7.1. Poly-encoders architectures and pre-training strategies for fast and accurate multi-sentence scoring

논문 : https://arxiv.org/pdf/1905.01969.pdf

참고자료

https://minhobby.tistory.com/50

https://blog.pingpong.us/ml-seminar-season-4/

Sentence간 연결의 방법론 2가지

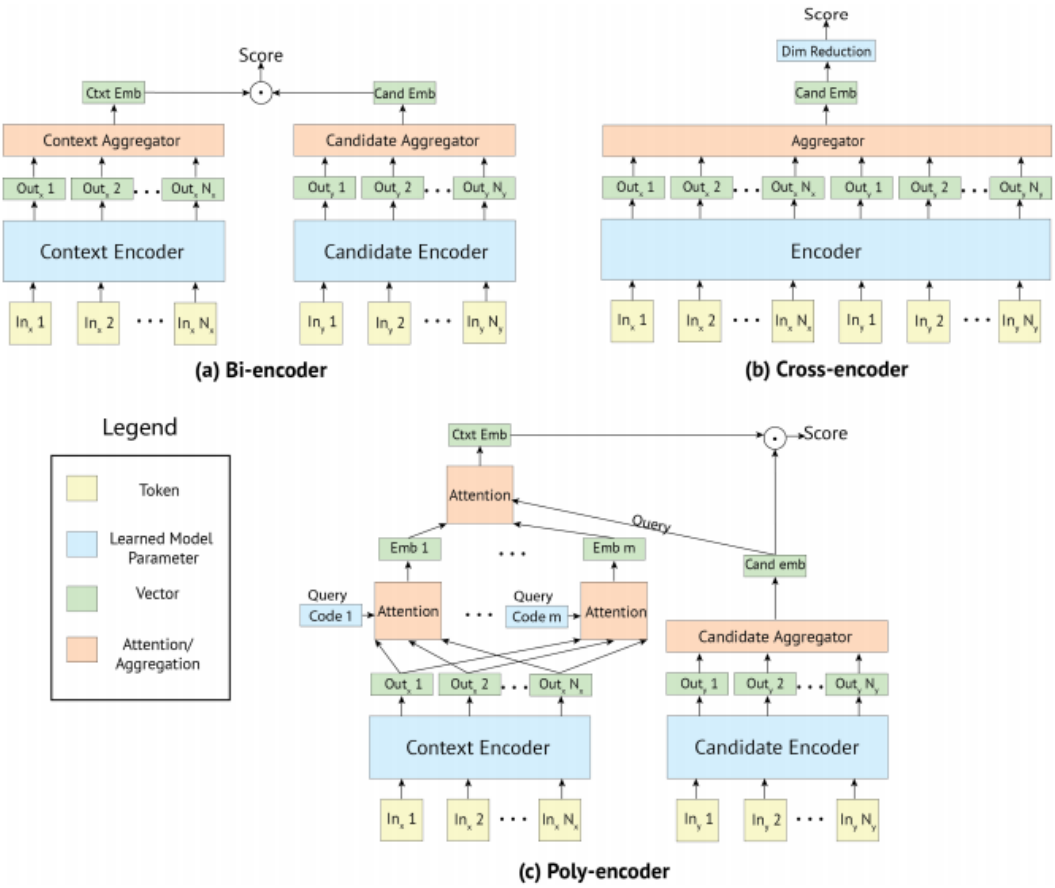


Figure 1: Diagrams of the three model architectures we consider. (a) The Bi-encoder encodes the context and candidate separately, allowing for the caching of candidate representations during inference. (b) The Cross-encoder jointly encodes the context and candidate in a single transformer, yielding richer interactions between context and candidate at the cost of slower computation. (c) The Poly-encoder combines the strengths of the Bi-encoder and Cross-encoder by both allowing for caching of candidate representations and adding a final attention mechanism between global features of the input and a given candidate to give richer interactions before computing a final score.

Cross-encoder

- 두 시퀀스를 동시에 입력으로 사용해 시퀀스 간의 full self-attention을 수행하는 방법
- Attention 크기가 토큰 길이 제곱으로 커져서 연산시간이 느림

$$y_{ctxt} = red(T_1(ctxt)) \quad y_{cand} = red(T_2(cand))$$

y_{ctxt} : Input Context에 대한 Representation

y_{cand} : Candidate에 대한 Representation

$T(x) = h_1, .., h_N$: Input Token에 대한 토큰 길이 N개에 대한 Token별 Representation 값 (Transformer 모델의 인코더류)

$red(\cdot)$: 전체 토큰 Representation을 하나의 백터로 줄이는 함수 (첫번째 토큰만 쓰거나 평균내거나 등)

$$s(ctxt, cand_i) = y_{ctxt} \cdot y_{cand_i}$$

Dot Product를 해서 pairwise scoring 계산

의문) 꼭 Dot Product를 써야하는가? 레이어를 추가하거나 KNN같은 머신러닝은 안되나?

Bi-encoder

- 두 시퀀스를 별도로 입력으로 사용해 두 시퀀스의 Representation 사이의 스코어를 계산하는 방법
- Cross-encoder보다 성능이 낮으나 연산속도는 빠름

$$y_{ctxt,cand} = h_1 = first(T(ctxt, cand))$$

Context와 Candidate를 같이 입력으로 전체 토큰중 첫번째 토큰(<CLS>류)를 추출

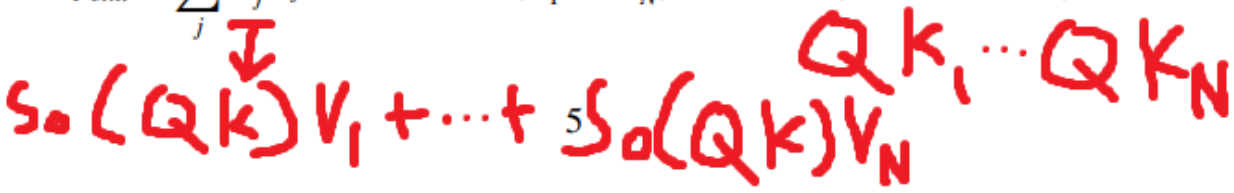
$$s(ctxt, cand_i) = y_{ctxt,cand_i} W$$

<CLS>토큰과 Weight를 행렬 곱(토큰 히든셀 크기, 클래스 수) 해서 scoring

Poly-encoder

- Bi-encoder와 비슷한 연산속도 높은 성능을 만들도록 모델 구조 제안
- Bi-encoder에 2중 Attention을 겹들임
    1. Learnable Vector와 Context Token Representation 간의 Attention
        - Query : Learnable Vector => Query 개수는 inference time에 맞춰 m 개 지정(하이퍼파라미터)
        - Key, Value : Context Token Representation
        - 일반적인 Attention임 => softmax(Q*K)*V
            - Q*K => (하이퍼파라미터 m, 토큰 수 n)
    2. 1번의 결과와 Candidate간의 Attention
        - Query : Candidate Vector
        - Key, Value : 1번 결과
        - 일반적인 Attention임 => softmax(Q*K)*V

$$y^i_{ctxt} = \sum_j w_j^{c_i} h_j \quad \text{where} \quad (w_1^{c_i}, .., w_N^{c_i}) = \text{softmax}(c_i \cdot h_1, .., c_i \cdot h_N)$$



$$y_{ctxt} = \sum_i w_i y^i_{ctxt} \quad \text{where} \quad (w_1, .., w_m) = \text{softmax}(y_{cand_i} \cdot y^1_{ctxt}, .., y_{cand_i} \cdot y^m_{ctxt})$$

실험

Bi-Encoder Cross-Encoder 비교

Negatives	31	63	127	255	511
R@1/20	81.0	81.7	82.3	83.0	**83.3**

Table 2: Validation performance on ConvAI2 after fine-tuning a Bi-encoder pre-trained with BERT, averaged over 5 runs. The batch size is the number of training negatives + 1 as we use the other elements of the batch as negatives during training.

Fine-tuned parameters	Bi-encoder	Cross-encoder
Top layer	74.2	80.6
Top 4 layers	82.0	86.3
All but Embeddings	**83.3**	**87.3**
Every Layer	83.0	86.6

Table 3: Validation performance (R@1/20) on ConvAI2 using pre-trained weights of BERT-base with different parameters fine-tuned. Average over 5 runs (Bi-encoders) or 3 runs (Cross-encoders).

Bi-Encoder Cross-Encoder Poly-Encoder 비교

Dataset	ConvAI2	DSTC 7		Ubuntu v2		Wikipedia IR
split	test	test		test		test
metric	R@1/20	R@1/100	MRR	R@1/10	MRR	R@1/10001
(Wolf et al., 2019)	80.7					
(Gu et al., 2018)	-	60.8	69.1	-	-	-
(Chen & Wang, 2019)	-	64.5	73.5	-	-	-
(Yoon et al., 2018)	-	-	-	65.2	-	-
(Dong & Huang, 2018)	-	-	-	75.9	84.8	-
(Wu et al., 2018)	-	-	-	-	-	56.8
pre-trained BERT weights from (Devlin et al., 2019) - Toronto Books + Wikipedia						
Bi-encoder	81.7 ± 0.2	66.8 ± 0.7	74.6 ± 0.5	80.6 ± 0.4	88.0 ± 0.3	-
Poly-encoder 16	83.2 ± 0.1	67.8 ± 0.3	75.1 ± 0.2	81.2 ± 0.2	88.3 ± 0.1	-
Poly-encoder 64	83.7 ± 0.2	67.0 ± 0.9	74.7 ± 0.6	81.3 ± 0.2	88.4 ± 0.1	-
Poly-encoder 360	83.7 ± 0.2	68.9 ± 0.4	76.2 ± 0.2	80.9 ± 0.0	88.1 ± 0.1	-
Cross-encoder	84.8 ± 0.3	67.4 ± 0.7	75.6 ± 0.4	82.8 ± 0.3	89.4 ± 0.2	-
Our pre-training on Toronto Books + Wikipedia						
Bi-encoder	82.0 ± 0.1	64.5 ± 0.5	72.6 ± 0.4	80.8 ± 0.5	88.2 ± 0.4	-
Poly-encoder 16	82.7 ± 0.1	65.3 ± 0.9	73.2 ± 0.7	83.4 ± 0.2	89.9 ± 0.1	-
Poly-encoder 64	83.3 ± 0.1	65.8 ± 0.7	73.5 ± 0.5	83.4 ± 0.1	89.9 ± 0.0	-
Poly-encoder 360	83.8 ± 0.1	65.8 ± 0.7	73.6 ± 0.6	83.7 ± 0.0	90.1 ± 0.0	-
Cross-encoder	84.9 ± 0.3	65.3 ± 1.0	73.8 ± 0.6	83.1 ± 0.7	89.7 ± 0.5	-
Our pre-training on Reddit						
Bi-encoder	84.8 ± 0.1	70.9 ± 0.5	78.1 ± 0.3	83.6 ± 0.7	90.1 ± 0.4	71.0
Poly-encoder 16	86.3 ± 0.3	71.6 ± 0.6	78.4 ± 0.4	86.0 ± 0.1	91.5 ± 0.1	71.5
Poly-encoder 64	86.5 ± 0.2	71.2 ± 0.8	78.2 ± 0.7	85.9 ± 0.1	91.5 ± 0.1	71.3
Poly-encoder 360	86.8 ± 0.1	71.4 ± 1.0	78.3 ± 0.7	85.9 ± 0.1	91.5 ± 0.0	**71.8**
Cross-encoder	**87.9 ± 0.2**	**71.7 ± 0.3**	**79.0 ± 0.2**	**86.5 ± 0.1**	**91.9 ± 0.0**	-

Table 4: Test performance of Bi-, Poly- and Cross-encoders on our selected tasks.

	Scoring time (ms)			
	CPU		GPU	
Candidates	1k	100k	1k	100k
Bi-encoder	115	160	19	22
Poly-encoder 16	122	678	18	38
Poly-encoder 64	126	692	23	46
Poly-encoder 360	160	837	57	88
Cross-encoder	21.7k	2.2M*	2.6k	266k*

Table 5: Average time in milliseconds to predict the next dialogue utterance from C possible candidates on ConvAI2. * are inferred.

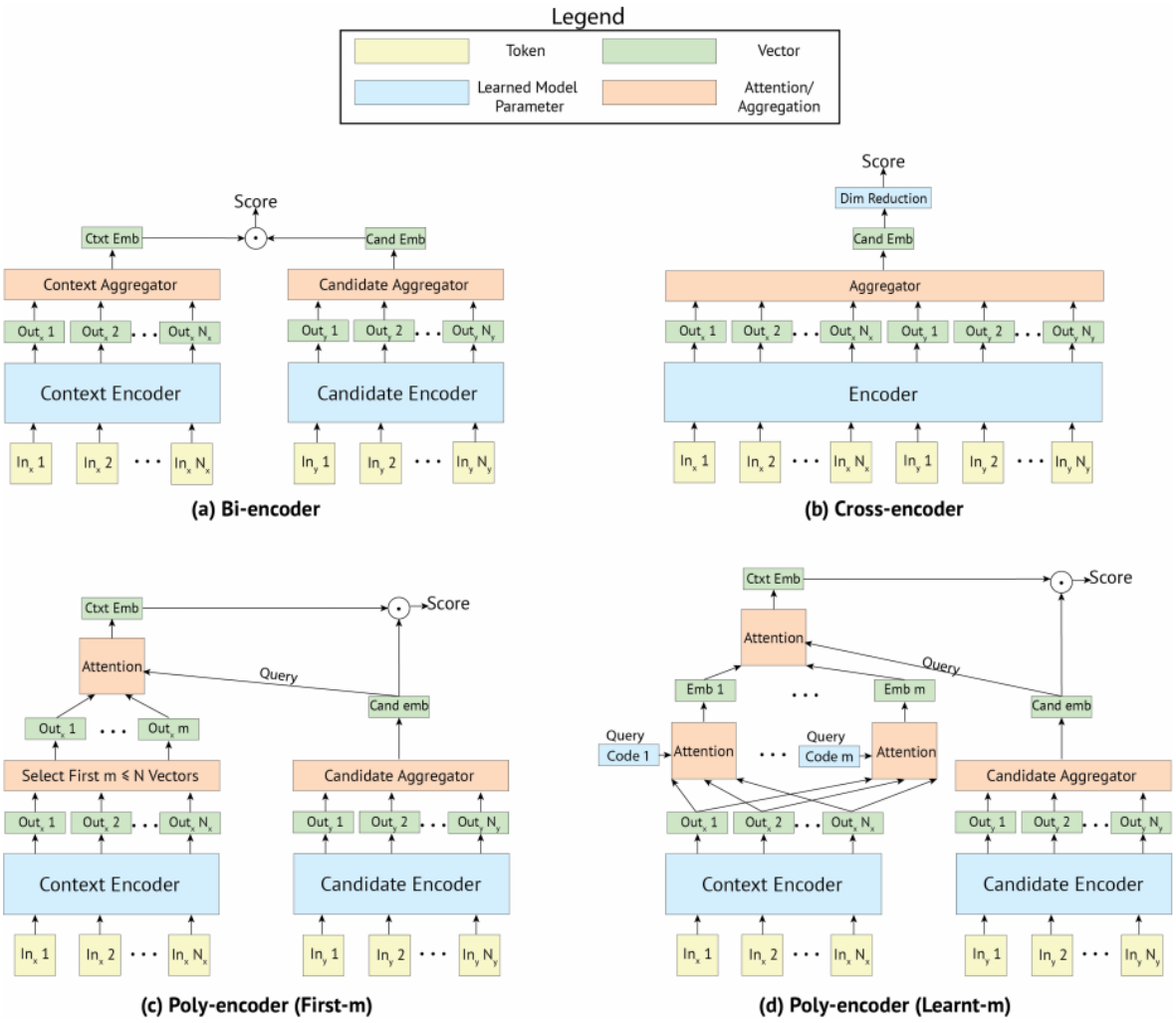


Figure 2: (a) The Bi-encoder (b) The Cross-encoder (c) The Poly-encoder with first m vectors. (d) The Poly-encoder with m learnt codes.