부스트캠프 AI Tech 2기

boostcamp ai tech

ALBERT

ALBERT_A LITE BERT FOR SELF-SUPERVISED LEARNING OF LANGUAGE REPRESENTATIONS

Email : ghy546@naver.com

GitHub : github.com/changyong93

Blog : velog.io/@changyong93

Email : j961224@naver.com

GitHub : github.com/j961224

0. Abstract

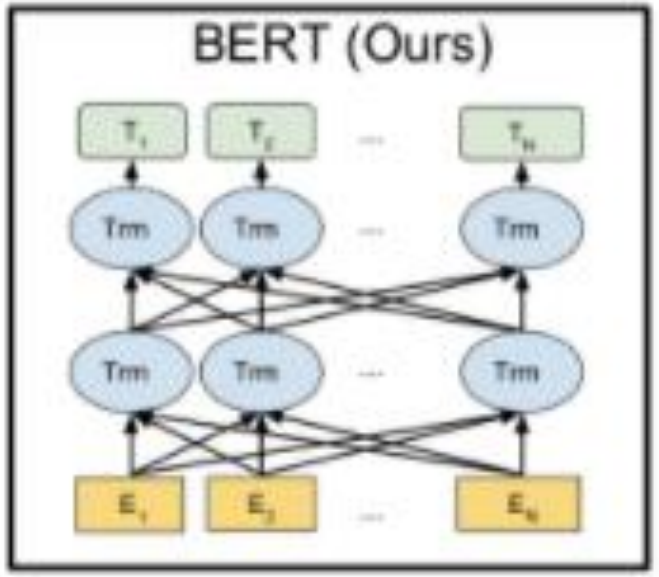
ABSTRACT

Increasing model size when pretraining natural language representations often results in improved performance on downstream tasks. However, at some point further model increases become harder due to GPU/TPU memory limitations and longer training times. To address these problems, we present two parameter-reduction techniques to lower memory consumption and increase the training speed of BERT (Devlin et al., 2019). Comprehensive empirical evidence shows that our proposed methods lead to models that scale much better compared to the original BERT. We also use a self-supervised loss that focuses on modeling inter-sentence coherence, and show it consistently helps downstream tasks with multi-sentence inputs. As a result, our best model establishes new state-of-the-art results on the GLUE, RACE, and SQuAD benchmarks while having fewer parameters compared to BERT-large. The code and the pretrained models are available at https://github.com/google-research/ALBERT.

- 모델 사이즈를 키울 시 downstream task의 성능 향상
  ➔ **But, 더 많은 GPU/TPU 자원 필요 & 학습시간 증가!**

- Sol 1) Parameter reduction
- Sol 2) Sentence 간의 일관성을 모델링하는 self-supervised loss

1. Introduction



- 제한된 사전학습 데이터를 활용하기 위해 모델 사이즈 증량 ➔ SOTA 달성
  - 모델 사이즈를 증량하여 pre-training
  - Distill pre-train large model for application
  - 의문제기 ➔이 과정을 small model로 하여 특정 task에 최적화된 모델로 학습은 불가?

- 기존 방법에 대한 문제점 제기
  - 가용 메모리에 따른 모델 사이즈 증량 제한(memory limitation)
  - Communication overhead로 인한 학습 속도 영향

    *Communication overhead : the proportion of time you spend communicating with your team instead of getting productive work done.

- Model Parallelization & clever memory management
  ➔ Memory limitation 해결 || Communication overhead 해결 X

- ALBERT(A Lite BERT)
  - Two parameter reduction technique 적용
    1. Factorized embedding parameterization
    2. Cross-layer parameter sharing
  - SOP(Sentence Order Prediction)
  - 이러한 방법으로 기존 문제를 해결하며, NLU dataset(GLUE, SQuAD, RACE)에 대해서 SOTA!!

2. Related Work

- Scaling up representation learning for natural language
  - larger model ➔ better performance
  - But, 1024-dim is optimal ➔ model size and computation cost problem
  - Natural languag에 대한 representation을 확장하는 것은 model size를 늘리는 것 만큼 단순하진 않음

  ➔ 우리가 제안한 방법을 적용하면 메모리 문제 해결 및 학습 속도 개선 가능!

- Cross-layer parameter sharing
  - Vaswani et al., 2017(attention is all you need)
    - Transformer 구조에서 착안, 단, transformer는 pretraining/finetuning보단 encoder/decoder 학습에 관심
  - Dehghani et al., 2018(UT, Universal Transformers)
    - Cross-layer parameter sharing은 language modeling과 subject-verb agreement에 더 나은 성능
  - Bai et al., 2019
    - DQE(Deep Equilibrium model)을 제안, 특정 layer의 input과 output embedding이 동일한 평형점 도달할 수 있음을 제시
    - 논문 저자에 의하면 BERT는 embedding이 수렴하기보단 진동한다고 함
  - Bai et al., 2019, modeling recurrence for transformer
    - parameter-sharing transformer를 standard transformer와 결합하여 standard transformer보다 더 높은 성능을 달성.

- Sentence Ordering Objectives
  - 기존에 다양한 paper들이 있지만, sentences에 초점에 맞춰져 있는 반면, 해당 논문에선 textual segments에 집중
  - BERT NSP의 negative는 두 문장 사이의 논리적 관계가 아닌 단순 비교로 매우 쉬운 학습

3-0. ALBERT 요소 – Model Architecture Choices

- ALBERT architecture : transformer의 encoder와 같은 구조를 사용하는 BERT와 유사함
- Activation function : GELU
  - RELU + dropout + zoneout

- Embedding size : E

- Num of layer : L

- Hidden size : H

- Feed-forward/filter size : 4H

- Num of attention head : H/64

3-1. ALBERT 요소 – Factorized embedding parameterization

Problem

- For WordPiece embedding(BERT, XLNet, RoBERTa) ➔ Embedding size E = hidden size H

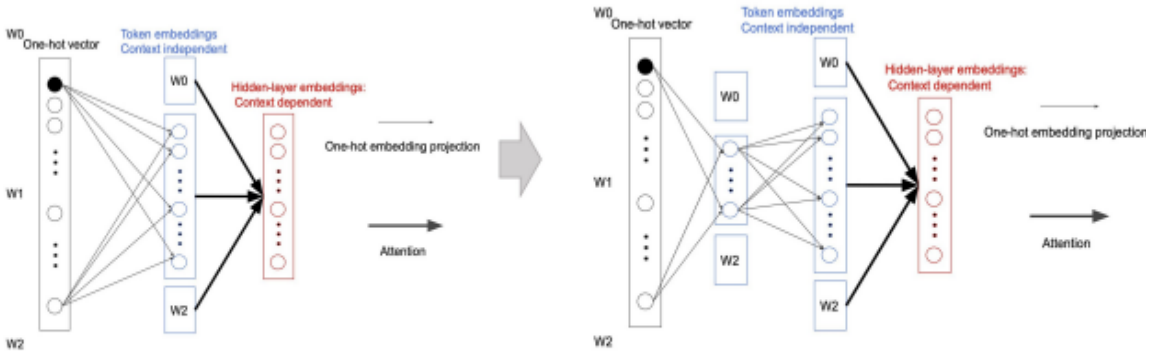
Why?

- WordPiece embedding ➔ context-independent representations 학습
- hidden-layer embedding ➔ context-dependent representations 학습

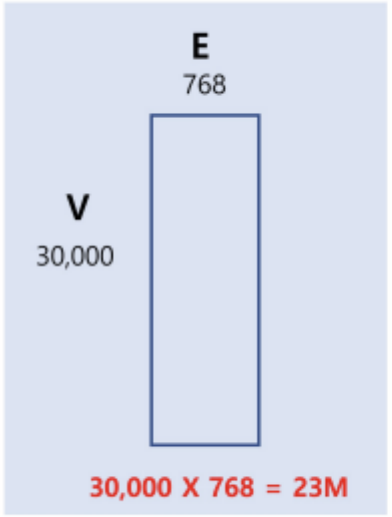
따라서, Modeling Needs에 맞는 E와 H의 사이즈 선정 시 parameter reduction 가능! (H〉〉E)

3-1. ALBERT 요소 – Factorized embedding parameterization

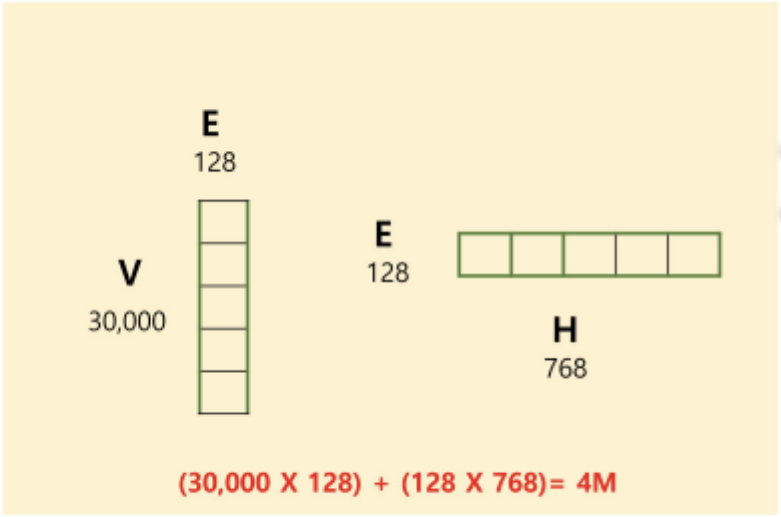
큰 vocabulary embedding matrix -> 2개의 small matrices로 분리!



: O(V x H) => O(V x E + E x H)

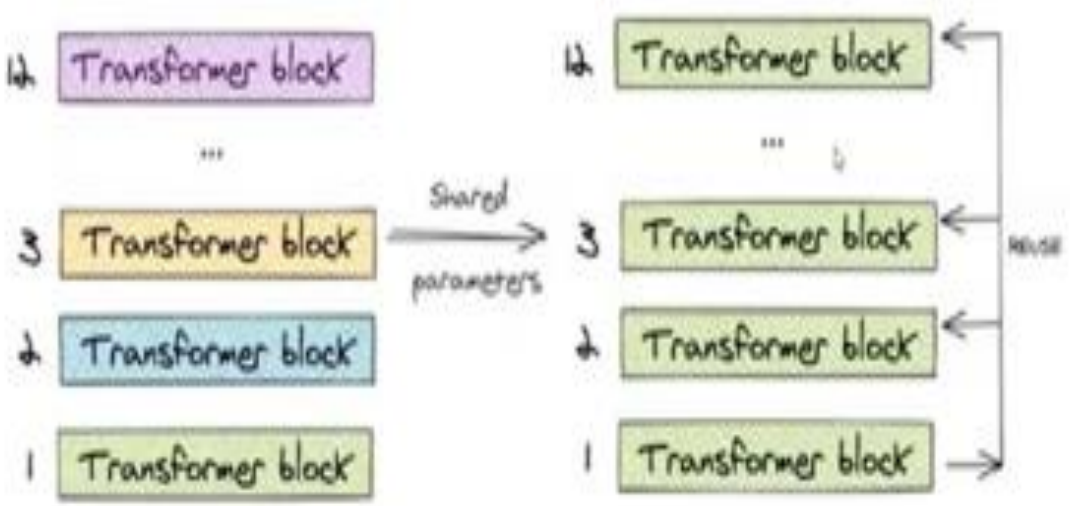


BERT base



Factorized embedding parameterization을 통해 파라미터가 많이 줄었다!

3-1. ALBERT 요소 – Cross-layer parameter sharing



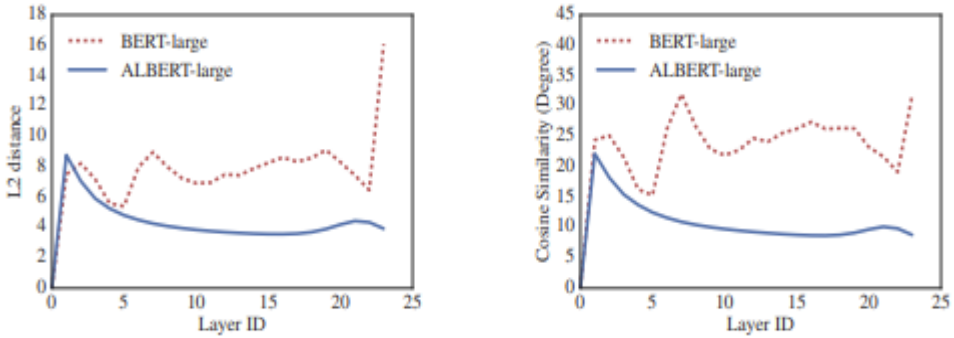


Figure 1: The L2 distances and cosine similarity (in terms of degree) of the input and output embedding of each layer for BERT-large and ALBERT-large.

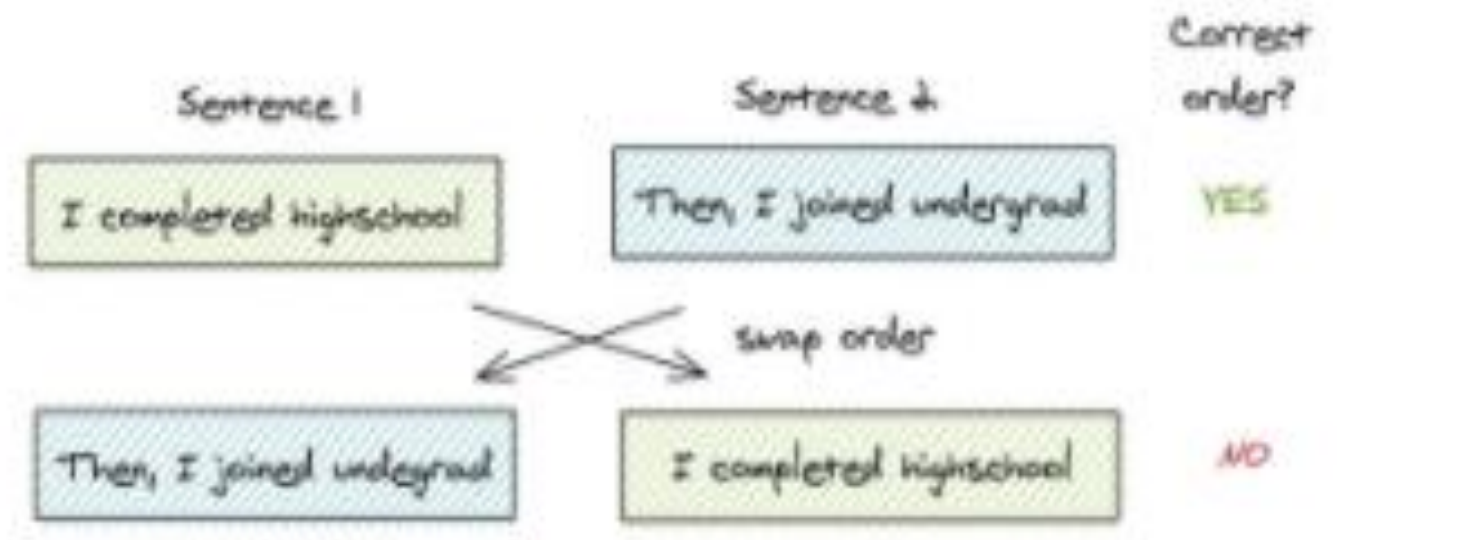
Proposition : To improve Parameter efficiency
Method : 1) shared-FFN 2) shared-attention 3) All-shared

Two metric에 대해, ALBERT가 안정적으로 한 값으로 수렴
BERT는 모든 layer를 통과하여도 발산

3-1. ALBERT 요소 – Inter-sentence coherence loss

NSP(Next Sentence Prediction)은 비교적 쉬운 task
➔MLM 대비 NSP은 쉬운 task로 학습에 부적합

문장간 통일성에 집중하는 SOP(Sentence Order Prediction)를 제안!



3-2. Model Setup

Model		Parameters	Layers	Hidden	Embedding	Parameter-sharing
BERT	base	108M	12	768	768	False
	large	334M	24	1024	1024	False
ALBERT	base	12M	12	768	128	True
	large	18M	24	1024	128	True
	xlarge	60M	24	2048	128	True
	xxlarge	235M	12	4096	128	True

- BERT Model 대비 Parameter 대폭 감소
  ➔ parameter 수 : bert-large 〉 18 x albert-large
- ALBERT-xxlarge는 24-layer와 12-layers의 성능이 비슷하여 12layers로 setup

4-1. EXPERIMENTAL SETUP

- Corpus for Pretraining baseline models : Bookcorpus, English Wikipedia
  ➔ 16GB, uncompressed text

- input format : [CLS] sentence1 [SEP] sentence2 [SEP] (Like BERT)
  ➔ max input length : 512 (shorter than 512 with a probability of 10%)

- Vocab size : 30,000(like BERT)

- Tokenizer : SentencePiece ➔ BPE + unigram language model

- MLM ➔ maximum length of n-gram : 3

- LAMB optimizer ➔ LARS method + adam optimizer (https://junseong.oopy.io/paper-review/lamb-optimizer)
  ➔ BERT와 같은 Large Batch Model에 최적화
  ➔ lr을 안정적으로 높여 학습속도 향상
  ➔ lr : 0.00176
- Cloud TPU V3(64 to 512 depending on model size)

- Evaluation for pre-training ➔ SQuAD, RACE(단, 모델 학습이 수렴하는 지 확인하는 용도로만 사용)
- Evaluation for fine-tuning ➔ GLUE, SQuAD(two vers), RACE || early stopping 적용

4-2. Overall comparison between BERT and ALBERT evaluation

Model		Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg	Speedup
BERT	base	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3	17.7x
	large	334M	92.2/85.5	85.0/82.2	86.6	93.0	73.9	85.2	3.8x
	xlarge	1270M	86.4/78.1	75.5/72.6	81.6	90.7	54.3	76.6	1.0
ALBERT	base	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1	21.1x
	large	18M	90.6/83.9	82.3/79.4	83.5	91.7	68.5	82.4	6.5x
	xlarge	60M	92.5/86.1	86.1/83.1	86.4	92.4	74.8	85.5	2.4x
	xxlarge	235M	**94.1/88.3**	**88.1/85.1**	**88.0**	**95.2**	**82.3**	**88.7**	1.2x

Model		Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg	Speedup
BERT	large	334M	92.2/85.5	85.0/82.2	86.6	93.0	73.9	85.2	3.8x
ALBERT	base	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1	5.6x
	large	18M	90.6/83.9	82.3/79.4	83.5	91.7	68.5	82.4	1.7x
	xlarge	60M	92.5/86.1	86.1/83.1	86.4	92.4	74.8	85.5	0.6x
	xxlarge	235M	**94.1/88.3**	**88.1/85.1**	**88.0**	**95.2**	**82.3**	**88.7**	0.3x

Table 2: Dev set results for models pretrained over BOOKCORPUS and Wikipedia for 125k steps. Here and everywhere else, the Avg column is computed by averaging the scores of the downstream tasks to its left (the two numbers of F1 and EM for each SQuAD are first averaged).

ALBERT-xxlarge(235M)가 BERT-large(334M)보다 더 적은 파라미터로 좋은 성능을 보였다!

ALBERT-large는 BERT-large보다 1.7배 빨리 학습한다!

4-3. Factorized Embedding parameterization

Model	E	Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
ALBERT base not-shared	64	87M	89.9/82.9	80.1/77.8	82.9	91.5	66.7	81.3
	128	89M	89.9/82.8	80.3/77.3	83.7	91.5	67.9	81.7
	256	93M	90.2/83.2	80.3/77.4	84.1	91.9	67.3	81.8
	768	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3
ALBERT base all-shared	64	10M	88.7/81.4	77.5/74.8	80.8	89.4	63.5	79.0
	128	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1
	256	16M	88.8/81.5	79.1/76.3	81.5	90.3	63.4	79.6
	768	31M	88.6/81.5	79.2/76.6	82.0	90.6	63.3	79.8

Table 3: The effect of vocabulary embedding size on the performance of ALBERT-base.

All-share인 경우, vocabulary embedding size 128일 때가 성능이 좋다!

4-4. Cross-layer parameter sharing

	Model	Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
ALBERT base E=768	all-shared	31M	88.6/81.5	79.2/76.6	82.0	90.6	63.3	79.8
	shared-attention	83M	89.9/82.7	80.0/77.2	84.0	91.4	67.7	81.6
	shared-FFN	57M	89.2/82.1	78.2/75.4	81.5	90.8	62.6	79.5
	not-shared	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3
ALBERT base E=128	all-shared	12M	89.3/82.3	80.0/77.1	82.0	90.3	64.0	80.1
	shared-attention	64M	89.9/82.8	80.7/77.9	83.4	91.9	67.6	81.7
	shared-FFN	38M	88.9/81.6	78.6/75.6	82.3	91.7	64.4	80.2
	not-shared	89M	89.9/82.8	80.3/77.3	83.2	91.5	67.9	81.6

Table 4: The effect of cross-layer parameter-sharing strategies, ALBERT-base configuration.

실험에서는, 주로 all-shared를 사용했지만 **all-shared 다른 것에 비해 좋은 결과가 좋지는 않다.**

Shared-attention과 not-shared와 성능은 비슷하다!

4-5. SOP / Train for same amount of time

	Intrinsic Tasks			Downstream Tasks					
SP tasks	MLM	NSP	SOP	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
None	54.9	52.4	53.3	88.6/81.5	78.1/75.3	81.5	89.9	61.7	79.0
NSP	54.5	90.5	52.0	88.4/81.5	77.2/74.6	81.6	**91.1**	62.3	79.2
SOP	54.0	78.9	86.5	**89.3/82.3**	**80.0/77.1**	**82.0**	90.3	**64.0**	**80.1**

Table 5: The effect of sentence-prediction loss, NSP vs. SOP, on intrinsic and downstream tasks.

Models	Steps	Time	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
BERT-large	400k	34h	93.5/87.4	86.9/84.3	87.8	94.6	77.3	87.2
ALBERT-xxlarge	125k	32h	**94.0/88.1**	**88.3/85.3**	87.8	**95.4**	**82.5**	**88.7**

NSP보다 SOP로 학습한 모델이 성능 향상을 보였다.

BERT-large와 ALBERT-xxlarge를 같은 train 시간을 가졌을 때, ABLERT-xxlarge가 좋은 성능을 보였다.

5. Summary

1. ALBERT는 factorized Embedding Parameterization, cross-layer Parameter sharing, SOP loss를 사용

2. **ALBERT-xxlarge가 BERT-large보다 파라미터 수가 적고 더 좋은 결과를 보여줬다.**

3. ALBERT-xxlarge가 BERT-large보다 더 느린 학습 속도를 보여줬지만,
   같은 학습 시간을 가지면 ALBERT-xxlarge가 더 좋은 결과를 보여줬다.

4. RACE benchmark에서 89.4% 정확도로 RoBERTa(88.4%)보다 좋은 결과를 보여줬고
   SQuAD 2.0 benchmark에서 F1-score 92.2로 가장 좋은 결과를 보여줬다.