Stand-Alone Self-Attention in Vision Models

Google Research, Brain Team

NIPS, 2019

컴퓨터소프트웨어학과 석사 2기 김 경 아

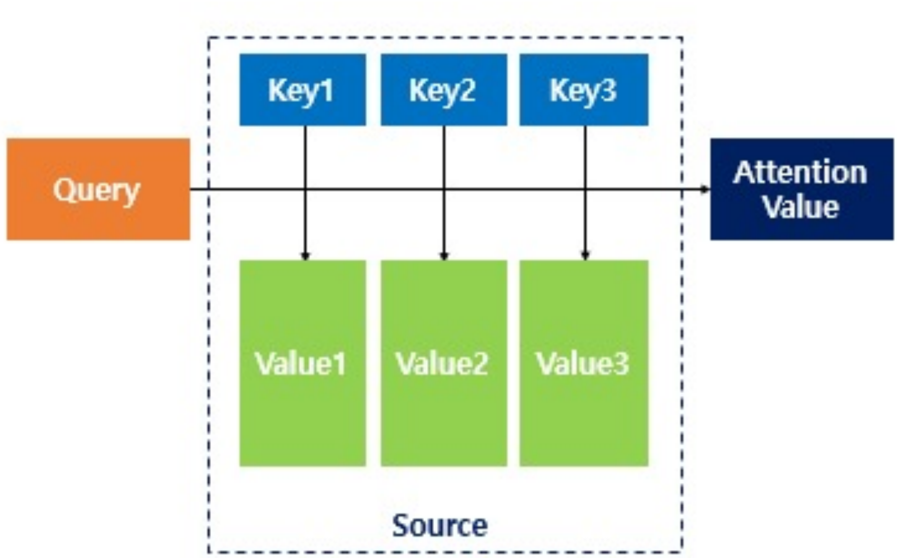
CONTENTS

01. Attention vs Self - Attention

01 Attention vs Self - Attention

Attention?

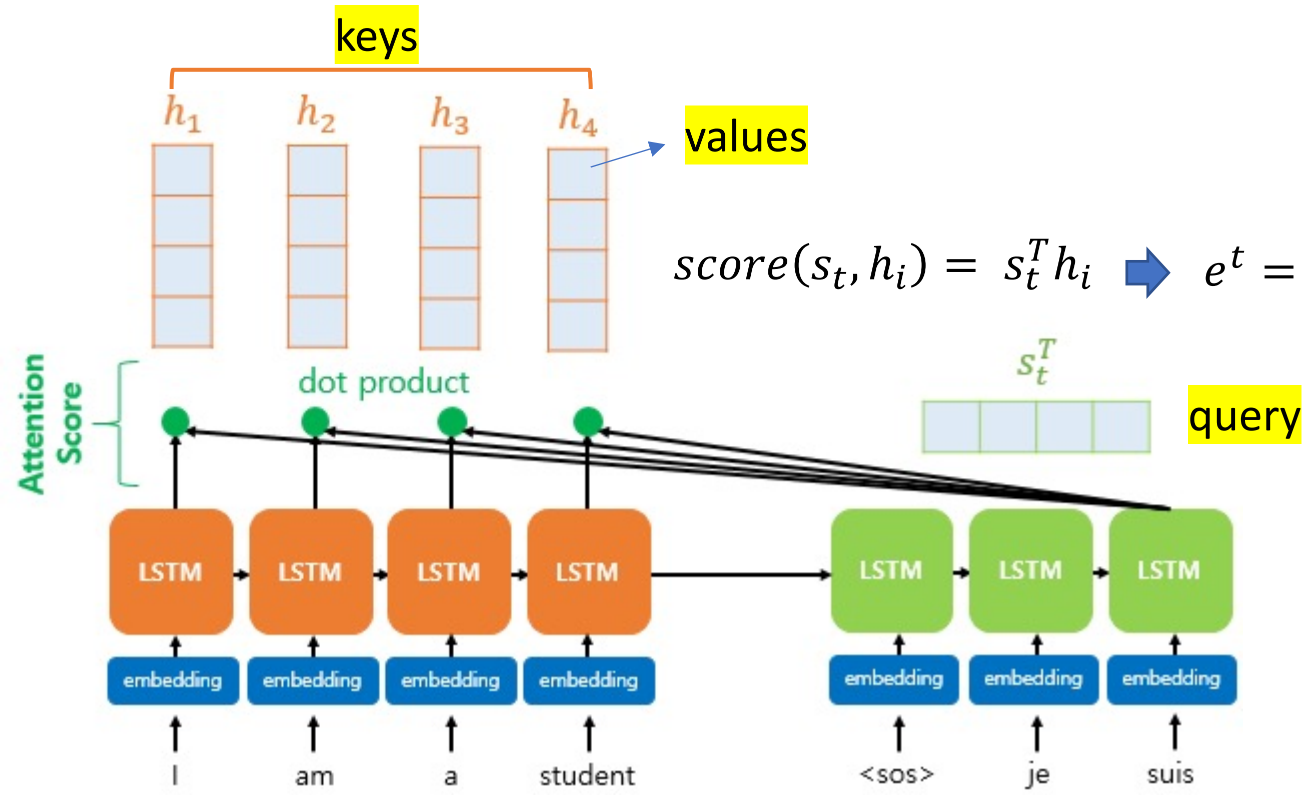
디코더에서 출력 단어를 예측하는 매 시점마다, 인코더에서의 전체 문장을 참고해서 해당 시점에서 **예측해야 할 단어**와 **연관이 있는 입력 단어 부분**을 좀 더 집중해서 보겠다.



Attention(Q, K, V) = Attention Value

01 Attention vs Self - Attention

Attention : Seq2Seq + Attention
(*NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE, ICLR 2015*)

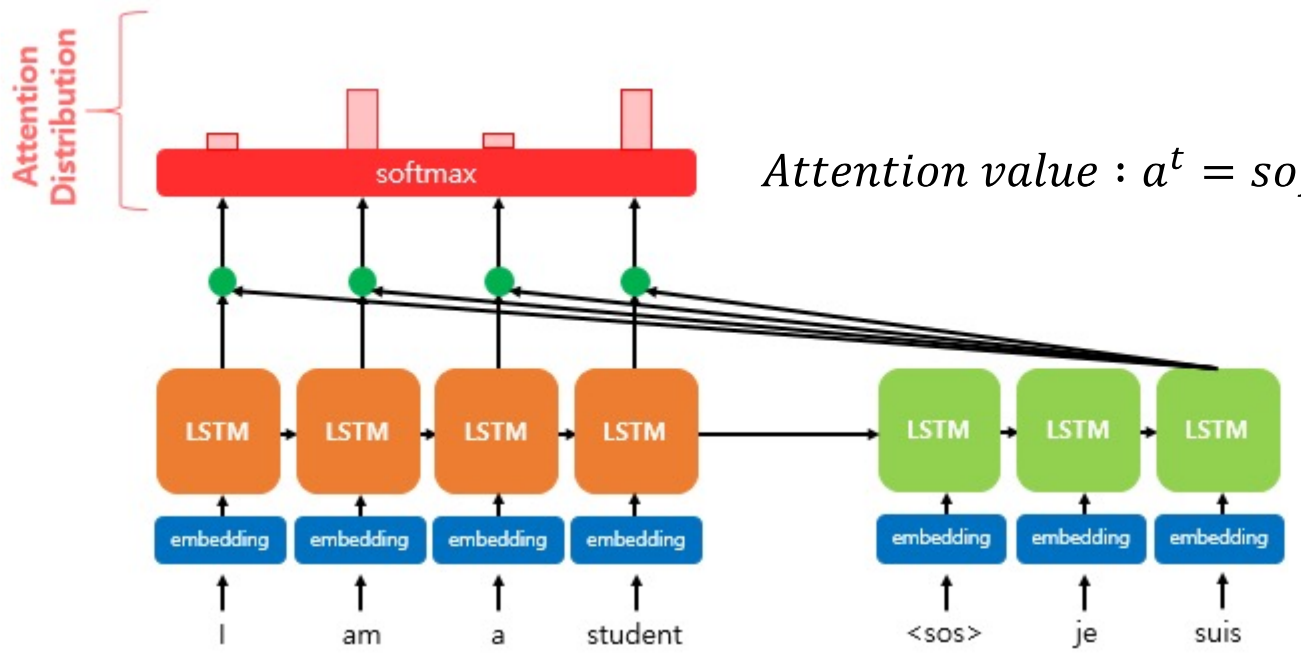


$$score(s_t, h_i) = s_t^T h_i \Rightarrow e^t = [s_t^T h_1, \ldots s_t^T h_N]$$

```
Q = Query : t 시점의 decoder 셀에서의 hidden state
K = Keys : 모든 시점의 encoder 셀의 hidden state
V = Values : 모든 시점의 encoder 셀의 hidden state
```

01 Attention vs Self - Attention

Attention : Seq2Seq + Attention

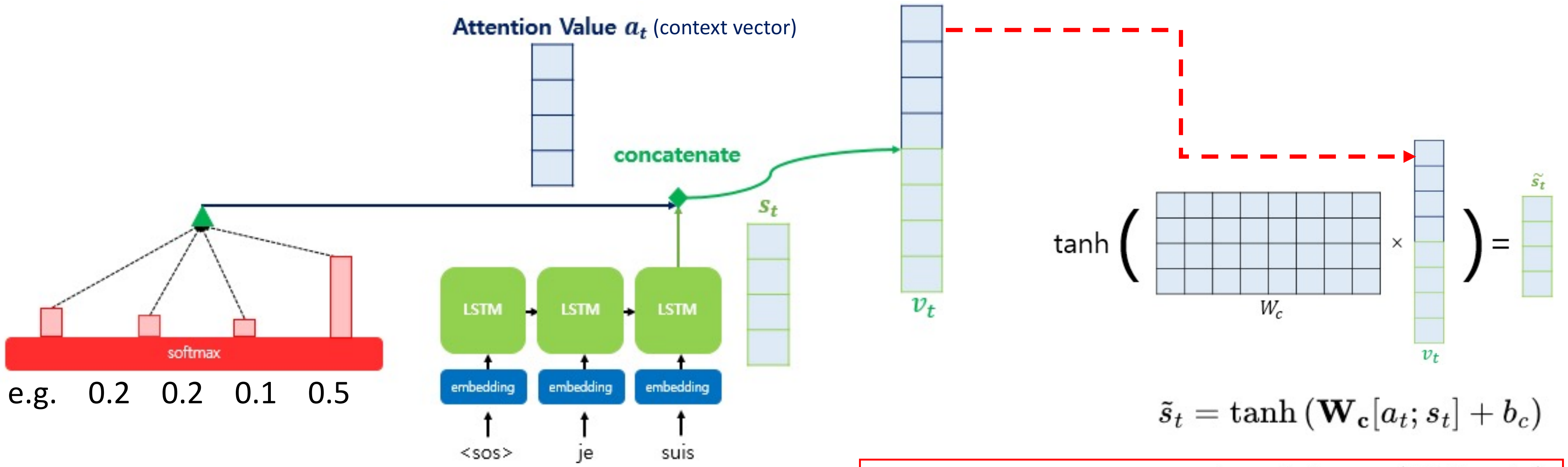


$Attention\ value : a^t = softmax(e^t)$

01 Attention vs Self - Attention

Attention : Seq2Seq + Attention

$$a_t = \sum_{i=1}^{N} \alpha_i^t h_i$$



e.g. 0.2 0.2 0.1 0.5

$$\tilde{s}_t = \tanh\left(\mathbf{W_c}[a_t; s_t] + b_c\right)$$

$\therefore$ 최종 예측 벡터 output $\hat{y}_t = \text{Softmax}\left(W_y \tilde{s}_t + b_y\right)$

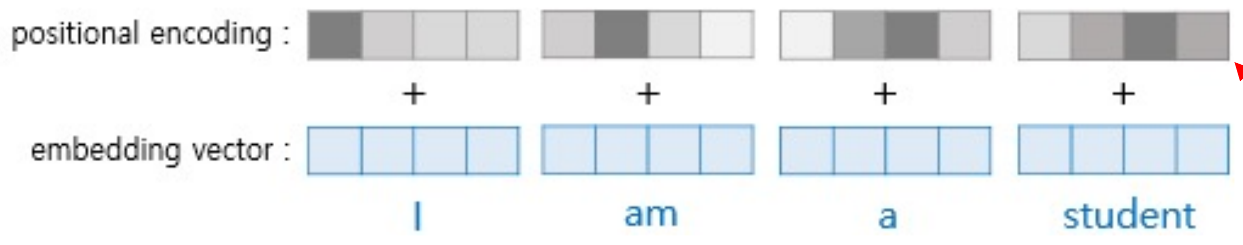
01 Attention vs Self - Attention

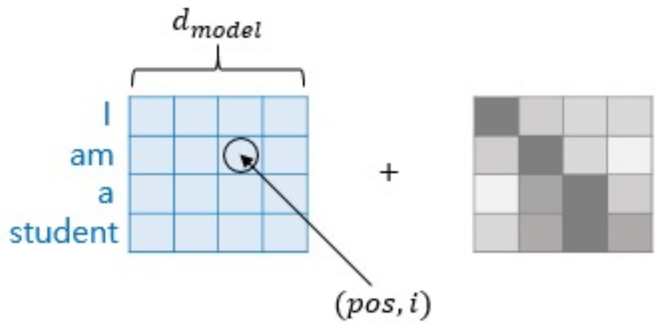
Self - Attention : Transformer (Attention Is All You Need, 2017)

1. Positional Encoding

Sequence에 있는 원소들의 위치에 대한 정보.

Sin or Cos encoding 사용





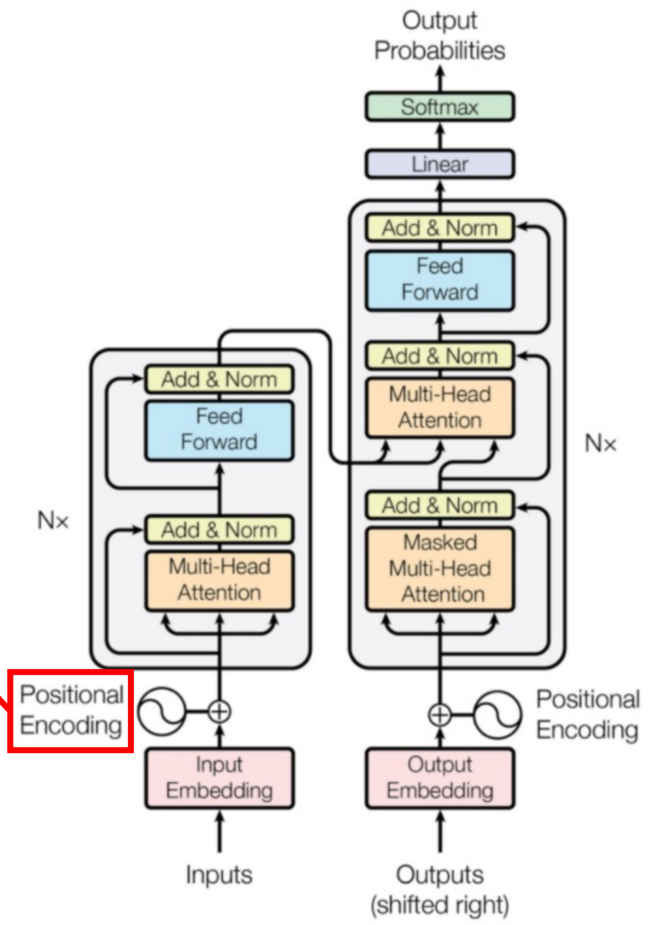


Figure 1: The Transformer - model architecture.

01 Attention vs Self - Attention

Self - Attention : Transformer (*Attention Is All You Need, 2017*)

2. Encoder and Decoder Stacks

각 sub-layer의 출력값은 $LayerNorm(x + Sublayer(x))$ 형태

- Multi head self-attention mechanism
  - Encoder Self-attention : Query = Key = Value (self)
  - Masked Decoder Self-attention : Query = Key = Value (self)
  - Encoder decoder attention : Query : decoder vector / Key = Value : encoder vector

- Position-wise fully connected feed-forward network

Encoder 와 Decoder 각 layer는 FC Feed-Forward 네트워크를 가지며, ReLU 와 2개의 선형 변환을 포함.

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

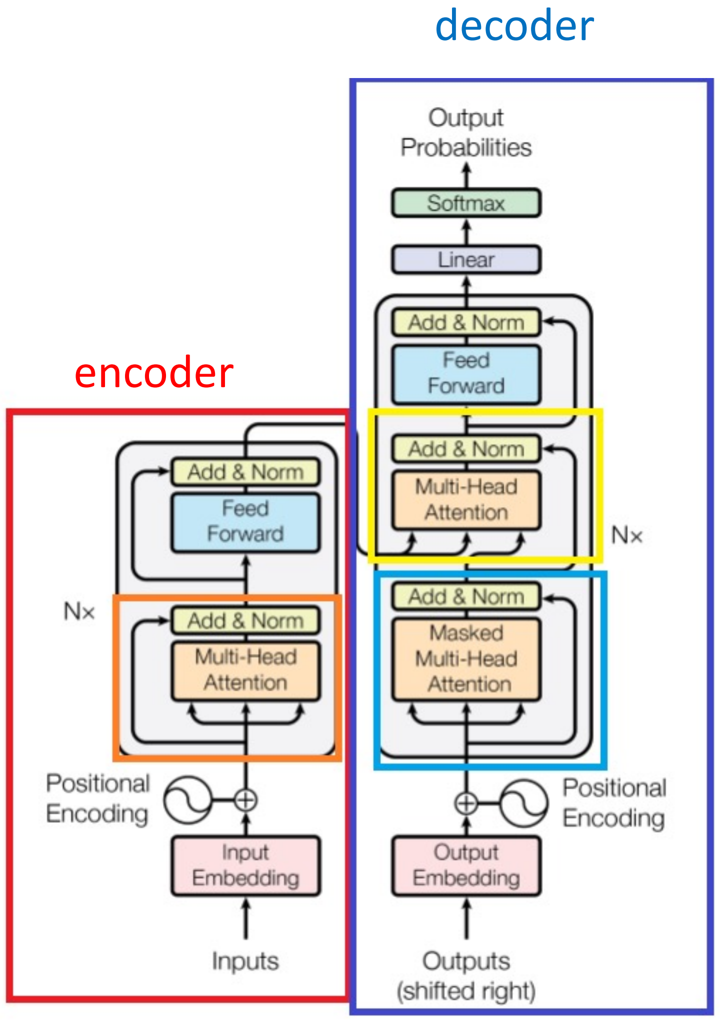
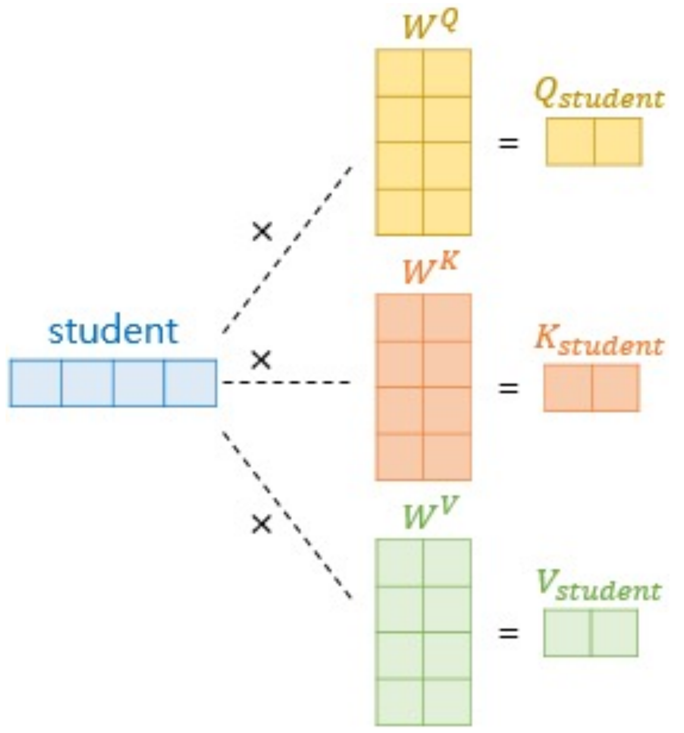


Figure 1: The Transformer - model architecture.

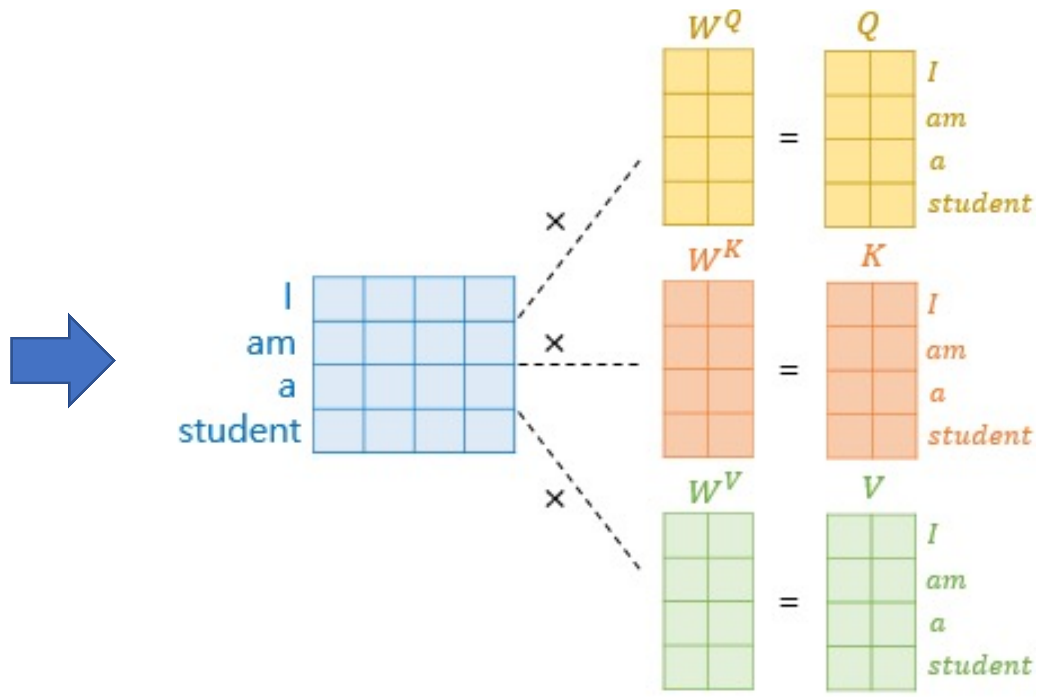
01 Attention vs Self - Attention

Self - Attention : Transformer ([Attention Is All You Need, 2017](#))

***Scaled dot-product Attention**



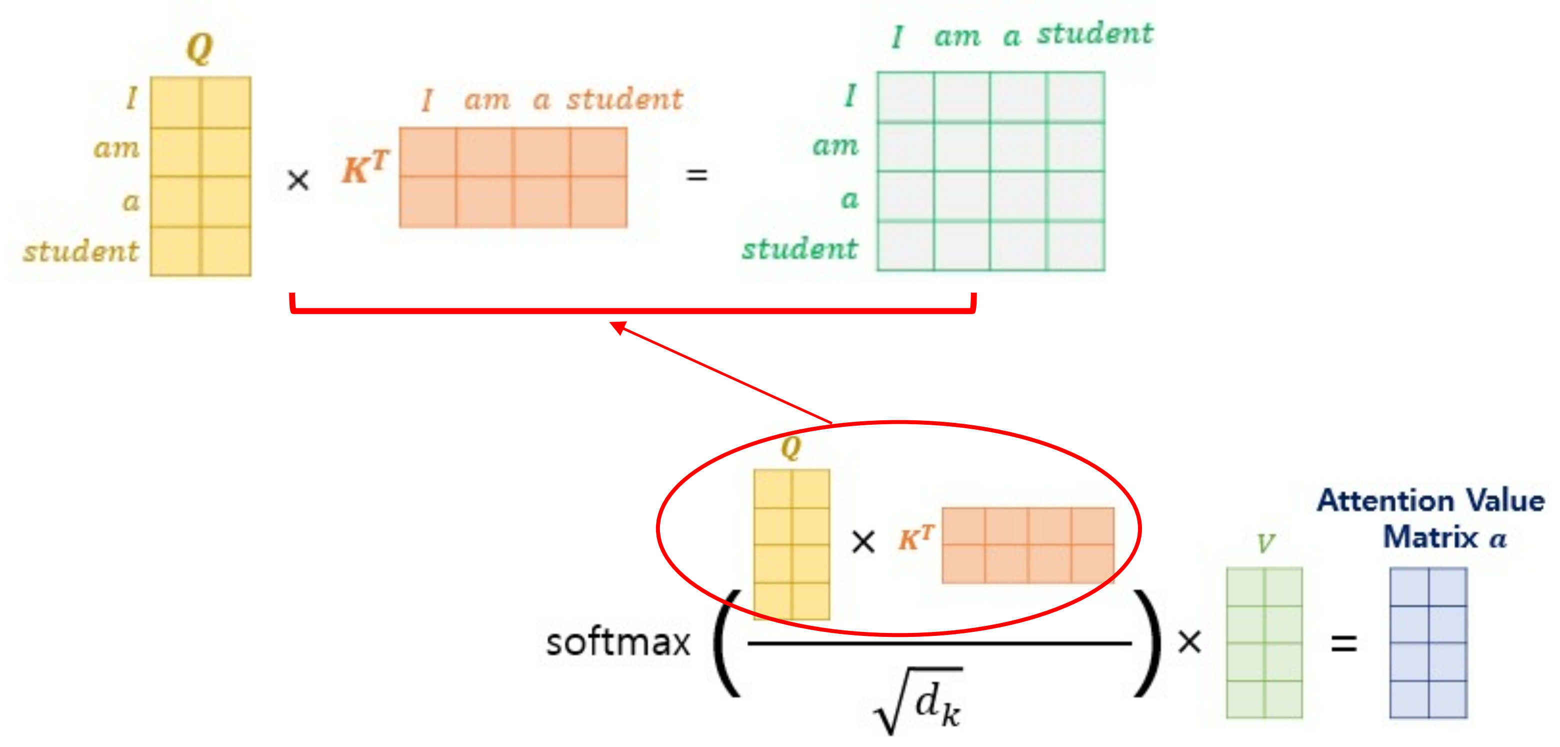
1 word



1 sentence

01 Attention vs Self - Attention

Self - Attention : Transformer (Attention Is All You Need, 2017)



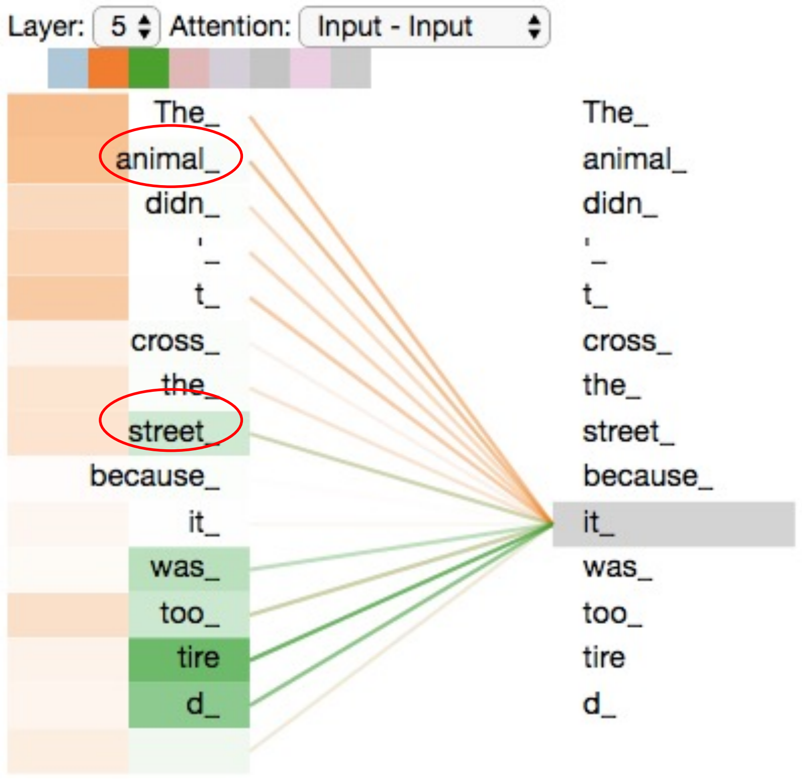
$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

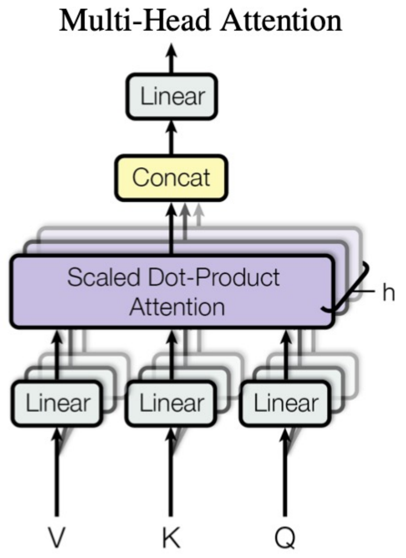
01 Attention vs Self - Attention

Self - Attention : Transformer (Attention Is All You Need, 2017)

*** Multi-Head Attention**

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





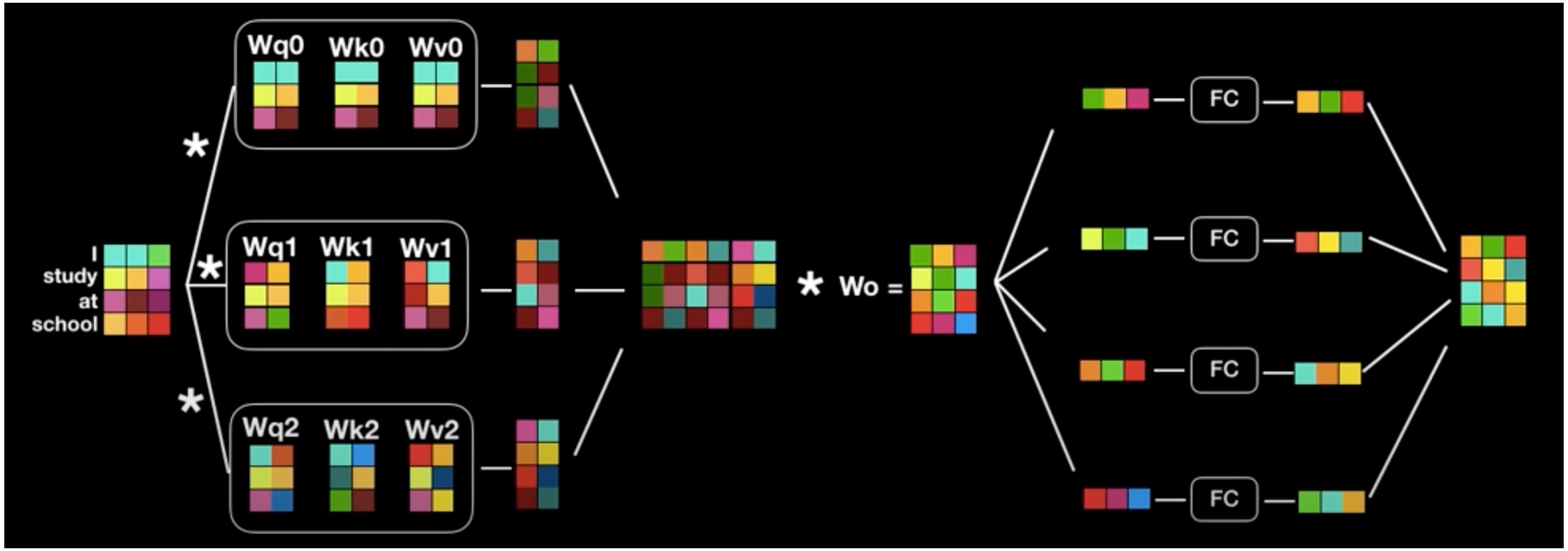
$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

01 Attention vs Self - Attention

Self - Attention : Transformer (Attention Is All You Need, 2017)

*** Multi-Head Attention**

[encoder]



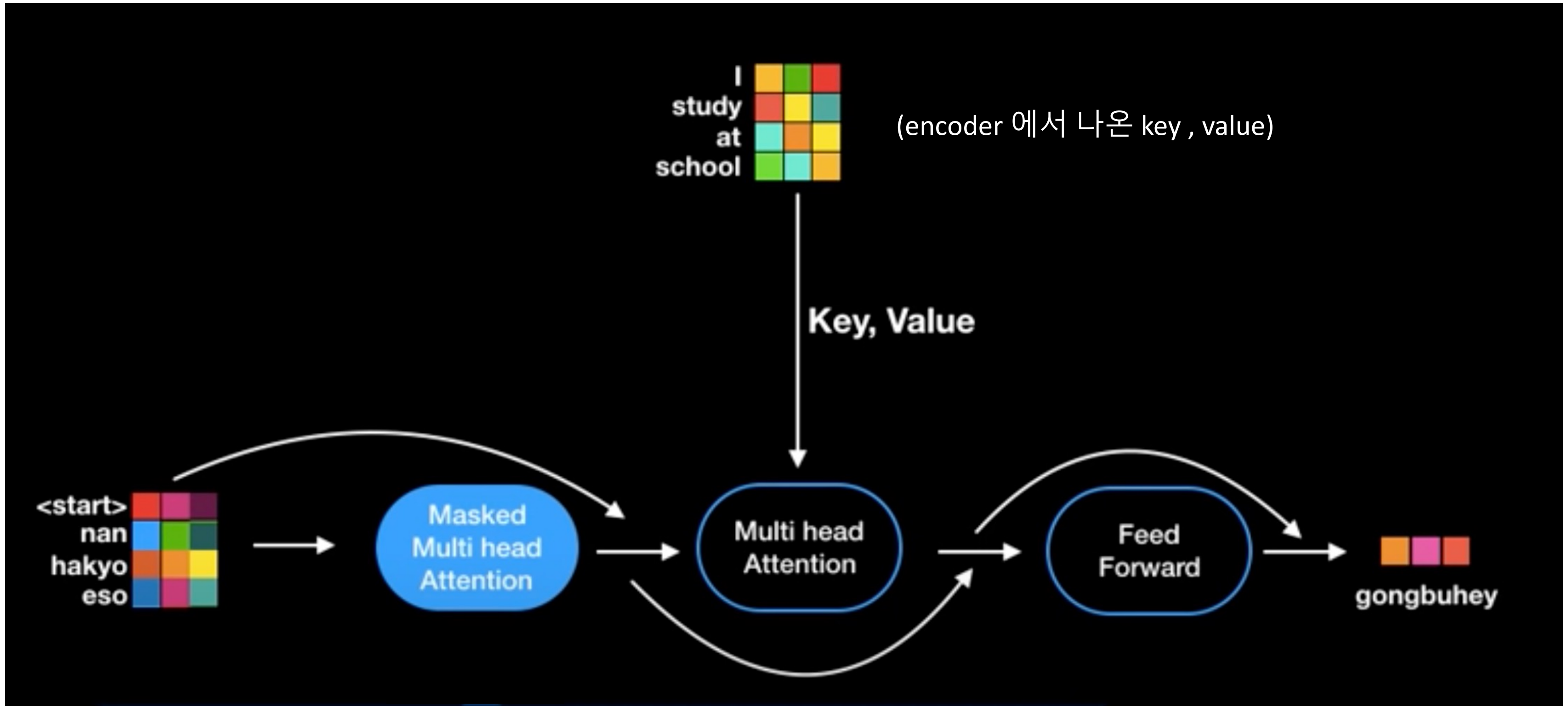
Youtube 참조 : https://www.youtube.com/watch?v=mxGCEWOxfe8

01 Attention vs Self - Attention

Self - Attention : Transformer (Attention Is All You Need, 2017)

*** Multi-Head Attention**

[decoder]



02. Background of Stand Alone self-attention

02 Background of Stand Alone Self-Attention

< 기존 문제점 >

1) Convolution 연산은 long range dependency를 고려하기 어렵다.

2) Global attention 형태인 attention block을 residual network 사이에 넣는 방법도 존재하나, 어느정도 spatial한 down Sampling이 이루어진 후에 추가하게 된다. 모든 픽셀간의 correlation을 구하기 때문에 computation cost가 많다.

“ Stand Alone Self-Attention ”

Convolution layer 대신에 **self attention block을 가지는 layer 만**으로 content-based interaction을 할 수 있는 방법.

Param 수는 훨씬 적고 computation cost이 적지만 성능은 기존의 베이스라인보다 더 좋음!

02 Background of Stand Alone Self-Attention

Convolutions

$$y_{ij} = \sum_{a,b \in \mathcal{N}_k(i,j)} W_{i-a,j-b}\, x_{ab}$$

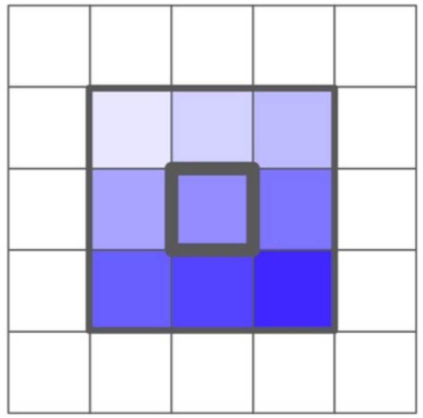


Figure 1: An example of a local window around $i = 3, j = 3$ (one-indexed) with spatial extent $k = 3$.

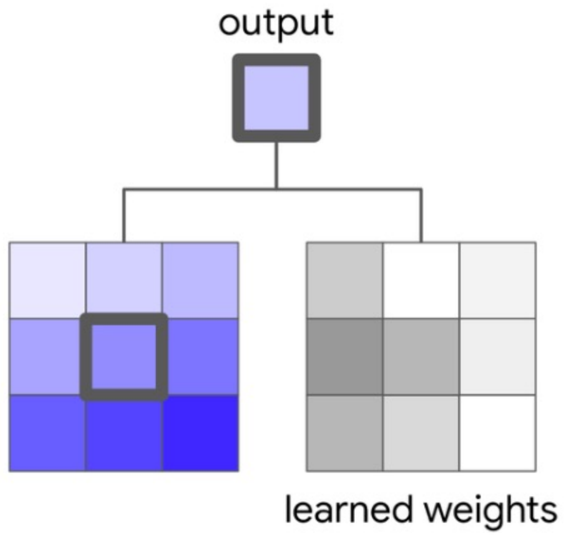


Figure 2: An example of a 3×3 convolution. The output is the inner product between the local window and the learned weights.

02 Background of Stand Alone Self-Attention

Self-Attention

$$y_{ij} = \sum_{a,b \in \mathcal{N}_k(i,j)} \texttt{softmax}_{ab}\left(q_{ij}^{\top} k_{ab}\right) v_{ab}$$

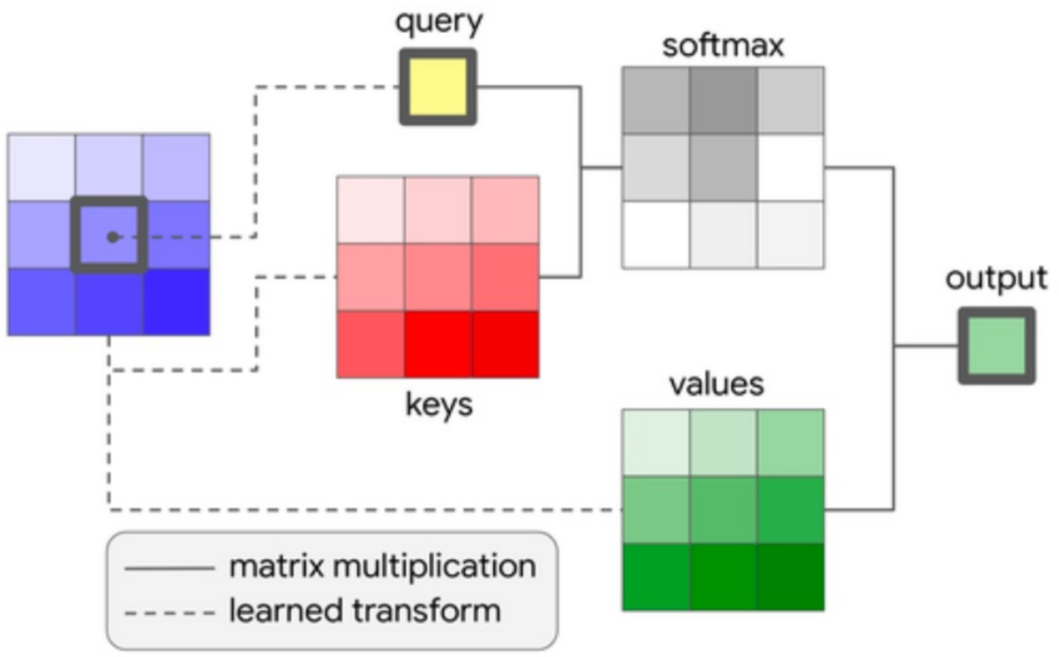


Figure 3: An example of a local attention layer over spatial extent of $k = 3$.

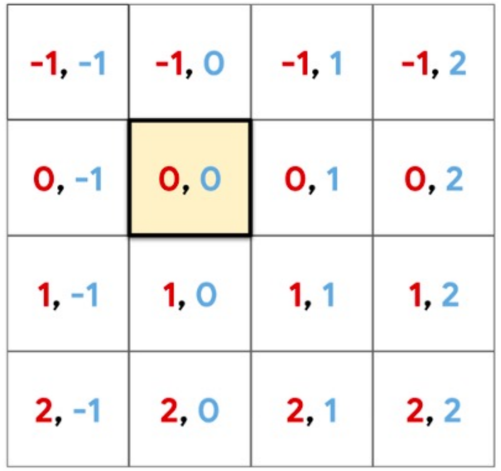
02 Background of Stand Alone Self-Attention

Self-Attention with Relative distance

$$y_{ij} = \sum_{a,b\in\mathcal{N}_k(i,j)} \texttt{softmax}_{ab}\left(q_{ij}^\top k_{ab} + \boxed{q_{ij}^\top r_{a-i,b-j}}\right) v_{ab}$$

relative distance (positional encoding)

- row offset : a-i
- column offset : b-j

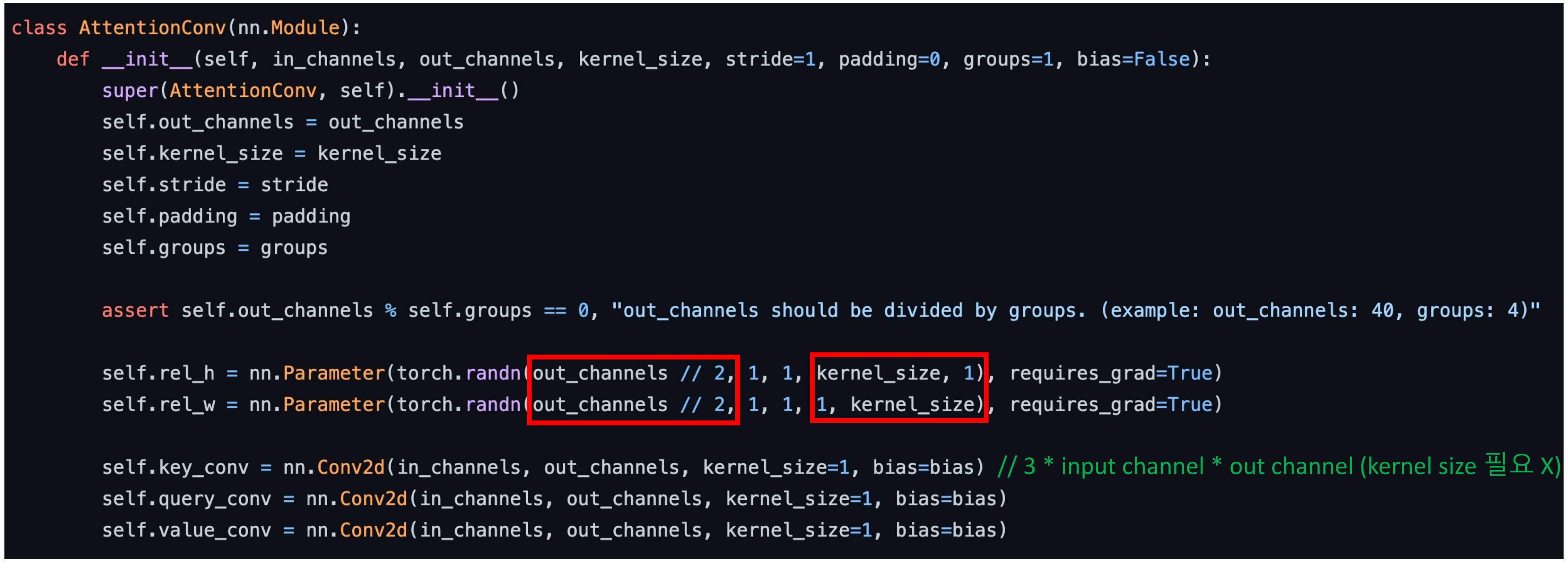


-1, -1	-1, 0	-1, 1	-1, 2
0, -1	0, 0	0, 1	0, 2
1, -1	1, 0	1, 1	1, 2
2, -1	2, 0	2, 1	2, 2

Figure 4: An example of relative distance computation. The relative distances are computed with respect to the position of the highlighted pixel. The format of distances is *row offset*, *column offset*.

02 Background of Stand Alone Self-Attention

Self-Attention with Relative distance



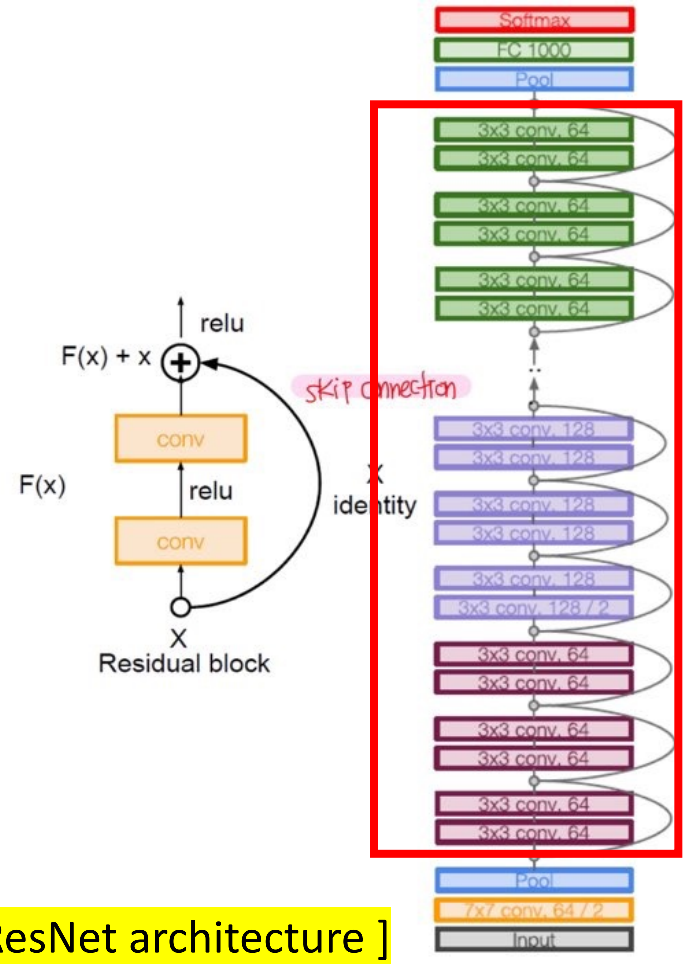
```
class AttentionConv(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, stride=1, padding=0, groups=1, bias=False):
        super(AttentionConv, self).__init__()
        self.out_channels = out_channels
        self.kernel_size = kernel_size
        self.stride = stride
        self.padding = padding
        self.groups = groups

        assert self.out_channels % self.groups == 0, "out_channels should be divided by groups. (example: out_channels: 40, groups: 4)"

        self.rel_h = nn.Parameter(torch.randn(out_channels // 2, 1, 1, kernel_size, 1), requires_grad=True)
        self.rel_w = nn.Parameter(torch.randn(out_channels // 2, 1, 1, 1, kernel_size), requires_grad=True)

        self.key_conv = nn.Conv2d(in_channels, out_channels, kernel_size=1, bias=bias) // 3 * input channel * out channel (kernel size 필요 X)
        self.query_conv = nn.Conv2d(in_channels, out_channels, kernel_size=1, bias=bias)
        self.value_conv = nn.Conv2d(in_channels, out_channels, kernel_size=1, bias=bias)
```

03. Methodology of Stand Alone self-attention

03 Replacing Spatial Convolutions

본 논문에서 제안하는 첫번째 방법은 **spatial 한 convolution을 전부 self attention layer로 변경**하는 것.

- ✓ spatial convolution이란 kernel size(filter size) > 1 을 가지는 convolution을 의미.
  (1 * 1 convolution을 제외)
- ✓ spatial downsampling 이 필요한 경우, attention layer 이후에 stride 2인 2 * 2 average pooling을 합니다.
- ✓ ResNet 아키텍쳐를 사용했으며, ResNet bottleneck block 중 3 * 3 convolution layer 만 self attention으로 대체.



[ResNet architecture]

03 Replacing the Convolutional Stem

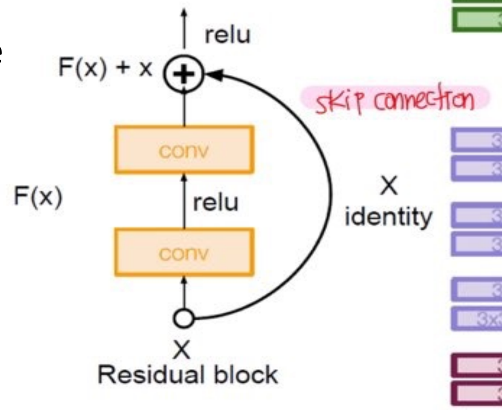
두번째로 제안하는 방법은 Stem 대체, 다시 말해 **CNN 앞단에서 사용하는 layer**를 변형하는 방법

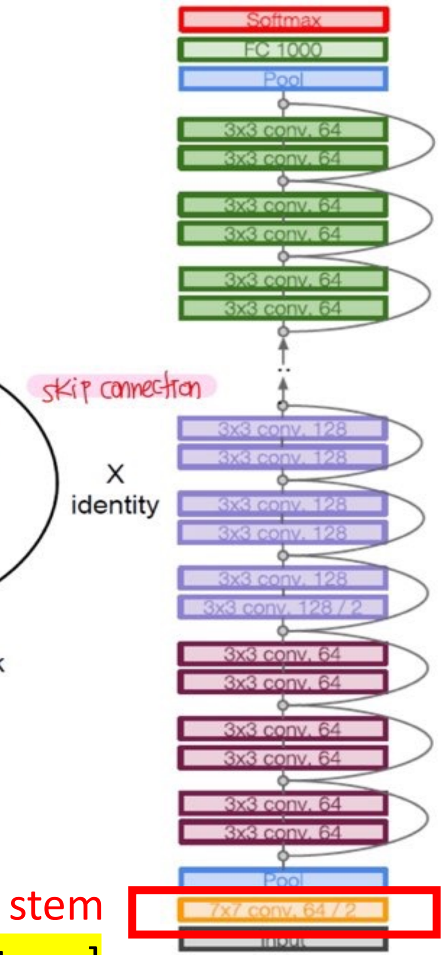
- ✓ Stem이란 initial layer.
- ✓ 전체를 Self attention으로 바꾸면 underperform 하다.
- ✓ 그래서, attention Stem을 제안. Stem layer의 value 값을 변형하여 spatial 한 정보(Distance based information)을 추가한 것. => Spatially-varying linear transform 추구

$$v_{ab} = \left(\sum_m \underline{p(a,b,m)} W_V^m \right) x_{ab}$$

$$p(a,b,m) = \texttt{softmax}_m \left(\left(\mathbf{emb}_{row}(a) + \mathbf{emb}_{col}(b) \right)^\top \nu^m \right)$$

주변 픽셀들의 위치를 확률로 나타냄. (absolute embedding a + b) * mixture embedding ν^m



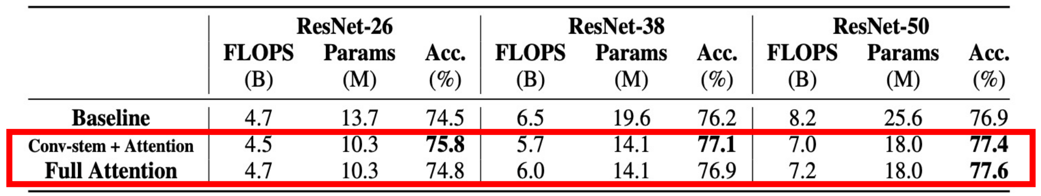


[ResNet architecture]

04. Experiment / Result

04 ImageNet Classification Result

[Result table]



	ResNet-26			ResNet-38			ResNet-50		
	FLOPS (B)	Params (M)	Acc. (%)	FLOPS (B)	Params (M)	Acc. (%)	FLOPS (B)	Params (M)	Acc. (%)
Baseline	4.7	13.7	74.5	6.5	19.6	76.2	8.2	25.6	76.9
Conv-stem + Attention	4.5	10.3	**75.8**	5.7	14.1	**77.1**	7.0	18.0	**77.4**
Full Attention	4.7	10.3	74.8	6.0	14.1	76.9	7.2	18.0	**77.6**

Table 1: ImageNet classification results for a ResNet network with different depths. *Baseline* is a standard ResNet, *Conv-stem + Attention* uses spatial convolution in the stem and attention everywhere else, and *Full Attention* uses attention everywhere including the stem. The attention models outperform the baseline across all depths while having 12% fewer FLOPS and 29% fewer parameters.

[Result figure]

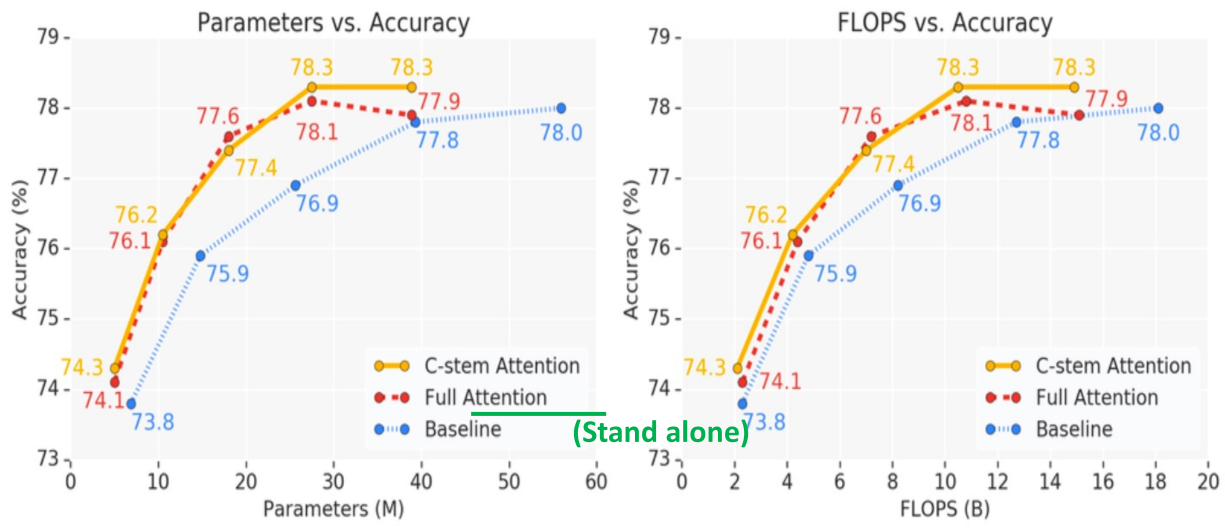
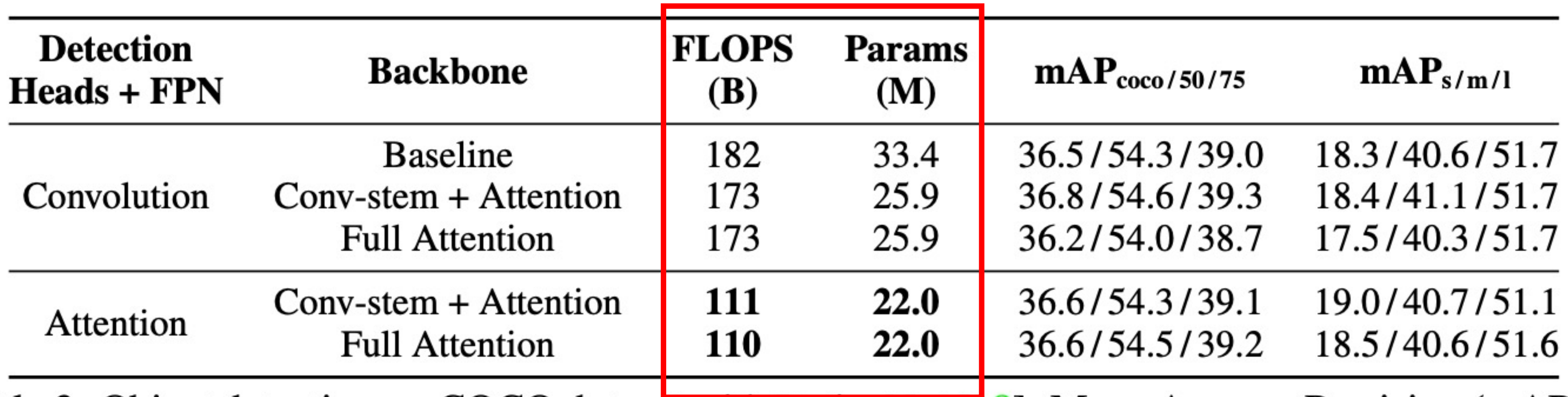


Figure 5: Comparing parameters and FLOPS against accuracy on ImageNet classification across a range of network widths for ResNet-50. Attention models have fewer parameters and FLOPS while improving upon the accuracy of the baseline.

04 COCO Object detection Result

[Result table]



Detection Heads + FPN	Backbone	FLOPS (B)	Params (M)	$\mathrm{mAP}_{\mathrm{coco}/50/75}$	$\mathrm{mAP}_{s/m/l}$
Convolution	Baseline	182	33.4	36.5 / 54.3 / 39.0	18.3 / 40.6 / 51.7
	Conv-stem + Attention	173	25.9	36.8 / 54.6 / 39.3	18.4 / 41.1 / 51.7
	Full Attention	173	25.9	36.2 / 54.0 / 38.7	17.5 / 40.3 / 51.7
Attention	Conv-stem + Attention	**111**	**22.0**	36.6 / 54.3 / 39.1	19.0 / 40.7 / 51.1
	Full Attention	**110**	**22.0**	36.6 / 54.5 / 39.2	18.5 / 40.6 / 51.6

Table 2: Object detection on COCO dataset with RetinaNet [18]. Mean Average Precision (mAP) is reported at three different IoU values and for three different object sizes (small, medium, large). The fully attentional models achieve similar mAP as the baseline while having up to 39% fewer FLOPS and 34% fewer parameters.

References

[research paper]

- *Ramachandran, P., Parmar, N., Vaswani, A., Bello, I., Levskaya, A., & Shlens, J. (NIPS 2019). Stand-alone self-attention in vision models. arXiv preprint arXiv:1906.05909.*
- *Ashish Vaswani, Llion Jones, Noam Shazeer Google Brain (2017). Attention Is All You Need*

[참고 자료]

- *PR-163 : CNN_Attention_Neworks*
- *PR-193 : Stand Alone Self Attention in vision models*
- *딥러닝을 위한 자연어처리 입문* : https://wikidocs.net/31379
- *https://towardsdatascience.com/self-attention-in-computer-vision-2782727021f6*
- *Pytorch code : https://github.com/leaderj1001/Stand-Alone-Self-Attention*

Q & A

감사합니다.

E-mail : kka960602@gmail.com
Github :https://github.com/Kyeonga-Kim/