Open-Domain Question Answering 프로젝트

Open-Domain Question Answering

Wrap-Up 리포트

2023.06.05 ~ 2023.06.22

NLP-4 조(데굴^2)

곽민석_T5014

이인균_T5150

임하림_T5178

최휘민_T5221

황윤기_T5229

1. 프로젝트 개요

프로젝트 주제

Open-domain Question Answering(ODQA)은 주어진 질문(query)에 대해 사전에 구축된 지식 체계 내에서 알맞은 대답을 찾아내는 과제이다. 일반적인 Question Answering 과제와는 달리 ODQA 에서는 특정한 지문이 주어지지 않고 방대한 지식 체계 내에서 지문을 찾아내는 과정이 추가되어야 하므로 더 어려운 문제라고 볼 수 있다.

프로젝트 구현 내용, 컨셉, 교육 내용과의 관련성 등

- 주어진 Wiki Documents 에서 질문에 해당하는 문서를 찾고, 정답 Span 을 예측한다.
- 2 Step 으로 문제를 해결하기 위해, 각각의 Step 에 맞게 모델을 따로 학습한다.
  - 주어진 문서에서, 질문에 대한 정답이 들어있는 문서를 검색하는 Retrieval 모델
  - 검색된 문서에서, 정답 Span 을 찾는 Reader 모델
- 효과적으로 검색 및 정답을 찾기 위해, 모델 구조 변경, 데이터 증강 등을 시도한다.

활용 장비 및 재료(개발 환경, 협업 tool 등)

(팀 구성 및 컴퓨팅 환경) 5 인 1 팀으로 각자에게 지원되는 V100 서버를 활용하여 각자 VSCode 혹은 PyCharm 등 편한 IDE 를 사용한다.

(협업 환경 및 의사 소통) Notion, Github, Wandb, Slack, Zoom

데이터셋 및 프로젝트 구조

데이터셋은 다음과 같다.

Wikipedia document 60613 개, Train Data 3952 개, Valid Data 240 개, Test Data 600 개

Wikipedia document 의 구조는 다음과 같다.

text	문서의 텍스트
corpus_source	문서의 출저
title	문서의 제목
document_id	문서의 고유 id

Train, Valid, Test 데이터셋의 구조는 다음과 같다.

title	문서의 제목
context	답변이 포함된 문서
id	질문의 고유 id
question	질문
answers	답변에 대한 정보. 하나의 질문에 하나의 답변만 존재한다
answer_start	답변의 시작 위치

text	답변의 텍스트
document_id	문서의 고유 id

프로젝트 팀 구성 및 역할

이름	역할
곽민석	Reader 모델 데이터 증강, Elastic Search 구현 및 적용, 모델 공유 툴 제작
이인균	Retriever 모델(DPR) 구현, Reader 모델 개선
임하림	Retriever 모델(BM25) 구현, Context 전처리
최휘민	Reader 모델 개선, Question Generation 실험, Ensemble 구현
황윤기	Retriever 모델(DPR), BiEncoder Trainer 구현, Re-Rank 구현

2. 프로젝트 수행 절차 및 방법

Retrieval

Sparse Retrieval

- **TF-IDF**

  TF-IDF 는 한 문서 내에서 단어가 얼마나 반복되는지에 대한 빈도(Term Frequency), 전체 문서 내에서 특정 단어가 등장한 빈도(Inverse Document Frequency)를 결합한 방법론이다.

- **BM25**

  Passage 의 길이를 분석했을 때 길이가 각기 다른 것을 확인하고, Sparse retrieval 에서는 SOTA 모델이고 passage 의 길이도 고려하는 BM25 를 도입했다.

- **Elastic Search**

  기존 BM25 라이브러리를 사용하는 경우 실행하고 있는 서버의 컴퓨팅 자원을 소모하고, 인덱싱을 매 실행 시 진행하기 때문에 Elastic Search 를 적용해 보았다.

  이를 구현하기 위해 Elastic Search 를 서버에 호스팅한 후 개발을 진행하려 했으나, 제공받은 서버가 Docker 에서 돌아가는 환경인데, 해당 환경에서는 특정 권한이 없어 진행이 불가능하여 부득이하게 외부 서버에 구축하여 실험해 보았다. 먼저 Elastic Search 에 RestAPI 를 이용하여 JSON 에 있는 위키백과의 데이터를 모두 외부 서버에 옮겼다. 그 후 프로젝트에서 사용할 수 있도록 Retrieval Class 내에서 REST API 를 호출하여 결과를 보여주도록 제작하였다.

Dense Retrieval

- **DPR**

  앞선 Sparse 계열 Retrieval 방법론은 질의와 문서 간의 어휘적인 유사도만을 고려하기 때문에, 질의와 문서 간의 의미적인 유사도를 고려하기 위해서 Dense 계열의 방법론을 도입할 필요가 있었다. 따라서 Query 와 Context 를 Transformers Encoding 을 진행하고, 유사도를 측정하는 DPR 모델을 도입하였다.

Re-Rank

Sparse Model 은 어휘적인 유사도가 높은 질의-문서 쌍을 잘 찾고, Dense Model 은 의미적인 유사도가 높은 질의-문서 쌍을 잘 찾는다는 점에서 착안하여, Sparse Retrieval 결과물을 미리 학습시킨 Dense Retrieval Model 로 다시 순위를 부여하는 작업을 진행하였다.

Reader

기본 베이스라인에 구현되어있는 Reader 모델은 Bert-base 로 규모가 작은 PLM 이었다. 파라미터의 크기를 키우면 성능이 좋아질 것이로 생각해 Roberta-Large 로 변경하였다.

Retrieval 모델보다 Reader 모델의 성능이 크게 뒤떨어졌기 때문에 조금이나마 올리기 위해 미세조정을 시행하였다.

Data

외부 데이터셋 증강

현재 기본 데이터셋에는 3,952 개의 학습용 데이터, 240 개의 테스트 데이터 그리고 600 개의 Validation 데이터가 있다. 여기서 생각해 본 문제점은 데이터셋의 개수가 적기 때문에 Reader 모델의 성능이 낮게 나오는 것으로 판단하여 데이터를 증강해보기로 하였다.

이번 Task 에 맞는 데이터셋을 찾아본 결과 총 3 가지의 데이터셋을 찾아냈다. AIHub 의 "기계독해"와 "도서자료 기계독해" 그리고 KorQuAD 1.0 데이터셋이다. AIHub 의 두 데이터셋은 각각 뉴스 기사로 이루어진 것과 도서 자료로 이루어진 데이터셋이다. KorQuAD 2.0 이아닌 1.0 을 사용한 이유는 2.0 이 1.0 보다 질문 자체가 조금 더 까다롭고, 테이블 해석이 가능해야 해결 가능한 문제가 있어서 1.0 을 사용하였다.

실험해 본 항목은 다음과 같다.

1. "기계독해"와 "도서 자료 기계독해" 두 데이터셋을 모두 합쳐 학습.
   - "도서 자료 기계독해" 데이터셋과 기본 제공된 데이터셋을 1 epoch 마다 번갈아 가며 학습.
   - 해당 실험이 좋지 않은 결과를 내에 각 데이터셋을 학습시켜 실험.
2. Task 에 맞지 않는 정답(문장 형태, 너무 긴 답변 등)이 들어간 데이터는 모두 제거 후 새로 데이터셋 구축.
   - KorQuAD 1.0 데이터셋과 기본 데이터셋을 바탕으로 학습.

3. 일정 epoch 동안 KorQuAD 1.0 데이터셋을 이용하여 학습시킨 후 기본 데이터셋으로 학습.
   - KorQuAD 1.0 데이터셋 학습을 각각 1, 2 epoch 후 기본 데이터셋으로 1 부터 5 epoch 까지 학습.

Question Generation

주어진 데이터에는 답변의 시작 위치를 포함하기 때문에 적절한 데이터 증강 방법으로 context 와 answer 를 입력해 question 을 생성하는 Question Generation 증강법을 시도하였다. SKT KoGPT2 모델을 사용하여 기존의 Question Answering Dataset 에서 input 으로 context 와 answers 를 입력하고 target 을 question 으로 지정해 학습하여 context 와 answers 로 question 을 생성하는 Question Generation 방법을 구현하였다.

결과는 적절성과는 상관없이 질문 자체는 잘 생성하였기 때문에 정확도를 높이기 위하여 대회의 dataset 과 KorQuAD 1.0, AIHub 의 "기계독해" 데이터를 합쳐서 학습을 해보는 실험도 하였다.

3. 프로젝트 수행 결과

Retrieval

Top-K retrieval rate 는 전체 질문에 대하여 각각 지문 K 개를 선정하였을 때 그 속에 정답이 되는 지문이 존재하는 비율이다. 이해하기 쉽고 구현이 빠르다는 장점이 있다. 또한 Retrieval 모델이 Top-K 지문을 탐색하면 그 지문이 모두 Reader 모델에 활용되기 때문에 정답 지문의 포함 여부가 성능을 잘 나타낸다고 생각하였다. 따라서 성능 지표로 retrieval rate 를 사용하기로 결정하였다. K 를 10 이 아닌 다른 숫자로 변경하여도 눈에 띄는 특이사항 없이 단조증가 하는 모습을 보였다.

- **TF-IDF**

  기본 baseline 에 있었던 Retreival 방식이었다. Top-10 retrieval rate 는 66.666%(tokenizer: Roberta-large)로 좋지 않았다.

- **BM25**

  Bm25 에서 문장을 토큰화할 때 쓰이는 기본 방식은 띄어쓰기이다. 이때 Top-10 retrieval rate 는 19.166%로 나쁜 성능을 보였다. 단어의 빈도수를 셀 때 단어가 "고구마를"과 같이 조사도 포함된 형태이기 때문으로 추정되었다. 따라서 좀 더 자세한 토큰화가 필요하였다. Bert-base-multilingual-cased 의 tokenizer 를 통해 넣어보았고 성능이 64.166%로 증가하였지만, 이 역시 좋은 성능을 보이지는 못했다. tokenizer 가 다국어를 지원하긴 하나 상당수의 한국어 단어가 UNK 로 토큰화되기 때문이라 생각하였다. 이후 한국어에 특화되어 있는 klue/roberta-large 를 통해 성능을 더욱 증가시켰다. 그렇게 최종 성능은 Top-10 retrieval rate = 90.000%으로 뛰어난 성능을 보였다. 질문과 지문 사이에서 완벽히 일치하는 단어를 잡아내면서 서로 다른 지문의 길이도 고려하기 때문에 좋은 결과를 보인 것으로 추정된다.

  아래의 Elastic Search 와 비교해 본 결과 BM25 의 top-1 retrieval rate = 54%로 elastic search 의 36%에 비해 더 좋은 성능을 보여주었다. 이러한 결과를 바탕으로 기본 BM25 를 선택하였고, 최종 결과물 또한 BM25 를 사용하였다

- **Elastic search**

  Elastic Search 가 default 설정이 BM25 기반으로 동작하게 되어있다. Elastic Search 는 Top-1 retrieval rate 가 36%(자체 tokenizer)의 성능을 보였다.

- **DPR**

  DPR 을 사용한 Retrieval 모델의 성능은 Top-10 retrieval rate = 49.583%으로 좋지 않았다. 이처럼 안 좋은 성능이 나온 이유는 klue/mrc 데이터셋은 주석자가 지문을 보는 상태에서 질문을 작성하기 때문에 지문의 단어를 그대로 질문에 사용하는 경향이 있기 때문으로 추측된다. 데이터셋을 눈으로 본 결과 지문과 질문의 단어가 일치하는 경향이 상당했다.

  ReRank

  BM25 로 Top-50 개를 뽑고, 미리 학습시켜 둔 DPR 모델로 다시 Top-20 을 도출한 결과 Train Dataset 에 대해서, Top-20 retrieval rate 가 64.217%이다.

Reader

최종적으로 리더보드에 사용되는 성능 지표가 exact match 이기 때문에 Reader 모델의 성능 지표로도 exact match 를 사용하였다. F1 Score 의 경우 exact match 와 정비례하는 모습을 보였다.

Bert-base 를 사용하는 Reader 모델의 성능은 exact match = 55.833%로 좋지 않았다. PLM 모델의 파라미터 크기가 작기 때문이라 추측된다.

Robeta-large 를 사용하는 reader 모델의 성능은 exact match = 69.583%로 개선되는 모습을 보였다. 파라미터 크기가 커질수록 성능이 좋아지는 경향이 있고 Bert 보다는 Roberta 가 언어를 더 잘 이해하는 모습을 보인다.

추가로 미세조정을 진행한 결과 exact match = 72.083%로 개선되는 모습을 보였다. 미세조정을 했기 때문에 성능이 좋아지는 것은 당연한 것이고 과적합의 우려가 있기 때문에 많이 사용하지는 않았다.

외부 데이터셋 증강 이후 미세조정을 하지 않은 reader 모델의 성능은 71.25%로 성능이 개선되면서 과적합의 우려가 없기 때문에 해당 모델을 사용하였다. 구체적인 방법에 대해서는 Data 문단에서 더 자세히 서술하였다.

Data

외부 데이터셋 증강

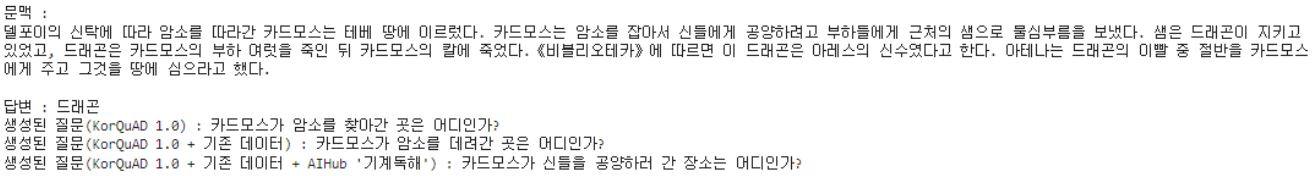
1 번 실험에 대해 klue/roberta-large 모델에 학습시킨 후 본 데이터셋으로 테스트해 본 결과 EM Score 가 55 점이 나오게 되었다. 1 번의 후속 실험에 대해 학습이 잘되지 않는 경향이 확인되어 "기계독해" 데이터셋은 제거하였다.

2 번 실험의 결과로 총합 3 epoch 때 EM Score 가 약 65 점대로 나온 후 50 까지 떨어진 후 학습이 되지 않았다.

3 번 실험 결과 "KorQuAD 1.0"는 1 epoch, 기본 데이터셋은 4 epoch 를 학습시켰을 때가 가장 좋은 점수를 보여주었다.

결과적으로 외부 데이터셋 증강을 이용하여 Evaluation 기준 EM 71.25, F1 79.7982 이 나왔다.

Question Generation



Question Generation 결과 적절하지 않은 질문들이 많이 포함되었다. 초기 KorQuAD 1.0 만 사용하여 학습하였을 때, 생성한 문장이 질문형태로는 온전히 만들어졌기 때문에 answer 와 상관없는 question 을 만드는 문제점은 학습데이터의 부족과 KorQuAD 와 기존 대회 데이터의 문맥 스타일의 차이로 생각하여 KorQuAD 에 기존 데이터를 추가해 학습하였고 추가로 3 만 개의 AIHub 의 데이터도 추가해 학습하였다.

그 결과로 위의 이미지처럼 학습데이터를 추가하면 질문의 표현은 풍부해 지지만 잘못된 질문을 생성하는 문제는 해결할 수 없었다. 위의 이미지는 문맥이 주어지고 답변은 "드래곤" 이지만 생성된 질문은 전부 적절한 질문이 아니거나 "근처의 샘"을 가정하고 만들어진 것처럼 보인다. 이러한 문제는 100 개의 sample 의 뽑았을 때 절반 이상의 데이터에서 나타나는 부분이기 때문에 reader 모델 개선에 사용할 수 없었다. 생성 관련 하이퍼 파라미터를 변경해도 개선할 수 없던 부분이었고 모델 부분을 개선해야 한다고 생각했지만, 대회 기간이 촉박하고 새로운 모델의 학습 시간을 생각하면 더 진행할 수 없었다. 그래서 Question Generation 은 사용하지 않았다.

시연 결과

```
***** eval metrics *****
  epoch        =    3.98
  eval_samples =     474
  exact_match  =   71.25
  f1           = 79.7982
***** test metrics *****
  eval_samples =    5349
  exact_match  = 60.4167
  f1           = 68.7082
```

첫 번째 결과는 klue/roberta-large 에 가장 좋았던 조합인 KorQuAD 1 epoch 후 기본 데이터셋 4 epoch 를 학습시킨 reader 모델의 validation 결과이다.

두 번째 결과는 이번 Task 에서 가장 성능이 좋았던 Retrieval 인 BM25 와 위의 실험을 통해 나온 Reader 모델을 이용한 end-to-end 의 validation 결과이다.

4. 자체 평가 의견

● **잘한 점들**

- 프로젝트가 어중간하게 떠버리는 일들과 다소 지치는 일을 겪고 이것들은 최종 프로젝트 때 일어날 수도 있었던 일이기에 회고 때 잘 정리해서 재발을 방지했다.

- Git-Flow 형식을 통해, 기능별 브랜치 관리와 버져닝이 잘 되었다.

- 최신논문 기반 구현을 시도하였다.

● **시도했으나 잘 되지 않았던 것들과 개선사항**

- DPR 모델의 성능이 생각보다 BM25 보다 더 좋아지지 않아서, Encoder 모델의 성능 문제라 생각하여 규모가 더 큰 모델로 변경하여 수행해 보았다. 그 과정에서 메모리 이슈 때문에, Batch Size 를 줄이면서 학습을 진행했는데, 규모가 작은 모델에 비해 오히려 성능이 아주 크게 감소했다. 이로 인해, DPR 모델은 학습 과정에서 아주 큰 Batch Size 가 필요로 한다는 것을 깨달았다.

- 또한 DPR 모델에서 포함되는 BM25 의 Hard Negative 결과를 3 개 정도로 늘리면, 일반화 성능이 올라가지 않을까 했지만, 성능은 개선이 되지 않고 학습시간만 크게 늘어났다.

- Elastic search 에서 언어 설정을 unviersal 로만 작동 시켜본 것이 아쉽다. 언어 설정을 Korean 으로 다시 시도했을 때 API 문제와 시간 문제로 작동을 못 시켜봤는데 다른 팀 발표에서 elastic search 가 더 좋았다는 것을 듣고 성능을 더 올릴 수 있었을 것 같아 아쉽다. 또한 BM25 에서 Tokenizer 를 변경하여 성능이 올라갔었는데, API 를 이용하여 문서 업로드 시 Tokenizing 을 먼저 한 후 서버로 보냈으면 조금 더 좋은 Indexing 결과가 나오지 않았을까 생각한다.

- Context 전처리를 시도했으나 이것을 정량적으로 평가 못 한것이 아쉽다. 마지막에 문제점을 파악해서 구현도 늦었고 마음도 급해서 찬찬히 했으면 잘 마무리 하고 성능도 올렸을 것 같은데 아쉽다. 아무리 급하더라도 문서에 정리하면서 확실하게 해야겠다고 생각했다.

- BM25 와 DPR 을 10 개씩 뽑아 20 개를 Retrieval 한 것이 잘되지 않아 아쉽다. BM25 와 DPR 의 문서 뽑는 기준이 상이하게 다르기 때문에 같이 뽑아서 inference 시키면 성능이 오르겠다고 생각을 했는데 아쉽다. 이후 Re2G 의 논문에서 Rerank 하는 방식만 가져와서 해보려고 했는데 시간도 많지 않았고 논문도 잘 이해하지 못해서 구현을 못 해본 것이 더 아쉽다.

● **아쉬웠던 점들과 개선사항**

- 데이터셋에 맞는 모델을 사용하지 못한 점이 아쉬웠다. 데이터셋 분석을 통해서, 잘 맞는 모델을 선택해서 집중하여 개선해야 했지만 구현하기 쉬운 모델을 우선적으로 선택해서 시간을 썼다. 하지만 생각보다 결과가 좋게 나오지 않았기에, 해당 모델을 개선하는 방향으로 계속해서 시간을 소비했지만, 결론적으로는 우리 데이터셋과 모델과의 궁합이 좋지 않았던 것이 문제였다.

- 기존 프로젝트에는 데이터 EDA 에 시간을 상당히 썼었다. 그러나 이번에는 강의에서 배운 점들이 많다보니 그것들을 프로젝트에 적용하는 것에만 시간을 썼고 데이터의 특징에 대해서는 거의 분석을 하지 않았다. 이로 인해 늦은 시점에서 크고 중대한 시행착오가 발생했다. 앞으로는 데이터 분석을 먼저 수행해야 한다.

- 실패하였던 시도에 관한 분석을 소홀히 했다고 생각한다. 단순히 판단하여 넘어가는 것이 아닌 면밀한 분석과 팀원과의 의논을 통하여 실패를 분석할 수 있도록 개선해야 한다고 생각한다.

- 평가 지표의 정의와 공유가 잘 이루어지지 않았다. 분업을 하긴 했으나 같은 분야를 맡은 사람들끼리는 동일한 평가 지표를 사용했어야 했다. 앞으로는 평가 지표도 중요시해야한다.

● 프로젝트를 통해 배운 점 또는 시사점

- 데이터를 증강하면서, Task 해결에 맞는 데이터를 사용해야 한다는 점을 배우게 되었다. 아무리 큰 데이터셋을 구축하였다 해도, 학습 내용이 해결하고자 하는 Task 의 데이터와 다르게 된다면 결과가 좋지 않기 때문이다.

- 데이터를 정량적으로 평가할 수 있는 다양한 방법론이 있지만, 자연어를 처리하는 만큼 반드시 해야 하는 것은 사람이 직접 데이터를 찬찬히 뜯어보고 분석하는 시간을 가지는 것이다.

- 최종 프로젝트에는 수집한 데이터셋의 분석과 우리가 도출해낼 결과를 고려하여, 모델의 설계를 진행해야 한다.

1. 성능 향상을 위한 노력

- 이번 Task에서 주어진 데이터셋은 이전의 기본 데이터셋보다 현저이 적어 데이터셋 증강 위주로 진행해보았다.
  - AIHub에서 찾아본 결과 “기계독해”, “도서자료 기계독해”를 찾게 되었고, 추가로 KorQuAD 1.0을 사용하기로 하였다.
- 위의 데이터를 이용하여 “외부 데이터셋 증강” 부분을 맡아 진행하였는데, 여러 실험 끝에 KorQuAD 데이터셋을 이용하여 1 epoch를 학습 시킨 후 기본 학습데이터를 이용하여 4 epoch를 학습시키는 방법을 사용하였다.
- 이렇게 학습된 Reader는 팀 내에서 가장 좋은 성능을 보여주어, 최종 Reader로 사용되었다.
- 또한, Retrieval 부분에서 Elastic Search를 적용해보았다.
  - 구현 이전에 이미 BM25가 구현되어있었으나, 해당 방법은 프로그램 작동시 매번 Indexing을 하여 실험에 어려움이 있어 구현하게 되었다.
  - Retrieval의 Elastic Search 부분에서 언급한것과 같이, 서버 내부에 설치가 불가능하여 외부에 서버를 설정하여 작업을 진행하였다.
- 위의 작업을 모두 마친 후 문서 검색을 진행했을 때, 기존 BM25의 bert-base-multilingual-cased의 tokenizer를 사용한것보다 성능이 좋았었다.

2. 전과 비교해서, 새롭게 시도한 것과 효과

- 이번 Repository에서는 dev branch를 따로 관리해보았다.
  - Dev branch에서 실제 작업 branch를 만들어 작업 후 다시 dev branch에 병합한 뒤, 작업물의 결과를 모두 확인 한 후 완전히 작동 가능한 버전을 main branch에 merge 하였다.
  - 이로인해 버전관리가 이전보다 한층 쉬웠었다.
- Retrieval의 구현될 부분이 많아 부모 Class를 정의 후 상속하여 사용할수 있도록 제작해두었다.
  - Retrieval의 공통되는 부분을 부모 Class에 지정해두어 Retrieval 제작 시 다른곳을 수정하지 않아도 요구에 맞게 제작하면 작동하도록 하여 개발시 불편한점을 제거하였다.

3. 한계점과 아쉬운 점 그리고 해결 방안

- 학습 시키기 전에 최대한 어떤 데이터인지 파악하고 접근했어야 했다.
  - 이번에 데이터를 수집하며 느꼈던 점은, 먼저 어떤 데이터를 사용하여 어떤 결과를 가져올것인지 우선 확인해야 한다는 것이다.
  - 데이터가 아무리 많다 하여도 요구사항에 맞지 않는 데이터는 완전히 필요가 없음을 깨닳았다.
- 프로젝트 초반 베이스라인 코드를 알기위해 많은 시간을 할애하였는데, 이때 다른사람의 코드를 이해하는것이 힘들다는것을 다시한번 깨닳게 되었다.

- 이전에 프로젝트를 진행하면서 알고는 있었지만, 제작자의 구두 설명이 있다면 훨씬 이해가 잘 되기 때문에 최종 프로젝트에서 내 코드를 사용하는 다른 팀원에게 도움이 되도록 세세한 설명이 필요할것 같다.

4. 다음 프로젝트에서…

나는 이번 프로젝트를 최종 프로젝트 이전의 연습이라 생각한다. 이전의 프로젝트의 경우 하나의 명시적인 Task가 있어서 팀원 모두 같이 관리가 가능했으나, 이번 프로젝트는 크게 두 부분으로 나뉘어 관리를 했어여 했다. 또한 팀원 개인간에 서로 일을 나누어 분담했었는데, 이때 일을 잘 나누어야한다는 점 또한 느꼈었다.

이러한 이유로 인해 최종 프로젝트에서는 UML을 만들고 최대한 작업 이전에 어떤 부분을 담당할지 많은 얘기를 해보고 진행하는것이 좋을것 같다 생각했다.

이인균

나는 내 학습목표를 달성하기 위해 무엇을 어떻게 했는가?

- 이번 프로젝트의 학습 목표는 논문을 바탕으로 한 모델링과 성능은 좋지 않더라도 나만의 생각이 담긴 창의적인 시도를 하는 것이었다.

- 구체적으로 말하자면 Dense Passage Retrieval for Open-Domain Question Answering 논문과 Learning Dense Representations of Phrases at Scale 논문을 읽었고 이를 바탕으로 한 DPR 모델, in-batch negatives, pre-batch negatives 를 구현하고 싶었다. 또한 hard negatives 에 대한 새로운 아이디어가 떠올랐다. 팀원을 Retrieval 파트와 reader 파트로 나누기로 했는데, 나는 Retrieval 파트와 그중에서도 DPR 모델을 구현하는 것을 자원을 하였다. 구현은 하였지만 시간이 생각보다 오래 걸렸고 성능이 좋지 않았다. 이후에서야 데이터 분석을 시도하였고 성능이 좋지 않을 수밖에 없는 데이터셋이라는 점을 뒤늦게 파악했다.

- 시간과 성능의 한계 때문에 DPR 모델과 관련된 아이디어는 구현을 해보지 못했다.

- 모델의 성능이 안 나오는 이유를 분석한 결과, BM25 를 활용한 retrieval 모델의 성능은 뛰어나나 reader 모델의 성능이 좋지 않았었고 이점이 일종의 보틀넥으로 작용했다고 생각하였다. reader 모델의 개선에 집중하였으나 미세조정을 제외하면 별다른 소득이 없었다. 다른 팀의 방식을 들은 후에야 데이터 전처리를 하는 것이 성능에 더 중대한 영향을 끼친다는 점을 뒤늦게 파악했다.

전과 비교해서, 내가 새롭게 시도한 변화는 무엇이고, 어떤 효과가 있었는가?

- 팀원들과 논문 스터디를 다시 시작하여 이전 프로젝트보다 논문을 더 많이 읽었다. 1 개의 논문에서는 접할 수 없었던 발상을 새로 익힐 수 있었고 이를 바탕으로 나만의 생각 역시 할 수 있었다.

- 미완성이었지만 나만의 아이디어를 생각 후 구현을 시도했다. 기존 논문들의 hard negatives 구현의 문제점은 BM25 와 같은 별도의 모델을 사용하기 때문에 비효율적이고 BERT 인코더가 잘 구별하지 못하는 어려운 지문과는 다른 지문을 제시할 수 있다는 점이다. 따라서 BERT 인코더를 똑같이 사용 후 지문들에 대해 출력되는 임베딩 벡터를 정렬한 뒤 sequential sampler 를 통해 DPR 모델을 학습시킬 경우, in-batch negatives 가 자연스럽게 어려운 예시들로 구성이 될 수 있을 것이라고 생각했다. 논문을 읽은 것에 그치지 않고 논문의 한계점을 생각하고 그에 대한 해결책을 떠올리는 논리적 단계를 밟기 위해 노력했다. 그대로 따라만 했던 전과 비교했을 때 더 논리적이고 새로운 접근법을 취할 수 있었다.

마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

- 프로젝트 초기에는 외부 대회, 마지막 프로젝트, 기업 연계 프로젝트를 준비하느라 팀적으로 시간을 온전히 쏟지 못했다. 이뿐만 아니라 DPR 모델 구현에 있어서 나의 속도가 느렸었고 코드를 리팩토링하는 과정에서 팀적인 비효율성이 있었다. 첫 번째 한계는 어쩔 수 없었지만 두 번째 한계는 파이썬을 다루는 미숙함이 드러났다. 알고리즘 코딩 테스트와 부스트캠프를 통해 파이썬 실력이 좋아지고 있지만 클래스 형식과 입출력을 다루는 것은 여전히 쉽지 않다. 세 번째 한계는 팀원들과의 공감이 있었고 다음 프로젝트에서는 효율적으로 이뤄질 수 있을 듯하다.

- 모델의 성능이 좋지 않았고 결과적으로 그동안 줄곧 중상위권~상위권을 유지했었는데 이번에는 리더보드 꼴등을 하였다. 강의에서 배운 것들이 많았고 해당 사항들을 적용하는 데에 시선이 끌려서 데이터에 집중하지 못했던 점이 원인으로 보인다. 정작 성능에 큰 영향을 미친 것은 커다랗고 기발한 최신 모델링이 아니라 데이터 전처리와 작은 코드의 수정이었다. 그동안은 데이터에 집중을 잘 해왔었는데 이번에는 놓친 점이 아쉽다.

한계/교훈을 바탕으로 다음 프로젝트에서 시도해보고 싶은 점은 무엇인가

- 프로젝트가 1 개만 남은 만큼 그동안의 미진했던 점들과 좋았던 점들을 놓치지 않고 해보고 싶다. 구체적으로 말하자면 모델링 구현, 데이터 전처리, 데이터 분석, 더 나은 협업, 더 효율적인 협업을 하고 싶다.

- 임하림 님의 리포트를 보고 떠올린 것인데 프로젝트를 하면서 기억이 휘발되는 느낌이 강했다. 대략적으로 3 일의 기간마다 기록을 하긴 했지만 이로써는 충분하지 않다고 생각한다. 이번에는 나 역시 하루 단위로 내가 했던 것들을 기록해서 나를 객관적으로 확인을 하고 싶다. 또한 구체적인 시간 소요가 정의된다면 문제 해결, 즉 다음 프로젝트에서는 조금 더 빠른 일 처리를 할 수 있을 듯하다.

임하림

나는 내 학습목표를 달성하기 위해 무엇을 어떻게 했는가?

- Baseline 을 잘 쓸 수 있도록 기능에 따른 모듈을 세분화해서 refactoring 을 진행했다.
- Retrieval 에 구현을 진행했고, 특히 Sparse Retrieval 에 집중했다.
    a. Passage 의 길이가 많이 차이 나는 것을 확인하고 기존의 TF-IDF 보다, SOTA 모델이고 passage 의 길이까지 고려하는 BM25 를 구현했다.
    b. BM25 를 구현하기 위해 tokenizer 를 써야 해서 기본에 있던 띄어쓰기 단위로 passage 전체를 나누어 Retrieval 을 시도했다.
    c. 하지만 '~를', '~가'처럼 조사가 붙어서 Retrieval 의 성능 하락이 있는 것을 확인했고, 단어가 나오는 것보다 의미로 나눈 subword tokenizer 의 기준으로 나누어 해결하는 것이 맞다고 생각해서 huggingface 의 autotokenizer 를 이용했다.
    d. 처음에는 baseline 의 Reader 모델인 bert-base-multilingual 의 tokenizer 로 시도했지만, 민석님이 말씀해주신 klue/roberta-large 의 tokenizer 를 사용한 것이 더 잘 된다 라는 의견에서 성능이 올라간 것을 확인했고, 한국어 기반을 통해 올랐다 라고 결론을 내렸다.
    e. 최종적으로 klue 기반의 tokenizer 를 통한 BM25 의 성능이 Top-k=20 일때 90%에 달하는 성능을 도달했지만, 정작 미리 학습된 reader 모델을 통해 inference 할 때는 passage 의 개수가 너무 많아서 오히려 정확한 passage 가 담겨 있지 않을 확률이 높은 Top-k=10 일 때 inference 성능이 더 나은 것을 확인하고, Rerank 를 통한 passage 개수를 줄이는 것을 목표로 삼았다.
    f. Re2G 라는 논문에 DPR 과 BM25 를 통해 각각 10 개씩 뽑고, 총 20 개를 새로운 Rerank 통해 다시 나열하는 과정을 가져와서 Rerank 를 시도하다가 프로젝트가 끝났다.
    g. '₩₩n' 처럼 context 에 들어있는 것을 확인하고 실제 답안에도 있는 걸 확인, 학습에 방해가 된다고 판단해서 많은 요소들을 없앴지만 너무 늦어 프로젝트 성능에는 기여하지 못했다.

전과 비교해서, 내가 새롭게 시도한 변화는 무엇이고, 어떤 효과가 있었는가?

- 해당하는 수업과 프로젝트에 관련되어 있는 논문을 읽고 정리했다.
    - 확실히 수업과 프로젝트로만 공부하는 것보다 훨씬 더 깊이 볼 수 있어서 이해가 깊어졌다.
- github 를 다루는 실력이 부족해서 불편하더라도 github 내에서 코드를 다루고, 개인 브랜치와 기능에 따른 브랜치를 구분해서 만들어 협업에 녹아들 수 있게 했다.
    - 이전에 부족했던 commit message 작성, 기능에 따른 branch 개설과 dev branch 를 활용해서 PR 과 issue 를 통한 협업을 제대로 익혔다.

마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

논문을 열심히 읽었지만 그것을 실제로 구현하기까지는 쉽지 않았다.

논문을 이해하기 위한 초석의 지식은 많은 부분 생긴 것 같지만, 아직 구현하는 능력은 빠르게 하기에는 부족하다. 논문을 분야별로 그 안에서 연대별로 정리해서 구현하기 쉽게 만드는 시간을 가져야겠다.

데이터를 정량적으로 평가하는 것 보다 human evaluation 을 못 해본 것이 아쉽다.

자연어를 처리하는 만큼, 정량적으로 평가하기 어려운 부분이 많다는 것을 인지했다. 다음에는 데이터를 찬찬히 뜯어보는 시간을 반드시 가져야겠다는 생각을 했다.

한계/교훈을 바탕으로 다음 프로젝트에서 시도해보고 싶은 점은 무엇인가?

- 좀 더 면밀하게 문제점과 가설을 설정해서 모든 과정을 논리적으로 남기고 싶다.
- 최신 논문을 읽고 구현하는 것을 시도해 보고 싶다.
- 모델을 개선하는 것 보다, 어떤 데이터가 있는 지 더 찬찬히 뜯어보고 생각을 정리해 보고 싶습니다.
- 저번 랩업 리포트에서도 적었지만 마지막 최종 프로젝트에서는 하루 단위로 내가 했던 것들을 기록해서 나를 객관적으로 확인을 하고 싶다.

최휘민

나는 내 학습목표를 달성하기 위해 무엇을 어떻게 했는가?

- 이번 프로젝트에서 최대한 최신 논문 기반으로 이전 기수에서 하지 않았던 새로운 시도를 도입해 보고 싶었다. 따라서 ODQA 와 관련한 논문들을 시간 순서대로 정리하고 다양한 논문을 읽었다.
- 프로젝트를 위해 읽은 논문은 ORQA, DPR, ColNERT, DensePhrases, Re2G 논문을 읽었고 팀원들과 이전의 기술과 새로운 기술의 차이점과 우리의 프로젝트에 적용할 수 있는 부분을 같이 논의하였다.
- Reader 모델의 성능 향상을 위해 Question Generation 방법을 시도하였다. Retrieval 부분은 BM25 의 Top-k 20 기준으로 90%이상이었기 때문에 최종 성능 향상은 Reader 모델 부분의 개선이 효율이 높아 보였다.
- Ensemble 방법으로 Hard Voting 을 구현하여 최종 성능을 높였습니다.

전과 비교해서, 내가 새롭게 시도한 변화는 무엇이고, 어떤 효과가 있었는가?

- 기존에도 다양한 방법을 시도해 보았다고 생각하지만 논문기반의 구현이 아니었거나 최신 논문이 아닌 2 년전 녹화한 강의에서 소개한 방법을 시도해 보았던 것 같다. 그래서 이번에는 강의에서 DPR 을 소개하였기 때문에 그 이후에 나온 논문을 읽고 구현을 시도하였고 새로운 논문에서 언급한 기존의 논문의 문제점 등을 논의하여 기존 방법론의 개선에 집중하였습니다.
- DensePhrases 를 공개된 Github Code 와 논문을 보고 구현을 시도해 보았습니다. 이전에는 공개된 방법론을 구현할 때 단순히 다른 사람이 구현한 코드를 검색해 따라 하는 느낌이였다면 이번에는 공개된 Github Code 를 분석하며 구현하려고 시도하였습니다. 논문에서 수식적으로 어려웠던 부분이 코드 구현 방법을 보며 구조를 이해할 수 있었습니다.
- Question Generation 을 시도하면서 결과물을 직접 보며 평가하고 팀원에게 문제점을 공유하였습니다. 이전에는 단순히 혼자 추측하여 문제점을 정의하고 개선하려고 시도하였습니다. 최종적으로 문제를 해결할 수 있지는 않았지만 문제 해결을 위한 좋은 방향이었다고 생각했다.

마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

- DensePhrases 구현을 시도하며 시간을 낭비했다고 생각한다. 구현을 시도하며 Github Code 와 논문을 끝임없이 읽고 시도했지만 결국 구현하지는 못했다. 가장 아쉬운점은 구현을 못한 것이 아니고 무작정 구현을 시도했던 것 같다. 어떻게 보면 실험적인 시도였는데 어느정도의 시간을 투자할지 정하고 진행했어야 했다고 생각한다. 결국 시간을 낭비하여 이전에 계속했던 결과 분석을 소홀히 했던 것 같다.
- Question Generation 의 시도가 실패하였는데 결과 분석을 잘못했던 것이 아쉬웠다. 단순히 원하는 결과가 안나온 것이 학습데이터의 부족이라 생각하고 외부데이터를 계속 증가하는 방법만 사용하였다. 어느정도 진행하고 다시 문제정의를 했어야 했다고 생각한다. 결국 너무 늦게 모델의 변경을 생각하여 개선할 시간이 부족했다.

한계/교훈을 바탕으로 다음 프로젝트에서 시도해보고 싶은 점은 무엇인가?

- 다음 프로젝트에서는 일정을 철저히 정하고 시작하고 싶다. 단순히 한 업무가 끝나면 다음 업무를 하는 것이 아닌 실험에도 시간을 얼마나 투자할지 정하는 것이 좋다고 이번 프로젝트에서 생각했다.
- 문제를 정의하는 것을 좀 더 철저히 한다고 생각한다. 귀찮을 수 있지만 문제 정의를 자주 하고 팀원과 자주 논의했으면 더 좋은 결과를 얻을 수 있었을 것이다.

황윤기

효율적인 학습을 위한 노력

- 베이스라인 코드의 빠른 이해와 분석 및 방법론 조사
- 해당 Task 를 해결하기 위한 논문 스터디 주도
- 논문에서 제시한 방법론 모델 구현
- 효율적인 학습 파이프라인 구성(DPR 모델 BiEncoder Trainer)
- 여러 방법론의 결합

성능 개선을 위해 시도해본 점

- 먼저 여러 논문들(DPR, ColBERT, ORQA, DensePhrase)을 리서치해보고, 가장 구조가 간단하고, 코드로의 구현이 복잡하지않고, 논리적으로 옳아보이는 시도를 우선으로 선택한 결과 Retrieval Model 로 DPR(Desne Passage Retrieval) 방법론을 결정했다.

DPR

- DPR 모델은 Query Encoder 와 Passage Encoder 를 개별적으로 구성함으로써, 각자 모델이 질문과, 주어진 문서에 대한 이해를 바탕으로, 연관성을 잘 도출해낼 것으로 생각하였다.
- 논문에서는 상당히 좋은 결과를 내고 있었지만, 같은 성능을 내기에 우려됐던 부분이 2 개 존재하였다.
    - 학습에 사용되는 데이터셋의 Batch Size 와 In-Batch Negative Sample 의 크기를 상당히 크게 가져간 것.
    - SQuAD 데이터셋에 대해서, BM25 보다 좋은 성능을 내지 못했던 것.
- 첫번째 문제점에 대해선, 우리가 조절할 수 있는 부분이 거의 없었기에, Batch Size 를 메모리한도 내애서 가장 크게 가져갈 수 밖에 없었다.
    - Reader 를 "Klue/roberta-large" 모델로 고정, DPR 모델을 여러조건으로 학습시켰을 때 의 결과.

	Batch Size	Hard Negative Num.	EM	F1
DPR Retrieval	30	1	36.67	46.48
DPR Retrieval	12	4	28.75	37.72

    - 다른 것은 조정하지 않고, BatchSize 를 가장 키우고, Negative Num 을 1 개로 가져갔을 때가, DPR 단일 모델의 결과는 가장 높게 나왔다.
    - Hard Negative 의 개수를 많이 가져가면, 일반화성능이 높아질 것이라고 생각했지만, 그렇지만은 않았다.
- 두번째 문제점에 대해선, BM25 만을 단일로 사용했을 때 실험을 통해 결과를 알아보았다.

	Top-K	EM	F1
BM25	20	54.58	65.57
Single DPR Model(Best)	-	36.67	46.48

    - 실험 결과와 같이 BM25 의 성능의 월등히 좋게 나왔다.
    - 이 결과를 바탕으로 몇가지 생각들을 정리했다.
        - 정답이 구문에 주어져 있기 때문에 어휘적인 유사도가 더 좋은 결과를 내는 것이다. 그렇기 때문에 BM25 가 좋은 성능을 내는 것이다.
        - DPR 에 사용하는 인코더의 성능이 부족했나?

- 위와 같이 두가지 생각을 바탕으로 두가지에 대해 모두 실험을 진행했다.
  - 첫번째는 BM25 의 결과를 바탕으로 기학습시켜둔 모델을 이용해서, 다시 순위를 매겨주는 작업이었다.(ReRank)
  - 두번째는 DPR 인코더를 더 큰 기학습모델로 교체하는 것이었다. 해당 내용은 Batch Size 를 너무 작게 가져감으로 인해 성능이 잘 나오지 않아서 아래 서술에선 생략한다.

ReRank

- 먼저 BM25의 단일 성능이 좋았기에, BM25 임베딩을 이용해서 Top-50개의 Query에 대한 Passage들의 랭크를 나열했다.
- Query 당 50 개의 Passage 들을 학습시켜둔 DPR 모델을 이용해서, 다시 점수를 계산하고, Top-20 개의 Passage 를 추출한다.
- 그 뒤에 Reader 모델에 넘겨줌으로써, Span 을 예측할 수 있도록 구성한다.
- BM25 의 어휘적으로 유사도가 높은 Passage 들을 먼저 필터링하고, DPR 모델로 한번 더 점수를 내준다면, 두 모델의 장점들을 서로 보완할 수 있을 것이라 생각했다.

	Top-K	Batch Size	Hard Negative	EM	F1
$DPR_{(1)}$	-	30	1	36.67	46.48
$BM25_{(2)}$	20	-	-	54.58	65.57
$BM25_{(2)}$ + $DPR_{(1)}$	20	-	-	57.08	68.29

- 내가 생각했던 논리대로 예측을 진행해본 결과, 일반 BM25 만 단일로 사용했을 때보다 EM 성능이 3 점정도 올랐다.

아쉬웠던 점

- 우리의 데이터셋에 대한 이해가 많이 부족했다. 데이터를 정량적인 분석은 물론, 직접 살펴보며 분석을 진행했어야 하지만, 정량적인 분석에 그쳐서 데이터가 실제 어떤 형식으로 구성되었는지 파악하질 못했다. 그에 따라 DPR 에서 SQuAD 데이터셋에서 성능이 나오지 않았다는 점에서 의심을 품고 시간을 너무 많이 소비하지 말았어야 했는데, DPR 을 개선하기 위해서 너무 많은 시간을 들였다. 이번 대회를 계기로 데이터셋에 대해서 정확한 이해와, 논문의 저자가 미리 실험해봤던 결과를 바탕으로 시도해볼 것과 안할 것을 명확히 구분해야한다는 것을 깨닫게 되었다.
- 두번째 아쉬운 점은 위의 내용과 맥락이 이어진다. 모델의 성능이 예상보다 안나온다면, 빠르게 다른 시도를 하기 위해 피보팅을 선택해야하는데, 잘 해내질 못한 것 같다. 조금만 더 수정하면 좋은 결과를 나올 것 같다는 막연한 희망과 구현한 내용들의 아쉬움으로 인해 시간 낭비를 범하였다.

다음 프로젝트에는...

- 프로젝트를 진행하다보면, 데이터 분석이 중요성보다 모델 구조의 개선에서 상당히 큰 점수 향상이 있었다보니 데이터 분석을 소홀히 진행한 점을 보완해야겠다.
- 이를 보완하기 위해, 아래와 같이 나만의 학습 단계를 정의하고 따라야 한다.
  - 데이터 분석 -> 실제 데이터 관찰 -> 분포나 성질이 비슷한 다른 데이터 서치
  - 기존재 하는 데이터셋에 대해 좋은 결과를 낸 모델 구조 리서치
  - 모델 구현 및 개선
  - 결과 분석 및 보완점 탐색 / 다시 반복

- 또한 기록의 중요성은 항상 느끼듯이, 내가 당시 경험했던 내용들을 바로 정리할 필요가 있다. 너무 바쁘더라도 주말에는 Summary 를 하는 시간을 가져야 한다.