MRC 랩업 리포트

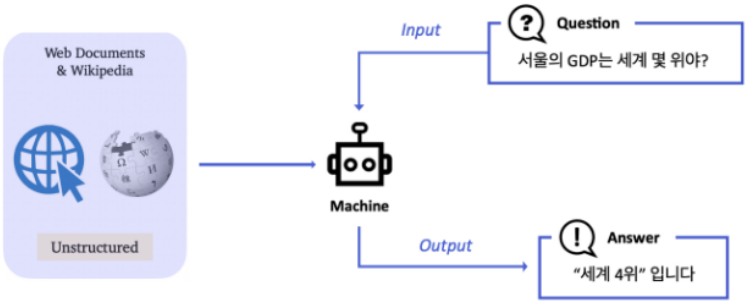
✍️ **순서**

1. Task 소개

- **Open-Domain Question Answering(ODQA)**는 질문에 관한 지문이 사전에 존재하지 않고 구축 되어 있는 knowledge resource 에서 질문에 대답할 수 있는 문서를 찾아 다양한 종류의 질문에 대답하는 인공지능을 만드는 Task 입니다.

Linking MRC and Retrieval: Open-domain Question Answering (ODQA)

ODQA: 지문이 따로 주어지지 않음. 방대한 World Knowledge에 기반해서 질의응답



- 평가 방법
  - **Exact Match(EM)** : 모델의 예측과 실제 답이 정확하게 일치할 때만 점수가 주어짐
  - **F1-Score** : EM과 다르게 겹치는 단어도 있는 것을 고려해 부분 점수를 받음

팀 구성 및 역할

- 이효정(PM): 프로젝트 역할 분담 및 진행 상황 관리, Reader 모델링
- 정지훈(Data): 데이터 분석 및 증강, 후처리 담당
- 김진호(Research): 리서치 및 Dense Retriever 구현 및 훈련
- 신혜진(Code review): 코드 리뷰, Sparse Retriever, Reader 모델링
- 이상문(Code review): 코드 리뷰, Dense Retriever 구현 및 훈련

수행 절차

- 팀 리포지토리는 Git-Flow 규칙에 따라 관리 (Git-flow를 간소화하여 `feat` → `dev` → `master` 로)

- 커밋 메시지 알아보기 쉽게 룰에 따라 작성
- Zoom 필수 접속 시간 외에 Gather, Slack을 이용하여 자유롭게 소통
- Notion의 각자의 실험결과를 정리하여 공유하기

2. 실험 환경 설정

- 모델 학습을 위해서 **Huggingface Trainer** 사용
- 모델별 성능을 확인하기 위해서 **Weight & biases** 사용
- 모델의 hyper-parameter search를 위해서 **Wandb Sweep** 사용

Project Directory 구조

```
├──📁config
│   └── base_config.yaml
│
├──📁data_loaders
│   └── data_loader.py  # reader모델을 위한 데이터셋 로드 및 전처리
│
├──📁dataset  # huggingface load_dataset으로 불러오는 데이터셋
│   ├──📁test_dataset
│   ├──📁train_dataset
│   └── wikipedia_documents.json  # retriever가 검색할 문서 corpus
│
├──📁retriever
│   ├── 📁dpr
│   │   ├── 📁 config # dpr 관련 config
│   │   ├── dense_dataloader.py
│   │   ├── dense_model.py
│   │   ├── dense_retrieval.py
│   │   ├── hard_negative.py
│   │   └── dense_train.py
│   ├── elastic_retrieval.py
│   ├── faiss_retrieval.py
│   ├── retrieval.py  # retriever만 테스트할 때 실행
│   └── sparse_retrieval.py
│
├──📁ssm
│   ├── mlm.py
│   ├── ssm.py
│   └──📁pretrained # pretrained 모델 저장 경로
│
├──📁models # reader 모델이 저장되는 경로.
│   └──📁18-14-42
│       ├── 📁eval # train_eval dataset inference 결과
│       ├── 📁pred # test dataset inference 결과
│       │    ├── nbest_predictions.json  # soft voting에 사용
│       │    └── predictions.json  # 제출
│       └── pytorch_model.bin
│
├──📁trainer
│   └── trainer_qa.py
└──📁utils
    └── utils_qa.py
│
├── README.md
├── arguments.py
├── requirements.txt
├── run_mrc.py
├── run_retrieval.py
└── train.py → train 코드
```

3. 데이터

▼ EDA

MRC 대회에서 주어진 데이터는 다음과 같음

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)		

각각 train data와 valid data의 answer, context, question 컬럼에 대하여 `describe()`를 통하여 기초 통계를 확인하였더니 별다른 특징은 없었음.

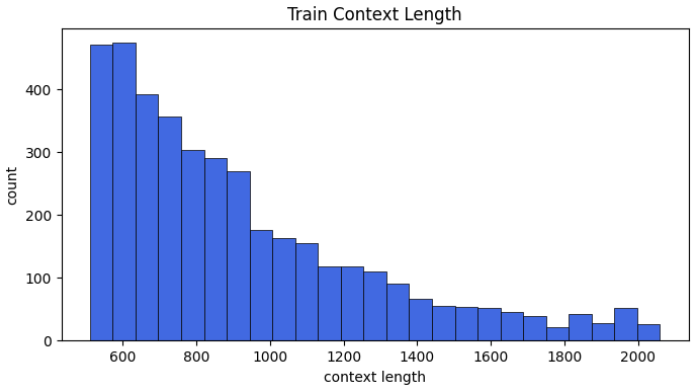
	ans_start	ans_len	ctx_len	qu_len
count	3952.000000	3952.000000	3952.000000	3952.000000
mean	376.794028	6.275051	920.220648	29.322368
std	309.122555	5.346842	356.500514	8.727421
min	0.000000	1.000000	512.000000	8.000000
25%	138.000000	3.000000	645.000000	23.000000
50%	310.000000	5.000000	819.000000	29.000000
75%	538.000000	8.000000	1099.250000	35.000000
max	1974.000000	83.000000	2059.000000	78.000000

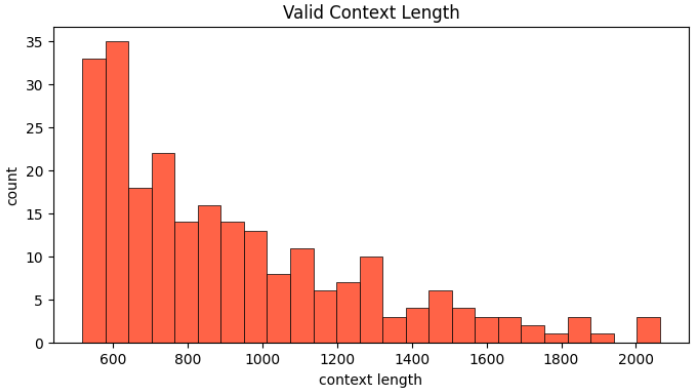
train dataset 기초 통계

	ans_start	ans_len	ctx_len	qu_len
count	240.000000	240.000000	240.000000	240.000000
mean	391.516667	6.912500	916.725000	29.195833
std	311.943965	6.858755	360.032122	8.728301
min	0.000000	1.000000	517.000000	9.000000
25%	154.000000	3.000000	616.750000	23.000000
50%	317.000000	5.000000	820.500000	29.000000
75%	536.000000	8.000000	1107.250000	35.000000
max	1429.000000	64.000000	2064.000000	59.000000

valid dataset 기초 통계

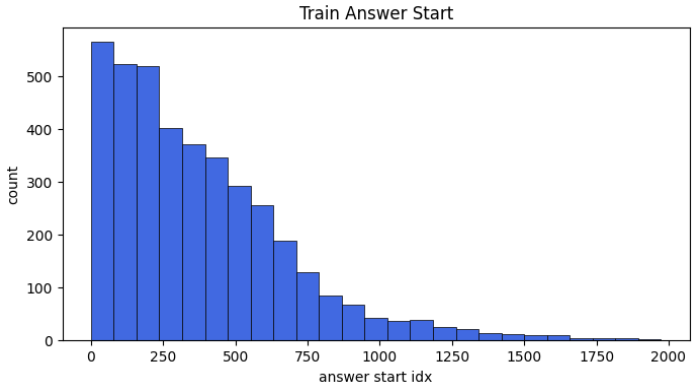
train과 valid의 context 컬럼의 **length**에 대한 분포를 각각 확인하였음.

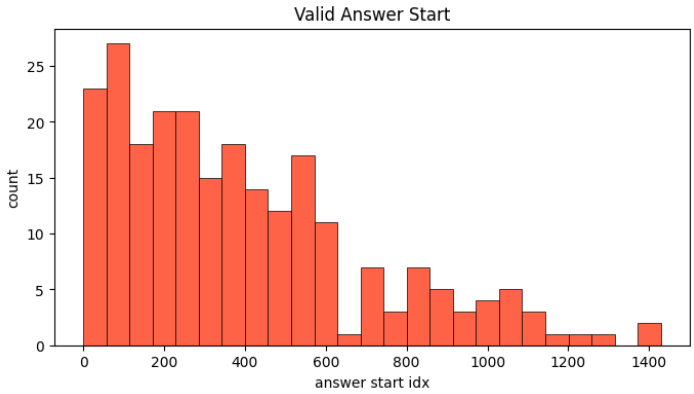




→ context 길이는 대부분 **1000 이하**로 이루어져 있음.

train과 valid의 answer 컬럼 **start index**에 대한 분포를 각각 확인하였음.





→ answer start index 도 역시 대부분 **1000 이하**로 이루어져 있음.

wikipedia document 내 특수문자

1. 한문, 일어, 러시아어 : (青空文庫, あおぞらぶんこ, раф Лев Никола́евич)
2. 개행문자 : \n, \n\n
3. 괄호 : 《》, 〈〉, ()
4. 기타 문자 : •, 『』, ⑥, ↑, ≪, °, ç, *, **, “, ‘

→ wikipedia document 내 특수문자가 존재하지만 tokenizer에서 [UNK]로 나와 따로 처리 X

wikipedia document 문서를 text 기준으로 중복을 제거

- 60613 → 56737로 줄어듦

▼ Post processing

모델 예측 결과, 단어나 문장 끝에 조사가 함께 나오는 경우가 드물게 있음.

konlpy 라이브러리의 Mecab, Hannanum, Okt을 사용하여 예측된 결과의 형태소 분석을 진행함.

1. 예측 결과를 형태소 단위로 분리
2. 예측 결과가 조사로 끝나는 경우 조사를 제거

4. Retriever

▼ Sparse Retriever

- Elastic Search 기본 세팅

```
filter : shingle
tokenizer : nori_tokenizer
decompound_mode : mixed
similarity : BM25
```

- Retriever별 top-k 성능 비교
  - 각 Top-1, 5, 10, 30, 50, 100 별로 성능을 비교

	Top-1	Top-5	Top-10	Top-30	Top-50	Top-100
TF-IDF	0.2791	0.5833	0.6666	0.8000	0.8458	0.9041
FAISS	0.0750	0.0916	0.0916	0.1083	0.1125	0.1166
Elastic BM25	0.5916	0.8666	0.9166	0.9625	0.9666	0.9791

- topk, similarty, wiki 데이터 전처리를 다양한 형태로 실험해봄
  - reader model은 **klue/roberta-large** 고정

Sparse Retriver setting \ 리더보드 public score	**EM**	**F1-score**
TF_IDF topk30 + normal wiki_document data	56.25	67.45
Elastic Search BM25 topk50 + normal wiki_document data	61.25	71.41
Elastic Search BM25 topk25 + normal wiki_document data	62.92	73.27
Elastic Search BM25 topk20 + normal wiki_document data	63.33	73.62
Elastic Search DFR topk20 + wiki_document data(전처리 + 제목)	**65.00**	**75.34**
Elastic Search BM25 topk20 + wiki_document data(전처리 + 제목)	**65.00**	**75.34**

- TF_IDF 보다는 Elastic Search가 월등한 성능을 보여줌
- topk는 20이 제일 좋은 성능을 보여주었으며 topk가 20에서 증가할 수록 성능은 떨어졌음
- 이전 기수에서는 bm25보다는 DFR이 조금 더 나은 성능을 보여주었다고 하였으나 이번 대회에서는 성능상에 차이점을 발견하지 못함 ⇒ 이후 BM25를 계속 사용하기로 결정
- wiki_document에 간단한 전처리와 제목을 붙여주니 성능이 소폭 증가함 ⇒ 제목에 정답과 근접한 키워드가 많이 포함이 되어있다라고 추측

▼ Dense Retriever

- 본 대회 특성상 Sparse Retriever가 성능이 매우 좋게 나오나 Sparse Retriever로 커버가 불가능한 question들이 존재
  - '프랑크 왕족의 무덤의 위치는?'이라는 query에 '전라북도 고성군'이라는 응답이 나옴
  - 단어 기반의 유사도 알고리즘으로는 찾을 수 없는 정답이 포함된 passage들이 있음을 추측함
- **'Dense Passage Retrieval for Open-Domain Question Answering'** 논문을 참조하여
  in-batch-negative와 hard negative 기반의 Dense Passage Retriever를 실험
  - 대회에서 제공하는 기본 데이터로 DPR을 학습 및 검증 (train : 3952, valid : 240)
  - hard negative는 하나의 question에 대하여 Elastic Search로 가장 유사한 passage 2개(top2)중에서 하나를 hard-negative passage로 추가
    - top1, top2중에서 top1이 정답 passage일 경우 ⇒ top2를 hard-negative로 추가
    - top2가 정답이거나 정답이 top1, top2중에 없을 경우 ⇒ top1을 hard-negative로 추가

setting\accuracy	top-1	top-5	top-30	top-50	top-100
in-batch-num_neg 3	0.1167	0.3167	0.4500	0.5250	0.6458
in-batch-num_neg 7	0.2000	0.3542	0.5750	0.6083	0.6875
in-batch-num_neg 3 + hard_negative	0.1917	0.3542	0.5375	0.5792	0.6375
in-batch-num_neg 7 + hard_negative	0.2208	0.4124	0.6125	0.6375	0.7000

- 추가적으로 squad 데이터셋을 추가적으로 활용하여 DPR을 훈련함
  - 기본 훈련세트에 squad 데이터 세트 3000개를 추가로 훈련함

setting\accuracy	top-1	top-5	top-30	top-50	top-100

in-batch-num_neg 5 + hard_negative	**0.3542**	**0.5083**	**0.7250**	**0.8042**	**0.8667**

- Dense Retriever 단독으로는 성능이 나오지 않아 Sparse Retriver와 앙상블로 이용함

▼ Re-Ranking

- Sparse Retriver와 Dense Retriver에서 각각 top 20개의 passage를 뽑아서 유사도 점수를 기준으로 Re-Ranking을 하여 다시 top passage를 선정함
  - Sparse Retriver
    - 위의 세팅과 동일하게 사용 + wiki document(전처리 + 제목)
    - bm25 score를 사용함(score range : 60 ~ 0)
  - Dense Retriever
    - 가장 성능이 좋았던 Dense Retriever 사용
    - dot_prod_scores를 사용(score range : 250 ~ 120)
  - 수식 : **argsort(Sparse Retriver Scores*α, Dense Retriver Scores*(1-α))**

setting \ score	EM	F1
(α=0.1), after re-rank top 20	60.83	73.48
(α=0.1), after re-rank top 30	**64.58**	**76.46**

- Sparse Retriver 실험때와는 달리 top 30 passage가 더 성능이 좋았으나 Sparse Retriver를 단독으로 사용했을 때보다 성능이 떨어짐
  - 추후 Sparse Retriver만 단독으로 사용해서 달성한 sota 모델의 결과와**(EM:65)**
    Sparse Retriver와 Dense Retriver를 Re-Ranking한 결과를 soft-voting하니 성능이 향상됨 (**EM:66.25)**

5. Reader

▼ Pre-training

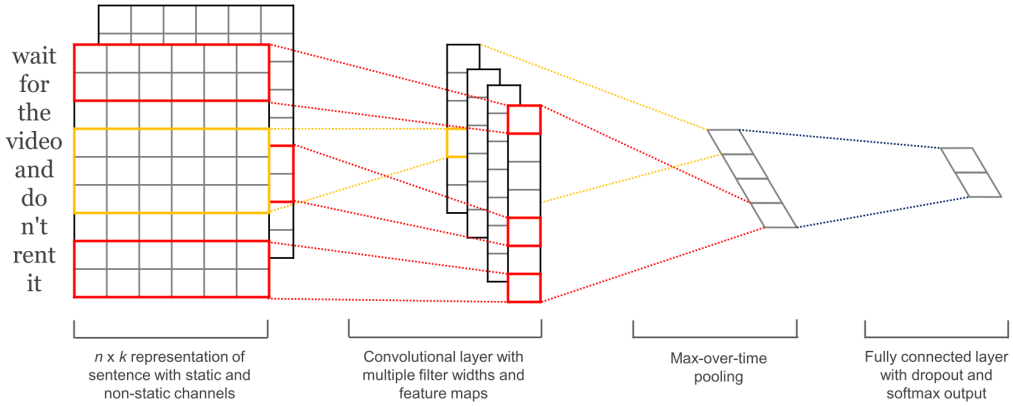
1. **Masked Language Modeling**
   - Reader모델이 답을 찾아야 하는 문서에 대한 사전지식을 Task Adaptive Pre-Training처럼한 번 더 학습하면 좋을 것이라고 가정함.
   - Reader가 읽고 이해해야 하는 wikipedia 문서 전체에 대해서 추가적으로 MLM을 수행.
   - klue/roberta-large모델을 사용해 15% 확률로 token masking해서 1 epoch MLM 결과, evaluation 점수는 EM, F1 점수 모두 1점 정도, 제출 점수는 3~5점 정도 하락함.
   - klue/roberta-large모델이 매우 큰 corpus에 대해 이미 MLM으로 학습되었기 때문에 wikipedia문서에 대한 추가적인 MLM은 불필요하다고 결론 내림.
2. **Salient Span Masking**
   - MLM은 token을 무작위로 masking하지만, Sailent Span Masking은 인물, 장소, 수량 등 named entity에 대해서만 masking해서 pre-training하는 기법임.
   - QA task는 대부분의 정답이 named entity이므로 SSM Pretraining이 MLM보다 도움 될 것이라고 생각.
   - Pororo Ner로 named entity를 추출 후 50%의 확률로 token masking 해서 1 epoch, 2 epoch Pre-training 진행.
   - 두 번째 epoch때는 첫 번째에서 제외된 나머지 50%의 token을 마스킹하여 학습함.
   - 실험 결과 1 epoch SSM 학습 시 evaluation점수는 비슷하게 유지되었고 제출 점수는 5점 정도 하락했지만, 2 epoch 학습했을 때에는 evaluation점수는 2점 상승, 제출 점수는 거의 비슷함.
   - 또한 MLM에 비해서도 약간의 성능 향상을 보임.
   - epoch을 더 여러 번 학습한다면 성능 향상이 있을 것이라고 기대되지만 시간 관계상 실험하지 못함.
3. **Salient Span Masking with QA dataset**
   - 추출된 entity가 QA answer가 아닌 경우에 성능 향상에 방해가 될 수도 있다고 생각해 train dataset의 passage에서 정답인 단어들을 masking하고 pre-training하는 방법 시도.
   - passage하나 당 한 개의 단어만 masking하면 너무 적기 때문에 정답 위치가 아니더라도 정답 단어와 일치한다면 모두 마스킹함.
   - 실험 결과 wikipedia dataset보다 훨씬 적은 데이터로 학습했음에도 SSM 1epoch와 비슷한 성능을 보임. 하지만 epoch를 늘리면 성능이 떨어졌는데, 데이터셋과 masking할 token 수의 한계 때문인 것으로 추측함.

reader	eval/EM	eval/F1	public/EM	public/F1	private/EM	private/F1	
klue/roberta-large	**68.75**	**77.79**	**64.17**	**74.39**	**60.83**	**73.87**	

klue/roberta-large + MLM 1 epoch	67.08	76.36	56.25	65.99	56.50	68.73	
klue/roberta-large + SSM 1 epoch	68.75	77.80	56.25	67.62	56.56	69.04	
klue/roberta-large + SSM 2 epoch	**70.0**	**78.38**	**57.92**	**68.67**	**60.28**	**72.26**	
klue/roberta-large + SSM with QA dataset	67.91	77.52	50.42	64.04	54.44	68.11	

▼ Reader + CNN

- CNN은 토큰의 지역적인 특징을 반영할 수 있다. 이를 이용하여, 모델이 좀더 제대로 위치를 예측할 수 있도록 CNN 을 모델 끝에 추가함.
- 모델 구조는 아래와 같다. 마지막 Fully Connected layer의 경우 lstm으로 바꿔 사용하는 실험도 진행함.



- 결과
  실험은 transformer model(klue/roberta-large)을 epoch 2로 학습한 뒤 이를 freeze 하고 cnn과 lstm 모델만 epoch 5로 학습을 진행함.
  실험결과 cnn은 점수에 영향을 주었지만 lstm은 점수에 영향을 주지 않음.
  cnn과 klue/roberta-large를 동시에 epoch 2로 학습시켰을 때 학습이 제대로 이루어지지 않았다. 이러한 결과는 transformer model, cnn, lstm이 각각 학습에 있어 필요한 하이퍼 파라미터 수가 다르기 때문으로 추측함.

reader	eval/EM	eval/F1	public/EM	public/F1	private/EM	private/F1	
klue/roberta-large + cnn	68.75	77.19	60.45	72.70	63.06	74.95	
klue/roberta-large + cnn + lstm	68.75	76.99	60.83	73.20	62.78	74.94	

- 추가 실험

  deberta와 electra에 대해서도 실험을 진행했는데 이 두 모델 모두 baseline에 대해 30점대의 낮은 성능을 보였을 뿐만 아니라, cnn에 대해서도 학습이 제대로 이뤄지지 않음 baseline보다 더 좋지 않은 pred 값을 내보낸 것으로 보아 학습된 모델을 불러오는 것에 문제가 있었던 것으로 보임.

▼ Curriculum Learning

- Curriculum Learning에서는 데이터의 난이도에 따라 데이터 종류를 상/중/하로 나누고 난이도 하, 난이도 중, 난이도 상 순서로 데이터를 학습함.
- 방법
  - klue/roberta-large로 1차 inference를 진행하여 train dataset에 대한 예측값을 계산. 각 학습의 epoch은 2로 고정함.
  - start_index와 end_index 예측값에 대해 각각 L2 Loss를 계산. 두 값의 합을 각 데이터의 Loss로 계산함.
  - Loss가 작은 순서로 나열하여 전체 데이터의 1/3을 난이도 하로 지정하고 이를 학습함.
  - 난이도 하에 대해 학습된 모델로 다시 train dataset을 예측함.

    - train dataset의 L2 loss 합을 계산하여 전체 데이터의 1/3을 난이도 중으로 지정한다. 처음 난이도 하로 1/3을 사용하고, 이 단계에서 1/3을 난이도 중으로 사용한다. 남은 데이터는 난이도 상으로 본다.
    - 난이도 중, 난이도 상으로 학습을 진행함.
- 결과

  실험 결과 학습이 제대로 진행되지 않았다. 난이도를 L2 Loss 값을 기준으로 하지 않고 데이터의 양으로 한 것이 문제일 수도 있다. 다른 이유로는 CNN 실험과 마찬가지로 난이도 중, 난이도 상 학습에는 이전에 학습된 가중치를 불러와 업데이트하는 방향으로 학습을 진행했는데 가중치 업데이트가 제대로 이뤄지지 않았을 가능성이 있다.

reader	eval/EM	eval/F1	public/EM	public/F1	private/EM	private/F1	
Curriculum Learning	**61.66**	71.28	50.00	63.91	54.44	67.76	

6. 앙상블

- **Soft voting**
    - Ensemble model (reader 기본 모델은 klue/roberta-large)

model \ public score	EM	F1-score
Elastic bm25 top 20	65.42	75.66
Re-Rank(Elastic bm25 top20 + DPR top20) dpr 가중치 0.1	64.58	76.46
Re-Rank(Elastic bm25 top20 + DPR top20) dpr 가중치 0.115	62.33	75.65
Salient Span masking + Elastic bm25 top20	57.92	68.6700
CNN layer + Elastic bm25 top20	60.83	73.2

    - 위의 다섯개 모델을 앙상블 + 후처리

public EM score	public F1-score	private EM score	private F1-score
67.5	**78.17**	**65.56**	**77.25**

7. References

- Kumar, Varun, Ashutosh Choudhary, and Eunah Cho. "Data augmentation using pre-trained transformer models." *arXiv preprint arXiv:2003.02245* (2020).
- Karpukhin, Vladimir, et al. "Dense passage retrieval for open-domain question answering." *arXiv preprint arXiv:2004.04906* (2020).
- Guu, Kelvin, et al. "Retrieval augmented language model pre-training." *International Conference on Machine Learning*. PMLR, 2020.
- Bengio, Yoshua, et al. "Curriculum learning." *Proceedings of the 26th annual international conference on machine learning*. 2009.

8. 자체 평가 의견

- 잘했던 부분
    - Retriever(DPR)에서 다양한 실험을 진행했다.
    - 이전 기수와 달리 Salient Span masking을 시도했다.
- 아쉬웠던 점
    - 데이터 분석이 부족했다.
    - 시간이 부족해 다양한 실험을 진행하지 못했다.

9. 개인 회고

▼ 이효정

- 대회에 집중하지 못해 다양한 실험을 진행하지 못한 것이 아쉽다.
- baseline code가 이전과 달리 복잡하게 설계되어 있어 이해하는 것이 어려웠지만, 많은 도움이 되었다.

▼ 정지훈

- 잘했던 부분

    - 다른 팀원들이 작성한 코드를 돌려보면서 전체적인 코드 구성과 흐름을 파악하는데 집중함. 덕분에 MRC 일련의 과정에 대한 이해를 할 수 있었음.
    - 대회 난이도에 비해 짧은 기간에도 불구하고, 다른 팀원들이 작성한 코드를 통해 다양한 조건에서 retrieval을 실험해 보았고 reader에서도 한두 가지 모델을 적용해 볼 수 있었음.
    - 외부 데이터가 사용 가능했기 때문에 데이터 증강은 필수적이지 않았지만, mT5 모델을 사용한 질문 생성을 시도해 보았음. 다만 생성된 질문의 질이 좋지 않아 사용하지는 않았음.
  - 아쉬웠던 점들
    - 입출력 데이터 분석을 자세히 해보고 싶었는데, 대회 특성상 수치화 또는 시각화를 통한 데이터 분석의 폭이 넓지 않아 아쉬웠음.
    - 새로운 모델을 시도해보지 않고, 다른 팀원들이 빌드업한 코드를 돌리는 수준에 그쳤음.
    - 임베딩 단에서 출력을 조정하거나 앙상블 시도를 하지 못한 점은 아쉬움.

▼ 김진호

- 잘했던 부분
  - 리서치 부분을 담당하여 시도해볼만한 내용들을 상세히 정리하였음
  - Dense Passage Retriever 논문을 구현하는 과정을 거친 좋은 경험을 하였음
  - 기존 Klue-MRC와 KorQuAD 데이터 약 3000개를 더하여 학습한 것과 비교
  - Sparse retrieval와 dense retrieval를 top-k에 대하여 상세히 비교하였음
- 아쉬웠던 점들
  - 다양한 방식으로 EDA를 진행하지 못하였음
  - Retrieval 및 Reader 모델이 inference을 통해 나온 결과에 대하여 분석을 못하였음
  - 이번 대회는 절대적으로 시간이 부족했다 보니 여러 가지 시도를 못한 것이 많이 아쉬움

▼ 신혜진

- Salient Span Masking에 대해 새로 알게 되었고, 직접 구현하고 테스트해 보면서 많이 배울 수 있었다. 또한 이전 기수들이 시도해보지 않은 방법을 시도해본 것이라서 좋았다.
- 첫 번째 주는 강의 듣느라, 두 번째 주는 새로고침 데이 때문에 프로젝트에 집중할 시간이 많이 없어서 이전 대회들만큼 다양한 방법들을 시도해보지 못한 것이 조금 아쉽다.
- QA task는 데이터 EDA나 output분석이 어려워서 거의 하지 못했는데, 어떻게 하면 더 잘 할 수 있을지 알아보고 싶다.
- Eval 점수, 리더보드 Public 점수, Private점수 차이가 심해서 SSM의 eval score는 좋았지만 public점수가 좋지 않아 성능이 안 좋은 줄 알고 더 실험하지 않았는데 private점수 공개 후 점수가 나쁘지 않아서 더 실험해봤어도 좋을 것 같다.

▼ 이상문

- 잘했던 부분
  - Dense Passage Retriver 논문에 기반하여 해당 내용을 구현해본것은 매우 좋은 경험이 였다고 생각함
- 아쉬웠던 부분
  - Dense Retriver부분에서 좀 더 다양한 시도를 해보지 못한점은 매우 아쉬웠음
  - Private score가 공개된 후 Reader쪽에서 다양한 시도를 해본 모델들이 오히려 점수가 올라간 것을 확인함.
  - Public score 점수만 보고 Reader쪽의 실험들이 효과가 없다고 너무 빠르게 결론을 내려버렸음
  - 전체적으로 다양한 시도들을 실험해 볼 시간이 너무 적었던 부분이 너무 아쉬웠음