Natural Language Processing

언어 모델: BERT 파생 모델

강사 양석환

BERT의 문제점과 파생 모델

BERT의 문제점과 BERT 파생 모델

- 수많은 변수로 구성되어 있다.
  - BERT-base의 경우 약 1억1천만 개의 변수로 구성

    → 모델의 학습이 어렵고, 추론 시 많은 시간이 소요됨
  - 모델의 크기가 커지면 성능은 향상되지만 계산 시 리소스의 제한 발생
- 해결을 위하여 다양한 BERT 파생형 등장
  - 프로세스를 변경한 파생모델: RoBERTa, ALBERT, ELECTRA, SpanBERT 등
  - 지식 증류 기반의 파생모델: DistilBERT, TinyBERT 등
  - 특수 파생모델: sentence-BERT, domain-BERT(ClinicalBERT, BioBERT) 등

RoBERTa

RoBERTa

- **RoBERTa 란?**
  - **Robustly Optimized BERT pre-training Approach**
    - 강력하게 최적화된 BERT 사전 학습 접근 방식
  - **BERT 파생 모델 중에서 가장 많이 사용되는 방법(특히 최신 시스템에서)**
  - **BERT와 유사하지만 사전 학습 단계에서 차이가 있음**

발표된 논문(2019)
RoBERTa: A Robustly Optimized BERT Pretraining Approach
(https://arxiv.org/pdf/1907.11692.pdf)

RoBERTa에 적용된 기법

- BERT는 충분히 학습되지 않은 모델임을 확인

  → BERT 모델을 사전 학습 시킬 때 사용할 수 있는 방법 제안

  - MLM 태스크에서 정적 마스킹이 아닌 동적 마스킹 방법 적용
  - NSP 태스크를 제거하고 MLM 태스크만 학습에 사용
  - 배치 크기를 증가시켜 학습 수행
  - 토크나이저로 BBPE를 사용함

- 사전 학습 시의 변경 사항을 제외하고는 RoBERTa는 기본적으로 BERT와 동일

RoBERTa에 적용된 기법

- 정적 마스크 대신 동적 마스크 사용
  - BERT의 MLM 태스크의 경우
    - 주어진 토큰을 15% 확률로 무작위로 마스크된 토큰으로 변경한 후
    - 모델에서 해당 토큰을 예측함
    - 예: We arrived at the airport in time
      - tokens = [[CLS], we, arrived, at, the, airport, in, time, [SEP]]
      - tokens = [[CLS], we, [MASK], at, the, airport, in, [MASK], [SEP]]
      - 토큰을 BERT 모델에 입력하여 마스크된 토큰을 예측하도록 학습
      - 이때, 마스킹은 데이터 전처리 단계에서 한 번만 수행, 에폭 별로 동일한 마스킹을 예측하도록 모델 학습이 이루어짐
        → 정적 마스킹

토크나이징 후 [CLS], [SEP] 추가

15% 확률로 무작위 마스킹

RoBERTa에 적용된 기법

- **RoBERTa의 경우**
  - 동적 마스킹 적용
  - 하나의 문장을 10개 복사
  - 10개의 문장에 대해 무작위로 15% 확률 마스킹 작업 수행 → 10개의 각기 다른 마스크 된 토큰을 가진 문장 확보

문장	토큰
문장 1	[[CLS], we, [MASK], at, the, airport, in, [MASK], [SEP]]
문장 2	[[CLS], we, arrived, [MASK], the, [MASK], in, time, [SEP]]
...	
문장 10	[[CLS], we, arrived, at, [MASK], airport, [MASK], time, [SEP]]

RoBERTa에 적용된 기법

- 모델을 에폭 40까지 학습 시킴. 이때 에폭 별로 다른 마스크가 적용된 문장을 입력
  - 문장 1의 경우, 에폭 1, 11, 21, 31에서만 사용됨
  - 문장 2의 경우, 에폭 2, 12, 22, 32에서만 사용됨
  - ...
- 이처럼 RoBERTa에서는 정적 마스킹 대신 동적 마스킹 방법을 모델 학습에 사용함

에폭	문장
에폭 1	문장 1
에폭 2	문장 2
...	...
에폭 10	문장 10
에폭 11	문장 1
에폭 12	문장 2
...	...
에폭 40	문장 10

RoBERTa에 적용된 기법

- **NSP 태스크 제거**
  - **여러 연구 결과, NSP 태스크가 BERT 모델의 사전 학습에 유용하지 않음을 확인 → 사전 학습 시 MLM 태스크만 사용**
  - **NSP 태스크의 작업 내용의 중요성을 이해하기 위한 실험**
    - SEGMENT-PAIR + NSP → NSP를 사용해 BERT 학습. 원래의 BERT 모델 학습방법과 유사
      → 입력은 512개 이하의 토큰 쌍으로 구성
    - SENTENCE-PAIR + NSP → NSP를 사용해 BERT 학습
      → 입력 값은 한 문서의 연속된 부분 또는 다른 문서에서 추출한 문장 쌍으로 구성. 입력은 512개 이하의 토큰 쌍으로 구성
    - FULL SENTENCES → NSP를 사용하지 않고 BERT 학습
      → 입력 값은 하나 이상의 문서에서 지속적으로 샘플링한 결과를 사용. 입력은 512개 이하의 토큰 쌍으로 구성
      → 하나의 문서 마지막까지 샘플링 후 다음 문서로 이음
    - DOC SENTENCES → NSP를 사용하지 않고 BERT 학습
      → FULL SENTENCE와 전체적으로 유사하지만 입력 값은 하나의 문서에서만 샘플링
      → 하나의 문서 마지막까지 샘플링 후 다음 문서는 사용하지 않음

RoBERTa에 적용된 기법

- 태스크별 모델의 점수 측정 결과(설정 별 BERT 성능)

모델	SQuAD 1.1/2.0	MNLI-m	SST-2	RACE
SEGMENT-PAIR	90.4/78.7	84.0	92.9	64.2
SENTENCE-PAIR	88.7/76.2	82.9	92.1	63.0
FULL-SENTENCES	90.4/79.1	84.7	92.5	64.8
DOC-SENTENCES	90.6/79.7	84.7	92.7	65.6

RoBERTa 모델의 학습

- 더 많은 데이터로 학습
  - BERT: 토론토 책 말뭉치 + 영어 위키피디아 데이터셋 → 16GB
  - RoBERTa: BERT용 데이터셋 + CC-News, Open WebText, Stories → 160GB
- 큰 배치 크기로 학습
  - BERT: 256개 배치로 100만 단계 동안 사전 학습 진행
  - RoBERTa: 8,000개 배치 크기로 30만 단계에 대하여 사전 학습
    (또는 8,000개 배치 크기로 50만 단계에 대하여 사전 학습)
  - 배치 크기를 키우면 → 학습 속도 및 모델 성능 향상

ALBERT 모델의 학습

- BBPE 토크나이저 사용
  - BERT: 워드피스 토크나이저 사용, 어휘사전의 크기는 30,000
  - RoBERTa: BBPE 토크나이저 사용, 어휘사전의 크기는 50,000

ALBERT

ALBERT (A Lite BERT)

- **ALBERT란?**
  - **BERT의 Light 버전**
  - **BERT가 가지는 수많은 변수를 줄여서 학습 시간과 추론 시간 감소**
    - 크로스 레이어 변수 공유(Cross-Layer Parameter Sharing)
    - 팩토라이즈 임베딩 레이어 변수화(Factorized Embedding Layer Parameter)

발표된 논문(2020)
ALBERT: A Lite Bert For Self-supervised Learning Of Language Representations
(https://arxiv.org/pdf/1909.11942.pdf)

ALBERT에 적용된 기법

- **크로스 레이어 변수 공유(Cross-Layer Parameter Sharing)**
  - **BERT: 동일한 형태를 가진 N개의 인코더로 구성됨**

    예: BERT-base는 12개의 인코더 레이어로 구성됨

    → 학습이 진행되면 인코더 레이어에 있는 모든 변수에 대하여 학습 수행
  - **크로스 레이어 변수 공유**
    - 모든 인코더 레이어의 변수를 학습시키는 것이 아니라
    - 첫 번째 인코더 레이어의 변수만 학습시킨 후
    - 첫 번째 인코더 레이어의 변수를 다른 모든 인코더 레이어와 공유함

ALBERT에 적용된 기법

- **계층 간 변수 공유 방법**
  - All-Shared: 첫 번째 인코더의 하위 레이어에 있는 모든 변수를 나머지 인코더와 공유
    - ALBERT의 기본 옵션으로 사용 중
  - Shared Feedforward Network: 첫 번째 인코더 레이어에서 피드포워드 네트워크의 변수만 다른 인코더 레이어의 피드 포워드 네트워크와 공유
  - Shared Attention: 첫 번째 인코더 레이어의 멀티 헤드 어텐션의 변수만 다른 인코더 레이어와 공유

ALBERT에 적용된 기법

- **팩토라이즈 임베딩 레이어 변수화**
  - **변수 감소(Parameter Reduction) 방법**
  - **BERT의 경우**
    - **워드피스 토크나이저 사용 → 임베딩 크기는 은닉레이어의 임베딩 크기와 같음**
      - **워드피스 임베딩**
        - 비맥락적(Non-Contextual) 표현
        - 원핫 인코딩(One-Hot Encoding)된 어휘 벡터에서 학습 수행
      - **은닉 레이어 임베딩**
        - 인코더를 통해 맥락적 의미를 가지는 형태로 학습 수행

ALBERT에 적용된 기법

- 어휘 사전 크기: V, 은닉 레이어 임베딩 크기: H, 워드피스 임베딩 크기: E
  - BERT의 어휘사전 크기는 30,000
  - 일반적으로 더 많은 정보를 은닉 레이어 임베딩으로 인코딩하기 위하여
    → 은닉 레이어의 임베딩 크기를 높게 설정함
  - BERT-base의 경우, 은닉 레이어 임베딩의 크기는 768
    → 은닉 레이어 임베딩 크기는 $\boldsymbol{V \times H = 30,000 \times 768}$
  - 워드피스 임베딩 크기(각 토큰의 인덱스 값을 임베딩한 경우)=사전 크기 × 임베딩 차원
    → 워드피스 임베딩 크기는 $\boldsymbol{V \times H = 30,000 \times 768}$
  - 따라서 은닉 레이어의 임베딩 크기 H를 키우면 워드피스 임베딩 크기인 E도 같이 증가함

ALBERT에 적용된 기법

- 워드피스 임베딩, 은닉 레이어 임베딩의 학습은 모델 학습이 진행될 때 수행되므로

  → “워드피스 임베딩 크기=은닉 레이어 임베딩 크기” 이면 학습할 변수도 증가

- 이런 문제점을 해결하기 위하여
  - 임베딩 행렬을 더 작은 행렬로 분해하는 방법인 “팩토라이즈 임베딩 변수화“ 적용

    → 워드피스 임베딩의 크기인 $V \times H$를 $V \times E$와 $E \times H$로 분해하여 적용

    - $V = 30000, E = 128, H = 768$ 이라고 한다면 $V \times H$는

      워드피스 임베딩 크기 ≠ 은닉 레이어 임베딩 크기

      - 먼저 사전의 원핫인코딩 벡터 V를 저차원의 워드피스 임베딩 공간 E로 투영($V \times E$)

        → $V \times E = 30{,}000 \times 128$

      - 다음으로 워드피스 임베딩 공간 E를 은닉 레이어 H로 투영($E \times H$)

        → $E \times H = 128 \times 768$

  - 크로스 레이어 변수 공유, 팩토라이즈 임베딩 레이어 변수화 적용 → 모델의 변수 감소 성공

ALBERT 모델의 학습

- 공통점
  - ALBERT 모델은 “영문 위키피디아,” “토론토 책 말뭉치” 데이터셋을 사용하여 사전학습 진행(BERT와 동일)
- 차이점
  - BERT: MLM, NSP태스크를 통해 사전 학습 진행
  - ALBERT
    - MLM 태스크 사용
    - NSP 태스크 → 문장 순서 예측 (Sentence Order Prediction, SOP) 태스크 사용으로 대체

ALBERT 모델의 학습

- 왜 NSP를 사용하지 않는가?
  - 다양한 연구 결과에 의해 다음의 사실을 알게 됨
    - NSP 태스크를 사용하는 것은 실제로는 유용하지 않음
    - MLM 태스크와 비교할 때, 난이도가 그리 높지 않음
    - NSP 태스크는 주제에 대한 예측, 문장의 일관성에 대한 예측을 하나의 작업으로 결합해서 수행
      - NSP 수행 시, 문서 내의 문장과 임의의 다른 문서의 문장을 비교함
      - 다음 순서에 나오는 문장인 경우: 문장의 순서, 일관성에 대한 예측을 수행
      - 그렇지 않은 경우: 서로 다른 문서의 임의의 문장을 비교 → 주제의 일관성 예측
    - 상황에 따라서는 필요 없는 예측에 자원을 소모하게 됨 → 비효율적
  - SOP는 주제 예측이 아니라 문장 간의 일관성만을 고려함

ALBERT 모델의 학습

- **문장 순서 예측**
  - SOP 태스크: NSP 태스크와 유사한 이진 분류 형태의 태스크
    - NSP: 한 쌍의 문장이 isNext 또는 notNext인지를 예측하는 형태로 학습 진행
    - SOP: 주어진 한 쌍의 문장이 문장 순서가 바뀌었는지 여부를 예측하는 형태로 학습 진행
  - 예시
    - 문장 1: She cooked pasta (그녀는 파스타를 요리했다).
    - 문장 2: It was delicious (맛있었다).

    문장 2가 문장 1 다음에 온다는 것을 알 수 있음 → Positive

    - 문장 1: It was delicious (맛있었다).
    - 문장 2: She cooked pasta (그녀는 파스타를 요리했다).

    문장의 순서가 바뀐 것을 것을 알 수 있음 → Negative

SOP는 주어진 한 쌍의 문장이 Positive(순서가 바뀌지 않음)인지 Negative(순서가 바뀜)인지를 판단하는 분류 문제이다.

단순히 하나의 문서에서만 한 쌍의 문장을 가져와서 순서가 바뀌었는지만 살펴보면 된다.

ALBERT와 BERT의 비교

- **ALBERT와 BERT를 비교하면**
  - **모든 경우에서 ALBERT는 BERT보다 적은 변수를 사용함**

모델	파라미터	레이어(L)	은닉(H)	임베딩(E)
BERT-base	110M	12	768	768
BERT-large	334M	24	1024	1024
ALBERT-base	12M	12	768	128
ALBERT-large	18M	24	1024	128
ALBERT-xlarge	60M	24	2048	128
ALBERT-xxlarge	235M	12	4096	128

334M: 3억3,400만개
18M: 1,800만개

  - **ALBERT-xxlarge**
    - SQuAD1.1/12.0, MNLI, SST-2, RACE 등의 태스크에서 BERT-base, BERT-large보다 월등한 성능을 보임

ELECTRA

ELECTRA

- **ELECTRA란?**
  - **Efficiently Learning an Encoder that Classifies Token Replacements Accurately**
  - **생성기(Generator)와 판별기(Discriminator)를 사용**
  - **사전 학습 과정에 “교체된 토큰 판별 태스크(Replaced Token Detection Task)”를 사용**

발표된 논문(2020)
ELECTRA: Pre-training Text Encoders As Discriminators Rather Than Generators
(https://arxiv.org/pdf/2003.10555.pdf)

ELECTRA에 적용된 기법

- BERT와의 차이점
  - MLM 태스크를 사전 학습에 사용하는 대신
  - 교체한 토큰 판별(Replaced Token Detection) 태스크 사용
- 교체한 토큰 판별 태스크
  - MLM 태스크와 기본적으로는 유사함
  - MLM: [MASK] 토큰으로 마스킹
  - 교체한 토큰 판별 태스크: 마스킹 대상인 토큰을 다른 토큰으로 변경한 후, 이 토큰이 실제 토큰인지 아니면 교체한 토큰인지 판별하는 형태로 학습 진행

ELECTRA에 적용된 기법

- **MLM 대신 교체한 토큰 판별 태스크를 사용하는 이유**
  - **MLM 태스크의 문제점**
    - 사전 학습 중에는 [MASK] 토큰을 사용하지만 파인 튜닝 시에는 [MASK] 토큰을 사용하지 않음

      → 사전 학습과 파인 튜닝 사이의 불일치 발생
  - **이런 문제로**
    - 토큰을 다른 토큰으로 교체하고 주어진 토큰이 실제 토큰인지 아니면 대체된 토큰인지 분류하도록 모델 학습을 진행

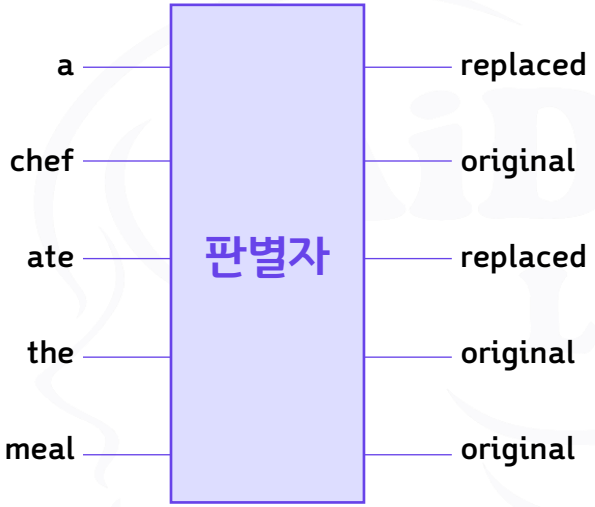
      → 사전 학습과 파인 튜닝 사이의 불일치 문제 해결 가능

ELECTRA에 적용된 기법

- 교체한 토큰 판별 태스크
  - MLM, NSP 태스크를 사용하는 BERT와 달리
  - ELECTRA는 교체된 토큰 판별 태스크만 사용하여 사전 학습을 진행함
  - 처리과정
    - The chef cooked the meal

      → tokens = [The, chef, cooked, the, meal]

      → tokens = [a, chef, ate, the, meal]
    - 토큰의 교체에는 MLM에서의 방법을 사용함



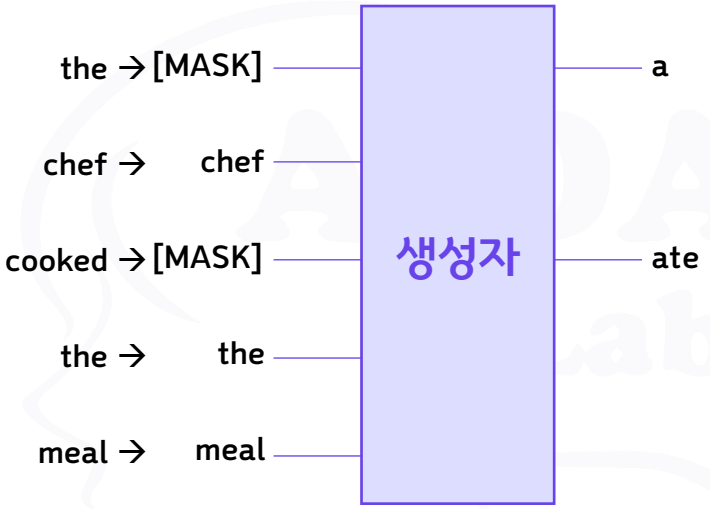
토큰이 원본인지 대체되었는지 판단하는 판별자

ELECTRA에 적용된 기법

- **토큰 교체하기**
  - tokens = [The, chef, cooked, the, meal]
  - tokens = [[MASK], chef, [MASK], the, meal]
  - 토큰을 다른 BERT 모델에 입력하여 마스크된 토큰 예측

    → 토큰에 대한 확률 분포를 결과로 제공하기 때문에 BERT를 “생성자”라고 부름

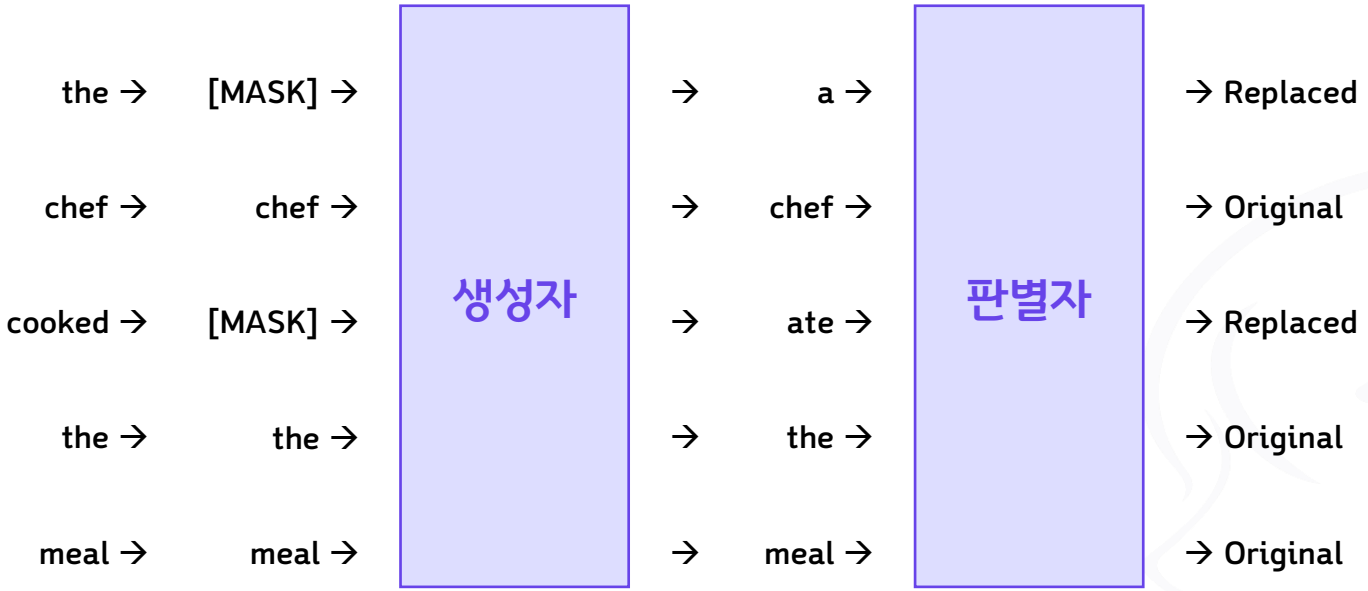
    → 마스크된 토큰을 생성자에 입력하면 마스크된 토큰에 대한 예측이 이루어지고 그 결과를 출력하게 됨
  - 교체한 토큰을 판별자에 입력하여 원래의 토큰인지 아닌지를 판별하도록 모델 학습



마스크된 토큰을 예측하는 생성자

ELECTRA에 적용된 기법

- **ELECTRA의 생성자와 판별자**

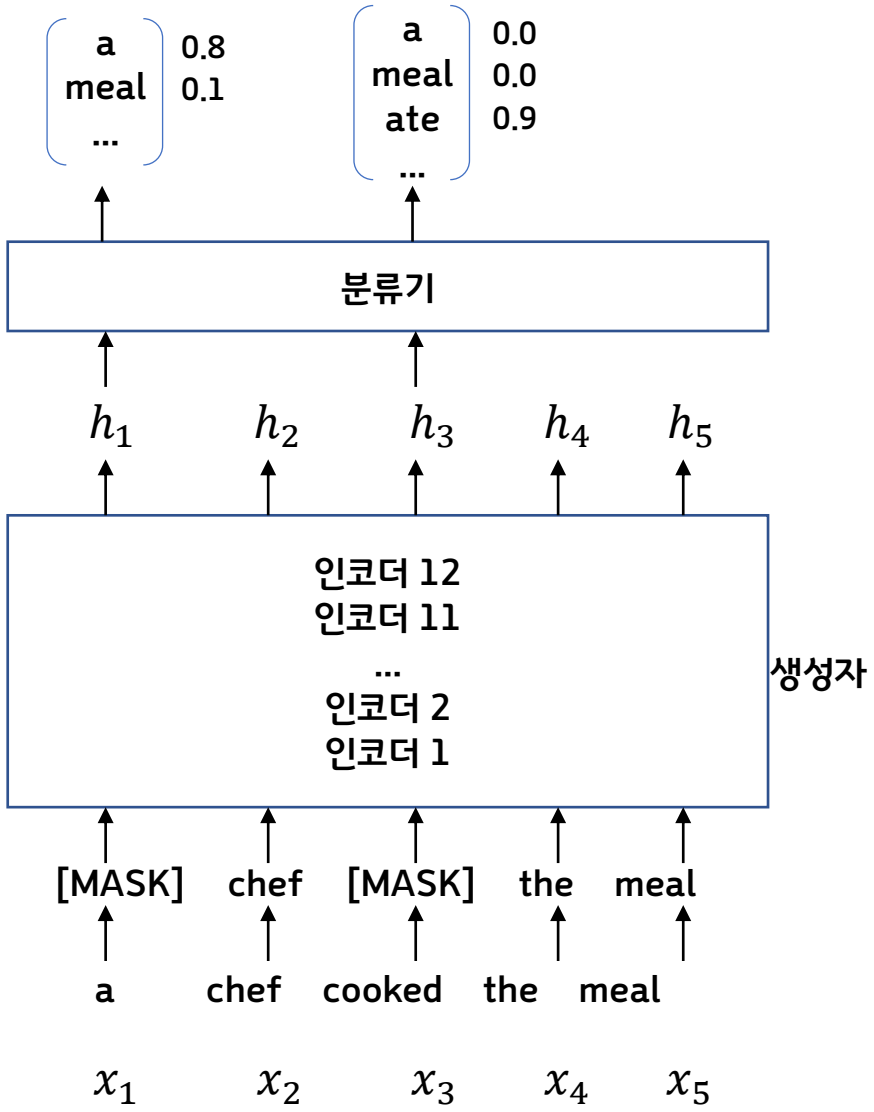


ELECTRA에 적용된 기법

- **ELECTRA의 생성자**
  - **생성자는 MLM 태스크를 수행함**
    - 15% 확률로 전체 토큰을 마스크된 토큰으로 교체
    - 생성기에서 마스크된 토큰을 예측하도록 학습 진행
  - **입력 토큰이 $x = [x_1, x_2, \dots, x_n]$ 이라면**
    - 무작위로 일부 토큰 마스킹하고 생성기에 입력하여 각 토큰의 결과값을 얻음
      - $h_G(X) = [h_1, h_2, \dots, h_n]$ → 생성기를 통해 얻은 각 토큰의 표현 값
    - 마스킹한 토큰에 대한 표현 → FFN의 softmax에 입력 → 토큰에 대한 확률분포 결과 출력
      (마스킹한 토큰이 사전 안에 있는 단어 중 어느 것에 속할지에 대한 확률분포 결과)

ELECTRA에 적용된 기법

ELECTRA의 생성자



- x_t를 t 위치의 마스크된 단어라고 하면
- 생성자는 소프트맥스 함수를 적용해 사전의 각 단어가 마스크된 단어일 확률을 출력함

$$P_G(x_t|X) = \frac{\exp(e(x_t)^T h_G(X)_t)}{\sum_{x'} \exp(e(x')^T h_G(X)_t)}$$

$e(\cdot)$: 토큰 임베딩을 의미

- 생성자에서 출력한 확률 분포를 통해 가장 확률이 높은 단어를 마스킹한 토큰으로 선택
- 왼쪽 그림에 표시된 확률 분포를 기반으로 마스크된 토큰 x_1은 a로 예측되고, x_3은 ate로 예측함

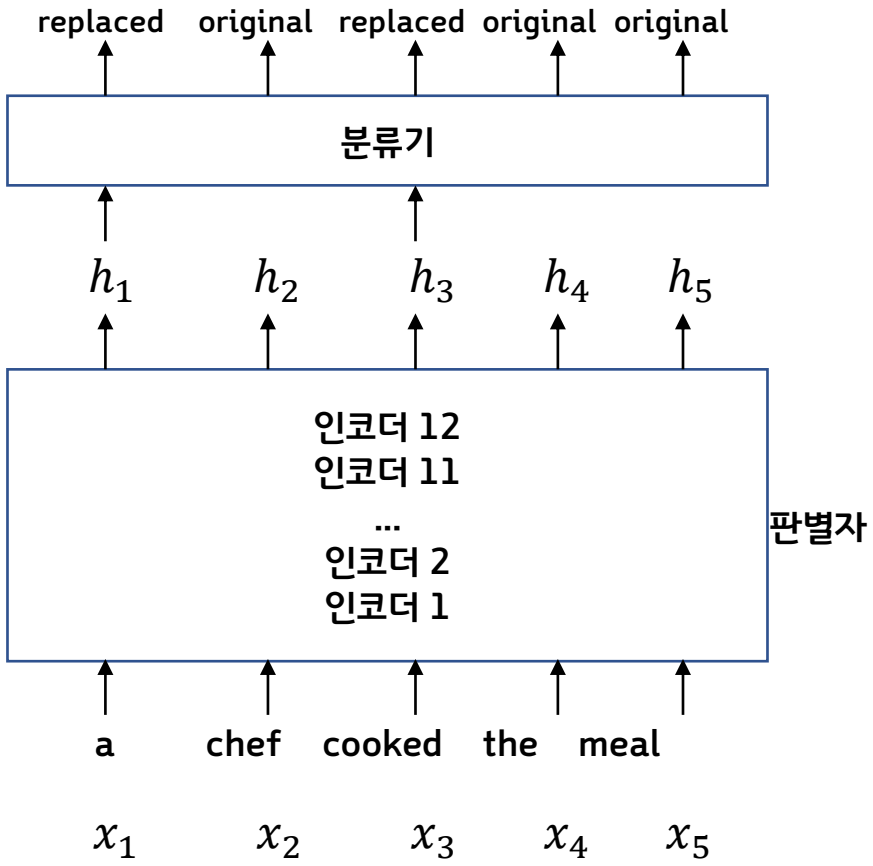
- 입력 토큰을 생성자에 의해 예측한 토큰으로 변경한 후 이를 판별자에 입력함

ELECTRA에 적용된 기법

- ELECTRA의 판별자
  - 판별자의 목표
    - 주어진 토큰이 생성자에 의해 만들어진 토큰인지 아니면 원래 토큰인지를 판별하는 것
  - 처리 과정
    - 먼저 토큰을 판별자에 입력해서 각 토큰에 대한 표현을 확보
      - $h_D(X) = [h_1, h_2, \dots, h_n]$ → 판별자를 통해 얻은 토큰별 표현값
    - 각 토큰의 표현을 시그모이드 함수를 가지고 있는 피드포워드 네트워크 형태의 분류기에 입력 → 주어진 토큰이 원래 토큰인지 아니면 교체된 토큰인지 판별하는 값 획득

ELECTRA에 적용된 기법

ELECTRA의 판별자



- 위치 t의 토큰을 x_t라고 하면
- 판별자가 시그모이드 함수를 사용해 해당 토큰이 원래 토큰인지 아닌지 여부를 아래의 식을 통해 계산함

$$D(X,t) = sigmoid(w^T h_D(X)_t)$$

- 마스킹한 토큰을 생성자에 입력하면
- 생성자는 마스크된 토큰을 예측함
- 다음으로 입력 토큰을 생성자에 의해 생성한 토큰으로 교체하고
- 이 값을 판별자에 입력함
- 판별자는 주어진 토큰이 원본인지 아닌지 여부를 판단함

ELECTRA 모델의 학습

- 모델 학습
  - 생성자는 MLM 태스크를 사용해 학습함
  - 주어진 입력에 대해 일부 위치를 선택해서 마스킹
  - $X^{masked} = replace(X, M, [MASK])$

$$\mathbf{X} = [\mathbf{x_1}, \mathbf{x_2}, \ldots \mathbf{x_n}], M = [m_1, m_2, \ldots, m_n]$$

주어진 입력 마스킹으로 선택한 위치

  - 마스킹 이후에 X^{masked} 토큰을 생성기에 입력하고 마스크된 토큰에 대한 예측 결과를 얻음

ELECTRA 모델의 학습

- 입력 토큰 X 중 일부를 생성자에 의해 생성한 토큰으로 변경
- 변경한 토큰: $X^{corrupted}$ → 생성자에 의해서 일부 토큰을 변경했기 때문
- 생성자의 손실함수
  - $L_G(X, \theta_G) = E(\sum_{i \in m} -\log P_G(x_i | X^{masked}))$
- 변경한 토큰 $X^{corrupted}$를 판별자에 입력하면 판별자에서는 해당 토큰이 원래 토큰인지 아닌지를 판단함
- 생성자와 판별자의 손실을 최소화 하는 방향으로 모델을 학습시킴

ELECTRA 모델의 학습

- 효율적인 학습 방법 탐색
  - 생성자와 판별자의 가중치를 공유함

    → 생성자와 판별자의 크기가 같다면 인코더의 가중치를 공유할 수 있음