Project 2
Machine Reading Comprehension

Team 5
박석훈, 고명준, 손기훈, 서경원

목차

1. 프로젝트 개요

[프로젝트 목표]
1. “성능”

[성과 달성 지표]
편집 거리(Levenshtein Distance)

[개발 환경]
Google Colab Pro + (언어: Python)

2. 프로젝트 팀 구성 및 역할

훈련생	담당 업무
박석훈(팀장)	- 프로젝트 총괄, 결과 보고서 작성
고명준	- 코드 작성, 디버깅
서경원	- 논문 리뷰, 자료 조사
손기훈	- 디버깅, 자료 조사

* 담당 업무는 주요 담당 업무이며, 전원 각 업무에 참여

3. 프로젝트 진행 프로세스

구분	기간	활동	비고
팀 빌딩, 업무 분담	2/14(월)	착수 회의, 역할 분담	
베이스라인 코드 리뷰	2/14(월)~2/16(수)	베이스라인 코드 확인	각 역할 전원 참여
디버깅	2/16(수)~2/24(목)	파라미터 조정, 모델 변경	각 역할 전원 참여
결과 정리, 보고서 작성	2/24(목)~2/25(금)	결과 정리, 보고서 작성	각 역할 전원 참여

ELECTRA(Efficiently learning an Encoder that Classifies Token Replacements Accurately)

- 기존 MLM(Masked Language Modeling) 태스크를 통해서 학습하는 방식의 문제점

  1. 전체 토큰 중 15%에 대해서만 loss가 발생하며, 비용이 증가

  2. 학습 때는 Mask 토큰을 모델이 참고하여 예측하지만, 실제로는 Mask 토큰이 존재하지 않음

- 대안: RTD(Replaced Token Detection)
- 실제 입력의 일부 토큰을 가짜 토큰으로 바꾸고 각 토큰이 실제 입력에 있는 진짜 토큰인지, Generator가 생성한 가짜 토큰인지를 맞추는 이진 분류 문제

ELECTRA(Efficiently learning an Encoder that Classifies Token Replacements Accurately)

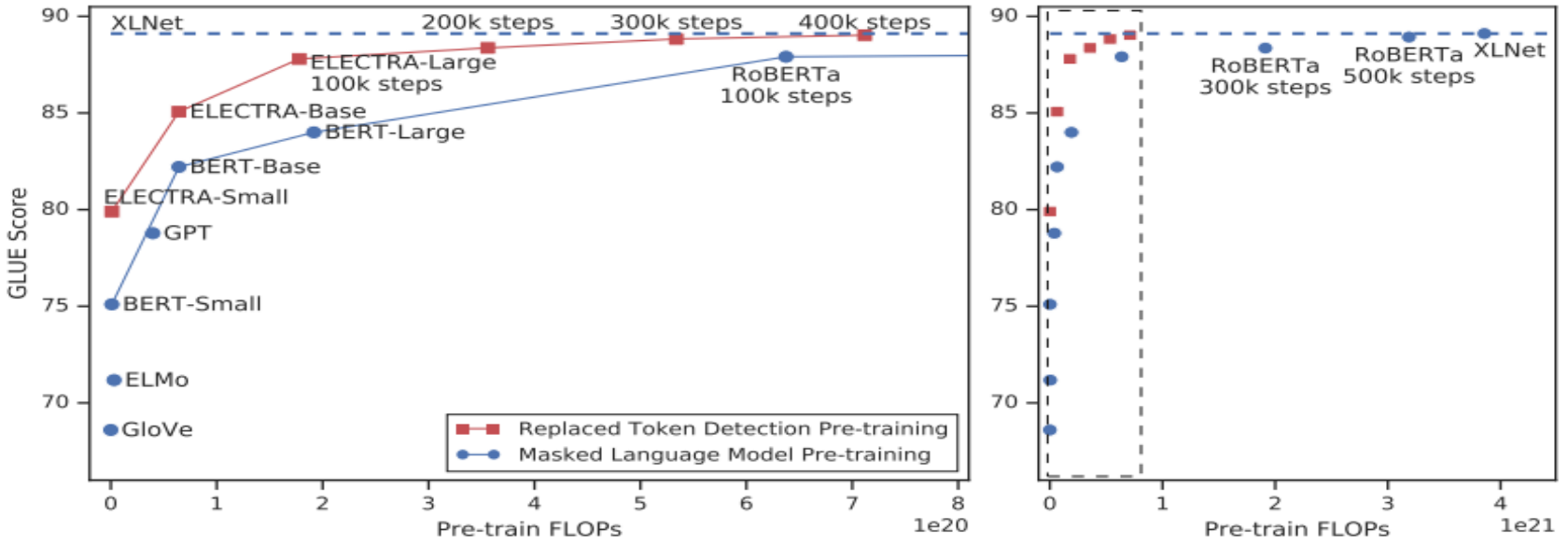

Figure 1: Replaced token detection pre-training consistently outperforms masked language model pre-training given the same compute budget. The left figure is a zoomed-in view of the dashed box.

ELECTRA: Pre-training Text Encoders as Discriminators Rather than Generators

https://openreview.net/pdf?id=r1xMH1BtvB

ELECTRA(Efficiently learning an Encoder that Classifies Token Replacements Accurately)

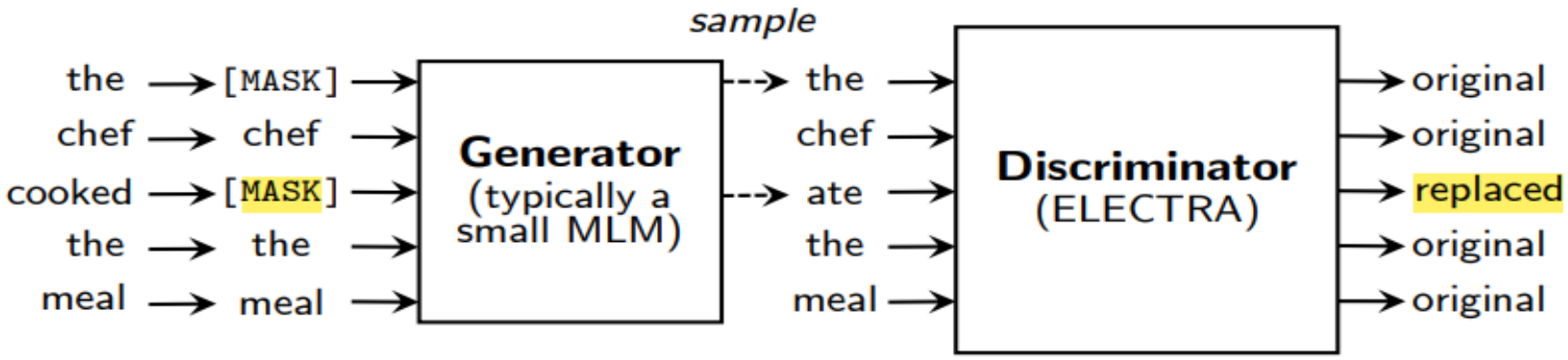

Figure 2: An overview of replaced token detection. The generator can be any model that produces an output distribution over tokens, but we usually use a small masked language model that is trained jointly with the discriminator. Although the models are structured like in a GAN, we train the generator with maximum likelihood rather than adversarially due to the difficulty of applying GANs to text. After pre-training, we throw out the generator and only fine-tune the discriminator (the ELECTRA model) on downstream tasks.

Koelectra(Pretrained ELECTRA Model for Korean)

- 기존 한국어 PLM 모델의 문제점
- - KoBERT: Vocab Size가 상대적으로 작음
- - HanBERT: Ubuntu 환경에서만 사용 가능
- - KorBERT: API 신청 등 과정 필요

- 목표
- - Vocab Size가 어느 정도 커야 함
- - 모든 OS에서 사용 가능
- - Tokenization 파일을 만들 필요 없이 곧바로 사용 가능
- - 성능

Code

```python
# 가장 짧은 답 하나만 사용
if answers is not None:
    # 들어가야 할 세 개의 토큰과 질문을 제외한 길이
    min_len = MAX_LENGTH - 3 - len(question)
    min_idx = 0
    for idx, answer in enumerate(answers):
        if len(answer["text"]) < min_len:
            min_idx = idx

    answer = answers[min_idx]
    answers = [answer]
```

Class KoMRC 부분에 추가

여러 답 중 가장 짧은 답 하나만 사용
오답이 나도 길이가 길게 생성되지 않도록 설정

Code

```python
if morph == "[UNK]":  # [UNK]의 앞쪽 위치 저장
    if sentence[position] == " ":
        position += 1
    tokens.append((token, (position, position + 1)))
    position += 1
else:
    position = sentence.find(morph, position)
    tokens.append((token, (position, position + len(morph))))
    position += len(morph)

# [UNK]의 뒤쪽 위치 저장
if idx > 0 and tokens[idx - 1][0] == "[UNK]":
    tokens[idx - 1] = list(tokens[idx - 1])
    if idx == 1:  # 첫번째 토큰이 [UNK]일 때
        if tokens[idx][0] == "[UNK]":
            tokens[idx - 1][1] = (0, position - 1)
        else:
            tokens[idx - 1][1] = (0, position - len(morph))
    else:
        # 두 번 연속 [UNK]가 나왔을 때
        if tokens[idx][0] == "[UNK]":
            tokens[idx - 1][1] = (tokens[idx - 1][1][0], position - 1)
        else:
            # 현재 토큰과 [UNK] 사이에 공백이 있을 때
            if sentence[tokens[idx][1][0] - 1] == " ":
                tokens[idx - 1][1] = (
                    tokens[idx - 1][1][0],
                    tokens[idx][1][0] - 1,
                )

                tokens[idx - 1][0] = sentence[
                    tokens[idx - 1][1][0] : tokens[idx - 1][1][1]
                ]
            # 현재 토큰과 [UNK] 사이에 공백이 없을 때
            else:
                tokens[idx - 1][1] = (
                    tokens[idx - 1][1][0],
                    tokens[idx][1][0],
                )
                tokens[idx - 1][0] = (
                    "##"
                    + sentence[
                        tokens[idx - 1][1][0] : tokens[idx - 1][1][1]
                    ]
                )
        tokens[idx - 1] = tuple(tokens[idx - 1])

# [UNK]가 마지막에 있을 때
if tokens[-1][0] == "[UNK]":
    tokens[-1] = list(tokens[-1])
    tokens[-1][1] = (tokens[-1][1][0], len(sentence))
    tokens[-1][0] = sentence[tokens[-1][1][0] :]
    tokens[-1] = tuple(tokens[-1])
```

Code

```python
# 답이 주어진 경우
if sample["answers"] is not None:
    max_seq_len = MAX_LENGTH - len(question) - 3
    answer = sample["answers"][0]

    # context가 최대 길이를 넘어가면 잘라준다.
    if len(context) > max_seq_len:
        # 답의 위치를 중간에 두고 자름
        start_token = (
            0
            if answer["start"] < max_seq_len // 2
            else answer["start"] - max_seq_len // 2
        )
        end_token = (
            answer["start"] + max_seq_len // 2
            if answer["start"] + max_seq_len // 2 < len(context)
            else len(context)
        )
        new_context = context[start_token:end_token]

        # 답의 위치도 잘린 context에 맞춰서 변경
        start = answer["start"] - start_token + len(question) + 2
        end = answer["end"] - start_token + len(question) + 2

        input_ids = (
            [self.cls_id]
            + question
            + [self.sep_id]
            + new_context
            + [self.sep_id]
        )
        # electra 모델은 중간에 있는 [sep] 토큰까지 0으로 취급
        token_type_ids = [0] * (len(question) + 2) + [1] * (
            len(new_context) + 1
        )
```

Indexer

Context가 최대 길이를 넘어갈 경우, 답을 중간에 두고 자름

Koelectra는 질문 - context 순서로 들어가며, 첫 [sep] 토큰까지 질문으로 포함

Code

```
# context에서 제외할 토큰은 맨 뒤에 있는 sep 토큰 하나
start_prob = start_logits[token_type_ids.bool()][:-1].softmax(-1)
end_prob = end_logits[token_type_ids.bool()][:-1].softmax(-1)
probability = torch.triu(start_prob[:, None] @ end_prob[None, :])
index = torch.argmax(probability).item()
```

첫 번째 [sep] 토큰이 질문에 해당하므로 softmax할 범위를 변경

```
29 torch.manual_seed(42)
30 model = ElectraForQuestionAnswering.from_pretrained(MODEL)
31 device = "cuda" if torch.cuda.is_available() else "cpu"
32 model.to(device)
33 optimizer = torch.optim.AdamW(model.parameters(), LEARNING_RATE)
34 total_steps = len(train_loader) * EPOCH
35 scheduler = get_linear_schedule_with_warmup(
36     optimizer, num_warmup_steps=total_steps // 10, num_training_steps=total_steps
37 )
38
```

Scheduler 추가