

# Open-Domain Question Answering (MRC)

- NLP-05 Wrap-up Report -



브레멘 음악대

변성훈 서보성 이상민

이승우 이예원

## <목 차>

### I. 서론

- 1. 프로젝트 개요 ----- 3
- 2. 평가 방법 및 룰 ----- 4
- 3. 프로젝트 팀 구성 및 역할 ----- 4

### II. 본론

- 1. 프로젝트 수행 절차 및 방법
  - 1-1 공통 환경 설정 ----- 4
- 2. 프로젝트 수행 및 완료 과정
  - 2-1 Data Analysis ----- 5-6
  - 2-2 Retriever ----- 6-10
  - 2-3 Reader ----- 10-12
  - 2-4 Sentence-level Approach (Bremen Special) ----- 12-13
  - 2-5 Ensemble ----- 12
- 3. 프로젝트 수행 결과 ----- 12

### III. 결론

- 1. 자체 평가 의견 ----- 13

### IV. 개인회고

- 참고문헌 ----- 20

## I. 서론

### 1. 프로젝트 개요

Question Answering은 다양한 종류의 질문에 대해 대답하는 인공지능을 만드는 연구 분야이다. 특히 Open-Domain Question Answering (ODQA)은 주어지는 지문이 따로 존재하지 않고 사전에 구축된 Knowledge resource에서 질문에 답할 수 있는 문서를 찾는 과정이 추가된다. 본 대회에서 우리가 만들어야 했던 모델은 질문에 관련된 문서를 찾아주는 **retriever** - 관련된 문서를 읽고 적절한 답변을 찾거나 만들어주는 **reader**의 two-stage로 구성되어 있다.

분류(디렉토리 명)	세부 분류	샘플 수	용도	공개여부
train_dataset	train	3952	학습용	모든 정보 공개 (id, question, context, answers, document_id, title)
	validation	240		
test_dataset	validation	240 (Public)	제출용	id, question 만 공개
		360 (Private)		

```
20]: train['train'][0]
```

```
20]: {'title': '미국 상원',  
      'context': '미국 상의원 또는 미국 상원(United States Senate)은 양원제인 미국 의회의 상원이  
다.\n\n미국 부통령이 상원의장이 된다. 각 정당 2명의 상원의원이 선출되어 100명의 상원의원으로 구  
성되어 있다. 임기는 6년이며, 2년마다 50개주 중 1/3씩 상원의원을 새로 선출하여 연방에 보낸다.\n\n미국 상원은 미국 하원과 다른 미국 대통령을 수반으로 하는 미국 연방 행정부에 각종 동의를 하  
는 기관이다. 하원이 세금과 경제에 대한 권한, 대통령을 포함한 대다수의 공무원의 파면할 권한을 갖고  
있는 국민을 대표하는 기관인 반면 상원은 미국의 주를 대표한다. 즉 캘리포니아주, 일리노이주 같이 주  
정부와 주 의회를 대표하는 기관이다. 그로 인하여 군대의 파병, 관료의 임명에 대한 동의, 외국 조약에  
대한 승인 등 신속을 요하는 권한은 모두 상원에게만 있다. 그리고 하원에 대한 견제 역할(하원의 법안을  
거부할 권한 등)을 담당한다. 2년의 임기로 인하여 급진적일 수밖에 없는 하원은 지나치게 급진적인 법안  
을 만들기 쉽다. 대표적인 예로 건강보험 개혁 당시 하원이 미국 연방 행정부에게 퍼블릭 옵션(공공건강보  
험기관)의 조항이 있는 반면 상원의 경우 하원안이 지나치게 세금이 많이 든다는 이유로 퍼블릭 옵션 조항  
을 제외하고 비영리건강보험기관이나 보험회사가 담당하도록 한 것이다. 이 경우처럼 상원은 하원이나 내각  
책임제가 빠지기 쉬운 국가들의 국회처럼 걸핏하면 발생하는 의회의 비정상적인 사태를 방지하는 기관이다.  
상원은 급박한 처리사항의 경우가 아니면 법안을 먼저 내는 경우가 드물고 하원이 만든 법안을 수정하여  
다시 하원에 되돌려보낸다. 이러한 방식으로 단원제가 빠지기 쉬운 합정을 미리 방지하는 것이다. 날짜=20  
17-02-05',  
      'question': '대통령을 포함한 미국의 행정부 견제권을 갖는 국가 기관은?',  
      'id': 'mrc-1-000067',  
      'answers': {'answer_start': [235], 'text': ['하원']},  
      'document_id': 18293}
```

데이터 구성 및 예시는 위 그림과 같으며, 질문 (question), 답변이 포함된 문서 (context), 질문에 대한 답변 (answers), 문서의 제목 (title)으로 이루어진다. answers에는 context 내에 존재하는 답변의 시작 index (answer\_start)와, 답변 내용 (text)으로 이루어지며, 하나의 질문에 하나의 답변만 존재한다. 추가로, Retriever를 위한 wikipedia 문서 데이터 역시 따로 제공되었다.

## 2. 평가 방법 및 룰

**[평가지표]** Exact Match (EM)을 사용한다. 모델의 예측과 실제 답이 정확하게 일치할 때만 점수가 주어진다. 즉 모든 질문은 0점 또는 1점으로 처리된다. 단, 띄어쓰기나 “.”과 같은 문자는 제외한 후 판단한다.

보조 지표로 F1 Score를 사용한다. EM의 경우 0점을 받는 답변이더라도 겹치는 단어가 있으면 부분 점수를 받을 수 있지만, 참고용으로만 활용되며 평가에는 반영되지 않았다.

**[외부 데이터셋]** KLUE-MRC 데이터셋을 제외한 모든 외부 데이터 사용이 허용된다.

**[기학습 가중치 사용]** KLUE-MRC 데이터로 학습된 기학습 가중치 (pretrained weight)를 제외한 모든 기학습 가중치 사용이 허용된다. 가중치는 모두 public에 공개되어 있고 저작권 문제 없이 누구나 사용 가능해야 한다.

**[평가 데이터 활용]** 학습 효율 측면에서 테스트셋을 분석하고 사용하는 행위는 금지한다.

## 3. 프로젝트 팀 구성 및 역할

**[변성훈]** Elasticsearch 적용, Tokenizer fine-tuning & Pre-train, Bremen Special 및 Independent Document Inference ideation

**[서보성]** Dense Retriever(Bi-encoder, ColBERT) 구현, Kobigbird 모델 테스트 (Negative Sampling 방식으로 학습)

**[이상민]** Data Preprocessing, Model Tuning, Curriculum Learning 구현

**[이승우]** Baseline code customizing, Data augmentation(KorQuAD), Transfer Learning, Dense Retriever(Cross-encoder) 구현

**[이에원]** Prompt tuning, Independent Document Inference, BM25, Bremen Special 및 ensemble 구현

## II. 본문

### 1. 프로젝트 환경

#### 1-1. 프로젝트 환경

**[컴퓨팅 환경]** 인당 V100 서버 할당, VSCode와 SSH 연결해 사용

**[모듈]** PyTorch, Hugging Face, wandb (logger)

**[협업 환경]** GitHub, Discord

#### 1-2. 공통 환경 설정

기존 **baseline code**에서 터미널로 **argument**를 받는 방식이 다소 복잡하고, 이후 추가될 다양한 **parameter**들을 **customizing**하기 불편하여 **yaml** 파일로 **config**를 관리하는 방식으로 변경하였다. 또한 학습 **logger**로 사용한 **wandb**에 **evaluation EM** 및 **F1 score**가 기록되도록 수정하였다. 성능 비교를 위해 **seed** 고정도 함께 세팅하였다.

## 2. 프로젝트 수행 및 완료 과정

### 2-1. Data Analysis

#### (1) Data augmentation

외부 데이터 사용이 허용됨에 따라, LG CNS가 운영하는 표준 한국어 QA 데이터셋인 KorQuAD (Korean Question Answering Dataset) 1.0을 추가로 사용했다. 해당 데이터셋은 기존의 주어진 데이터셋과 스키마가 같고, 그 내용이 비슷해 적용에 무리가 없었다. 또 데이터의 개수가 **train 60,407개**, **validation 5,774개**로, 각각 **3,952개**, **240개**인 기존 데이터셋만 사용하는 것에 비해 성능이 향상될 것이라 판단했다.

실험은 KorQuAD만 사용하기, 기존 데이터와 섞어 사용하기, 샘플링해 사용하기 등 다양하게 진행했다. 결과, 각 경우에서 최적의 **hyperparameter (learning rate 등)**를 가정할 때, 기존 데이터셋만 쓴 경우보다 기존 데이터셋에 KorQuAD의 **train** 및 **validation**까지 섞어 쓴 경우에 **public**에서 성능이 올랐지만 (**EM 60.83 → 62.08**), **private**에서는 성능이 떨어졌는데 (**EM 59.44 → 57.50**), 따라서 KorQuAD는 기존 **test dataset**과 어느 정도 다른 성질을 가지고 있는 것으로 추측된다. 하지만 KorQuAD는 이후에도 단순 증강 뿐 아닌 다양한 튜닝에 사용했다.

#### (2) Preprocessing

**train, validation data**에 '\n'라는 불용어가 다수 포함되어 있었다. '\n'이 **tokenizer**에 들어갔을 때, '\n'같은 경우 **[UNK]**으로 **tokenized**되고, **n**같은 경우 뒤의 단어와 결합되어 뒤의 단어에 영향을 끼친다는 것을 확인했다. '\n'은 뒤 단어의 의미까지 변질을 줄 수 있기에 제거를 해주었다.

*EX* '날씨 좋다.\n해가 짙쨌' → **[날씨][좋다][##다][.][\n(UNK)][n해][##가][짙쨌]**

#### (3) Prompt Tuning

**train, valid, test data** 모두 극소수의 **example**을 제외하면 **question**의 의문형 어미가 '은?', '는?', '요?', '나?', '가?'로 끝난다는 사실을 확인할 수 있었고 '은?', '는?'으로 끝나는 경우를 제외하고는 **question**에 극소수의 **example**을 제외하고 의문사가 존재했다. 또한, **train, valid, test data**에서 의문형 어미, 의문사 유형의 분포는 동일하였다.

이와 같은 분석을 토대로 의문형 어미 통일, 체언과 용언의 강조 등 다양한 실험을 진행했지만

다음의 예시만이 public에서 성능 상승을 보였고 (EM 67.08 → 69.17) private에서는 성능 상승이 없었다.

#### [KorQuAD 1.0 data preprocessing]

question이 '?' 로 안 끝나면 '?' 추가, 끝에서 두번째에 '.', '?', ';' 의 문자가 있으면 제거했다.

기존 데이터에 없는 의문형 어미, 의문사를 가진 데이터 제거했다. (train 2,017개, valid 154개)

#### [[MASK] special token 추가]

의문형 어미가 '은?', '는?' 으로 끝나는 경우는 answer가 있어야 할 부분이 원래 바로 뒤  
이므로 아래와 같은 예시로 [MASK] special token을 추가했다.

EX) NLP 5조의 이름은? → NLP 5조의 이름은 [MASK]인가?

이외의 의문형 어미는 의문사가 있는 부분이 answer가 나와야 할 부분이므로 의문사 뒤에  
[MASK]를 추가했다. (의문사 자체에도 정보가 있으므로 의문사 자체는 [MASK]로 대체하지  
않았다.)

EX) 브레멘 음악대에는 누가 있는가? → 브레멘 음악대에는 누가[MASK] 있는가?

## 2-2. Retriever

### (1) Retriever Tokenizers

Retriever는 inference 과정에서 Reader와 독립적으로 동작한다. 그러므로 Reader에서  
사용한 Model의 tokenizer와는 다른, retriever 성능이 좋은 pretrained model의 것을 사용해  
성능을 높이하고자 하였다. 같은 조건에서의 실험 결과는 다음 표와 같다.

retrieval tokenizer	evaluation EM
klue/roberta-large	55.42
<b>monologg/koelectra-base-v3-finetuned-korquad</b>	<b>56.25</b>
jinmang2/kpfbert	56.25
KoNLPy - Mecab	37.50
KoNLPy - Kkma	35.00
KoNLPy - Komoran	32.08

실험 결과, Reader model에서 주로 사용한 klue/roberta-large보다  
monologg/koelectra-base-v3-finetuned-korquad와 jinmang2/kpfbert가 성능이 좋을 것을 확인할  
수 있었다. KoNLPy 내장 형태소 분석기들은 성능이 좋지 못했다.

이후 조건이 다른 실험들에서 kpfbert 보다는 KorQuAD, 즉 MRC task 용으로 finetuning한  
monologg/koelectra-base tokenizer가 대체적으로 우세했기에, 주로 이를 사용했다.

## (2) Sparse Retriever (Tf-idf, BM25, BM25+Tf-idf, Elastic Search)

Sparse retriever는 키워드 기반의 문서 검색 방식으로 질문과 관련성 높은 문서를 찾아낸다.

Baseline code에 Tf-idf 방식이 주어졌고, 추가로 이를 개선한 BM25 방식을 사용하였다.

Retriever에서는 관련성 높은 상위 k개의 문서를 선별해 사용하며, 문서의 개수 (이하 topk) hyperparameter가 성능에 큰 영향을 주었다. 실험 결과, Tf-idf는 topk 40, BM25는 topk 30의 문서를 추출하는 것이 대부분의 경우 최적이었다.

같은 조건 (topk=30)에서 train data 기준으로는 Retriever의 accuracy (topk 내 정답 context 포함 정확도)는 Tf-idf가 80%, BM25가 92%로 BM25가 크게 높지만 public score는 Tf-idf와 거의 유사하였으며 서로 다른 prediction 양상을 보여주었다.

이처럼 Tf-idf와 BM25가 문서의 다른 특징을 잡아낼 수 있다는 점을 발견해 두 방식을 같이 사용하는 방식을 사용했다 (BM25+Tf-idf). 각각 2k개씩, 도합 4k개의 문서를 선별한 뒤 두 결과에서 겹치는 문서가 있을 경우 두 score를 normalize한 후, 평균을 낸 뒤 ranking을 다시 매겨 k개를 뽑는 방법을 구현했다. 결과적으로 public 성능을 올릴 수 있었지만, private에서는 BM25를 단독으로 사용한 것이 성능이 좋았다 (하단 표 참조).

여러 tokenizer를 조사하던 도중 Elastic Search에서 제공하는 nori-tokenizer를 알게되었다. Elastic Search 자체가 검색엔진이므로 본 프로젝트의 Retriever에 적용할 수 있을 것이라고 생각하였다.

Elastic Search는 Ngram과 비슷한 방식인 shingle filter, 원본 단어와 분리시킨 단어를 모두 포함시키는 Mixed decompound를 이용하여 각 문서를 Indexing하는 방식인데, 단어 집합에 많은 경우의 수가 들어가기 때문에 다른 방식들과 비교하여 상대적으로 일반화 성능이 높을 것이라는 가설을 세웠다.

실제 결과에서 public score는 다른 방식과 차이가 크지 않았지만 private score에서 하락폭이 매우 적은 것을 확인할 수 있었다. private에서의 score를 확인할 수 없었기 때문에 public에서 좋은 score를 보여준 topk 30~40정도의 하이퍼파라미터를 사용하였다.

	topk	lr	batch	retrieval tokenizer	public EM	private EM
Tf-idf	40	5e-5	16	klue/roberta-large	60.42	53.61
BM25	30	5e-5	16	klue/roberta-large	60.83	61.11
BM25	30	5e-5	16	monologg/koelectra	63.75	61.94
BM25+Tf-idf	35	5e-5	16	monologg/koelectra	64.17	60.83
Elastic Search	10	5e-5	16	nori-tokenizer	60.83	61.39
Elastic	30	5e-5	16	nori-tokenizer	61.67	61.39

Search						
Elastic Search	40	5e-5	16	nori-tokenizer	61.67	60.56
Elastic Search	100	5e-5	16	nori-tokenizer	58.33	57.78

### (3) Dense Retriever

Dense Retriever는 Sparse Retriever와 달리 Dense vector 표현을 사용해 문서 간 유사도를 계산하는 방식으로 문서를 찾아낸다. Encoder에 적절한 pretrain된 BERT모델을 불러온 뒤 Positive pair와 Negative pair 사이의 Negative Log Likelihood를 최소화하는 방향으로 fine-tuning하여 사용하는 방식이다.

먼저 학습에는 In Batch Negative Sampling 기법을 사용했다. 질문 (Query)와 Golden Passage (Query에 해당하는 정답 문서)를 Encoder에 통과시킨 뒤, 정답 Query-Passage 쌍은 1로 labeling하고, 추가로 정답이 아닌 Passage들과 해당 Query의 쌍을 여러 개 만들어 0으로 labeling했다.

정답이 아닌 Passage들은 BM25를 이용해 Silver Passage (Golden Passage와 유사하지만 정답은 아닌 상위권 Passage들)를 이용하는 방법과, Silver Passage 및 다른 Random한 문장을 같이 이용하는 방법을 사용해 Sampling했다.

사용한 Encoder 모델들의 구조는 다음과 같다.

#### [Bi-encoder]

Bi-encoder에서는 두 개의 학습된 Encoder를 통해서 계산한 Query와 Passage 각각의 hidden state [CLS]의 벡터값을 내적하여 유사도를 계산한다.

#### [Cross-encoder]

Cross-encoder는 Query와 Passage의 Encoder를 따로 사용하지 않고, 둘을 concat해 [SEP] special token으로 구분한 후 하나의 Encoder에 넣는 방식이다. Bi-encoder와 달리 Query와 Passage 사이의 token level에서의 상호작용을 고려할 수 있다.

#### [Colbert]

Colbert에서는 두 개의 학습된 Encoder를 통해서 계산한 Query와 Passage 각각의 hidden state 들을 Query의 모든 Token들과 Passage의 모든 Token들을 각각 내적한 뒤 각각의 Query Token에 대해서 계산된 값의 최대값을 모두 더하여 유사도를 계산한다.



#### (4) Inference Method

##### [[SEP] special token 사용]

Top k개의 문서를 처리할 때, **Baseline**에서는 문서를 공백을 이용해 연결했다. 서로 다른 **passage**를 같은 **context**로 생각해 방해줄 수 있다는 가설을 세웠고, 따라서 문서를 구분해주기 위해 **[SEP] special token**으로 이어주었다. 그 결과 **public**에서는 조금의 점수 향상이 있었지만 (EM 60.83 → 61.67), **private**에서는 조금의 점수 하락이 있었는데 (EM 57.7800 → 57.2200), **Reader** 단계에서 **[SEP] token**에 대한 학습이 되지 않았기 때문으로 추정한다. **Reader**에서 학습 시 여러 개의 **passage**를 붙여 학습시키지 않기에 그 의미를 알지 못했던 것으로 보이며, **[SEP]**를 **output**으로 내놓는 경우도 있었다. 따라서 여러 개의 **passage**를 **[SEP]**로 이어 붙여서 학습시켜 보고자 하였으나 시간 관계상 하지 못했다.

##### [Independent Documents Inference]

훈련 시에 문서 하나에 대해 **answer**를 찾기 때문에, **Inference**에서도 **topk**의 문서를 길게 연결하는 방식이 아닌, 문서 각각에 대해 **inference**를 한 후 **answer logits**에 **normalized document score** (logits이 음수일 경우 1-score)를 곱해주어서 가장 큰 값을 가지는 **answer**를 채택하는 방식을 사용하였다. 이 때 **probs**가 아닌 **logits**에 **score**를 곱한 이유는 관련 없는 문서에서 유일하게 답이 될 만한 부분이 있을 때, **softmax**를 취하는 **probs**는 높을 수 있다는 점을 염려했기 때문이다. 문서를 개별적으로 **inference**할 경우 **case**마다 다르지만 대체로 최적의 **topk** 값이 높아지는 경우를 확인할 수 있었으며, 큰 성능 향상이 있었다 (**public** EM 64.17 → 67.08, **private** EM 60.83 → 65.00).

### 2-3. Reader

#### (1) Model Tuning

**Reader Pretrained Model**의 경우 실험 결과 다른 **task**와 마찬가지로 **klue/roberta-large**가 모든 경우에 가장 우월한 성능을 내어 이를 주로 사용했다.

Huggingface에서 제공하는 **Pretrained** 모델에서 **QA** 모델의 마지막 **layer**가 **Linear Layer** 한 층으로 이루어진 것을 확인하여 마지막 **layer**의 변화를 시도해 보았다. “**LSTM(Reverse , Bidirectional) + 활성화 함수 + DropOut layer** 추가”와 “**Linear Layer + 활성화 함수 + DropOut layer** 추가”를 시도했다. (**LSTM** 추가시, **public** EM 59.58 → 52.50, **private** EM 54.72 → 52.22) **LSTM** 이용 시 성능이 하락한 것은 **sequential**한 데이터를 이용하기에 **max\_len 512**는 너무 길었기 때문으로 보인다. **Linear Layer**를 추가했을 때는 **wandb** 상에서 성능 향상이 없어 별도의 제출을 해보지 않았는데, 활성화 함수가 학습 시 영향을 끼친 것으로

추정된다.

또한, 단어의 정보가 pretrained 모델에서 잘 학습되어 있을 것이라 생각하고 embedding layer를 freeze 후 학습을 진행해 보았을 때, public에서는 향상이 있었지만 private에서는 성능 하락을 보았다 (public EM 61.67 → 63.75, private EM 57.22 → 55.56)

## (2) Tokenizer Tuning

“만약 tokenizer에서 label을 [UNK]로 인식한다면 성능을 저하시키는 원인이지 않을까”라는 가설을 세웠고, train dataset에서 등장하는 명사들을 KoNLPy의 Okt 라이브러리를 사용하여 얻어낸 뒤 pre-train tokenizer에 토큰을 추가해주었다.

1차 시도로 embedding layer를 제외한 상단의 layer를 모두 freezing한 후 fine-tuning하였지만 훈련이 끝난 후 모델을 불러왔을 때 embedding layer에 들어가면 안되는 범위의 값들이 들어가 오류가 발생하였다.

2차 시도로 freezing을 하지 않고, 단순히 토큰을 추가해준 후 fine-tuning을 진행하였다. 이 경우에는 baseline code에서의 리더보드 EM score와 다를 바 없는 score를 보여주었다. 새로 추가해준 토큰은 pre-train이 진행되어 있지 않기 때문에 모델의 train을 오히려 방해하였고, 이것이 성능 저하로 이어진 것이라고 생각하였다. (public EM 43.33, private EM 45.28)

## (3) Curriculum Learning

from\_pretrained 모델로 Question, Documents 별로 start\_logits과 end\_logits에 대해 CrossEntropyLoss를 구하고 합쳐 total loss를 산정해주었다. total loss를 기준으로 난이도를 측정했고, loss가 클수록 난이도가 높은 문제라고 생각했고, 난이도를 기준으로 train data를 4개로 나누어 학습을 진행했다.

f1 score가 아닌 Logits의 Loss 값으로 난이도를 측정해서 그런지, 이미 학습된 모델이 아닌 from\_pretrained 모델로 난이도를 측정해서 그런지 wandb 상 성능 향상이 보이지 않았고, 따라서 별도의 제출을 하지 않았다.

## (4) Transfer Learning

삼성SDS에서 진행한 Techtonic 2020 KorQuAD 1.0 성능 개선 프레젠테이션의 전이 학습 관련 내용 ([https://youtu.be/ovD\\_87gHZO4](https://youtu.be/ovD_87gHZO4))을 참고해, 먼저 KorQuAD (train+validation) 데이터셋으로만 1차로 fine tuning한 후, 이후 2차로 기존 데이터셋을 이용해 Transfer

learning을 진행했다. 실험 결과 1차 lr 3e-5, 2차 lr 9e-6에서 매우 큰 성능 향상이 있었다 (pubilc EM 64.17 → 67.08, private EM 60.83 → 67.22).

## (5) Negative Sampling

Retriever의 Inference Method에서 언급했듯이 Inference 단계에서는 Retriever를 통해서 찾아낸 유사도가 높은 문장들을 단순히 concat한다. 하지만 Reader의 경우 단순히 Golden Passage에 대해서만 학습을 진행하기 때문에 이를 개선하기 위해서 Reader의 학습시 BM25를 이용하여 찾은 유사도가 높은 문장(Silver Passage)를 기존 Golden Passage에 Concat하여 학습을 진행하였다.

## 2-4. Sentence-level Approach (Bremen Special)

전체 문서에서 answer를 찾는 방법보다는 먼저 정답이 포함된 문장을 찾고, 문장 내에서 answer를 찾으면 더 정확한 답을 찾을 것이라는 가설을 세우고, 정답 sentence에 가중치를 주어 더 정확한 정답을 찾고자 고안한 방법이다.

### [Semantic Textual Similarity model finetuning]

먼저 KorQuAD를 이용해 question과 sentence 쌍을 입력받아 해당 문장이 answer를 포함하고 있는 문장이면 1로 labeling, 아니라면 0으로 labeling한 STS task의 상황을 만들어 학습시켰다 (1 epoch, lr=7e-6). 모델은 이전 STS 대회 때 사용한 electra model을 사용했다. 여기서 train data를 구성할 때, 정답 sentence보다 정답이 아닌 쪽이 더 많다는 점을 고려해 1:3의 비율로 Sampling 해주었다. 이후 기존 dataset으로 test했을 때, 93% 정도의 정확도를 보여주었다.

### [Train method]

정답 문장에 대한 logits에는 0.1의 가중치를, 정답 문장이 없을 경우에는 CLS에 0.1의 가중치를 주어 학습을 진행하였다.

### [Inference method]

retriever로 불러온 wiki document를 문장으로 분리해서 각 문장이 question에 대한 정답 sentence 일지를 위 finetuning 모델로 예측한다. 정답이라 예측한 경우에 대해서만 softmax를 취해 확률을 구한 후, 해당 sentence 부분의 logits에 0.1\*probs의 가중치를 준다. (train과 마찬가지로 없을 경우 CLS에 0.1 가중치)


구현을 완료한 시점이 늦어 시간상 제대로 된 실험을 할 수가 없어 정확한 성능을 측정할 수 없었지만, **parameter tuning** 없이 간단하게 진행해 본 실험에서는 따로 성능 향상을 이루지는 못했다.


## 2-5. Ensemble

최종적으로 사용한 앙상블 방법은 **nbest**의 **probs**를 활용한 **soft voting**이다. 각 **Inference** 시 지정한 **nbest**개 만큼의 확률이 높은 **answer**들을 뽑고, 각 **answer**들의 **probs**에 제출 **score**를 곱한 값들을 **answer** 별로 모두 합한 후, 그 값이 가장 큰 **answer**를 채택하는 방식을 사용했다. 최종적으로 제출한 것은 아래 조합의 **predictions**을 **nbest 3**으로 조합하였다.

1. KorQuAD Transfer Learning + Elastic Search (topk 40) + Independent Document Inference
2. KorQuAD Masked Data + Tf-idf+BM25 (topk 100) + Independent Document Inference
3. Original Data + Tf-idf+BM25 (topk 100) + Independent Document Inference
4. original data + Elastic Search (topk 30) + Independent Document Inference
5. KorQuAD masked data (Back-translation으로 의문사 추가) + Tf-idf+BM25 (topk 100) + Independent Document Inference
6. KorQuAD Transfer Learning + Tf-idf+BM25 (topk 100) + Independent Document Inference
7. KorQuAD masked data + Bremen Special + Elastic Search (topk 40) + Independent Document Inference

## 3. 프로젝트 수행 결과

순위	팀 이름	팀 멤버	EM ↕	F1 ↕	제출 횟수	최종 제출
3 (-)	NLP_05조		72.5000	82.9600	120	3d

순위	팀 이름	팀 멤버	EM ↕	F1 ↕	제출 횟수	최종 제출
2 (1 ▲)	NLP_05조		71.3900	81.7400	120	3d

**[public score]** EM: 72.50, F1: 82.96, 13팀 중 3위

**[private score]** EM: 71.39, F1: 81.74, 13팀 중 최종 2위

## III. 결론

## 1. 자체 평가 의견

### 1-1. 잘한 점들

MRC Task에 대한 이해도가 뒷받침 되어야한다고 판단해 학습에 많은 시간을 할애했다. 덕분에 그렇지 않았을 경우보다 문제 인식이 수월하게 진행되었을 것이다. 또, 그런 철저한 문제 인식과 원인 분석 덕분에 적용할 작업의 리서치 및 아이디어이션이 곧잘 이루어지고 기존에 없던 새로운 시도도 해보며 다양한 경험을 할 수 있었으며, 최종 결과 또한 만족스러웠다.

### 1-2. 아쉬웠던 점들

시간 분배의 실패로 계획한 작업들을 수행하지 못했고, 구현을 완료한 몇몇 작업들도 실험의 부족으로 최종 결과에 사용하지 못했다.

## IV. 개인 회고

### 1. 변성훈

나는 내 학습목표를 달성하기 위해 무엇을 어떻게 했는가?

문제가 발생하였을 때 답을 바로 얻는 것을 지양하는 편이다. 그래서 문제를 스스로 해결하기 위해 학습에 시간을 많이 투자하였다. 하지만 팀원들과 정해놓은 마감 기한이 있었기 때문에 급하게 처리해야 할 문제들은 팀원들의 도움을 많이 받았다.

전과 비교하여 새롭게 시도한 변화는 무엇이고, 어떤 효과가 있었는가?

지금까지는 데이터 처리 혹은 시각화쪽을 주로 담당했었다. 이번에는 모델 쪽의 개선을 진행하면서 딥러닝 모델 개선에 대한 두려움을 많이 없앨 수 있었다. 모델 부분은 코드가 길면 읽기 두렵고, 잘못 수정하면 오류가 발생하기 때문에 쉽게 다가갈 수 없었지만, 이번 대회에서는 오류가 발생하더라도 어떻게든 해결하려 노력하고, 결국 기능을 완성시킴으로써 이러한 걱정들을 조금이나마 해소할 수 있었던 것 같다.

마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

지금까지 진행했던 프로젝트(문장 유사도, 문장 내 관계 추출)의 경우 문제점이 뚜렷하였고, 그

원인을 파악한 후 해결하는 과정을 거쳐왔지만 이번 프로젝트에서는 그 과정을 진행하기에 너무 많은 걸림돌이 있었다. wandb상의 **evaluation EM score**와 리더보드의 **EM score**가 최대 20점까지 차이가 났으며, **retriever**와 **reader**로 나뉘는 대회 특성상 모델이 어떤 데이터를 잘 학습하지 못하는지 파악하는 것조차 쉽지 않았다.

한계/교훈을 바탕으로 다음 프로젝트에서 시도해볼 것은 무엇인가?

코드 컨벤션을 제대로 확립하고 프로젝트를 시작하는 것이 좋을 것 같다. 파이썬이라는 언어 자체가 다른 언어에 비하여 제한이 크지 않기 때문에 서로 다른 코딩 스타일을 갖게 되는 경우가 많은 것 같다. 이번 대회는 코드의 길이도 매우 길었기 때문에 새로운 기능이 추가될 때마다 코드를 해석하는데 힘든 점이 많았다.

나는 어떤 방식으로 모델을 개선했는가?

**retriever** 쪽에 많은 신경을 썼다. **reader**의 성능이 좋아도 **retriever**가 잘못된 문서를 가져온다면 **reader**의 성능이 무의미해지기 때문이다. 그리하여 이미 잘 만들어져있는 검색엔진을 사용하고자 **ElasticSearch**를 이번 프로젝트에 적용시켰고, **public EM score**와 **private EM score**가 크게는 5점가량 차이나는 다른 **retriever**에 비해 **Elasticsearch**는 1~2점 밖에 하락하지 않아 많은 도움이 되었다.

협업 과정에서 잘된 점 / 아쉬웠던 점은 어떤 점이 있는가?

서로 많은 시도를 하다보니 코드가 꼬이는 경우가 자주 발생했던 것 같다. 어느 팀원은 코드가 잘 실행되고, 어느 팀원은 오류가 계속 발생했었다. 이번 대회는 과정이 여러 개로 나뉘었기 때문에 더욱 이런 상황이 자주 발생했다고 생각한다.

## 2. 서보성

나는 내 학습목표를 달성하기 위해 무엇을 어떻게 했는가?

이번 프로젝트의 주제가 이전에 했던 프로젝트에 비해서 조금 난이도가 있는 편이었기 때문에 이번 프로젝트에서는 어떤 방식으로 모델이 예측을 진행하는지를 **Baseline Code**를 이해하여 개선점을 생각하는 것을 학습목표로 정했다. 먼저 동작에 대한 이해를 하기 위해서 **breakpoint** 내장 함수를 사용하여 어떻게 동작되고 있는지를 중간중간마다 확인하였다. 또한 정답이 들어있는 **sequence**를 잘 가져오는 것이 가장 중요하다고 생각했기 때문에 **Reader**보다는 **Retriever**의 성능을 높이는 것을 우선적으로 생각했다. **Baseline code**에서 **Retriever**는 **Tf-idf**를 이용해서 동작하는 것을 확인하였고, 이것을 다양한 방법의 **Dense Retriever**로 바꾸는 방법을 시도했다.

전과 비교하여 새롭게 시도한 변화는 무엇이고, 어떤 효과가 있었는가?

기본적으로 **Dense Retriever**를 구현하기 위해서는 따로 **Dense embedding** 모델을 학습을 시켜야한다. 기존에 **Baseline**을 토대로 기능들을 추가하여 구현을 한 적은 많았지만 모델을 학습하기 위한 코드 전체를 구현한 적은 없었다. 그래서 이번에 **Dense Retriever Passage**를 구현하면서 직접 모델에 대한 학습코드를 구현해보았으며, 이를 통해서 어떤 부분이 어떻게 동작하는지 어떤 것이 모델 학습의 **Parameter**로 중요한 역할을 하는지에 대해서 알 수 있었다.

마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

**Dense Retriever**를 구현하면서 많은 한계를 마주했지만 가장 큰 부분은 성능적인 부분이었다. 의도한 대로 동작할 수 있도록 제대로 구현했으나 성능적으로 상승이 없어서 제대로 구현했는지 점검하는 과정에 많은 시간을 쏟았다. 이 부분에서 쏟은 많은 시간들 때문에 다른 작업들을 많이 진행하지 못했고 이것이 이 프로젝트에서 가장 아쉬웠던 점이다.

한계/교훈을 바탕으로 다음 프로젝트에서 시도해볼 것은 무엇인가?

성능적인 상승에 없는 부분에 대해서 프로젝트 중에 직면했던 한계에 대한 해답을 다른 팀의 이야기를 듣고 알게 되었다. 이번에 겪은 문제의 이유는 데이터의 양과 **hyper parameter**의 문제였다. 단순히 한 가지의 경우만 실험한 뒤 포기한 것이다. 그래서 이것을 토대로 다음 프로젝트에서는 방법이 옳다고 생각되면 다양한 시도를 해야겠다고 생각한다.

나는 어떤 방식으로 모델을 개선했는가?

먼저 **Dense Retriever**로 **BiEncoder**와 **Colbert**를 사용해서 모델 개선을 시도했다. 두 **Dense Retriever** 모두 **Negative Sampling** 방식으로 학습을 진행했으며, **Negative Sample**로 랜덤한 **Context**나 **BM25**로 계산된 유사도가 높은 문장을 사용했다. 또한 **Reader**의 **inference** 과정에서 유사도가 높은 문장들을 **concat**해서 입력해주는 것을 확인하여 학습시에도 같은 방식으로 학습할 수 있도록 **BM25**로 추출한 유사도 높은 문장을 정답 **context**에 **concat**해서 학습을 진행하였다.

협업 과정에서 잘된 점 / 아쉬웠던 점은 어떤 점이 있는가?

이전에 진행된 프로젝트보다 **Github**를 잘 활용했던 부분은 좋았고 소통도 전보다 더 잘 진행되었던 것 같다. 하지만 이번 프로젝트에서 내가 이런 부분에 참여가 잘 되지 않았던 것 같다. 특히 작업의 성능이 잘 나오지 않았던 부분에 대해서만 집중을 하다보니 협업하는 것을 놓친 것이 아쉬웠다.

### 3. 이상민

대회에 들어가면서 목표

- input data 변화주기 (prompt tuning)
- lr scheduler 이용하여 Learning Rate 파악 및 분석
- Model tuning

전과 비교하여 새롭게 도전한 시도

저번 RE task에서는 hugging face에서 불러온 모델에 대한 이해 및 분석을 중심으로 프로젝트를 진행했다. 한 번 해보았기에 모델 튜닝 부분은 가볍게 하고 데이터를 이용한 성능향상을 이뤄내고 싶었다. 처음 대회를 들어가며 목표로 했던, input data에 변화를 주는 시도를 해보지는 못했다. 하지만 curriculum learning 시, 난이도를 측정하기 위해 input data를 확인하고 결과를 낼 때, input data의 전처리 부분에 필요한 부분을 확인했고, 어떤 식으로 data가 모델에 input되는 지 등을 분석할 수 있었다.

마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

hugging face 에서의 Trainer.train() 과 같은 기본적인 함수에 대한 이해가 많이 부족했다. 각 question과 context 별로 predict answer를 뽑아내어 정답과 f1-score를 내어 문제 별 난이도를 측정하고 싶었는데, Trainer.train()과 같은 함수들의 args에 대한 기본적인 이해가 부족해 baseline으로 preprocessing data를 가지고 원하는 목표를 달성할 수 없었다.

너무 baseline code를 이용하려 했던 점이 아쉬웠다. 내가 원하는 output과 현재의 진행 상황을 잘 match 시킬 수 없다면, 처음부터 나만의 코드를 작성해 봐야겠다는 생각을 했다. 이번 대회 그렇지 못했던 점이 아쉽다.

한계/교훈을 바탕으로 다음 프로젝트에서 시도해볼 것은 무엇인가?

baseline 코드를 이용하되, 자신만의 편리한 코드를 만들어 대회를 진행해 보고 싶다. 그리고 이번 대회에서 input data의 변형(prompt tuning)을 시도하지 못한 것이 너무 아쉬웠다. 모델도 중요하지만 data의 정제 및 변형이 실제 output에 더욱 큰 변화를 줄 수 있을 것이라 생각하기에 최종 프로젝트에서는 꼭 input data의 정제 및 형식의 변화를 주는 시도를 해보고 싶다.

느낀점

이번 task는 retrieval과 mrc, 크게 보면 2가지의 문제를 해결했어야 했던 task이다. 해결해야 할 task가 많아져서 복잡해졌고, 성능 개선을 위한 순차적 진행 및 비교가 힘들었다. 그리고 2가지 task이기에 reader와 data 부분을 맡은 나는 retrieval 시 코드의 오류가 발생 시, 오류에 대한 이해가



많이 부족했다. 분업 시, 다른 부분에 대한 이해도 필요하다는 것을 느꼈다.

개인적으로 **Open-Domain**에서 문서를 불러오는 것이 낯설었다. 얼마나 많은 **Data**를 받아오는지, 그 **data**에 대한 비교가 시간을 얼마나 이용하는지 등이 궁금하고 신기했다. 이렇게 실시간 데이터를 끌어올 수 있고, 데이터 저작권만 잘 해결이 된다면 **chat gpt**에 **data injection**과 같은 **task**를 수행할 수 있을까? 에 대한 궁금증이 생기게 되었다. 많이 발전되었다고 생각했던 **NLP** 분야에 흥미가 생기고 새롭게 도전하고 싶은 주제 하나가 더 생겨 좋은 프로젝트 경험이 되었다고 생각한다.

대회 들어가면서 **learning rate** 파악 및 분석과 같이 목표했던 **hyperparameter**에 대한 분석이 잘 이루어지지 못했고 경험적으로 좋은 결과를 내는 **hyperparameter**를 이용하게 되었다. 다음에는 꼭 **hyperparameter**의 채택에도 조금의 논리는 가지고 선택할 수 있게 분석 해보고 싶다.

## 4. 이승우

### 학습 목표 및 목표 달성 과정

지난 대회들을 통해 먼저 현재 주어진 상황을 이해한 뒤 문제의 원인을 분석해 개선 방법을 찾아나가는 방식의 중요성을 깨닫고 이러한 **flow**로 움직이는 것을 목표로 삼았다. 이 목표를 대회 내내 리마인드해가며, 새로운 작업을 무작정 적용해보기 보다는 지금 성능이 안 좋은 이유가 무엇인지, 이 방법이 정말 도움이 될 것인지 근거를 가져가며 움직였다. 그 결과, 나 개인을 넘어 팀 단위로도 대부분의 작업에 근거가 있는 채로 나름의 “스토리 라인”을 만들어가며 대회를 진행할 수 있게 되었다. 나 개인적으로는 먼저 프로젝트에 빠르게 익숙해지고 초기 상태의 문제 원인 분석을 위해 **Baseline code**를 **customizing**을 스스로 맡아 코드를 수정했고, 그 과정에서 프로젝트의 전체적인 구조를 파악하고자 했다. 이 과정이 전체적인 **Git Flow** 관리에도 도움이 되었다.

### 모델 개선 방식

다른 작업 역시 진행했지만, 특히 데이터를 중심으로 봤던 것 같다. 기존에 주어진 데이터셋은 그 양이 적다고 느껴 **KorQuAD**를 추가하고, 이를 중심으로 다양한 작업을 시도해봤던 것 같다. 내가 맡은 작업 중 가장 많은 실험과 가장 큰 성능 향상을 이뤄낸 것은 **Transfer Learning** 파트로, 모델을 2번 사용하는 방법임에 따라 전처리 및 **hyperparameter** 튜닝에 많은 신경을 썼던 것 같다.

### 마주한 한계 및 아쉬웠던 점

시간 분배의 실패가 가장 아쉬웠다. 나 포함 팀원 모두 **Task** 자체의 어려움 때문에 강의를 철저히 듣고 이해하는데 많은 시간을 할애했고, 그럼에도 이해가 부족해 프로젝트 진행 중에도 간과하거나 다시 찾아본 경우가 잦았다. 또, 원인을 파악함에 있어 그 과정이 이전 **Task**에 비해 복잡해 어디에

문제가 있는지는 커녕 어떤 문제가 있는지조차 알기 힘들었던 점도 컸다. 이 때문에 구현을 미처 해보지 못하거나 구현을 해놓고도 제대로 된 실험을 해보지 못해 사용하지 못한 방법들이 많았다. 다만 **MRC Task**가 손에 꼽히는 어려운 **Task**이기도 하고, 대회 덕에 많은 경험을 한 것 같아 다음 프로젝트부터는 체계적인 시간 분배가 가능할 것 같다.

다음 프로젝트에서 시도해볼 것

다음으로 진행할 프로젝트는 **Product Serving**이 포함된 서비스 개발이다. 개인적으로는 대회에서 좋은 성능을 내는 것도 중요하지만 모델이 인간의 삶에 유익함을 주지 못하면 아무 의미가 없다 생각해, 모델을 서비스에 적용하는 과정을 경험하고 싶었다. 서비스 개발인 만큼 나의 개발 경험을 살려 프로젝트 진행 계획을 철저히 수립하고, 협업도 지금보다 꼼꼼히 관리해서 많은 경험을 쌓는 것을 목표로 할 것이다.

## 5. 이에원

이번 프로젝트에서 나의 목표는 무엇이었는가?

이번 **mrc** 대회는 난이도가 있는 **task**라고 알고있던 만큼 **task** 자체의 이해를 제대로 하고 **baseline** 코드에 대한 이해를 초반에 확실히 하고자 하였다.

나는 내 학습목표를 달성하기 위해 무엇을 어떻게 했는가?

우선 첫 일주일을 **mrc** 강의와 **baseline** 코드를 이해하기 위해 거의 썼던 것 같다. 그렇기에 **mrc task** 자체와 **QA** 모델의 구조, 코드 이해는 잘 잡혔고 덕분에 구현에 있어서 기존 코드를 효율적으로 잘 사용할 수 있었다.

나는 어떤 방식으로 모델을 개선했는가? 그리고 어떠한 깨달음을 얻었는가?

**inference** 방식에서 **document** 각각에 대해 **answer**를 추론하는 방식을 구현함으로써 모델을 개선하였고 가장 많은 시간을 투자한 것은 **prompt tuning**이었다. **gpt**와 같은 **LLM**을 효율적으로 이용하기 위한 **prompt tuning**이 중요하다는 관련 자료들을 보고 이번 **task**에 적용하면 무조건 적으로 성능이 오를 줄 알았다. 특히 **MLM**으로 학습된 **bert** 모델을 쓰는 만큼, **question**에서 답이 나와야 할

곳에 [MASK] token을 사용하는 것은 이론상 확실하게 성능향상이 있을 것이라 기대했는데 예상과 다르게 성능이 잘 오르지 않았고 원인 분석이 쉽지가 않았다. 이 과정에서 prompt에 대한 흥미도 생겼지만 생각보다 예민한 아이라는 것을 깨달았다.

마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

마주한 한계는 프로젝트 과정에 개인적인 상태가 영향을 미치지 않도록 하는 것이 상당히 미숙하다는 점이었다. 경험의 부족함으로 인한 한계를 다소 느꼈었던 것 같다. 또한 아쉬웠던 점은 MRC에서 dense retriever와 관련된 연구가 많이 진행되었고 특히 중요한 부분이라고 생각하기에 이에 대해 깊이 있게 공부하고 싶었다. 그러나 막상 맡은 파트가 아니다 보니 관련 논문을 자세히 읽어보지 못한 것이 너무 아쉬웠다.

한계/교훈을 바탕으로 다음 프로젝트에서 스스로 새롭게 시도해볼 것은 무엇일까?

다음 프로젝트는 지금까지처럼 대회 형식의 프로젝트가 아니라 모델의 성능을 올리기 위해 다양한 실험을 해보기 보다는 아마 특정 기능을 구현하기 위해 직접 모델을 훈련시키고 baseline을 직접 만드는 과정이 더 많고 중요할 것이라 예상된다. 지금까지와는 다른 만큼 특정한 목표를 잡을 생각은 없고 최대한 새로운 작업에 빨리 적응하고, 관련 논문을 많이 공부해보는 것이 목표이다.