인공지능프로그래밍

Lab6

GPT2 Model for Language Understanding

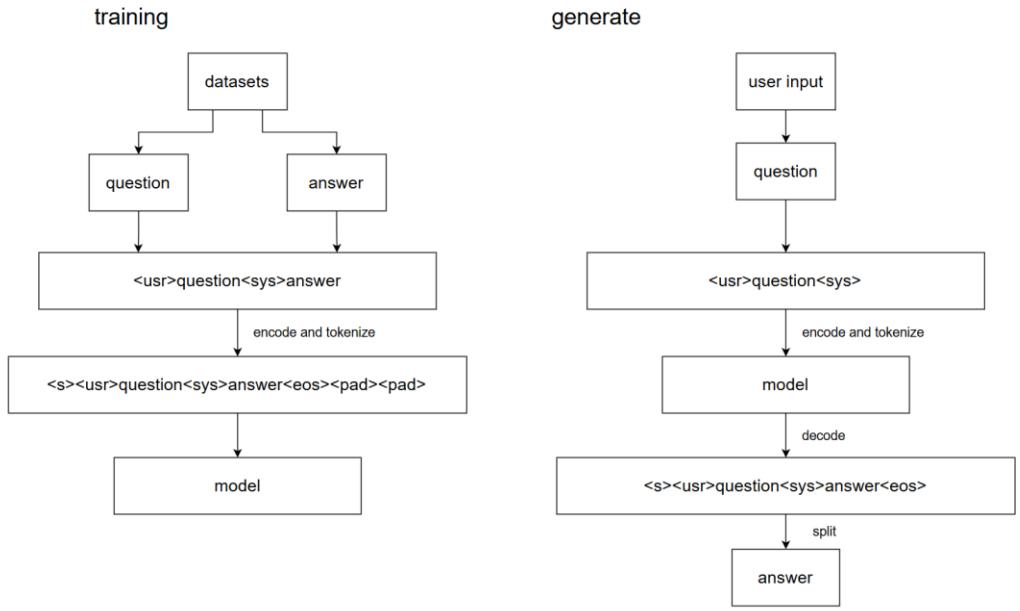
2024/11/07

2019202050 이강현

Lab Objective

훈련된 GPT2 model을 사용하여 언어를 이해하고 학습하는 과정을 살펴본다. 언어 데이터를 전처리하는 과정에서 문장의 토큰화와 인코딩 디코딩 개념을 코드를 통해 살펴본다. 다양한 문장 예시를 통해 모델에 입력되는 문장과 출력되는 문장을 확인하고 코드의 흐름을 이해할 수 있다.

Program Flow



먼저 프로그램의 흐름은 모델의 훈련과정과 생성과정으로 크게 나눌 수 있다. 훈련과정은 데이터셋에서 질문과 답변을 서로 구별할 수 있게끔 토큰을 구분자로 이어 붙이고 encoding한다. 이후 전체 문장의 시작과 끝을 나타내는 토큰을 붙이고 문장들이 모두 고정된 길이를 가지게끔 padding한 후 입력으로 넣게된다. 이렇게 학습된 모델을 통해 질문을 통해 답변을 생성할 수 있게되는데 이것이 생성과정에 나타나 있다. 모델이 학습되고 질문만 토큰을 붙이고 encoding되어 모델의 입력으로 들어오게 되면 출력으로 질문과 답변을 모두 생성하게 되고 이를 decoding하고 split하여 답변만을 출력한다. 위 흐름도에서는 설명을 용이하게 하기 위해 encoding된 문장을 정수 형태로 나타내지 않았다. 세부적인 동작은 코드를 통해 확인한다.

Result

```
#define necessary libraries
RunningInCOLAB = 'google.colab' in str(get_ipython()) #check execution environment

if RunningInCOLAB: # if you run this code in colab
    from tqdm.notebook import tqdm # use tqdm.notebook
else:              # else
    from tqdm import tqdm # use tqdm

import os
os.environ["KERAS_BACKEND"] = "tensorflow" # set the keras backend to tensorflow

import tensorflow as tf
import keras
from transformers import AutoTokenizer # tokenizer
from transformers import TFGPT2LMHeadModel #model
```

먼저 라이브러리를 import하는 부분이다. get_ipython 함수를 통해 실행되는 환경에 대한 문자열을 얻고 만약 google.colab에서 실행된다면 사용되는 tqdm 라이브러리를 다르게 import하겠다는 의미이다. 그 후는 필요한 tensorflow, keras, model, tokenizer를 각각 import하는 부분이다.

```
### START CODE HERE ###
# define tokenizer and model
# find & assign tokenizer and model: 'skt/kogpt2-base-v2'

tokenizer = AutoTokenizer.from_pretrained('skt/kogpt2-base-v2',bos_token='<s>',eos_token='</s>',
                                          unk_token='<unk>', pad_token='<pad>', mask_token='<mask>'
                                          ,clean_up_tokenization_spaces=True)       # define various tokens while loading tokenizer
model = TFGPT2LMHeadModel.from_pretrained('skt/kogpt2-base-v2',from_pt=True)          # load KOGPT model
#load tokenizer and model as pretrained model and set special tokens
### END CODE HERE ###
```

```
/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 secr
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% 1.00k/1.00k [00:00<00:00, 58.0kB/s]
tokenizer.json: 100% 2.83M/2.83M [00:00<00:00, 11.8MB/s]
pytorch_model.bin: 100% 513M/513M [00:03<00:00, 182MB/s]
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFGPT2LMHeadModel: ['transformer.h.2.attn.masked_bia
- This IS expected if you are initializing TFGPT2LMHeadModel from a PyTorch model trained on another task or with another architecture
- This IS NOT expected if you are initializing TFGPT2LMHeadModel from a PyTorch model that you expect to be exactly identical (e.g. ini
```

이 부분은 skt/kogpt2-base-v2로 학습된 tokenizer와 model을 transformer 라이브러리에서 가져오는 부분이다. 먼저 AutoTokenizer는 텍스트를 모델이 이해할 수

있는 토큰으로 변환하여 전처리하는 데 도움이 된다. TFGPT2LMHeadModel은 텍스트 생성 작업을 위해 설계된 GPT-2 모델의 TensorFlow로의 구현이다. LMHead 부분은 시퀀스의 다음 단어를 예측하는 데 사용되는 레이어인 언어 모델링 헤드이다. TFGPT2LMHeadModel은 일반적으로 챗봇이나 텍스트 완성 작업과 같은 애플리케이션에서 일관된 텍스트 응답을 생성하는 데 주로 사용된다고 한다. Eos, bos, unk등 필요한 토큰들을 어떤 문자로 사용할 것인지를 tokenizer의 인자로 넣어줄 수 있다. 이렇게 사전에 훈련된 모델을 사용하게 되면 학습시간을 줄일 수 있고 좋은 데이터셋으로 학습된 모델을 사용하는 것이기에 학습에 용이하다.

```
#summary of created model
model.summary()
```

```
Model: "tfgpt2lm_head_model"
_________________________________________________________________
 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)
```

모델의 요약을 보여주는 부분이다. 파라미터의 수와 크기등의 정보를 얻을 수 있다.

```
# configuration of created 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
  }
```

위는 모델의 구성 요소를 보여준다. 다양한 값들이 내부에 존재하지만 일부분만 가져온 모습이다.

```
# check special token setting
print(tokenizer.bos_token_id) # begin of sentence is <s>
print(tokenizer.eos_token_id) # end of sentense is </s>
print(tokenizer.pad_token_id) # padding is <pad>
print(tokenizer.unk_token_id) # unknown is <unk>

print('-' * 10)

for i in range(10):
    print(i, tokenizer.decode(i)) # list of token decoding
print(tokenizer.decode(51200)) # Check for undefined token
```

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

위는 사용된 주요 토큰인 begin of sentence, end of sentence, padding, unknown의 encoding과 decoding 형태를 확인하는 모습이다. 각각 0,1,3,5로 encoding되는 것을 확인한다. 51200은 모델 어휘의 범위를 벗어나기 때문에 아무것도 반환하지 않는다.

```
# import libraries to load datasets from csv file
import pandas as pd
import urllib.request
```

위는 데이터셋을 가져오고 처리하기 위한 라이브러리를 import하는 모습이다.

```
#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")
train_data = pd.read_csv('ChatBotData.csv') #Load the downloaded CSV file into a pandas DataFrame
```

```
#show loaded datasets
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

위는 라이브러리를 통해 모델 학습시 필요한 데이터를 가져오고 어떤 데이터가 들어있는지 보여주는 모습이다. 질문과 답변 그리고 label로 이루어진 데이터가 11823개 들어있는 것을 확인한다. 이는 csv파일내에 들어있으며 모두 쌍을 이루는 특징이 있다. 행 단위로 하나씩 뽑아 사용할 것이다.

```
#manufacturing datasets
def get_chat_data():

    bos_token = tokenizer.bos_token_id          # get token_id for begin of sentence token
    eos_token = tokenizer.eos_token_id          # get token_id for end of sentence token
    unk_token = tokenizer.unk_token_id          # get token_id for unknown word token
    max_token_value = model.config.vocab_size   # range of words that can be processed

    conversations = [] # empty list to append sentences
    for question, answer in zip(train_data.Q.to_list(), train_data.A.to_list()):

        ### START CODE HERE ###

        qna_line = tokenizer.encode('<usr>' + question + '<sys>' + answer)  # encode q & a dialog line

        dialog = [bos_token]        # replace overshooting tokens with unk and enclose with bos and eos
        for token in qna_line:
            if token<max_token_value: # if token is less than max_token_value
                dialog.append(token) # append token
            else:
                dialog.append(unk_token) # append unknown token
        dialog.append(eos_token) # append end of sentence token

        ### END CODE HERE ###

        conversations.append(dialog) # finally insert created sentence
    return conversations # return all sentences
```

위는 데이터를 처리하는 부분이다. 먼저 encoding된 token들을 저장해두고 질문과 답변을 이어 붙이고 적절하게 토큰을 이용해 구분한다. 이어 붙인 문장은 encoding하여 사전에 encoding해둔 token과 붙인다. Begin of sentence는 <s>로, end of sentence는 </s>로, 벗어난 단어는 <unk>이다. 해당 함수는 그렇게 만들어진 문장들을 conversation에 모두 append하여 return한다. 만들어진 문장에 대한 내용은 아래 예시에 존재한다.

```
# fill pad in sentences for uniform length
chat_data = keras.utils.pad_sequences(get_chat_data(), padding='post', value=tokenizer.pad_token_id)
```

위는 모든 문장을 고정된 길이로 만들기 위해 padding하는 부분이고, 결과적으로 빈 공간에 <pad>라는 토큰이 채워진다.

```
# data shuffling and setting size of batch
buffer = 500
batch_size = 32
#create tensorflow dataset
dataset = tf.data.Dataset.from_tensor_slices(chat_data)
dataset = dataset.shuffle(buffer).batch(batch_size,drop_remainder=True)
```

위는 데이터를 500개씩 뽑아 섞고 학습시에는 32개씩 batch형태로 만들어 학습하도록 dataset을 만드는 부분이다.

```
# extract as batch size
for batch in dataset.take(1):
    print(batch.shape) # print size
    print(batch[0])    # print one sample
```

```
(32, 47)
tf.Tensor(
[    0     2  9718  7182  7601 10648  8006     4 41664  8102  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     3], shape=(47,), dtype=int32)
```

위는 dataset에서 1개의 batch만 뽑아내어 어떤 형태로 저장되어 있는지 확인하는 부분이다. Batch 크기가 32이므로 문장의 개수는 32개임을 확인한다. 먼저 0은 <s> 문장의 시작을 나타내는 토큰이고 2는 <usr> user가 입력한 문장의 시작점을 나타내는 토큰이다. 그 후 질문에 해당하는 문장이 encoding, tokenize되어 저장되고 4는 <sys> 이제 답변이 시작되는 부분을 나타내는 토큰에 해당한다. 그 후 답변이 encoding, tokenize되어 저장되어 있고 끝으로 1은 </s>로 문장의 끝을 나타낸다. 이후 모든 문장을 고정된 길이로 만들기 위한 padding 3 <pad>가 저장되어 있는 것을 확인한다. 따라서 모든 문장은 47이고 batch 하나당 32 x 47의 크기를 가진다.

```
# show one decoded sentence
str = tokenizer.decode(batch[0])
print(str)
```

```
<s><usr> 꽃다발 받았어<sys> 부러워요!</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><pad>
```

```
# show one encoded sentence
print(tokenizer.encode(str))
```

```
[0, 2, 9718, 7182, 7601, 10648, 8006, 4, 41664, 8102, 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, 3]
```

위는 encoding된 문장과 decoding된 문장의 차이를 보인다.

```
# set optimizer
adam = keras.optimizers.Adam(learning_rate=3e-5, epsilon=1e-08)
# how many steps do we have?
steps = len(train_data) // batch_size + 1
print(steps)
```

```
370
```

위는 사용할 optimizer와 훈련데이터를 배치 크기로 나누어 총 step 수를 계산하는 부분이다. Optimizer는 adam을 사용한다.

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`.

위는 훈련에서 사용할 label은 입력을 그대로 넣을 것인데 실제로는 한칸 shift되어 들어가 입력으로 begin of sentence 토큰이 들어갔을 때 질문의 첫 단어를 예측하도록 모델을 학습시켜야 하는데 이를 내부에서 자동으로 shift 해준다는 의미이다.

```
# number of iteration
EPOCHS = 3

for epoch in range(EPOCHS):
    epoch_loss = 0

    for batch in tqdm(dataset, total=steps):
        with tf.GradientTape() as tape:
            #forward
            ### START CODE HERE ###
            result = model(batch, labels=batch, training=True) # insert batch to model with label and train
            loss = result[0] # get loss
            batch_loss = tf.reduce_mean(loss) # calculate mean loss

            ### END CODE HERE ###
        #backpropagation
        grads = tape.gradient(batch_loss, model.trainable_variables) # gradient
        adam.apply_gradients(zip(grads, model.trainable_variables)) # optimizer step
        epoch_loss += batch_loss / steps # loss per epoch

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

```
100% 369/370 [09:21<00:01,  1.37s/it]
[Epoch:    1] cost = 1.25304067
100% 369/370 [08:21<00:01,  1.30s/it]
[Epoch:    2] cost = 1.00575387
100% 369/370 [08:09<00:01,  1.35s/it]
[Epoch:    3] cost = 0.889541388
```

위는 모델을 학습하는 부분이다. Batch를 입력으로 넣어주고 labels=batch로 라벨 또한 batch를 넣어준다. Labels 인자를 채우게 되면 자동으로 모델의 예측과 라벨의 loss를 계산하여 모델의 첫번째 return으로 내보낸다. 따라서 loss는 result[0]으로 얻을 수 있고 이것의 평균을 구하면 batch_loss를 구할 수 있다. 이를 기반으로 adam을 통한 backpropagation을 진행하는 것이 모델의 학습과정이다.

아래는 모델에 질문이 들어왔을 때 답변을 생성하는 과정이다.

```
# make question example with tokens
text = '오늘도 좋은 하루!'
sent = '<usr>' + text + '<sys>'
```

오늘도 좋은 하루라는 문장 앞에 질문을 나타내는 토큰과 답변을 나타내는 토큰을 붙인다. 답변은 존재하지 않기 때문에 모델이 <sys>뒤에 답변을 생성해줄 것을 예상할 수 있다.

```
# concatenate begin of sentence and encoded question
input_ids = [tokenizer.bos_token_id] + tokenizer.encode(sent)
input_ids = tf.convert_to_tensor([input_ids]) # convert to type of tensor
```

문장을 encoding하고 문장의 앞에 begin of sentence 토큰을 붙여 tensor형태로 변환한다.

```
#model's output
output = model.generate(input_ids, max_length=50, do_sample=True, eos_token_id=tokenizer.eos_token_id)
```

Model의 입력으로 encoding된 문장을 넣어 답변을 생성하게 한다. 이때 output은 답변의 개수 x 문자열의 형태로 생성되게 된다.

```
# decode output
# split based on '<sys>' and remove end of sentence token
decoded_sentence = tokenizer.decode(output[0].numpy().tolist())
decoded_sentence.split('<sys> ')[1].replace('</s>', '')
```

'좋은 하루만 있을 거예요.'

output에서 0번째인 문자열을 뽑아내고 이를 decoding한 후 답변만을 뽑아내기 위해 생성된 문장에서 <sys>를 기준으로 문장을 분할하고 end of sentence를 공백문자로 바꾸는 모습이다.

```
# extract top 10 tokens
output = model.generate(input_ids, max_length=50, do_sample=True, top_k=10)
tokenizer.decode(output[0].numpy().tolist())
```

'<s><usr> 오늘도 좋은 하루!<sys> 축하해요!</s>'

위는 오늘도 좋은 하루라는 똑 같은 질문을 넣었을 때 top_k인자를 다르게 하여 축하해요라는 답변을 얻는 것을 보여준다. Top_k=10이라는 것은 다음 단어를 예측할 때 그 단어가 나올 확률이 가장 높은 상위 10개 단어 중에 고르도록 하여 제일 높은 확률을 가진 단어만을 뽑는 것이 아니라서 좀 더 부정확할 수는 있으나 더 창의적이고 다양한 답변을 뽑아낼 수 있다.

```
# make function
def return_answer_by_chatbot(user_text):
  sent = '<usr>' + user_text + '<sys>' # make question
  input_ids = [tokenizer.bos_token_id] + tokenizer.encode(sent) # concatenate begin of sentence token
  input_ids = tf.convert_to_tensor([input_ids]) # convert to tensor
  output = model.generate(input_ids, max_length=50, do_sample=True, top_k=20) # extract top 20 tokens
  sentence = tokenizer.decode(output[0].numpy().tolist()) # decode model's output
  chatbot_response = sentence.split('<sys> ')[1].replace('</s>', '') #split question and answer and remove question
  return chatbot_response # return answer
```

질문을 모델의 입력에 맞게 처리하는 것부터 답변을 반환하는 것까지의 위 과정을 함수화 한 것이다.

```
[23] return_answer_by_chatbot('안녕! 반가워~') # example
```

```
'안녕이네요.'
```

```
[24] return_answer_by_chatbot('너는 누구야?')# example
```

```
'저는 이 사람입니다.'
```

```
[25] return_answer_by_chatbot('나랑 영화보자')# example
```

```
'같이 보자고 해보세요.'
```

```
return_answer_by_chatbot('너무 심심한데 나랑 놀자')# example
```

```
'같이 놀자고 말해보세요.'
```

```
[27] return_answer_by_chatbot('영화 해리포터 재밌어?')# example
```

```
'영화는 영화처럼 여러명이 함께해요.'
```

```
[28] return_answer_by_chatbot('너 딥 러닝 잘해?')# example
```

```
'딥 러닝이 가능한지 생각해보세요.'
```

```
[29] return_answer_by_chatbot('커피 한 잔 할까?')# example
```

```
'제가 마시기 좋은 음료 추천 좀 해주세요.'
```

위는 다양한 질문에 따라 모델이 답변한 예시를 보여준다.

Discussion

LLM 모델 중 하나인 TFGPT2LMHeadModel을 사용하여 질문에 대한 답변을 생성하는 과제를 진행해보았다. Pretrain된 model과 tokenizer를 사용하는 법과 추가로 데이터셋으로 학습시켜보면서 미리 학습된 모델을 나의 데이터셋으로 fine-tuning할 수 있겠다라는 생각을 했다. 한 가지 어려운 부분은 fine-tuning하는 부분이었는데 모델을 직접 구현하지 않고 가져온다라는 것은 큰 장점이지만 그 내부를 정확히 알기 어렵다는 단점이 있었다. 모델의 labels 인자에 데이터를 전달하게 되면 loss를 계산하여 첫번째로 return 한다는 것을 공식 source에서는 알기 어려웠는데 이는 TFGPT2LMHeadModel이 또 다른 기본 모델을 상속받아 만들어진 모델이며 그 모델에서 구현되어 있는 것이라 잘 알지 못했다. 추후 모델을 구현할 때 document를 잘 구성하는 것이 중요할 것 같았다.