

Student ID: 2019147531

Name: 박윤정

Q1. Describe Transformer model based on its three main properties: Self-attention, Multi-head attention, and Parallel operation (one-page answer, 25 points).

Transformer 는 "Attention is All you need"란 논문에서 처음 제안된 architecture 로 multi-head self-attention 을 이용해 sequential computation 을 줄여 더 많은 부분이 병렬처리 가능하게 하면서 동시에 더 많은 토큰 간의 dependency 를 모델링합니다. 이런 transformer 는 sequence-to-sequence 과제 수행에 특화된 모델로 인코더, 디코더 구조로 이루어져있습니다. 인코더와 디코더는 인코더 블록과 디코더블록 여러 개로 구성되어 있으며 인코더 블록은 multi-head attention 과 feedforward NN, residual connection(ADD)&Norm(레이어정규화)로 이루어져있습니다. 그리고 디코더 블록은 Masked Multi-Head Attention(masked decoder self-attention)과 Multi-head attention (encoder-decoder attention), feedforward NN, residual connection(ADD) & Norm (레이어정규화)로 이루어져있습니다.

먼저 self-attention 은 자기 자신에 수행하는 attention 기법으로 인코더 블록과 디코더 블록에서 모두 쓰입니다. 인코더 블록에서 self-attention 은 단어를 인코딩할 때 더 관련있는 단어를 강하게 반영하는 데 사용합니다. 먼저 문장내 각 단어가 embedding 과 positional encoding 을 통해 벡터 형태로 바뀌고 각 벡터가 행렬의 행이 되는 matrix X 에 학습가능한 행렬 W_q, W_k, W_v 를 곱해 query, key, value 세가지로 변환되고 query 행렬과 $\text{transpose}(K)$ 를 곱한 값에 K 의 dimension 의 root 값을 나눠주면 attention score 를 갖는 행렬이 됩니다. 이 행렬에 softmax 함수를 사용한 후 V 행렬을 곱하면 각 단어의 attention 값을 갖는 attention value 행렬이 나옵니다. 이 행렬은 문장에 속한 모든 단어쌍 사이의 관계가 녹아 있습니다.

그 중 multi-head attention 은 self-attention 을 병렬적으로 사용하였다는 의미입니다. 한번의 attention 을 하는 것보다 여러 번의 attention 을 병렬로 사용하는 것이 더 효과적이고 논문에서는 8 개의 병렬어텐션을 하였습니다. 이때, 각각의 attention value 행렬을 attention head 라고 하고 $W_{q1}, \dots, W_{q8}, W_{k1}, \dots, W_{k8}, W_{v1}, \dots, W_{v8}$ 과 같이 가중치 행렬(learnable)은 8 개의 attention head 마다 전부 다릅니다. 이렇게 attention 을 병렬로 수행하면서 다른 시각에서 단어사이의 연관도를 볼 수 있습니다. Parallel attention 수행 후 모든 attention head 를 concatenate 한 후 다른 가중치 행렬 W_0 와 곱해 나온 값이 multi-head attention 의 최종값이고 FFNN 을 지나 다음 인코더 블록으로 입력됩니다.

디코더에서도 인코더와 마찬가지로 embedding+positional encoding 을 거쳐 matrix 형태로 입력되고 이때 현재 시점의 예측시 미래 단어를 참고 못하도록 masked-multi-head self-attention 을 거칩니다. 그런 다음 Encoder-decoder multi-head attention 을 거치는데 이는 key 와 value 가 encoder 에서 오고 query 는 decoder 에서 와 self-attention 이 아닙니다. 그런 다음

position-wise FFNN 을 거치고 이런 식으로 6 개의 decoder block 을 거친 다음 dense layer 와 softmax 를 통과한 후 최종 output 을 냅니다.

RNN 의 경우 순차적으로 정보를 처리해서 parallel operation 이 불가능 하였지만 transformer 는 RNN 구조를 사용하지 않고 attention 만을 사용했기에 병렬적으로 input 을 처리할 수 있고 대신 positional encoding 을 통해 위치 정보를 전달합니다.

Q2. Explain your own idea to advance or extend the first term-project (landmark engine) using Transformer/Language models and other NLP benchmark datasets (one-page answer, 25 points).

(Evaluation factors are Soundness, Impact, Creativity, Clarity, and overall Quality)

당시 추가적인 input 으로 Seattle landmark 이름들을 주고 각 visual cluster 의 hashtag 들과 비교하려고 하였는데 어려웠던 점으로 tag.csv file 에 있던 해시태그의 경우 띄어쓰기가 되어 있지 않았고 띄어쓰기 되지 않은 한 해시태그 안에서의 단어들의 조합 순서가 다양했고 한 랜드마크에 대해 여러 명칭이 있었던 것이었습니다. 따라서, 띄어쓰기를 하는 것이 landmark 를 recognition 하는 데 도움이 될 거라고 생각했습니다.

따라서 Hashtag segmentation 에 대해 찾아보다 Zero-shot hashtag segmentation for multilingual sentiment analysis 논문에서 제안된 hashtag segmentation framework 를 이용하여 hashtag 들을 먼저 segmentation 하는 방식을 생각해보았습니다. 이 모델에서 제안한 hashtag segmentation framework 는 pretrained language model 인 GPT-2 를 segmenter 로 BERT 를 re-ranker 로 사용하고 변형된 beam search 방식을 사용하였습니다. (language model 로 각 time step 에서 확률이 높은 단어나 토큰 구하고 수정된 beam search 에 따라 expand 하는 방식)

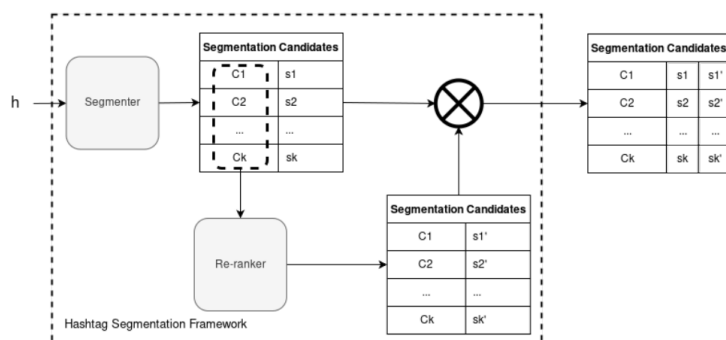
hashtag 가 주어졌을 때, segmenter 에서 이 논문에서 다르게 정의한 beam search algorithm 에 따라 segmentation 후보인 C_i 를 score s_i 와 함께 output 하고, 다시 re-ranker 을 통해 각 C_i 에 대해 새로운 score(masked language model scoring 방식으로 구한 값)인 s'_i 를 output 합니다. 그런 다음, segmenter 와 re-ranker 에서 구해진 score 들을 정의된 Ensembler 를 통해 합치는 방식으로 segmentation 후보 중 하나를 고릅니다.

이 방식을 이용해 다음과 같이 hashtag 를 띄어쓰기 한 후,

예시) seattleaquarium-> seattle aquarium

Instagramvenue name centurylinkfied -> Instagram venue name centurylink fied

Seattle 의 각 landmark 를 key 로, 해당하는 hashtag 조합들(띄어쓰기 되어 있는)을 value 로 하는 dictionary 를 추가적인 input 으로 주고 segment 된 visual cluster 들의 tag 들과 비교하여 어떤 landmark 인지 판단하는 방식을 생각해보았습니다.



Q3. Perform the inference phase using Transformer-based language models (e.g., BERT) fine-tuned on one of "Machine Reading Comprehension", "Commonsense Reasoning", "Information Retrieval", or "Dialogue Generation" benchmarks. Briefly describe your selected task/model/performance (your code or github link should be attached to your assignment submission). Then, analyze the error cases (i.e., incorrect inference) of the model to discuss its limitation. (unlimited length, 50 points).

Code link:

https://colab.research.google.com/drive/1d3XPLgp_mBQwErwQwhpwXMgs8mpQqiRK?usp=sharing

<selected task: MRC>

Machine reading comprehension, commonsense reasoning, information retrieval, dialogue generation 중 제가 선택한 task는 Machine reading comprehension으로 주어진 text를 모델이 읽고 이해했는지 확인하는 것입니다. 이는 모델에 context와 question을 input으로 주고 context내 구절을 output하여 ground truth와 비교해 학습하는 question answering의 type입니다.

<Model>

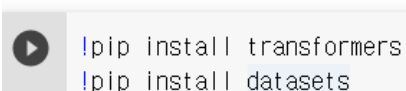
사용한 모델은 huggingface의 bert-large-uncased-whole-word-masking-finetuned-squad 모델입니다. BERT는 transformer-based language 모델로 MLM, NSP 두가지 방식으로 대량데이터에 대해 사전 훈련된 모델입니다. Wikipedia, bookcorpus dataset을 사용하여 한 단어에 해당하는 모든 토큰이 한번에 마스킹 되고(whole word masking) 마스킹된 토큰을 독립적으로 예측하도록 pretraining한 후, SQuAD dataset(ver1.1)을 이용해 fine-tuning된 모델입니다. SQuAD dataset은 Stanford Question Answering Dataset으로 pretrained BERT 모델은 query와 그에 대한 답을 갖고 있는 passage(context)가 주어졌을 때, 답에 해당하는 span을 찾으려(start index와 end index를 찾으려) fine-tuning되었습니다. 따라서, 이 모델은 context와 query가 주어졌을 때, answer를 output하는데 사용할 수 있습니다.

<Performance>

이 모델의 f1 score는 93.15이고, exact_match값은 86.91입니다.

먼저 필요한 library등을 import해줍니다.

import libraries



```
!pip install transformers
!pip install datasets
```

```
[ ] import pandas as pd
import numpy as np
import torch
from transformers import BertForQuestionAnswering #model
from transformers import BertTokenizer #tokenizer
```

그리고 inference에 사용할 SQuAD ver1.1 dataset을 load합니다.

dataset loading from huggingface

```
[ ] from datasets import load_dataset

dataset = load_dataset("squad")
```

Downloading builder

그 중 학습에 사용하지 않은 validation dataset을 사용합니다.

Use SQuAD validation dataset to inference model fine-tuned on SQuAD train dataset

```
[ ] test_data = dataset["validation"]
```

```
[ ] print(test_data)
```

```
Dataset({
  features: ['id', 'title', 'context', 'question', 'answers'],
  num_rows: 10570
})
```

그리고 fine-tuned BERT model과 tokenizer를 load합니다.

load model, tokenizer

```
▶ model = BertForQuestionAnswering.from_pretrained('bert-large-uncased-whole-word-masking-finetuned-squad')
tokenizer = BertTokenizer.from_pretrained('bert-large-uncased-whole-word-masking-finetuned-squad')
```

⏏ Downloading: 100%  443/443 [00:00<00:00, 12.8kB/s]
 Downloading: 100%  1.25G/1.25G [00:27<00:00, 54.3MB/s]
 Downloading: 100%  226k/226k [00:00<00:00, 311kB/s]
 Downloading: 100%  28.0/28.0 [00:00<00:00, 85.9B/s]

Test dataset에서 index에 해당하는 question, text, GT를 이용해 inference합니다.

Inference stage

```
[ ] random_num = 581 #choose random index in test dataset
```

```
[ ] # random_num = np.random.randint(0, len(test_data))
question = test_data["question"][random_num]
text = test_data["context"][random_num]
GT = test_data["answers"][random_num]
```

```
[ ] # print(test_data[84])
```

```
[ ] # print(GT['text'][:])
```

```
[ ] print("random_num is {}".format(random_num))
```

random_num is 581

Encoding using loaded tokenizer

```
[ ] input_ids = tokenizer.encode(question, text)
print("The input has a total of {} tokens.".format(len(input_ids)))
```

The input has a total of 81 tokens.

```
[ ] tokens = tokenizer.convert_ids_to_tokens(input_ids)
# for token, id in zip(tokens, input_ids):
#     print('{:8}{:8}'.format(token, id))
```

그리고 BERT 모델에 input으로 주기 위해 tokenizer를 사용해 question과 text를 encoding합니다.

segment embedding to differentiate question, text

```
[ ] #first occurrence of [SEP] token
sep_idx = input_ids.index(tokenizer.sep_token_id)
print("SEP token index: ", sep_idx)
#number of tokens in segment A (question) - this will be one more than the sep_idx as the
num_seg_a = sep_idx+1
print("Number of tokens in segment A: ", num_seg_a)
#number of tokens in segment B (text)
num_seg_b = len(input_ids) - num_seg_a
print("Number of tokens in segment B: ", num_seg_b)
#creating the segment ids
segment_ids = [0]*num_seg_a + [1]*num_seg_b
#making sure that every input token has a segment id
assert len(segment_ids) == len(input_ids)
```

SEP token index: 10
Number of tokens in segment A: 11
Number of tokens in segment B: 70

이때, question과 text를 구별하기 위해 segment embedding해줍니다.

feed question, text to loaded model

```
[ ] #token input_ids to represent the input and token segment_ids to differentiate our segmer
    output = model(torch.tensor([input_ids]), token_type_ids=torch.tensor([segment_ids]))
```

모델에 input_ids과 segment_ids를 입력으로 주고 답을 예측하게 합니다.

get most probable start, end words answers are provided only if the end token is after the start token

```
[ ] print("\nText:\n{}".format(text.capitalize()))
    print("\nQuestion:\n{}".format(question.capitalize()))

    #tokens with highest start and end scores
    answer_start = torch.argmax(output.start_logits)
    answer_end = torch.argmax(output.end_logits)
    print("\nanswer_start is {}".format(answer_start))
    print("answer_end is {}".format(answer_end))
    # print(tokens[answer_start:answer_start+1])
    # print(tokens[answer_end:answer_end+1])

    if answer_end >= answer_start:
        answer = " ".join(tokens[answer_start:answer_end+1])
        print("\nPredicted Answer:\n{}".format(answer.capitalize()))
    else:
        print("\nPredicted Answer:")
        print("unable to find the answer to this question. Can you please ask another question")

    print("\nOriginal answer:\n{}".format(GT))
    # print("\nQuestion:\n{}".format(question.capitalize()))
    # print("\nAnswer:\n{}".format(answer.capitalize()))
```

이때, start일 확률이 가장 높은 token과 end일 확률이 가장 높은 token을 구하고 그 token들 사이의 구절이 질문에 대한 답이 됩니다. 따라서, start의 위치가 end보다 뒤에 있으면 답을 구할 수 없다고 print합니다.

잘 나온 결과의 예시로는 다음과 같습니다.

```
Text:
Super bowl 50 was an american football game to determine the champion of the national football league (nfl) for the 2015 season. the american football conference (afc) champion d

Question:
Which nfl team represented the nfc at super bowl 50?

answer_start is 57
answer_end is 58

Predicted Answer:
Carolina panthers.

Original answer:
{'text': ['Carolina Panthers', 'Carolina Panthers', 'Carolina Panthers'], 'answer_start': [249, 249, 249]}
```

Incorrect answers의 예시는 다음과 같습니다.

1) Start token index가 end token index보다 큰 경우

```
text:
In 1893, george westinghouse won the bid to light the 1893 world's columbian exposition in ct

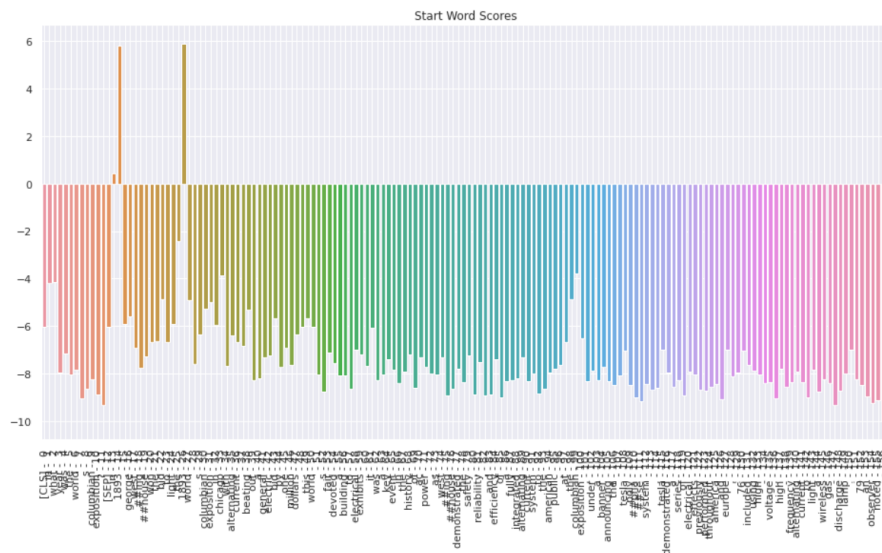
Question:
In what year was the world's columbian exposition?

answer_start is 26
answer_end is 14
['1893']
['1893']

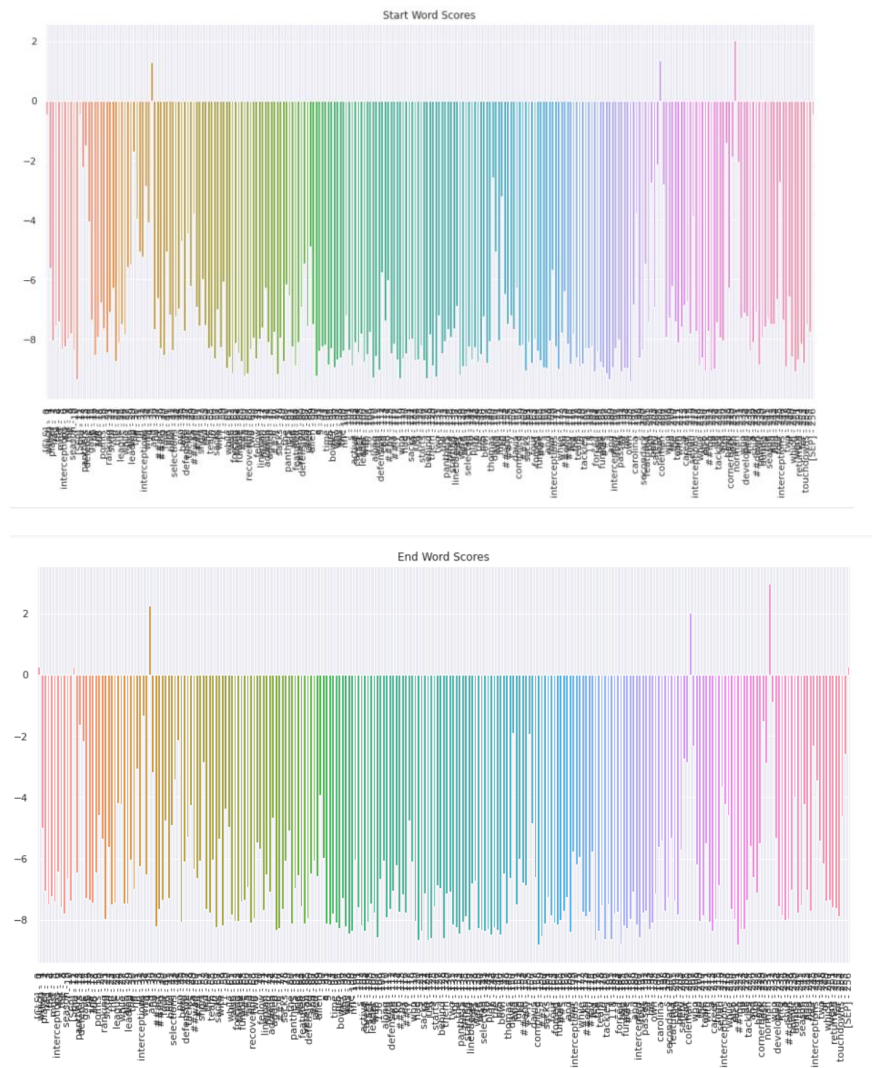
Predicted Answer:
unable to find the answer to this question. Can you please ask another question?

Original answer:
{'text': ['1893', '1893', '1893'], 'answer_start': [3, 54, 3]}
```

Start word의 score와 end word의 score를 확인했을 때, 1893인 두 token 위치(index)가 비슷한 start, end score를 가짐을 확인할 수 있었습니다. start와 end index를 예측하고 두 index사이의 단 어들을 answer로 예측하도록, 즉 context내의 span을 예측하도록 하였기에 context내에 답이 불 연속적으로 여러 번 등장할 경우, 이 case처럼 먼저 나온 단어가 end word로써의 가장 높은 확률 값을 갖게 예측되고, 뒤에 나온 단어가 start word로써 가장 높은 확률값을 갖게 예측된 경우 문 제가 생김을 알 수 있습니다.



야 하는데, 4번한 josh norman이 나온 case입니다.



이 역시 interception을 가장 많이 한 선수를 답해야하는데, interception과 which player에 연관되어 있을 만한 span을 예측한 것으로 볼 수 있습니다.

따라서, error cases을 통해 SQuAD ver1.1에 fine-tuned된 bert모델의 한계에 분석해보자면

먼저 SQuAD dataset은 answerable한 질문들에 초점이 맞춰져 있어서 항상 답이 주어진 context 내에 존재할 거라고 가정하여 모델이 context를 읽고 이해하여 질문에 대한 답이 존재하는 지 여부를 따지지 않고 가장 답에 근접해보이는 span을 찾는 데 집중한 것으로 이 Q&A task로 진정한 machine reading comprehension이라고 볼 수 있는 가라는 한계가 있습니다.

마찬가지로 2번, 3번 case도 질문에 있는 특정 단어들을 근거로 contexts내에서 가장 답일 것 같은 span을 찾은 것으로 볼 수 있습니다. 또, 주어진 context에서 답일 span을 찾는 방식 즉 start와 end token을 찾는 방식이기 때문에 정답이 연속적이지 않은 경우, 1번 case와 같이 문제가 생기는 것을 확인하였습니다.

따라서, unanswerable한 question을 포함한 데이터셋에 대해 추가적으로 fine-tuning하고 답하기 전 passage를 읽고 답할 수 있는 질문인가를 학습하는 것과 불연속적인 답을 찾는 것을 학습하면 개선하는데 도움이 될 것이라고 생각합니다.

또 질문의 단어들로 matching하는 방식으로 예측하는 것이 아니라 passage를 읽고 이해하여 답할 수 있도록 adversarial example을 paragraph안에 넣어 fine-tuning하는 것도 개선에 도움이 될 것이라고 생각합니다.

<참고자료>

<https://mccormickml.com/2020/03/10/question-answering-with-a-fine-tuned-BERT/>

<https://towardsdatascience.com/question-answering-with-a-fine-tuned-bert-bc4dafd45626>

<https://medium.com/analytics-vidhya/bert-question-answering-finetune-visualization-f86198279c13>

<https://u.cs.biu.ac.il/~yogo/squad-vs-human.pdf>

<https://arxiv.org/pdf/2112.03213.pdf> (ZERO-SHOT HASHTAG SEGMENTATION FOR MULTILINGUAL SENTIMENT ANALYSIS)