

Understanding **Transformer Model**

Attention is all you need

00 Outline

- **Attention in seq2seq**
- **Self-Attention**
- **Multi-headed Attention**
- **Matrix Flow**
- **Sample Code**

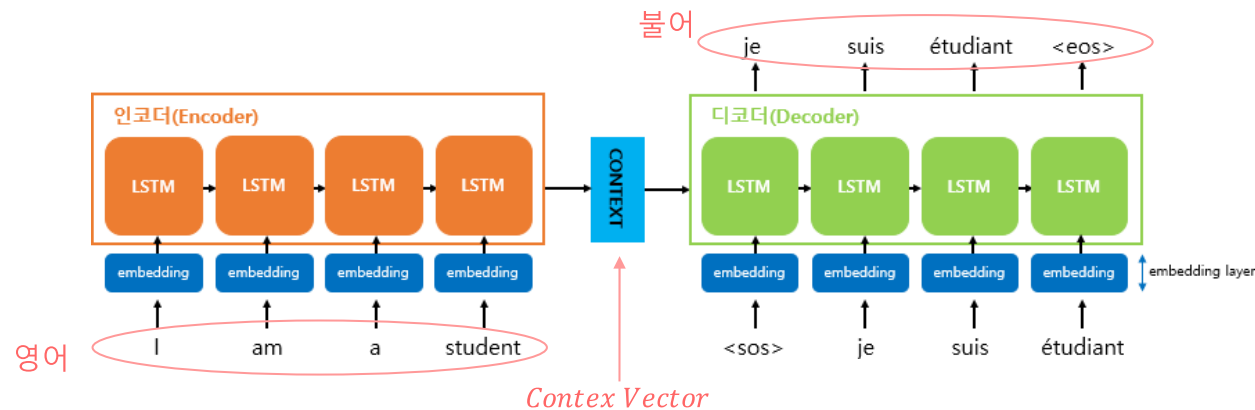
Attention in Seq2Seq (복습)

... transfer a sequence to another sequence ...

1 Attention 개념

seq2seq 모델의 단점

- 인코더에서 입력 sequence 전체에 대한 정보를 context vector 한개에 압축하여 디코더로 전달
- 디코더는 단 하나의 context vector에 의존하여 출력 Sequence 생성
- 한개의 context vector에 모든 정보를 담아야 하므로 정보 손실 발생 (정확성 저하)
- RNN의 근본적인 약점인 Vanishing Gradient (기울기 소실) 문제 상존



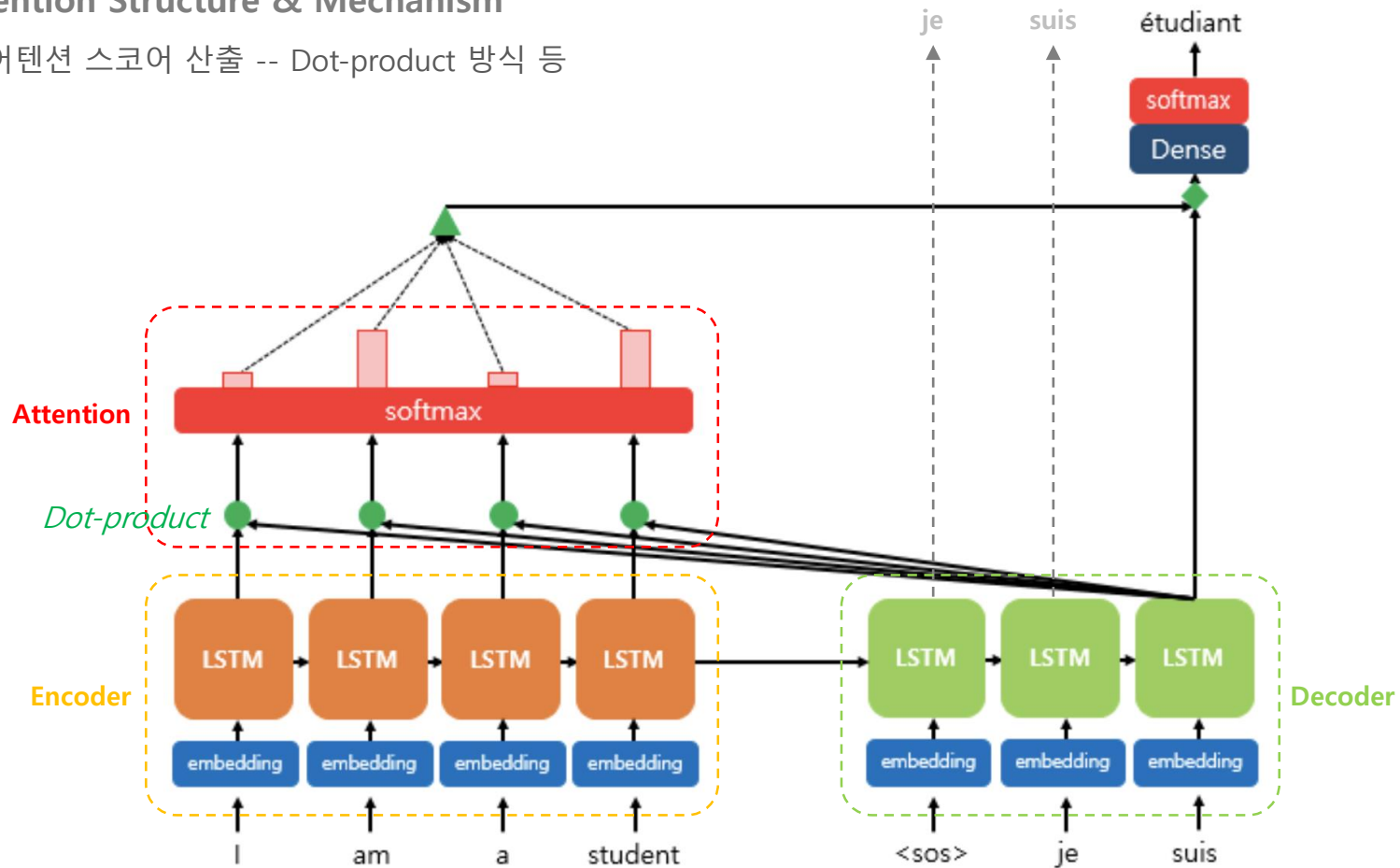
Attention 개념

- 디코더에서 출력 단어를 예측하는 매 시점마다 인코더에서의 전체 입력 문장을 다시 한 번 참고
- 각 시점에서 예측해야 할 출력 단어와 연관있는 입력 단어를 집중(어텐션)하여 참고함
(모든 입력단어를 동일한 비율로 참고하는 것이 아님)

2 Attention 개념

Attention Structure & Mechanism

- 어텐션 스코어 산출 -- Dot-product 방식 등

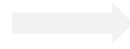


3 RNN-based seq2seq

General issues on translation

- Different word order

I love you
└─┘ └─┘ └─┘
S V O



난 널 사랑해
└─┘ └─┘ └─┘
S O V

- Different sentence length

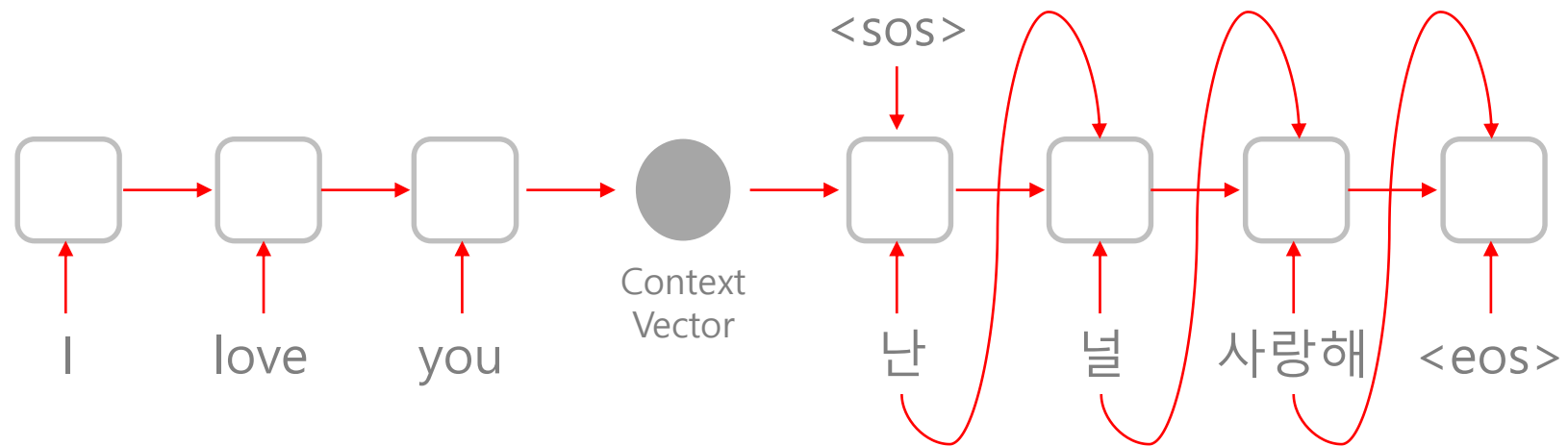
How are you ?
└──────────┘
3 words



잘 지내니 ?
└────────┘
2 words

4 RNN-based seq2seq

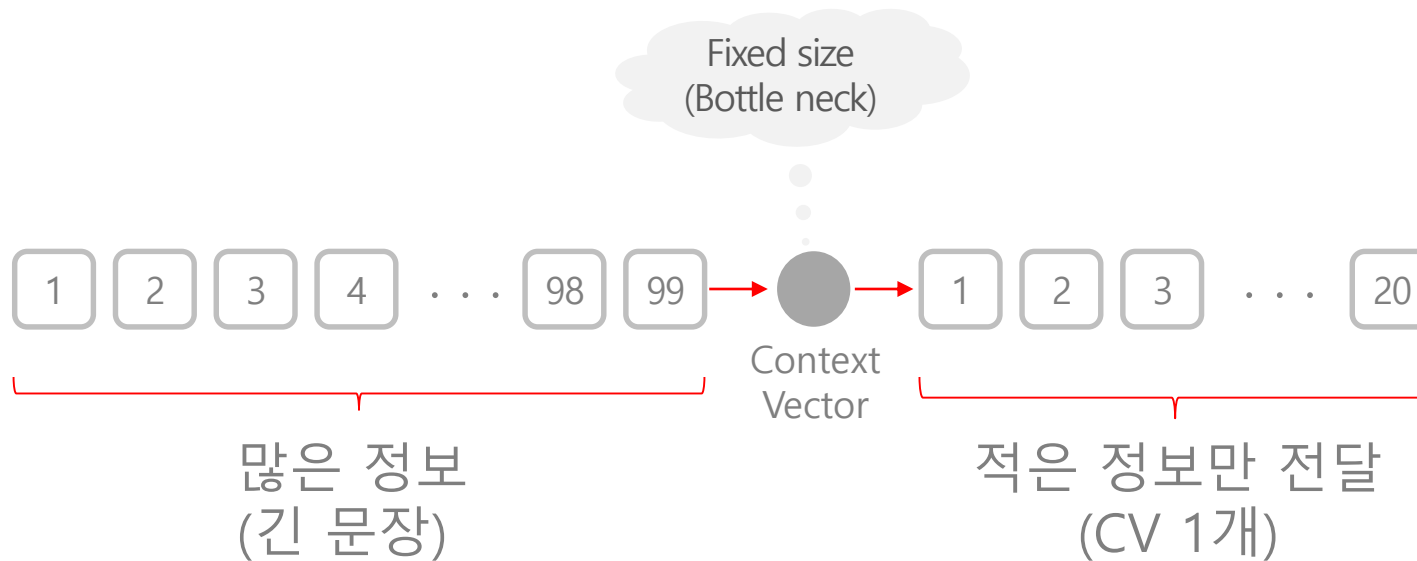
Basic seq2seq using RNN blocks



- RNN 구조를 이용하여 **길이**와 **순서**의 차이를 어느 정도 극복

5 RNN-based seq2seq

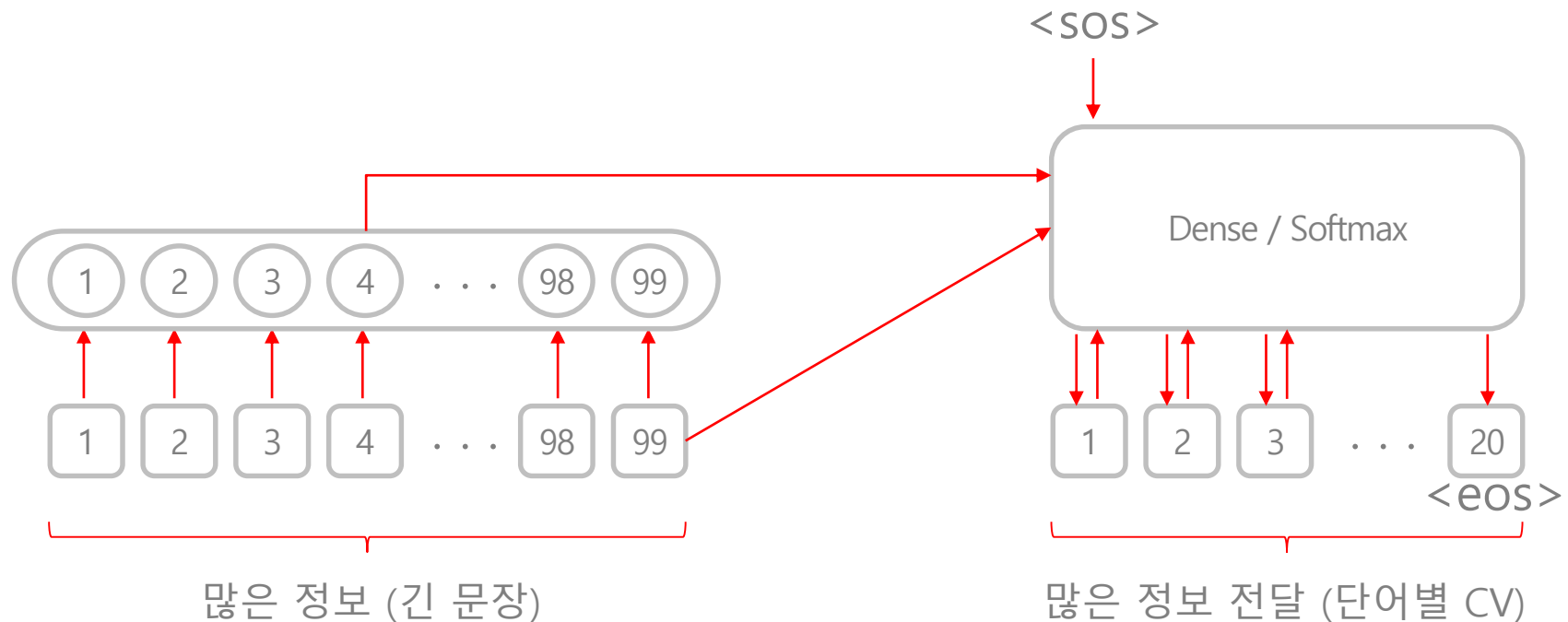
A **context vector** can be a bottleneck



- 한개의 context vector에 모든 정보를 담아야 하므로 정보 손실 발생 (정확성 저하)

6 RNN-based seq2seq with attention

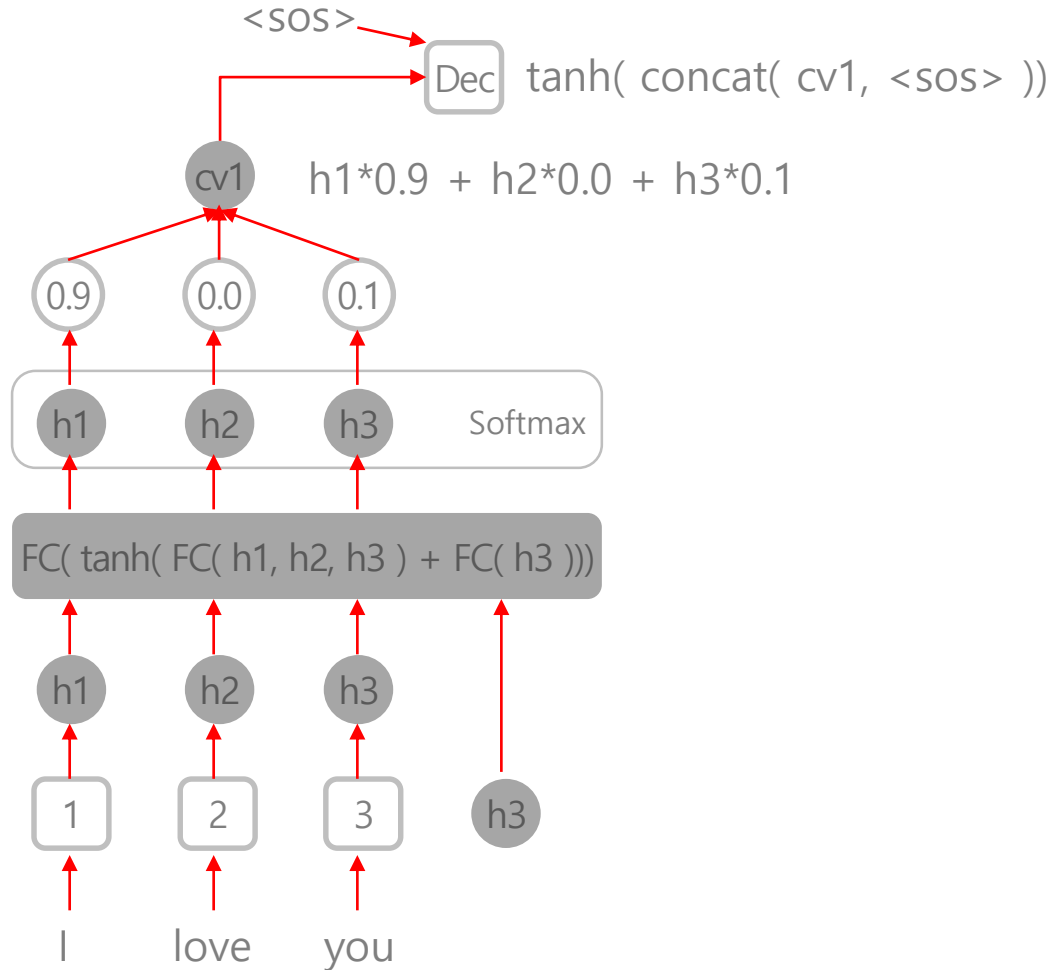
Dynamic **context vectors** for each **step** of translation (for each word)



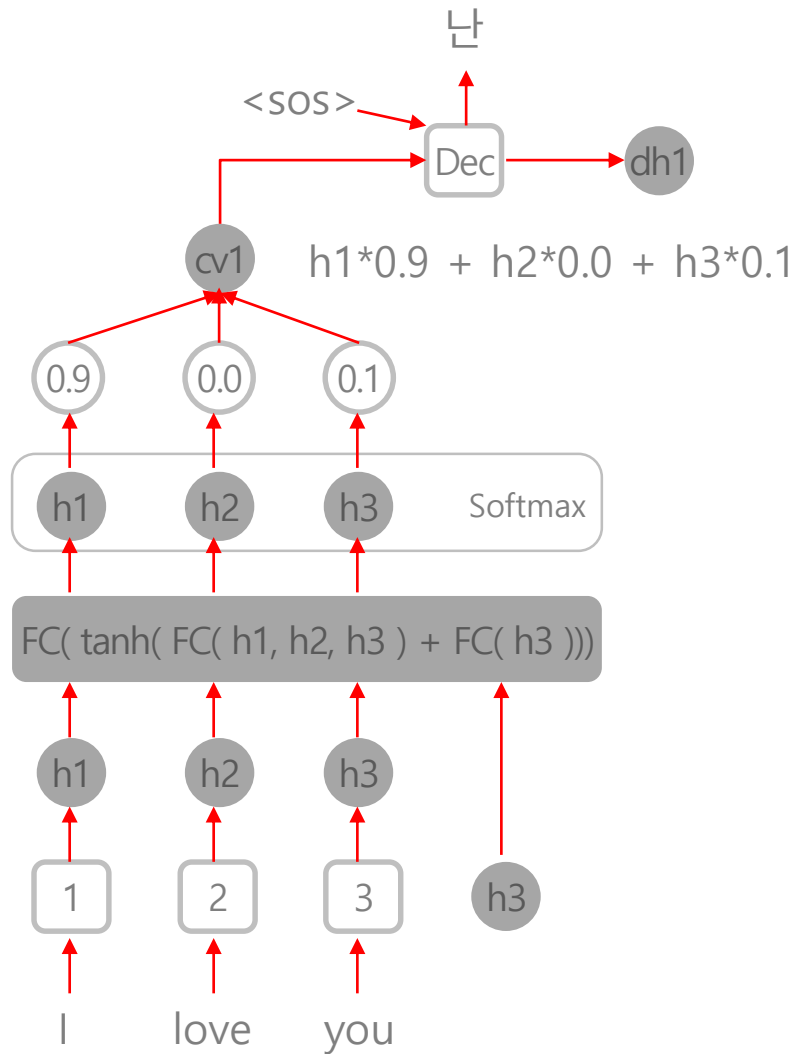
- 전체 시퀀스 정보를 하나의 최종 CV로 변환하는 것이 아니라 단계별로 다이내믹한 CV로 변환
- 디코딩 시에 각 단계별로 그 time-step에 해당하는 CV를 적용

7 RNN-based seq2seq with attention

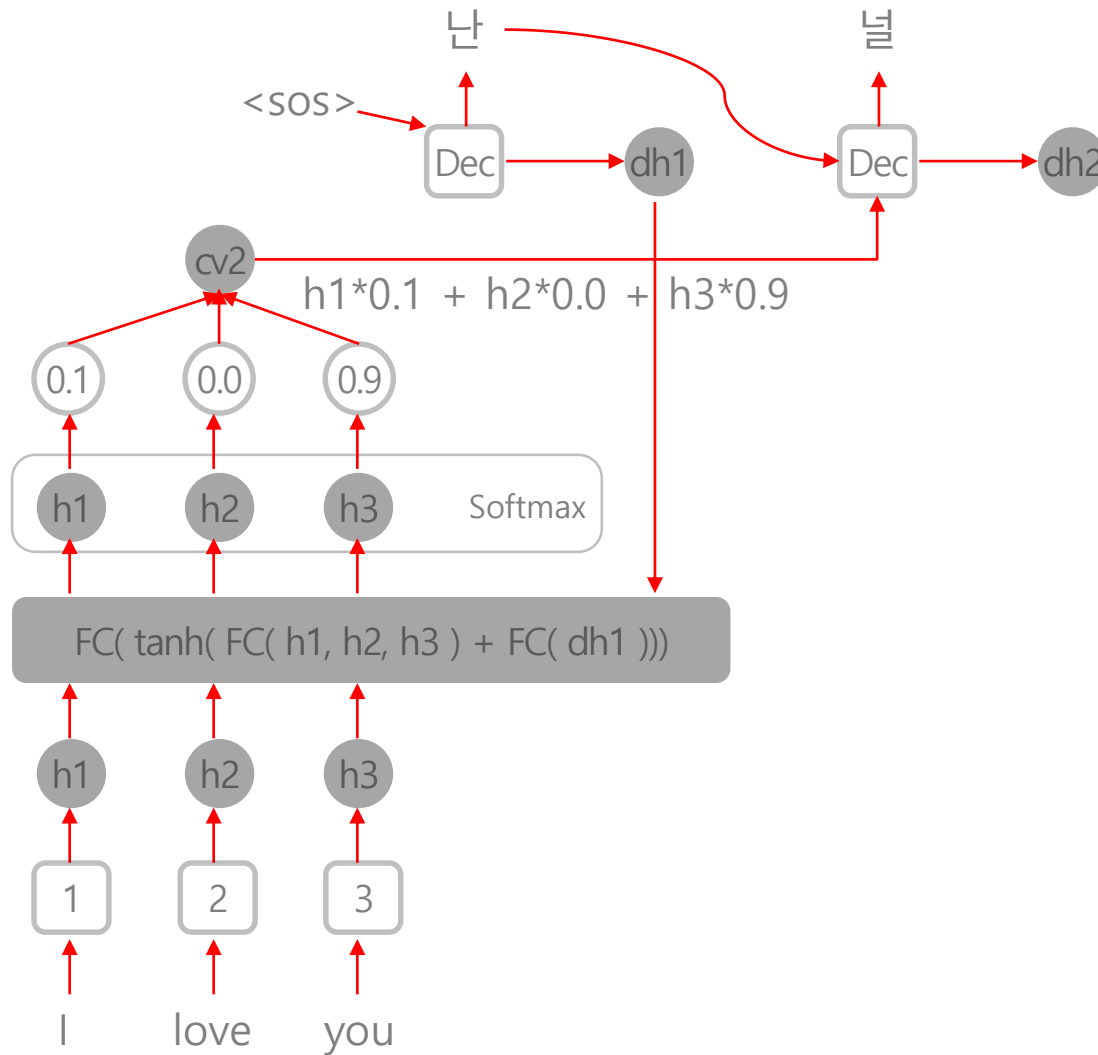
Dynamic **context vectors** for each **step** of translation (for each word)



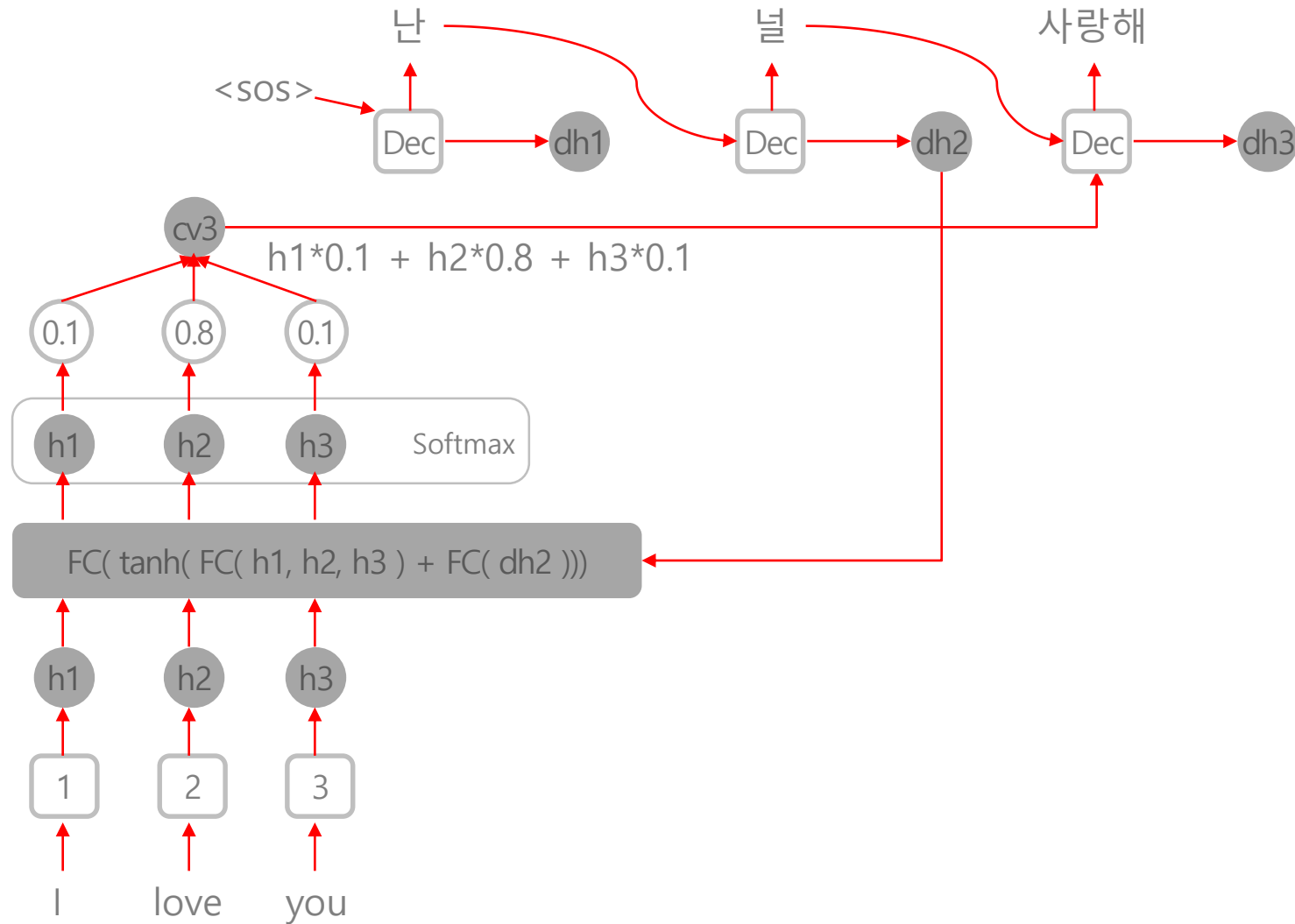
8 RNN-based seq2seq with attention



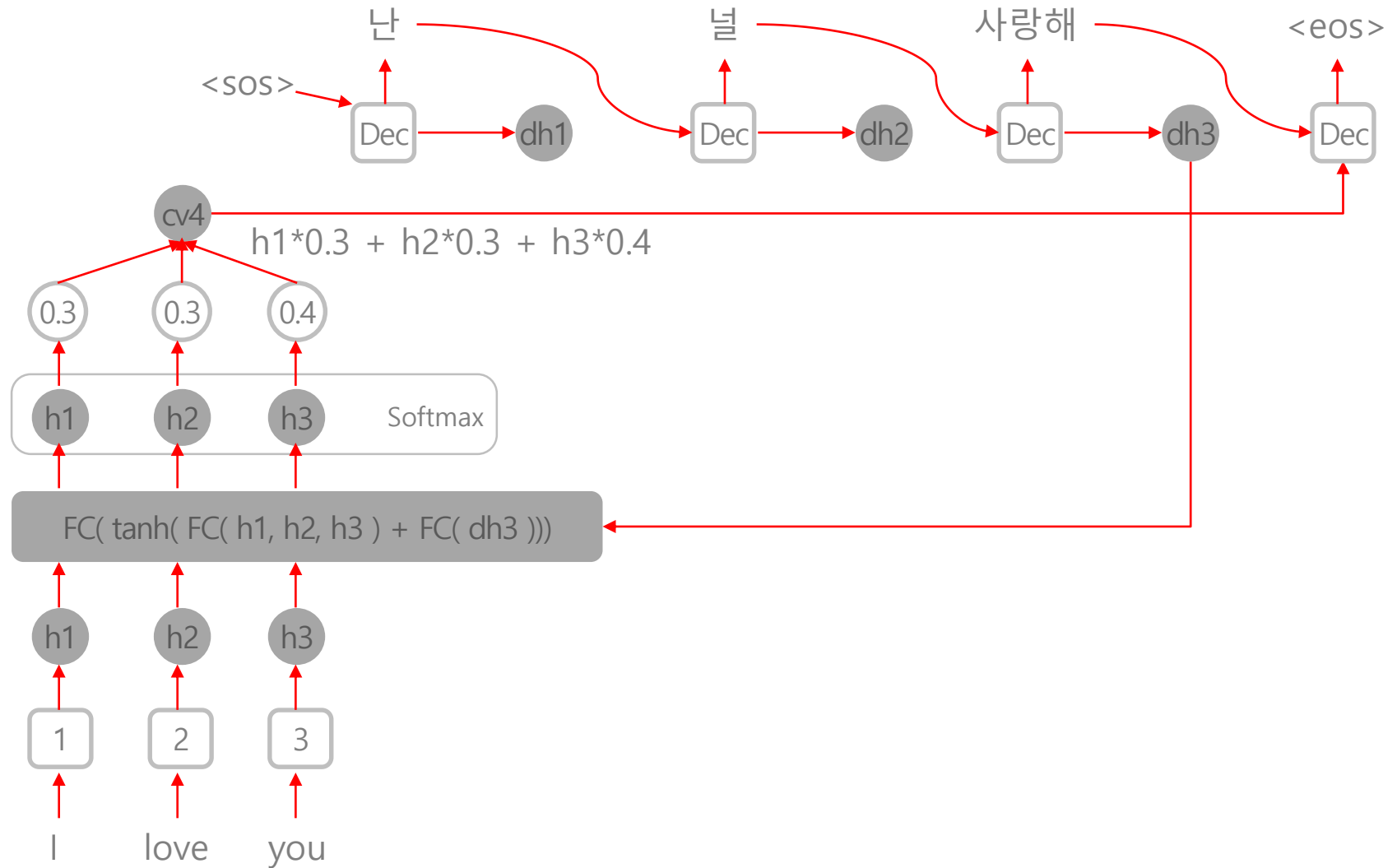
9 RNN-based seq2seq with attention



10 RNN-based seq2seq with attention



11 RNN-based seq2seq with attention



12 RNN-based seq2seq with attention

Attention successfully translates **longer sentences** without degradation

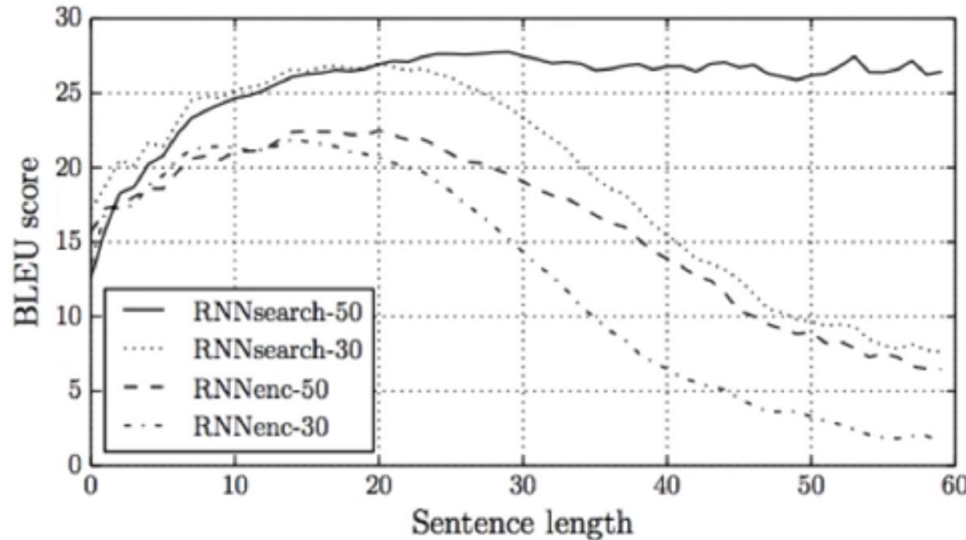


Figure 2: The BLEU scores of the generated translations on the test set with respect to the lengths of the sentences. The results are on the full test set which includes sentences having unknown words to the models.

- RNNsearch# : Attention 적용 모델
- RNNenc# : Attention 개념 없이 기본적인 RNN만으로 구성된 모델
- RNN***30 or RNN***50 : 30 또는 50개 단어 이하로 구성된 문장을 학습 데이터로 사용
- BLEU : 예측된 문장과 target 문장이 얼마나 같은지를 알려주는 성능지표

Attention is all you need

... based solely on attention mechanism ...

01 “Attention is all you need”(*)

(*) 2017년 구글에서 발표한 논문, 모델 이름은 Transformer, SOTA 대비 BLEU 최대 2점 향상, 훈련량 1/3로 축소

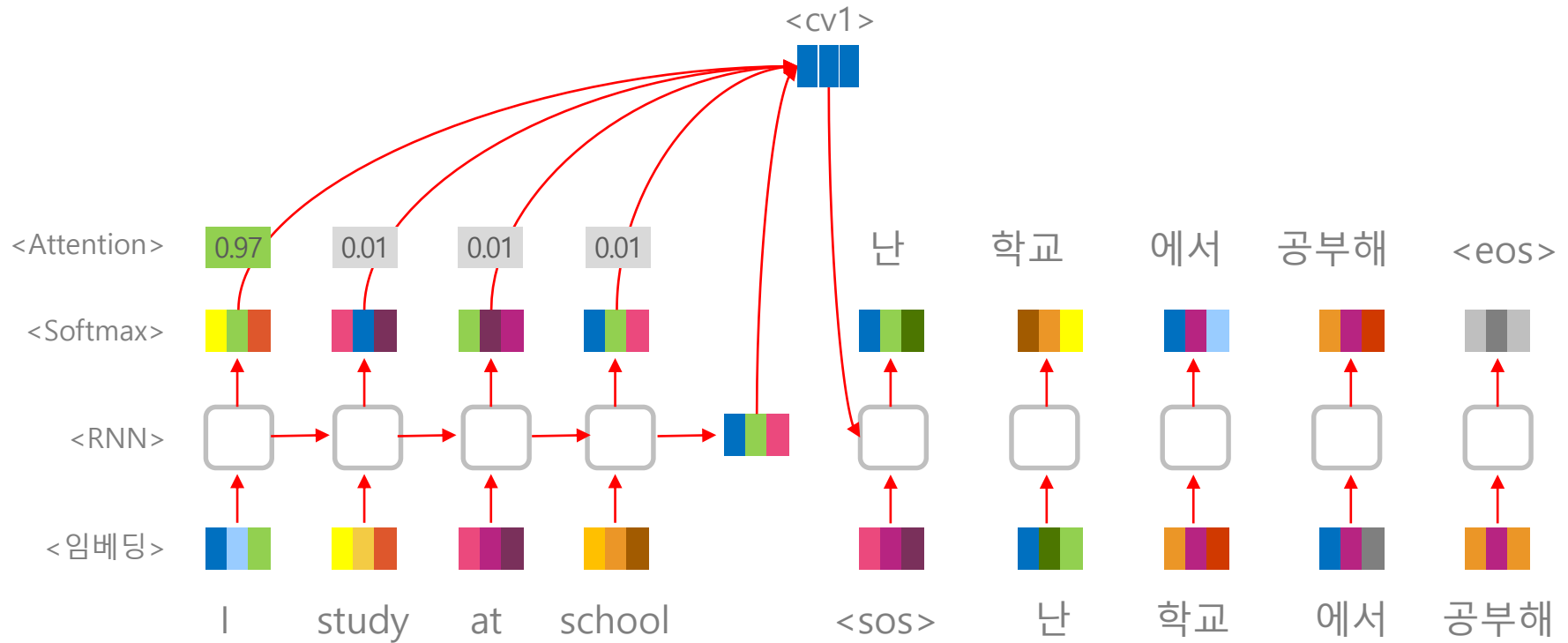
Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$3.3 \cdot 10^{18}$	
Transformer (big)	28.4	41.8	$2.3 \cdot 10^{19}$	

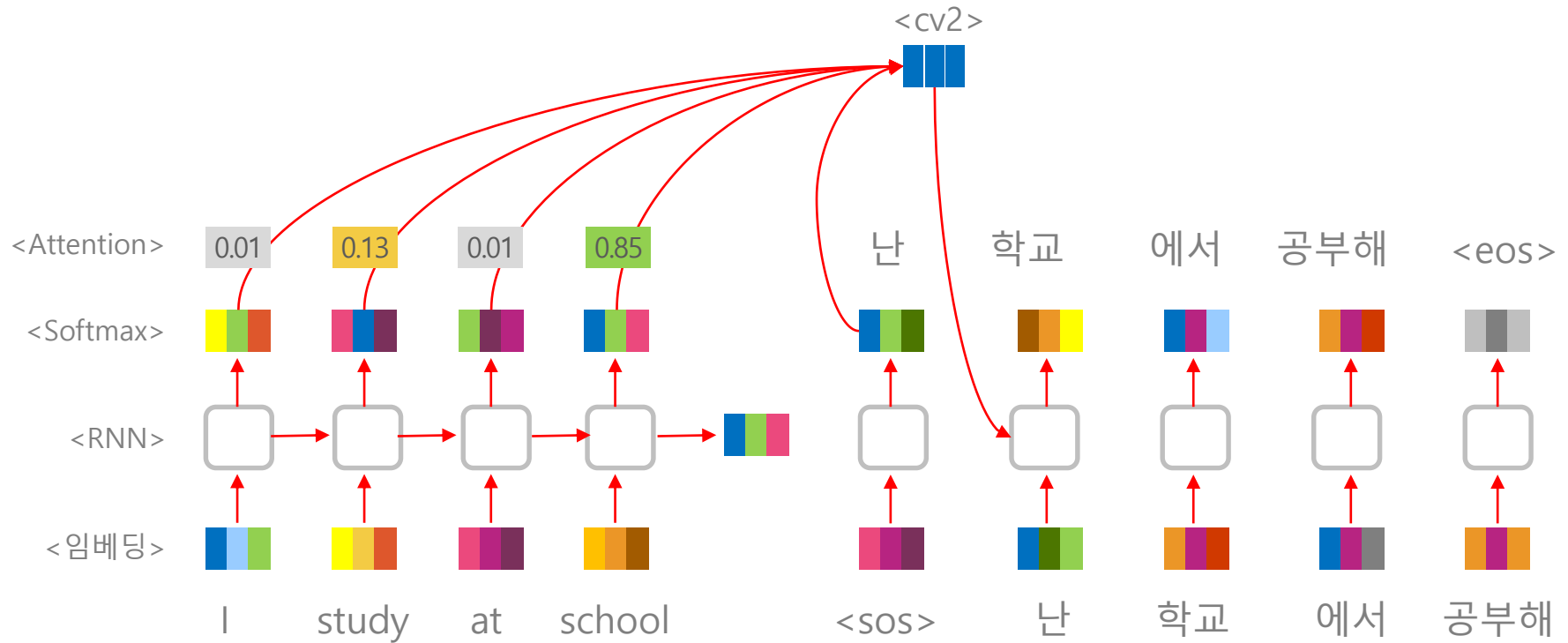
- RNN 계층 제거, CNN 기술도 사용 안함
- Attention 계층만 사용하여 학습시간 단축 / 성능 향상
- 전체 계산량 축소 / 병렬 처리 가능한 영역 확대
- WMT 2014 EN-DE 번역 성능: 28.4 BLEU
- WMT 2014 EN-FR 번역 성능: 41.8 BLEU
(GPU 8대 사용, 3.5 일 동안 훈련)

- BLEU (Bilingual Evaluation Understudy): 기계번역의 Quality를 평가하는 지수
- WMT (Workshop on Statistical Machine Translation): 기계번역 워크숍에서 제공하는 번역 데이터 셋

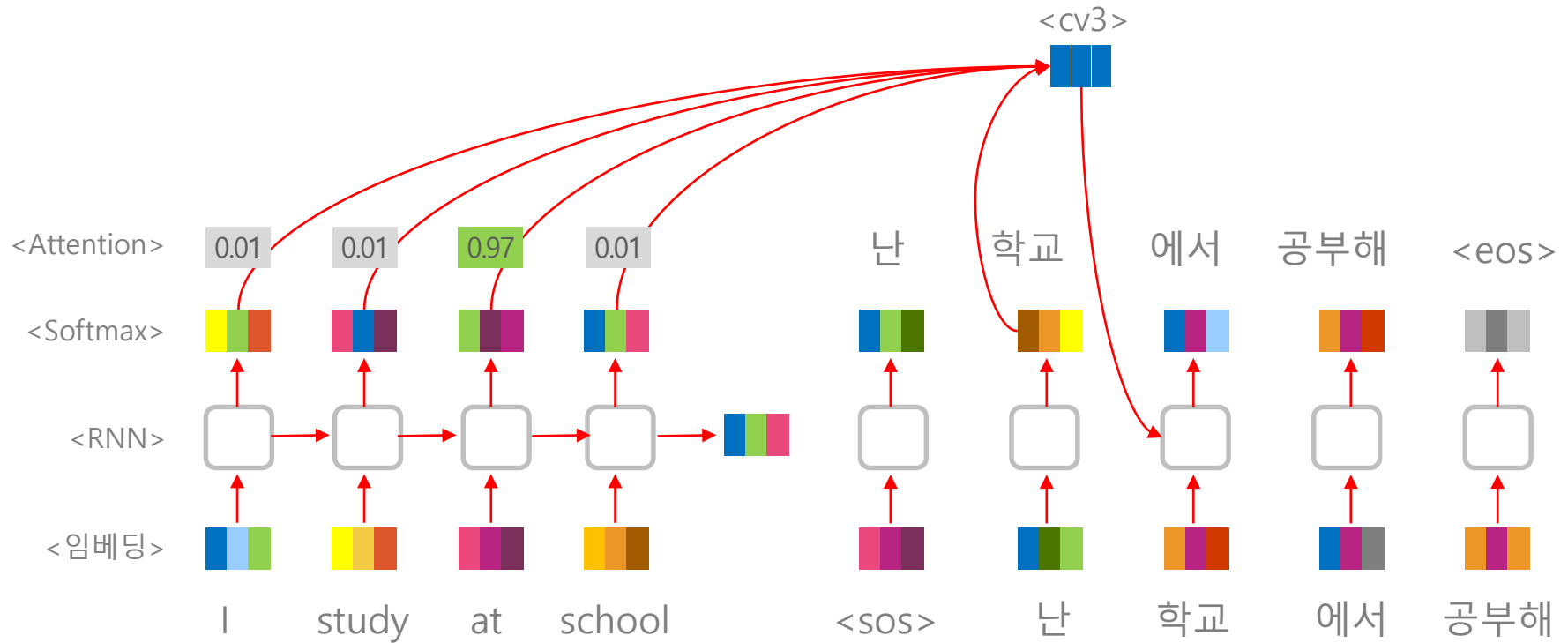
01 RNN removal



01 RNN removal

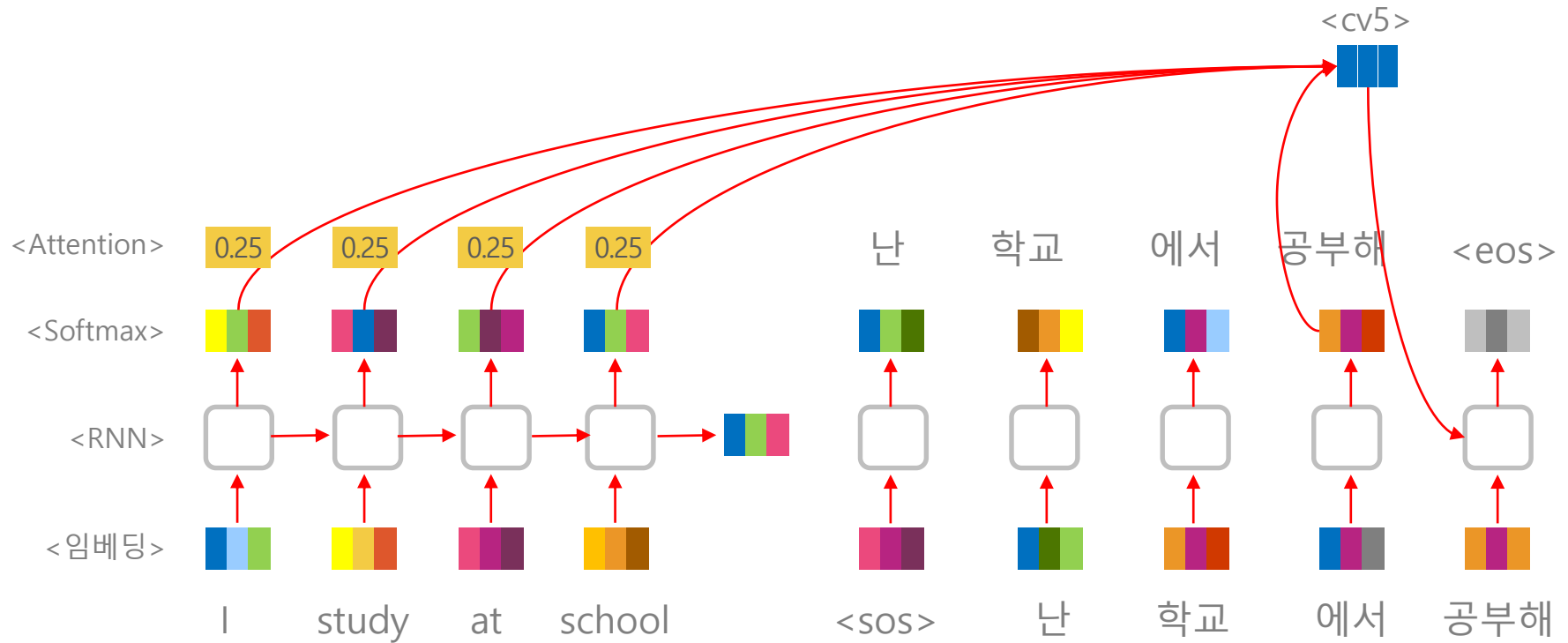


01 RNN removal



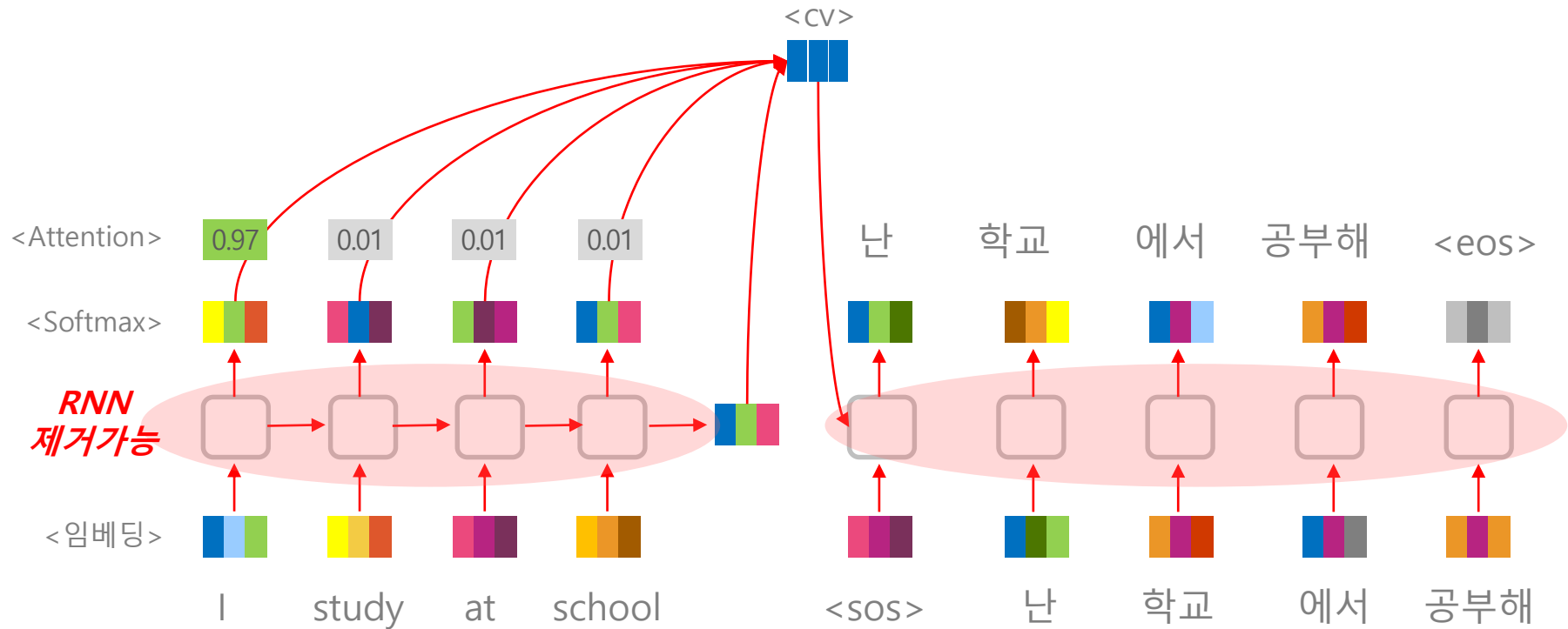


01 RNN removal



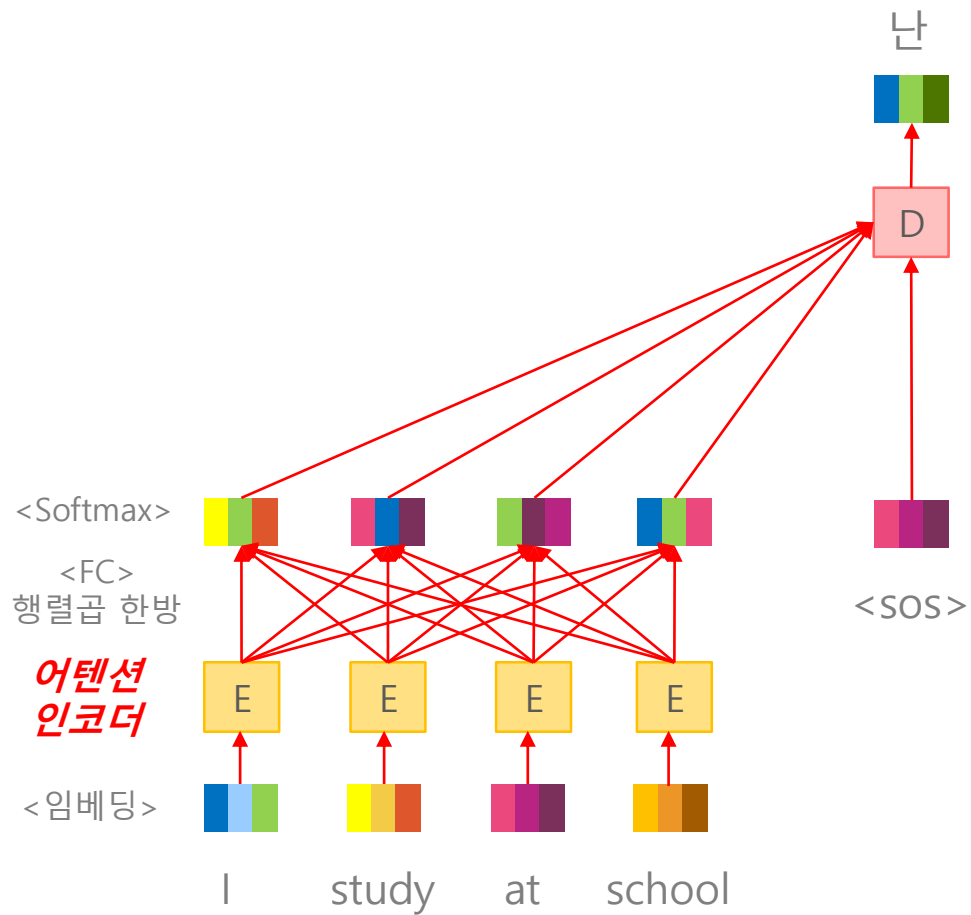
01 RNN removal

이 그림에서 RNN 계층이 굳이 필요한가?

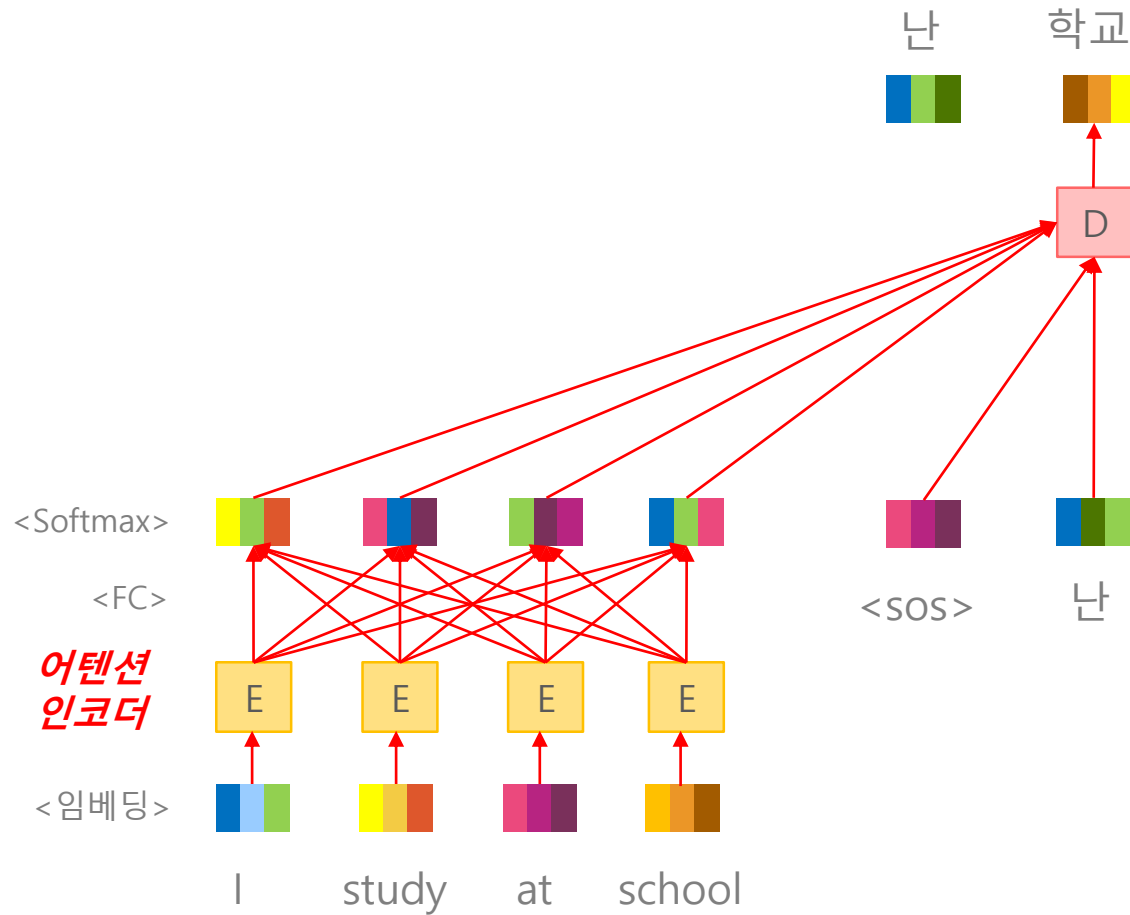


- 남아있는 <임베딩>, <Softmax>, <Attention> 등을 잘 조합하여 단순화 → **Transformer** 모델 구현
- RNN 계층 제거로 계산량이 줄고 Sequence 간에 연결이 없어지므로 병렬처리 가능

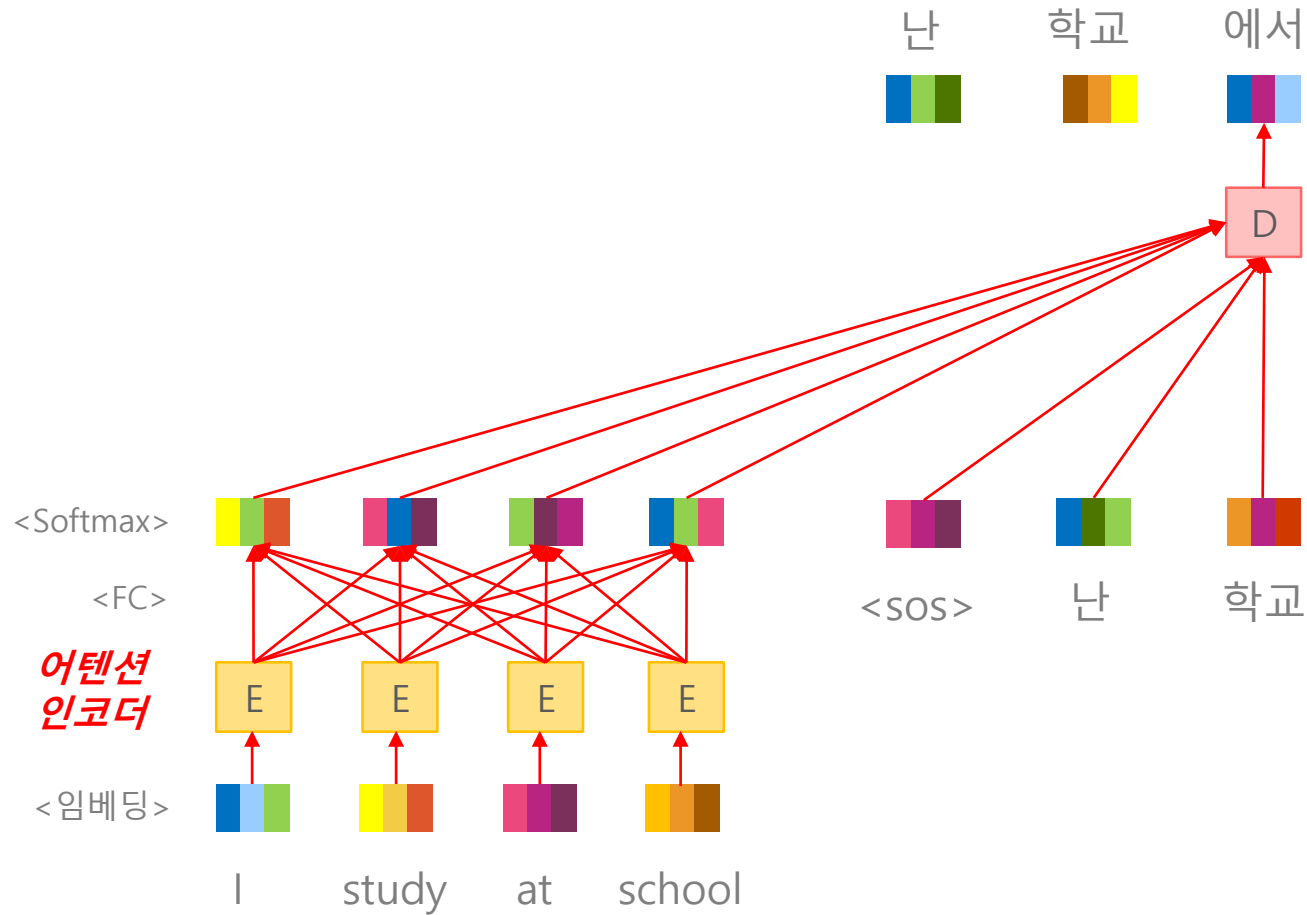
01 Without RNN



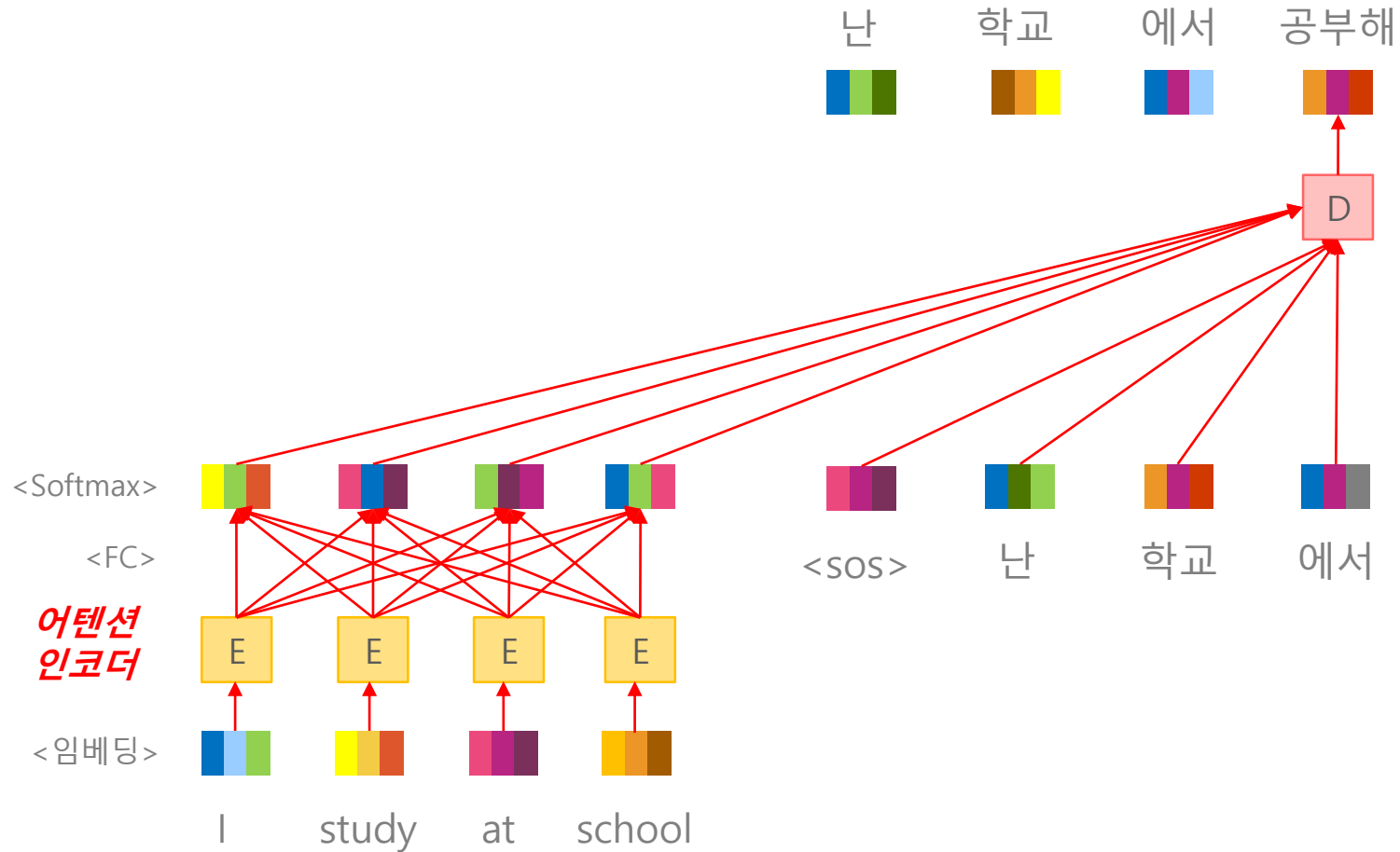
01 Without RNN



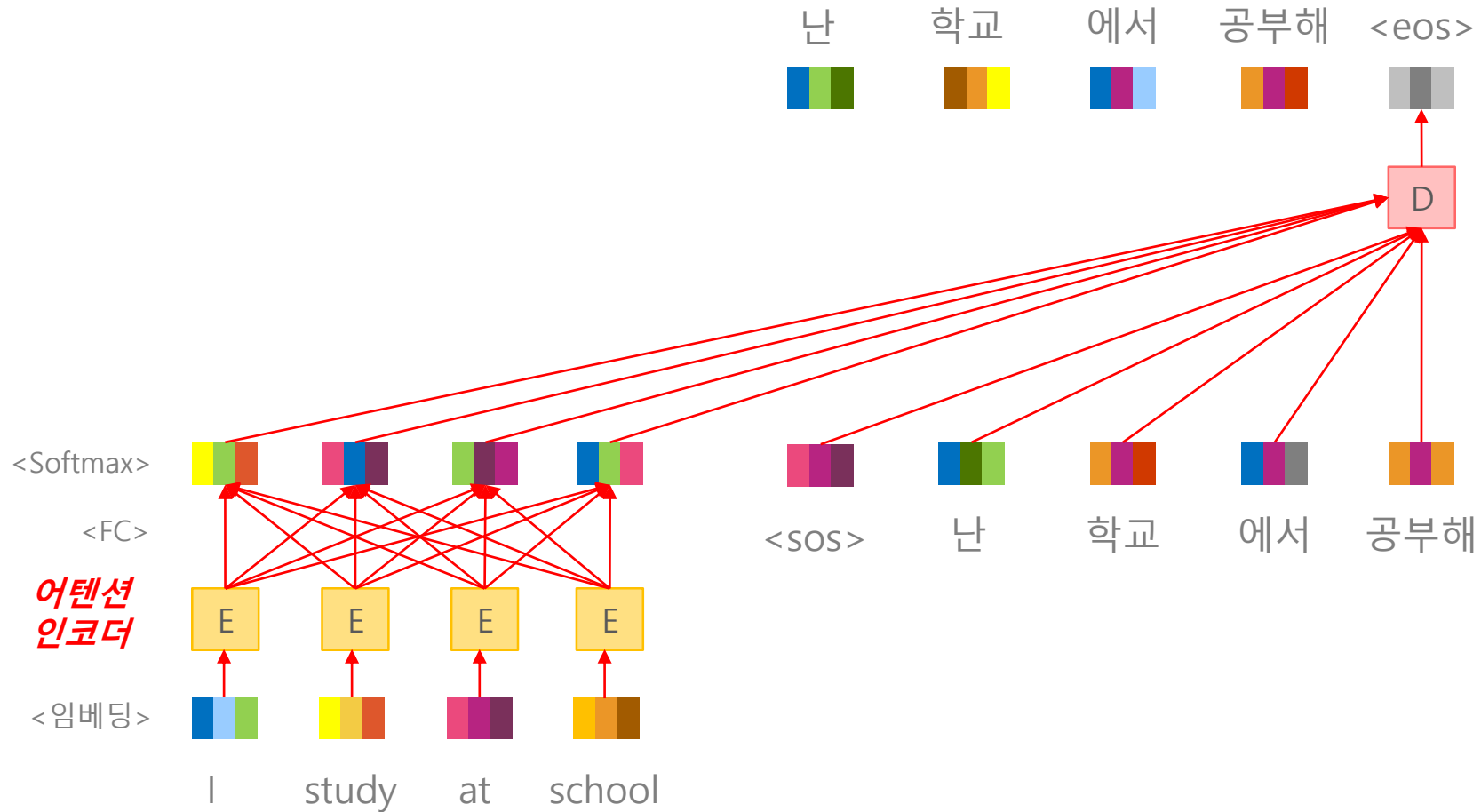
01 Without RNN



01 Without RNN

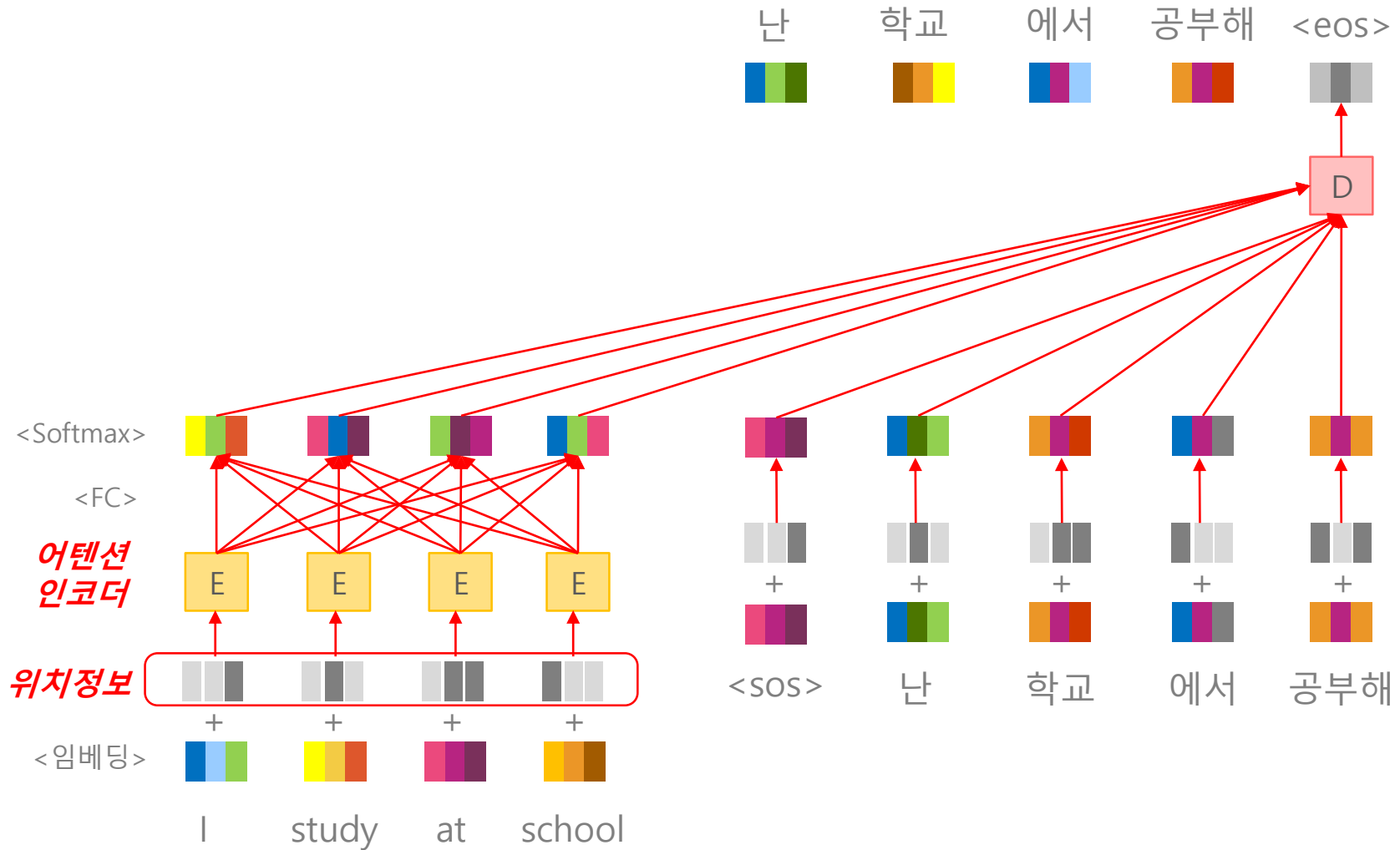


01 RNN Without RNN

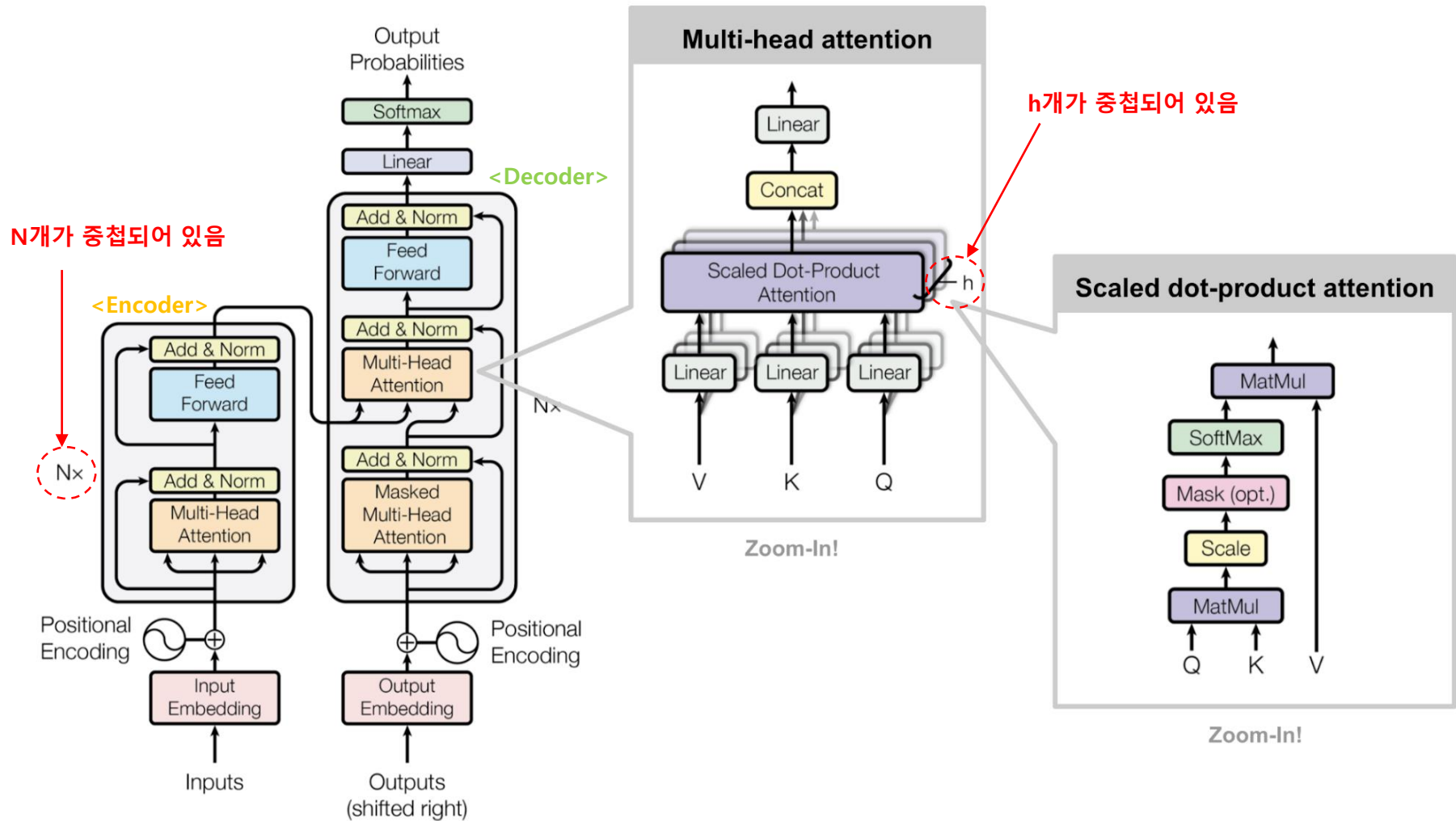


01 Positional Encoding

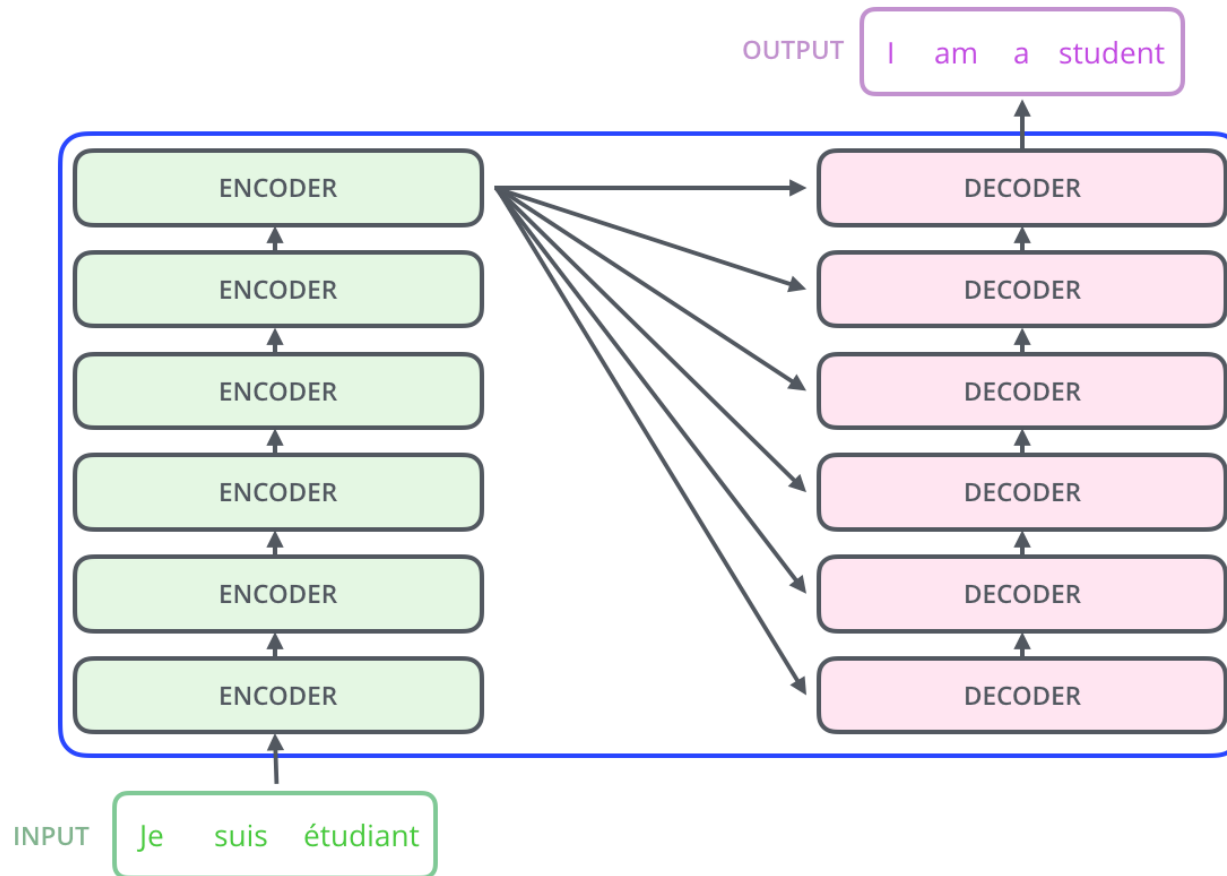
RNN에 내재되었던 **순서** 개념을 추가



01 Transformer Architecture

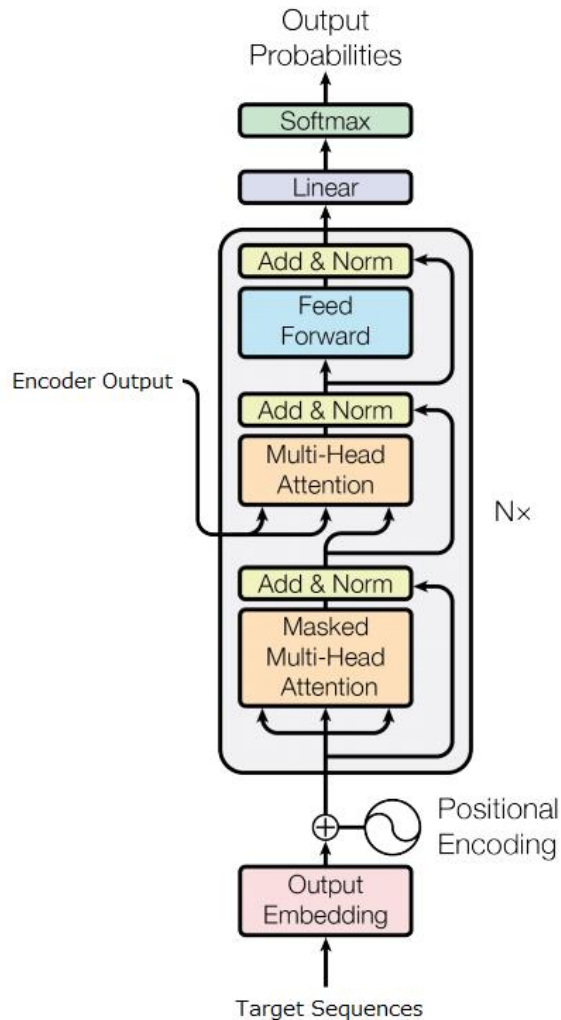


01 Layered encoder/decoder



- "Attention is all you need" 원 논문에는 인코더와 디코더를 6층씩 쌓아 올렸음
- 각 계층의 출력 벡터의 크기가 통일되어 있으므로 필요에 따라 더 많이 쌓을 수도 있음

01 Basic Components

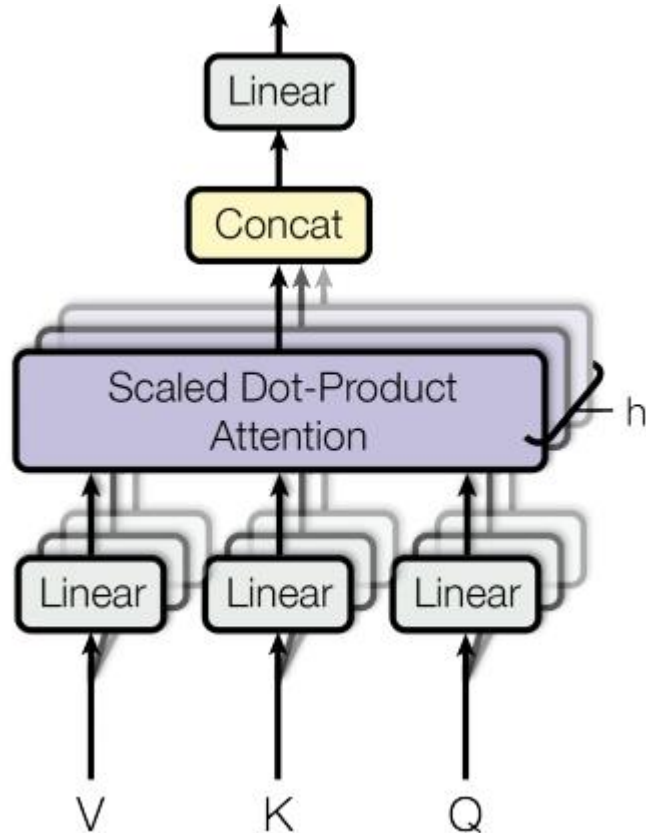


- **Multi-Head Attention:** Self-attention 기능을 수행. 특정 단어가 입력 문장 내의 다른 모든 단어들과의 연관성을 수치로 표현
- **Masked Multi-Head Attention:** 디코딩을 위한 Self-Attention시 자신의 time step 이후 word는 가려 Self-Attention 되는 것을 막는 역할. 현재까지 디코딩된 단어들만 적용하고 아직 출력되지 않은 단어들은 attention에 포함시키지 않음
- **Add & Normalization:** Residual Connection을 통해 들어오는 원본 값을 더한 후 정규화를 수행해 줌으로써 gradient가 exploding 또는 vanishing 하는 문제 예방

$$\text{LayerNorm} (x + \text{Sublayer} (x))$$
- **Feed Forward:** Multi-Head Attention에서 각 head가 자신의 관점으로 문장을 self-attention한 결과를 균등하게 섞어 줌. 두개의 linear transformation으로 구성

$$\text{FFN}(x) = \text{relu}(xW_1 + b_1)W_2 + b_2, x \text{는 각 head별 출력 값}$$
- **Input Embedding:** 임베딩 알고리즘을 이용하여 각 단어를 벡터로 변환 (단어 당 512바이트 크기의 word embedding 적용)
- **Positional Encoding:** 단어가 문장내 어느 위치에 배치되어 있는가를 표현 (원 논문에서는 단어의 출현 순서가 짝수냐 홀수냐에 따라 \sin 또는 \cos 함수를 각각 사용함)

01 Multi-Head Attention



- 각 단어에 대해 Q, K, V 벡터를 생성하고 여러장의 head로 나누어 scaled dot-product attention을 구함
- 먼저 embedding word vector들을 Q, K, V 벡터로 구성
- 각 벡터는 여러 개의 부분으로 나뉘어 head별로 할당됨

[원 논문의 구성]

- 단어들은 512바이트로 embedding
- Q, K, V 벡터도 이 값을 그대로 받아 512바이트로 구성
- 그것을 64바이트 크기로 나누어서 8개로 분리한 뒤 각 조각을 8개의 head에 할당 ($h=8$)
- Scaled dot-product attention이 완료되면 head 단위로 결과를 concatenation
- 연결된 결과를 Fully connect 형식으로 섞어 준 후 Linear 연산

01 Query, Key, Value

01 Attention Mechanism

- 디코더의 다음 상태 S_i 계산식:

$$S_i = f(S_{i-1}, y_{i-1}, c_i)$$

- S_{i-1} : 디코더의 이전 상태
- y_{i-1} : 이전 단어 (디코딩된 것 또는 정답지)
- c_i : 현재 단계에서의 context vector

- 각 단계(i) 별 context vector c_i :

$$e_{ij} = a(S_{i-1}, h_j)$$

$$\alpha_{ij} = \frac{e_{ij}}{\sum_{k=1}^n e_{ik}}$$

$$c_i = \sum_{k=1}^n \alpha_{ik} h_k$$

- a : 유사성 계산 함수 (dot product 형태)
- h_j : 현재 인코더측 히든 노드의 출력
- e_{ij} : 이전 디코더 값과 현재 인코더 값 간의 유사도
- α_{ij} : 벡터 e_{ij} 에 대한 softmax 값

01 Attention Type

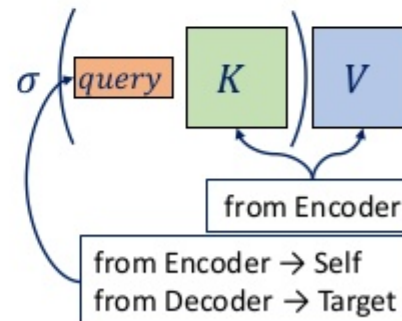
Source-Target or Self Attention

■ 2 types of Dot-Product Attention

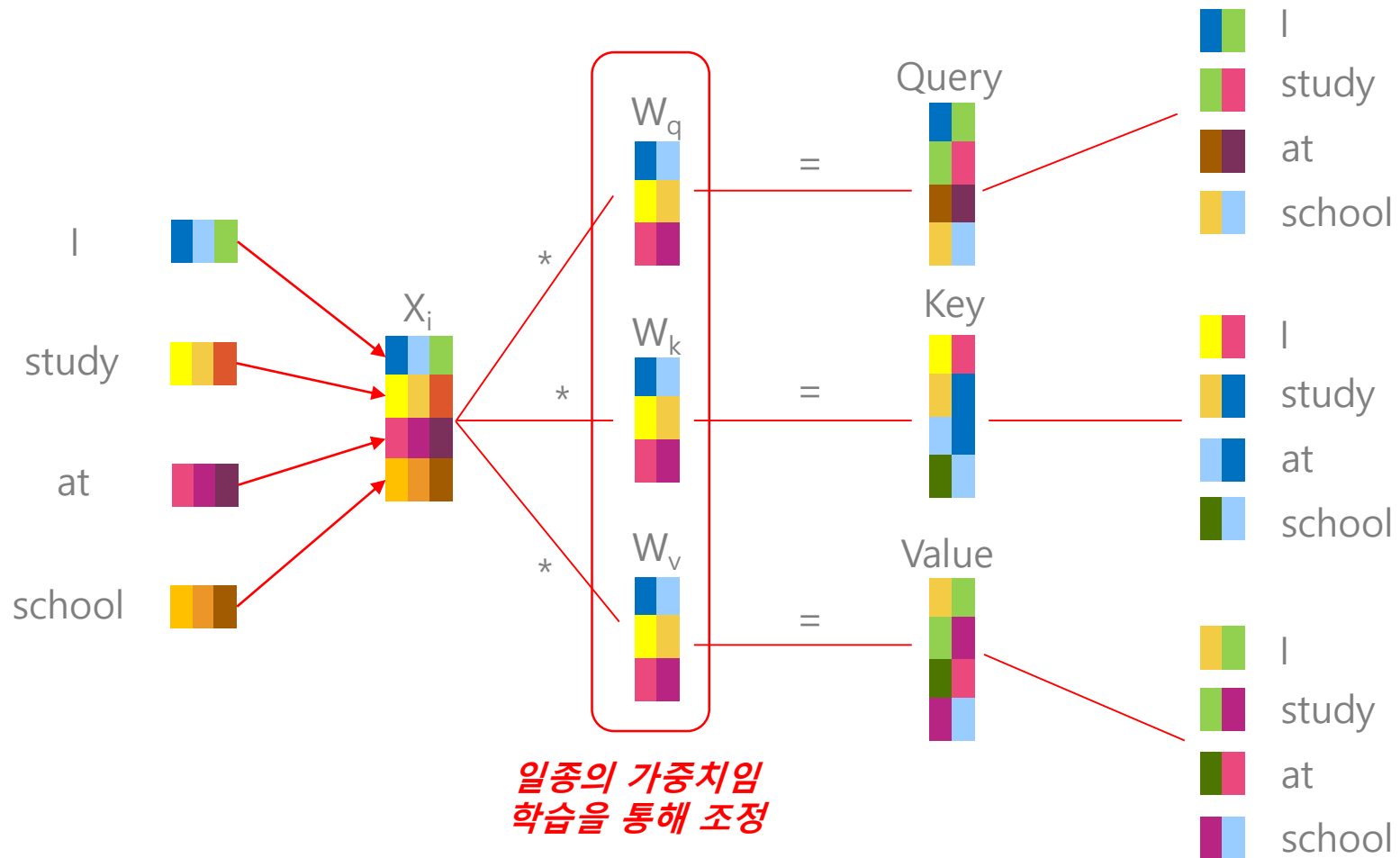
- Source-Target Attention
 - Used in the 2nd Multi-Head Attention Layer of Transformer Decoder Layer
- Self-Attention
 - Used in the Multi-Head Attention Layer of Transformer Encoder Layer and the 1st one of Transformer Decoder Layer

■ What is the difference?

- Depends on where query comes from.
 - query from Encoder → Self-Att.
 - query from Decoder → Source-Target Att.



















01 Self Attention











01 Self Attention

asdf

	$\langle X_i \rangle$	$\langle \text{Query} \rangle$	$\langle \text{Key} \rangle$	$\langle \text{Value} \rangle$
I				
study				
at				
school				

01 Self Attention


















asdf

	Query	*	Key ^T	Score	Softmax
I * I		*		= 130	0.92
I * study		*		= 50	0.05
I * at		*		= 20	0.02
I * school		*		= 10	0.01

$$\text{softmax}\left(\frac{\text{Score}}{\text{sqrt}(\text{dimension of key})}\right)$$

01 Self Attention

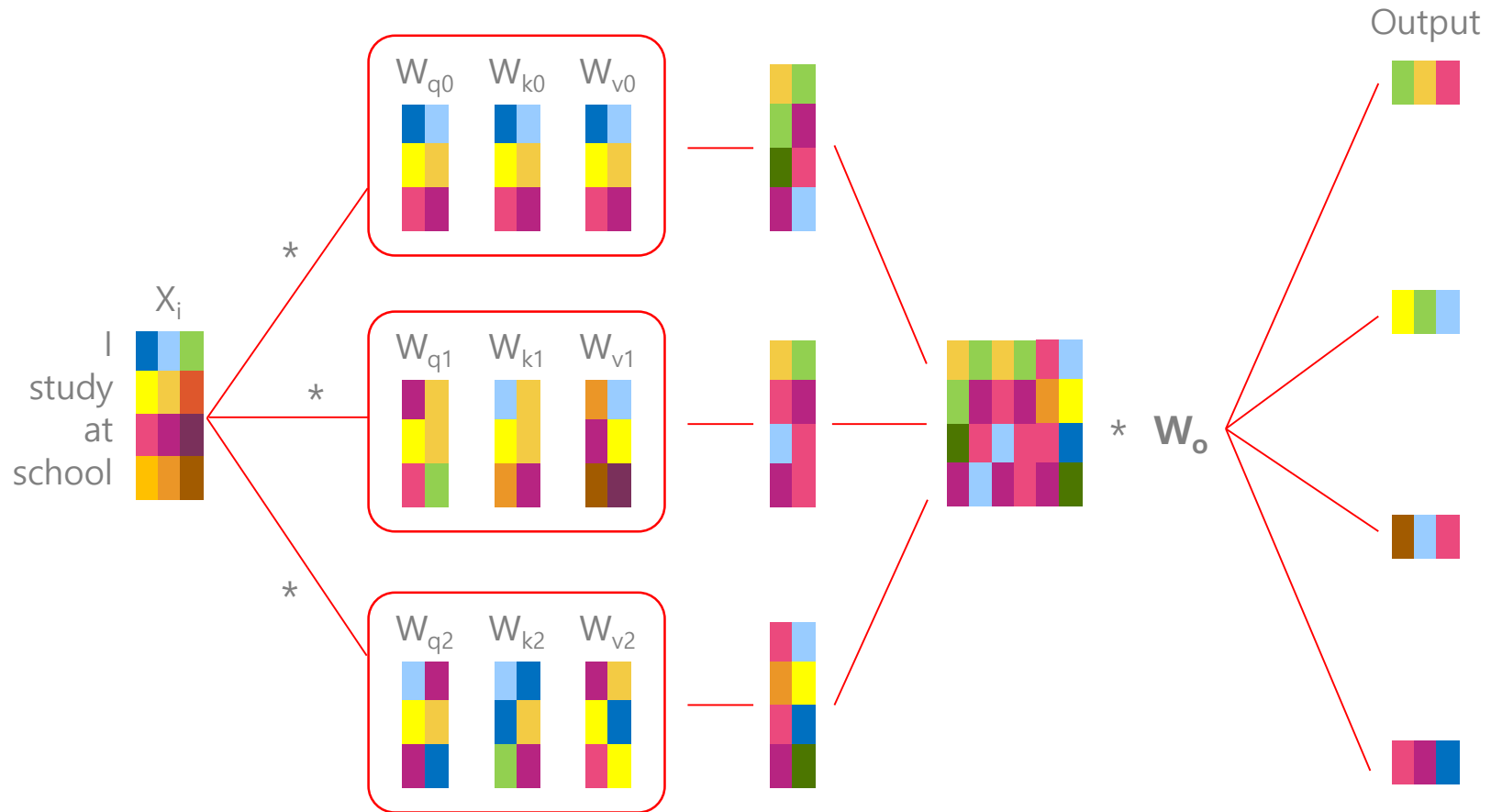
asdf

	Query	*	Key ^T	Score	Softmax	Value	Softmax *	Σ Softmax *
							Value	Value
I * I		*		= 130	0.92	I 		
I * study		*		= 50	0.05	study 		
I * at		*		= 20	0.02	at 		
I * school		*		= 10	0.01	school 		
								(Attention layer output)

01 Self Attention

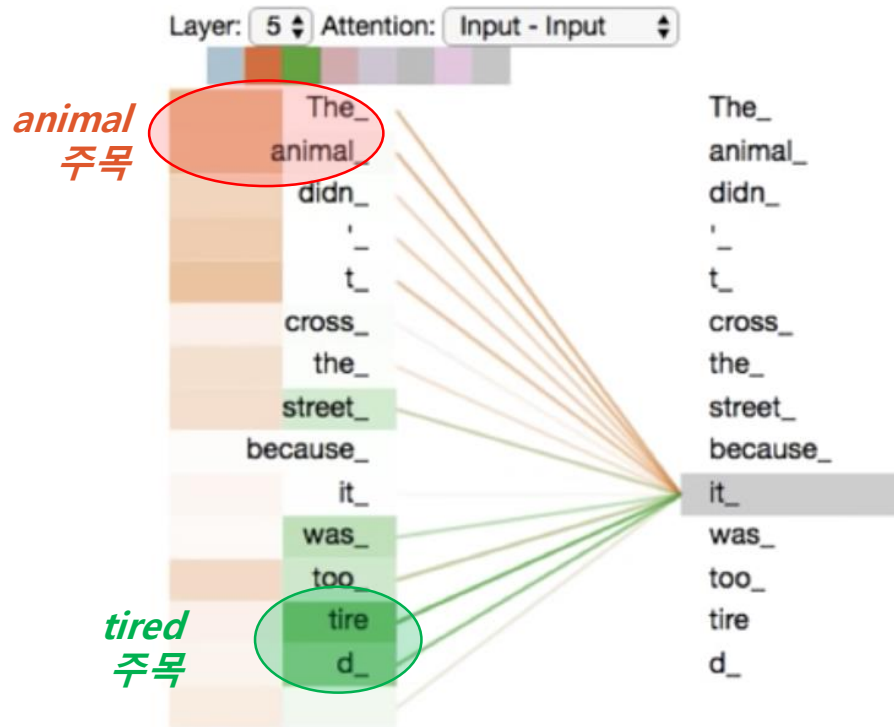
	Query	*	Key ^T	=	Score	Softmax	Value	Softmax * Value	(Attention layer output) \sum Softmax * Value
I * I		*		=	130	0.92	I		
I * study		*		=	50	0.05	study		
I * at		*		=	20	0.02	at		
I * school		*		=	10	0.01	school		
study * I		*		=	30	0.02	I		
study * study		*		=	110	0.70	study		
study * at		*		=	20	0.03	at		
study * school		*		=	70	0.25	school		
at * I		*		=	30	0.03	I		
at * study		*		=	50	0.10	study		
at * at		*		=	90	0.80	at		
at * school		*		=	40	0.07	school		
school * I		*		=	30	0.01	I		
school * study		*		=	80	0.27	study		
school * at		*		=	23	0.02	at		
school * school		*		=	160	0.70	school		

01 Multi-head Attention



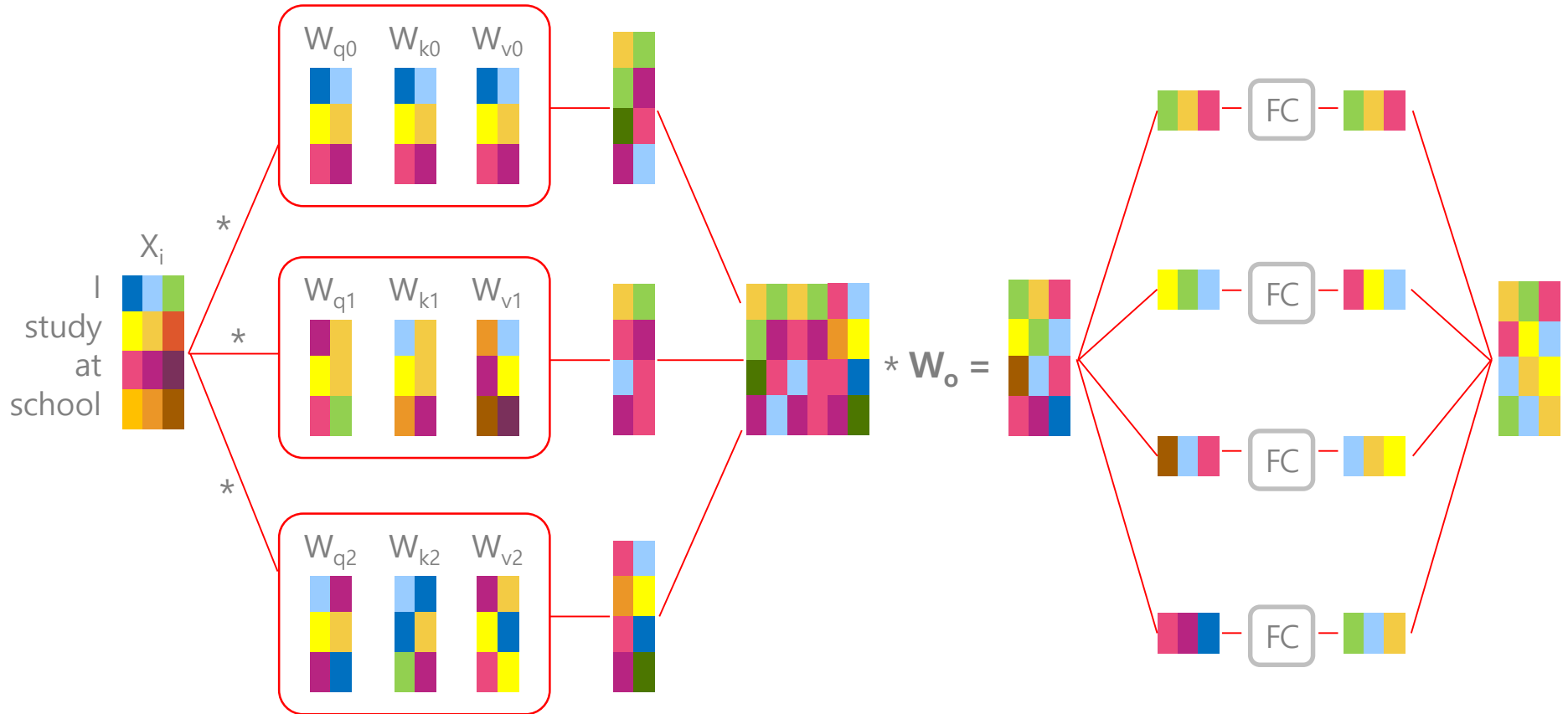
01 Multi-head Attention

asdf



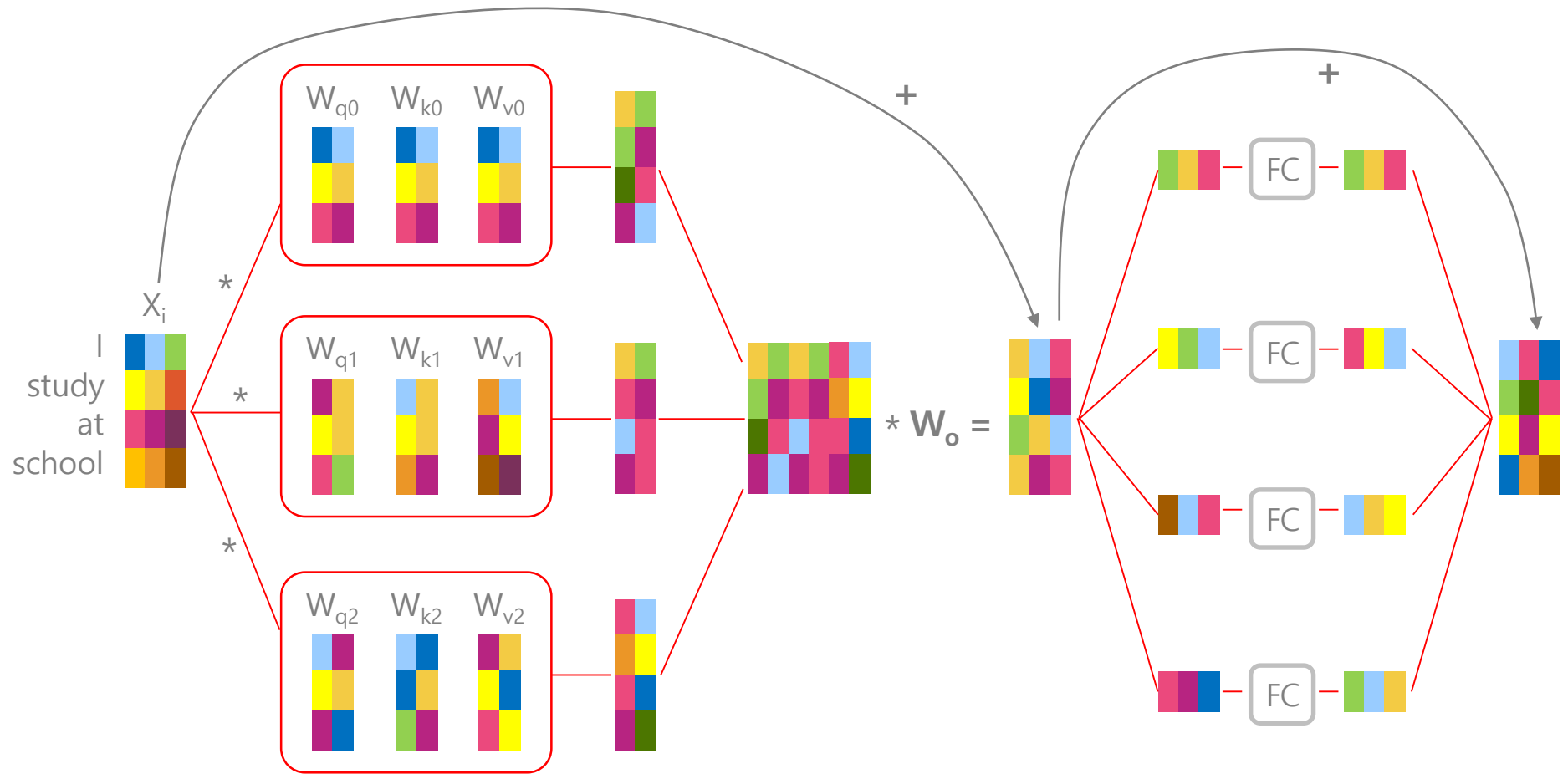
- 단어 “it”를 인코딩할 때 **첫번째 어텐션 헤드(갈색)**는 “the animal”에 주목하고 있고, **두번째 어텐션 헤드(녹색)**는 “tired”에 주목하고 있음.
- 멀티 헤드 어텐션 모델을 통해 단어 “it”가 “animal”과 “tired”에 동시에 주목한다는 사실을 복합적으로 표현

01 Multi-head Attention

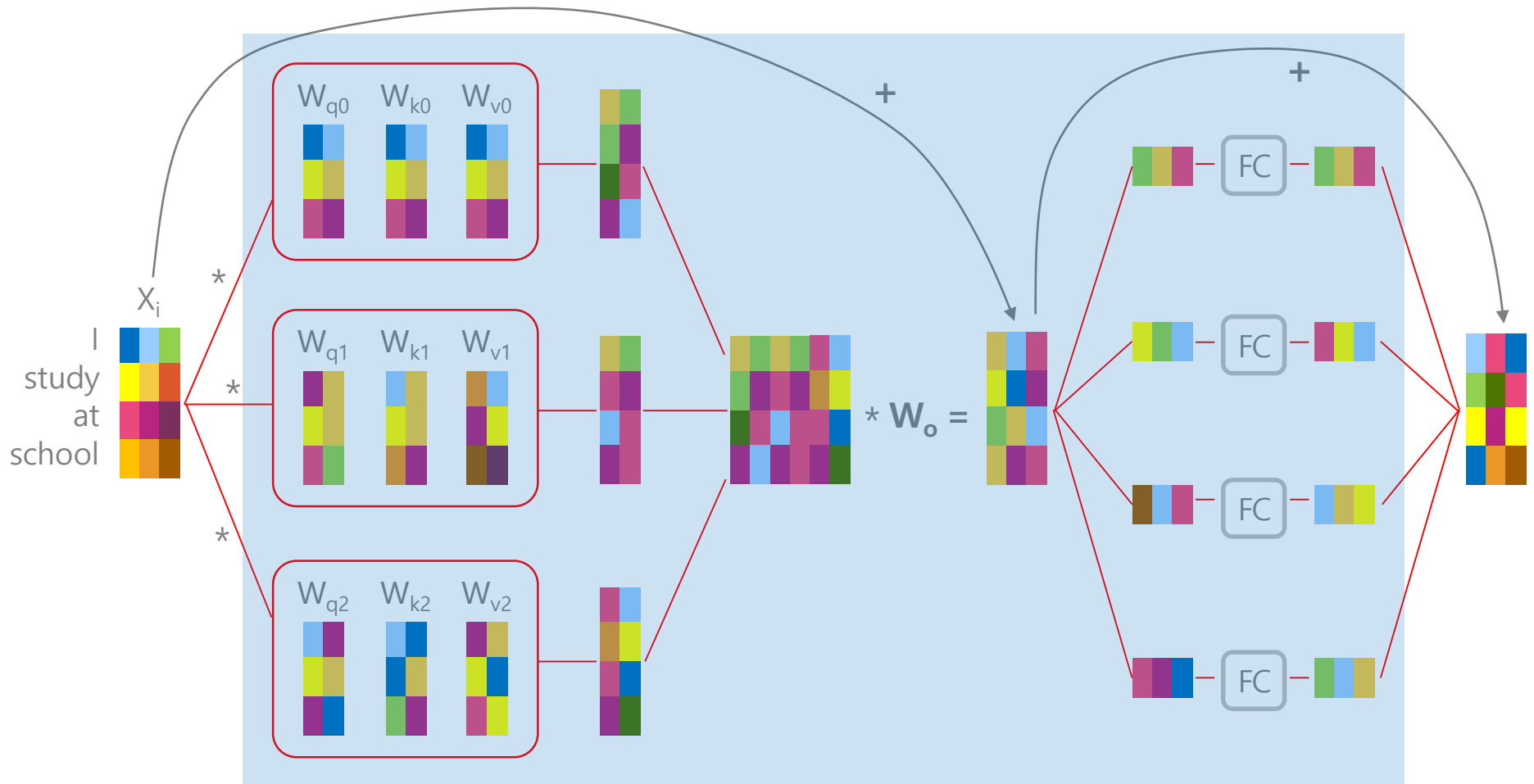


- 입력 행렬의 크기 = 출력 행렬의 크기

01 Residual Connection

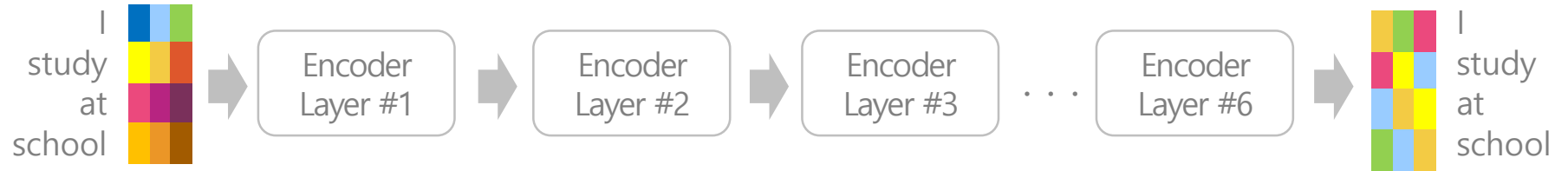


01 Encoder Layer



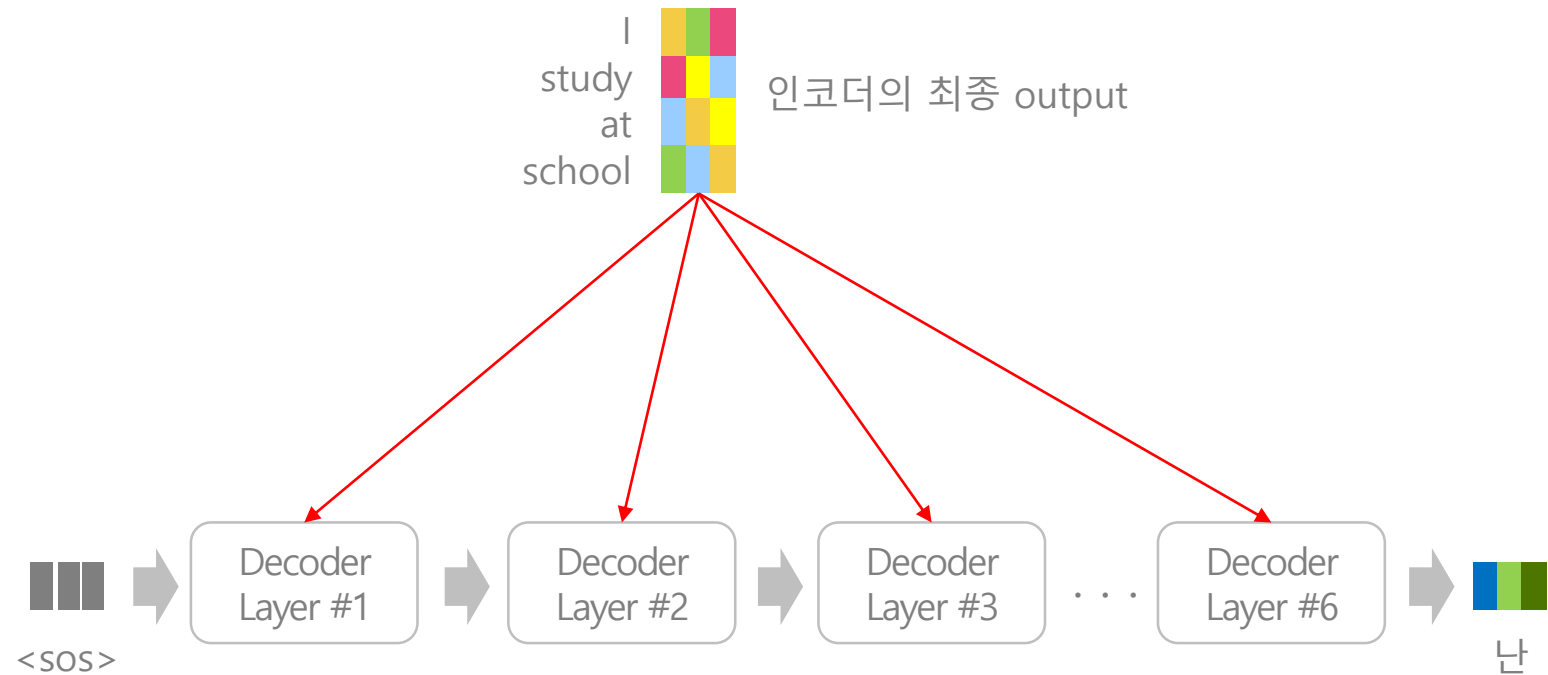
- 여러개의 인코더를 직렬로 붙여서 사용

01 Layered Encoders



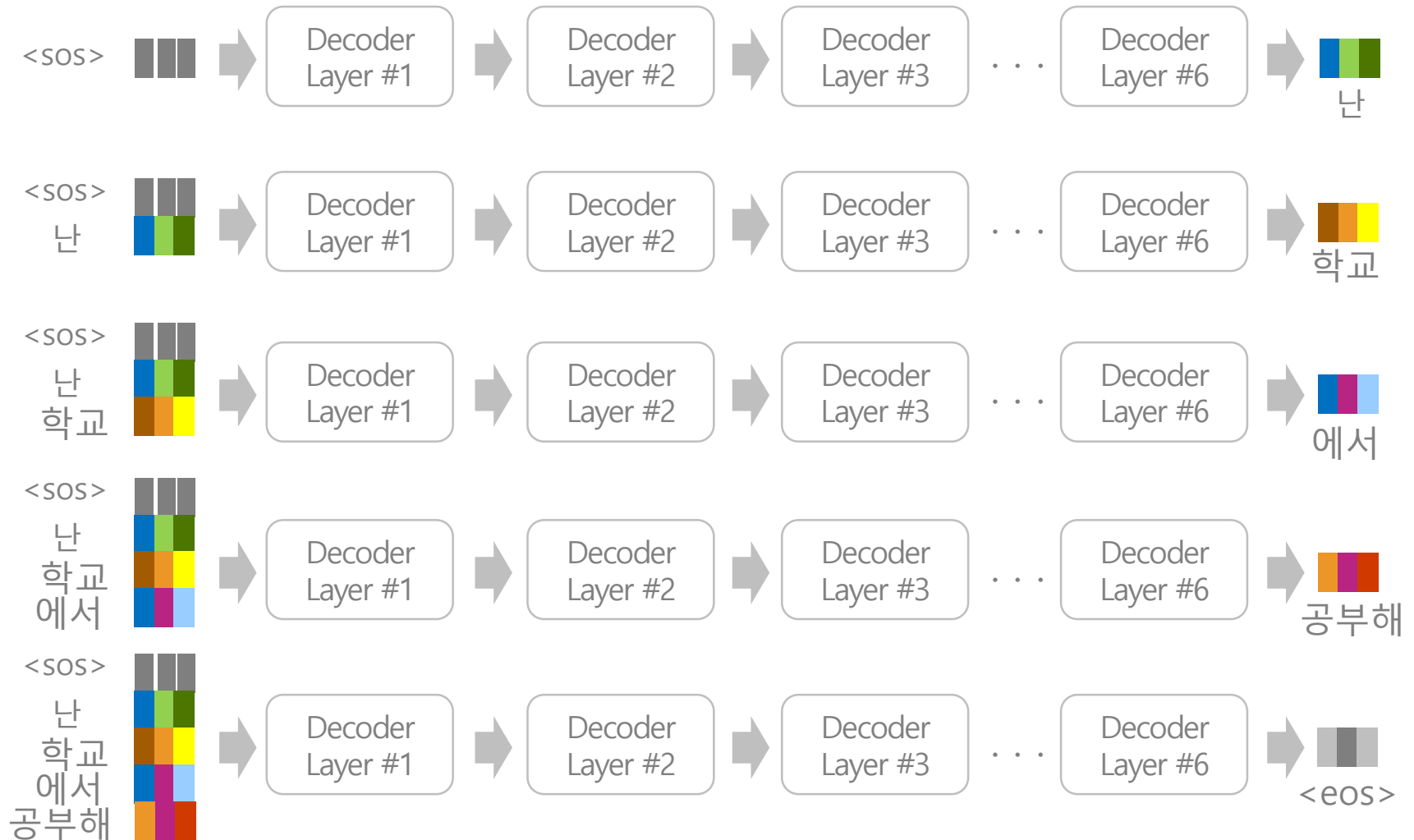
- 동일한 인코더 계층을 여러겹 쌓아서 사용함
- 동일한 구조이나 Weight 들은 각자 업데이트

01 Layered Decoders

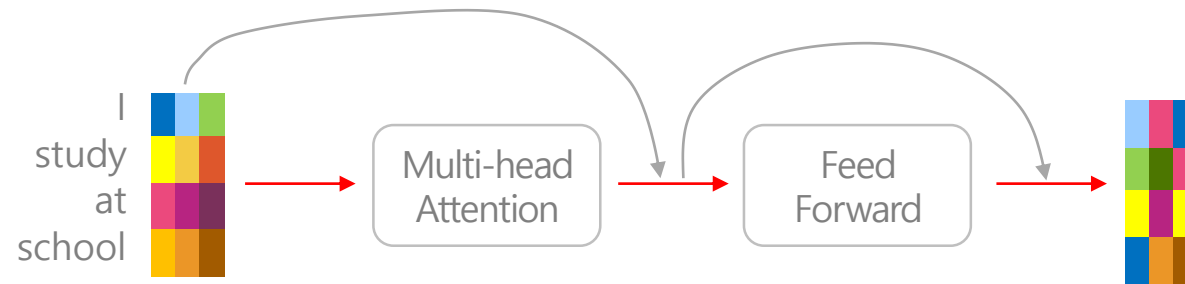


- 디코더는 한 스텝 당 하나씩 번역된 단어를 출력
(인코더는 한 문장 전체를 한꺼번에 입력)

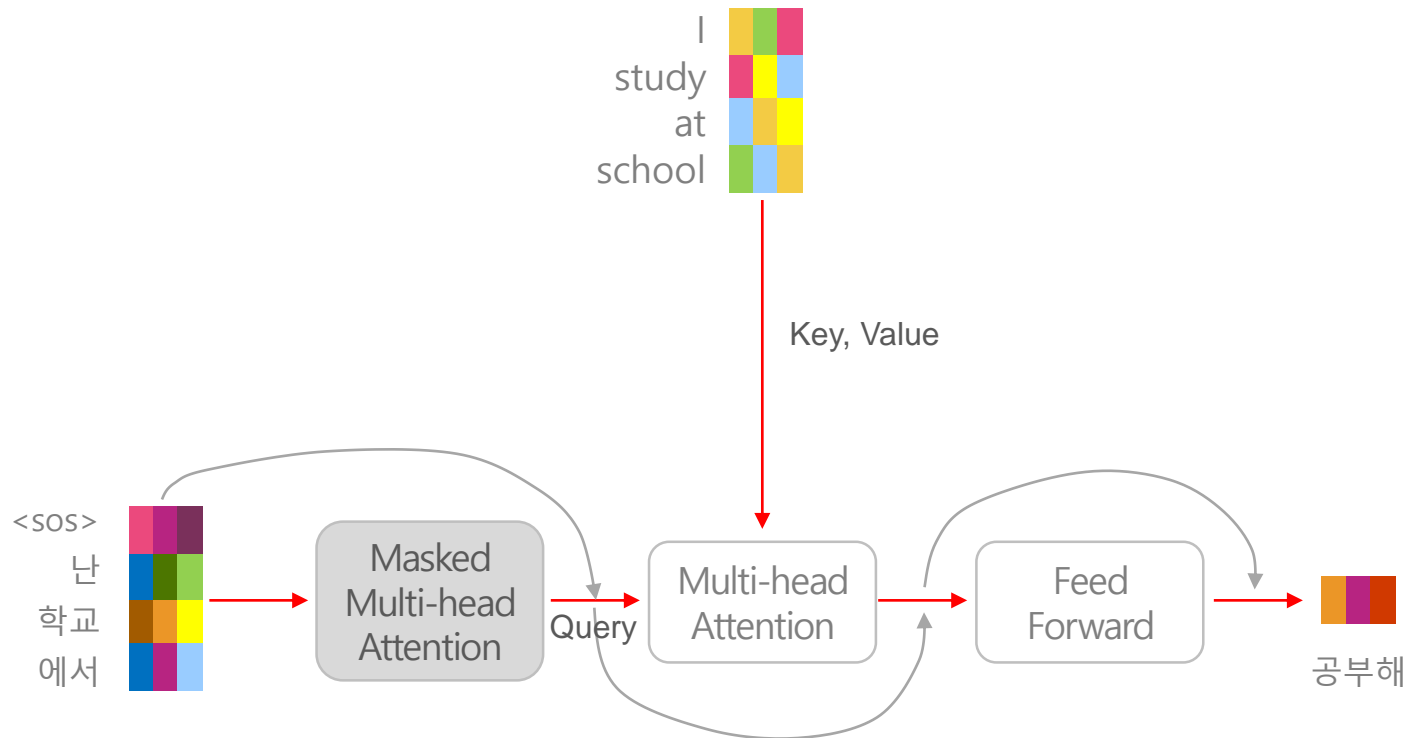
01 Layered Decoders



01 Translator Encoder

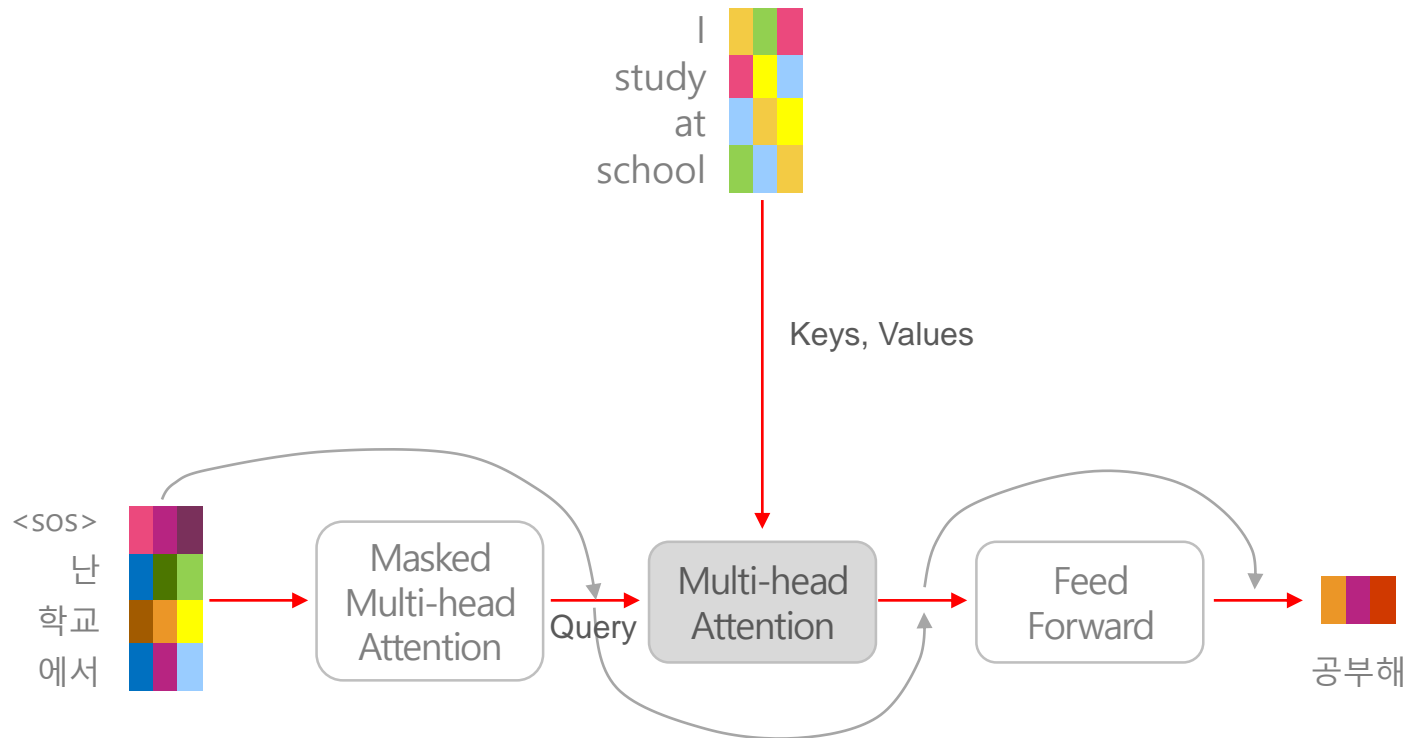


01 Translator Decoder



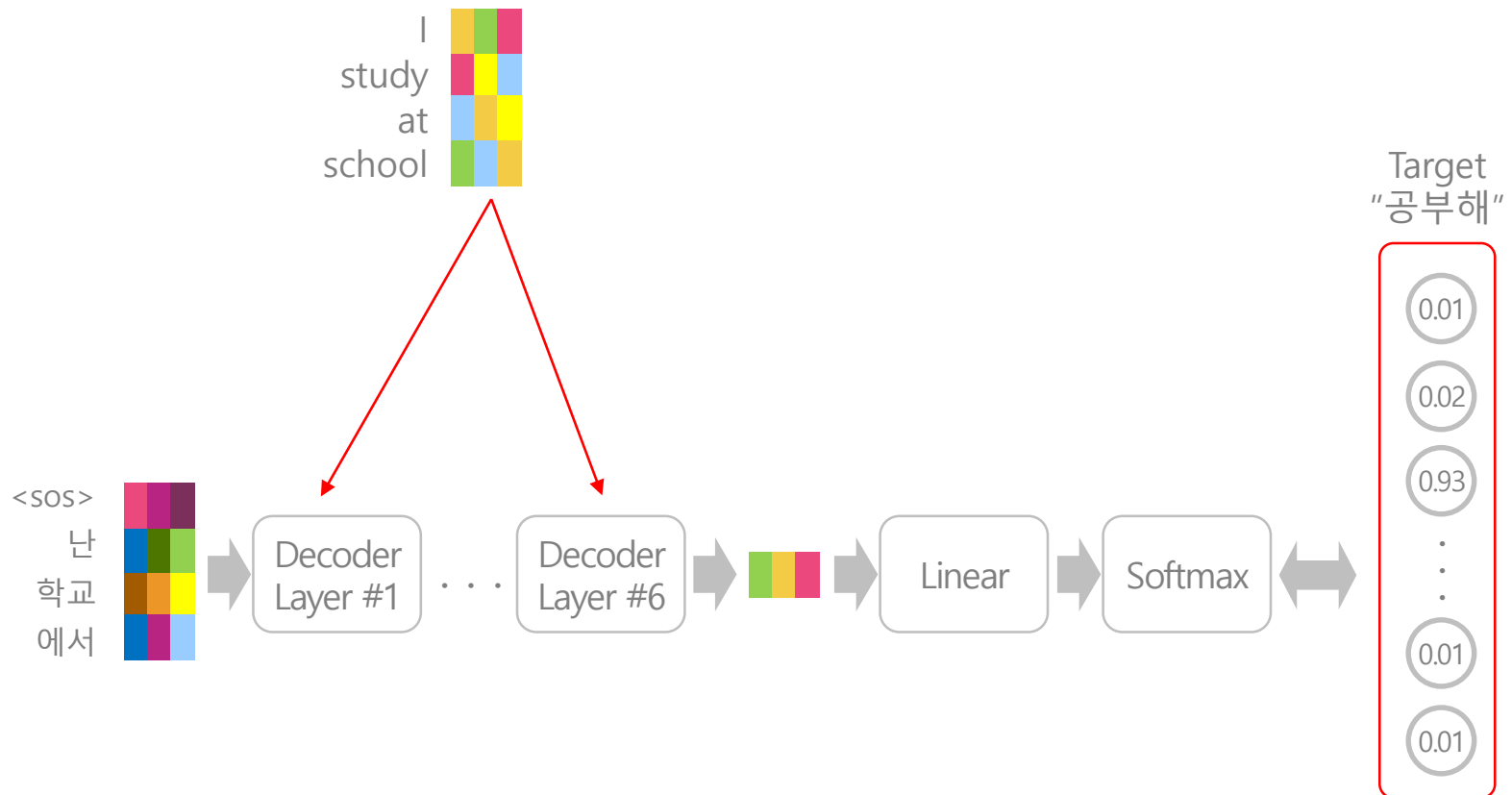
- 첫번째 노드는 Mask된 Multi-head Attention임
(지금까지 출력된 단어들만 Attention 적용하고 뒤에올 단어들은 Masking)

01 Translator Decoder

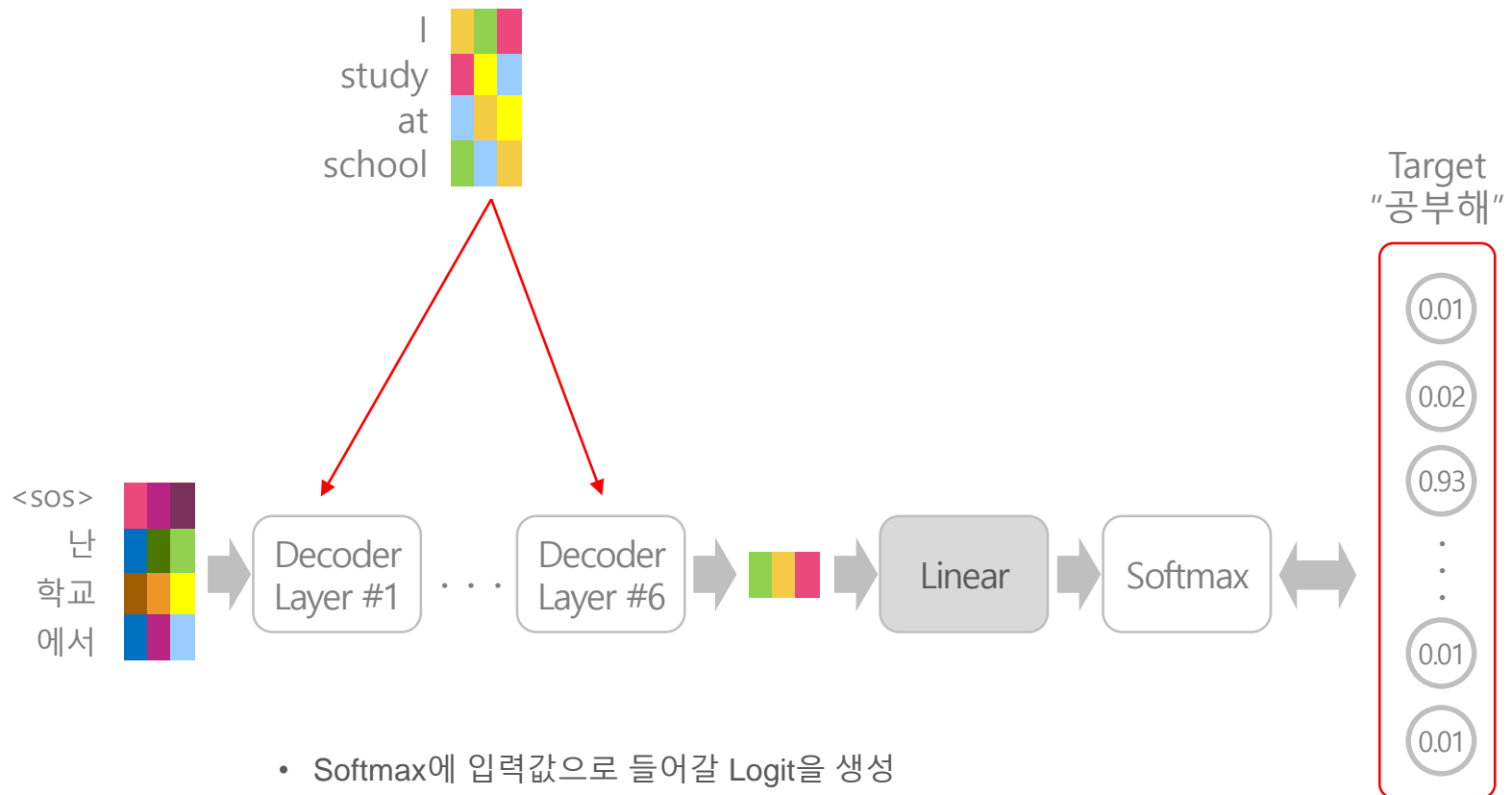


- Query값은 이전 디코더 계층으로부터 전달됨
- Keys 들과 Values 들은 인코더의 출력으로부터 전달됨
(디코더의 현재 단어가 Query이며 이를 인코더에 질의하여 Key와 Value를 획득하여 디코더가 출력할 다음 단어와 가장 잘 매칭되는 후보를 결정)

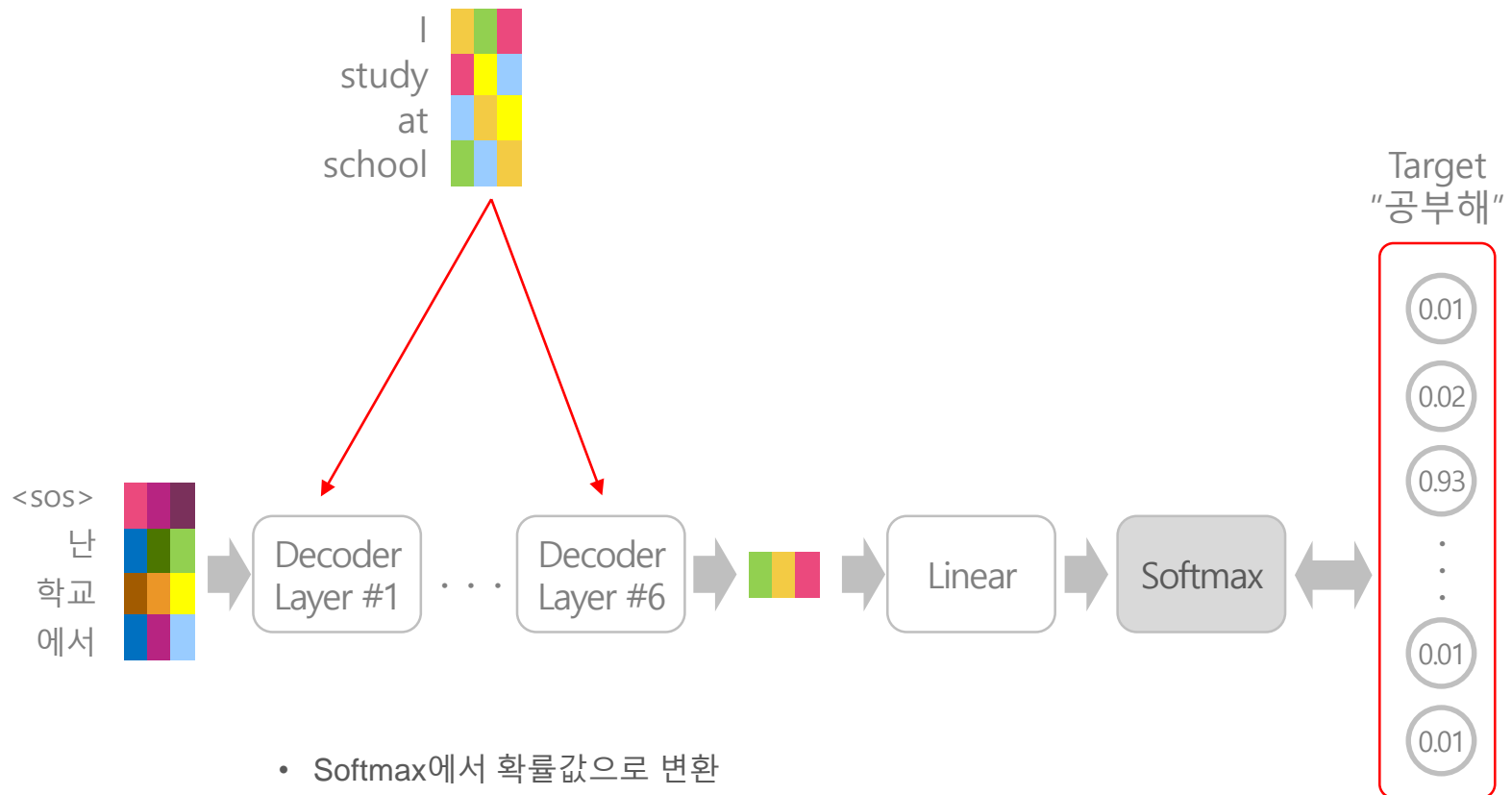
01 Final Output Vector



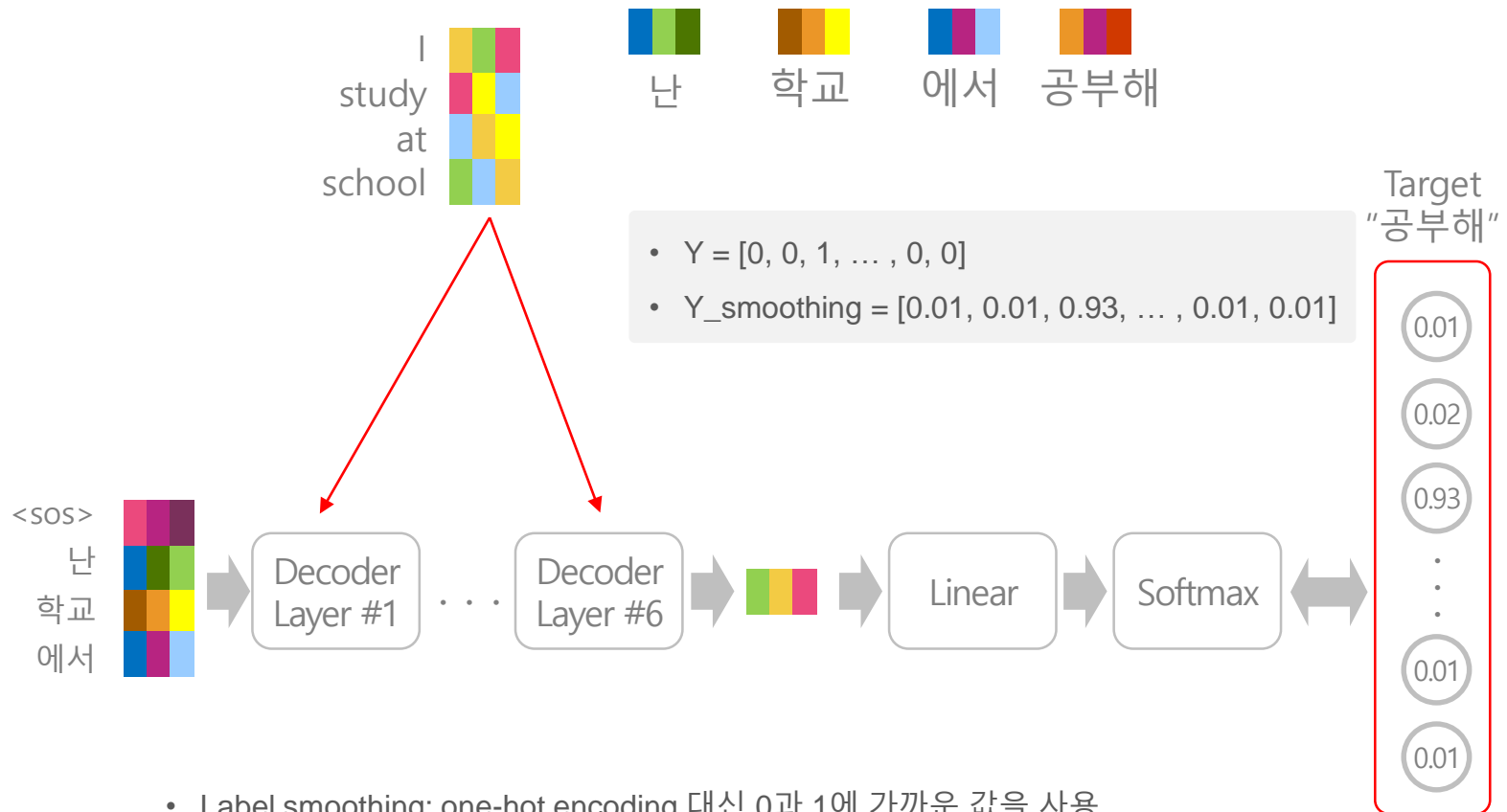
01 Final Output Vector



01 Final Output Vector



01 Final Output Vector



- Label smoothing: one-hot encoding 대신 0과 1에 가까운 값을 사용
- Label이 noisy하여 0/1로 깔끔하게 표현되지 않는 경우 학습 효과 향상 (BLEU 점수 및 예측 정확도 향상)

02 Positional Encoding

개념 예제

For example, for word w at position $pos \in [0, L-1]$

in the input sequence $\mathbf{w}=(w_0, \dots, w_{L-1})$, with 4-dimensional embedding e_w , and $d_{model}=4$, the operation would be

$$\begin{aligned} e'_w &= e_w + [\sin(pos/100000), \cos(pos/100000), \sin(pos/10000^{2/4}), \cos(pos/10000^{2/4})] \\ &= e_w + [\sin(pos), \cos(pos), \sin(pos/100), \cos(pos/100)] \end{aligned}$$

where the formula for positional encoding is as follows

$$\begin{aligned} PE(pos, 2i) &= \sin(pos/10000^{2i/d_{model}}) & - \text{단 } i \in [0, d_{model}/2] \\ PE(pos, 2i+1) &= \cos(pos/10000^{2i/d_{model}}) \end{aligned}$$

with $d_{model}=512$ (thus $i \in [0, 255]$) in the original paper.

02 Positional Encoding

구현 예제

예제 문장 $e_w := \mathbf{I \ am \ a \ student}$, $L=4$

- i 는 차원을 browse 하는 첨자, 단 $i \in [0, d_{\text{model}} / 2]$, 본 예제에서는 $4/2 = 2$
- $\text{PE} = [\sin(\text{pos}), \cos(\text{pos}), \sin(\text{pos}/100), \cos(\text{pos}/100)]$

(1) 'I'에 대한 PE

- 단어 위치 $\text{pos} = 0$
- $\text{PE} = [\sin(\text{pos}), \cos(\text{pos}), \sin(\text{pos}/100), \cos(\text{pos}/100)]$
 $= [\sin(0), \cos(0), \sin(0/100), \cos(0/100)]$
 $= [0, 1, 0, 1]$

(2) 'am'에 대한 PE

- 단어 위치 $\text{pos} = 1$
- $\text{PE} = [\sin(\text{pos}), \cos(\text{pos}), \sin(\text{pos}/100), \cos(\text{pos}/100)]$
 $= [\sin(1), \cos(1), \sin(1/100), \cos(1/100)]$
 $= [0.8414, 0.5403, 0.0099, 0.9999]$