Lecture 13.

Contextual Word Embeddings

CS224n Natural Language Processing with Deep Learning

Kang Changgu

2021.01.31

Contents

Contents

Word Representation

Deep Learning is all about Representation Learning.

Building the right tools for learning representations is an important factor in achieving empirical success.

- Ashish Vaswani

Word Representation = 단어에 대한 표현

Word Embedding 을 통해 단어를 벡터로 표현함으로써 자연어를 실수 공간의 벡터로 변환

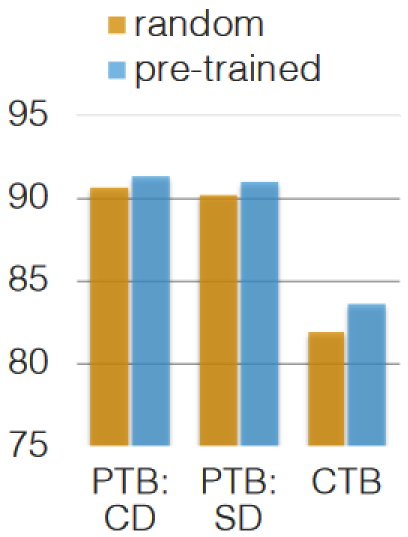
Pre-trained Word Vectors: The early years/Current Sense

	POS WSJ (acc.)	NER CoNLL (F1)
State-of-the-art*	97.24	89.31
Supervised NN	**96.37**	**81.47**
Unsupervised pre-training followed by supervised NN**	**97.20**	**88.87**
+ hand-crafted features***	97.29	89.59

-> rule based

-> Supervised Learning

-> Supervised Learning +
Unsupervised word representation
(Word2vec, Glove)



- Chen and Manning (2014)
  Dependency parsing
- Random: uniform(-0.01, 0.01)
- Pre-trained:
  - PTB (C & W): +0.7%
  - CTB (word2vec): +1.7%

Random Initialized < Pre-trained

(because we can train them for more words on much more data)

Tips for unknown words with word vectors

pre-trained word vector 사용 시 <UNK> 토큰 처리

<UNK> 1. dataset 에서 5회 이하로 등장하는 단어

2. train set 이 아닌, test set 에서만 등장하는 단어

Problem) No way to distinguish different <UNK> words

Solution)

1. char-level models to build vectors
2. if <UNK> word at test time appears in unsupervised word embeddings, use that vector
3. assign <UNK> word a random vector, adding them to vocabulary

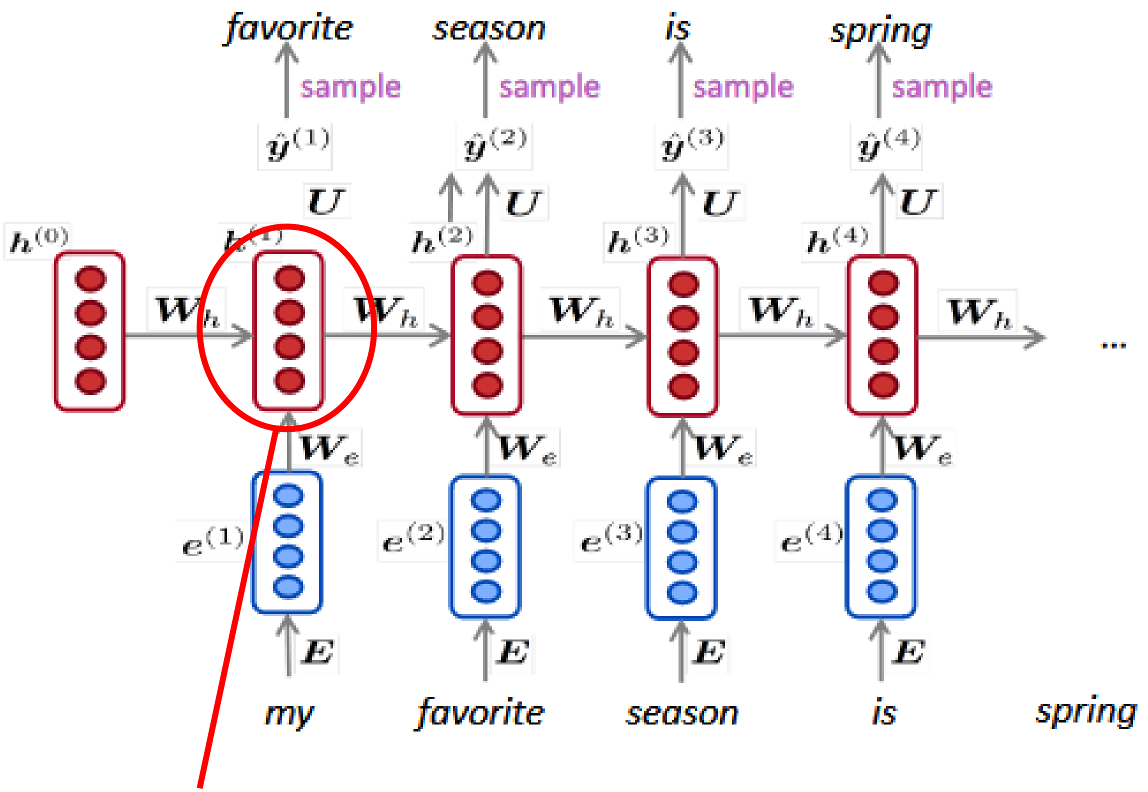
Representation for a word

one representation of words with word vectors (Word2Vec, GloVe, fastText)

Problems)

1. Always the same representation for a word type regardless of the context in which a word token occurs
2. Have one representation for a word, but words have different aspects, including semantic, syntactic behavior, and register/connotations

Representation for a word



NLM (Neural Language Modeling)

LSTM 층은 다음 단어를 예측하도록 학습

각 LSTM 층에서 context-specific word representation 을 계산하므로 문맥 유지

actual representation of a word in context → already invented a way to have context-based

Contents

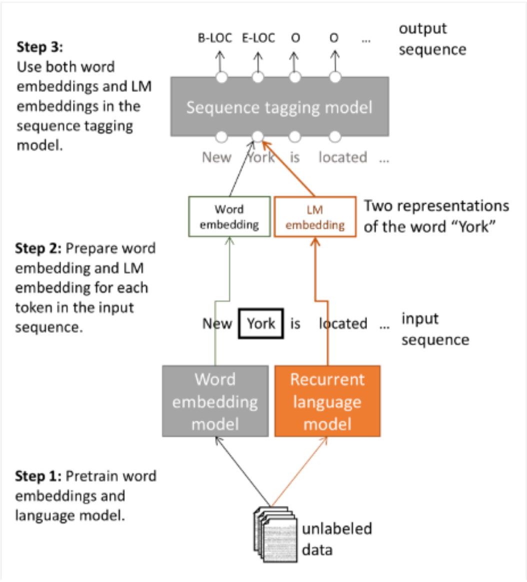
Pre-ELMo

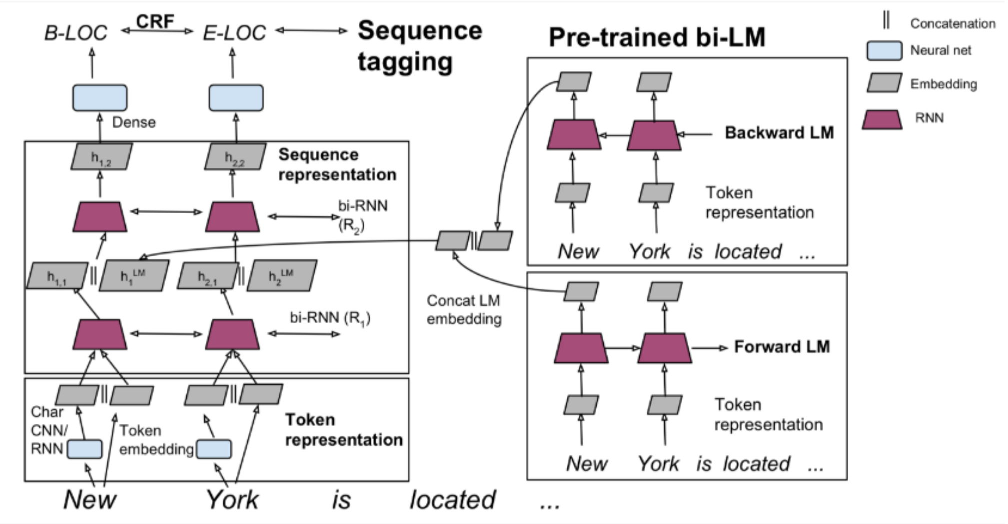
TagLM – "Pre-ELMo"

semi-supervised approach where we train NLM on large unlabeled corpus,

rather than just word vectors

=> semi-supervised learning with pre-trained word vector





$$\mathbf{h}_{k,1} = [\overrightarrow{\mathbf{h}}_{k,1}; \overleftarrow{\mathbf{h}}_{k,1}; \mathbf{h}_k^{LM}].$$

Pre-ELMo

[STEP 1]

- Word Embedding model 로 word vector 학습 (Word2Vec, GloVe)
- NLM model 로 word vector 학습 (bi-LSTM)

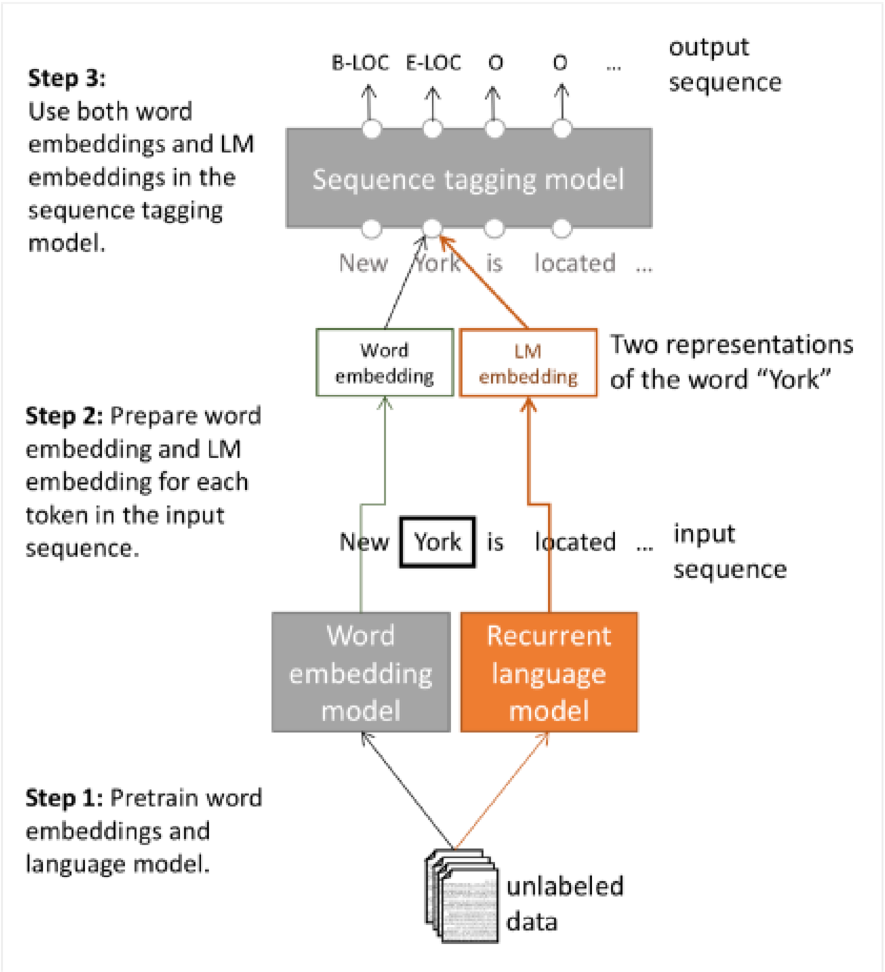
[STEP 2]

- Word Embedding 과 LM Embedding 을 input sequence 로 구성

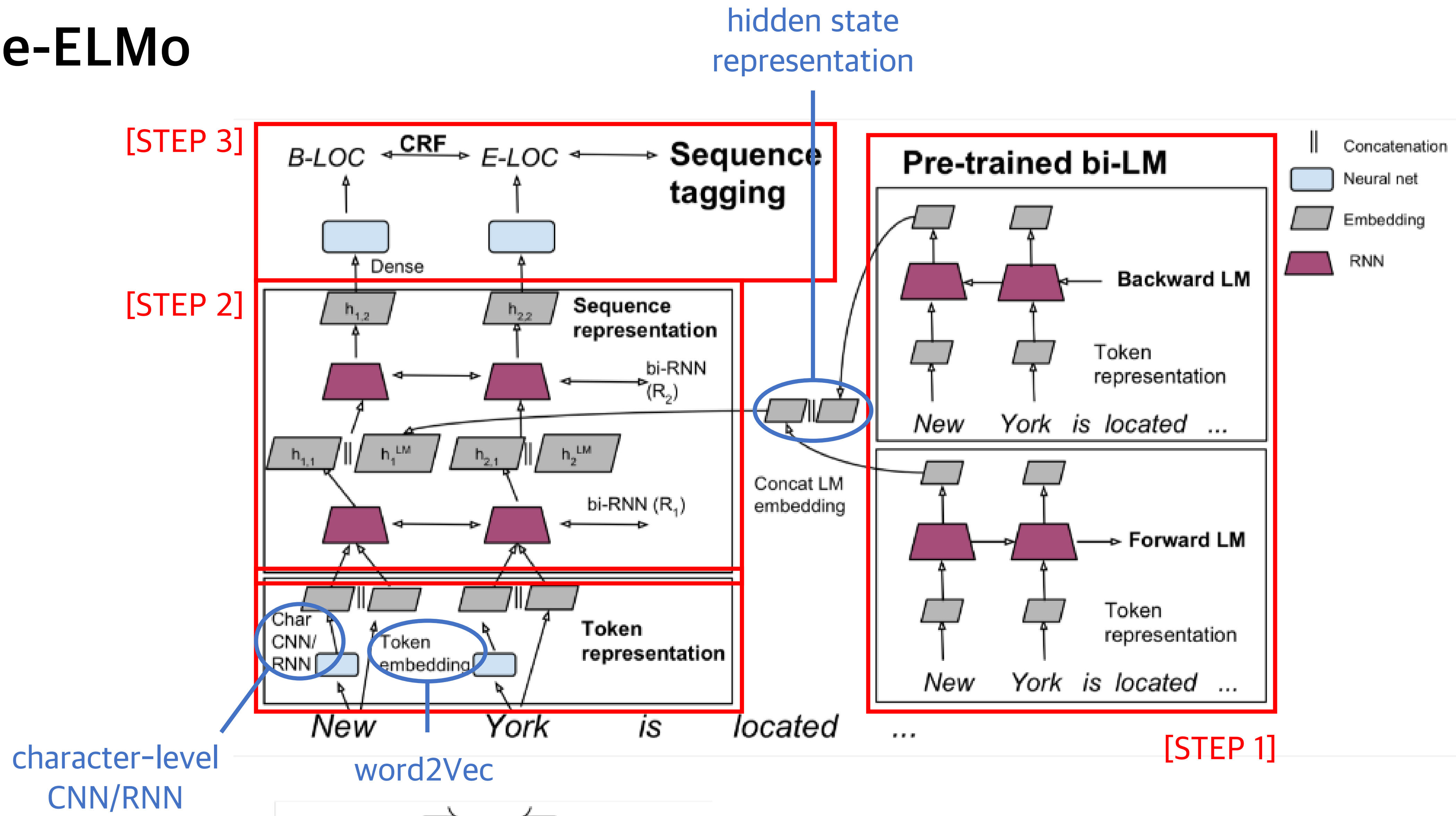
[STEP 3]

- sequence tagging model 에 Word Embedding 과 LM Embedding 을 input 으로 입력

-> feature word vector 와 context word vector 모두 사용



Pre-ELMo



$$\mathbf{h}_{k,1} = [\overrightarrow{\mathbf{h}}_{k,1}; \overleftarrow{\mathbf{h}}_{k,1}; \mathbf{h}_k^{LM}].$$

ELMo

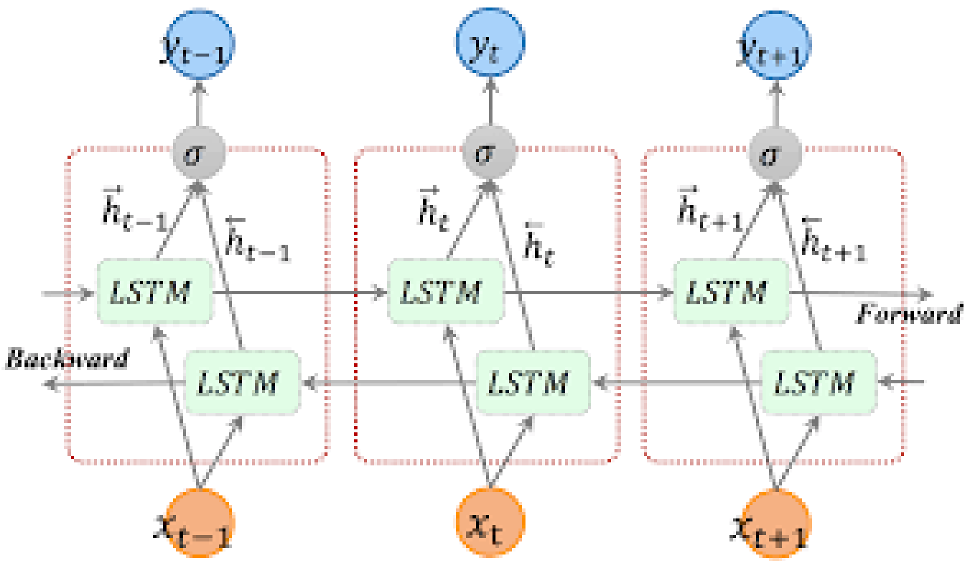
ELMO: Embeddings from Language Models

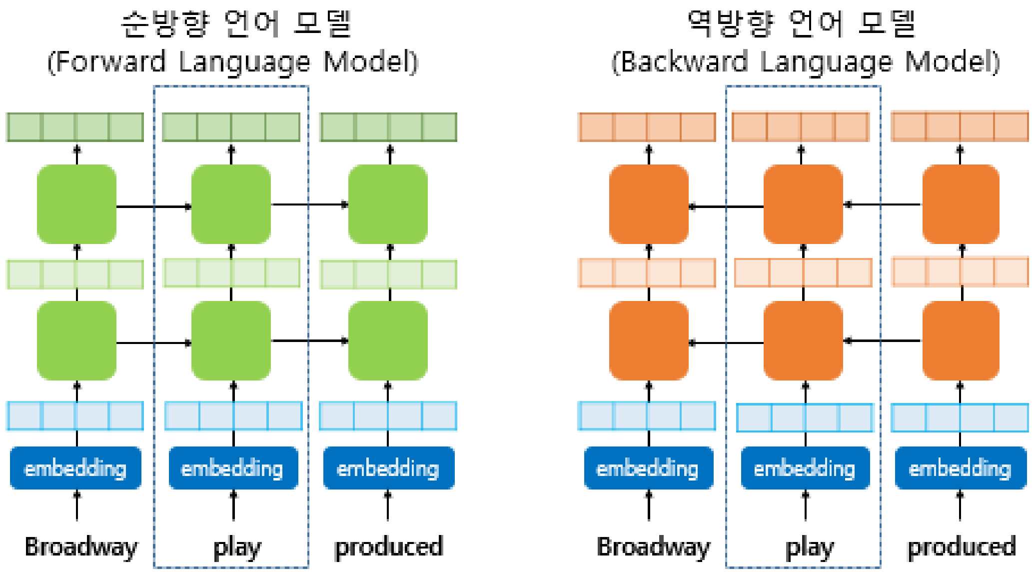
Deep contextualized word representations

- Breakout version of word token vectors/contextual word vectors
- Learn word token vectors using long contexts not context windows

- Use 2 bi-LSTM layers
    - Lower layer is better for lower-level syntax e.g. Part-of-speech tagging, NER
    - Higher layer is better for higher-level sematics e.g. Sentiment, Question Answering
- Use character CNN to build initial word representation only
  (= dispense with having word representation together)
- Use 4096 dim hidden/cell LSTM state with 512 dim projections to next input

ELMo





- 양방향 RNN

  순방향 RNN 의 은닉 상태와 역방향 RNN의

  은닉상태를 다음 층의 입력으로 보내기 전에

  concatenate

- biLM(Bidirectional Lanauage Model)

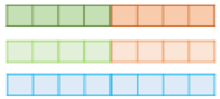
  순방향 언어 모델과 역방향 언어 모델이

  <u>각각의 은닉 상태만을</u> 다음 은닉층으로 전달하여

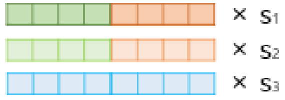
  훈련 시킨 후에 은닉 상태를 concatenate

ELMo

1) 각 층의 출력값을 연결(concatenate)한다.

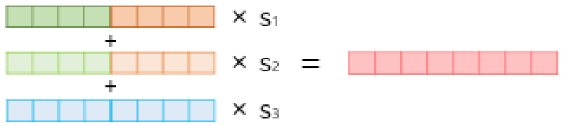


2) 각 층의 출력값 별로 가중치를 준다.



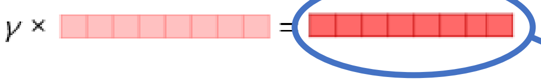
이 가중치를 여기서는 s_1, s_2, s_3라고 합시다.

3) 각 층의 출력값을 모두 더한다.

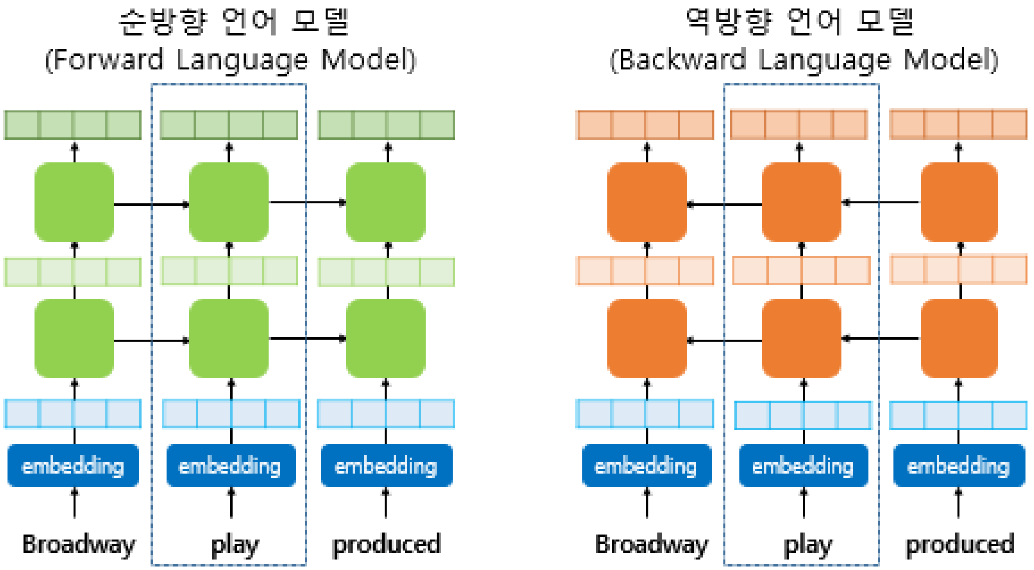


2)번과 3)번의 단계를 요약하여 가중합(Weighted Sum)을 한다고 할 수 있습니다.

4) 벡터의 크기를 결정하는 스칼라 매개변수를 곱한다.



이 스칼라 매개변수를 여기서는 γ이라고 합시다.



$$
\begin{aligned}
R_k &= \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\
&= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\},
\end{aligned}
$$

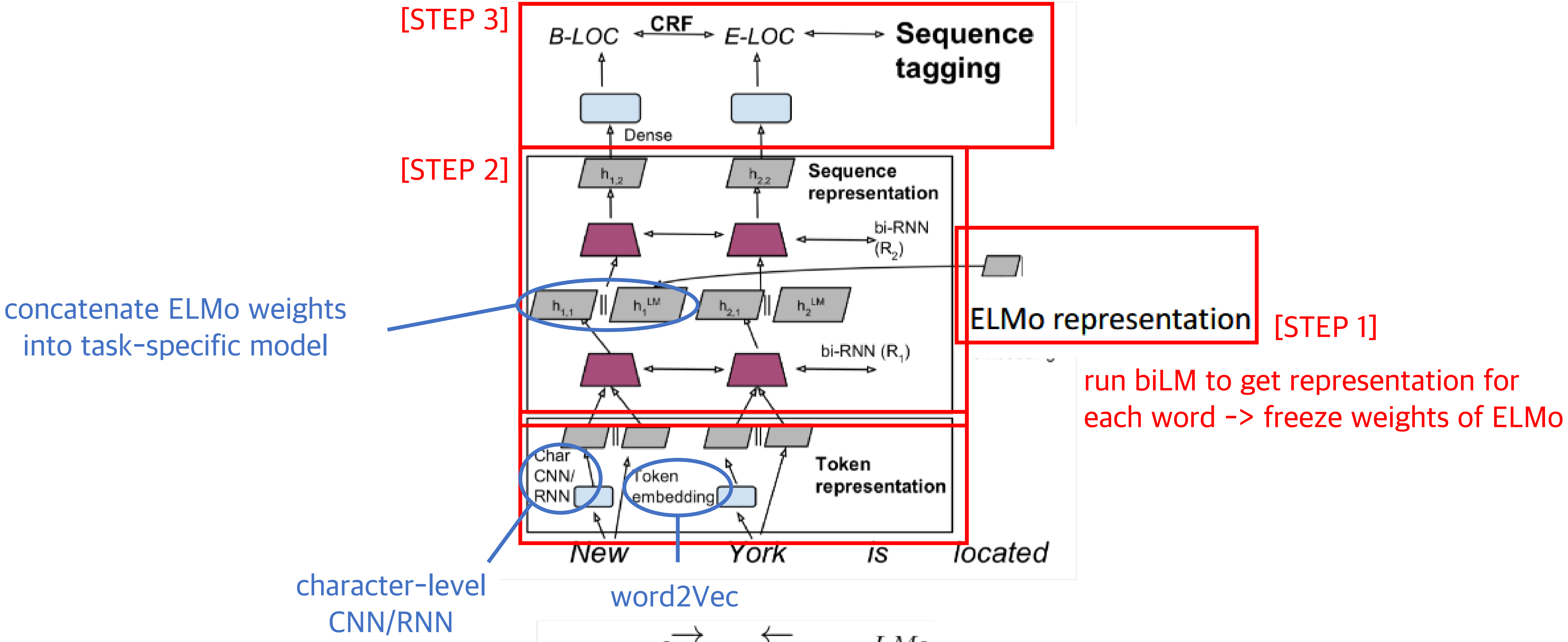
hidden state of kth layer

-> use all layers of NLM

$$
\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}
$$

ELMo representation

ELMo



$$\mathbf{h}_{k,1} = [\overrightarrow{\mathbf{h}}_{k,1}; \overleftarrow{\mathbf{h}}_{k,1}; \mathbf{h}_k^{LM}].$$

Contents

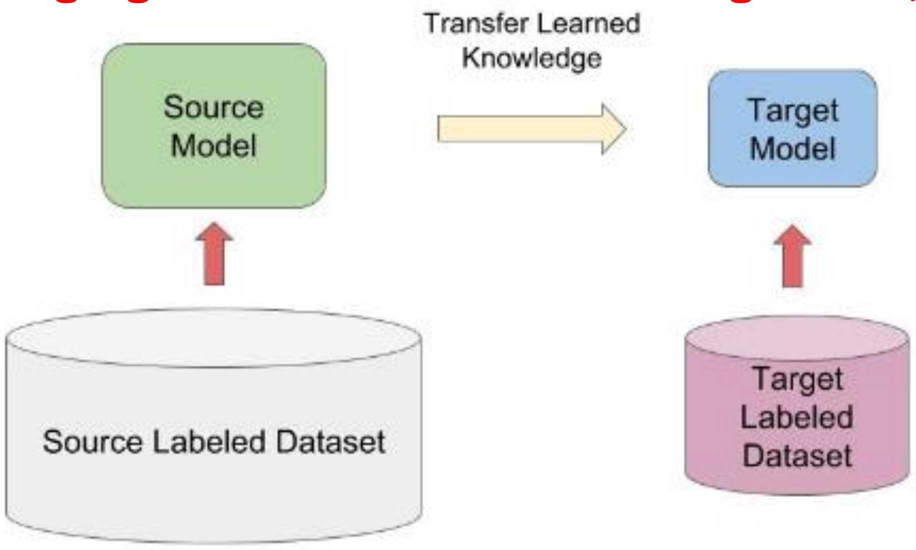
ULMFit

ULMFit : Universal Language Model Fine-tuning for Text Classification

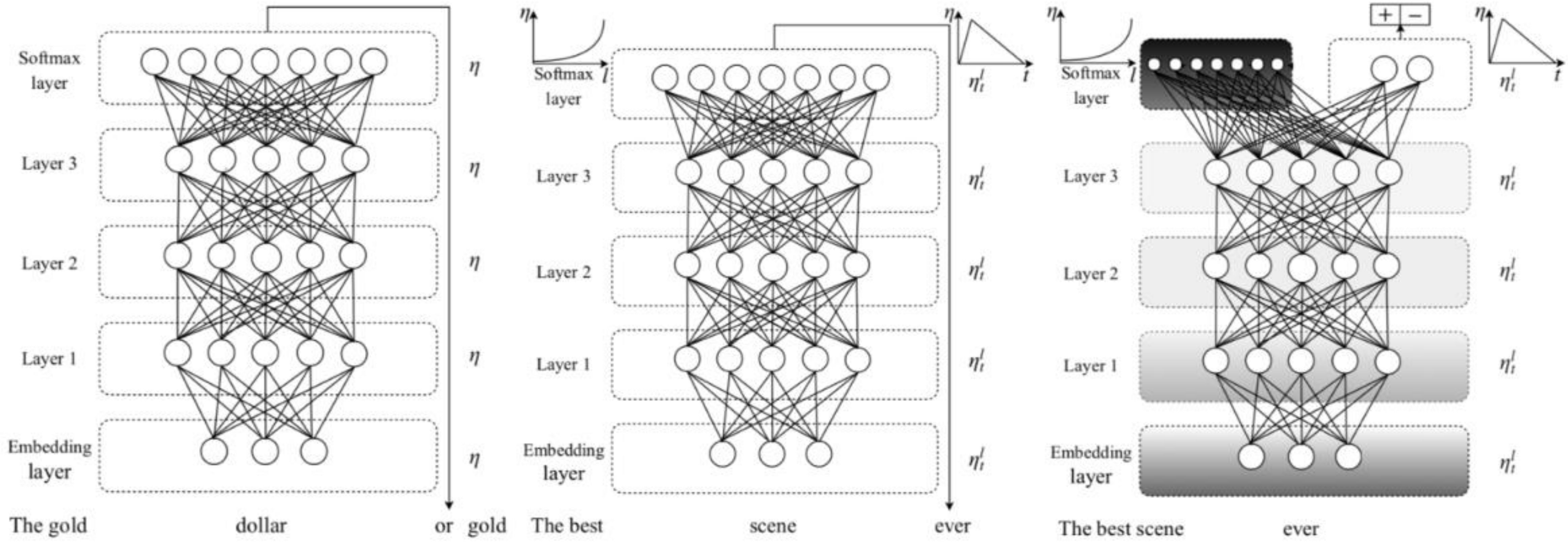
general idea of transferring NLM knowledge, applied to text classification

big language model

target task (e.g. NER)



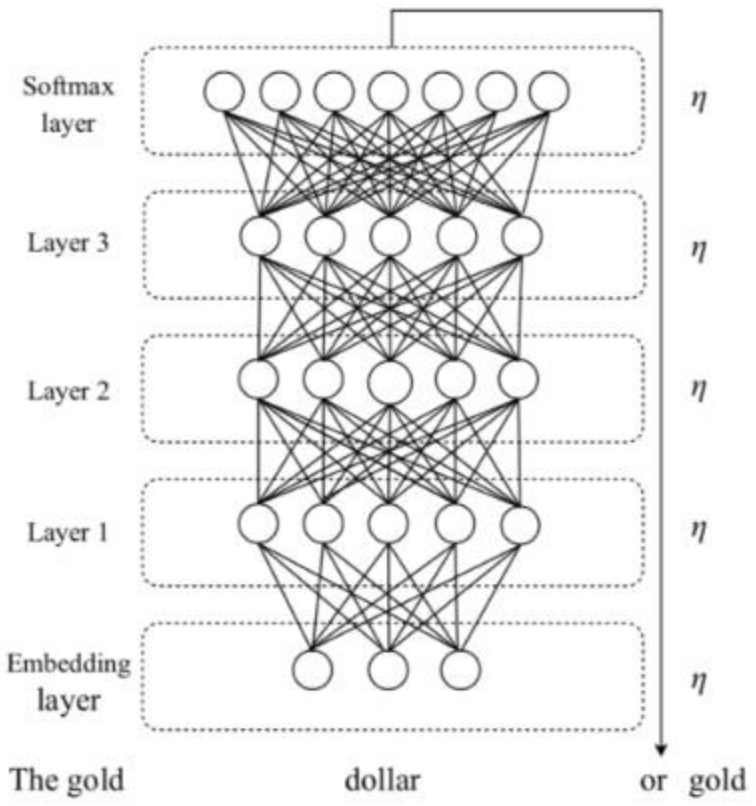
ULMFit



(a) LM pre-training (b) LM fine-tuning (c) Classifier fine-tuning

일반 언어 모델 학습 과제 맞춤형 언어 모델 튜닝 과제 분류기 튜닝

ULMFit

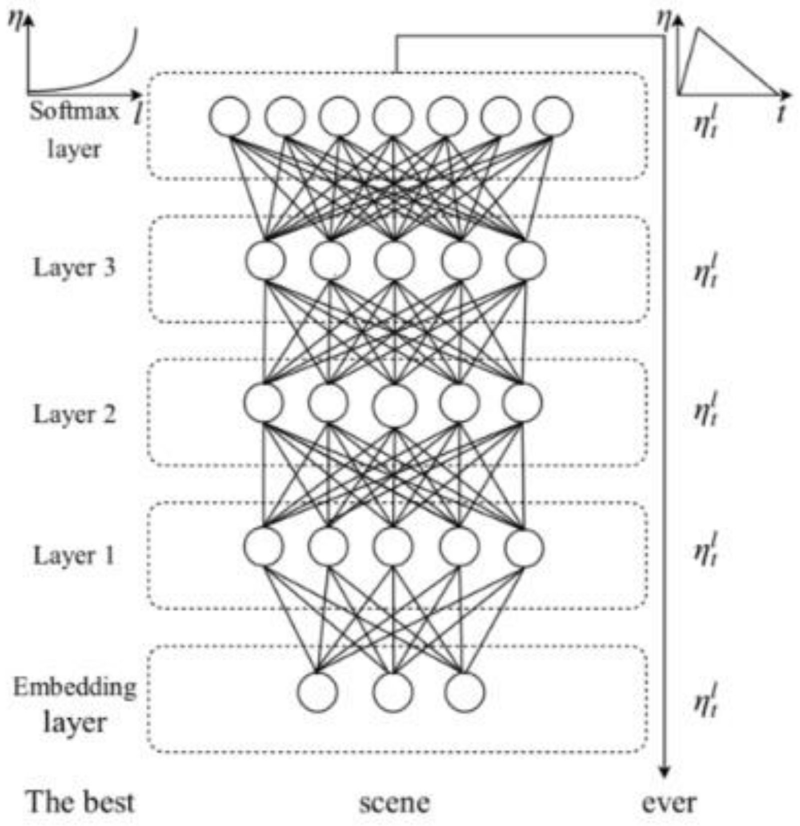


(a) LM pre-training

일반 언어 모델 학습

- 3-layer bi-LSTM Language Model 로 pre-trained
- 논문에서는 Wikipedia 의 정제된 28,595 개의 article 과 1억 개의 word 를 pre-training

ULMFit



(b) LM fine-tuning

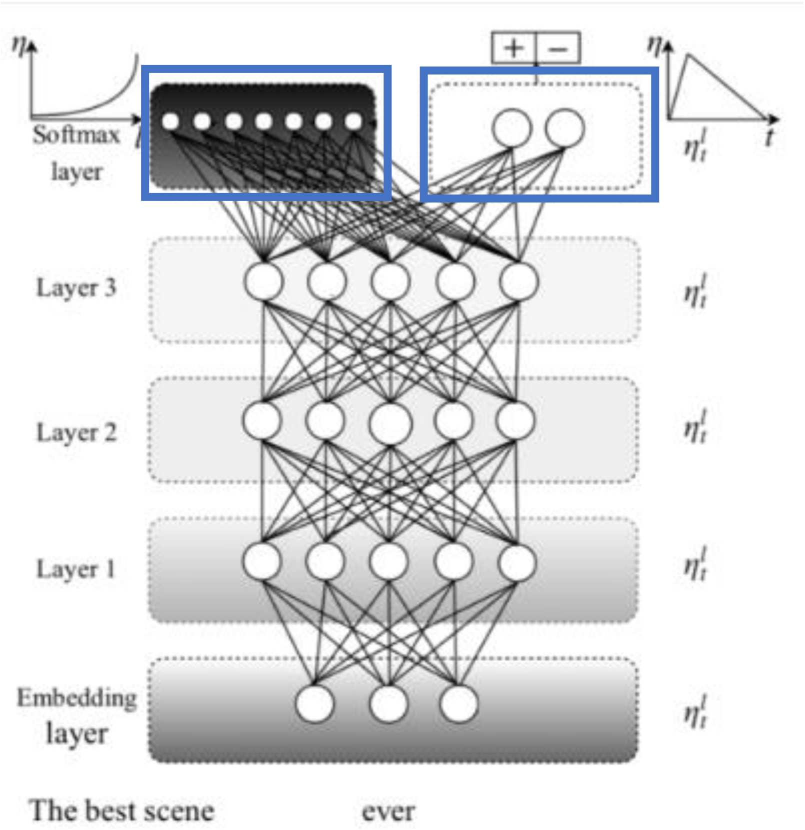
과제 맞춤형 언어 모델 튜닝

- 일반 언어 모델을 주어진 task에 맞게 조정하여 추가 학습
  학습 시 다른 네트워크가 아닌 동일한 네트워크를 사용하며, 네트워크에 대해 다른 objective 를 적용

- 주어진 task 에 100개의 텍스트와 대응되는 100개의 label 이 있는 경우, 100개의 텍스트를 사용하여 언어 모델을 튜닝

- 튜닝 시 사용하는 2가지 방법
  - Discriminative Fine-tuning
  - Slanted triangular learning rates

ULMFit



(c) Classifier fine-tuning

과제 분류기 튜닝

- 상위 계층의 softmax 층을 얼리고,
  다른 prediction unit 을 연결하여 특정 과제에 대한 예측 수행

Contents

Transformer models

All of these models are Transformer architecture models … so maybe we had better learn about Transformers?

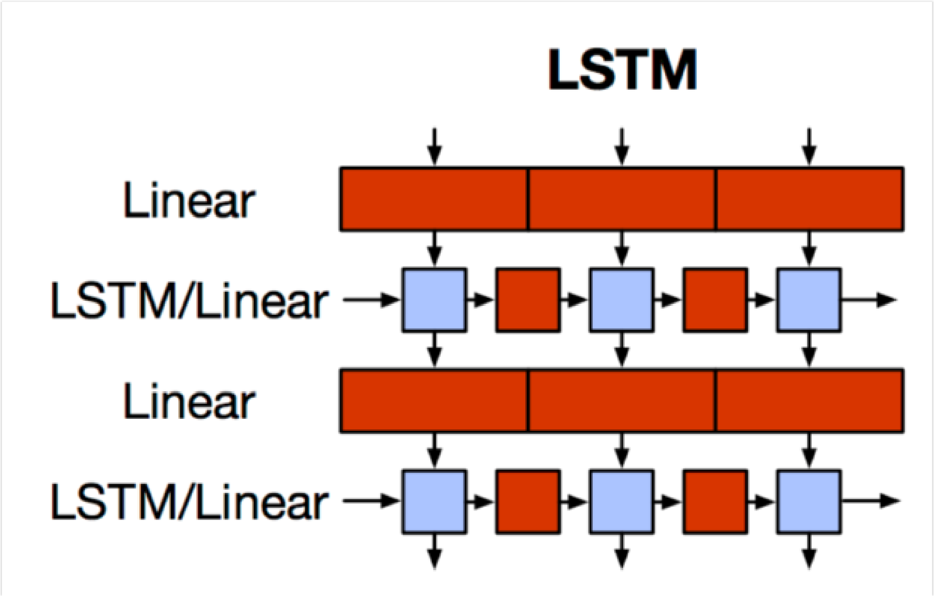
ULMfit	GPT	BERT	GPT-2
Jan 2018	June 2018	Oct 2018	Feb 2019
Training:	Training	Training	Training
1 GPU day	240 GPU days	256 TPU days ~320–560 GPU days	~2048 TPU v3 days according to a reddit thread
fast.ai	OpenAI	Google AI	OpenAI

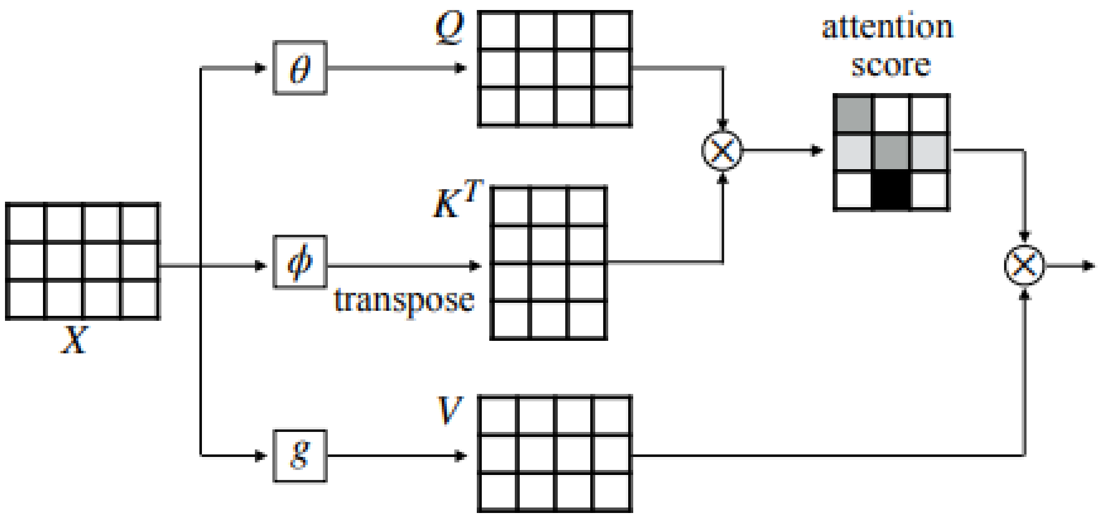
Transformer

RNNs are inherently sequential but we want parallelization

Despite GRUs and LSTMs, RNNs still need attention mechanism to deal ling range dependencies

-> What if attention gives us access to any state?

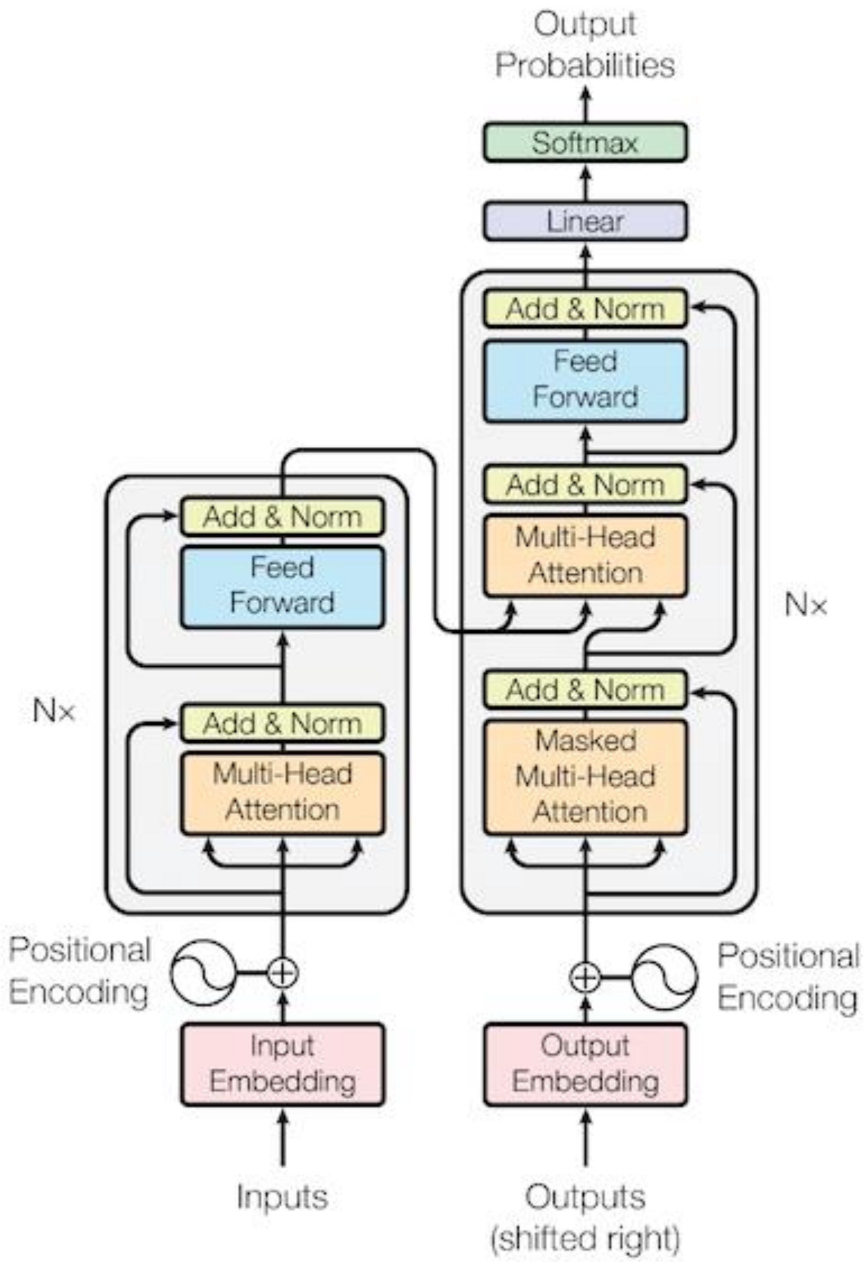




Transformer

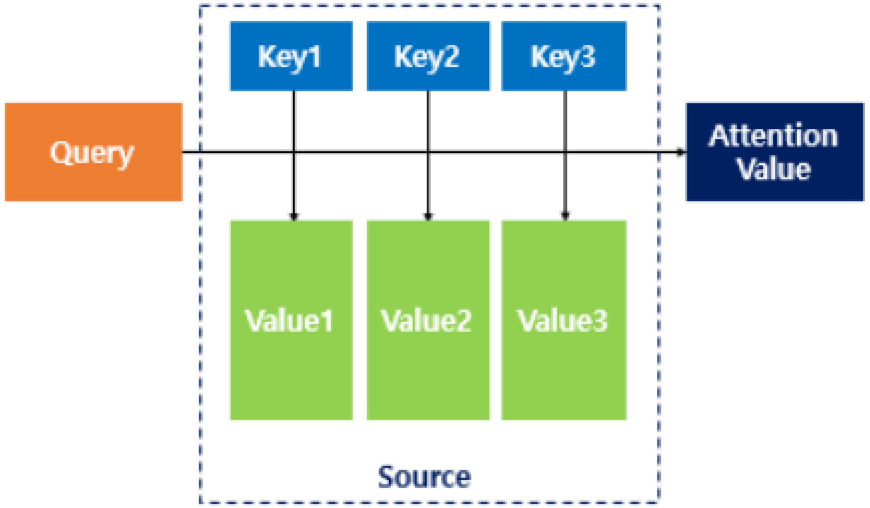
Transformer Overview

- Non-recurrent sequence-to-sequence encoder-decoder model
- Predict each translated word
- Final cost/error function is standard cross-entropy error on top of a softmax classifier
- Task : machine translation with parallel corpus



Transformer

Self-attention

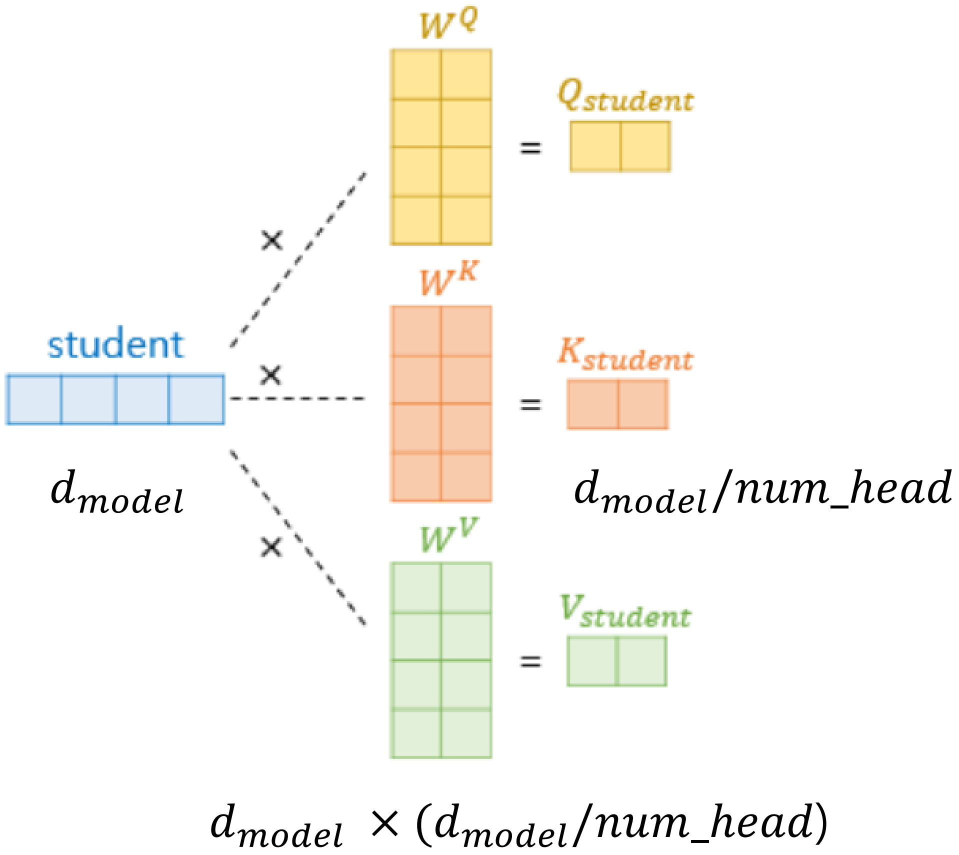


Query : 입력 문장의 모든 단어 벡터들

Key : 입력 문장의 모든 단어 벡터들

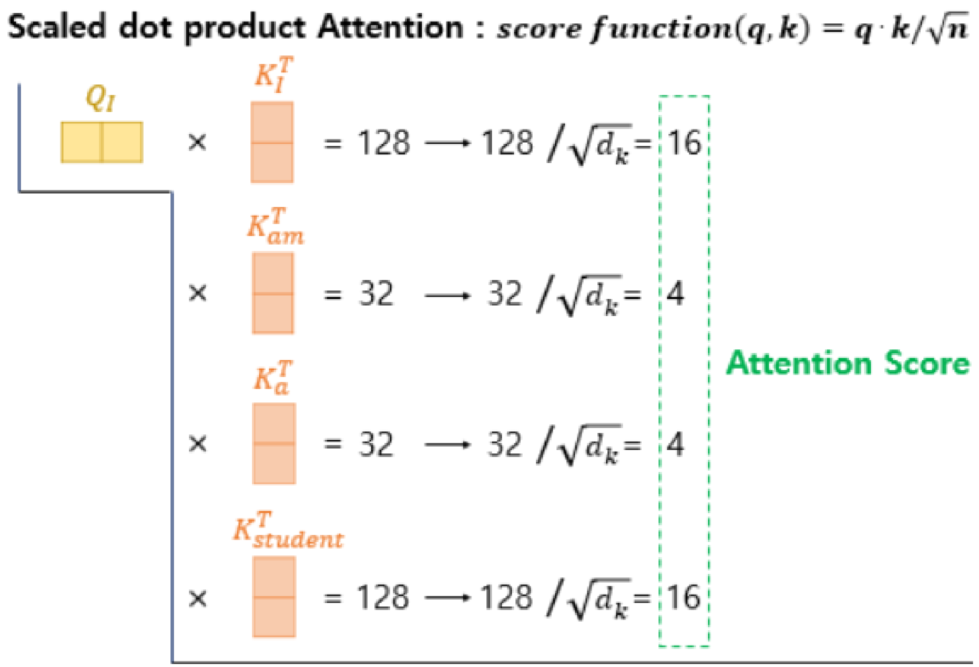
Value : 입력 문장의 모든 단어 벡터들

Dot-Product Attention

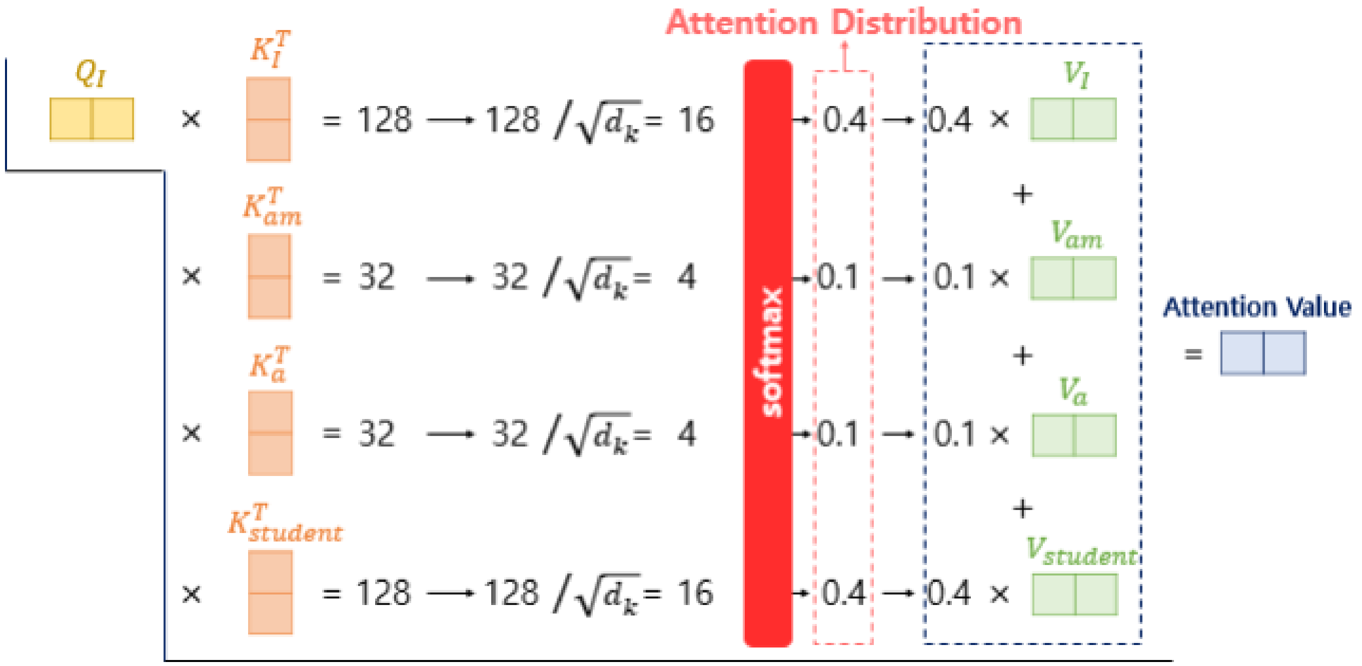


Transformer

Dot-Product Attention



attention score 계산

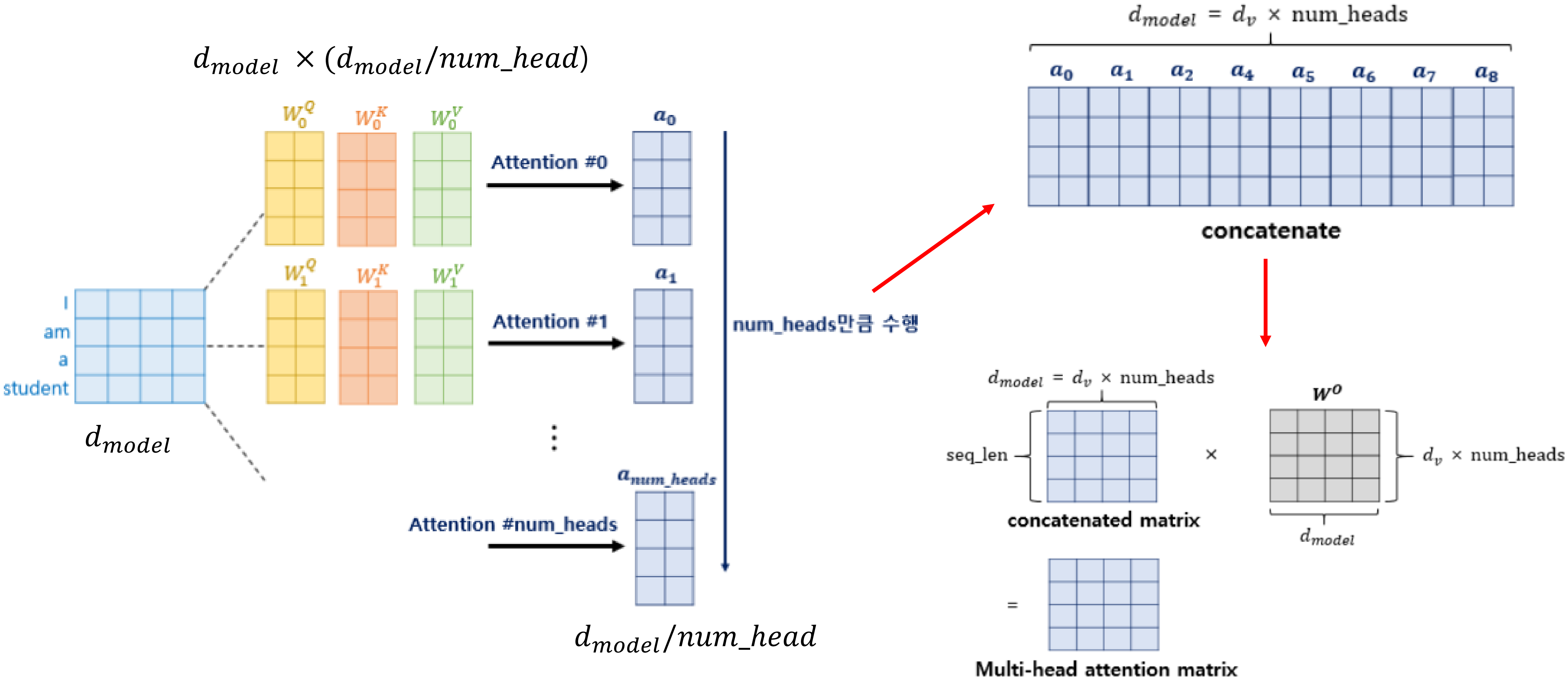


attention distribution/value 계산

$$=> \quad A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i$$

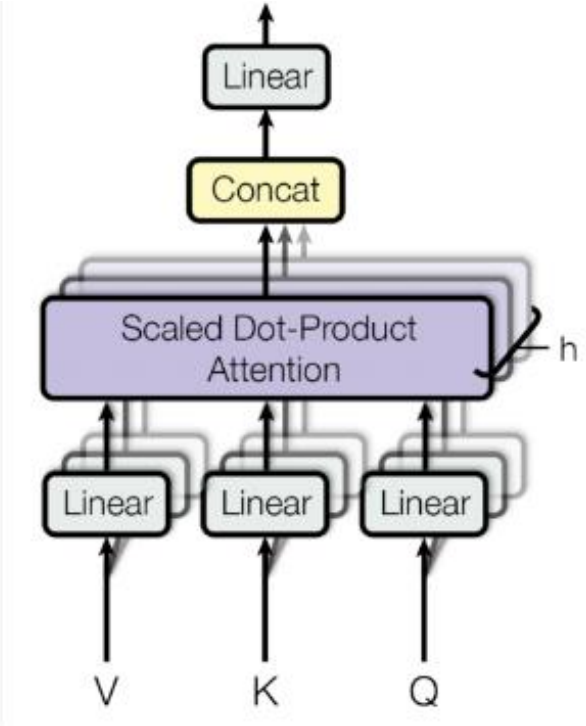
Transformer

Multi-head attention



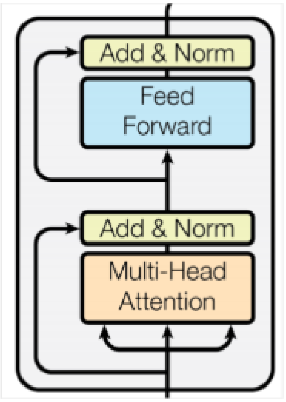
Transformer

Multi-head attention



$$\mathrm{MultiHead}(Q,K,V) = \mathrm{Concat}(\mathrm{head}_1, ..., \mathrm{head}_h)W^O$$
$$\text{where } \mathrm{head}_i = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Transformer Block



- Each block has two "sublayers"

1. Multihead attention
2. 2-layer feed-forward NNet (with ReLU)

Each of these two steps also has:

Residual (short-circuit) connection and LayerNorm

LayerNorm(x + Sublayer(x))

Layernorm changes input to have mean 0 and variance 1,
per layer and per training point (and adds two more parameters)

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$$

$$h_i = f\left(\frac{g_i}{\sigma_i}(a_i - \mu_i) + b_i\right)$$

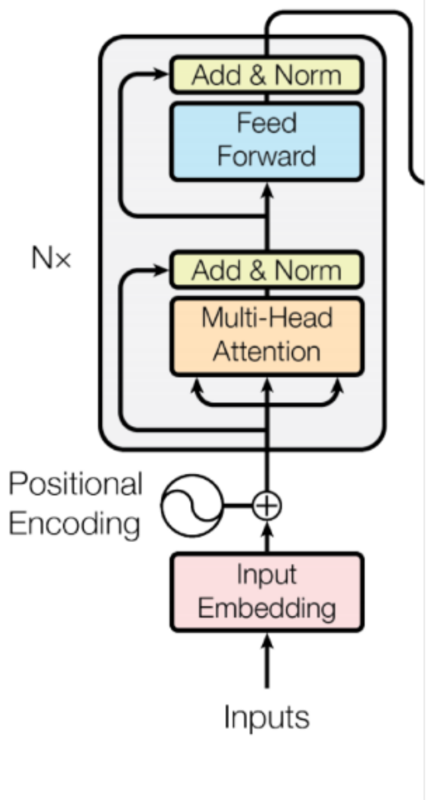
Transformer

Positional Encoding

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Transformer Encoder

- For encoder, at each block, we use the same Q, K and V from the previous layer
- Blocks are repeated 6 times
  - (in vertical stack)



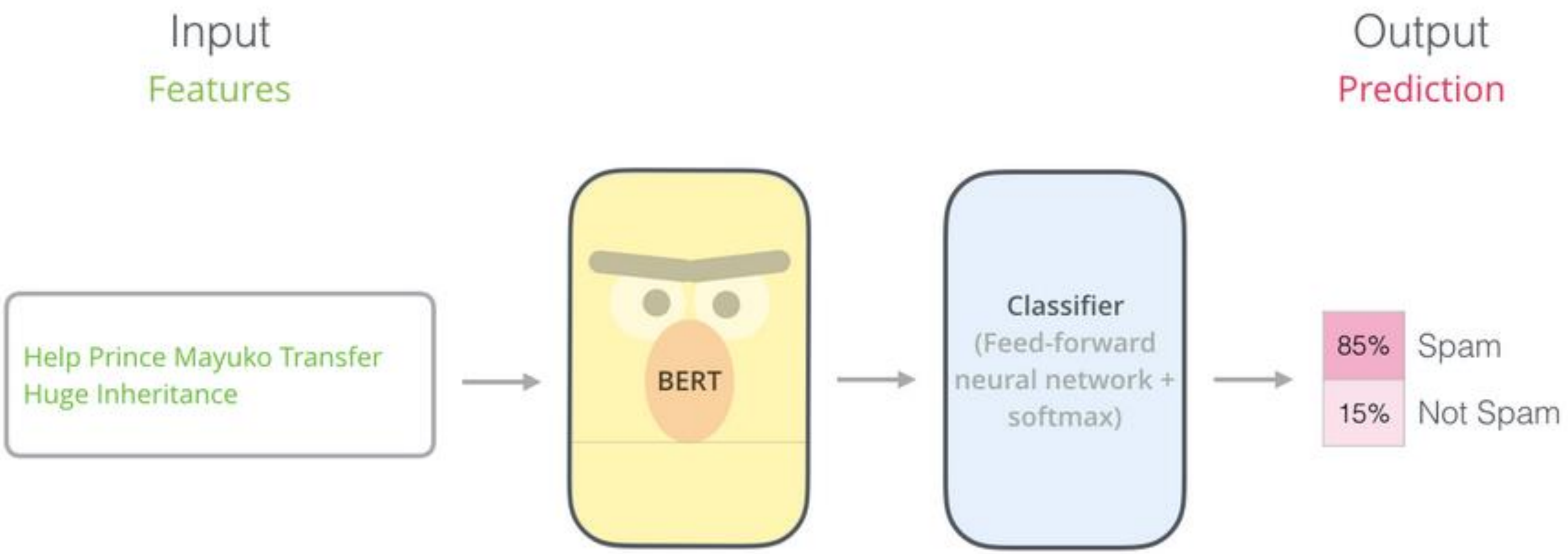
-> BERT, GPT 에 사용되는 Block

Contents

BERT

BERT : Bidirectional Encoder Representations from Transformers

Pre-training of Deep Bidirectional Transformers for Language Understanding



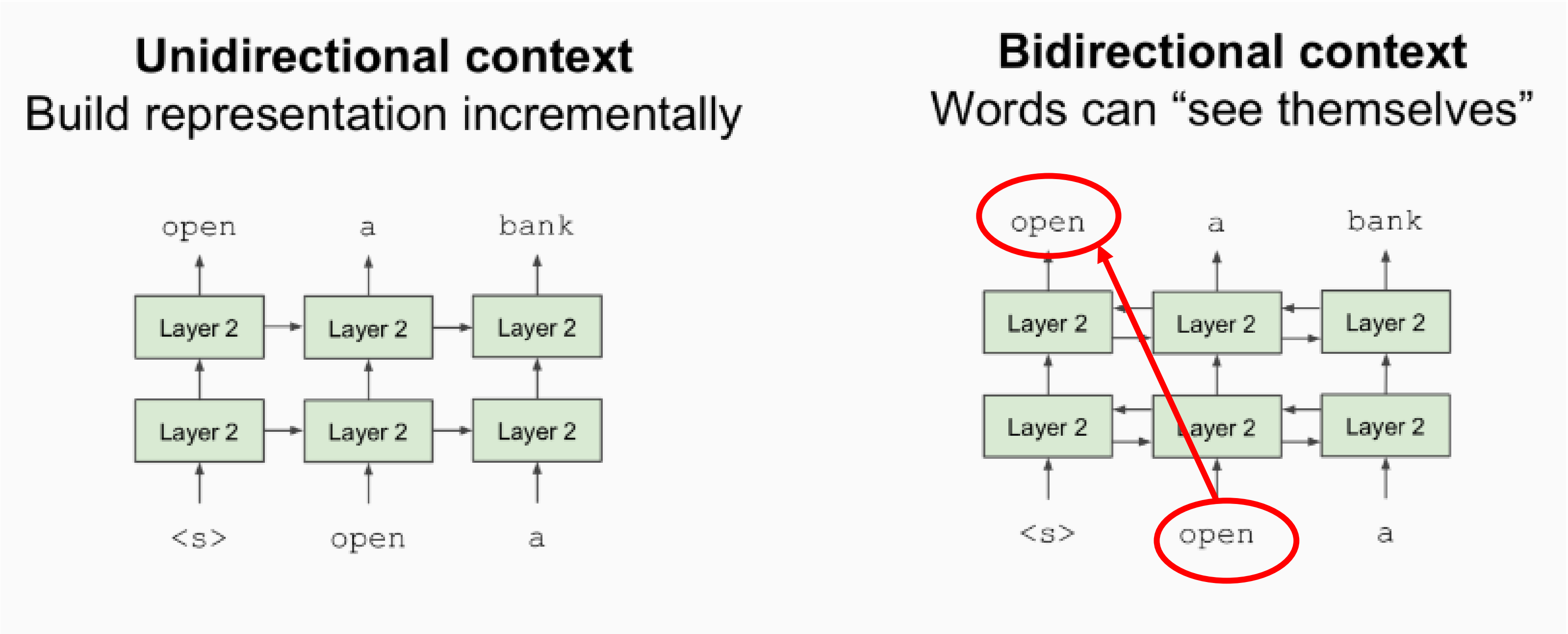
BERT

Problem) Language models only use left/right context, but language understanding is bidirectional

Why are LMs unidirectional?

Reason 1 : Directionality is needed to generate a well-formed probability distribution

Reason2 : Words can “see themselves” in a bidirectional encoder



BERT

Solution) Mask out k% of input words, and then predict the masked words

- They always use k = 15%

store ↑ gallon ↑

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

Too little masking : Too expensive to train

Too much masking : Not enough context

BERT

Maksed LM (MLM)

Solution) Mask out k% of input words, and then predict the masked words

- They always use *k* = 15%

store ↑ gallon ↑

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

Too little masking : Too expensive to train

Too much masking : Not enough context

BERT

Next Sentence Prediction (NSP)

To learn relationships between sentences, predict whether sentence B is actual sentence

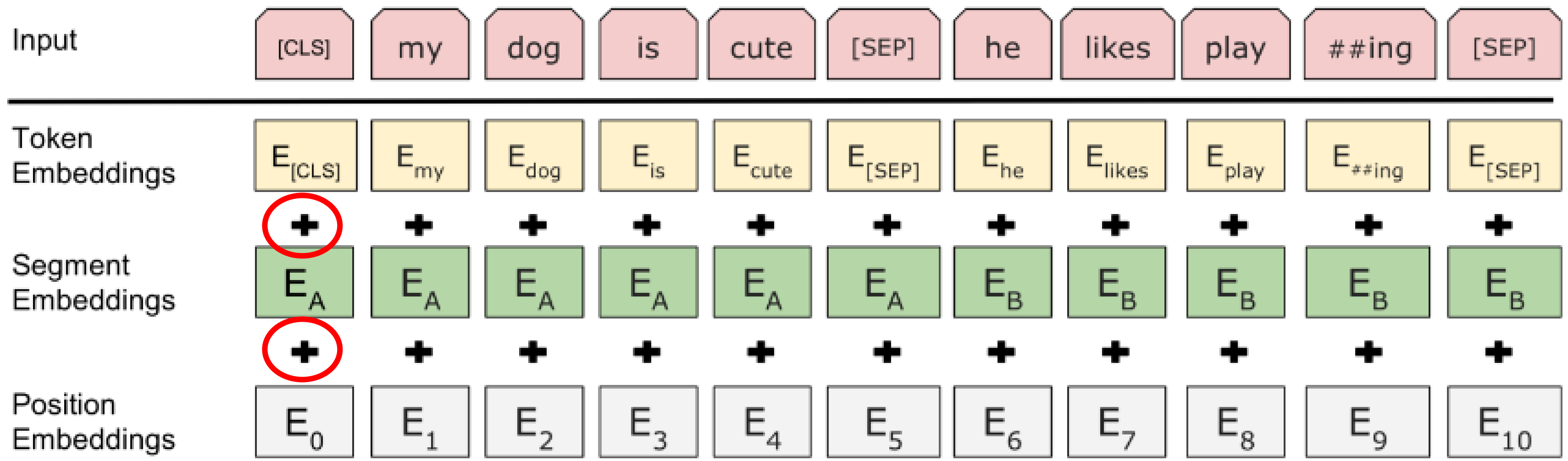
thar proceeds sentence A, or a random sentence

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`

BERT

Input representation - sentence pair encoding



use summed representation

token Embedding : word piece

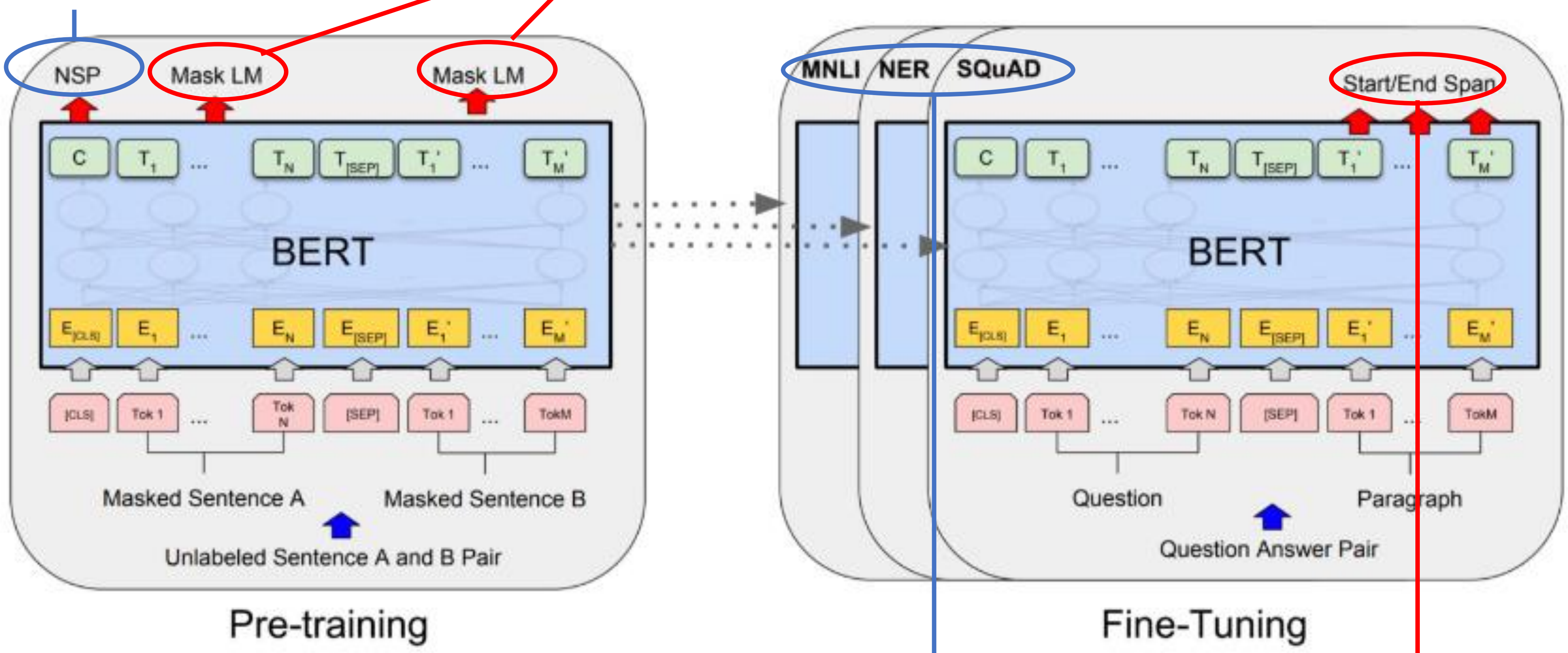
segment Embedding : for each word piece whether it comes from the first/second sentence before/after the separator

Positional Embedding : for other Transformer architectures

BERT

binary prediction function as to whether a correct next sentence or not

losses to extent that you can't predict the masked words



useful for various tasks

substitute with a final prediction layer

BERT

Train 2 model sizes:

BERT-Base : 12-layer, 12-head, 768-hidden

BERT-Large : 24-layer, 1024-hidden, 16-head

Use transformer encoder

- self-attention => no locality bias
- single multiplication per layer => efficiency on GPU/TPU

BERT

transformer encoder