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

NeurIPS 2019 (# Citation: 8479회)

가짜연구소 김 성 은

2025.03.20

- Introduction
- Related work
- The Elements of ALBERT
- Experimental Result

Introduction

- Full network pre-trainning의 발전으로 NLP 성능이 크게 개선됨
- BERT 기반 pre-train model들은 HW의 제약으로 인해 크게 2가지 문제 발생
  - **memory limitation**
  - **communication overhead**

Introduction

- Full network pre-trainning의 발전으로 NLP 성능이 크게 개선됨
- BERT 기반 pre-train model들은 HW의 제약으로 인해 크게 2가지 문제 발생
  - **memory limitation**
  - **communication overhead**
    - communication: 통신
    - overhead: 어떤 처리를 하기 위해 들어가는 간접적인 처리 시간

Introduction

- Full network pre-trainning의 발전으로 NLP 성능이 크게 개선됨
- BERT 기반 pre-train model들은 HW의 제약으로 인해 크게 2가지 문제 발생
  - **memory limitation** → **parallelization 및 memory 관리 적용**
  - **communication overhead** 🤔

Introduction

- Full network pre-trainning의 발전으로 NLP 성능이 크게 개선됨
- BERT 기반 pre-train model들은 HW의 제약으로 인해 크게 2가지 문제 발생
    - **memory limitation**
    - **communication overhead**
- 본 논문에서는 위의 두 문제를 모두 해결하기 위한 2가지 **parameter reduction 기법**을 제안
    - **factorized embedding parameterization**
    - **cross-layer parameter sharing**
- 더 나아가, BERT의 NSP 대신 **SOP(Sentence Order Prediction)**를 도입해 성능 ↑

Related work

2.1. Scaling up Representation Learning for Natural Language

- BERT, GPT 등 대형 모델이 성능 향상에 기여했으며, 모델 크기가 클수록 성능이 향상된다고 알려짐.
  (larger hidden size, more hidden layers, and more attention head → better performance)

🤔 👍 👍

[BERT]

hidden layers나 attention head 늘리면 성능 향상되지만,
hidden size는 1024를 넘어서면 성능 크게 향상 X, 계산 비용 ↑

- 결국, AlBERT에서는 모델 크기는 오히려 줄이고, 효율적인 구조로 성능 개선

Related work

2.2. Cross-Layer Parameter Sharing

- Universal Transformer, DQE 등에서 이미 연구된 개념
- ALBERT의 경우 교차 레이어 매개변수 공유가 모델 크기를 크게 줄이면서 성능을 유지하는 방식으로 활용

2.3. Sentence Ordering Objectives

- BERT → NSP

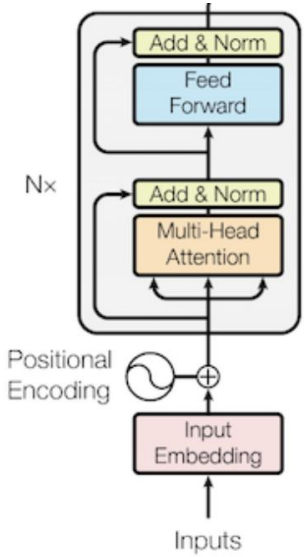
  : 두 문장을 입력하고 두 번째 문장이 첫 번째 문장의 다음 문장인지 예측하도록 학습하는 이진 분류 task
- ALBERT → SOP

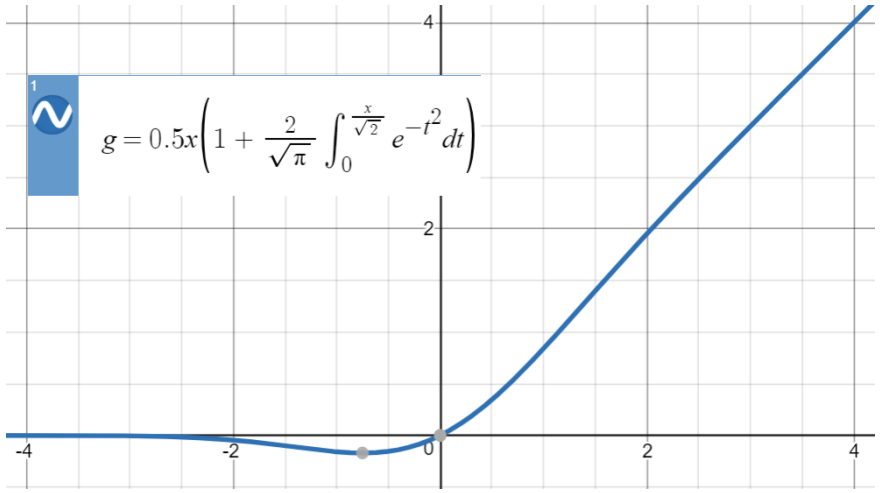
  : 한 쌍의 문장이 문장 순서가 바뀌었는지 여부를 예측하도록 학습하는 task

The Elements of ALBERT

Model Architecture Choices

- transformer encoder와 GELU activation function을 사용





The Elements of ALBERT

Factorized embedding parameterization

- embedding parameter를 factorization하여 2개의 작은 행렬로 인수분해(분리)
- Token Embedding Size(E) 를 Hidden layer size(H) 보다 작게 하여 Parameter 수를 줄임
- 기존의 BERT 기반 모델은
  - Vocabulary size (V) ≈ 30,000 → 사용할 수 있는 고유한 WordPiece 개수
  - WordPiece embedding size (E) = 768 → 각 WordPiece를 벡터로 변환할 때 벡터의 차원 수
    = Vocabulary embedding size
  - Hidden layer size (H) = 768 → 모델이 문맥을 학습할 때 사용하는 벡터의 차원 수

$$E = H$$

The Elements of ALBERT

Factorized embedding parameterization

E를 H만큼 크게 하지 않아도 돼! 유행은 H를 키우는 것!

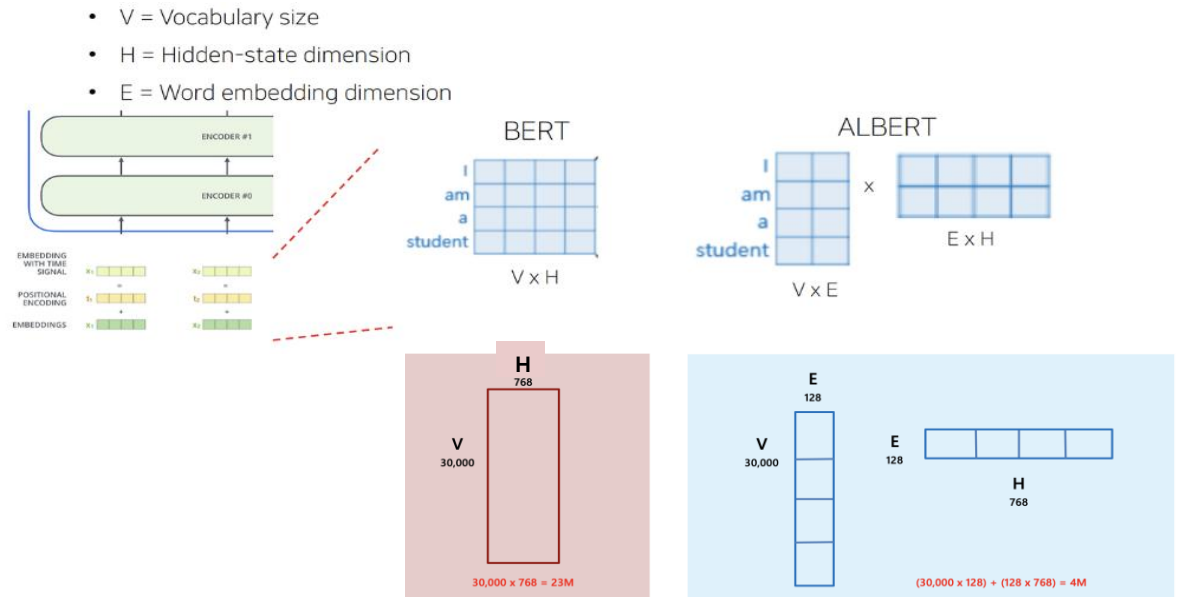
- Embedding size (E) → context–independent 한 representation
- Hidden layer size (H) → context–dependent 한 representation

즉, 두 embedding의 목적이 다르기에 size가 같을 필요 X

The Elements of ALBERT

Factorized embedding parameterization

- BERT에서는 E=H 이므로 vocabulary size(V)에 대해서 Embedding parameters는 O(V×H)임.
- ALBERT의 경우 decompostion을 통해 두개의 matrix로 factorization E<<H 이므로 Embedding parameters는 O(V×E + E×H)가 훨씬 줄어듦



The Elem

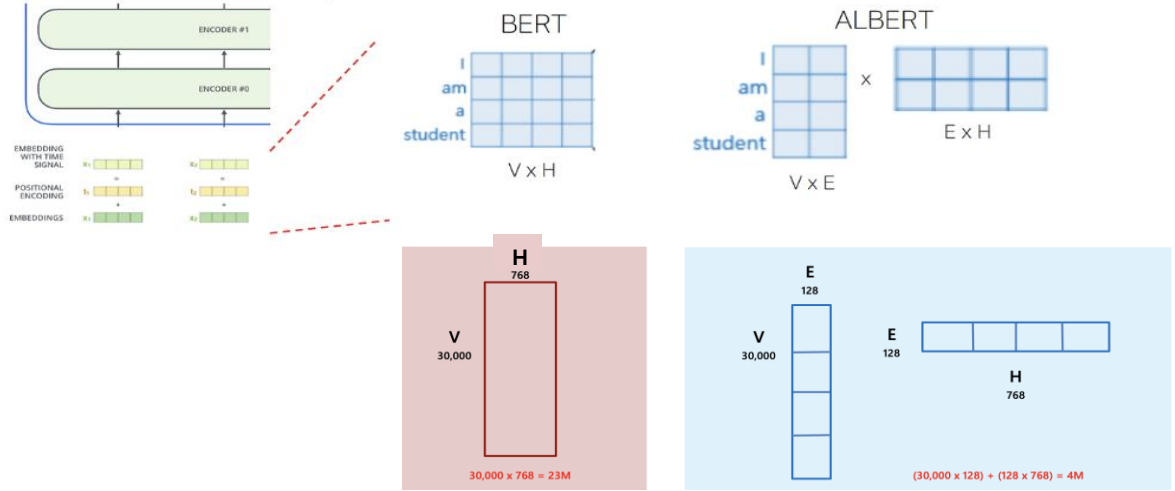
Factorized embeddin

- BERT에서는 E=H 이므 meters는 O(V×H)임.
- ALBERT의 경우 deco <H 이므로

  Embedding parame

```python
class Embeddings(nn.Module):
    "The embedding module from word, position and token_type embeddings."
    def __init__(self, cfg):
        super().__init__()
        # Original BERT Embedding
        # self.tok_embed = nn.Embedding(cfg.vocab_size, cfg.hidden) # token embedding

        # factorized embedding
        self.tok_embed1 = nn.Embedding(cfg.vocab_size, cfg.embedding) # embedding = 128
        self.tok_embed2 = nn.Linear(cfg.embedding, cfg.hidden) # hidden = 784

        self.pos_embed = nn.Embedding(cfg.max_len, cfg.hidden) # position embedding
        self.seg_embed = nn.Embedding(cfg.n_segments, cfg.hidden) # segment(token type) embedding

        self.norm = LayerNorm(cfg)
        # self.drop = nn.Dropout(cfg.p_drop_hidden)

    def forward(self, x, seg):
        seq_len = x.size(1)
        pos = torch.arange(seq_len, dtype=torch.long, device=x.device)
        pos = pos.unsqueeze(0).expand_as(x) # (S,) -> (B, S)

        # factorized embedding
        e = self.tok_embed1(x)
        e = self.tok_embed2(e)
        e = e + self.pos_embed(pos) + self.seg_embed(seg)
        #return self.drop(self.norm(e))
        return self.norm(e)
```

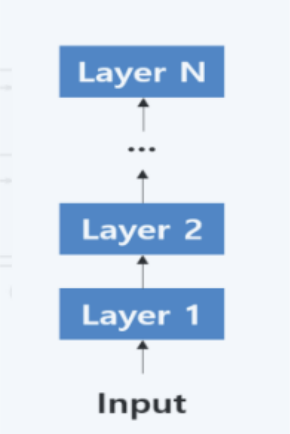
- V = Vocabulary size
- H = Hidden-state dimension
- E = Word embedding dimension

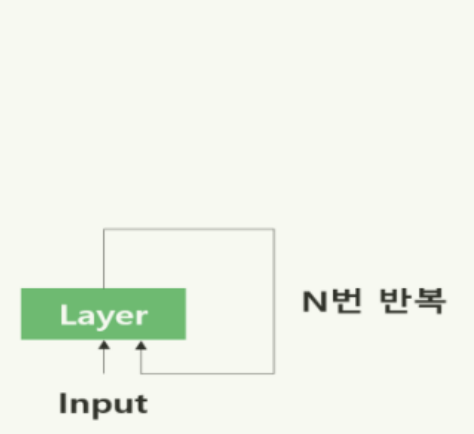


The Elements of ALBERT

Cross-Layer Parameter Sharing

- Transformer Layer 간 같은 Parameter를 공유하며 사용하는 것을 의미
- 기존의 BERT는 Transformer Block을 1~12번 거쳤음
- ALBERT에서는 이와 같은 효과를 누리기 위해 단 하나의 Transformer Block을 12번 거침





The Elements of ALBERT

Cross-Layer Parameter Sharing

안정적

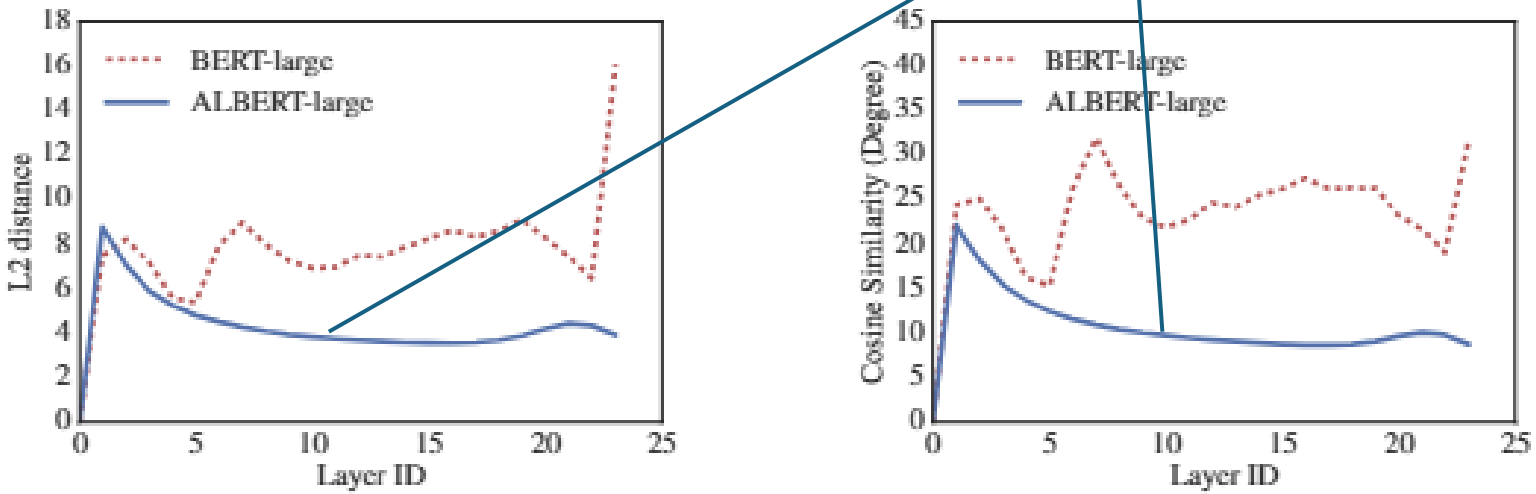
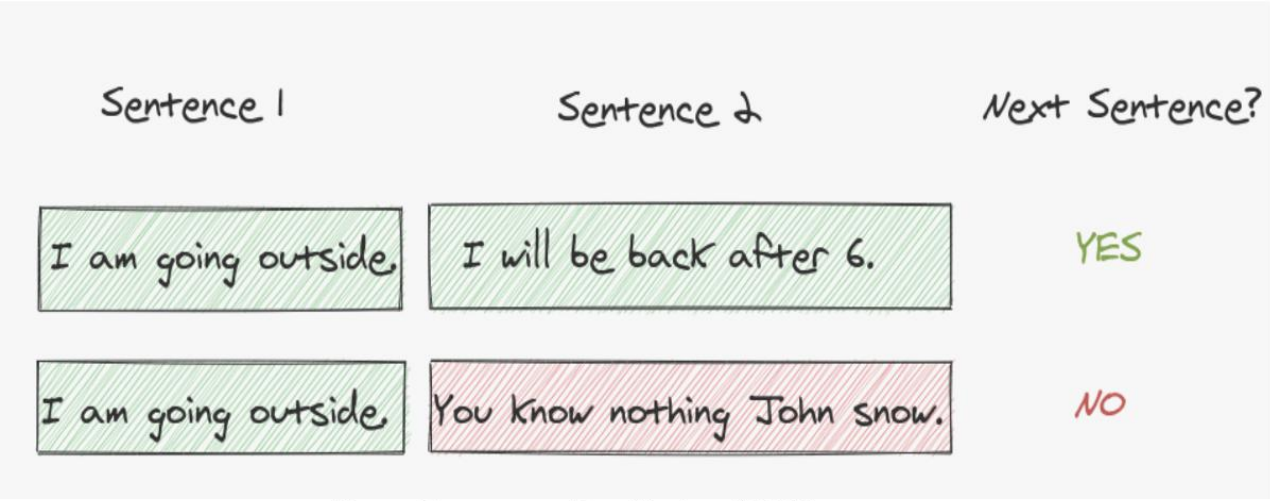


Figure 1: The L2 distances and cosine similarity (in terms of degree) of the input and output embedding of each layer for BERT-large and ALBERT-large.

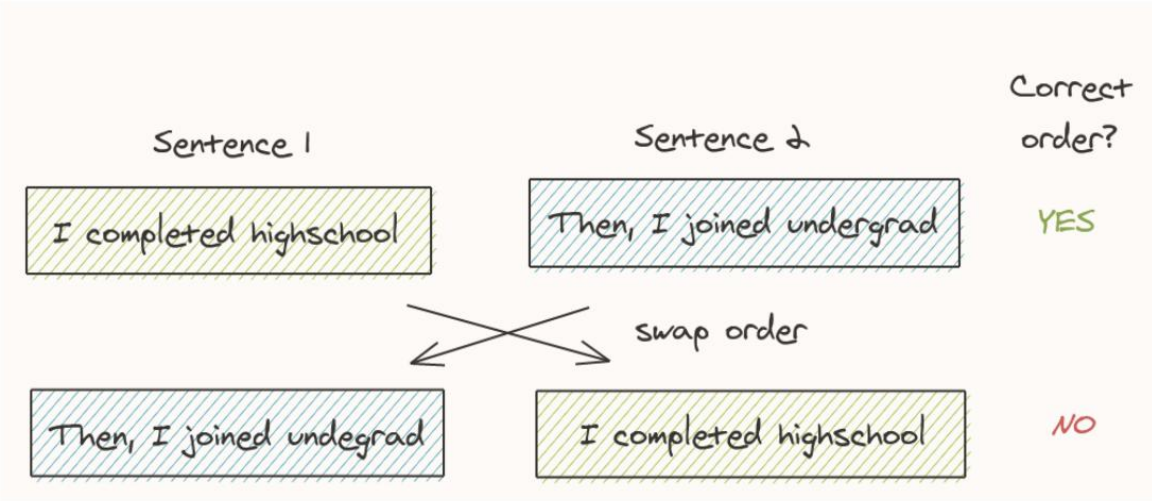
The Elements of ALBERT

Sentence Order Prediction

- 기존의 BERT는 문장과 문장이 서로 연속된 문장인지 맞추는 NSP 사용
- ALBERT에서는 2개의 문장의 순서를 맞추는 Sentence Order Prediction(SOP) 사용



Next Sentence Prediction(NSP)



Sentence Order Prediction(SOP)

The Elements of ALBERT

Model Setup

- ALBERT_large는 BERT보다 18배 적은 parameter 수를 가짐

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

Table 1: The configurations of the main BERT and ALBERT models analyzed in this paper.

Experimental Results

Experimental Setup

- Training data
  - BookCorpus
  - English Wikipedia

→ BERT와 같음

- Input $[\mathrm{CLS}]\ x_1\ [\mathrm{SEP}]\ x_2\ [\mathrm{SEP}]$
- 최대 입력 길이 512로 제한, vocabulary size (V)는 30,000로 함
- SentencePiece 사용

Experimental Results

BERT와 ALBERT의 성능 비교

F1 EM

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	4.7x
	large	334M	92.2/85.5	85.0/82.2	86.6	93.0	73.9	85.2	1.0
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_xlarge, ALBERT_xxlarge 는 BERT_large 보다 parameter 적게 사용했음에도 좋은 성과를 냄
- 그러나 추론 속도는 각각 6배, 3배 느리다는 것을 알 수 있음

Experimental Results

Vocabulary embedding size에 대한 성능 비교

- Not shared (BERT-style)에서는 embedding size가 클수록 모델 성능이 증가하지만 상승폭이 크지 않음
- all-shared (ALBERT-style)에서는 embedding size와 성능이 비례하지 않고, 128이 가장 좋은 결과를 보임

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.

Experimental Results

- 모든 all-shared 한 경우, E에 상관없이 성능 하락
- Attention parameter를 공유하는 것만으로는 성능이 크게 하락 X

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.

공유하는 파라미터가 증가할수록 전체 파라미터 수는 감소하지만 성능이 저하됨

Experimental Results

- NSP 과제만으로는 SOP 과제 성능을 높이지 못하지만, SOP 과제는 NSP 과제 성능을 높이는 것을 확인
- 결론적으로 SOP는 모든 downstream task에서 성능 ↑

	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.

Experimental Results

What if we train for the same amount of time?

- ALBERT-xxlarge 가 BERT-large 에 대비했을 때 3.17 배 낮은 데이터 처리량을 보여 더 많은 학습 시간이 필요

  → 동일한 학습 시간만큼 학습할 때 BERT-large 와 ALBERT-xxlarge 모델의 성능을 비교

- BERT-large로 400K Step학습한 것보다, ALBERT-xxlarge로 125K step 을 학습한 것이 좋은 성능을 보임

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	4.7x
	large	334M	92.2/85.5	85.0/82.2	86.6	93.0	73.9	85.2	1.0
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).

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

Experimental Results

Additional training data and dropout effects

- pre-train 단계에서 추가적인 dataset을 학습시키는 것, dropout을 제거하는 것으로 실험을 진행
- XLNet과 RoBERTa에서 사용했던 추가 dataset을 통해 추가적인 pre-train을 진행

  →추가적인 dataset을 사용하는 것이 downstream task에서도 성능을 향상시킨다는 것을 확인
- 다만, SQuAD에서는 성능이 하락 → 이는 SQuAD는 Wikipedia-based task이기 때문
- Dropout 제거시에 오히려 downstream 과제의 성능이 향상됨

	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**

	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**

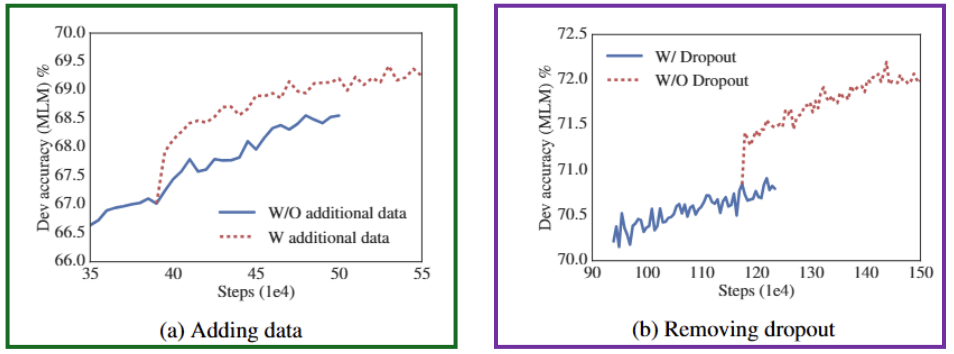


Figure 2: The effects of adding data and removing dropout during training.

Experimental Results

Current State-Of-The-Art on NLU Tasks

- ALBERT 모델이 기존 NLP 과제에서 모두 SOTA 와 동일하거나 더 높은 성능을 달성

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 9: 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.

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

Discussion

- factorized embedding parameterization, cross-layer parameter sharing로 모델을 경량화한 것이 포인트
- GLUE, SQuAD, RACE Task에서 BERT보다 좋은 성능을 보임