqdm

You can check it via the following command

```
pip list
```

Download 117M model

The first thing to do is to download the base model. In the command prompt, point it to the root folder which contains the *download_model.py*. Type the following command:

```
python download_model.py 117M
```

Once you have completed the download, you should have a 117M folder with the following files:

1. checkpoint
2. encoder.json

3. hparams.json
4. model.ckpt.data-00000-of-00001
5. model.ckpt.index
6. model.ckpt.meta
7. vocab.bpe

We will now move on to the next section.

[Section 2] Preparing custom text dataset

You can use any kind of text data that you can find as long as they are in English. Example includes:

1. Light novels
2. Poems
3. Song lyrics
4. Questions and answers
5. Synopsis or abstract
6. News, letter or articles

You can still train it using other languages but bear in mind that you may not achieve a good results with it even if you have sufficient data and time. You can combine all the data into one single text file or split them into multiple text files in one directory. If you are putting them in one basket, remember to delimit each data with

```
<|endoftext|>
```

For example, if I am using Jpop lyrics as training data, I will delimit each lyrics with **<|endoftext|>**.

```
Kimi ni sayonara
Tsugeru made de ii
Dare yori soba ni ite hoshii
```

```
Sonna futari no ketsumatsu o shitte mo
Deae te yokatta to omoiaeru made

suki da yo
imasara da kedo
iwasete
sayonara no mae ni

<|endoftext|>

sake ta mune no kizuguchi ni afure nagareru pain in the dark
kasane ae ta shunkan no tsunagaru omoi tokashi te
same nai netsu ni unasare te
saigo no koe mo kikoe nai

Don't cry
koware sou na hodo dakishime tara kimi ga furue te i ta
sotto kazasu tenohira ni fure te mise te
never untill the end

koboreochiru suna no you ni hakanai negai o close to the light
toji ta kimi no omokage ni kare nai namida nijin de
hodoi ta yubi no sukima kara
inori ga fukaku tsukisasaru
```

On a side note, you don't really need the delimiter to train but having a delimiter allows the model to learn the formatting of the training data. For example, if you just wanted the lyrics, you can lump all of them together without the delimiter. However, if you would like to generate the artist name, title and genre, kindly include such data in your training data and delimit each of them. It is advisable to scale down the scope to one genre or label for best performance.

In this tutorial, I retrained GPT-2 with Jpop lyrics (in romaji format) due to the following reasons:

1. Curiosity
2. Some Jpop lyrics have English words
3. Song lyrics usually contains quite a lot of repetition especially during the chorus.

I built the training data manually via copy and paste method from the following website:

I browsed through the first few song lyrics to make a 8000+ lines of text file as training data. By definition, it is not the best training data but good enough for a simple tutorial.

Once you have completed your training data. Move the folder or files to the **src** directory. I saved the training data in one single text file as *lyric.txt*. The next step is to encode it so that you can use it for multiple runs. Let's run the *encode.py*. Change the command prompt directory to the **src** folder and type the following command (remember that we have copied the *encode.py* file to **src** folder, you have to specify the PYTHONPATH if you are using the one at the root folder):

```
python encode.py lyric.txt lyric.npz
```

lyric.txt is the input file while *lyric.npz* is the output file. Feel free to modify the name accordingly but make sure that you point to the correct path if your dataset is located at another folder. The output file *lyric.npz* will be generated after the encoding. It should just take a few seconds for it depends on the size of your dataset. Kindly refer to the *encode.py* file to find out more:

Once you are all set, we can now move to the next section to train GPT-2.

[Section 3] Training GPT-2

Before we start the training, double check that your command prompt point to the same directory as *train.py*. It should be the same directory if you just performed the encoding process.

Normal training

Kindly refer to the train.py file to find out more about the available arguments:

Type the following command:

```
python train.py --dataset lyric.npz
```

If you would like to see more samples, you can modify it accordingly (default is once every 100 steps). For example, to output 3 samples every 50 steps, type the following command instead:

```
python train.py --dataset lyric.npz --sample_every 50 --sample_num 3
```

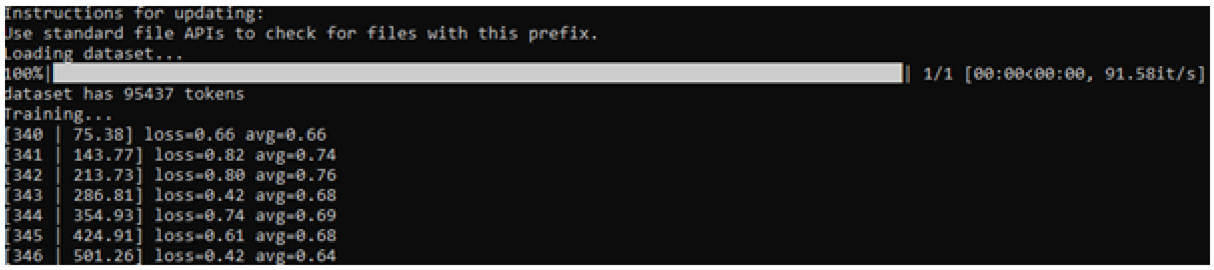
There is also an option for you to increase the **batch size** and **learning rate**. Make sure that you have sufficient RAM to handle the increase in batch size (the default is 1). Learning rate is used for fine-tuning the model. If you noticed that the model is not learning anything (loss is not decreasing), you can further reduce the learning rate. For example, you can use the following command to do training with **batch size** of 2 and **learning rate** of 0.0001:

```
python train.py --dataset lyric.npz --batch_size 2 --learning_rate
0.0001
```

If you are using the latest python modules, do not be alarmed if you happened to see a lot of warning messages such as:

1. Tensorflow binary was not compiled to use: AVX2
2. … is deprecated and will be removed in future version.
3. deprecated in favor of operator or …

You should have the following output after a while:



```
Instructions for updating:
Use standard file APIs to check for files with this prefix.
Loading dataset...
100%|██████████████████████████████████████████████████████████| 1/1 [00:00<00:00, 91.58it/s]
dataset has 95437 tokens
Training...
[340 | 75.38] loss=0.66 avg=0.66
[341 | 143.77] loss=0.82 avg=0.74
[342 | 213.73] loss=0.80 avg=0.76
[343 | 286.81] loss=0.42 avg=0.68
[344 | 354.93] loss=0.74 avg=0.69
[345 | 424.91] loss=0.61 avg=0.68
[346 | 501.26] loss=0.42 avg=0.64
```

Example of the training output after running train.py

From the image above, we can decipher the output **[340 | 75.38] loss=0.66 avg=0.66** as follow:

1. **340**: Refers to the number of training step. Think of it as a counter that will increase by 1 after each run.
2. **75.38**: Time elapsed since the start of training in seconds. You can use the first step as reference to determine how long does it take to run one step.
3. **loss and avg**: Both of them refers to the cross-entropy (log loss) and the average loss. You can use this to determine the performance of your model. In theory, as

training steps increases, the loss should decrease until it converge at certain value. The lower, the better.

Training using horovod

For those that wish to distribute on multiple GPUs to train GPT-2, you can try the following code (all of them in one line):

```
mpirun -np 4 -H localhost:4 -bind-to none -map-by slot -x
NCCL_DEBUG=INFO -x LD_LIBRARY_PATH -x PATH -x PYTHONPATH=src -mca
pml ob1 -mca btl ^openib train-horovod.py --dataset lyric.npz
```

How to stop training?

You can stop the training by using Ctrl+C. By default, the model will be saved once every 1000 steps and a sample will be generated once every 100 steps. After you have interrupted the process, a **checkpoint** folder and **samples** folder will be generated for you. Inside each folder, you can find another folder called **run1** (you can modify this via the run_name argument). Samples will contain the example output from the model, you can view it in any text editor to evaluate your model. The **checkpoint** folder will contains the necessary data for you to resume your training in the future. Each saved model will contain a post-fix according to the number of steps ran. If you saved at the 100 step, you should have the following:

1. model-100.data-00000-of-00001
2. model-100.index
3. model-100.meta

How to resume training from last checkpoint?

You can simply use the following code to resume the training:

```
python train.py --dataset lyric.npz
```

How to resume training from different checkpoint?

If you have created different checkpoints and wanted to resume the training, you can use the following code with the path to the **checkpoint** folder

```
python --restore_from path/to/checkpoint train.py --dataset
lyric.npz
```

How to resume training using different run_name?

You can modify the output folder by specifying the run_name argument. This allows you to keep track of the running process so that you can fine-tuning it and resume it at a particular checkpoint.

```
python train.py --dataset lyric.npz --run_name run2
```

This will output the saved model into a new folder called run2. I managed to train 500 steps over 3 days period with an average loss of 0.25. We will see how to generate samples using the saved model in the next section.

[Section 4] Generate samples

Create a folder for the model

In the **src/models** folder, you should have just one folder called **117M**. Create another folder to store your model alongside with the original model. I made a new folder called **lyric**. Now, I have two folders in **src/models**, one is called **117M** and the other is called **lyric**.

Go to **src/checkpoint/run1** folder, and copy the following files:

1. checkpoint
2. model-xxx.data-00000-of-00001
3. model-xxx.index
4. model-xxx.meta

xxx refers to the step number. Since I have trained for 501 steps, I have *model-501.index*.

Paste them into the newly created folder (in my case, the folder is called **lyric**). Next, go to the 117M folder and copy the following files:

1. encoder.json
2. hparams.json
3. vocab.bpe

Paste them into the lyric folder. Double check that you should have 7 files in it. With this, we are ready to generate samples. There are two ways to generate samples.

Generate unconditional sample

Unconditional sample refers to randomly generate sample without taking into account any user input. Think of it as random sample. Make sure you are in the **src** directory and type the following in the command prompt:

```
python generate_unconditional_samples --model_name lyric
```

Wait for it to run and you should be able to see samples being generated one by one. It will keep on running until you stop it with Ctrl+C. This is the default behavior. We can fine-tune it to obtain different variations of samples. Kindly refer to the *generate_unconditional_samples.py* for more information:

The most important parameters are top_k and temperate.

- **top_k**: Integer value controlling diversity. 1 means only 1 word is considered for each step (token), resulting in deterministic completions, while 40 means 40 words are considered at each step. 0 (default) is a special setting meaning no restrictions. 40 generally is a good value.
- **temperature**: Float value controlling randomness in boltzmann distribution. Lower temperature results in less random completions. As the temperature approaches zero, the model will become deterministic and repetitive. Higher temperature results in more random completions. Default value is 1.

For example, to generate using 0.8 temperate and 40 top_k, type in the following:

```
python generate_unconditional_samples --temperature 0.8 --top_k 40 -
-model_name lyric
```

Generate interactive conditional sample

Interactive conditional sample refers to generate samples based on user input. In other words, you input some text and GPT-2 will do its best to fill in the rest. The command and parameters available is the same as that of unconditional sample. Do not enter your input together in the command as you will be prompt for the input later on in the command prompt. Type the following command:

```
python interactive_conditional_samples --temperature 0.8 --top_k 40
--model_name lyric
```

Once it is running, you can type in your input and press enter. The downside to this interactive mode is that you will face some issues if you have line breaks in your input. For those that experienced this issue, the best way is to type it out in a text file and modify the code to read it. Kindly refers to *interactive_conditional_samples.py* to learn more:

Feel free to play around with the model generated to see what can been generated by your model. There might be surprise findings that can make or break your day. In the next section, we will analyse some samples that I have generated and wrap up this tutorial.

[Section 5] Conclusion

Analyse generated samples

By default, a random sample will be generated once every 100 steps. Since I have ran 500 steps, I have 5 samples generated during the training process. Let's have a look at each one of them find out the interesting parts.

Sample 1

```
...
Kokoro no hibi namida kono demo you ni
(Sou, tatta hikeru no ai wa, shittomo mai you ni yo)
Kore ikiteku kizutsu ga kimi wa jikiriteru
Kore ikiteku kizutsu ga kimi wa jikiriteru
(Sou, tatta hikeru no ai wa, jibun ni yo)
Kokoro no hibi dake na hoshi ga hanau kedo
(Tsukamete hiroi hanabiru ni zutto yoko)
Kore ikiteku kizutsu ga kimi wa jikiriteru
Kore ikiteku kizutsu ga kimi wa jikiriteru
(Sou, tatta hikeru no ai wa)
...
```

We can see that there are quite a lot of repetitions. However, a few sentences begin with the same words but end with different words (as shown by the words in bold).

Sample 2

```
...
Nani mo itsumo kodoku dake
Itsumo itsumo kodoku
Kirei no shirin no youni
Yoko ni shitteiru
Kizutsuki kizutsukeru
Kitto yaru kodoku
...
```

The sentence that is highlighted in bold shows that the words "itsumo" is being repeated twice and "itsumo kodoku" is actually part of the sentence above.

```
...
Kumo no yureta toki ni irete
Kirei no kodoku
Suteki no oku ni furetakute futteyosu
Yuuki no kumo de you ni
Nogaretesu kizutsukeru ne
Yuuki no yureta kowareru ka
...
```

We can also noticed that in some part of the lyrics, the word *no* is being repeated in almost every single sentence.

In Japanese, の which is being written as *no* in romaji, is a possessive particle used to connect nouns together. For example, to say *this is my book*, you will say これは私の本です(kore ha watashi no hon desu). It is not a surprise that you will often encounter this particle in song lyrics but having so many of them is plain wrong. Not to mention that it is wrongly used. In the sentence "kirei no kodoku" (highlighted in bold):

- **kirei**: refers to 綺麗（きれい）which means beautiful. It is na-adjective and should be followed by a *na* particle and not *no* particle which makes the sentence grammatically wrong.

- **kodoku**: refers to either 孤独(loneliness) or 蠱毒(a type of poison called Gu, associated with the culture of South China). Both words are represented by the same hiragana（こどく）.

As a result, the meaning for the sentence will end up as beautiful loneliness or beautiful poison which makes it a dark-theme song lyric. Definitely not what we wanted to hear from a normal Jpop song.

Sample 3

```
aki
Natsukashii Kyou mo kawaranai you ni
Kanawanai you ni kirakiri
Tama mo todoka atte hoshikute ii

Konya iki mo tomatta toki ha hen ni
Kakehiki hanashi hen dashitatte
Shiranai furueteta toki ha

Itsumori nanika ni kogeta michi
Tsumetai kokoro ha otozure nami ni
Koyuki ni konya wo
Tsumetai yoru wo kiete yo

Itsuji nara momiji no
Tsuyogari da kedo watashi ni kiete
Ai wa itsuka kono te wo tozashita sono hanabi no yo
Itsumori nanika ni kogeta michi
Tsumetai kokoro ha otozure nami ni
Koyuki ni konya wo
Tsumetai yoru wo kiete yo

Itsuji nara momiji no
Tsuyogari da kedo watashi ni kiete
Ai wa itsuka kono te wo tozashita sono hanabi no yo
```

Personally, I like this part the most (highlighted in bold). It reads:

- **aki**: refers to 秋（あき）which means autumn
- **natsukashii kyou**: refers to 懐かしい今日 which means today is a nostalgic day
- **mo**: refers to a particle も which means also

- **kawaranai you ni**: refers to 変わらないように which means may it not change

To sum it up, it means *autumn, may this nostalgic day stay as it is*.

```
...
"itsu de mirai" made
ma sura natsukashii mie tte
sakyuu ga kareru? te wo mada chiriyuku
jibun mae no bokura nda
dakedo kamawanai you ni
tooku made ita
...
```

There are also some random words that does not make sense in Japanese such as **nda** (highlighted in bold).

Sample 4

```
...
Happy Days azu pregnancy?
Doudou saishuu atsuku
Hageshiku atsuku dakara
Trouble spots ai no oto deepakou atashi ni sakaramono
Ato me mo nakute
Kimi no te wo meguri
Kirawaranai ippai
Manmaru tte kankei abishime yo

Happy Days ai no oto deepakou soko de anata ga
Hitori nagara dake de
Tondeiki ne
Yami ame mo nai
...
```

Up until now, the generated samples are all in broken Japanese. However, remember that I did include samples with mixture of English and Japanese as well. In this sample, we finally able to see a sample that include both languages. However, it is far from what I have expected. As you can see, the model mixed up a few random English that do not make sense and even created a new word called "deepakou".

Sample 5

```
...
We're gonna make it
Katachi no nai futari
Hikisakarete shimawanai
We're gonna make it real

We're gonna make it
Tatoe no hane wo
Ima wo ai shiteru you ni

We're gonna make it real

We're gonna make it real
Katachi no nai futari
Kanashii tsuki no koe wo
Hikisakarete shimawanai
We're gonna make it real

We're gonna make it real
Katachi no nai futari
Kanashii tsuki no koe wo
...
```

GPT-2's attempt at making it real. However, pasting it all over the places is not a good choice.

```
...
Strong, be strong wasurenaide
Hisshi ni shita nara  tsuki ni aru no ni
Strong, be in control jibun no aida ga
Kokoro no naka no youni  omoi wo maretari kitto
bokura ga nia wazu ni  nijimari iyasora no naka de

nakanai de natta toki da nagasete
nakitaku natta toki da inai no ni
nakitaku natta toki da inai no ni
nakitaku natta toki da inai no ni
nakitaku natta toki da inai no ni
...
```

GPT-2 is trying to appear strong but lost control at the last moment. The result is still pretty bad at this moment. If you ever got yourselves in this situation, the solutions are to increase training time or training data. Since this is just a simple tutorial, I will stop the training for now. Let's move on to the summary to wrap up this tutorial.

Summary

Congratulations, you have completed a simple tutorial in retraining GPT-2 based on your own custom dataset. What can we learn from this? Well, GPT-2 provides us with an idea on how useful AI is at generating content. Although this is just the smaller version of GPT-2, it is incredibly useful at generating some samples that might be useful. The main gist is not how AI will replace us in generating content but rather on how we can leverage AI to help us generate valuable content in shorter time frame. For example, I can retrain GPT-2 to generate random song lyrics as samples. Then, I can use the samples as reference when writing my own song lyrics. Who knows I might be able to create a masterpiece? Most of the time, we only lack some hints and guidance from the people around us to move forward with our work. That's all from this tutorial, I hope that you enjoyed it. I will look forward on how AI can be used to ignite the creativity and spark joy in our daily life!

Reference

Feel free to check out the following links for more information on GPT-2:

1. https://openai.com/blog/better-language-models/
2. https://github.com/nshepperd/gpt-2
3. https://www.gwern.net/GPT-2

Python Artificial Intelligence Gpt 2