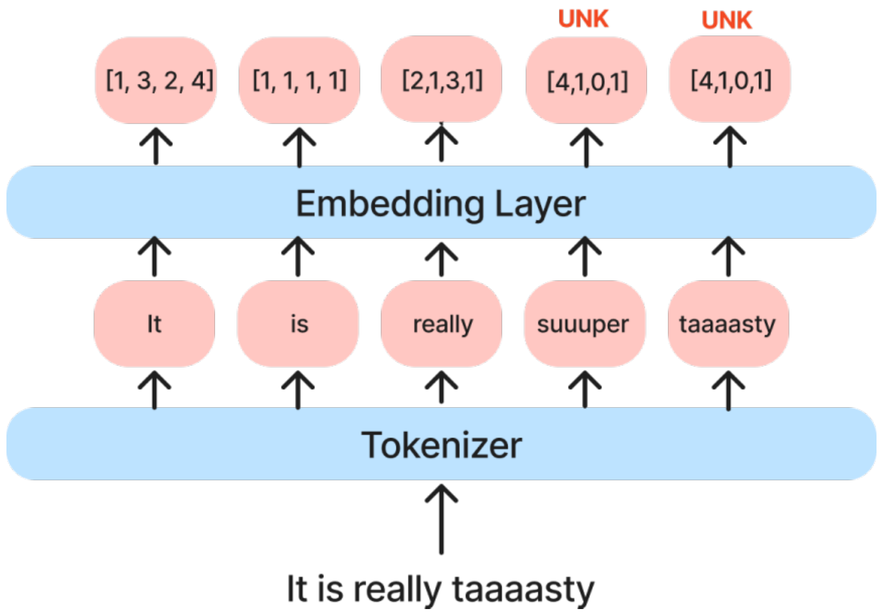
Transformers & Pretraining

Out of Vocabulary Problem

■ 임베딩 과정과 문제점



1. 문자열 입력
2. 학습된 토크나이저로 토큰화
3. 각 토큰별 학습된 임베딩 벡터 변환

- 학습 데이터에 존재하지 않는 단어
  → UNK 토큰으로 토큰화
  → 다른 토큰임에도 불구하고 동일 벡터로 변환되는 문제점
  ⇒ 신조어, 오탈자에 대응하기 어려움

The byte-pair encoding algorithm

■ Subword 단위 Token화

- 단어와 문자 단위 토큰화의 중간 위치의 형태
- 대표적인 서비워드 단위 토큰화 기법: Byte Pair Encoding
  GPT 모델: BPE / BERT 모델: wordpiece

■ Byte Pair Encoding : BPE

1. 단어를 문자 단위로 분해 — a a a b d a a a b a c
2. 전체 코퍼스에서 가장 자주 등장하는 연속된 두 문자열 병합 — aa a b d aa a b a c
3. 병합된 토큰을 기반으로 조합 — ZabdZabac
4. 종료 조건(병합 횟수, 토큰 개수 등)을 만족할 때 까지 1~3 반복

⇒ 신조어, 오타에 대응 가능
⇒ OOV 문제에서 비교적 자유로움
⇒ 사전 크기 증가를 억제하면서도 정보를 효율적으로 압축할 수 있는 알고리즘

Contextualized Word Embedding

■ Fixed Word Embedding

- 각 단어에 대해 사전에 학습된 임베딩 벡터를 부여
→ 다른 문맥임에도 동일 벡터가 맵핑
→ 문맥을 고려하지 못한 임베딩 방식

배 위에서 **배**를 긁으며 **배**를 먹는다. → '배'라는 단어가 의미하는 것이 다르지만 동일 벡터가 맵핑 됨

■ Pretrain Model

Input의 일부를 훼손하고 모델이 훼손된 input을 복원하면서 언어가 가진 의미적/문법적 구조를 학습

- Downstream Task (실제 수행할 구체적인 태스크)
: pretrain한 이유는 downtream task를 잘 하기 위해서이다.
Downstream Task의 학습 방식은 Pretrain Model의 weight를 initial weight으로 삼아 fine tuning을 진행

Pretrain Objective functions

■ Language Modeling

$$L = \sum \log P(w_t | w_0, \dots, w_{t-1})$$

- 이전 토큰들을 바탕으로 현재 시점의 토큰을 예측
→ 별도의 label이 없어도 되기 때문에 다량의 데이터 확보가 가능
- LM은 그 자체로 언어구조/문법 및 의미적 정보 파악이 가능

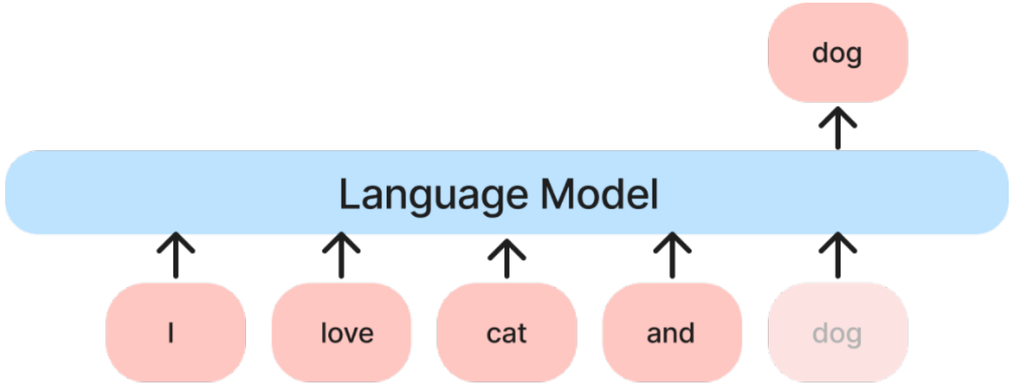
■ SGD with Pretrain/Finetune

Pretrain 과정에서 학습한 parameter들을 Finetuning 과정의 initial weight로 사용하게 되면 Random initializing 된 weight에 비해서 global optimum에 가까울 수 있다.
→ 적은 데이터로도 잘 학습, 쉽게 수렴 가능

Pretrain Model: GPT-1

■ Language Modeling

- 트랜스포머의 Decoder만 사용
- 단방향 모델
- 일반적인 LM을 통해 Pretrain
  : LM은 label이 필요 없음



$$L = \sum_i \log P(w_i \mid w_0, \ldots, w_{i-1}; \theta)$$

$$h_0 = UW_e + W_p$$

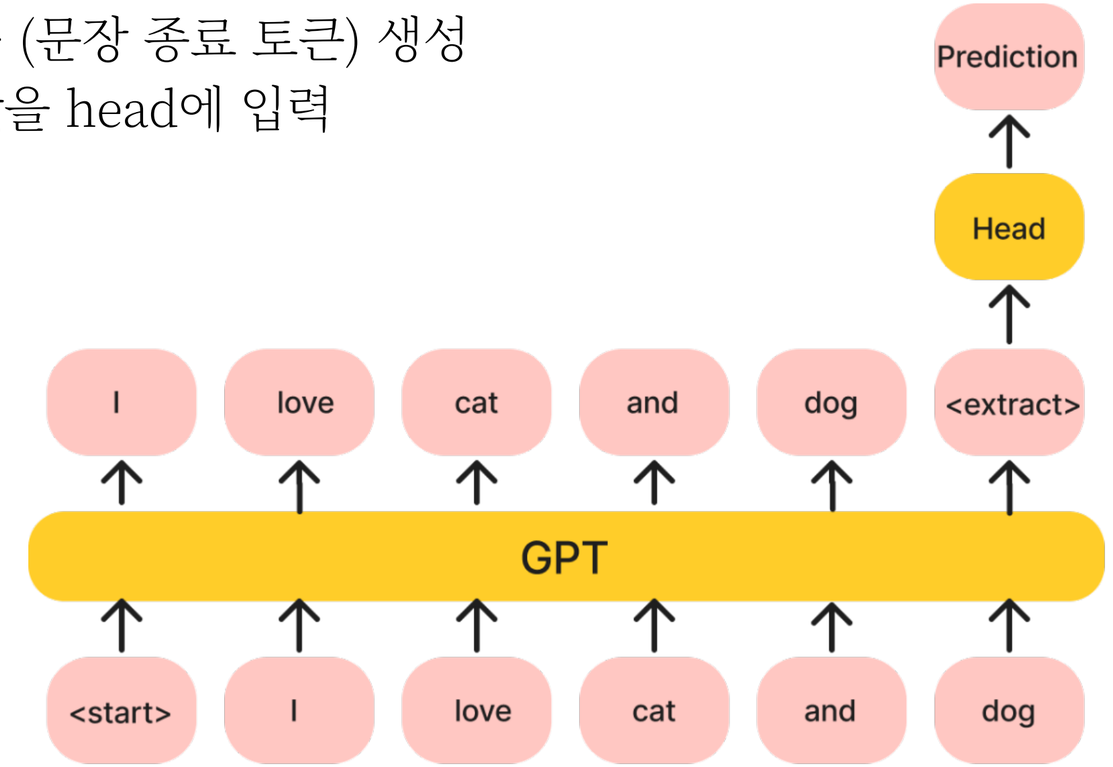
$$h_l = Decoder\ block_i(h_{l-1})$$

$$P(w_i) = softmax(h_n W_e^T)$$

Pretrain Model: GPT-1

■ Finetuning

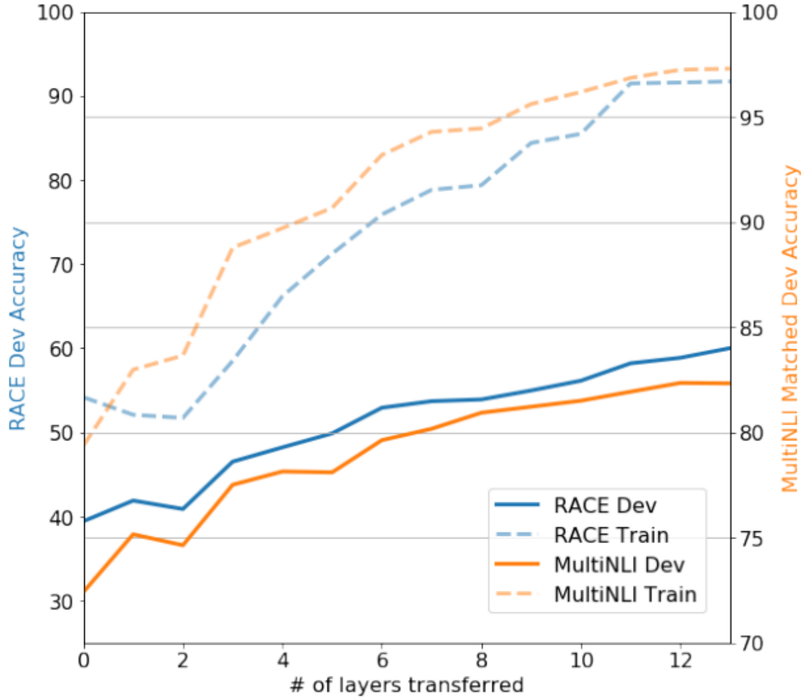
- 문장 마지막에 <Extract> (문장 종료 토큰) 생성
- <Extract> 위치의 출력값을 head에 입력

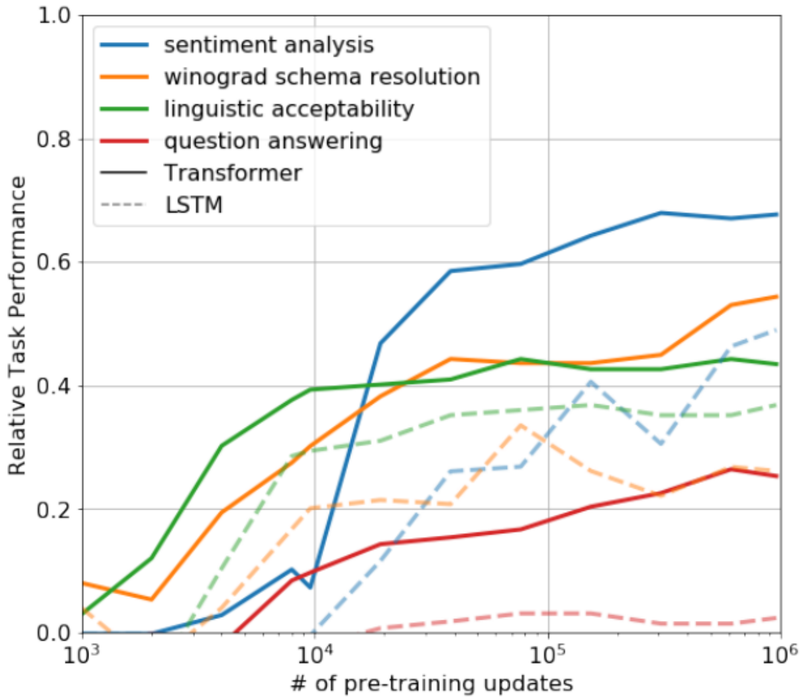


Pretrain Model: GPT-1

■ Pretrain이 Downstream task에 효과적

- Pretrain layer를 사용할수록 성능이 향상
- Pretrain을 진행한 모델일수록 성능이 향상





Pretrain Model: BERT

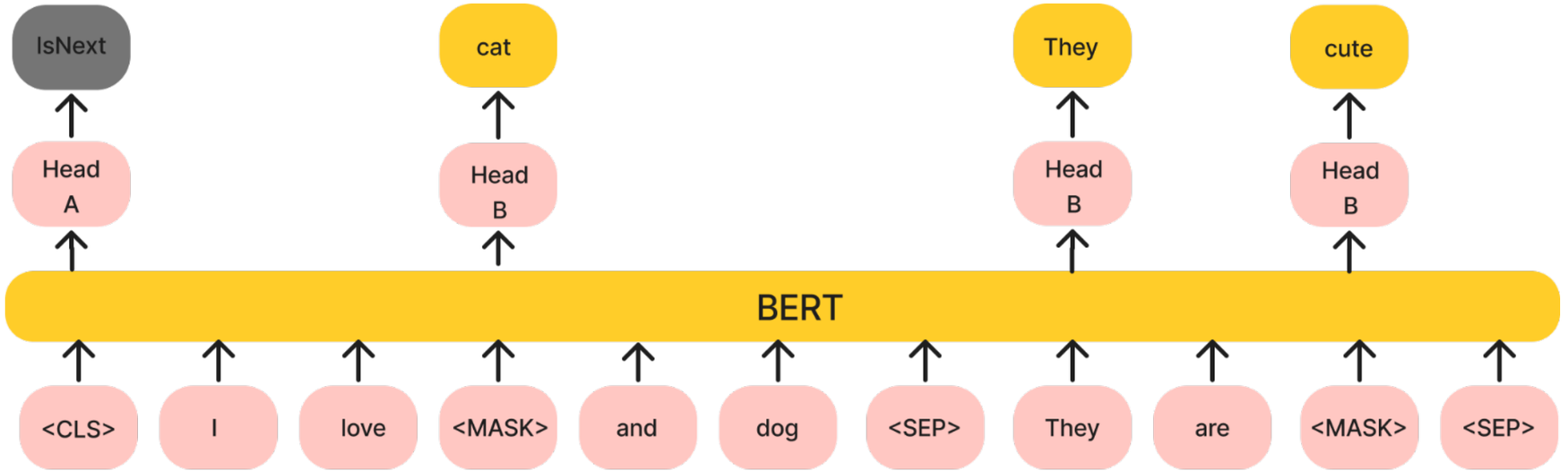
■ BERT

- 트랜스포머 Encoder만 사용
- 양방향 모델
- Pretrain task는 MLM과 NSP가 있음

$$h_0 = UW_e + W_p$$

$$h_l = Encoder\ block_i(h_{l-1})$$

$$P(w_i) = softmax(h_n W_e^T)$$



Pretrain Model: BERT

■ Masked Language Modeling (MLM)

$$L_{MLM} = -\sum_{i \in M} \log P(w_i \mid w_0, \ldots, w_n; \theta)$$

(M : Masked tokens, n : sequence length)

- 양방향 토큰의 정보를 활용하기 때문에 Language Model objective function을 사용할 수 없다.

-→ 입력 토큰의 15%를 아래와 같이 변형하고 이를 예측하는 방식
  - 80% : [MASK]
  - 10% : 랜덤한 다른 토큰
  - 10% : 원본 토큰

Pretrain Model: BERT

■ Next Sentence Prediction (NSP)

$$L_{NSP} = -\log \hat{y}P(IsNext\ |w_0, \dots, w_n) - (1 - \hat{y})(1 - P(IsNext|w_0, \dots, w_n))$$

(M : Masked tokens, n : sequence length)

- MLM은 문장 간 관계를 다루지 않기 때문에 이를 반영한 pretrain task가 필요

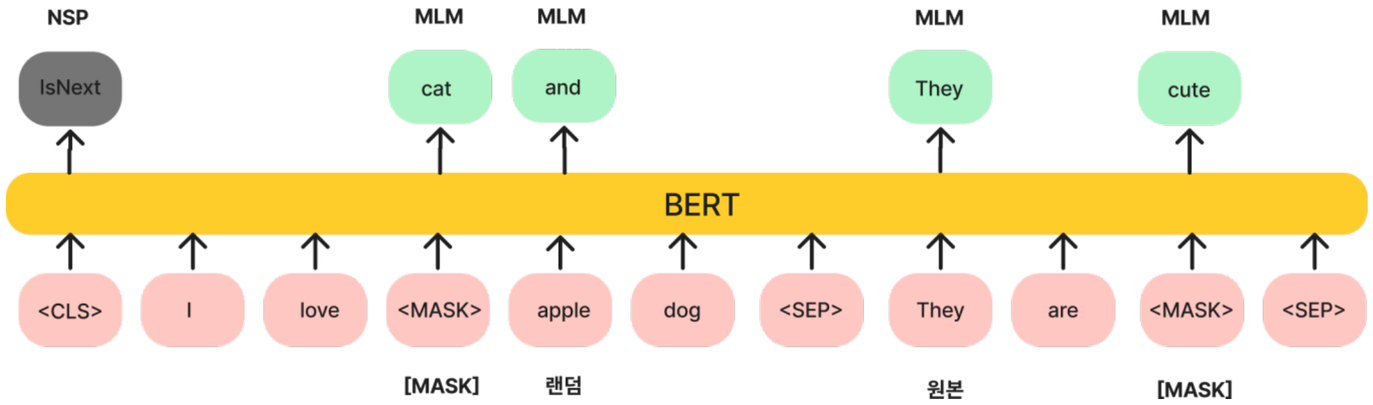
→ 두 문장 입력

- 50%: 학습 데이터에서 연속된 문장
- 50%: 서로 다른 문서에서 사용된 문장

⇒ 연속된 문장 여부 판단(IsNext)

→

- 문장 간 관계 학습
- <CLS> 토큰이 입력 문장의 정보를 담음



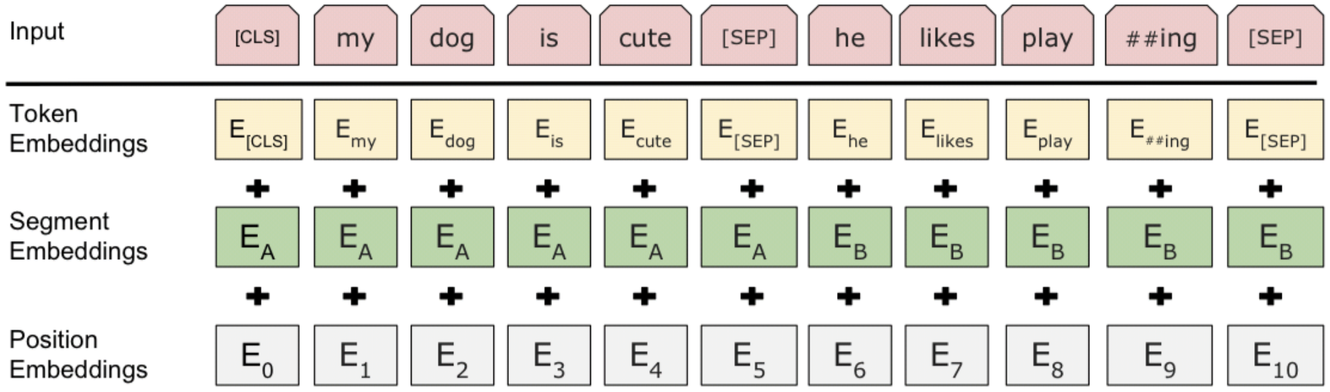
+ RoBerta 논문에서 NSP가 학습에 도움이 되지 않고, BERT가 과소적합 되어 있음을 보임

Pretrain Model: BERT

■ Input

- BERT는 Token, Segment, Position 임베딩 벡터를 더하여 input 임베딩 벡터를 생성

- ▸ Token Embedding : Wordpiece 기반 토큰 단위 임베딩
- ▸ Segment Embedding : 두 문장을 구분하기 위한 임베딩
- ▸ Position Embedding : 트랜스포머의 positional encoding



Pretrain Model: BERT

■ Input

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
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

- GPT-1과 동일한 파라미터 수이면서 더 좋은 성능
  (양방향 모델링에서 오는 이점)
- 모델의 크기를 키움으로서 더 성능을 높일 수 있음을 보임
- 인코더 모델의 한계로 생성 downstream task에 파인튜닝이 어려움

Pretrain Model: BERT

■ RoBERTa

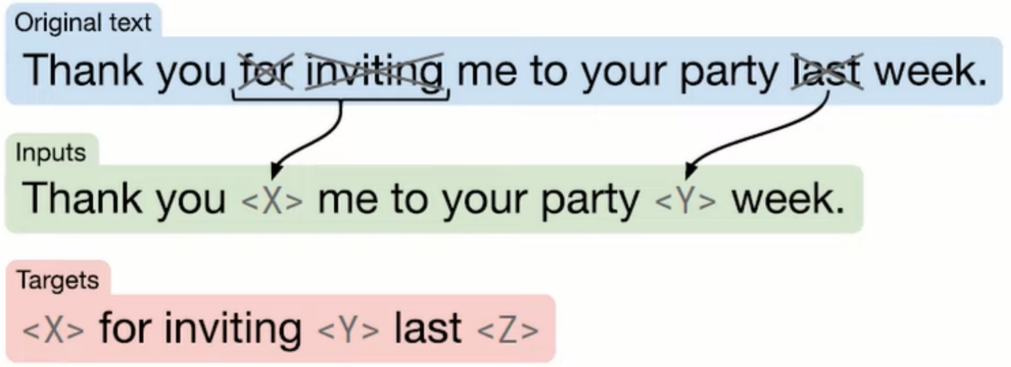
Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7

- BERT 구조를 그대로 가져오되, NSP를 제외하고 추가적인 하이퍼파라미터 튜닝을 진행
- Pretrain 시 추가적인 데이터셋과 epoch로 더 높은 성능을 보일 수 있음을 증명

Pretrain Model: T5

■ Pretrain

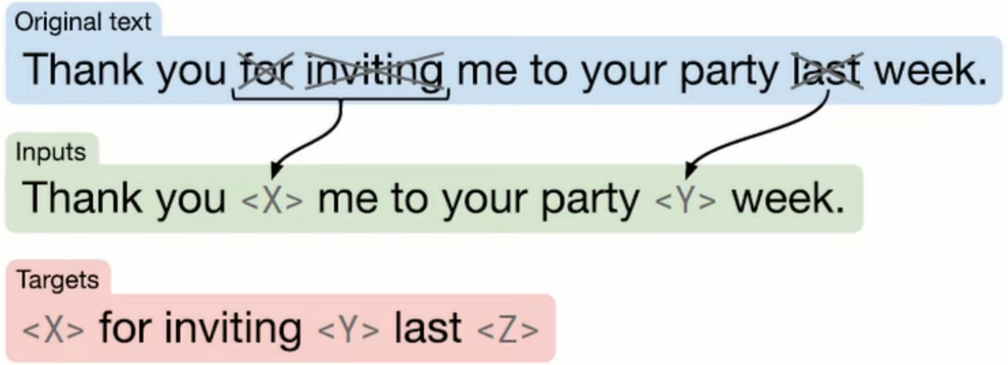
- 트랜스포머 구조를 가져오되 **데이터셋과 모델을 키우고 pretrain 과정을 정교화**
- pretrain task로 span corruption 사용



▸ Original text에서 단어를 masking 하는 것은 동일하지만 연속된 토큰이 지워질 가능성이 높다.

Pretrain Model: T5

■ Pretrain task: span corruption



- 인코더의 입력으로 MASK 사용
- 디코더의 출력으로 MASK ID와 해당하는 문장 생성
- 인코더는 MLM을 디코더는 LM을 수행하는 pretrain task

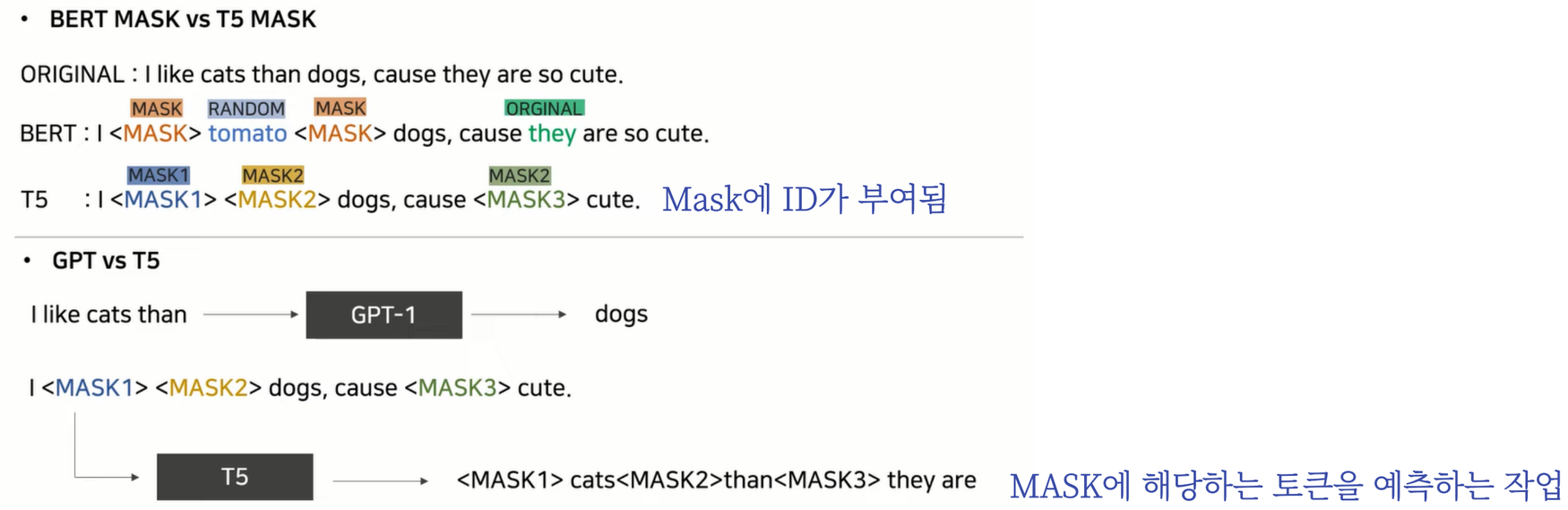
⇒ GPT보다 생성을 잘하고, BERT보다 파악을 잘하는 모델

Pretrain Model: T5

■ Pretrain task: span corruption

BERT의 Masking : 1대1 Masking
Span Corruption의 Masking : 1대다 Masking

- 전체 토큰 중 15%를 25개의 MASK로 Masking
→ BERT는 모델이 하나의 토큰을 **복원**한다고 볼 수 있다면, T5는 마스킹된 부분을 **생성**한다고 볼 수 있음

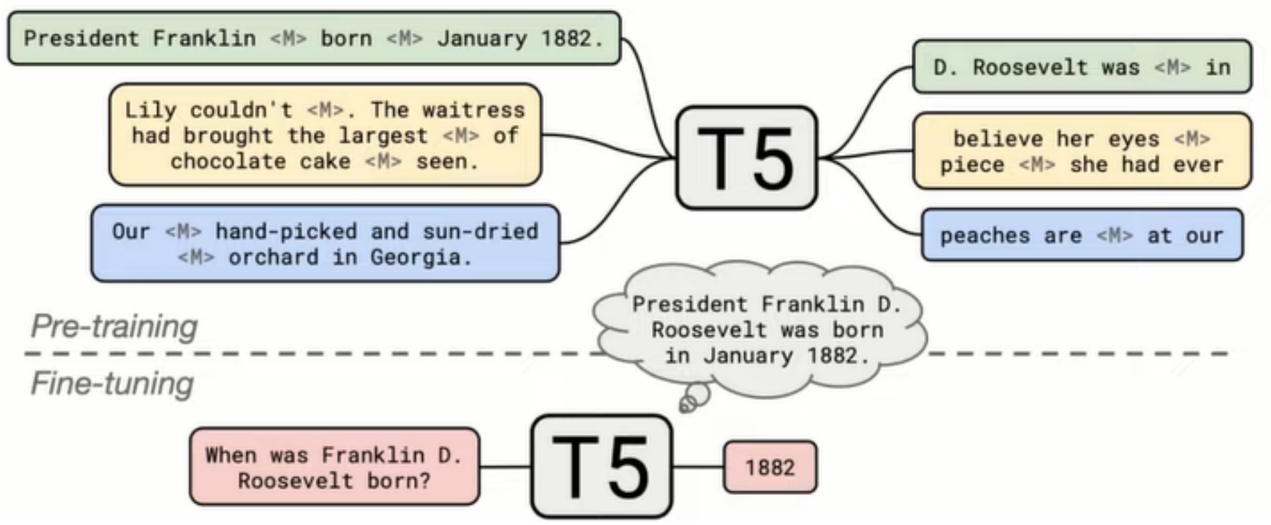


Pretrain Model: T5

■ Open Domain

- 기존의 Question & Answering Task
  : 질문과 정답이 포함된 문장이 입력되면 모델이 정답을 생성하거나 정답에 해당하는 토큰의 인덱스를 반환

- T5는 질문만 주어져도 어느정도 답변이 가능함
  : 대량의 데이터셋(750GB)로 학습했기 때문
  : 모델이 정보를 파라미터화 하여 기억



Pretrain Model

■ Summary

- Task 별 모델 설계에서 Pretrain Model로 패러다임이 변화함
- 훈련 데이터 및 모델 크기가 기하급수적으로 늘어남
- 모델의 성능은 파라미터 수와 비례
- 적절한 pretrain task + 대량의 데이터 + 큰 모델은 매우 효과적인 fine tuning을 보장