AI Seminar

- Self Attention -

한양대학교 AI Lab
이 주 홍

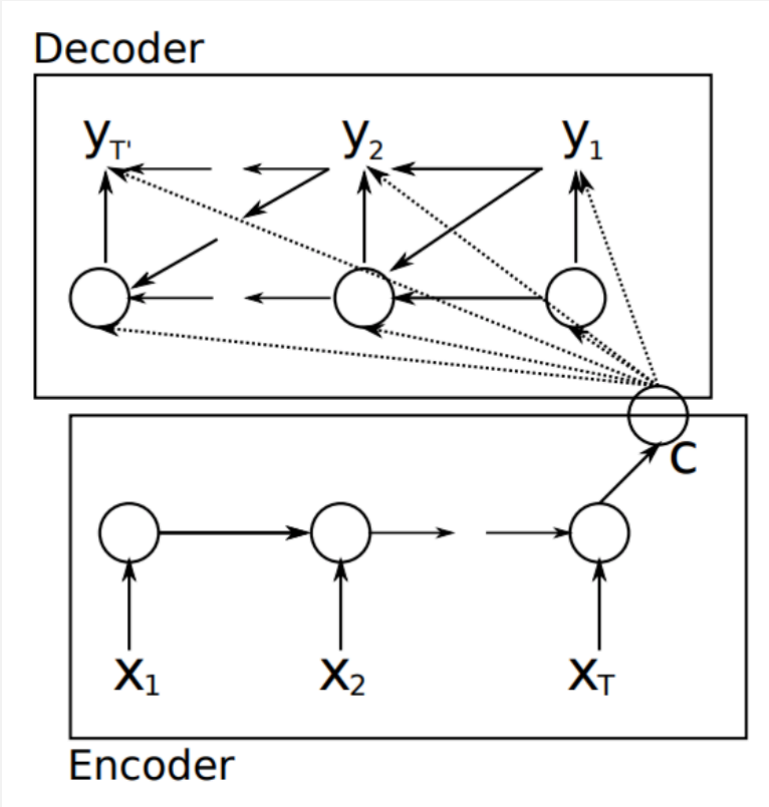
Introduction

- Attention is All You Need
  - ICLR 2017
  - Google Brain & Research
  - Transformer를 제안
  - End2End Encoder-Decoder Model
  - Neural Machine Translation 문제를 풀려고 함

Neural Machine Translation

- Neural Machine Translation
  - 대표적인 NLP Task
  - Source sentence를 받아서 Target sentence로 번역
    - Source : I am happy
    - Target : 나는 행복하다
  - Encoder-Decoder Model로 많이 접근
  - Seq2Seq, Attention 등이 유명

Seq2Seq



* f, g 모두 non-linear function
f는 RNN, g는 전체 모델을 의미

- Encoder (일반 RNN)

$$\mathbf{h}_{\langle t \rangle} = f\left(\mathbf{h}_{\langle t-1 \rangle}, x_t\right),$$

- Decoder (context 추가)

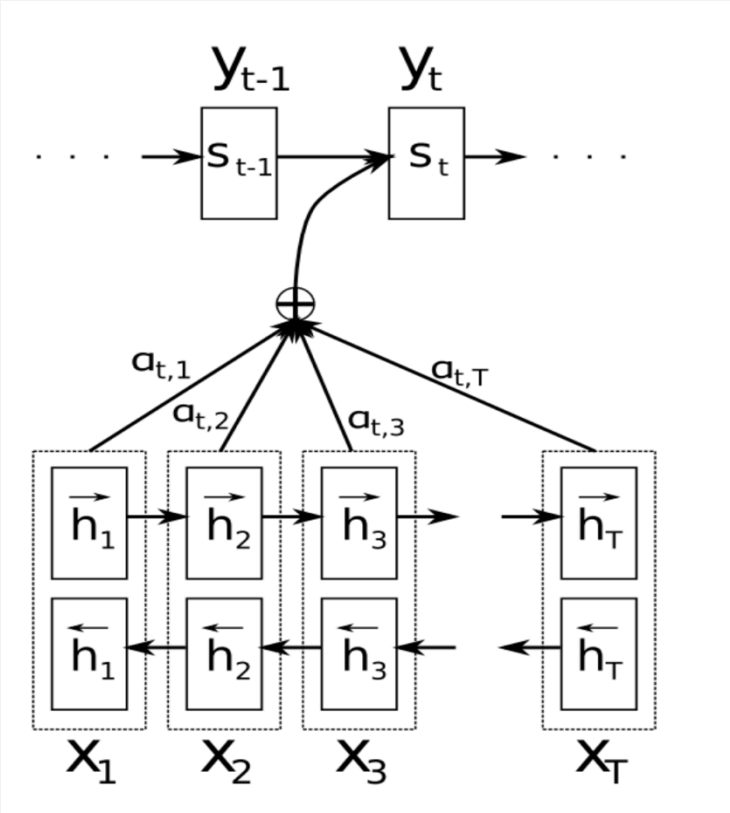
$$\mathbf{h}_{\langle t \rangle} = f\left(\mathbf{h}_{\langle t-1 \rangle}, y_{t-1}, \mathbf{c}\right),$$

- Output

$$P(y_t | y_{t-1}, y_{t-2}, \ldots, y_1, \mathbf{c}) = g\left(\mathbf{h}_{\langle t \rangle}, y_{t-1}, \mathbf{c}\right).$$

- Encoder Input으로 Source Sentence,
  Decoder Output으로 Target Sentence, Input은 shifted right 한 거

Attention based Seq2Seq



$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$

$$p(y_i|y_1, \ldots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i),$$

- Decoder (context 추가)

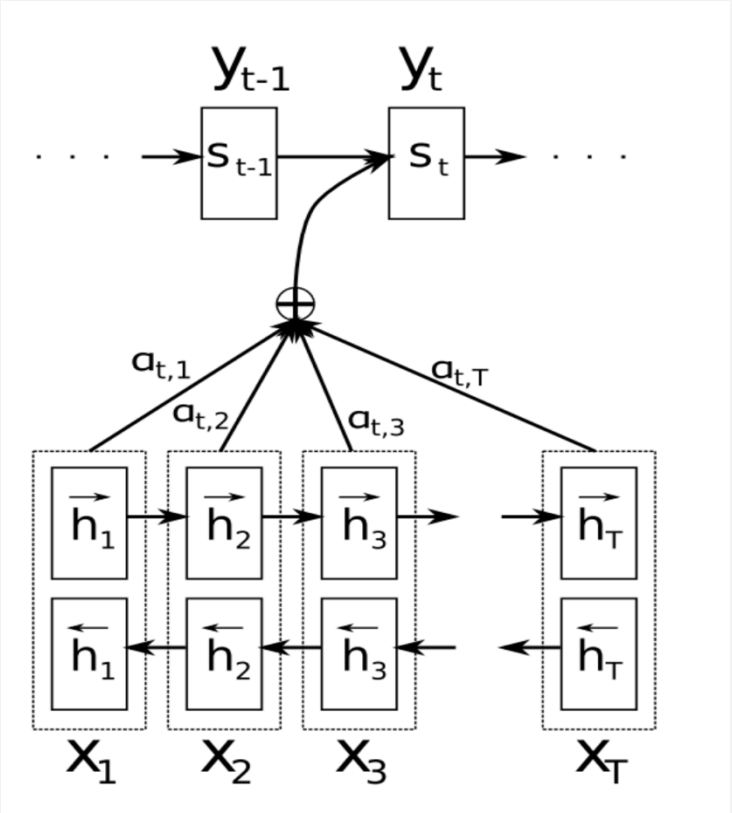
$$\mathbf{h}_{\langle t \rangle} = f\left(\mathbf{h}_{\langle t-1 \rangle}, y_{t-1}, \mathbf{c}\right),$$

- Output

$$P(y_t|y_{t-1}, y_{t-2}, \ldots, y_1, \mathbf{c}) = g\left(\mathbf{h}_{\langle t \rangle}, y_{t-1}, \mathbf{c}\right).$$

- Seq2Seq 수식과 (거의) 똑같다!
- 최종 Encoder output c 대신 Input 전체를 의미하는 x로 바뀜
- 하나의 c 대신 각 Decoder(i^{th})에 집중(attention)된 c_i로 바뀜

Attention based Seq2Seq



$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$

$$p(y_i|y_1, \ldots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i),$$

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j.$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

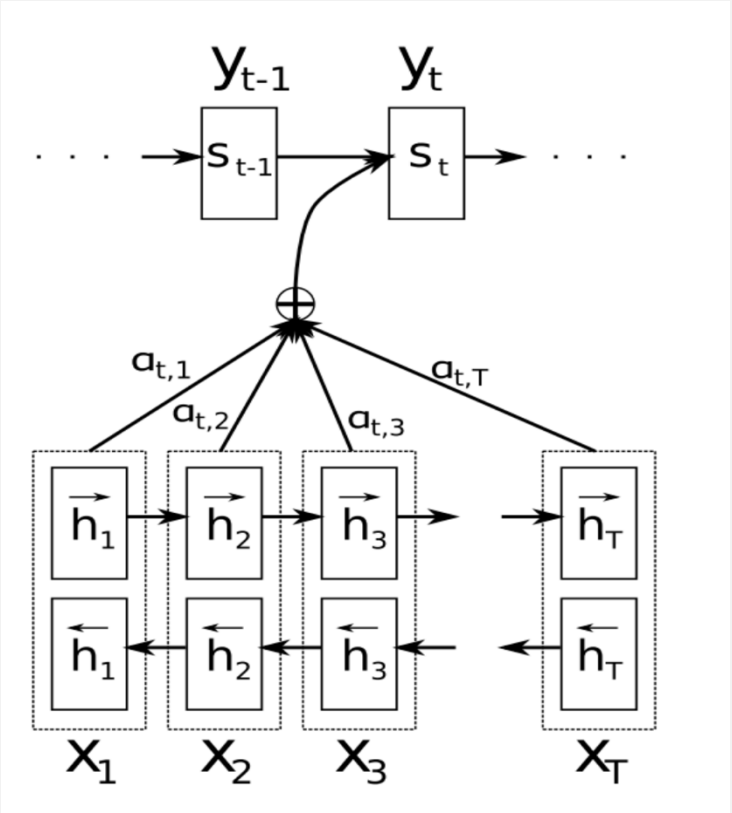
$$e_{ij} = a(s_{i-1}, h_j)$$

- c_i 를 구해야 한다!
- 왜? s_i 와 $p(y_i|y_{1;i-1}, x)$ 를 구하기 위해서!

현재(i) output

Decoder의 현재(i) state

Attention based Seq2Seq



$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$

$$p(y_i|y_1, \ldots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i),$$

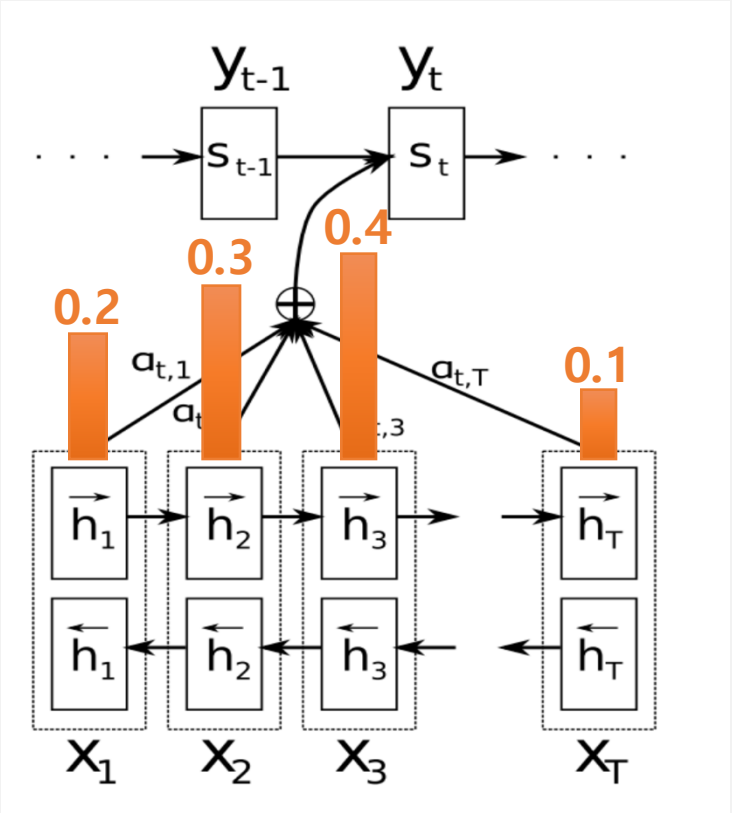
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j.$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

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

- Decoder의 직전(i-1)번째 state와 Encoder의 j번째 state의 유사도
- 유사도 계산 방법은 과감히 패스! ~~(뒤에도 나옴)~~

Attention based Seq2Seq



$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$

$$p(y_i|y_1, \ldots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i),$$

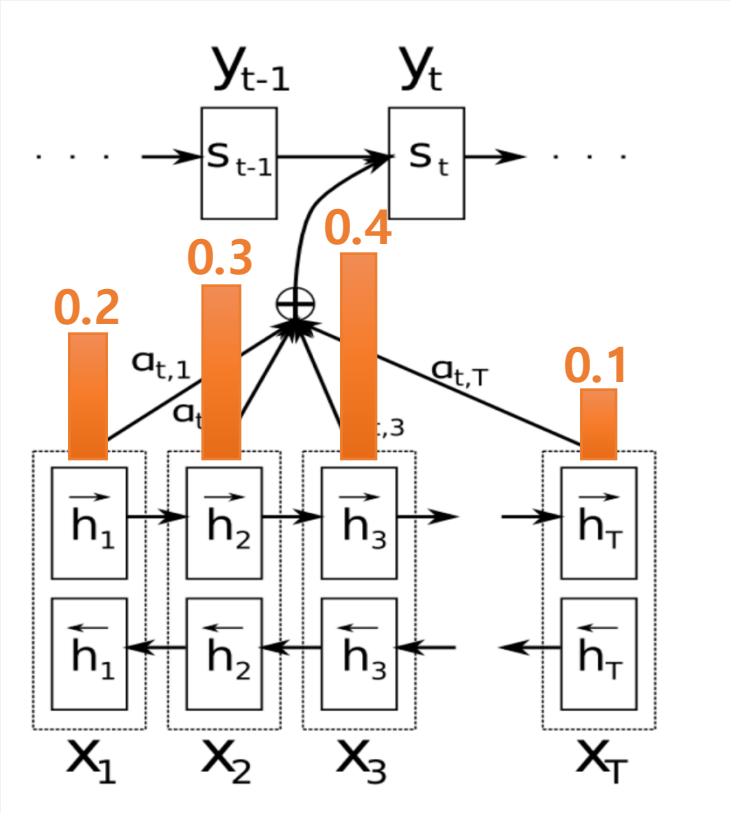
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j.$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

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

- 이 유사도를 **확률**로 표현
- i는 변하지 않고, j만 변함 (분모의 summation)
- 즉 Decoder i state에 대한 모든 **Encoder의 유사도**를 확률로 나타냄

Attention based Seq2Seq



$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$

$$p(y_i|y_1, \ldots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i),$$

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j.$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$ 유사도(확률 ver.)

$$e_{ij} = a(s_{i-1}, h_j)$$ 유사도

- c_i는 [각 Encoder state] 곱하기 [Decoder 직전 state와의 유사도]의 합
- 모든 Encoder state의 가중합 (weighted-sum)

Attention based Seq2Seq

- NMT에서 최고의 성능을 내는 모델 중 하나 (state-of-the-art)
- 하지만 seq2seq의 두가지 단점이 있음
  1. **병렬처리가 불가능 → 속도 느림**
  2. Long Term Dependency를 잡을 수 없음 (CNN도 마찬가지)
- Attention이 long term dependency 문제를 어느정도 완화시켰으나, 결국 RNN기반이기에 병렬 처리 불가, 속도 느림
- **그래서 RNN, CNN 없이 only attention만 가지고 해보자!**

  Attention is All You Need !!! ~~**(어텐션만 있으면 돼!)**~~

Transformer

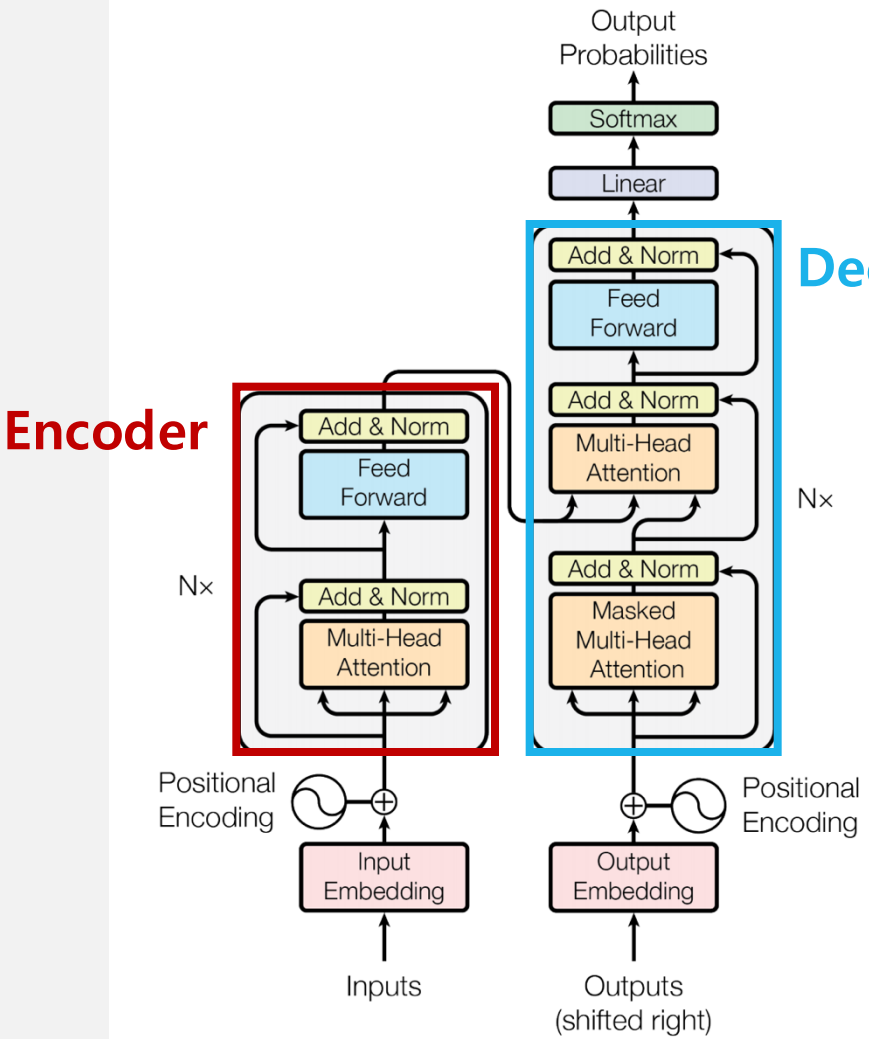


Figure 1: The Transformer - model architecture.

- Embedding
- Positional Encoding
- (Masked) Multi-Head Attention
- Scaled Dot-Product Attention
- Position-wise FF Network
- Residual Connection
- Layer Normalization

Transformer

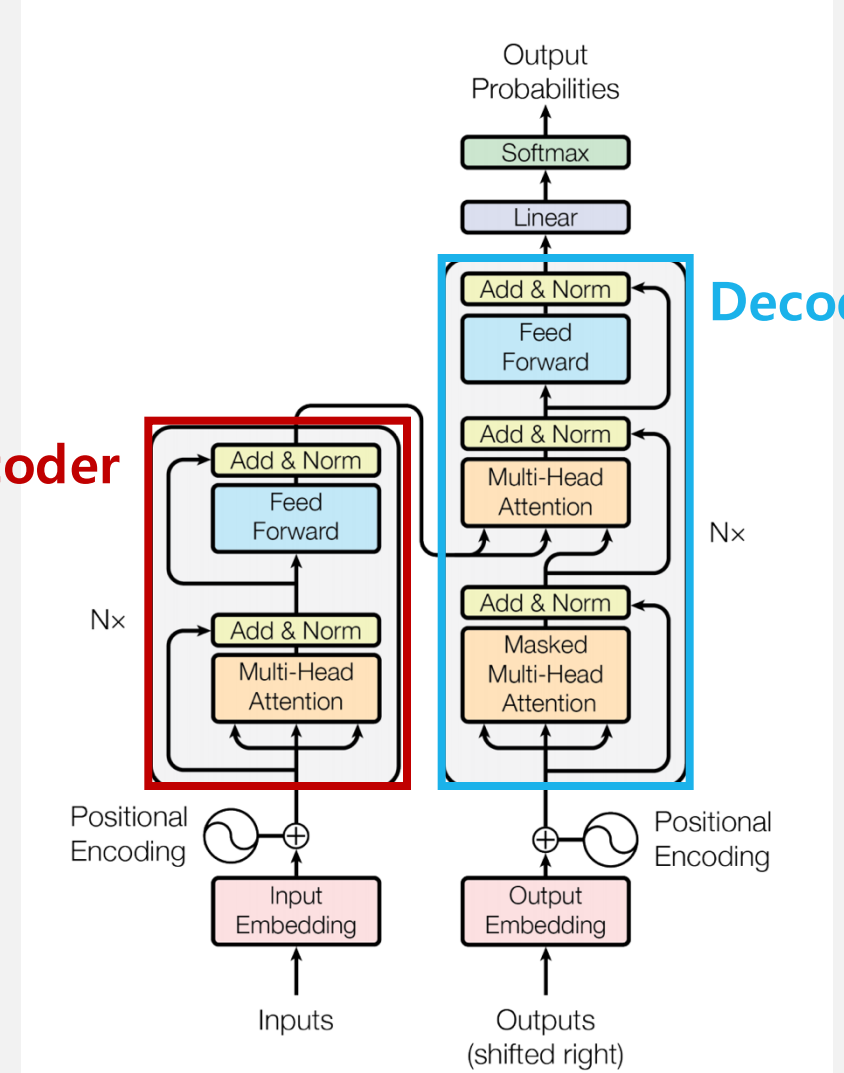


Figure 1: The Transformer - model architecture.

- Encoder Input

  = Source Sentence

- Decoder Input

  = right shifted Target Sentence

  ex) <S> I am happy <E> <P> …

- Decoder Output

  = Target Sentence

  ex) I am happy <E> <P> …

Embedding & Position Encoding

- Embedding은 그냥 흔히 쓰는 word embedding
- Position Encoding
  - CNN, RNN은 순서를 알 수 있음
  - Transformer는 순서를 알지 못함

Position Encoding

- CNN

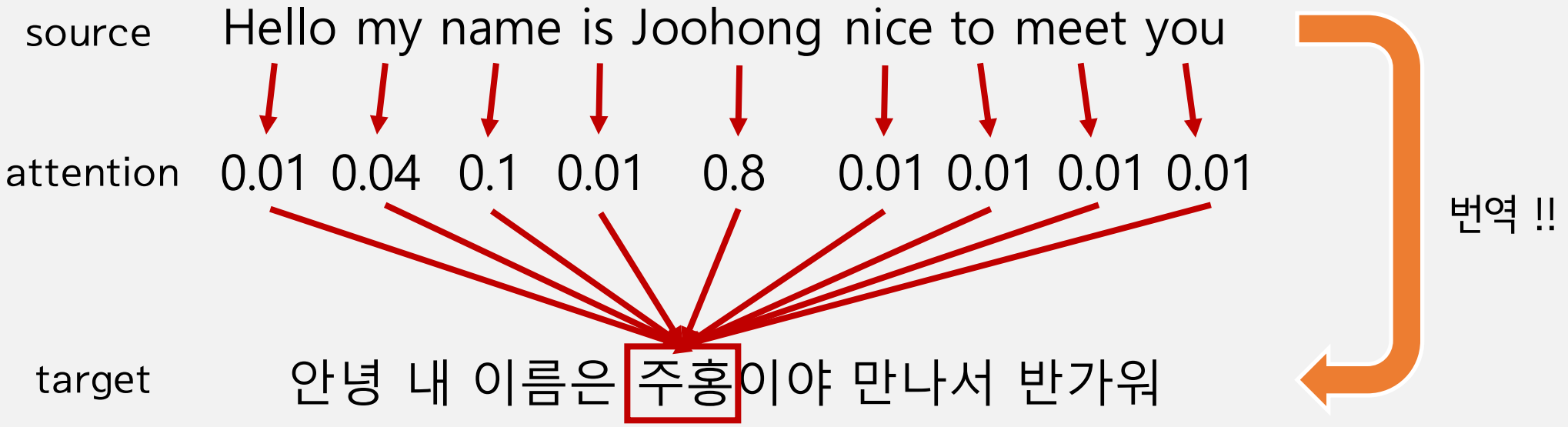
Hello my name is Joohong nice to meet you

  - N-gram과 유사함
  - (1,2), (2,3) … (n-1, n) 이렇게 묶여서 들어가기에
    근처에 나타나는 단어를 알 수 있고 이는 순서의 의미를 담음

- RNN
  - 생략 ㅎㅎ

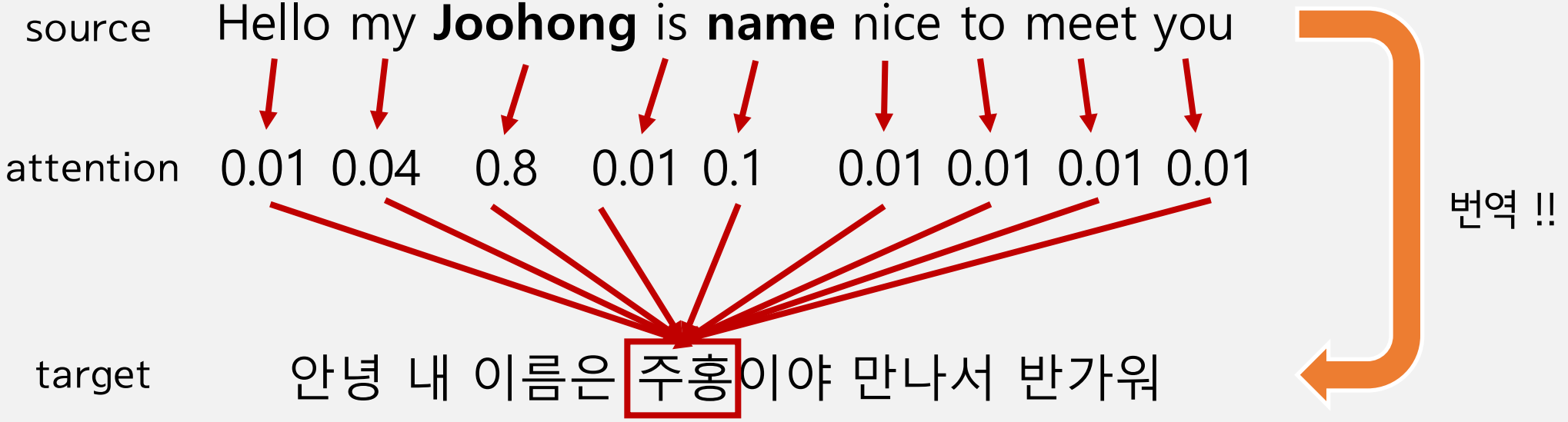
Position Encoding

- Transformer



Position Encoding

- Transformer

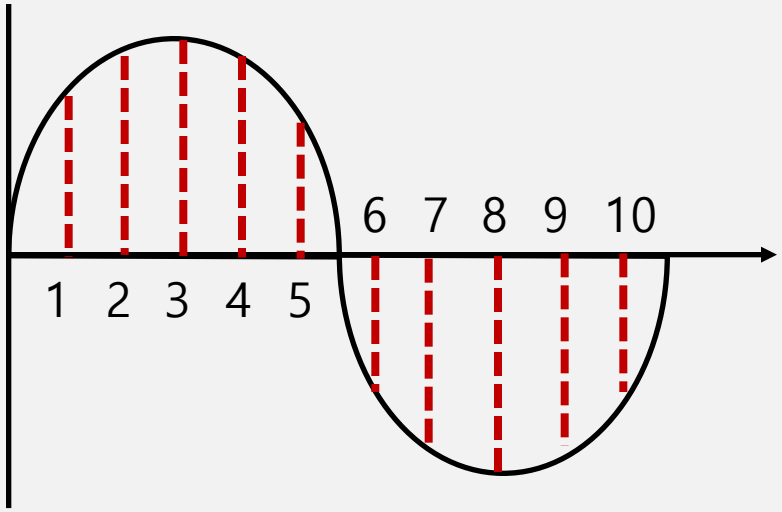


Position Encoding

- 그래서 Position을 Encoding 시켜준다!

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

Word Embedding
+
Position Encoding
=
Final Input



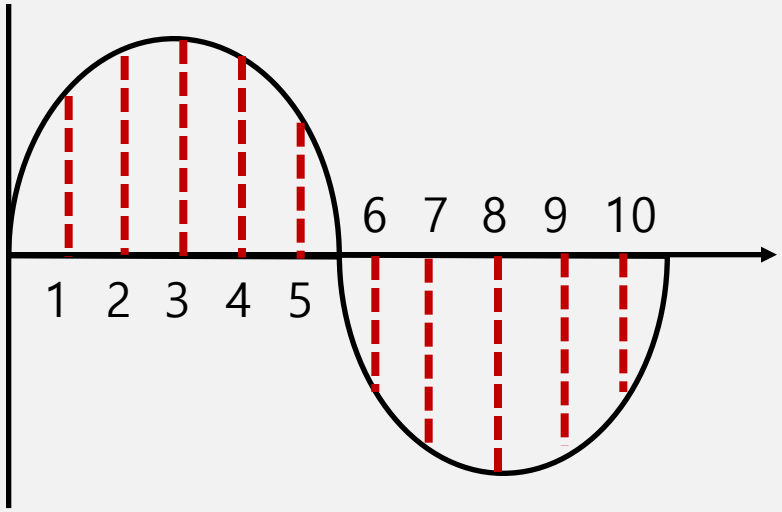
- 만약 $\sin x$가 1이면 ? → 3번째
- 만약 $\sin x$가 -1이면 ? → 8번째
- 만약 $\sin x$가 0.5이면 ? → 1? 4?

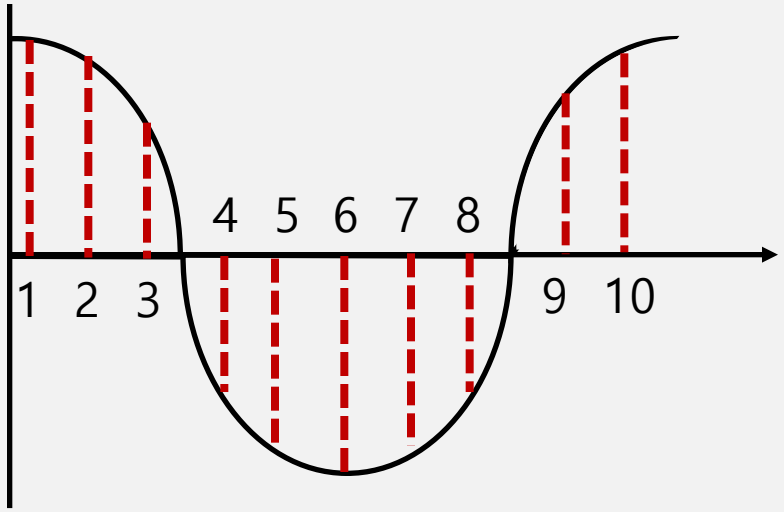
Position Encoding

- 그래서 Position을 Encoding 시켜준다!

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

Word Embedding
+
Position Encoding
=
Final Input



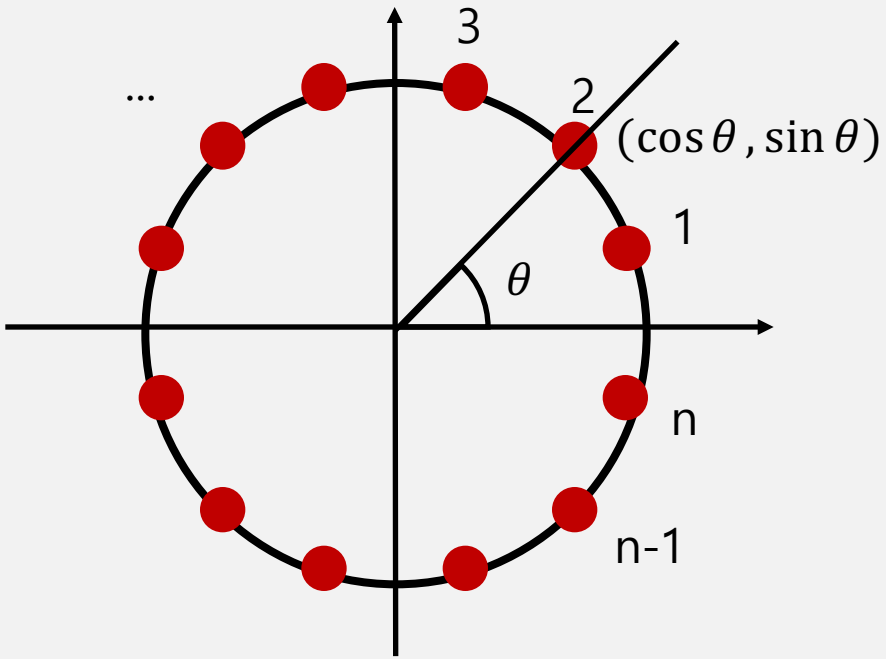


Position Encoding

- 그래서 Position을 Encoding 시켜준다!

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

Word Embedding
+
Position Encoding
=
Final Input



sinusoidal 방법을 통해
Position 정보를
Encoding 시켜줄 수 있다!!

Multi-Head Attention

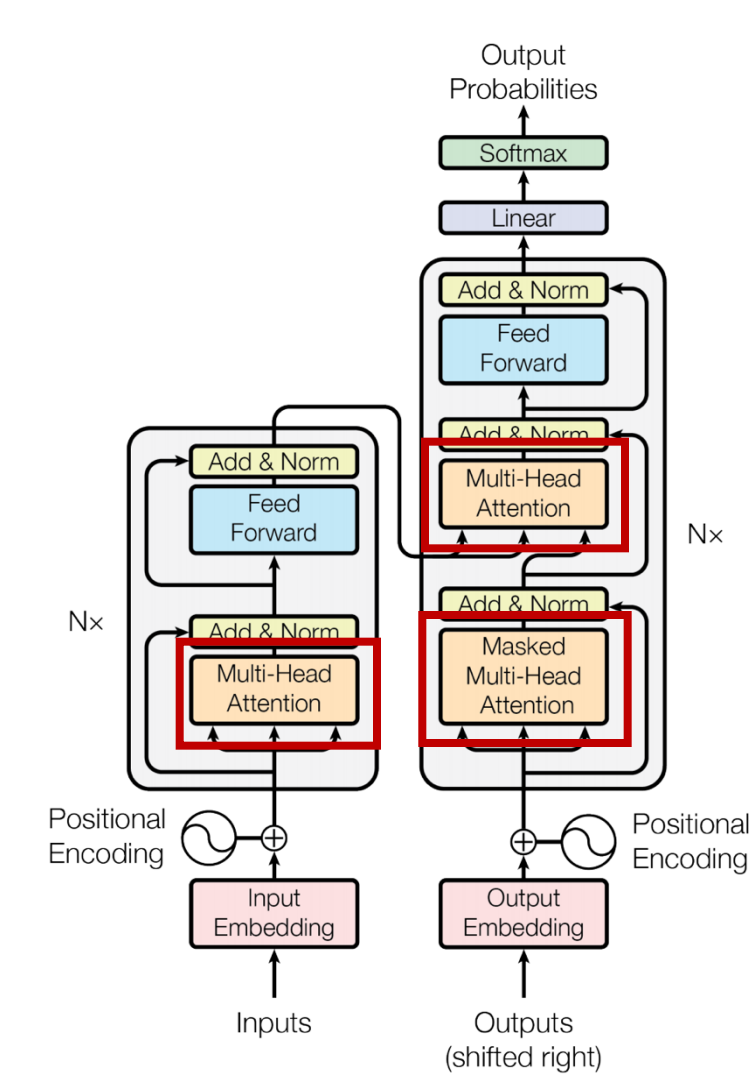
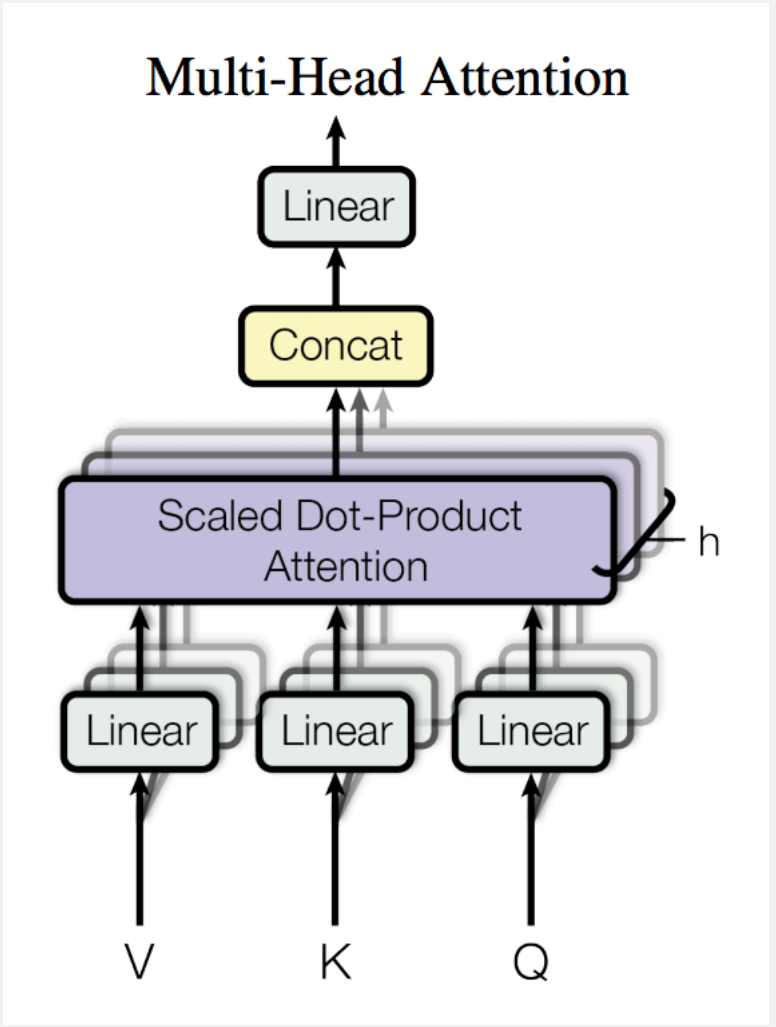
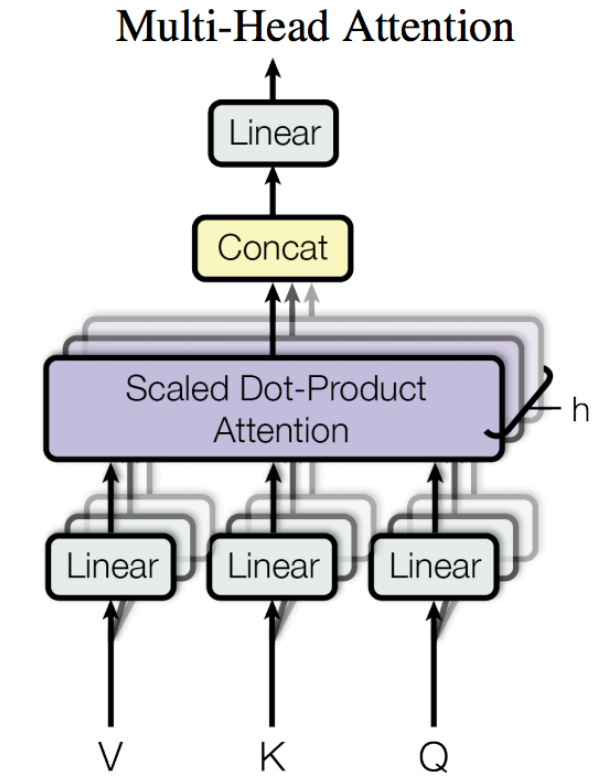


Figure 1: The Transformer - model architecture.



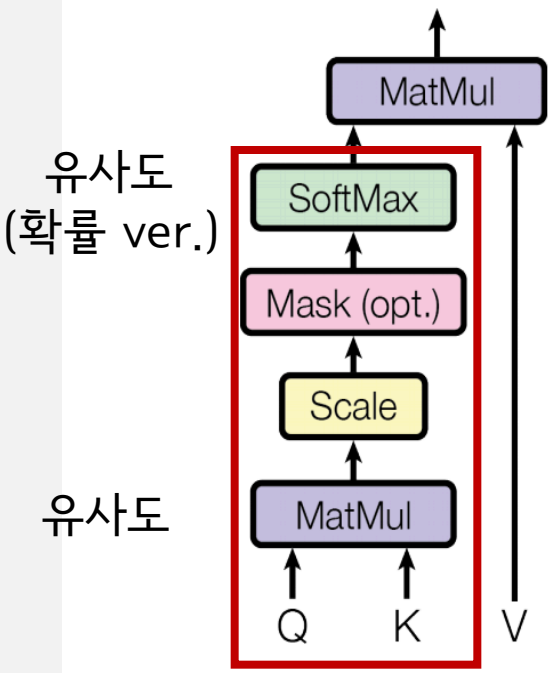
Multi-Head Attention



- Query, Key, Value에 대해서 Scaled Dot-Product Attention를 여러번(Multi) 시행한 뒤, 이어 붙여서(concat) 내보냄

- Query, Key, Value와 Scaled Dot-Product Attention을 알아보자!

Scaled Dot-Product Attention



- Query, (Key, Value) Attention

  → Query와 Key의 유사도만큼

  Attention을 줘서 Value를 가중합

- 앞의 seq2seq로 치자면,
  - Query = Decoder
  - Key = Encoder
  - Value = Encoder

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

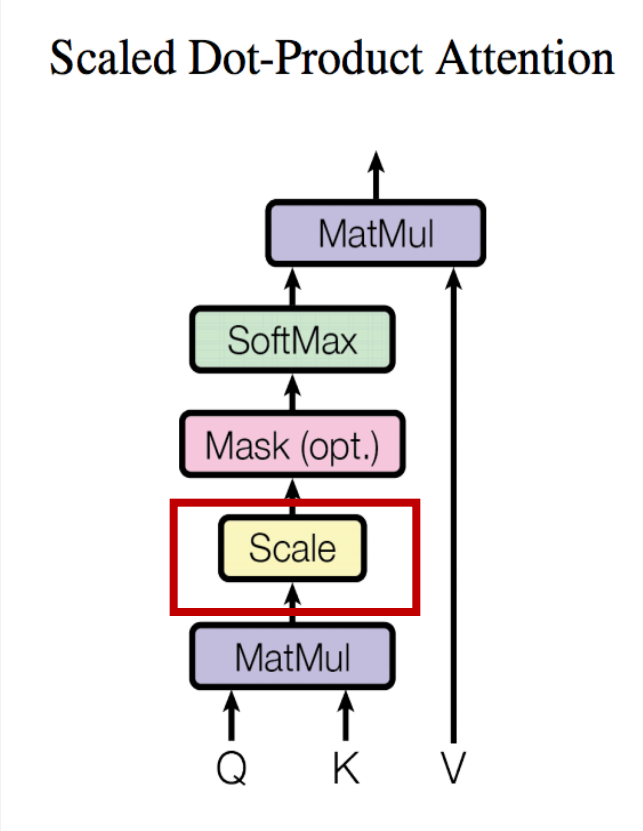
유사도
(확률 ver.)

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

유사도

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

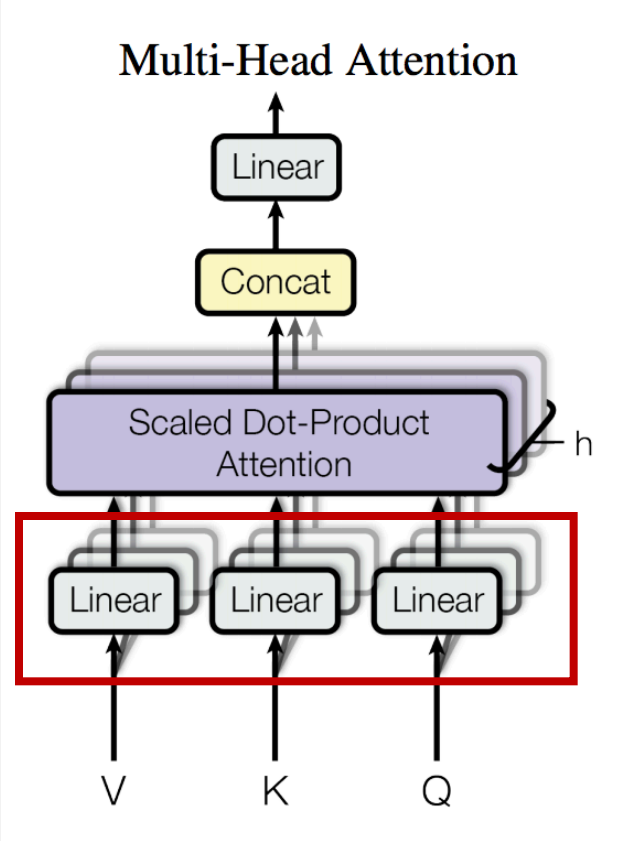
Scaled Dot-Product Attention



- Scaling을 하는 이유

- Q, K가 Dot-Product을 하게 되는데,
  이러면 값이 너무 커짐
- Q, K $\sim standard\ N(0, dim)$ 이라 할 때,
  $Q \cdot K = \sum_{i=1}^{dim} q_i k_i$ 가 되고
  $Q \cdot K \sim standard\ N(0, dim)$ 이 됨
  따라서, $\sqrt{dim}$ 으로 나눠주어 scaling 해줌

Multi-Head Attention



- Query, Key, Value에 대해서 Scaled Dot-Product Attention를 여러번(Multi) 시행한 뒤, 이어 붙여서(concat) 내보냄
- Linear는 일반 FC (projection) layer
  - 정보 압축
  - 차원 축소 → 속도 향상

Multi-Head Attention

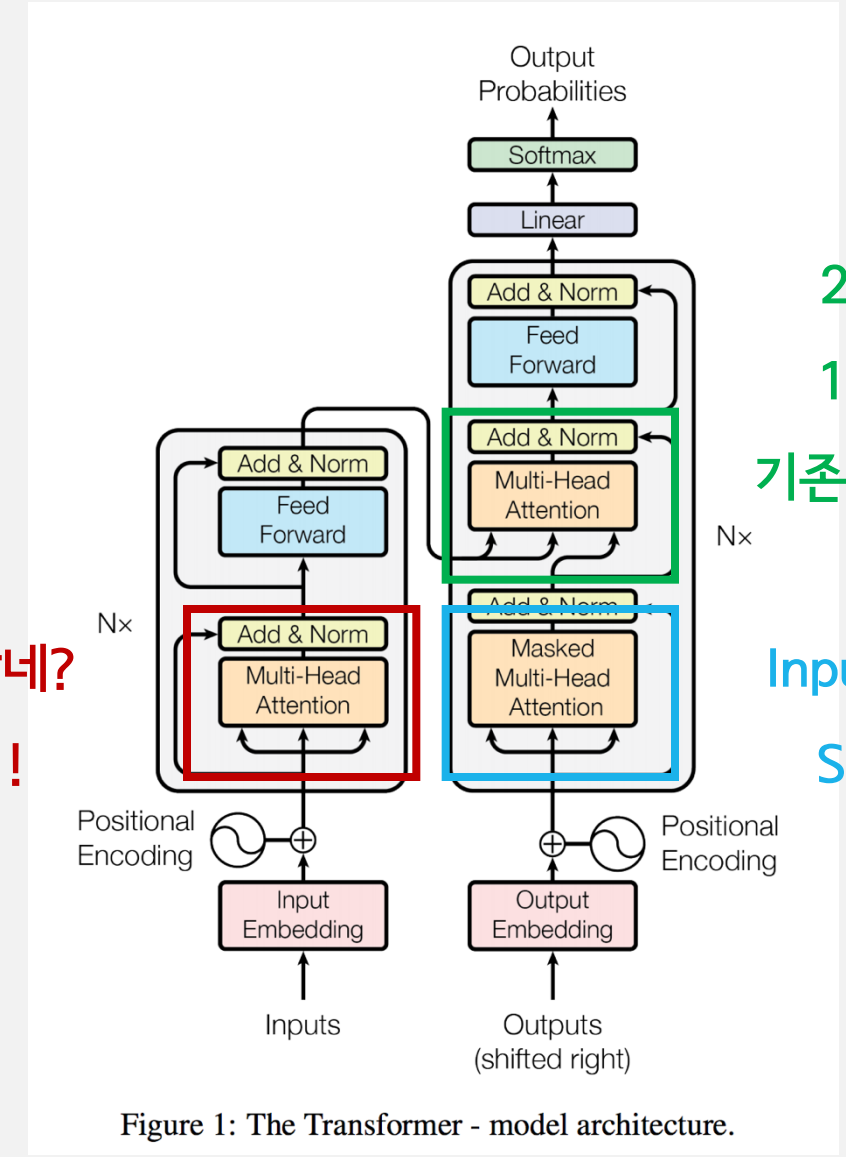


Figure 1: The Transformer - model architecture.

2개는 Encoder에서

1개는 Decoder에서

기존 seq2seq Attention!

Input 3개가 다 같네?

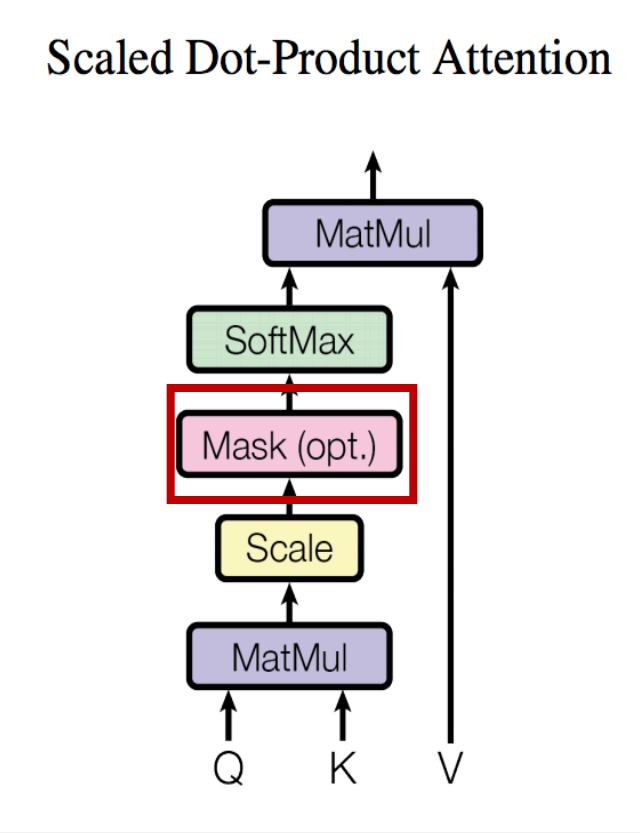
Self Attention !

Input 3개가 다 같네?

Self Attention !

그럼 Masked는 뭐지...?

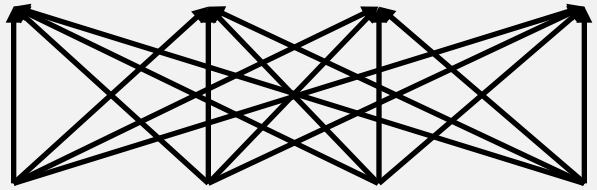
Masked Multi-Head Attention



- 일단 Masked가 적용된 부분은 **Decoder의 Self Attention 뿐임**

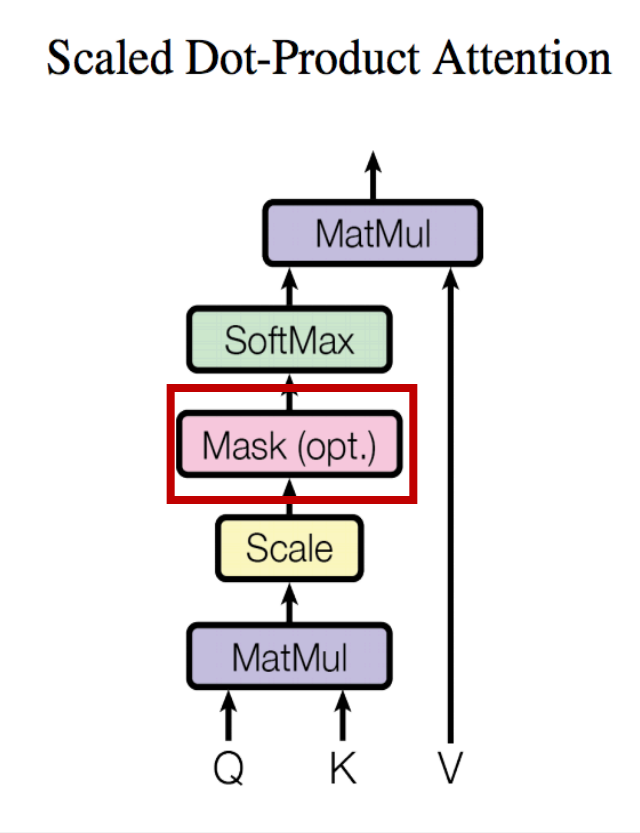
- 일반 Self Attention

My name is Joohong



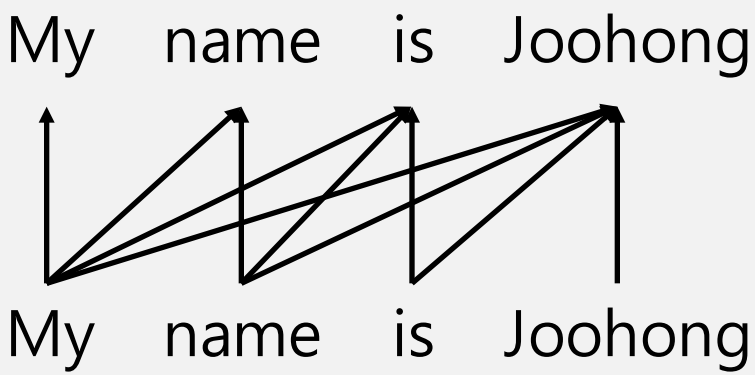
My name is Joohong

Masked Multi-Head Attention



- 일단 Masked가 적용된 부분은
  Decoder의 Self Attention 뿐임

- Masked Self Attention



- Auto regressive 하게
  앞의 단어들에 대해서만 Attention 하겠다!

Masked Multi-Head Attention

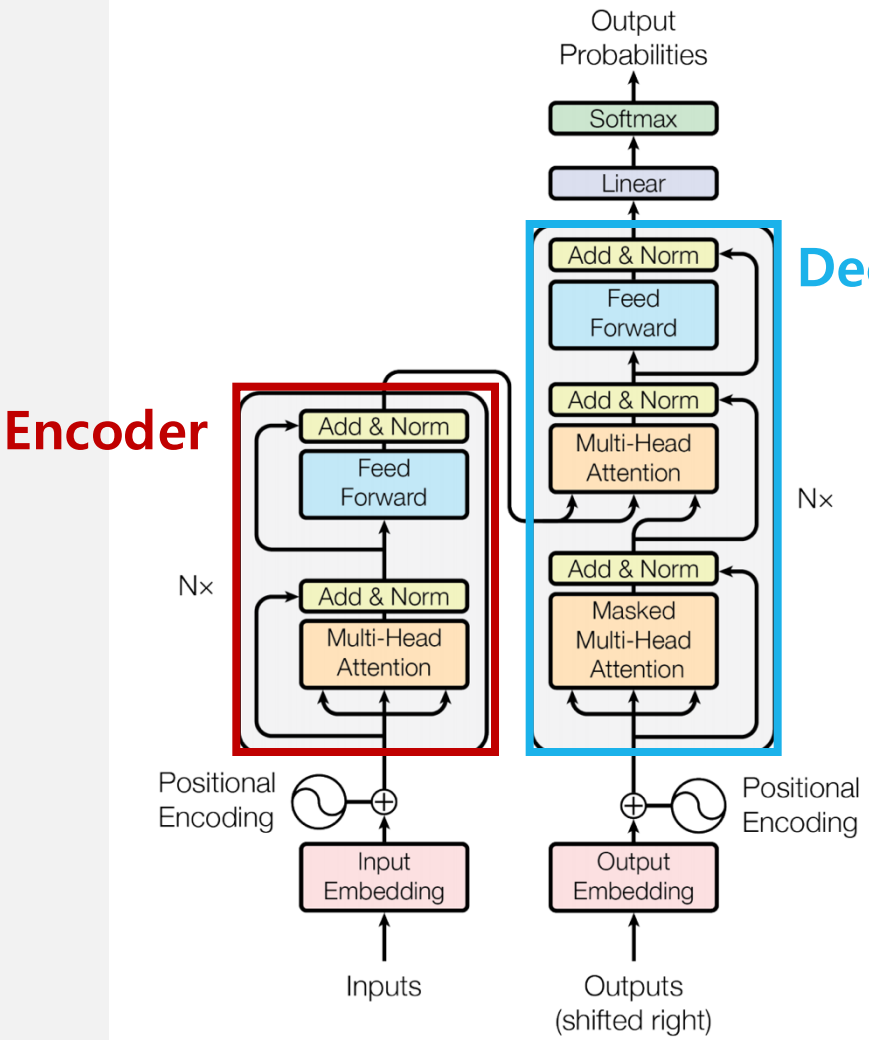
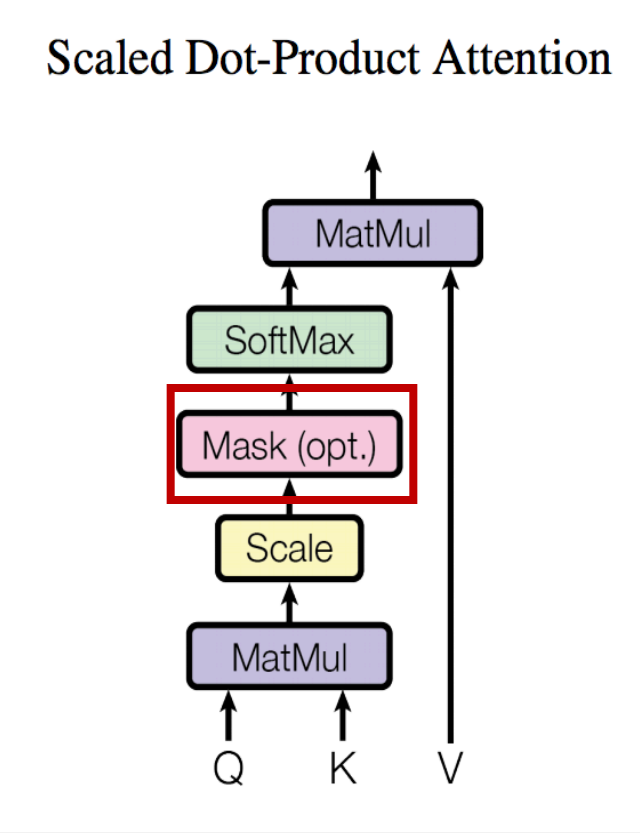


Figure 1: The Transformer - model architecture.

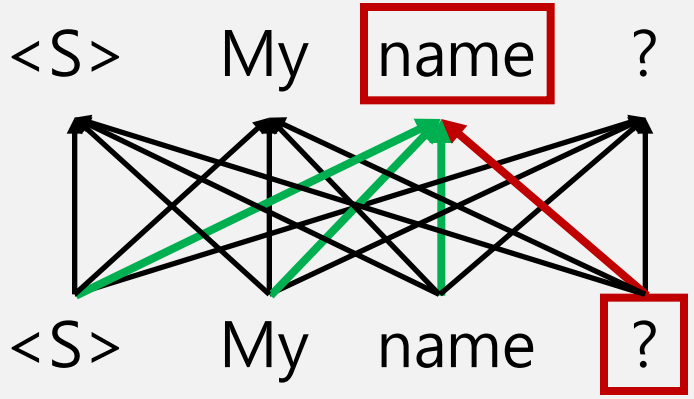
- 학습 후, Inference 단계에서 Decoder의 In/Out은 무엇일까?
  → <S> ? ? ? ? ? ? ...
- 위의 sequence를 입력, 예측을 반복하며, 한 글자씩 알아가는 것
- seq = [<S>, ?, ?, ?, ..., ?]
  Loop(i):
  result = predict(seq)
  seq[i+1] = result[i]

Masked Multi-Head Attention

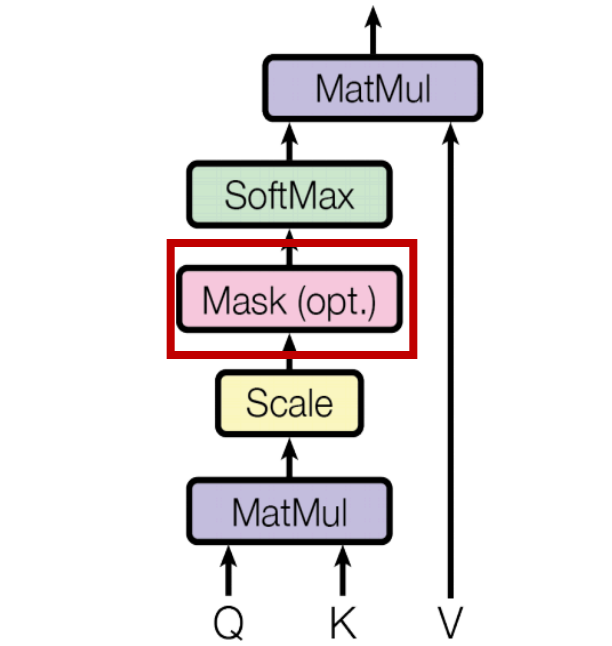


- 일단 Masked가 적용된 부분은 **Decoder의 Self Attention 뿐임**

- 일반 Self Attention

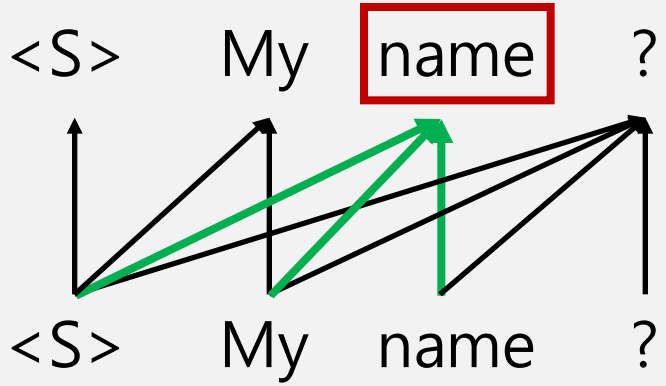


Masked Multi-Head Attention



- 일단 Masked가 적용된 부분은 **Decoder의 Self Attention 뿐임**

- Masked Self Attention



Transformer - 중간점검

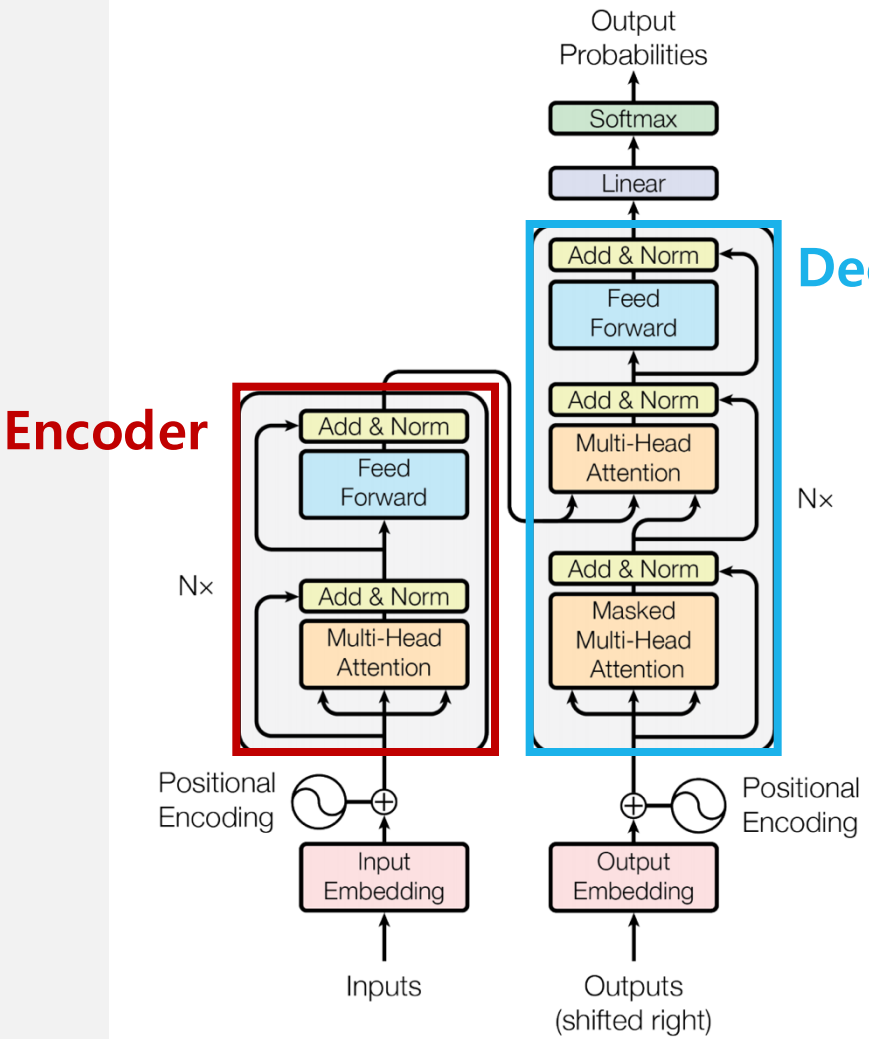
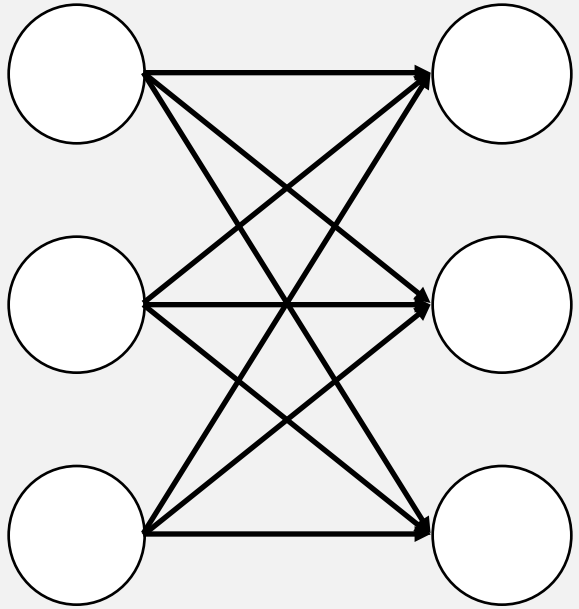


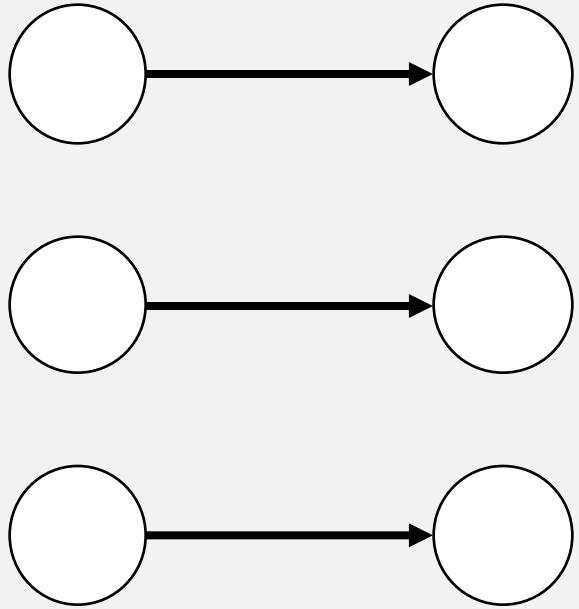
Figure 1: The Transformer - model architecture.

- ~~Embedding~~
- ~~Positional Encoding~~
- ~~(Masked) Multi-Head Attention~~
- ~~Scaled Dot-Product Attention~~
- Position-wise FF Network
- Residual Connection
- Layer Normalization

Position-wise Feed Forward Network



Feed Forward Network



Position-wise
Feed Forward Network

- 실제 구현할 때도, 1-D Convolution으로 함
- 구체적으로 어떤 효과와 의미가 있는지는 자세히 모르겠음

Residual(Skip) Connection

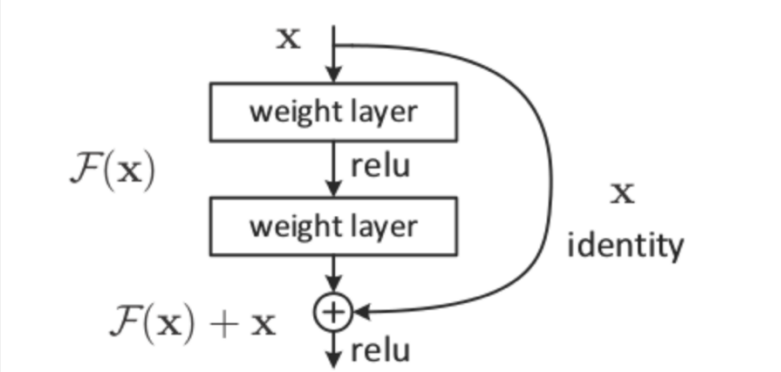
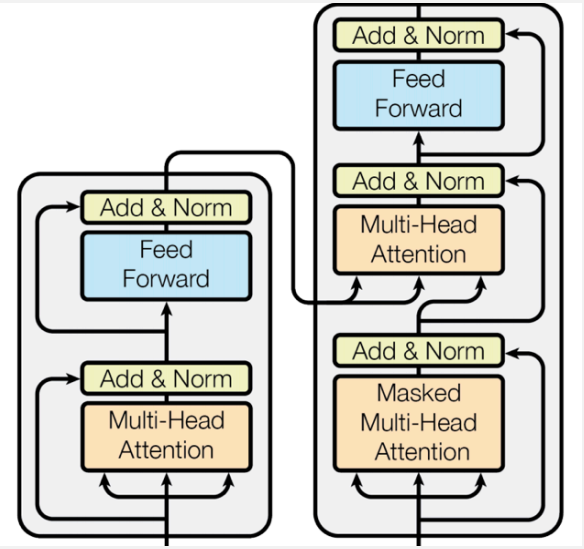


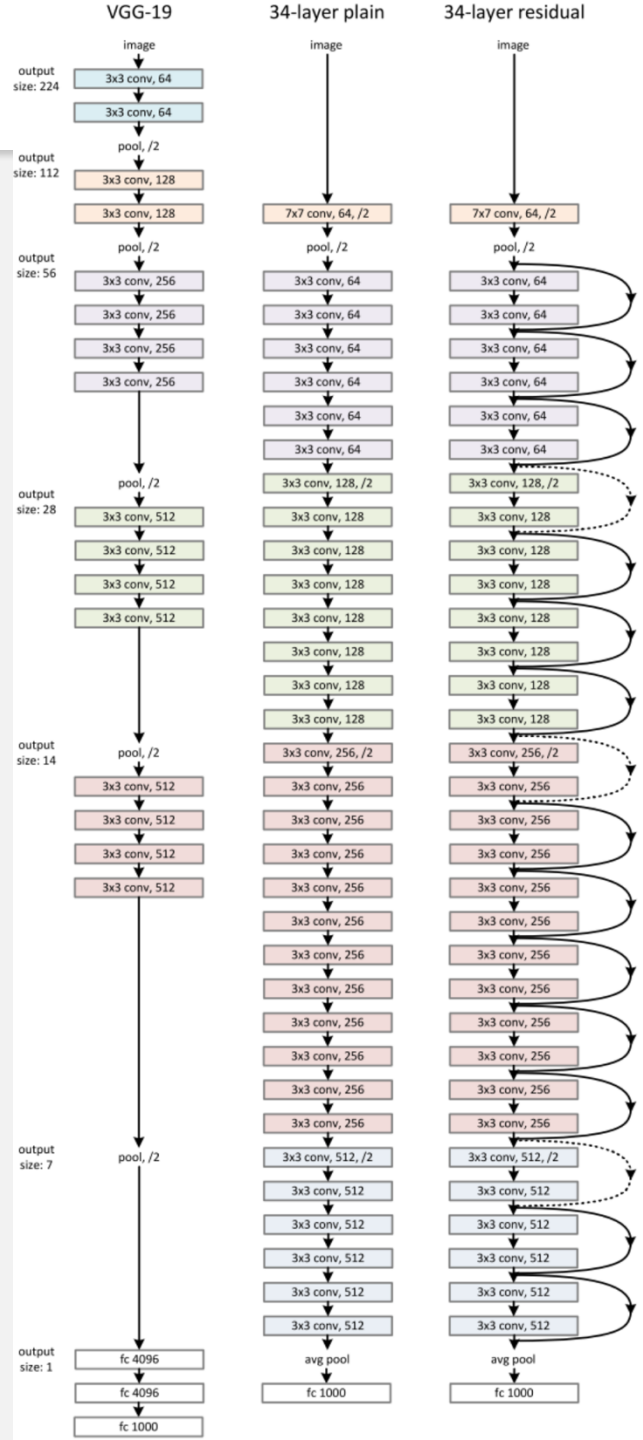
Figure 2. Residual learning: a building block.



- 이미지 분류로 유명한 ResNet에 적용된 기법
- 원래 x와 layer를 거친 F(x)를 더함
- Gradient가 (역)전파될 수 있는 길이 많음

  → Layer가 멀어도 금방 skip해서 감, LSTM의 철학
- Gradient Vanishing 문제 해결

  → Layer를 더 깊게 쌓을 수 있음



Layer Normalization

- Hinton 연구실에서 제안
- arXiv preprint (https://arxiv.org/abs/1607.06450)

- Batch normalization과 유사하게 layer의 output에 대해 normalization을 취해줘서 분포를 맞추는 느낌

- 잘은 모르니 궁금하시면 논문 읽고 리뷰 ㄱㄱ

Transformer - 중간점검

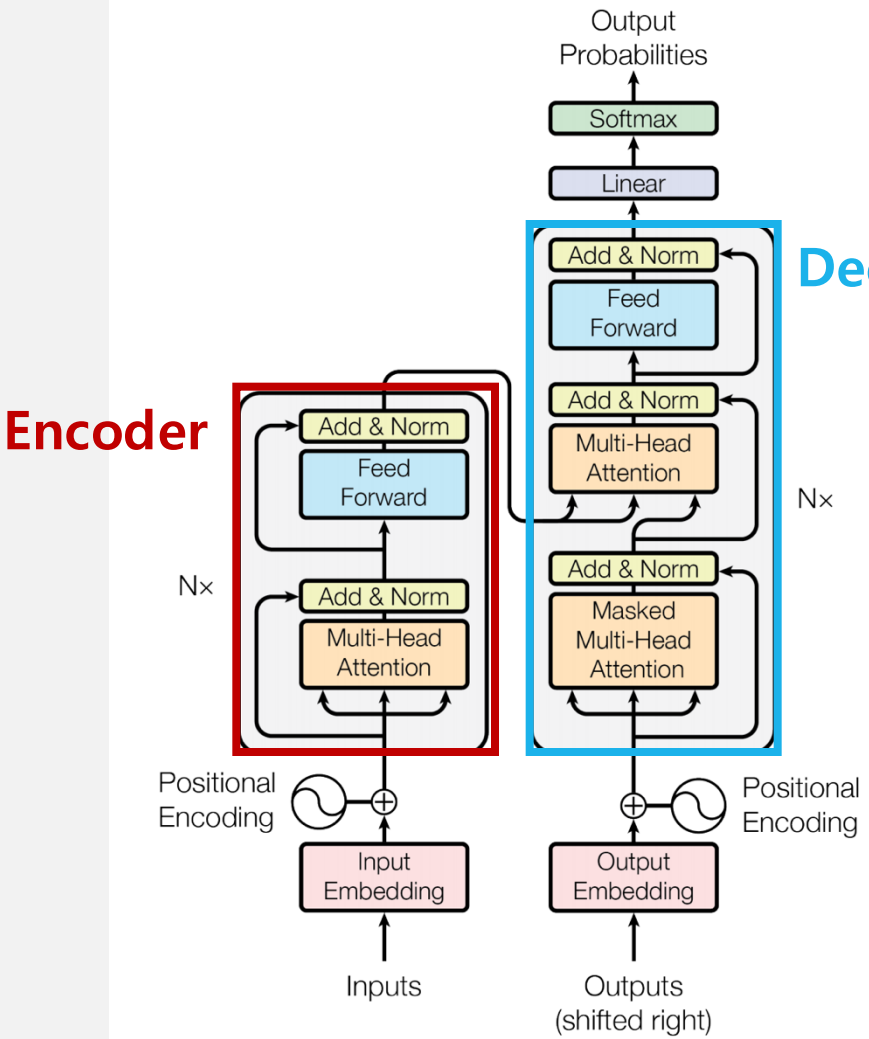


Figure 1: The Transformer - model architecture.

- ~~Embedding~~
- ~~Positional Encoding~~
- ~~(Masked) Multi-Head Attention~~
- ~~Scaled Dot-Product Attention~~
- ~~Position-wise FF Network~~
- ~~Residual Connection~~
- ~~Layer Normalization~~

- CLEAR !!

Discussion : Distant Dependency (Phrase)

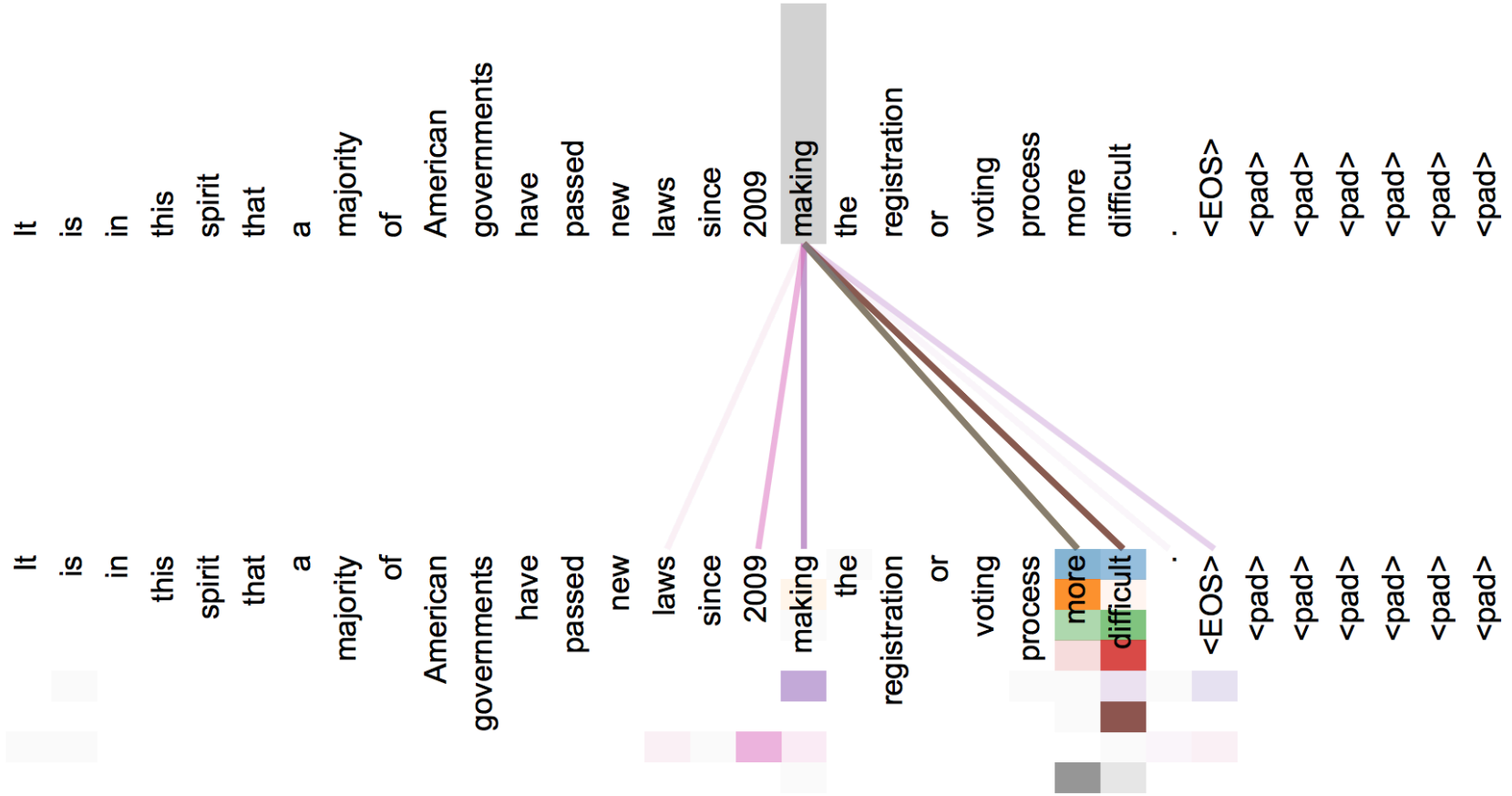


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word 'making'. Different colors represent different heads. Best viewed in color.

Discussion : Anaphora Resolution

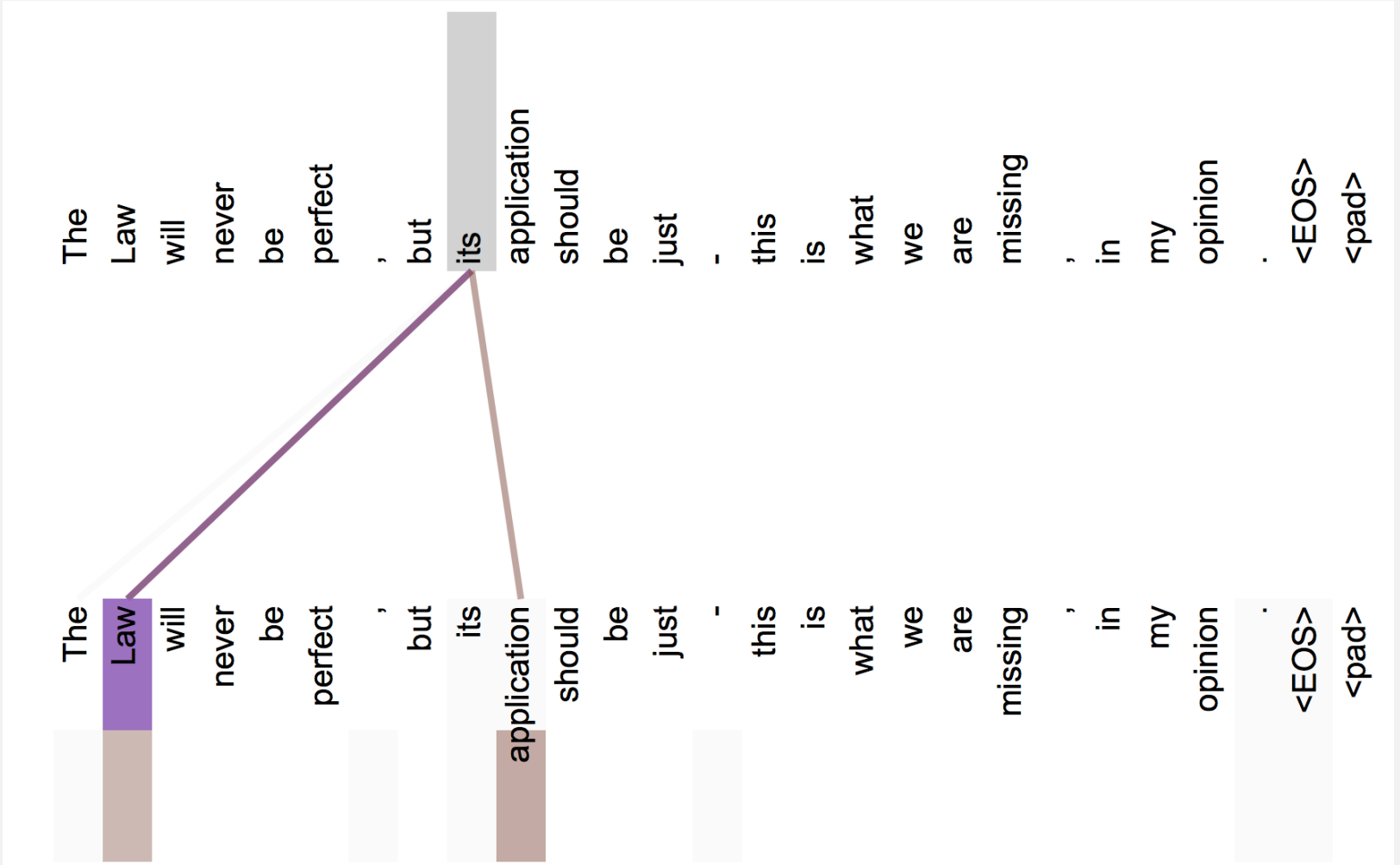


Figure 4: Two attention heads, also in layer 5 of 6, apparently involved in anaphora resolution. Top: Full attentions for head 5. Bottom: Isolated attentions from just the word 'its' for attention heads 5 and 6. Note that the attentions are very sharp for this word.

Discussion : Multi-Head

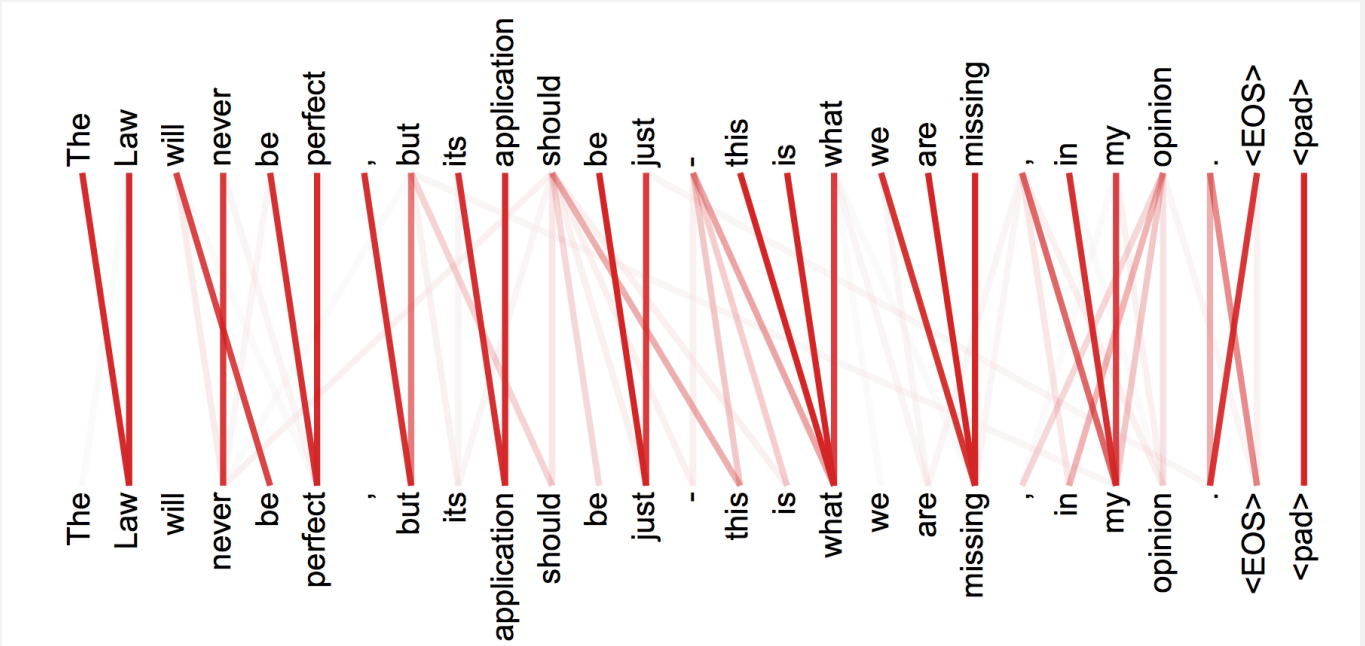


Figure 5: Many of the attention heads exhibit behaviour that seems related to the structure of the sentence. We give two such examples above, from two different heads from the encoder self-attention at layer 5 of 6. The heads clearly learned to perform different tasks.

Conclusion

- Contribution
  - Attention만을 사용한 Transformer 제안
  - 기존 모델들의 병렬 처리 한계 개선
  - Long term dependency 문제 해결
  - RNN, CNN 보다 연산량이 작은 Attention을 통해 속도 개선
  - NMT에서 SOTA를 찍음
- Limitation
  - 생각보다 그리 빠르지 않음, ~~제가 잘 사용을 못한 걸 수도...~~
  - 이후 (self) attention, transformer 기반의 많은 연구가 나오고 있지만, 완벽히 RNN을 대체할 수 있을지는 잘... 그래도 매우 큰 가능성 !!

Thank You

made by. 이 주 홍

AI Lab - Deep Learning Seminar