



BERT and Other Pre-trained Language Models

송민경 주연우

목차

#01 Contextual representation

#02 History of Contextual Representation

#03 BERT

#04 Post-Bert pre-training Advancement

#05 Distillation



1. Contextual representation



#01. Contextual representation

#1 Contextual representation

bank가 [0.3, 0.2, -0.8,...]으로 문맥 고려x / 동음이의어 구분x

- **Problem:** Word embeddings are applied in a context free manner

open a bank account on the river bank

← [0.3, 0.2, -0.8, ...] →

- **Solution:** Train *contextual* representations on text corpus

[0.9, -0.2, 1.6, ...] [-1.9, -0.4, 0.1, ...]

↑ ↑

open a bank account on the river bank

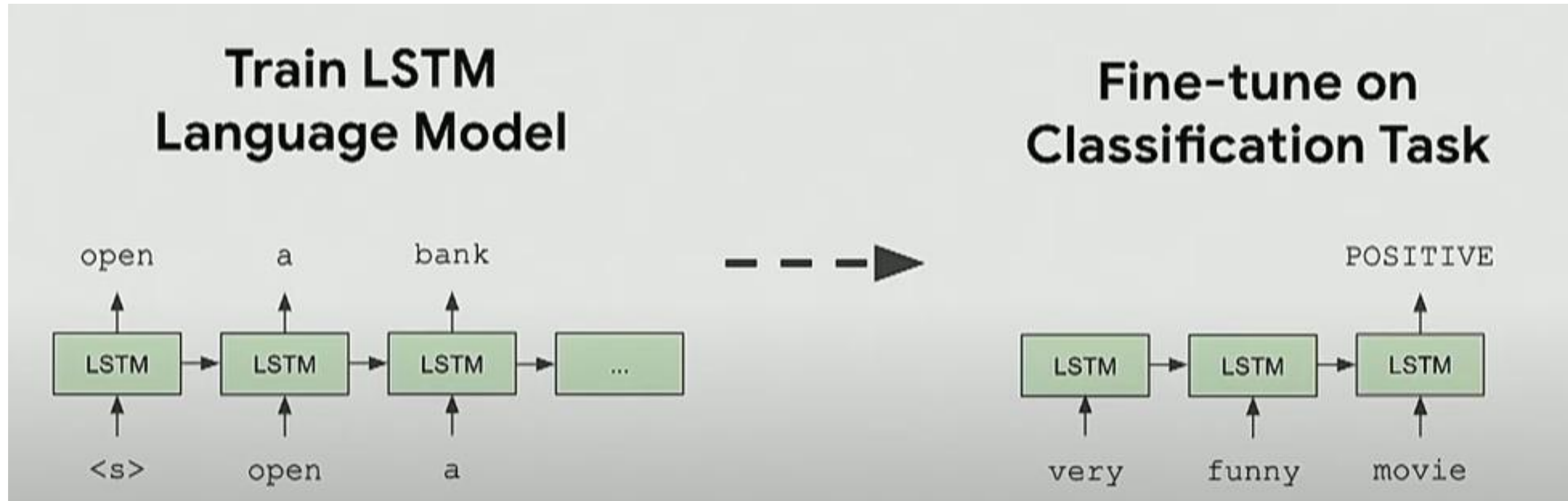
텍스트 corpus 문맥상의 표현을 학습하는 Train contextual representation on corpus가 제안됨

2. History of Contextual Representation



#02. History of Contextual Representation

#1 semi-supervised sequence learning, Google, 2015

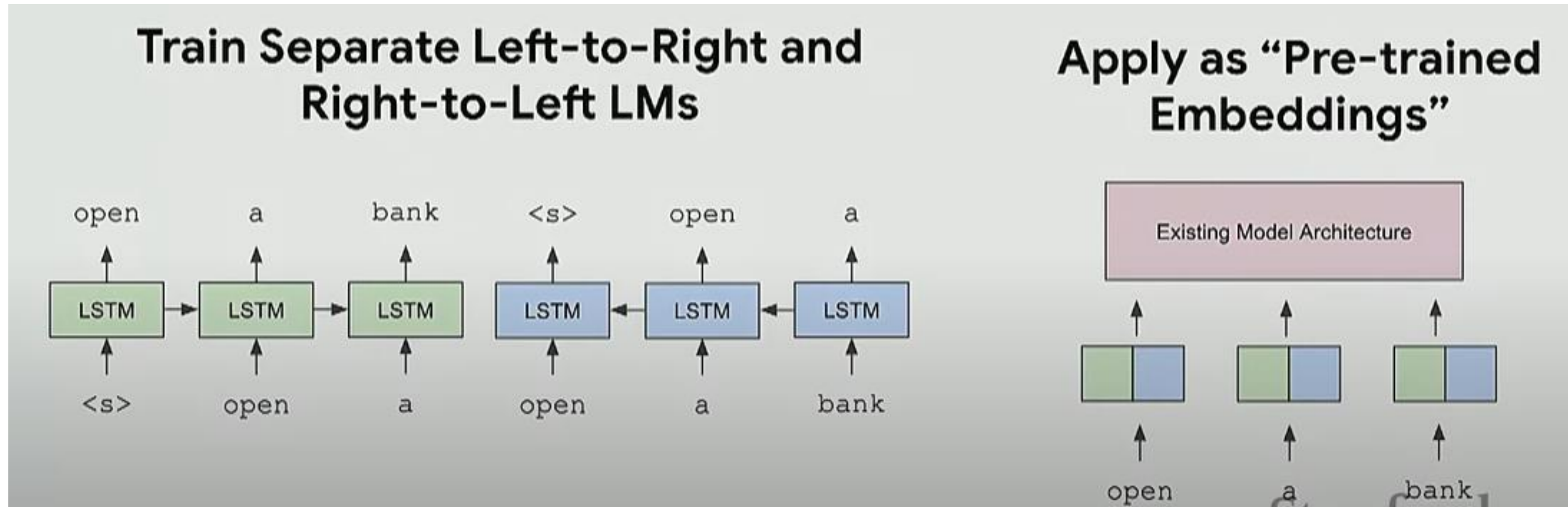


전체 모델을 pretrain한 뒤에 classification을 위해 fine tuning을 진행하는 방식으로
영화 리뷰데이터 감성분석을 진행

충분한 데이터가 없어 좋은 결과를 얻지 못했음

#02. History of Contextual Representation

#2 Elmo(Dep deep contextual word embedding)



ELMO는 큰 언어 corpus를 순방향 역방향 LSTM으로 학습하는 양방향 모델
역방향 언어 모델도 사용함으로써 기존의 GLOVE 등에 있던 워드 임베딩 문제점을 해결

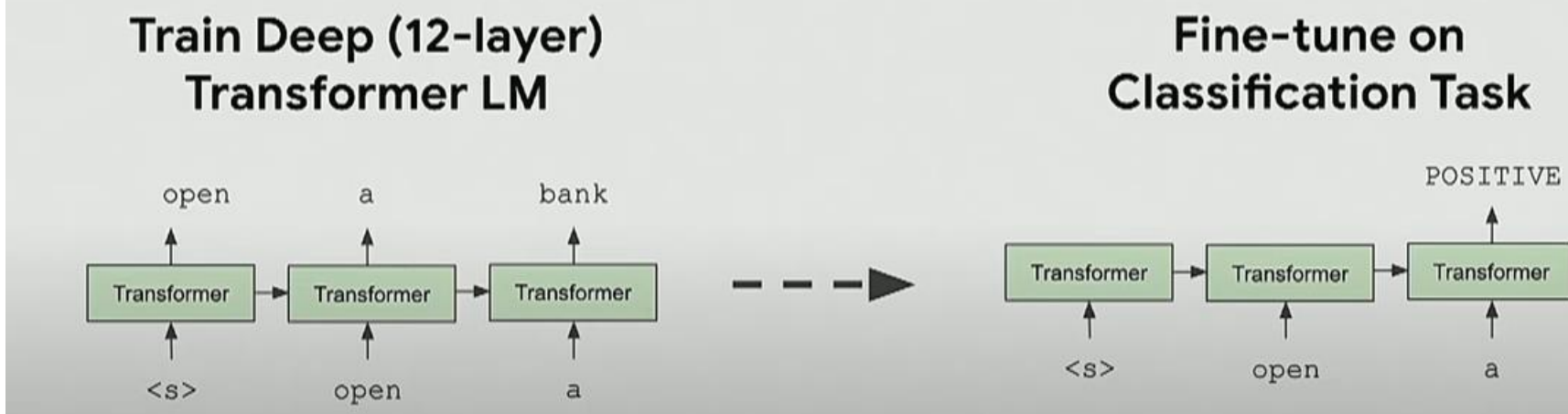
#02. History of Contextual Representation

#3 GPT-1

* Transformer VS LSTM

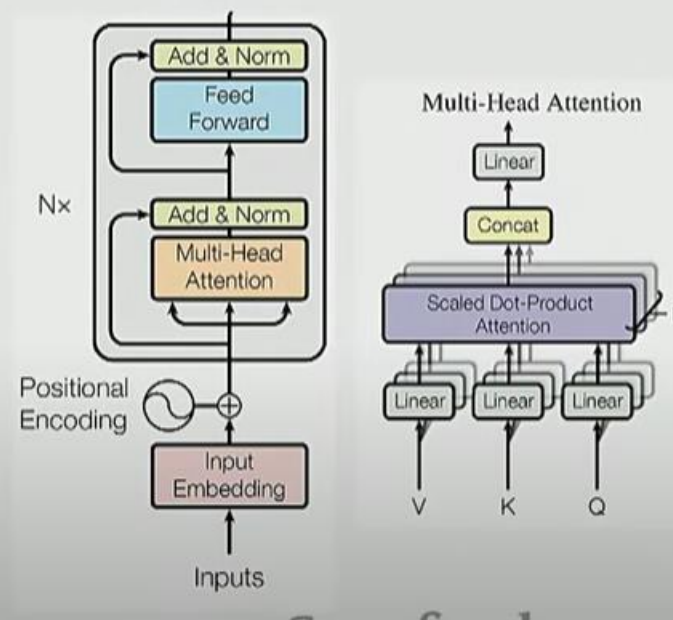
Transformer 등장 -> 언어 모델 학습 시 Transformer를 이용한 연구 발전

- *Improving Language Understanding by Generative Pre-Training*, OpenAI, 2018



Transformer encoder

- Multi-headed self attention
 - Models context
- Feed-forward layers
 - Computes non-linear hierarchical features
- Layer norm and residuals
 - Makes training deep networks healthy
- Positional embeddings
 - Allows model to learn relative positioning



#02. History of Contextual Representation

#3 GPT-1

* Transformer VS LSTM

LSTM은 단어를 순차적으로 입력 받아서 처리

-> 각 단어의 위치 정보를 가질 수 있어 멀리 있는 단어보다 가까이 있는 단어가 관련성이 높다고 판단하는 locality bias 문제가 발생

- Empirical advantages of Transformer vs. LSTM:

1. Self-attention == no locality bias

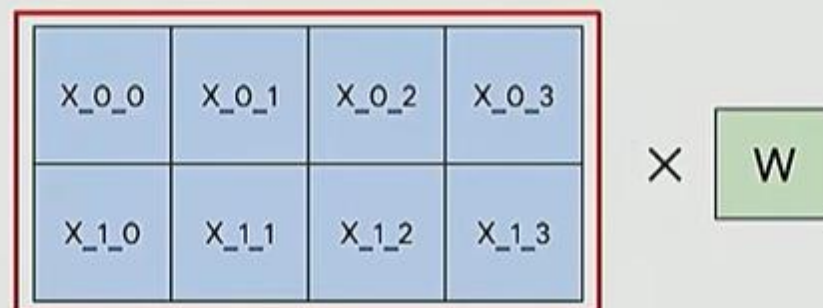
- Long-distance context has "equal opportunity"

transformer는 단어 입력을 순차적으로 받는 것이 아니라 각 단어의 embedding 벡터에 positional encoding을 통해 위치정보를 더하여 locality bias 문제를 해결

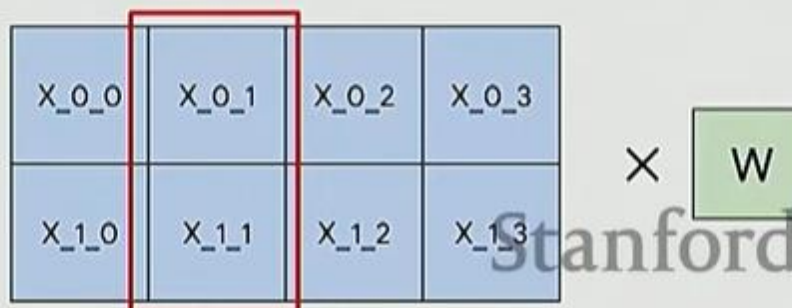
2. Single multiplication per layer == efficiency on TPU

- Effective batch size is number of *words*, not *sequences*

Transformer



LSTM



self-attention에선 맥락을 concatenate함으로써
long-distance context가 "equal opportunity"를 가지도록 함

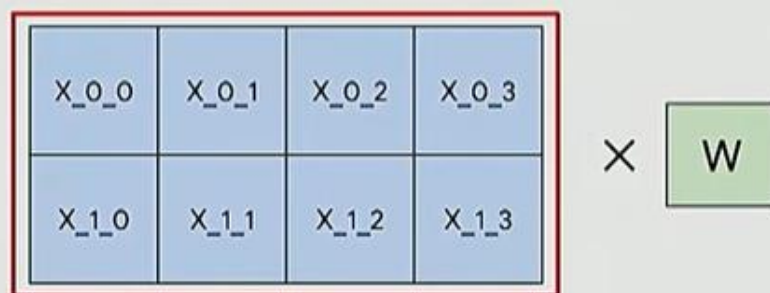
#02. History of Contextual Representation

#3 GPT-1

* Transformer VS LSTM

- Empirical advantages of Transformer vs. LSTM:
 1. Self-attention == no locality bias
 - Long-distance context has “equal opportunity”
 2. Single multiplication per layer == efficiency on TPU
 - Effective batch size is number of *words*, not *sequences*

Transformer



LSTM

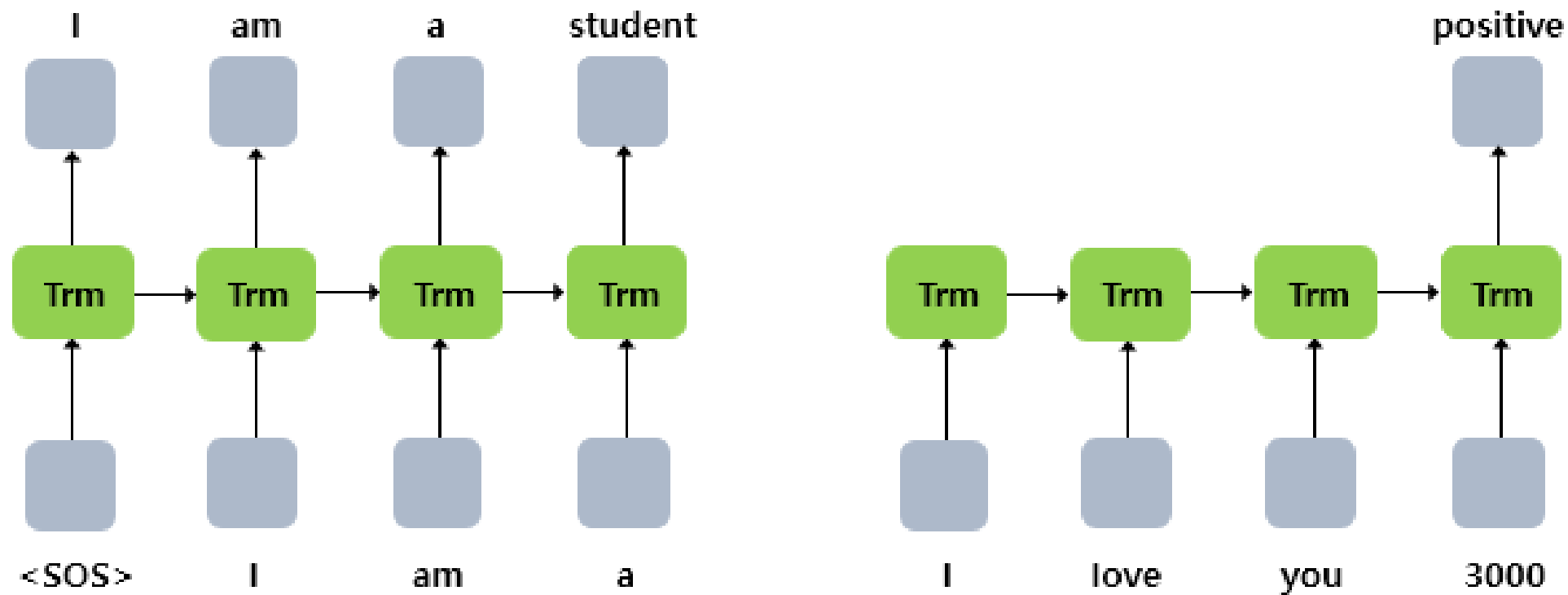


LSTM은 문장(몇개의 과거 데이터를 볼 것인지)을 이용 vs. Transformer는 단어 임베딩을 사용
만약 500개의 단어와 32개의 sentence가 있다면 Transformer는 512*32개의 batch 사이즈를 사용
->TPU와 GPU를 효율적으로 사용 가능

#02. History of Contextual Representation

#3 GPT-1

Improving Language Understanding by Generative Pre-training, OpenAI, 2018



Deep(12-layer) Trm 언어 모델을 사전 훈련

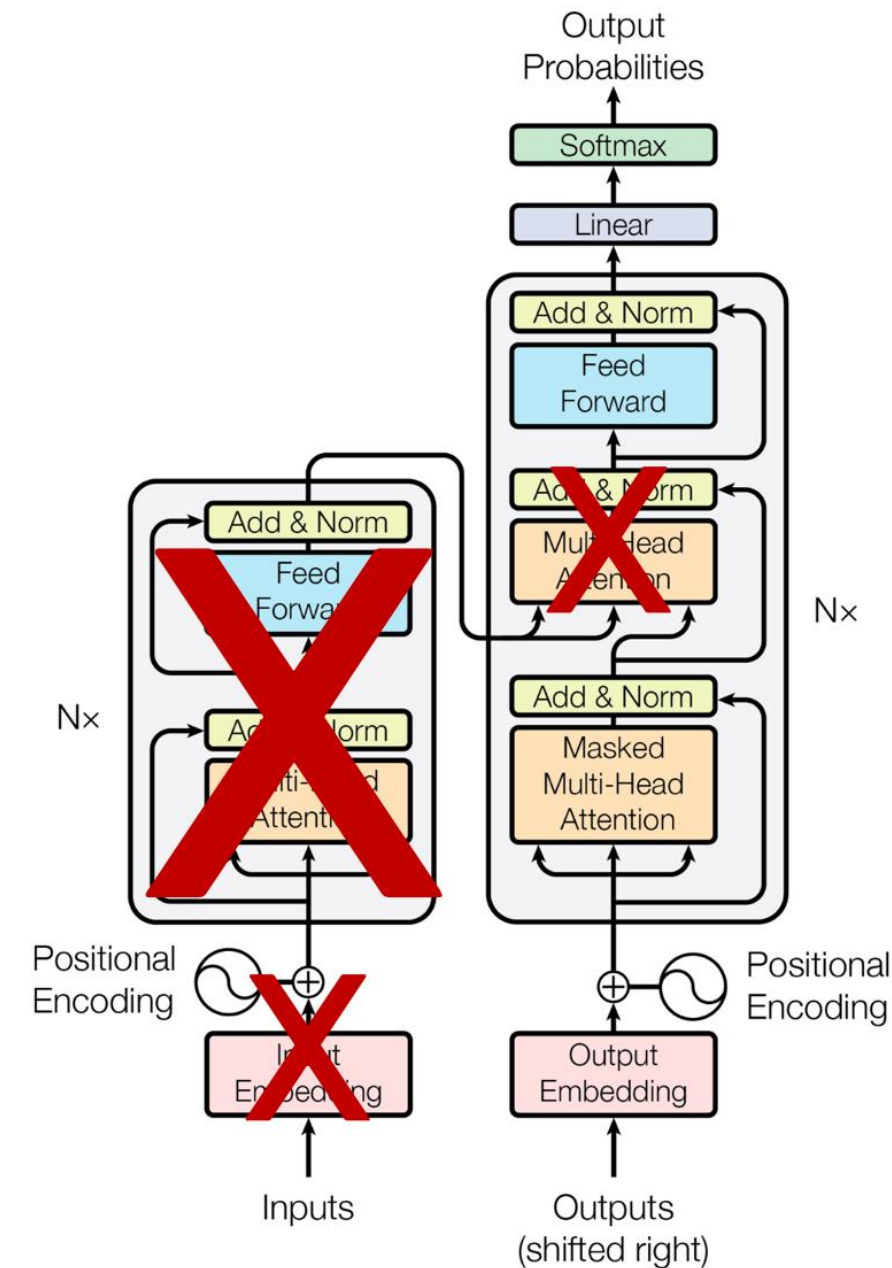
분류 문제에 파인 튜닝

표준 Transformer의 Encoder는 사용하지 않고 Decoder만 사용

- > Decoder에서 Encoder의 출력과 Attention을 하는 부분인 Encoder-Decoder Multi-Head attention 부분을 제거
- > 12개의 transformer 층을 쌓은 후 방대한 텍스트 데이터를 이용하여 GPT-1 모델을 생성

=>

- 1) 큰 말뭉치에서 대용량의 언어모델 학습
- 2) 분류 데이터를 써서 과제에 맞춰 모델을 fine-tuning하는 방식으로 진행 -> 12개 중 9개의 nlp task에서 sota 달성



3. BERT



#03. BERT

#1 problem with previous methods

-1. bidirectional/see themselves

LM은 left, right 맥락으로만 사용, 그러나 언어는 양방향 이해가 필요

- **Problem:** Language models only use left context or right context, but language understanding is bidirectional.
- Why are LMs unidirectional?
- Reason 1: Directionality is needed to generate a well-formed probability distribution.
 - We don't care about this.
- Reason 2: Words can “see themselves” in a bidirectional encoder.

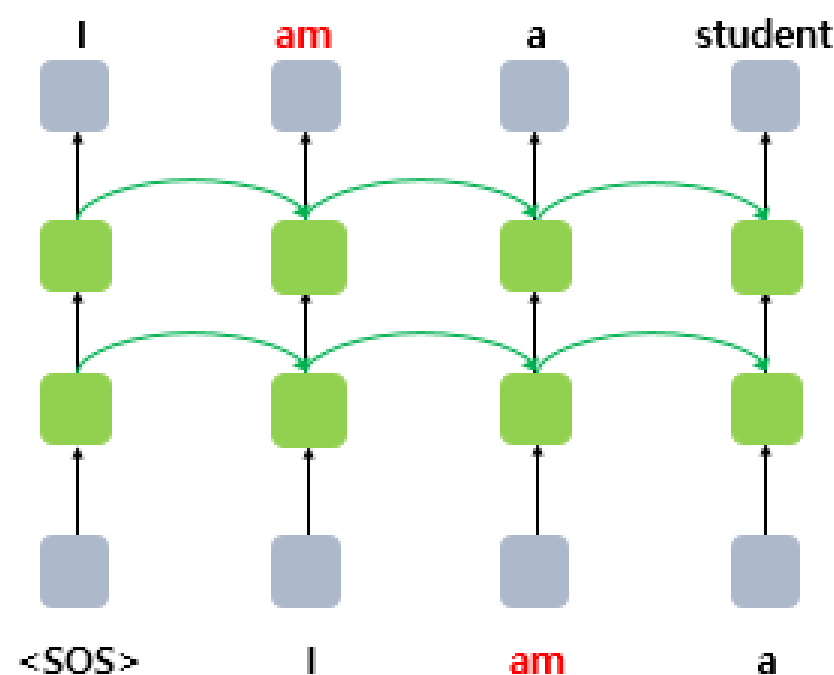
Stanford

#03. BERT

#1 problem with previous methods

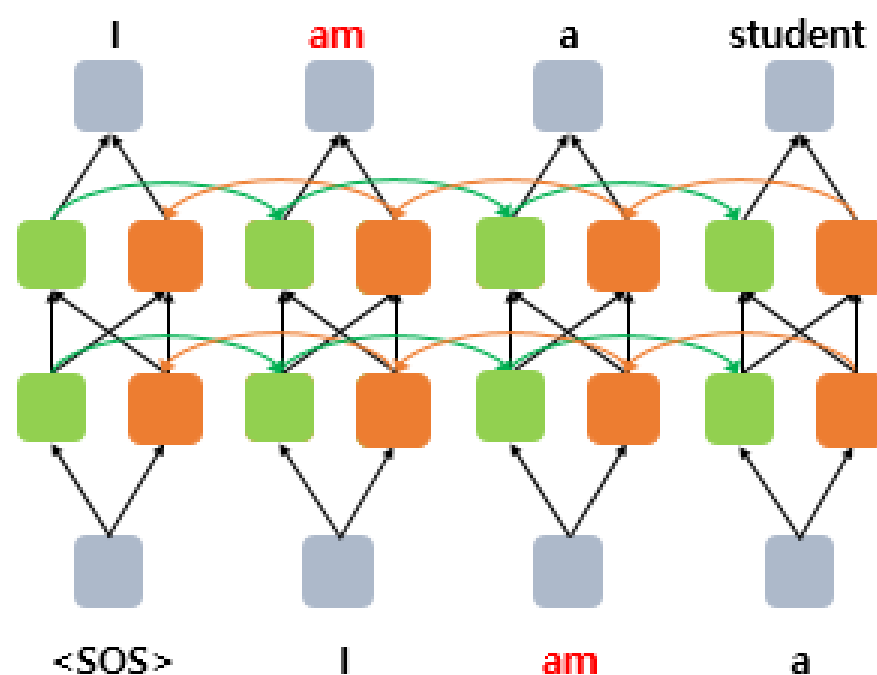
-1. bidirectional

단방향 언어모델



순차적으로 단어를 생성한다.

양방향 언어모델



단어들이 자기 자신을 본다.

- 확률 분포를 잘 형성하기 위해서는 방향성이 필요
- 양방향 encode에서는 단어가 자기를 볼 수 있음

양방향 LM은 am을 예측할 때 순방향 모델의 정보(I,)와 역방향 모델의 정보(a, am, i)를 가짐.
즉, am을 예측하기 위해 am의 정보를 가지고 있는 상황이 발생

#03. BERT

#2 문제를 해결한 BERT : Masked LM → 양방향 학습의 문제를 해결하기 위해서 Masked LM이 탄생

Masked LM

- **Solution:** Mask out $k\%$ of the input words, and then predict the masked words
 - We always use $k = 15\%$

the man went to the [MASK] to buy a [MASK] of milk

store gallon

↑ ↑

masking 개수가 적으면 학습하는데 많은 비용이 들고, 너무 많다면 맥락의 수가 적어져 예측이 어려움.

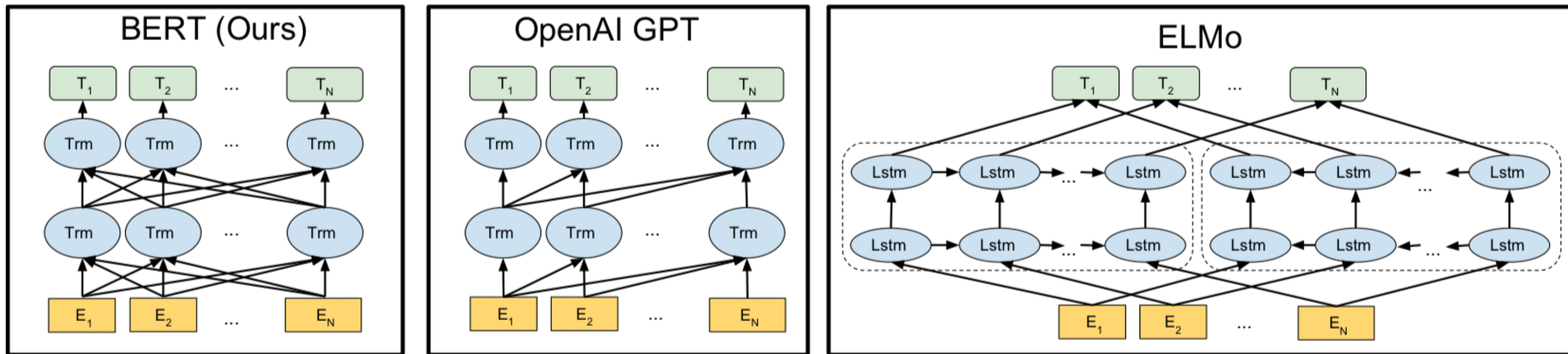
- Too little masking: Too expensive to train
- Too much masking: Not enough context

Masked LM이란, 입력 텍스트 단어 집합의 15%의 단어를 랜덤으로 masking하고 masking된 단어를 예측하는 방법

#03. BERT

#2 문제를 해결한 BERT : Masked LM

* Bert vs GPT vs ELMo



- ELMo는 정방향/역방향 LSTM을 각각 훈련시키고 합쳐 양방향 언어 모델을 생성
- GPT-1은 transformer의 decoder를 이전 단어로 다음 단어를 예측하는 단방향 언어모델을 생성
- Bert는 GPT와 달리 Masked LM을 사용하여 양방향 학습을 사용하는 모델

#03. BERT

#3 Bert pretraining 방법, input, procedure

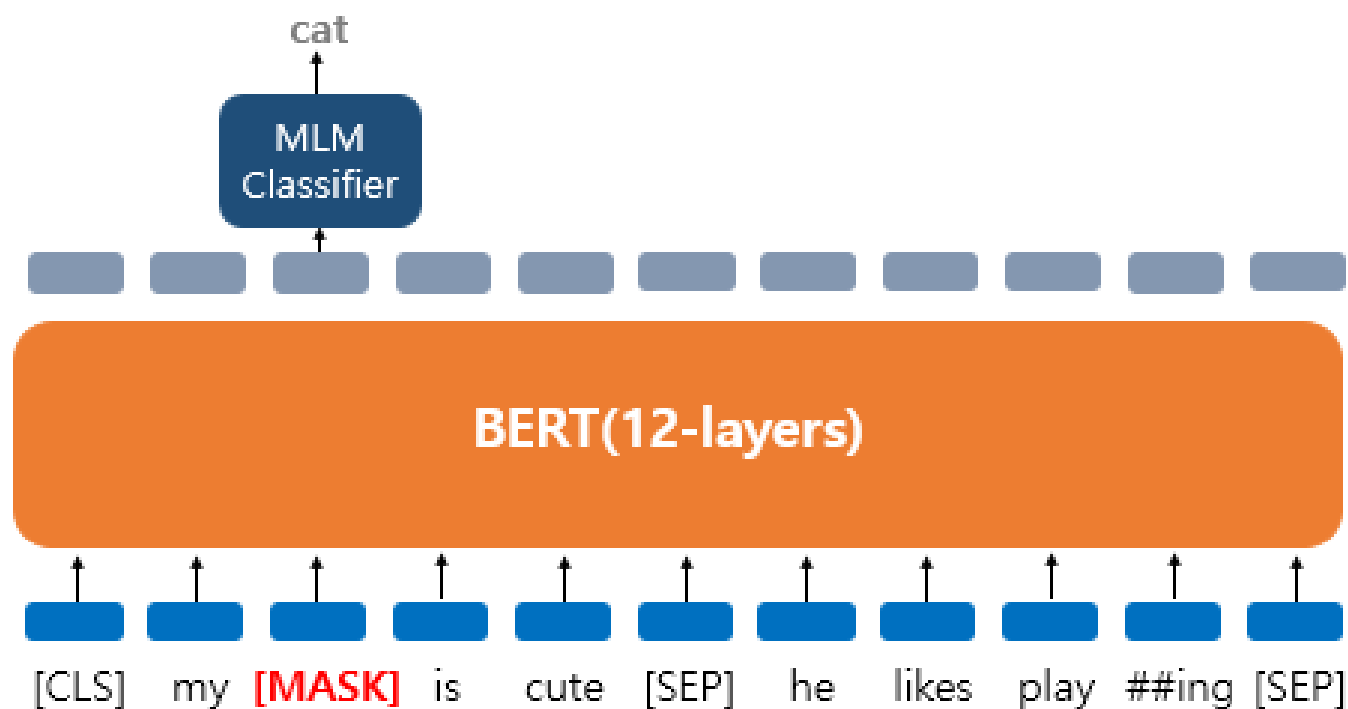
-1. masked language model(MLM)

- Problem: Mask token never seen at fine-tuning
- Solution: 15% of the words to predict, but don't replace with [MASK] 100% of the time. Instead:
 - 80% of the time, replace with [MASK]
went to the store → went to the [MASK]
 - 10% of the time, replace random word
went to the store → went to the running
 - 10% of the time, keep same
went to the store → went to the store **Stanford**

Bert는 사전 훈련을 위해 신경망 input으로 들어가는 입력 text의 15%를 랜덤으로 masking하고 masking을 예측

#03. BERT

#3 Bert pretraining 방법, input, procedure
-1. masked language model(MLM)



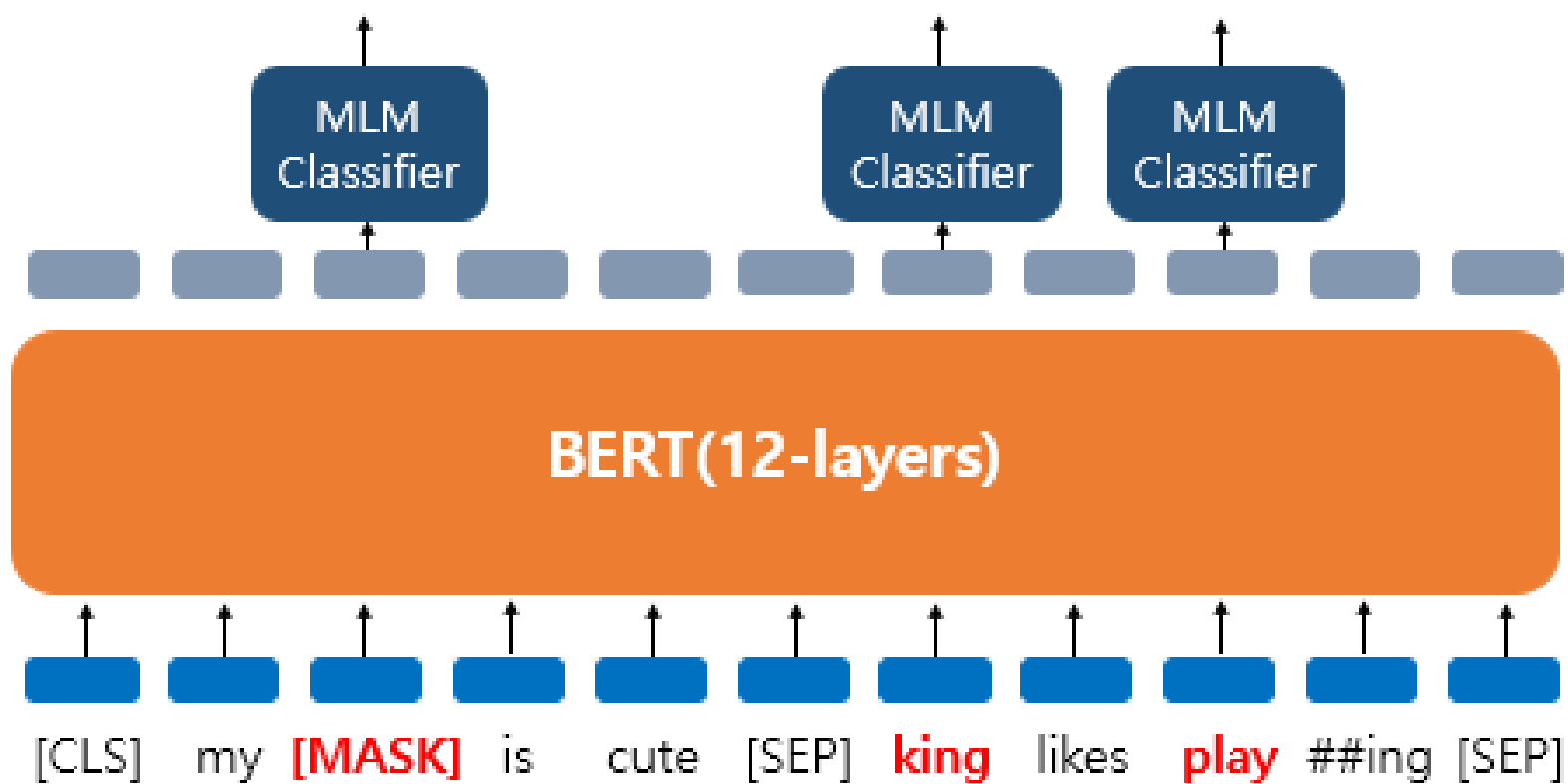
My dog is cute. He likes playing이라는 문장에 대해 masked language 모델을 학습하려고 할 때 [my, dog, is, cute, he, likes, play, ##ing]로 토큰화가 되어 BERT 입력으로 사용됨

여기서 dog가 [mask]되었는데 Bert 모델이 [mask]된 단어를 맞추려고 하고 이때 dog 위치의 출력층 벡터만 사용

#03. BERT

#3 Bert pretraining 방법, input, procedure

-1. masked language model(MLM)



[Mask] 토큰만 사용하면
mask token이 fine tuning 단계에서 나타나지 않음.

이를 해결하기 위해

15%의 80%는 [mask]로

EX) went to the store -> went to the [mask]

15%의 10%는 랜덤으로 단어 변경

EX) went to the store -> went to the running

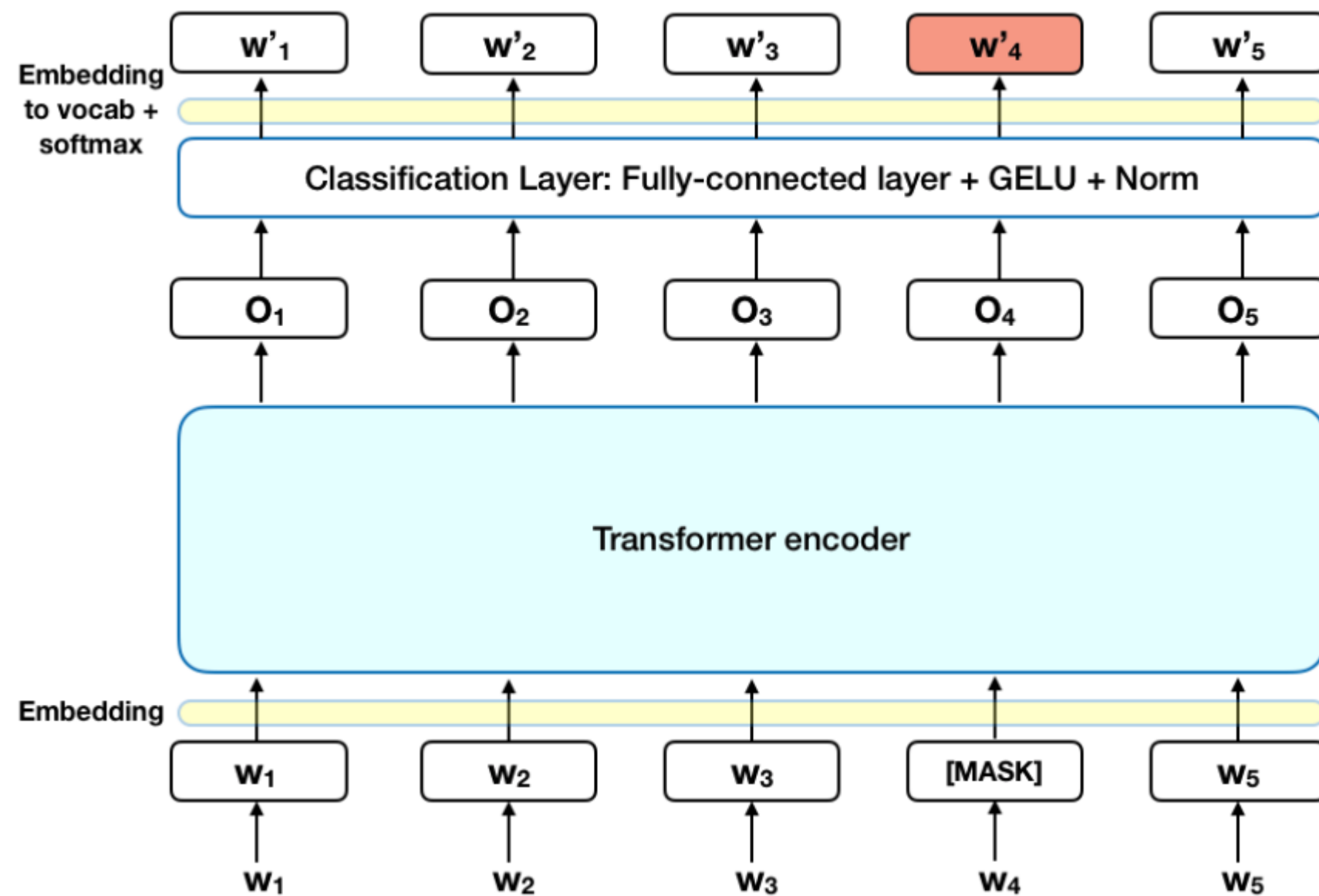
15%의 10%는 동일하게

EX) went to the store -> went to the store

여기서는 He -> king / play를 play 그대로 사용, [mask], 'king', 'play'에서도 원래 단어를 예측

#03. BERT

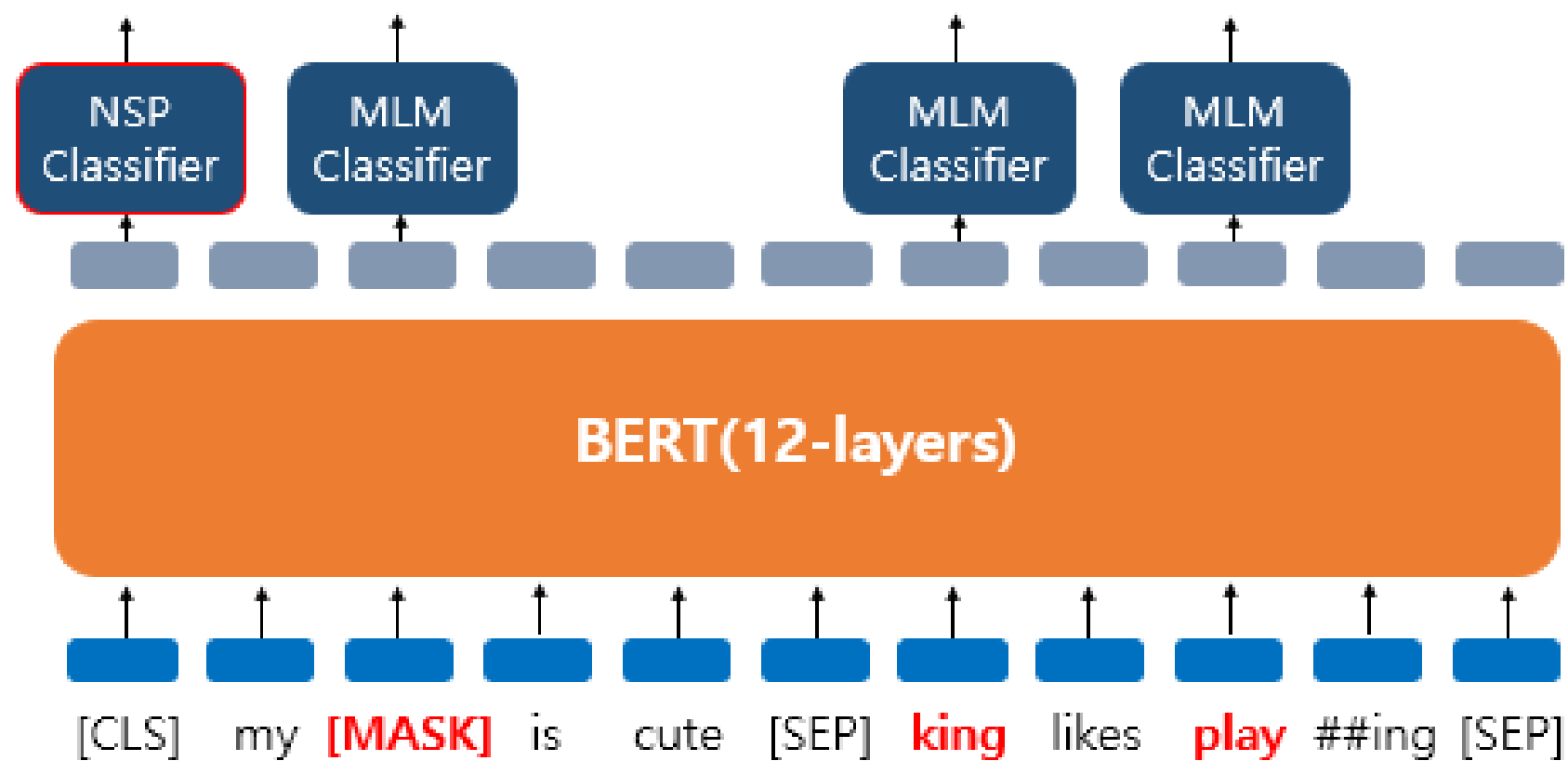
#3 Bert pretraining 방법, input, procedure -1. masked language model(MLM)



따라서, Bert에서는 input과 masked된 token을 Transformer encoder에 넣고 token을 예측하므로 양방향 학습을 함.

#03. BERT

#3 Bert pretraining 방법, input, procedure
-2. Next sentence prediction(NSP)



QAL나 Natural language inference와 같이 두 문장 사이의 관계를 이해하도록 두 문장을 이어서 맞추는 것
pre-training 시에는 50:50 비율로 실제 이어지는 문장과 랜덤한 문장을 넣어서 Bert가 맞추도록 함.

#03. BERT

#3 Bert pretraining 방법, input, procedure -2. Next sentence prediction(NSP)

- To learn *relationships* between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence

Sentence A = The man went to the store.
Sentence B = He bought a gallon of milk.
Label = IsNextSentence

Sentence A = The man went to the store.
Sentence B = Penguins are flightless.
Label = NotNextSentence

Bert의 입력에 [Sep]라는 토큰을 넣어 문장을 구분

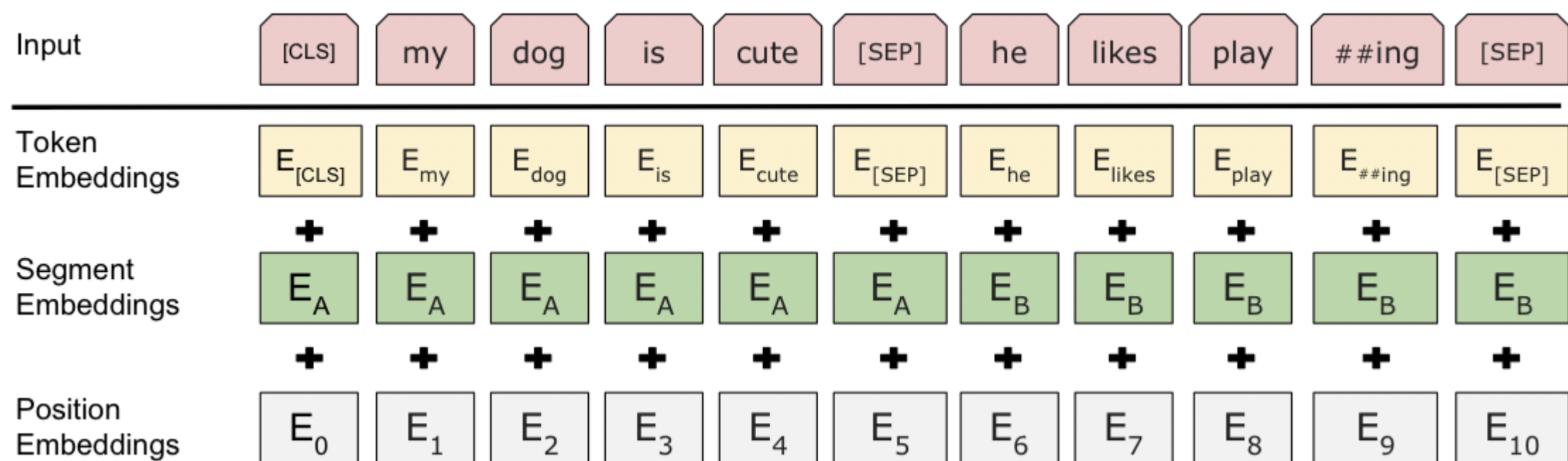
문장 분류 문제를 위해 [CLS] 토큰을 추가하여 [CLS]의 출력층에서 NSP 분류 문제를 해결
이때 NSP와 MLM은 loss를 합하여 학습이 이루어진다.

#03. BERT

#3 Bert pretraining 방법, input, procedure

- 3. Bert pretraining의 input

Bert의 input은 3가지 embedding 값의 합으로 이루어짐



Token Embedding : WordPiece Embedding 사용. Embedding 벡터의 종류는 단어 집합의 크기

* WordPiece : 단어보다 더 작은 단위로 쪼개는 tokenizer

자주 등장하는 단어는 단어집합에 추가, 자주 등장하지 않는 단어는 더 작은 단위의 서브워드로 분리

->서브 워드들이 단어집합에 추가

해당 토큰의 첫번째 서브워드를 제외한 나머지 서브워드들은 앞에 ##을 붙인 것을 토큰으로 함.

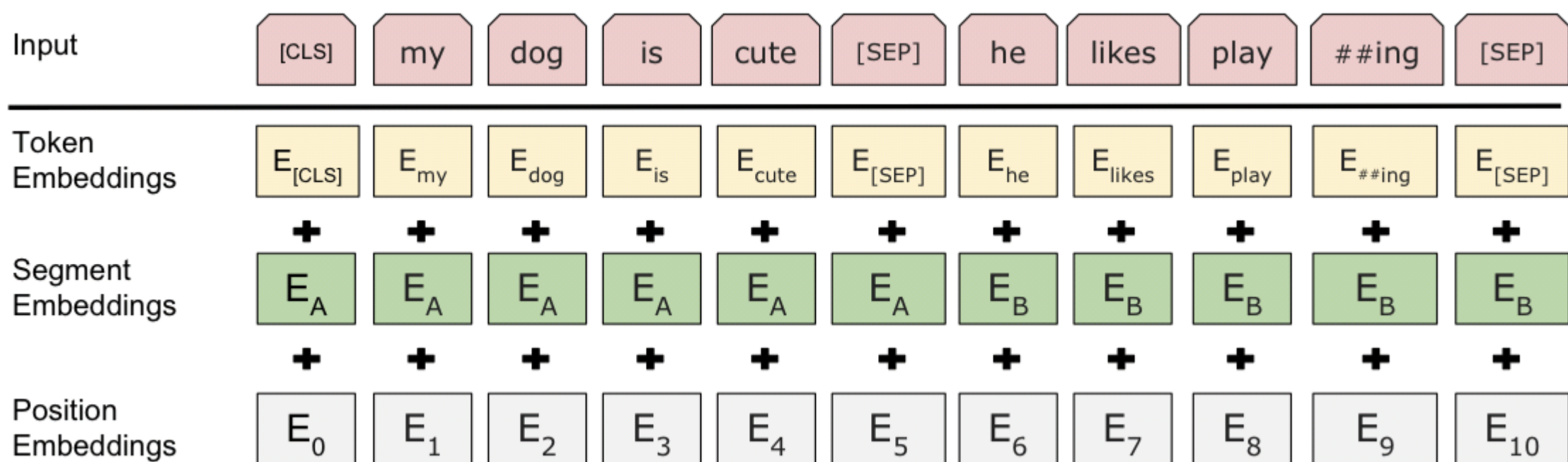
위의 예시인 Playing의 경우 Play, ###ing로 분리 되었다. 이때 ###은 단어의 중간부터 등장하는 서브워드라는 것을 알리기위해 표시해둔 기호. 단어 집합을 기반으로 토큰화를 수행.

#03. BERT

#3 Bert pretraining 방법, input, procedure

- 3. Bert pretraining의 input

Bert의 input은 3가지 embedding 값의 합으로 이루어짐



Token Embedding : WordPiece Embedding 사용. Embedding 벡터의 종류는 단어 집합의 크기

Segment Embedding : QA 등과 같은 두개의 문장 입력이 필요한 task를 풀 때 segment embedding 사용.
만약 문장이 하나면?? Sentence A embedding만 사용

Position Embedding : 위치 정보를 학습하기 위한 Embedding. Embedding 벡터의 종류는 문장의 최대 길이인 512개, Transformer에서 positional encoding 방법과 같음.

#03. BERT

- #3 Bert pretraining 방법, input, procedure
- 3. Bert pretraining의 procedure

Model Details

- Data: Wikipedia (2.5B words) + BookCorpus (800M words)
- Batch Size: 131,072 words (1024 sequences * 128 length or 256 sequences * 512 length)
- Training Time: 1M steps (~40 epochs)
- Optimizer: AdamW, 1e-4 learning rate, linear decay
- BERT-Base: 12-layer, 768-hidden, 12-head
- BERT-Large: 24-layer, 1024-hidden, 16-head
- Trained on 4x4 or 8x8 TPU slice for 4 days Stanford

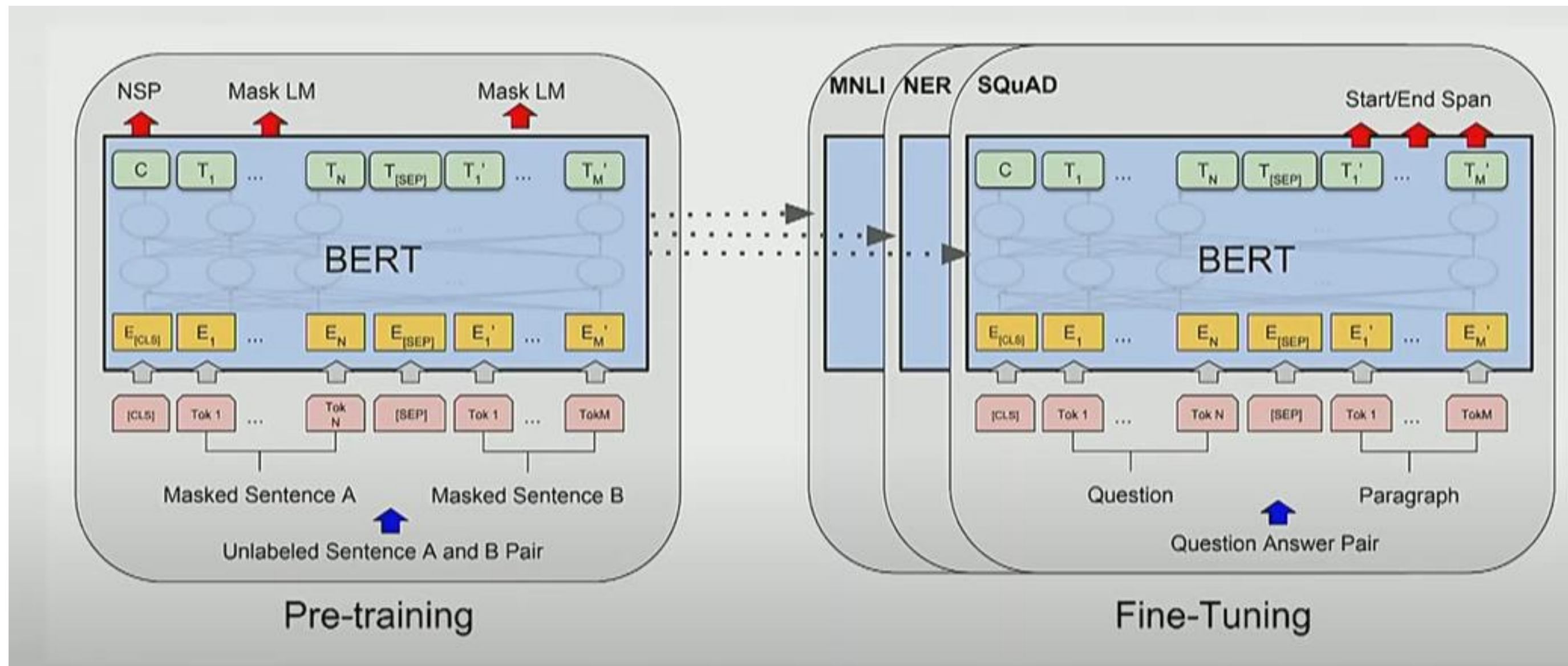
1단계 : 위키피디아, book corpus 데이터 이용

2단계 : NSP를 위해 sentence를 뽑아서 Sentence embedding을 넣음
(이때, 50%는 진짜 sentence, 나머지는 random sentence)

3단계: masking 작업을 하고 masking 예측

#03. BERT

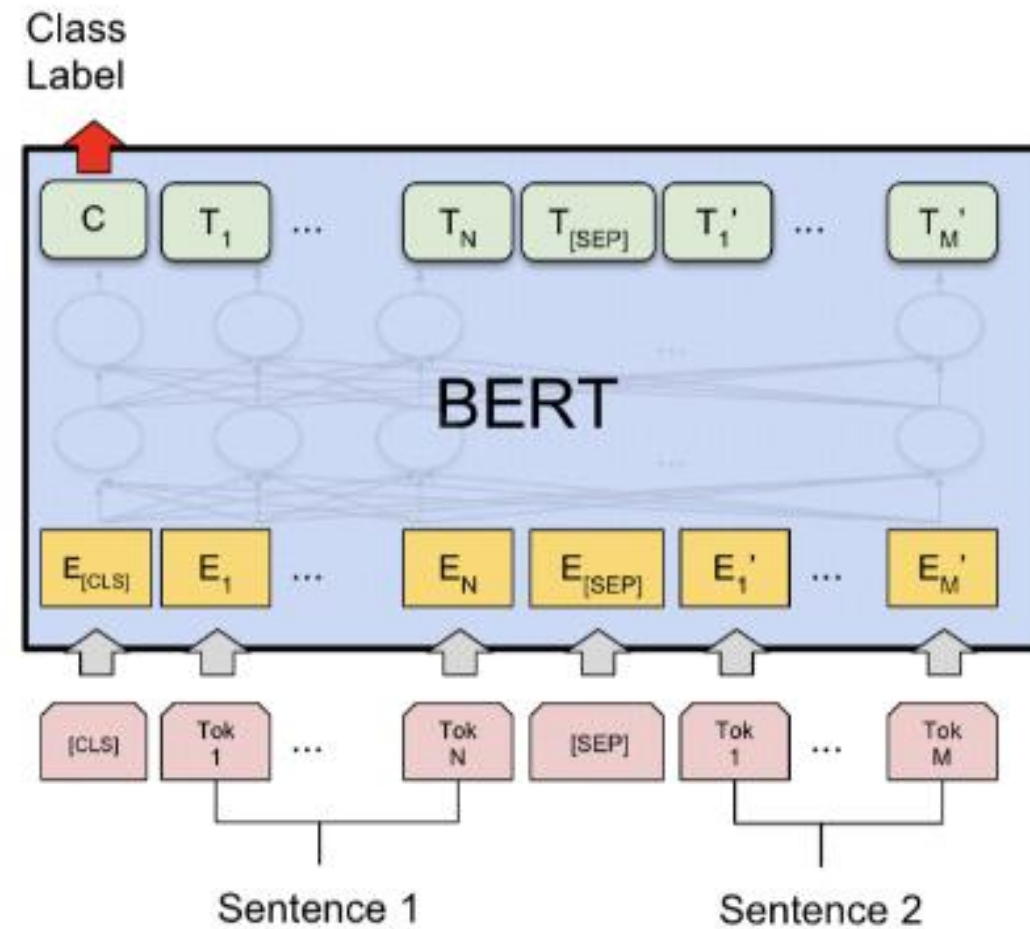
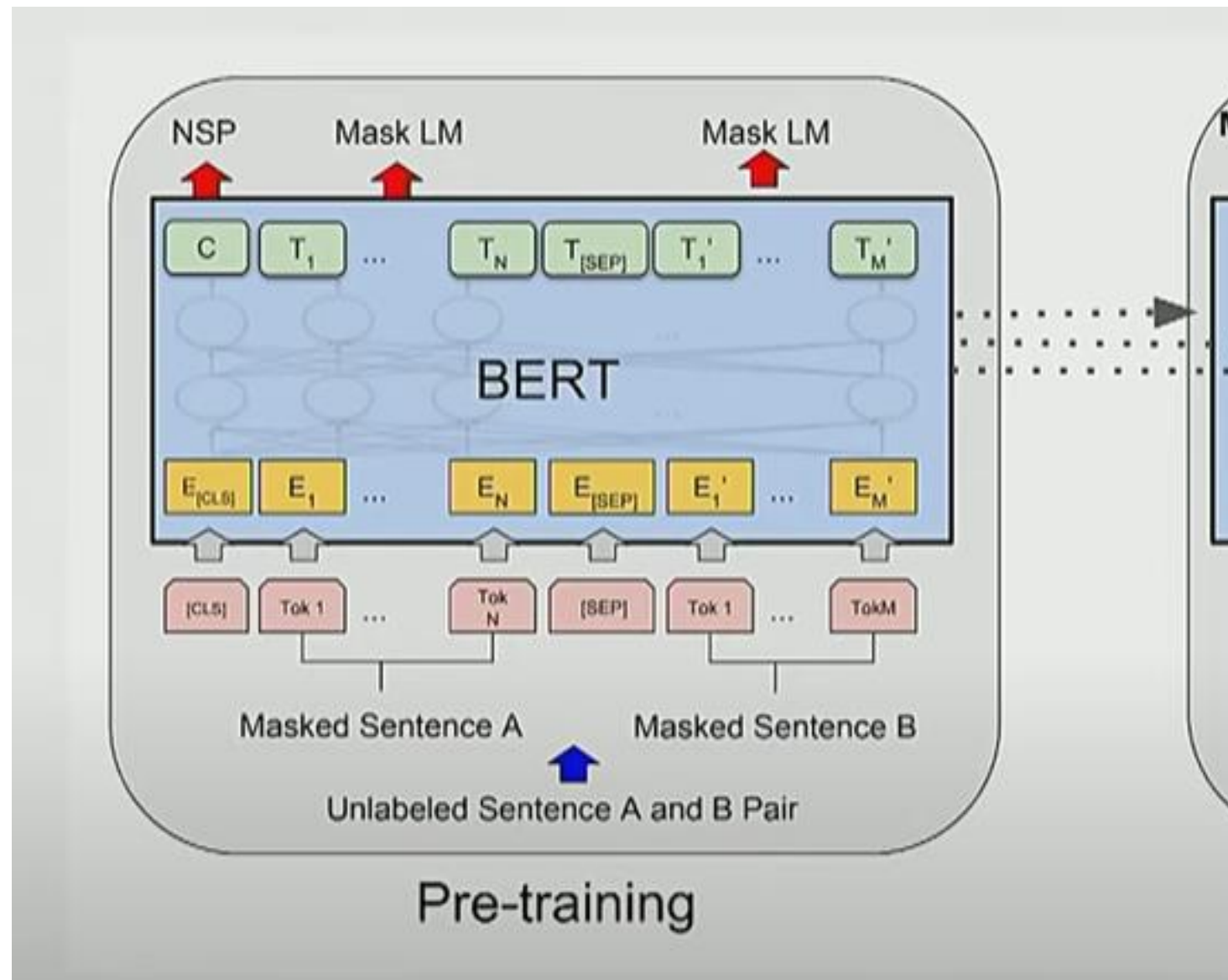
#4 Bert fine tuning procedure



#03. BERT

#4 Bert fine tuning procedure - **sequence-level task**

- 1. Sentence Pair Classification Tasks : 텍스트 쌍에 대한 분류 문제

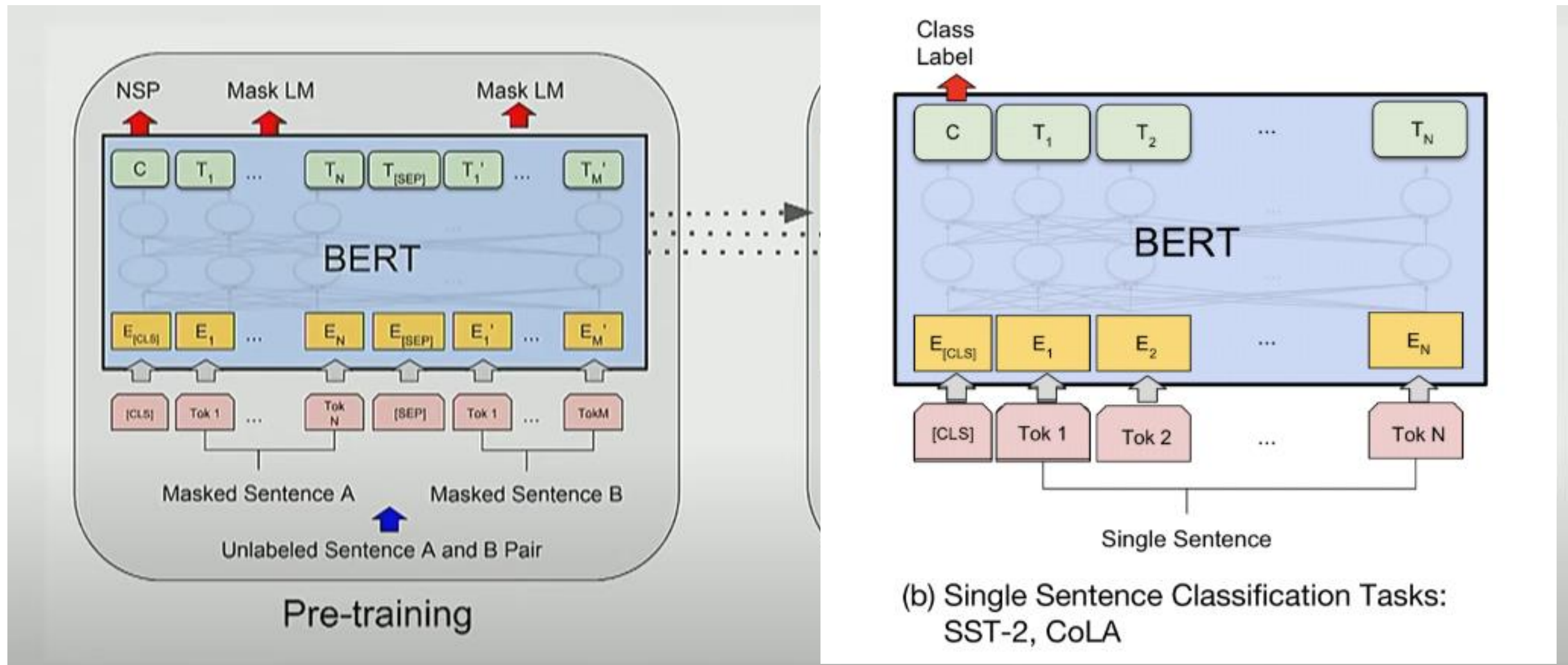


task: NLI(자연어 추론) – 두 문장이 주어졌을 때 하나의 문장이 다른 문장과 어떤 관계가 있는지 추론
입력 텍스트가 1개가 아니므로 text 사이에 [sep]토큰을 넣고 두 종류의 segment embedding 사용

#03. BERT

#4 Bert fine tuning procedure - **sequence-level task**

- 2. single sentence Classification Tasks : 하나의 텍스트에 대한 텍스트 분류 유형



영화 리뷰 감성분류, 뉴스 분류 등 입력한 문서에 대해서 분류를 하는 유형으로 [CLS] 토큰을 사용하여 토큰의 위치 출력층에서 Dense layer 또는 FC layer를 추가하여 분류에 대한 예측 실행

#03. BERT

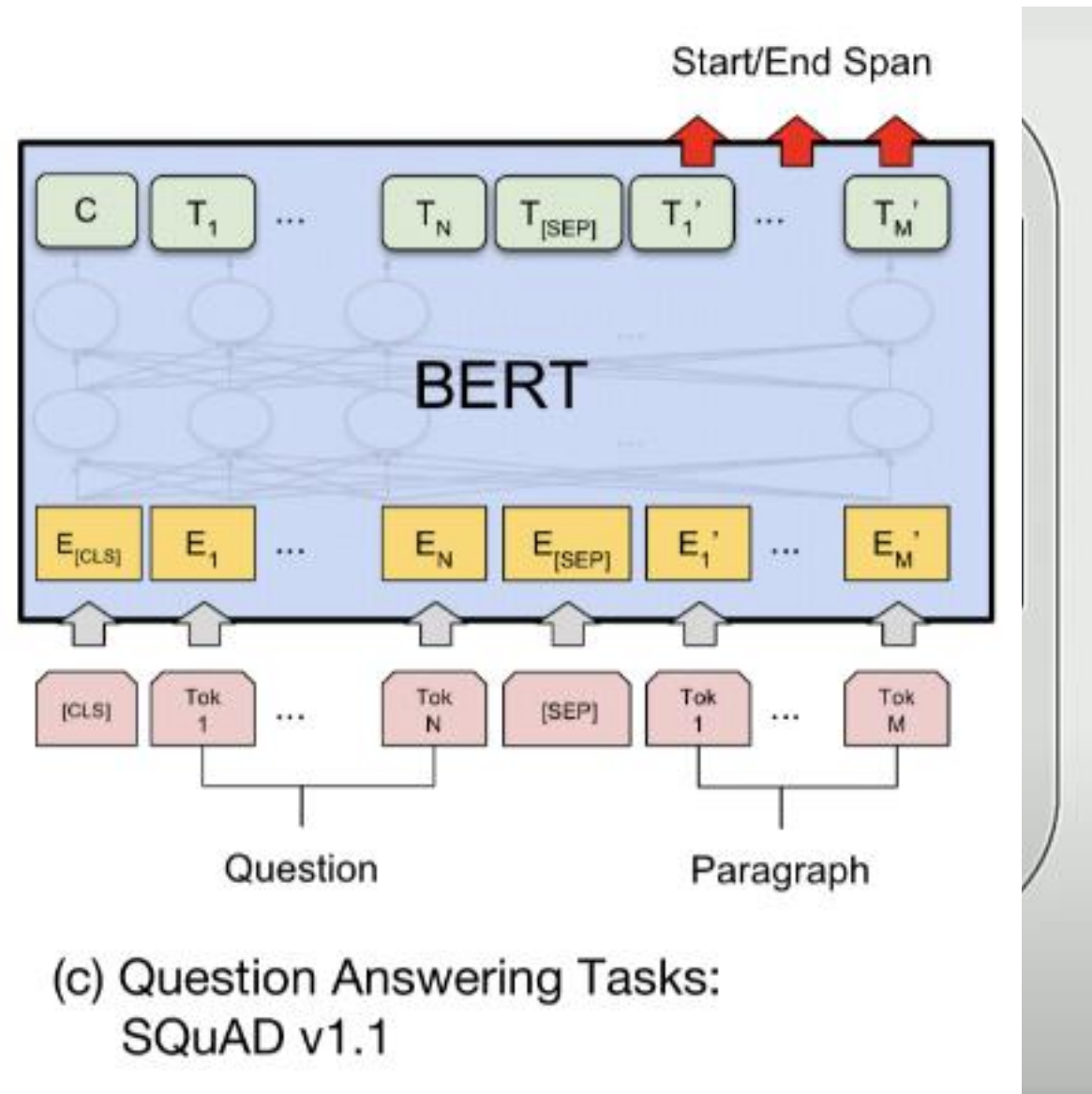
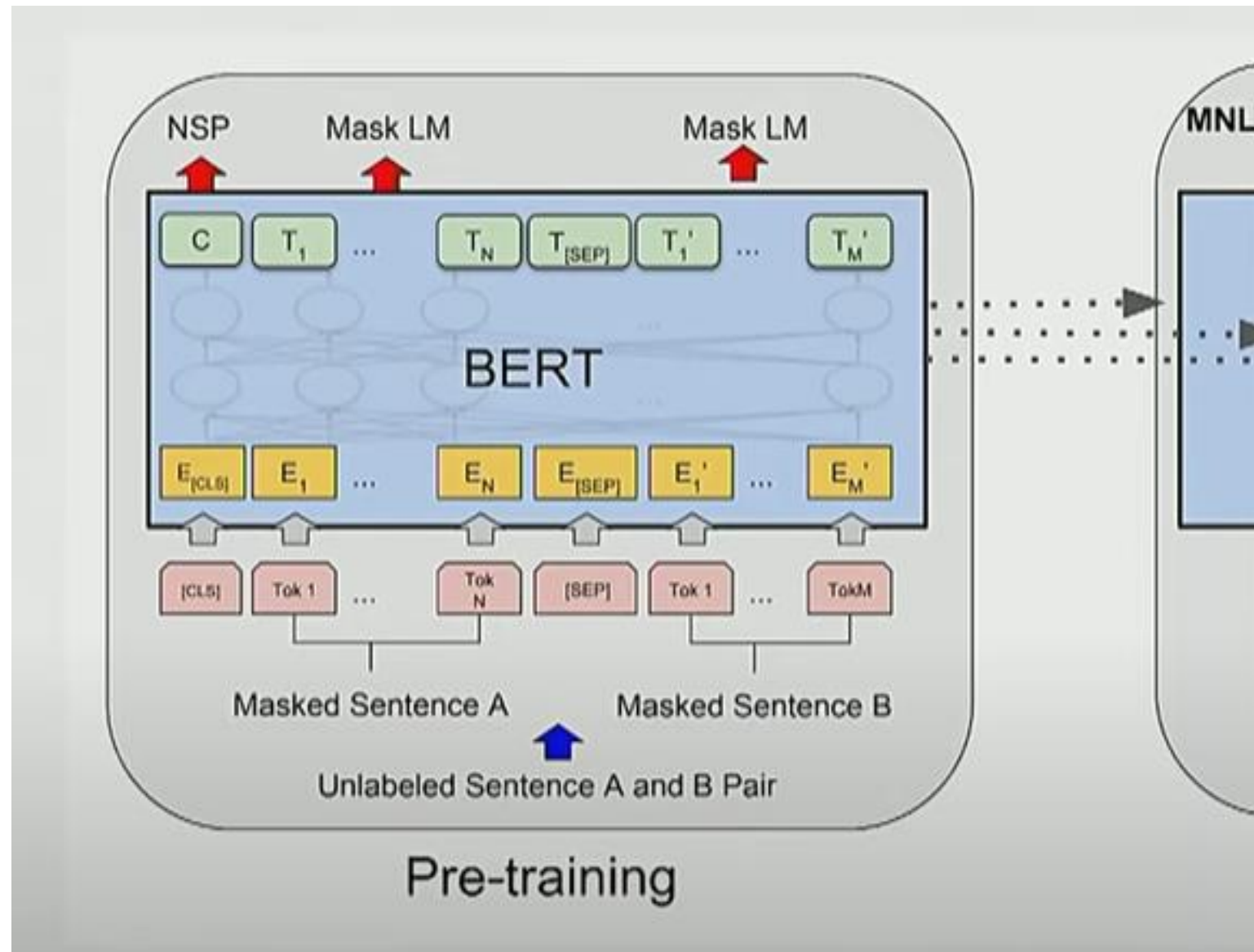
#4 Bert fine tuning procedure - **sequence-level task**

- Sequence-level task에 대한 BERT-fine tuning과정

- 1.[CLS] token의 output값을 사용하고
- 2.이때 [CLS] token의 벡터는 H차원(hidden size)
- 3.classify하고 싶은 K에 따라 classification layer를 붙여 $K \times H$ 의 classification layer를 만듦
- 4.softmax를 통과하여 label probabilities 도출

#03. BERT

- #4 Bert fine tuning procedure - **token-level task**
- 3. Question Answering Tasks : 질의응답

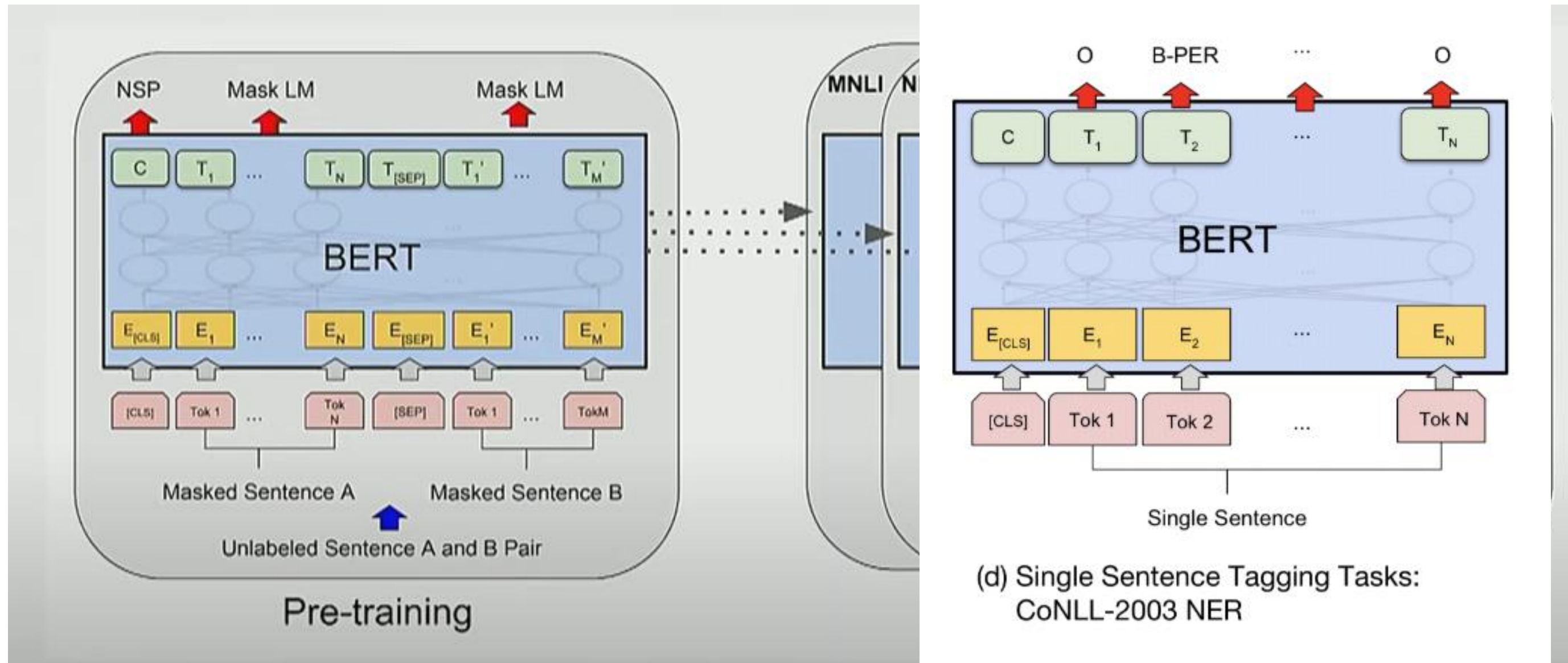


task: 텍스트의 쌍을 입력 받는 QA
질문과 본문을 입력 받으면 본문의 일부를 추출해서 질문에 답변하는 것
token들에서 Start/end span을 찾아냄.

#03. BERT

#4 Bert fine tuning procedure - **token-level task**

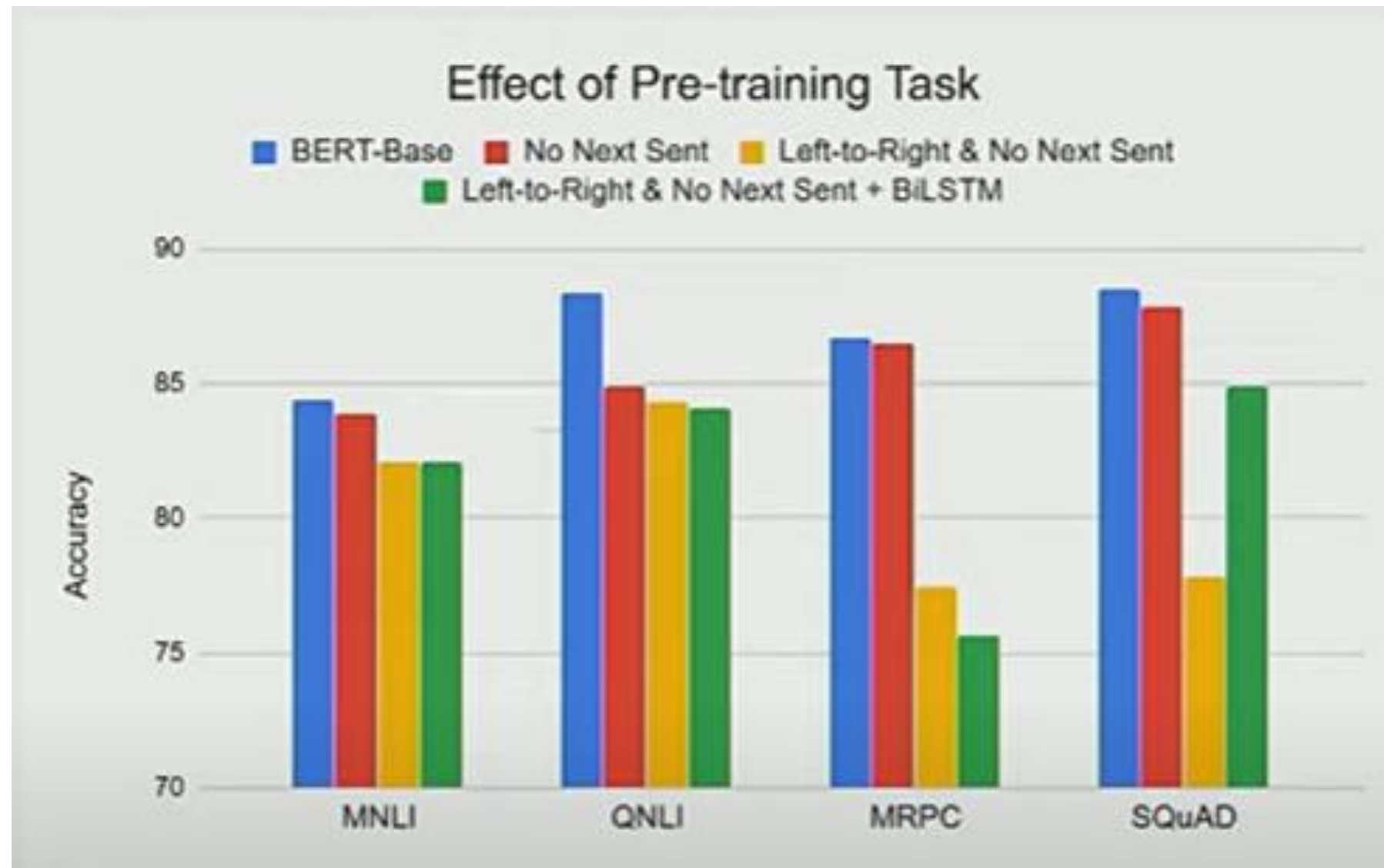
- 4. single Sentence Tagging Tasks : 하나의 텍스트에 대한 태깅 작업



Named entity Recognition(NER)이나 형태소 분석과 같이 single sentence에서 각 토큰이 어떤 class를 갖는지 모두 classifier적용

#03. BERT

#5 Effect of pre-training task

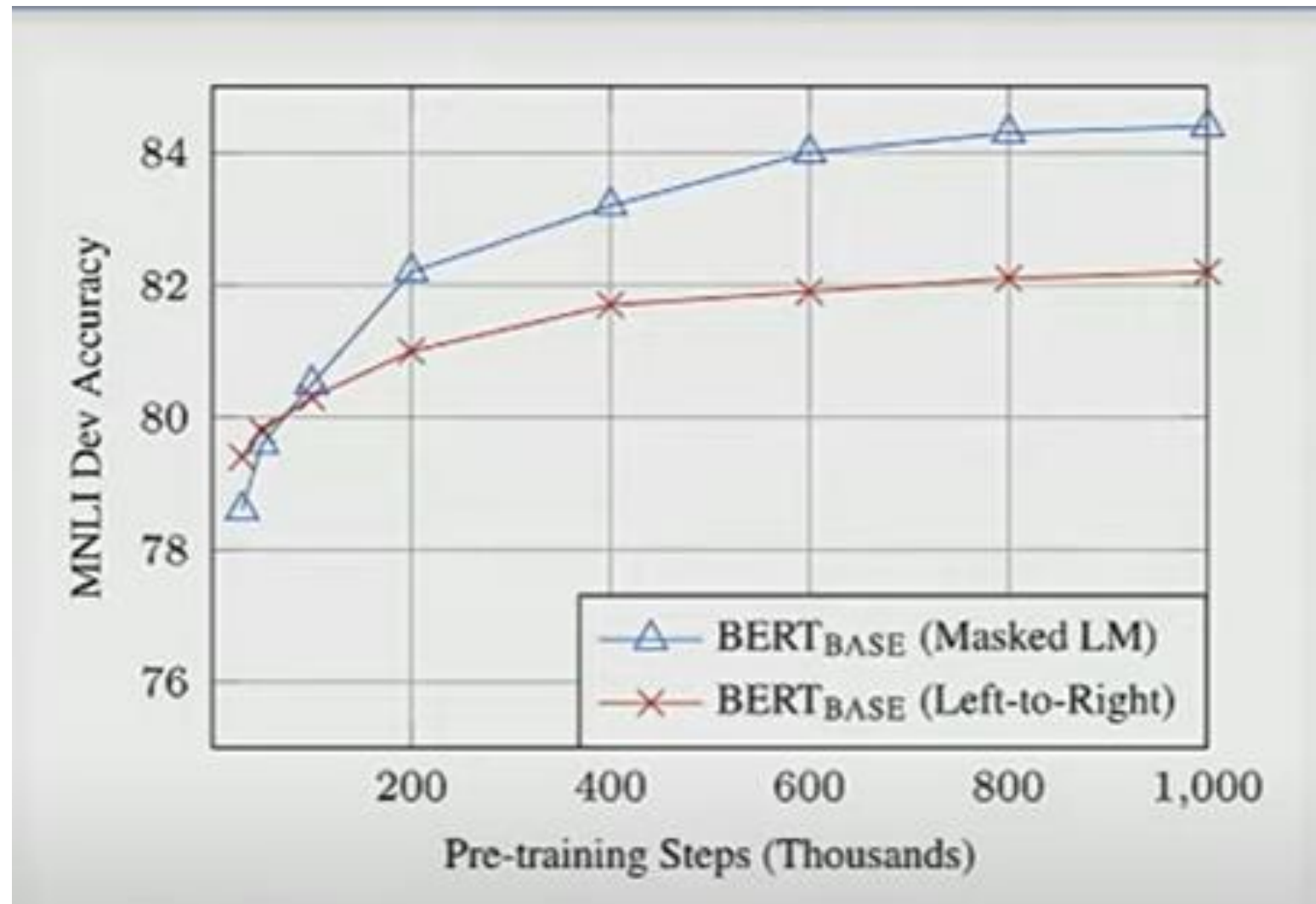


- Pre-training task가 하나라도 제거되는 경우 성능이 떨어진다
- NSP는 **문장간 논리적 구조 파악에** 중요한 역할을 하기 때문에 NO NSP의 경우 NLI에서 성능이 떨어지는 것을 확인할 수 있다
- left-to-right 모델은 단어수준 작업인 SQuAD에서 열악하지만 **BiLSTM에 의해 완화된다**
- MLM 대신 left-to-right를 사용하면 BiLSTM을 사용하더라도 결과적으로 성능 저하를 보인다.

#03. BERT

#5 Effect of pre-training task

- Effect of Directionality and Training Time

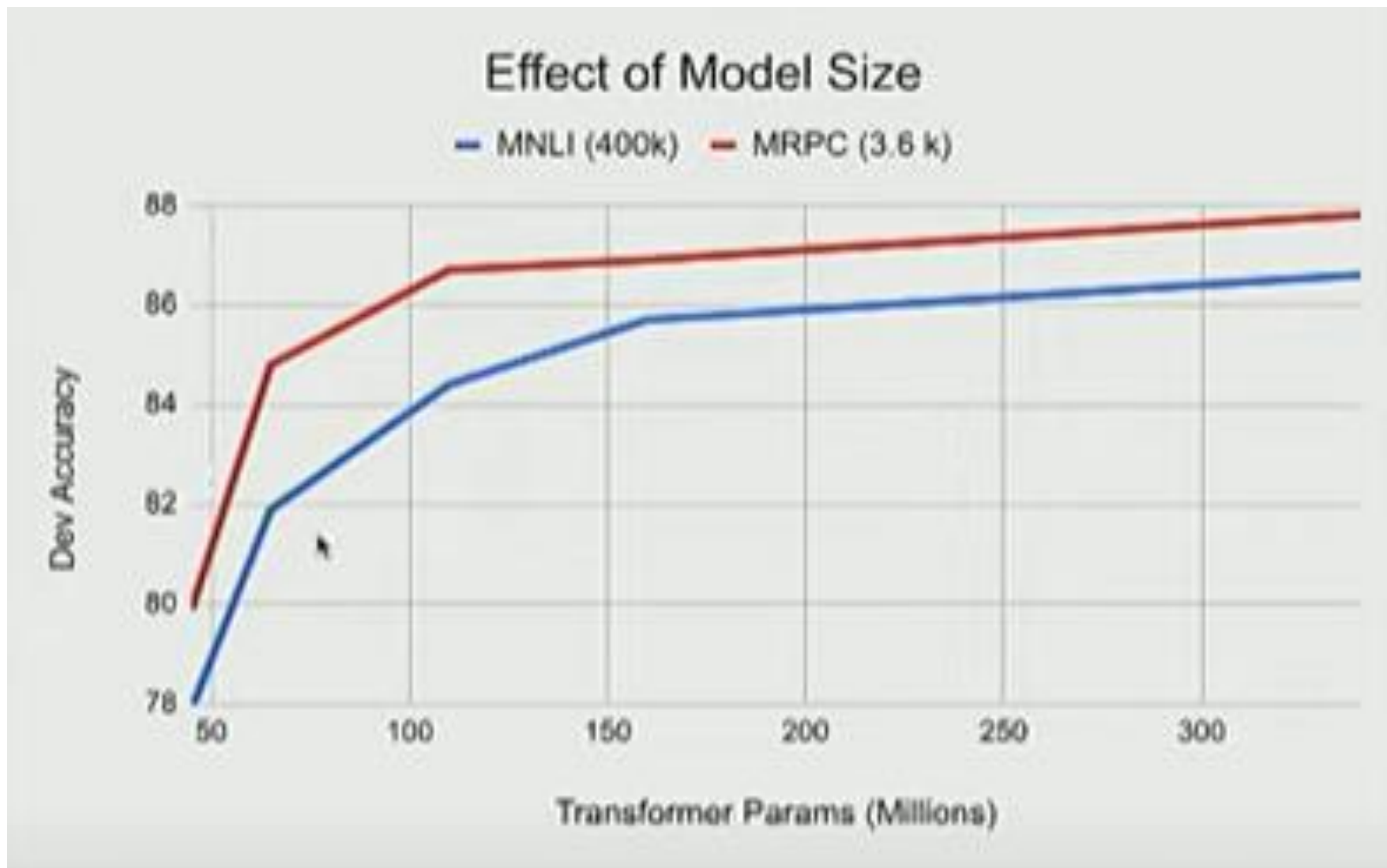


- MLM은 100%가 아닌 15%만 예측하기 때문에 수렴하는데 더 많은 시간이 걸린다
- 그러나 결과는 훨씬 좋다

#03. BERT

#5 Effect of pre-training task

- Effect of Model Size



- 모델의 크기가 클수록 정확도 증가
- 두 그래프는 비슷한 추이를 보임
- 결과적으로 두 그래프 사이에 눈에 띄는 변화는 경미하다

4. Post-Bert pre-training Advancement



#04. Post-Bert pre-training Advancement

#1 RoBERTA : A Robustly Optimized BERT Pretraining Approach
(Liu et al, University of Washington and Facebook, 2019)

BERT가 underfit한 상황으로 여겨 모델을 더 오래 학습하고 더 많은 데이터를 넣어 성능을 높임

- 학습 데이터: BERT에 비해 더 많은 데이터로 더 오래 학습을 진행
- 결과: 데이터의 양과 다양성이 중요, 오랜 학습에도 overfitting을 보이지 않음

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
<i>Single-task single models on dev</i>										
BERT _{LARGE}	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	Stanford	
XLNet _{LARGE}	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8		
RoBERTa	90.2/90.2	94.7	92.2	86.6	96.4	90.9	68.0	92.4		-

#04. Post-Bert pre-training Advancement

#2 XLNET : Generalized Autoregressive Pretraining for Language Understanding (Yang et al, CMU and Google, 2019)

1. AR(auto-regressive) 모델 : 데이터를 순차적으로 처리하는 기법

ex) ELM0, GPT

나는 고구마를 먹는다 \Rightarrow 나는 \rightarrow 고구마를 \rightarrow 먹는다

단점: 문맥을 양방향으로 볼 수 없다

2. AE(autoencoding) 모델: 입력값을 복원하는 기법

ex) BERT-masking

나는 [mask] 먹는다 \Rightarrow 나는 \rightarrow [mask] \leftarrow 먹는다

단점: masking 된 토큰을 서로 독립으로 가정하기 때문에 token들 사이의 의존관계를 고려할 수 없다

#04. Post-Bert pre-training Advancement

#2 XLNET : Generalized Autoregressive Pretraining for Language Understanding
(Yang et al, CMU and Google, 2019)

AR, AE 모델의 한계 극복을 위해 XLNET 제안

- Input sequence index는 모든 순열(permutation)을 고려하는 AR 방식
- 각 순열 AR 모델에 objective function을 적용하여 특정 token에 양방향 context 고려 가능

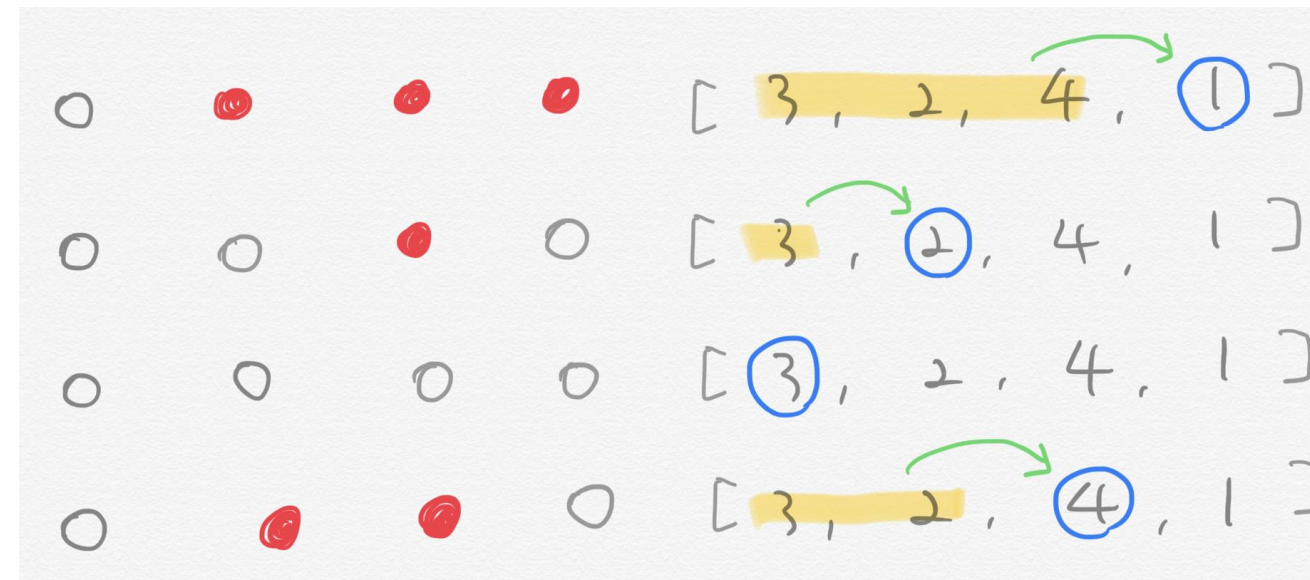
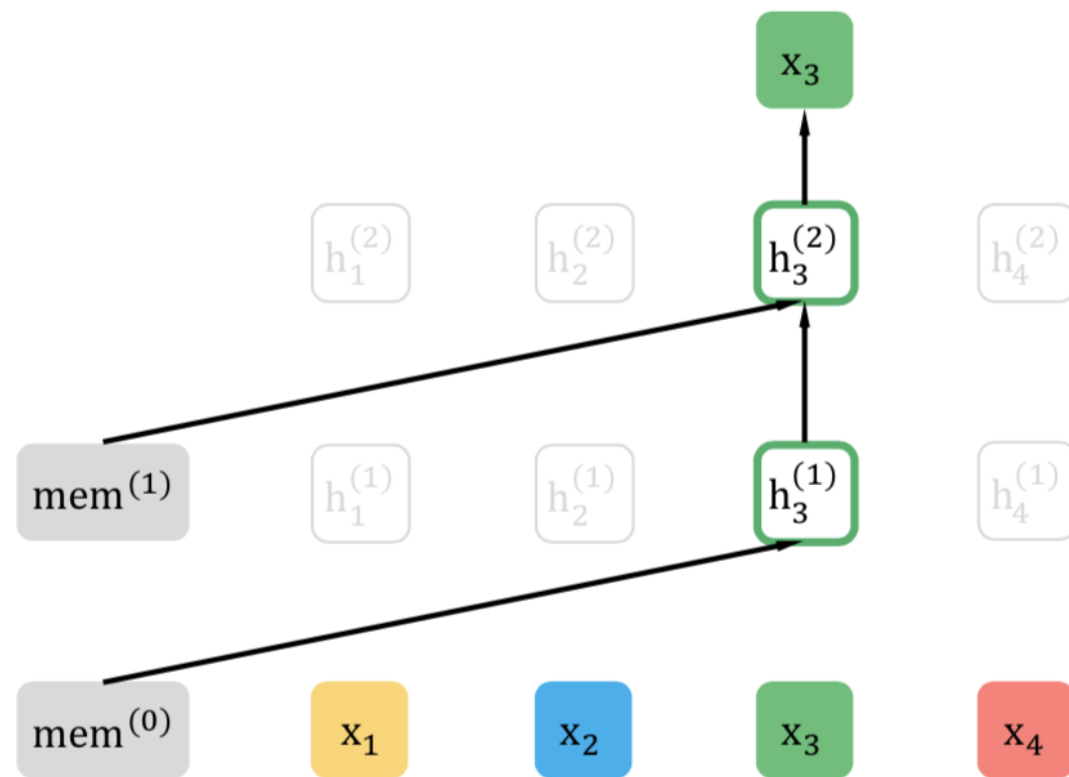
$$\text{input sequence} : x = (x_1, x_2, \dots, x_T)$$

$$\text{likelihood} : \mathbb{E}_{z \sim Z_T} [\prod_{t=1}^T p(x_{z_t} \mid x_{z < t})]$$

$$\text{training objective} : \max_{\theta} \mathbb{E}_{z \sim Z_T} [\sum_{t=1}^T \log p_{\theta}(x_{z_t} \mid x_{z < t})]$$

#04. Post-Bert pre-training Advancement

#2 XLNET : Generalized Autoregressive Pretraining for Language Understanding (Yang et al, CMU and Google, 2019)



- permutation은 attention mask로 실현
- Ex) 3번 단어를 맞추기 위해서는 정보를 사용할 수 없고, 2번 단어를 맞추기 위해 앞에 있는 3번 단어 사용
- 이 방법은 학습 시 permutation을 하므로 예측할 token이 명확하지 않아 standard transformer에서 작동하지 않음. 따라서 object function 적용을 위해 two-stram self-attention 제안

#04. Post-Bert pre-training Advancement

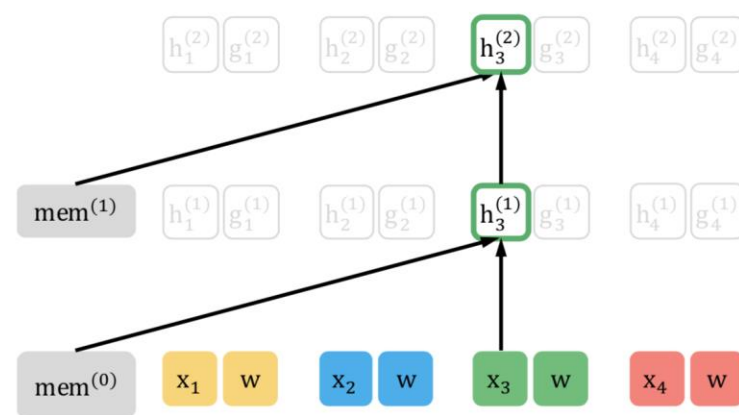
#2 XLNET : Generalized Autoregressive Pretraining for Language Understanding (Yang et al, CMU and Google, 2019)

-Two-stram self-attention

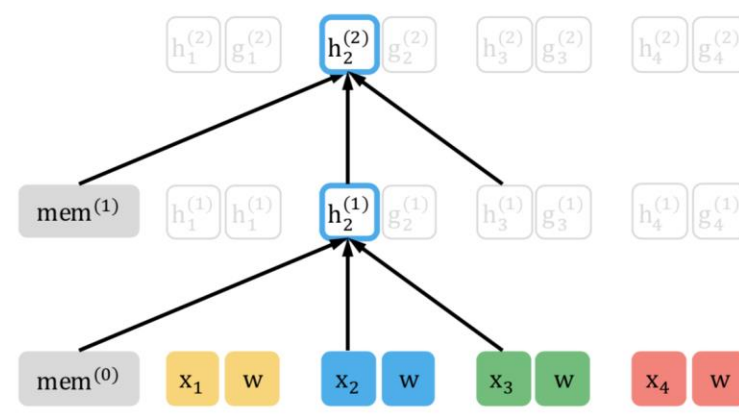
: 쿼리 스트림과 컨텍스트 스트림을 합하여 2개의 hidden representation을 가지고 업데이트하는 기법

*컨텍스트 스트림: t 시점과 t 이전 시점의 token 정보

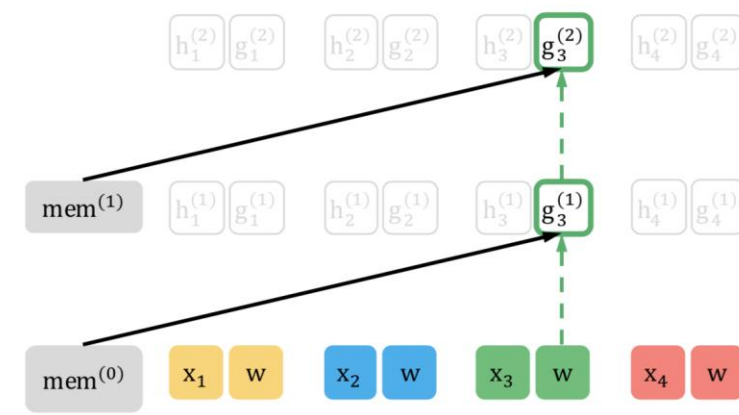
*쿼리 스트림: 토큰과 위치 정보를 활용한 self-attention 기법 (t 이전 시점의 token 정보 + t 시점의 정보)



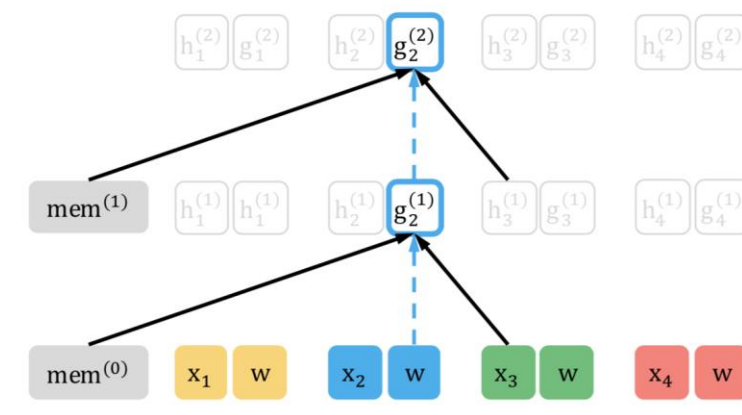
Position-3 View



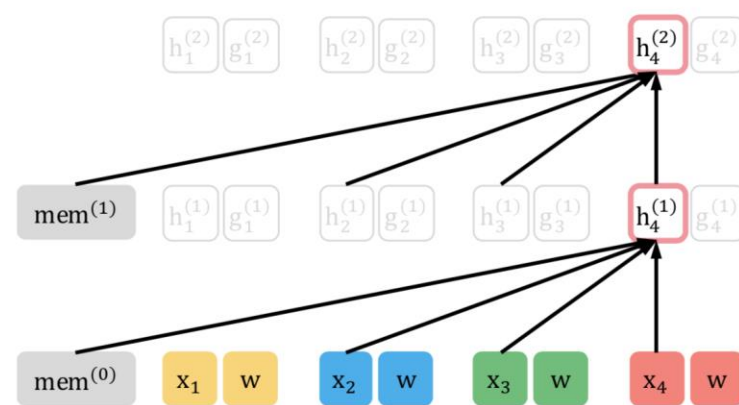
Position-2 View



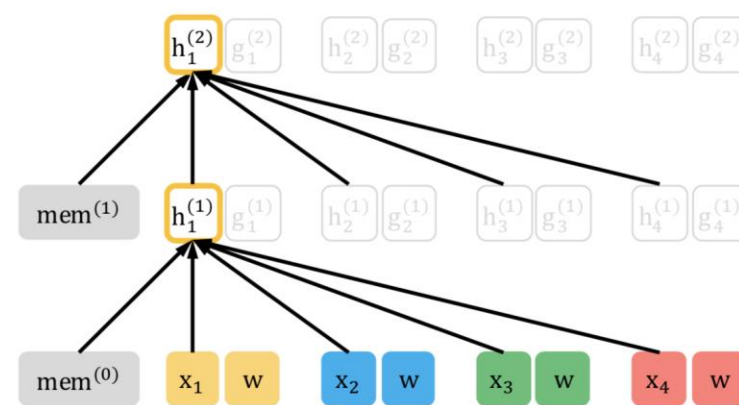
Position-3 View



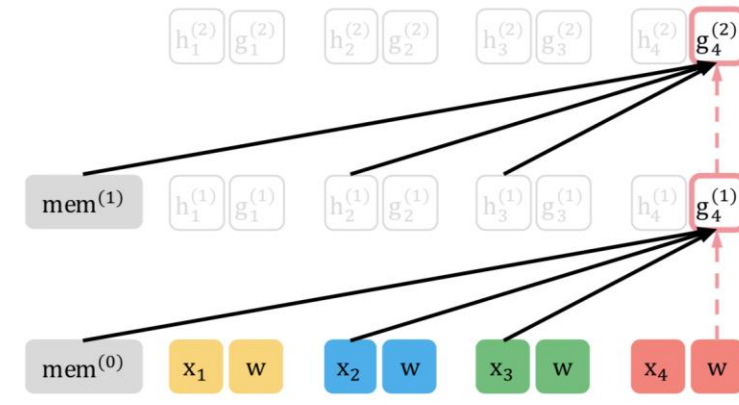
Position-2 View



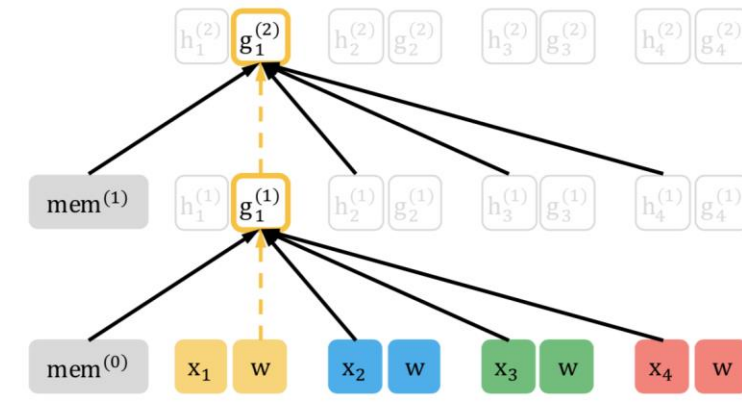
Position-4 View



Position-1 View



Position-4 View



Position-1 View

#04. Post-Bert pre-training Advancement

#3 ALBERT : A Lite BERT for Self-supervised Learning of Language Representations (Lan et al, Google and TTI Chicago, 2019)





- 워드 임베딩을 embedding size로 설정하여 파라미터 수를 감소시킴
- Attention과 FFNN 레이어 간 파라미터를 공유하는 Cross-layer parameter sharing 방법을 사용하여 파라미터 수를 줄이고, 안정적인 학습을 진행
- BERT에 비해 파라미터 수가 적고 training 속도가 빠르다

Model		Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg	Speedup
BERT	base	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3	4.7x
	large	334M	92.2/85.5	85.0/82.2	86.6	93.0	73.9	85.2	1.0
ALBERT	base	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1	5.6x
	large	18M	90.6/83.9	82.3/79.4	83.5	91.7	68.5	82.4	1.7x
	xlarge	60M	92.5/86.1	86.1/83.1	86.4	92.4	74.8	85.5	0.6x
	xxlarge	235M	94.1/88.3	88.1/85.1	88.0	95.2	82.3	88.7	0.3x

#04. Post-Bert pre-training Advancement

#4 T5 : Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer
(Raffel et al, Google, 2019)

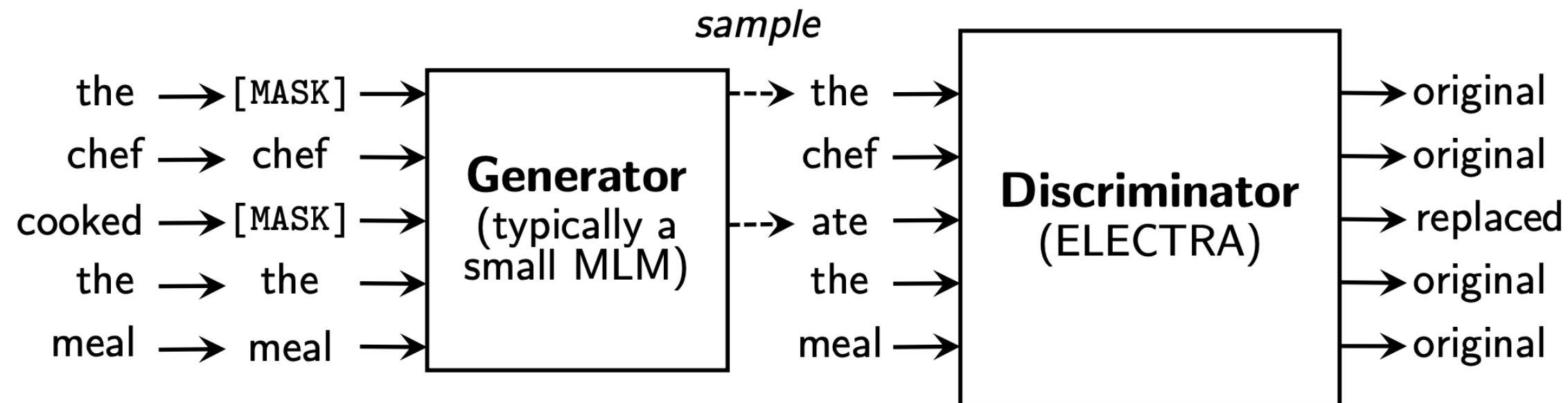
- 모든 NLP task를 통합할 수 있도록 text-to-text 프레임 워크 사용
- 모든 NLP task에서 T5라는 동일한 모델, loss, hyperparameter 사용 가능
- 파라미터 수가 BERT의 2 배이므로 비교적 비용이 높다

Rank	Name	Model	URL	Score	BoolQ	CB	COPA	MultRC	ReCoRD	RTE	WiC	WSC	AX-b	AX-g
1	SuperGLUE Human Baselines	SuperGLUE Human Baselines		89.8	89.0	95.8/98.9	100.0	81.8/51.9	91.7/91.3	93.6	80.0	100.0	76.6	99.3/99.7
+	2	T5 Team - Google		89.3	91.2	93.9/96.8	94.8	88.1/63.3	94.1/93.4	92.5	76.9	93.8	65.6	92.7/91.9
3	Zhuji Technology	RoBERTa-ctrl-adv		85.7	87.1	92.4/95.6	91.2	85.1/54.3	91.7/91.3	88.1	72.1	91.8	58.5	91.0/78.1
4	Facebook AI	RoBERTa		84.6	87.1	90.5/95.2	90.6	84.4/52.5	90.6/90.0	88.2	69.9	89.0	57.9	91.0/78.1
5	IBM Research AI	BERT-ctrl		73.5	84.8	89.6/94.0	73.8	73.2/30.5	74.6/74.0	84.1	46.2	61.0	21.6	97.8/57.3
6	SuperGLUE Baselines	BERT++		71.5	79.0	84.8/90.4	73.8	70.0/24.1	72.0/71.3	79.0	69.6	64.4	38.0	99.4/51.4
		BERT		69.0	77.4	75.7/83.6	70.6	70.0/24.1	72.0/71.3	71.7	69.6	64.4	23.0	97.8/51.7

#04. Post-Bert pre-training Advancement

#5 ELECTRA : Pre-training Text Encoders as Discriminators Rather Than Generators (Clark et al, 2020)

- 실제 text와 locally plausible text를 구별하도록 하는 모델 학습
- Generator : BERT MLM
- Discriminator : 입력 토큰 sequence에서 토큰이 original인지 replaced인지 분류
- 대용량 corpus에서 generator loss와 discriminator loss 합이 최소가 되도록 학습



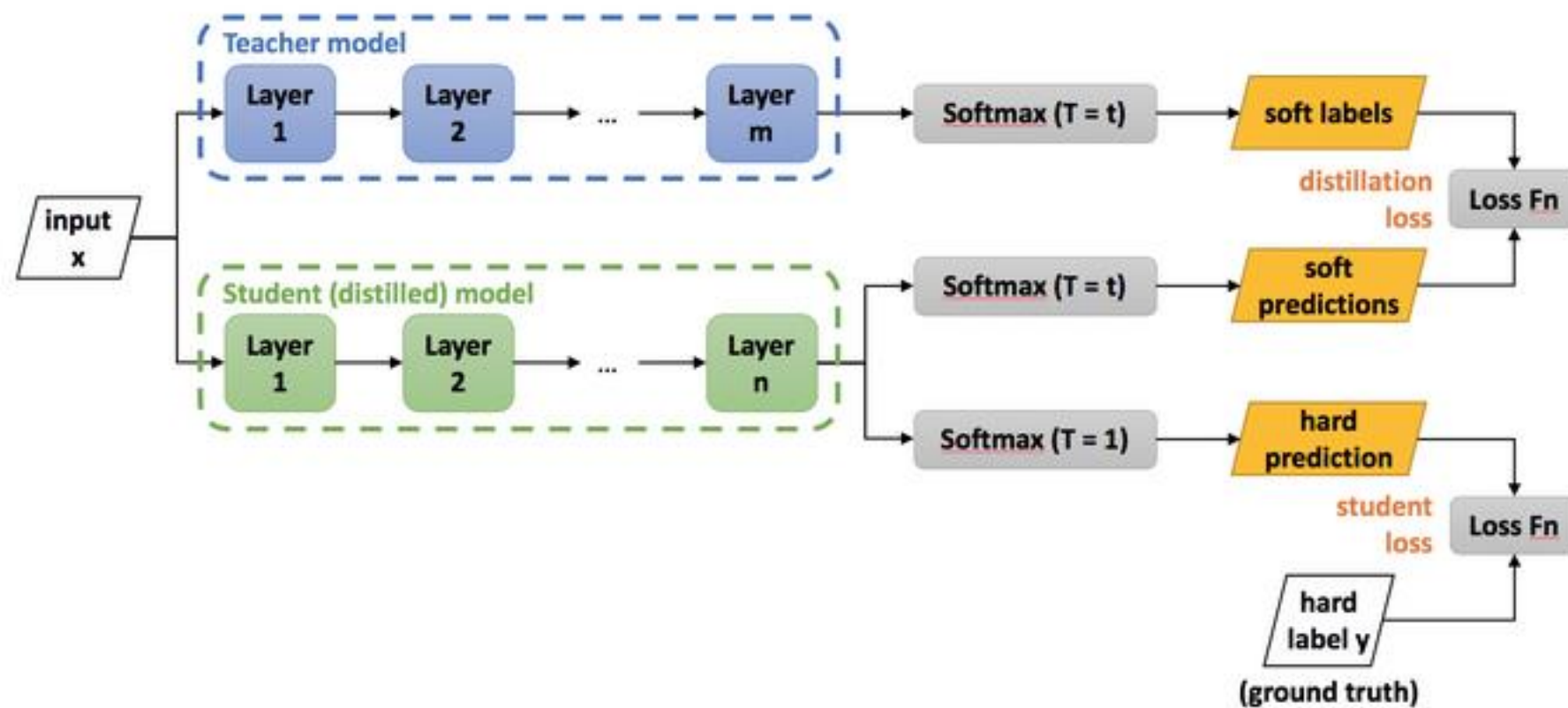
5. Distillation



#05. Distillation

BERT와 다른 pretrained language models은 매우 크고 비용이 많이 든다

⇒ Distillation을 통해 model compression



- Train Teacher: training set으로 sota pre-training + fine-tuning 하여 모델 정확성 극대화
- Label a large amount of unlabeled input examples with teacher
- Train student: teacher output을 흉내내는 작은 모델 훈련
- MSE, CE로 loss를 최소화

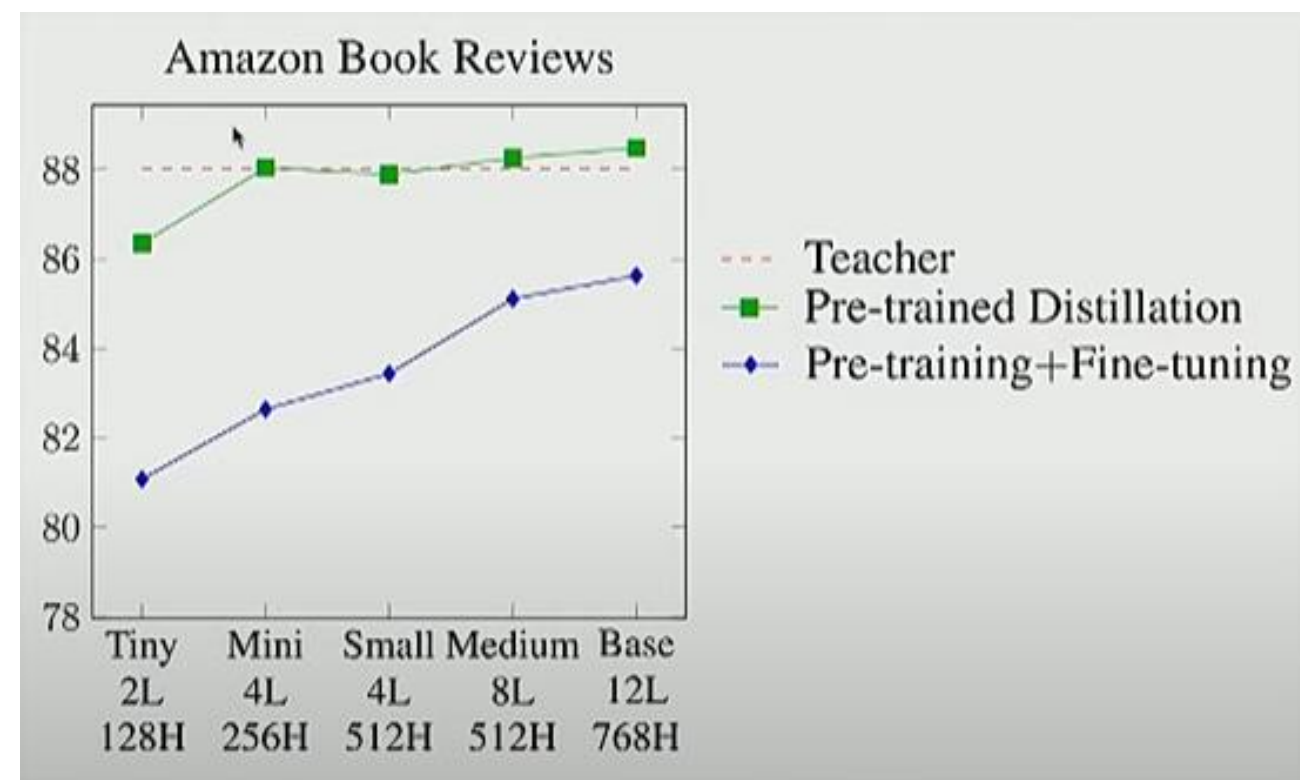
#05. Distillation

Example distillation results

- Distillation을 통해 성능은 유지하면서 model을 압축하는 것이 목표

Distillation의 성능이 좋은 이유

- training a massive language model learns millions of latent features which are useful for these other NLP tasks
- finetuning mostly just picks up and tweaks these existing latent features
- this requires an oversized model, because only a subset of the features are useful for any given task



THANK YOU

