고려대학교 KOREA UNIVERSITY

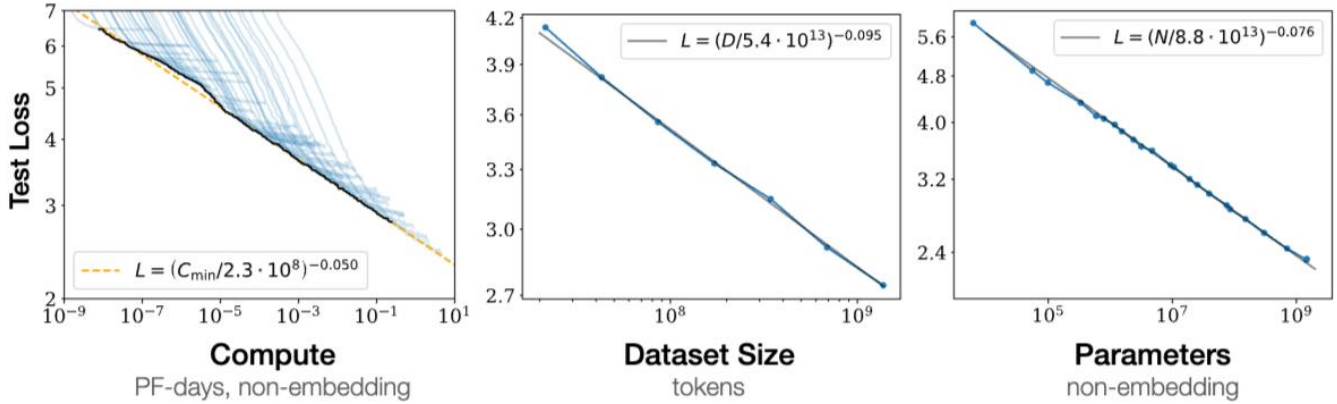
Natural Language Processing & Artificial Intelligence

연구실 세미나

고려대학교 NLP&AI 연구실
발표자: 손준영

◆ Introduction

Scaling Laws (Kaplan et al. "Scaling laws for neural language models")

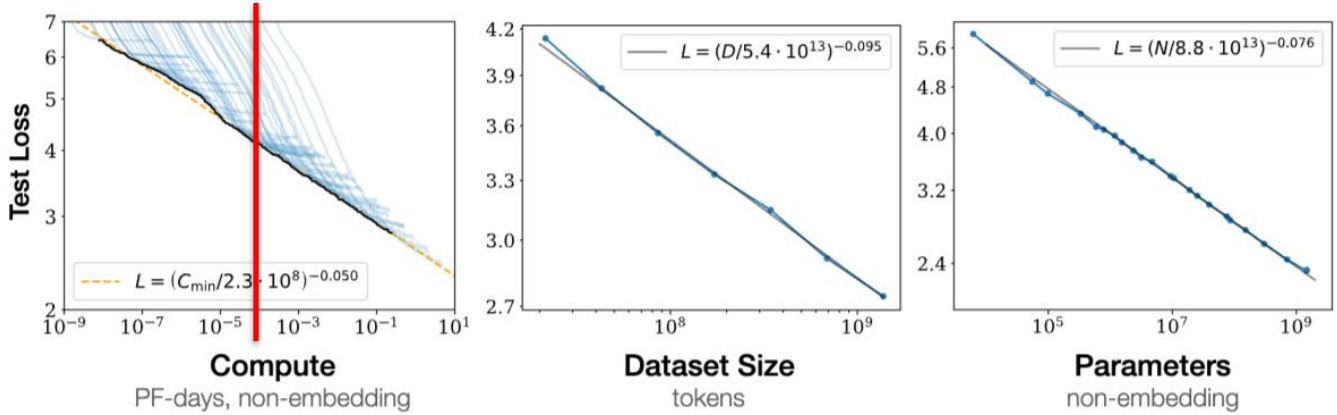


- Test Loss가 모델 크기, 데이터셋 크기, 계산량과 멱법칙(Power-law) 관계를 가지며 매끄럽게 감소함

→ **규모의 중요성**: 모델 성능이 아키텍처의 세부 사항보다는 **"규모(scale)에 가장 크게 의존적"**
➔ 이러한 "규모(scale)"의 힘은 큰 모델을 만들려는 경쟁 촉발
➔ 결과적으로 대부분의 연구자와 실무자가 언어 모델 훈련을 시도하기 어려워짐

◆ Introduction

Scaling Laws under limited compute



- 하루동안 GPU 1개로 시간·자원이 고정되어도 스케일링 법칙이 성립할까?
- 관점 1: 고정된 C 내에서 최적의 N·D 찾기
  - $C \approx N \times D$ 관계가 그대로 유지 → 따라서 여전히 $L \propto C^{-a}$
  - 따라서 C 안에서 N이 너무 크면(D가 작으면) : under-fitting
  - N이 작고 D가 크면: over-fitting
- 관점 2: 주어진 장비에서 토큰(T) /sec을 최대화해 하루동안 학습할 수 있는 총 C를 키운다
  - Flash Attention과 같은 최적화 알고리즘

◆ Cramming: Training a Language Model on a Single GPU in One Day

CRAMMING: TRAINING A LANGUAGE MODEL ON A SINGLE GPU IN ONE DAY

Jonas Geiping
University of Maryland, College Park
jgeiping@umd.edu

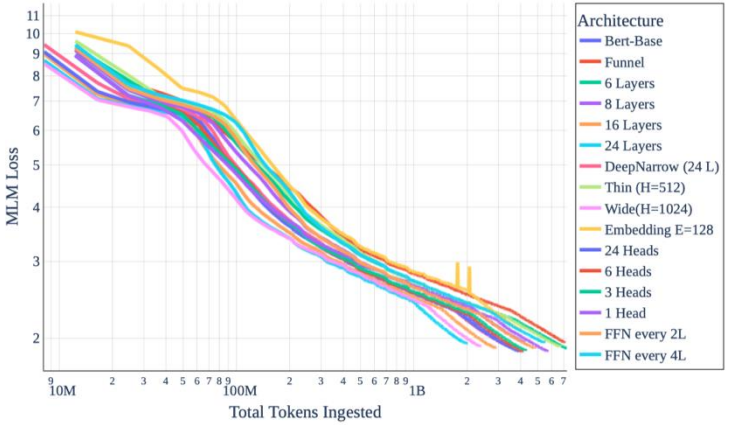
Tom Goldstein
University of Maryland, College Park
tomg@umd.edu

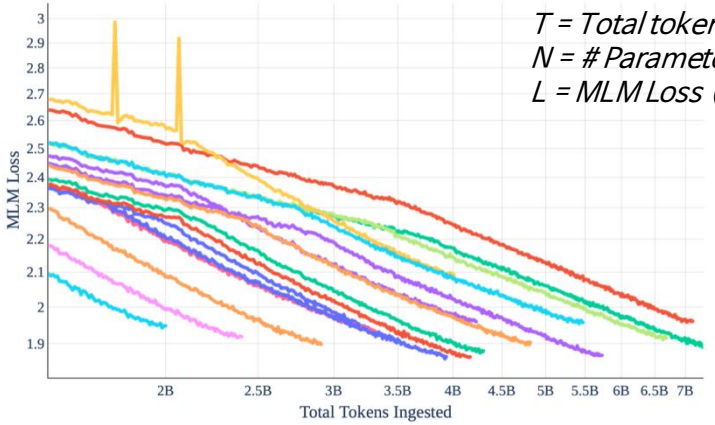
- ICML 2023 Accepted
- Cramming: "보통의 연구자가 단일 GPU로 단 하루 만에 BERT를 처음부터 훈련했을 때 어느 정도의 다운스트림 성능을 달성할 수 있는지?"

➔ 이 논문은 기존의 스케일링 법칙이 저사양 환경에서도 여전히 유효함을 보여주면서도,
[1]아키텍처, [2]훈련 방식, [3]데이터셋 최적화를 통해 효율성을 극대화하는 방법을 제시하고 있음

◆ Cramming: Training a Language Model on a Single GPU in One Day

Scaling Laws under a Cramming Regime





T = Total tokens ingested (x-축)
N = # Parameters (색상/곡선)
L = MLM Loss (y-축)

하루만에 달성할 수 있는 성능을 높이려면?

1. 토큰수(**T**) 증가 ⇒ Loss(**L**) 감소
2. 모델 크기(**N**) ↑ ⇒ 초기 Loss(**L**) 감소, 하지만 토큰수(**T**) ↓ (데이터당 효율 증가 but throughput 감소)
3. 결과: 서로 다른 모델 크기(**N**)에도 1B 토큰 이후 Loss(**L**)≈1.9에서 수렴

➔ 모델 크기(**N**)·토큰수(**T**) "등가교환"만으론 한계

◆ Cramming: Training a Language Model on a Single GPU in One Day

Finding the One-Day Optimum

Group	Target	Accelerator	Time Limit	Total exaFLOP
(Devlin et al., 2019)	BERT	16 TPU	4 days	680
(Dettmers, 2018)	BERT	8 V100	11 days	950
(Narasimhan, 2019)	BERT-large	1472 V100	47 min	519
(Raffel et al., 2020)	T5-base	16 TPUv3	1 day	170
(Iandola et al., 2020)	squeezeBERT	8 Titan RTX	4 days	361
(Narang et al., 2021)	T5 variations	16 TPUv3	1.75 days	298
(Tay et al., 2021)	T5-small-L16	16 TPUv3	11.2 hours	82
(Izsak et al., 2021)	BERT variation	8 V100	1 day	86
(Liu et al., 2019)	roBERTa-base	1024 V100	1.25 day	13 824
(Chowdhery et al., 2022)	PaLM	6144 TPUv4	50 days	7 299 072
Our Setup 1	BERT variation	1 rtx2080ti	1 day	5
Our Setup 2	BERT variation	1 rtxa4000	1 day	8
Our Setup 3	BERT variation	1 rtxa6000	1 day	13

How to reach the "sweet spot" on 1 GPU?

① Kernel-level tricks — Flash-Attn 3, Unpadding, torch.compile (단위 시간당 토큰 처리량(**T**) **증가**)
② Parameter pruning — Bias-off, Head ↓, GeGLU (모델 크기 최적화(**N 감소**))
③ Long-seq data packing — 8 k tokens / sample (배치 연산 효율성 증가 (**T 증가**)))

◆ Cramming: Training a Language Model on a Single GPU in One Day

Modifying the Architecture (1)

- **Attention Block:**
  1. Disabling QKV bias — 파라미터 수 1-2% 감소, GPU Throughput 1% 증가 ➔ 성능 손실 없이 효율화
  2. Reducing Attention Heads (12개 활용) — 계산량↓, GPU 병렬화 효율↑

- **Feedforward Block:**
  1. Disabling Linear bias — 파라미터 감소, gradient 계산 속도 개선
  2. Gated Linear Unit (GLU) 기반 활성화 함수 — 표현력, 수렴속도, 성능 ↑

[Detail. GLU 기반 Activation + FFN]
$g(x) = \sigma(xW_g) \in (0,1)$ ➔ 게이트
$a(x) = xW_a$ ➔ 정보 흐름
$h = (g(x) \otimes a(x))W_{FFN}$ ($\otimes$: element-wise multiplication)
➔ **정보 흐름 제어**

◆ Cramming: Training a Language Model on a Single GPU in One Day

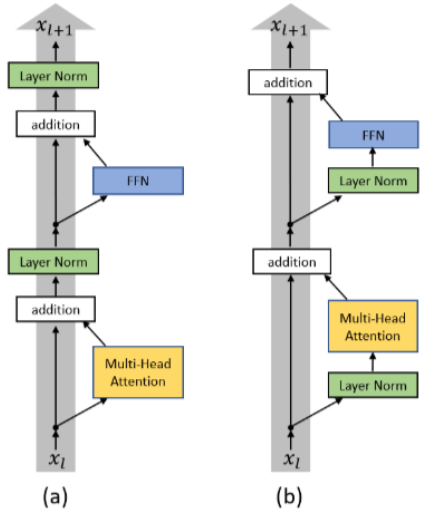
Modifying the Architecture (2)

- **Normalization:**
  1. Layer Normalization after Embedding Layer:
  2. Pre-normalization: Stable training

- **Head Block:**
  1. Removing non-linear heads and decoder bias:
  2. Optimizing token prediction:

Token distributions 안정화
gradient 폭발/소실 완화, 훈련 안정화

경량화
학습비용 ↓



Xiong, Ruibin, et al. 에 따르면, Layer Normalization (LN)의 위치가 학습 초기의 gradient의 규모에 결정적인 역할을 미치며,
Pre-LN의 경우, 더 깊은 모델일 수록 학습이 더 안정적으로 수렴하는 결과를 보였음

◆ Cramming: Training a Language Model on a Single GPU in One Day

Modifying the Training Setup

- **Training Objective:**
  1. Masked Language Modeling (MLM) 단일 목표
     - 마스킹 비율 15 % (Devlin et al., 2019 원본)
     - 마스킹 40 % / MSE / L1 Loss → 이득 없음

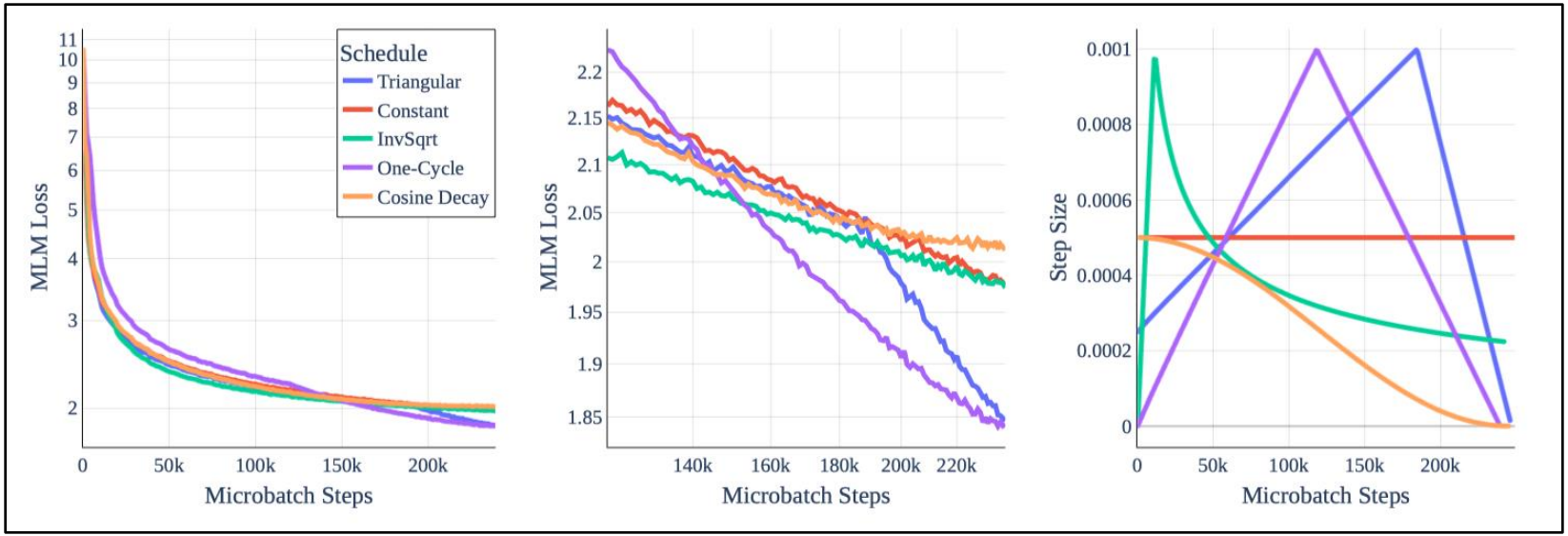
- **Optimizer:**
  1. Adam W ($\beta 1 = 0.9, \beta 2 = 0.98, \varepsilon = 1e{-}12$, wd = 0.01)
  2. Gradient Clipping 0.5 → 수렴 안정화, 추가 비용 0
  3. 다른 적응형/고차 옵티마이저(Adafactor 등)나 SGD는 이 제한된 예산에서는 전반적 이득이 없었음

- **Learning Rate Schedule**
  1. Budget-aware one-cycle (Izsak et al., 2021)
  2. 다양한 curve 실험 중 최저 사전-훈련 Loss 달성

- **Batch Size:**
  1. 2080Ti 기준 가능한 micro-batch = 96 ➔ Gradient accumulation (다른 스케쥴링도 조정 필요)
  2. Batch Size Scheduling: 훈련 진행과 함께 누적 step 선형 ↑
     ➔ One-cycle LR과 조합이 좋음 ➔ 학습 속도·성능 ↑

◆ Cramming: Training a Language Model on a Single GPU in One Day

Modifying the Training Setup



◆ Cramming: Training a Language Model on a Single GPU in One Day

Optimizing The Dataset

- **Data-Source Exploration:**
  1. The Pile 하위 셋 (Gutenberg + Books3 + Wiki-en)
  2. C4 (20 M items streaming)
  3. 각 소스별로 WordPiece 토크나이저 재학습 → 성능 비교 후 The Pile 토크나이저 활용

- **Filtering & Sorting Existing Data:**
  1. Deduplication (정확 부분 중복) → 큰 효과 없음
     ➔ 토큰 압축 비율에 따른 필터링: 압축 불가능" 시퀀스 필터링 (t = #tok / #char 〉 0.3)
     (필터 기준: 한 샘플의 토큰 수가 원문 문자 수의 t 배를 넘으면 그 샘플을 버림)
  2. 토큰 빈도 기준 시퀀스 정렬 → 흔한(가능성 높은?) 문장 먼저 소비 → 수렴 속도 ↑
     ➔ 초기에 '좋은' 데이터부터 보게 하면 학습 초기 단계에서 더 빠르게 유의미한 패턴을 잡을 수 있음

- **Vocabulary:**
  1. 학습 막판 batch ↑ (최대 4032/4096) → C4에서 손실 ↓, 분포 변동 완화 → 안정적인 학습 신호 제공
     ➔ 데이터가 정렬되어 있을 때(유사한 샘플이 묶여있을 때) 효과 극대화 (gradient 추정의 분산 감소)
  2. Vocab size sweep (16k ~ 64k) → GLUE 평균은 크기 ↑에 비례하나, MNLI에서는 32k에서 포화

- **Take-away:**
  데이터 품질·순서를 다듬는 것만으로도 아키텍처·커널 최적화 이후 추가 2 - 3 pt downstream 이득 확보.
  "Scaling Laws ≠ 토큰 품질 한계" — Better tokens beat more compute.

◆ Cramming: Training a Language Model on a Single GPU in One Day

Optimizing The Dataset

Dataset	Batch Size	MNLI (m)
Bookcorpus-Wikipedia	1536	79.8
The Pile	1536	80.5
The Pile (natural data subset)	1536	80.8
C4-Subset	1536	79.1
Bookcorpus-Wikipedia, Deduduplication > 100	1536	79.9
Bookcorpus-Wikipedia, Deduduplication > 50	1536	79.5
Bookcorpus-Wikipedia, filtered with $t = 0.3$, sorted	1536	80.8
Bookcorpus-Wikipedia, sorted	1536	81.0
C4-Subset, Deduduplication > 100	1536	79.2
C4-Subset, filtered with $t = 0.3$	1536	79.9
C4-Subset, filtered with $t = 0.3$, sorted	1536	81.4
C4-Subset, filtered with $t = 0.3$, larger, sorted	1536	81.9
Bookcorpus-Wikipedia	4032	80.5
C4-Subset, filtered with $t = 0.3$	4032	82.2
C4-Subset, filtered with $t = 0.3$, sorted	4032	82.5
C4-Subset, filtered with $t = 0.3$	8064	80.9

◆ Cramming: Training a Language Model on a Single GPU in One Day

Comparison in GLUE-dev performance of baseline BERT to the crammed model

	MNLI	SST-2	STSB	RTE	QNLI	QQP	MRPC	CoLA	GLUE
BERT-base (Fully trained)	83.2/83.4	91.9	**86.7**	**59.2**	**90.6**	**87.7**	**89.3**	**56.5**	**80.9**
BERT-base (No Pretrain)	34.1/34.1	79.9	17.8	47.3	50.0	68.6	77.9	0.0	45.5
Trained for 1 day on a **2080ti**:									
BERT (normal protocol)	58.7/57.8	79.8	16.6	50.9	55.4	71.1	70.1	7.3	52.0
BERT ((Izsak et al., 2021))	75.0/75.7	-	-	52.3	84.6	84.4	82.2	33.8	69.7
crammed BERT	82.8/83.4	91.5	83.1	54.0	89.0	87.2	86.2	47.2	78.3
Trained for 1 day on an **A4000**:									
BERT (normal protocol)	58.0/56.5	79.4	17.0	51.6	54.2	70.6	74.1	8.2	52.2
BERT ((Izsak et al., 2021))	58.8/59.6	-	-	-	-	-	81.4	0.0	49.9
crammed BERT	83.0/83.2	91.6	84.8	54.7	88.5	86.9	86.4	43.7	78.1
Trained for 1 day on an **A6000**:									
BERT (normal protocol)	56.3/54.8	81.2	21.8	49.5	56.4	65.1	74.8	10.3	52.2
BERT ((Izsak et al., 2021))	76.2/76.5	87.4	78.5	49.1	85.0	84.1	83.2	36.3	72.9
crammed BERT	**83.9/84.1**	**92.2**	84.6	53.8	89.5	87.3	87.5	44.5	78.6

◆ Cramming: Training a Language Model on a Single GPU in One Day

Ablation study, which improvements were most important?

	MNLI	SST-2	STSB	RTE	QNLI	QQP	MRPC	CoLA	GLUE
crammed BERT	**83.9 / 84.1**	**92.2**	84.6	53.8	**89.5**	**87.3**	87.5	**44.5**	**78.6**
+ original data	82.2 / 82.7	92.0	83.6	49.8	**89.5**	87.0	85.9	42.5	77.3
+ original train	50.0 / 50.4	80.7	13.7	52.0	59.8	65.1	73.2	7.2	50.2
+ original arch.	35.4 / 35.2	49.1	-	52.7	49.5	0.0	0.0	0.0	27.7
+ minimal train mod.	81.9 / 82.6	91.4	**85.5**	**54.9**	88.2	87.0	**88.4**	43.6	78.1
+ minimal arch. mod.	83.2 / 83.5	91.7	82.0	52.0	88.9	86.8	83.6	38.3	76.7

- Original data: 데이터만 원래로 돌려도 성능 저하는 작음 → 데이터 개선(필터링/정렬)이 유의미하지만 엄청 큰 폭은 아님
- Original train: 학습 설정(특히 LR schedule·dropout) 없이는 성능이 급락 ➔ 1GPU·하루라는 제약에 부적합
- Original arch: 제외시 거의 학습 불가

- "적은 자원 = 모델만 줄인다"는 오해 ➔ 균형 잡힌 전체 레시피가 필요.
- 아키텍처·학습 방식·데이터 3축이 서로 보완 → 합산 +26 ~ +51 pt.

  e.g. Pre-Norm → 안정화 → 공격적인 LR 스케줄링 → 이에 맞는 batch size 스케줄링 + 더 품질 좋은 데이터

◆ Cramming: Training a Language Model on a Single GPU in One Day

WHAT HAPPENS WHEN TRAINING LONGER?

	MNLI	SST-2	STSB	RTE	QNLI	QQP	MRPC	CoLA	GLUE
BERT-Base (Fully trained)	83.2/83.4	91.9	86.7	59.2	90.6	87.7	89.3	56.5	80.9
BERT-Base (No Pretrain)	34.1/34.1	79.9	17.8	47.3	50.0	68.6	77.9	0.0	45.5
ROBERTA-Base	**86.6**/86.4	93.7	**90.4**	**77.3**	92.1	**88.3**	**91.4**	**60.2**	**85.1**
Crammed BERT (A6000)	83.9/84.1	92.2	84.6	53.8	89.5	87.3	87.5	44.5	78.6
Trained for 2 days on 8 **A6000**:									
Crammed BERT	86.5/**86.7**	**93.8**	86.8	53.4	91.6	88.0	88.2	42.9	79.8
Crammed BERT (no clipping)	86.1/**86.7**	93.2	87.1	55.2	**92.1**	88.3	90.2	46.6	80.6

- **크래밍 레시피 "확장성" 검증**
  - 1 GPU·24 h → 8 GPU·48 h 로 Compute ×16
  - 동일 하이퍼파라미터, 데이터 1-pass 반복 허용

- **성능 결과 (GLUE)**

MNLI 86.5 / SST-2 93.8 → BERT-base 완전학습 surpass
GLUE 평균 79.8 → RoBERTa-base 85.1 에 근접
Gradient clipping OFF 해도 동일 (80.6) → 안정성 확보

- **태스크별 관찰**
  - 대형 셋(MNLI, QQP, QNLI, SST-2) : 예산 ↑ 만큼 선형 개선
  - CoLA·RTE : +16× compute에도 거의 정체
    - CoLA → 하이퍼파라미터 민감도 / 문법 수용능 용량 한계?
    - RTE → 훈련 셋 소량, scaling benefit 작음

- **인사이트?**

1. Cramming은 compute-invariant — 더 큰 예산에서도 효과O
2. 데이터 정렬 상태 + 반복 training 이 "오버핏" 없이 실질적 개선
3. Task-specific 병목(CoLA, RTE) 해결방안 필요..

◆ Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference

Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference

Benjamin Warner[1†] Antoine Chaffin[2†] Benjamin Clavié[1†]
Orion Weller[3] Oskar Hallström[2] Said Taghadouini[2]
Alexis Gallagher[1] Raja Biswas[1] Faisal Ladhak[4*] Tom Aarsen[5]
Nathan Cooper[1] Griffin Adams[1] Jeremy Howard[1] Iacopo Poli[2]

[1]Answer.AI [2]LightOn [3]Johns Hopkins University [4]NVIDIA [5]HuggingFace

*†: core authors, *: work done while at Answer.AI*

Correspondence: {bw,bc}@answer.ai, antoine.chaffin@lighton.ai

- ACL 2025 Accepted
- ModernBERT는 최근 LLM 개발에서 배운 것들을 이러한 작은 모델들에 적용하였음 ➔ 더 나은 효율성, 성능

◆ Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference

Architectural Improvements

- **바이어스 항 (Bias Terms) 비활성화**
  - Dense / FFN / LayerNorm 전부 bias=False
  - 단 MLM Decoder Linear만 bias 유지 → 파라미터·메모리 ↓ (~-2 %), TensorCore 정렬 ↑
- **위치 임베딩 (Positional Embeddings) → RoPE**
  - 절대 PE → Rotary PE (Su et al., 2024)
  - 긴 문맥(8192+) 확장 용이, 구현 간단, 동일 FLOP
- **정규화 (Normalization)**
  - 블록 앞 Pre-LN → 그래디언트 안정, Warm-up 불필요
  - Embedding 뒤 LN 한 층 추가 → Token 분포 정규화
- **활성화 함수 (Activation)**
  - GeLU → GeGLU(Shazeer, 2020)
  - FFN 게이팅으로 표현력 ↑, BEIR 기준 +0.2 pt

◆ Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference

Efficiency Improvements

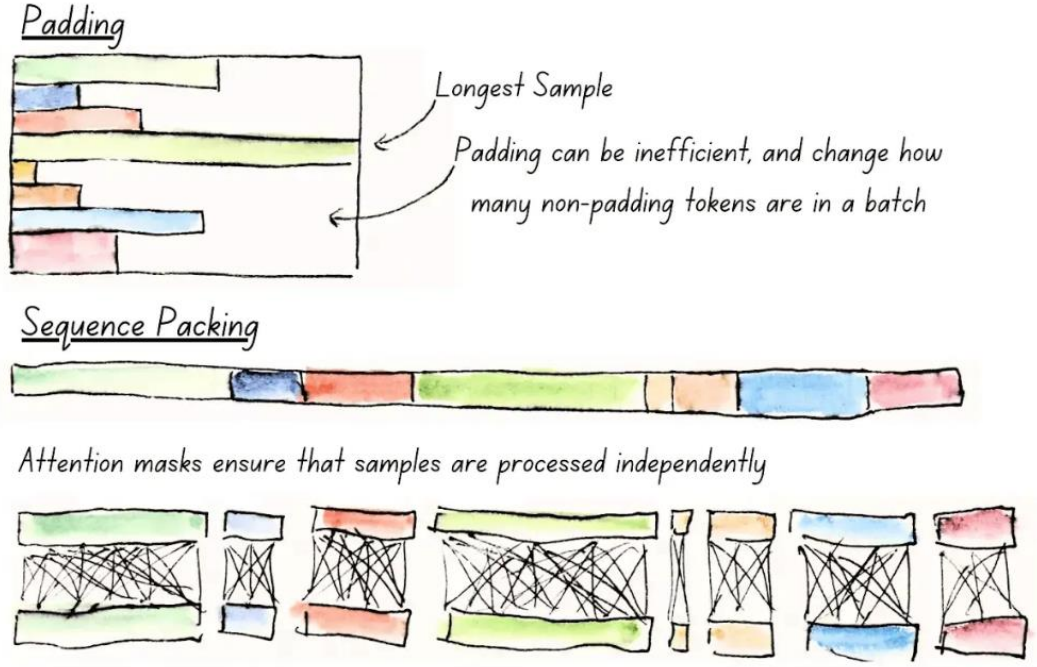
- **교대 어텐션 (Alternating Attention)**
  - Global Attention: 매 세번째 레이어마다 모두 상호 참조 가능
  - Local Attention: 작은 슬라이딩 윈도우에서만 상호 참조 가능 → Local 128-window Attention
    ➔ Global-only 수준 성능을 유지하면서 · FLOP 감소

- **언패딩 (Unpadding)**
  - Variable-len batch → Token IDs 압축 후 Flash-Attn 호출
  - 10 - 20 % throughput ↑, 메모리 ↓

- **Flash Attention**
  - Global layer → Flash-Attn 3 (H100 최적)
  - Local layer → Flash-Attn 2
  - end-to-end wall-time ~1.3 × faster vs. vanilla

- **torch.compile (PyTorch 2)**
  - JIT compile 가능한 모듈 전체 최적화
  - 추가 코드 변경 없이 학습 TPS ≈ +10%

◆ Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference



◆ Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference

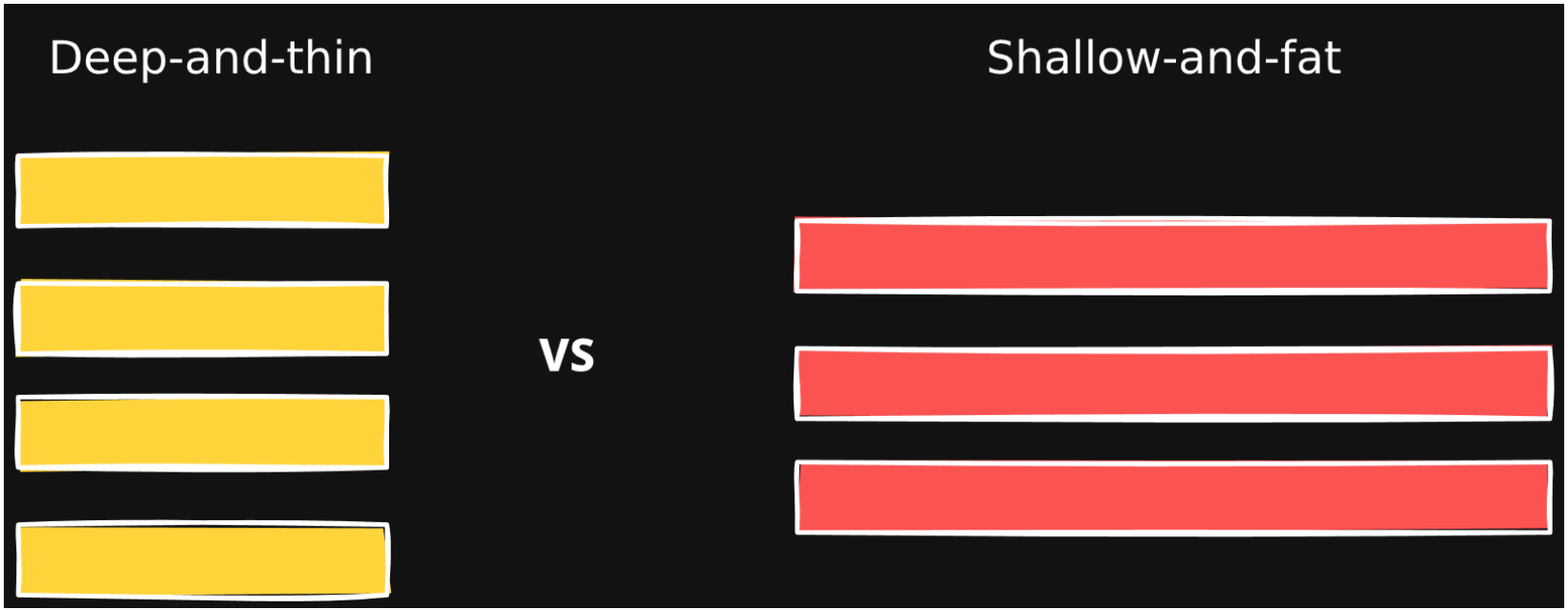
Model Design

- **하드웨어 인지 모델 설계: GPU-Centric 목표**
  - 서버(T4·A10·A100·H100) + 소비자(RTX 3090·4090) 모두에서
    "한 번의 ckpt → 최대 추론 효율" 달성 Global-only 수준 성능을 유지하면서 · FLOP 감소
  - 행·열 차원 모두 64 배수 → Tensor Core full-util

- **Deep & Narrow 아키텍처**
  - Deep & Narrow vs. Shallow & Wide: Deep·Narrow는 성능 / Wide는 속도
  - ModernBERT는 이 두 가지 사이의 성능·속도의 절충점을 타협
    레이어 수 ↑, hidden size ↓ → 더 작은 MatMul tile • 더 긴 시퀀스 캐시

- **모델 스펙**
  - Base 22L / H 768 → 149 M params
  - Large 28L / H 1024 → 395 M params

◆ Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference



◆ Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference

	ModernBERT-large	jina-XLM-RoBERTa	RoBERTa-large
Parameters	400M	550M	355M
Hidden states	1,024	1,024	1,024
Intermediate dims	2,624	4,096	4,096
Attention heads	16	16	16
Layers	28	24	24
Vocabulary size	50,368	250,002	50,265

◆ Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference

◆ Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference

Training Data & Tokenizer

- **학습 데이터**
  - 2조 토큰 (웹 + 코드 + 과학 문서)
    여러 ablation를 통해 mixure 비율 결정

- **OLMo 토크나이저 채택**
  - GPT-NeoX-20B의 BPE-based tokenizer의 수정된 버전
  - 코드 토큰 효율 향상에 집중
  - Vocab size = 50,368 (64 배수) → 64배수 맞추려고 unused 83개 추가

- **Sequence Packing + Unpadding**
  - Unpadding으로 인한 batch size의 불균형 방지를 위해 Greedy packing 적용 → 99% 효율

◆ Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference

Training Details

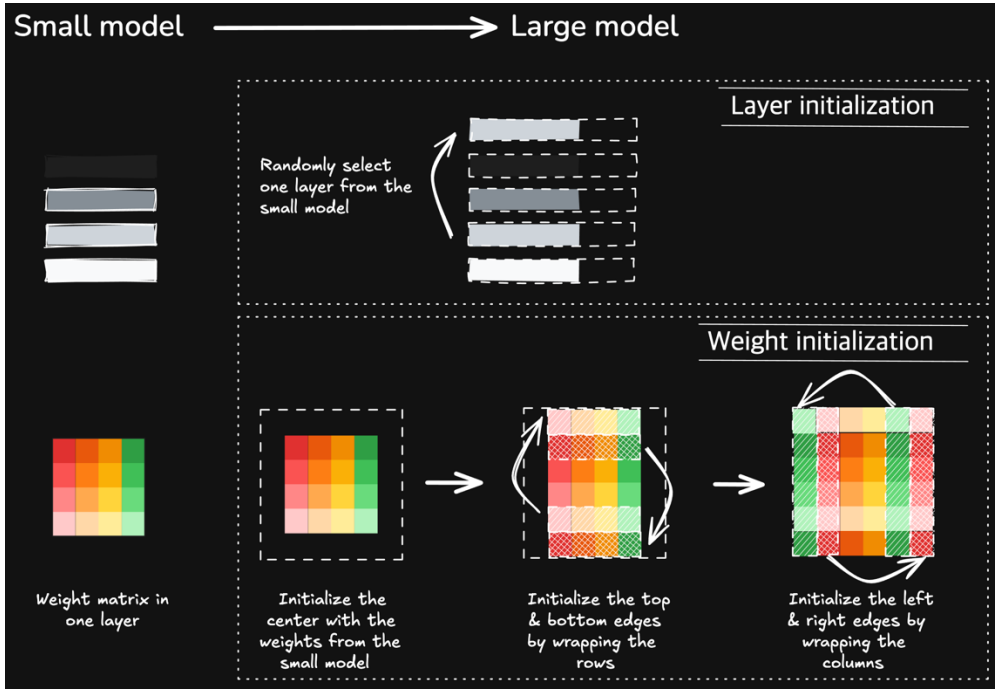
- **Masked Language Modeling (MLM)**
  - 기존 BERT의 15% 대신 30% 마스킹 비율 사용
  - Next-Sentence Prediction (NSP) 제거 → 기존 흐름과 동일
- **Optimizer**
  - StableAdamW 사용: 더 안정적인 학습 및 다운스트림 성능
    - *"per-parameter update clipping"이 추가된 변형*
- **Learning Rate Schedule**
  - 사다리꼴 형태의 학습률 스케줄링 활용 (Warmup-Stable-Decay): 웜업 후 유지, 감소
- **Batch Size Schedule**
  - 훈련 초기에는 작은 배치 → 점진적으로 배치 크기 증가 → 훈련 속도 가속화
- **Context Length Extension**
  - 1024 학습 → 8192로 확장

◆ Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference

Weight Initialization and Tiling

- Layer 확장
  - Small 모델의 layers 중 랜덤 선택하여 초기화
- Weight 확장
  - Tiling 방식: small 모델의 가중치를 매트릭스 중앙에 배치하고 회전하며 채우는 방식

➔ 이미 학습된 패턴을 효과적으로 활용



◆ Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference

Language Benchmark Evaluation

		IR (DPR)			IR (ColBERT)		NLU	Code	
	Model	BEIR	$MLDR_{OOD}$	$MLDR_{ID}$	BEIR	$MLDR_{OOD}$	GLUE	CSN	SQA
Base	BERT	38.9	23.9	32.2	49.0	28.1	84.7	41.2	59.5
	RoBERTa	37.7	22.9	32.8	48.7	28.2	86.4	44.3	59.6
	DeBERTaV3	20.2	5.4	13.4	47.1	21.9	88.1	17.5	18.6
	NomicBERT	41.0	26.7	30.3	49.9	61.3	84.0	41.6	61.4
	GTE-en-MLM	41.4	**34.3**	**44.4**	48.2	69.3	85.6	44.9	71.4
	ModernBERT	**41.6**	27.4	44.0	**51.3**	**80.2**	**88.4**	**56.4**	**73.6**
Large	BERT	38.9	23.3	31.7	49.5	28.5	85.2	41.6	60.8
	RoBERTa	41.4	22.6	36.1	49.8	28.8	88.9	47.3	68.1
	DeBERTaV3	25.6	7.1	19.2	46.7	23.0	**91.4**	21.2	19.7
	GTE-en-MLM	42.5	**36.4**	**48.9**	50.7	71.3	87.6	40.5	66.9
	ModernBERT	**44.0**	34.3	48.6	**52.4**	**80.4**	90.4	**59.5**	**83.9**

➔ 더 빠르고, 더 길게 볼 수 있으면서 더 좋은 모델

- **전반적인 성능 개선**
  - ModernBERT가 base 및 large 크기 모두에서 가장 높은 성능 달성
- **Information Retrieval 성능**
  - Single-vector(DPR)와 Multi-vector(ColBERT) 상관 없이 높은 성능 달성
- **Natural Language Understanding 성능**
  - DeBERTaV3 출시 이후 처음으로 이 성능을 능가
- **Code Tasks**
  - 코드 특화 토크나이저 + 데이터로 매우 높은 성능

◆ Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference

Memory and Inference efficiency

	Model	Params	Short			Long		
			BS	Fixed	Variable	BS	Fixed	Variable
Base	BERT	110M	1096	**180.4**	90.2	–	–	–
	RoBERTa	125M	664	179.9	89.9	–	–	–
	DeBERTaV3	183M	236	70.2	35.1	–	–	–
	NomicBERT	137M	588	117.1	58.5	36	46.1	23.1
	GTE-en-MLM	137M	640	123.7	61.8	38	46.8	23.4
	GTE-en-MLM$_{xformers}$	137M	640	122.5	128.6	38	47.5	67.3
	ModernBERT	149M	**1604**	148.1	**147.3**	**98**	**123.7**	**133.8**
Large	BERT	330M	**792**	**54.4**	27.2	–	–	–
	RoBERTa	355M	460	42.0	21.0	–	–	–
	DeBERTaV3	434M	134	24.6	12.3	–	–	–
	GTE-en-MLM	435M	472	38.7	19.3	28	16.2	8.1
	GTE-en-MLM$_{xformers}$	435M	472	38.5	40.4	28	16.5	22.8
	ModernBERT	395M	770	52.3	**52.9**	**48**	**46.8**	**49.8**

[NVIDIA RTX 4090 GPU에서 효율성 테스트 수행]
BS (Batch Size), Short (512 context), Long (8192 context),
Fixed (고정 길이에서 초당 처리 토큰 수, 천 단위)
Variable (가변 길이에서 초당 처리 토큰 수, 천 단위)

- **BS**: Base에서 1604, Large에서 770의 배치 크기를 기록
  ➔ 다른 모든 모델보다 훨씬 높은 메모리 효율성을 보여줌
- **Short_Fixed**: 기존 BERT와 RoBERTa 모델보다는 약간 느리지만, 다른 최신 인코더에 비해서는 더 빠르거나 유사한 성능
- **Short_Variable**: 다른 모든 경쟁 모델보다 훨씬 뛰어난 효율성을 보여줍니다. 특히, 최대 약 30.9% 더 많은 토큰을 처리
- **Long_Fixed**: 다른 모든 경쟁 인코더보다 2.65배(Base)에서 3배(Large) 더 빠른 속도
- **Long_Variable**: 최대 약 118.8% 더 많은 토큰을 처리

감사합니다.