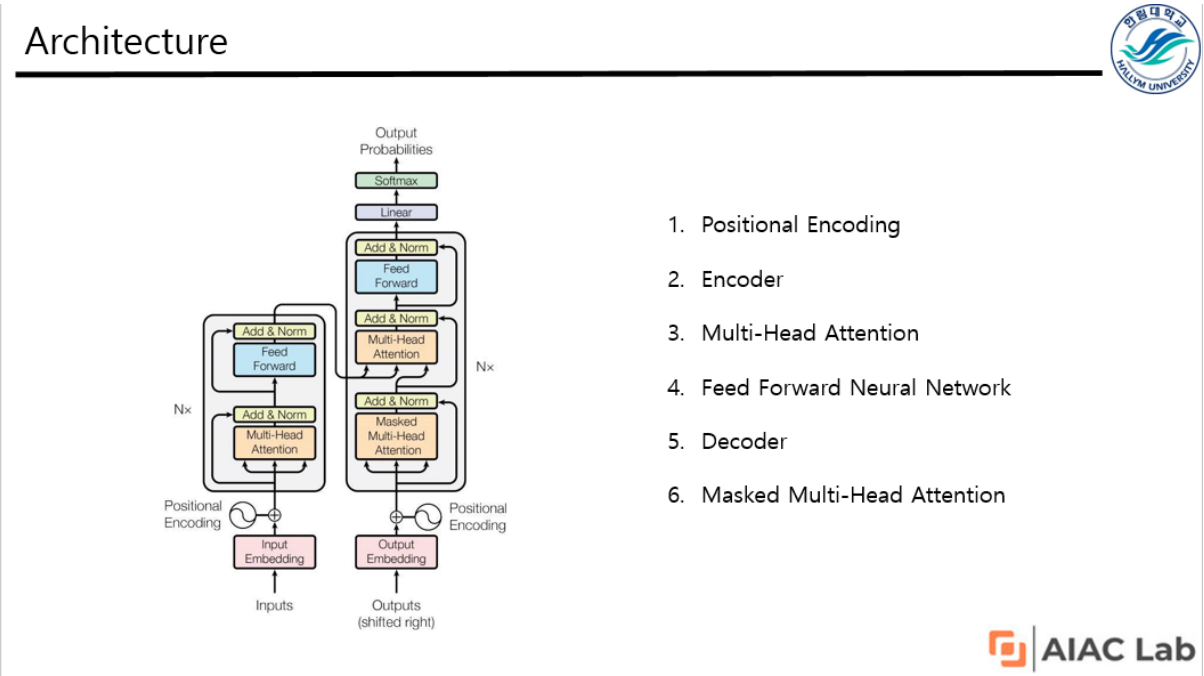
6 / 25 (일)

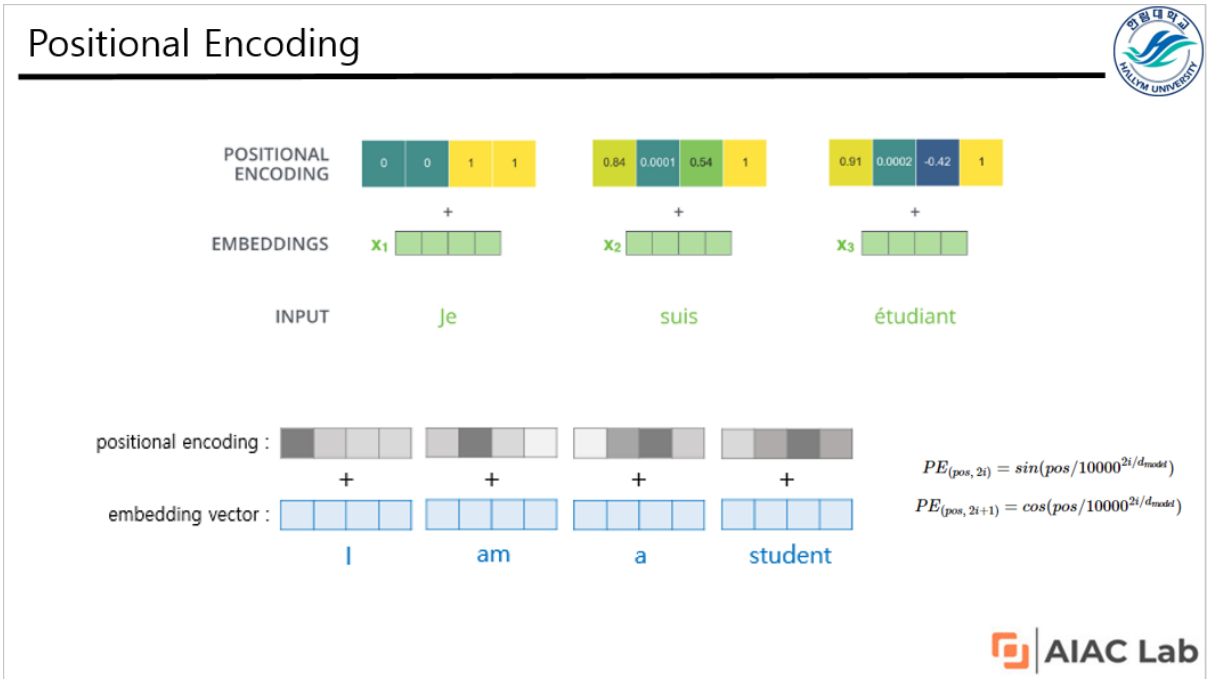
Transformer 정리

Transformer.pptx

Architecture



Positional Encoding

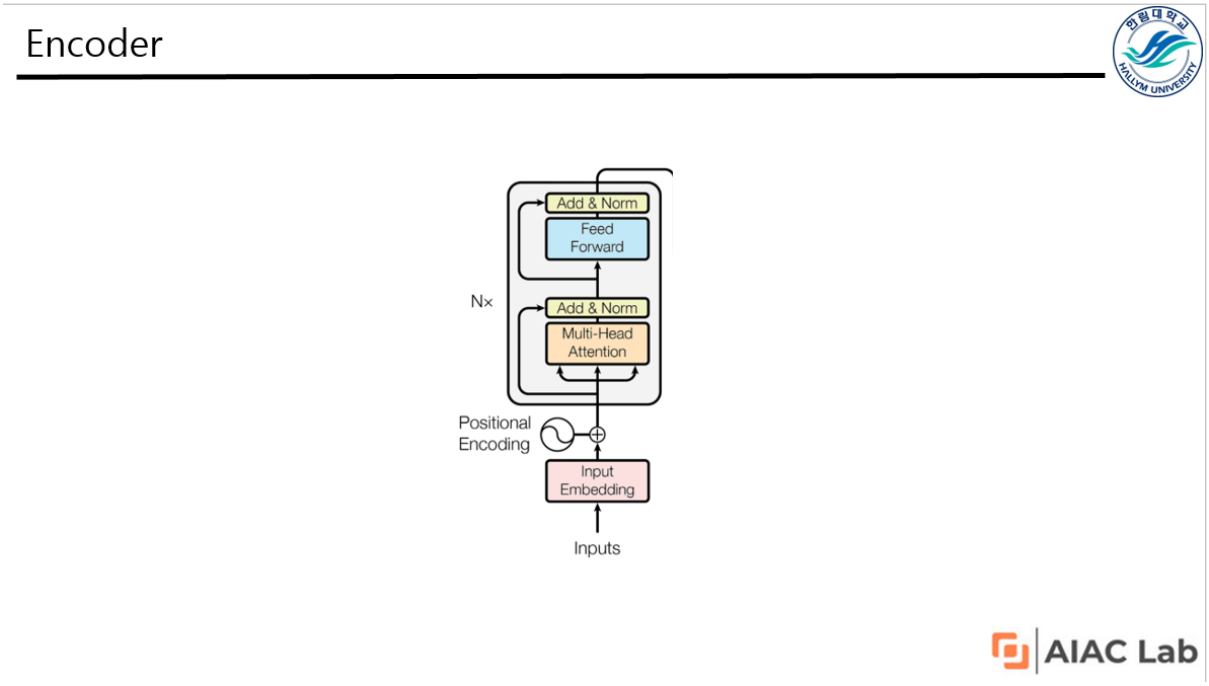


Transformer말고 자연어 처리에서 유용하게 사용되었던, RNN은 단어의 위치에 따라 단어를 순차적으로 입력 받아서 처리하는 RNN의 특성으로 인해서 각 단어의 위치 정보를 가질 수 있음.

하지만 Transformer에서는 단어의 위치 정보를 얻기 위해서 각 단어의 임베딩 벡터에 위치 정보들을 더해서 모델의 입력으로 사용하는데, 이것을 Positional Encoding이라고 함.

여기서 생길 수 있는 의문점이 “Positional Encoding 값은 어떤 값이길래 위치 정보를 반영할 수 있을까?” 생각이 드는데, Transformer는 위치 정보를 가진 값을 만들기 위해서 Sine, Cosine 함수를 사용해서 만들게 된다. 간단하게 말하면 임베딩 벡터 내의 각 차원의 인덱스가 짝수인 경우에는 Sine 함수를 사용하고, 홀수인 경우에는 Cosine 함수를 사용함.

Encoder



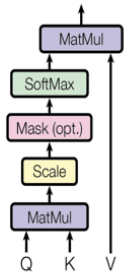
논문에서는 Encoder, Decoder를 각각 6개씩 사용했다고 말을 하는데, Encoder 하나는 Multi-head-attention과 Feed forward neural network로 이루어지게 됨.

Encoder는 Input Embedding에 Positional Encoding을 한 후, Multi-head-attention을 진행하고 Feed forward neural network를 통해서 Output을 만들게 됨.

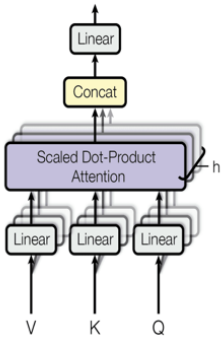
Self Attention

Self Attention

Scaled Dot-Product Attention



Multi-Head Attention



"I am a student" 에 대해서 학습을 한다고 가정 할 때

1. "I am a student"를 Vector화
2. Vector화 된 데이터를 Query, Key, Value로 활용할 수 있도록 Linear layer를 통과
3. Attention Score를 계산하기 위해서 Query와 Key에 대해서 연산
4. Softmax를 통과하여 합이 1이 되도록 설정
5. 최종적으로 Value에 attenstion score를 반영하여 최종 output 연산

AIAC Lab

앞에서 Single attention을 구하고, 이걸 num_heads만큼 진행하면 그게 Multi-head Attention임.

어떠한 단어가 문장 내에서 하나의 관계만 갖지 않고 여러 경우의 수가 있기 때문에 여러 방면에서 보기 위해서 Multi-head attention을 사용함.

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

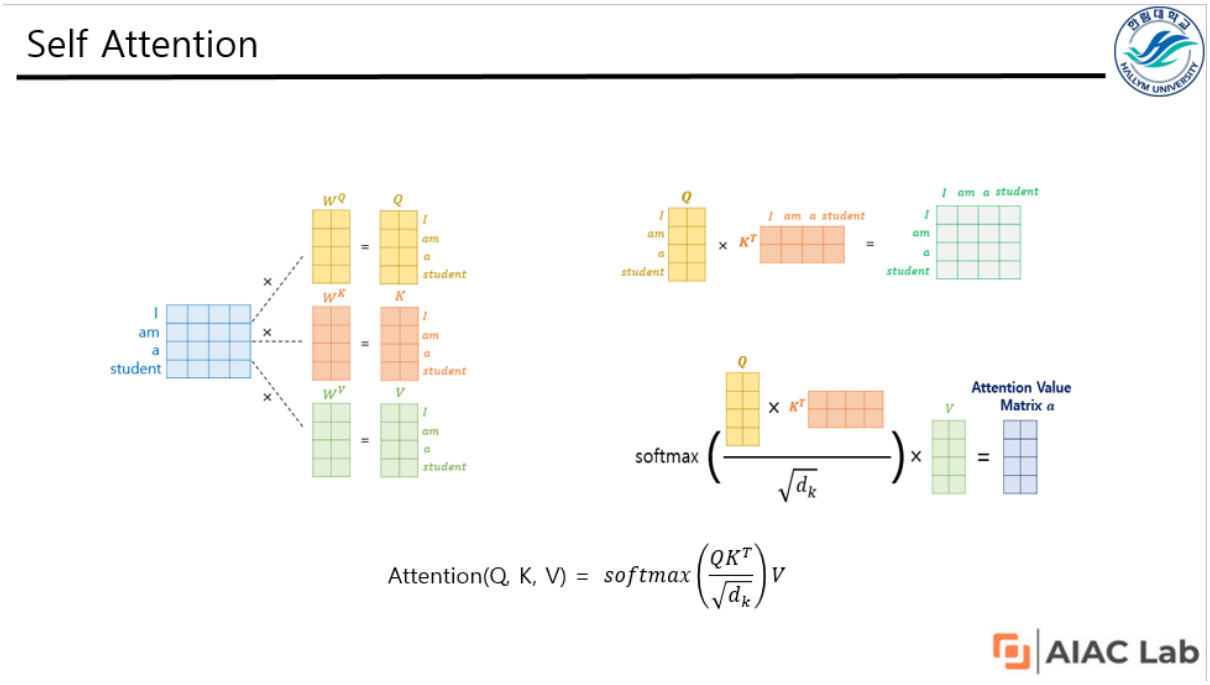
$$\text{MultiHead(Q, K, V)} = \text{Concat}(head_1, \dots, head_h)W^O$$

$$where\ head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

$$W_i^Q \in \mathbb{R}^{d_{model} x\, d_k},\ W_i^K \in \mathbb{R}^{d_{model} x\, d_k},$$
$$W_i^V \in \mathbb{R}^{d_{model} x\, d_k} \text{ and } W^O \in \mathbb{R}^{h d_v x\, d_{model}}$$

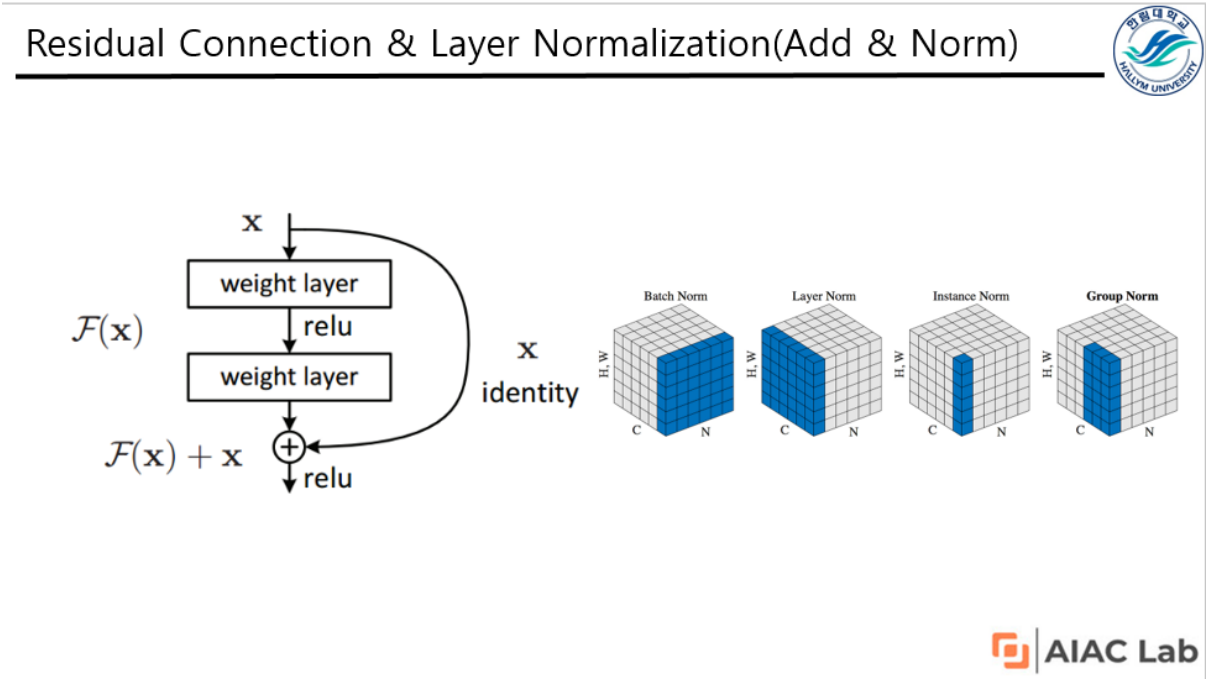
$$\text{h} = 8,\ d_k = d_v = \frac{d_{model}}{h} = 64$$

Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality



I am a student의 문장이 vector화 되어서 Self attention이 일어나는 과정

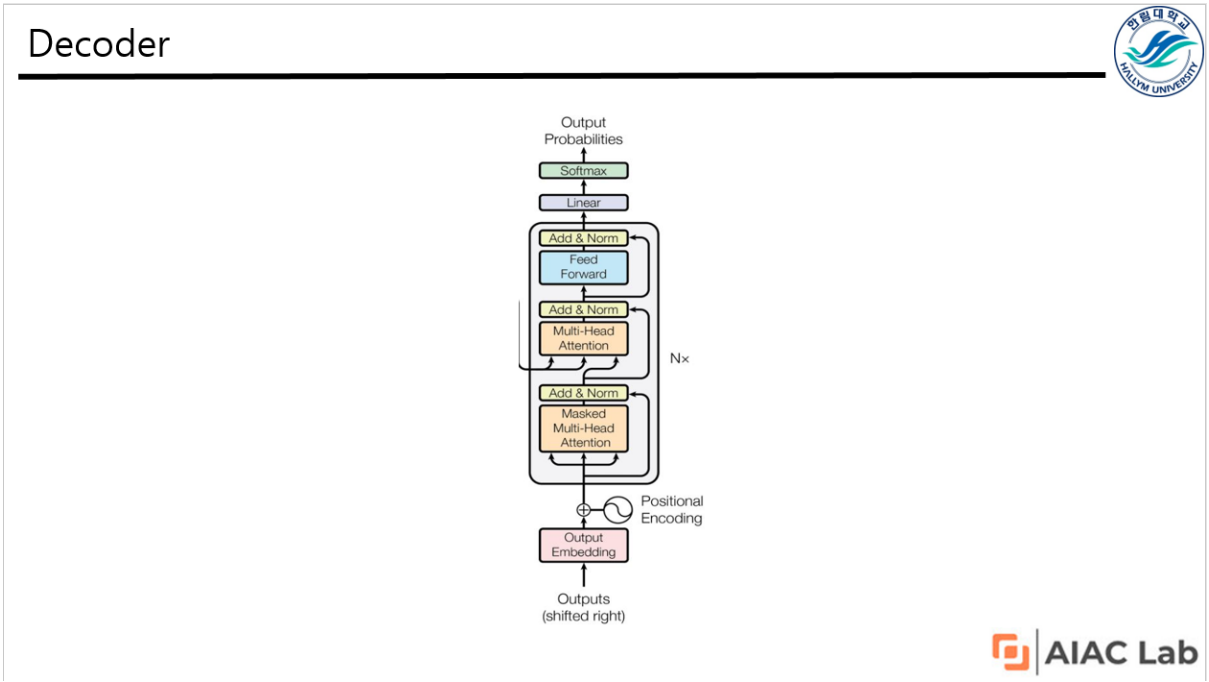
Residual Connection & Layer Normalization(ADD & Norm)



Residual Connection은 이전의 입력과 출력을 더하는 것. 깊은 신경망에서 하위 층에서 학습된 정보가 데이터 처리 과정에서 손실되는 것을 방지하기 위해서 사용.

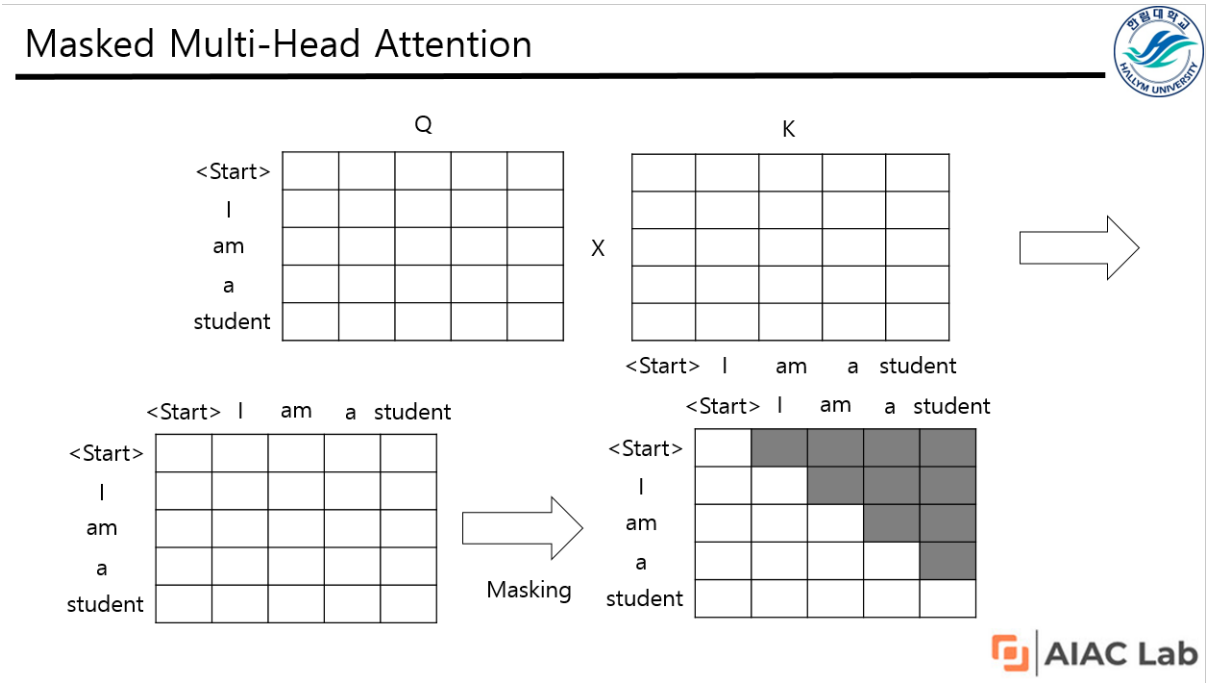
Self Attention층에서 출력된 행렬은 FFNN(Feed Forward Neural Network)층으로 가기 전에 Residual Block과 Layer Normalization을 거치게 되는데, Residual Block은 Self-Attention층을 통과한 출력 벡터에 원래의 입력 벡터를 더해주는 과정.

Decoder



Decoder는 Encoder에서 Key값과, Value 값을 가져오게 되고, Encoder와의 차이점으로는 Masked multi head attention이 있음.

Masked Multi-Head Attention



Masked, Masking은 가린다는 의미임.

Decoder에서의 Self-Attention Layer는 반드시 자기자신보다 앞쪽에 있는 Token들의 Attention score만을 볼 수 있음.

Decoder에서는 현재 내가 알고 있는 정보까지만 Self-Attention이 가능함.

Token이 그 이후에 등장하는 단어 정보에 관해 인코딩된 벡터를 생성하게 되면 이는 결과적으로 앞 단어의 인코딩 벡터가 뒤의 단어의 정보를 반영하게 되는 현상이 발생함 이러한 문제를 막기 위해서 Query와 Key vector 간의 내적을 통해서 Attention score를 구한 후 해당 index 단어의 뒤에 등장하는 단어와의 Attention scoresms 0으로 만들어서 후처리하는 것이 Masked multi head attention임.

앞에 등장하는 단어를 인코딩할 때 뒤에 등장하는 단어의 정보를 반영하지 못하도록 정보의 접근 권한을 차단하는 역할을 한다고 할 수 있음.

요약하면, Masking은 Self-Attention 수행 시 행렬 Q와 K를 곱한 결과인 가중치 행렬에 적용하는데, 이는 Self-Attention 수행 시 앞 단어를 예측할 때 뒤쪽의 정보가 반영되는 것을 사전에 차단하기 위한 용도임.