

Text Generation Techniques Using GPT-2

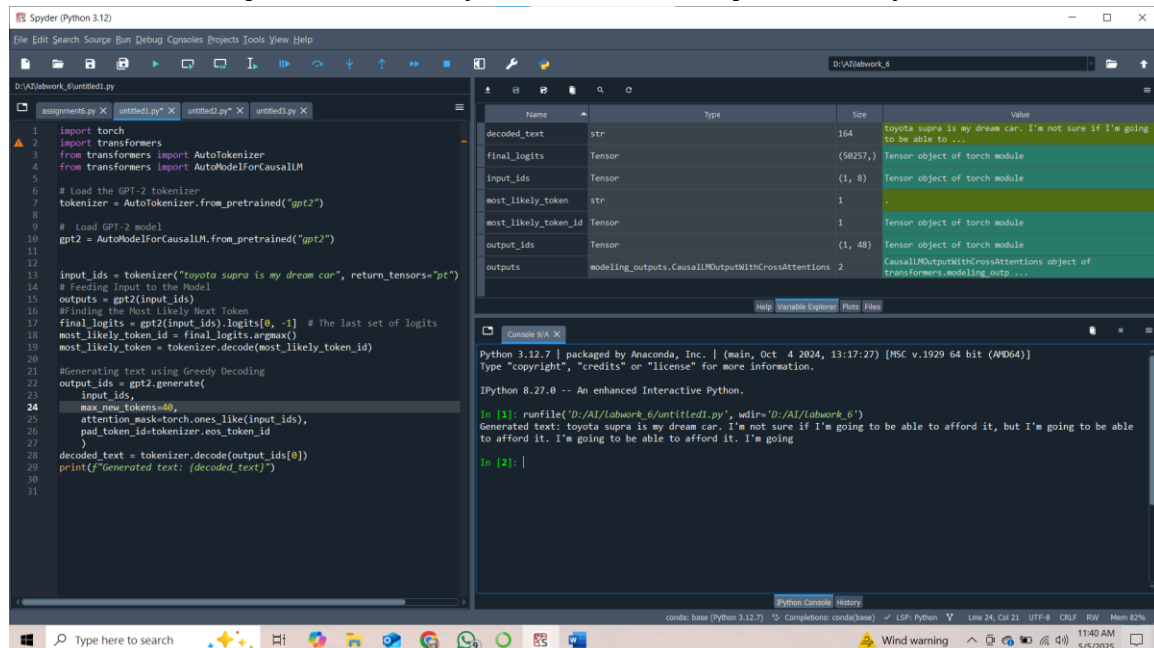
1. Obtained Results

Multiple text generation methods were applied to the phrase "toyota supra is my dream car" using the GPT-2 model. The following decoding strategies and parameters were tested:

1. Greedy Decoding:

Output: "toyota supra is my dream car. I'm not sure if I'm going to be able to afford it, but I'm going to be able to afford it. I'm going"

- Observation: Repetitive and overly deterministic. It loops awkwardly.



The screenshot shows the Spyder Python IDE with a Python script for GPT-2 Greedy Decoding. The script imports torch, transformers, and AutoTokenizer, AutoModelForCausalLM. It loads the GPT-2 tokenizer and model. The input text "toyota supra is my dream car" is tokenized. The model generates the next token by finding the most likely token from the output logits. This process is repeated to generate the full output text. The output is printed as "Generated text: {decoded_text}". The Variable Explorer shows the decoded_text variable as a string with the value "toyota supra is my dream car. I'm not sure if I'm going to be able to ...". The Console shows the execution of the script, including the command to run the file and the output of the generated text.

```
1 import torch
2 import transformers
3 from transformers import AutoTokenizer
4 from transformers import AutoModelForCausalLM
5
6 # Load the GPT-2 tokenizer
7 tokenizer = AutoTokenizer.from_pretrained("gpt2")
8
9 # Load GPT-2 model
10 gpt2 = AutoModelForCausalLM.from_pretrained("gpt2")
11
12
13 input_ids = tokenizer("toyota supra is my dream car", return_tensors="pt")
14 # Feeding Input to the Model
15 outputs = gpt2(input_ids)
16 # Finding the Most likely Next Token
17 final_logits = gpt2(input_ids).logits[0, -1] # The last set of logits
18 most_likely_token_id = final_logits.argmax()
19 most_likely_token = tokenizer.decode(most_likely_token_id)
20
21 #Generating text using Greedy Decoding
22 output_ids = gpt2.generate(
23     input_ids,
24     max_new_tokens=40,
25     attention_mask=torch.ones_like(input_ids),
26     pad_token_id=tokenizer.eos_token_id
27 )
28 decoded_text = tokenizer.decode(output_ids[0])
29 print(f"Generated text: {decoded_text}")
30
31
```

2. Beam Search (num_beams=5):

Output: "toyota supra is most expensive car in the world. In the United States, the average cost of a car in the United States is about"

- Observation: Better structure and informativeness, but includes redundancy.

```

1 import torch
2 import transformers
3 from transformers import AutoTokenizer
4 from transformers import AutoModelForCausalLM
5
6 # Load the GPT-2 tokenizer
7 tokenizer = AutoTokenizer.from_pretrained("gpt2")
8
9 # Load GPT-2 model
10 gpt2 = AutoModelForCausalLM.from_pretrained("gpt2")
11
12
13 input_ids = tokenizer("toyota supra is most expensive car", return_tensors="pt")
14 # Feeding Input to the Model
15 outputs = gpt2(input_ids)
16 # Finding the Most Likely Next Token
17 final_logits = gpt2(input_ids).logits[-1] # The last set of logits
18 most_likely_token_id = final_logits.argmax(-1)
19 most_likely_token = tokenizer.decode(most_likely_token_id)
20 # Generating text using Beam Search
21 beam_output = gpt2.generate(
22     input_ids,
23     num_beams=5,
24     max_new_tokens=40,
25     attention_mask=torch.ones_like(input_ids),
26     pad_token_id=tokenizer.eos_token_id
27 )
28 print(tokenizer.decode(beam_output[0], skip_special_tokens=True))

```

Name	Type	Size	Value
beam_output	Tensor	(1, 48)	Tensor object of torch module
final_logits	Tensor	(50257,)	Tensor object of torch module
input_ids	Tensor	(1, 8)	Tensor object of torch module
most_likely_token	str	1	.
most_likely_token_id	Tensor	1	Tensor object of torch module
outputs	modeling_outputs.CausalLMOutputWithCrossAttentions	2	CausalLMOutputWithCrossAttentions object of transformers.modeling_outp ...

```

Python 3.12.7 | packaged by Anaconda, Inc. | (main, Oct 4 2024, 13:17:27) [MSC v.1929 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.

IPython 8.27.0 -- An enhanced Interactive Python.

In [1]: runfile('D:/AI/labwork_6/untitled2.py', wdir='D:/AI/labwork_6')
toyota supra is most expensive car in the world.
In the United States, the average cost of a car in the United States is about $1,000. The average cost of a car in the United States is about

In [2]:

```

3. Beam Search + Repetition Penalty (penalty=1.2):

Output: "toyota supra is most expensive car. In the United States, the average cost of a Toyota Prius is about \$1,000. In Japan, the average cost of a Toyota Prius is"

- Observation: Improved variation; avoids extreme repetition but still has redundancy.

```

1 import torch
2 import transformers
3 from transformers import AutoTokenizer
4 from transformers import AutoModelForCausalLM
5
6 # Load the GPT-2 tokenizer
7 tokenizer = AutoTokenizer.from_pretrained("gpt2")
8
9 # Load GPT-2 model
10 gpt2 = AutoModelForCausalLM.from_pretrained("gpt2")
11
12
13 input_ids = tokenizer("toyota supra is most expensive car", return_tensors="pt")
14 # Feeding Input to the Model
15 outputs = gpt2(input_ids)
16 # Finding the Most Likely Next Token
17 final_logits = gpt2(input_ids).logits[-1] # The last set of logits
18 most_likely_token_id = final_logits.argmax(-1)
19 most_likely_token = tokenizer.decode(most_likely_token_id)
20 # Text generation using Repetition Penalty
21 beam_output = gpt2.generate(
22     input_ids,
23     num_beams=5,
24     repetition_penalty=1.2,
25     max_new_tokens=40,
26     attention_mask=torch.ones_like(input_ids),
27     pad_token_id=tokenizer.eos_token_id
28 )
29 print(tokenizer.decode(beam_output[0], skip_special_tokens=True))

```

Name	Type	Size	Value
beam_output	Tensor	(1, 48)	Tensor object of torch module
final_logits	Tensor	(50257,)	Tensor object of torch module
input_ids	Tensor	(1, 8)	Tensor object of torch module
most_likely_token	str	1	.
most_likely_token_id	Tensor	1	Tensor object of torch module
outputs	modeling_outputs.CausalLMOutputWithCrossAttentions	2	CausalLMOutputWithCrossAttentions object of transformers.modeling_outp ...

```

Python 3.12.7 | packaged by Anaconda, Inc. | (main, Oct 4 2024, 13:17:27) [MSC v.1929 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.

IPython 8.27.0 -- An enhanced Interactive Python.

In [1]: runfile('D:/AI/labwork_6/untitled3.py', wdir='D:/AI/labwork_6')
toyota supra is most expensive car in the world.
In the United States, the average cost of a Toyota Prius is about $1,000. In Japan, the average cost of a Toyota Prius is about $

In [2]:

```

4. Sampling with Temperature (temperature=0.2):

Output: "toyota supra is most expensive car. The court found that the cost of a single-family home was \$1,582 per year for one person and an additional 1 percent to 2"

- Observation: Limited randomness and some coherence. Output turns to unrelated

content.

```
1 import torch
2 import transformers
3 from transformers import AutoTokenizer
4 from transformers import AutoModelForCausalLM
5
6 # Load the GPT-2 tokenizer
7 tokenizer = AutoTokenizer.from_pretrained("gpt2")
8
9 # Load GPT-2 model
10 gpt2 = AutoModelForCausalLM.from_pretrained("gpt2")
11
12
13 input_ids = tokenizer("toyota supra is most expensive car", return_tensors="pt")
14 # Feeding input to the Model
15 outputs = gpt2(input_ids)
16 # Finding the Most Likely Next Token
17 final_logits = gpt2(input_ids).logits[-1] # The last set of logits
18 most_likely_token_id = final_logits.argmax()
19 most_likely_token = tokenizer.decode(most_likely_token_id)
20 # Text generation using Temperature
21 sampling_output = gpt2.generate(
22     input_ids,
23     do_sample=True,
24     temperature=0.2,
25     max_length=40,
26     repetition_penalty=1.2,
27     top_k=0,
28     attention_mask=torch.ones_like(input_ids),
29     pad_token_id=tokenizer.eos_token_id
30 )
31 print(tokenizer.decode(sampling_output[0], skip_special_tokens=True))
32
```

Name	Type	Size	Value
final_logits	Tensor	(50257,)	Tensor object of torch module
input_ids	Tensor	(1, 8)	Tensor object of torch module
most_likely_token	str	1	.
most_likely_token_id	Tensor	1	Tensor object of torch module
outputs	modeling_outputs.CausalLMOutputWithCrossAttentions	2	CausalLMOutputWithCrossAttentions object of transformers.modeling_outp ...
sampling_output	Tensor	(1, 40)	Tensor object of torch module

```
Python 3.12.7 | packaged by Anaconda, Inc. | (main, Oct 4 2024, 13:17:27) [MSC v.1929 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.

Python 8.27.0 -- An enhanced Interactive Python.

In [1]: runfile('D:/AI/Labwork_6/untitled4.py', wdir='D:/AI/Labwork_6')
toyota supra is most expensive car.
The court found that the cost of a single-family home was $1,500 per year for one person and an additional 1 percent to 2 percent

In [2]:
```

5. Top-K Sampling (top_k=10, top_p=0.94):

Output: "toyota supra is most expensive car. A single-vehicle dealer could easily sell a Toyota Tacoma or Lexus for \$1,900 in its first year of production if it wanted to avoid"
- Observation: Balanced randomness with more specific content.

```
1 import torch
2 import transformers
3 from transformers import AutoTokenizer
4 from transformers import AutoModelForCausalLM
5
6 # Load the GPT-2 tokenizer
7 tokenizer = AutoTokenizer.from_pretrained("gpt2")
8
9 # Load GPT-2 model
10 gpt2 = AutoModelForCausalLM.from_pretrained("gpt2")
11
12
13 input_ids = tokenizer("toyota supra is most expensive car", return_tensors="pt")
14 # Feeding input to the Model
15 outputs = gpt2(input_ids)
16 # Finding the Most Likely Next Token
17 final_logits = gpt2(input_ids).logits[-1] # The last set of logits
18 most_likely_token_id = final_logits.argmax()
19 most_likely_token = tokenizer.decode(most_likely_token_id)
20 # Text generation Top-K Sampling
21 sampling_output = gpt2.generate(
22     input_ids,
23     do_sample=True,
24     max_length=40,
25     top_p=0.94,
26     top_k=10,
27     repetition_penalty=1.2,
28     attention_mask=torch.ones_like(input_ids),
29     pad_token_id=tokenizer.eos_token_id
30 )
31 print(tokenizer.decode(sampling_output[0], skip_special_tokens=True))
32
```

Name	Type	Size	Value
final_logits	Tensor	(50257,)	Tensor object of torch module
input_ids	Tensor	(1, 8)	Tensor object of torch module
most_likely_token	str	1	.
most_likely_token_id	Tensor	1	Tensor object of torch module
outputs	modeling_outputs.CausalLMOutputWithCrossAttentions	2	CausalLMOutputWithCrossAttentions object of transformers.modeling_outp ...
sampling_output	Tensor	(1, 40)	Tensor object of torch module

```
Python 3.12.7 | packaged by Anaconda, Inc. | (main, Oct 4 2024, 13:17:27) [MSC v.1929 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.

Python 8.27.0 -- An enhanced Interactive Python.

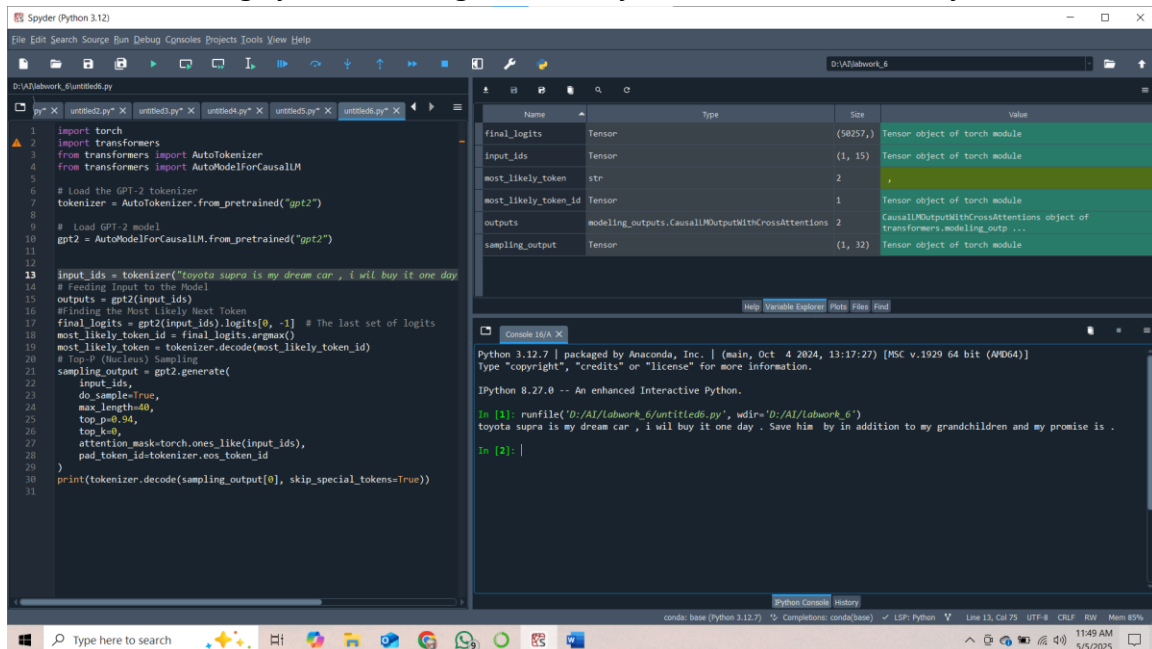
In [1]: runfile('D:/AI/Labwork_6/untitled5.py', wdir='D:/AI/Labwork_6')
toyota supra is most expensive car. A single-vehicle dealer could easily sell a Toyota Tacoma or Lexus for $1,900 in its first year of production if it wanted to avoid

In [2]:
```

6. Top-P (Nucleus) Sampling (top_p=0.94, top_k=0):

Output: "toyota supra is my dream car , i wil buy it one day . Save him by in addition to my grandchildren and my promise is ."

- Observation: Highly creative but grammatically incorrect and lacks clarity.



2. Discussion of Results

The outputs reveal key strengths and weaknesses of each decoding strategy:

- Greedy Decoding is fast but repetitive and lacks depth or diversity.
- Beam Search improves fluency but can be redundant without further tuning.
- Adding a Repetition Penalty helps reduce looping patterns and boosts lexical variety.
- Temperature Sampling controls randomness. Lower values like 0.2 give more deterministic outputs but risk monotony and topic drift.
- Top-K Sampling filters vocabulary to the top-k likely words, promoting diversity without losing relevance.
- Top-P (Nucleus) Sampling selects words from a dynamic probability mass, encouraging creativity. However, it can generate odd or incoherent sentences.

3. Conclusions

Each generation method offers trade-offs between structure, diversity, and creativity. For reliable, informative text, Beam Search with repetition penalty is preferred. For exploratory or imaginative tasks, Top-K or Top-P sampling is ideal. The choice depends on the use case and desired output style.

Appendix: Code Used

```
import torch
import transformers
from transformers import AutoTokenizer
from transformers import AutoModelForCausalLM

# Load the GPT-2 tokenizer
tokenizer = AutoTokenizer.from_pretrained("gpt2")

# Load GPT-2 model
gpt2 = AutoModelForCausalLM.from_pretrained("gpt2")

input_ids = tokenizer("toyota supra is my dream car",
return_tensors="pt").input_ids

# Feeding Input to the Model
outputs = gpt2(input_ids)

# Finding the Most Likely Next Token
final_logits = gpt2(input_ids).logits[0, -1] # The last set of logits
most_likely_token_id = final_logits.argmax()
most_likely_token = tokenizer.decode(most_likely_token_id)

# Generating text using Greedy Decoding
output_ids = gpt2.generate(
    input_ids,
    max_new_tokens=40,
    attention_mask=torch.ones_like(input_ids),
    pad_token_id=tokenizer.eos_token_id
)
decoded_text = tokenizer.decode(output_ids[0])
print(f"Generated text: {decoded_text}")

#i used it step by step as you can see in the screenshot.....

#Generating text using Beam Search
beam_output = gpt2.generate(
    input_ids,
    num_beams=5,
    max_new_tokens=40,
    attention_mask=torch.ones_like(input_ids),
    pad_token_id=tokenizer.eos_token_id
```

```

)

print(tokenizer.decode(beam_output[0], skip_special_tokens=True))

# Text generation using Repetition Penalty

beam_output = gpt2.generate(

    input_ids,

    num_beams=5,

    repetition_penalty=1.2,

    max_new_tokens=40,

    attention_mask=torch.ones_like(input_ids),

    pad_token_id=tokenizer.eos_token_id

)

print(tokenizer.decode(beam_output[0], skip_special_tokens=True))

# Text generation using Temperature

sampling_output = gpt2.generate(

    input_ids,

    do_sample=True,

    temperature=0.2,

    max_length=40,

    repetition_penalty=1.2,

    top_k=0,

    attention_mask=torch.ones_like(input_ids),

    pad_token_id=tokenizer.eos_token_id

)

print(tokenizer.decode(sampling_output[0], skip_special_tokens=True))

# Text generation Top-K Sampling

sampling_output = gpt2.generate(

    input_ids,

    do_sample=True,

```

```

        max_length=40,

        top_p=0.94,

        top_k=10,

        repetition_penalty=1.2,

        attention_mask=torch.ones_like(input_ids),
        pad_token_id=tokenizer.eos_token_id
    )

print(tokenizer.decode(sampling_output[0], skip_special_tokens=True))

# Top-P (Nucleus) Sampling

sampling_output = gpt2.generate(

    input_ids,

    do_sample=True,

    max_length=40,

    top_p=0.94,

    top_k=0,

    attention_mask=torch.ones_like(input_ids),

    pad_token_id=tokenizer.eos_token_id

)

print(tokenizer.decode(sampling_output[0], skip_special_tokens=True))

```