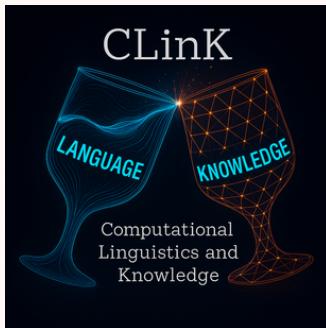
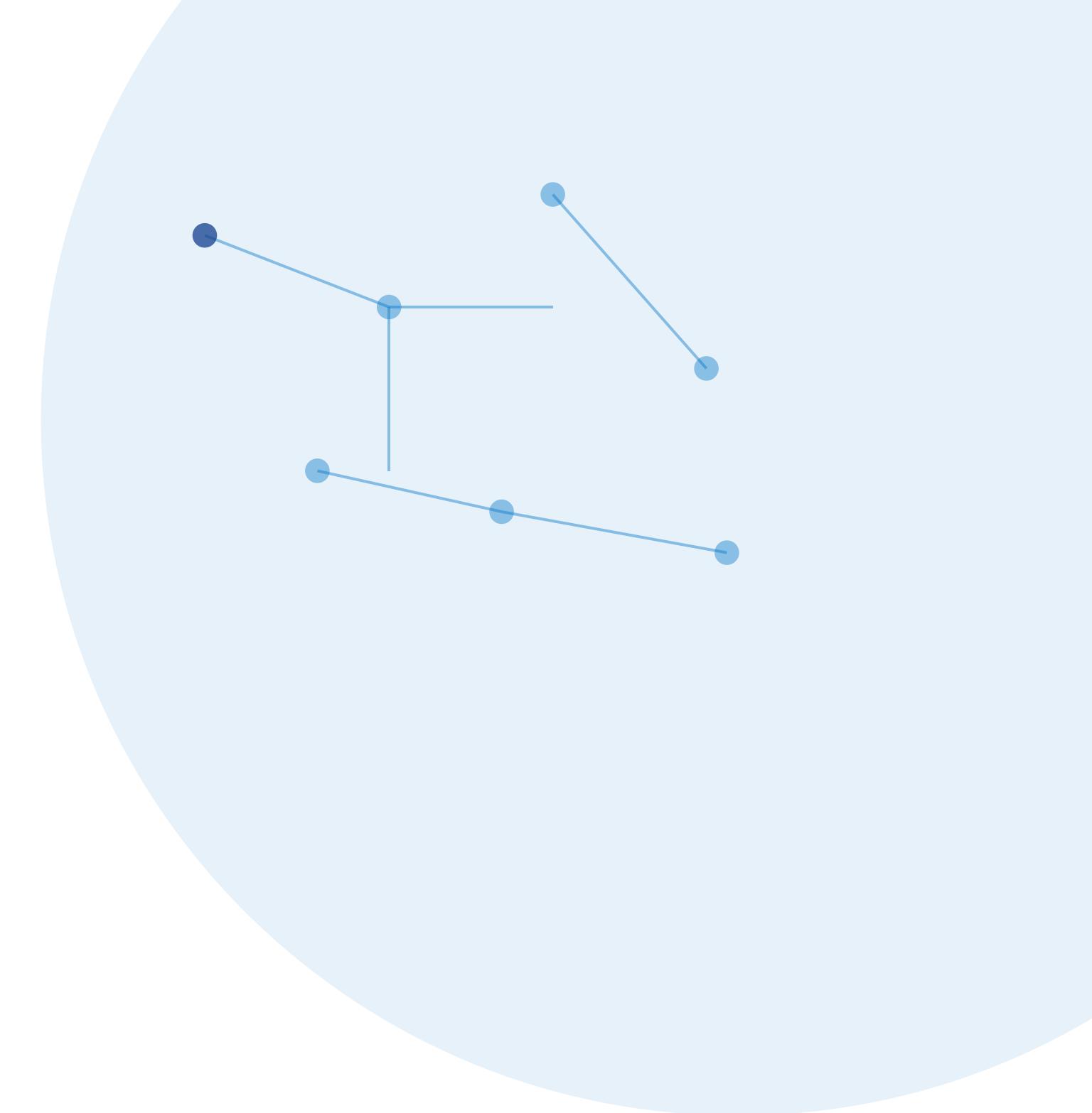


BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding



Contents

01

Introduction

02

Related work

03

Architecture

04

Experiments

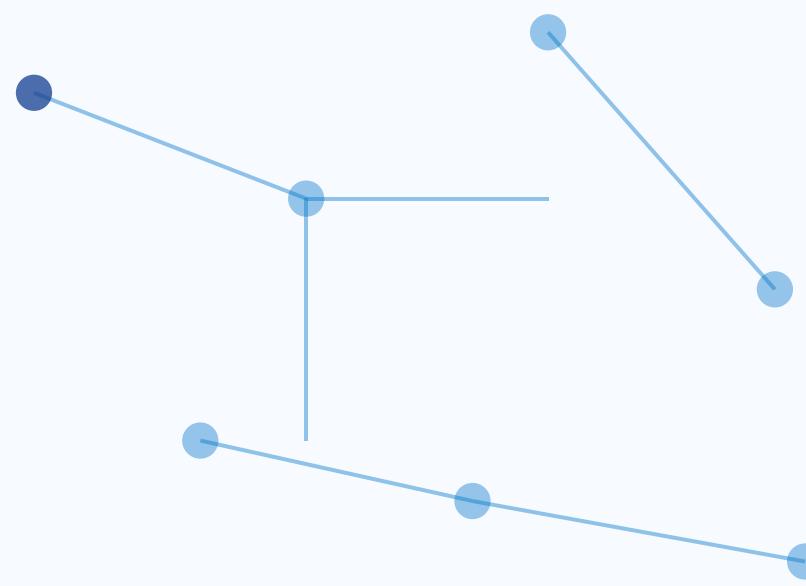
05

Ablation Studies

06

Conclusion

Introduction



| What is Bert

Bidirectional Encoder Representations from Transformers

→ transformer의 encoder 부분을 기반으로, 문장의 양방향 문맥을 고려하여 단어들의 의미로 벡터로 표현

| Past method

특성 기반(Feature-based): 사전 학습된 표현을 추가적인 특성으로 사용 / 사전 학습된 모델을 건드리지 않고, 결과값만 사용해서 내 모델 입력 값으로 사용 (예: ELMo)

파인 투닝(Fine_tuning): 사전 학습된 파라미터를 가져와 전체를 미세 조정 (예: OpenAI GPT)
→ 단방향 언어 모델

| Solution

transformer의 encoder 부분만 사용

Masked LM: 입력 토큰의 일부를 무작위로 masking하고 문맥을 통해 예측 → Deep Bidirectional 문맥 학습 가능

NSP: 텍스트 쌍의 관계를 파악하기 위해 다음 문장을 예측하는 task 수행

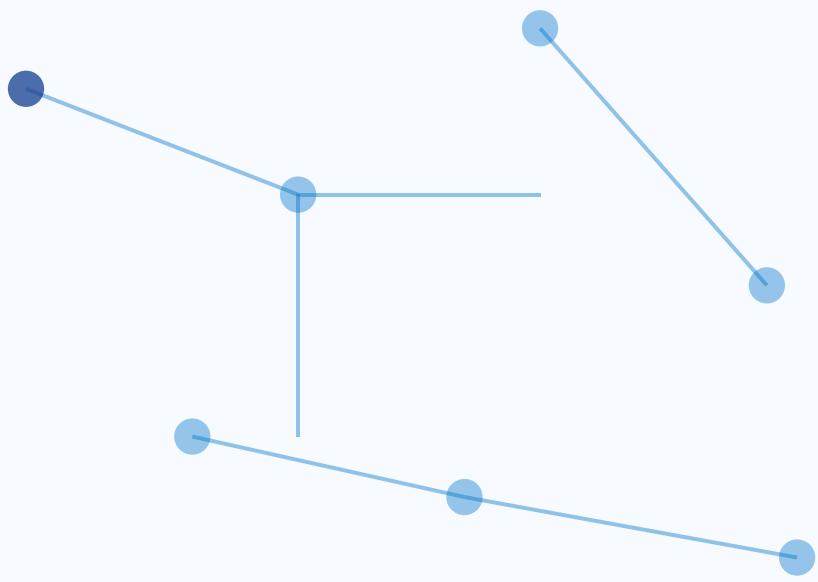
| What is the difference between ELMo and BERT?

| 비교 항목 (Criteria) | ELMo (Shallow Bidirectional) | BERT (Deep Bidirectional) |
|------------------|---|---|
| 1. 학습 방식 | 독립적 학습 후 결합 (Concatenation) | 동시 학습 (Jointly Conditioned) |
| | 왼쪽→오른쪽(Forward) 모델과 오른쪽→왼쪽(Backward) 모델을 따로 학습시킨 뒤 단순히 이어 붙임. | 모든 레이어에서 왼쪽과 오른쪽 문맥을 한 번에 동시에 참조하여 학습함. |
| 2. 문맥 파악 | 순차적 (Sequential) | 전역적 (Global) |
| | 왼쪽 문맥을 볼 때는 오른쪽을 못 보고, 오른쪽을 볼 때는 왼쪽을 못 봄. | Self-Attention을 통해 문장 내 모든 단어 간의 관계를 즉시 파악함. |

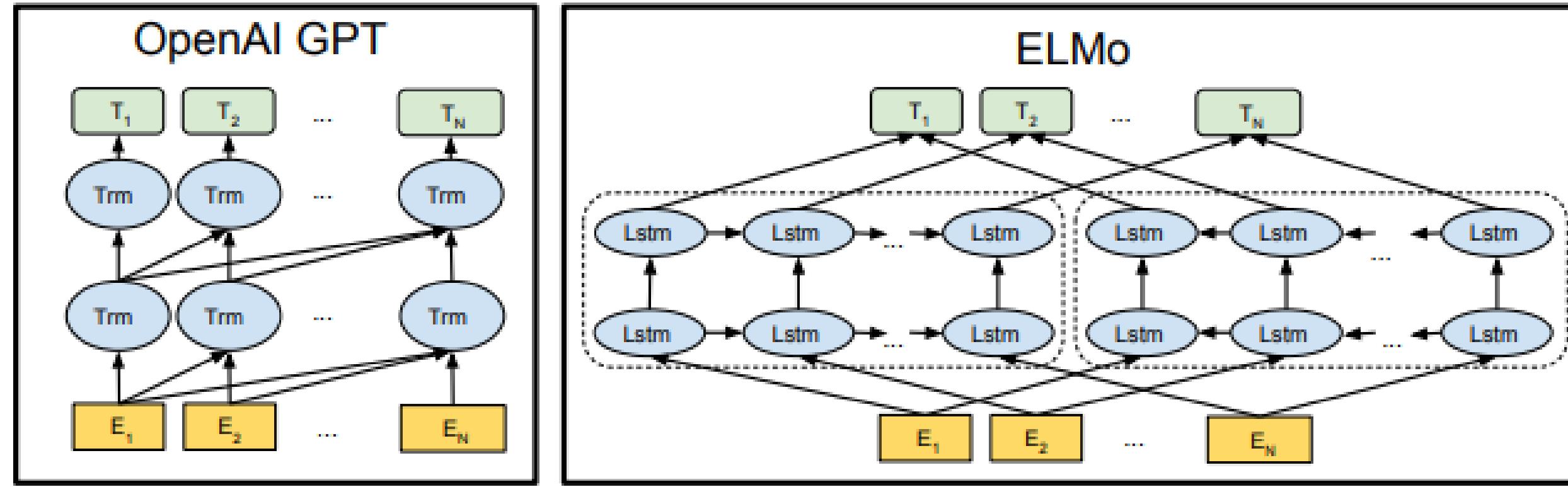
ELMo: 왼 → 오 LSTM, 오 → 왼 LSTM Concat 함.

Bert: Transformer의 Self-Attention을 활용 → 모든 레이어에서 왼쪽과 오른쪽 문맥을 Jointly Conditioned

Related work



Related work



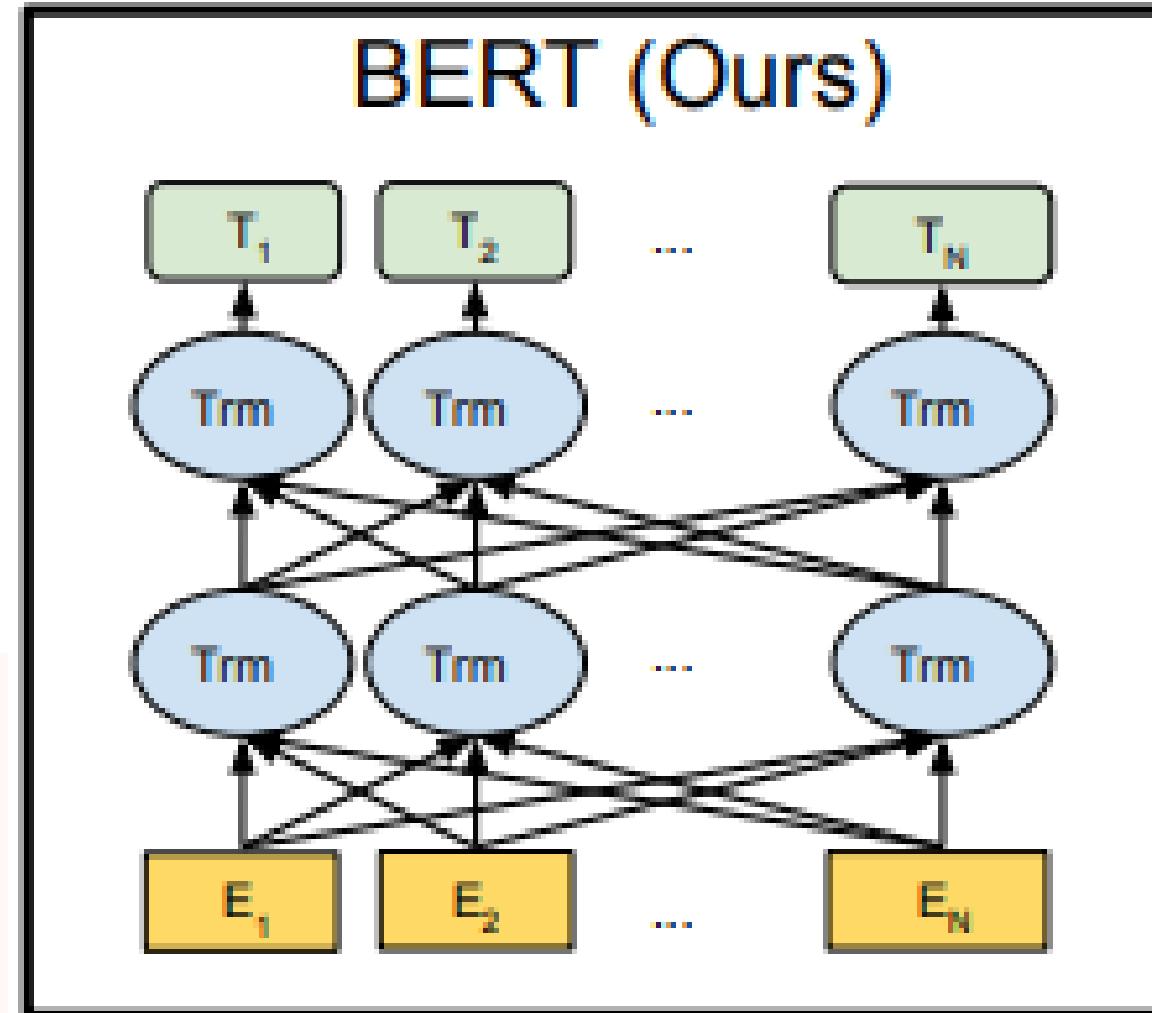
ELMo (Shallow Bidirectional):

- 정방향 LSTM과 역방향 LSTM을 따로 학습 후, Concat하여 word representation을 나타냄
- 한계: 동시에 문맥을 고려하는건 아님

OpenAI GPT (Unidirectional):

- Transformer의 Decoder 부분을 활용하여, 뒷 부분을 Masking 해서 Left-to-Right 한 방향으로만 학습하는 모델
- 한계: 단방향 구조이므로 현재 단어를 예측할 때 뒤에 나오는 문맥을 볼 수 없음

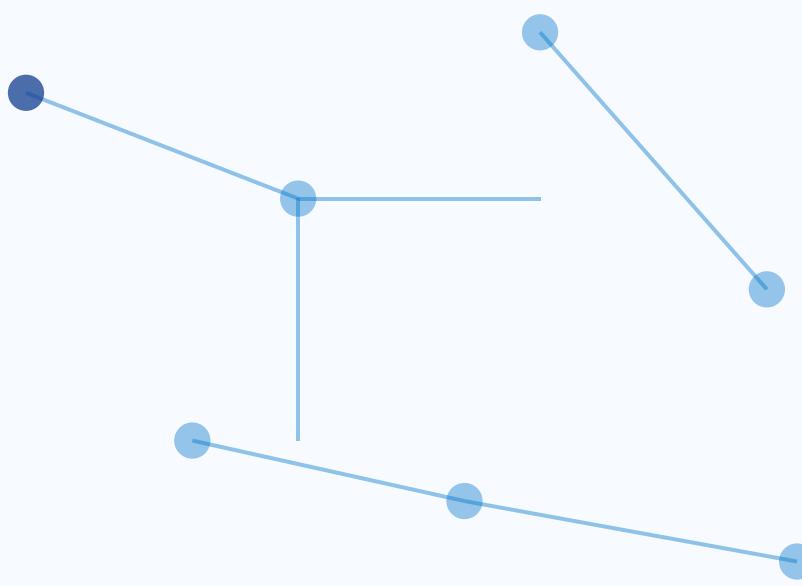
Related work



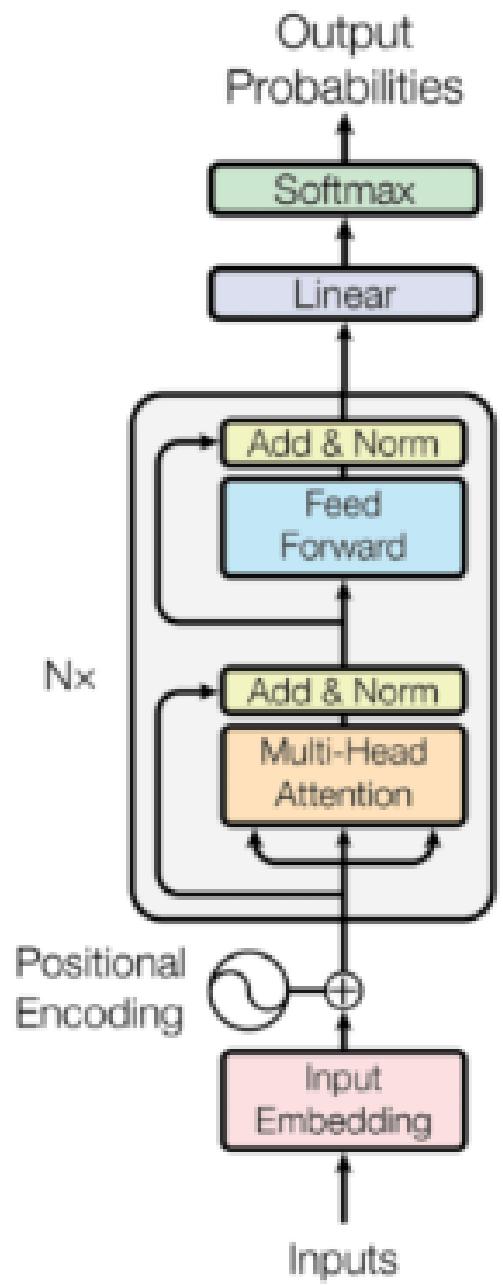
BERT (Deep Bidirectional):

- Transformer의 Encoder를 활용하여, 입력의 일부분(15%)을 Masking하고 Bidirection으로 문맥을 파악해 가려진 부분을 맞추도록 학습하는 모델.
- 강점: 모든 레이어에서 왼쪽과 오른쪽 문맥을 Jointly 깊게 고려함.

Architecture



Architecture



Transformer의 encoder 구조 활용

L: Number of Layers, H: Hidden Size, A: Number of Self-Attention Heads

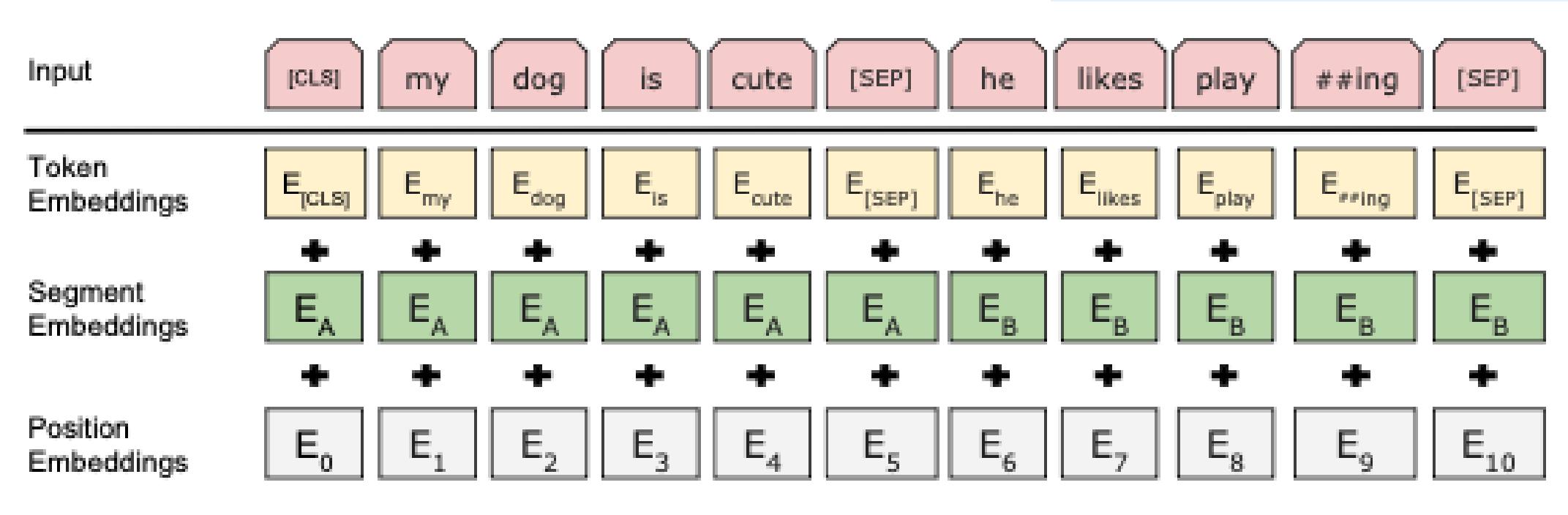
- BERT-Base: L=12, H=768, A=12 (Total Parameters: 110M)
- BERT-Large: L=24, H=1024, A=16 (Total Parameters: 340M)

BERT의 총 파라미터 수는 크게 임베딩(Embedding) 부분과 트랜스포머 레이어(Encoder Layers) 부분의 합입니다.

$$\text{Total Parameters} \approx \underbrace{(V \times H)}_{\text{Embedding}} + \underbrace{L \times (12 \times H^2)}_{\text{Transformer Layers}}$$

- **V (Vocabulary Size):** 단어 사전의 크기 (BERT는 약 30,522개)
- **L (Layers):** 레이어 수
- **H (Hidden Size):** 히든 벡터 크기
- **A (Attention Heads):** 파라미터 수 계산에는 직접 영향을 주지 않음 (내부적으로 쪼개질 뿐 총량은 같음)

Input/Output Representations



① Token Embeddings (단어 정보)

- WordPiece: 단어를 의미 단위로 쪼개어 처리
 - playing -> play (어근) + ##ing (접미사)
- [CLS]: 문장 시작 토큰 / [SEP]: 문장 구분 토큰

② Segment Embeddings (문장 구분)

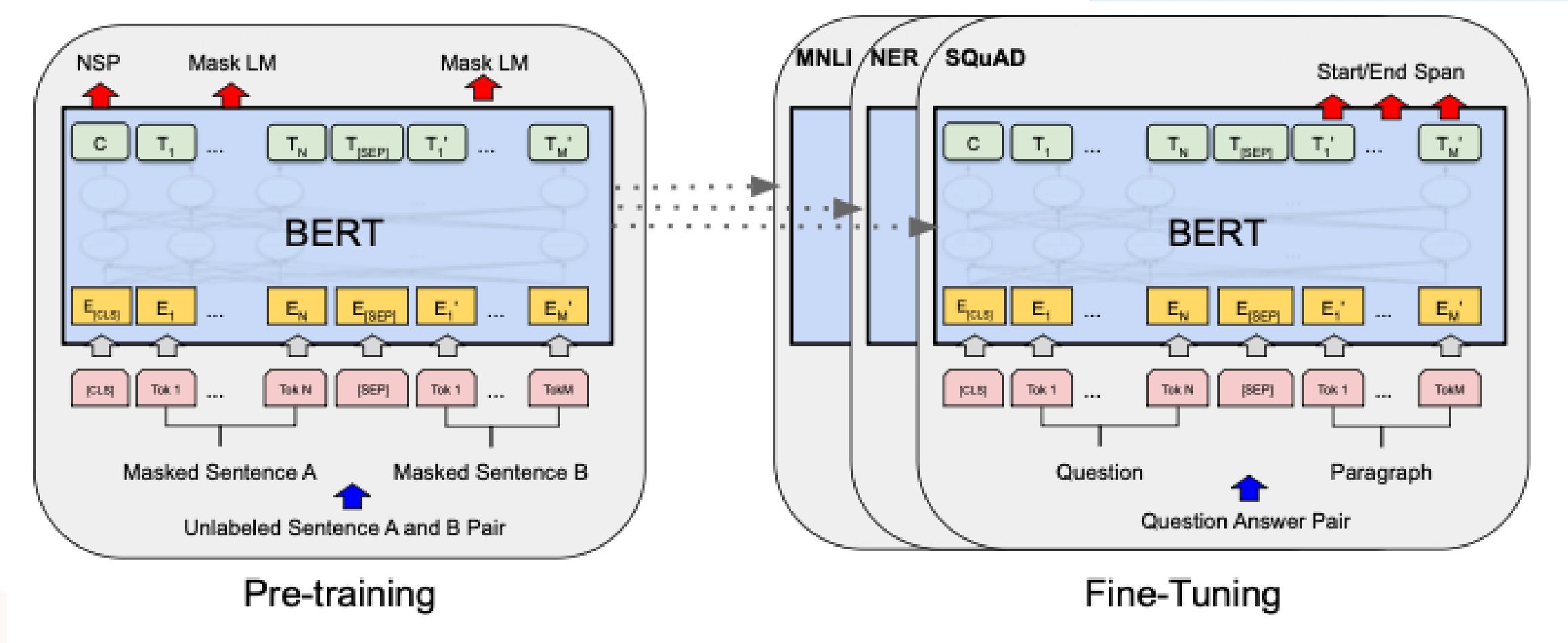
- 현재 처리 중인 토큰이 앞 문장인지 뒷 문장인지 구별

③ Position Embeddings (위치 정보)

- 순서 정보가 없는 Transformer에 단어의 위치 순서를 주입

$$\text{최종 입력} = \text{Token Emb} + \text{Segment Emb} + \text{Position Emb}$$

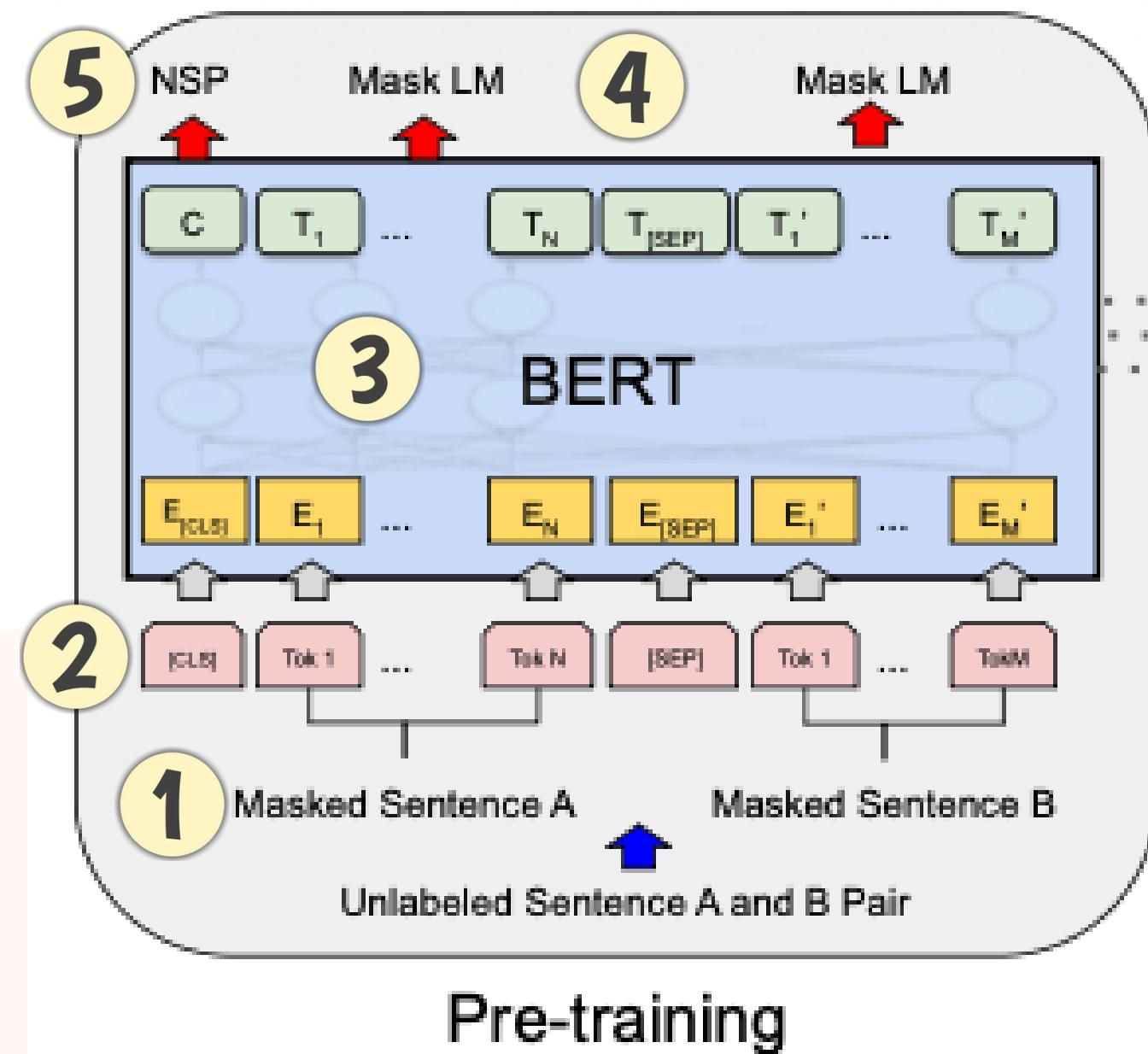
Pre-training, Fine-tuning



1단계 Pre-training
라벨이 없는 데이터로 MLM(Masked LM), NSP(Next Sentence Prediction)등을 수행함

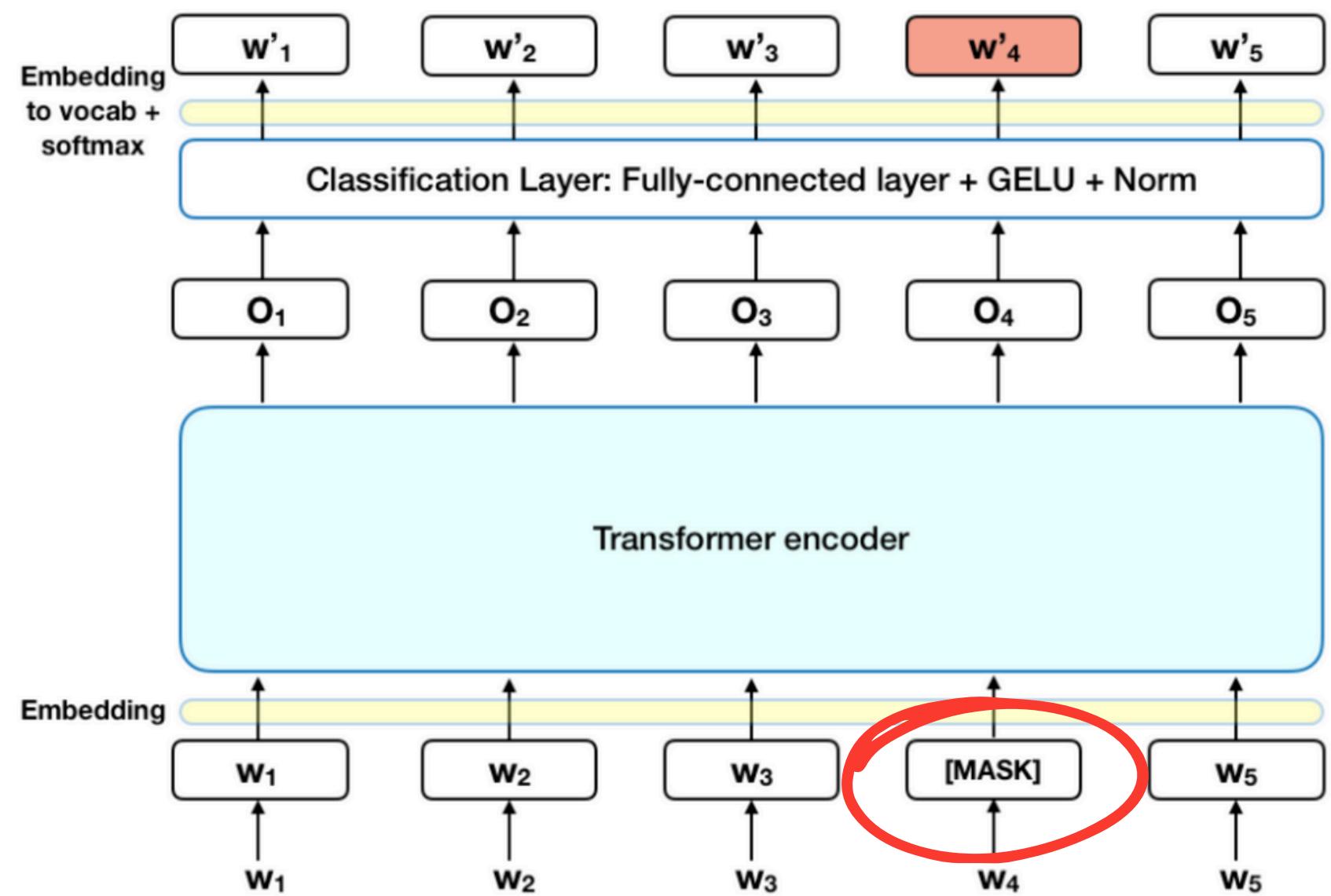
2단계 Fine-Tuning
라벨이 있는 데이터로 전체 파라미터를 미세 조정

Pre-training Process



- ① Input Data (입력)
 - Label이 없는 방대한 텍스트에서 두 문장(Sentence A, B)을 가져옴
- ② Tokenizing & Masking (전처리)
 - 입력 토큰의 15%를 [MASK]로 가림
 - 문장 시작([CLS])과 끝([SEP])에 특수 토큰 부착
- ③ BERT Encoder (연산)
 - Deep Bidirectional Transformer 층을 통과하며 문맥 파악
- ④ Task 1: Masked LM (빈칸 추론)
 - 주변 문맥을 통해 가려진 [MASK] 토큰의 원래 단어 예측
- ⑤ Task 2: NSP (다음 문장 예측)
 - [CLS] 토큰의 출력값(C)을 확인하여, 두 문장이 이어지는지 판단 (IsNext/NotNext)

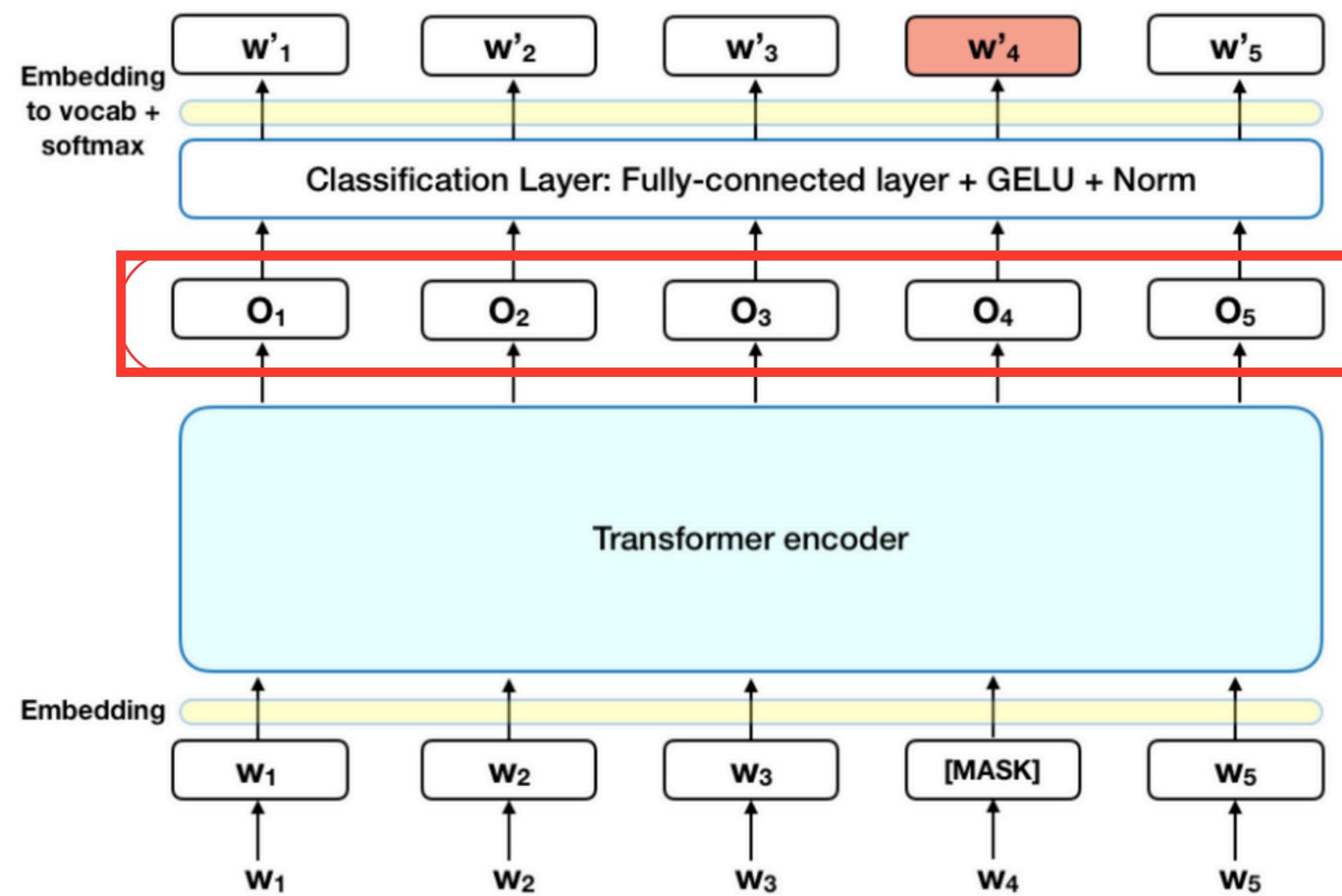
Masked LM



- 일반적인 구조에서 양방향을 적용하면, 모델이 예측해야 할 단어를 미리 봐버리는 문제가 있음

→ 입력 토큰 중 15%를 마스킹해서 단어가 뭔지 맞추게 시킴
예: the man went to [MASK] store → 정답: the

Masked LM



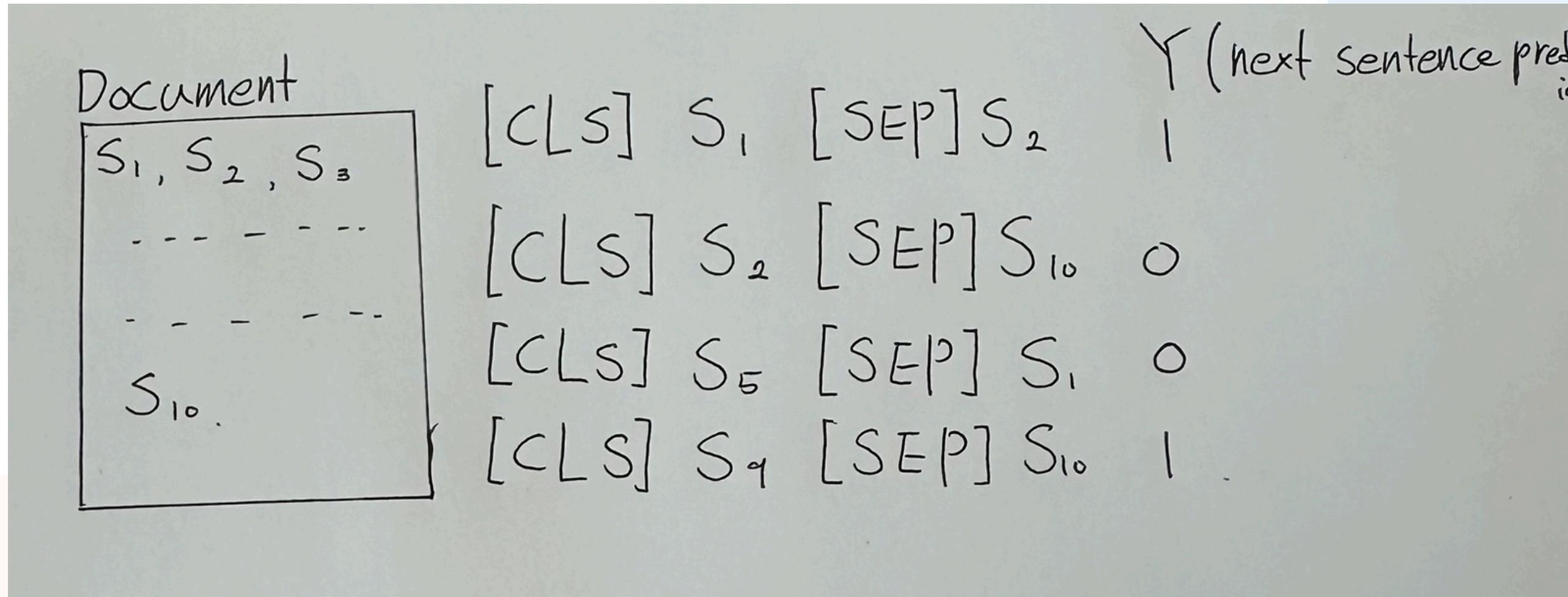
- pre-training 과정에서는 mask token이 등장하지만, fine-tuning 과정에서는 mask token이 등장하지 않음 → mismatch 발생

→ mask token의 80%는 그대로, 10% 오답, 나머지 10% 정답

- (80%) → my dog is [MASK]
- (10%) → my dog is apple → 문맥상 틀린 걸 수정
- (10%) → my dog is hairy → 정답이어도 의심하고 확인

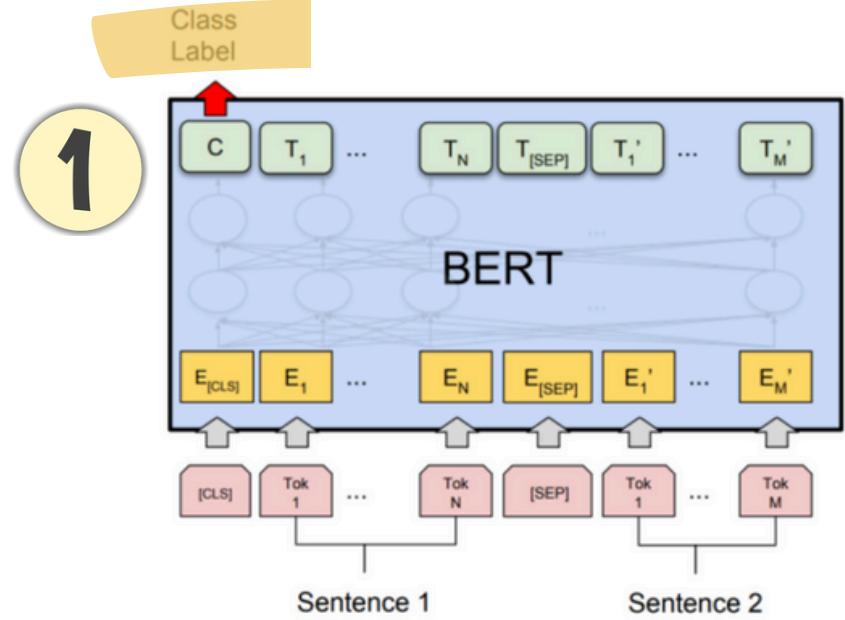
| MASK | Masking Rates | | | Dev Set Results | | |
|------|---------------|------|------|-----------------|-----------|---------------|
| | SAME | RND | MNLI | NER | | |
| | | | | Fine-tune | Fine-tune | Feature-based |
| 80% | 10% | 10% | 84.2 | 95.4 | 94.9 | |
| 100% | 0% | 0% | 84.3 | 94.9 | 94.0 | |
| 80% | 0% | 20% | 84.1 | 95.2 | 94.6 | |
| 80% | 20% | 0% | 84.4 | 95.2 | 94.7 | |
| 0% | 20% | 80% | 83.7 | 94.8 | 94.6 | |
| 0% | 0% | 100% | 83.6 | 94.9 | 94.6 | |

Next Sentence Prediction (NSP)

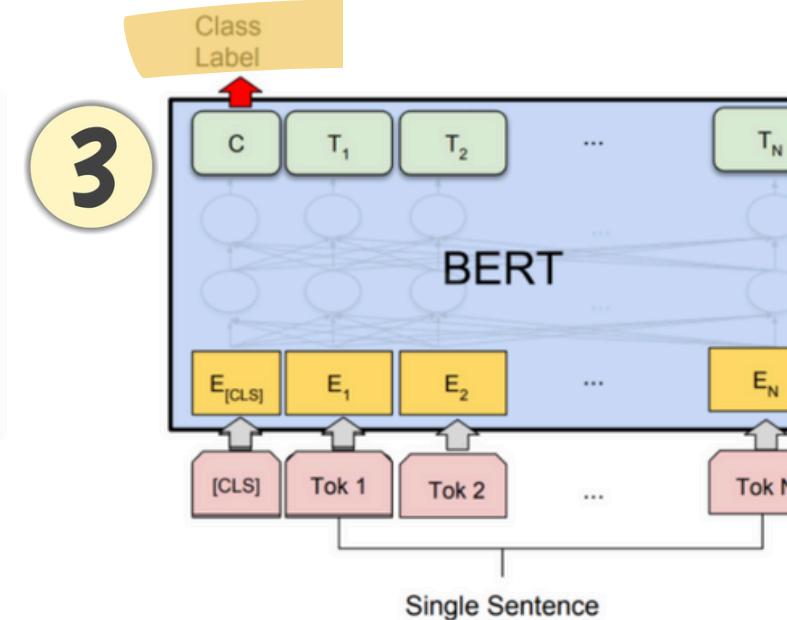


- 문장 사이에 관계를 이해시키기 위해 NSP를 진행함(QA, NLI)
 - 문장 A와 B를 주고, B가 A다음 문장인지 맞추게함
- 50%: 진짜 다음 문장 (IsNext)
- 50%: 램덤하게 가져온 아무 문장 (NotNext)
- A: 남자가 가게에 갔다 / B: 그는 우유를 샀다 → Label: IsNext C: 펭귄은 날지 못한다. → Label: NotNext

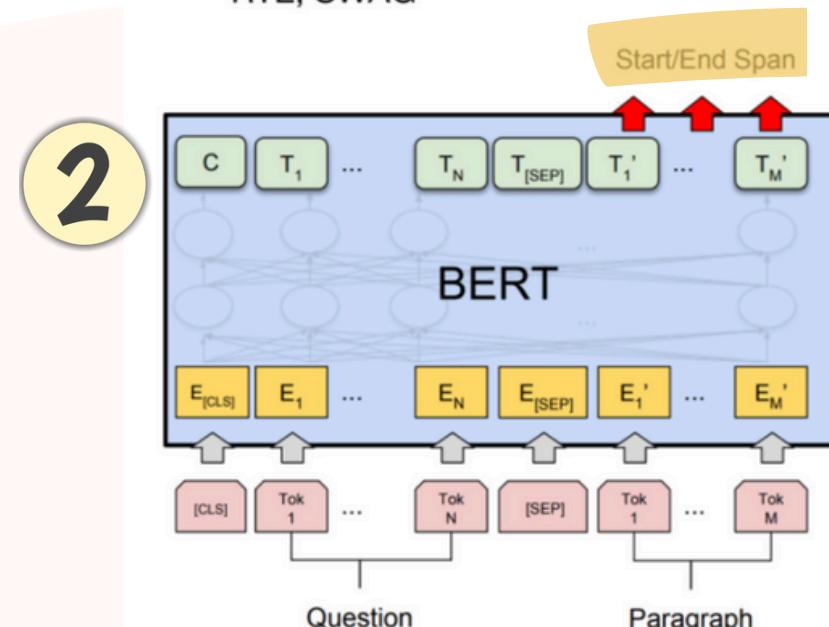
Fine-tuning



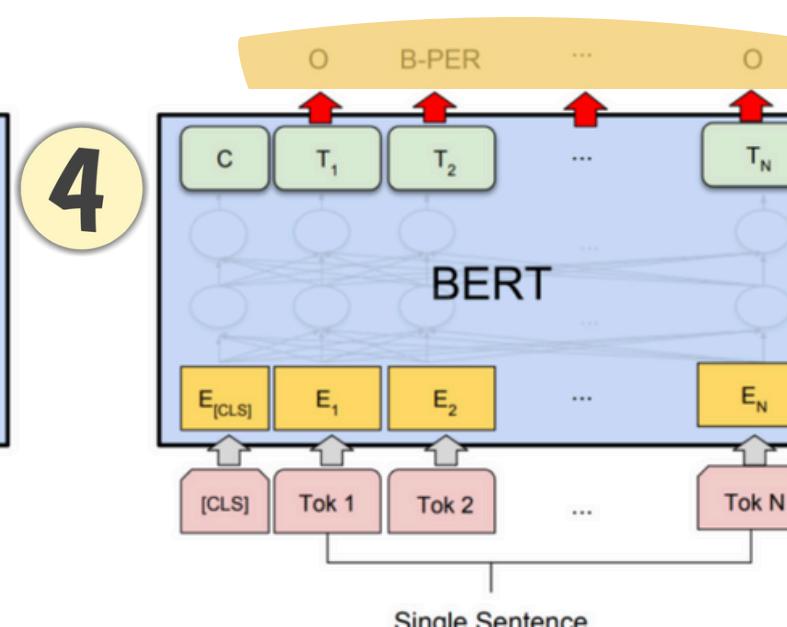
(a) Sentence Pair Classification Tasks:
MNLI, QQP, QNLI, STS-B, MRPC,
RTE, SWAG



(b) Single Sentence Classification Tasks:
SST-2, CoLA



(c) Question Answering Tasks:
SQuAD v1.1



(d) Single Sentence Tagging Tasks:
CoNLL-2003 NER

1. 문장 쌍 분류 (Sentence Pair Classification)

- 입력: 문장 A + 문장 B (예: MNLI, 문장 유사도)
- 역할: 두 문장의 논리적 관계나 유사성을 파악
- 출력: 맨 앞 [CLS] 토큰 하나를 사용하여 관계를 분류함

2. 질의응답 (Question Answering)

- 입력: 질문(Question) + 지문(Paragraph) (예: SQuAD)
- 역할: 지문을 읽고 질문에 대한 정답 위치를 찾음.
- 출력: 정답 구절의 시작과 끝 위치를 찾아냄.

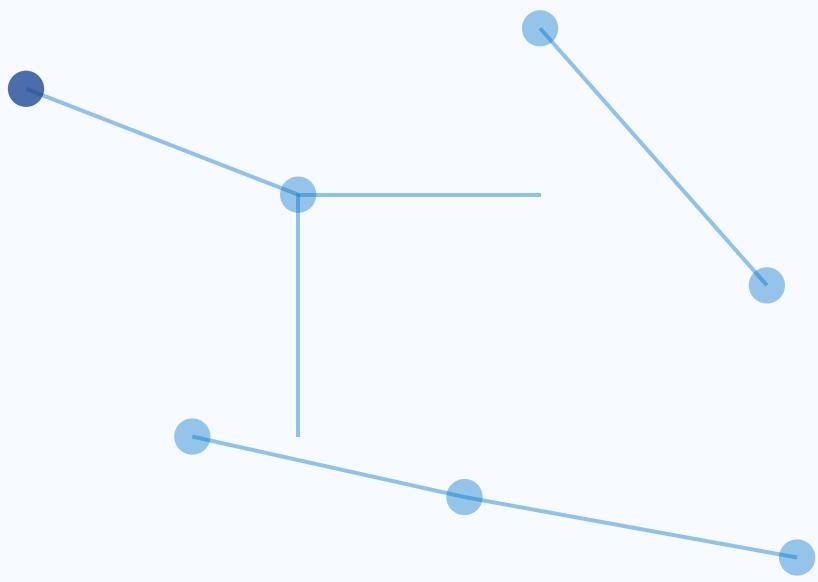
3. 단일 문장 분류 (Single Sentence Classification)

- 입력: 문장 1개 (예: 감성 분석, 스팸 분류)
- 역할: 문장 전체의 주제나 분위기를 파악.
- 출력: 맨 앞 [CLS] 토큰 하나를 사용하여 문장의 종류를 분류함.

4. 개체명 인식 (Single Sentence Tagging / NER)

- 입력: 문장 1개 (예: 형태소 분석, 개체명 인식)
- 역할: 문장이 아닌 각 단어(Token)의 정체를 파악.
- 출력: 모든 토큰의 벡터를 사용하여 각 단어마다 태그를 붙임.

Experiments



GLUE

| System | MNLI-(m/mm) 392k | QQP 363k | QNLI 108k | SST-2 67k | CoLA 8.5k | STS-B 5.7k | MRPC 3.5k | RTE 2.5k | Average |
|-----------------------|---------------------|-------------|--------------|--------------|--------------|---------------|--------------|-------------|-------------|
| Pre-OpenAI SOTA | 80.6/80.1 | 66.1 | 82.3 | 93.2 | 35.0 | 81.0 | 86.0 | 61.7 | 74.0 |
| BiLSTM+ELMo+Attn | 76.4/76.1 | 64.8 | 79.8 | 90.4 | 36.0 | 73.3 | 84.9 | 56.8 | 71.0 |
| OpenAI GPT | 82.1/81.4 | 70.3 | 87.4 | 91.3 | 45.4 | 80.0 | 82.3 | 56.0 | 75.1 |
| BERT _{BASE} | 84.6/83.4 | 71.2 | 90.5 | 93.5 | 52.1 | 85.8 | 88.9 | 66.4 | 79.6 |
| BERT _{LARGE} | 86.7/85.9 | 72.1 | 92.7 | 94.9 | 60.5 | 86.5 | 89.3 | 70.1 | 82.1 |

- GLUE란 자연어 이해 능력 평가를 위한 9개의 다양한 task 집합
 1. MNLI, QQP, QNLI, STS-B: 문장 두 개를 주고 관계 파악하기 (예: 두 문장이 같은 뜻인가?)
 2. SST-2, CoLA: 문장 하나를 보고 판단하기 (예: 긍정/부정, 문법이 맞나?)
 3. MRPC, RTE, WNLI: 적은 데이터로 추론하기
- GPT와 비슷한 Size의 Bert base 모델도 GPT보다 높은 성능
- Bert Large의 경우도 GPT보다 높음
- 현재 RoBERTa / ALBERT 등 모델의 기반이 될 정도로 우수한 모델

SQuAD v1.1

| System | Dev | | Test | |
|---|-------------|-------------|-------------|-------------|
| | EM | F1 | EM | F1 |
| Top Leaderboard Systems (Dec 10th, 2018) | | | | |
| Human | - | - | 82.3 | 91.2 |
| #1 Ensemble - nlnet | - | - | 86.0 | 91.7 |
| #2 Ensemble - QANet | - | - | 84.5 | 90.5 |
| Published | | | | |
| BiDAF+ELMo (Single) | - | 85.6 | - | 85.8 |
| R.M. Reader (Ensemble) | 81.2 | 87.9 | 82.3 | 88.5 |
| Ours | | | | |
| BERT _{BASE} (Single) | 80.8 | 88.5 | - | - |
| BERT _{LARGE} (Single) | 84.1 | 90.9 | - | - |
| BERT _{LARGE} (Ensemble) | 85.8 | 91.8 | - | - |
| BERT _{LARGE} (Sgl.+TriviaQA) | 84.2 | 91.1 | 85.1 | 91.8 |
| BERT _{LARGE} (Ens.+TriviaQA) | 86.2 | 92.2 | 87.4 | 93.2 |

- SQuAD란 10만 개의 질문-정답 쌍으로 구성된 기계 독해 능력 평가 벤치마크
주워진 지문을 읽고 질문에 대한 정답의 시작과 끝을 찾아냄
- BERT_Base → 88.5
- BERT_Large → 90.9
- 사람 평균 → 91.2 / BERT_Large (Ensemble + TriviaQA) → 93.2
→ 역사상 최초로 인간보다 뛰어난 독해 능력

SQuAD v2.0

| System | Dev | | Test | |
|--|------|------|------|------|
| | EM | F1 | EM | F1 |
| Top Leaderboard Systems (Dec 10th, 2018) | | | | |
| Human | 86.3 | 89.0 | 86.9 | 89.5 |
| #1 Single - MIR-MRC (F-Net) | - | - | 74.8 | 78.0 |
| #2 Single - nlnet | - | - | 74.2 | 77.1 |
| Published | | | | |
| unet (Ensemble) | - | - | 71.4 | 74.9 |
| SLQA+ (Single) | - | - | 71.4 | 74.4 |
| Ours | | | | |
| BERT _{LARGE} (Single) | 78.7 | 81.9 | 80.0 | 83.1 |

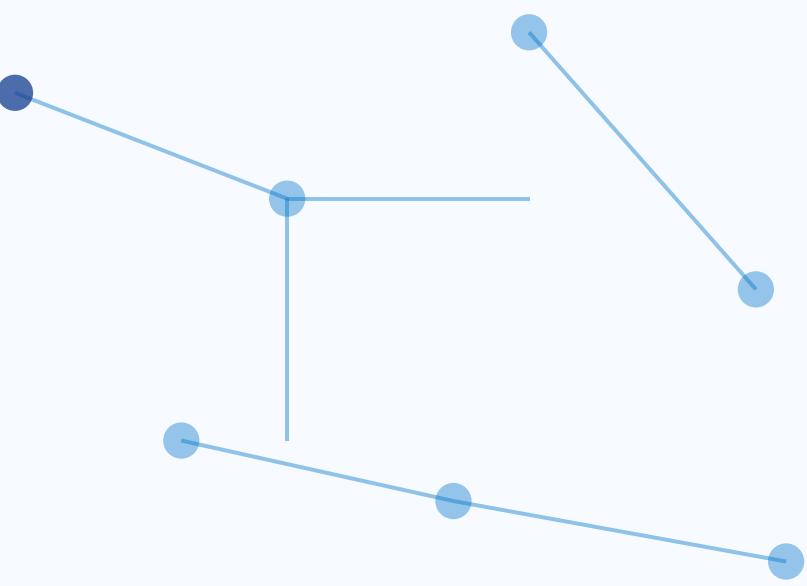
- SQuAD v1.1의 확장판으로, "Unanswerable Questions"이 추가된 데이터셋 모델은 정답을 찾는 것뿐만 아니라, "정답이 없다"는 사실을 스스로 판단해야 함
- Previous SOTA → 78.0
- BERT_{LARGE} → 83.1
→ BERT의 Deep Bidirectionality 덕분

SWAG

| System | Dev | Test |
|------------------------------------|-------------|-------------|
| ESIM+GloVe | 51.9 | 52.7 |
| ESIM+ELMo | 59.1 | 59.2 |
| OpenAI GPT | - | 78.0 |
| BERT _{BASE} | 81.6 | - |
| BERT _{LARGE} | 86.6 | 86.3 |
| Human (expert) [†] | - | 85.0 |
| Human (5 annotations) [†] | - | 88.0 |

- 주어진 문장 다음에 이어질 가장 자연스러운 상황을 4개의 보기 중에서 선택하는
상식 추론 task
- OpenAI GPT → 78.0
- BERT_Large → 86.3
- 사람 평균 → 85.0 / BERT_Large → 86.3

Ablation Studies



Effect of Pre-training(NSP) Tasks

| Tasks | Dev Set | | | | |
|----------------------|-----------------|---------------|---------------|----------------|---------------|
| | MNLI-m (Acc) | QNLI (Acc) | MRPC (Acc) | SST-2 (Acc) | SQuAD (F1) |
| BERT _{BASE} | 84.4 | 88.4 | 86.7 | 92.7 | 88.5 |
| No NSP | 83.9 | 84.9 | 86.5 | 92.6 | 87.9 |
| LTR & No NSP | 82.1 | 84.3 | 77.5 | 92.1 | 77.8 |
| + BiLSTM | 82.1 | 84.1 | 75.7 | 91.6 | 84.9 |

- MRPC,SST-2 한 문장만 필요하는 단순한 task에서는 성능 향상이 비교적 작게 일어남
두 문장 사이의 관계가 중요한 QNLI,SQuAD,MNLI-m task에서는 유의미한 성능 차이가 일어남

Effect of Model Size

| Hyperparams | | | | Dev Set Accuracy | | |
|-------------|------|----|----------|------------------|------|-------|
| #L | #H | #A | LM (ppl) | MNLI-m | MRPC | SST-2 |
| 3 | 768 | 12 | 5.84 | 77.9 | 79.8 | 88.4 |
| 6 | 768 | 3 | 5.24 | 80.6 | 82.2 | 90.7 |
| 6 | 768 | 12 | 4.68 | 81.9 | 84.8 | 91.3 |
| 12 | 768 | 12 | 3.99 | 84.4 | 86.7 | 92.9 |
| 12 | 1024 | 16 | 3.54 | 85.7 | 86.9 | 93.3 |
| 24 | 1024 | 16 | 3.23 | 86.6 | 87.8 | 93.7 |

- BERT는 충분히 잘 pre-training 되어 있으면 아주 작은 데이터셋에도 큰 효과를 기대할 수 있음

Feature-based Approach with BERT

| System | Dev F1 | Test F1 |
|--|--------|-------------|
| ELMo (Peters et al., 2018a) | 95.7 | 92.2 |
| CVT (Clark et al., 2018) | - | 92.6 |
| CSE (Akbik et al., 2018) | - | 93.1 |
| <hr/> | | |
| Fine-tuning approach | | |
| BERT _{LARGE} | 96.6 | 92.8 |
| BERT _{BASE} | 96.4 | 92.4 |
| <hr/> | | |
| Feature-based approach (BERT _{BASE}) | | |
| Embeddings | 91.0 | - |
| Second-to-Last Hidden | 95.6 | - |
| Last Hidden | 94.9 | - |
| Weighted Sum Last Four Hidden | 95.9 | - |
| Concat Last Four Hidden | 96.1 | - |
| Weighted Sum All 12 Layers | 95.5 | - |

- BERT의 파라미터를 Fine-tuning하는 방식과, BERT를 고정하고 Feature-based하는 방식의 성능을 비교
- Fine - tuning → 96.6, 96.4
- 입력 임베딩만 사용 → 91
- 두번째부터 마지막 Hidden Vector를 참고 → 95.6
- 마지막 Hidden Vector만 참고 → 94.9
- 마지막에서 4번째까지의 Hidden Vector를 자중합 → 95.9
- 마지막 4개의 Hidden Vector concat → 96.1
- 모든 Hidden Vector를 가중합 → 95.5

→ BERT는 굳이 무겁게 Fine-tuning을 하지 않고, 내부 레이어의 값을 Feature로 추출해서 사용해도 매우 강력한 성능을 냄

Why Feature-based Approach Works?

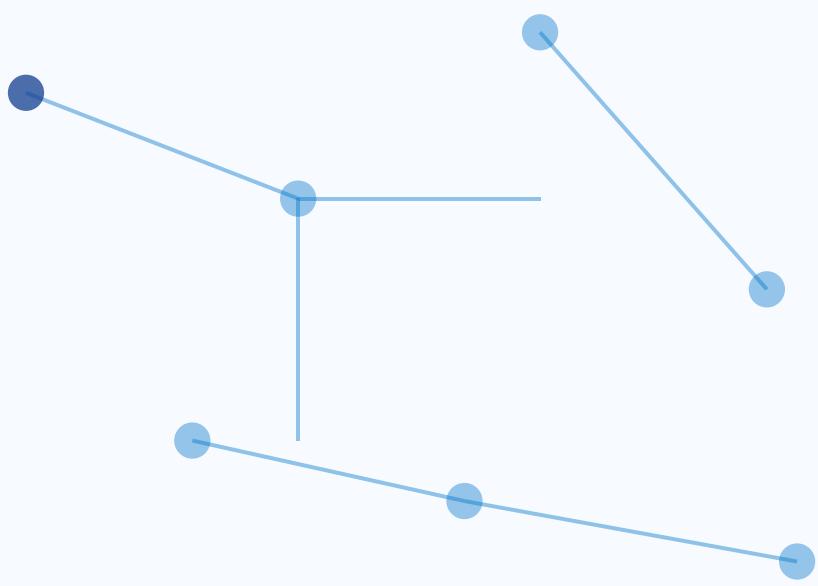
1. 이미 완성형인 "문맥화된 임베딩" (Contextualized Embeddings)

- 기존 모델(Word2Vec)의 한계: '배(Ship)'와 '배(Pear)'를 똑같은 숫자(Vector)로 처리함. 문맥을 모름.
- BERT의 혁신: 사전 학습(Pre-training) 단계에서 Deep Bidirectional 구조를 통해, 문장 내의 모든 단어 관계를 이미 파악함.
- 결과: BERT가 뱉어내는 내부 레이어의 값은 단순한 숫자가 아니라, "이 문맥에서 이 단어는 이런 뜻이야"라고 완벽하게 해석된 고품질의 정보임.

2. 레이어별 정보의 풍부함 (Rich Hierarchy)

- BERT의 12개(또는 24개) 레이어는 각자 다른 정보를 배움.
- 하위 레이어: 문법, 구문 정보 (Syntax).
- 상위 레이어: 의미, 문맥 정보 (Semantics).
- Concatenation의 힘: 마지막 4개 레이어를 합친다는 것은, 문법적인 정보와 의미적인 정보를 한꺼번에 가져다 쓴다는 의미. 그러나 성능이 좋을 수밖에 없음.

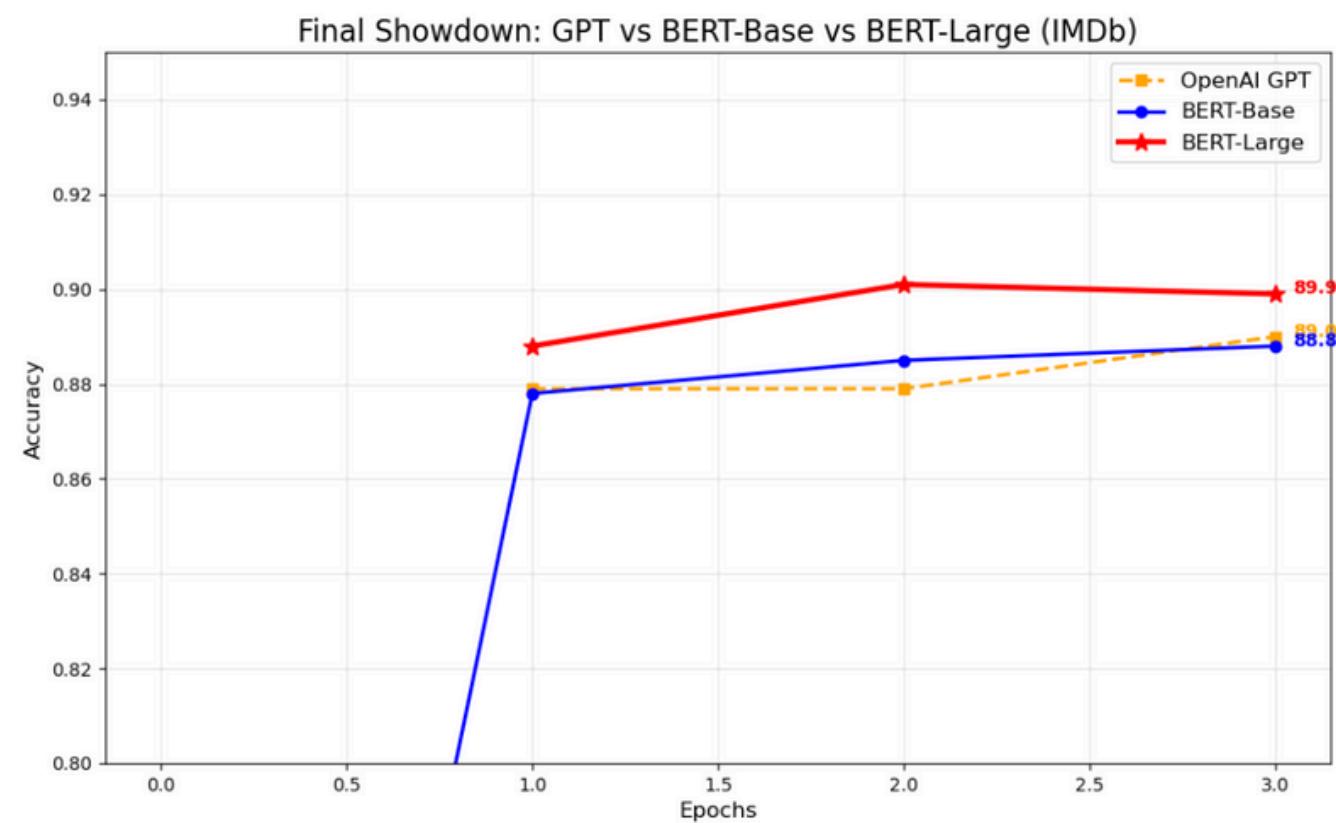
conclusion



| conclusion

- BERT는 Deep Bidirectional 학습을 시킨 모델
- Pre-training을 시켜서 각 down-stream task에 맞게 Fine-tuning 해주는 방식으로 학습 시키면 됨
- Pre-training 과정에서 MLM, NSP의 두 가지 주요 task를 수행
- 이후 RoBERT, ALBERT 등의 모델의 기반이 되었고, SOTA를 달성할 만큼 뛰어난 성능을 가짐

BERT vs. GPT (IMDb Analysis)



- 데이터셋: IMDb Movie Reviews (50k) - 긍정/부정 이진 분류.
- Bidirectional이 성능이 더 좋음, Size Matters
- 3 Epoch때 왜 GPT 성능이 BERT_Base 보다 좋지?
→ 고난도 task에선 BERT가 압도적이지만 단순한 task는 GPT도 잘 함
- 3 Epoch때 왜 BERT_Large의 성능이 소폭 감소하지?
→ 과적합(Overfitting) 사소한 노이즈나 특이점까지 억지로 외우기 시작 모델이 커서 적은 횟수만으로도 이미 충분히 학습 됨. 논문에서도 2,3,4회 추천