Accelerating Training of Transformer-Based Language Models with Progressive Layer Dropping

Minjia Zhang and Yuxiong He

집현전 중급반
구혜연

INDEX

01 Introduction

02 Preliminary Analysis

03 Our Approach : Progressive Layer Dropping

04 Evaluation

05 Conclusion

01. Introduction

1. 기존 연구와 한계

NLP의 패러다임

- Pre-training :
  Transformer 기반 모델 + 큰 학습 데이터

$$h_i = f_{LN}(x_i + f_{S-ATTN}(x_i))$$
$$x_{i+1} = f_{LN}(h_i + f_{FFN}(h_i))$$
$$\times L$$

- unsupervised pre-training 모델을 학습하기 위해서
  많은 computational 비용 발생
- 이러한 문제를 해결하기 위한
  고성능 하드웨어와 병렬화가
  없다면, language model을
  활용하기가 힘듦

학습 속도 개선 노력

ELECTRA

- 토큰 Masking과 예측에서
  Adversarial Training을 차용
- 전체 학습 step 수를 감소시켜
  속도 향상 효과

FLOPS

- Floating-point Operation
  연산 수를 줄이기 위해
  트랜스포머의 Depth를 감소
- 속도 증가, **정확도 하락**

[31]

- 처음은 3-layer로 학습하고
  점진적으로 6, 12-layer로 늘림
- 각 Depth의 step 수를 정해야
  하므로 **현실 적용 어려움**

단점 극복 방안

Stochastic Depth

- 필요한 만큼 Depth를 줄여
  효과가 있음을 보임
- 그러나, Random으로
  Layer를 Drop하면
  Local Minimum에 수렴
- 그래서, LR를 증가시키면
  Warm-up step을
  설정하더라도 **발산**

무엇이
Layer Dropping을 도입한
BERT의 Pre-training을
어렵게 하는가?

2. 한계를 극복하기 위한 생각의 흐름

Try	별도의 하드웨어 자원을 추가하지 않고 아키텍처 변경과 학습 테크닉을 통해 학습 속도를 향상시켜보자!
Idea1	Transformer Layer 수에 따라 학습 비용이 늘어나니까 **Transformer Network의 Depth를 줄여볼까**? **Nope!** 모델의 용량이 작아지므로, Downstream Task의 정확도가 낮아짐
Idea2	**Stochastic Depth** 테크닉이 이미지 인식 분야에서 지도학습 가속에 효과가 있던데 적용해볼까? **Nope!** 성능이 불안정. Local Minimum에 빠지거나 발산하는 경우가 있음
Challenge	왜 Transformer Network는 Stochastic Depth로 학습이 어려울까? Downstream Task의 성능을 저해하지 않고 Transformer의 Pre-training의 속도를 증가시킬 수 없을까?

3. Challenge Solve를 위한 단계

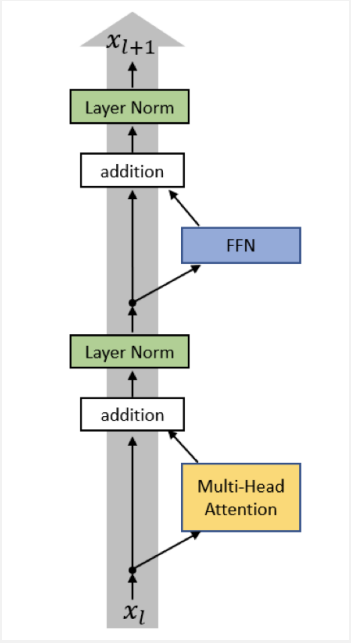
Analysis	"무엇이 Transformer Network에 Stochastic Depth 적용을 어렵게 하는가 " 를 찾기 위한 분석
New Method	각 Transformer Layer를 On/Off할 수 있는 **Switchable-Transformer(ST) Block** unit 제시 Layer Dropping과 함께 Transformer Network를 안정화할 수 있는 **Progressive Schedule** 제시
Evaluation	동일한 샘플을 이용해 Pre-training했을 때, Original BERT보다 **24% 빠름** Downstream Task에서 비슷한 Accuracy에 도달하는 데에 Original BERT보다 **2.5배 빠름** Original BERT보다 더 빠르게 학습함에도 불구하고, **1.1% 높은 GLUE 달성**

02. Preliminary Analysis

무엇이 Transformer Network에 Stochastic Depth 적용을 어렵게 하는가?

1. Training Stability : PostLN or PreLN

PostLN

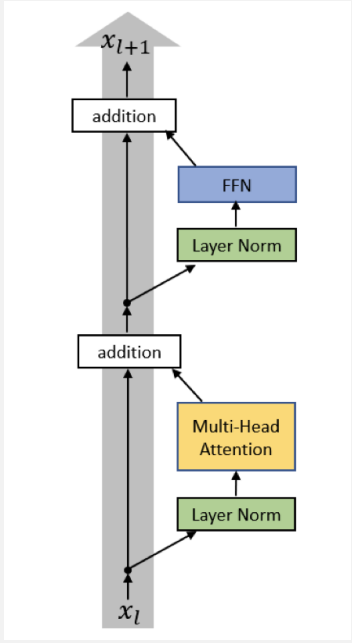


$$h_i = f_{LN}(x_i + f_{S-ATTN}(x_i))$$

$$x_{i+1} = f_{LN}(h_i + f_{FFN}(h_i))$$

VS

PreLN



$$h_i = x_i + f_{S-ATTN}(f_{LN}(x_i))$$

$$x_{i+1} = h_i + f_{FFN}(f_{LN}(h_i))$$

분석 결과

※ 이미지 출처 : On Layer Normalization in the Transformer Architecture

1 Unbalanced Gradient

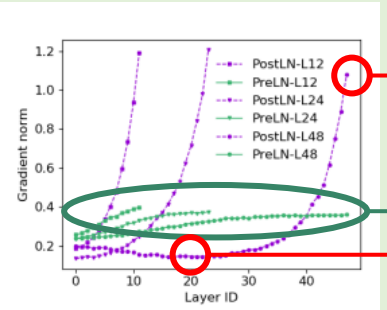


Figure 1: The norm of the gradient with respect to the weights, with PostLN and PreLN.

PostLN은 Layer가 깊어지면서 Gradient Norm이 증가

PreLN은 **Gradient 불균형 문제를 해결**함 Gradient Norm이 모든 Layer에서 거의 동일

2 Norm Preserving Ratio

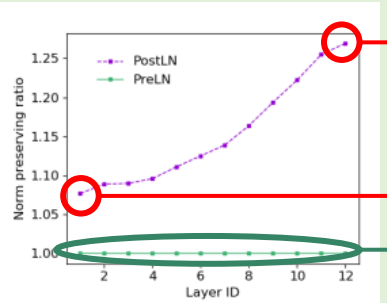


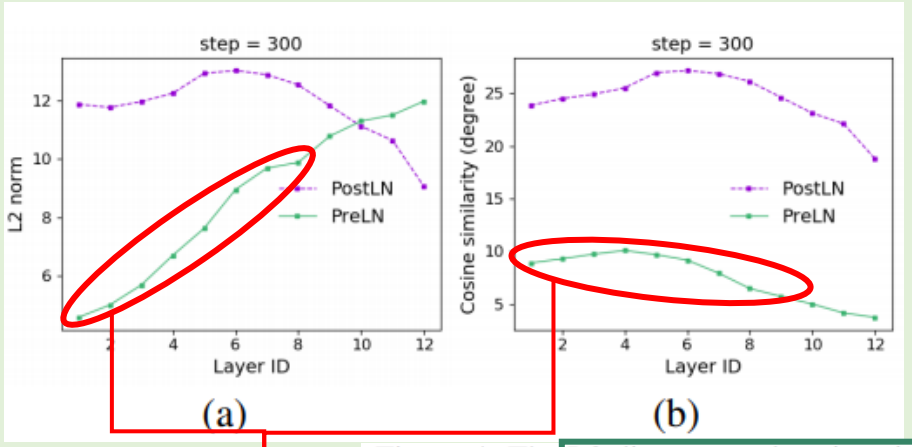
Figure 2: The norm preserving ratio with respect to the inputs, with PostLN and PreLN.

PostLN은 일정 수준의 Gradient Norm을 유지하지 못함

PreLN은 Gradient 규모를 일정하게 유지 안정적으로 학습을 수행

2. Corroboration of Training Dynamics

1 학습 초기에 각 층의 input과 output의 유사도



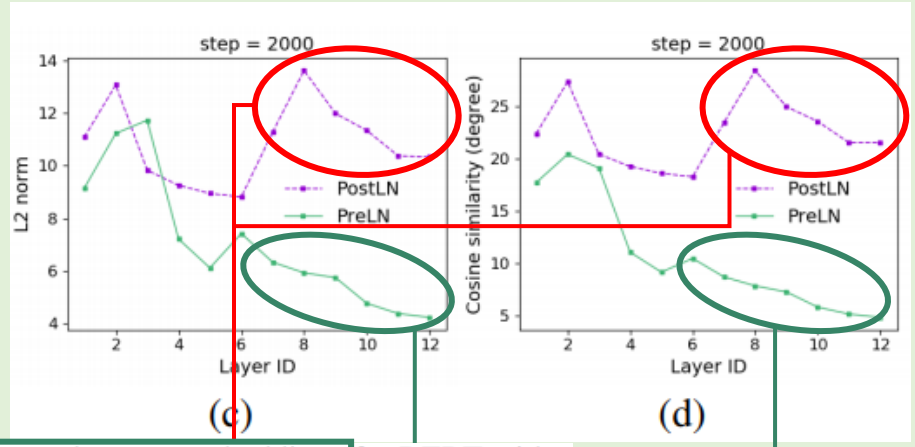
낮을수록 유사도 높음

Figure 4: The L2 distance and cosine similarity of the input and output embeddings for BERT with PostLN and PreLN, at different layers and different steps. We plot the inverse of cosine similarity (arccosine) in degrees, so that for both L2 and arccosine, the lower the more similar.

PostLN과 PreLN 모두
L2 Norm과 Cosine Similarity의
관련성이 떨어짐

- 학습 초기 단계이므로 input에 대해 풍부한 feature를 찾기 위해 노력하는 단계
- **이 단계에서 Layer Dropping은 좋지 않음**

2 학습 중후반기에 각 층의 input과 output의 유사도



PostLN에서는 울퉁불퉁하고
여전히 dissimilarity가 높음

PreLN에서는
연속적인 Layer에서
similarity가 증가하기 시작함

- PostLN은 여전히 새로운 feature를 찾고 있음에 반해 PreLN은 상위 Layer에서는 similar estimation을 찾음
- PreLN은 **unrolled iterative refinement** 현상을 보임

3. Effect of Lesioning

1 Layer Dropping은 학습을 저해하는가?

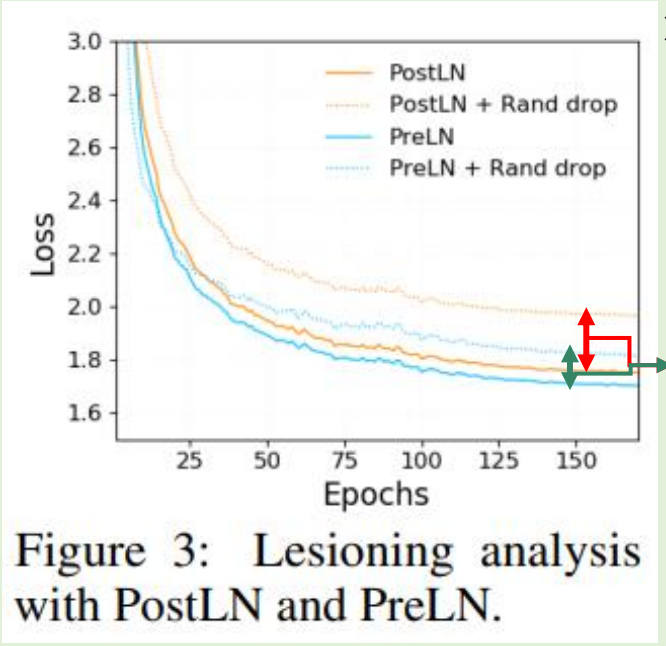


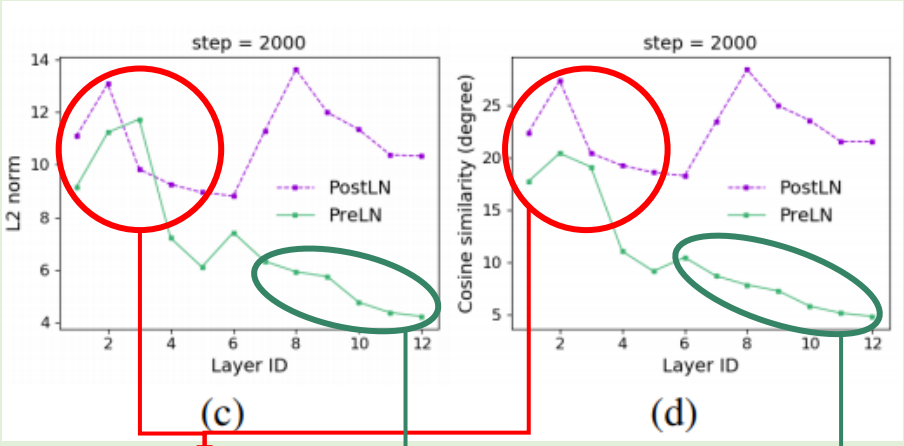
Figure 3: Lesioning analysis with PostLN and PreLN.

※ layer keep ratio : 0.5

Layer Drop 시,
PostLN은
성능 저하가
크게 발생
PreLN은
PostLN보다
성능 저하가 작음

- PostLN은 Learning Rate를 크게 하면 발산함
- **PreLN**의 상위 Layer들은 새로운 feature를 찾기보다 기존 representation을 refine하므로, **Layer Dropping에 대한 성능 저하가 적음**

2 어떤 Layer를 Drop해야 하는가?



하위 Layer들은
계속 새로운 feature를
찾기 위해
높은 dissimilarity를 가짐

PreLN에서
상위 Layer들은
Input의 영향을 덜 받음

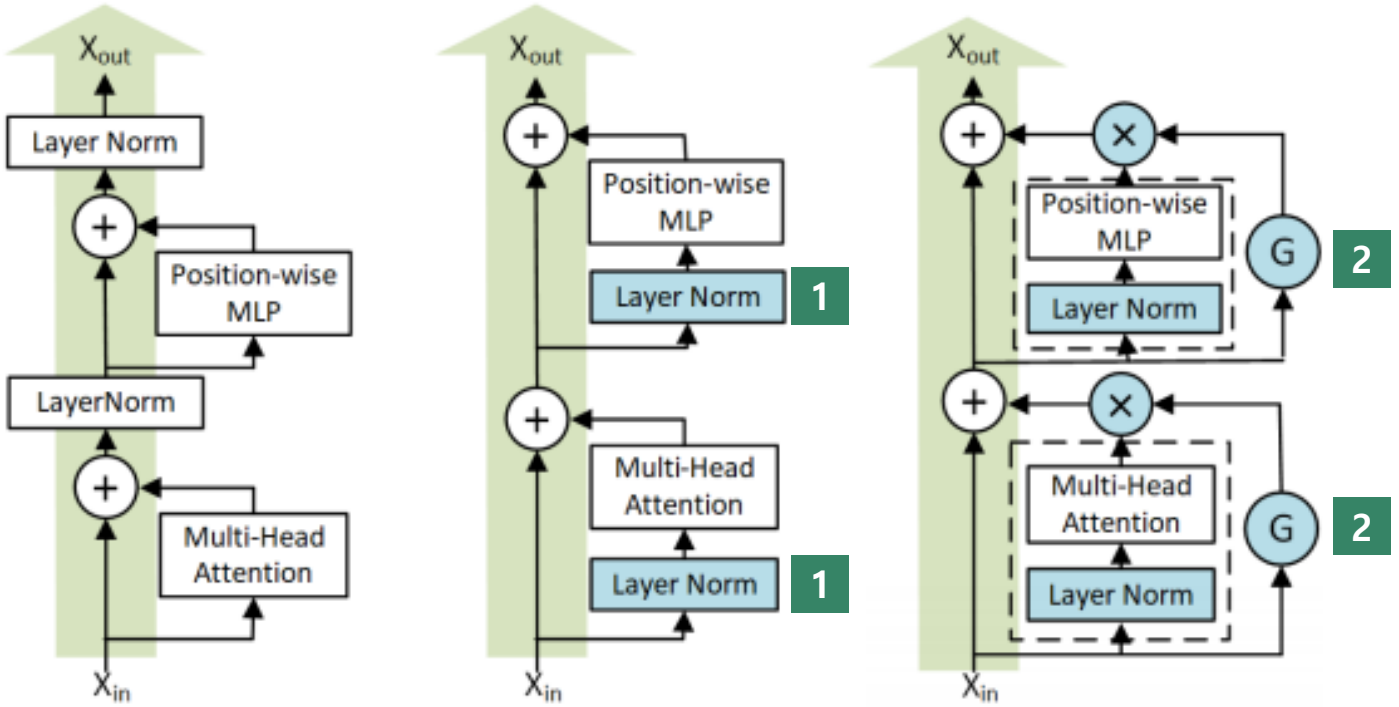
- 상위 Layer에서는 전체 estimation을 바꾸지 않으므로 Layer Dropping이 output에 미치는 영향이 작음
- **하위 Layer들은 계속해서 새로운 feature를 찾으므로 Layer Dropping이 적게 되어야 함**

03. Our Approach: Progressive Layer Dropping

각 Transformer Layer를 On/Off할 수 있는 Switchable-Transformer(ST) Block unit 제시

Layer Dropping과 함께 Transformer Network를 안정화할 수 있는 Progressive Schedule 제시

1. Switchable Transformer Blocks



(a) Original (b) Identity mapping reordering (c) Switchable Transformer

Figure 5: Transformer variants, showing a single layer block.

1 Identity mapping reordering

- Layer Norm을 sublayer들의 input stream에만 적용
- 안정성을 위해 PreLN 사용

2 Switchable Gates

- 각 미니배치마다 두 sublayer의 두 gate가 해당 transformer function을 제거하고 단순히 identity mapping connection만 유지할지 결정

$$h_i = x_i + G_i \times f_{S-ATTN}(f_{LN}(x_i)) \times \frac{1}{p_i}$$

$$x_{i+1} = h_i + G_i \times f_{FFN}(f_{LN}(h_i)) \times \frac{1}{p_i}$$

2. Progressive Layer Dropping Schedule

1 Definition

- Layer keep ratio를 단조 감소 함수로 설정해, layer dropping의 likelihood를 시간 축에 따라 증가하도록 함
- 이 때, keep ratio는 실험 결과 0.5 이상, 0.9 이하로 설정

Definition 4.1. *A progressive schedule is a function* $t \to \theta(t)$ *such that* $\theta(0) = 1$ *and* $\lim_{t\to\infty} \theta(t) \to \bar{\theta}$, *where* $\bar{\theta} \in (0, 1]$.

- $\theta(t)$는 학습 후반에 $\bar{\theta}$에 수렴
- keep ratio는 0보다 크고 1보다 작거나 같음

We constrain $\theta(t)$ to be $\theta(t) \geq \bar{\theta}$ for any t, where $\bar{\theta}$ is a limit value, to be taken as $0.5 \leq \bar{\theta} \leq 0.9$

2 Along the time dimension

- layer drop이 없는 $\theta(0) = 1$부터 서서히 layer를 drop하고, t가 충분히 클 때 $\theta(t) = \bar{\theta}$가 되도록 감소

$$\bar{\theta}(t) = (1 - \bar{\theta})exp(-\gamma \cdot t) + \bar{\theta}, \gamma > 0$$

$$\gamma = \frac{100}{T}$$

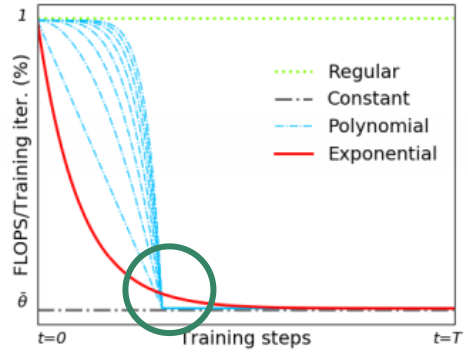
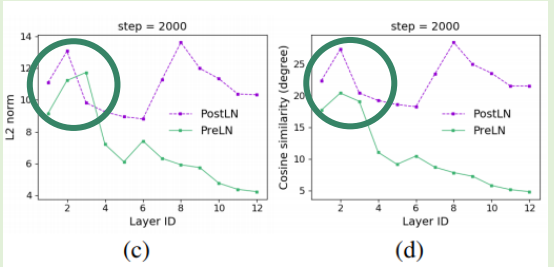


Figure 6: FLOPS per training iteration normalized to the baseline.

3 Along the depth dimension



- 각 step에서 계산된 $\bar{\theta}$를 각 ST-block마다 다르게 해서, 선형적으로 층이 낮으면 낮은 drop 확률을 가지도록 함

$$p_l(t) = \frac{i}{L}(1 - \bar{\theta}(t))$$

- 한 block 내 sublayer들은 같은 schedule을 공유

$$\theta_i(t) = \frac{i}{L}(1 - (1 - \bar{\theta}(t))exp(-\gamma \cdot t) - \bar{\theta}(t))$$

3. Putting it together

Algorithm 1 **Progressive_Layer_Dropping**

Input: $keep\_prob\ \bar{\theta}$
InitBERT($switchable\_transformer\_block$)
$\gamma \leftarrow \frac{100}{T}$
for t ← 1 to T **do**
  $\theta_t \leftarrow (1-\bar{\theta})exp(-\gamma \cdot t) + \bar{\theta}$
  step ← $\frac{1-\theta_t}{L}$
  $p \leftarrow 1$
  for l ← 1 to L **do**
    action ~ Bernoulli(p)
    if action == 0 **then**
      $x_{i+1} \leftarrow x_i$
    else
      $x_i^{'} \leftarrow x_i + f_{ATTN}(f_{LN}(x_i)) \times \frac{1}{p}$
      $x_{i+1} \leftarrow x_i^{'} + f_{FFN}(f_{LN}(x_i^{'})) \times \frac{1}{p}$
    $x_i \leftarrow x_{i+1}$
    p ← p - step
  Y ← $output\_layer(x_L)$
  loss ← $loss\_fn(\bar{Y}, Y)$
  backward(loss)

layer keep 비율
block당 drop 비율

베르누이 분포에 따라
해당 블록의 drop 여부 결정

Identity mapping

ST block의 상위 stack에
가까워질수록 drop 비율 증가

1 Identity Mapping

- ST block이 특정 iteration을 건너 뛸 때
  Identity Mapping을 진행하므로,
  forward, backward, gradient update가 필요하지 않음
- 더 direct한 경로로, 더 적은 network를 업데이트

2 Significantly Improves the Pre-training Speed

$$E(\bar{L}) = \sum_{t=0}^{T} \sum_{i=1}^{L} \theta(i,t).$$

With $\bar{\theta} = 0.5$

$$E(\bar{L}) = (3L-1)/4 \text{ or } E(\bar{L}) \approx 3L/4$$

- 12-Layer BERT를 9-Layer 속도로 학습시킬 수 있음
- **BERT의 Pre-training Speed를 크게 개선**할 수 있음
- Keep ratio가 0.5인 drop schedule에서 약 25%의 FLOPS를 절약
- **Test 및 Fine-tuning 시에는 모델을 full-depth로 복구**

04. Evaluation

Original BERT와 비교

1. Evaluation Details

1 Datasets

1 Vocab

- BERT(3,330M)와 다르게 2.8B word token 사용
- Wordpiece tokenizer로 최종 30,528 vocab 사용
- final vocab은 30,528

2 Pre-training

- BERT와 동일하게
  English Wikipedia Corpus와 BookCorpus 사용
- Training set : Val set = 300 : 1

3 Fine-tuning

- GLUE 사용

2 Training details

1 Model

- 자체 implementation BERT 모델 사용
- pytorch DDP 라이브러리로 data parallelism 핸들링
- 추가적인 컴퓨팅 오버헤드 제거

2 Baseline

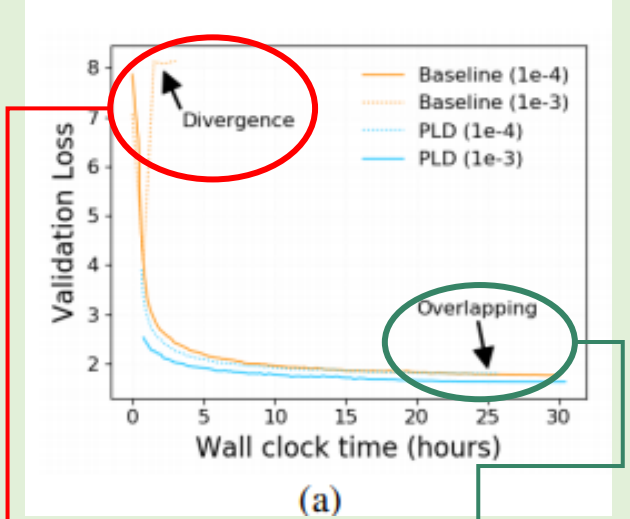
- 12-layer BERT-base, Adam, 4K batch,
  2000K steps(약 186 epochs)

3 Fine-tuning

- 각 GLUE task를 5epochs씩 학습
- 5개의 random initialization을 만들고,
  학습 결과 중 중간값 사용

2. Experimental Results(1)

1 Pre-training convergence comparisons

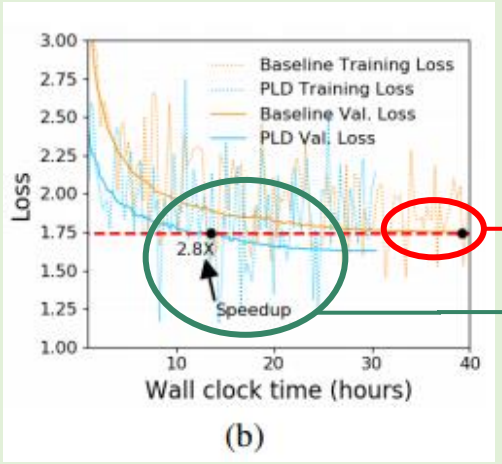


lr이 클 때,
baseline은 발산하지만
PLD는
안정적으로 수렴함

baseline과 PLD의
수렴이 거의 비슷함
PLD는 모델이 수렴하는
것을 방해하지 않음

Figure 7: The convergence curve of the baseline and our proposed method regarding the wall-clock time.

2 Speedup



Baseline은
epoch 186에서
val loss 1.75 달성

PLD는 같은 val loss를
epoch 87에 도달

Table 1: Training time comparison. Sample RD standards for sample reduction. SPD represents speedup.

	Training Time	Sample RD	SPD
Baseline ckp186	38.45h	0	1
PLD ckp186	29.22h	0	1.3×
PLD ckp100	15.56h	46%	2.5×
PLD ckp87	13.53h	53%	2.8×

같은 샘플 수일 때,
24%의 시간 절약

Training sample을
53% 적게 사용

PLD는 basline보다
2.8배 빠름

2. Experimental Results(2)

3 Downstream task accuracy

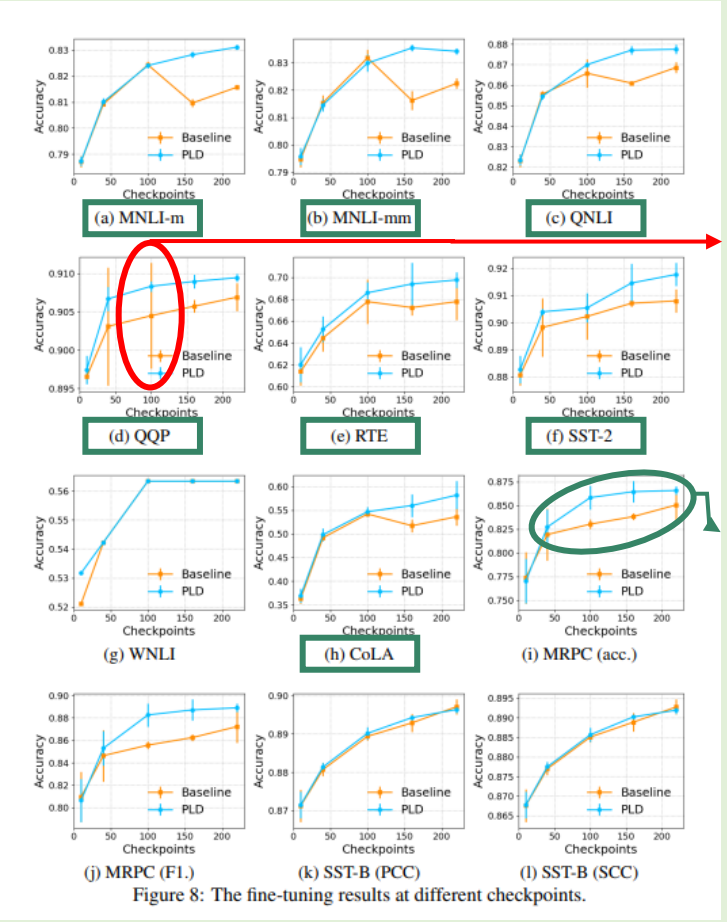


Figure 8: The fine-tuning results at different checkpoints.

baseline은 val에서 변동성이 큼

layer drop이 추가되면서 성능에 차이가 발생

Table 2: The results on the GLUE benchmark. The number below each task denotes the number of training examples. The metrics for these tasks can be found in the GLUE paper [6]. We compute the geometric mean of the metrics as the GLUE score.

Model	RTE (Acc.) 2.5K	MRPC (F1/Acc.) 3.7K	STS-B (PCC/SCC) 5.7K	CoLA (MCC) 8.5K	SST-2 (Acc.) 67K	QNLI (Acc.) 108K	QQP (F1/Acc.) 368K	MNLI-mm -/m (Acc.) 393K	GLUE
BERT_{base} (original)	66.4	88.9/84.8	87.1/89.2	52.1	**93.5**	**90.5**	71.2/89.2	**84.6**/83.4	80.7
BERT_{base} (Baseline, ckp186)	67.8	88.0/86.0	89.5/**89.2**	52.5	91.2	87.1	89.0/90.6	82.5/83.4	82.1
BERT_{base} (PLD, ckp87)	66	88.2/85.6	88.9/88.4	54.5	91	86.3	87.4/89.1	81.6/82.4	81.6
BERT_{base} (PLD, ckp100)	68.2	88.2/85.8	89.3/88.9	56.1	91.5	86.9	87.7/89.3	82.4/82.6	82.3
BERT_{base} (PLD, ckp186)	**69**	**88.9/86.5**	**89.6**/89.1	**59.4**	91.8	88	**89.4/90.9**	83.1/**83.5**	**83.2**

1. BERT_{base}(original) vs BERT_{base}(Baseline, ckp 186)
   - 연구진이 implementation한 BERT는 경쟁력 있음
2. BERT_{base}(Baseline, ckp 186) vs BERT_{base}(PLD, ckp 87)
   - Baseline이 더 좋은 성능을 보임
3. BERT_{base}(Baseline, ckp 186) vs BERT_{base}(PLD, ckp 100)
   - **PLD가 더 좋은 성능**을 보임
4. BERT_{base}(Baseline, ckp 186) vs BERT_{base}(PLD, ckp 186)
   - PLD는 15.56h, Baseline은 39.15h가 걸려 **PLD가 약 2.5배 빠름**

3. Ablation Study(1)

1 Downstream task fine-tuning sensitivity

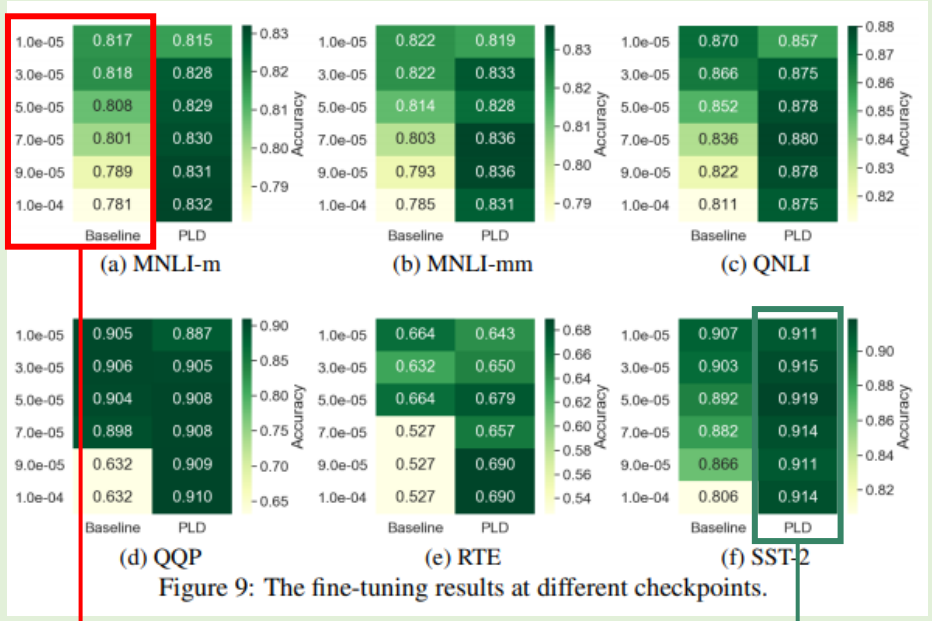


Figure 9: The fine-tuning results at different checkpoints.

Baseline은
lr 선택에 sensitive
종종 높은 lr에서
성능이 나쁨

PLD는 더 robust
종종 높은 lr에서
더 좋은 성능을 보임

2 The Effect of $\bar{\theta}$

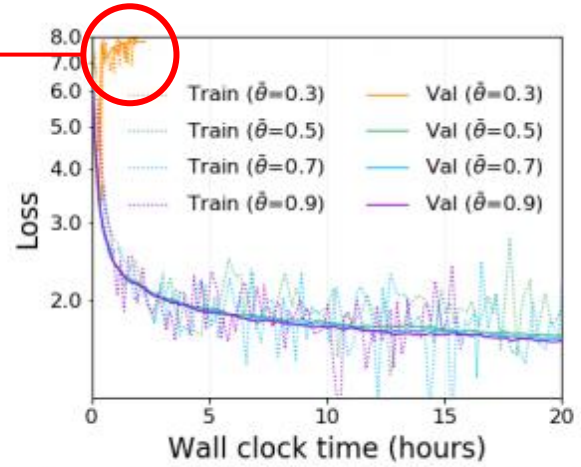


Figure 10: Convergence curves varying the keep ratio $\bar{\theta}$.

$\bar{\theta}$가 너무 작으면
알고리즘이 발산하는 경향을 보임

keep ratio $\bar{\theta}$ and identify $0.5 \leq \bar{\theta} \leq 0.9$ as a good range,

3. Ablation Study(2)

Table 3: Ablation studies of the fine-tuning results on the GLUE benchmark.

Model	RTE (Acc.)	MRPC (F1/Acc.)	STS-B (PCC/SCC)	CoLA (MCC)	SST-2 (Acc.)	QNLI (Acc.)	QQP (F1/Acc.)	MNLI-m/mm (Acc.)	GLUE
BERT (Original)	66.4	**88.9**/84.8	87.1/89.2	52.1	**93.5**	**90.5**	71.2/89.2	**84.6/83.4**	80.7
BERT + PostLN	67.8	88.0/86.0	89.5/89.2	52.5	91.2	87.1	89.0/90.6	82.5/83.4	82.1
BERT + PreLN + Same lr	66.0	85.9/83.3	88.2/87.9	46.4	90.5	85.5	89.0/90.6	81.6/81.6	80.2
BERT + PreLN + lr↑	67.8	86.7/84.5	**89.6/89.1**	54.6	91.9	88.1	89.3/**90.9**	**83.6/83.7**	82.6
Shallow BERT + PreLN + lr↑	66.0	85.9/83.5	89.5/88.9	54.7	91.8	86.1	89.0/90.6	82.7/82.9	81.8
BERT + PreLN + lr↑ + Rand.	68.2	88.2/86.2	89.3/88.8	56.8	91.5	87.2	88.6/90.3	82.9/83.3	82.7
BERT + PreLN + lr↑ + TD	68.2	88.6/**86.7**	89.4/88.9	55.9	91.3	86.8	89.1/90.7	82.7/83.1	82.7
BERT + PreLN + lr↑ + PLD	**69.0**	**88.9**/86.5	**89.6/89.1**	**59.4**	91.8	88.0	**89.4/90.9**	83.1/83.5	**83.2**

3 PLD vs. PreLN

1. BERT + PostLN vs BERT + PreLN + Same lr
   - PreLN보다 PostLN이 더 성능이 좋음

2. BERT + PreLN + lr↑ vs BERT + PreLN + lr↑ + PLD
   - PreLN은 PLD보다 24%의 자원을 더 사용했음에도 성능이 낮음

4 PLD vs. shallow network

1. Shallow BERT + PreLN + lr↑ vs BERT + PreLN + lr↑
   - Shallow BERT는 BERT보다 성능이 떨어짐

2. Shallow BERT + PreLN + lr↑
   vs BERT + PreLN + lr↑ + PLD
   - Shallow BERT는 PLD와 GFLOPS가 똑같지만 (9-Layer 연산) 성능이 더 많이 떨어짐

3. Ablation Study(3)

Table 3: Ablation studies of the fine-tuning results on the GLUE benchmark.

Model	RTE (Acc.)	MRPC (F1/Acc.)	STS-B (PCC/SCC)	CoLA (MCC)	SST-2 (Acc.)	QNLI (Acc.)	QQP (F1/Acc.)	MNLI-m/mm (Acc.)	GLUE
BERT (Original)	66.4	**88.9**/84.8	87.1/89.2	52.1	**93.5**	**90.5**	71.2/89.2	**84.6/83.4**	80.7
BERT + PostLN	67.8	88.0/86.0	89.5/89.2	52.5	91.2	87.1	89.0/90.6	82.5/83.4	82.1
BERT + PreLN + Same lr	66.0	85.9/83.3	88.2/87.9	46.4	90.5	85.5	89.0/90.6	81.6/81.6	80.2
BERT + PreLN + lr↑	67.8	86.7/84.5	**89.6/89.1**	54.6	91.9	88.1	89.3/**90.9**	**83.6/83.7**	82.6
Shallow BERT + PreLN + lr↑	66.0	85.9/83.5	89.5/88.9	54.7	91.8	86.1	89.0/90.6	82.7/82.9	81.8
BERT + PreLN + lr↑ + Rand.	68.2	88.2/86.2	89.3/88.8	56.8	91.5	87.2	88.6/90.3	82.9/83.3	82.7
BERT + PreLN + lr↑ + TD	68.2	88.6/**86.7**	89.4/88.9	55.9	91.3	86.8	89.1/90.7	82.7/83.1	82.7
BERT + PreLN + lr↑ + PLD	**69.0**	**88.9**/86.5	**89.6/89.1**	**59.4**	91.8	88.0	**89.4/90.9**	83.1/83.5	**83.2**

5 PLD vs. Random drop

1. BERT + PreLN + lr↑ + Rand.
   - 같은 computational cost에서
     shallow BERT보다 성능이 좋고 preLN보다 빠름
2. BERT + PreLN + lr↑ + Rand.
   vs BERT + PreLN + lr↑ + PLD
   - Dynamic을 고려하지 않아 PLD보다 성능이 낮음

6 Schedule impact analysis

1. BERT + PreLN + lr↑ + Rand. + TD *only(32-bit*)*
   - GLUE score가 Rand.과 동일함
   - PLD의 성능 향상은 대부분 DD에서 옴
   - 하지만, TD 없이는 half-precision에서 NaN과 함께
     발산하므로 반드시 full-precision으로 학습해야 하며,
     이는 학습 속도를 느려지게 함

05. Conclusion

1. Conclusion

Unsupervised Language Model의 Pre-training은
NLP task에서 SOTA로 가는 중요한 Step

이러한 모델을 학습시킬 때 매우 긴 시간이 걸리므로,
학습 소요 시간을 단축하는 것은 매우 중요함

연구진은 NLP task의 대표적인 Pre-training model인 BERT를 위한
효과적인 학습 알고리즘을 연구

연구진은 광범위한 분석을 수행했고,
트랜스포머 기반 모델에서 Stochastic Depth가 중요하다는 것을 깨달음

이 인사이트를 가지고,
연구진은 **Switchable-Transformer block**과
Progressive Layer-wise Drop Schedule(PLD)를 제안함

이 새로운 학습 전략은
더 빠르게 깊은 모델을 밑바닥부터 학습하고,
성능에서도 경쟁력이 있음

감사합니다.