Transformer(attention is all you need)

유범곤, beomgon.yu@gmail.com

참조

https://wikidocs.net/24996

https://amitness.com/2020/03/illustrated-transformer/

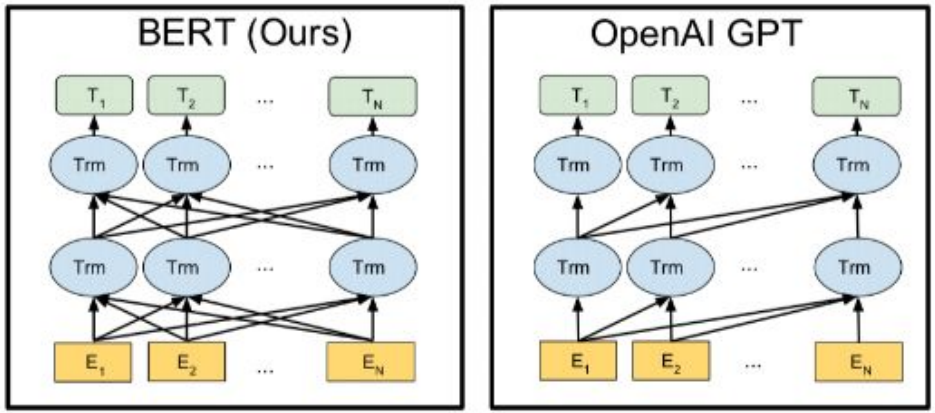
https://www.youtube.com/watch?v=xhY7m8QVKjo

http://nlp.seas.harvard.edu/2018/04/03/attention.html

Contents

Why Transformer

현재 NLP의 대세인 BERT 및 OPEN-GPT계열 모두 Transformer에서 나왔다.
Transformer에 대해 알아볼 필요 존재함.



https://huggingface.co/transformers/model_doc/bert.html

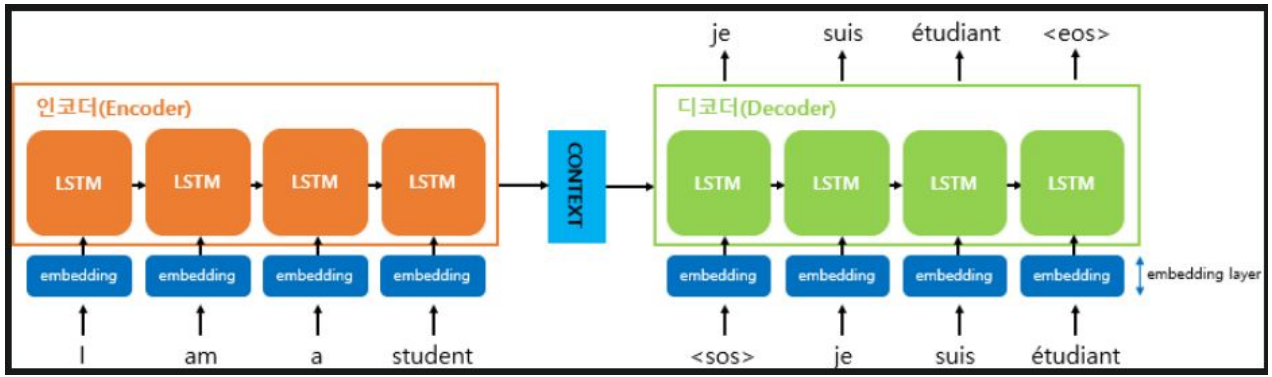
- BERT
- OpenAI GPT
- Transformer XL
- OpenAI GPT2
- XLM
- XLNet
- RoBERTa
- DistilBERT
- CTRL
- CamemBERT
- ALBERT
- XLM-RoBERTa
- FlauBERT
- Bart
- T5
- ELECTRA
- DialoGPT
- Reformer
- MarianMT
- Longformer

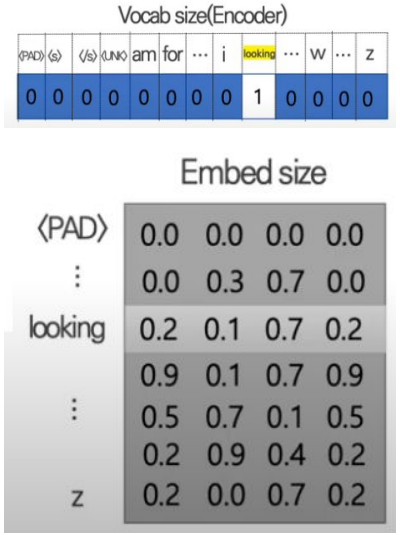
기존 seq2seq 모델에 대한 소개

encoder/decoder 구조

for getting sequencial context, RNN 계열 신경망이 주로 사용되었음(LSTM, GRU)

But exploding or vanishing gradient problem, difficult parallel training, long inference/training time
문장이 길수록 처음 token의 정보가 전달되기 힘들다.





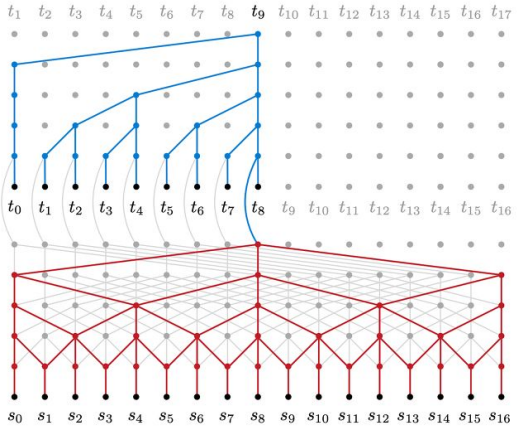
기존 seq2seq 모델에 대한 소개

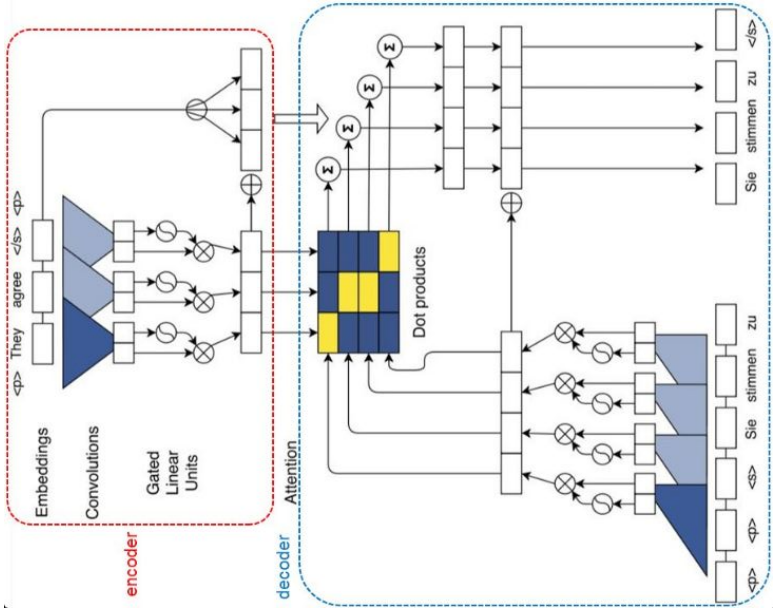
instead of RNN, Use CNN

ByteNet - use dilated convolution

convS2S(Facebook)

Gated linear units, residual Connection, attention



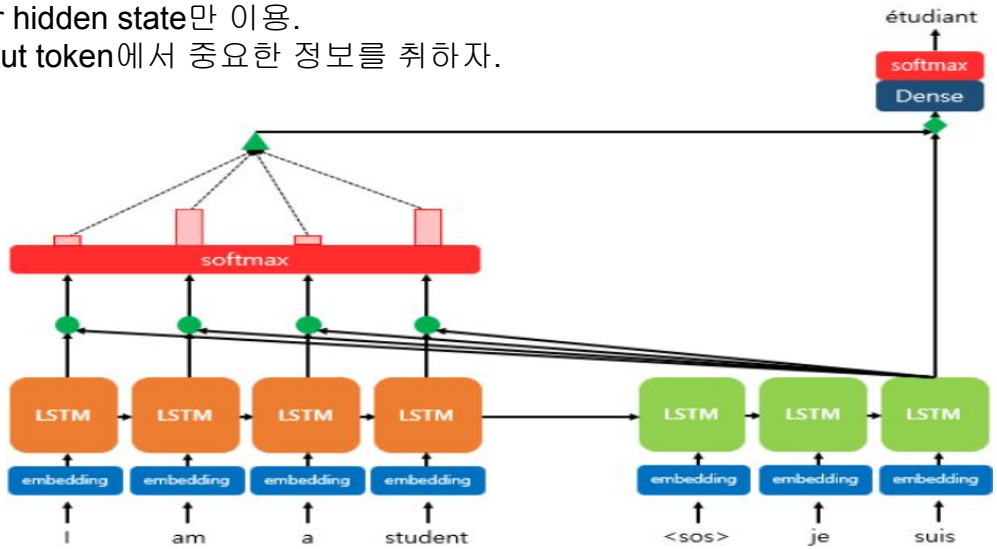


ConvS2S : https://reniew.github.io/44/

기존 seq2seq 모델에 대한 소개

RNN based seq2seq using attention

기존 RNN은 context vector or hidden state만 이용.
attention을 이용하여 전체 input token에서 중요한 정보를 취하자.



https://wikidocs.net/31379

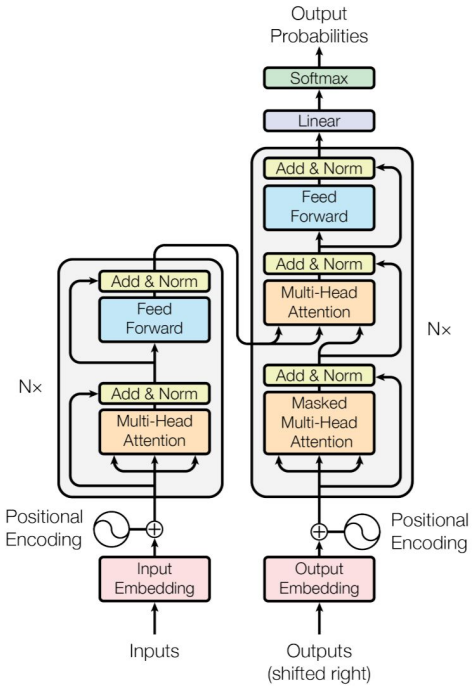
Transformer 설명

for getting sequential context, RNN or CNN based or attention plus model들이 소개됨.

But only attention만으로 Encoder/Decoder구현하면 어떨까??
Transformer가 최초이다

Transformer/ Model

- 모델의 목적
  - Downstream Task for NLP
  - Translation(독일어 -> 영어)
- 데이터의 구성
  - Pair sentences(독일어, 영어)
- Attention 방법
  - Key, Query, Value
  - K,Q,V 는 Encoder, Decoder에서 각각 생성됨



Encoder

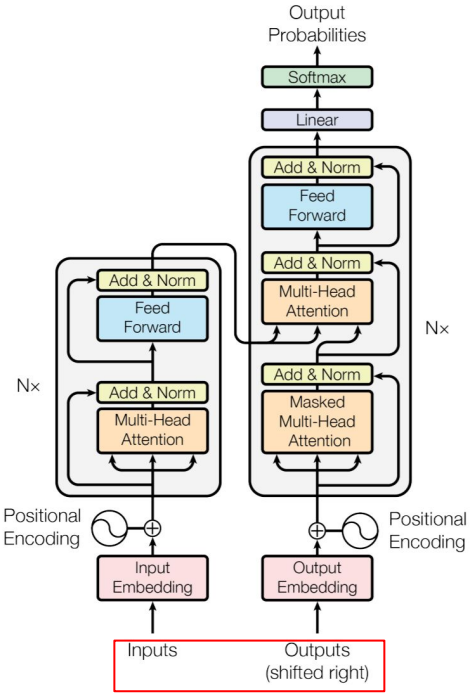
token embedding

Encoder, max_length = 6일 때,

- Sent1: i am looking for happiness $\{x_1, x_2, x_3, x_4, x_5, \text{</s>}\}$
- Sent2: i am donghwa $\{x_1, x_2, x_3, \text{</s>}, \text{<PAD>}, \text{<PAD>}\}$
- Sent3: i am @#$ $\{x_1, x_2, \text{<UNK>}, \text{</s>}, \text{<PAD>}, \text{<PAD>}\}$

Decoder, max_length = 6일 때,

- Sent1: 나는 행복을 찾고 있다 $\{\text{<s>}, x_1, x_2, x_3, x_4, \text{</s>}\}$
- Sent2: 나는 동화다 $\{\text{<s>}, x_1, x_2, \text{</s>}, \text{<PAD>}, \text{<PAD>}\}$
- Sent3: 나는 @#$ $\{\text{<s>}, x_1, \text{<UNK>}, \text{</s>}, \text{<PAD>}, \text{<PAD>}\}$



Encoder

Positional encoding - sequential한 offset을 각 token에 주기 위함

determine the position of each word, or the distance between different words in the sequence

The intuition here is that adding these values to the embeddings provides meaningful distances between the embedding vectors once they're projected into Q/K/V vectors and during dot-product attention.

$$PE_{(pos,2i)} = \sin(pos/10^{4\cdot 2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10^{4\cdot 2i/d_{model}})$$

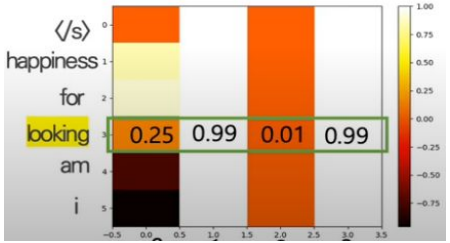
d_{model} : embedding size

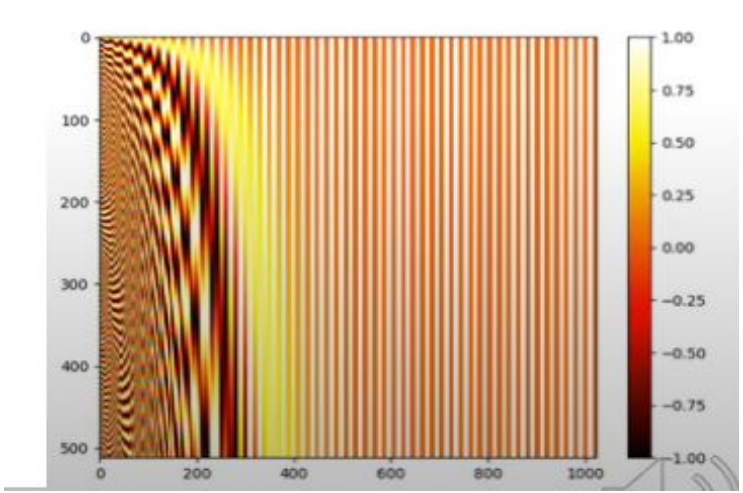
i: embedding feature의 인덱스

Pos: 단어의 인덱스(a word)

Encoder, max_length = 6일때,

- Sent1: i am looking for happiness





max_legnth = 512

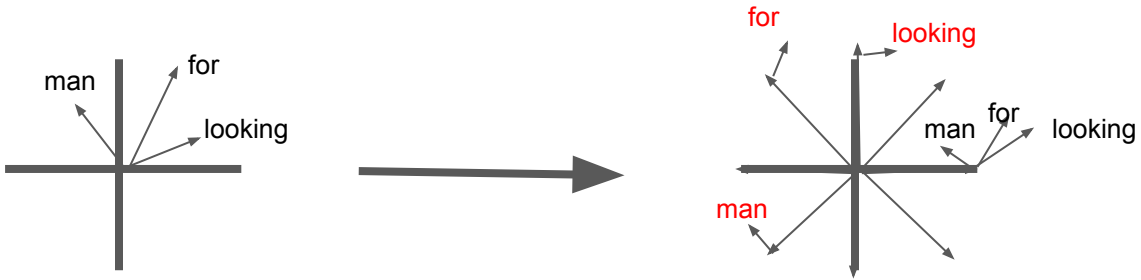
embed size = 1024

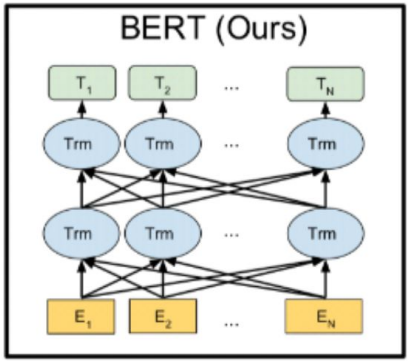
Encoder

Positional encoding

“ I am looking for a man”

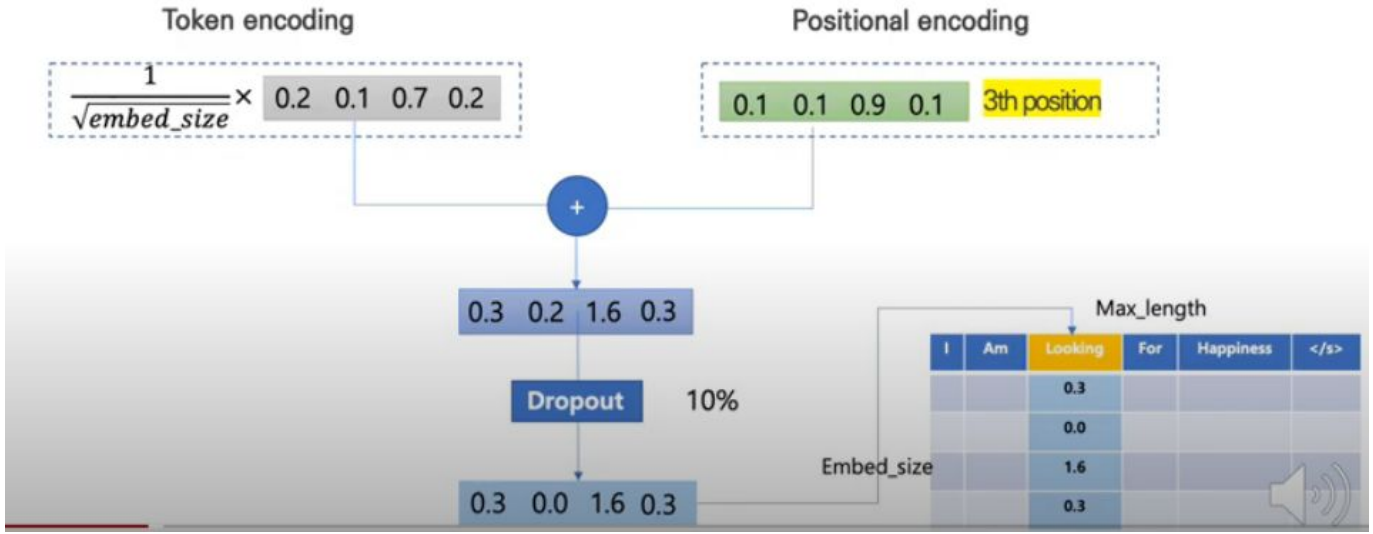
there is **no notion of word order** (1st word, 2nd word, ..) in the proposed architecture. All words of input sequence are fed to the network with no special order or position (unlike common RNN or ConvNet architectures), thus, model has no idea how the words are ordered. Consequently, a position-dependent signal is added to each word-embedding to help the model incorporate the order of words.
In the case of RNNs, we feed the words sequentially to RNN, i.e. n-th word is fed at step n, which helps the model incorporate the order of words.





Encoder

Embedding/ positional encoding/ dropout



Encoder

attention : scaled dot product attention

additive attention(fcn with 1 hidden layer)

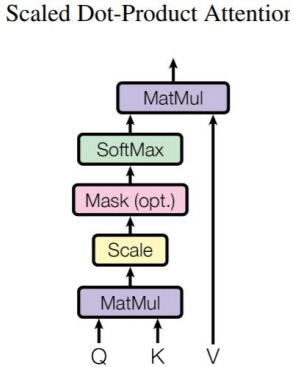
dot product attention

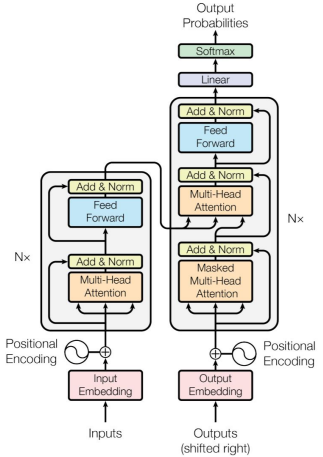
when dk is large, additive attention outperform dot product attention. assume that this is because of large number(softmax gradient extremly small)
-> so scaled by root(dk)

it seemed that tow(temperature) in contrastive loss

$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)/\tau)},$$

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$





Encoder

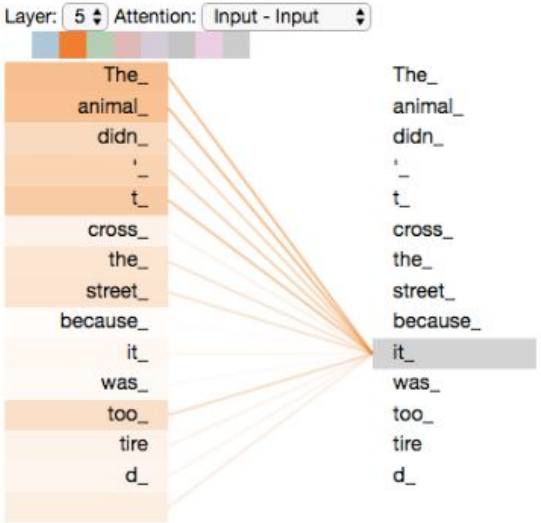
why self attention

- complexity
- Sequential Op
- Maximum path Length

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

”

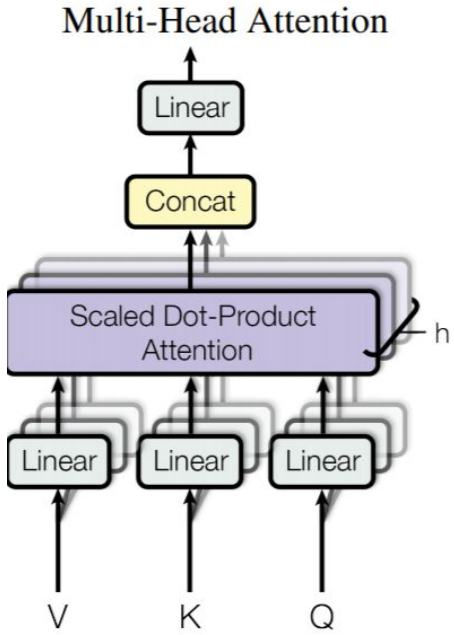
The animal didn't cross the street because it was too tired”



Encoder

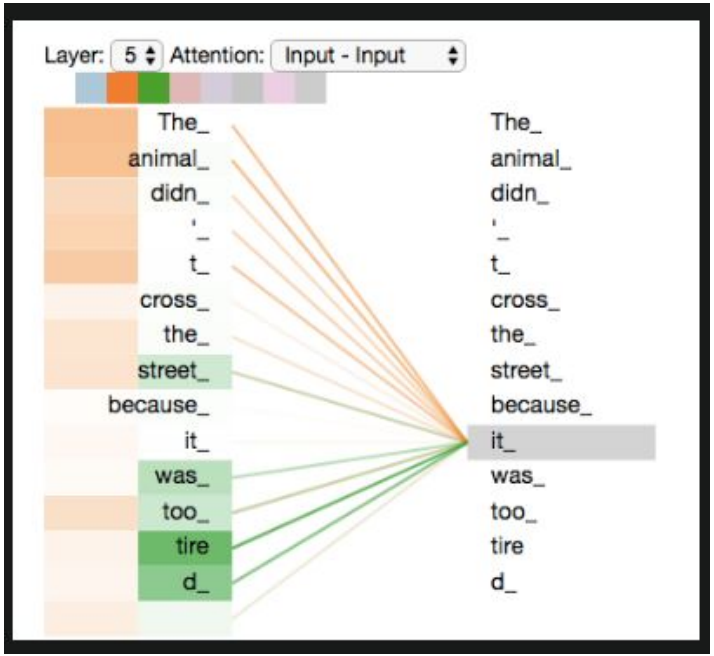
multi head attention - jointly attend to information from different representation subspace

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



Encoder

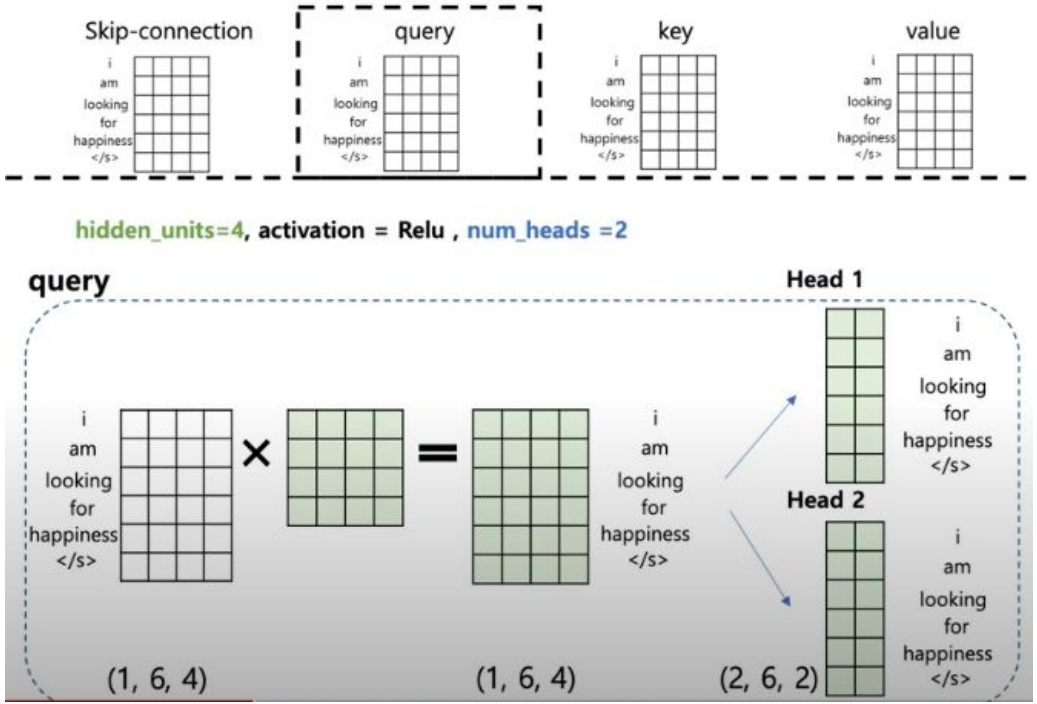
multi head attention



Encoder

multi-head attention

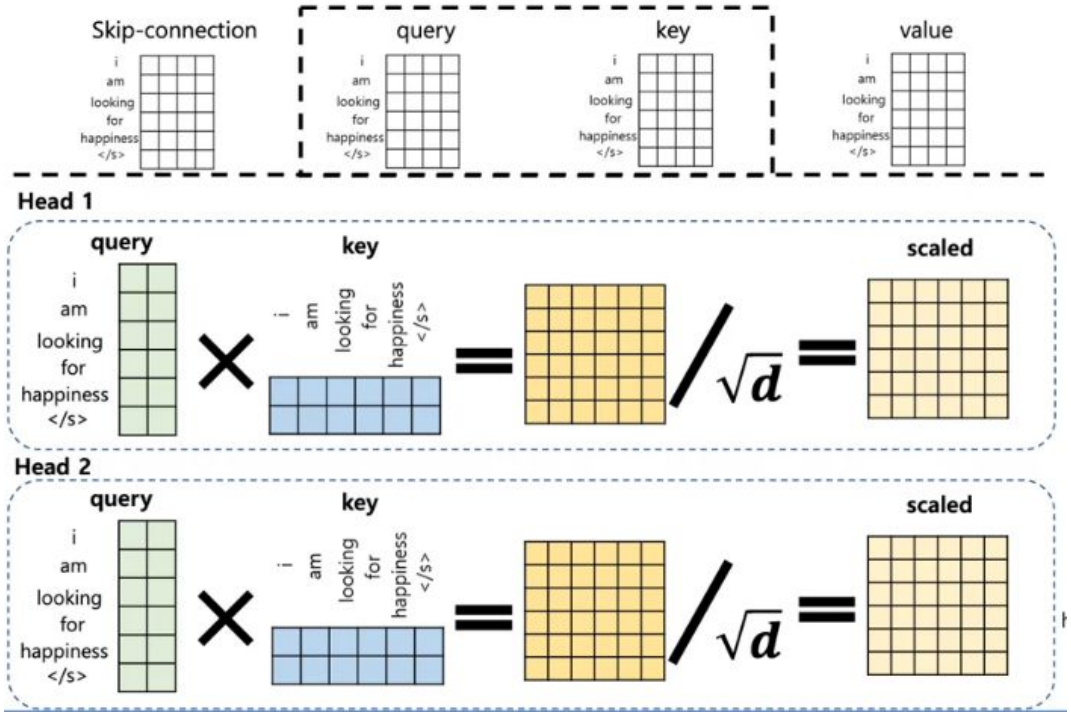
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



Encoder

multi-head attention

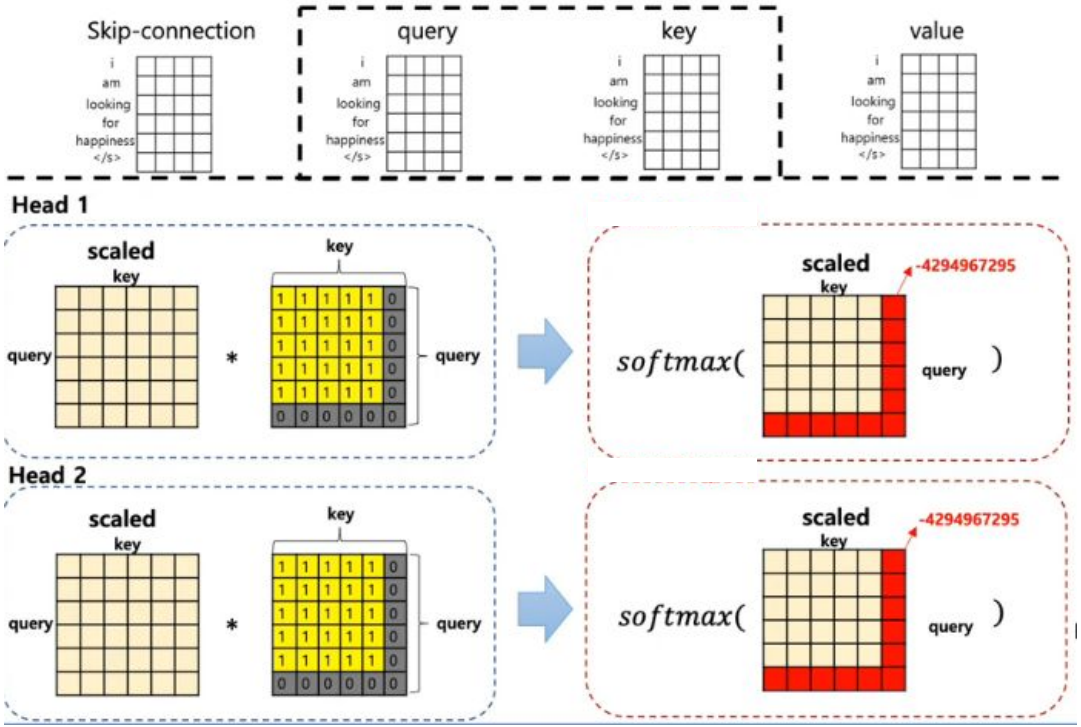
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



Encoder

multi-head attention

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



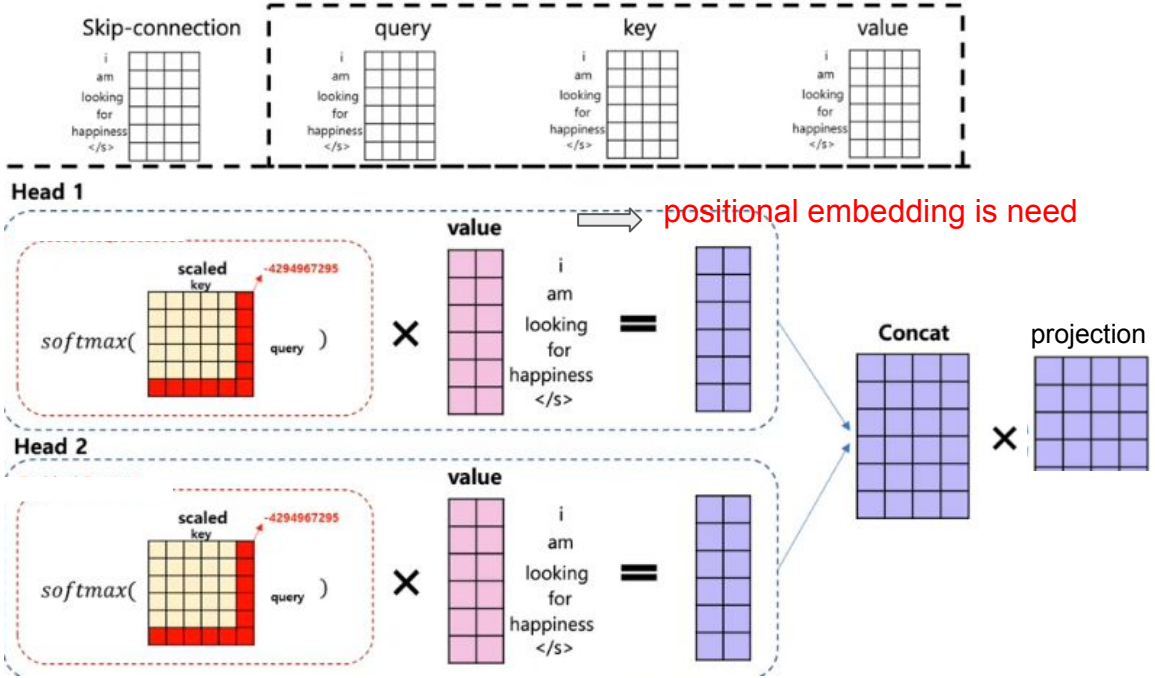
Encoder

multi-head attention

after concat, linear transformation is followed

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

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



Encoder

residual connection

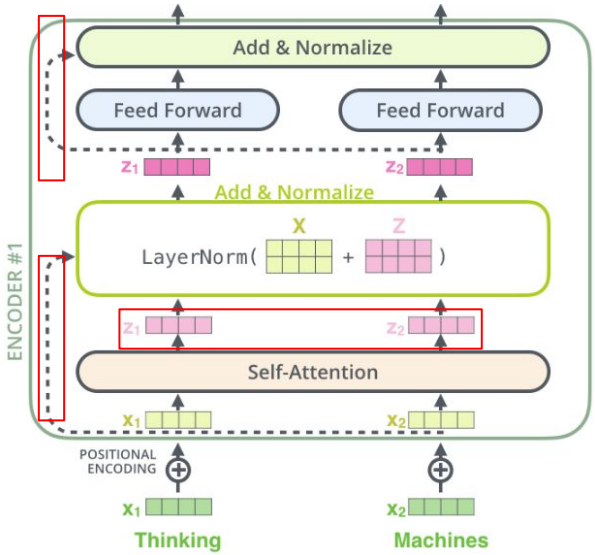
backpropagation is easy

block rapid change of token embedding

output = LayerNorm(x + sublayer(x))

residual dropout

apply dropout to the output of each sublayer, before it is added and normalized(rate : 0.1)

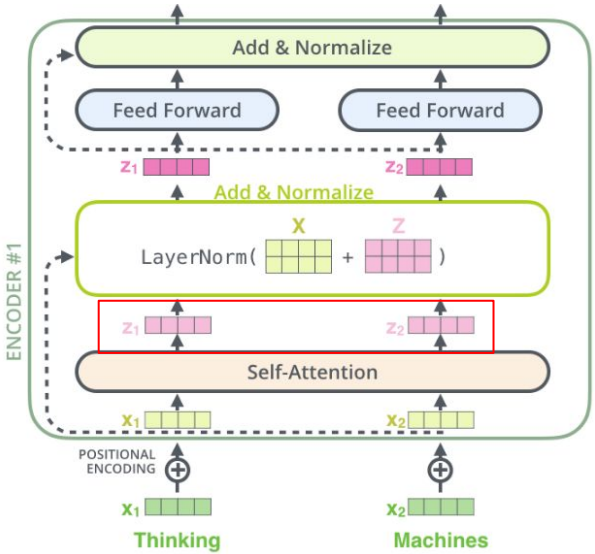


Encoder

residual dropout

```
class SublayerConnection(nn.Module):
    """
    A residual connection followed by a layer norm.
    Note for code simplicity the norm is first as opposed to last.
    """
    def __init__(self, size, dropout):
        super(SublayerConnection, self).__init__()
        self.norm = LayerNorm(size)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x, sublayer):
        "Apply residual connection to any sublayer with the same size."
        return x + self.dropout(sublayer(self.norm(x)))
```



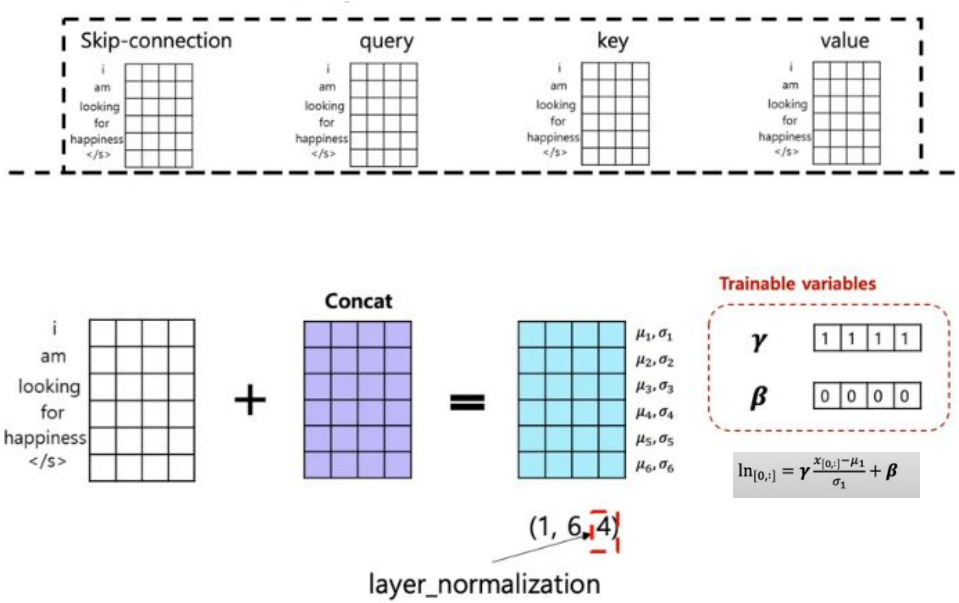
Encoder

Add & Norm

output = LayerNorm(x + sublayer(x))

```
class LayerNorm(nn.Module):
    "Construct a layernorm module (See citation for details)."
    def __init__(self, features, eps=1e-6):
        super(LayerNorm, self).__init__()
        self.a_2 = nn.Parameter(torch.ones(features))
        self.b_2 = nn.Parameter(torch.zeros(features))
        self.eps = eps

    def forward(self, x):
        mean = x.mean(-1, keepdim=True)
        std = x.std(-1, keepdim=True)
        return self.a_2 * (x - mean) / (std + self.eps) + self.b_2
```

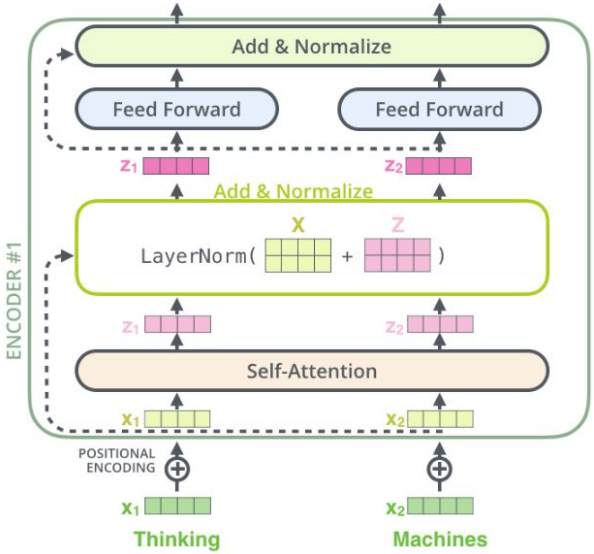


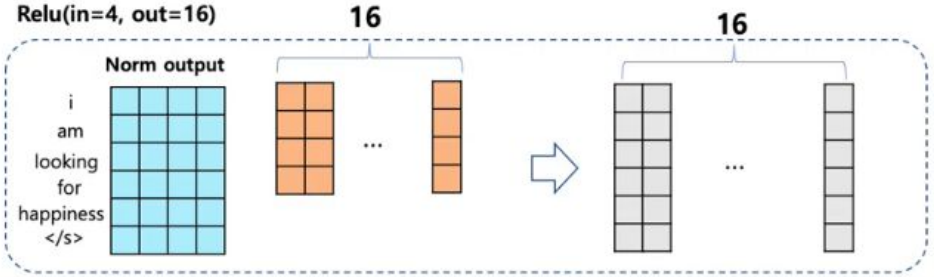
Encoder

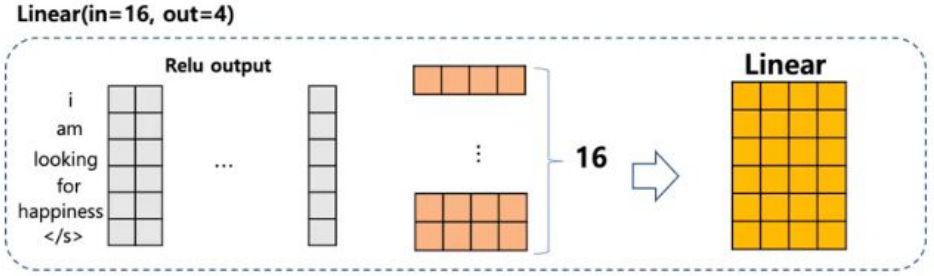
pointwise Feed Forward Network - linear transformation

relu -> grelu(in bert)

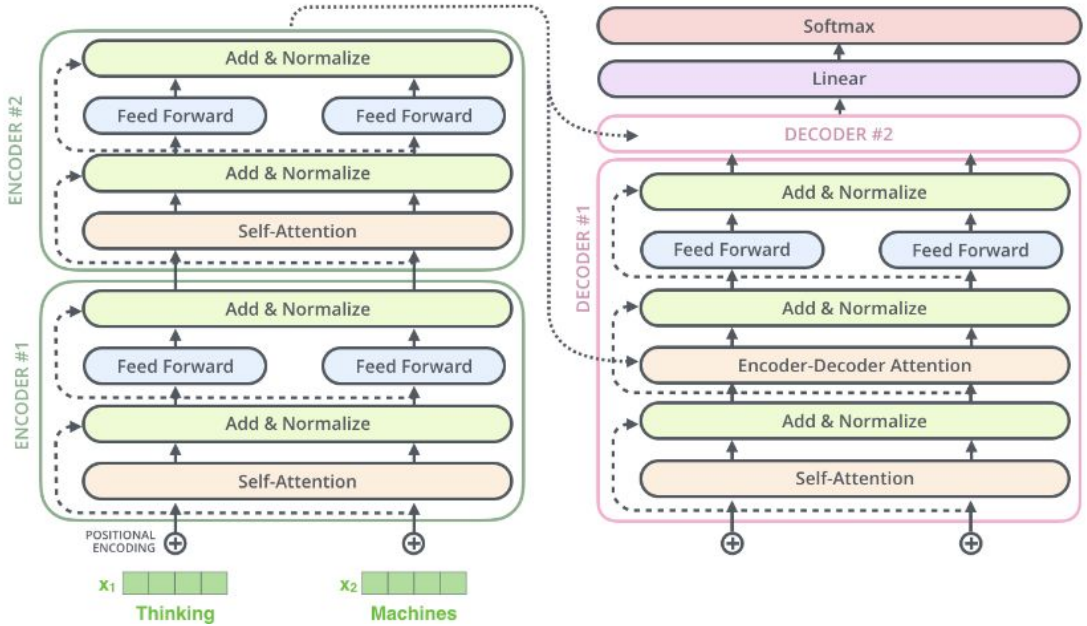
$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$







Encoder/Decoder

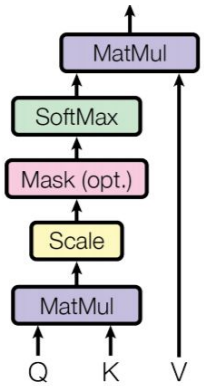


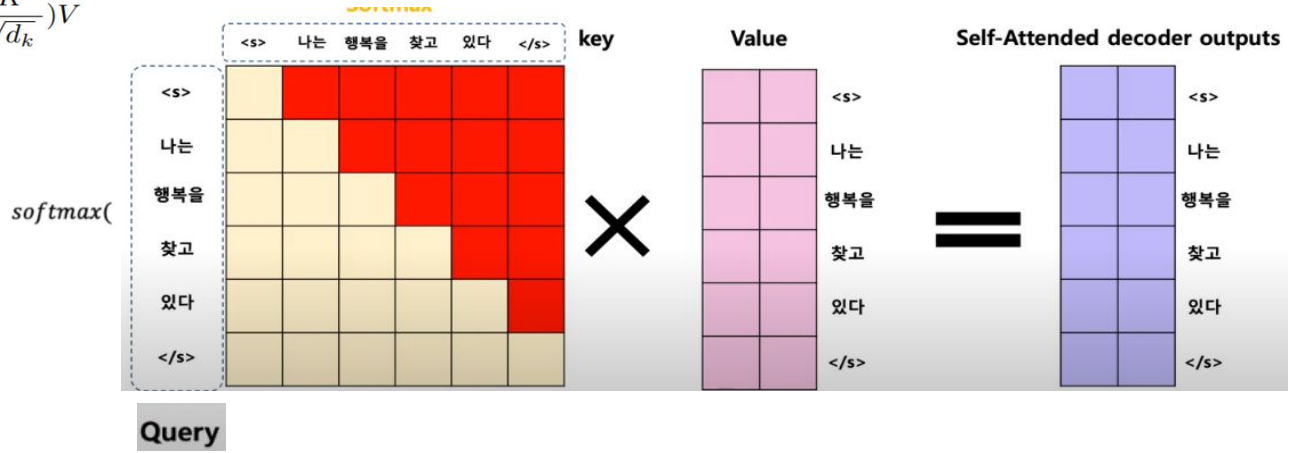
Decoder

- Token Embedding + postion Embedding, dropout
- BOS, EOS
- Masked Multi-Head Attention

  the self-attention layer is only allowed to attend to earlier positions in the output sequence. This is done by masking future positions (setting them to `-inf`) before the softmax step

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

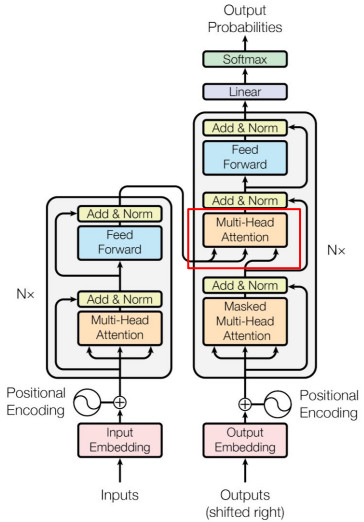


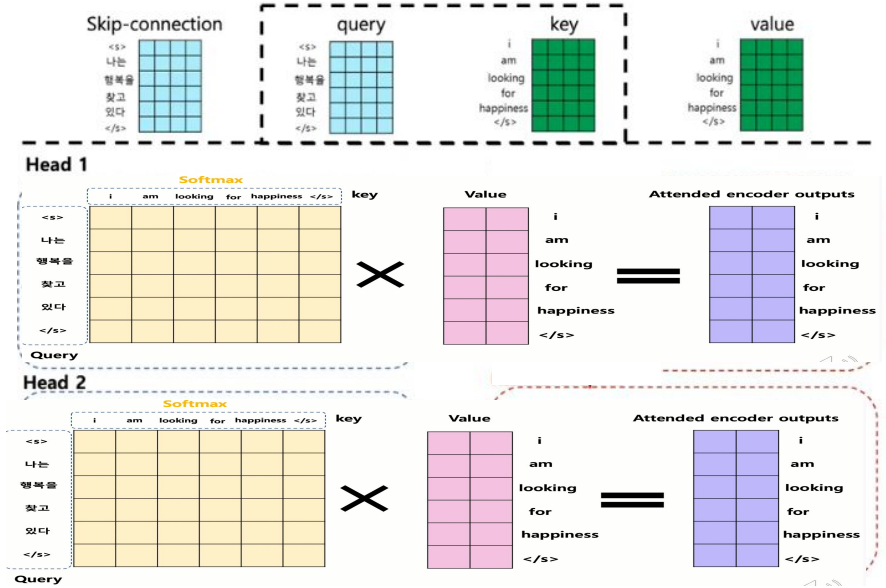


Decoder

Encoder/Decoder attention

it creates its Queries matrix from the layer below it, and takes the Keys and Values matrix from the output of the encoder stack

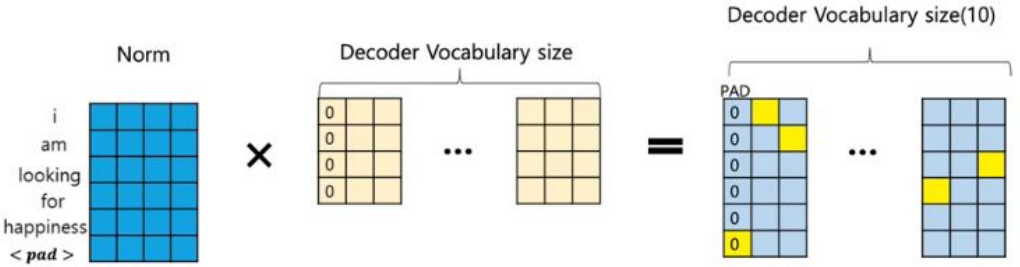




ex) 행복을 token의 encoder의 happiness에 좀더 attention을 준 상태로 새롭게 encoding 된다.

Decoder

Linear/Softmax



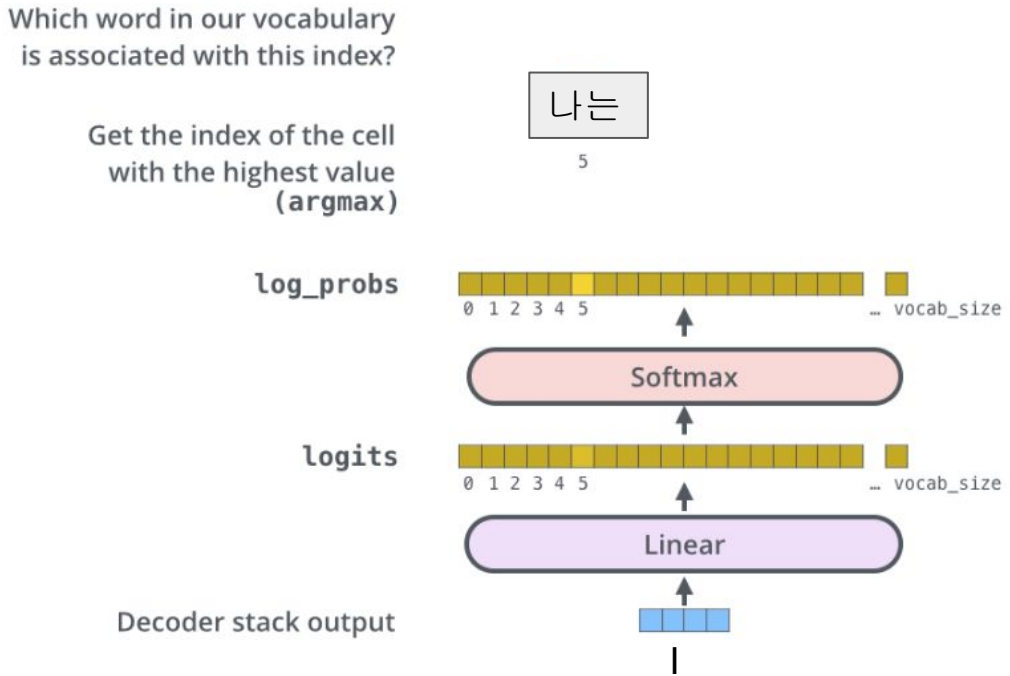
- Target
  - Label smoothing을 적용할 경우 perplexity는 높아지지만 BLEU score 성능 향상시킬 수 있음
    - [0, 0, 1]

$$((1-\epsilon)\times[0,0,1]) + \left(\frac{\epsilon}{len(V)}\right), \qquad \epsilon = 0.1$$

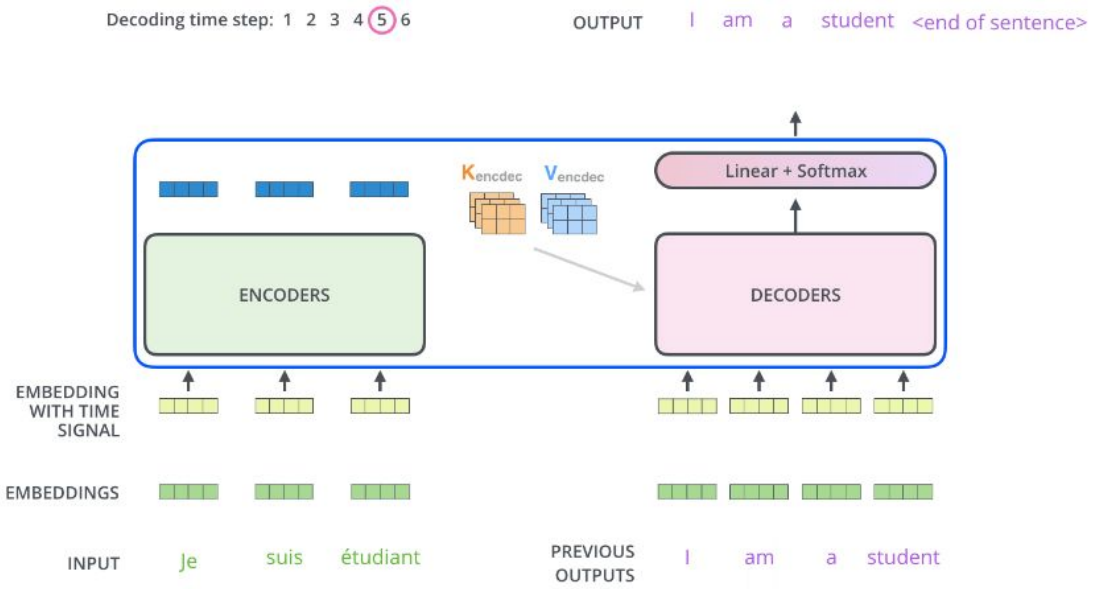
    - [0, 0, 0.9] + 0.1/3 = [0.03, 0.03, 0.93]
- Logit에 Softmax를 적용하여 확률 계산, Cross-entropy 손실함수 Optimize
- Accuracy
  - 〈PAD〉를 제외한 token에 대해서 구함

Decoder

Linear/Softmax



summary



http://jalammar.github.io/illustrated-transformer/

Training

Data : standard WMT 2014 English-German dataset consisting of about 4.5 million sentence pairs, s a shared source, target vocabulary of about 37000 tokens

Hardware : 8 NVIDIA P100 GPUs, e base models for a total of 100,000 steps or 12 hours

optimizer We used the Adam optimizer [20] with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$. We varied the learning rate over the course of training, according to the formula:

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$

regularization :

- residual dropout, dropout for input embedding
- label smoothing

Results

Machine Translation

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**	$2.3 \cdot 10^{19}$	

Results

Model variations

-positional encoding

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

attention visualization

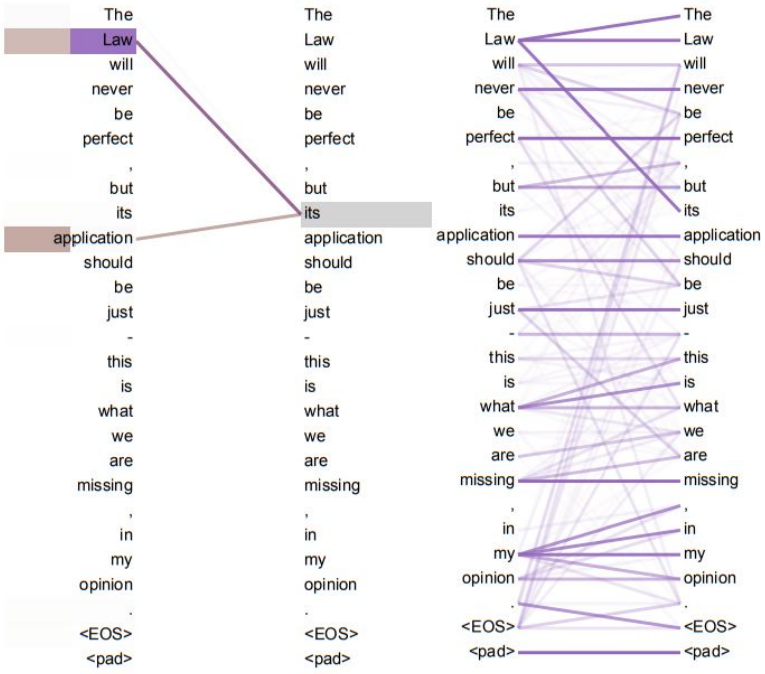
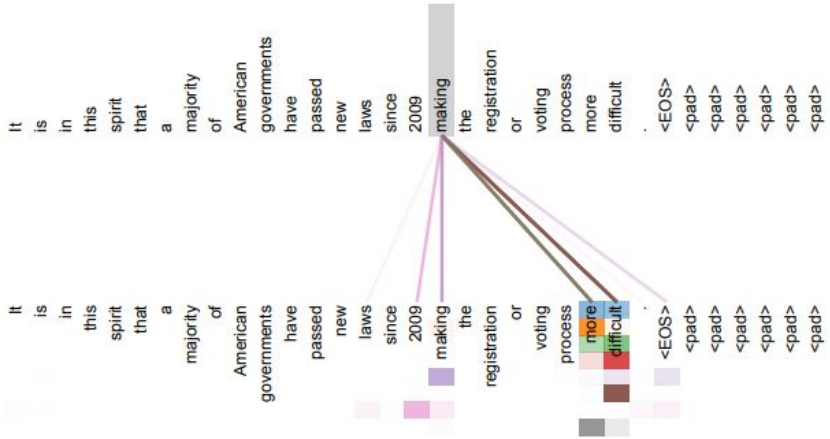


Figure 4: Two attention heads, also in layer 5 of 6, apparently involved in anaphora resolution. Top: Full attentions for head 5. Bottom: Isolated attentions from just the word 'its' for attention heads 5 and 6. Note that the attentions are very sharp for this word.



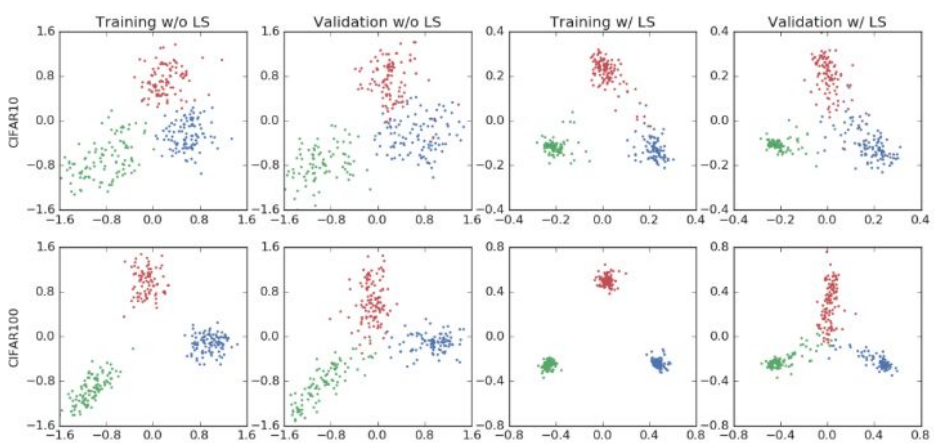
Transformer/label smoothing

label smoothing(2015년 구글 논문에서 발표)

temperate scaling

hard label의 경우 해당 레이블에 과적합하게 학습될 수 있다.

general solution을 원할 때, soft label을 쓰자.(A=0.1)



label smoothing을 적용함으로써 overfit이

완화되었음을 확인할 수 있다.

https://medium.com/@nainaakash012/when-does-label-smoothing-help-89654ec75326

The End

Thank you!