GPT2를 활용한 텍스트 생성하기

학습 목표

- GPT-2를 활용하여 텍스트를 생성해 본다.

학습 내용

- 라이브러리 설치
- GPT-2을 활용한 텍스트 생성

개발 환경 구축

- torch, torchvision, torchaudio, transformers 설치

```
!pip install torch torchvision torchaudio transformers --upgrade
```

```
!pip install torch torchvision torchaudio transformers --upgrade
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.3.1+cu121)
Collecting torch
  Downloading torch-2.4.0-cp310-cp310-manylinux1_x86_64.whl.metadata (26 kB)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (0.18.1+cu121)
Collecting torchvision
  Downloading torchvision-0.19.0-cp310-cp310-manylinux1_x86_64.whl.metadata (6.0 kB)
Requirement already satisfied: torchaudio in /usr/local/lib/python3.10/dist-packages (2.3.1+cu121)
Collecting torchaudio
  Downloading torchaudio-2.4.0-cp310-cp310-manylinux1_x86_64.whl.metadata (6.4 kB)
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.42.4)
Collecting transformers
  Downloading transformers-4.44.2-py3-none-any.whl.metadata (43 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.7/43.7 kB 1.4 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.15.4)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch) (4.12.2)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch) (1.13.2)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch) (2024.6.1)
Collecting nvidia-cuda-nvrtc-cu12==12.1.105 (from torch)
  Using cached nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-manylinux1_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-runtime-cu12==12.1.105 (from torch)
  Using cached nvidia_cuda_runtime_cu12-12.1.105-py3-none-manylinux1_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-cupti-cu12==12.1.105 (from torch)
  Using cached nvidia_cuda_cupti_cu12-12.1.105-py3-none-manylinux1_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cudnn-cu12==9.1.0.70 (from torch)
  Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cublas-cu12==12.1.3.1 (from torch)
  Using cached nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch)
  Using cached nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-curand-cu12==10.3.2.106 (from torch)
  Using cached nvidia_curand_cu12-10.3.2.106-py3-none-manylinux1_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cusolver-cu12==11.4.5.107 (from torch)
  Using cached nvidia_cusolver_cu12-11.4.5.107-py3-none-manylinux1_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cusparse-cu12==12.1.0.106 (from torch)
  Using cached nvidia_cusparse_cu12-12.1.0.106-py3-none-manylinux1_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-nccl-cu12==2.20.5 (from torch)
  Using cached nvidia_nccl_cu12-2.20.5-py3-none-manylinux2014_x86_64.whl.metadata (1.8 kB)
Collecting nvidia-nvtx-cu12==12.1.105 (from torch)
  Using cached nvidia_nvtx_cu12-12.1.105-py3-none-manylinux1_x86_64.whl.metadata (1.7 kB)
Collecting triton==3.0.0 (from torch)
  Downloading triton-3.0.0-1-cp310-cp310-manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (1.3 kB)
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-cu12==11.4.5.107->torch)
  Using cached nvidia_nvjitlink_cu12-12.6.20-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from torchvision) (1.26.4)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.10/dist-packages (from torchvision) (9.4.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (24.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2024.5.15)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.4.4)
Requirement already satisfied: tokenizers<0.20,>=0.19 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.19.1)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.66.5)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch) (2.1.5)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.3.2)
```

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer
import torch

# GPT-2 모델과 토크나이저 로드
model_name = "gpt2"
model = GPT2LMHeadModel.from_pretrained(model_name)
tokenizer = GPT2Tokenizer.from_pretrained(model_name)

# 모델을 평가 모드로 설정
model.eval()

# 텍스트 입력
input_text = "Once upon a time"
input_ids = tokenizer.encode(input_text, return_tensors="pt")

# 텍스트 생성
with torch.no_grad():
    output = model.generate(input_ids, max_length=50, num_return_sequences=1)

# 생성된 텍스트 디코딩
generated_text = tokenizer.decode(output[0], skip_special_tokens=True)
print(generated_text)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
config.json: 100%                                                 665/665 [00:00<00:00, 38.6kB/s]
model.safetensors: 100%                                           548M/548M [00:05<00:00, 133MB/s]
generation_config.json: 100%                                      124/124 [00:00<00:00, 8.94kB/s]
tokenizer_config.json: 100%                                       26.0/26.0 [00:00<00:00, 1.92kB/s]
vocab.json: 100%                                                  1.04M/1.04M [00:00<00:00, 3.94MB/s]
merges.txt: 100%                                                  456k/456k [00:00<00:00, 3.49MB/s]
tokenizer.json: 100%                                              1.36M/1.36M [00:00<00:00, 6.79MB/s]
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_tokenization_spaces` was not set.
  warnings.warn(
The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please pass your input's `attention_
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
The attention mask is not set and cannot be inferred from input because pad token is same as eos token. As a consequence, you may observe unexpec
Once upon a time, the world was a place of great beauty and great danger. The world was a place of great danger, and the world was a place of gre
```

코드 설명

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer
import torch
```

- transformers 라이브러리에서 GPT2LMHeadModel과 GPT2Tokenizer를 임포트합니다.
- GPT2LMHeadModel은 GPT-2 모델을 로드하고 텍스트 생성을 수행하는 데 사용되며,
- GPT2Tokenizer는 텍스트를 토큰으로 변환하거나 토큰을 텍스트로 변환하는 데 사용됩니다.
- torch는 PyTorch 라이브러리로, 모델의 연산과 텐서 조작을 위해 사용됩니다.

```
model_name = "gpt2"
model = GPT2LMHeadModel.from_pretrained(model_name)
tokenizer = GPT2Tokenizer.from_pretrained(model_name)
```

- model_name 변수에 사용할 GPT-2 모델의 이름을 지정합니다. "gpt2"는 기본 모델입니다.
- GPT2LMHeadModel.from_pretrained(model_name) 호출을 통해 사전 훈련된 GPT-2 모델을 로드합니다.
- GPT2Tokenizer.from_pretrained(model_name) 호출을 통해 해당 모델에 맞는 토크나이저를 로드합니다.

```
model.eval()
```

- model.eval()을 호출하여 모델을 평가 모드로 설정합니다.
- 이는 모델이 학습이 아닌 추론을 수행하도록 하며, Dropout과 BatchNorm과 같은 학습 중에만 활성화되는 기능을 비활성화합니다.
- 학습 모드에서 평가 모드 전환
    - Dropout 비활성화,

- Batch Normalizaion 고정(학습 중, 배치마다 평균과 분산을 계산하여 정규화하는 Batch Normalization이 평가 모드에서는 학습된 평균과 분산을 사용하여 고정.)

```
input_text = "Once upon a time"
input_ids = tokenizer.encode(input_text, return_tensors="pt")
```

- input_text 변수에 생성할 텍스트의 시작 부분을 지정합니다.
- tokenizer.encode()를 호출하여 입력 텍스트를 모델이 이해할 수 있는 형태인 토큰 ID(숫자 형태로 매핑)로 변환합니다. return_tensors="pt"는 pt는 PyTorch를 의미한다. PyTorch 텐서 형식으로 반환되도록 지정합니다. 텐서는 PyTorch에서 효율적인 계산을 위해 사용하는 다차원 배열의 형태로, 특히 GPU에서 계산이 유리.

```
with torch.no_grad():
    output = model.generate(input_ids, max_length=50, num_return_sequences=1)
```

- with torch.no_grad() 블록은 모델 평가나 추론할 때 사용.
  - 모델의 가중치를 업데이트하기 위해 그래디언트를 계산합니다.
  - 모델 평가, 추론할 때는 가중치 업데이트가 필요없음. 이때 그래디언트 계산을 비활성화하여 메모리와 계산 비용을 줄입니다.
- model.generate()를 호출하여 텍스트 생성을 시작합니다. input_ids는 입력 텍스트를 토큰화한 ID이며, max_length=50은 생성할 텍스트의 최대 길이를 50으로 설정합니다. num_return_sequences=1은 하나의 생성된 텍스트 시퀀스만 반환하도록 지정합니다.

```
generated_text = tokenizer.decode(output[0], skip_special_tokens=True)
print(generated_text)
```

- tokenizer.decode()를 호출하여 모델이 생성한 토큰 ID를 사람이 읽을 수 있는 텍스트로 변환합니다. - skip_special_tokens=True는 특별 토큰(예: 패딩, 시작, 종료 토큰 등)을 제거합니다.
  - 패딩, 시작, 종료 토큰을 제거하면 최종 출력이 더 깔끔하고 사람이 읽기 쉬운 형태가 된다.
- 최종적으로 생성된 텍스트를 출력합니다.

코딩을 시작하거나 AI로 코드를 생성하세요.