BERT Sparse: BERT를 활용한 키워드 기반 실시간 문서 검색

김영민[O], 임승영, 유인국, 박소윤

LG CNS AI빅데이터연구소 기술팀
hong@mail.ac.kr, kim@mail.re.kr

BERT Sparse: Keyword-based Document Retrieval using BERT in Real time

Youngmin Kim[O], Seungyoung Lim, Inguk Yu, Soyoon Park
LG CNS AI Technology Team

요 약

문서 검색은 오래 연구되어 온 자연어 처리의 중요한 분야 중 하나이다. 기존의 키워드 기반 검색 알고리즘 중 하나인 BM25는 성능에 명확한 한계가 있고, 딥러닝을 활용한 의미 기반 검색 알고리즘의 경우 문서가 압축되어 벡터로 변환되는 과정에서 정보의 손실이 생기는 문제가 있다. 이에 우리는 BERT Sparse라는 새로운 문서 검색 모델을 제안한다. BERT Sparse는 쿼리에 포함된 키워드를 활용하여 문서를 매칭하지만, 문서를 인코딩할 때는 BERT를 활용하여 쿼리의 문맥과 의미까지 반영할 수 있도록 고안하여, 기존 키워드 기반 검색 알고리즘의 한계를 극복하고자 하였다. BERT Sparse의 검색 속도는 BM25와 같은 키워드 기반 모델과 유사하여 실시간 서비스가 가능한 수준이며, 성능은 Recall@5 기준 93.87%로, BM25 알고리즘 검색 성능 대비 19% 뛰어나다. 최종적으로 BERT Sparse를 MRC 모델과 결합하여 open domain QA환경에서도 F1 score 81.87%를 얻었다.

주제어: 자연어처리, 문서검색, 문서랭킹, BM25, BERT, 기계독해, open-QA

1. 서론

딥러닝 기반의 기계독해 모델은 다양한 질의응답 데이터셋에 대해 뛰어난 성능을 보이고 있다. 주어진 문단에서 질문에 대한 답변을 찾을 수 있는 경우 구체적인 답변 영역을 반환하는 SQuAD 2.0[1]태스크에서는 F1 기준으로 SOTA 모델이 사람보다 3%p이상 좋은 성능을 보인다. 한국어 위키피디아의 원본 페이지를 대상으로 자연어 질의에 대한 답변을 찾아내는 KorQuAD 2.0[2] 태스크에서도 단일 모델의 성능이 사람의 성적을 크게 앞질렀다.

그러나 이러한 기계독해 모델은 자연어 질문에 대해 답변을 찾을 수 있는 대상 문서가 주어져야 한다는 제약사항이 있다. 따라서 MRC 기술을 도메인의 범위가 넓은 챗봇, 자연어 검색 등의 어플리케이션에 활용하기 위해서는 입력으로 들어온 자연어 쿼리와 관련된 문서를 찾아오는 문서 검색 단계가 반드시 필요하다. 이렇게 문서 검색과 기계독해를 결합한 것이 Open domain QA 시스템이다. 이 시스템에서는 자연어 질문이 들어왔을 때 전체 문서 도메인으로부터 질문에 대한 구체적인 답변 영역을 반환하는 것을 목표로 한다.

높은 성능의 Open domain QA 시스템을 만들기 위해서는 기계독해뿐만 아니라 문서 검색의 품질을 향상시키는 것이 중요하다. 전통적으로는 문서 검색에서는 TF-IDF나 BM-25와 같은 키워드 기반의 방법들을 활용하였다. 예를 들어 DrQA[3]는 키워드 매칭 기반 알고리즘을 사용해 관련된 문서를 받아오고, 뉴럴 네트워크 모델을 사용하여 정확한 답변 영역을 반환하는 시스템을 제안하였다.

그러나 이러한 키워드 기반의 검색 방법은 유의어와 패러프레이징에 취약하다. 자연어 질문은 문서에 없는 단어를 사용할 수 있고, 추상화된 표현을 포함하기 때문에 단순 키워드 이상의 언어 지식이 필요하다. 유의어 사전 구축을 통해 성능을 개선하는 방법 있으나 도메인에 따라 수작업으로 사전을 수정해야 한다는 점에서 비용이 많이 든다.

이에 따라 최근에는 딥러닝을 활용한 문서 검색이 대두되고 있다. 딥러닝 모델 기반의 검색은 자연어에 대한 문맥적 이해에 기반하기 때문에 다양한 질문에 유연하게 대처할 수 있다. 특히 Transformer[4]를 기반으로 한 cross-attention model 모델[5,6]들은 문서 검색에서 좋은 성능을 보였다. 하지만 이 모델들은 자연어 쿼리가 들어올 때마다 모든 문서들에 대해서 새로 연산을 하기 때문에 대량의 문서에 대해 scale-out하기 어렵다는 단점이 있다.

이러한 한계를 보완하기 위해 임베딩 기반 모델[7,8,9]이 제안되었다. 이 방법에서는 검색쿼리와 문서를 독립적으로 연산하여 일정 크기의 벡터로 임베딩 한 후 벡터 간의 유사도를 계산해 관련도가 높은 문서를 받아온다. 문서를 미리 인코딩 해둘 수 있기 때문에 검색 속도가 상대적으로 빠르지만, 문서에 있는 모든 정보를 한정된 크기의 벡터로 표현하는 과정에서 많은 정보가 손실된다.

위와 같은 단점들을 극복하기 위해 이 논문에서는 딥러닝을 활용한 새로운 문서 검색 모델을 제안한다. BERT 기반의 딥러닝 인코더를 활용하지만 문서를 고정된 차원의 벡터가 아닌 키워드 별 점수의 형태로 변환하는 키워드 기반 검색 모델이다. 이때 언어 딥러닝 모델을 활용하였기 때문에 문서에 등장하지 않는 키워드에도 점수를 부여하여 유사한 의미의 키워드들을 검색할 수 있다(부록의 표 4 예시 참고). 실험 결과 제안한 모델은 키워드 기반 방법과 임베딩 기반 방법들에 비해 검색 성능이 뛰어나다. 또한, 모델 학습을 완료한 후 문서를 인코딩 해두면 추론 시에는 GPU를 사용하지 않고도 자연어 질문과

관련된 문서를 받아올 수 있기 때문에 추론 비용이 저렴하다. 본 논문에서는 최종적으로 MRC 모델을 결합하여 open domain QA 환경에서 질의응답 성능까지 측정하였다.

2. 관련 연구

2.1 기존 키워드 기반 검색

주어진 사용자 쿼리를 이용한 문서 검색에는 다양한 알고리즘들이 활용되고 있지만, 핵심적으로는 Term Frequency(TF)와 Inverse Document Frequency(IDF)를 이용한 키워드 기반의 검색 방식을 많이 활용한다. 한 문서 안에서 얼마나 자주 등장하는 단어인지, 그리고 여러 문서에 걸쳐서 자주 등장하는 단어인지 그 빈도를 파악하여 단어 별 중요도 점수를 합산하여 관련 있는 문서를 반환하는 방식이다.

가장 널리 사용되는 오픈소스 검색엔진인 lucene 에서는 TF-IDF 를 조금 변형한 방식의 BM25 알고리즘을 기본 검색 모델로 사용하고 있고, lucene 을 기반으로 개발된 solr, Elasticsearch 에서도 역시 BM25 기반의 알고리즘을 활용하고 있다.

DrQA[3]에서는 이러한 키워드 기반의 검색과 기계 독해 모델을 순차적으로 적용하여, 사용자 쿼리에 대한 구체적 답변을 반환하도록 함으로써, 기계 독해에 어떻게 검색을 적용하는지에 대해 서술하였다. 이 연구에서는 bigram 단위의 해싱과 TF-IDF 알고리즘을 사용해 답을 찾을 후보 문단을 가져오고, 뉴럴 네트워크 모델을 사용하여 후보 문단에서 정확한 답변 영역을 반환하는 Open domain QA 시스템을 제안하여 주목을 받은 바 있다.

그러나 이런 키워드 기반의 검색은 문서에 쿼리와 정확히 일치하는 키워드가 없는 경우, 또는 쿼리가 자연어로 이루어져 있어 추상화된 표현을 포함하는 경우 매우 취약하다. "휴대폰", “모바일”과 같이 비슷하지만 다른 표현들을 지닌 쿼리들에 대해서도 동일한 검색 결과를 보장할 수 없으며, 이러한 유의어, 패러프레이징에 상당히 취약하다. 우리의 연구에서는 기존 키워드 기반의 한계를 극복한 의미 기반 검색을 제안하며, 이어지는 기계 독해의 성능을 상당히 끌어올리는데 기여하였다.

2.2 딥러닝을 활용한 의미 기반 검색

앞서 언급한 키워드 기반 검색의 한계점을 극복하고자, BERT[10,11]와 같이 대용량의 corpus로 pre-training 과정을 거친 transformer 기반 NLU 모델을 이용한 연구들이 진행되고 있다.

이러한 임베딩 기반의 검색들은 다음과 같은 과정을 거친다. (1) 쿼리와 문서를 BERT를 이용하여 같은 벡터 공간 위의 벡터로 변환한 뒤, (2) 내적이나 코사인 유사도를 이용해 두 벡터 간의 유사도를 측정하고 (3) 가장 높은 유사도 점수 갖는 문서 순으로 반환하는 순이다.

(1)에 해당하는 단계를 수행하기 위해서 쿼리와 문서를 인코딩하는 방식은 크게 두 가지로 구분할 수 있다. 첫째는 Cross-encoder 방식으로, 최종 representation을 얻기 위해 쿼리와 문서를 인코딩하는 BERT 인코더를 1개만 두고, 쿼리와 문서 간의 cross-attention 연산을 거치는 방식이다. 이는 BERT에서 기계 독해를 수행할 때의 연산과 같은 방식으로, 쿼리가 들어올 때마다 문서를 인코딩해야 하기 때문에 정확도는 높지만 많은 계산 비용으로 인해 실제 검색 서비스에 적용하기는 어렵다. 둘째는 Bi-encoder를 사용하는 방식으로, 쿼리와 문서를 인코딩하는 BERT 인코더를 각각 따로 두어 분리하는 것이다. 이러한 방식을 이용하면 쿼리와 무관하게 문서를 먼저 인코딩하여 캐싱 해둘 수 있기 때문에 빠르고 효율적인 검색이 가능하다.

Poly-encoder[12]는 위에서 언급한 두 가지 유형의 인코딩 방식의 장점을 결합한 아키텍처를 가지고 있다. Bi-encoder 방식으로 문서를 각각 인코딩하여 feature 개수를 줄인 후에 캐싱하여 저장하였다가, 쿼리가 들어오면 각각의 문서 별로 캐싱된 feature들로부터 Cross-encoder 방식으로 attention 연산을 수행 후 문서 별 최종 점수를 계산한다.

Sentence-BERT[8]에서는 문장 간 유사도를 비교하기 위해서 Bi-encoder 방식을 차용해 속도를 개선하지만, Siamese BERT-Network를 활용하여 인코더를 하나로 통합하여 활용함으로써 파라미터의 수를 줄이는 시도를 하기도 하였다.

DenSPI[13]에서는 Bi-encoder 방식으로 쿼리와 문서를 각각 인코딩 하였다. 이 연구의 특이한 점은 문서단위의 인코딩이 아니라, 문서에서 답변이 될만한 구문들을 모아 BERT로 인코딩한 뒤 캐싱한다. 쿼리 입력 시, 이러한 구문들 전체를 대상으로 검색하기 때문에 검색, 기계 독해 모델을 순차적으로 거치지 않고 바로 정답을 반환할 수 있다는 장점이 있다.

그러나 위 BERT를 활용한 의미 기반 검색 연구들 역시 명확한 한계점이 존재하는데, 바로 많은 내용을 포함하는 문서 또는 구문을 한정된 길이의 벡터로 축약하는 데서 오는 '정보의 손실'이다. 이 정보의 손실 때문에, 의미 기반 검색은 높은 정확도를 보장하지 못하고 있으며, 이를 위해 의미 기반 검색을 키워드 기반 검색과 앙상블하여 기존 키워드 기반 검색의 장점을 살리고 단점을 보완하는 기법[13,14]을 사용하기도 하였다.

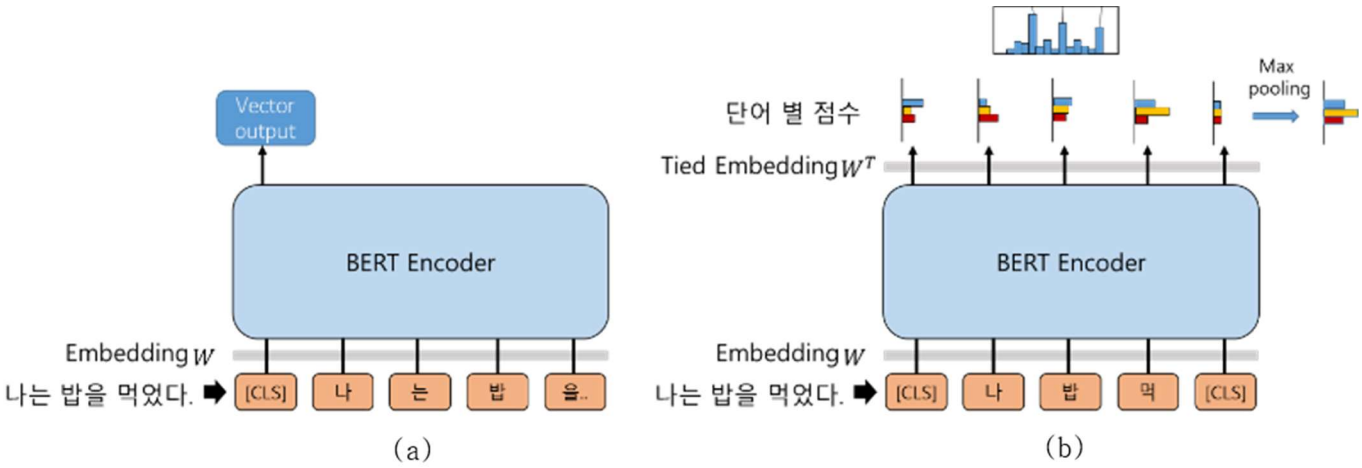
우리의 연구에서는 BERT를 활용하여 단어 사전에 있는 모든 단어 별로 점수를 새롭게 계산하는 독창적인 방식을 고안하여 기존 키워드 기반 검색의 장점은 살리면서도, 유사어와 패러프레이징에 취약한 단점은 BERT로부터 얻은 문맥 기반 단어 점수로 보완하여 의미 기반 검색 기법의 장점도 취하였다. 높은 정확도를 유지하면서도 검색 속도를 실 서비스 가능한 수준으로 보장하기 위해 위 논문들의 방식을 따라 Bi-encoder의 구조를 차용함으로써 매우 빠른 검색 속도를 얻었다.

3. 검색 모델

3.1 BERT Dense

BERT Dense는 Bi-encoder를 사용하는 기존의 임베딩 기반 모델[7,9]과 유사하다. BERT 인코더를 활용하여 쿼리와 문서를 각각 임베딩 하였다. 임베딩 벡터로는 그림 1(a)와 같이 BERT의 sequence 출력 결과에서 ‘[CLS]’에 해당하는 첫 번째 벡터를 사용했다. 이후 dot-

그림 1 BERT Dense(a), BERT Sparse(b) 모델 구조



product 연산을 통해 쿼리와 문서의 유사도를 계산한다.

$$\mathrm{Q}_{\mathrm{emb}} = BERT_q(Q) \in R^h, \qquad h = featuer\ size$$
$$\mathrm{d}_{\mathrm{emb}} = BERT_d(doc) \in R^h$$
$$\mathrm{score}(\mathrm{Q}, \mathrm{doc}) = < \mathrm{Q}_{\mathrm{emb}}, d_{emb} >$$

Sentence-BERT[8]와 같이 Siamese 구조를 사용할 수도 있지만 쿼리와 문서의 분포, 길이 등이 다르기 때문에 서로 다른 모델을 학습하여 사용하였다. 그리고 벡터간의 유사도로는 코사인 유사도를 많이 사용하기도 하지만 논문에서는 연산량이 더 적은 dot-product를 사용하였다.

3.2 BERT Sparse

BERT Sparse는 BERT를 활용한 키워드 기반의 검색 모델이다. BERT를 사전학습 할 때 MLM과제에서 최종적으로 토큰마다 단어의 등장 가능성을 계산하는데, 이를 응용하여 단어의 등장 가능성(logit)을 단어 점수로 해석하였다. 그림 1(b)의 예시처럼 '밥'의 자리에는 비슷한 의미를 가지는 '식사'나 '끼니'도 높은 값을 가질 것이다. 우리는 logit 시퀀스에 max-pooling을 취해서 최종적으로 문서가 단어 수 크기의 벡터 S로 표현되도록 만들었다. 벡터는 각 단어의 점수를 의미하며, 수식은 다음과 같다.

$$W_{emb} \in R^{v \times h}, \qquad v = vocabulary\ size$$
$$\mathrm{O} = \mathrm{BERT}_{\mathrm{sparse}}(\mathrm{doc}) \in R^{l \times h}, \qquad l = sequence\ length$$
$$\mathrm{S}_{\mathrm{doc}} = max\ pool_i(\mathrm{O} \times {W_{emb}}^T)_{ij} \in R^v$$

이를 통해 문서는 문맥정보가 고려된 단어 점수 벡터로 인코딩 된다.

쿼리와 문서간의 점수는 단어 별 점수를 합하여 계산하였다.

$$\mathrm{score}(\mathrm{Q}, \mathrm{doc}) = \sum_{q_i \in Q} S_{doc}(q_i)$$

BM-25와 같이 쿼리를 따로 인코딩 할 필요가 없기 때문에 기존의 bi-encoder를 활용한 의미 기반 모델들보다 연산량이 적다는 장점이 있다.

Denspi[13,14]에서도 키워드 기반 정보를 사용하기 위해 sparse 구조를 활용하지만 문단에 등장하는 키워드에 대해서만 인덱싱이 된다. 하지만 본 논문의 모델은 문단에 등장하지 않는 키워드들에 대해서도 인덱싱이 가능하다는 차이가 있다.

3.3 목적함수

전체 문서(D) 중에서 쿼리에 대한 정답 문서의 확률은 다음과 같다.

$$\mathrm{p}(Q, \mathrm{d}_Q) = \frac{\exp\left(score(Q, d_Q)\right)}{\sum_{d_j \in D} \exp\left(score(Q, d_j)\right)} \approx \frac{\exp\left(score(Q, d_Q)\right)}{\sum_{d_j \in Batch+NS} \exp\left(score(Q, d_j)\right)}$$

손실함수는 이 확률의 cross-entropy 로 정의하였는데 학습할 때 항상 모든 문서에 대한 확률을 계산하는 것은 비용이 크다. 그래서 전체 문서가 아닌 샘플 문단에 대해서만 연산을 하는 sampled-softmax[15]를 사용하였다. 샘플 문단으로는 연산량을 최소화하기 위해 학습 배치 내의 문단을 활용하는 방법이 사용된다[7,9]. 그런데 이런 방법으로 학습하는 경우 배치 내에서만 확률이 높으면 되기 때문에 세부적인 내용이 아니라 포괄적인 주제만 학습하는 경향이 생길 수 있다. 이러한 점을 보안하기 위해 2가지 방식으로 적대적인 문단을 배치에 추가하여 학습하였다. 하나는 같은 문서 내의 다른 문단을 추가하는 것이고 다른 하나는 다른 문서에서 BM25를 통해 상위로 뽑힌 유사한 문단을 추가하는 것이다. 이런 negative sampling(NS) 방식을 통해 문서의 세부적인 내용까지 학습할 수 있도록 유도하였다.

추가적으로 효율적인 학습을 위해서 손실함수를 하나 더 추가하였다. 앞서 설명한 손실함수는 여러 문서 중에서 쿼리와 레이블로 매칭된 문서를 찾는 과제(Q→doc)에 대한 것이다. 추가 손실함수는 반대로 여러 쿼리 중에서 해당 문서를 찾는 방식(doc→Q)으로 계산하였다. 기존 손실함수에서 이미 계산한 배치 내의 문서와 쿼리에 대해서 연산이 이루어지기 때문에 추가적인 연산이 적다.

$$\mathrm{Loss} = \mathrm{Loss}_{\mathrm{Q}\rightarrow\mathrm{doc}} + Loss_{\mathrm{doc}\rightarrow\mathrm{Q}} * \lambda$$

최종 손실함수는 메인 과제인 Q→doc가 더 잘 학습되도록 doc→Q의 손실함수에 1보다 작은 양수 λ를 곱해주었다.

마지막으로 학습 과정에서 단순히 weight의 크기 증가로

score(Q,doc)가 높아지는 문제가 생길 수 있다. 이런 과적합으로 인해 loss가 감소하는 것을 방지하기 위해 학습 가능한 스케일 조절 인자 α를 softmax의 온도로 넣어주었다. 이런 re-scale 인자들이 실험적으로 좋은 성능 얻는데 도움이 될 수 있다[12].

4. 실험 및 결과

4.1 실험 환경

4.1.1 데이터

학습 및 평가 데이터로는 KorQuAD 2.0[2]을 활용했다. KorQuAD 2.0은 위키피디아 문서로 이루어진 HTML 형식의 데이터이다. 원래는 위키피디아 문서 한 페이지 내에서 질문에 대한 답을 찾는 MRC 과제이지만 이를 전체 문서에서 질문에 대한 답이 있을 것으로 예측되는 문단을 찾아야 하는 문서 검색과제로 변환하였다. 평가 시에는 검증데이터에 있는 전체 문서를 검색 대상으로 하였고 검색되는 단위는 문서를 일정 stride로 나눈 문단으로 하였다. 한 문단은 길이가 480 토큰 안팎이 되도록 나누었는데 그 이유는 BERT MRC 모델을 연결하는 open-QA를 염두하였기 때문이다. 검색 결과로 나온 480토큰 길이의 문단을 질문과 합쳐 512 토큰 이내가 되도록 하고 이를 통해 BERT MRC 모델과 연결할 때 모델 최대 길이인 512를 넘어가서 문단이 더 나뉘는 것을 방지하였다. 또한 문단의 앞에 문서의 제목도 추가적으로 넣었다. 문단의 내용에 주어가 없거나 대명사를 활용하는 경우가 있어 부족한 정보를 보충하기 위해서이다. Stride는 128로 설정하였고 Stride로 인해 실제로 질문에 대한 답을 포함되는 문단도 여러 개가 될 수 있다.

표 1 데이터 통계

	문서 수	문단 수	질문 수
학습데이터	38,496	-	83,486
검증데이터	4,736	113,614	9,880

4.1.2 모델 별 설정

BM25는 python 라이브러리 Rank-BM25의 Okapi BM25 모델을 사용하여 테스트 하였다. BERT Dense 및 Sparse 학습에 사용된 BERT는 뉴스데이터, 위키피디아와 나무위키 말뭉치를 수집하여 자체적으로 사전학습을 하였다. 이 모델은 Google BERT base[10]와 같은 크기로 은닉층의 수(h)는 768, 레이어의 개수는 12개이다. 학습률은 5e-5 로 설정하였고 배치사이즈는 GPU당[2] negative sample을 포함하여 9개이다. BERT Sparse 모델에는 추가적인 텍스트 전처리로 조사와 어미를 제거하여 키워드 검색의 특징을 더 강화하였다.

4.2 문서 검색 결과

표 2에서는 제안한 방법과 BM25, 임베딩 방법의 성능을 비교하였다. 그리고 세가지 방법을 앙상블하여 성능을 확인하였다. 성능 평가 지표는 쿼리를 검색하여 평가 데이터의의 모든 문단에서 점수가 높은 상위 n개 문서를 뽑고 그 중 답을 포함하는 문단이 있으면 정답으로 보는 방식으로 정확도 Recall@n을 측정하였다. Latency는 python CPU환경에서 쿼리당 처리 속도를 측정하였다.

먼저 BM25의 성능을 보면 R@5 74.7% 정도로 간단한 방법임에 비해 좋은 성능을 보여준다. BERT Dense는 BM25보다 정확도가 낮은데 이는 문단 단위를 벡터로 압축해야 하기 때문에 세세한 부분에 대한 정보가 필요한 질문에 잘 답하지 못하기 때문으로 보인다. 단일모델로써는 BERT Sparse가 가장 좋은 성능을 보여주었고 속도 또한 가장 빠르다.

앙상블은 모델 별 가중치 합을 통해 쿼리와 문서간의 점수를 구하였다. 세 모델 모두 합쳐 앙상블한 경우 BERT Sparse 보다 R@5 기준 1.4% 포인트 상승했다. 특이한 점은 단일 모델로는 BERT Dense 모델이 BM25보다 낮은 성능을 가지고 있지만 BERT Sparse 모델과 앙상블 하는 경우 오히려 Dense 모델이 더 좋은 효과를 보여준다. 이는 BERT Sparse 모델은 문단의 세부적인 정보를 잘 저장하고 BERT Dense 모델은 전체적인 정보를 요약하여 서로 다른 특징을 추출한다는 점이 서로의 단점을 보완해 줄 수 있기 때문으로 보인다.

4.3 BERT Sparse 인덱싱 가지치기

실제 서비스 단계에서는 검색 속도뿐만 아니라 인덱싱 용량도 중요한 문제이다. 이를 위해 BERT Sparse 인덱싱

표 2 모델 별 Recall@n 성능 및 쿼리당 처리 속도

모델	R@1	R@3	R@5	R@10	R@50	Latency(ms)
BM25	49.39	68.87	74.71	80.84	88.88	1,146[1]
BERT Dense	46.93	62.72	69.03	76.58	88.45	343.5
BERT Sparse	**78.6**	**90.39**	**93.87**	**96.53**	**98.7**	60.3
BERT Sparse w/o fine-tuning	44.84	63.62	70.63	77.95	84.56	-
앙상블 BERT Sparse + BM25	78.83	90.7	94.08	96.74	98.93	-
BERT Sparse + Dense	**81.35**	91.98	95.1	97.16	99.11	-
BERT Sparse + Dense + BM25	81.11	**92.02**	**95.27**	**97.27**	**99.18**	-

[1] Python 라이브러리가 최적화 되어있지 않아 다소 느림.
Elasticsearch를 활용하는 경우 100ms 이내의 latency를 확인하였음.

[2] Sampled-softmax 연산이 GPU마다 이루어져서 GPU당 배치 사이즈가 중요함

결과 크기를 줄이는 두 가지 방법을 실험하였다. 첫 번째 방법은 stride 조절을 통한 문단 수를 줄이는 것이다. Stride는 문단 수에 직접적으로 영향을 주는데 값이 작을수록 문단 간에 중복되는 부분이 많아져 비효율적이다. 표 3을 보면 stride를 128에서 256으로 조정하면 문단 수는 거의 절반가까이 줄어들고 정확도는 1.7% 정도 감소하게 된다.

두 번째 방법은 각 문단마다 일정 값 이상의 점수를 갖는 단어들만 저장하도록 하는 것이다. BERT Sparse 모델은 기본적으로는 각 문단마다 모든 단어들에 대한 점수를 가지게 되는데 문단과 관련이 없는 낮은 점수의 단어들도 포함되어 비효율적이다. Threshold가 높아질수록 중요 단어들만 남아 문단 별 평균 단어 수가 줄어들지만 성능도 같이 하락하기 때문에 상황에 따라 적정 값을 찾는 것이 필요하다. Threshold가 10 이하 인 경우 오히려 stride 256 기준보다 성능이 약간 높아지기도 하였다. 이는 노이즈가 되는 키워드들이 제거되어 성능이 향상되었을 수 있다.

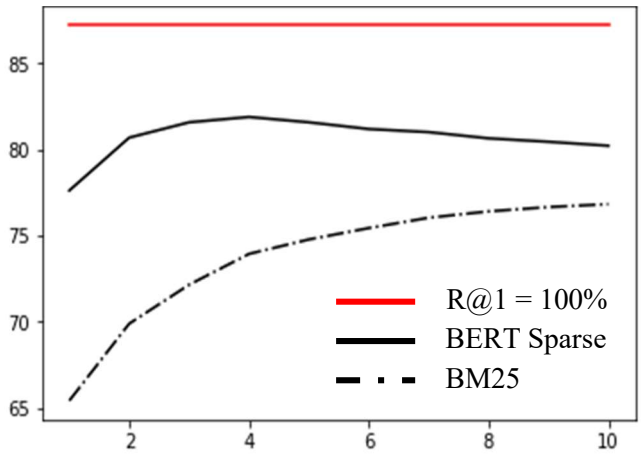
표 3 BERT Sparse 인덱싱 경량화에 따른 성능변화. Threshold 조절은 stride 256에서 진행하였음.

방법	문단 수	문단 별 평균 단어 수	R@5
Baseline	113,614	99,694	93.87
Stride 128 -> 256	60,355	99,694	92.18
Threshold >5	60,355	22,634	92.38
>10	60,355	4,201	92.2
>15	60,355	814	91.73
>20	60,355	280	90.72

4.3 MRC

Open-QA를 수행하기 위해 문서 검색 이후 MRC 모델을 연결하여 성능을 측정하였다. MRC 모델은 KorQuAD 2.0 기존의 과제 방식을 따라 한 문서 내에서 답을 찾도록 학습을 하였다. 이 모델을 사용하여 답을 포함하고 있는 문단 하나에서 답을 찾는 경우 F1 score 87.2가 나왔다. 이것은 검색 성능 R@1 이 100%일 때 얻을 수 있는 최대 점수이다. 그림 2는 검색 모델이 반환하는 문단 수를 바꿔가며 실험한 결과이다. 검색모델로 BERT Sparse를 사용하는 경우 후보 문단이 늘어날수록 성능이 오르다가 4개에서 F1 score 81.87로 가장 높은 성능을 보였고 이후로는 감소하는 경향을 확인할 수 있다. 이는 MRC 모델이 한 문서 내에서만 답을 찾도록 학습되어 있어 여러 문서의 문단이 섞이는 경우에 취약한 것으로 보인다. 그래서 성능을 높이기 위해서는 negative sample을 추가하여 다른 문서에서 비슷한 내용의 문단들을 추가하여 MRC 모델을 학습할 필요가 있다. BM25는 조금 다른 경향을 보이는데 후보 문단이 늘어날수록 성능이 계속 증가하였다. 이는 검색성능이 낮아 일정 성능까지는 계속 증가하는 것으로 보인다.

그림 2 후보문단 개수에 따른 MRC F1 성능



5. 결론

이 논문에서는 문서 검색을 위한 BERT Sparse 모델을 제안하여 Recall@5 기준 93.87%으로 다른 모델 대비 뛰어난 성능을 확인하였다. 속도 측면에서도 키워드 기반 검색 방식과 비슷하다. 또한 BERT Sparse, Dense 모델과 BM25의 앙상블을 통해 성능이 더 높아짐을 확인하였다. 그리고 문단의 수와 단어 수를 줄이는 방식을 통해 가지치기를 실험하였다. 최종적으로 MRC 모델과 결합하여 Open-QA에서 F1 score 81.87의 성능을 내었다.

BERT Sparse 모델은 문서는 문맥정보를 활용하여 인코딩 되지만 검색 쿼리는 단순히 키워드 방식으로 사용되어 속도는 빠르지만 문맥정보는 고려하지 못하는 단점이 있다. 앞으로의 연구에서는 전후 처리 및 모델 개선을 통해 이와 같은 단점을 보완해 나갈 것이다. 또한 현실에서 사용 가능한 수준의 성능과 속도를 내는 자연어처리 모델 연구에 힘쓰고자 한다.

참고문헌

[1] Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. Squad: 100,000+ questions for machine comprehension of text. arXiv preprint arXiv:1606.05250, 2016.

[2] 김영민, 임승영, 이현정, 박소윤, & 김명지. (2020). KorQuAD 2.0: 웹문서 기계독해를 위한 한국어 질의응답 데이터셋. *정보과학회논문지*, *47*(6), 577-586, 2020.

[3] Chen, D., Fisch, A., Weston, J., & Bordes, A. Reading wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051*, 2017

[4] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008). 2017.

[5] Yang, W., Zhang, H., & Lin, J. Simple applications of

BERT for ad hoc document retrieval. *arXiv preprint arXiv:1903.10972*. 2019.
[6] Nogueira, R., Yang, W., Cho, K., & Lin, J. Multi-stage document ranking with BERT. *arXiv preprint arXiv:1910.14424*. 2019
[7] Lee, K., Chang, M. W., & Toutanova, K. Latent retrieval for weakly supervised open domain question answering. *arXiv preprint arXiv:1906.00300*. 2019.
[8] Reimers, N., & Gurevych, I. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*. 2019
[9] Chang, W. C., Yu, F. X., Chang, Y. W., Yang, Y., & Kumar, S. Pre-training tasks for embedding-based large-scale retrieval. *arXiv preprint arXiv:2002.03932*. 2020.
[10] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*. 2018.
[11] Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. Improving language understanding by generative pre-training. 2018.
[12] Humeau, S., Shuster, K., Lachaux, M. A., & Weston, J. Poly-encoders: Transformer architectures and pre-training strategies for fast and accurate multi-sentence scoring. *arXiv preprint arXiv:1905.01969*. 2019.
[13] Seo, M., Lee, J., Kwiatkowski, T., Parikh, A. P., Farhadi, A., & Hajishirzi, H. Real-time open-domain question answering with dense-sparse phrase index. *arXiv preprint arXiv:1906.05807*. 2019.
[14] Lee, J., Seo, M., Hajishirzi, H., & Kang, J. Contextualized Sparse Representations for Real-Time Open-Domain Question Answering. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (pp. 912-919). 2020.
[15] Chen, W., Grangier, D., & Auli, M. Strategies for training large vocabulary neural language models. *arXiv preprint arXiv:1512.04906*. 2015.

부록

표 4 BERT Sparse 모델 인코딩 결과 상위 단어 점수 예시.

동의어(강아지->애완견, 요금제->통신비)나 오탈자 (함께->함꼐)가 같이 등장하는 것을 확인할 수 있음.

제목	나		아이유		요금제	
내용	나는 초등학교를 졸업 후 강아지와 함께 아파트에서 살았다.		음색이 좋고 감각이 뛰어나다는 평을 받으며, 정식 데뷔 전 여러 비공식 무대를 거쳐		본 요금제는 만 4세부터 18세까지 청소년만 가입 가능합니다	
단어 별 점수	초등	53.1	아이유	58.3	요금제	59.8
	아파트	52.4	데뷔	54.4	청소년	52.3
	나	52.1	음색	49.7	가입	51.9
	강아지	49.0	정식	48.8	본	48.1
	학교	47.4	공식	48.4	세	47.8
	졸업	46.9	감각	45.3	18	43.2
	함께	39.9	평	43.0	가능	41.8
	후	39.8	뛰어나	41.5	요금	39.4
	초등학생	30.8	무대	39.3	살	37.4
	졸업식	29.7	좋	35.0	가입자	30.8
	난	28.5	뛰어난	33.2	나이	29.8
	같이	28.3	평가	32.4	통신비	29.5
	애완견	27.9	뛰어났	29.3	미성년자	27.6
	함께	27.8	목소리	29.2	해야	26.5
	주택	27.2	비@@	28.3	가입비	26.0
	나@@	27.0	보컬	28.0	불가	25.7
	애견	26.9	호평	26.8	단말기	25.4
	내	26.7	노래	26.5	연령	24.7
	살	26.5	창법	26.4	출고가	24.7
	입학	25.5	음악	26.2	운임	24.5
	대학교	25.5	데뷔작	26.0	소년	24.4
	초교	24.3	음원	25.4	통화료	24.3
	중학교	22.8	탁월	24.2	요금@@	23.7
	국민학교	22.8	음질	24.0	약관	23.6
	댕댕이	20.5	출중	23.9	이용료	23.4
	개	20.2	공연	23.6	기본요금	23.3
	오피스텔	19.9	곡	23.4	중학생	23.3