NLP bootcamp - Transformer Is All You Need (5th, 19.10.12 ~ 19.12.28)
Week11 (19.12.18)
발표자: 김동완

ALBERT: A Lite BERT for Self-supervised Learning of Language Representations

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, Radu Soricut
(26 Sep 2019)
https://arxiv.org/abs/1909.11942

Abstract

- 파라미터 수를 늘리면 NLP 모델의 성능이 좋아지는 경우가 많지만, GPU/TPU 메모리 제약으로 모델 크기를 늘리는데 한계가 있음. 성능을 떨어뜨리지 않으면서 메모리 사용량을 줄이는 방법 필요 본 논문에서는 아래 3가지 아이디어 제시
  - BERT류 모델의 파라미터 수를 줄이는 방법 두가지 방법
    - (1) Factorized Embedding Parameterization
    - (2) Parameter Sharing(FF & Attention)
  - Inter-sentence coherence 성능 개선을 위한 loss
    - (3) Sentence Order Prediction loss(text segment간의 순서를 예측)

1. Introduction

- full network pre-training & task-specific fine-tuning 이 대세
- 모델 크기를 키우는 것도 성능 향상에 큰 도움이 되다고 알려져 있음 → 하지만 모델 크기만 키우면 항상 좋은건가? 그리고 가능한가?
  - GPU/TPU 메모리 제약
  - 파라미터 수에 따른 느린 학습/추론 속도
  - 파라미터 수를 늘렸더니 성능이 떨어지는 경우도 발견됨 (BERT-xlarge 의 성능이 BERT-large 보다 좋지 않음, 아래 그림)

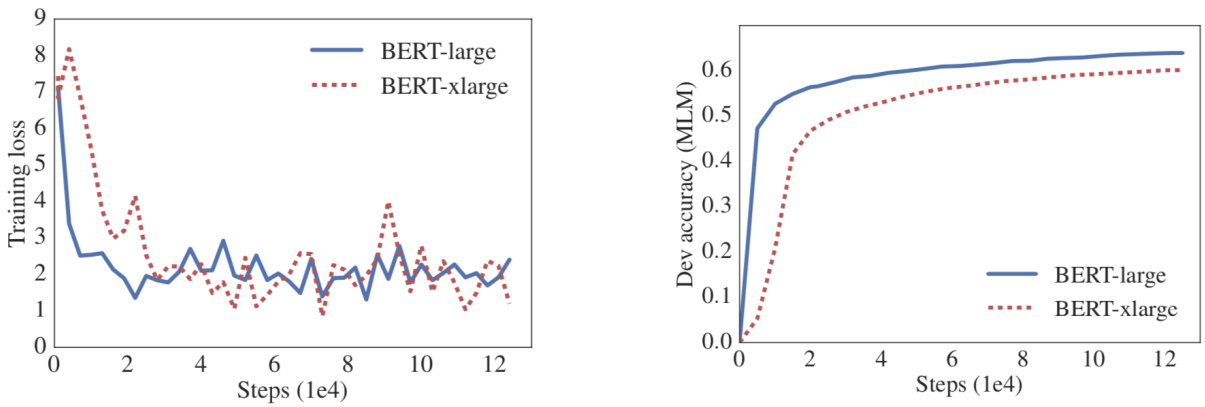


Figure 1: Training loss (left) and dev masked LM accuracy (right) of BERT-large and BERT-xlarge (2x larger than BERT-large in terms of hidden size). The larger model has lower masked LM accuracy while showing no obvious sign of over-fitting.

Model	Hidden Size	Parameters	RACE (Accuracy)
BERT-large (Devlin et al., 2019)	1024	334M	72.0%
BERT-large (ours)	1024	334M	73.9%
BERT-xlarge (ours)	2048	1270M	54.3%

Table 1: Increasing hidden size of BERT-large leads to worse performance on RACE.

- 이런 문제에 대한 대안으로 파라미터 수를 줄이는 두가지 방법과 text segment의 순서를 맞추는 형태의 self-supervised loss를 제시

2. Related work

2.1. Scaling up representation learning for natural language

- 요즘 큰 흐름
  - word embedding을 pre-training하는 대신 full-network를 pre-training 이후 task-specific fine-tuning하는 걸로 연구 방향이 바뀌고 있음.
- 모델 크기
  - Devlin et al., 2019(BERT) - hidden size가 클수록, hidden layer 수가 많을 수록, attention head 수가 많을 수록 성능이 좋아진다. → 하지만 본 논문(ALBERT)의 실험에 따르면 hidden size를 1024에서 2048로 늘렸을 때 성능이 감소
- 메모리 문제에 대한 다른 접근들
  - 최근 모델은 파라미터수가 수억개에서 수십억개에 이르는데, 이에 따른 메모리 부족 문제를 해결하기 위한 방법들이 소개되었지만(Chen et al., 2016 & Gomez et al., 2017) 학습 속도를 희생하면서 메모리 사용량을 줄인 접근법. → 본 논문의 접근법을 사용하면 메모리 사용량과 학습 시간 모두 줄어듬.

2.2. Cross-layer parameter sharing

- 레이어간 파라미터를 공유하는 다른 연구들 (기존 연구들은 encoder-decoder를 처음부터 학습시키는 방식 vs 본 논문은 pretraining/finetuning 방식)
  - Dehghani et al., 2018 - Universal Transformer - 기존 transformer보다 language modeling과 subject-verb agreement task에 더 나은 성능
  - Bai et al., 2019 - Deep Equilibrium Model(DQE) - (transformer) layer의 입력/출력 임베딩이 같아지는 상태(equilibrium point)가 됨.
  - Hao et al., 2019 - 파라미터를 공유하는 버전의 transformer와 그렇지 않은 버전을 결합한 형태(파라미터 수가 더 많아짐)

2.3. Sentence ordering objectives

- 담화(discourse)의 일관성(coherence)과 응집성(cohesion)은 인접한 text segment에 의해 크게 좌우된다. NLP 모델이 일관성과 응집성을 잘 담아내려면 어떤 형태의 'pre-training objective'를 사용해야할까? 기존 연구에서는 아래와 같은 것들이 사용됨
  - 이웃 문장에 속하는 단어를 예측(Kiros et al., 2015 - Skip-thought // Hill et al., 2016 - FastSent)
  - (꼭 바로 인접한 문장이 아니더라도) 미래에 나올 문장을 예측(Gan et al., 2017)
  - 두 문장의 순서 예측, 첫 세문장 이후 나올 문장 예측, 문장간 접속사 예측(Jernite et al., 2017 // Nie et al., 2019) → 본 논문의 접근법은 이와 가장 유사하며, 문장 간 순서 대신 'text segment'간의 순서를 예측 (A segment is usually comprised of more than one natural sentence, which has been shown to benefit performance by Liu et al.)

3. The elements of ALBERT

3.1. Model architecture choices

- 모델 구조
  - 기본 구조는 BERT와 동일 (feed-forward/filter size: $4H$, number of attention head = $H/64$) (E: vicabulary embedding size, L: number of encoder layers, H: hidden size)

Factorized embedding parameterization

- WordPiece embedding은 컨텍스트와 무관한 표현을 학습한다.(learn context-independent representation) 반면 hidden-layer embedding은 컨텍스트와 관련된 표현을 학습한다. (learn context-dependent representation)
- BERT류 모델의 강점은 컨텍스트를 잘 학습하는데서 나오기 때문에 H가 커야 할 것이다. 그런데 XLNet, RoBERTa를 포함한 BERT류의 모델에서 embedding size E는 hidden size H와 같다. ($E \equiv H$)
- 이런 세팅에서는 H를 크게 하면 불필요하게 embedding matrix 또한 커진다.
- ALBERT에서는 기존 $\mathbb{R}^{V \times H}$인 embedding matrix를 $\mathbb{R}^{V \times E}$와 $\mathbb{R}^{E \times H}$의 곱으로 분해해서 파라미터수와 계산량을 줄인다.
  - 예를들어 $V = 30000, H = 1024, E = 128$이라면 (30000 *1024)* → *(30000* 128 + 128 * 1024) 와 같이 기존 BERT 대비 embedding matrix 크기가 87% 감소

Cross-layer parameter sharing

- 공유할만한 파라미터는 아래와 같이 두가지가 있다. (Sec. 4.5.에 각 공유 케이스에 따른 결과 비교)
  - feed-forward network
    $$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$
  - attention
    $$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$
    $$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$
- 파라미터를 공유한다는 아이디어에 대한 다른 논문들
  - Dehghani et al. (2018) (Universal Transformer, UT)
    - vanilla Transformer 보다 나은 성능을 보인다 주장.
  - Bai et al. (2019) (Deep Equilibrium Models, DQE)
    - 학습시키다보면 각 transformer unit의 입력 임베딩과 출력 임베딩이 같아지는 상태에 도달(equilibrium point)
    - 반면 ALBERT에서는 어떤 파라미터에 수렴(converging)하기보다는 진동(oscillating)하는 결과를 얻음 (입출력 임베딩 벡터간 'L2 distance'와 'cosine similarity'를 비교) → Q) 여기서 oscillating한다는게 L2 distance나 cosine similarity가 변한다는 건가?, 0이 아니라는 것인가? → (이하 설명을 보면) 차이가 0이 아니라는 의미로 보임
- BERT와 ALBERT의 각 transformer unit의 입력 임베딩과 출력 임베딩 사이의 L2 distance와 cosine similarity 관찰
  - BERT에서는 transformer unit을 지날때마다 값(L2, cos sim) 변화가 불규칙적이었다. 반면 ALBERT에서의 값 변화가 보다 부드러웠다. 반면 → ALBERT에서의 네트워크 파라미터 변화가 보다 안정적.
  - ALBERT에서는 레이어를 지날수록 두 값(L2 & Cos sim)이 줄어들긴 했지만 24개 레이어를 지나도 그 값이 0이 되지는 않았다.(수렴하지는 않음)

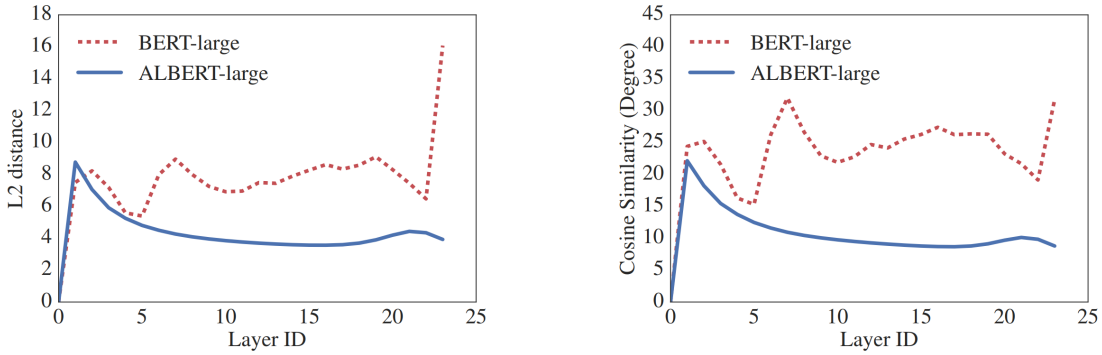


Figure 2: The L2 distances and cosine similarity (in terms of degree) of the input and output embedding of each layer for BERT-large and ALBERT-large.

Inter-sentence coherence loss

- BERT는 masked language model(MLM) loss와 next-sentence prediction (NSP) loss를 사용
  - NSP는 문장간 관계를 추론하는 문제에 대한 성능을 높히기 위해 도입됐지만 후속 연구(Yang et al., 2019와 Liu et al., 2019)에 따르면 NSP를 없애면 더 성능이 좋아지는게 발견됨.
- 왜 NSP가 좋지 않은가?
  - NSP는 topic prediction과 coherence prediction이 섞여 있다. (conflates topic prediction and coherence prediction) → 그런데 NSP에서의 negative sample은 앞/뒤 문장의 일관성(topic)과 응집성(coherence)이 모두 어긋나 있고, topic prediction이 coherence prediction보다 쉽기 때문에 네트워크는 topic prediction을 주로 학습하게 된다.
    - 예를들어 영화에 대한 문서에서 앞 문장을 뽑고, 정치에 대한 문서에서 뒷 문장을 뽑았다면 NSP에 대해 모델은 Coherence를 따지기도 전에 Topic가 다르다는 판단을 먼저 하게 된다. → 즉 NSP로는 coherence를 효과적으로 학습시키지 못한다.
    - 또한 topic prediction은 MLM 테스크와 겹치므로 NSP가 의도한 coherence 학습이 잘 되지 않는다. → topic prediction, MLM 모두 주어진 문장에 등장하는 token 과 빈번히 같이 사용되는 token 하나 혹은 조합을 찾는 문제)
- SOP
  - coherence를 보다 잘 학습시키기 위해, 문장(혹은 더 큰 단위)간 순서를 예측하도록 해 본다.(Sentence- Order Prediction, SOP)
    - Positive sample: 한 문서에서 뽑은 연속한 두개의 'text segment'
    - Negative sample: 한 문서에서 뽑은 연속한 두개의 'text segment'의 순서를 바꾼 것.

    → coherence prediction에서는 positive sample과 negative sample 모두

    `한 문서내 token들의 조합인데, 이들의 나열 순서를 예측하는 문제`

3.2 Model setup

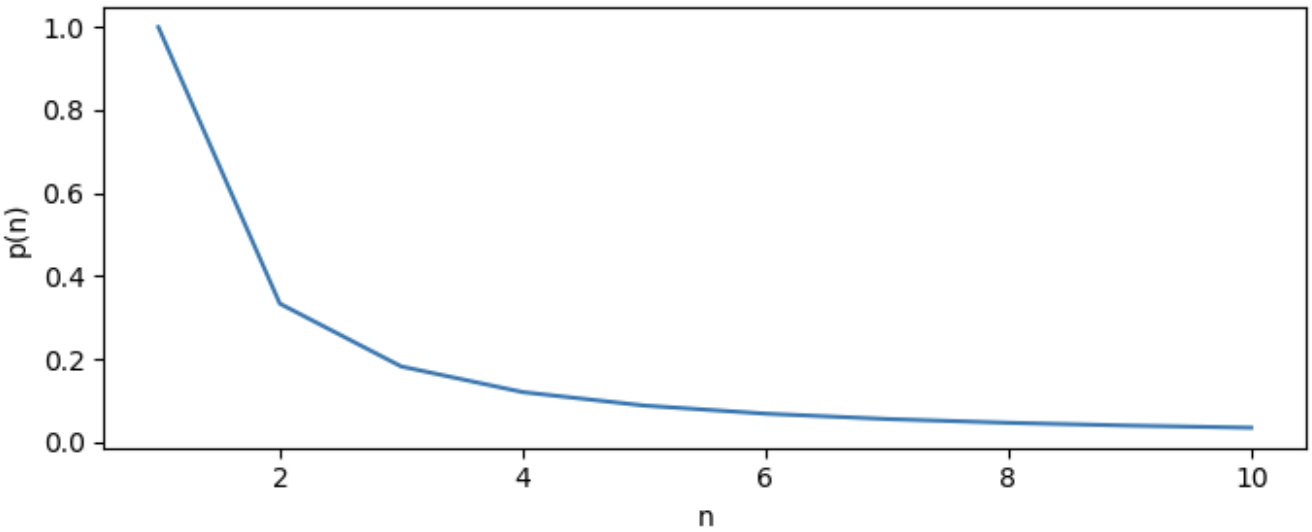
- 하이퍼파라미터(임베딩 크기, 히든 크기, 헤드 수, 트랜스포머 레이어 수 등) 세팅을 base, large, xlarge, xxlarge로 나누고 BERT와 ALBERT의 파라미터 수를 비교
- 예를들어 large 세팅의 파라미터수는 BERT는 334M개인 반면 ALBERT는 18M개 BERT는 Hidden = Embedding인 반면 ALBERT는 Hidden ≠ Embedding

4. Experimental results

4.1. Experimental setup

- Baseline model 학습에는 BookCorpus 와 English Wikipedia 데이터셋 사용 BERT와 ALBERT 모두 아래의 세팅 사용.
- [CLS] x_1 [SEP] x_2 [SEP] 형태의 데이터 포멧 사용
  - $x_1 = x_{1,1}, x_{1,2}, \cdots$ & $x_2 = x_{2,1}, x_{2,2}, \cdots$인데, $x_?$는 text segemnt로서 하나 이상의 문장으로 구성됨
- max input length: 512 10% 확률로 512보다 짧은 시퀀스 생성)
- vocabulary size: 30,000
- tokenizer: SentencePiece
- n-gram masking mask하나가 최대 3개 token을 가림. mask 길이가 n(=1, 2, 3)일 확률은 아래와 같다. Q) p(length ≥ n) 이어야 할듯? 즉 mask 길이가 n보다 크거나 같을 확률

$$p(n) = \frac{1/n}{\sum_{k=1}^{N} 1/k}$$



- batch size: 4096
- optimizer: LAMB
- learning rate: 0.00176
- training steps: 125,000
- machine: Cloud TPU V3 (# of TPU: 64 ~ 1024)

4.2. Evaluation benchmarks

4.2.1. Intrinsic evaluation

- 학습 진행을 모니터 하기 위해 SQuAD와 RACE dataset의 dev set data를 이용해 dev set을 만들고, MLM과 classification task에 대한 accuracy를 살펴봄 (물론 이 결과를 model selection 에는 사용하지 않음)

4.2.2. Downstream evaluation

- 다운스트림 테스크의 모델 성능 평가에는 GLUE benchmark와 SQuAD, RACE dataset을 사용함. (Appendix A.1.)

4.3. Overall comparison between BERT and ALBERT

- 비교 대상
  - 하이퍼파라미터(임베딩 크기, 히든 크기, 헤드 수, 트랜스포머 레이어 수 등) 세팅을 base, large, xlarge, xxlarge로 나누고 BERT와 ALBERT의 성능 비교
- 결과
  - ALBERT-xxlarge는 BERT-large 대비 70% 수준의 파라미터만으로 더 나은 성능을 보임.

    `(233M vs 334M)`

  - BERT-xlarge는 성능이 좋지 않음 → BERT는 파라미터 수가 일정 수준 이상 늘어나면 학습시키기 어려워짐.
  - ALBERT가 BERT 대비 학습 속도 또한 빠름. (ALBERT base vs BERT base, ALBERT large vs BERT large, ALBERT xlarge vs BERT xlarge)

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

Table 3: 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).

4.4. Factorized embedding Parameterization

- 비교 대상 (ALBERT-base를 기본으로)
  - Factorized Embedding Parameterization과 SOP는 사용하면서

    Parameter Sharing을 사용하지 않는 버전(ALBERT base not-shared)과 사용하는 버전(ALBERT base all-shared)에 대해 임베딩 크기(E)를 바꿔가면서 성능 비교
- 결과
  - Parameter Sharing을 사용하지 않는 경우 임베딩 크기(E)가 커질수록 성능이 좋아지긴 했으나, 성능 향상이 그렇게 크지는 않았다.
  - Parameter Sharing을 사용하는 세팅에서는 $E = 128$일 때 성능이 가장 좋았다.
  - 여타 실험에서는 embedding size 기본 값으로 128를 사용

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

4.5. Cross-layer parameter sharing

- 비교 대상 (ALBERT-base를 기본으로)
  - 두가지 임베딩 크기(768, 128)에 대해 아래 4가지 공유 케이스의 성능 비교
    - attention과 FFN을 공유(all-shared)
    - attention만 공유(shared-attention)
    - FFN만 공유(shared-FFN)
    - 공유하지 않음(not-shared)
- 결과
  - all-shared 의 성능이 가장 좋지 않음
  - 상대적으로 FFN을 공유하면 성능이 많이 나빠짐
  - attention을 공유하는 경우 성능 감소가 매우 작음 (특히 임베딩크기가 128인 경우 성능 감소 더 작음)
  - 레이어 L개를 크기가 M인 N개 그룹으로 나누고 각 그룹의 파라미터를 공유하는 방법도 있음 (M이 작아질수록 성능이 좋아지지만 파라미터 수가 늘어남)
  - 파라미터 수 관점에서 성능비교할 것이므로, 여타 실험에서는 attention과 FFN을 공유(all-shared)하는 것을 사용

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

4.6. Sentence order prediction(SOP)

- 비교 대상 (ALBERT-base를 기본으로)
  - None, Inter-sentence loss를 사용하지 않는 경우 (XLNet, RoBERTa style)
  - NSP만 쓰는 경우 (BERT style)
  - SOP만 쓰는 경우 (ALBERT style)
- Metric
  - MLM, NSP, SOP accuracy
- 결과
  - 3가지 세팅 모두 MLM loss가 포함되어 있으므로 None, NSP, SOP의 성능 차이가 크지 않음
  - NSP를 넣으면 NSP accuracy가 급격히 올라가지만 SOP accuracy는 거의 변화 없음 → 즉 NSP로는 문장간 순서를 학습시키지 못한다.
  - SOP를 넣으면 None에 비해 NSP accuracy와 SOP accuracy가 올라감 또한 downstream task 성능이 가장 좋음. → 즉 SOP로는 NSP도 어느정도 학습시킬 수 있다.

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

4.7. Effect of network depth and width

- 비교 대상
  - transformer layer 수에 따른 성능 비교
- 결과
  - 레이어 수가 늘어날 수록 성능이 좋아지지만, 레이어 수에 따른 성능 향상 효과는 점차 감소.
  - 레이어 수가 3개 이상인 모델은 그보다 레이어 수가 작은 모델을 fine-tuning함. (예를들어 레이어 수가 3개인 모델을 처음부터 학습시킨게 아니라 레이어 수가 1개인 모델의 checkpoint에서부터 학습 진행)
  - base, xxlarge: $L = 12$ large, xlarge: $L = 24$

Number of layers	Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
1	18M	31.1/22.9	50.1/50.1	66.4	80.8	40.1	52.9
3	18M	79.8/69.7	64.4/61.7	77.7	86.7	54.0	71.2
6	18M	86.4/78.4	73.8/71.1	81.2	88.9	60.9	77.2
12	18M	89.8/83.3	80.7/77.9	83.3	91.7	66.7	81.5
24	18M	**90.3/83.3**	**81.8/79.0**	83.3	91.5	**68.7**	**82.1**
48	18M	90.0/83.1	**81.8/78.9**	**83.4**	**91.9**	66.9	81.8

Table 7: The effect of increasing the number of layers for an ALBERT-large configuration.

- 비교 대상
  - hidden size(H)에 따른 성능 비교
- 결과
  - hidden size를 1024부터 4096까지 늘리면 성능이 올라가지만 6144까지 늘렸을 때에는 성능이 감소함 (best-performing ALBERT에 비해 실험한 4개 세팅의 모델은 학습 데이터에 overfit 되지는 않았음)
  - base: $H = 768$ large: $H = 1024$ xlarge: $H = 2048$ xxlarge: $H = 4096$

Hidden size	Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
1024	18M	79.8/69.7	64.4/61.7	77.7	86.7	54.0	71.2
2048	60M	83.3/74.1	69.1/66.6	79.7	88.6	58.2	74.6
4096	225M	**85.0/76.4**	**71.0/68.1**	**80.3**	**90.4**	**60.4**	**76.3**
6144	499M	84.7/75.8	67.8/65.4	78.1	89.1	56.0	74.0

Table 8: The effect of increasing the hidden-layer size for an ALBERT-large 3-layer configuration.

4.8. What if we train for the same amount of time?

- 비교 대상
  - BERT와 ALBERT를 비슷한 시간 동안 학습 시킨 후 성능 비교
    - BERT-large를 34시간 & 400k training step 학습 (파리미터수: 334M) ALBERT-xxlarge를 32시간 & 125k training step 학습 (파라미터수: 235M)
- 결과
  - ALBERT-xxlarge가 BERT-large보다 평균 1.5% 좋은 성능 보임.

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**

Table 9: The effect of controlling for training time, BERT-large vs ALBERT-xxlarge configurations.

4.9. Do very wide ALBERT models need to be deep(er) too?

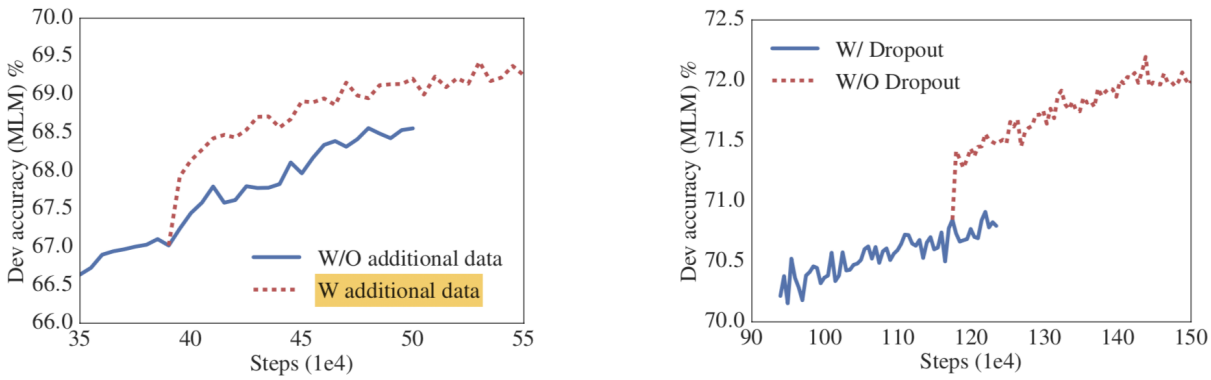
- 비교 대상
  - [4.7]에서 ALBERT-large($H = 1024$)의 레이어 수를 변화시켰을 때 성능 변화를 봤고, 레이어가 12개일 때와 24개일때 성능차이가 크지 않았다. 이번에는 ALBERT-xxlarge($H = 4096$)에 같은 실험을 해 본다.
- 결과
  - 마찬가지로 레이어가 12개일 때와 24개일 때 성능 차이가 크지 않음. (결국 레이어 수를 12개 이상 늘릴 필요는 없어 보임)

Number of layers	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
12	94.0/88.1	88.3/85.3	87.8	95.4	82.5	88.7
24	94.1/88.3	88.1/85.1	88.0	95.2	82.3	88.7

Table 10: The effect of a deeper network using an ALBERT-xxlarge configuration.

4.10. Additional training data and dropout effects

- 비교 대상(a)
  - 앞서 테스트들에서는 Wikipedia와 BookCorpus 데이터셋을 사용했었다. 여기에 XLNet과 RoBERTa에서 공통적으로 사용된 **데이터를 추가했을 때** 성능 변화를 살펴본다.
- 결과(a)
  - 추가 데이터를 사용한 경우 'dev accuracy' 상승
- 비교 대상(b)
  - drop-out를 사용하는 경우, 사용하지 않는 경우
- 결과(b)
  - drop-out을 제거 했을 때 'dev accuracy' 상승
  - CNN 모델에 Batch norm과 dropout을 썼을 때 성능에 나쁜 영향을 준다는 기존 연구 있음
    - Szegedy et al., 2017: Inception-v4, inception-resnet and the impact of residual connections on learning
    - Li et al., 2019: Understanding the disharmony between dropout and batch normalization by variance shift)



(a) Adding data (b) Removing dropout

Figure 3: The effects of adding data and removing dropout during training.

	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
No additional data	**89.3/82.3**	**80.0/77.1**	81.6	90.3	64.0	80.1
With additional data	88.8/81.7	79.1/76.3	**82.4**	**92.8**	**66.0**	**80.8**

Table 11: The effect of additional training data using the ALBERT-base configuration.

	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
With dropout	94.7/89.2	89.6/86.9	90.0	96.3	85.7	90.4
Without dropout	**94.8/89.5**	**89.9/87.2**	**90.4**	**96.5**	**86.1**	**90.7**

Table 12: The effect of removing dropout, measured for an ALBERT-xxlarge configuration.

4.11. Current state-of-the-art on NLU tasks

- 앞서 실험들을 종합해 최종적으로 선택한 ALBERT 세팅은 아래와 같다.
  - 학습 데이터: Devlin et al., 2019 (BERT) + Liu et al., 2019 (RoBERTa) + Yang et al., 2019 (XLNet)
  - 임베딩 크기(E), 히든 크기(H), 헤드 수($H/64$), 트랜스포머 레이어 수(L)는 ALBERT-xxlarge 세팅을 사용 ($L = 12, H = 4096, E = 128$, Parameter-sharing: All shared, Parameters: 235M)
  - MLM & SOP loss
  - no dropout
- GLUE benchmark

Models	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
Single-task single models on dev										
BERT-large	86.6	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
XLNet-large	89.8	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa-large	90.2	94.7	**92.2**	86.6	96.4	**90.9**	68.0	92.4	-	-
ALBERT (1M)	90.4	95.2	92.0	88.1	96.8	90.2	68.7	92.7	-	-
ALBERT (1.5M)	**90.8**	**95.3**	**92.2**	**89.2**	**96.9**	**90.9**	**71.4**	**93.0**	-	-
Ensembles on test (from leaderboard as of Sept. 16, 2019)										
ALICE	88.2	95.7	**90.7**	83.5	95.2	92.6	**69.2**	91.1	80.8	87.0
MT-DNN	87.9	96.0	89.9	86.3	96.5	92.7	68.4	91.1	89.0	87.6
XLNet	90.2	98.6	90.3	86.3	96.8	93.0	67.8	91.6	90.4	88.4
RoBERTa	90.8	98.9	90.2	88.2	96.7	92.3	67.8	92.2	89.0	88.5
Adv-RoBERTa	91.1	98.8	90.3	88.7	96.8	93.1	68.0	92.4	89.0	88.8
ALBERT	**91.3**	**99.2**	90.5	**89.2**	**97.1**	**93.4**	69.1	**92.5**	**91.8**	**89.4**

Table 13: State-of-the-art results on the GLUE benchmark. For single-task single-model results, we report ALBERT at 1M steps (comparable to RoBERTa) and at 1.5M steps. The ALBERT ensemble uses models trained with 1M, 1.5M, and other numbers of steps.

- SQuAD & RACE benchmark

Models	SQuAD1.1 dev	SQuAD2.0 dev	SQuAD2.0 test	RACE test (Middle/High)
Single model (from leaderboard as of Sept. 23, 2019)				
BERT-large	90.9/84.1	81.8/79.0	89.1/86.3	72.0 (76.6/70.1)
XLNet	94.5/89.0	88.8/86.1	89.1/86.3	81.8 (85.5/80.2)
RoBERTa	94.6/88.9	89.4/86.5	89.8/86.8	83.2 (86.5/81.3)
UPM	-	-	89.9/87.2	-
XLNet + SG-Net Verifier++	-	-	90.1/87.2	-
ALBERT (1M)	94.8/89.2	89.9/87.2	-	86.0 (88.2/85.1)
ALBERT (1.5M)	**94.8/89.3**	**90.2/87.4**	**90.9/88.1**	**86.5 (89.0/85.5)**
Ensembles (from leaderboard as of Sept. 23, 2019)				
BERT-large	92.2/86.2	-	-	-
XLNet + SG-Net Verifier	-	-	90.7/88.2	-
UPM	-	-	90.7/88.2	
XLNet + DAAF + Verifier	-	-	90.9/88.6	-
DCMN+	-	-	-	84.1 (88.5/82.3)
ALBERT	**95.5/90.1**	**91.4/88.9**	**92.2/89.7**	**89.4 (91.2/88.6)**

Table 14: State-of-the-art results on the SQuAD and RACE benchmarks.

5. Discussion

- ALBERT가 BERT 대비 적은 파라미터로 더 나은 성능을 보임.(ALBERT-xxlarge vs BERT-large)
  - 다만 동일한 BERT 세팅에서 파라미터 공유만 하면 성능이 떨어질 수 있음.
- ALBERT의 training/inference 속도를 높이는게 과제 (sparse attention, block attention 등 해보면 좋을 듯)
- SOP 보다 더 나은 방법도 있지 않을까?

.

Major References

Lai et al., 2017 - RACE: Large-scale ReAding Comprehension dataset from Examinations

- Dataset: **RACE** (task designed for middle and high-school English exams in China)
- For this task, SOTA machine accuracy at that time was 44.1%

Devlin et al., 2019 - **BERT**: Pre-training of deep bidirectional transformers for language understanding

- about full network pre-training
- importance of large network

Liu et al., 2019 - **RoBERTa**

- performance for RACE test: 83.2%

Yang et al., 2019 - **XLNet**

Radford et al., 2019 - Language models are unsupervised multitask learners

- about full network pre-training
- importance of large network

Sun et al., 2019 - Patient knowledge distillation for BERT model compression

- pre-training large models and distilling down.

Turc et al., 2019 - Well-read students learn better: The impace of student initialization on knowledge distillation.

- pre-training large models and distilling down.

Szegedy et al., 2017 - Inception-v4, inception-resnet and the impact of residual connections on learning

Li et al., 2019 - Understanding the disharmony between dropout and batch normalization by variance shift)