Attention Is All You Need

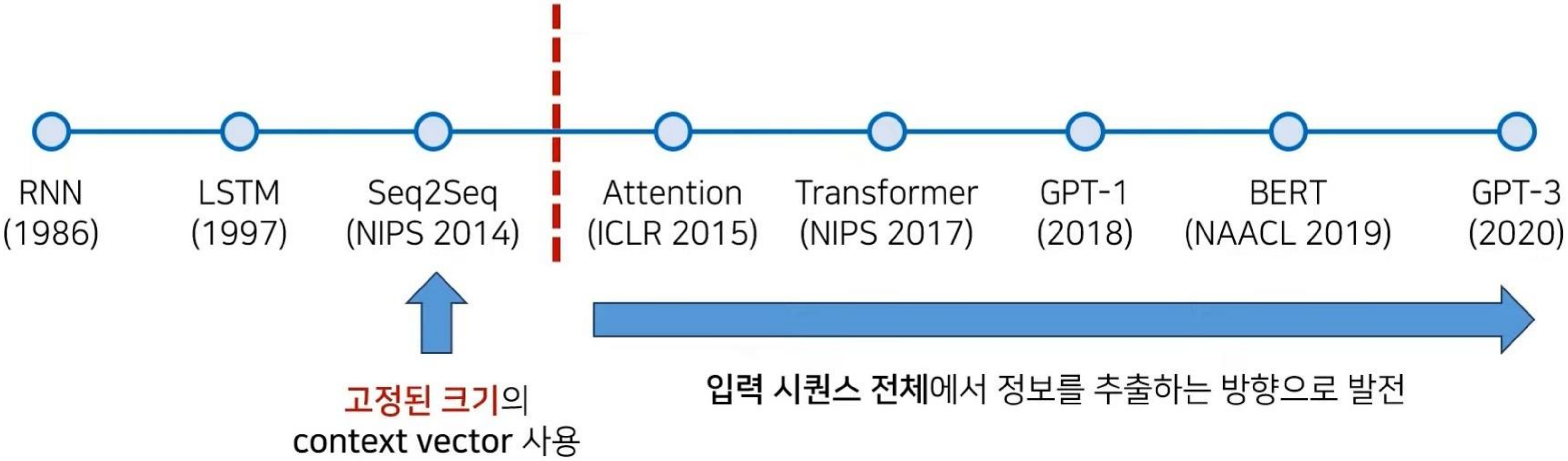
발표자 : 진아람

INDEX

Background

Background

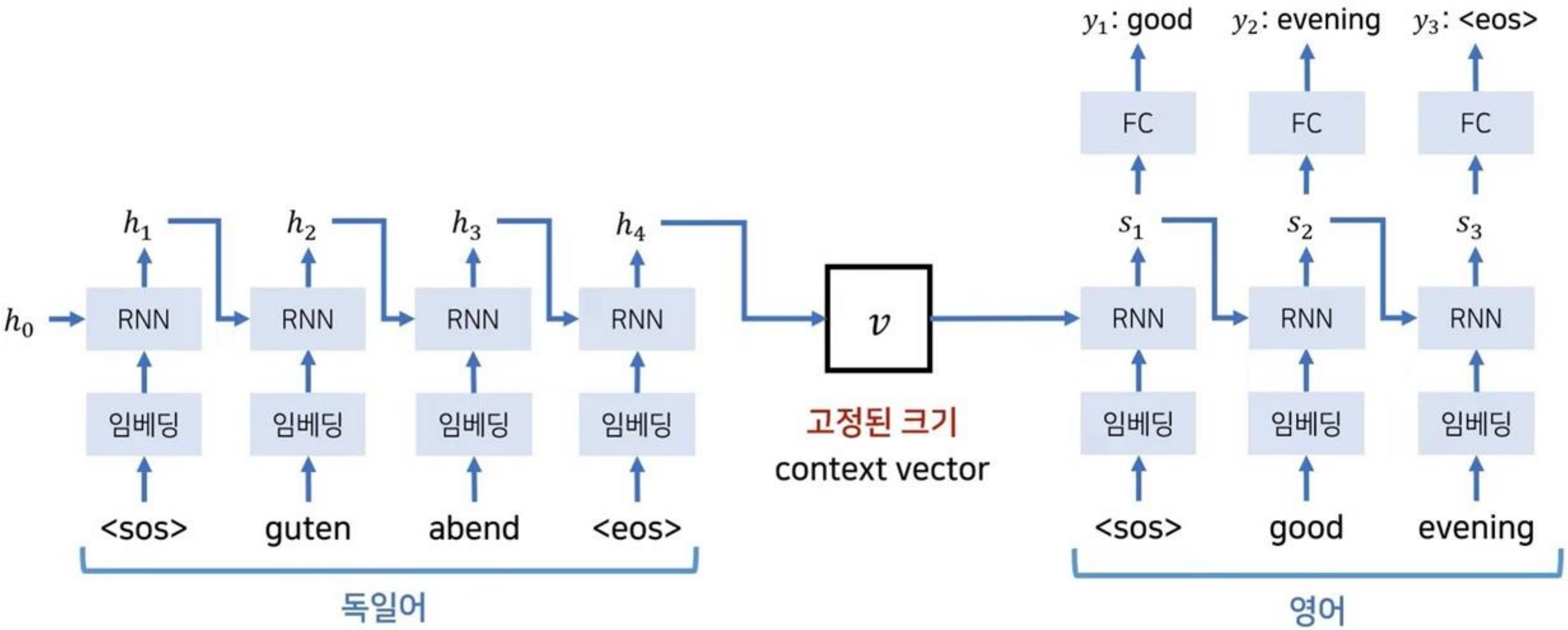
- 자연어 처리에 고성능 모델들은 Transformer 아키텍처기반
- 자연어처리에서 가장 대표적이면서 중요한 Task중 하나는 기계번역



Sequence to Sequence

Seq2seq

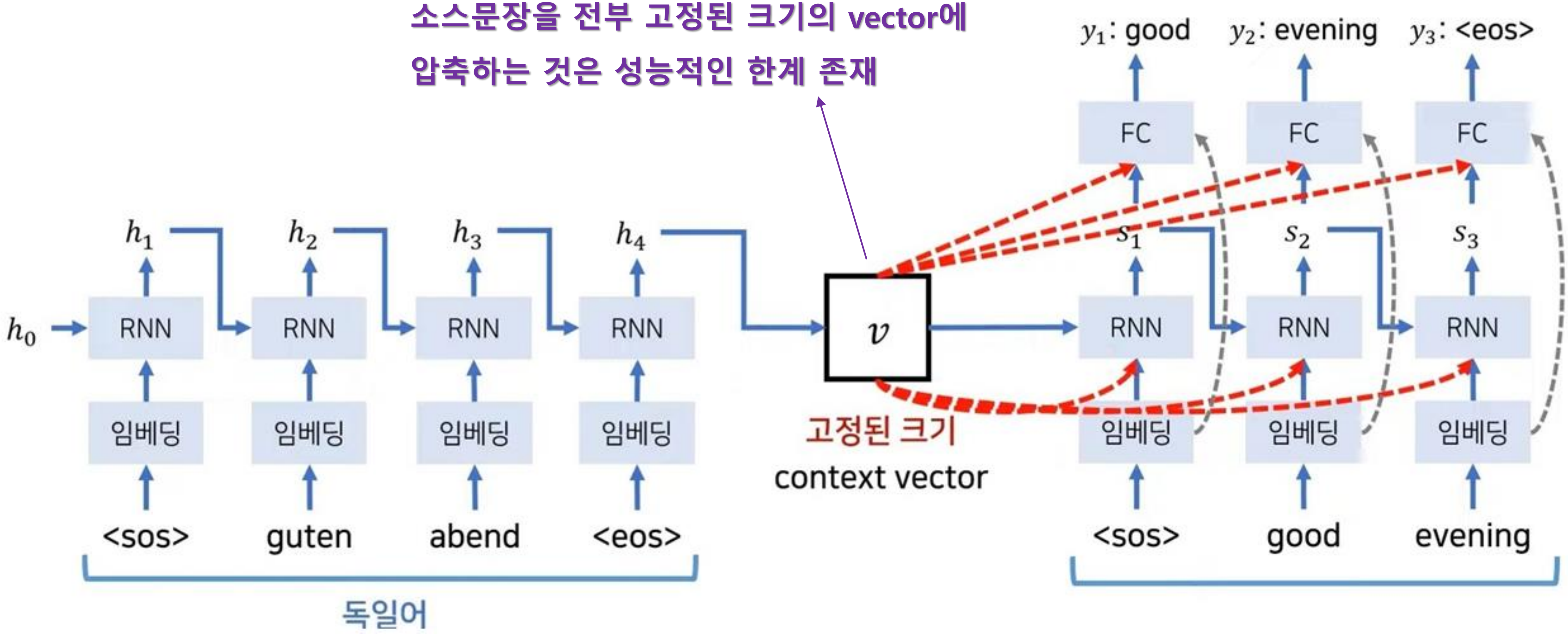
Basics (image)



Seq2seq

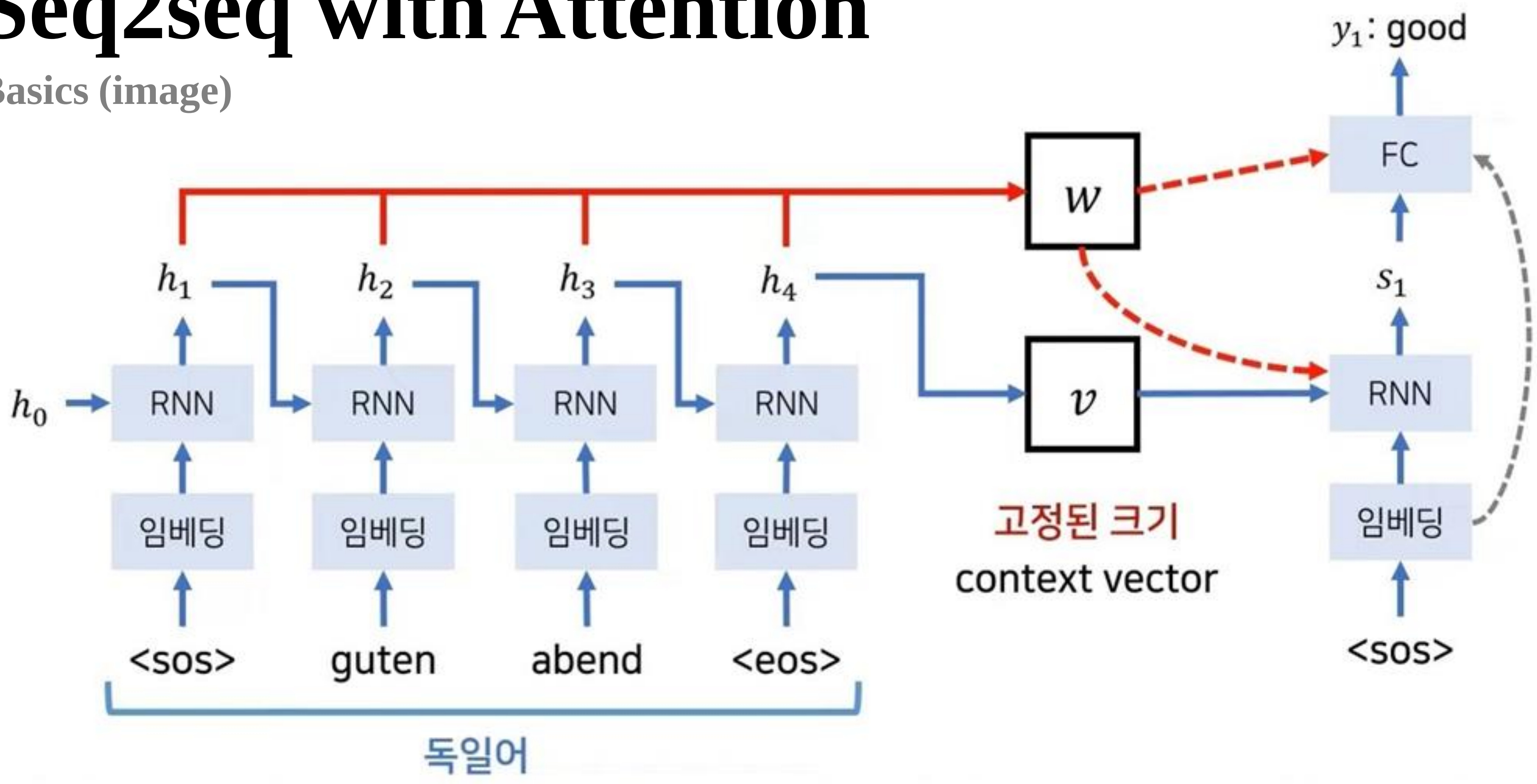
Basics (image)

소스문장을 전부 고정된 크기의 vector에
압축하는 것은 성능적인 한계 존재



Seq2seq with Attention

Basics (image)



입력되는 문장을 전부 기억하기위해 Attention매커니즘 도입(디코더는 인코더의 모든 출력을 참고)

But, RNN연산이기때문에 속도가 느리다.

Transformer

Transformer

Attention Is All You Need

- ✓ RNN,CNN연산을 사용하지 않음, **Attention**만을 사용 → 효율적인 **병렬화**
- ✓ 기존 기계번역 모델의 문제점인 <u>느린 속도</u>와 <u>위치정보 기억 소실 문제</u> 해결
- ✓ **Self-attention** 사용 (쿼리, 키 ,벨류)

Transformer

Architecture (overview)

✓ 속도 향상

→ Attention만으로 병렬연산

✓ 위치(순서) 정보 Embedding

→ Positional Encoding

(RNN이나 CNN을 사용하지 않음)

Encoder

Decoder

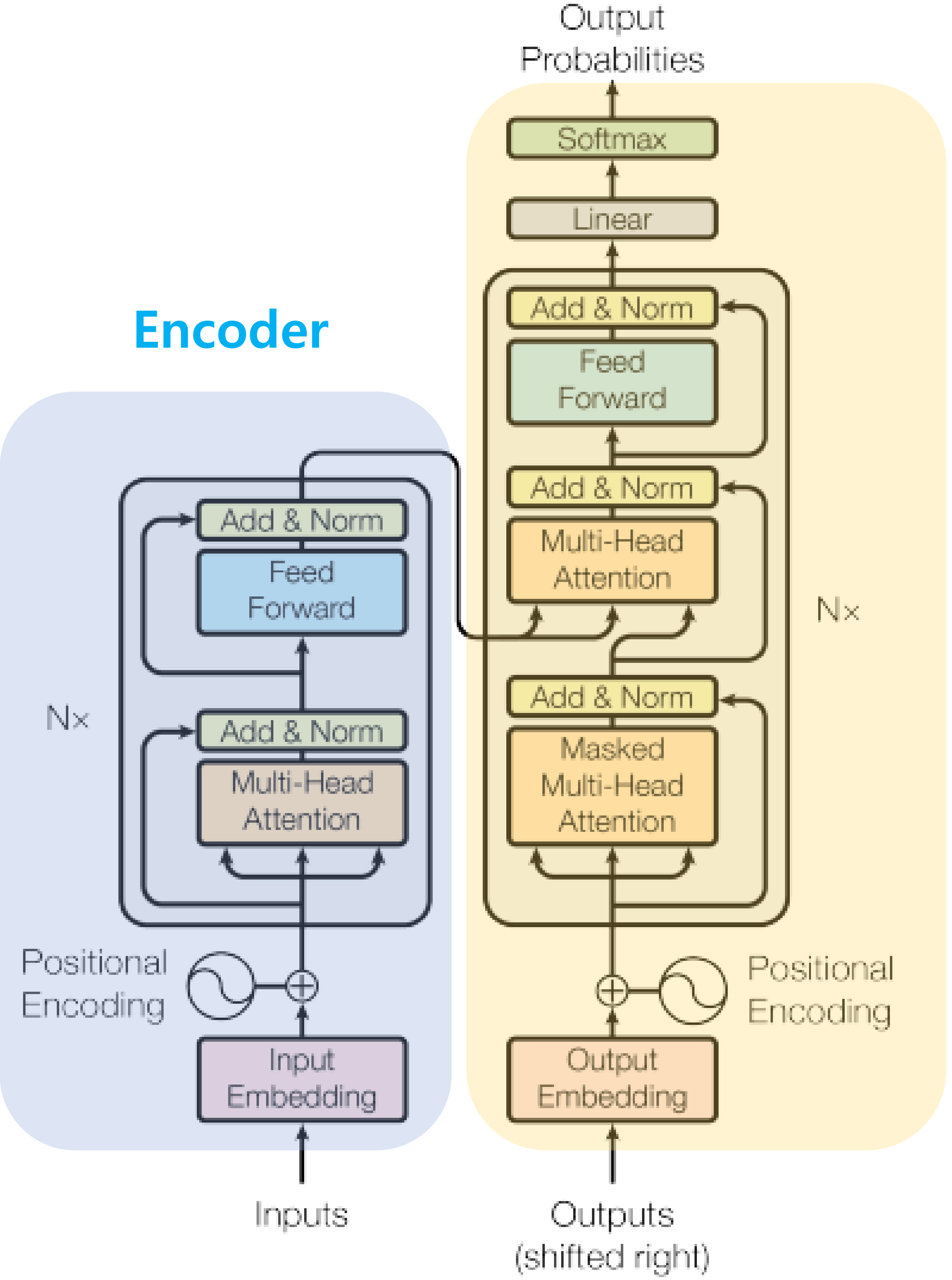
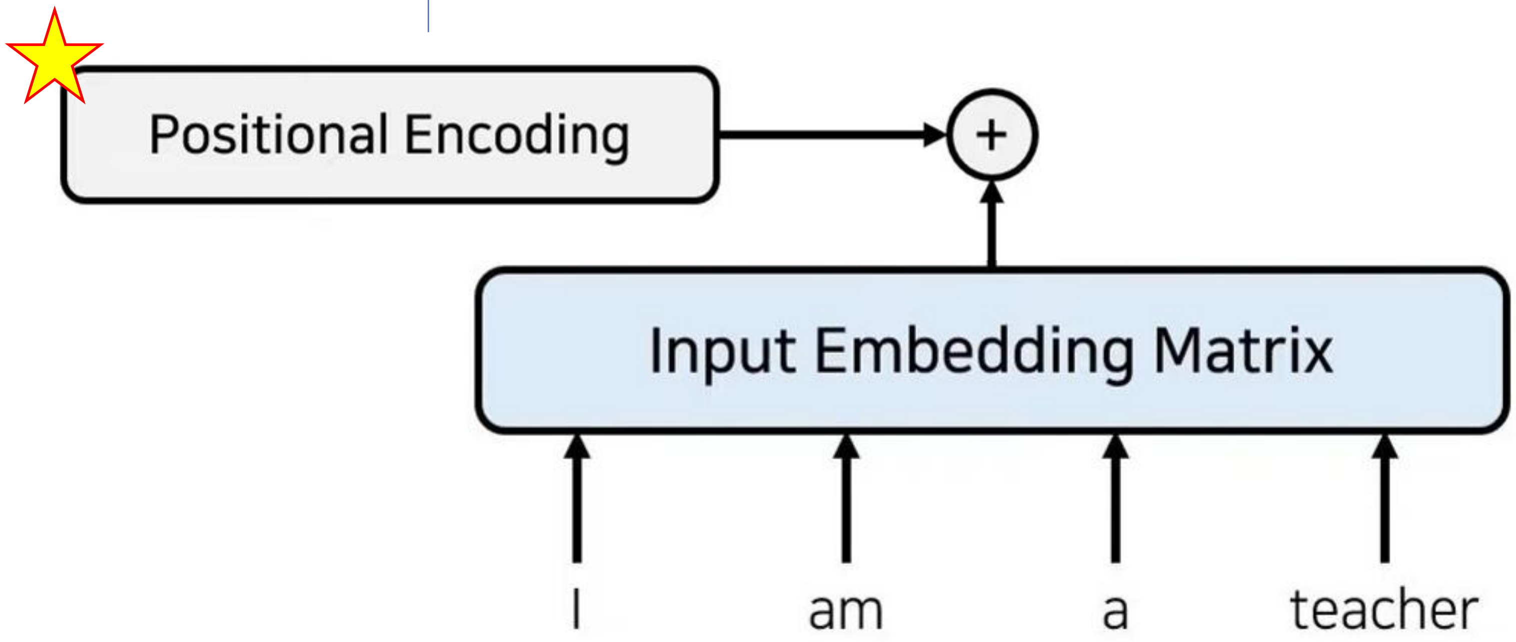


Figure 1: The Transformer - model architecture.

Transformer

Embedding

RNN을 사용하지 않기때문에 순서정보를 포함하고 있는 임베딩 사용

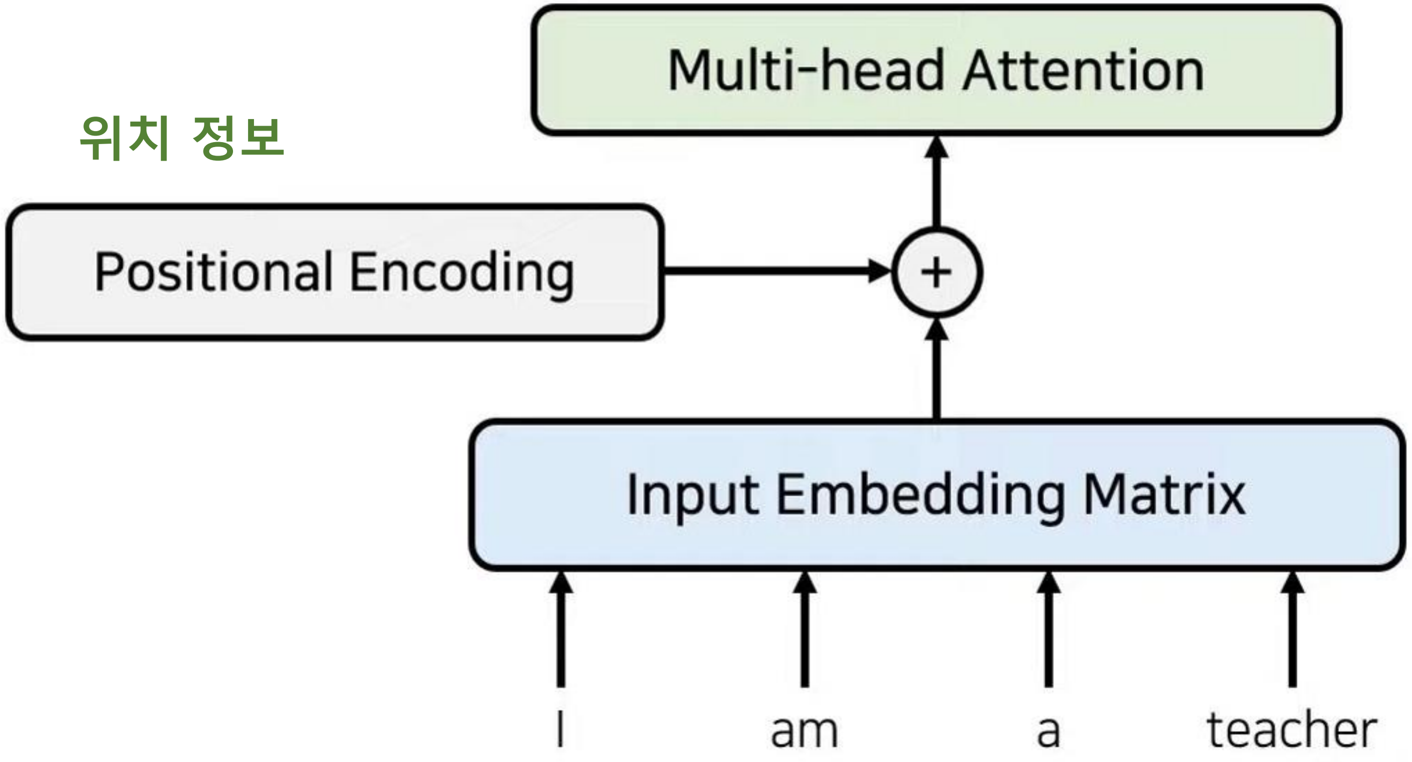


Transformer

Attention

인코더파트 : Self-Attention

위치 정보



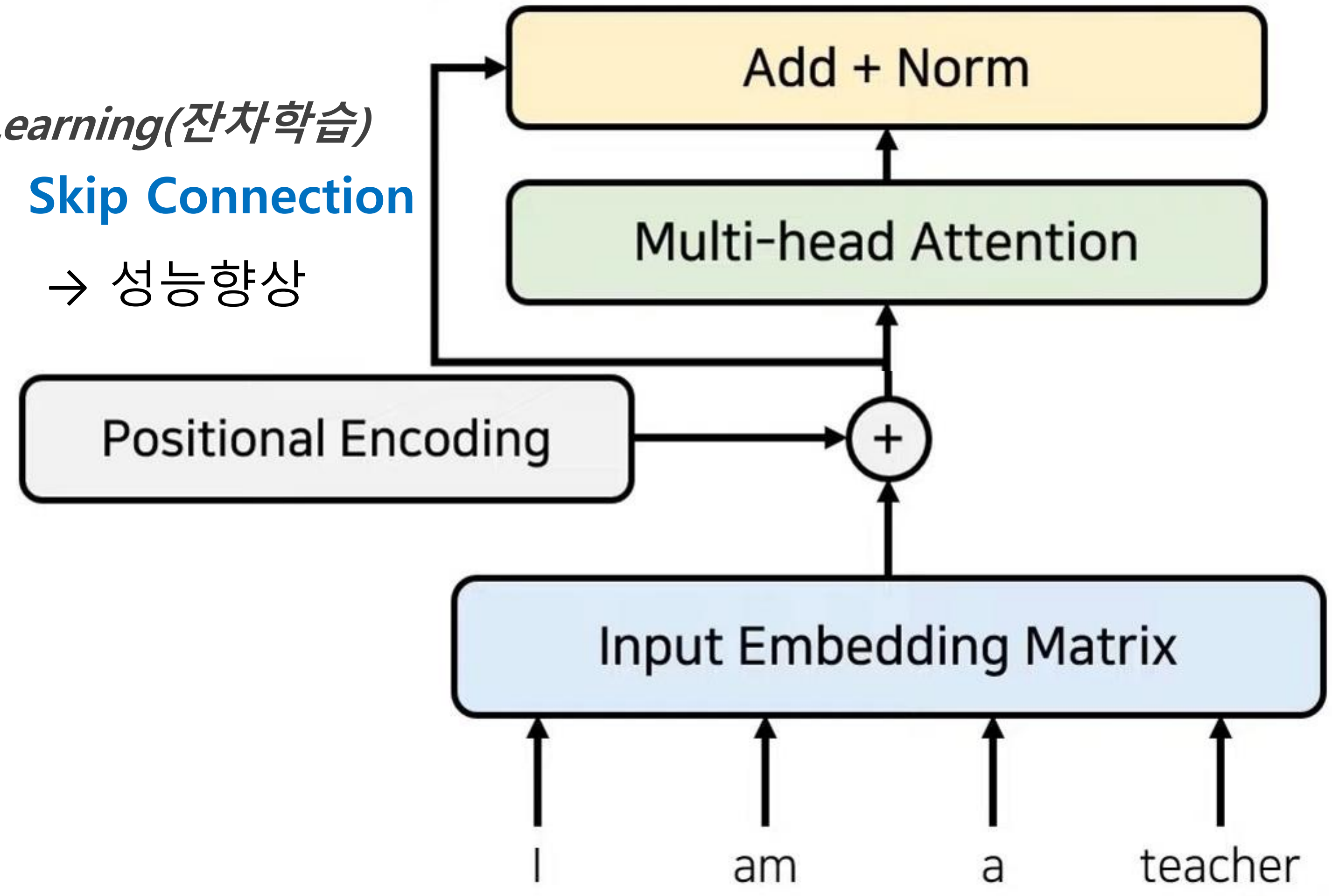
Transformer

Attention & Residual Learning

Residual Learning(잔차학습)

Skip Connection

→ 성능향상

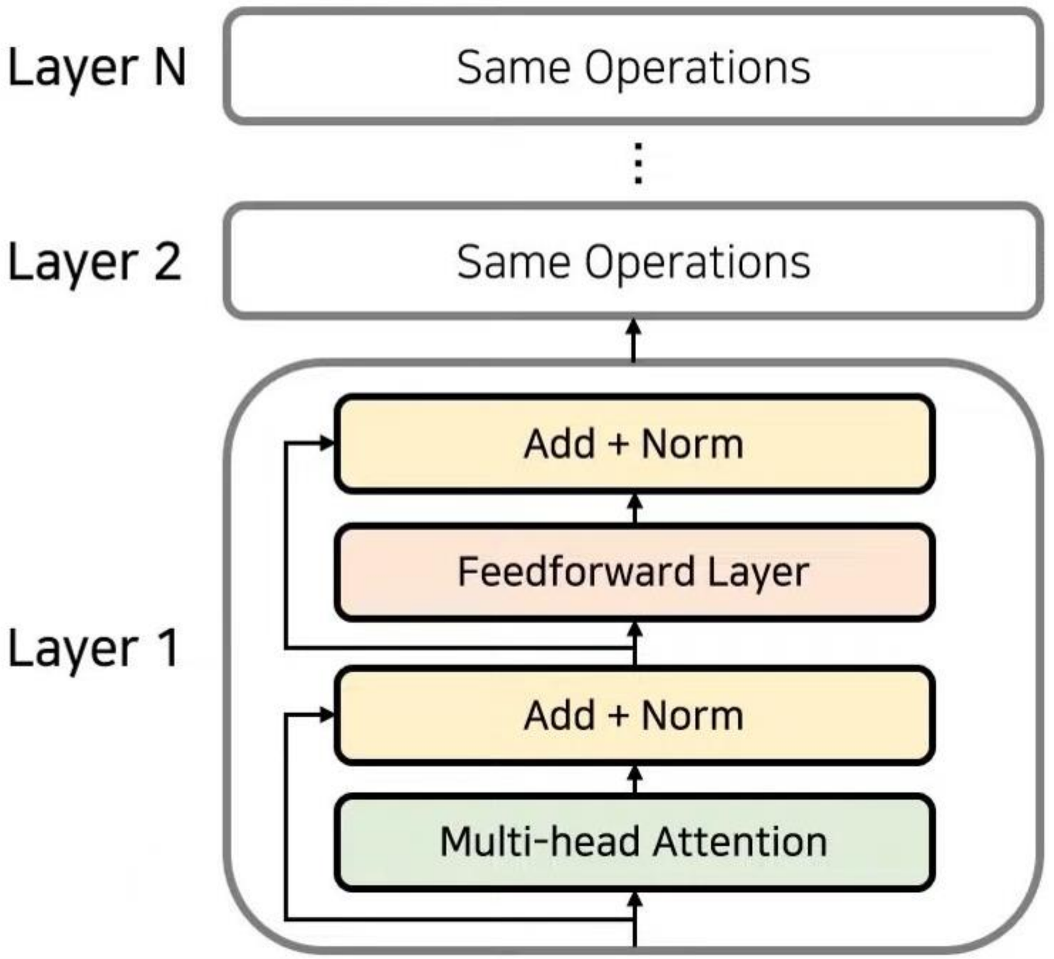


Transformer

Encoder

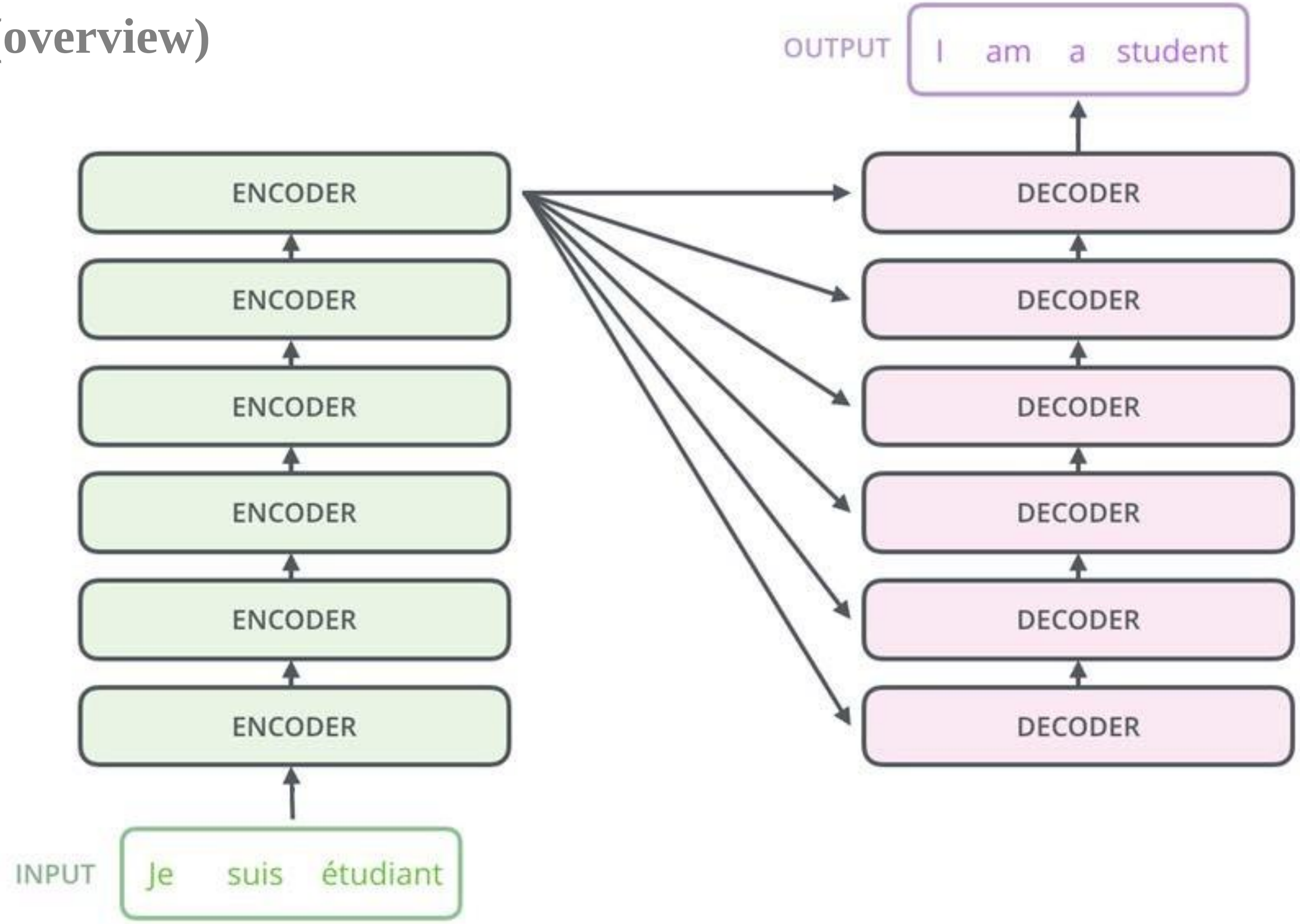
어텐션(Attention)과

정규화(Normalization)과정을 N번 반복



Transformer

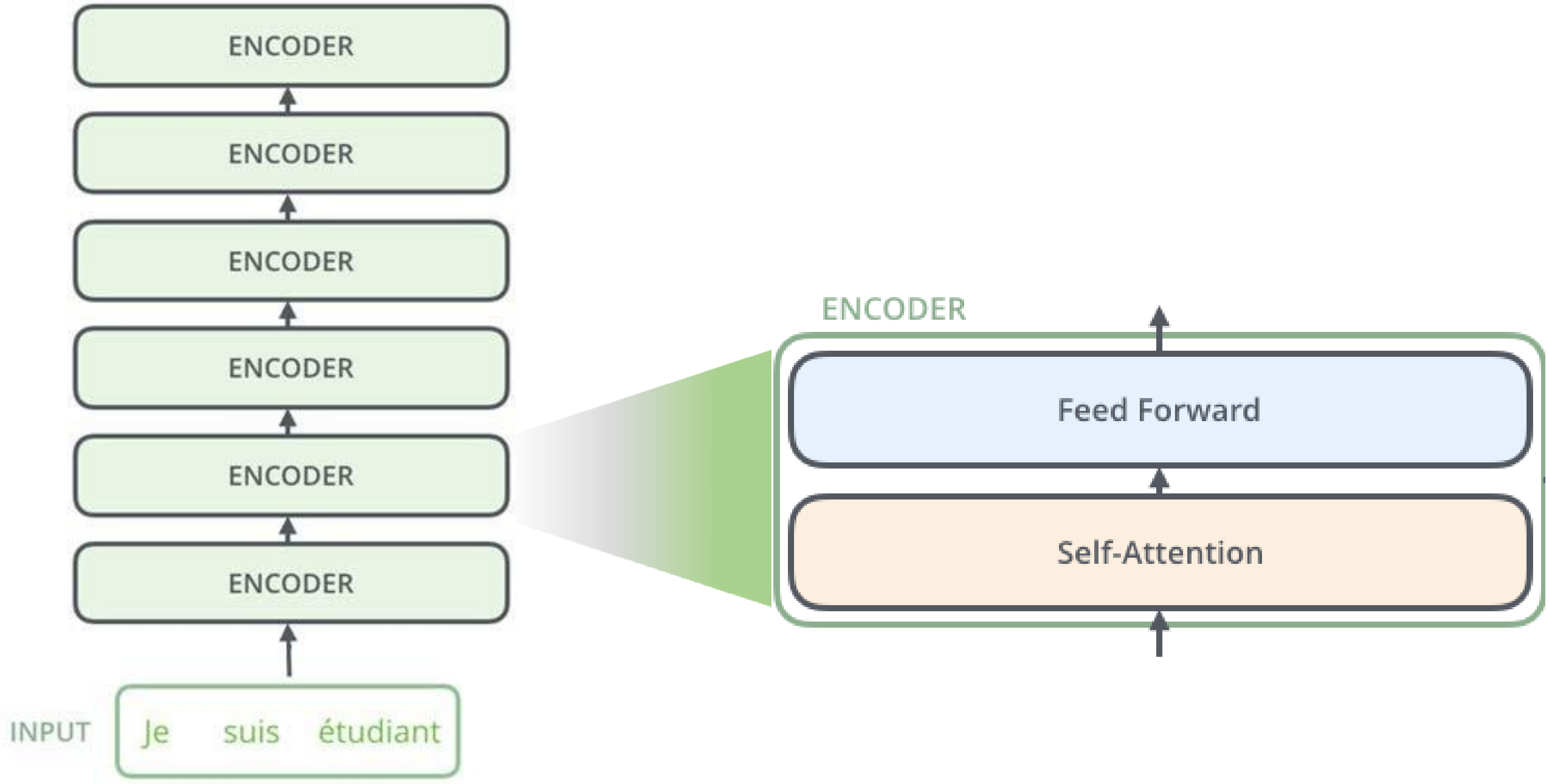
Architecture (overview)

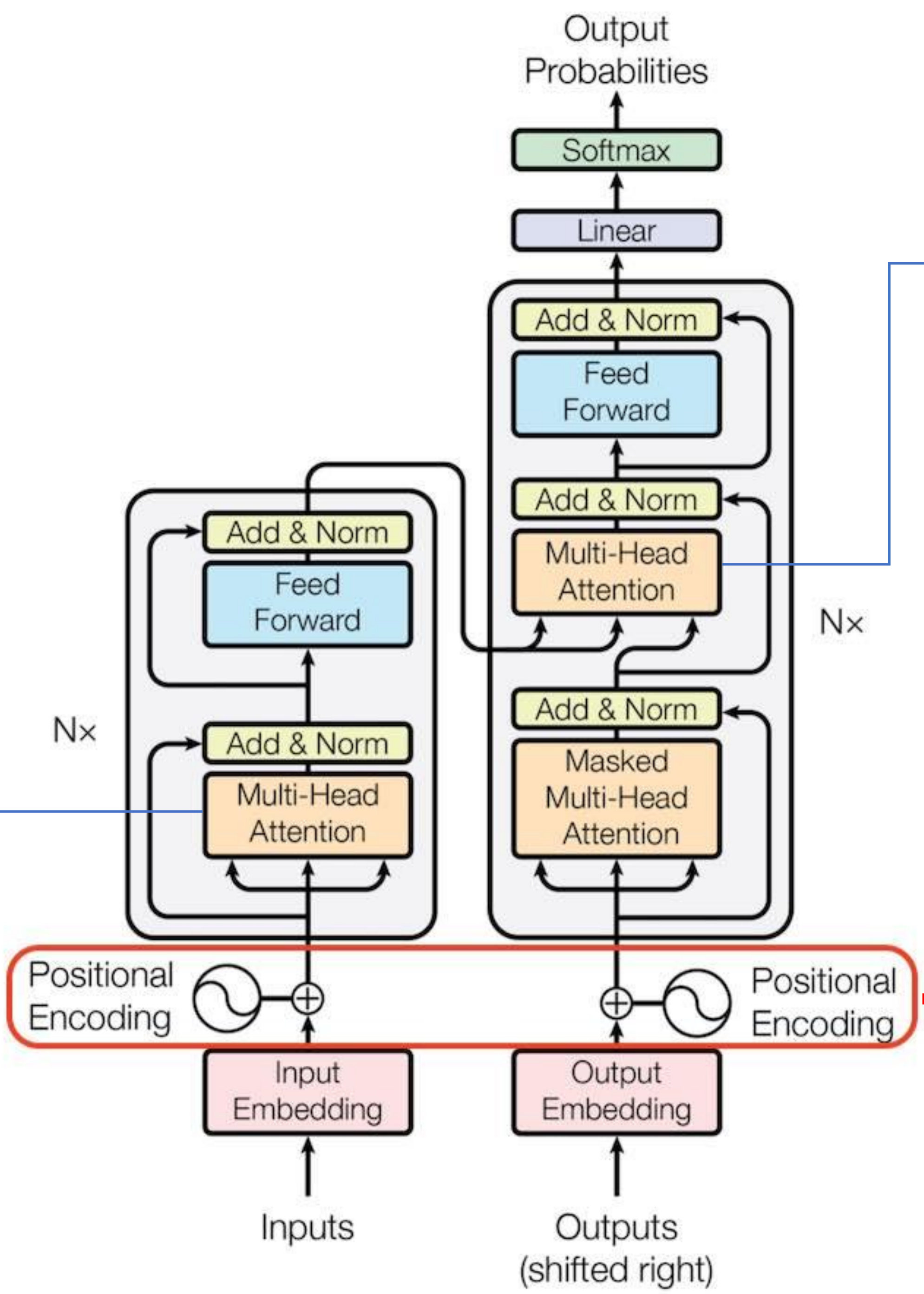


마지막 인코더 레이어의 출력이 모든 디코더 레이어에 입력

Transformer

Architecture (encoder)





Encoder Decoder Attention

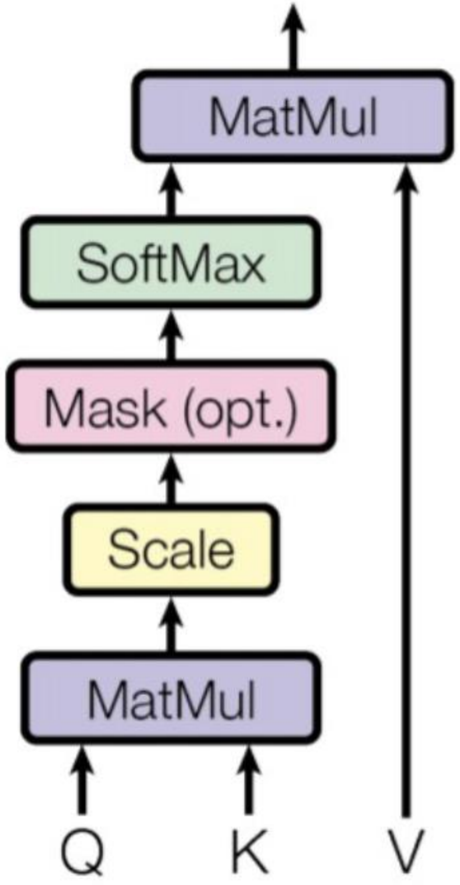
Self-Attention

RNN을 대신하여 위치정보를 준다

Self-Attention

Self-Attention

Architecture (overview)



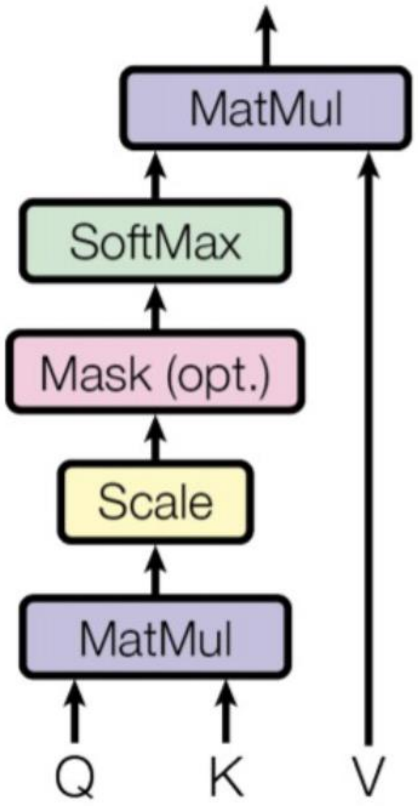
Query : 물어보는 주체

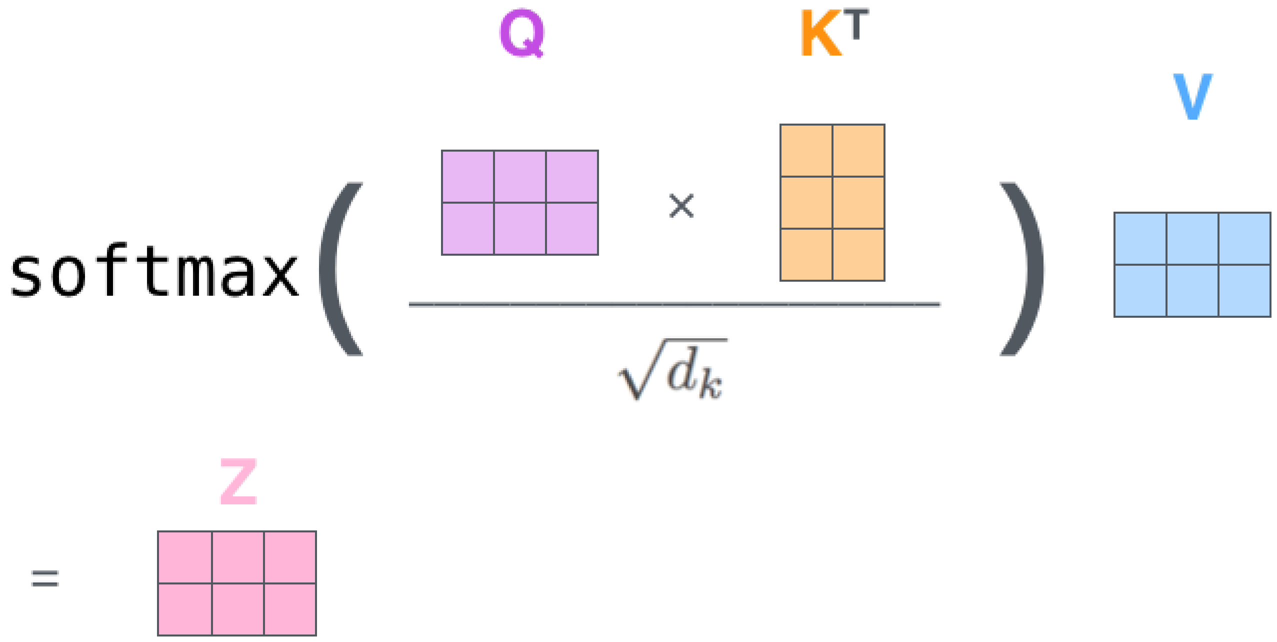
Key : 물어보는 대상

Value : key에 대한 의미적 결과

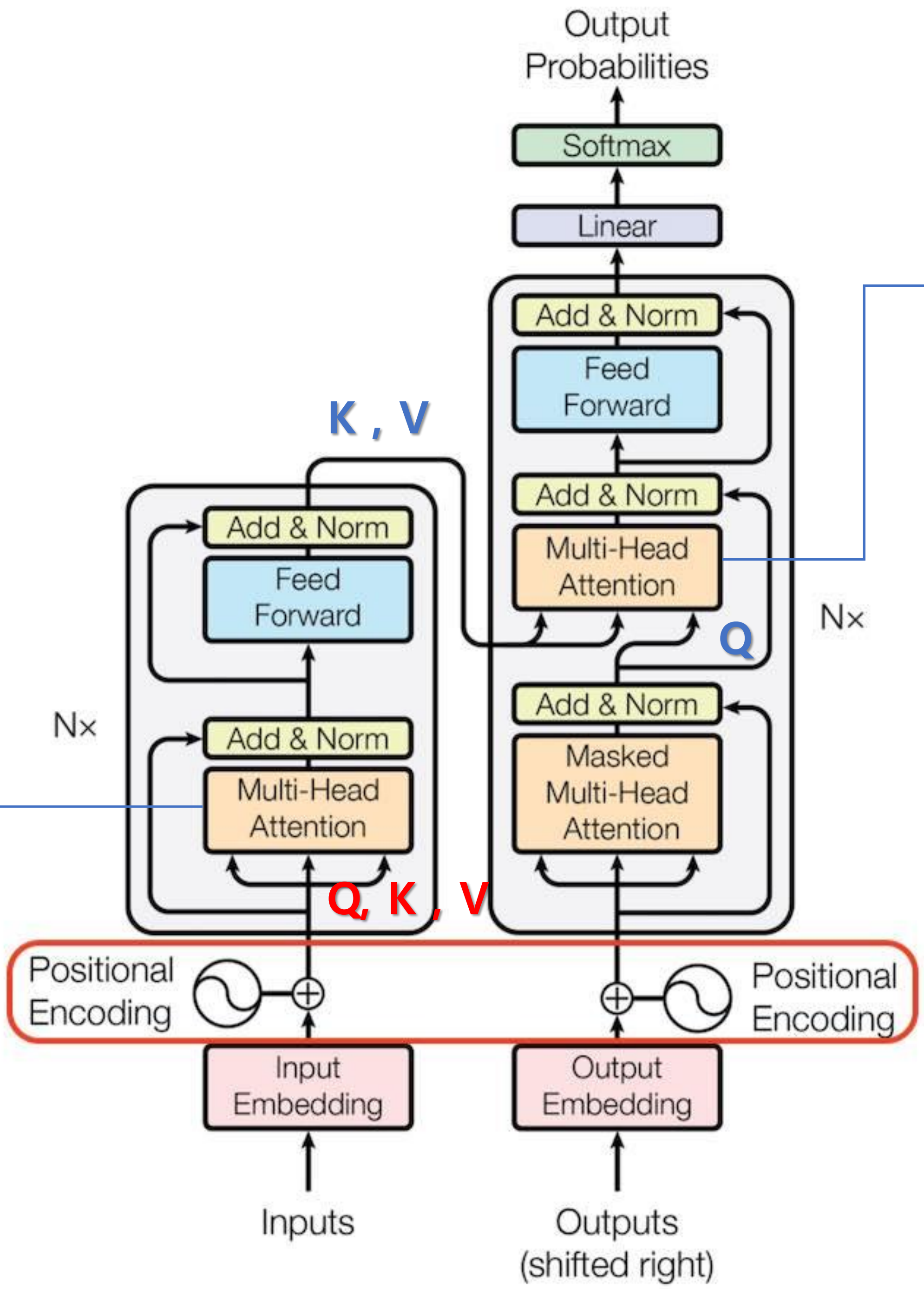
Self-Attention

Architecture (overview)





Encoder - Decoder Attention



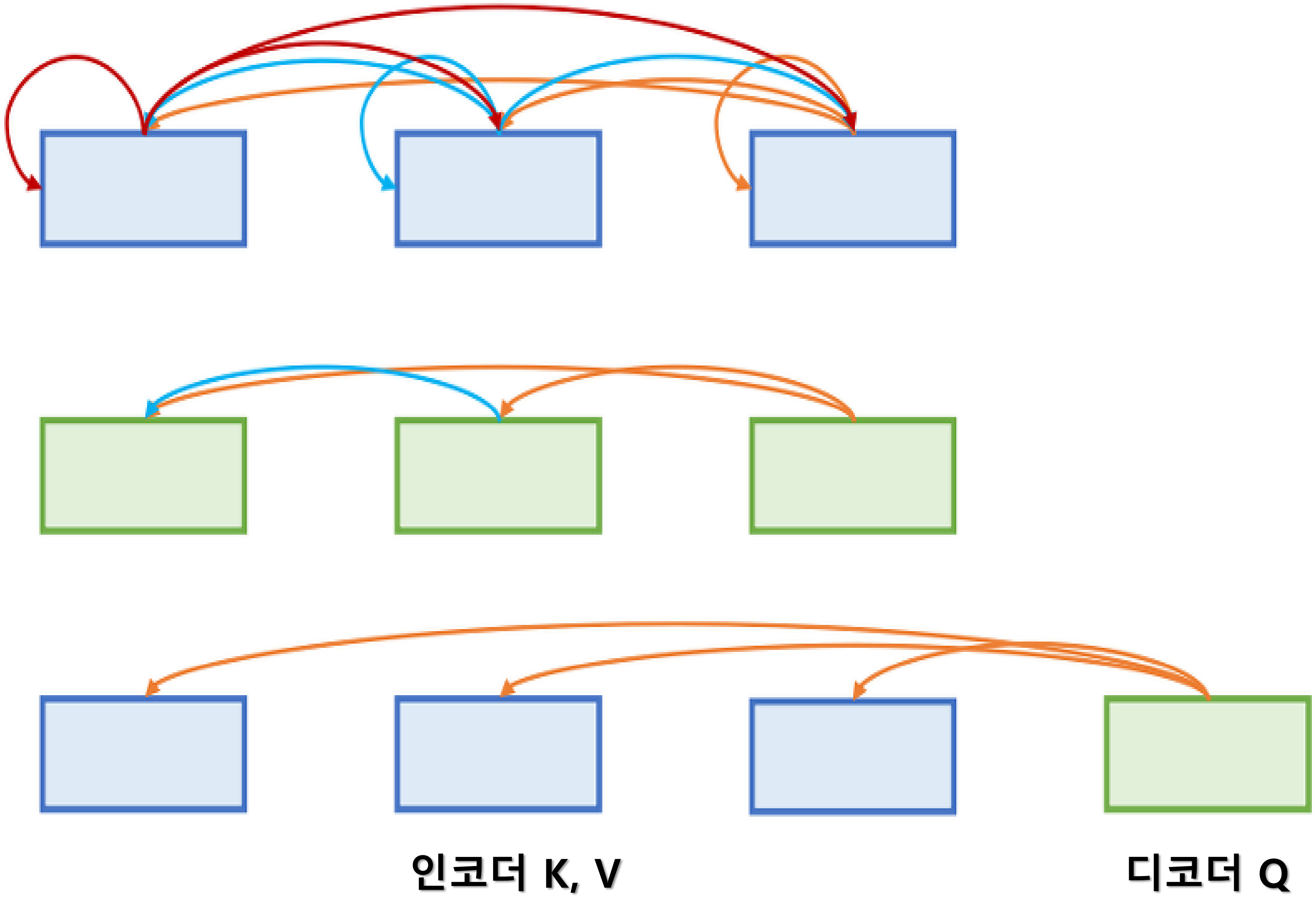
Self-Attention

Transformer

Encoder Self-Attention:

Masked Decoder Self-Attention:

Encoder-Decoder Attention:

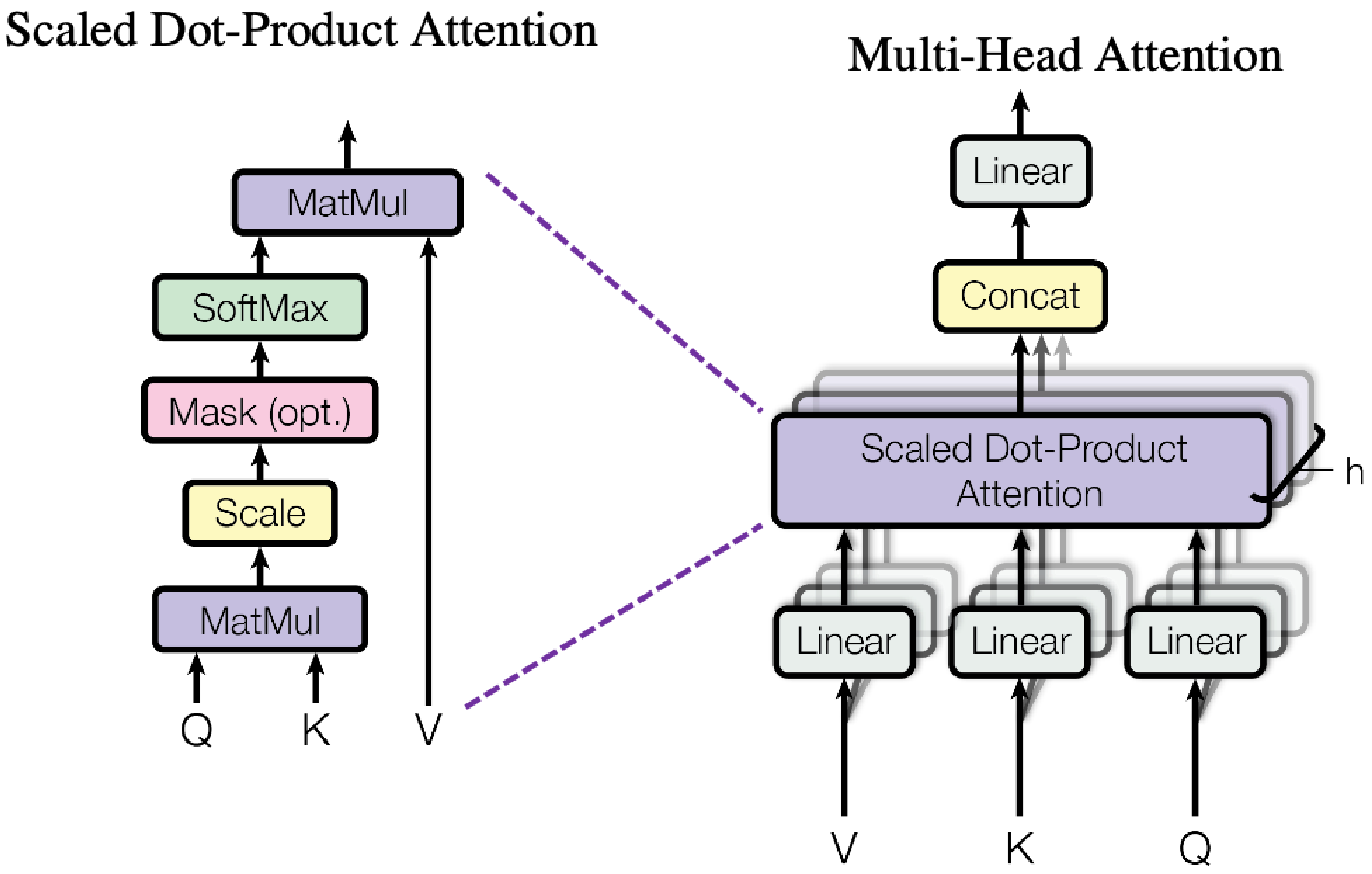


Multi-head Attention

Multi-Head Attention

Figure from paper

Muti-Head Attention을 수행한 후에도 차원(dimension)이 동일하게 유지됨



Multi-Head Attention

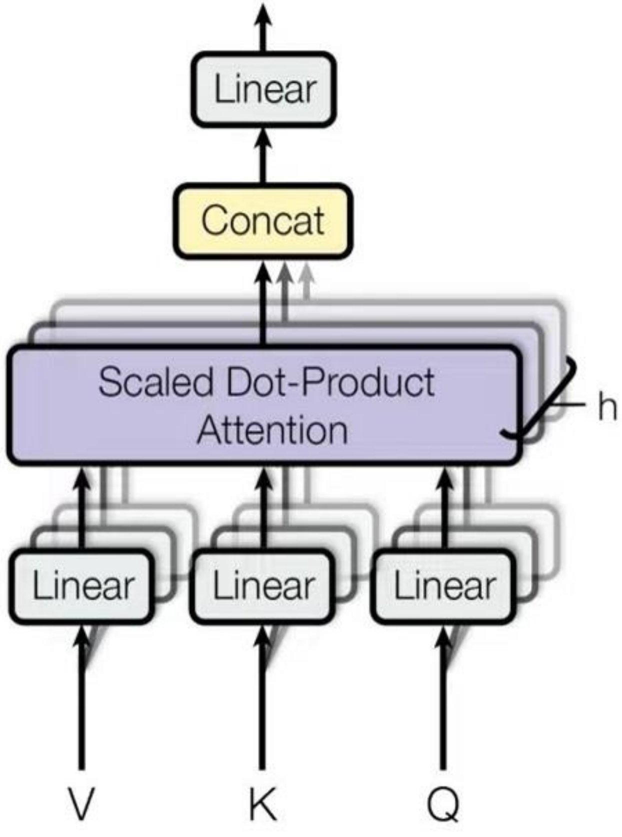
Figure from paper

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

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

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

h: 헤드(head)의 개수



Multi-Head Attention

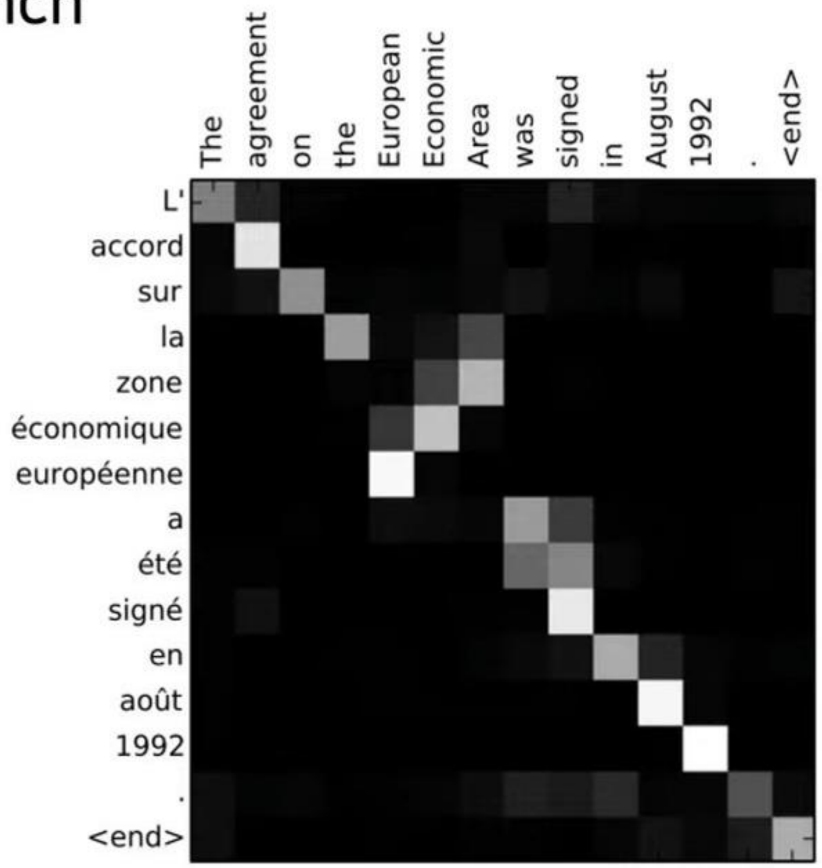
Transformer

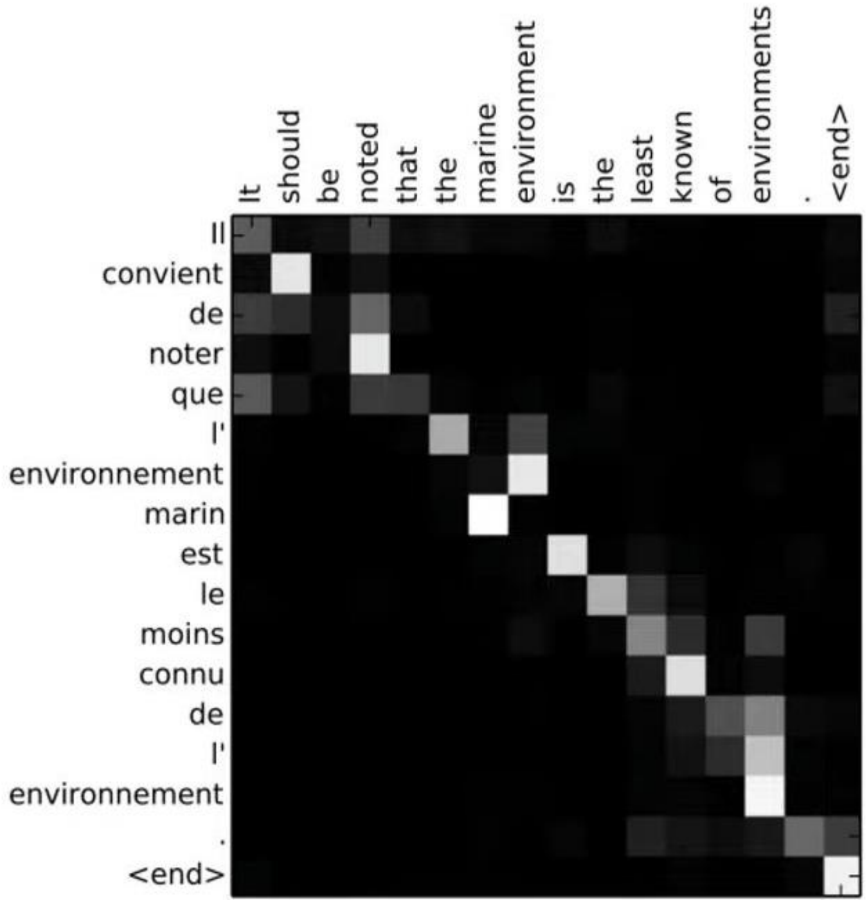
Key Point

Attention으로 문장 사이에 단어들의 연관성 정보를 알 수 있음

→ 문맥 파악 능력치 향상!

- **어텐션(attention) 가중치**를 사용해 각 출력이 어떤 입력 정보를 참고했는지 알 수 있습니다.
  - English → French





OUTPUT

```python
embed_dim = 256
latent_dim = 2048
num_heads = 8

encoder_inputs = keras.Input(shape=(None,), dtype="int64", name="encoder_inputs")
x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(encoder_inputs)
encoder_outputs = TransformerEncoder(embed_dim, latent_dim, num_heads)(x)
encoder = keras.Model(encoder_inputs, encoder_outputs)

decoder_inputs = keras.Input(shape=(None,), dtype="int64", name="decoder_inputs")
encoded_seq_inputs = keras.Input(shape=(None, embed_dim), name="decoder_state_inputs")
x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(decoder_inputs)
x = TransformerDecoder(embed_dim, latent_dim, num_heads)(x, encoded_seq_inputs)
x = layers.Dropout(0.5)(x)
decoder_outputs = layers.Dense(vocab_size, activation="softmax")(x)
decoder = keras.Model([decoder_inputs, encoded_seq_inputs], decoder_outputs)

decoder_outputs = decoder([decoder_inputs, encoder_outputs])
transformer = keras.Model(
    [encoder_inputs, decoder_inputs], decoder_outputs, name="transformer"
)
```

```
She handed him the money.
[start] ella le pasó el dinero [end]

Tom has never heard Mary sing.
[start] tom nunca ha oído cantar a mary [end]

Perhaps she will come tomorrow.
[start] tal vez ella vendrá mañana [end]

I love to write.
[start] me encanta escribir [end]

His French is improving little by little.
[start] su francés va a [UNK] sólo un poco [end]

My hotel told me to call you.
[start] mi hotel me dijo que te [UNK] [end]
```

Q & A