(2025.10.26) SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking

연구 배경

정보 검색(Information Retrieval, IR)은 사용자가 필요로 하는 정보를 문서나 데이터베이스, 웹 등에서 찾아내는 기술과 연구 분야를 말합니다. 이 분야는 사용자의 쿼리(query) 또는 질문에 가장 관련성 높은 정보를 신속하고 정확하게 제공하는 것을 목표로 합니다. IR 분야도 ML 특히 DL이 결합되면서 NN 기반의 ranking model이 쿼리에 맞는 정보를 '순위'(rank)를 매겨 응답하는 연구가 많이 이루어 지고 있습니다.

일반적인 NN기반 IR Model들은 일반적으로 2단계 파이프라인을 사용합니다.
1번째 stage에서는 BoW(bag-of-words) 모델 기반의 retrieval model를 통해 쿼리에 적합한 documents들을 document collection에서 먼저 추출합니다.
2번째 stage에서는 1번째 stage에서 빠르게 추출된 documents들을 input으로 조금 더 정교한 모델을 이용하여 사용자의 쿼리에 보다 적합한 결과로 re-rank 한 결과를 리턴합니다.
이 중 1번째 stage는 "문자 그대로의 매칭"으로 빠르게 document들을 추출하기 때문에 속도가 빠르지만, 대신 "relevant" 하지만 "exact-matching" 하지 않는 단어들을 고려하지 못하기 때문에 semantic-level의 결과는 기대할 수 없었습니다. (vocabulary mismatch problem)

이 1번째 stage의 문제를 해결하기 위해 최근(2020~21년)의 경향은 BoW 모델을 사용하는 대신 Bert 등 LLM을 이용해서 쿼리와 document를 dense embedding으로 표현 후 이 벡터들 간의 유사도(approximate nearest neighbor search) 를 계산하는 방법이 제안되었습니다. 하지만 이 방법만으로는 semantic level의 응답값을 리턴할 수 있지만 계산 비용 상승으로 인한 효율성 감소, 그리고 쿼리의 특정 키워드가 document에 정확히 존재하는지 여부(exact-match)를 명시적으로 확인할 수 없는 문제점이 있었습니다.

그래서 최근에는 "sparse representation"을 NN을 이용해 학습하는 데 대한 관심이 증가하고 있습니다.(SparTerm)
이를 통해 1st-stage 모델은 정확한 매칭(exact-match)과 효율성 같은 BOW 모델의 장점들을 가져오면서, dense embedding 을 사용하는 확장된 IR 모델의 semantic-level 결과값도 기대할 수 있는 장점도 가져올 수 있습니다.

논문 핵심

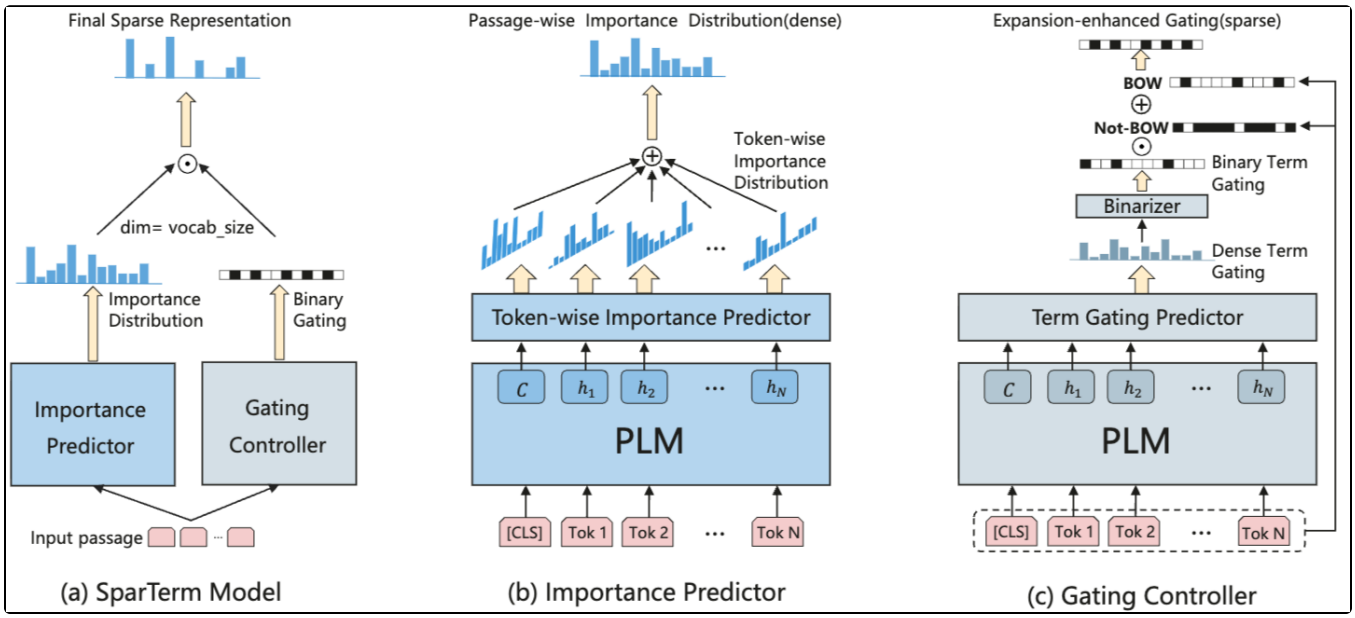
논문에서 발표한 SPLADE 모델은 SparTerm 모델 베이스의 variation 모델로 다른 희소 검색 방법들(BM25, DeepCT, doc2query-T5, SparTerm)을 능가하는 퍼포먼스를 낼 수 있다.

사용된 이론 및 방법론

SPLADE 모델은 SparTerm 모델을 베이스로 아래와 같은 부분이 변경되었습니다. 그래서 이 장에서 먼저 SparTerm 모델의 구조를 살펴본 후, SPLADE 가 적용한 변경점을 따라가면 이해가 쉬울 것 같습니다.

1. weight term 수정
2. rank loss function에 "In-Batch Negatives" 방법 적용
3. sparse representation을 위한 FLOPS regularization term 추가

SparTerm



$$p' = \mathcal{F}(p) \odot \mathcal{G}(p)$$

SparTerm 모델은 Importance predictor(IP)와 Gating Controller(GC) 으로 이루어져 있습니다. 이 두 컴포넌트의 결과값을 아다마르 곱을 한 결과가 최종 "sparse representation" 이 됩니다.

Importance Predictor

Model

IP는 Bert의 encoder를 이용해서 dense embedding을 만듭니다.
이 임베딩을 이용해서 만든 쿼리 토큰의 임베딩과 전체 단어 임베딩 매트릭스를 내적(dot product) 후 ReLu 활성화 함수를 거쳐서 중요도 분포(Importance Distribution)를 만듭니다.
즉, 이 중요도 분포는 전체 vocabulary 공간에서 각 term의 semantic importance를 dense 벡터 형태로 나타냅니다.

$$I_i = Transform(h_i)E^T + b$$

$$I = \sum_{i=0}^{L} \text{Relu}(I_i)$$

"Transform"은 GELU activation을 의미하고 "h_i" 는 bert encoder를 통해 만들어진 토큰의 dense embedding, "E"는 전체 vocabulary embedding matrix 입니다.
토큰의 임베딩과 전체 vocabulary 임베딩의 내적을 통해 해당 토큰과 관련있는 vocabulary의 분포를 "I_i"로 나타내고, 이를 모든 input token 에 반복해 input passage 에 해당하는 분포를 도출합니다.

Training

$$L_{\text{rank}}(q_i, p_{i,+}, p_{i,-}) = -\log \frac{e^{\text{sim}(q_i', p_{i,+}')}}{e^{\text{sim}(q_i', p_{i,+}')} + e^{\text{sim}(q_i', p_{i,-}')}}$$

학습 단계에서는 최종 "sparse representation"에서 도출되는 쿼리(q_i')와 관련 있는(p_{i,+}'), 관련 없는(p_{i,-}') passage를 이용합니다.

Gating Controller

Model

GC는 최종 "sparse representation" 에 나타날 vocabulary 제어하기 위해 binary 게이팅 벡터를 생성합니다.

$$G' = \text{Binarizer}(G)$$

$$G_e = G' \odot (\neg \text{BoW}(p))$$

$$G_{le} = G_e + BoW(p)$$

IP와 같은 방식으로 생성된 (그러나 별도의) Gating Distribution G를 만든 후, 이 분포를 0,1 두 값 중 하나로 만드는 binarizer 를 통화해 전체 vocabulary 공간에 대해 0,1로 표현된 G'를 만듭니다.

그리고 1차적으로 G'와 vocabulary 공간에 속하지만 쿼리에서는 나타나지 않은 단어들과의 아다마르 곱을 통해 "쿼리에는 없지만 semantic 으로 연관이 있는 vocabulary를 활성화" 합니다.

그리고 2차로 1차에서 만들어진 벡터에 추가로 실제로 쿼리에 등장한 vocabulary를 추가합니다.

Training

$$L_{\text{exp}} = -\lambda_1 \sum_{j \in \{m|T_m=0\}} \log(1 - G_j) - \lambda_2 \sum_{k \in \{m|T_m=1\}} \log G_k$$

주어진 입력 텍스트에 대해, 타겟 텍스트(T_m)에 나타나는 용어들의 활성화 확률을 높이고(첫번째 term), 타겟 텍스트에 나타나지 않는 용어들의 활성화 확률을 낮추도록 학습시킵니다. (두번째 term)

(G_j는 GC가 특정 용어 w_j를 최종 표현에 포함시킬 확률을 나타내고, T_m=1은 타겟 텍스트의 Bag-of-Words 에서 포함된, T_m=0은 포함되지 않음을 뜻합니다.)

Overall

Model

$$w_{ij} = \text{transform}(h_i)^T E_j + b_j \quad j \in \{1, \ldots, |V|\}$$
$$w_j = g_j \times \sum_{i \in t} \text{ReLU}(w_{ij})$$

Training

$$L_{\text{rank}}(q_i, p_{i,+}, p_{i,-}) = -\log \frac{e^{\text{sim}(q_i', p_{i,+}')}}{e^{\text{sim}(q_i', p_{i,+}')} + e^{\text{sim}(q_i', p_{i,-}')}}$$

$$L_{exp} = -\lambda_1 \sum_{j \in \{m | T_m = 0\}} \log(1 - G_j) - \lambda_2 \sum_{k \in \{m | T_m = 1\}} \log G_k$$

$$L = L_{\text{rank}} + L_{\text{exp}}$$

SPLADE

변경점 1 - weight 계산 방식 변경

기존 SparTerm의 최종 weight 계산 방식에서 gating 메커니즘을 제거하고 대신 log 함수를 이용하였습니다.
이를 이용해서 두 가지 효과를 얻을 수 있다고 합니다.

1. log 함수 그래프의 성질을 이용해서 입력인자 간 상대적인 차이가 줄어들고 값들이 특정 범위 안에서 압축되는 경향을 이용해서 특정 용어가 지배하는 현상을 방지합니다.
2. "정규화 없이" sparse representation을 만들어 낼 수 있다.
    a. 이 부분이 신기합니다. 일반적으로 l1 regularization 을 loss function에 추가하면 상대적으로 약한 coefficient(weight)를 0으로 낮출 수 있는 거를 알려져 있는데요.
    여기서는 log 함수만으로 그 어떤 regularization 없이 가능하다고 합니다 (?!)
    b. 물론 이후에 L1 regularization 과 비슷한 역할을 하는 FLOPS 라는 정규화도 쓰긴 합니다.

SparTerm (IP * GC)

$$w_{ij} = \text{transform}(h_i)^T E_j + b_j \quad j \in \{1, \ldots, |V|\}$$
$$w_j = g_j \times \sum_{i \in t} \text{ReLU}(w_{ij})$$

SPLADE

$$w_j = \sum_{i \in t} \log(1 + \text{ReLU}(w_{ij}))$$

결론적으로 IP, GC 두 컴포넌트로 나누어져있던 SparTerm 과 달리 하나의 컴포넌트로 변경되었습니다.

변경점 2 - loss function 에 "In-Batch Negatives" 적용

다른 점은 동일하나 분모에 항이 하나 추가 되었습니다. 이 항은 다른 쿼리들의 긍정 문서들을 현재 쿼리에 대한 부정 문서로 간주하는 역할을 합니다. 이 방식은 많은 수의 negative 샘플을 확보할 수 있게 하여 학습 속도를 높이고 성능을 개선한다고 합니다.

SparTerm

$$L_{\text{rank}}(q_i, p_{i,+}, p_{i,-}) = -\log \frac{e^{\text{sim}(q_i', p_{i,+}')}}{e^{\text{sim}(q_i', p_{i,+}')} + e^{\text{sim}(q_i', p_{i,-}')}}$$

$$L_{exp} = -\lambda_1 \sum_{j \in \{m|T_m=0\}} \log(1 - G_j) - \lambda_2 \sum_{k \in \{m|T_m=1\}} \log G_k$$

$$L = L_{\text{rank}} + L_{\text{exp}}$$

SPLADE

$$\mathcal{L}_{rank-IBN} = -\log \frac{e^{s(q_i, d_i^+)}}{e^{s(q_i, d_i^+)} + e^{s(q_i, d_i^-)} + \sum_j e^{s(q_i, d_{i,j}^-)}}$$

변경점 3 - "sparse representation"을 위해 loss function에 FLOPS regularization term 추가

변경점 2의 loss function 에 더해 sparse representation이 되도록 약한 weight를 0로 밀어줄 FLOPS regularization term이 추가 되었습니다. L1 norm 보다 효율적이기 때문에 이 regularization을 사용했다고 합니다.

SparTerm

$$L_{\text{rank}}(q_i, p_{i,+}, p_{i,-}) = -\log \frac{e^{\text{sim}(q_i', p_{i,+}')}}{e^{\text{sim}(q_i', p_{i,+}')} + e^{\text{sim}(q_i', p_{i,-}')}}$$

$$L_{exp} = -\lambda_1 \sum_{j \in \{m|T_m=0\}} \log(1 - G_j) - \lambda_2 \sum_{k \in \{m|T_m=1\}} \log G_k$$

$$L = L_{\text{rank}} + L_{\text{exp}}$$

SPLADE

$$\mathcal{L}_{rank-IBN} = -\log \frac{e^{s(q_i, d_i^+)}}{e^{s(q_i, d_i^+)} + e^{s(q_i, d_i^-)} + \sum_j e^{s(q_i, d_{i,j}^-)}}$$

$$\ell_{\text{FLOPS}} = \sum_{j \in V} \bar{a}_j^2 = \sum_{j \in V} \left(\frac{1}{N} \sum_{i=1}^{N} w_j^{(d_i)} \right)^2$$

$$\mathcal{L} = \mathcal{L}_{rank-IBN} + \lambda_q \mathcal{L}_{reg}^q + \lambda_d \mathcal{L}_{reg}^d$$

앞서 log 함수를 통해 sparsity 를 만들 수 있다고 하였지만 별도로 regularization 이 사용되었습니다.
직관적으로 생각해보면 기존의 SparTerm은 0,1만을 출력하는 gating 컴포넌트가 output layer 에 존재하였기 때문에 sparsity 를 보장 할 수 있었지만 SPLADE 는 그런 장치가 없기 때문에 regularization의 효과를 빌리지 않았나 합니다.

실험 결과 및 결론

model	MS MARCO dev		TREC DL 2019		FLOPS
	MRR@10	R@1000	NDCG@10	R@1000	
Dense retrieval					
Siamese (ours)	0.312	0.941	0.637	0.711	-
ANCE [25]	0.330	0.959	0.648	-	-
TCT-ColBERT [15]	0.335	0.964	0.670	0.720	-
Sparse retrieval					
BM25	0.184	0.853	0.506	0.745	0.13
DeepCT [4]	0.243	0.913	0.551	0.756	-
doc2query-T5 [18]	0.277	0.947	0.642	0.827	0.81
ST lexical-only [1]	0.275	0.912	-	-	-
ST expansion [1]	0.279	0.925	-	-	-
Our methods					
ST lexical-only	0.290	0.923	0.595	0.774	1.84
ST exp-ℓ_1	0.314	0.959	0.668	0.800	4.62
ST exp-ℓ_{FLOPS}	0.312	0.954	0.671	0.813	2.83
SPLADE-ℓ_1	0.322	0.954	0.667	0.792	0.88
SPLADE-ℓ_{FLOPS}	0.322	0.955	0.665	0.813	0.73

SPLADE 모델은 다른 희소 검색 방법들(BM25, DeepCT, doc2query-T5, SparTerm)을 크게 능가할 뿐만 아니라, ANCE 및 TCT-ColBERT와 같은 최신 밀집 검색(dense retrieval) 방법들보다는 살짝 떨어지는 그러나 경쟁할만한 결과를 낼 수 있었습니다.

참고자료

https://www.microsoft.com/en-us/research/wp-content/uploads/2017/06/fntir2018-neuralir-mitra.pdf

https://arxiv.org/pdf/2107.05720

https://arxiv.org/pdf/2010.00768