

# BERT and GPT 1, 2, 3

팀 레모네이드

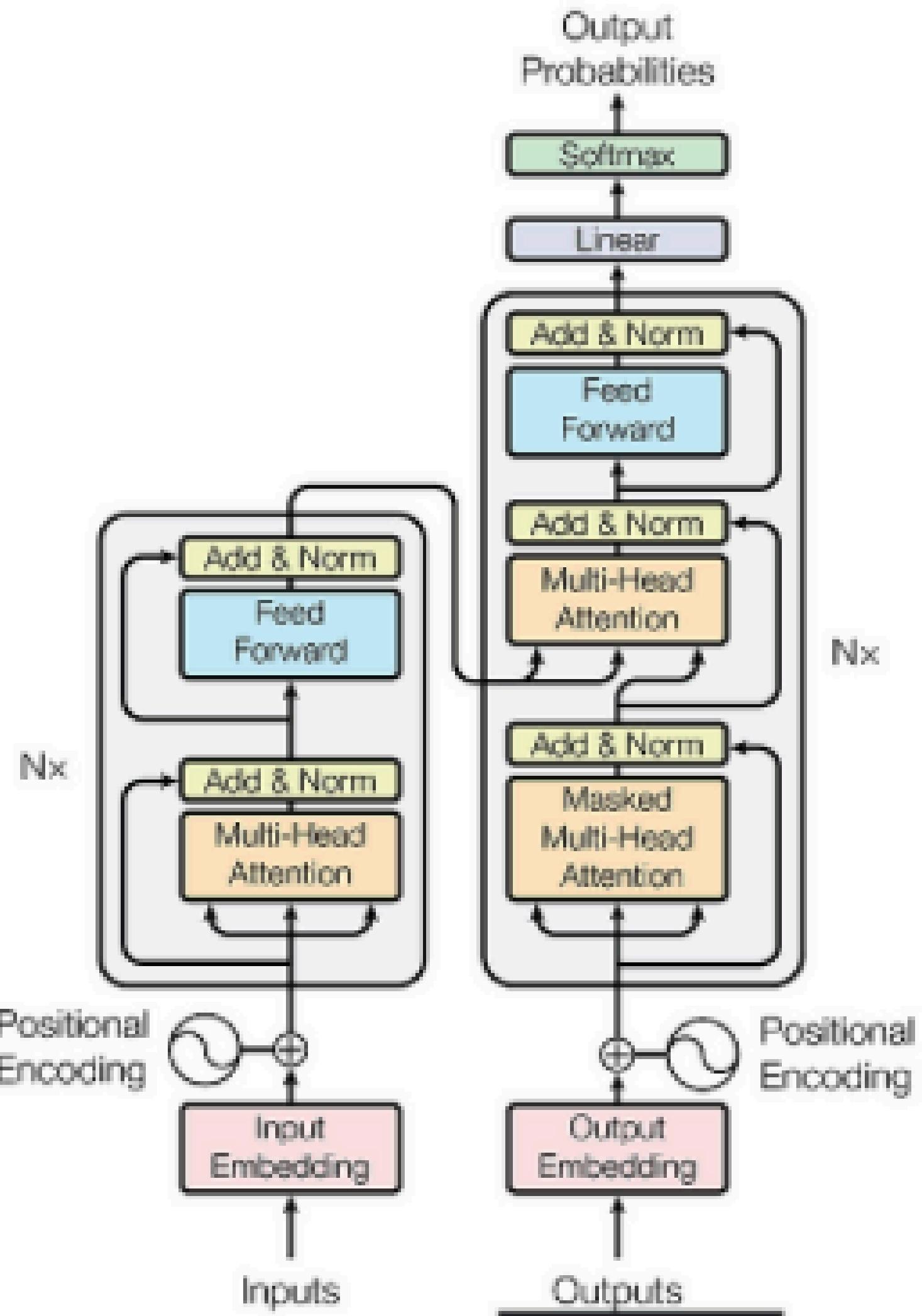
---

장재우 | 강희준 박은수 장우진 김선준

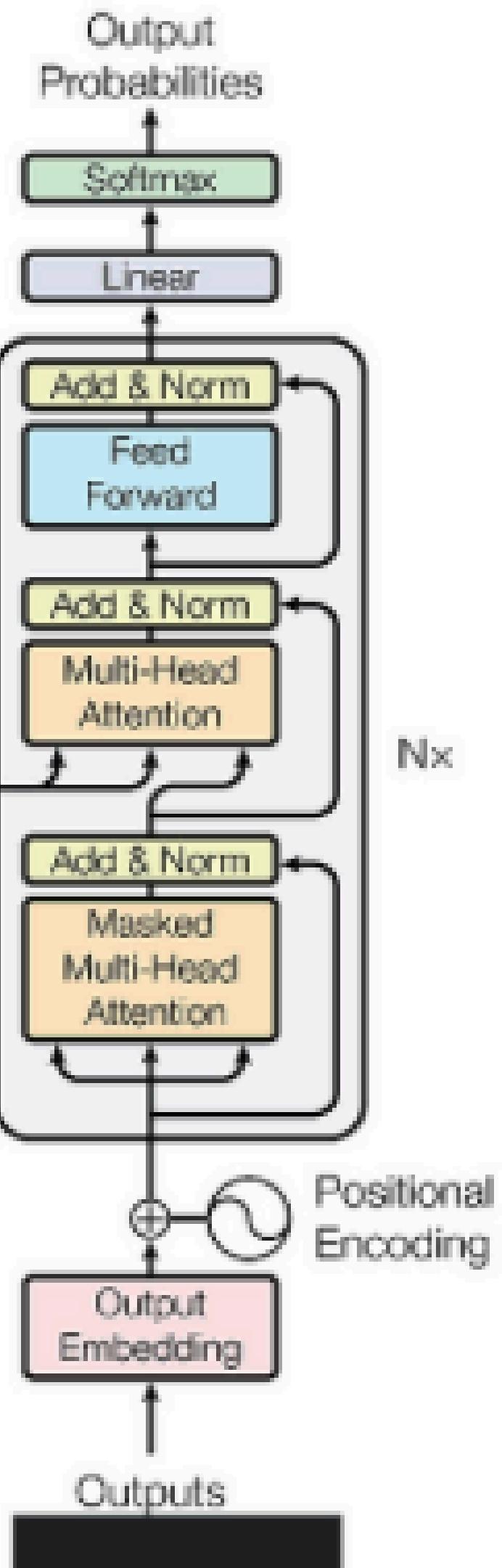
# Contents

- 01** \_\_\_\_\_ 배경
- 02** \_\_\_\_\_ BERT
- 03** \_\_\_\_\_ GPT 1,2,3

# BERT Encoder

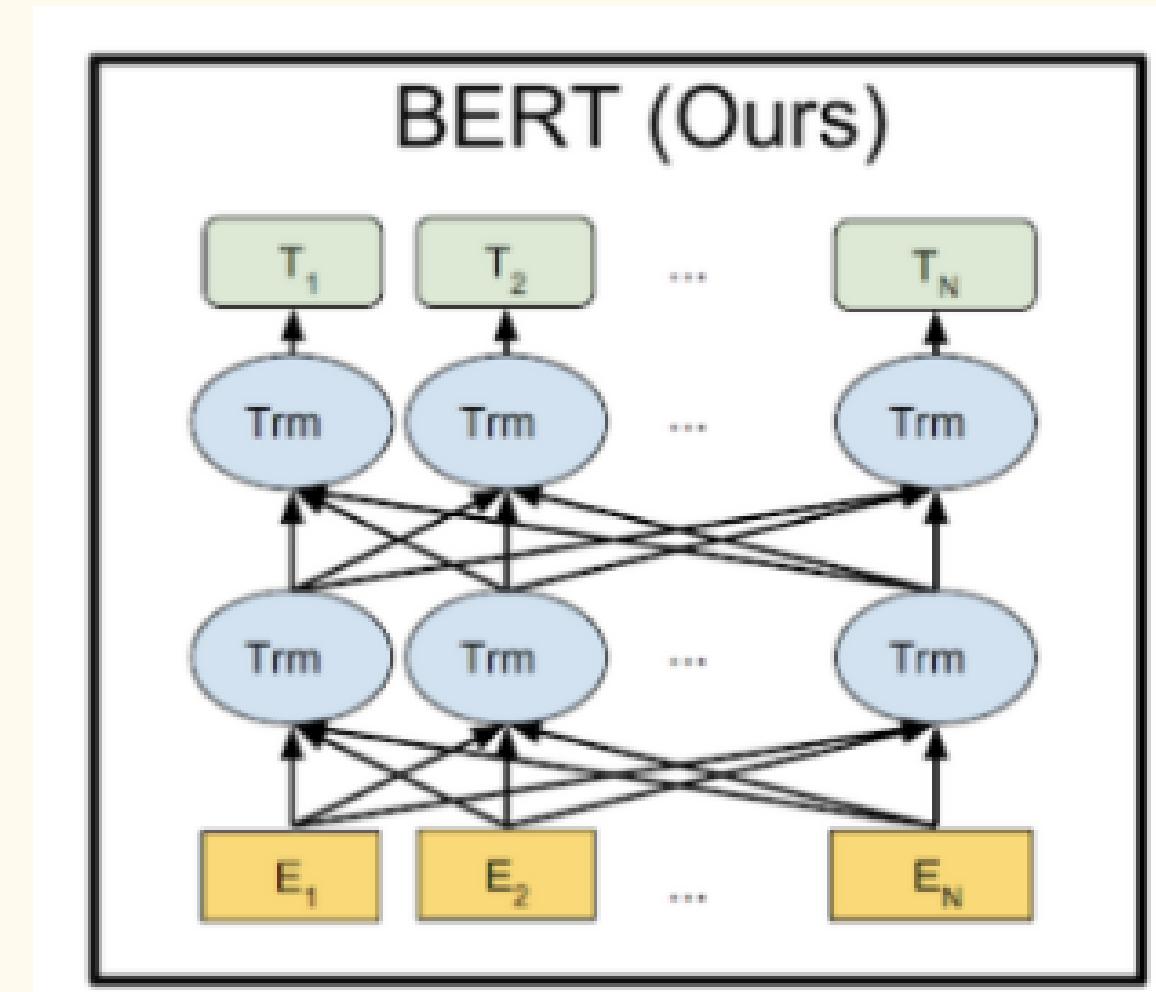


# GPT Decoder



## BERT 소개

- Bidirectional Encoder Representations from Transformers
- 기존 모델과 달리 \*\*양방향 문맥(좌·우 모두)\*\*을 동시에 학습
- 라벨 없는 텍스트로 사전학습(pre-training) → 다양한 태스크에 파인튜닝만으로 적용 가능
- 태스크별 복잡한 아키텍처 없이, 출력층만 추가하면 다양한 문제 해결
- 기반: Transformer Encoder (Vaswani et al., 2017)
- BERTBASE: L=12, H=768, A=12 → 110M params
- BERTLARGE: L=24, H=1024, A=16 → 340M params





## 데이터 준비

- Wikipedia + BookCorpus  
(대규모 무라벨 텍스트)
- WordPiece 토크나이저  
( $\approx 30k$  vocab)
- 특수 토큰: [CLS], [SEP], [PAD]



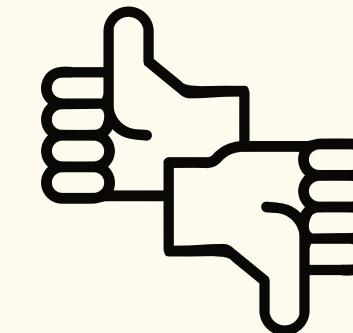
## 사전학습

- MLM (Masked Language Model)  $\rightarrow$  15% 토큰 가려놓고 원래 단어 예측
- NSP (Next Sentence Prediction)  $\rightarrow$  문장 A 다음에 문장 B가 실제 이어지는지 판별



## 인코더 구조

- 12층(BERT Base) / 24층(BERT Large)
- 멀티헤드 Self-Attention + Feed Forward + Residual + LayerNorm
- 양방향 문맥 학습 (좌→우, 우→좌 동시에 활용)



## 파인튜닝

- 문장 분류  $\rightarrow$  [CLS] 토큰 사용
- 토큰 단위 태깅 (NER, QA)  $\rightarrow$  각 토큰 벡터 활용
- QA  $\rightarrow$  시작/끝 위치 예측
- 태스크별 작은 출력 레이어만 붙이고, 전체 파라미터 end-to-end 학습

# 기존 접근의 한계와 BERT의 해법

## 기존

### 1. 기존 접근법

- Feature-based (ELMo): 사전학습 표현을 특징으로 사용
- Fine-tuning (GPT-1): 전체 파라미터를 태스크별 미세조정

### 2. 한계점 (Limitations)

- 대부분 단방향 모델 (Unidirectional)
  - GPT: Left-to-Right (앞만 본다)
  - ELMo: 양방향이지만 얇은 결합
- 문제:
  - 문장 수준 → 비효율적
  - 토큰 수준 (QA 등) → 좌·우 문맥이 필요해서 치명적

## BERT

### BERT의 해결책 (Solution)

- MLM (Masked LM): 일부 토큰을 가려놓고 좌·우 문맥을 동시에 학습
- NSP (Next Sentence Prediction): 문장쌍 관계까지 학습

### Contributions

- Bidirectional pre-training으로 표현력 강화
- 태스크별 복잡한 아키텍처 불필요 → 범용 적용 가능
- 11개 NLP 태스크에서 SOTA 달성 (문장 단위 & 토큰 단위 모두 outperform)

## BERT가 필요했던 이유

### Background: Word Embeddings

- 오래 전부터 단어 표현 학습은 NLP 핵심 과제.
- 사전학습된 임베딩은 scratch 학습보다 성능 크게 향상.
- 주로 Left-to-Right LM 또는 단어 예측/판별 목표 사용.

### Beyond Words → Sentences/Paragraphs

- Sentence/Paragraph Embeddings 연구로 확장.
- - Skip-thought vectors (Kiros, 2015): 이전 문장으로 다음 문장 생성
  - Next sentence ranking (Logeswaran & Lee, 2018)
  - Denoising autoencoder 방식 (Hill, 2016)

### ELMo: Contextual Embeddings

- ELMo (Peters et al., 2018): 좌→우 + 우→좌 LM을 따로 학습 → 결합해서 문맥 반영.
- 기존 태스크별 아키텍처에 feature로 추가하는 방식.
- QA, 감정 분석, NER 등 여러 태스크에서 SOTA 성능.

### Limitations

- Feature-based: 사전학습 표현을 단순히 가져다 쓰는 구조.
- 완전한 양방향(bidirectional)이 아님 → 좌·우 문맥을 얇게만 결합.

## Unsupervised Fine-Tuning Approaches

이전

- Collobert & Weston (2008): 단어 임베딩만 사전학습.

최근

- Sentence/Document Encoder를 사전학습 → 이후 다운스트림 태스크에 파인튜닝.
- 장점: 새로운 태스크에서 적은 파라미터만 학습하면 됨.
- 대표 연구:
  - Dai & Le (2015): 언슈퍼바이즈드 pre-training → supervised fine-tuning.
  - Howard & Ruder (2018, ULMFiT): LM 파인튜닝 프레임 워크.
  - Radford et al. (2018, GPT-1): Left-to-right Transformer LM → GLUE에서 SOTA 달성.

# Embedding

## 1. WordPiece 토크나이저

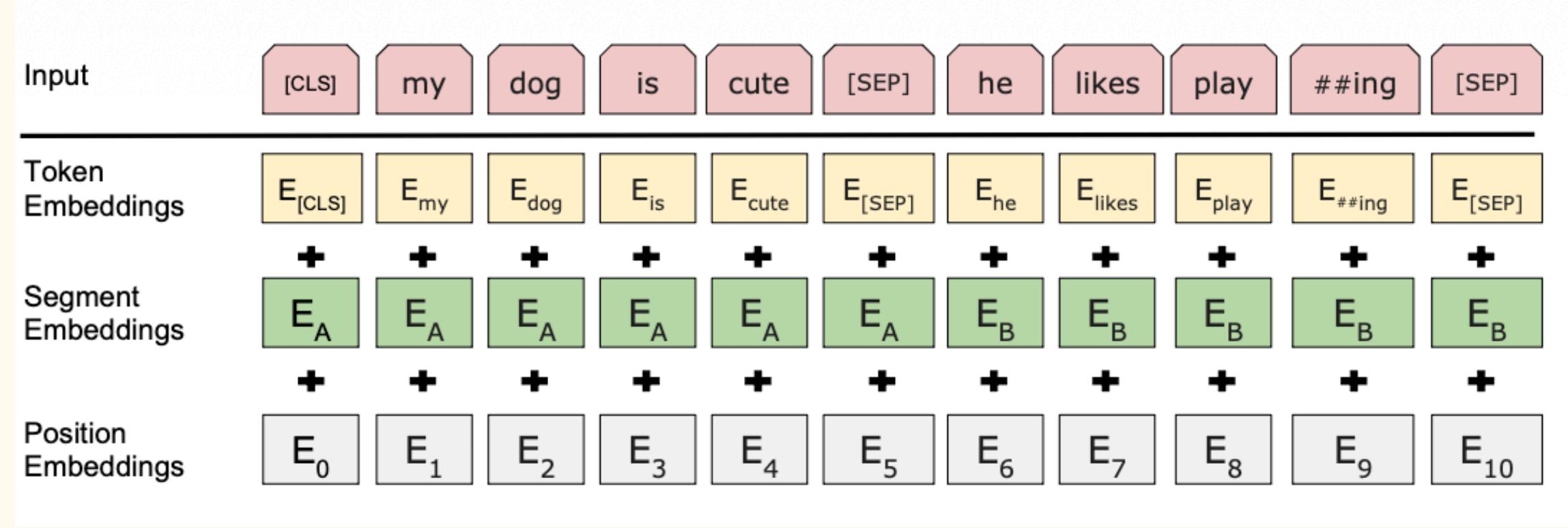
- BERT는 WordPiece 방식으로 단어를 서브워드 단위까지 분해.
- Vocabulary 크기  $\approx 30,000 \rightarrow$  영어 기준.
- 장점:
  - 신조어나 희귀어도 쪼개서 표현 가능.
  - OOV(Out-Of-Vocabulary) 문제 해결.

예시:

- “unbelievable”  $\rightarrow$  [un] [##believ] [##able]
- “playing”  $\rightarrow$  [play] [##ing]

## 2. 입력 시퀀스 구조

- 입력은 하나의 문장 또는 문장쌍.
- 문장쌍 예시:
  - (질문, 답변), (전제, 가설), (문장1, 문장2)
- 두 문장을 하나의 시퀀스로 연결(concatenate) 해서 입력



## 3. 특수 토큰 (Special Tokens)

- [CLS]
  - 항상 맨 앞에 붙음.
  - 최종 hidden state가 \*\*문장 단위 태스크(분류 등)\*\*에서 대표 벡터로 활용됨.
  - 예: 감정 분류  $\rightarrow$  [CLS] 벡터로 긍정/부정 판별.
- [SEP]
  - 문장과 문장 사이, 혹은 문장의 끝을 구분하는 경계 토큰.
  - 문장쌍 태스크에서 Sentence A, Sentence B 구분에 필수.
- [PAD]
  - 문장 길이를 맞추기 위해 사용. (짧은 문장은 [PAD]로 채움)

## Token Embedding

토큰: [CLS] un ##believ ##able movie ! [SEP]

ID: 101 432 9876 2451 3185 106 102

임베딩:  $e_{\text{CLS}} \ e_{\text{un}} \ e_{\text{##believ}} \ e_{\text{##able}} \ e_{\text{movie}} \ e_{\text{!}} \ e_{\text{SEP}} \in \mathbb{R}^H$

### 개념

입력 문장을 WordPiece로 쪼개 각 토큰을 의미 벡터로 바꾼 것.

BERT는 약 30,000개(vocab) 토큰을 갖고, 각 토큰마다 학습 가능한 벡터를 가짐.

임베딩 테이블  $E_{\text{token}} \in \mathbb{R}^{V \times H}$  ( $V \approx 30k$ ,  $H=768/\text{Base}$ )

토큰 ID로 테이블을 lookup하여  $e_{\text{token}} \in \mathbb{R}^H$  벡터 획득.

### 왜 필요한가

모델이 텍스트를 수치로 이해하려면 단어(또는 서브워드)의 의미가 벡터화되어야 함.

WordPiece 덕분에 OOV(사전 밖 단어) 문제를 최소화(신조어/희귀어도 분해해서 표현).

### 어떻게 동작

#### 텍스트 정규화

WordPiece 토크나이징 → 토큰 ID 시퀀스 생성

각 ID로  $E_{\text{token}}$ 에서 임베딩 벡터를 검색

## Segment Embedding

토큰: [CLS] 그는 갔다 . [SEP] 그는 왜 갔나 ? [SEP]

세그먼트: 0 0 0 0 0 1 1 1 1

### 개념

BERT는 문장 쌍을 한 시퀀스로 합쳐서 넣음:

[CLS] Sentence A [SEP] Sentence B [SEP]

각 토큰에 세그먼트 ID를 부여해 “이 토큰이 A인지 B인지”를 알려줌.

보통 A=0, B=1 (임베딩 테이블에서 두 벡터를 학습)

### 왜 필요한가

Self-Attention은 모든 토큰이 서로를 볼 수 있음 →

“서로 다른 문장” 경계를 알면 문장 간 상호작용을 더 안정적으로 학습.

### 어떻게 동작

최종 입력 벡터를 만들 때, 각 토큰에 Segment Embedding을 더함.

단일 문장 태스크는 전부 A(0)로 채움.

.

# Position Embedding

토큰: [CLS] 그는 갔다 . [SEP] 그는 왜 갔나 ? [SEP]  
포지션: 0 1 2 3 4 5 6 7 8 9

## 개념

Transformer는 순서를 직접 모르기 때문에, 각 토큰에 위치 임베딩을 더해 순서를 알려줌.  
BERT는 학습된(learned) 절대 위치 임베딩 사용. (기본 최대 길이 512)

## 왜 필요한가

“나는 오늘 학교에 갔다” vs “학교에 나는 오늘 갔다”처럼 순서가 의미에 영향 →  
위치 정보 없이는 Self-Attention이 순서 구분 불가.

## 어떻게 동작

시퀀스 앞에서부터 0,1,2,⋯로 위치 번호를 매김.  
문장 B로 넘어가도 위치는 리셋하지 않고 계속 증가.  
[PAD]도 위치 번호는 갖지만 attention\_mask=0으로 무시됨.

## Final

### 개념

BERT에 들어가는 실제 입력 벡터는 세 가지 임베딩(Token + Segment + Position)의 합.  
각 토큰마다 고유하게 계산 → 이후 Transformer Encoder로 전달됨.

토큰: [CLS] 그는 집에 갔다 . [SEP] 그는 왜 갔나 ? [SEP]

Token ID: 101 1030 2090 3111 119 102 1030 777 3112 136 102

Segment: 0 0 0 0 0 0 1 1 1 1 1

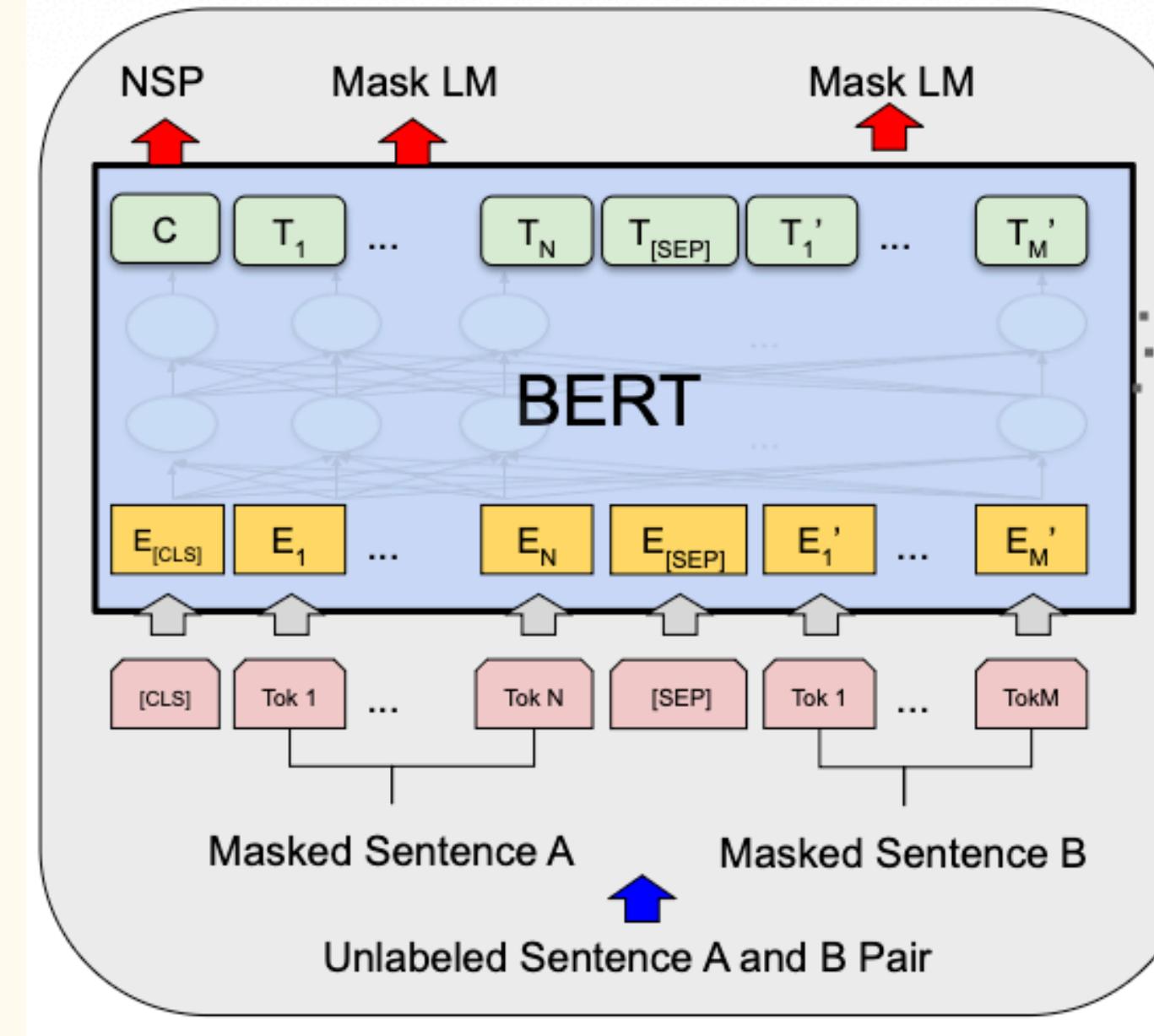
Position: 0 1 2 3 4 5 6 7 8 9 10

## Pre-training BERT

### 1 Masked Language Model (MLM)

- 입력 토큰 중 15% 무작위 선택 → 예측 대상으로 설정.
- 선택된 토큰 처리:
  - 80%: [MASK]로 교체
  - 10%: 랜덤 단어로 교체
  - 10%: 원래 단어 유지
- 출력: 마스킹된 자리의 hidden state → Softmax → 정답 단어 예측.
- 특징: 양방향 문맥 활용 가능, 기존 LM보다 강력.
- 문제점: [MASK]는 실제 파인튜닝 입력에 안 쓰임 → 이 때문에 랜덤·원래 단어 섞어서 mismatch 완화.
- 문장 쌍 관계 학습 (QA, NLI 대비).
- 데이터 생성:
  - 50%: 실제 다음 문장 (IsNext)
  - 50%: 랜덤 문장 (NotNext)
- 입력: [CLS] 벡터 → Binary Classifier로 NSP 예측.
- 효과: 문장 관계 이해 성능 크게 향상 (QA/NLI).
- BERT는 NSP accuracy 97~98% 달성.

### 2 Next Sentence Prediction (NSP)

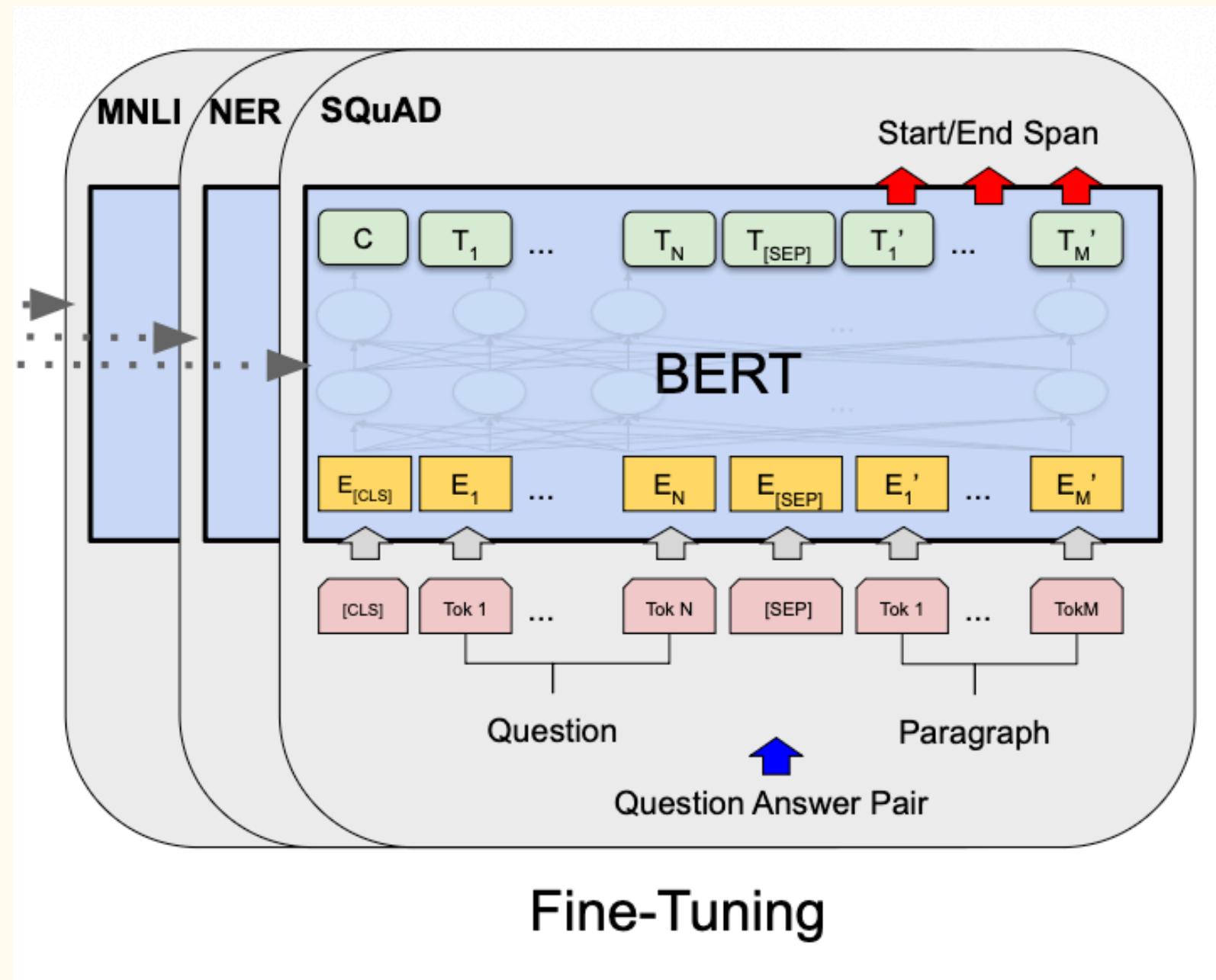


Pre-training

# Fine-tuning BERT

How?

- 입력
  - 사전학습 시 Sentence A, B → 태스크별 입력쌍으로 매핑.
  - 예시:
    - i. Paraphrasing → (문장1, 문장2)
    - ii. NLI → (Premise, Hypothesis)
    - iii. QA → (Question, Passage)
    - iv. Classification/NER → (문장, Ø)
- 출력
  - Token-level 태스크 (QA, NER): 각 토큰 벡터 사용
  - Sentence-level 태스크 (분류, NLI, 감정분석): [CLS] 벡터 사용



Efficiency

- Fine-tuning은 비용이 적음 (사전학습이 가장 무거움).
- 논문 기준: Cloud TPU로 1시간 이내, GPU로도 몇 시간 내 재현 가능.
- 모든 태스크는 동일한 pre-trained BERT에서 출발 → 태스크별 헤드만 다름.

## 1. 문장 분류 (Sentiment Analysis, NLI 등)

입력: [CLS] 문장 [SEP]

출력: [CLS] 최종 hidden state → Linear Layer → Softmax

예: 긍정/부정, entail/neutral/contradict 분류

## 2. 개체명 인식 (NER)

입력: 문장

출력: 각 토큰 hidden state  $T_i$  → Linear Layer → BIO 태그

예측

예: “홍길동 → B-PER, 서울 → B-LOC”

## 3. 문장 유사도 / STS

입력: [CLS] 문장A [SEP] 문장B [SEP]

출력: [CLS] hidden state → Linear → 점수 (0~5 유사도 등)

## 4. 질의응답 (SQuAD, QA)

입력: [CLS] 질문 [SEP] 본문 [SEP]

출력: 본문 토큰 hidden state  $T_i$  → Linear Layer(2) → Start/End 위치 예측

예: “조선의 수도는 어디인가?” → “서울”

# GPT 시리즈 발전사: 언어모델의 혁신적 진화



GPT-1 (2018)

"Pretrain → Finetune"의 표준 절차를 확립  
두 단계 학습으로 GLUE 등 전이학습의 기반  
을 마련



GPT-2 (2019)

규모 확대 + 제로샷 성능을 본격적으로 입증  
프롬프트만으로 다양한 과제 수행 가능성을  
보여줌



GPT-3 (2020)

초대규모 파라미터로 Few-shot in-context  
learning을 정립  
미세조정 없이도 예시만으로 고성능 달성

- **공통 철학:** 대규모 말뭉치로 언어모델(다음 단어 맞히기)을 먼저 학습 → 작은 라벨 데이터로 과제 적용

# GPT-1: 전이학습의 표준화

01

...

## 핵심 제안사항

- **두 단계 학습:** ① 비지도 사전학습(LM) → ② 감독 미세조정(라벨 과제)
- **한 모델로 다수 과제:** 입력을 토큰열로 통일(Start/Delim/Extract 포맷)해 분류  
·NLI·유사도·MCQ 처리

## 아키텍처 & 수식

**모델:** 멀티레이어 Transformer Decoder(Masked Self-Attention) × n

**초기 표현:**  $h^0 = U \cdot W_e + W_p$  (단어임베딩 + 위치임베딩)

**사전학습 손실:**  $L_1 = \sum_i \log P(u_i | u_{i-k} \dots u_{i-1}; \theta) \rightarrow$  "앞의 k단어로 다음 단어 맞히기"

**미세조정:** 최종 토큰 벡터  $h_m$ 을 뽑아 Linear+Softmax로 라벨 예측

$$P(y|x_1 \dots x_m) = \text{softmax}(h_m W_\gamma)$$

**보조손실:**  $L_3 = L_2 + \lambda \cdot L_1(C)$  (언어감각 유지·수렴 가속)



## 평가셋 & 지표

- GLUE(SST-2 Acc, CoLA MCC, MRPC/QQP F1, STS-B 상관계수, QNLI/MNLI/RTE Acc)
- RACE/StoryCloze Acc
- SciTail Acc 등

# 핵심 요약 (Executive Summary)

## GPT-2: 제로샷의 가능성 입증

### 핵심 성과

규모 확대( $117M \rightarrow 1.5B$ )만으로 제로샷  
성능이 폭발적으로 증가함을 실증

프롬프트 엔지니어링으로 미세조정 없이  
요약·번역·QA 등 다과제 수행

### 핵심 구성

Pure Decoder Transformer(GPT-1과  
동일 계열)

대규모 웹코퍼스 + 더 긴 시퀀스,  
LayerNorm/Dropout/초기화 개선

### 평가 방식

제로샷 위주: LAMBADA(Perplexity +  
Accuracy), Winograd(Accuracy),  
CoQA(F1)

요약(CNN/DM): ROUGE-1/2/L, 번역  
(WMT): BLEU, 오픈QA: Exact  
Match(EM) 등

# GPT-3: Few-shot Learning의 혁신

02

## 혁신적 변화

### 175B 파라미터의 초거대 모델로 Few-shot In-Context Learning 확립

미세조정 없이 프롬프트 안에 예시(k-shot)를 넣기만 해도 SOTA급

#### 작동 원리(인컨텍스트 러닝)

프롬프트 = "문제 설명 + 예시(K개) + 질문"

모델은 프롬프트 안의 패턴을 학습한 것처럼 따라 한다(내부 가중치 업데이트 없음)

## 평가셋 & 지표

- HellaSwag/StoryCloze/ARC: Accuracy
- NQ/WebQ/TriviaQA: Acc/EM
- CoQA/DROP: F1
- PTB/LAMBADA: Perplexity/Accuracy
- SuperGLUE: 표준 종합점(Acc/F1/MCC 등 소과제 집계)

## Few-shot 프롬프트 예(분류)

Task: Is the review positive(1) or negative(0)?

Example:

Text: "I absolutely loved it." → 1

Text: "It was a waste of time." → 0

Now classify:

Text: "Surprisingly enjoyable!" →

# 세 모델 비교 분석

항목	GPT-1	GPT-2	GPT-3
학습 전략	Pretrain → Finetune	대규모 Pretrain + Zero-shot	초대규모 Pretrain + Few-shot
주 메시지	전이학습 일반성 증명	스케일만 키워도 다과제 제로샷 가능	미세조정 없이 예시만으로 고성능
입력 포맷	Start/Delim/Extract, 후보별 스코어	지시문 프롬프트	지시문 + 예시(K-shot)
대표 지표	GLUE 집계(Acc/F1/MCC/상관)·RACE Acc	LAMBADA ppl/Acc, ROUGE/BLEU, EM	Accuracy/EM/F1, SuperGLUE 종합, ppl
강점	라벨 적은 과제에 강함	범용 제로샷, 데이터 효율	세팅 비용 ↓, 범용성 ↑
한계	과제마다 헤드/미세조정 필요	프롬프트 민감성, 지식 신선도	대규모 비용, 프롬프트 민감성

# 구현 핵심 포인트

1

## GPT-1 구현 체크리스트

- 토크나이저: BPE or WordPiece(논문은 BPE 계열) / 소문자화 일관성
- 포지션 임베딩: 고정 길이(논문은 학습형) · 시퀀스 길이 관리
- 학습: Adam, 학습률 워밍업, 드롭아웃, 레이어노름
- **자주 겪는 함정:** Extract 위치(마지막 토큰) 착오 → 성능 급락

2

## GPT-2 구현 체크리스트

- 스케일이 곧 성능: 폭·깊이·컨텍스트 길이 증가 → 성능 곡선 확인
- 프롬프트 템플릿: "문제지 스타일" 지시문, 예시 포맷 일관성
- 샘플링: top-k/p, 길이/반복 패널티 (문장 품질 좌우)
- **함정:** 프롬프트 포맷 불안정 → 분산 큰 결과

3

## GPT-3 구현 체크리스트

- 프롬프트 길이 관리: 예시 수(k) vs 컨텍스트 창
- 예시 품질: 경계 케이스 포함, 태그/형식 일관성
- 디코딩: greedy vs sampling 선택 (정확도 과제는 보통 greedy)
- **함정:** 예시 순서/표현 민감성(순서 고정, 포맷 엄격)

## 핵심 질문과 답변

Q. 왜 미세조정 대신 프롬프트만으로 가능한가?

A. 대규모 LM이 일반 패턴을 이미 내재화했고, 프롬프트가 조건부 규칙을 지정한다.

Q. SuperGLUE에서 어떤 지표?

A. 과제별 표준(Acc/F1/MCC 등)을 집계 점수로 보고한다.

Q. 가장 큰 재현 리스크?

A. 토크나이저·포맷·Extract 위치·디코딩 파라미터 불일치.

## 결론

GPT-1은 '사전학습→미세조정' 표준을 세웠고, GPT-2는 규모로 제로샷의 문을 열었으며, GPT-3는 초대규모에서 프롬프트 예시만으로도 학습 효과를 구현—즉, 규모·포맷·프롬프트가 현대 LLM의 성능을 결정한다.

# Reference

1. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," arXiv preprint arXiv:1810.04805, 2018.
2. A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving Language Understanding by Generative Pre-Training," OpenAI Technical Report, 2018.
3. A. Radford et al., "Language Models are Unsupervised Multitask Learners," OpenAI Technical Report, 2019.
4. T. B. Brown et al., "Language Models are Few-Shot Learners," arXiv:2005.14165, 2020.

**THANK YOU**

**감사합니다.**

