인공지능프로그래밍

Lab 06: GPT2 Model for Language Understanding

학 번 : 2021202058

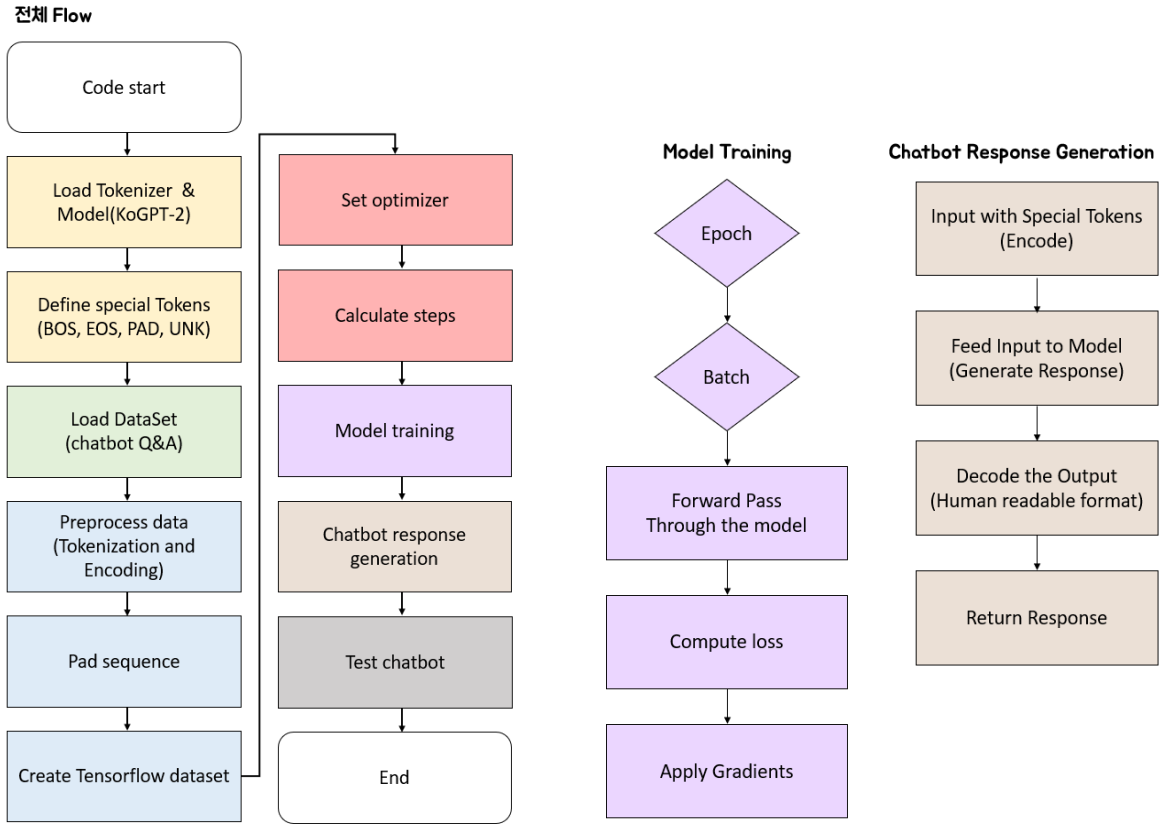
성 명 : 송채영

날 짜 : 2024.11.07

Lab Objective

이번 과제의 목적은 사전 학습된 언어 모델인 KoGPT-2와 TensorFlow 및 Keras를 사용하여 chatbot 모델을 구현해보는 것이다. Special token을 사용하여 입력 text를 처리하기 위해 tokenizer를 구현해보고 입력 encoding, 입력을 기반으로 응답을 생성, 사람이 읽을 수 있는 형식으로 decoding하는 전체 프로세스를 이해하고 구현해본다. 또한 이를 통해 생성된 텍스트에서 chatbot의 응답을 추출하여 사용자에게 반환하는 과정까지 실습해본다.

Program flow



우선 왼쪽 사진은 전체적인 코드의 흐름이다. 먼저 노란색 박스에 해당하는 부분이다. 필요한 라이브러리를 가져온 후 transformers 라이브러리를 사용하여 사전 훈련된 KoGPT-2 tokenizer 및 모델을 로드한다. 이후 BOS, EOS, PAD, UNK, special tokens을 설정한다. 이 부분은 설정 및 초기화해주는 부분에 해당한다. 다음으로 연두색 박스에 해당하는 부분으로 데이터세트를 처리하는 과정이다. 질문과 답변이 포함된 간단한 챗봇 데이터세트(CSV 파일)을 다운로드하였다. 이후 하늘색 박스에 해당하는 부분으로 데이터 전처리를 해주었다. 질문과 답변 pair를 토큰화하여 숫자 토큰 ID로 인코딩 해주었다. 그리고 시퀀스의 길이가 모두 동일하도록 padding을 해준 후 Tensorflow 데이터 세트로 변환해주었다. 다음으로 빨간색 박스는 training 준비에 해당한다. Adam optimizer를 사용하여 설정

해주었고 데이터세트 크기와 배치 크기를 기반으로 각 훈련 에폭에 필요한 step 수를 계산해주었다. 이어서 보라색 박스는 모델 training에 해당하며 자세한 과정은 오른쪽 사진에서 설명하겠다. 이어서 갈색 박스는 chatbot 응답을 생성하는 과정이며 마지막 회색 박스는 사용자 입력을 기반으로 응답을 생성한 후 사용자에게 출력해주는 과정이다. 오른쪽 사진은 모델 training과 chatbot 응답 생성을 좀 더 자세하게 나타낸 것이다. 우선 모델 training의 경우 각 에폭에서 각 배치에 대해 정방향 전달을 통해 출력을 얻게 된다. 손실을 계산하고 모델의 가중치를 업데이트 하기 위해 Adam optimizer를 사용해 gradient를 얻는다. 이어서 chatbot 응답 생성의 경우, 사용자 입력 형식을 지정한다. (<usr> 와 <sys>) 이후 형식이 지정된 입력이 토큰 ID로 인코딩 되며 이것을 기반으로 응답을 생성한다. 이후 사람이 읽을 수 있도록 다시 디코딩 된 후 응답을 반환하는 방식으로 동작한다.

Result

```
[1]  # Check Colab Environment
     RunningInCOLAB = 'google.colab' in str(get_ipython())

     # Import tqdm for notbook if Colab
     if RunningInCOLAB:
         from tqdm.notebook import tqdm
     else:
         from tqdm import tqdm

     # Set the Keras backend to TensorFlow
     import os
     os.environ["KERAS_BACKEND"] = "tensorflow"

     # Importing TensorFlow and Keras
     import tensorflow as tf
     import keras
     # Importing required modules from the Transformers library
     from transformers import AutoTokenizer
     from transformers import TFGPT2LMHeadModel
```

우선 현재 코드가 어떤 환경에서 실행 중인지를 확인한다. get_ipython의 결과가 goolge.colab 문자열을 포함하는지 체크하는 방식을 사용한다. Google Colab환경에서 실행될 때는 Jupyter notebook과 같은 환경에 맞게 조정된 tqdm 라이브러리를 가져오며 로컬 환경에서 실행될 때는 일반 tqdm 라이브러리를 가져온다. 이후 Keras의 백엔드를 Tensorflow로 설정한 후 이번 프로젝트에서 필요한 라이브러리들을 import 해주었다.

The GPT2 Model transformer for TensorFlow with a language modeling head on top (linear layer with weights tied to the input embeddings).

If you choose this second option, there are three possibilities you can use to gather all the input Tensors in the first positional argument :

a single Tensor with input_ids only and nothing else: `model(inputs_ids)`

a list of varying length with one or several input Tensors IN THE ORDER given in the docstring: `model([input_ids, attention_mask])` or `model([input_ids, attention_mask, token_type_ids])`

a dictionary with one or several input Tensors associated to the input names given in the docstring: `model({'input_ids': input_ids, 'token_type_ids': token_type_ids})`

https://huggingface.co/transformers/v3.0.2/index.html

이부분은 TensorFlow에서 GTP-2 모델을 사용할 때 입력 데이터를 모델에 전달하는 방법을 설명한다. 첫 번째 방법은 단일 텐서인(input_ids)만 모델에 전달하는 것이다. 두 번째 방법은 (input_ids, attention_mask) 이렇게 여러 텐서 목록을 전달하는 것이다. 세 번째 방법은 (input_ids, attention_mask, token_tupe_ids) 이렇게 세 가지 입력을 모두 포함하는 텐서 목록을 전달하는 것이다.

```
### START CODE HERE ###

# find & assign tokenizer and model; 'skt/kogpt2-base-v2'

# Initialize tokenizer for KoGPT-2 model with special tokens
tokenizer = AutoTokenizer.from_pretrained('skt/kogpt2-base-v2',
                                          bos_token='<s>',
                                          eos_token='</s>',
                                          pad_token='<pad>',
                                          unk_token='<unk>')                # define various tokens while loading tokenizer
model = TFGPT2LMHeadModel.from_pretrained('skt/kogpt2-base-v2', from_pt=True)   # Load the KoGPT-2 model

### END CODE HERE ###
```

```
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFGPT2LMHeadModel: ['transformer.h.3.attn.masked_b
- This IS expected if you are initializing TFGPT2LMHeadModel from a PyTorch model trained on another task or with another architectur
- This IS NOT expected if you are initializing TFGPT2LMHeadModel from a PyTorch model that you expect to be exactly identical (e.g. i
All the weights of TFGPT2LMHeadModel were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFGPT2LMHeadModel for predictions
```

위 코드는 사전 훈련된 KoGPT-2 모델과 해당 tokenizer를 초기화하고 load한다. KoGPT-2는 SK텔레콤이 개발한 한국어 text 학습 GPT-2 모델 버전이다.
AutoTokenizer.from_pretrained 해당 함수는 사전 훈련된 모델과 연결된 토크나이저를 자동으로 로드하는데 사용되며 skt/kogpt2-base-v2는 Hugging Face Model Hub에서의 모델 식별자로 한국어에 맞게 조정된 버전이다. 이어서 beginning of sentence(BOS), end of sentence(EOS), padding token(pad), unknown_token(unk)인 special tokens를 정의해주었다. BOS의 경우 문장의 시작 토큰으로 <s>로 설정하였다. 이어서 EOS는 문장의 끝 토큰으로 </s>로 설정하였다. <pad>는 패딩 토큰으로 텍스트 시퀀스를 동일한 길이로 만드는데 사용된다. <unk>는 모델어휘의 일부가 아닌 입력 토큰을 나타낼 때 사용된다. 이어서 TFGPT2LMHeadModel.from_pretrained함수는 사전 훈련된 KoGPT-2 모델을 텍스트 생성에 적합한 언어 모델링 헤드와 함께 로드한 것이다. 아래 사진과 같은 오류가 발생하여 from_pt =True로 설정하여 from_pretrained 메소드가 pytorch 체크포인트에서 모델 가중치를 로드하도록 해주었다.

```
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
<ipython-input-3-95aed9b5157c> in <cell line: 6>()
      4
      5 tokenizer = AutoTokenizer.from_pretrained("skt/kogpt2-base-v2")          # define various tokens while loading tokenizer
----> 6 model = TFGPT2LMHeadModel.from_pretrained("skt/kogpt2-base-v2")         # load KOGPT model
      7
      8 ### END CODE HERE ###

/usr/local/lib/python3.10/dist-packages/transformers/modeling_tf_utils.py in from_pretrained(cls, pretrained_model_name_or_path, config, cache_dir, ignore_mismatched_sizes, force_downloa
   2871                         is_sharded = True
   2872                     elif has_file(pretrained_model_name_or_path, WEIGHTS_NAME, **has_file_kwargs):
-> 2873                         raise EnvironmentError(
   2874                             f"{pretrained_model_name_or_path} does not appear to have a file named"
   2875                             f" {TF2_WEIGHTS_NAME} but there is a file for PyTorch weights. Use `from_pt=True` to"

OSError: skt/kogpt2-base-v2 does not appear to have a file named tf_model.h5 but there is a file for PyTorch weights. Use `from_pt=True` to load this model from those weights.
```

또한 기본 값이 향후 버전에서 변경될 예정임을 알리는 Future Warning도 발생하였는데 이는 기능에 영향은 주지 않기 때문에 무시하였다.

```
# Display a summary of the model
model.summary()
```

```
Model: "tfgpt2lm_head_model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 transformer (TFGPT2MainLay  multiple                  125164032
 er)

=================================================================
Total params: 125164032 (477.46 MB)
Trainable params: 125164032 (477.46 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

위 코드는 로드한 KoGPT-2 모델의 summary로 레이어, 출력 형태, 매개변수 수에 대한 정보를 포함하고 있다. 모델명은 tfgpt2lm-head-model이며 type은 GPT-2 모델의 핵심 계층으로 transformer 아키텍처를 통해 입력 텍스트를 처리한다. Output shape은 다양한 길이의 입력 시퀀스를 처리하기 때문에 출력 모양이 동적이며 출력된 것을 보면 multiple로 표시된 것을 확인할 수 있다. Param의 경우 학습 가능한 가중치인 125,164,032의 매개변수가 있음을 알 수 있다. 모델의 입력은 일반적으로는 (batch_size, sequence_length) 형태의 텐서로 전달되며 출력은 (batch_size, sequence_length, vocab_size)의 형태로 출력된다.

```
# Access the configuration settings of the model
model.config
```


```
GPT2Config {
  "_name_or_path": "skt/kogpt2-base-v2",
  "_num_labels": 1,
  "activation_function": "gelu_new",
  "architectures": [
    "GPT2LMHeadModel"
  ],
  "attn_pdrop": 0.1,
  "author": "Heewon Jeon(madjakarta@gmail.com)",
  "bos_token_id": 0,
  "created_date": "2021-04-28",
  "embd_pdrop": 0.1,
  "eos_token_id": 1,
  "gradient_checkpointing": false,
  "id2label": {
    "0": "LABEL_0"
  },
  "initializer_range": 0.02,
  "label2id": {
    "LABEL_0": 0
  },
  "layer_norm_epsilon": 1e-05,
  "license": "CC-BY-NC-SA 4.0",
  "model_type": "gpt2",
  "n_ctx": 1024,
  "n_embd": 768,
  "n_head": 12,
  "n_inner": null,
  "n_layer": 12,
  "n_positions": 1024,
  "pad_token_id": 3,
  "reorder_and_upcast_attn": false,
  "resid_pdrop": 0.1,
  "scale_attn_by_inverse_layer_idx": false,
  "scale_attn_weights": true,
  "summary_activation": null,
  "summary_first_dropout": 0.1,
  "summary_proj_to_labels": true,
  "summary_type": "cls_index",
  "summary_use_proj": true,
  "task_specific_params": {
    "text-generation": {
      "do_sample": true,
      "max_length": 50
    }
  },
  "transformers_version": "4.44.2",
  "use_cache": true,
  "vocab_size": 51200
}
```


모델의 구조, 훈련 설정, 하이퍼파라미터 등 다양한 속성을 포함하는 출력 결과로, 주요 속성만 살펴보면 모델의 정보(경로, 모델 유형, 저자, 생성 날짜, 라이선스), 모델의 아키텍처(아키텍처, 활성화, 초기화 범위), 입력 및 출력 토큰(bos_token_id, eos_token_id, vocab_size) 등을 확인할 수 있다.

```python
# Print the token IDs for special tokens
print(tokenizer.bos_token_id)
print(tokenizer.eos_token_id)
print(tokenizer.pad_token_id)
print(tokenizer.unk_token_id)

print('-' * 10)

# Print the first 10 token IDs
for i in range(10):
    print(i, tokenizer.decode(i))
print(tokenizer.decode(51200)) # Print token ID 51200
```

```
0
1
3
5
----------
0 <s>
1 </s>
2 <usr>
3 <pad>
4 <sys>
5 <unk>
6 <mask>
7 <d>
8 </d>
9 <unused0>
```

Tokenizer와 관련된 정보를 출력하는 코드로 BOS, EOS, PAD, UNK의 토큰 ID를 출력한 결과가 위에서 설정한 것에 알맞게 0, 1, 3, 5로 할당되어 출력되는 것을 확인하였다. 이후 처음 10개의 토큰 ID를 인쇄한 것인데 0부터 9까지에 해당 문자열 값을 출력한 것이다. 이서 토큰 ID 512000에 해당하는 토큰 문자열을 출력한 것인데, 0~51199의 범위를 벗어났기 때문에 출력 결과가 없는 것을 확인할 수 있다.

```python
[7]  # Import library
     import pandas as pd
     import urllib.request
```

Import a Korean chatbot dataset made by songys:
https://github.com/songys/Chatbot_data
To find more Korean dataset, check this site:
https://github.com/ko-nlp/Korpora

```python
[8]  # Download the dataset from the provided URL and save it as ChatBotData.csv
     urllib.request.urlretrieve("https://raw.githubusercontent.com/songys/Chatbot_data/master/ChatbotData.csv",
                                filename="ChatBotData.csv")
     # Load the downloaded CSV file into a pandas DataFrame
     train_data = pd.read_csv('ChatBotData.csv')
```

위 코드는 우선 Pandas 라이브러리를 가져오며 이는 데이터 분석을 위해 import 하였다. Urlib.request 라이브러리는 URL을 읽기 위해 import 하였다. 이어서 지정된 URL에서 데이터세트를 다운로드하여 pandas의 DataFrame에 로드한 후 train_data 변수에 할당하였다. ChatBotData.csv 파일은 대화형 AI 모델 또는 chatbot training에 사용되는 데이터세트로 Q-A(질문-답변)쌍으로 포함되어 있다.

```
# Display the contents of the train_data DataFrame
display(train_data)
```

	Q	A	label
0	12시 땡!	하루가 또 가네요.	0
1	1지망 학교 떨어졌어	위로해 드립니다.	0
2	3박4일 놀러가고 싶다	여행은 언제나 좋죠.	0
3	3박4일 정도 놀러가고 싶다	여행은 언제나 좋죠.	0
4	PPL 심하네	눈살이 찌푸려지죠.	0
...	...	...	...
11818	훔쳐보는 것도 눈치 보임.	티가 나니까 눈치가 보이는 거죠!	2
11819	훔쳐보는 것도 눈치 보임.	훔쳐보는 거 티나나봐요.	2
11820	흑기사 해주는 짝남.	설렜겠어요.	2
11821	힘든 연애 좋은 연애라는게 무슨 차이일까?	잘 헤어질 수 있는 사이 여부인 거 같아요.	2
11822	힘들어서 결혼할까봐	도피성 결혼은 하지 않길 바라요.	2

11823 rows × 3 columns

train_data, DataFrame의 내용을 표시하는 코드로, ChatBotData.csv 파일의 내용이 출력된 결과를 볼 수 있다. 위 데이터는 Q, chatbot에 대한 질문 또는 사용자의 입력에 해당하며 A, chatbot의 응답, 답변에 해당한다.

```
[10] def get_chat_data():

        # Get the token ID for the beginning of sentence (BOS) token
        bos_token = tokenizer.bos_token_id
        # Get the token ID for the end of sentence (EOS) token
        eos_token = tokenizer.eos_token_id
        # Get the token ID for the unknown of sentence (UNK) token
        unk_token = tokenizer.unk_token_id
        # Get the maximum token value based on the model's vocab size
        max_token_value = model.config.vocab_size

        conversations = [] # Initialize an empty list to store conversations

        # Iterate over each question and answer pair in training data
        for question, answer in zip(train_data.Q.to_list(), train_data.A.to_list()):

            ### START CODE HERE ###

            # Encode the question and answer as a tokenized sequence
            qna_line = tokenizer.encode('<usr>' + question + '<sys>' + answer)  # encode q & a dialog line

            # Initialize a list to store the dialog tokens with the BOS token
            dialog = [bos_token]

            # Loop through the encoded tokens
            for token in qna_line:
                if token<max_token_value:     # If the token is within the model's vocabulary size
                    dialog.append(token)      # Append the token to the dialog list
                else:                         # If the token is out of vocabulary
                    dialog.append(unk_token)  # Replace it with the unknown token

            dialog.append(eos_token)          # Append the EOS token to the dialog
            ### END CODE HERE ###

            conversations.append(dialog)      # Add the dialog to the conversations list
        return conversations                  # Return the list of conversations
```

위 코드는 CSV파일의 데이터세트를 transformer 기반 chatbot 모델에 입력하기에 적합한 토큰화된 시퀀스로 처리하는 함수를 정의한 것이다. 우선 BOS, EOS, UNK, 토큰 ID를 검색한다. max_token_value의 경우 모델의 vocab에서 최대 토큰 ID를 찾는다. 이후 질문-답변 쌍에 대한 토큰화된 대화 데이터를 저장하는 목록을 초기화한다. train_data.Q.to_list()는 질문 목록을 제공하고, train_data.A.to_list()는 답변 목록을 제공하며 이러한 질문-답변 쌍을 반복하는 것이 for문에 해당한다. zip()을 통해 루프의 각 반복에서 쌍이 함께 처리되도록 하는 것이다. 이어서 <usr> + 질문 + <sys> + 답변 이렇게 형식을 지정하여 사용자의 입력과 시스템의 응답을 표시하였다. 이어서 tokenizer.encode 메소드를 통해 질문-답변 문자열을 모델이 처리할 수 있는 토큰 ID 목록으로 토큰화한다. Dialog는 BOS 토큰으로 초기화한다. 이를 통해 대화의 시작을 표시해 문장이 시작되는 위치를 모델이 알 수 있다. 다음으로 for문에서 인코딩된 질문-답변 쌍인 qna_line을 통과하며 해당 값이 max_token_value 보다 작은 경우 dialog.append(token)하며 없는 경우, 즉 max_token_value 보다 크거나 같은 경우 UNK 토큰으로 대체된다. 이후 EOS 토큰을 추가하여 문장의 끝을 의미한다. 마지막으로 완전히 토큰화된 대화를 list에 추가한 후 반환한다.

```
# Pad the sequences of chat data using the pad token ID to ensure uniform length
chat_data = keras.utils.pad_sequences(get_chat_data(), padding='post', value=tokenizer.pad_token_id)
```

```
[12] buffer = 500     # Set buffer size for shuffling
     batch_size = 32  # Set batch size for the dataset

     # Create a TensorFlow dataset
     dataset = tf.data.Dataset.from_tensor_slices(chat_data)
     dataset = dataset.shuffle(buffer).batch(batch_size,drop_remainder=True)
```

```
[13] # Take one batch from the dataset and print its shape and the first sequence
     for batch in dataset.take(1):
         print(batch.shape)
         print(batch[0])
```

```
(32, 47)
tf.Tensor(
[    0     2 25883 14701  7991     4 10586  9802  9846  9122  8046  8084
   376     1     3     3     3     3     3     3     3     3     3     3
     3     3     3     3     3     3     3     3     3     3     3     3
     3     3     3     3     3     3     3     3     3     3     3], shape=(47,), dtype=int32)
```

우선 첫 번째 코드는 keras.utils.pad_sequences 함수를 사용해 토큰화된 모든 시퀀스를 동일한 길이로 채워주었다. Get_chat_data 함수를 통해 토큰화된 대화를 생성한 후 padding=post로 시퀀스의 끝 부분을 채운다. Value=tokenizer.pad_token_id는 패딩에 PAD 토큰 ID를 사용함을 의미한다. 이어서 두 번째 코드는 buffer 및 batch size를 설정한 후 Tensorflow dataset를 만든 것으로 buffer를 500으로 설정하여 셔플하는 동안 500개의 요소가 다시 셔플되기 전에 샘플링 된다는 것을 의미한다. Batch_size는 32로 32개의 시퀀스가 각 배치에 그룹화 됨을 의미한다. 이후 이를 바탕으로 데이터세트를 만들었다. 세 번째 코드는 데이터세트(32개의 시퀀스)에서 하나의 배치를 뽑아 배치의 shape과

배치의 첫 번째 시퀀스를 출력한 것으로 BOS, EOS, PAD 등의 토큰 ID가 알맞게 출력된 것을 확인할 수 있다.

```
# Decode the first sequence in the batch and print it
str = tokenizer.decode(batch[0])
print(str)
```

```
<s><usr> 공시생이야<sys> 좋은 결과 있을 거예요!</s><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad><pad>
```

```
[15] # Encode the string back into token IDs and print it
print(tokenizer.encode(str))
```

```
[0, 2, 25883, 14701, 7991, 4, 10586, 9802, 9846, 9122, 8046, 8084, 376, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
```

```
[16] # Initialize the Adam optimizer
adam = keras.optimizers.Adam(learning_rate=3e-5, epsilon=1e-08)
```

```
# Initialize the Adam optimizer
adam = keras.optimizers.Adam(learning_rate=3e-5, epsilon=1e-08)

# Calculate the number of training steps
steps = len(train_data) // batch_size + 1
print(steps)
```

```
370
```

첫 번째 코드는 배치의 첫 번째 시퀀스를 decoding한 것으로 문장의 시작, 사용자의 질문 또는 입력, 시스템의 응답(sys) 문장의 끝, 패딩으로 출력된 것을 확인할 수 있다. 이어서 두 번째 코드는 문자열을 다시 token ID로 인코딩하는 코드로 위의 결과와 비교했을 때 인코딩이 잘 됐음을 알 수 있다. 세 번째 코드는 Adam Optimizer를 초기화한다. Learning_rate와 epsilon을 설정해주었다. 이어서 네 번째 코드는 optimizer와 training step 수를 계산하는 코드로 len(train_data) // batch_size는 한 에폭의 훈련에 필요한 배치 수를 계산 한 후 +1을 하여 나머지 샘플이 하나의 최종 배치로 포함된다. 결과가 370이 나왔으므로 에폭당 370개의 training step이 있음을 알 수 있다.

In standard text generation fine-tuning, since we are predicting the next token given the text we have seen thus far, the labels are just the shifted encoded tokenized input. However, GPT's CLM (causal language model) uses look-ahead masks to hide the next tokens, which has the same effect as the labels are automatically shifted inside the model. Therefore, we can set as `labels=input_ids`.

```
EPOCHS = 3 # Set the number of training epochs

# Loop over the number of epochs
for epoch in range(EPOCHS):
    epoch_loss = 0 # Initialize loss

    # Iterate through the dataset
    for batch in tqdm(dataset, total=steps):
        # Record operations
        with tf.GradientTape() as tape:

            ### START CODE HERE ###
            # Forward pass through the model and computing the loss
            result = model(batch, labels=batch)
            # Extract the loss value
            loss=result[0]
            # Calculate the average loss
            batch_loss=tf.reduce_mean(loss)
            ### END CODE HERE ###

        # Compute the gradients of the loss
        grads = tape.gradient(batch_loss, model.trainable_variables)
        # Apply the gradients to the model's weights
        adam.apply_gradients(zip(grads, model.trainable_variables))
        # Accumulate the average loss
        epoch_loss += batch_loss / steps

    # Print the average loss
    print('[Epoch: {:>4}] cost = {:>.9}'.format(epoch + 1, epoch_loss))
```

```
100% 369/370 [07:43<00:01,  1.21s/it]
[Epoch:    1] cost = 1.19391048
100% 369/370 [08:21<00:01,  1.20s/it]
[Epoch:    2] cost = 0.923893332
100% 369/370 [07:38<00:01,  1.21s/it]
[Epoch:    3] cost = 0.746905565
```

우선 코드 위에 정의된 글을 살펴보면, 텍스트 생성을 위한 fine-tuning에서 모델의 목표는 이미 본 텍스트를 기반으로 시퀀스의 다음 토큰을 예측하는 것인데 LSMT이나 Transformer 기반 모델을 사용한 언어 모델링과 같은 일반적인 시퀀스 예측에서는 입력 토큰을 한 위치씩 이동해 모델에 대한 레이블을 만든다. ex) ilovecoding 입력 : ["I", "love", "coding"] 레이블:["love", "coding"] 이는 사실상 입력시퀀스(input)의 한 단계 이동된 버전임을 알 수 있다. 이 변화를 통해 이전 텍스트를 기반으로 향후의 각 토큰을 예측하여 학습할 수 있다. GPT는 다음 토큰을 숨기기위해 look-ahead masks를 적용하는 CLM 아키텍처를 사용하는데, 이 때문에 각 토큰이 뒤에 오는 토큰이 아닌 앞에 오는 토큰만 볼 수 있다. Look-ahead masks는 모델의 forward 전달 중 각 토큰 위치가 아닌 시퀀스의 이전 토큰만 고려할 수 있음을 의미하기 때문에 모델이 이미 과거 토큰을 기반으로 다음 토큰을 예측하도록 구현되었다. 따라서 레이블에 대한 입력의 이동된 버전이 필요하지 않다. 이 때문에 input_ids와 labels는 동일한 시퀀스를 사용할 수 있다.

코드를 살펴보면, 각 에폭 동안 loss를 0으로 초기화한 후 시각적으로 bar를 보여주기 위해 tqdm을 사용해 batch를 반복한다. 이후 tf.GradientTape() 함수를 사용하는데 역전파를 위한 그래디언트를 계산하는데 사용된다. 이후 구현한 부분으로, result=model(batch, labels=batch)를 통해 정방향 전달을 수행하고 손실을 계산한다. Batch는 토큰 시퀀스의 입력 배치이고 위에서 설명한 것을 기반으로 레이블을 입력 자체로 설정하였다. 이후 loss=result[0]은 모델 출력(첫번째 요소)에서 loss를 검색하며 tf.reduce_mean(loss)는 배치 전체의 loss 평균을 계산하는 것이다. 이후 gradient를 계산하고 Adam optimizer를 사용해 gradient를 업데이트 한다. 이후 누적된 에폭 loss를 구하고 print하여 확인한 결과이다. 결과를 살펴보면 epoch을 더 돌려봐야 정확히 알겠지만 cost가 감소하는 것으로 보여 괜찮게 나온다고 생각하였다.

```
[19]  # Add user and sys tags to the text
      text = '오늘도 좋은 하루!'
      sent = '<usr>' + text + '<sys>'
```

```
[20]  # Encode the sentence with the BOS token
      input_ids = [tokenizer.bos_token_id] + tokenizer.encode(sent)
      # Convert the result into a TensorFlow tensor
      input_ids = tf.convert_to_tensor([input_ids])
```

```
[21]  # Generate text from the model
      output = model.generate(input_ids, max_length=50, do_sample=True, eos_token_id=tokenizer.eos_token_id)
```

```
      # Decode the token IDs to a string
      decoded_sentence = tokenizer.decode(output[0].numpy().tolist())
      # Get the part after <sys> and remove the end token </s>
      decoded_sentence.split('<sys> ')[1].replace('</s>', '')
```

```
'그 날 하루 조금 더 참고 기다려봐요.'
```

```
[23]  # Generate text from the model with sampling and top-k sampling
      output = model.generate(input_ids, max_length=50, do_sample=True, top_k=10)
      # Decode the generated token IDs
      tokenizer.decode(output[0].numpy().tolist())
```

```
'<s><usr> 오늘도 좋은 하루!<sys> 좋은 하루!</s>'
```

우선 첫 번째 코드는 입력 텍스트에 <usr>, <sys>를 추가하여 사용자의 입력과 시스템의 응답의 위치를 나타내주는 것이다. 이어서 두 번째 코드는 BOS 토큰이 있는 문장을 인코딩하는 것인데 이때 인코딩된 입력 앞에 BOS 토큰 ID를 추가해준 후 이를 텐서로 변환하였다. 이어서 세 번째 코드는 generate 함수로 제공된 입력을 기반으로 응답을 생성하는데 이때 생성된 응답은 최대 50개의 토큰으로 제한되며, 샘플링을 True로 설정해 random하게 하고 시퀀스 끝 토큰인 EOS를 설정하여 이 토큰이 나타날 시 생성이 중지된다. 이어서 output에서 생성된 시퀀스는 토큰 ID에서 다시 문자열로 변환된다. <sys> 뒤의 텍스트를 추출한 후(시스템 응답의 시작) 출력에 나타나는 시퀀스 끝 토큰인 </s>를 제거하면 그 아래 출력 예시와 같다. 이어서 top_k 샘플링으로 텍스트를 생성하는데 tok_k=10은 각 단계에서 모델의 토큰을 선택하는데 이때 가장 가능성이 높은 10개의 토큰으로 제한됨을 의미한다. 최종으로 생성된 응답은 사람이 읽을 수 있는 문자로 디코딩되어 사용자의 입력에 대한 모델의 답변으로 반환된다. 이를 출력한 결과를 확인하였다.

```
# Function defined to receive chatbot response based on user input
def return_answer_by_chatbot(user_text):
  sent = '<usr>' + user_text + '<sys>' # Format the input by adding usr and sys tags
  input_ids = [tokenizer.bos_token_id] + tokenizer.encode(sent) # Encode the input sentence and add the BOS token
  input_ids = tf.convert_to_tensor([input_ids]) # Convert the input IDs to a TensorFlow tensor
  output = model.generate(input_ids, max_length=50, do_sample=True, top_k=20) # Generate a response from the model
  sentence = tokenizer.decode(output[0].numpy().tolist()) # Decode the output tokens
  chatbot_response = sentence.split('<sys> ')[1].replace('</s>', '') # Extract the chatbot's response
  return chatbot_response  # Return the chatbot's response
```

해당 함수는 사용자의 입력 텍스트를 가져와 chatbot 모델에서 생성된 응답을 반환한다. 우선 user_text를 <usr>와 <sys> 태그로 매핑한다. Ex <usr> 문장 <sys> 이렇게 하면 모델이 사용자 메시지의 어느 부분에서 응답 생성을 시작해야 하는지 알 수 있다. 이어서 KoGPT-2 모델의 토크나이저를 사용해 sent 문자열을 코느화하고 BOS 토큰 ID를 input_ids에 추가해주었다. 이렇게 하면 입력 시퀀스 시작 부분에 BOS 토큰을 추가함으로써 모델이 대화가 시작되는 위치를 알 수 있다. 이어서 입력을 tensorflow tensor로 변환한 후 모델의 generate를 사용해 입력을 기반으로 응답을 생성하였다. 각 인자에 대한 설명은 위에서 이미 했으므로 생략한다. 생성된 출력은 토크나이저를 사용해 문자열로 디코딩된다. 마지막으로 생성된 텍스트 중 <sys>뒤에 챗봇의 응답 부분을 출력한다. 이때 </s>, EOS 토큰이 나타나면 이를 제거한다.

```
# Get the chatbot's response for the given user input
return_answer_by_chatbot('안녕! 반가워~')
```

```
'반갑습니다.'
```

```
# Get the chatbot's response for the given user input
return_answer_by_chatbot('너는 누구야?')
```

```
'누구야?'
```

```
[27] # Get the chatbot's response for the given user input
return_answer_by_chatbot('나랑 영화보자')
```

```
'영화도 보고 뮤지컬도 볼 수 있는 곳이 좋겠어요.'
```

```
[28] # Get the chatbot's response for the given user input
return_answer_by_chatbot('너무 심심한데 나랑 놀자')
```

```
'좋은 소식도 많을 거 같아요.'
```

```
[29] # Get the chatbot's response for the given user input
return_answer_by_chatbot('영화 해리포터 재밌어?')
```

```
'좋아하는 영화 해봐도 될까요.'
```

```
[30] # Get the chatbot's response for the given user input
return_answer_by_chatbot('너 딥 러닝 잘해?')
```

```
'딥 러닝은 스스로 더 잘할 거예요.'
```

```
[31] # Get the chatbot's response for the given user input
return_answer_by_chatbot('커피 한 잔 할까?')
```

```
'깔끔한걸 좋아하면 할 수 있어요.'
```

우선 첫 번째 예시를 보면, 사용자 입력으로 함수를 호출하면

user_text = "안녕! 반가워~ -> sent = <usr>안녕! 반가워~<sys> -> <s><usr>안녕! 반가워~<sys> -> generate -> decode -> "<s><usr>안녕! 반가워~<sys> 반갑습니다.</s>" -> "반갑습니다."

이런식으로 동작함을 알 수 있다. 이런식으로 여러 예시들을 실행해 본 결과 대부분이 알맞게 응답하는 것을 확인하였다.

Discussion

이번 과제는 지금까지 했던 과제들 중 가장 어려웠던 것 같다. 우선 처음 접해보는 라이브러리들이 많았기에 공식 홈페이지를 엄청 찾아봤던 것 같다. Epoch이 3번 밖에 되지 않음에도 불구하고 한 번 돌릴 때 시간이 많이 소요도 됐고 google colab의 GPU의 실행 시간도 한정되어 있기 때문에 과제에 많은 시간을 썼던 것 같다. 결론적으로는 코드를

구현하고 한 줄 한 줄 보고서에 설명하며 완벽하게 이해할 수 있었다. 또한 5차 과제에서 진행한 토크나이저에 대한 개념도 이해하고 있었기에 이해하는데 있어서는 수월했다. 이번 과제를 통해 GPT 동작 방식에 대해서도 공부할 수 있어 좋았다. 항상 느끼지만 이론으로 배웠을 땐 확 이해가 되는 것 같지 않았는데, 과제를 진행하고 시간을 많이 쓰면서 느끼고 배우는 점이 많은 것 같다.