Euron 최종 발표

- 유린이들(김민지, 김정현, 조현지)

목차

지원 공모전

지원 공모전 : 도배하자 질의응답 처리
- 한솔데코 시즌2 AI 경진대회

목표 : NLP(자연어 처리) 기반의 QA(질문-응답) 시스템을 통해
도배하자와 관련된 깊이 있는 질의응답 처리 능력을 갖춘
AI 모델 개발

EWHA,
THE FUTURE
WE CREATE

LLM(Large Language Model)이란

대규모 언어 모델(LLM)은 대화 또는 기타 자연 언어 입력에 대해 인간과 유사한 응답을 생성하기 위해 방대한 양의 텍스트 데이터에 대해 훈련된 인공 지능의 하위 집합

이러한 자연어 응답을 생성하기 위해 LLM은 다층 신경망을 사용하여 복잡한 데이터를 처리, 분석 및 예측하는 심층 학습 모델을 사용

목표

- **정량 목표** : Baseline 코드(0.5811)보다 개선된 (score 0.70이상)

  평가 지표 : Cosine Similarity(코사인 유사도)

- **정성 목표** : LLM에 대해 이해하고 활용 능력 기르기

실험

1. 데이터 EDA

2. 데이터 전처리

- 특수문자 제거
- nltk를 이용한 데이터 증강

3. Fine Tuning

- scheduler이용
- optimizer 변경 (adamw, adamWR, SGDWR)
- learning rate, batch size, epochs 조정

실험 - 데이터 EDA

훈련데이터

- 데이터 6440행 / 질문1, 질문2, category, 답변 1~5 총 8가지 feature
- 유사한 질문 2개에 대한 답변 5가지로 구성
- 답변 5가지는 주요 단어의 순서만 바뀐 문장

	id	질문_1	질문_2	category
0	TRAIN_000	면진장치가 뭐야?	면진장치에 사용되는 주요 기술은 무엇인가요?	건축구조

답변_1	답변_2	답변_3	답변_4	답변_5
면진장치란 지반에서 오는 진동 에너지를 흡수하여 건물에 주는 진동을 줄여주는 진동 ...	면진장치란 건물의 지반에서 발생하는 진동 에너지를 흡수하여 건물을 보호하고, 진동을...	면진장치란 지반으로부터 발생하는 진동 에너지를 흡수하여 건물에 전달되는 진동을 줄여...	면진장치는 건물의 지반으로부터 오는 진동 에너지를 흡수하여 건물에 전달되는 진동을 ...	면진장치는 건물에 오는 지반 진동의 영향을 최대한으로 흡수하여 건물에 전달되는 진동...

실험 - 데이터 전처리

1. 데이터 특수문자 제거

   -> 특수문자가 학습에 방해되어 일반적으로 특수문자를 제거

```
# 데이터 로드 과정에서 특수문자 제거

data.replace(regex=True, inplace=True, to_replace=r'[^a-zA-Z0-9가-힣ㅂs]', value=r'')

data.head()
```

	id	질문_1	질문_2
0	TRAIN_000	면진장치가 뭐야?	면진장치에 사용되는 주요 기술은 무엇인가요?

	id	질문_1	질문_2
0	TRAIN000	면진장치가 뭐야	면진장치에 사용되는 주요 기술은 무엇인가요

실험 - 데이터 전처리 (데이터 증강: NLTK)

```python
# 데이터 증강(Augmentation)
def synonym_replacement(sentence, n=1):
    words = sentence.split()
    new_words = words.copy()
    random_word_list = list(set([word for word in words if word not in ['<s>', '</s>']]))
    random.shuffle(random_word_list)
    num_replaced = 0
    for random_word in random_word_list:
        synonyms = []
        for syn in wordnet.synsets(random_word):
            for lemma in syn.lemmas():
                synonyms.append(lemma.name())
        if len(synonyms) >= 1:
            synonym = random.choice(synonyms)
            new_words = [synonym if word == random_word else word for word in new_words]
            num_replaced += 1
        if num_replaced >= n:
            break
    new_sentence = ' '.join(new_words)
    return new_sentence

augmented_data = []
for text in formatted_data:
    text = tokenizer.decode(text[0], skip_special_tokens=True)
    augmented_text = synonym_replacement(text)
    augmented_input_ids = tokenizer.encode(augmented_text, return_tensors='pt')
    augmented_data.append(augmented_input_ids)

formatted_data += augmented_data
```

synonym_replacement():

- 입력 문장을 단어 단위로 분리
- 각 단어에 대해 WordNet을 이용하여 동의어를 찾음
- 찾은 동의어 중 하나를 선택하여 원래 단어를 대체
- 대체된 단어로 이루어진 새로운 문장 반환

실험 - 데이터 전처리 (데이터 증강: NLTK)

```
/usr/local/lib/python3.10/dist-packages/transformers/optimization.py:411: FutureWarning: Th
  warnings.warn(
Epoch 1 - Avg Loss: 2.3149: 100%|██████████| 12880/12880 [16:58<00:00, 12.64it/s]
Epoch 1/10, Average Loss: 2.3149474678665216
Epoch 2 - Avg Loss: 0.9638: 100%|██████████| 12880/12880 [16:56<00:00, 12.67it/s]
Epoch 2/10, Average Loss: 0.9638495988276157
Epoch 3 - Avg Loss: 0.4813: 100%|██████████| 12880/12880 [16:59<00:00, 12.63it/s]
Epoch 3/10, Average Loss: 0.4813285306558846
Epoch 4 - Avg Loss: 0.3001: 100%|██████████| 12880/12880 [16:58<00:00, 12.64it/s]
Epoch 4/10, Average Loss: 0.3001278351249046
Epoch 5 - Avg Loss: 0.2250: 100%|██████████| 12880/12880 [16:56<00:00, 12.67it/s]
Epoch 5/10, Average Loss: 0.22502954261513392
Epoch 6 - Avg Loss: 0.1860: 100%|██████████| 12880/12880 [16:59<00:00, 12.64it/s]
Epoch 6/10, Average Loss: 0.18601771943178608
Epoch 7 - Avg Loss: 0.1632: 100%|██████████| 12880/12880 [16:57<00:00, 12.66it/s]
Epoch 7/10, Average Loss: 0.16321169785006978
Epoch 8 - Avg Loss: 0.1479: 100%|██████████| 12880/12880 [16:56<00:00, 12.67it/s]
Epoch 8/10, Average Loss: 0.1479040135619352
Epoch 9 - Avg Loss: 0.1355: 100%|██████████| 12880/12880 [16:47<00:00, 12.78it/s]
Epoch 9/10, Average Loss: 0.13545621516749187
Epoch 10 - Avg Loss: 0.1272: 100%|██████████| 12880/12880 [17:07<00:00, 12.53it/s]
Epoch 10/10, Average Loss: 0.12722696882415194
('./hansoldeco-kogpt2/tokenizer_config.json',
 './hansoldeco-kogpt2/special_tokens_map.json',
 './hansoldeco-kogpt2/tokenizer.json')
```

NLTK:

epoch=5, 55%

epoch=10, **61%**

epoch=10, batch_size=8, **64%**

Baseline(original):

epoch=10, **58%**

EWHA,
THE FUTURE
WE CREATE

실험 - Fine Tuning

1. optimizer 조절: AdamWR, SGDWR

2. learning rate : scheduler 사용

3. Batch size 조절

실험 - Fine Tuning: adamWR

```
Epoch 1 - Avg Loss: 2.0367: 100%|██████████| 12880/12880 [16:39<00:00, 12.89it/s]
Epoch 1/7, Average Loss: 2.0366963992554763
Epoch 2 - Avg Loss: 0.5505: 100%|██████████| 12880/12880 [16:20<00:00, 13.14it/s]
Epoch 2/7, Average Loss: 0.5505486513250826
Epoch 3 - Avg Loss: 0.2344: 100%|██████████| 12880/12880 [16:11<00:00, 13.26it/s]
Epoch 3/7, Average Loss: 0.23439554464858697
Epoch 4 - Avg Loss: 0.1641: 100%|██████████| 12880/12880 [16:15<00:00, 13.20it/s]
Epoch 4/7, Average Loss: 0.16406016845403598
Epoch 5 - Avg Loss: 0.1426: 100%|██████████| 12880/12880 [16:18<00:00, 13.17it/s]
Epoch 5/7, Average Loss: 0.14255241694027176
Epoch 6 - Avg Loss: 0.1222: 100%|██████████| 12880/12880 [16:18<00:00, 13.16it/s]
Epoch 6/7, Average Loss: 0.12223414033795968
Epoch 7 - Avg Loss: 0.1159: 100%|██████████| 12880/12880 [16:36<00:00, 12.93it/s]
Epoch 7/7, Average Loss: 0.11589305583830137
('./hansoldeco-kogpt2/tokenizer_config.json',
 './hansoldeco-kogpt2/special_tokens_map.json',
 './hansoldeco-kogpt2/tokenizer.json')
```

```
Epoch 1 - Avg Loss: 2.3149: 100%|██████████| 12880/12880 [16:58<00:00, 12.64it/s]
Epoch 1/10, Average Loss: 2.3149474678665216
Epoch 2 - Avg Loss: 0.9638: 100%|██████████| 12880/12880 [16:56<00:00, 12.67it/s]
Epoch 2/10, Average Loss: 0.9638495988276157
Epoch 3 - Avg Loss: 0.4813: 100%|██████████| 12880/12880 [16:59<00:00, 12.63it/s]
Epoch 3/10, Average Loss: 0.4813285306558846
Epoch 4 - Avg Loss: 0.3001: 100%|██████████| 12880/12880 [16:58<00:00, 12.64it/s]
Epoch 4/10, Average Loss: 0.3001278351249046
Epoch 5 - Avg Loss: 0.2250: 100%|██████████| 12880/12880 [16:56<00:00, 12.67it/s]
Epoch 5/10, Average Loss: 0.22502954261513392
Epoch 6 - Avg Loss: 0.1860: 100%|██████████| 12880/12880 [16:59<00:00, 12.64it/s]
Epoch 6/10, Average Loss: 0.18601771943178608
Epoch 7 - Avg Loss: 0.1632: 100%|██████████| 12880/12880 [16:57<00:00, 12.66it/s]
Epoch 7/10, Average Loss: 0.16321169785006978
Epoch 8 - Avg Loss: 0.1479: 100%|██████████| 12880/12880 [16:56<00:00, 12.67it/s]
Epoch 8/10, Average Loss: 0.1479040135619352
Epoch 9 - Avg Loss: 0.1355: 100%|██████████| 12880/12880 [16:47<00:00, 12.78it/s]
Epoch 9/10, Average Loss: 0.13545621516749187
Epoch 10 - Avg Loss: 0.1272: 100%|██████████| 12880/12880 [17:07<00:00, 12.53it/s]
Epoch 10/10, Average Loss: 0.12722696882415194
('./hansoldeco-kogpt2/tokenizer_config.json',
 './hansoldeco-kogpt2/special_tokens_map.json',
 './hansoldeco-kogpt2/tokenizer.json')
```

epoch=7까지 돌린 adamWR => 57%

epoch=5까지 돌린 adamW => 55%, 10까지 돌린 adamW=>61%

EWHA,
THE FUTURE
WE CREATE

실험 - Fine Tuning: SGDWR

```
Epoch 5/10, Average Loss: 3.243801327151541 3
Epoch 6 - Avg Loss: 3.1821: 100%|██████████| 6440/6440 [04:58<00:00, 21.56it/s]
Epoch 6/10, Average Loss: 3.1821034506240986
Epoch 7 - Avg Loss: 3.1455: 100%|██████████| 6440/6440 [04:59<00:00, 21.53it/s]
Epoch 7/10, Average Loss: 3.1454772184724393
Epoch 8 - Avg Loss: 3.1243: 100%|██████████| 6440/6440 [04:58<00:00, 21.59it/s]
Epoch 8/10, Average Loss: 3.12431514546367 7
Epoch 9 - Avg Loss: 3.0776: 100%|██████████| 6440/6440 [04:58<00:00, 21.58it/s]
Epoch 9/10, Average Loss: 3.0775543538865096
```

epoch=9까지 돌린 SGDWR avg Loss: 3.0776

=> loss값 변화 거의 없음

EWHA,
THE FUTURE
WE CREATE

실험 - Fine Tuning : Cosine Annealing Scheduler

```
Epoch 5 - Avg Loss: 0.2235: 100%|██████████| 12880/12880 [16:51<00:00, 12.73it/s]
Epoch 5/10, Average Loss: 0.223530029122549 93
Epoch 6 - Avg Loss: 0.1856: 100%|██████████| 12880/12880 [16:52<00:00, 12.73it/s]
Epoch 6/10, Average Loss: 0.18557534234841233
Epoch 7 - Avg Loss: 0.1624: 100%|██████████| 12880/12880 [16:54<00:00, 12.70it/s]
Epoch 7/10, Average Loss: 0.162431183863598 98
Epoch 8 - Avg Loss: 0.1464: 100%|██████████| 12880/12880 [16:52<00:00, 12.72it/s]
Epoch 8/10, Average Loss: 0.14644229052854435
Epoch 9 - Avg Loss: 0.1349: 100%|██████████| 12880/12880 [16:59<00:00, 12.63it/s]
Epoch 9/10, Average Loss: 0.1348957475847478
Epoch 10 - Avg Loss: 0.1264: 100%|██████████| 12880/12880 [16:52<00:00, 12.73it/s]
Epoch 10/10, Average Loss: 0.12640001553093425
```

learning rate scheduler를 이용 => 61.5%

실험 - Fine Tuning: batch size 조절

```python
def pad_sequences(sequences, max_len):
    # 각 시퀀스를 주어진 길이로 패딩
    padded_sequences = []
    for seq in sequences:
        seq_len = seq.size(0)
        if seq_len < max_len:
            padded_seq = torch.cat([seq, torch.zeros(max_len - seq_len, dtype=torch.long)], dim=0)
        else:
            padded_seq = seq[:max_len]
        padded_sequences.append(padded_seq)
    return torch.stack(padded_sequences)

def change_batch_size(dataset, batch_size, max_seq_length):
    # 데이터셋을 새로운 배치 크기로 DataLoader에 로드
    return DataLoader(dataset, batch_size=batch_size, shuffle=True, collate_fn=lambda x: (pad_sequences([i[0] for i in x], max_seq_length),))

# 최대 시퀀스 길이 설정
max_sequence_length = 128

# 기존의 formatted_data를 새로운 배치 크기로 변경
new_data_loader = change_batch_size(formatted_data, batch_size=16, max_seq_length=max_sequence_length)
```

THE FUTURE
WE CREATE

실험 - Fine Tuning: batch size 조절

```
Epoch 1 - Avg Loss: 1.4890: 100%|██████████| 805/805 [04:00<00:00,  3.35it/s]
Epoch 1/5, Average Loss: 1.489024793587706
Epoch 2 - Avg Loss: 0.9704: 100%|██████████| 805/805 [04:00<00:00,  3.34it/s]
Epoch 2/5, Average Loss: 0.9704005560519532
Epoch 3 - Avg Loss: 0.7271: 100%|██████████| 805/805 [04:00<00:00,  3.34it/s]
Epoch 3/5, Average Loss: 0.727140222906326
Epoch 4 - Avg Loss: 0.5705: 100%|██████████| 805/805 [04:00<00:00,  3.34it/s]
Epoch 4/5, Average Loss: 0.5704571322994944
Epoch 5 - Avg Loss: 0.4560: 100%|██████████| 805/805 [04:03<00:00,  3.30it/s]
Epoch 5/5, Average Loss: 0.4560293406062985
```

batch size 8 : 점수 0.6149

batch size 16 : 점수 0.6136

최종 결과

가장 score가 높았던 실험 조건

데이터 전처리 : nltk를 이용한 데이터 증강

batch size = 8

epoch = 10

=> 64%

아쉬운 점

GPU 사이즈 한계로 여러 모델을 적용해보지 못한 것

Test time augmentation 적용 시 Embedding vector 크기 차이로 인한 null값 발생

램 사이즈 한계 -> 훈련 시간 감소, 모델 크기 제한

감사합니다