Wrap up

▼ 목차

아쉬웠던 점들
한계/교훈을 바탕으로 다음 스테이지에서 새롭게 시도해볼 것

4주에 걸친 Stage 3 - MRC 대회가 끝이 났다

최종 리더보드에서는 조금 아쉬웠지만, 많은 것을 시도하면서 성장했다고 생각한다

💡 최종 제출 내용

최종 순위

- Public LB - 6 / 9

  EM : 59.58%, F1 : 72.27%

- Private LB - 7 / 9

  EM : 58.61%, F1 : 69.08%

최종 제출 모델

Model Architecture & Hyper parameters

MRC

- Koelectra-base-v3-finetuned-korquad (https://huggingface.co/monologg/koelectra-base-v3-finetuned-korquad)
- learning rate : 1e-4
- epoch : 1

  ⇒ epoch을 늘리면 학습이 더 안되는 현상이 발생하여 epoch 조정

  ⇒ validation 240개의 데이터에 대해 EM : 60%, F1 : 70%대의 score 형성

Retrieval

- Elastic search의 BM25 사용
- 가장 점수가 높은 문서를 기준으로 특정 threshold값을 넘는 문서를 모두 candidates로 생각

  ex) 가장 점수가 높은 문서 BM25값: 20점. Threshold 80%면 16점 이상인 모든 문서 사용

  최종 제출 모델은 threshold 0.7로 설정

Post processing

- 단어 Ban list

  ["이따금","아마","절대로","무조건","한때","대략","오직","오로지","감히","최소","아예","반드시","꼭","때때로","이미", "심지어",,"종종","졸곧","약간","기꺼이", "비록","꾸준히","일부러","어쩔", "문득", "어쨌든", "순전히", "필수","자칫", "다소", "간혹", "적어도", "왜냐하면", "아무래도"]

- Start logit 상위 20개, End logit 상위 20개 ⇒ 400개 조합에서 해당 단어를 포함하는 후보는 삭제하는 파트 추가

Score

- $\sigma(Score_{start}) * \sigma(Score_{end})$

의견

- 마지막날 성능을 많이 올렸던 최종 제출 모델
- 시간이 있으면 조금 더 향상된 모델이 나오지 않을까 하는 아쉬움을 피어세션에서 많이 나누었다
- 모델 측면에서 learning rate만 나와 다른데, 저것보다 더 좋은 모델을 사용할 수 있을 것 같다
- Retrieval에서 threshold에 따라 추출한 context의 개수가 다양한데, 적게는 1개부터 많게는 30개 이상의 문서가 포함되는 경우가 있다

  ⇒ 이런 경우에 Dense embedding이 적용되면 좋을 것 같은데, 그러지 못한 아쉬움

💡 피어세션 진행 방식

협업 방식

- 각자 파트를 분배해서 나누는 분업 방식이 아닌, 각자 전체적인 ODQA의 task(retrieval, reader)를 모두 수행
- 그렇게 해서 잘 되는 내용을 피어세션에서 공유

장점

- 모든 피어들이 전체적인 부분에 대해서 내용을 숙지하고 있다

단점

- 실험 관리가 어렵고, 코드 단에서 통일하기가 어렵다

- 시간이나 효율성에서 떨어진다 (같은 실험을 각자 모두 수행한다던가)

💡기술적인 도전 + 시도했으나 잘 되지 않았던 것들

Retrieval

1. 다양한 retrieval 방식 적용
    - 문제 제기 : MRC 모델의 성능이 1.0이어도, retrieval의 성능이 0.1이면, 최종 모델은 0.1짜리

      ⇒ retrieval의 성능을 올리는 것이 중요하다, 다양한 방식을 찾아보자
    - Sparse embedding : BM25 (https://pypi.org/project/rank-bm25/)
    - Open source : Elastic Search

    성능 비교 (train + validation의 4192개의 데이터에서 ground truth context가 top-k개의 context에 포함될 확률)

Aa 종류	Top-1	Top-3	Top-5	Top-10	Top-20
TF-IDF (baseline)	17.9		40.5	50.3	58.8
TF-IDF (n-gram max feature 해제)	41.0		73.5	80.9	86.5
BM-25 (직접 짜본거)	26.1		49.4	56.7	63.6
BM-25 (Kapi)	54.2	78.5	83.0	86.6	89.0
BM-25 (도윤님 전처리)	60.6 (→ 70)	80.0	84.4	87.2	90.0
Elastic search (No tuning)	57.1	81.3	86.0	88.9	91.2

    - 그 밖에 BM-25 reranking(1000 → 3) 방식도 사용하였는데, 성능 지표가 날아가서 표에는 추가하지 않음
    - 시간은 오래 걸리는데 비해, 성능은 기존 BM-25보다 하락해서 사용하지 않음
2. 전처리된 문서를 Elastic에 적용
    - 문제 제기 : 도윤님이 전처리 하신 분서를 BM-25에 활용했을 때, 성능 향상이 있었는데, Elastic search에 적용하면 성과가 있지 않을까?

- 전처리 내용
  - Stopword 적용
  - 전처리된 wiki data 적용
  - Query에서 질문에 특정적인 용어를 삭제하고, 형태소 단위에서 특정 품사를 제거하는 방식
- 결과
  - Stopword 적용

    ⇒ 성능의 급격한 하락 (Top 1 : 40%), elastic 자체에 stopword (`__korean__`) 이 들어가 있고, stopword들이 더 작은 단위로 쪼개져서 정상적으로 동작이 안되는 것 같음
    (그냥 구현을 못한거일수도)
  - 전처리된 wiki data 적용

    ⇒ 약간의 성능 향상
  - Query에서 질문에 특정적인 용어를 삭제하고, 형태소 단위에서 특정 품사를 제거하는 방식

    ⇒ 약간의 성능 향상

3. Elastic search tuning
   - Elastic search guide : https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html
   - Elastic 가이드북 : https://esbook.kimjmin.net/

   을 참고하여 elastic tuning을 진행
   - ~~완건드리는게 가장 좋았다~~
4. Dense embedding
   - 다른 캠퍼님들이 맡아서 진행한 부분인데, Top-1 기준으로 40%정도의 성능

     ⇒ 잘 안되는 이유는 데이터의 특성 때문일 수도 있고, 데이터의 개수 때문일 수도 있음

     ⇒ 논문에서는 큰 batch size로 학습시켰다고 했는데, 그러지 못한 원인일 수도 있음
   - 다만, 이 성능은 전체 wiki data가 아닌, batch에서의 성능이기 때문에 실제로는 더 떨어질 것으로 생각하고 적용하지 않음

- 다른 조에서 dense를 활용한 방법을 들었을 때, 성능이 좋지 못해도 Sparse와 가중 결합을 사용하면 sparse보다 더 향상된 성능을 얻었다고 하는것으로 봐서, 우리도 적용하면 어떤 결과가 나올까 궁금해지는 방법
- 그리고, 최종 모델에서 threshold를 기준으로 나누는데, 이 잘린 문서의 개수가 일정 개수 이상이면 sparse로 찾을 수 없다고 판단하고, dense를 사용하는 방법도 좋을 것이라고 생각

MRC(Reader)

3주차 내내 매달렸던 부분,,,

⇒ Retriever가 sparse만으로 어느 정도 성능이 나오는데, 모델의 EM : 62, F1 : 71에 upper bound가 걸려있던 상황

Backbone model

- XLM-Roberta-large (multilingual)
  - 학습이 오래 걸리지만, 성능이 Koelectra보다 좋은 편
  - 다만, EM과 F1 Score의 차이가 조금 났는데, multilingual tokenizer다보니, 조사를 떼주는 디테일에서 아쉬움이 있었음
- Koelectra : https://huggingface.co/monologg/koelectra-base-v3-discriminator
- Koelectra-finetuned-korquad : https://huggingface.co/monologg/koelectra-base-v3-finetuned-korquad

다양한 입력

Baseline에서는 Query + Context로 학습

1. Query + Title + Context
   - 문제 제기 : EDA결과, query에 대한 답이 context에 없던 경우 존재, title에는 존재
     - Span prediction 방식에서는 context에서 답을 추출하는 것인데, 없으면 답을 못찾는다
   - 해결 방법 : Title도 붙여서 학습을 해보자!
   - 결과 : 성능의 향상은 없었다
     - https://wandb.ai/gyjeong/huggingface?workspace=user-gyjeong 에서 볼 수 있듯이, EM : 60.4정도의 분포로, maximum EM인 62에 미치지 못함

- 실패 이유 : 나도 모르겠다,,, 아마 title이 noise인 context가 몇개 있지 않았을까 하는 생각

2. Query + Context (Ground truth) + Negative sample
    - 문제 제기 : 모델을 더 어렵게 학습해보자
    - 해결 방법 : Negative sample context를 이어붙여서 train
    - Negative sample을 뽑는 근거는, Ground truth는 아니지만 BM-25 기준 상위 2개 context
    - 결과 : 성능의 향상은 없었다
    - 1번과 마찬가지로 EM : 60정도의 분포
    - 실패 이유 : 이것도 모르겠다,,, 아마 negative sample을 붙이면 max token length가 여러 개로 들어가는데, 그럼 그 부분은 통채로 오답이 학습되는 것이어서 그런 듯

모델 구조의 변화

Baseline에서는 1 Linear layer를 사용 (768 → 2)

- 문제 제기 : 너무 간단한 방식이기 때문에 쉽게 오버피팅 될 가능성과 제대로 된 결과를 내지 못한다는 가정

1. 2 Linear layer
    - Linear + Dropout(0.7) + Linear
    - 기존 방식에서는 학습이 불안정한 측면이 있었는데(중간에 튀거나 하는 경우 + 금방 overfitting 되는 경우)
      이 경우에는 그런 현상은 발생하지 않았음
    - https://wandb.ai/gyjeong/huggingface/runs/32mrz8t1?workspace=user-gyjeong
    - EM이 62점대 정도로 형성되는데, 1개의 layer와 비슷하게 나타남
2. Bi-GRU(LSTM) ⇒ 도윤님이 시도
    - 1번보다 더 복잡한 구조
    - Bidirection을 태워서 나온 output을 linear에 태워서 사용

Loss의 변화 ⇒ 도윤님이 시도

- Trainer의 loss는 `CrossEntropyLoss`였는데, `CrossEntropyLossDynamic` 적용
- 성능의 향상 : Public LB 기준, EM 51% → 55%

Inference

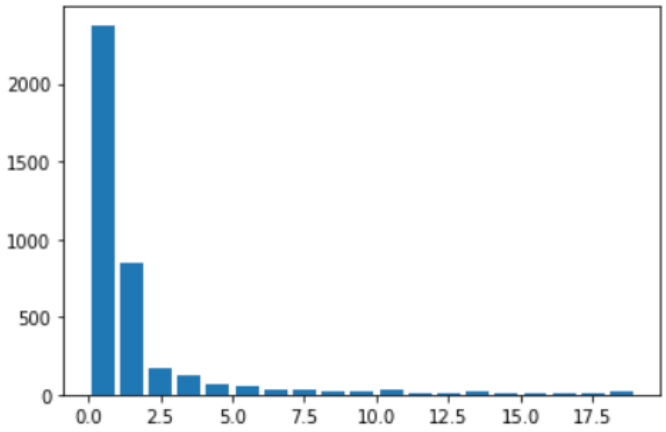
- Top-k개의 context를 어떻게 활용할까?
- 같은 모델을 inference할 때, 여러 개의 context를 concat(space 단위)하는 작업을 통해, 최적의 개수를 찾을 때는,

Koelectra-korquad backbone, EM : 62, F1 : 71 모델 기준

Aa Metric	Top-1	Top-3	Top-5	Top-10	Top-20
EM	45.42%	50.00%	49.17%	48.33%	47.08%
F1	59.00%	62.65%	62.45%	61.48%	60.82%

- 로 확인했을 때, Top-3개의 context를 concat하는 것이 가장 좋게 나타났다

  ⇒ 너무 길게 주면 오답 context가 들어가 있을 확률이 큰 것으로 나타남



Ensemble

Soft voting

- `nbest_prediction.json`에서 뽑힌 확률을 더하는 방식
- EM 55%가 나온 두 파일에서 확률을 더하는 방식으로 진행했는데,
- 큰 성능 향상은 없었다
  - 각 nbest_prediction의 확률 분포가 달라 단순 합을으로 진행했을 때, 한쪽 결과가 묻히는 현상 발생
  - 하나의 파일은 sigmoid로 전체적으로 균등한 확률 분포, 다른 하나의 파일은 균등하지 않은 확률 분포

Hard voting

- `prediction.json` 에서 뽑힌 최종 답을 counting해서 maximum을 구하는 방식
- 4개의 파일 중, 가장 잘나온 파일에 가중치를 두고 진행
- 마지막 제출을 시도할 때, EM : 59.58% F1 : 71.82%의 성능
- 이 성능은 유출된 답 24개를 적용한 것이 아니기 때문에, public lb를 기준으로 보았을 때는, 최종 제출 모델보다 성능이 더 좋을 가능성이 높음

💡 학습과정에서의 교훈

- 이전 Stage 1, 2과는 다르게 난이도가 많이 높아졌고, 팀간 competition이다보니 이전과 분위기가 조금 달랐던 것 같다
- 데이터를 다루는 것의 중요성을 알게 되었다. 다른 훌륭한 팀들의 발표를 들으니, 데이터를 가공하는 데에서 많은 노력을 들인 것을 알 수 있었다

  KorQuAD와 대회 데이터를 합쳐서 학습을 시켰을 때, 오히려 validation score가 떨어지는 것을 보고, 대회 데이터를 가공하는 방법을 시도해보았어야 했는데, 그러지 못해서 아쉽다
- 실험 로그를 팀 단위로 정리하는 것이 중요하다고 느꼈다
- WANDB로 실행 로그를 기록하다 보니 한눈에 비교하기 너무 수월했다

  https://wandb.ai/gyjeong/huggingface?workspace=user-gyjeong
- 멘토링 과정에서 멘토님께서 어떤 것을 시도할 때, 왜 이 방법이 잘 될 것 같은지 생각해보고 적용해보라는 말씀을 많이 하셨는데,

  그렇게 생각하는 과정의 중요성에 대해 알게되었다
- MRC는 알고있었는데, retrieval이 포함된 ODQA task는 처음 대해보았다.

  재미있는 대회였고, 많은 것을 알게 되어 좋은 경험이라고 생각한다.

💡 마주한 한계와 도전 숙제

아쉬웠던 점들

- MRC 모델의 성능을 올리기 위해 정체된 기간이 조금 길었는데, 여기서 너무 오래 묶여있었던 것이 아쉽다

- 모델의 성능을 validation EM, F1으로 측정했는데, 240개의 데이터에 모델의 성능 전체를 의존하는 것이 좋지 않은 방법이라고 생각한다
- 팀별 competition은 처음이다 보니 분업에 대해 많은 생각을 하지 못했는데, 분업을 하면 이번 대회보다 효율적인 실험 관리가 이루어 질 수 있다고 생각

한계/교훈을 바탕으로 다음 스테이지에서 새롭게 시도해볼 것

- 팀별 분업과 코드 단위 협업
- Baseline을 새로 짜보기
- 논문을 찾아보고 구현해보기
