3.5. PoWER BERT

논문 : https://arxiv.org/abs/2001.08950

2020년 1월 출시

PoWER BERT : Progressive Word-Vector Elimination for inference time Reduction of BERT

요약

1. Language Model로 Classification Task를 해결할 때, 마지막 레이어로 갈수록 토큰간 데이터가 유사해지는 것을 확인
2. 굳이 입력 토큰을 유지하지말고 Attention Matrix(= Attention Weight)를 바탕으로 중요토큰만 살려서 inference time을 줄여보자!
3. BERT, ALBERT에 장착해본결과 1.6~6.8배 까지 빨라지고 정확도는 1%내외로 감소!

문제 제기

- BERT의 모델이 사용하는 파라미터는 많고, inference time도 느림
- ALBERT는 파라미터 공유 기법을 제안해서 모델의 파라미터 수를 감소시켰으나, inference time은 유지됨
- DistilBERT, BERT-PKD와 같은 기법은 모델의 파라미터 수와 inference time을 감소시켰으나, accuracy가 상당히 낮아짐
- BERT로 **Classification Task**를 해결하고자 Fine-tuning 할 경우, 각 Encoder에서 연산된 Word-Vector 간의 Cosine Similarity를 그림 2로 나타낼 수 있음
    - 그림 2와 같이 마지막 레이어로 갈수록 Word-Vector들이 유사한 값을 띄는 것을 볼 수 있음 => diffusion of information을 의미함
    - 그러면 마지막 레이어로 갈수록 Word를 적게 써도(생략해도) 중요한 정보가 유지되어서, Classification Task를 해결하는데 문제가 없을 것이다!
    - 참고 : "word-vector to refer to an intermediate vector output generated by the encoders"

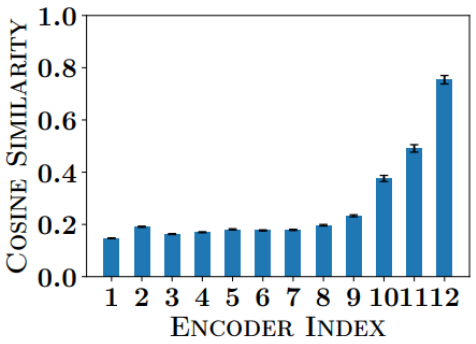


Figure 2: Cosine similarity for BERT encoders on the SST-2 dataset. The j^{th} bar represents cosine similarity for the j^{th} encoder, averaged over all pairs of word-vectors and all inputs.

제안기법

- 그림 1
    - BERT(위)는 Encoder Layer를 지나가도 Sequence Length X Hidden Size를 유지
    - PoWER BERT(아래)는 Encoder Layer를 지나갈 때 일부 Word-Vector를 eliminate
- 본 논문은 토큰을 제거하는 기법에 대해 제안함
    - 모델구조 자체를 건들이지 않기때문에 ALBERT와 같이 다른논문이 제안한 모델의 파라미터 감소 기법들을 적용할 수 있음
- Word-Vector를 제거하려면 설정해야되는 두가지
    - 얼만큼 제거할 것이냐? (= 얼만큼 보존시킬 것이냐?) : Encoder Layer 마다 추출되는 Sequence Length => ***Retention configuration***
        - ex) 그림 1 처럼 (128, 80, 73, 70, 50, 50, 40, 33, 27, 20, 15, 13, 3)의 근거를 만들기
    - 어떤 Word-Vector를 제거할 것이냐?(= 어떤 Word-Vector를 보존할 것이냐?) : 남길 Word-Vector를 구분하는 기준점 => ***Word-vector selection***

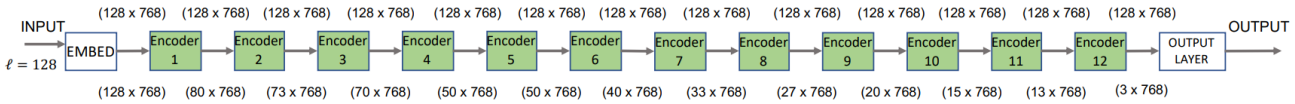


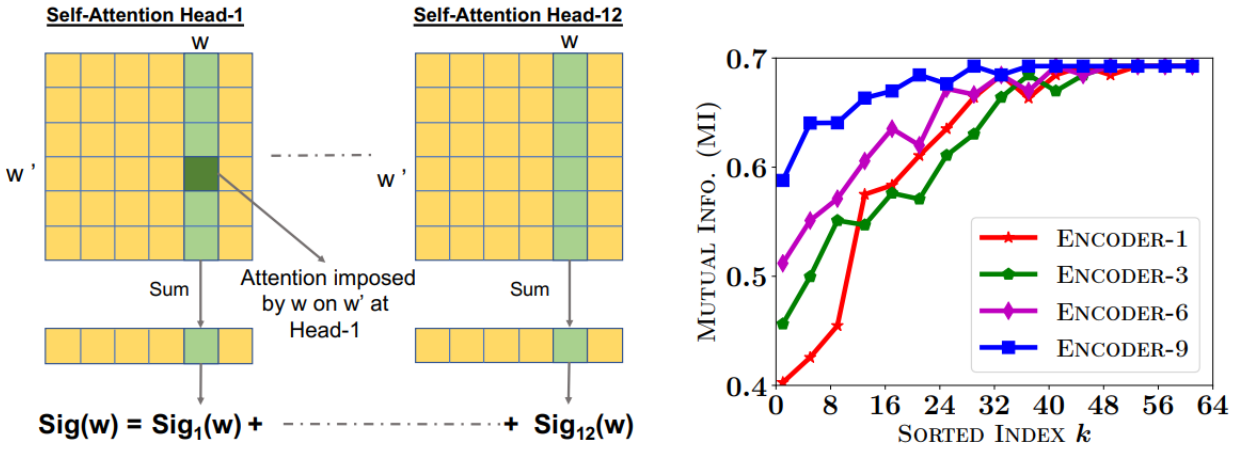
Figure 1: Illustration of PoWER-BERT scheme over $\text{BERT}_{\text{BASE}}$ that has $L = 12$ encoder layers, each having $A = 12$ self-attention heads and a hidden size of $H = 768$.The words are first embedded as vectors of length $H = 768$. The numbers on top are output sizes for each encoder layer of $\text{BERT}_{\text{BASE}}$ for input sequence length $\ell = 128$. The numbers at the bottom are output sizes for each encode layer of PoWER-BERT. In this example, the first encoder eliminates 48 and retains 80 word-vectors, whereas the second encoder eliminates 7 more and retains 73 word-vectors. The hidden size remains at 768.

Word-vector Selection

- 본논문에서는 Static Strategies와 Dynamic Strategies를 구분함
  - Static Strategies
    - 데이터셋마다 특정 위치가 중요할 것이다라는 관점.
    - 고정적으로 보존시킬 위치를 Random으로 지정하거나 (Rand-WS, Random Word-Vector Selection), 처음 워드부터 sequence length까지 추출하는 방법(Head-WS, Head Word-Vector Selection)
    - 본 논문에선 크게 다루지 않음
  - **Dynamic Strategies**
    - 특정 인코더마다 중요한 토큰을 뽑는다? 인코더마다 Self-Attention이 있다? Self-Attention의 Attention Matrix(=Attention Weight)로 주요 Word-vector을 판별하자!
    - Attention Mechanism (그림 6) 에서 Softmax연산이 끝난 값의 크기는 Sequence Length X Sequence Length = Attention Distribution Matrix
      - Attention Distribution Matrix은 어떤 단어(행)가 다른단어(열)를 얼만큼 Attention할지 가중치가 기록되어있음
      - 해당 Attention Distribution Matrix를 **열단위**로 합치고, 그 값을 **모든 head끼리 더하면** 해당 단어가 얼마나 가중치를 받았는지(중요한지) 알수있을 것임! (그림 3 왼쪽)
        - 이러한 연산 값을 **Significance score**라 명시함
      - 가중치를 높게받았다 => Attention Mechanism에서 중요한 단어이다 => 해당 Word를 보존시키면 되겠다!
    - 본 논문은 Dynamic Strategies로 **Attn-WS**(Attetion Word-Vector Selection)를 제안하고 사용함.
    - **그림 4와 같이 Multi-head Self-Attention Layer 뒤에 Extract Layer를 배치해서 Word-Vector를 추출하는 방식으로 구현**
    - **실험 : Attn-WS가 정말 유효한가?**
      - 실험방식 : 1,3,6,9 번째 Encoder의 Significance score을 연산 및 내림차순으로 정렬한 뒤 k번째 Word-Vector를 지웠을 때의 Classification Output과 BERT base의 Classification Output을 가지고 Mutual Information을 계산해봄
        - 실험 데이터 셋 : SST-2 (문장에 대한 긍/부정이 붙어있는 데이터)
        - 1,3,6,9번째 Encoder만 사용한 이유 : 가독성때문에 나머지 인코더 레이어는 생략
        - Mutual Information ? : 두 확률변수가 서로 어떤 관계를 가지고 있는지 나타내는 정보량 중 하나.
          - Baseline Entropy를 0.69로 설정함.
            - binary classification에서 0과 1이 나올 확률이 같은 데이터셋이면, H(X) = -(0.5 * ln 0.5) * 2 = 0.693147...
          - 두 확률변수의 MI가 어느 하나의 확률변수의 정보량(Entropy)에 가까울수록, 두 확률변수는 서로를 구별하는 정보가 적다고 말할 수 있다(BERT Base와 비슷한 성능을 보이는 것으로 판단하면됨)
            - H(X|X) = 0; X에 대해 설명하지 않는 X는 없다. H(Y|X) ~ X가 알려진 조건에서, Y에 남아있는 불확실성의 양. X가 Y에 대해서 많이 말한다면 H(Y|X)는 낮을 것이고, 거의 말하지 않는다면 높을 것이다.
            - I(X;Y) = I(Y;X) = H(X) - H(Y|X) = H(Y) - H(X|Y)
            - I(X;X) = H(X); I(X;X) >= I(X;Y); 동일한 분포를 갖는 확률변수라면, 두 확률변수는 서로 최대의 정보량을 가지고 있다.
      - 실험결과
        - k가 커질수록(= Significance score가 작은 Word-Vector를 제거할수록) Mutual Information에 영향을 주지 않는것을 확인할 수 있음
        - => 중요하지 않는 Word-Vector를 지워도 된다는 것을 확인

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

그림 6. Scaled Dot-Product Attention



(a) Significance score (b) Mutual information

Figure 3: Word-vector selection. Part (a) shows score computation for word-vector w. Part (b) presents mutual information when the k^{th} significant word-vector is eliminated at the specified encoder.

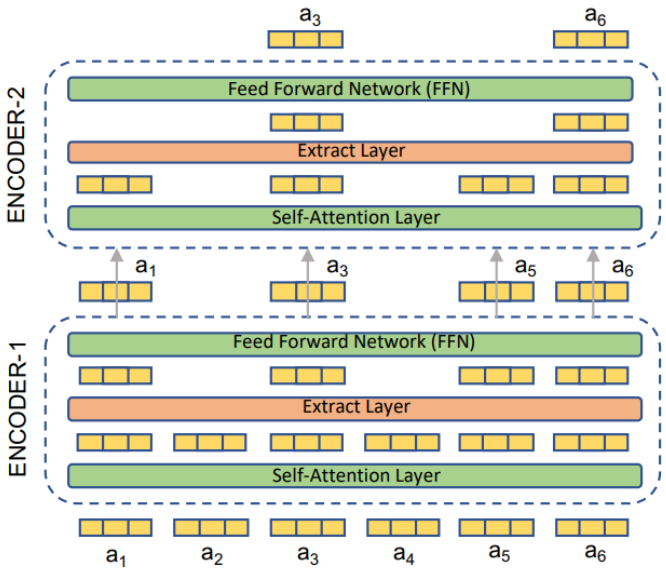


Figure 4: Word-vector selection over the first two encoders. Here, $\ell = 6$, $\ell_1 = 4$ and $\ell_2 = 2$. The first encoder eliminates two word-vectors a_2 and a_4; the second encoder further eliminates word-vectors a_1 and a_5.

Retention Configuration

- 인코더 레이어당 보존시킬 Word-Vector를 선정해야되는 상황
- 논문은 인코더 레이어당 Sequence Length를 **학습**시키기 위해 Fine-tuning 뒤에 **Retention Search** 과정 을 제안함
  - Retention Configuration : fine-tuning 과정에서 사용하는 데이터셋, Loss Function에 **2가지 기법을 추가**해서 적절한 인코더 레이어의 Sequence length를 구함
    1. Soft Extract Layer **(수식 1, 그림 5)**
       - Retention Configuration 단계에서 Extract Layer 대신 사용되는 것으로 0~1 사이의 값을 학습할 수 있는 파라미터 r
       - Retention Configuration 초기단계에서 모든 인코더의 r들을 1로 초기화 한뒤, Loss Function에 의해 학습이 이뤄짐
       - Extract Layer와 다르게 r이 가진 값을 Word-vector에 곱해 값에 변화를 주는 방식을 사용함 (0에 가까울수록 Word-Vector 값이 없어짐)
       - **Retention Search 에서 Re-training 단계로 넘어갈땐 인코더 레이어별로 r을 합쳐서 소수점 올림한 정수값을 해당 인코더 레이어의 Sequence Length로 지정**
    2. Loss Function **(수식 2)**

- Fine-Tuning 과정에서 사용했던 Loss Function외에 Soft Extract Layer의 r을 학습할 수 있도록 Loss Function을 수정해야함
- 기존의 Loss + 모든 인코더의 r 총합 (r이 크다? 인코더별로 사용하는 Sequence Length가 많다는 뜻)
- j번째 인코더가 말단으로 갈수록 토큰간의 유사도가 높아지므로 Loss 증가
- λ가 클 수록 Loss가 커지므로 학습과정에서 Elimination이 많이 발생

$$\mathbf{E}^{out}[i,:] = \mathbf{r}_j[\mathtt{Sig}_{idx}(a_i)] \cdot \mathbf{E}^{in}[i,:].$$

수식 1. Soft Extract Layer 수식. E_in : Self-Attention Layer의 출력 (Sequence Length X Hidden size), E_out : Feed Forward Network의 입력 (Sequence X Hidden size),

a_i : i번째 행렬의 Attention matrix, Sig_idx : 내림차순으로 정렬된 Word 별 Significance Score 백터, r_j : j번째 Significance Score를 가진 Word-Vector에 대한 가중치 [0,1]

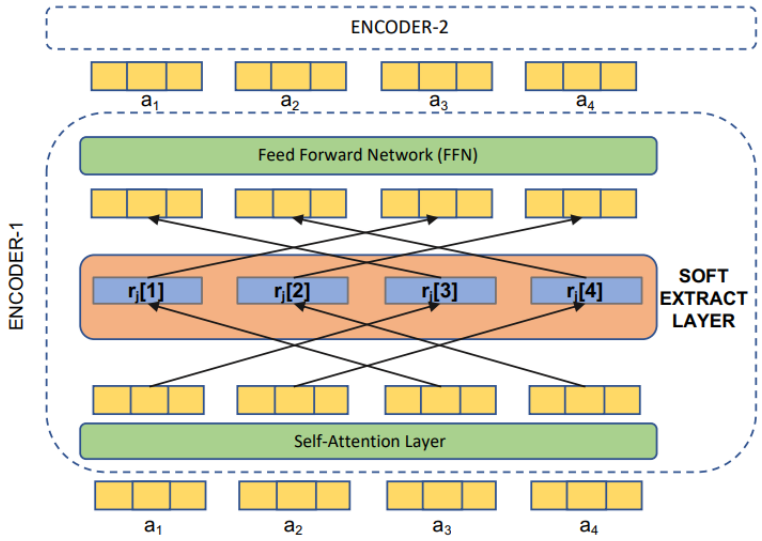


Figure 5: Illustration of the `soft-extract` layer. First encoder is shown, taking $\ell = 4$. Here, the sorted sequence of the word-vectors are a_3, a_4, a_1, a_2; hence their ordered positions are $3, 4, 1, 2$. For each word-vector, the parameter by which it is multiplied is shown.

$$\mathrm{mass}(j\ ;\ \mathbf{r}) = \sum_{k=1}^{\ell} \mathbf{r}_j[k].$$

$$\min_{\mathbf{\Theta},\mathbf{r}} \quad \left[\mathcal{L}(\mathbf{\Theta}, \mathbf{r}) \ + \ \lambda \cdot \sum_{j=1}^{L} j \cdot \mathrm{mass}(j\ ;\ \mathbf{r}) \right] \quad \text{s.t.} \ \ \mathbf{r}_j[k] \in [0,1],$$

where L is the number of encoders, $j \in [1, L]$, and $k \in [1, \ell]$. While $\mathcal{L}(\mathbf{\Theta}, \mathbf{r})$ controls the accuracy, the regularizer term controls the aggregate mass. The hyper-parameter λ tunes the trade-off.

수식 2. 논문이 제안한 Retention Configuration에서의 Loss Function. L : Fine-Tuning에서 사용하던 Loss Function, λ(람다) : 하이퍼파라미터 논문이 사용한 범위 [0.0001,0.01], mass(j;r) : j번째 인코더에서 사용한 r의 총합

학습의 형태

PoWER BERT 모델을 생성하려면....

1. Fine-Turning : pre-trained BERT을 데이터셋에 맞게 학습

2. Configuration-search : 인코더 레이어 별 Sequence Length를 학습하기위해 fine-tuning에 soft extract layer 추가 및 loss function 변경
3. Re-training : Configuration-search에서 사용한 soft extract layer 대신 extract layer로 교체, 인코더 레이어별 sequence length를 지정하고 fine-tuning에서 사용한 loss function으로 재학습

실험

K80 GPU + Batch Size 128로 설정했을 때 모델 성능과 inference time 비교 (테이블 1, 2)

	Method	CoLA	RTE	QQP	MRPC	SST-2	MNLI-m	MNLI-mm	QNLI	STS-B	IMDB	RACE
Test Accuracy	BERT	52.5	68.1	71.2	88.7	93.0	84.6	84.0	91.0	85.8	93.5	66.9
	PoWER-BERT	52.3	67.4	70.2	88.1	92.1	83.8	83.1	90.1	85.1	92.5	66.0
Inference Time (ms)	BERT	898	3993	1833	1833	898	1867	1867	1833	898	9110	20040
	PoWER-BERT	201	1189	405	674	374	725	908	916	448	3419	10110
Speedup		**(4.5x)**	(3.4x)	**(4.5x)**	(2.7x)	(2.4x)	(2.6x)	(2.1x)	(2x)	(2x)	(2.7x)	(2.0x)

Table 1: Performance comparison between PoWER-BERT and $\text{BERT}_{\text{BASE}}$. Inference done on a K80 GPU with batch of 128 for all datasets except RACE (batch of 64 is used). Matthew's Correlation reported for CoLA, F1-score for QQP and MRPC, Spearman Correlation for STS-B, and Accuracy for the rest. Input sequence length used are 64 for CoLA, SST-2 and STS-B, 512 for IMDB and RACE, 256 for RTE and 128 for the rest.

	Method	CoLA	RTE	QQP	MRPC	SST-2	STS-B	MNLI-m	MNLI-mm	QNLI
Test Accuracy	$\text{ALBERT}_{\text{BASE}}$	42.8	65.6	68.3	89.0	93.7	80.9	82.6	82.5	89.2
	PoWER-BERT	43.8	64.6	67.4	88.1	92.7	80.0	81.8	81.6	89.1
Inference Time (ms)	$\text{ALBERT}_{\text{BASE}}$	940	4210	1950	1950	940	940	1960	1960	1950
	PoWER-BERT	165	1778	287	813	442	604	589	922	1049
Speedup		(5.7x)	(2.4x)	**(6.8x)**	(2.4x)	(2.1x)	(1.6x)	(3.3x)	(2.1x)	(1.9x)

Table 2: Performance comparison between PoWER-BERT and $\text{ALBERT}_{\text{BASE}}$. Here PoWER-BERT represents application of our scheme on ALBERT. The experimental setup is same as in Table 1