MRC NLP Wrap up Report (6조)

1. 프로젝트 개요

1.1. MRC Task

MRC task에서는 모델의 ODQA(Open-Domain Question Answering) 의 수행 능력을 평가하여 모델의 기계독해 성능을 측정한다.

ODQA 모델은 two-stage로 구성된다. 첫 번째 단계는 입력된 질문에 대해 관련된 문서를 찾아주는 retriever model이고, 두 번째 단계인 reader model에서는 retriever model이 전달한 context를 이용해 입력된 query에 대한 정답을 찾게 된다.

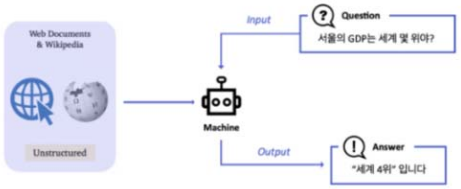

Figure 1. MRC Overview

1.2. 팀 구성

이름	역할
박승현	PM, Reader, Model Tuning, 결과 분석
류재환	코드 리뷰어, Elastic Search
김준휘	DPR, Retriever
박수현	EDA, 전처리, Reader
설유민	DPR, Retriever, Model Tuning

Table 1. 팀 역할 분배

1.3. Timeline

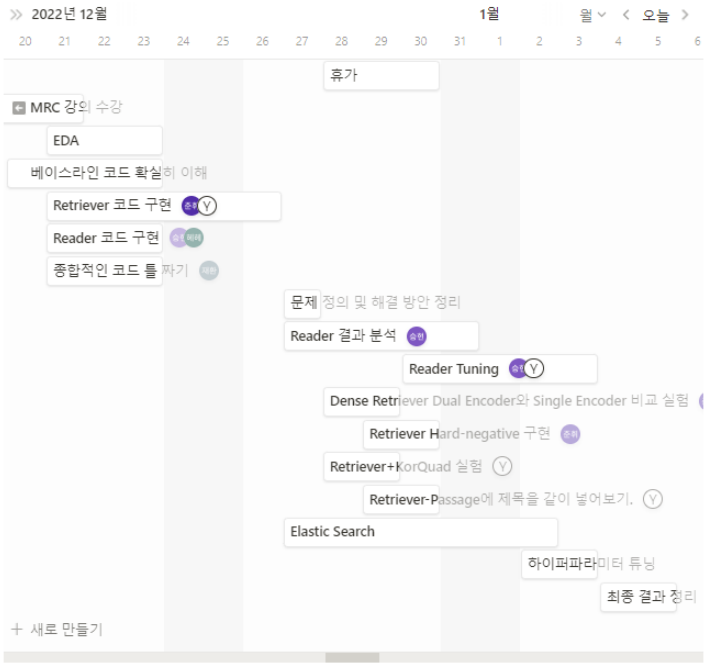

Figure 2. Timeline

2. EDA

대회에서 제공된 dataset에는 train dataset과 test dataset으로 이뤄져있다. train dataset 중에 train은 3962개 validation은 240로 구성되어 있고, test dataset에는 600개의 데이터가 포함되어 있다. 또한, 대회에서 추가로 사용한 Korquad v-1.0 같은 경우에는 제공되는 train dataset과 validation dataset을 제공해주며, train은 60,407개, validation은 5,774개로 구성되어 있다.

대회 데이터와 Korquad 데이터에는 결측치와 pair duplication은 존재하지 않았으나, 대회 data에서는 동일 context에 대해 다른 question이 제시된 사례가 612개 존재했다.

2.1. Length of Passage & Question

일반 pre-trained 모델의 경우, input token의 길이가 512로 제한되기에 passage와 question의 길이와 klue/roberta tokenizer를 사용한 tokenized 길이를 분석해봤다.

	Median	Min	Max
Passage Length	815	512	2,046
Question Length	29	8	78

Table 2. Length of Passage & Question

	Median	Min	Max
Passage Length	429	234	1,135
Question Length	5	5	43

Table 3. Length of Tokenized Passage & Question

2.2. Length of Answer

max answer length를 설정하기 위해서 answer의 길이를 분석하는 것도 매우 중요하기에 answer length에 대해 분석해봤으며, 아래는 대회 dataset에 대한 분석이다.

	Mean	Min	Max
Train	6	1	83
Validation	6	1	64

Table 4. Length of Answer (Our Dataset)

	Mean	Min	Max
Train	5	1	84
Validation	5	1	43

Table 5. Length of Answer (Korquad v-1.0)

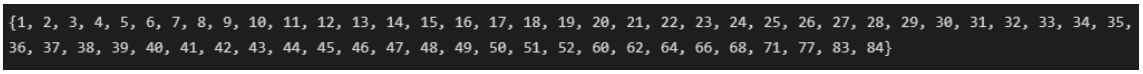

Figure 3. Length of Answer (Total)

실제로 확인 결과 answer length가 50 이상인 데이터가 많지 않았으며, 특별한 질문이 아닌 이상 대부분 답변의 길이는 길지 않은 것으로 확인됐다. max answer length를 너무 길게 설정할 경우, 모델이 학습을 제대로 할 수 없을 수 있기에 중간값보다 조금 큰 50으로 max answer length를 설정하기로 했다.

3. Retrieval Model

3.1. 모델 평가

Retriever 모델의 평가는 top-k 정확도와 validation loss로 수행했다. Top-k 정확도는 모델이 추출한 문서가 라벨 문서와 같은지를 판단한다.

3.2. Dense Retriever & Sparse Retriever

Wiki corpus에서 passage를 가져오는 방식이 여러 가지 존재하지만, 이번 대회에서는 dense retriever(DPR)과 sparse retriever(BM25 & Elastic Search)를 적용했다.

3.3. Dual Encoder vs Single Encoder (DPR)

DPR은 기본적으로 Dual Encoder를 사용해 passage와 question을 각각 처리하지만 연산량이 많이 필요하고, 배치 크기도 키울 수 없다. 이보다는 연산량이 가벼운 Single Encoder를 사용하고 배치 크기를 키우는 것이 모델 성능 향상에 더 도움이 될 수 있다고 생각했기에 아래와 같은 실험을 진행했다.

	top-5	top-20	top-100
Dual-Encoder (batch size = 8)	34.27%	55.64%	73.38%
Single-Encoder (batch size = 8)	**43.95% (+9.7%)**	**58.06% (+3.6%)**	**75.80% (+1.5%)**
Single-Encoder (batch size = 64)	39.11% (+5%)	56.04% (+1%)	72.98% (-1%)

Table 6. Single Encoder와 Dual Encoder의 정확도 비교

실험 결과 Dual Encoder보다 Single Encoder의 성능이 오히려 더 좋았으며, 배치 크기도 64까지 키울 수 있어 훨씬 빠른 학습이 가능했다.

3.4. Wiki Corpus Stride (DPR)

Wiki 문서들은 대부분 길이가 매우 길기 때문에 모델에 한 번에 넣을 수가 없다. 따라서 문서를 자르게 되는데, 자르는 길이와 겹치는 영역의 길이를 조절할 필요가 있다. 이는 huggingface의 tokenizer를 사용해서 쉽게 할 수 있고 조절 가능한 부분은 max_length와 stride가 있다.

최대 길이는 512로 설정했다. 여기서 질문과 passage를 합쳐서 Reader 모델에 input으로 제공할 것을 생각해 384로 잘랐으며, 어떻게 문장을 나누느냐에 따라서 성능이 달라질 것을 고려해 다양한 stride에 대한 실험을 진행했다(with DPR).

	top5-accuracy	top20-accuracy	top100-accuracy
stride = 32	32.23%	53.71%	73.14%
stride = 64	**34.14%**	53.25%	**73.57%**
stride = 128	31.77%	**54.51%**	72.56%
stride = 256	31.00%	49.72%	71.50%

Table 7. Stride Experiment

실험 결과, stride에 따라 성능 차이가 크지 않은 것으로 판단되어 실험 결과에서 조금 더 좋아보이는 stride = 64를 선정했다.

3.5. Pretraining Wiki Corpus with Similarity Score (DPR)

Wiki corpus의 수는 60,000개지만 주어진 데이터셋이 4,000개밖에 되지 않는다. 이를 이용해 훈련을 한다면 질문과 관련 없는 56,000개의 위키 문서들은 모델이 학습하지 못하게 되는 문제가 있다. 그렇기 때문에 질문이 없는 wiki corpus들만을 활용해 비지도 학습을 먼저 하는 방법을 시도해 보고 싶었다.

Wiki corpus 간의 유사도를 측정하기 위해 BM25와 BLEU score를 사용해 봤지만 6만 개의 wiki corpus를 모두 계산하기에는 너무 많은 시간이 소모된다. 따라서 gpu를 사용해 빠르게 유사도를 측정해낼 수 있는 pretrain된 STS 모델을 사용해 유사도를 측정하고 positive 쌍들을 뽑아내 보기로 하였다. STS 모델을 활용해 각 wiki corpus마다 가장 유사한 positive corpus들을 하나씩 뽑아내고, in-batch-negative 학습을 통해 pre-training을 진행해 보기로 했다.

	without pre-training	with pre-training
top-5	**42.74%**	41.93%
top-20	**54.83%**	52.82%
top-100	**71.77%**	67.74%

Table 8. Pre-training top-k Accuracy

3.6. In-batch-negative (DPR)

Dense Retriever의 학습을 위해 질문과 관련 있는 positive passage와 관련 없는 negative passage를 구성할 필요가 있다. 그러나 직접 negative passage를 구성하는 것은 자원이 많이 들고 한 번에 많이 학습할 수 없다. DPR 논문의 in-batch-negative는 이런 문제를 해결하기 위해 같은 배치의 다른 positive 쌍들을 negative로 보는 방식으로 연산의 효율성을 크게 높인다.

	top-5	top-20	top-100
without-in-batch-negative	33.06%	46.77%	66.53%
with-in-batch-negative	**41.12%**	**53.62%**	**69.75%**

Table 9. in-batch-negative 사용 유무에 따른 top-k 정확도

In-batch-negative를 사용한 것이 정확도가 압도적으로 더 좋을 뿐만 아니라, 연산의 효율성에도 큰 차이가 있었다. In-batch-negative를 사용하지 않고 질문마다 negative passage를 2개씩 구성했을 때는 배치 크기가 8이 한계였다. 그러나 in-batch-negative를 사용하면 더 높은 배치 크기(64)도 활용이 가능하다.

3.7. 외부 데이터셋 사용 (DPR)

In-batch negative 방법을 유지하면서, 상대적으로 규모가 작은 우리 데이터에 같은 task의 다른 데이터셋을 합쳐 학습에 활용하여 모델이 retrieval 학습 자체에 사용하는 데이터를 늘려주어 성능 향상을 이끌어내고자 했다.

외부 데이터셋으로는 KorQuAD 1.0를 사용했다. KorQuAD는 우리 데이터와 동일하게 위키백과의 문서로 knowledge base를 구성했고, 질문 및 답변 생성 과정 역시 유사해 우리 데이터의 경향을 유지한 채로 학습 데이터 부족으로 발생하는 성능 정체를 해소할 수 있을 것이라 판단했다. Validation 과정에서는 우리 데이터셋의 validation set만을 사용했다. PLM은 klue/bert-base를 사용했다.

	top-5	top-20	top-100
대회 데이터	43.50%	63.41%	87.80%
대회 데이터 + KorQuAD	49.59% (+6.09%)	69.11% (+5.7%p)	86.59% (-1.21%p)
KorQuAD 후 대회 데이터	**57.32% (+7.73%p)**	**72.36% (+3.25%p)**	**88.21% (+1.62%p)**

Table 10. 외부 데이터 적용 후 정확도 분석

top-5, top-20 정확도가 우리 데이터만을 사용했을 때보다 KorQuAD를 함께 사용했을 때 더 높았다. 우리 knowledge base의 문서를 직접적으로 학습에 더 활용하지 않더라도, multitask learning이 모델의 retrieval 성능 향상에 도움을 줌을 확인할 수 있었다. Top-100 정확도의 경우 단일 데이터를 사용했을 때보다 오히려 감소했지만, reader에게 너무 많은 후보 passage를 전달할 경우 오히려 정답을 찾지 못하는 경우가 발생할 것을 고려하면, multitask learning이 실질적인 성능 향상을 이끌었다고 할 수 있다.

KorQuAD를 함께 사용하여 fine-tuning을 한 번만 진행했을 때보다, KorQuAD로 먼저 학습 후 우리 데이터로 학습했을 때의 top-k 정확도가 더 높았다. PLM을 KorQuAD로 먼저 QA task에 맞게 fine-tuning하고, 우리 데이터에 맞게 재조정한 효과가 있음을 확인했다.

3.8. Hard Negative (DPR)

모델이 문장의 의미보다 단어가 얼마나 겹치냐에 의존하여 결과를 내는 것을 방지하기 위해 단어가 크게 겹치지만 정답이 아닌 passage를 negative 쌍으로 모델에 의도적으로 보여준다.

DPR 논문을 따라 데이터셋의 모든 질문들에 대해 bm25를 활용해 단어가 가장 많이 겹치지만 정답 passage가 아닌 passage들을 hard negative로 구성했다.

	top-5	top-20	top-100
without hard-negative	**50.83%**	**69.17%**	**81.25%**
hard-negative (1)	45.56%	62.09%	79.43%

Table 11. Hard Negative 적용 유무 차이

KorQuad까지 Hard negative를 적용해서 실험해본 결과는 오히려 hard-negative를 사용하지 않은 모델이 정확도가 더 높았다.

3.9. Elastic Search

성능이 우수한 오픈소스 검색 엔진인 Elasticsearch를 사용하여 Retriever를 구현했다.

처음에 Elasticsearch를 적용하였을 때, 기존에 사용하던 BM25 메커니즘에 비해 성능이 떨어지는 결과를 확인하였다.

	Public EM	Public F1
기존의 BM25	**44.17**	**59.17**
Elasitcsearch를 통한 BM25	30.42	41.86

Table 12. BM25 오픈소스 비교

Default로 Elasticsearch 역시 똑같은 BM25 메커니즘을 사용하는데 성능이 크게 차이가 나는것에 의문을 가지고 원인을 분석해 보았다. 같은 메커니즘 속에서 차이가 날 만한 부분을 탐색해본 결과, BM25에 사용하는 토크나이저에 차이가 있음을 발견하였고 추가적인 검색 결과 Elasticsearch에서 사용할 수 있는 한국어 전용 토크나이저인 nori tokenizer를 발견하였다. 이를 사용하여 다시 결과를 얻었을 때 기존보다 성능이 많이 오른 것을 확인했다.

이어서 Elasticsearch에서 기본으로 설정된 BM25 메커니즘 말고도 다양한 다른 검색 메커니즘을 실험해 보았다.

	top10 EM	top10 F1	top20 EM	top20 F1	top30 EM	top30 F1
BM25	**58.75**	**67.06**	61.25	**69.56**	61.25	69.76
DFI	56.25	63.56	59.17	66.20	59.58	66.02
DFR	**58.75**	66.38	**61.67**	69.09	**62.50**	**70.13**
IB	56.25	64.04	60.00	68.00	60.41	68.21
LMDirichlet	58.33	65.46	59.58	67.23	61.25	68.70
LMJelinekMercer	55.00	62.93	57.08	65.18	58.33	66.22

Table 13. Elastic Search 알고리즘 별 성능 차이

결과적으로 BM25와 DFR 두 가지 방법의 성능이 가장 뛰어났고, 두 가지 방법을 바탕으로 Public에 제출한 결과 public EM score가 각각 66.6, 67.5로 기존보다 3점 가량의 성능 향상이 있었다.

3.10. Rerank with STS

관련 자료[1]를 토대로 bm25의 결과와 sts pretrained 모델을 사용해 rerank하는 방식을 실험해 봤으나 오히려 정답을 담은 passage의 rank가 하락했다.

3.11. 최종 결과

아래의 top-k 정확도는 모델이 추린 문서에 정답 단어를 포함 여부로 측정했다. 최종 실험은 Retriever 모델만이 아니라 Reader 모델의 결과까지 볼 것이기 때문에 Reader가 정답을 추출할 수 있는 환경을 마련해 줬는지를 판별하기 위해서이다.

DPR 모델은 KorQuad→Trainset으로 훈련된 모델이며, Hard-negative는 한 개를 추가한 모델이다.

	BM25	DPR	DPR(Hard-negative)
top-20	**85.41%**	76.66%	70.83%

Table 14. Dev셋에 대한 top-20 정확도

결과적으로 BM25의 정확도를 넘어서지는 못했다. BM25의 정확도가 확실히 높았고, 그 다음 DPR 모델, 그 다음 Hard negative를 적용한 DPR 모델 순이었다. 그러나 DPR 모델은 학습 가능한 Dense Retriever로써, 오로지 단어의 유사도만을 비교하는 BM25가 예측하지 못한 문서를 찾을 수 있다고 판단했다. 따라서 BM25의 결과와 DPR 결과를 섞어서 Reader에 제공하기로 했다.

Retriever	Accuracy
BM25(top30)	86.25%
BM25(top20) + DPR(top10)	92.50%
BM25(top20) + DPR(hard-negative)(top10)	91.66%
BM25(top15) + DPR(top15)	**92.91%**
BM25(top15) + DPR(hard-negative)(top15)	91.66%

Table 15. Dev셋에 대해 Retriever의 결과를 받아 Reader로 낸 EM과 F1 점수

	EM	F1
BM25(top30)	63.89	77.21
BM25(top15) + DPR(top15)	61.67	74.53

Table 16. Test셋에 대해 Retriever의 결과를 받아 Reader로 낸 EM과 F1 점수

실제로 그 결과를 Reader 모델에 전달하여 Dev셋에 대해 점수를 냈을 때도, BM25만 사용한 것보다 DPR 결과를 같이 전달했을 때 EM 점수는 0.8점, f1 점수는 0.2점 정도 높게 나타났다.

그러나 Test셋에서는 EM 점수가 2점 감소하고, f1 점수가 3점 감소했다. Test셋이 Dev셋보다 단어가 겹치는 것이 더 중요했을 지도 모르겠다. Dev셋과 Test셋의 경향성이 좀 달랐던 것은 확실한 듯하다.

4. Reader Model

Reader Model의 모든 성능 평가는 dev set 기준으로 진행했다. KorQuAD를 학습시킬 때는 KorQuAD validation set, 대회 데이터를 학습시킬 때는 대회 validation set을 활용했다.

4.1. Pre-trained Model 실험

보다 해당 task에 적합한 모델을 찾기 위해 다양한 모델에 대해 통일된 hyperparameter로 실험했다.

Transformer Model	EM	F1	비고(default mx_token_length=512)
klue/bert-base	85.192	90.658	
klue/roberta-large	87.807	92.348	
klue/roberta-base	87.427	92.093	
klue/roberta-small	84.344	89.729	
monologg/koelectra-v3	85.937	90.886	
beomi/kcelectra-base	69.449	76.572	
monologg/kobigbird-bert-base	87.236	91.818	
monologg/kobigbird-bert-base	86.734	91.666	mx_token_length=2048
jinmang2/kpfbert	87.201	92.141	
monologg/kobert	4.295	6.655	
xlm-roberta-large	77.797	84.064	
xlm-roberta-base	75.234	81.954	
beomi/kcbert-large	64.254	72.509	mx_token_length=300(due to model setting)
beomi/kcbert-base	60.669	69.331	mx_token_length=300(due to model setting)
monologg/kocharelectra-base	7.724	10.497	
kykim-electra-kor-base	62.695	71.53	

Table 17. 다양한 Transformer 모델 실험

EM score 기준으로 사용 가능할 만한 모델은 klue/roberta-large, klue/roberta-base, jinmang2/kpfbert, monologg/kobigbird-bert-base 총 4가지이지만, 이 중 kobigbird의 2048 token length 같은 경우에 학습시간이 과도하게 많아 효율성 문제로 인해 사용하지 않기로 했다.

4.2. BM25 vs DPR

앞선 Retriever 실험에서 DPR의 성능이 BM25보다 떨어졌지만, 실제로 더 좋은 passage를 제공했을 가능성도 존재한다. 이에 따라 baseline code 설정으로 두 가지 결과를 비교해보기로 했다.

	top5 EM	top5 F1	top20 EM	top20 F1	top30 EM	top30 F1
BM25	47.5	60.17	44.58	56.75	42.08	54.15
DPR	35.0	50.26	39.17	51.88	37.92	50.62

Table 18. BM25 VS DPR

Retriever 실험에서와 같이 Reader에서도 DPR이 제공한 결과가 좋지 못한 성능을 보였다.

4.3. Best top-k

Top-k passage의 k도 일종의 하이퍼 파라미터이기에 실험이 필요하다. 우선, k의 중요성을 파악하기 위해 validation set을 기준으로 답안이 몇 번째 k에 위치하고 있는지를 확인해봤다. 정답이 top-20 내에 있을 확률은 50% 이상인 것으로 확인됐고, k가 증가함에 따라 성능도 자연스럽게 올라가는 것을 확인할 수 있었으며, 적정 k에 대한 실험이 필요했다.

	EM	F1
top-5	53.75	60.92
top-10	55.41	62.90
top-20	58.75	66.23
top-30	59.16	66.79
top-40	59.16	66.76
top-50	**60.00**	**67.11**
top-60	59.16	66.34

Table 19. Top-k Analysis

k 값이 커지면 커질수록 성능이 좋아졌지만, k=50 이상인 경우에는 성능이 점차 하락하는 모습을 보였다. 그 이유는 후 순위 passage에서 추출된 답안이 높은 score를 기록해 최종 답안 도출에 영향을 줬기 때문이라고 생각할 수 있다.

4.4. Retrain

우선, 대회 측에서 제공해준 train dataset은 3,952개 밖에 존재하지 않기 때문에 충분한 학습을 진행할 수 없다고 판단했다. 따라서 약 2만여개의 데이터를 가지고 있는 KorQuAD 데이터를 우선 사용하기로 했다. KorQuAD 데이터로만 train하는 경우에도 Reader가 자신의 역할을 충분히 해낼 것으로 예측된다.

대회 train data와 KorQuAD 조합 및 학습 방식에 따라 성능에 차이점이 발생할 것 같아 여러 설정으로 실험을 진행해봤다. 실험의 효율성을 높이기 위해 base 모델을 우선 사용하고 가장 좋은 방식으로 large 모델에 적용했다.

모델	epoch (KorQuAD)	epoch (대회 data)	EM	F1
roberta-base	2	1	65.41	73.76
roberta-base	2	2	64.16	72.97
roberta-base	2	3	65.83	74.10
kpfbert	2	1	64.22	73.12
kpfbert	2	2	67.49	76.15
kpfbert	2	3	**72.08**	**79.42**
kpfbert	2	4	68.46	78.64
kobigbird	2	3	67.08	76.97
roberta-large	2	3	65.83	75.89
kpfbert	1	2	68.33	76.96
kpfbert	1	3	68.33	77.10
roberta-large	1	3	73.75	80.74
roberta-large	1	4	**74.16**	**81.64**

Table 20. Retrain Option Experiment

실험 결과, KorQuAD와 대회 dataset에 대한 학습을 진행할 때 epoch에 따라 성능 차이를 확연하게 보이는 것을 확인할 수 있다. 그러나 모델에 따라 성능이 천차만별이고 어떤 방식이 좋은지는 명확하지 않은 것 같다. 그러나 대회 데이터를 3~4 epoch 정도 학습시킨

모델이 더 좋은 성능을 도출하는 것 같다.

4.5. Merge All Data

앞선 실험(4.4)에서는 KorQuAD를 학습시킨 이후 대회 데이터를 학습시키는 방식을 취했다. 만약 이 데이터를 병합해 학습시킨다면 더 좋은 성능을 도출할 수 있지 않을까하는 의구심이 들어 실험을 진행했다.

모델	설명	EM	F1
kpfbert	실험(4.4)에서 가장 좋은 모델	**72.08**	**79.42**
kpfbert	병합한 데이터를 2epoch 학습시킨 모델	63.75	73.64
kpfbert	병합한 데이터를 3epoch 학습시킨 모델	64.11	74.53

Table 21. Merged Data Experiment

데이터를 병합해 학습시키면 표현력을 확대할 수 있을 것이라는 기대감과 달리, 실제 결과는 하락했다. 그 이유는 KorQuAD 데이터의 question에는 passage에 사용된 단어가 들어있는 경우가 많기에 쉬운 데이터로 분류될 수 있고, 이에 반해 KLUE/MRC 데이터는 최대한 passage 내 단어와 겹치게 하지 않기 위해 노력을 기울였기에 어려운 데이터로 볼 수 있다. [2]논문에 따르면, 쉬운 데이터를 학습한 이후 어려운 데이터를 학습하면 더 좋은 성능을 도출해 낼 수 있다고 한다. 결과도 증명했듯이 KorQuAD와 대회 데이터의 난이도가 다르기에 KorQuAD를 먼저 학습시키는 것이 더 좋은 학습 방법이라고 볼 수 있다.

4.6. Passage Similarity

Reader의 성능을 명확하게 파악하기 위해서는 최종 answer만을 확인해 볼 것이 아닌 n best answer도 확인해봐야 한다. 비록 정답으로 선정되지 않았더라도 n best에 있다면 후처리를 통해 정답을 조율해볼 수 있지 않을까 하는 아이디어가 떠올랐다. 또한, n best answer에 대한 분석을 진행해 본 결과 정답을 포함한 passge의 위치는 앞 순위에 있는 반면 답안은 후 순위에 있는 경우가 어느 정도 존재했을뿐더러 4.3 실험에서 확인했듯이 후 순위의 passage에서 추출한 답변이 최종 답안 도출에 큰 영향을 주는 경우도 꽤 존재했기에 passage 순서에 따른 가중치 부여도 고려해봐야 할 것 같았다.

Retriever의 성능이 좋을 것이라는 가정 하에 passage score도 answer score에 반영해준다면 더 좋은 성능을 도출할 수 있지 않을까 하는 생각이 들었다. BM25와 Elastic Search의 경우 top-20~의 정확도가 90%로 높기에 passage score도 적용해 보면 좋을 것 같았다.

	EM	F1
without passage similarity	**58.31**	**63.48**
with passage similarity	57.10	61.94

Table 22. Experiment of Passage Similarity

위 실험 결과에는 passage similarity를 반영한 방식이 좋았으나, 실제 여러 차례 실험 결과 reader의 n best, max answer length, model type 등 여러 요소에 따라 실험 결과가 변동됐다. 그럼에도 불구하고 passage similarity를 적용한 이유는 성능이 좋은 reader의 결과값으로 answer score를 보완해주기 위함이다.

4.7. N-Best Answers

n개의 후보 답들 중에서 실제 답이 더 많이 등장했음에도 최종 답으로 채택되지 않아 문제를 틀리는 경우를 관찰했다. n개의 후보 답들 중 가장 첫번째 것을 답으로 바로 출력하지

않고, 가장 많이 등장한 후보 답을 최종 답으로 채택했을 때 실제로 답을 더 많이 맞힐 수 있는지 확인해보고자 했다.

Retriever로는 BM25, reader로는 klue/roberta-large, 후보 답으로는 20개를 선별한 validation 결과를 관찰했다.

n-best answers에서 가장 많이 나오는 답이 실제 정답인 경우는 240개 중 145개(60.47%)였다. 그러나 같은 빈도로 등장하는 다른 답들이 존재할 수 있어 확인한 결과, 답일 확률이 가장 높으면서 가장 많이 등장하는 답이 실제 정답인 경우는 120개(50%)였다. 반면, 현재 reader 모델이 정답이라고 판단한 답이 실제로 가장 많이 나오는 답인 경우는 182개(75.83%)로 오히려 실제 정답이 가장 많이 등장한 답인 경우보다 많았다. n-best answers 내 등장 빈도에 상관없이 답일 확률이 가장 높은 한 개의 답을 도출했을 때는 132개(55%)의 답을 맞혔다.

N-best answers에서 실제 정답이 더 많이 등장했지만, 등장 빈도가 더 적은 답을 선택함으로써 답을 틀린 경우는 14(5.83%)였다. 반면, n-best answers에서 실제 정답보다 고른 답이 더 많이 등장하는 경우는 63개(26.25%)였다. N-best answers에서 가장 많이 등장하는 답을 최종 답으로 출력했을 때 성능 개선을 기대하기 어려워 보였다.

	EM	F1
기존	**57.08**	**64.34**
가장 많이 등장하는 답 선택	52.08	60.95

Table 23. 답 선택 방식별 EM, F1

실제로 validation에서 기존 방식처럼 답일 확률이 가장 높게 책정된 답을 최종 답으로 채택하는 것이 더 좋은 성능을 보였다.

4.8. 전처리

dataset의 context는 한국어 위키백과가 출처이므로, 전처리 중 줄 바꿈을 나타내는 '\n' '\\n'이 처리되지 않았음을 확인하고, context내 해당 문자를 처리하고 answer_start index를 수정했다.

ODQA의 test set에서는 context 없이 질문만 주어지므로, wikipedia corpus에서 retriever가 적절한 context를 찾아 reader모델에 넘기는 과정이 필요하다. 이때 사용되는 wikipedia corpus도 마찬가지로 전처리가 되어 있지 않았기 때문에 전처리를 수행했다.

Wiki corpus 내에서 처리한 부분은 다음과 같다.

1) URL 및 </html> tag
2) 특수문자, 줄 바꿈(\n, \\n)

	EM	F1
without 전처리	60.42	72.77
with 전처리	64.58	74.77

Table 24. 전처리 적용 관련 실험

문장 학습에 필요 없는 부분을 지움으로써 모델이 문장 구조를 더 명확하게 파악하게 할 수 있었던 것 같다. 모델이 문장의 의미를 이해하기도 하지만 실제 결과 값과 모델의 경향성을 토대로 분석해보면, 모델은 문장에 대한 구조를 더 잘 학습한다. 따라서, 전처리를 통해 문장 구조를 통일함으로써 모델이 데이터 패턴을 명확하게 이해할 수 있다. 또한, 노이즈 데이터를 정제해줌으로써 학습이 올바른 방향으로 가도록 유도했을 수 있다.

4.9. 최종 결과 및 분석

이번 대회에서 최종적으로 적용한 방법은 아래와 같다.

1) Elastic Search(BM25 + DFR) retriever & Elastic Search(BM25) + DPR retriever
2) wiki 전처리
3) roberta-large reader
4) passage similarity
5) KorQuAD retrain

EM (Rank)	F1	개요	생성 시간	최종 제출
68.3300	79.3600	상세 보기	2023-01-04 18:07	
65.4200	75.0100	상세 보기	2023-01-05 17:26	

Figure 4. Final Score

최종 실험 결과를 통해 확인할 수 있듯이, 최종 실험의 F1 score가 높다는 것을 확인 가능하다. EM에 비해 F1이 현저히 높은 이유는 정답이 아니지만 정답과 매우 유사한 답이 많다는 것을 의미한다. n best answer와 ground truth를 비교 분석해 본 결과 모델이 가장 헷갈려하는 부분은 연도이다. ground truth가 연도인 경우 “1938”, “1938년”, “1942년” 등 다양한 답변이 n best에 존재하기도 한다. 이것을 통해 대략적으로 추측할 수 있는 점은 모델이 의미를 이해하기도 하지만 정확하게는 구조 혹은 패턴을 학습한다는 점이다.

마지막으로, 실제 private 결과에 따르면 roberta-large의 성능은 당연히 우수하지만, kpfbert의 성능도 상당히 우수하다는 것을 확인했다.

	EM	F1
roberta-large (best)	63.89	77.18
kpfbert (best)	62.22	73.89

Table 25. roberta-large VS kpfbert

5. 아쉬운 부분

1) 이번 대회의 목적은 차근차근 모든 부분을 고려하고 다양한 분석을 진행하는 것이 목표였기에 오직 성능만을 추구하려는 앙상블을 진행하지는 않았다. 그러나, 앙상블을 진행했다면 분명 더 좋은 성능을 도출했을 것이라 예상된다.
2) Reader와 Retriever 역할이 따로따로 분리되어 있어서 한 명이 필수적으로 실험을 진행해줘야만 했고, 그로 인해 generation based model을 구현하기 위한 시간이 부족했다.
3) n best, max answer length 등 일부 파라미터에 대한 고찰이 부족했다. 특히 max answer length 같은 경우, 너무 긴 답변은 확실히 틀리고 짧은 답변은 확실히 맞히는 방식의 실험을 진행했다면 더 좋은 성능을 낼 수 있었을 것이다.
4) 원래 해보고 싶은 방법론이 정말 많았다. generation, 영어 데이터 사용, pre-train 등이 있었지만, 대부분 공부 혹은 초기 단계까지만 진행해서 최종 결과를 도출해내지 못했다.

6. Reference

[1] https://github.com/jaeyeongs/BM25-KoSBERT
[2] Curriculum Learning for Natural Language Understanding (Xu et al., ACL 2020)

김준휘 개인 회고

1. 대회 목표

ODQA task에 대한 파악이 끝난 후 가장 하고 싶었던 일은 좋은 성능의 Retriever를 만들고 싶다는 것이었다. 앞선 STS 대회에서 수행했던 Contrastive learning을 여기서 활용할 수 있을 것이란 생각이 들었기 때문이다. 대회에서 주어진 데이터는 3000개였고, 이를 활용해서 질문과 문서 쌍을 학습한다면 Retriever 모델은 전체 60,000개의 문서 중 3,000개 밖에 학습하지 못하는 것이라 생각이 들었다.

2. 수행한 작업

2.1. Dense Retriever 코드 작성

우리 팀은 우선 Retriever와 Reader로 나눠 작업을 수행하기로 결정했다. 나는 팀원 1명과 Retriever를 맡았다. 제공된 베이스라인 코드에서 Retriever 코드는 제공되지 않았기 때문에 직접 구성해야 했다. 강의에서 제공해준 노트북 실습 코드를 바탕으로 처음부터 코드 틀을 잡았다. 코드 작성은 팀원 1명과 git flow를 사용하여 수행되었다.

이번 대회에 맞춰 시작한 논문 스터디 모임에서 ORQA와 DPR을 공부했는데, ORQA는 매우 큰 배치 크기가 필요한 pre-training 과정이 필요하기 때문에 사용이 어려울 것이라 판단하고, DPR 논문을 기반으로 코드 구성을 하였다.

2.2. 데이터셋

너무나도 긴 wiki corpus들의 길이를 적절하게 자를 필요가 있어 초기에 진행했던 실험이다. 자른 결과를 직접 봐도 뭐가 좋은지 알 수 없기 때문에 하이퍼 파라미터처럼 직접 조정했다. 최대 토큰 길이는 질문 길이를 고려하여 384로 지정했고, stride(겹치게 자를 영역)는 실험해본 결과 64가 제일 성능이 괜찮았다.

이번 대회는 데이터셋의 추가가 가능했기 때문에 wikipedia를 사용해 만들어진 korquad v1.0 데이터를 추가로 사용했다. 이 데이터는 1500여 개의 wiki 문서에서 추출한 1만 여개의 문단과 관련한 6만개의 질의응답 쌍을 갖고 있다. 실제로 실험한 결과 korquad를 추가로 사용하는 것은 성능 향상에 큰 영향을 미쳤다.

2.3. 모델 평가

사실 모델 평가가 어려운 부분 중에 하나였다. 결과로 top-k개의 문서를 내놓는데 이 문서들이 질문과 관련이 있는지를 일일이 보기도 어려웠고, 문서 양도 너무 많았기 때문이다. 앞선 대회들처럼 특별한 라벨이 존재하는 것도 아니여서 어떤 통계를 내기도 어려웠다. 결국 모델을 평가하는 데는 validation loss와 top-k accuracy만 사용되었다.

2.4. In batch negative

DPR의 핵심이라고 볼 수 있는 것 중 하나인 In batch negative의 효과를 실험해 보았다. 실험 결과 in-batch negative를 사용했을 때 top-5 정확도가 8%가 더 높았다. 그리고 in-batch negative를 사용하지 않을 때는 배치 크기를 8까지 밖에 사용할 수 없었지만, in-batch negative를 사용해 64까지도 키울 수 있게 되었다.

2.5. Dual Encoder와 Single Encoder

원래 DPR에서는 passage와 question에 Dual Encoder를 사용하여 학습하지만 연산량이 매우 크고, 배치 크기도 키울 수 없어 Single Encoder와 성능을 비교해보고 웬만하면 Single Encoder를 사용해 보고자 했다. 놀랍게도 실제로 성능을 비교했을 때 Single Encoder가 더 좋은 성능이 나왔고, 배치 크기도 훨씬 키울 수 있었기 때문에 이 뒤의 실험은 모두 Single Encoder로 진행되었다.

2.6. BM25를 활용한 Hard Negative

DPR 논문의 핵심2라고 볼 수 있는 Hard Negative도 실험해 보았다. BM25를 활용해 각 질문 별로 bm25 점수가 높지만 정답 passage가 아닌 것을 Hard negative로 사용했다. KLUE 데이터만으로 학습했을 때의 결과에선 top-5 정확도는 2% 향상되었지만 top-10 정확도는 4% 감소되었다. Korquad까지 hard negative를 적용해서 학습했을 땐 top-5에서 4%, top-10에서 7%나 정확도가 감소했다. Dev셋에서 bm25의 점수가 높은 것을 봤을 때, Dev셋의 질문과 정답 passage가 단어적으로 많이 겹치기 때문에 이런 결과가 나타났다고 판단했다.

3. 그 외

1) https://github.com/jaeyeongs/BM25-KoSBERT를 참고하여 STS 모델을 활용한 bm25 결과를 reranking하는 것도 시도해 봤지만 성능이 안 좋았다.
2) STS 모델을 활용해 질문 없이 wiki corpus만으로 positive 쌍을 구해서 in batch negative로 pre-training을 해봤지만 성능이 떨어졌다.

4. 소감

Task의 난이도에 비해 대회 기간도 짧은 편이었고, 연말에 새로고침 데이까지 겹쳐 기간이 정말 짧게 느껴졌고, 실제로 대부분의 시간은 코드 작성에 소모되었다. 코드 작성 능력은 지난 STS 대회나 RE 대회보다 확실히 늘었다고 생각이 되었지만, 이 나의 꼼꼼하지 못한 부분은 좀처럼 개선이 되지 않는 것 같다. (그렇게 검토를 많이 했는데!!) 코드 작성으로 인해 많은 실험과 리서치를 적용 해보지 못한 것은 아쉬웠고, 특히 wiki corpus를 이용한 다양한 pre-training 방법을 적용해 보는 것은 최종 프로젝트에도 사용될 수 있는 부분이라 생각되어 관심이 많았는데, 많이 실험해 보지 못해서 아쉬웠다.

그렇지만 지난 대회에 비해 협업 부분에서 정말 많이 발전된 것은 뿌듯한 점이었고, 최종 프로젝트를 수행하기 위한 초석은 어느 정도 마련이 되었다고 생각된다. (Retriever와 Generator 관련한 리서치)

설유민 개인 회고

1. 이번 프로젝트에서 나의 목표는 무엇이었는가?

1.1. 팀의 성장을 위한 목표

1) 공유하고 수렴하기: 누가 어떤 작업을 어떻게 진행하고 있는지 서로가 알고, 공유하는 문제점을 해결하도록 진행 방향을 수립하기

1.2. 개인 목표

1) Task와 관련된 다양한 연구들을 찾아보고 팀원들과 공유하기

2) 데이터 또는 모델로부터 발생하는 문제점을 찾고, 관성에 따라 하는 작업(hyperparameter tuning 등)이 아닌 개선 방안 적어도 1개 고안하고 제시하기

2. 나는 내 학습목표를 달성하기 위해 무엇을 어떻게 했는가?

1) NLP 논문 스터디에서 제시한 논문을 읽어보고자 함 → 실패

2) 내가 생각하는 문제(question-passage 쌍이 부족함)에 대한 연구 찾기 → 실패했고, 초점 자체를 잘못 맞춘 경향이 있는 것으로 판단됨 (Multitask learning으로 가성비 좋게 해결할 수 있었던 문제)

3. 나는 어떤 방식으로 모델을 개선했는가?

1) Reader 모델의 hyperparameter tuning

2) 외부 데이터를 사용해 1차 fine-tuning 후 우리 데이터 사용

4. 마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

1) 개인 목표 달성 실패: task 자체를 이해하고 기본적인 부분을 구현하는 데에 많은 시간을 소모하면서, 사실상 기본적인 retriever-reader 위에서만 작업

2) 여전히 부족한 실험 이유: 이 실험을 왜 할지에 대한 직관적인 이유는 있지만, 수치를 해석하고 증명하는 부분은 아직 부족함 (실험에서 변화시킬 것과 고정해야 할 것을 명확히 설정하고 실험을 시작해야 했는데, 조바심에 성능을 올리는 데에만 급급함)

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

1) Task 스스로 정의해보기: 내가 하고자 하는게 일반적인 generation인지, text style transfer인지, paraphrasing인지 빠르게 정할 필요가 있음

2) 많은 연구 찾기: 최종 프로젝트 주제와 관련해서는 별도의 로드맵이 주어지지 않기 때문에, '이건 이런 구조의 모델을 쓰면 된다'라는 게 없음. 내가 직접 찾아서 써야하는데, 지금 당장 시야가 좁으니까 그걸 빠르게 넓히는 작업이 필요함

3) 항상 '왜'라는 질문 달고 살기: 이번 대회에서 괜히 팀원을 방해하는 것 같아서 '이유가 있어서 그렇게 했을 거다'라고 생각하고 넘어간 부분이 있는데, 이때의 시간 소모가 조금 크리티컬했음

박수현 개인 회고

우선 mrc task자체가 내가 생각했던 것보다 어려웠던 탓에 개념과 코드를 이해하는데 생각보다 오랜 시간을 소요했다. 참 이 문제를 해결하는 데 얼마의 자원이 들지를 사전에 결정하는 건 어려운 작업 같다

1. 이번 프로젝트에서 개인 목표

1) baseline을 잘 이해하고 실험하기 : 성공
2) baseline refactoring 기여 : 큰 기여를 하지 못함.
3) 분석, 논리적으로 사고하고 실험하기 : 꾸준히 시도했으나 부족함이 있었음

2. 개인 목표를 달성하기 위해 한 활동

1) baseline refactoring 에 효과적으로 기여하지는 못했지만 그러기 위해 애쓰는 동안 baseline 을 잘 이해하고 실험에 참여할 수 있었다
2) 실험마다 실험 결과가 왜 나왔는지, 이게 왜 필요한지 찾아서 적용하려 노력했다.
3) EDA 및 데이터 전처리에 참여해 성능향상에 기여했다. EM score +4.2

3. 나는 어떤 방식으로 모델을 개선했는가?

1) Reader 모델에 evaluation loss를 추가하여 overfitting을 방지하고자 했다.
2) EDA를 수행하여 적절한 hyperparameter 선정을 위한 근거를 마련했다
3) wikipedia corpus/dataset 에 전처리를 수행

4. 마주한 한계 및 고쳐야 할 점

[올바르지 못한 방향설정으로 인한 낭비]

초반에 내가 했던 자원의 낭비는 SOTA를 찾아다니면서 시간을 꽤 허비한 것이다. KorQuad 2.0을 몇 달 전 카카오가 경신했다는 자료를 보고 해당 방법론을 학습하는데 시간을 썼고 우리가 쓰면 잘 나올지 한참 재고 생각하다가… 그럼 써보자! 하고 찾아봤는데 공개된 pretrained model이 없었다. 시간과 자원이 제약된 competition상 pretrained model의 사용이 필수적이기 때문에 결국 아무것도 적용할 수 없었고 프로젝트 진행 상 날린 시간이 되었다. 다음 대회가 있다면 SOTA를 찾는 것도 중요하지만 바로 적용할 수 있는 방법론인지 가부 판단을 먼저 한 뒤 더 조사해도 늦지 않다.

새로운 방법론 등을 이론적으로는 알게 되었기에 마냥 버려진 시간은 아니나, 팀 협업에서 내가 며칠을 허비한 건 꽤나 누수가 컸을 것이다. 답답했을 텐데 마냥 기다려준… 팀원들에게 그저 미안하다.

[비판적 논리적 사고 부족]

이번 대회에서 분석적 사고를 해보려 하니 약점이 그대로 드러났다. 왜 이 실험을 하는가 → 그래서 무얼 하는가 → 얻고자 하는 것은 무엇인가 → 그래서 얻고자 하는 걸 얻었는가? 왜 이런 결과가 나왔는가? 를 생각해 보려 했지만 논리적으로 뭔가를 하는 연습이 부족해서 그런지 팀원들의 질문 한두번이면 금방 논리가 파훼되었다. 어쨌든 꾸준한 연습이 필요한

부분이다.

그래서 코드리뷰, 리팩토링 잘하는 팀원들 보면 뭔가 고칠 점을 찾아낸다는 게 너무 신기하다. 신기하게만 생각하지 말고 왜 저걸 고치고자 했는지 따라해 보려 한다. 지식이 완전히 정리되지 않아서 그런지 뭔가를 설명하려 해도 준비한 걸 읽는 수준에서 벗어나지 못했다. 먼저 스스로에게 말로 설명하는 연습을 하자.

[덧붙여서 실수가 잦음]

실수가 잦은 건 빨리 바꿔야 할 내 업무습관이다. 이번에도 여러 차례의 실수로 팀원들이 했던 작업을 또 해야 하는 일이 있었다. 부캠 전까지는 대충대충 하고, 벼락치기 해도 대충 결과물의 질을 유지할 수 있었는데, 아무래도 새로운 분야고 하니 더욱 실수가 많아지는 듯하다. 한번에 좋은 결과물을 완성하지 못한다면, 충분한 여유시간을 가지고 작업을 미리 하고, 작업물의 결과를 여러 차례 랜덤샘플링해서 충분히 확인하고 넘겨야 한다. 덧붙여 팀원들과 이야기를 나눈 뒤 대화 내용과 결론을 요약해서 상대에게 말해주고 확인받는 시간을 갖는 습관을 가지자.

5. 다음 프로젝트에서 스스로 새롭게 시도해볼 것은 무엇일까?

코드 구현에 보다 많이 참여할 것. 리서치 결과를 머리로 아는 것도 중요하지만 결국 모델이 학습할 수 있게 구현해야 한다.

모델을 개선하고 싶어도, 아이디어가 떠오르지 않아 팀원의 도움을 많이 받았다. 어떠한 방법론이 있는지 알아야 우리 task에 적절한지 가부판단을 스스로 하고 실험해볼 수 있기 때문에, 최종 프로젝트에 적용하기 위해 시간을 내서 논문 조사 및 다양한 방법론을 익혀둘 것.

류재환 개인회고

1. 학습목표

MRC 분야가 쉽지 않은 task이기 때문에 아주 기초적인 부분부터 차근차근 이해하고자 하였고, 심화적인 부분은 대회 기간동안 따로 논문 스터디를 진행하면서 정보를 얻고자 하였다. 최종적으로는 End-to-end까지의 MRC 과정의 다양한 방법을 이해하고 왜 그러한 방법을 사용하는지 까지 숙지하고자 하였다.

2. 팀에서의 역할

팀에 기여할 수 있는 역할에 대해 고민을 한 결과, 코드 피드백을 통해 팀원들에게 많은 도움을 줄 수 있을 것 같아 저번에 이어서 한 번 더 코드 리뷰어를 맡아 깃허브를 통한 코드 관리를 하였다. 지난 대회와 달리 이번에는 혼자 코드 리뷰어를 맡아 역할이 더 중요하였다.

3. 모델 개선에서의 역할

3.1. Elasticsearch

Retriever의 일환으로 멘토님이 추천해주신 Elasticsearch 오픈소스를 사용하였다. 처음 다루는 툴이다 보니 시행착오도 많이 겪었지만, 제대로 설정을 마친 이후에는 기존에 사용하고 있었던 BM25 보다 더 좋은 결과를 얻을 수 있었고, 우리 모델의 최종 retriever로 사용되었다.

3.2. 코드 리뷰

코드 리뷰어로서 팀원들의 코드를 적극적으로 디버깅 해보면서 생기는 의문점을 팀원들과 공유하였고, 이를 통해 모델 개선에 기여하였다. 구체적으로 hard negative sampling코드에서의 shape를 팀원과 이야기해 보며 적절한 구현에 도움을 주었고, reader모델의 출력에서 스페셜 토큰을 지나치게 많이 예측하는 것을 팀원과 공유해보며 모델의 개선 방향을 설정하는 부분에서 기여할 수 있었다.

4. 마주한 한계

원래 계획한 학습 목표대로 베이스라인 코드를 자세하게 읽고 이해해 본 것은 좋았지만, 시간을 너무 사용한 나머지 스스로 찾아보고 연구하는 부분이 부족하였다.

코드 리뷰어로 팀원들에게 코드의 수정을 요구하는 단계에서 코드 관리가 아쉬운 부분이 있었다. 구체적으로는 코드 수정은 병합 이후에도 가능한데 지나치게 병합 이전에 오류를 다 잡으려고 한 나머지 통합 브랜치로의 병합이 매우 늦어졌다. 그만큼 코드의 안정성은 올라갔으나 대신 팀원이 다른 팀원의 코드를 사용할 때 불편함을 겪기도 하였다.

처음에 팀원들을 위한 베이스 코드 작성을 담당하였는데 결과물이 아쉬워 다른 팀원이 수정을 많이 해주었다. 캠프에서 제공해준 코드의 기능들을 전부 살리려고 했던 부분이 문제였으며, 코드의 취사선택이 아쉬웠음을 반성할 수 있었다.

5. 다음 프로젝트에서 스스로 새롭게 시도해볼 것은 무엇일까?

다음 대회에서는 리서쳐로서의 역할을 더 우선으로 하여 논문의 내용이나 여러가지 방법들을 직접 구현해보는 시도를 하면 좋을 것이다.

앞으로 코드를 리뷰 할 때, 내가 생각하는 바람직한 코드와 팀원들의 편의성 사이의 타협점을 잘 찾아보면 좋을 것이다.

박승현 개인 회고

1. 목표

지난 대회 및 여러 상황에서 팀원들이 각자 따로따로 노는 느낌이 들었기 때문이 이번에는 팀원들이 하나가 될 수 있도록 조율하고, 개인적인 공부와 개발 활동을 꾸준히 이어나가는 것이 목표였다.

2. 작업 목록

2.1. Reader Code refactoring

대회에서 제공해 준 baseline 코드가 개인적으로 난잡한 것 같아서 수정해줬다. 특히, 다른 팀원들이 모델 및 학습 과정에 대해 변화를 주고 싶을 때 손쉽게 적용해 볼 수 있도록 코드를 작성했다. 또한, Reader가 어떠한 방식으로 작동하고 학습되는지에 대해 학습했다.

2.2. Reader 모델 Tuning

Retriever와 Reader 모델을 모두 적용하는 end-to-end 실험을 많이 진행했다. Transformer 모델, learning rate, batch size, warm up, lr scheduler 등을 변경하면서 최적의 조건을 탐색하기 위해 노력했다.

2.3. 결과 심층 분석 및 후처리

실제로 Reader 모델에 변화를 줄 수 있는 부분이 마땅치 않아서 큰 변화를 주지 않았다. Reader 모델에 변화를 주지 않고 좋은 성능을 도출하기 위한 방법을 모색하기 위해, n best, top-k passage, max answer length 등 다각도에서 결과 분석을 진행해봤다. 그 결과, n best와 passage similarity 적용 등 부분에서 후처리를 적용했고, 성능 향상도 이끌어냈다.

2.4. 추가 데이터 사용 및 tuning

대회에서 제공된 데이터가 소수이기에 데이터를 증강하기 위해 외부 데이터를 사용했다. 외부 데이터를 사용하기 위한 방법은 다양하지만 이번에는 KorQuAD만 추가하여 train시키도록 했다. KorQuAD가 먼저이냐, 대회 train이 먼저이냐, 모두 섞어서 학습시키는게 좋은가 등에 대한 실험을 진행했다.

2.5. 팀원 모니터링

어떤 팀원이 어떤 업무를 하고 어떤 결과를 도출했으며, 어떤 분석을 할 수 있는지 같이 고민하고 전반적인 프로젝트의 진행 상황을 조율했습니다. 프로젝트의 흐름이 끊기지 않도록 하기 위해 남들이 잘 안하는 업무를 담당해 진행했습니다.

3. 고찰

1) 본인이 최대한 협업을 이끌어내려고 노력했지만, 아직은 팀원 각각 따로 노는 느낌을 지

우기는 힘들었던 것 같다. 특히, retriever와 reader로 역할을 분배하면서 다들 자신과의 싸움을 하기에 급급해 보였다.
2) 개인적으로 generation 부분을 마스터하고 싶은 욕심이 있어 reader 부분을 빠르게 마무리하고 generation 관련 논문도 읽고 GPT에 대한 학습도 진행했었다. 그러나, 내가 팀원들을 조율해주지 않으면 팀원들은 각자 맡은 업무만 담당했기에 모든 실험과 과정을 총괄해줄 필요도 있었고, 문제가 있는 팀원들과는 지속적인 소통도 유지했어야 했다. 따라서, generation을 포기하고 다른 팀원들의 조화로운 협업 형성에 더 집중했고, 해당 프로젝트가 산으로 가지 않도록 꾸준히 모니터링하는데 많은 시간을 쏟았다. 이로 인해 generation 부분은 거의 많이 보지 못했다.
3) 너무 많은 것을 다 잘하려고 했던 점이 독이 됐던 것 같다. 특히, MRC 하던 도중에 다른 공모전도 병행했기에 분신술이 시급했었다. 잠을 줄이고 최대한 효율적으로 시간을 사용함으로써 시간을 많이 낼 수 있었고, 그 결과 만족하지는 못했지만 공모전에서도 MRC 대회에서도 생각 내의 결과를 도출했던 것 같다.

4. 추후의 목표

1) 본인은 AI 엔지니어일 수도 있지만 개발자이다. 꾸준히 개발을 이어나가고 공부도 해야한다. 따라서, 꾸준히 개발 능력을 키워나갈 필요가 있다.
2) 정리하는 습관이 아직은 조금 부족한 것 같다. 다른 팀원은 정리를 잘하는 것 같은데 유독 나는 조금 산만하게 정리하는 듯했다. 따라서, 추후 프로젝트에서는 더 완벽한 정리가 필요할 것 같다.
3) 아직도 비슷한 생각을 가지고 있지만, 부스트캠프에 집중하는 것도 분명히 중요하지만 남들과 같은 것만 해서는 부캠 수료생 1 밖에 되지 못한다. 따라서, 남들과 차별점을 둘 수 있는 개발 능력이든 뭐든 해야 할 필요성을 느끼고 있으며, 현재 조금씩 실천 중이다.