Attention is All You Need

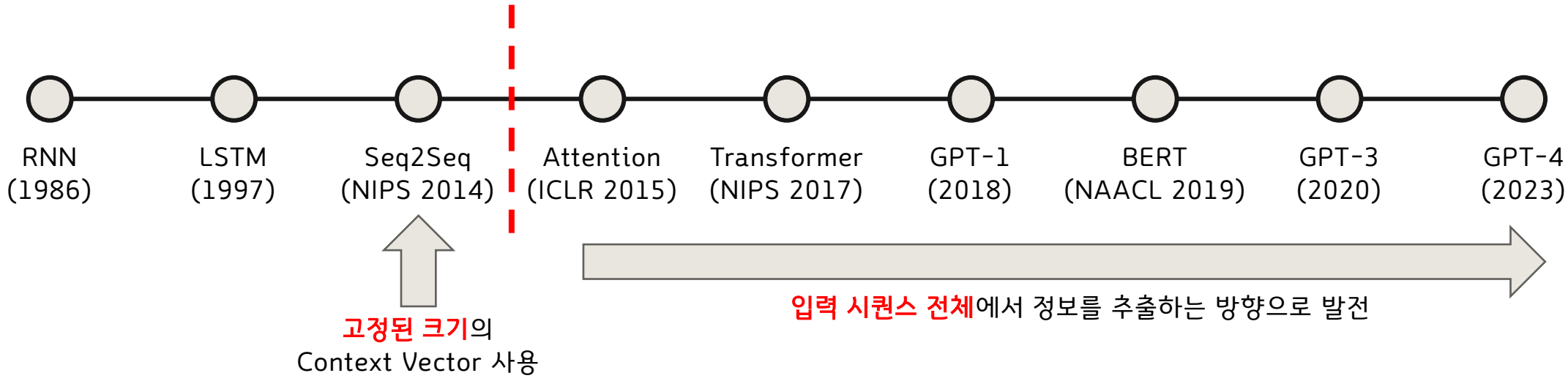
[6기-A] 임동주

NIPS 2017

목차

연구 배경 및 선행 연구

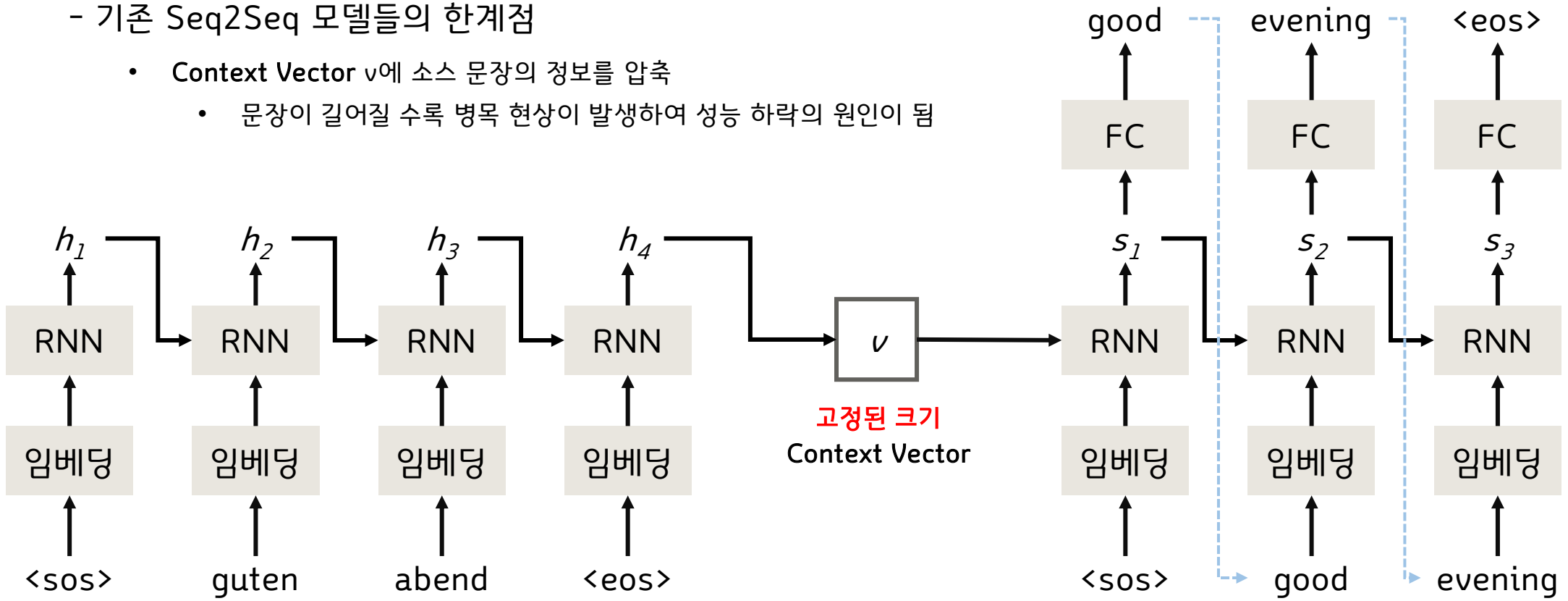
- 딥러닝 기반 기계 번역 발전 과정
  - **GPT**: Transformer의 **Decoder** 아키텍처를 활용
  - **BERT**: Transformer의 **Encoder** 아키텍처를 활용



(출처: Youtube 채널 <동빈나>, https://youtu.be/AA621UofTUA)
[딥러닝 기계 번역] Transformer: Attention Is All You Need (꼼꼼한 딥러닝 논문 리뷰와 코드 실습)

연구 배경 및 선행 연구

- 기존 Seq2Seq 모델들의 한계점
  - **Context Vector** v에 소스 문장의 정보를 압축
    - 문장이 길어질 수록 병목 현상이 발생하여 성능 하락의 원인이 됨

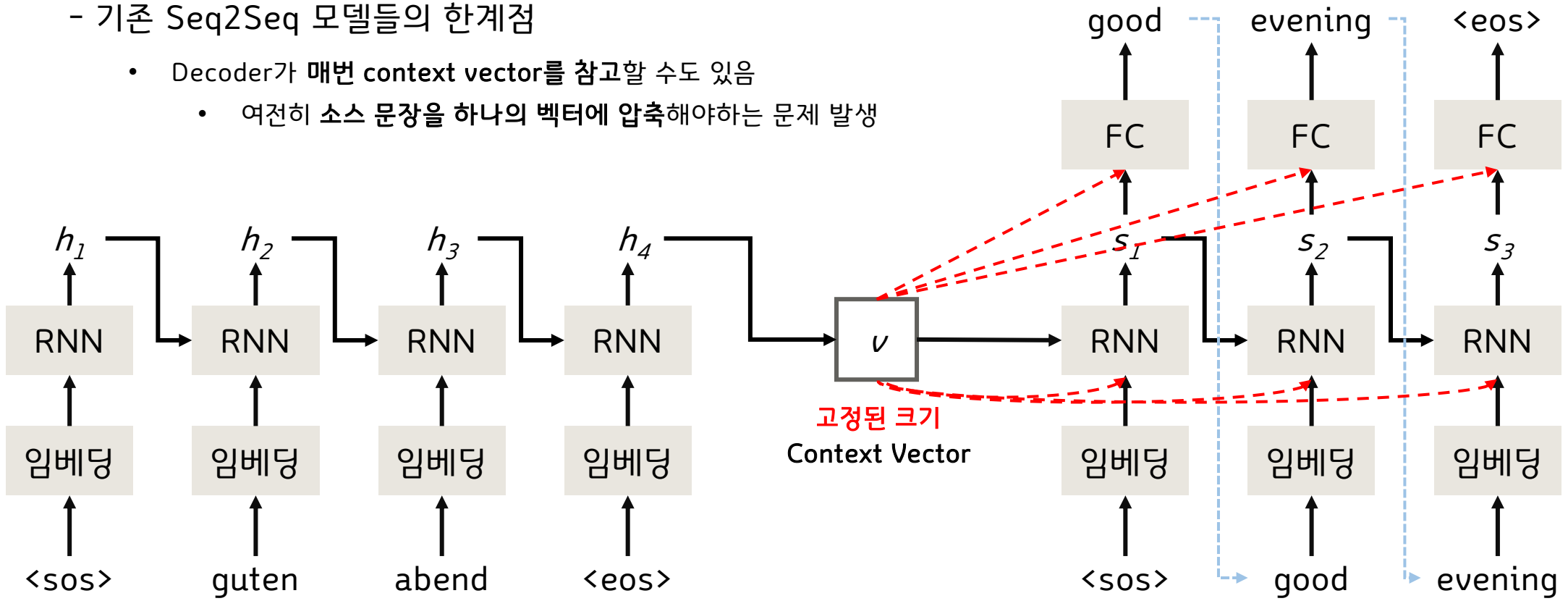


(출처: Youtube 채널 <동빈나>, https://youtu.be/AA621UofTUA)
[딥러닝 기계 번역] Transformer: Attention Is All You Need (꼼꼼한 딥러닝 논문 리뷰와 코드 실습)

Sequence to Sequence Learning with Neural Networks (NIPS 2014)

연구 배경 및 선행 연구

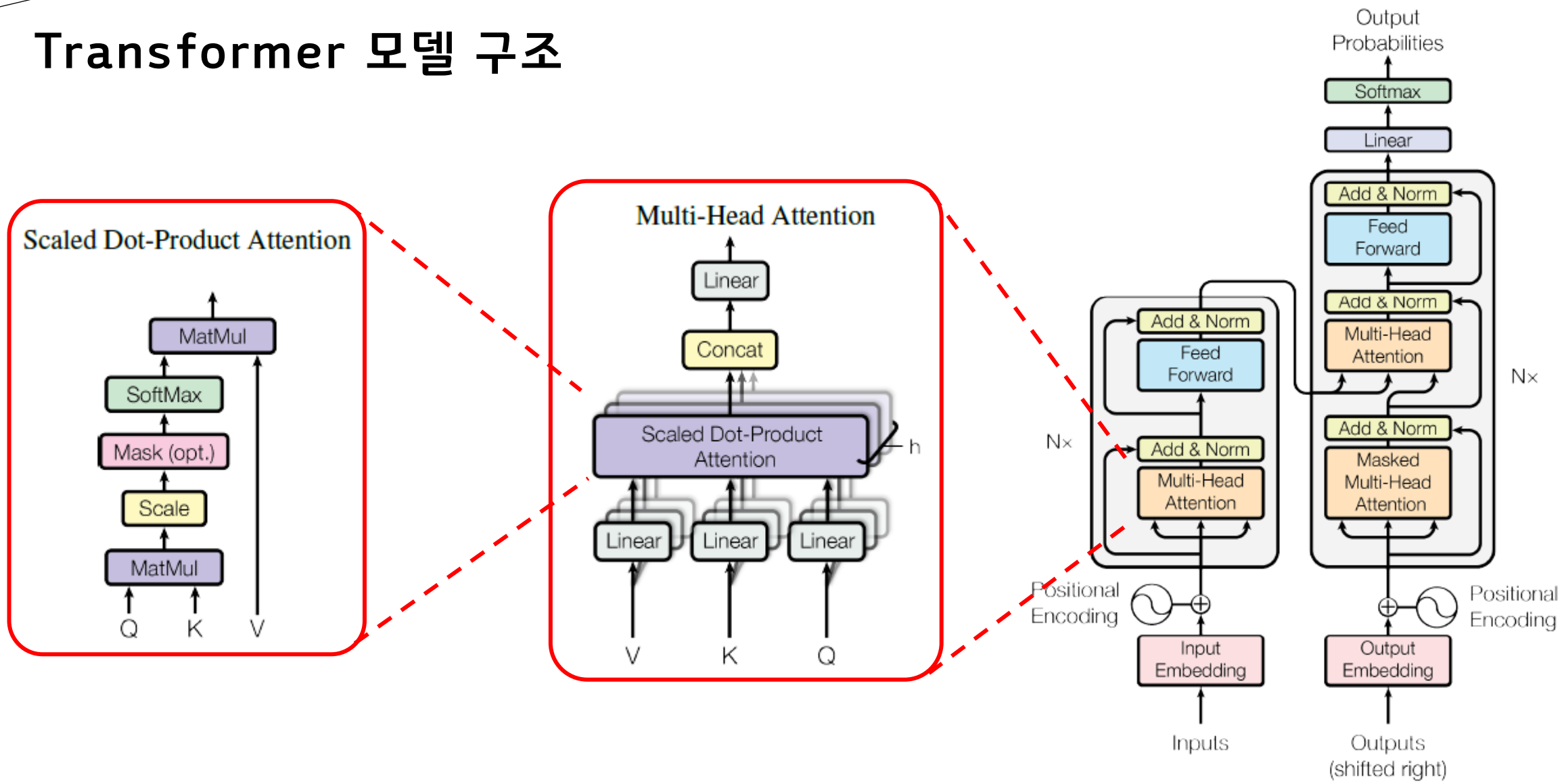
- 기존 Seq2Seq 모델들의 한계점
  - Decoder가 **매번 context vector를 참고**할 수도 있음
    - 여전히 **소스 문장을 하나의 벡터에 압축**해야하는 문제 발생



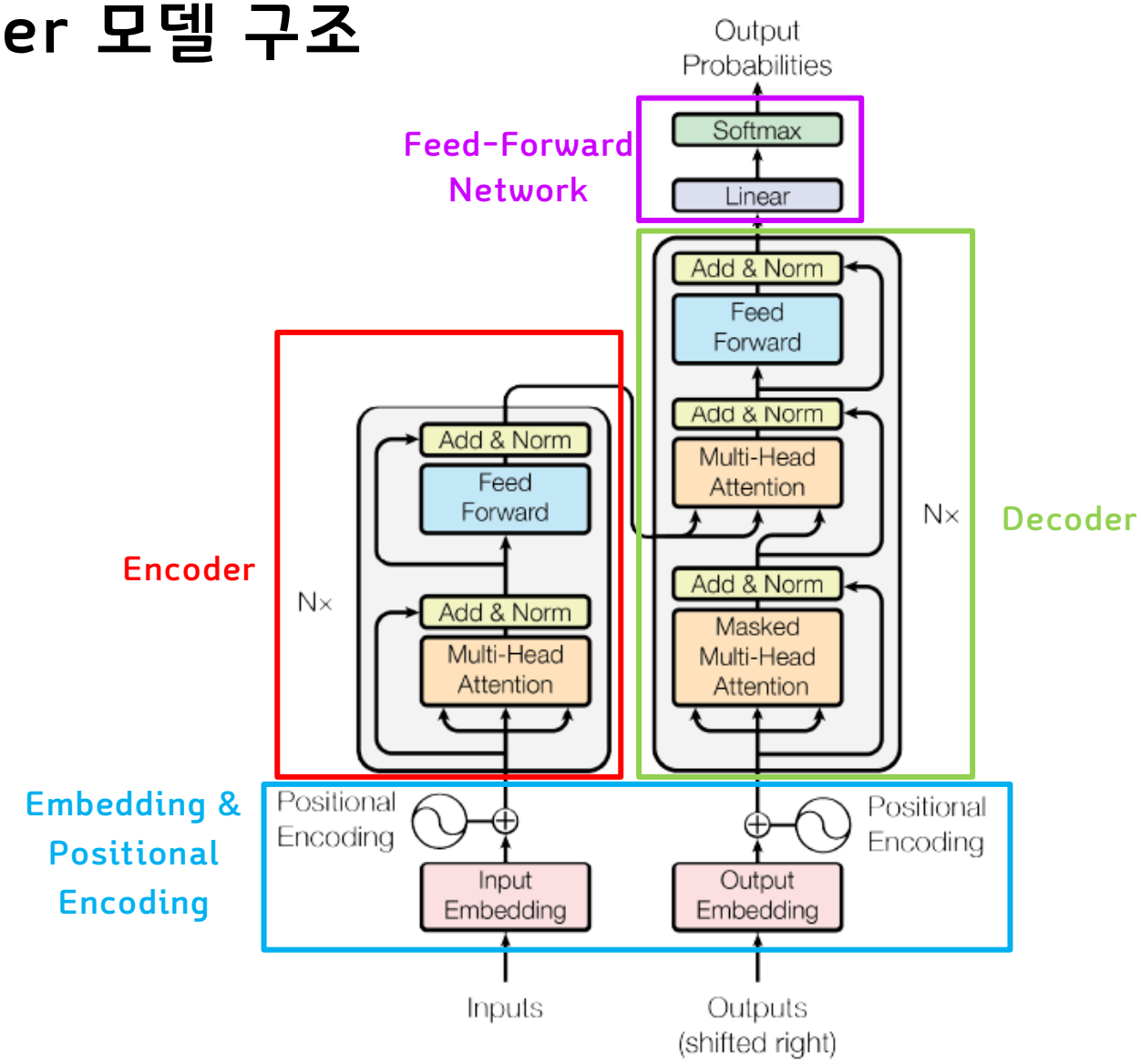
(출처: Youtube 채널 <동빈나>, https://youtu.be/AA621UofTUA)
[딥러닝 기계 번역] Transformer: Attention Is All You Need (꼼꼼한 딥러닝 논문 리뷰와 코드 실습)

Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation (EMNLP 2014)

Transformer 모델 구조



Transformer 모델 구조



Transformer 모델 구조

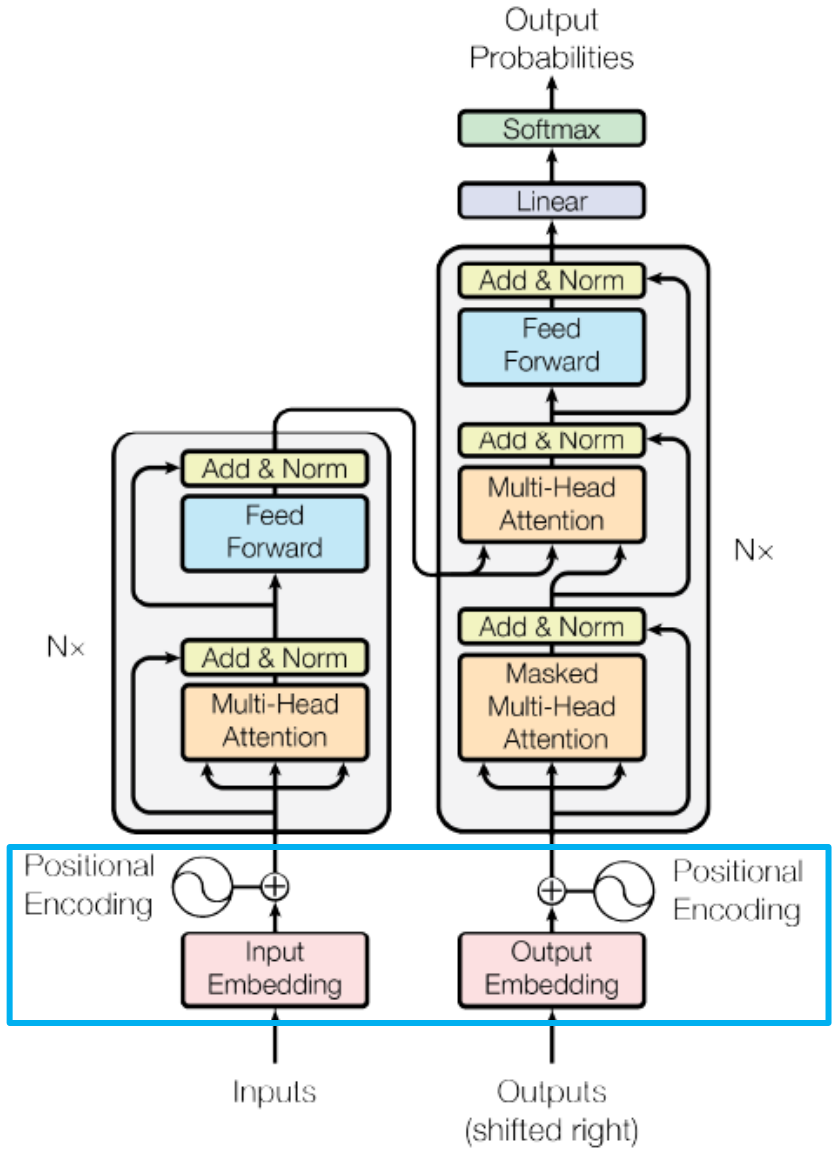
- Embedding & Positional Encoding

- Embedding
  - 일반적인 Embedding을 사용
- Positional Encoding
  - 단어의 위치정보를 Embedding Vector에 전달하기 위한 방법

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

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

Embedding & Positional Encoding

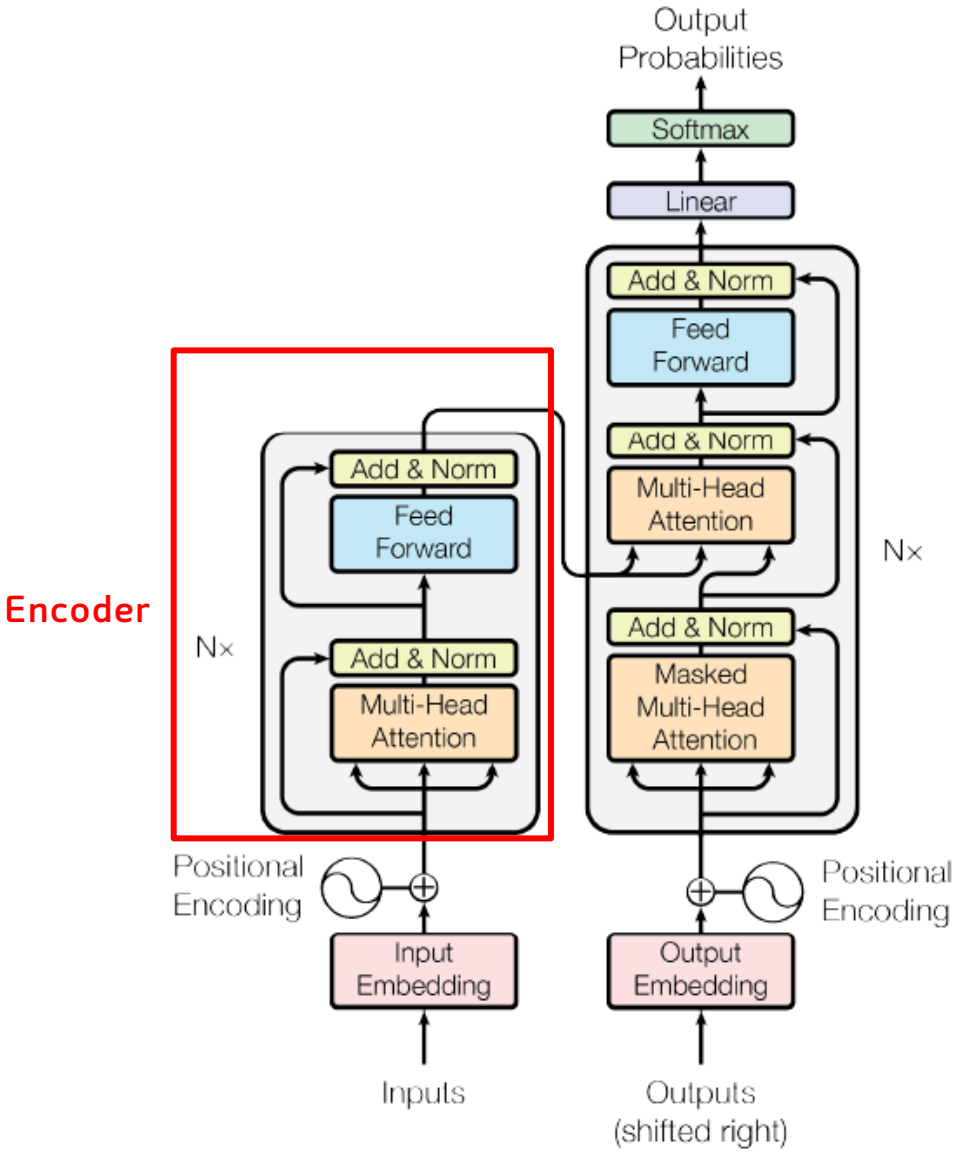


Transformer 모델 구조

- Encoder

- Self-Attention Mechanism을 활용하여 문맥을 분석, Attention Value 반환.
- Residual Connection을 활용.
- 입력과 출력의 크기가 같기 때문에 여러 번 중첩 가능.

- 본 논문에서는 6개의 Encoder를 중첩.

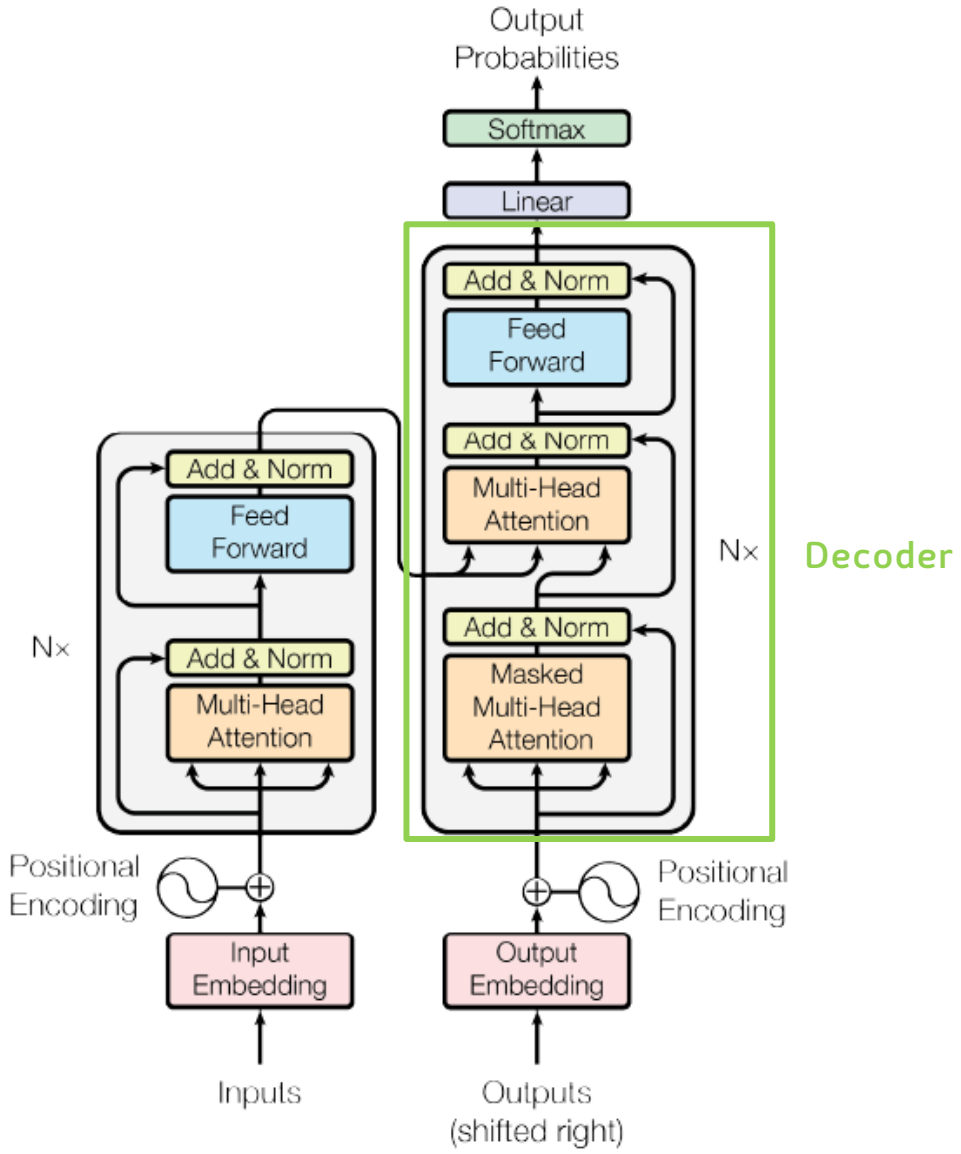


Transformer 모델 구조

- Decoder

- Encoder에서 생성된 문맥 정보(Attention Value)를 활용하여 입력된 단어에 이어 올 단어를 예측할 정보를 Fully Connected Layer와 softmax에 전달.
- Residual Connection을 활용.
- 입력과 출력의 크기가 같기 때문에 여러 번 중첩 가능.

- 본 논문에서는 6개의 Decoder를 중첩.
- Encoder에서 반환된 Attention Value가 모든 Decoder에 반영됨.

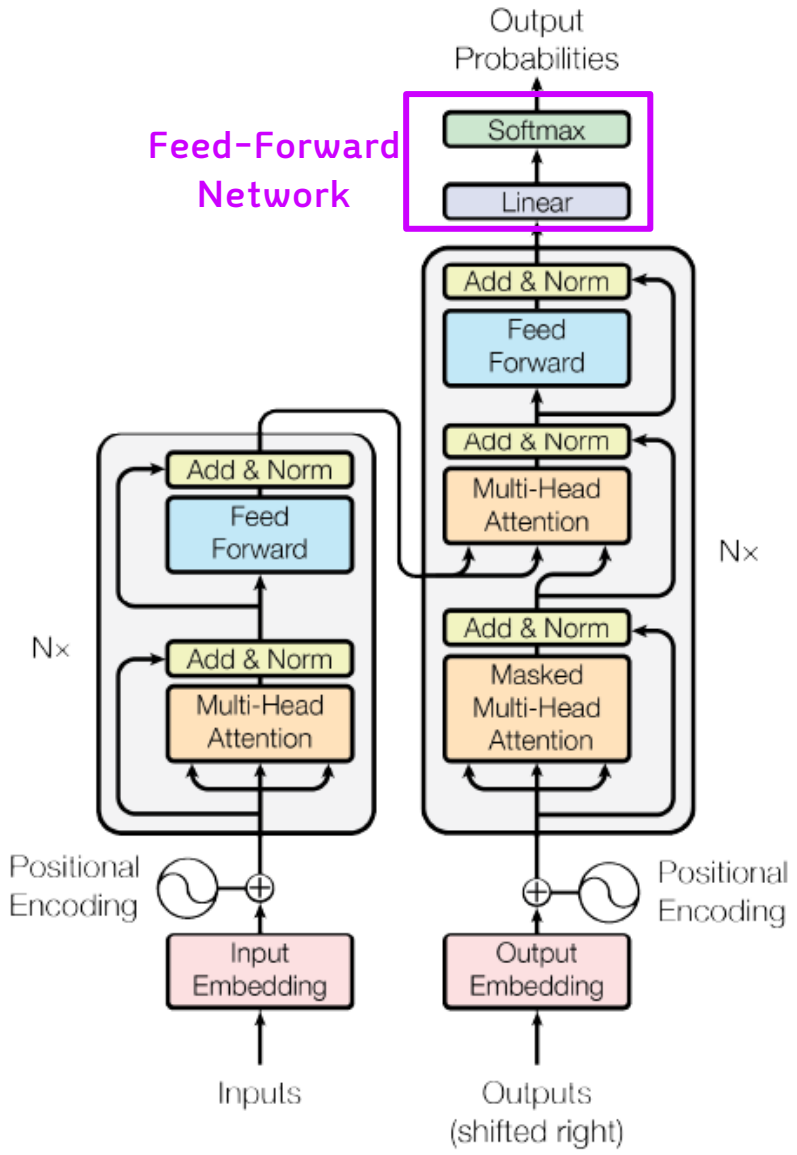


Transformer 모델 구조

- Feed-Forward Network

- Decoder의 연산 결과에 따라 입력 단어에 이어 올 단어를 순차적으로 예측.
- i번째 출력된 단어가 i+1번째 Decoder의 입력으로 사용됨.
- Linear과 ReLU로 구성

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



Positional Encoding

$(pos = \text{position},\ i = \text{dimension})$

$$PE(pos, i) = \sin\left(\frac{pos}{10000^{\frac{i}{d_{model}}}}\right)\ (i = 2k)$$

$$PE(pos, i) = \cos\left(\frac{pos}{10000^{\frac{i-1}{d_{model}}}}\right)\ (i = 2k + 1)$$

Positional Encoding

위치값 표시 원칙

1. 입력 1과 입력 2의 길이가 다르더라도 같은 위치라면 위치 벡터 값이 같아야 함.
2. 기존 Embedding 정보를 압도해서는 안 됨. (본 논문이 위치값을 Embedding Vector에 더하는 방법을 택했기 때문)

Positional Encoding

위치값 표시 원칙

1. 입력 1과 입력 2의 길이가 다르더라도 같은 위치라면 위치 벡터 값이 같아야 함.
2. 기존 Embedding 정보를 압도해서는 안 됨. (본 논문이 위치값을 Embedding Vector에 더하는 방법을 택했기 때문)

위치값 표시 방법

1. 위치에 따라 1씩 증가하는 정수로 위치를 표시

Positional Encoding

위치값 표시 원칙

1. 입력 1과 입력 2의 길이가 다르더라도 같은 위치라면 위치 벡터 값이 같아야 함.
2. 기존 Embedding 정보를 압도해서는 안 됨. (본 논문이 위치값을 Embedding Vector에 더하는 방법을 택했기 때문)

위치값 표시 방법

1. 위치에 따라 1씩 증가하는 정수로 위치를 표시
2. 0과 1 사이를 입력의 길이로 나누어 표시

Positional Encoding

위치값 표시 원칙

1. 입력 1과 입력 2의 길이가 다르더라도 같은 위치라면 위치 벡터 값이 같아야 함.
2. 기존 Embedding 정보를 압도해서는 안 됨. (본 논문이 위치값을 Embedding Vector에 더하는 방법을 택했기 때문)

위치값 표시 방법

1. 위치에 따라 1씩 증가하는 정수로 위치를 표시
2. 0과 1 사이를 입력의 길이로 나누어 표시
3. 주기함수를 활용 (sine, cosine)

Positional Encoding

$(pos = \text{position},\ i = \text{dimension})$

$$PE(pos, i) = \sin\left(\frac{pos}{10000^{\frac{i}{d_{model}}}}\right)\ (i = 2k)$$

$$PE(pos, i) = \cos\left(\frac{pos}{10000^{\frac{i-1}{d_{model}}}}\right)\ (i = 2k + 1)$$

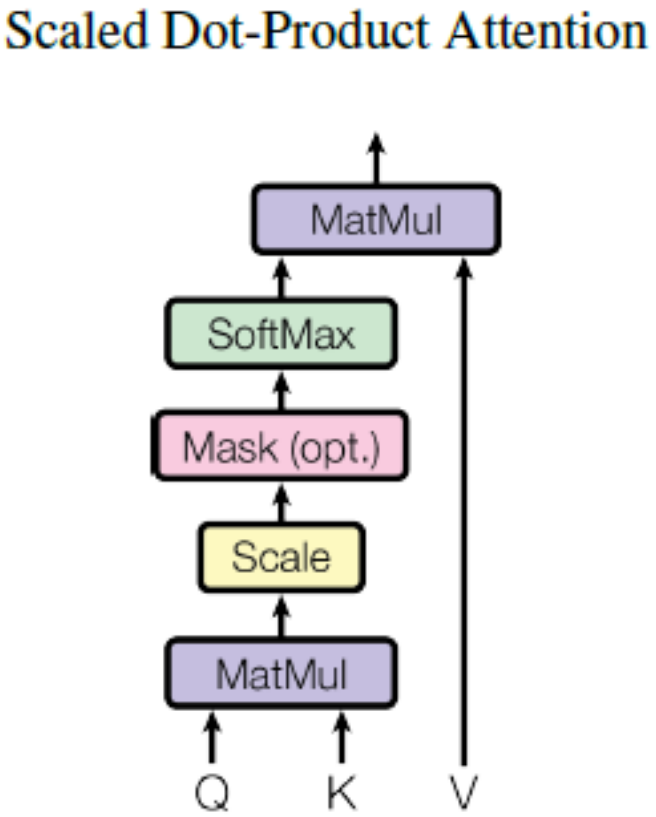
Transformer에 사용된 Attention

- Scaled Dot-Product Attention

- 기존 Attention 메커니즘에서의 용어 정리
  - Query : 각 단어들의 Attention Score을 구할 문장(A) 벡터
  - Key : Query의 Attention Score을 구할 기준 문장(B) 벡터
  - Value : Query가 Key에 대한 주목도를 구하기 위한 문장(B) 벡터
  - 즉, Query ≠ Key = Value

- **단, Transformer 모델에서의 Query, Key, Value는 모두 같은 값**
  - **그래서 Self-Attention!**

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

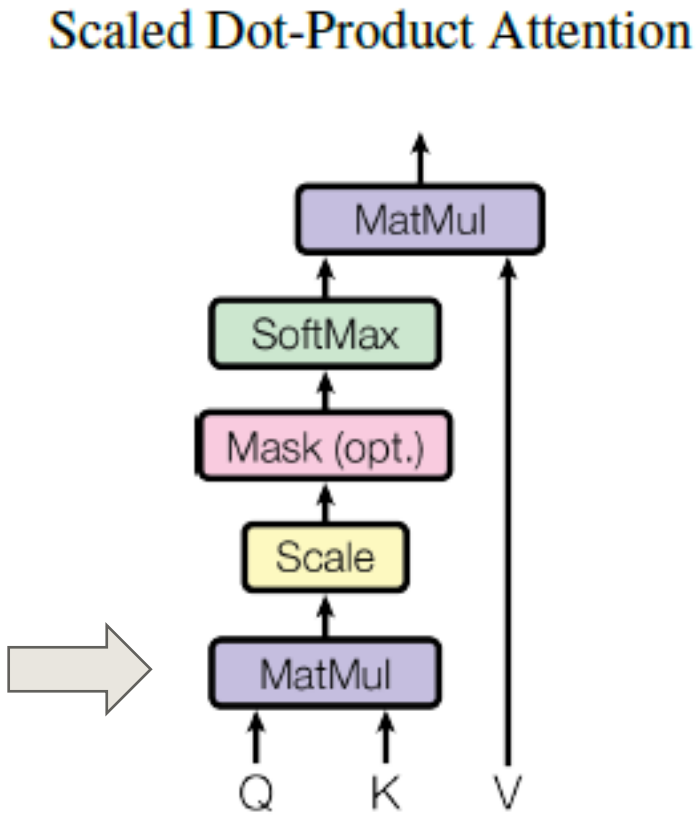


Transformer에 사용된 Attention

- Scaled Dot-Product Attention

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

- **1st Matmul**: Query와 Key 사이의 유사도 연산.
  (Output : 문장 길이 × 문장 길이)
  (내적 ≒ 코사인 유사도)
- **Scale**: 입력이 길어져 내적의 편차가 커질 경우를 대비한 Scaling.
  Key 행렬 차원에 루트를 취한 값으로 전체를 나눔.
- **SoftMax**: Scaling이 끝난 Attention Score을 확률값으로 변환.
- **2nd Matmul**: Attention Score 확률값을 weight로 활용하여 weighted sum 연산.
  (Output: 문장 길이 × 임베딩 길이)

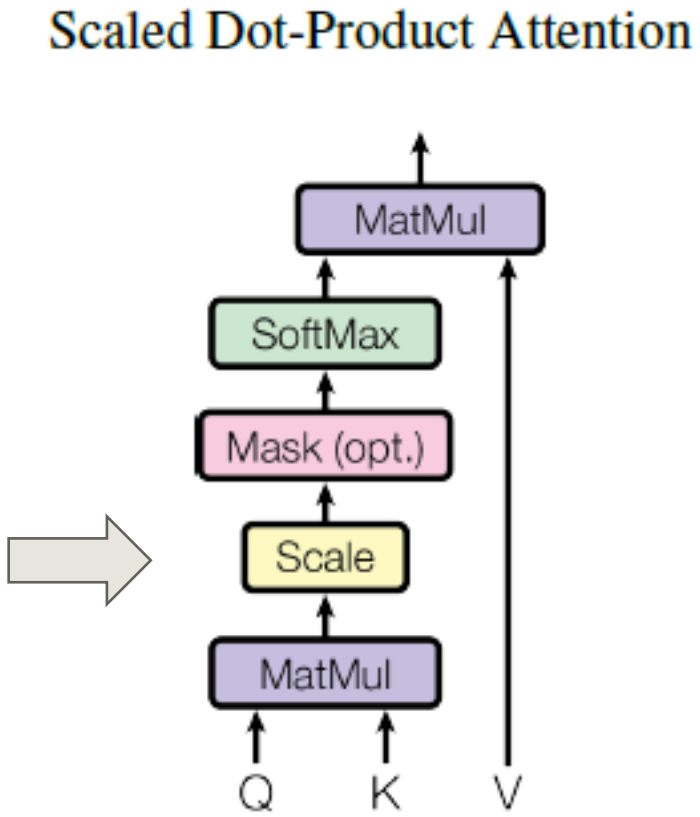


Transformer에 사용된 Attention

- Scaled Dot-Product Attention

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

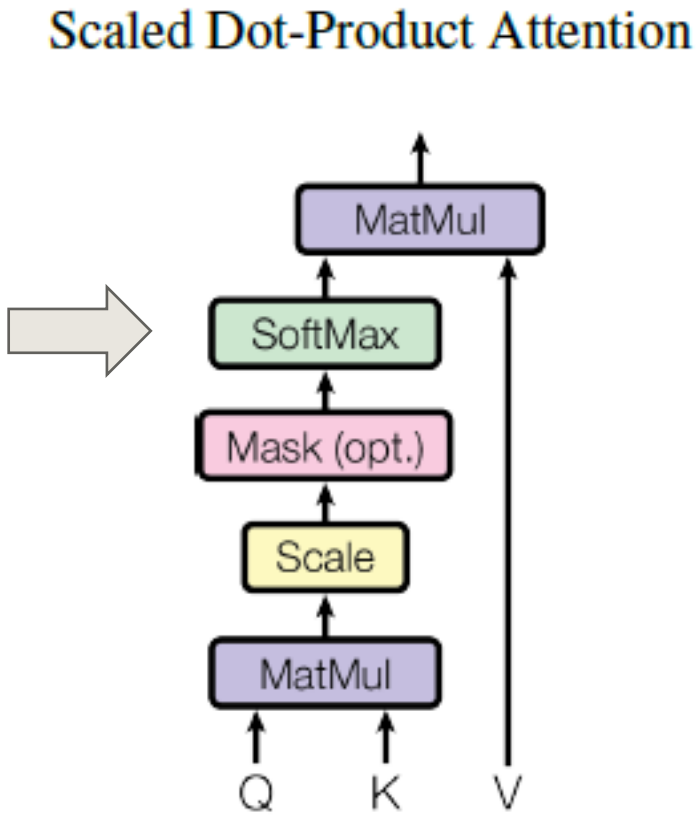
- **1st Matmul**: Query와 Key 사이의 유사도 연산.
  (Output : 문장 길이 × 문장 길이)
  (내적 ≒ 코사인 유사도)
- **Scale**: 입력이 길어져 내적의 편차가 커질 경우를 대비한 Scaling.
  Key 행렬 차원에 루트를 취한 값으로 전체를 나눔.
- **SoftMax**: Scaling이 끝난 Attention Score을 확률값으로 변환.
- **2nd Matmul**: Attention Score 확률값을 weight로 활용하여 weighted sum 연산.
  (Output: 문장 길이 × 임베딩 길이)



Transformer에 사용된 Attention

- Scaled Dot-Product Attention

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

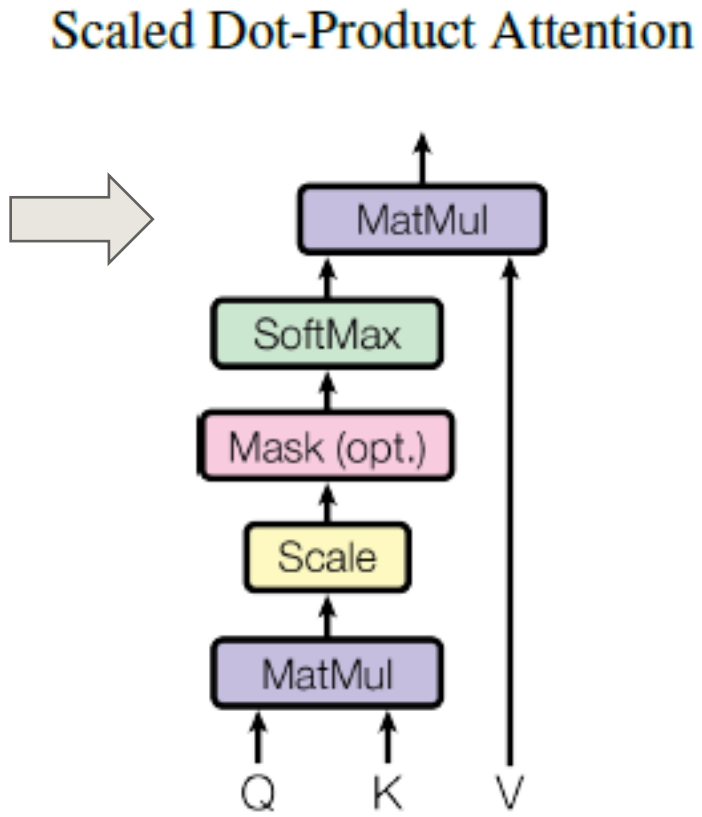


- **1st Matmul**: Query와 Key 사이의 유사도 연산.
  (Output : 문장 길이 × 문장 길이)
  (내적 ≒ 코사인 유사도)
- **Scale**: 입력이 길어져 내적의 편차가 커질 경우를 대비한 Scaling.
  Key 행렬 차원에 루트를 취한 값으로 전체를 나눔.
- **SoftMax**: Scaling이 끝난 Attention Score을 확률값으로 변환.
- **2nd Matmul**: Attention Score 확률값을 weight로 활용하여 weighted sum 연산.
  (Output: 문장 길이 × 임베딩 길이)

Transformer에 사용된 Attention

- Scaled Dot-Product Attention

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



- **1st Matmul**: Query와 Key 사이의 유사도 연산.
  (Output : 문장 길이 × 문장 길이)
  (내적 ≒ 코사인 유사도)
- **Scale**: 입력이 길어져 내적의 편차가 커질 경우를 대비한 Scaling.
  Key 행렬 차원에 루트를 취한 값으로 전체를 나눔.
- **SoftMax**: Scaling이 끝난 Attention Score을 확률값으로 변환.
- **2nd Matmul**: Attention Score 확률값을 weight로 활용하여 weighted sum 연산.
  (Output: 문장 길이 × 임베딩 길이)

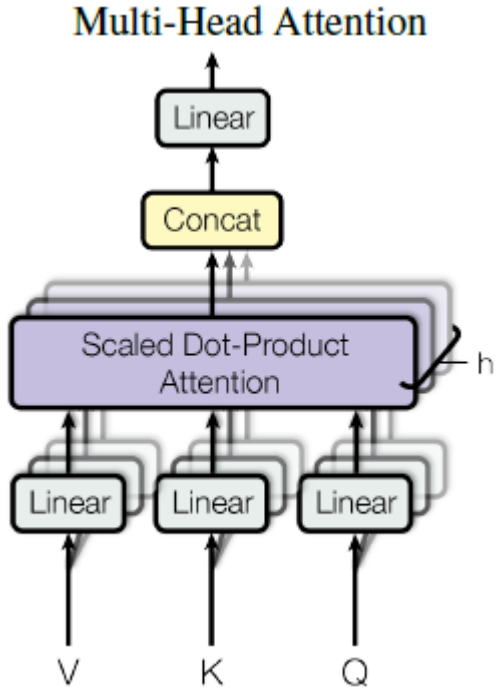
Transformer에 사용된 Attention

- Multi-Head Attention

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

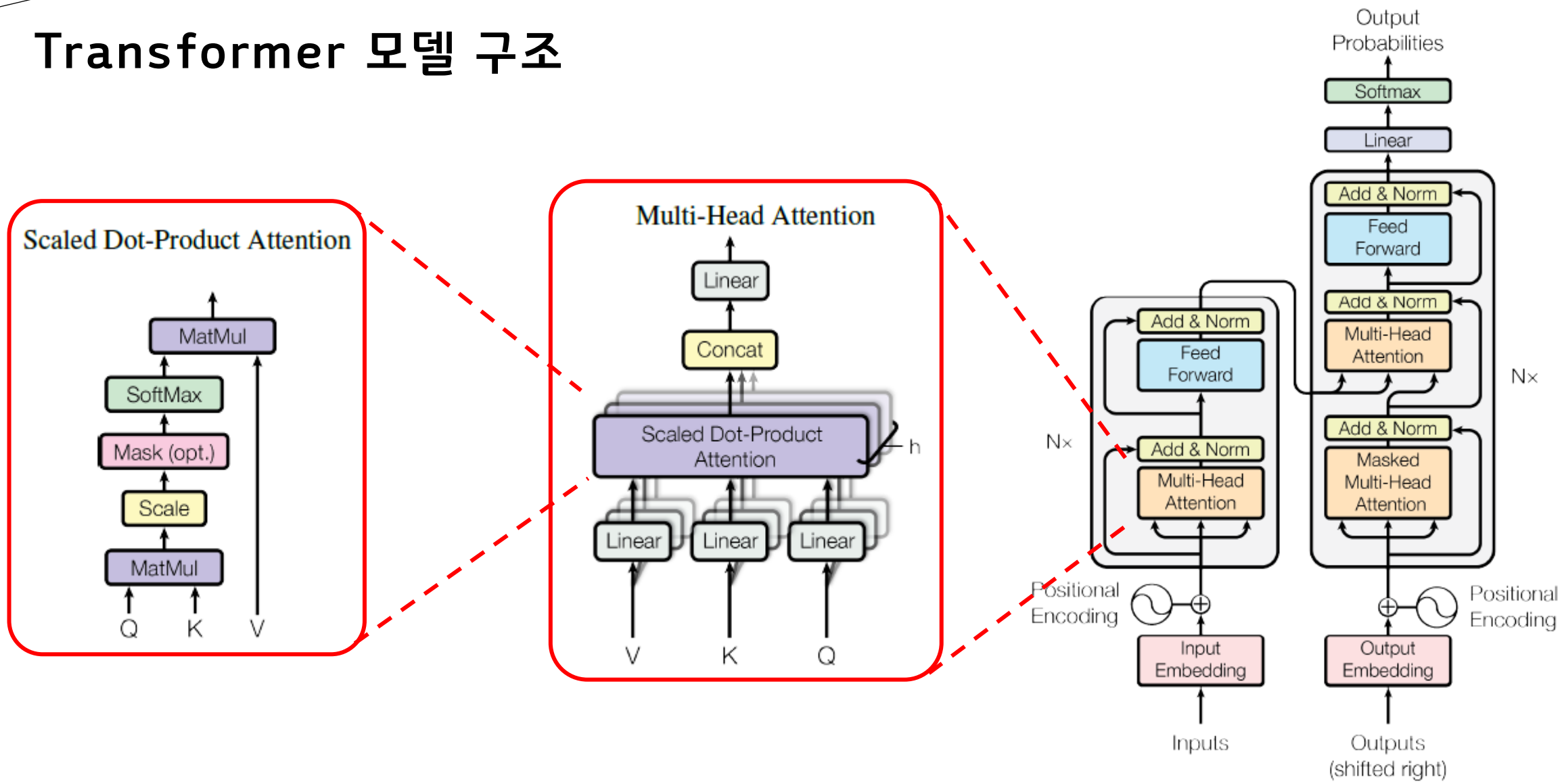
$$head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right)$$

$$MultiHead(Q, K, V) = Concat(head_1, \ldots head_h)W^O$$



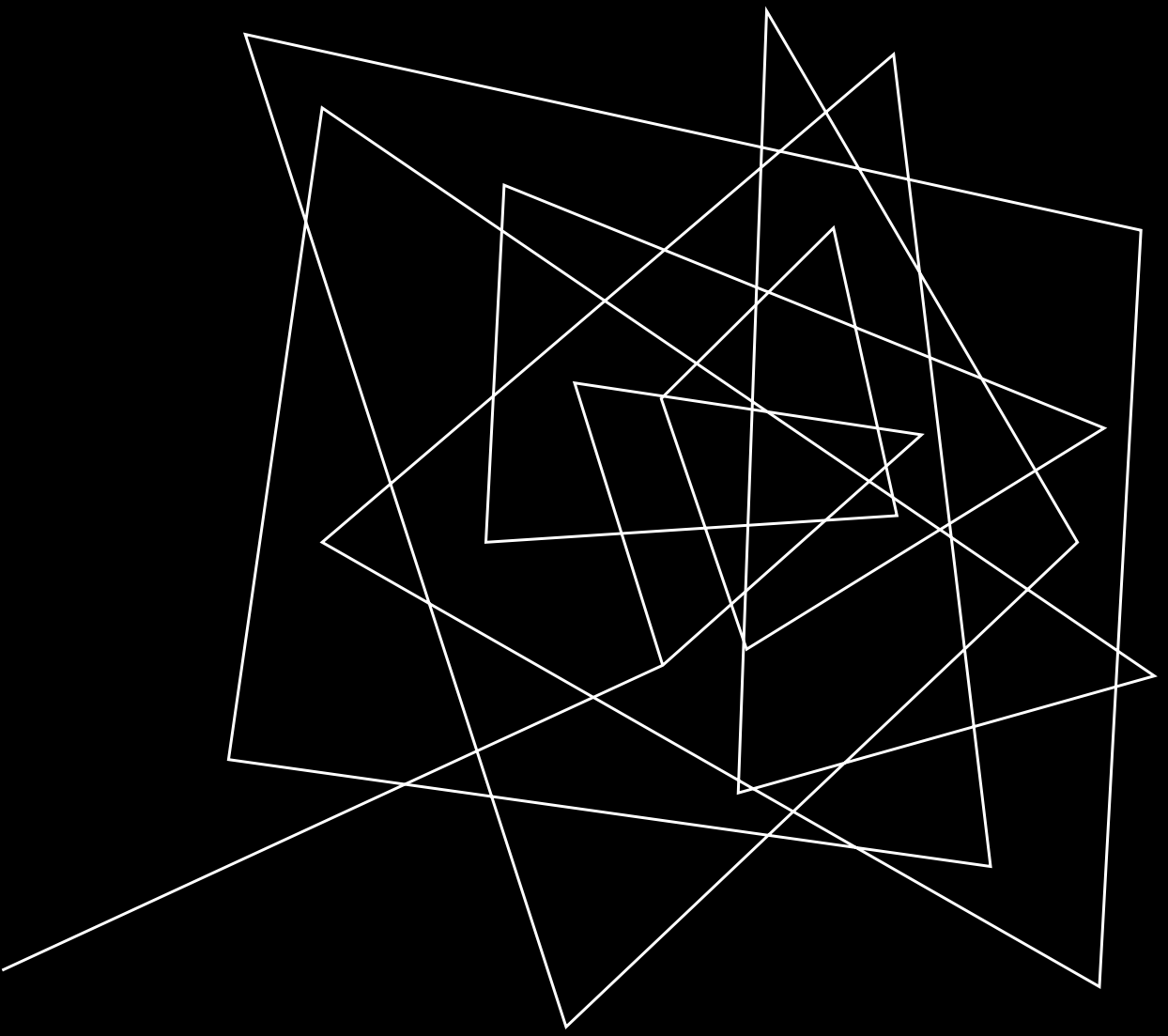
h : 헤드(head)의 개수

Transformer 모델 구조



(참고)Transformer 구현 코드

- **나동빈 강사님의 github**
  - https://github.com/ndb796/Deep-Learning-Paper-Review-and-Practice/blob/master/code_practices/Attention_is_All_You_Need_Tutorial_(German_English).ipynb



Q&A