




Wrap-Up Report

- [1. 프로젝트 개요](#)
 - [2. 프로젝트 팀 구성 및 역할](#)
 - [3. 데이터셋 설명](#)
 - [4. 프로젝트 수행 절차 및 방법](#)
 - [4.1. Workflow](#)
 - [5. 프로젝트 수행 결과](#)
 - [5.1. Retriever](#)
 - [Sparse Embedding](#)
 - [Elastic Search](#)
 - [Split Text](#)
 - [Retrieval Ensemble](#)
 - [5.2. Reader](#)
 - [Pretrained-model Fine-Tuning](#)
 - [Curriculum Learning](#)
 - [Data Augment with KorQuAD](#)
 - [Inference \(Retriever + Reader\)](#)
 - [5.3. Ensemble](#)
 - [soft-voting](#)
 - [6. 자체 평가 의견](#)
 - [6.1. 프로젝트 달성 요소](#)
 - [6.2. 프로젝트 자체 평가](#)
- [피드백](#)

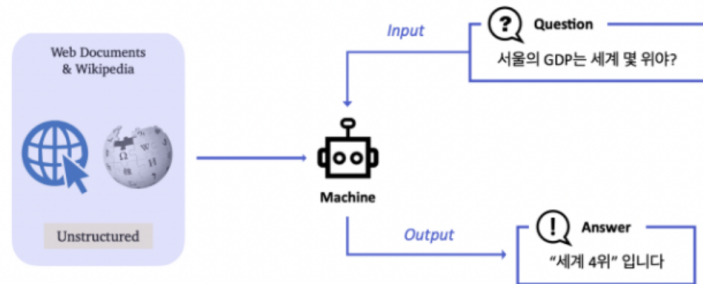
[최종 순위 : 3위]

순위	팀 이름	팀 멤버	EM ↕	F1 ↕	제출 횟수	최종 제출
3 (6 ▲)	NLP-02조	  상윤  우진	66.9400	78.1100	18	4d

1. 프로젝트 개요

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

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



- ODQA 대회에서 사용 할 모델은 two-stage로 구성되어 있다. 첫 단계는 질문에 관련된 문서를 찾아주는 "retriever" 단계이고, 다음으로는 관련된 문서를 읽고 적절한 답변을 찾거나 만들어주는 "reader" 단계이다. 두 가지 단계에 대한 모델을 각각 구성하고 그것들을 적절히 통합하여, ODQA 시스템 모델을 생성한다.
- EM 기준으로 리더보드 등수가 반영되고, F1은 참고용으로만 활용한다. 답이 하나가 아닐 수 있는데, 이런 경우는 하나라도 일치하면 정답으로 간주한다.
- KLUE-MRC 데이터셋을 제외한 모든 외부 데이터 사용 허용한다.

2. 프로젝트 팀 구성 및 역할

- 고우진 : Model fine-tuning, Curriculum learning, Data Augmentation
- 현승엽 : Elastic Search, Retrieval Ensemble, Ensemble
- 김상윤 : Model fine-tuning, Curriculum learning, Data Augmentation

3. 데이터셋 설명

- Train / Valid / Test dataset
 - Train과 Validation 데이터의 경우 각각 3952, 240개가 주어졌다. 각 데이터는 title, context, question, id, answers, document_id의 정보를 담고 있고 이를 이용하여 retriever, reader 파트에서 모델을 학습하고 선택하는데 사용한다.
 - 질문(query)과 질문에 대응하는 id가 주어진 600개의 test 데이터가 있고 이를 통해 inference한 결과를 제출하면 public, private에서 각각 240개, 360개의 데이터를 이용해 평가했다.
- wikipedia_documents.json
 - Retriever를 만들기 위해 wikipedia로 부터 가져온 60,613개의 데이터가 있고, 이 중 57000개가 unique한 문서 데이터이다. Text, document_id, text_length 를 주요 정보

로 가지고 있다.

4. 프로젝트 수행 절차 및 방법

4.1. Workflow

- EDA
- Retrieval



- Reader



- Inference

5. 프로젝트 수행 결과

5.1. Retriever

Sparse Embedding

- 베이스라인의 SparseRetrieval 클래스를 이용하면 TF-IDF를 이용한 sparse embedding을 구성하여 passage retrieval을 수행할 수 있다. TF-IDF에 기반한 sparse embedding을 구성 알고리즘인 BM25를 `rank_25` 패키지를 이용하여 구현하였다.

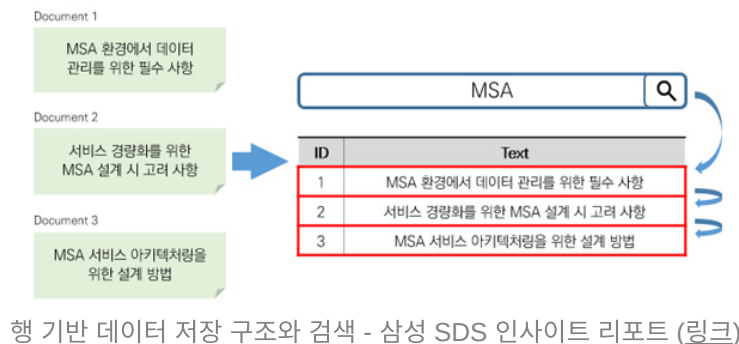
- retriever 모델이 얼마나 정확하게 문서를 선택하는지와 각 알고리즘 별 성능을 좀 더 다양하게 비교하기 위해 retriever 모델로 부터 유사도가 높은 상위 k개의 문서를 전달받아 정답을 포함하는지 확인해 정확도를 계산하였다.
- 베이스라인을 이용하여 TF-IDF와 BM25의 top-k(1, 10, 20, 40)에 대한 성능을 비교하였다. Retriever로 부터 받은 top k개의 문서 중 정답을 포함한 비율을 accuracy로 계산하였다. 아래 표의 값은 계산된 accuracy이다.

	Top-1	Top-10	Top-20	Top-40
TF-IDF	0.251908	0.635734	0.727814	0.796278
BM25	0.463814	0.729515	0.869394	0.881820

- 모든 k에 대하여 BM25가 TF-IDF보다 좋은 성능을 보임을 알 수 있는데 이는 BM25가 기존의 TF-IDF에서 추가적으로 문서의 길이를 고려하고 TF의 범위를 제한하여 일정한 범위를 유지하도록 하여, 좀 더 유용한 sparse embedding을 만들 수 있기 때문이다.

Elastic Search

- Elastic Search는 자바 언어로 이루어진 정보 검색 라이브러리 소프트웨어인 아파치 루씬 (Apache Lucene) 기반의 검색 엔진이다. Elastic Search는 inverted index 구조를 사용하여 행 기반 데이터 저장 방식을 사용하는 기존의 DBMS에 비해 빠르고, 정확하게 데이터를 검색할 수 있다.



- Elastic Search의 설치와 구현은 [부스트캠프 3기 NLP 9조의 깃허브](#)와 관련 블로그(Link)를 참고하였다. 서버에 설치를 마친 후 기존의 retrieval.py에 `ElasticRetrieval` 클래스를 추가하여 이를 사용할 수 있도록 하였고 sparse embedding을 생성하기 위한 세팅을 `settings.json` 를 통해 구성하였고 embedding 생성을 `elastic_setting.py` 를 통해 할 수 있도록 구현하였다.
- Elastic Search에서는 질문(query)과 문서(passage)의 유사도 계산에 기본적으로 BM25 알고리즘을 이용하고 있다. 또한 elastic search는 기존의 TF-IDF와 BM25에 더해서, 확률 모형을 고려한 유사도 알고리즘인 DFR(Divergence From Randomness), DFI(Divergence From Independence), IB(Information Based model), LM Dirichlet 그리고 LM Jelinek Mercer 를 제공한다.
- 베이스라인에서 주어진 기본 검색 방법과 elastic search의 비교와 각 알고리즘에 대한 성능 비교를 4가지 top k (1, 10, 20, 40)에 대해 비교하였다.

	Top-1	Top-10	Top-20	Top-40
TFIDF (Baseline)	0.251908	0.635734	0.727814	0.796278
BM25 (Baseline)	0.463814	0.729515	0.869394	0.881820
TFIDF	0.456106	0.844465	0.880486	0.914599
BM25	0.682352	0.924369	0.946218	0.966386
DFR	0.663865	0.915966	0.934453	0.951260
DFI	0.608403	0.889075	0.915966	0.932773
IB	0.620168	0.895798	0.914285	0.939495
LM Dirichlet	0.625210	0.894117	0.919327	0.934453
LM JelinekMercer	0.638655	0.904201	0.926050	0.946218

- 우선 elastic search가 기존의 모델보다 1.5배가량 성능이 좋음을 알 수 있다. Elastic search가 빠르고 정확하게 문서를 찾아준다는 것을 확인할 수 있다.
- 여러가지 similarity 알고리즘을 비교해봤을 땐 BM25의 성능이 가장 좋음을 알 수 있다. 각 similarity의 특성이 조금씩 다르기 때문에, 각 알고리즘을 이용한 모형들을 앙상블 단계에 이용하였다.

Split Text

- Corpus로 사용하는 wikipedia 데이터를 살펴봤을 때, 정답과 관련없는 부분이 포함되어 문서의 길이가 불필요하게 길어지는 경우가 많다는 것을 확인하였다.

- 한 문서에서 질문에 대한 정답을 추론하기 위해선 문서 전체가 필요하진 않겠다는 생각으로 validation 데이터의 문서들을 `\n` 을 기준으로 분할하여 정답이 포함된 부분만 데이터로 남겼다.
- 이를 이용하여 retriever 모델을 만들면 좀 더 정확하게 질문에 대한 답을 찾을 수 있을 것이라고 기대했다.
- 아래 표는 BM25를 기준으로 기존 elastic search에 기본 데이터를 이용한 모형과 분할된 데이터를 이용한 retrieval 모형의 accuracy를 보여주고 있다.

	Top 1	Top 10	Top 20	Top 40
Original data	0.682352	0.924369	0.946218	0.966386
Splited Data	0.601680	0.848739	0.878991	0.894117

- 성능이 기존보다 떨어졌는데, 우선 corpus의 문서들을 좀 더 세부적인 기준을 두고 나눴다면 향상된 성능을 보였을 것이라고 생각한다. 단순히 `\n` 로만 나누는 것이 아니라 문서에서 분할되는 부분의 길이도 일정하게 고려하고, 같은 `\n` 이라도 문서에 컴퓨터 코드 등이 포함된 경우엔 그 의미가 다르기 때문에 이런 점들을 데이터 분할에 적용 했다면 더 좋았을 것 같다.
- 데이터에서 잘라낸 부분들이 정답과는 관련이 없지만 질문과의 유사도를 생각했을 땐 유의미하게 작용할 수 있기 때문에 성능이 떨어졌을 수 있겠다고 생각했다.

Retrieval Ensemble

- Soft voting을 이용해 앙상블을 하기 위해선 각각의 retrieval가 입력받는 하나의 query에 대해 약 6만개의 문서에 대한 score를 저장하고 있어야하고 이를 계산해야하기 때문에 시간이 굉장히 많이 필요하다. 따라서 방법을 조금 달리하여 다른 similarity 알고리즘을 이용한 각 모형에서 top-30개의 문서에 대한 score만 저장한 뒤 score를 이용하여 soft voting을 진행하였다.
- 각 모형의 30개 문서에 대한 score에 min-max normalization을 적용한 뒤 softmax 함수를 통해 확률화하였다. 그리고 사용한 9개의 모형에서 나온 각각 30개의 문서에 대한 확률을 더하여 가장 확률이 높은 top-k개의 문서를 선택하였다. 이 때 top-2의 문서를 선택하는 모형을 만들어 정확도를 계산했고 이를 기존에 가장 성능이 좋았던 BM25의 top-2 모델과 비교했을 때 약 0.1%의 성능 향상을 보였다.

	BM25	Ensemble
Accuracy	0.825210	0.826890

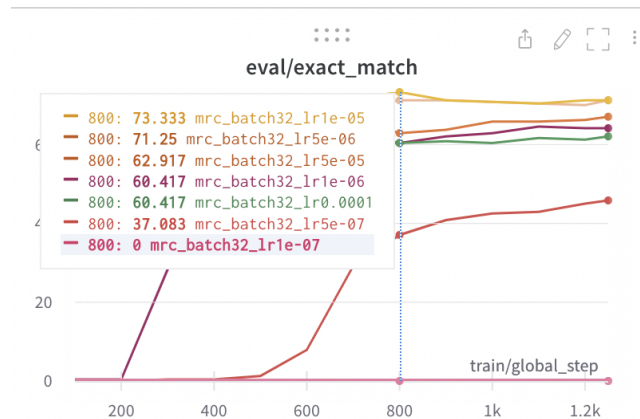
5.2. Reader

Pretrained-model Fine-Tuning

• 하이퍼파라미터 튜닝 1

◦ roberta-large

- roberta-large pretrained-model을 가져와서 하이퍼파라미터 튜닝 진행.
- batch-size = [4, 8, 12, 16, 32]
learning-rate = [1e-7, 5e-7, 1e-6, 5e-6, 1e-5, 5e-5, 1e-4]



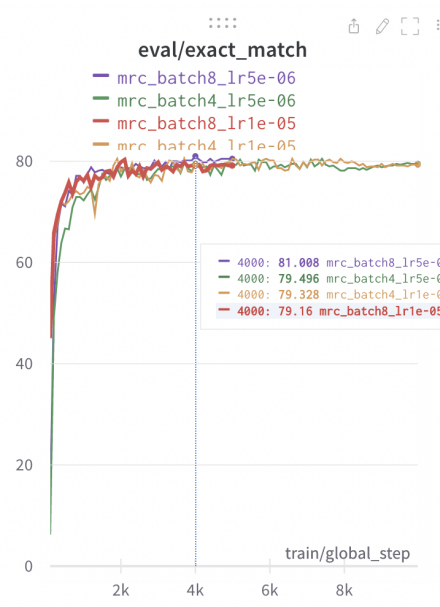
◦ kobigbird, bert-multilingual

- kobigbird, bert-multilingual pretrained-model을 가져와서 하이퍼파라미터 튜닝 진행.
- batch-size = [4, 8, 12, 16, 32]
learning-rate = [1e-7, 5e-7, 1e-6, 5e-6, 1e-5, 5e-5, 1e-4]
- roberta-large 대비 성능이 좋지 못하였고, 추후의 실험에 있어서 roberta-large pretrained model을 사용하여 진행하였다.

• 하이퍼파라미터 튜닝 2

- 대회에서 제공한 validation의 개수는 240개로 성능을 검증하는데에 있어 충분한 양이 아니라는 판단을 하였다. 따라서 대회에서 제공한 데이터와 형태가 유사한 KorQuAD 1.0 train 데이터셋에서 355개의 데이터를 추출해와 validation의 개수를 595개로 늘려 진행하였다.(KorQuAD 1.0 데이터셋에 대한 설명은 추후 아래에서 진행할 것)

◦ roberta-large



Model	Batch size	Learning rate	EM	F1
roberta-large	4	1e-05	80.504	84.539
roberta-large	4	5e-06	80.504	84.096
roberta-large	8	1e-05	80.336	84.376
roberta-large	8	5e-06	81.008	84.904
roberta-large	12	1e-05	79.832	83.928
roberta-large	12	5e-06	78.824	83.084
roberta-large	16	1e-05	80.168	84.41
roberta-large	16	5e-06	78.824	82.936
roberta-large	32	1e-05	78.487	83.346
roberta-large	32	5e-06	77.815	82.604

Curriculum Learning

- Curriculum Learning
 - Curriculum Learning for Natural Language Understanding** 논문을 참고해서 Reader 학습 개선을 시도해 보았다.
 - 1. train-dataset을 n개(10개로 진행)로 쪼갬 뒤 각 쪼개진 dataset으로 fine-tuning된 모델 10개를 생성
 - 2. Cross-Review
 - : 쪼개진 10개의 각 dataset에 대한 Inference를 10개의 모델로 모두 진행하여, 모든

data에 대해 난이도를 측정. (Difficulty : 10개의 모델 중 몇 개의 모델이 EM 기준으로 예측을 맞게 했는가)

3. 난이도를 기준으로 예측이 가장 쉬운 데이터셋부터 C_1, C_2, ..., C_10 순으로 이름을 붙인다.
4. 쉬운 난이도의 data(C_1)부터 점차 어려운 난이도(C_10)의 data가 학습될 수 있도록 dataset을 재구성해서 1epoch 학습. (case1 ~ case5 다양한 방법론으로 실험)
5. 기존 train-dataset를 이용해 5epoch 더 학습하여 EM기준 최고점 check-point로 모델 생성.

- **case1 - cross-review + annealing(no_shuffle)**

- i번째 stage(S_i)는 C_1의 1/10, C_2의 1/10, ... C_i의 1/10 을 순서대로 위에서부터 병합하여 구성된다. (i = 1, ..., 10)
- 데이터셋은 S_1, S_2, ..., S_10을 순서대로 위에서 부터 병합하여 구성된다. (S_1 + S_2 + ... + S_10)

- **case2 - cross-review + annealing(shuffle) → 논문에서 제시한 방법**

- i번째 stage(S_i)는 C_1의 1/10, C_2의 1/10, ... C_i의 1/10 을 순서에 상관없이 shuffle 후 병합하여 구성된다. (i = 1, ..., 10)
- 데이터셋은 S_1, S_2, ..., S_10을 순서대로 위에서 부터 병합하여 구성된다. (S_1 + S_2 + ... + S_10)

- **case3 - cross-review + naive-order**

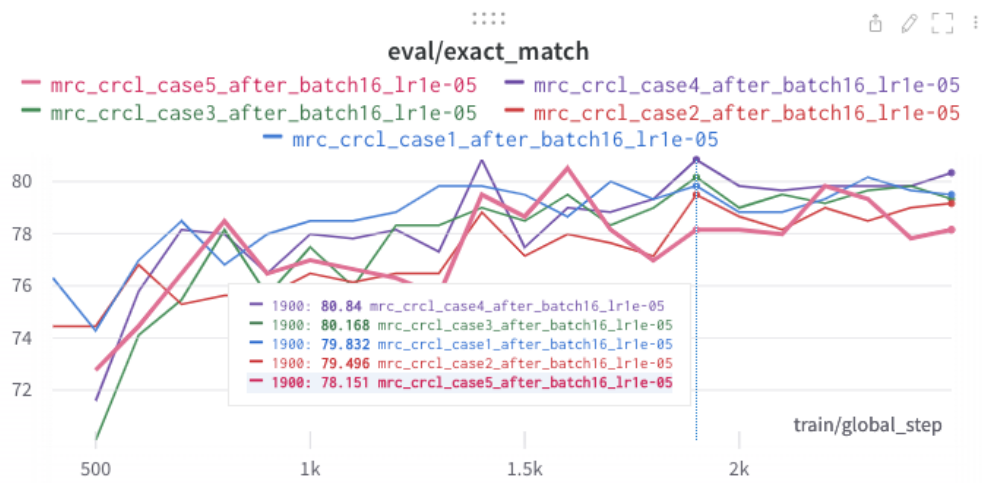
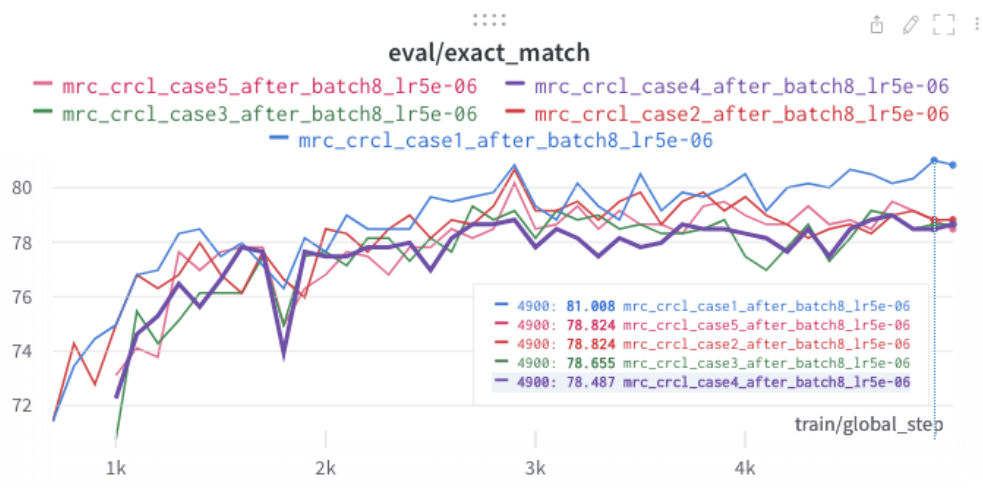
- i번째 stage(S_i)는 C_i 이다. (i = 1, ..., 10)
- 데이터셋은 S_1, S_2, ..., S_10을 순서대로 위에서 부터 병합하여 구성된다. (S_1 + S_2 + ... + S_10)

- **case4 - cross-review + annealing w/ full-data(no_shuffle) → 우리가 새로 고안한 방법**

- i번째 stage(S_i)는 C_1의 1/10, C_2의 1/9, ... C_i의 1/(11-i) 을 순서대로 위에서부터 병합하여 구성된다. (i = 1, ..., 10)
- case1이 일부데이터를 사용하지 못하는 반면, case4의 경우 모든 데이터를 사용할 수 있다.
- 데이터셋은 S_1, S_2, ..., S_10을 순서대로 위에서 부터 병합하여 구성된다. (S_1 + S_2 + ... + S_10)

- **case5 - cross-review + annealing w/ full-data(shuffle) → 우리가 새로 고안한 방법**

- i 번째 stage(S_i)는 C_1 의 $1/10$, C_2 의 $1/9$, ... C_i 의 $1/(11-i)$ 을 순서에 상관없이 shuffle 후 병합하여 구성된다. ($i = 1, \dots, 10$)
- case2가 일부데이터를 사용하지 못하는 반면, case5의 경우 모든 데이터를 사용할 수 있다.
- 데이터셋은 S_1, S_2, \dots, S_{10} 을 순서대로 위에서 부터 병합하여 구성된다. ($S_1 + S_2 + \dots + S_{10}$)



	Batch size	Learning rate	EM	F1
roberta-large w/ crcl - case1	8	5e-06	81.008	84.765

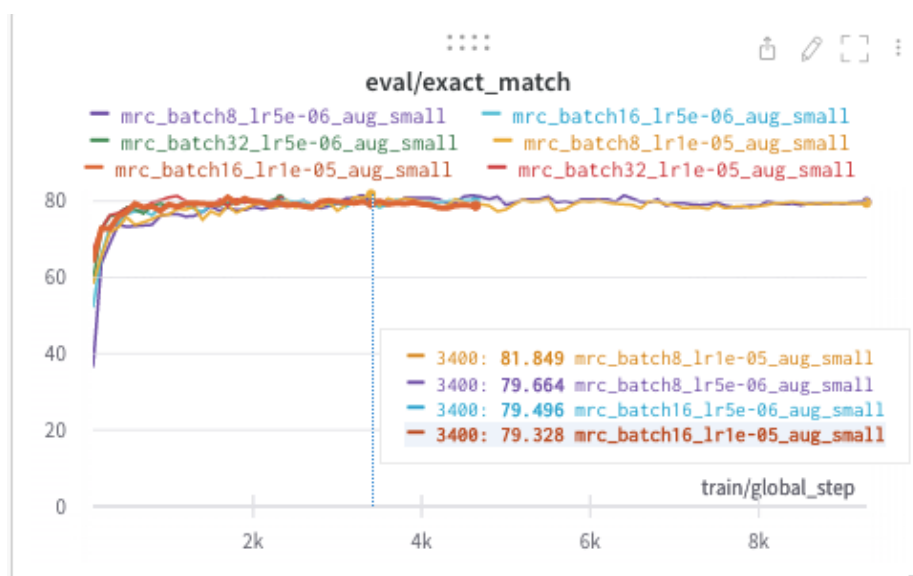
	Batch size	Learning rate	EM	F1
roberta-large w/ crcl - case2	8	5e-06	80.672	84.477
roberta-large w/ crcl - case3	8	5e-06	79.328	83.255
roberta-large w/ crcl - case4	8	5e-06	78.992	82.945
roberta-large w/ crcl - case5	8	5e-06	80.168	84.416
roberta-large w/ crcl - case1	16	5e-06	80.168	83.853
roberta-large w/ crcl - case2	16	5e-06	79.496	83.682
roberta-large w/ crcl - case3	16	5e-06	80.168	84.411
roberta-large w/ crcl - case4	16	5e-06	80.84	84.691
roberta-large w/ crcl - case5	16	5e-06	80.504	84.036

- Batch size = 8, lr = 5e-06의 경우 기존의 EM값은 **81.008**이었다. Curriculum Learning case1~case5 중에서 case1이 **81.008**의 동일한 EM값을 달성했고 나머지는 이보다 낮은 값을 기록했다.
- Batch size = 16, lr = 5e-06의 경우 기존의 EM값은 **78.824**이었다. Curriculum Learning case1~case5 중에서 case4가 **80.84**의 EM값을 달성해 기존 EM대비 약 2정도 증가했다. 나머지 역시 기존보다 높은 값을 기록했다.
- curriculum learning 방법을 사용했을때, 동일한 세팅(Batch size, lr)에서 기존 대비 성능이 동일하거나 증가함을 확인 할 수 있었다. 하지만 시간 제약상 모든 경우에 대해 실험해보지 못하였고, 각 경우마다 높은 값을 기록하는 curriculum learning의 case가 달랐다. 따라서 성능향상에 있어 curriculum learning이 실제 효과가 있는지를 정확히 검증하기 위해서는 추가적인 실험이 필요해 보인다.

Data Augment with KorQuAD

- KorQuAD 1.0은 한국어 Machine Reading Comprehension을 위해 만든 데이터셋으로 모든 질의에 대한 답변은 해당 Wikipedia article 문단의 일부 하위 영역으로 이루어진다. Stanford Question Answering Dataset(SQuAD) v1.0과 동일한 방식으로 구성되었다.

- KorQuAD 1.0의 전체 데이터는 1,560 개의 Wikipedia article에 대해 10,645 건의 문단과 66,181 개의 질의응답 쌍으로, Training set 60,407 개, Dev set 5,774 개의 질의응답쌍으로 구분되어있다.
- 대회에서 주어진 데이터는 train - 3952개, validation - 240개로 충분한 정보 학습과, 적절한 검증에 있어서 그 양이 충분치 않다고 생각하였다. 따라서 아래와 같이 기존 대회에서 주어진 데이터에, KorQuAD 1.0 데이터를 합친 데이터셋을 2개 얻었고, 각각을 이용해 fine-tuning을 진행해 보았다.
- aug_small
 - train - 9726개 (대회 train + KorQuAD 1.0 validation)
 - validation - 595개 (대회 validation + KorQuAD 1.0 train 일부)



	Batch size	Learning rate	EM	F1
roberta-large fine-tuned w/ aug_small	8	1e-05	81.849	85.232
roberta-large fine-tuned w/ aug_small	8	5e-06	81.681	84.615
roberta-large fine-tuned w/ aug_small	16	1e-05	81.008	84.596
roberta-large fine-tuned w/ aug_small	16	5e-06	80.84	84.159
roberta-large fine-tuned w/ aug_small	32	1e-05	81.345	84.9
roberta-large fine-tuned w/ aug_small	32	5e-06	80.672	84.281

- aug_large
 - train - 69893개 (대회 train + KorQuAD 1.0 train 일부 + KorQuAD 1.0 validation)
 - validation - 595개 (대회 validation + KorQuAD 1.0 train 일부)



	Batch size	Learning rate	EM	F1
roberta-large fine-tuned w/ aug_large	16	5e-06	83.697	87.462
roberta-large fine-tuned w/ aug_large	32	5e-06	83.193	86.149

대회에서 제공한 데이터셋만을 사용하여 fine-tuning을 하였을 때의 최고 EM값은 **81.008** 이었다. 이에 반해 KorQuAD 1.0 데이터를 사용해 학습데이터의 양을 늘린 결과 aug_small의 경우 **81.849**를, aug_large의 경우 **83.649**의 EM값을 달성할 수 있었다. 학습데이터의 양이 모델 성능 향상에 있어 매우 중요한 위치를 차지함을 확인할 수 있었다.

Inference (Retriever + Reader)

- Retriever와 Reader를 연결해 inference를 진행 했을 때 EM 점수가 Reader모델 단독 성능과(EM 점수) 완전히 비례하지 않았다.
- 그래서 val-dataset을 활용하여 앞서 만든 모든 Reader 모델에 대해 Retriever의 'top-k' 인자를 다양하게 적용해가며 최적의 모델을 찾아갔다.

모델이름	# batch	lr	# checkpoint	# wandb-EM	topk	in-EM
roberta	8	1e-05	2100	80.336	top2	0.6366
					top3	0.635
					top4	0.6333
					top5	0.6383
					top6	0.64
					top7	0.63
	8	5e-6	4000	81.008	top1	0.61
					top5	0.6516
					top10	0.6466
					top20	0.6333
roberta	16	1e-05	1400	80.168	top1	0.605
					top2	0.6466
					top3	0.6483
					top4	0.655
					top5	0.66
					top6	0.66
					top7	0.6583
					top10	0.6566
					top20	0.6566

모델이름	# batch	lr	# checkpoint	# wandb-EM	topk	in-EM
crcl-case3	16	1e-05		80.168	top3	0.6416
					top5	0.6433
	8	5e-6			top7	0.64
crcl-case4	16	1e-05		80.84	top5	0.665
					top3	0.6233
					top5	0.6233
	8	5e-6			top7	0.6516
					top5	0.6516
crcl-case5	16	1e-05		80.504	top3	0.6283
					top5	0.6266
					top7	0.6266
	8	5e-6			top5	0.6566
aug_large	16	5e-6	10000	83.697	top5	0.6816
					top3	0.675
					top4	0.6783
					top7	0.675
					top10	0.6766
					top15	0.6583

5.3. Ensemble

soft-voting

- Soft-voting을 이용한 앙상블은 각각의 모델이 주는 결과에 대한 확률들의 (가중)평균을 계산하여 새로운 결과를 얻는 방법이다. 그런데 이번 task에선 한 모델이 낼 수 있는 결과에 대한 경우의 수가 굉장히 다양하기 때문에 조금 다른 방법을 사용했다.
- Inference의 결과로 각 질문에 대한 정답을 얻을 수 있는 `predictions.json` 파일과 함께 각 모델에서 정답으로 예측한 후보들의 상위 20개의 확률값을 포함한 `nbest_predictions.json` 파일이 주어진다. 이를 이용해서 각 모델이 예측한 상위 20개 정답에 대한 확률을 더해서 가장 높은 확률을 갖는 정답을 최종 정답으로 제출하였다.
- 이 경우 각각의 상위 20개의 답이 모든 모델에서 같지 않기 때문에 여러 개의 모델에 대한 정답 확률을 더했을 때 얻을 수 있는 결과의 경우의 수가 20개보다 같거나 크게 된다. 그래서 기존의 soft voting과는 흐름이 조금 다르지만 다양한 모델들의 다른 성향을 고려할 수 있다는 점은 동일하다고 생각되어 위에서 설명한 방식의 앙상블을 진행하였다.

아래 표는 앙상블에 사용한 11가지 모형의 성능과 앙상블 모형의 결과를 정리한 표이다.

	EM
Model 0	0.66
Model 1	0.66
Model 2	0.66
Model 3	0.665
Model 4	0.6816

- 앙상블에 이용한 각 모델에 대한 간략한 설명은 아래와 같다.
 - Model 0~1 : roberta-large
 - Model 2~3 : curriculum learning
 - Model 4~10 : KorQuAD 추가 학습

	EM
Model 5	0.675
Model 6	0.6783
Model 7	0.675
Model 8	0.7033
Model 9	0.6966
Model 10	0.6783
Ensemble Model	0.7233

- 앙상블 결과 가장 성능이 좋았던 Model 8의 0.7033 보다 2% 높아진 0.7233의 EM를 얻을 수 있었다.

6. 자체 평가 의견

6.1. 프로젝트 달성 요소

- 매우 다양한 조건의 하이퍼파라미터 튜닝을 통해 최적의 모델을 생성했다.
- KorQuAD 1.0 data를 이용한 Data-Augment를 통해 모델 성능 개선을 이루어 냈다.
 - Reader 모델 eval/EM : **81.849** → **83.697**
- Curriculum Learning 구현 및 실험을 통해 같은 dataset이어도 학습 방법에 따라 모델 성능에 차이가 나타난다는 것을 확인할 수 있었다.
- Curriculum Learning을 통해 모델 성능 개선을 이루어 냈다.
 - Batch size = 8, lr = 5e-06 eval/EM : **81.008** → **81.008**
 - Batch size = 16, lr = 5e-06 eval/EM : **81.008** → **80.84**
- Elastic Search 사용 시 기존의 검색방법보다 빠르고 정확한 retriever 모델을 만들 수 있었다.
 - 다양한 similarity 알고리즘과 옵션에 대한 하이퍼 파라미터 튜닝으로 데이터에 맞는 최적의 알고리즘을 사용했다.
- 앙상블(soft voting)을 이용해 단일 모형보다 약 2%의 성능 향상을 이루어 냈다.

6.2. 프로젝트 자체 평가

- **잘한 점**
 - 미리 판단하지 않고 다양한 파라미터 구성으로 실험을 최대한 많이 진행했다.
 - Curriculum Learning, Data Augmentation, Split Text 등 다양한 방법을 고민하고 적용해 보았다.

- Curriculum Learning에 대한 논문 스터디를 팀원들과 함께 진행 후 그에 기반하여 정확한 구현을 했다.
- Curriculum Learning 관련 논문에서 제시한 방법만 사용하지 않고, 새로운 방법을 고안하여 사용하였다.
- Elastic Search에 대해 알게됐고, 이를 사용할 수 있도록 구현에 성공하였다.
- **시도 했으나 잘 되지 않았던 것**
 - 외부 dataset으로 먼저 fine-tuning 후 train-dataset으로 이어서 fine-tuning하는 continuous learning을 시도했지만 시간이 부족해서 마무리 짓지 못 했다.
 - Retrieval ensemble에서 query에 대응하는 약 6만개에 대한 score값을 정확하게 불러오지 못해 완전히 구현하지 못했다.
 - Context를 분할하여 sparse embedding을 구성하는 것을 제대로 구현하지 못했다. 분할의 기준을 더 구체적으로 세우고 context의 길이까지 고려한다면 지금보다 좋은 성능을 얻을 수 있다고 생각한다.
- **아쉬웠던 점**
 - Augment 한 dataset으로 Curriculum Learning을 진행해 보았으면 성능 향상을 확인할 수도 있었을 것 같은데 시간이 부족했다.
 - 협업 과정에서 github을 제대로 활용하여 버전관리, PR을 하지 못 했다.
 - 크게 두 파트로 나뉜 이번 task에서 담당하지 않은 부분에 대해 깊게 공부하지 못 하였다.
- **프로젝트를 통해 배운 점 또는 시사점**
 - 같은 Dataset이어도 학습 방법과 조건에 따라 모델의 성능이 다르게 나타났고, Reader 모델의 성능 순위가 Retriever와 결합 했을 때 그대로 유지되지 않았다. 이를 통해 특정 task에서 어떤 Data와 모델이 좋은 성능을 낼 것이라고 결론 지어버리는 것이 위험하다고 느꼈다.
 - 코드 구현 시 생각한대로 실행되는 코드가 작성된 것인지, 관심있는 결과에 대한 수치가 제대로 도출된 것인지 여러 번 확인하며 코드를 작성해야 한다는 것을 느꼈다. 나중에 잘못된 점을 발견하게 되면 이를 수정하는데에는 두 배, 세 배로 시간이 많이 걸려 비효율적으로 작업하게 된다는 것을 경험하였다.

피드백

리포트와 관련한 피드백은 아래와 같습니다.

- MRC의 경우 Task의 난이도가 높아서 large 모델을 사용하는 것이 좋지만, 그럼에도 roberta-base 모델을 이용했을 때의 점수를 비교해주면 좋습니다. 현업에서는 이 부분에 대한 챌린지를 상당히 많이 받습니다. Kobigbird-base와 bert-multilingual-base에 대한 시도는 했음에도 roberta-base에 대한 시도를 하지 않은게 아쉽습니다.

- Kobigbird-base와 bert-multilingual-base 의 결과가 좋지 않았다고 서술했는데, 점수가 어떻게 차이가 났는지 정확한 수치 차이를 기록해야 할 것 같습니다.

- Dense Retrieval을 고려하지 않은 이유가 분명 있을 것이라 생각합니다. 물론 현업에서는 latency, computation, 성능 등의 이슈로 Sparse로 가는 경우가 대부분입니다. 그럼에도 왜 Dense를 쓰지 않았는지에 대해서는 계속 챌린지를 받게 됩니다. (실 서비스에서는 Sparse Retrieval과 Dense Retrieval의 점수를 둘 다 이용하는 경우도 꽤 있습니다.)

- Augmentation에서 validation set을 만들때 “KorQuAD 1.0 train 일부” 약 200개를 사용한 것으로 보이는데, “일부”를 어떠한 기준으로 뽑았는지 적어야 할 것 같습니다.

- 또한 Augmentation에서 validation set은 그대로 유지했으면 어떨까 합니다. train set에 korquad가 포함되어 있기 때문에 validation에도 korquad 데이터가 포함되어 있으면 성능이 더 높게 나올 수 밖에 없습니다. train set에 korquad를 추가해도 validation은 순수한 대회 validation set으로만 가져가는 게 정확한 평가일 것 같습니다.

- Retriever의 경우는 어떠한 오픈소스 라이브러리를 쓰느냐가 상당히 중요합니다. 동일한 알고리즘이더라도 어떠한 라이브러리를 쓰느냐에 따라 구현방식, 추가적인 기능이 다릅니다. elasticsearch 를 사용하였는데 다른 후보 라이브러리도 꼭 조사했으면 합니다.

- korquad 이외의 다른 데이터를 찾아봐도 좋을 것 같습니다. AIHub에 MRC 데이터가 일부 존재합니다. 물론 AIHub 데이터셋의 퀄리티가 좋지 않은 경우가 많지만, 최소한 어떠한 데이터가 있는지 리서치 만이라도 했으면 좋겠습니다.

- EM 은 Tokenizer 결과물의 영향을 많이 받습니다 (e.g. 오늘이 → 오, ##날이). 대회 점수가 f1이 아닌 EM으로 결정되는 만큼 MRC 결과에 대한 후처리 (조사 떼어내기 등) 도 고민했으면 좋을 것 같습니다. 실제 mrc inference output을 본다면 느낌이 다를 것입니다.

대회 성적과 리포트 퀄리티 모두 굉장히 좋아져서 멘토로서 뿌듯합니다. (지난 Relation Extraction과 비교했을 때 굉장히 많이 성장했음을 느낍니다) 솔직히 제가 아쉬운 점을 쓰긴 했지만, 부스트캠프 일정과 과제양을 고려했을 때 지금 결과로도 이미 완벽합니다. 실제 현업에서 프로젝트를 해보면 시간적인 압박, 여러 자원 (인적, 물적)의 한계로 결과물이 완벽할 수 없습니다. 결국 가장 최적, 최선의 결과를 뽑아내는 것이 중요합니다!

여러 가지 우여곡절이 있었음에도 대회를 잘 마무리해서 감사하다는 말을 꼭 하고 싶습니다. 앞으로 살면서 모든 일이 항상 계획된 대로 되지만은 않을 것입니다. 분명 그 순간에는 그 장애물들이 너무 밍고 싫겠지만, 시간이 지나고 보면 그 경험을 미리 해본 것에 감사하게 될 것입니다. 남

은 부스트캠프 기간동안, 그리고 그 이후에도 여러 어려움을 이겨내고 성장하는 캠퍼들이 되었으면 합니다. 항상 응원하겠습니다!