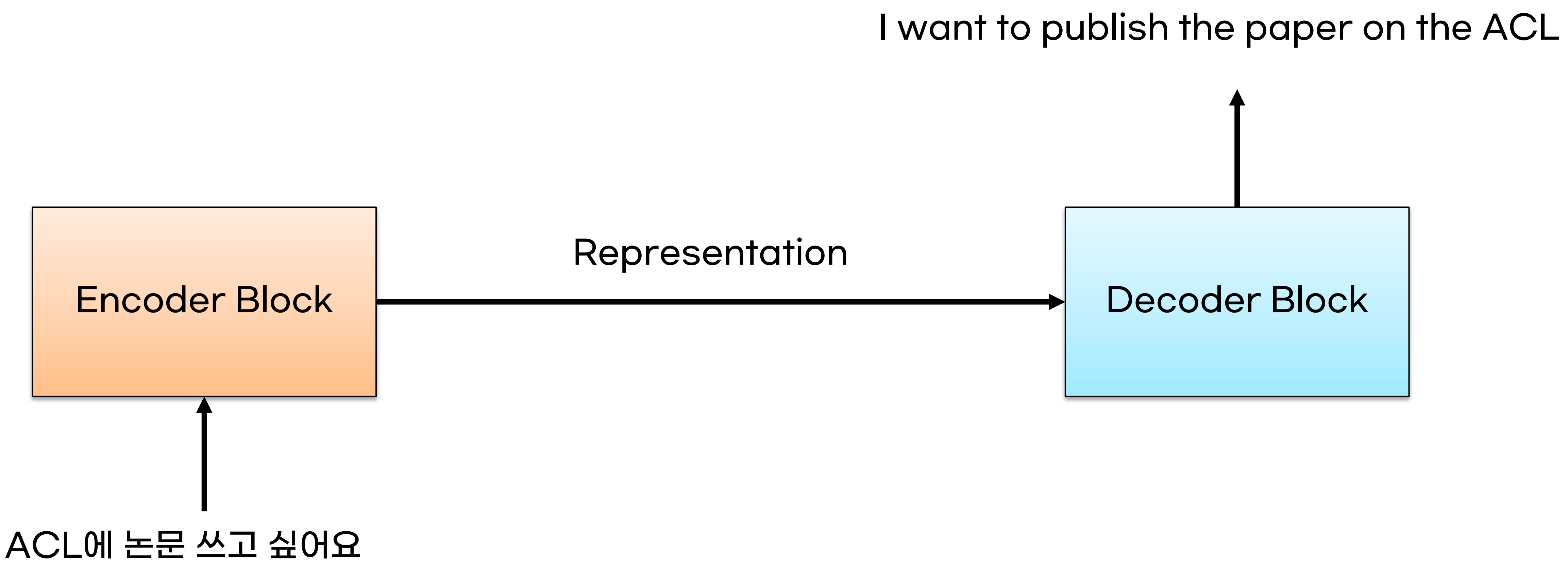
Transformer Decoder

MLP Lab.

Contents

01 Encoder-Decoder Structure

- Transformer Encoder-Decoder Architecture: Without Bottleneck
  - ✓ The size of input and output embedding are the same size
  - ✓ Since it encodes via self-attention, it does not seem to require a bottleneck.



02 Transformer Structure

- Tokenize the input sentence through a tokenizer, convert it to a token embedding, and combine the position embedding and token embedding to inject information about the position of the token into the encoder.

- Encoders and decoders consist of a stack of encoder layers and decoder layers, similar to the stack of convolutional layers in computer vision.

- The output of the encoder is injected into each decoder layer, and the decoder predicts the next most likely token in the sequence, predicts the <EOS> token, or repeats until the maximum length is reached.

- The transformer architecture is designed for seq2seq operations, but the encoder and decoder blocks are independent models.

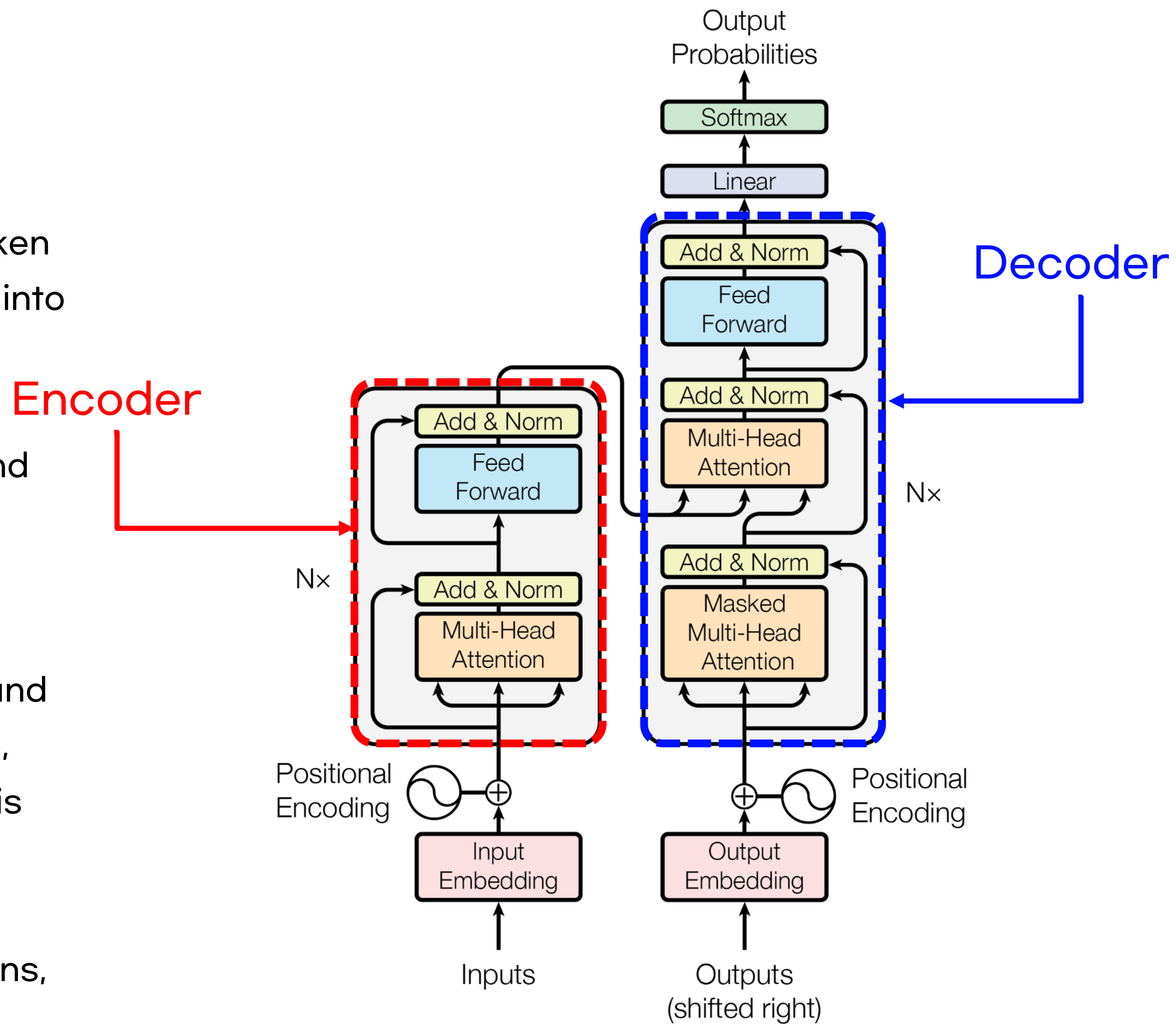


Figure 1: The Transformer - model architecture.

03 Usage of Encoder-Decoder

- Transformer Encoder-Decoder Architecture.

Encoder Block

➢ Encoder block : BERT Series Models for QA, text classification

Decoder Block

➢ Decoder block : GPT-like models for Chatbot, NLG, QA

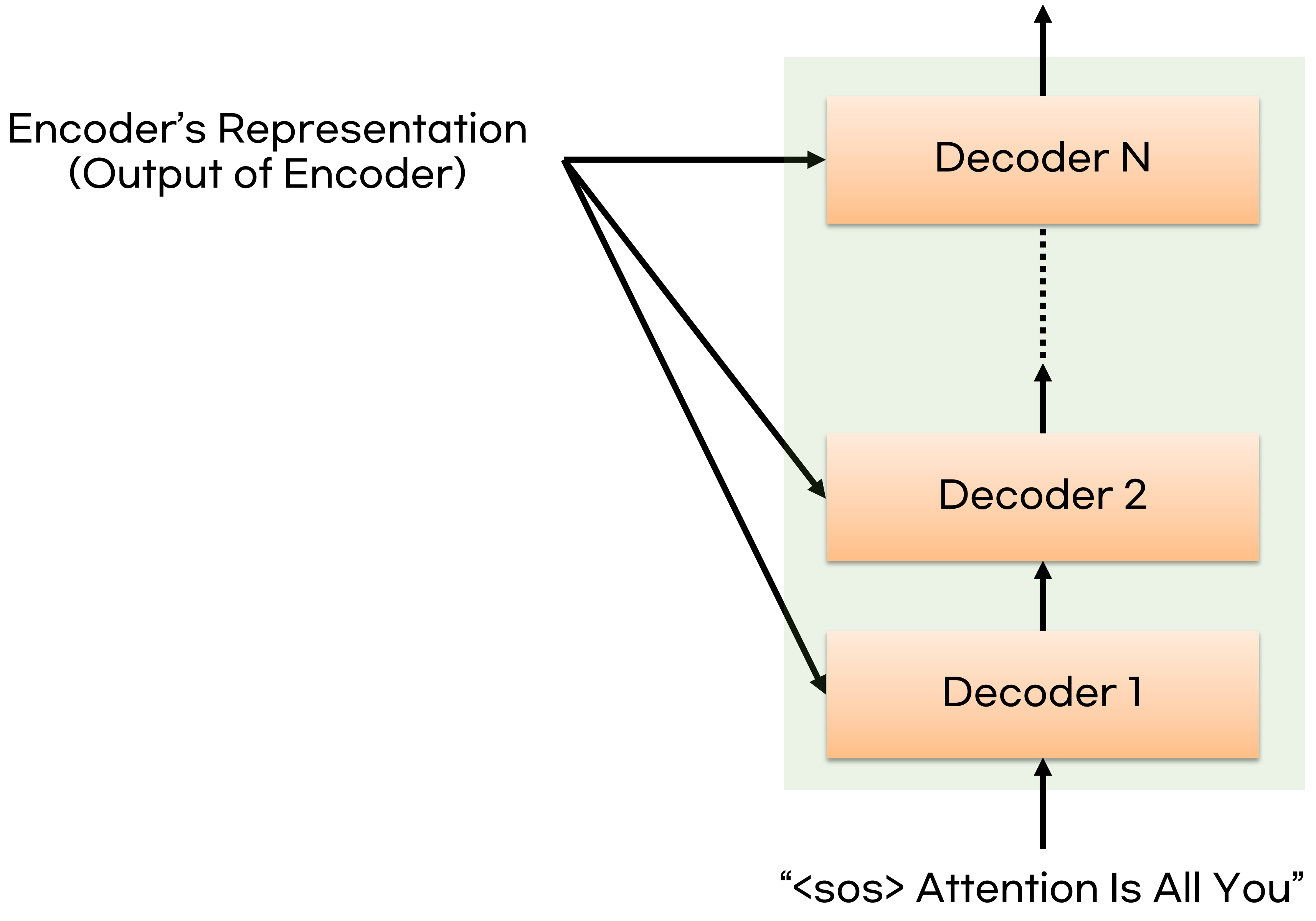
Encoder Block + Decoder Block

➢ Encoder block + Decoder block : BART, T5 for Machine Translation and Summarization

Contents

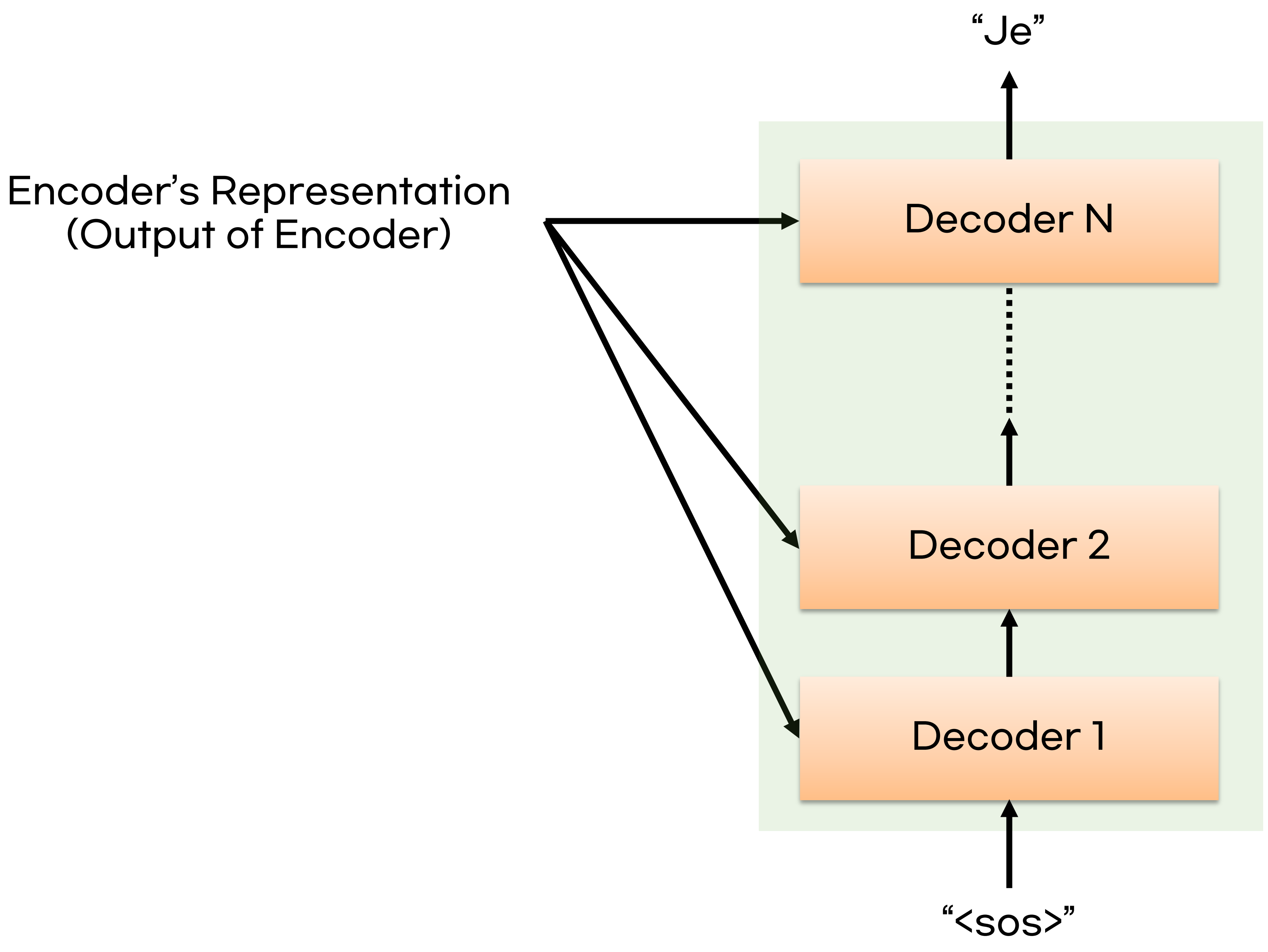
00 Transformer Decoder

- Use N decoders as a stacked structure
- The Decoder takes as input the output of the previous Decoder and the representation of the Encoder.



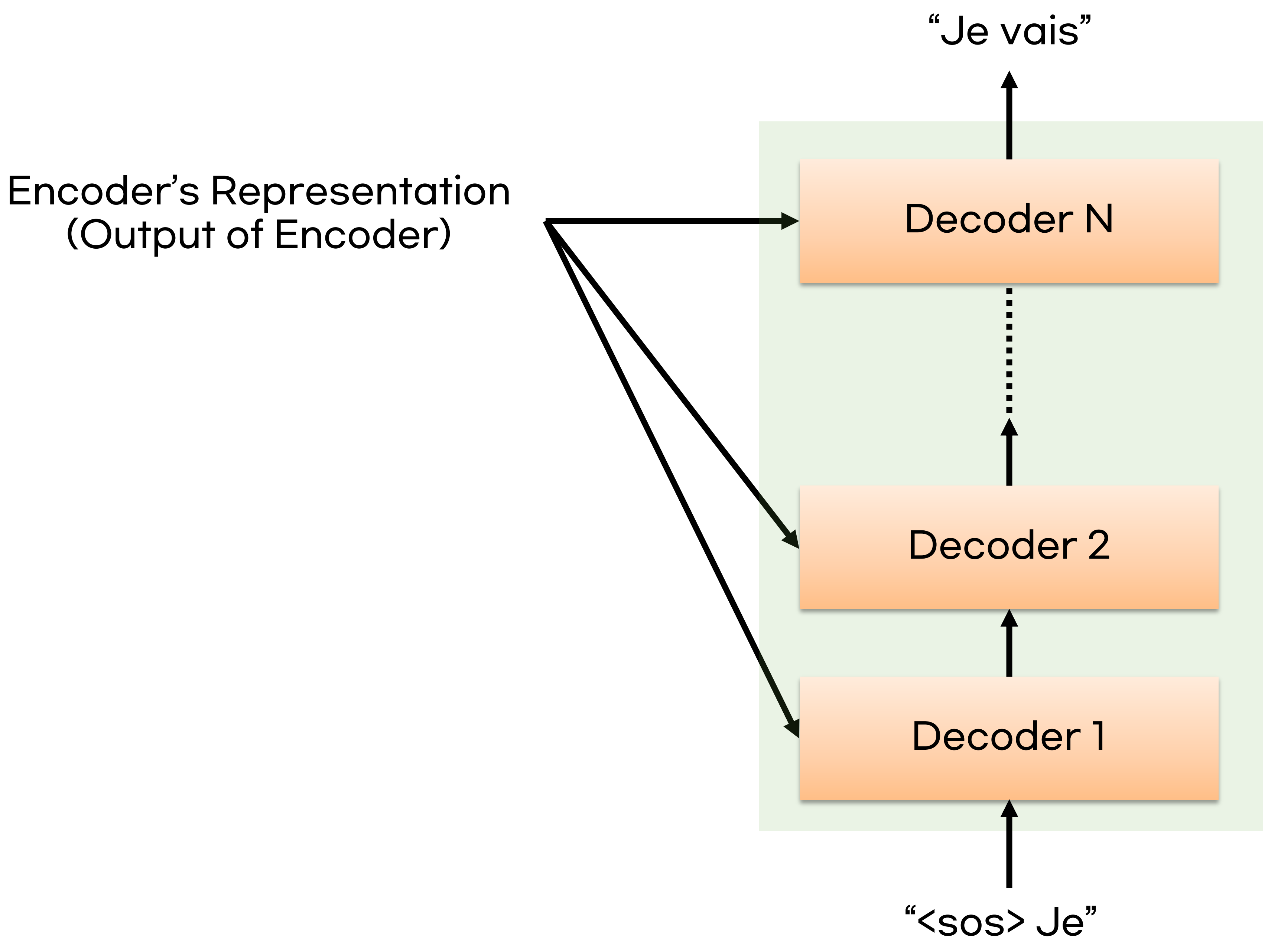
00 Transformer Decoder

- How the Decoder Works?
- Example of generating the French sentence "Je vais bien" from the English sentence "I am good".



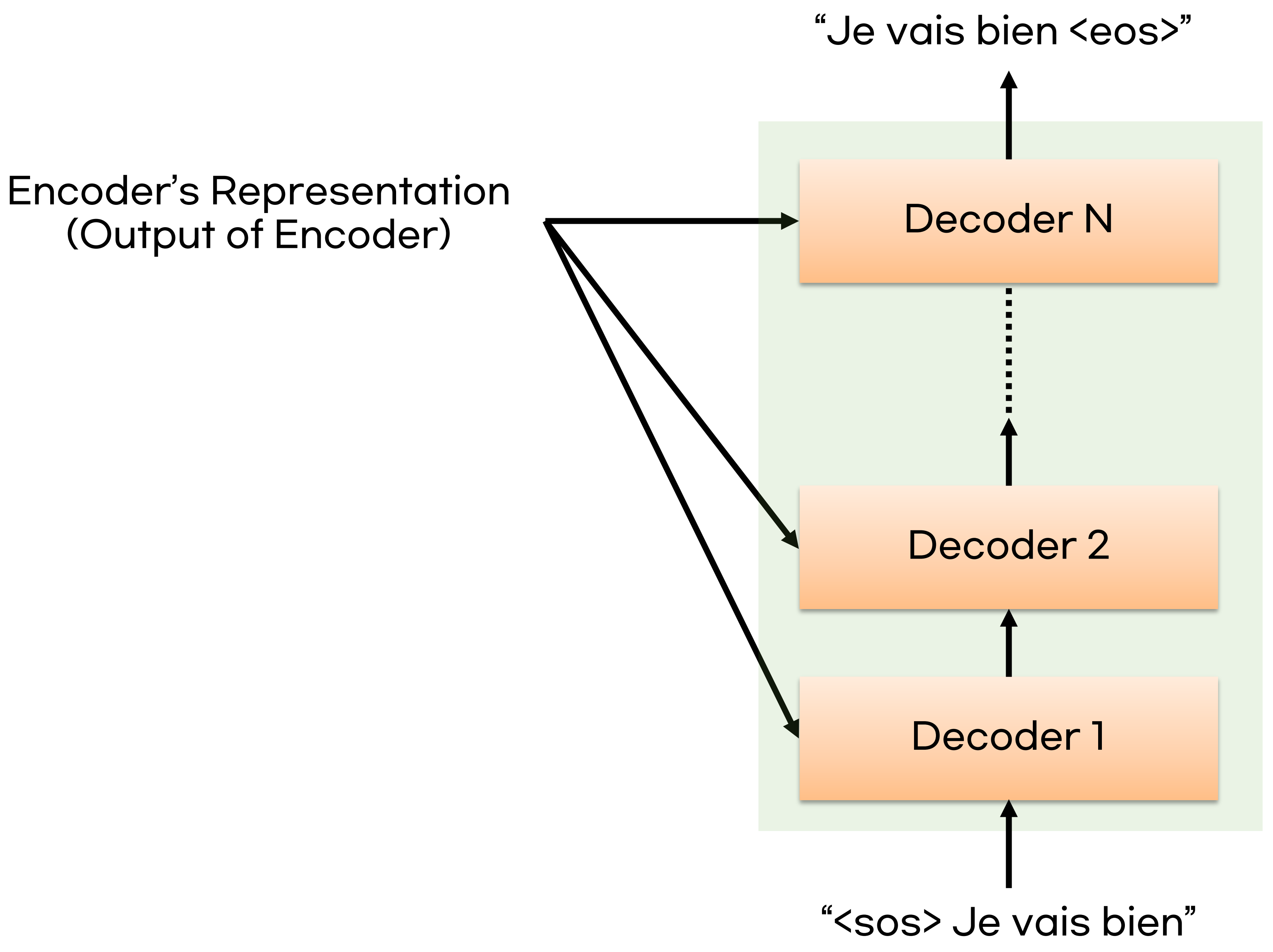
00 Transformer Decoder

- How the Decoder Works?
- Example of generating the French sentence "Je vais bien" from the English sentence "I am good".



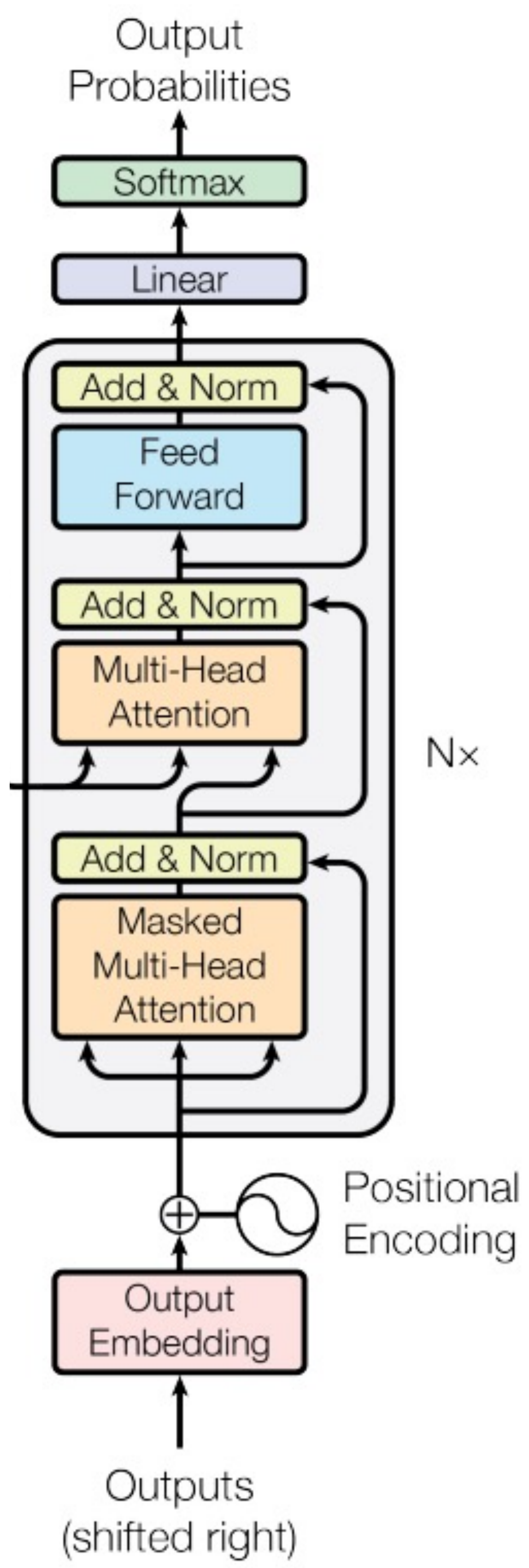
00 Transformer Decoder

- How the Decoder Works?
- Example of generating the French sentence "Je vais bien" from the English sentence "I am good".



00 Transformer Decoder

- Transformer Decoder Components
  - ✓ Embedding layer (Word, Position)
  - ✓ Masked Multi-Head Attention
  - ✓ Multi-Head Attention
  - ✓ Feed-Forward
  - ✓ Add & Norm

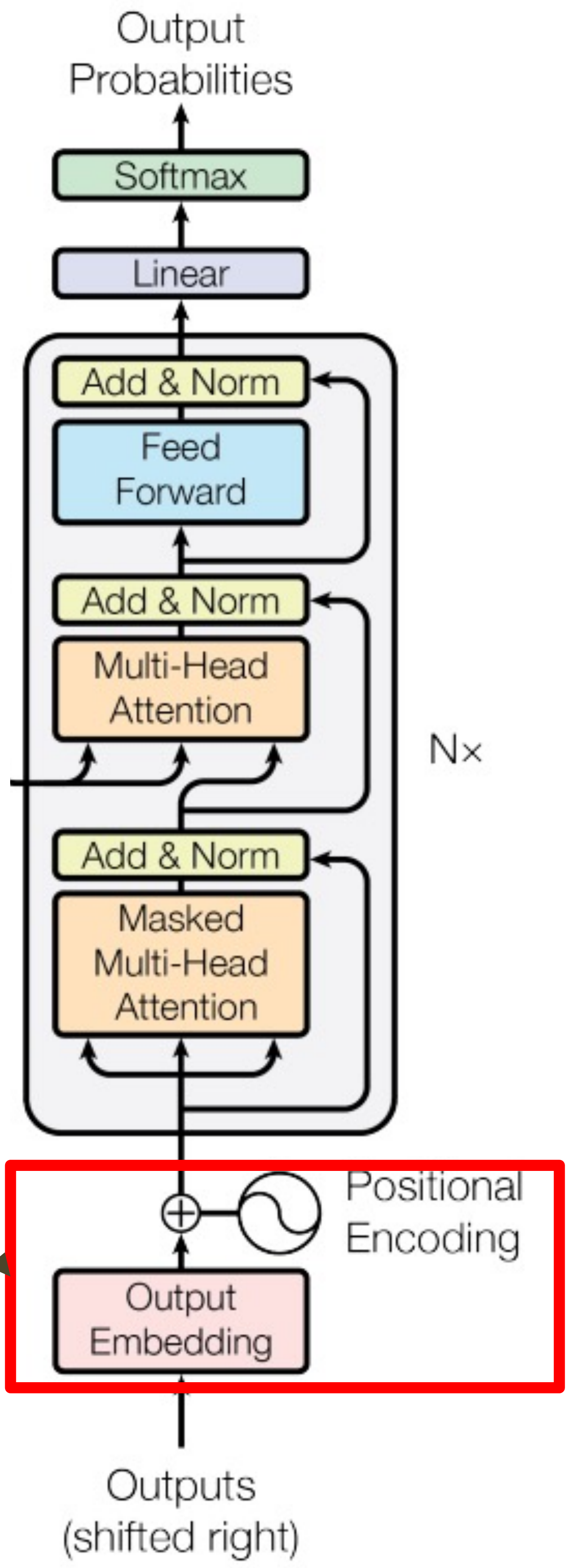


Transformer Decoder Architecture

00 Transformer Decoder

- Transformer Decoder Components
  - ✓ Embedding layer (Word, Position)
  - ✓ Masked Multi-Head Attention
  - ✓ Multi-Head Attention
  - ✓ Feed-Forward
  - ✓ Add & Norm

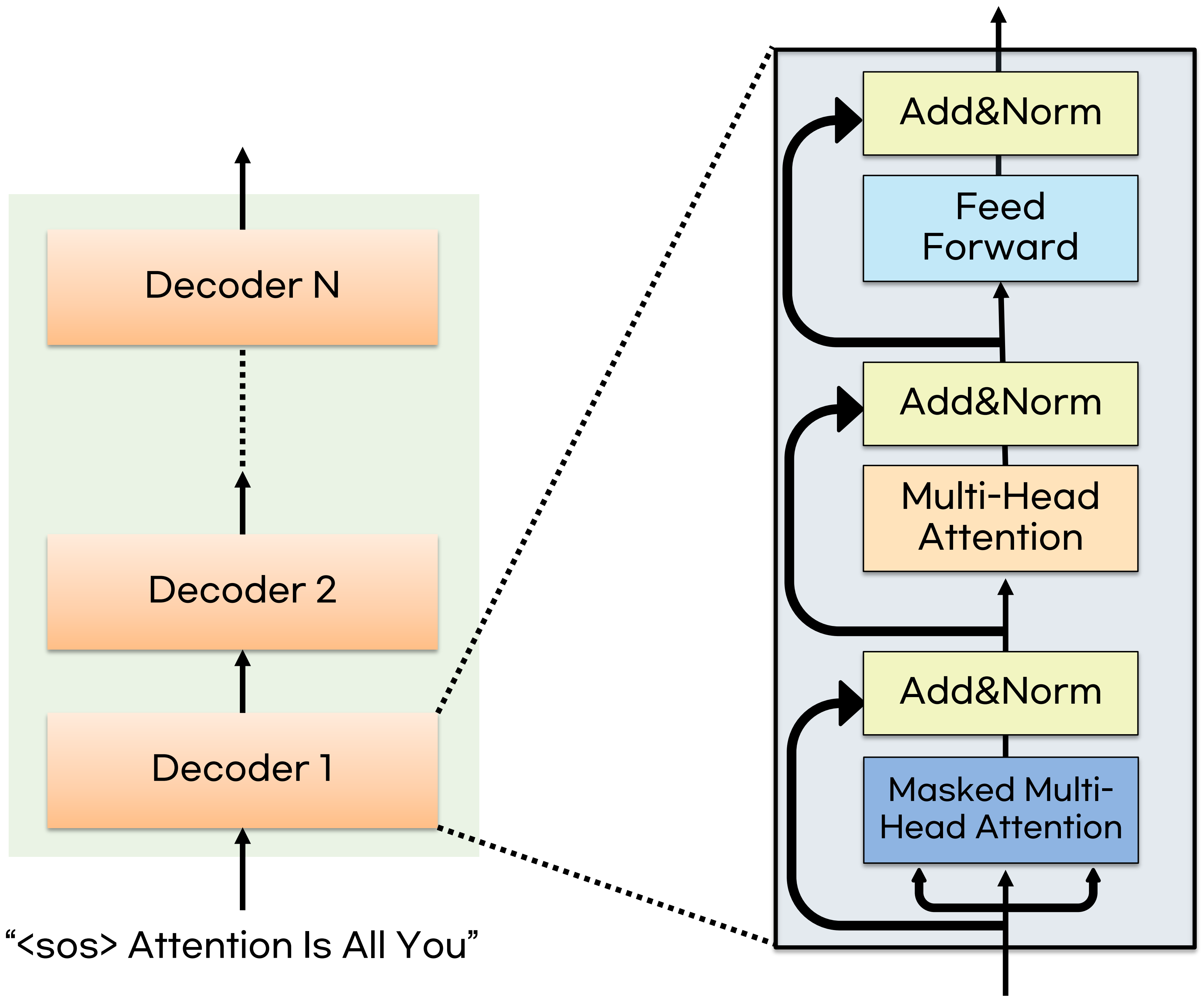
Those parts are same with the Encoder



Transformer Decoder Architecture

00 Transformer Decoder

- Transformer Decoder Components
  - ✓ Embedding layer (Word, Position)
  - ✓ Masked Multi-Head Attention
  - ✓ Multi-Head Attention
  - ✓ Feed-Forward
  - ✓ Add & Norm



Transformer Decoder Architecture

01 Masked Multi-Head Attention

Introduction of Masked Multi-Head Attention

- When the decoder generates a sentence, it only puts the words generated in the previous step into the input sentence.

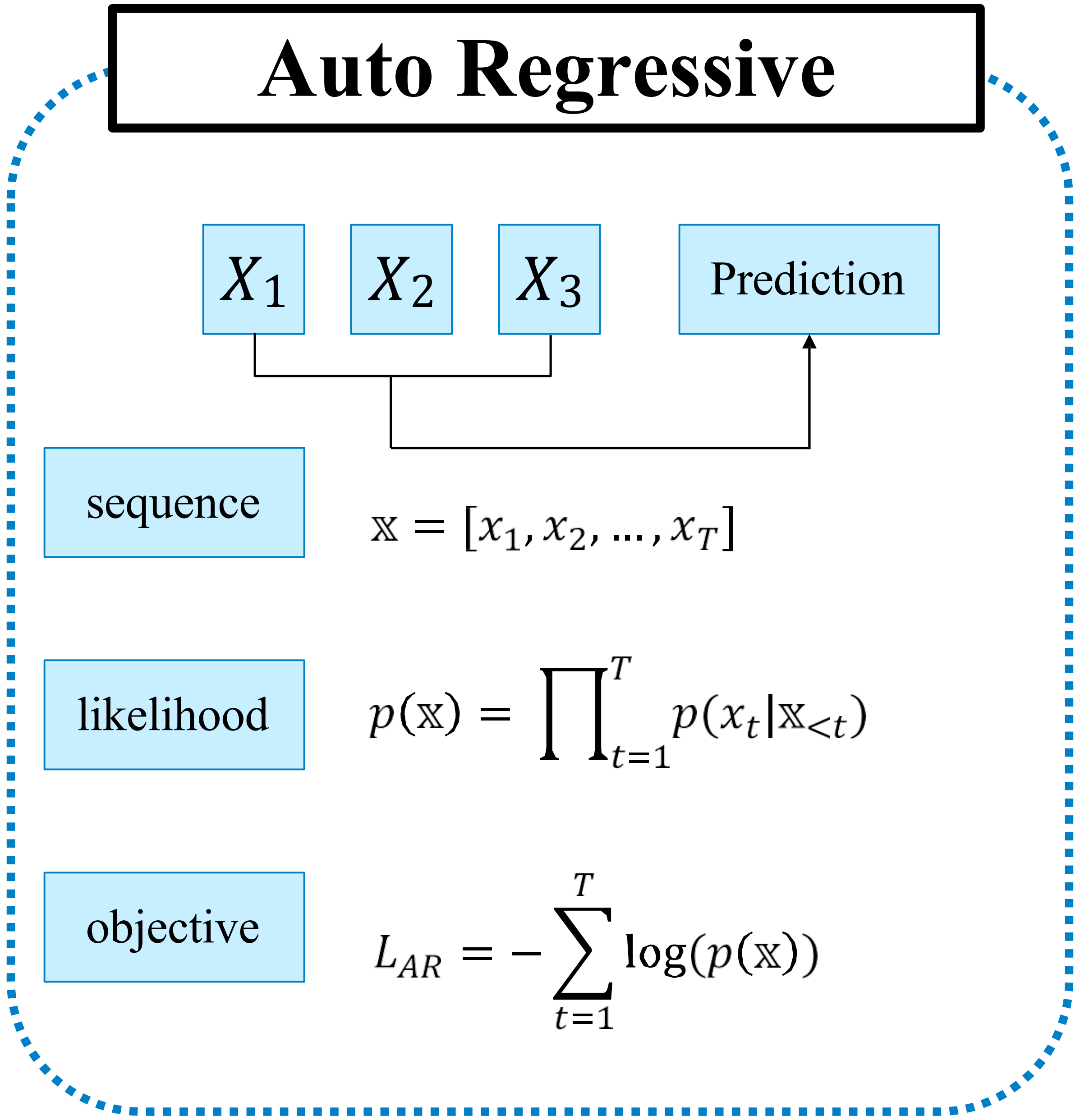
  - ex) input : [<sos>, ‘Je’], output : [‘Je’, ‘vais’]

- Decoder Characteristics + self-attention = Masked Multi-Head Attention

  - the characteristic is about **Auto-Regressive**
  - **Auto-Regressive** : Predicting the next token based on all previous tokens

01 Masked Multi-Head Attention

- **Auto-Regressive :** Predicting the next token based on all previous tokens



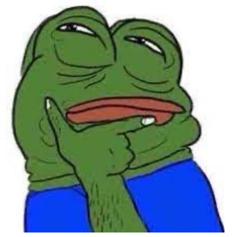
Sentence: I have a dream

- <SOS> I ___
- <SOS> I have ___
- <SOS> I have a ___

01 Masked Multi-Head Attention

Can we compute multi-head attention parallelly in the Auto-regressive model?

Masking the un sheet next to you! == masked multi-head attention



<sos>	Je	vais	bien
<sos>	Je	vais	bien
<sos>	Je	vais	bien
<sos>	Je	vais	bien

→

<sos>	mask	mask	mask
<sos>	Je	mask	mask
<sos>	Je	vais	mask
<sos>	Je	vais	bien

01 Masked Multi-Head Attention

Masked Multi-Head Attention - Process

1. Extract embeddings from a sentence: create an embeddings matrix
2. Create Key, Query, Value from Embedded Matrix
3. Calculate the similarity between Query and Key : $Q \cdot K^T$

Those parts are same with the Encoder

4. Scaling and Softmax (normalization) → score matrix : $softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}_{mask}\right)$
5. Multiply by Value to obtain Attachment matrix M

01 Masked Multi-Head Attention

Masked Multi-Head Attention - Process

1. Extract embeddings from a sentence: create an embeddings matrix
2. Create Key, Query, Value from Embedded Matrix
3. Calculate the similarity between Query and Key : $Q \cdot K^T$
4. Scaling and Softmax (normalization) → score matrix : $softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}_{mask}\right)$
5. Multiply by Value to obtain Attachment matrix M

Those parts are **same** with the **Encoder**

01 Masked Multi-Head Attention

4. Scaling and Softmax (normalization)

	<sos>	Je	vais	bien
<sos>	9.125	7.5	1.25	5.625
Je	5.0	12.37	3.12	8.75
vais	7.25	5.0	10.37	1.25
bien	1.5	1.37	1.87	10.0

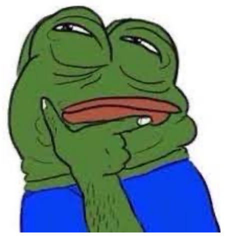
$$\frac{QK^T}{\sqrt{d_k}}$$

	<sos>	Je	vais	bien
<sos>	9.125	-inf	-inf	-inf
Je	5.0	12.37	-inf	-inf
vais	7.25	5.0	10.37	-inf
bien	1.5	1.37	1.87	10.0

$$\frac{Q \cdot K^T}{\sqrt{d_k}}_{mask}$$

01 Masked Multi-Head Attention

- Why mask with -inf?



  - $exp(-\inf) = 0$
  - $softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}_{mask}\right) =$

	<sos>	Je	vais	bien
<sos>	1	0	0	0
Je	0.37	0.62	0	0
vais	0.26	0.31	0.43	0
bien	0.21	0.26	0.26	0.27

01 Masked Multi-Head Attention

Masked Multi-Head Attention - Process

1. Extract embeddings from a sentence: create an embeddings matrix
2. Create Key, Query, Value from Embedded Matrix
3. Calculate the similarity between Query and Key : $Q \cdot K^T$

Those parts are same with the Encoder

4. Scaling and Softmax (normalization) → score matrix : $softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}_{mask}\right)$
5. Multiply by Value to obtain Attachment matrix M

01 Masked Multi-Head Attention (code)

- Get the score matrix in the same way as the encoder and replace the upper triangular matrix with -inf

  Masking effect because the value is zero after passing Softmax

```python
def scaled_dot_product_attention(query, key, value, mask=None):
    dim_k = query.size(-1)
    scores = torch.bmm(query, key.transpose(1, 2)) / sqrt(dim_k)
    if mask is not None:
        scores = scores.masked_fill(mask == 0, float("-inf"))
    weights = F.softmax(scores, dim=-1)
    return weights.bmm(value)
```

✓ 0.0s

mask is a half-triangular matrix

$$
\begin{matrix} 1 & 0 & 0 & \cdots \\ 1 & 1 & 0 & \\ 1 & 1 & 1 & \\ \vdots & & & \ddots \end{matrix}
$$

	<sos>	Je	vais	bien
<sos>	45.2	20.1	18.3	17.8
Je	22.8	63.2	7.3	16.3
vais	22.3	41.2	77.2	12.2
bien	12.3	23.9	44.2	84.2

$$\frac{Q \cdot K^T}{\sqrt{d_k}}$$

	<sos>	Je	vais	bien
<sos>	45.2	-∞	-∞	-∞
Je	22.8	63.2	-∞	-∞
vais	22.3	41.2	77.2	-∞
bien	12.3	23.9	44.2	84.2

$$\frac{Q \cdot K^T}{\sqrt{d_k}}$$

	<sos>	Je	vais	bien
<sos>	1	0	0	0
Je	0.03	0.97	0	0
vais	0.15	0.25	0.96	0
bien	0.01	0.01	0.01	0.97

$$softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right)$$

01 Masked Multi-Head Attention

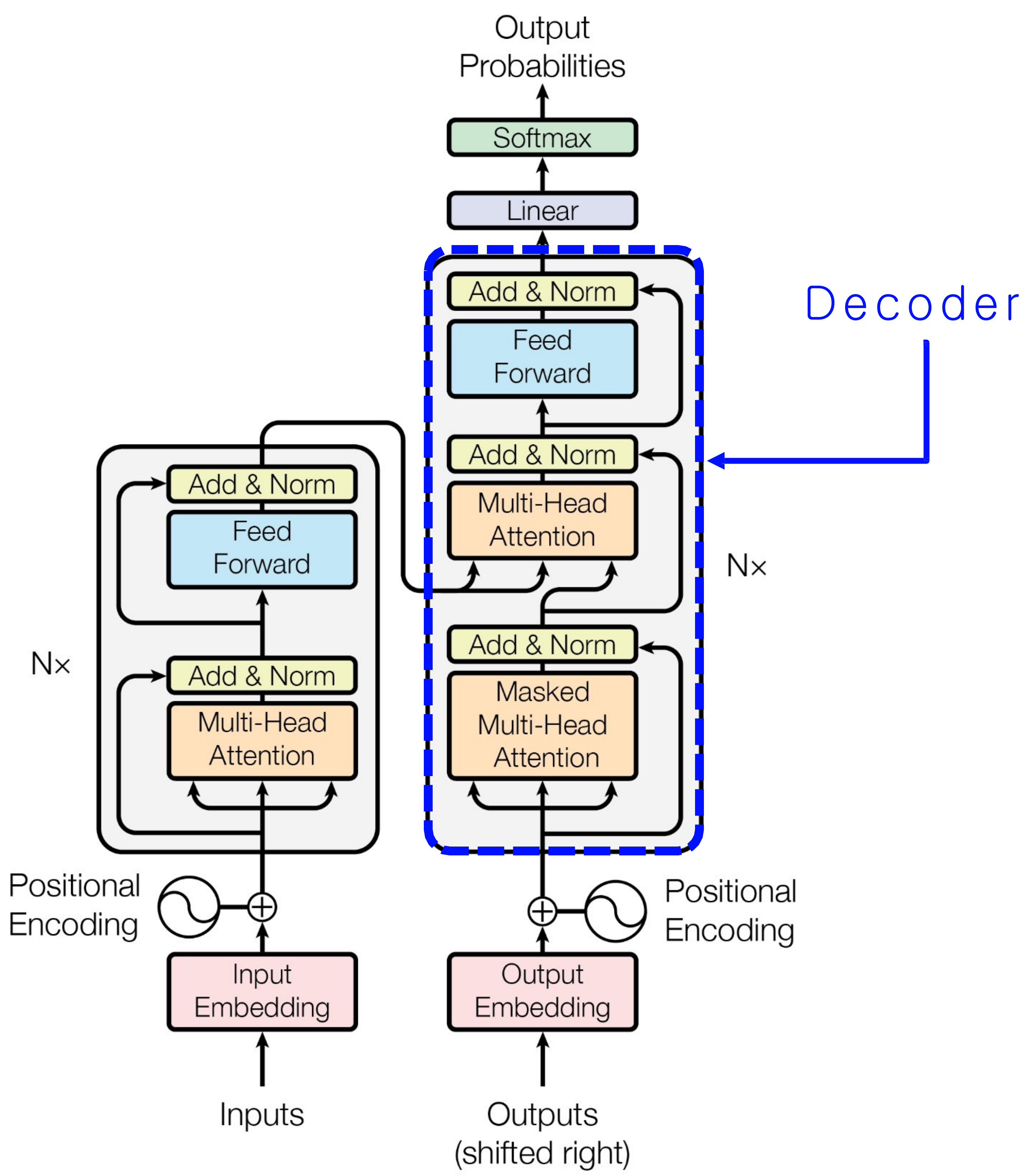
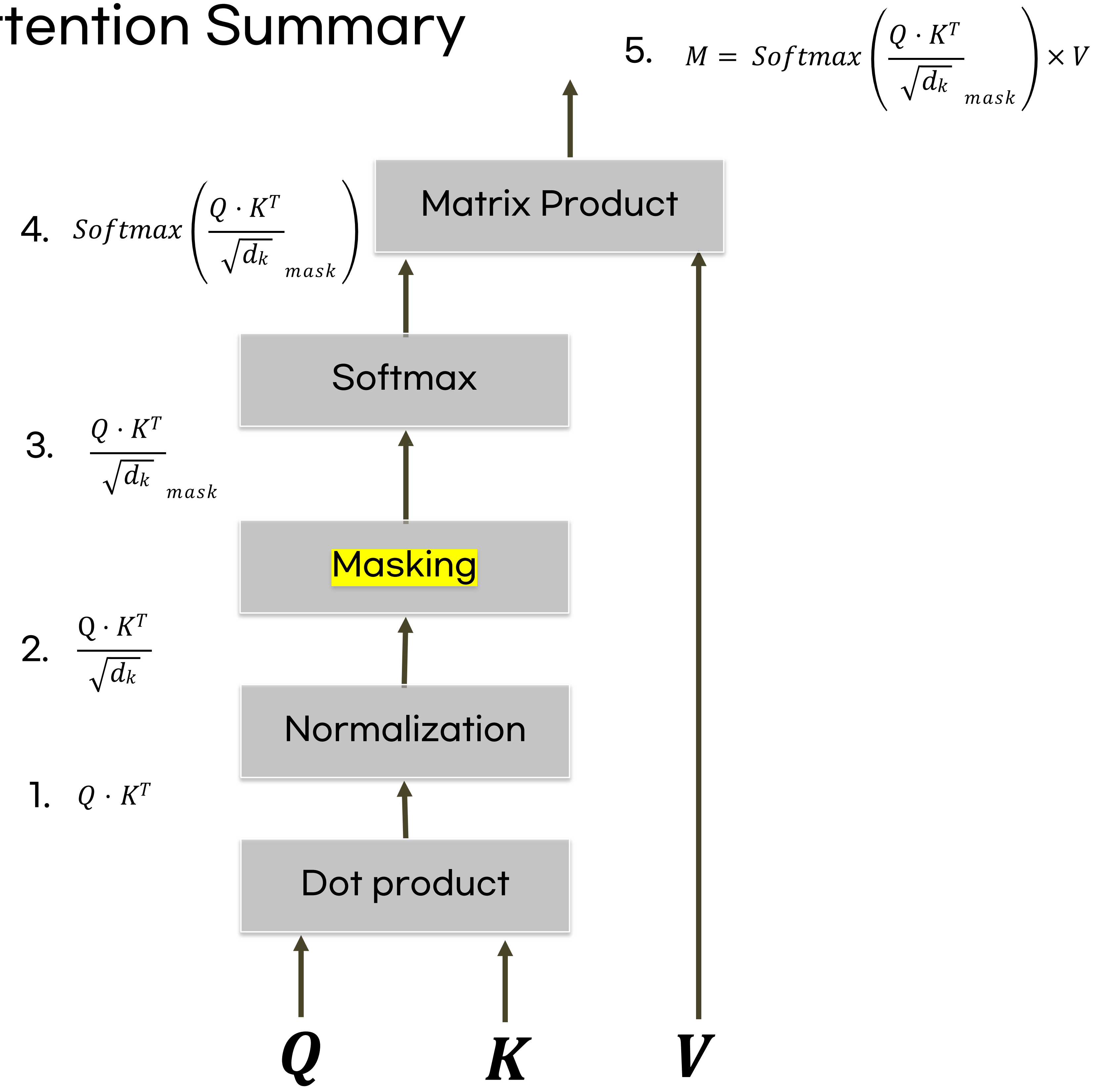


Figure 1: The Transformer - model architecture.

01 Masked Multi-Head Attention

Masked Multi-Head Attention Summary



01 Masked Multi-Head Attention (Example)

- Assuming we have two Multi-Head Attention, then the calculation is as follows

1. Compute attention heads M_1, M_2

1) Set W_1^Q, W_1^K, W_1^V and compute Q_1, K_1, V_1. Then get $M_1 = \text{softmax}\left(\frac{Q_1 K_1^T}{\sqrt{d_k}}\right) V_1$

2) Set W_2^Q, W_2^K, W_2^V and compute Q_2, K_2, V_2. Then get $M_2 = \text{softmax}\left(\frac{Q_2 K_2^T}{\sqrt{d_k}}\right) V_2$

2. Concatenate the obtained Attention Heads and use the Linear Projection

$$\text{concatenate}([M_1, M_2])W$$

02 Multi-Head Attention

Introduction of Multi-Head Attention for Decoder

- Input
  - Output of Masked Multi-Head Attention (Matrix M)
  - Output of Encoder

- Interaction between the results of the encoder and the results of the decoder

  - Called the Encoder-Decoder Attention Layer

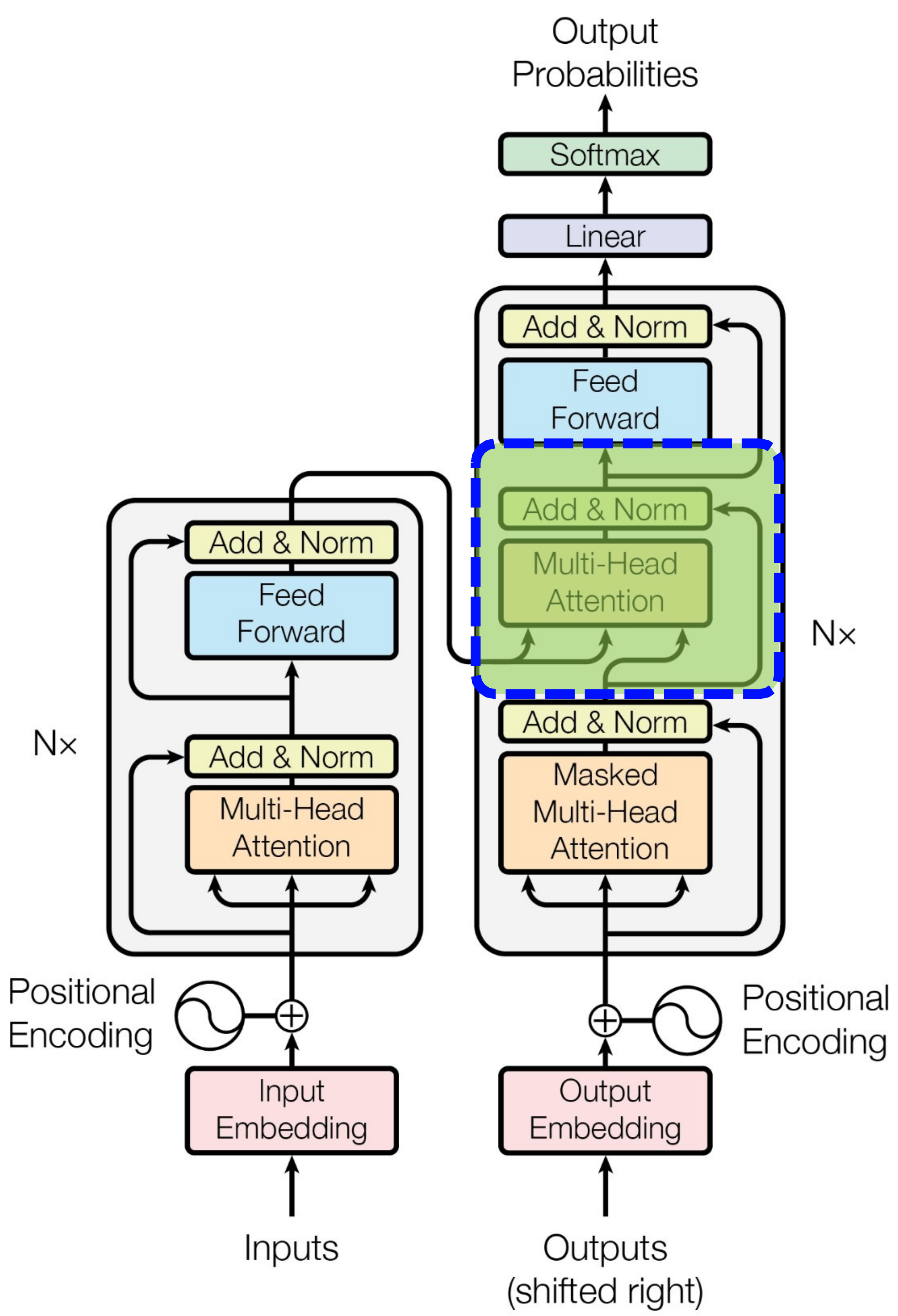
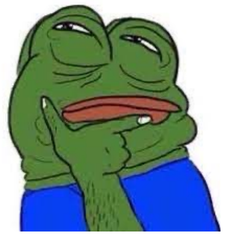


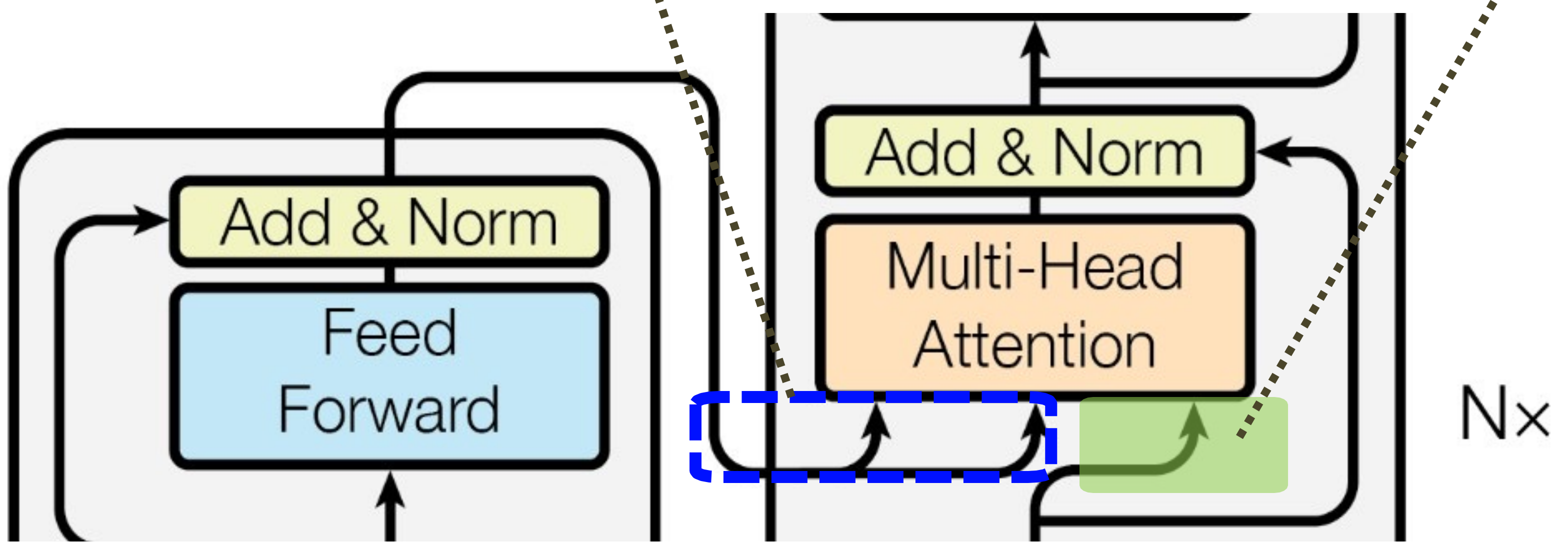
Figure 1: The Transformer - model architecture.

02 Multi-Head Attention

How do encoder results interact with decoder results? 

→ Generate the query matrix Q using output of decoder (Attention matrix M)

→ Generate a Key, Value matrix (K, V) using the result of the encoder (representation R)



02 Multi-Head Attention

Multi-Head Attention(in Decoder) - Process

1. Generate Query from Attention Matrix (M from Masked), Key and Value from Encoder Representation (R)
2. Calculate the similarity between a query and a key : QK^T
3. Scaling & Softmax -> score matrix : $softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right)$
4. Multiply by Value to get the Attention matrix Z

02 Multi-Head Attention

Multi-Head Attention(in Decoder) - Process

1. Generate Query from Attention Matrix (M from Masked), Key and Value from Encoder Representation (R)

2. Calculate the similarity between a query and a key : QK^T
3. Scaling & Softmax -> score matrix : $softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right)$

4. Multiply by Value to get the Attention matrix Z

02 Multi-Head Attention

1. Generate Query from Attention Matrix (M from Masked), Key and Value from Encoder Representation (R)

- For generating Query(Q) matrix, set W^Q

$$where\ W^Q \in \mathbb{M}^{d\_emb \times d_k}$$ are trainable **parameters**

- Generate Q by dot-product between W^Q and Attention matrix M

$$Q = MW^Q$$

$where\ M$ is Attention matrix from masked multi head attention

02 Multi-Head Attention

1. Generate Query from Attention Matrix (M from Masked), Key and Value from Encoder Representation (R)

- For generating Key (K) and Value(V), set W^K, W^V

$$where\ W^K, W^V \in \mathbb{M}^{d\_emb \times d_k}\ are\ \text{trainable}\ \mathbf{parameters}$$

- Generate K and V by dot-product as follows:

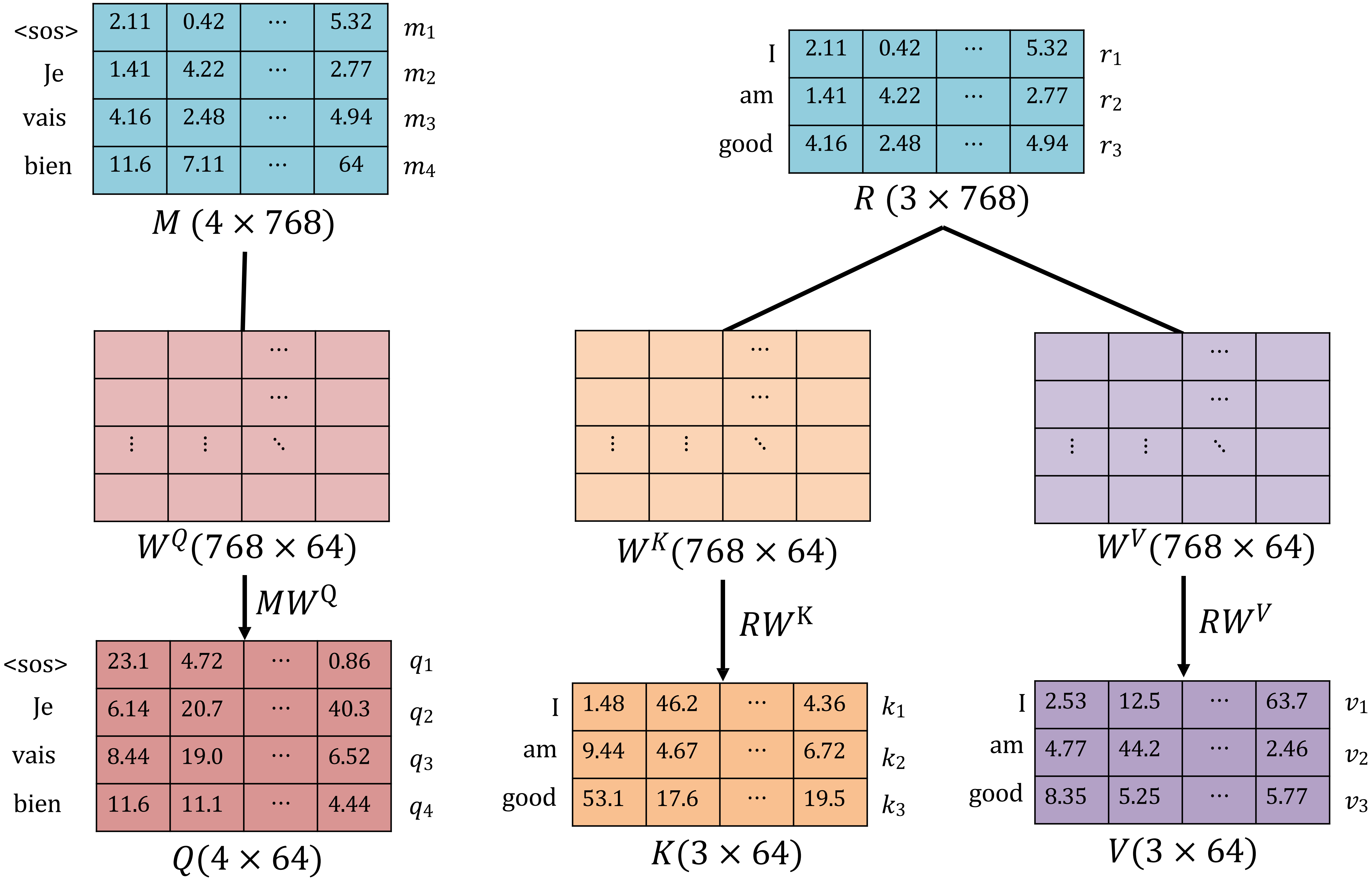
$$K = RW^k$$

$$V = RW^V$$

$$where\ R\ is\ representation\ matrix\ from\ encoder$$

02 Multi-Head Attention

1. Generate Query from Attention Matrix (M from Masked), Key and Value from Encoder output



$R, M \in \mathbb{M}^{n\_tok \times d\_emb}$

$W^K, W^Q, W^V \in \mathbb{M}^{d\_emb \times d_k}$

$Q, K, V \in \mathbb{M}^{n\_tok \times d_k}$

02 Multi-Head Attention

Multi-Head Attention(in Decoder) - Process

1. Generate Query from Attention Matrix (M from Masked), Key and Value from Encoder Representation (R)

2. Calculate the similarity between a query and a key : QK^T
3. Scaling & Softmax -> score matrix : $softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right)$

4. Multiply by Value to get the Attention matrix Z

02 Multi-Head Attention

2. Calculate the similarity between a query and a key : QK^T

- Performing the inner product of the Query and Key matrices
- Compute the inner product of two matrices to get the similarity for each token

$$
\begin{array}{r}
\text{<sos>} \\ \text{Je} \\ \text{vais} \\ \text{bien}
\end{array}
\begin{bmatrix}
23.1 & 4.72 & \cdots & 0.86 \\
6.14 & 20.7 & \cdots & 40.3 \\
8.44 & 19.0 & \cdots & 6.52 \\
11.6 & 11.1 & \cdots & 4.44
\end{bmatrix}
\begin{array}{l}
q_1 \\ q_2 \\ q_3 \\ q_4
\end{array}
\times
\begin{array}{c}
\begin{array}{ccc} \text{I} & \text{am} & \text{good} \end{array} \\
\begin{bmatrix}
1.48 & 9.44 & 53.1 \\
46.2 & 4.67 & 17.6 \\
\vdots & \vdots & \vdots \\
4.36 & 6.72 & 19.5
\end{bmatrix} \\
\begin{array}{ccc} k_1 & k_2 & k_3 \end{array}
\end{array}
=
\begin{array}{r}
\text{<sos>} \\ \text{Je} \\ \text{vais} \\ \text{bien}
\end{array}
\begin{array}{c}
\begin{array}{ccc} \text{I} & \text{am} & \text{good} \end{array} \\
\begin{bmatrix}
q_1 \cdot k_1 & q_1 \cdot k_2 & q_1 \cdot k_3 \\
q_2 \cdot k_1 & q_2 \cdot k_2 & q_2 \cdot k_3 \\
q_3 \cdot k_1 & q_3 \cdot k_2 & q_3 \cdot k_3 \\
q_4 \cdot k_1 & q_4 \cdot k_2 & q_4 \cdot k_3
\end{bmatrix}
\end{array}
$$

$Q(4 \times 64)$ $\quad$ $K^T(64 \times 3)$ $\quad$ $QK^T(4 \times 3)$

02 Multi-Head Attention

3. Calculate the similarity between a query and a key : QK^T

$$
\begin{array}{c}
\begin{array}{c|cccc|c}
 & & & & & \\
\text{<sos>} & 23.1 & 4.72 & \cdots & 0.86 & q_1 \\
\text{Je} & 6.14 & 20.7 & \cdots & 40.3 & q_2 \\
\text{vais} & 8.44 & 19.0 & \cdots & 6.52 & q_3 \\
\text{bien} & 11.6 & 11.1 & \cdots & 4.44 & q_4
\end{array} \\
Q(4 \times 64)
\end{array}
\times
\begin{array}{c}
\begin{array}{ccc}
\text{I} & \text{am} & \text{good} \\
\hline
1.48 & 9.44 & 53.1 \\
46.2 & 4.67 & 17.6 \\
\vdots & \vdots & \vdots \\
4.36 & 6.72 & 19.5 \\
\hline
k_1 & k_2 & k_3
\end{array} \\
K^T(64 \times 3)
\end{array}
=
\begin{array}{c}
\begin{array}{c|ccc|}
 & \text{I} & \text{am} & \text{good} \\
\hline
\text{<sos>} & q_1 \cdot k_1 & q_1 \cdot k_2 & q_1 \cdot k_3 \\
\text{Je} & q_2 \cdot k_1 & q_2 \cdot k_2 & q_2 \cdot k_3 \\
\text{vais} & q_3 \cdot k_1 & q_3 \cdot k_2 & q_3 \cdot k_3 \\
\text{bien} & q_4 \cdot k_1 & q_4 \cdot k_2 & q_4 \cdot k_3 \\
\hline
\end{array} \\
QK^T(4 \times 3)
\end{array}
$$

- For the first row

Query : <sos>, Key : I , am , good

-> Compute similarity to all tokens in a sentence for a query

-> Calculate how similar <sos> is to every word in the input sentence (I, am, good)

02 Multi-Head Attention

3. Query와 Key의 유사도 계산 : $Q \cdot K = QK^T$

$$
\begin{array}{c}
\begin{array}{c|cccc|c}
\text{<sos>} & 23.1 & 4.72 & \cdots & 0.86 & q_1 \\
\text{Je} & 6.14 & 20.7 & \cdots & 40.3 & q_2 \\
\text{vais} & 8.44 & 19.0 & \cdots & 6.52 & q_3 \\
\text{bien} & 11.6 & 11.1 & \cdots & 4.44 & q_4
\end{array} \\
Q(4 \times 64)
\end{array}
\times
\begin{array}{c}
\begin{array}{ccc}
\text{I} & \text{am} & \text{good} \\ \hline
1.48 & 9.44 & 53.1 \\
46.2 & 4.67 & 17.6 \\
\vdots & \vdots & \vdots \\
4.36 & 6.72 & 19.5 \\ \hline
k_1 & k_2 & k_3
\end{array} \\
K^T(64 \times 3)
\end{array}
=
\begin{array}{c}
\begin{array}{c|ccc|}
 & \text{I} & \text{am} & \text{good} \\ \hline
\text{<sos>} & q_1 \cdot k_1 & q_1 \cdot k_2 & q_1 \cdot k_3 \\
\text{Je} & q_2 \cdot k_1 & q_2 \cdot k_2 & q_2 \cdot k_3 \\
\text{vais} & q_3 \cdot k_1 & q_3 \cdot k_2 & q_3 \cdot k_3 \\
\text{bien} & q_4 \cdot k_1 & q_4 \cdot k_2 & q_4 \cdot k_3 \\ \hline
\end{array} \\
QK^T(4 \times 3)
\end{array}
$$

- For the second row

Query : Je, Key : I , am , good

-> Compute similarity to all tokens in a sentence for a query

-> Compute how similar Je is to every word in the input sentence (I, am, good)

02 Multi-Head Attention

3. Query와 Key의 유사도 계산 : $Q \cdot K = QK^T$

$$
\begin{array}{c} \text{<sos>} \\ \text{Je} \\ \text{vais} \\ \text{bien} \end{array}
\begin{bmatrix} 23.1 & 4.72 & \cdots & 0.86 \\ 6.14 & 20.7 & \cdots & 40.3 \\ 8.44 & 19.0 & \cdots & 6.52 \\ 11.6 & 11.1 & \cdots & 4.44 \end{bmatrix}
\begin{array}{c} q_1 \\ q_2 \\ q_3 \\ q_4 \end{array}
\times
\begin{array}{ccc} \text{I} & \text{am} & \text{good} \\ \hline 1.48 & 9.44 & 53.1 \\ 46.2 & 4.67 & 17.6 \\ \vdots & \vdots & \vdots \\ 4.36 & 6.72 & 19.5 \\ \hline k_1 & k_2 & k_3 \end{array}
=
\begin{array}{c|ccc} & \text{I} & \text{am} & \text{good} \\ \hline \text{<sos>} & q_1 \cdot k_1 & q_1 \cdot k_2 & q_1 \cdot k_3 \\ \text{Je} & q_2 \cdot k_1 & q_2 \cdot k_2 & q_2 \cdot k_3 \\ \text{vais} & \boxed{q_3 \cdot k_1 \quad q_3 \cdot k_2 \quad q_3 \cdot k_3} \\ \text{bien} & q_4 \cdot k_1 & q_4 \cdot k_2 & q_4 \cdot k_3 \end{array}
$$

$Q(4 \times 64)$ $\quad$ $K^T(64 \times 3)$ $\quad$ $QK^T(4 \times 3)$

- For the third row

Query : vais, Key : I , am , good

-> Compute similarity to all tokens in a sentence for a query

-> Compute how similar vais is to every word in the input sentence (I, am, good)

02 Multi-Head Attention

3. Query와 Key의 유사도 계산 : $Q \cdot K = QK^T$

$$
\begin{array}{c|cccc|c}
 & & & & & \\
\text{<sos>} & 23.1 & 4.72 & \cdots & 0.86 & q_1 \\
\text{Je} & 6.14 & 20.7 & \cdots & 40.3 & q_2 \\
\text{vais} & 8.44 & 19.0 & \cdots & 6.52 & q_3 \\
\text{bien} & 11.6 & 11.1 & \cdots & 4.44 & q_4 \\
\end{array}
\times
\begin{array}{ccc}
\text{I} & \text{am} & \text{good} \\
\hline
1.48 & 9.44 & 53.1 \\
46.2 & 4.67 & 17.6 \\
\vdots & \vdots & \vdots \\
4.36 & 6.72 & 19.5 \\
\hline
k_1 & k_2 & k_3
\end{array}
=
\begin{array}{c|ccc|}
 & \text{I} & \text{am} & \text{good} \\
\hline
\text{<sos>} & q_1 \cdot k_1 & q_1 \cdot k_2 & q_1 \cdot k_3 \\
\text{Je} & q_2 \cdot k_1 & q_2 \cdot k_2 & q_2 \cdot k_3 \\
\text{vais} & q_3 \cdot k_1 & q_3 \cdot k_2 & q_3 \cdot k_3 \\
\text{bien} & q_4 \cdot k_1 & q_4 \cdot k_2 & q_4 \cdot k_3 \\
\hline
\end{array}
$$

$Q(4 \times 64)$ $\quad$ $K^T(64 \times 3)$ $\quad$ $QK^T(4 \times 3)$

- For the fourth row

Query : bien, Key : I , am , good

-> Compute similarity to all tokens in a sentence for a query

-> Compute how similar bien is to every word in the input sentence (I, am, good)

02 Multi-Head Attention

Multi-Head Attention(in Decoder) - Process

1. Generate Query from Attention Matrix (M from Masked), Key and Value from Encoder Representation (R)

2. Calculate the similarity between a query and a key : QK^T
3. Scaling & Softmax -> score matrix : $softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right)$

4. Multiply by Value to get the Attention matrix Z

Those parts are same with the Encoder

02 Multi-Head Attention

4. Scaling and Softmax (normalization) → score matrix : $softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right)$

- Scaling: divide QK^T by the dimension of the key (scaled-dot product)

  → Scaling values that have become too large due to dot-product

  → Optimized gradient to be stable when back-propagating

- Softmax : Take the soft-max in $\frac{QK^T}{\sqrt{d_k}}$ and change it to a score matrix

02 Multi-Head Attention

5. Multiply by Value to obtain Attachment matrix Z

- Use the score matrix as a weight to give more weight to highly relevant words
- Multiply the Score matrix and the Value matrix to get the Attention matrix Z.

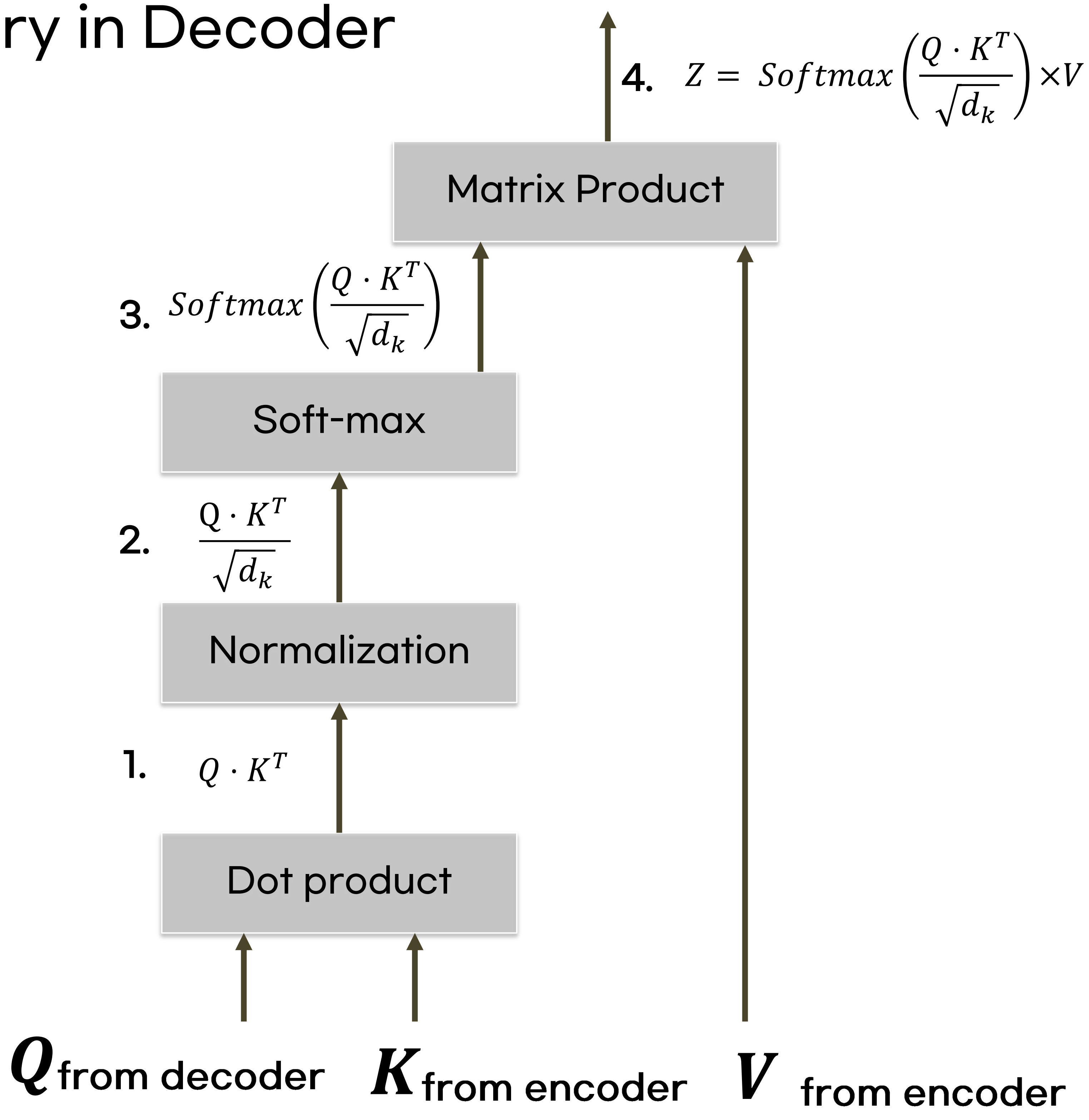
→ The model learns which words to focus on in which contexts through an attentional matrix.

$$
\begin{array}{c|ccc|}
 & \text{I} & \text{am} & \text{good} \\
\hline
\text{<sos>} & 0.84 & 0.017 & 0.14 \\
\text{Je} & 0.98 & 0.02 & 0 \\
\text{vais} & 0 & 1 & 0 \\
\text{bien} & 0 & 0 & 1 \\
\hline
\end{array}
\times
\begin{array}{c|cccc|c}
\hline
\text{I} & 2.53 & 12.5 & \cdots & 63.7 & v_1 \\
\text{am} & 4.77 & 44.2 & \cdots & 2.46 & v_2 \\
\text{good} & 8.35 & 5.25 & \cdots & 5.77 & v_3 \\
\hline
\end{array}
=
\begin{array}{c|ccc|}
\hline
\text{<sos>} & 3.38 & \cdots & 54.36 \\
\text{Je} & 2.65 & \cdots & 62.48 \\
\text{vais} & 4.77 & \cdots & 2.46 \\
\text{bien} & 8.35 & \cdots & 5.77 \\
\hline
\end{array}
$$

$softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right)\ (4 \times 3)$ $\quad$ $V(3 \times 64)$ $\quad$ $Z\ (4 \times 64)$

02 Multi-Head Attention

Self Attention Summary in Decoder



02 Multi-Head Attention (code)

- The decoder's attention head generates Q with the decoder's masked multi-head attentions, and K and V with the encoder's final representation.

```
class DecoderAttentionHead(nn.Module):
    def __init__(self, embed_dim, head_dim):
        super().__init__()
        self.W_q = nn.Linear(embed_dim, head_dim)
        self.W_k = nn.Linear(embed_dim, head_dim)
        self.W_v = nn.Linear(embed_dim, head_dim)

    def forward(self, encoder_hidden_state, decoder_hidden_state):
        attn_outputs = scaled_dot_product_attention(
            self.W_q(decoder_hidden_state),
            self.W_k(encoder_hidden_state),
            self.W_v(encoder_hidden_state)
        )
        return attn_outputs
```

```
def scaled_dot_product_attention(query, key, value, mask=None):
    dim_k = query.size(-1)
    scores = torch.bmm(query, key.transpose(1, 2)) / sqrt(dim_k)
    if mask is not None:
        scores = scores.masked_fill(mask == 0, float("-inf"))
    weights = F.softmax(scores, dim=-1)
    return weights.bmm(value)
```

✓ 0.0s

02 Multi-Head Attention

As the same way decoder concateate all the heads

$$\mathrm{Concatenate}([Z_1, Z_2, \cdots, Z_h])\boldsymbol{W}$$

03 Add&Norm

- Add : Skip-connection

  Using Skip-Connection to Prevent Vanishing Gradients
- Norm : Layer Normalization

  - Normalizes the sequence to each

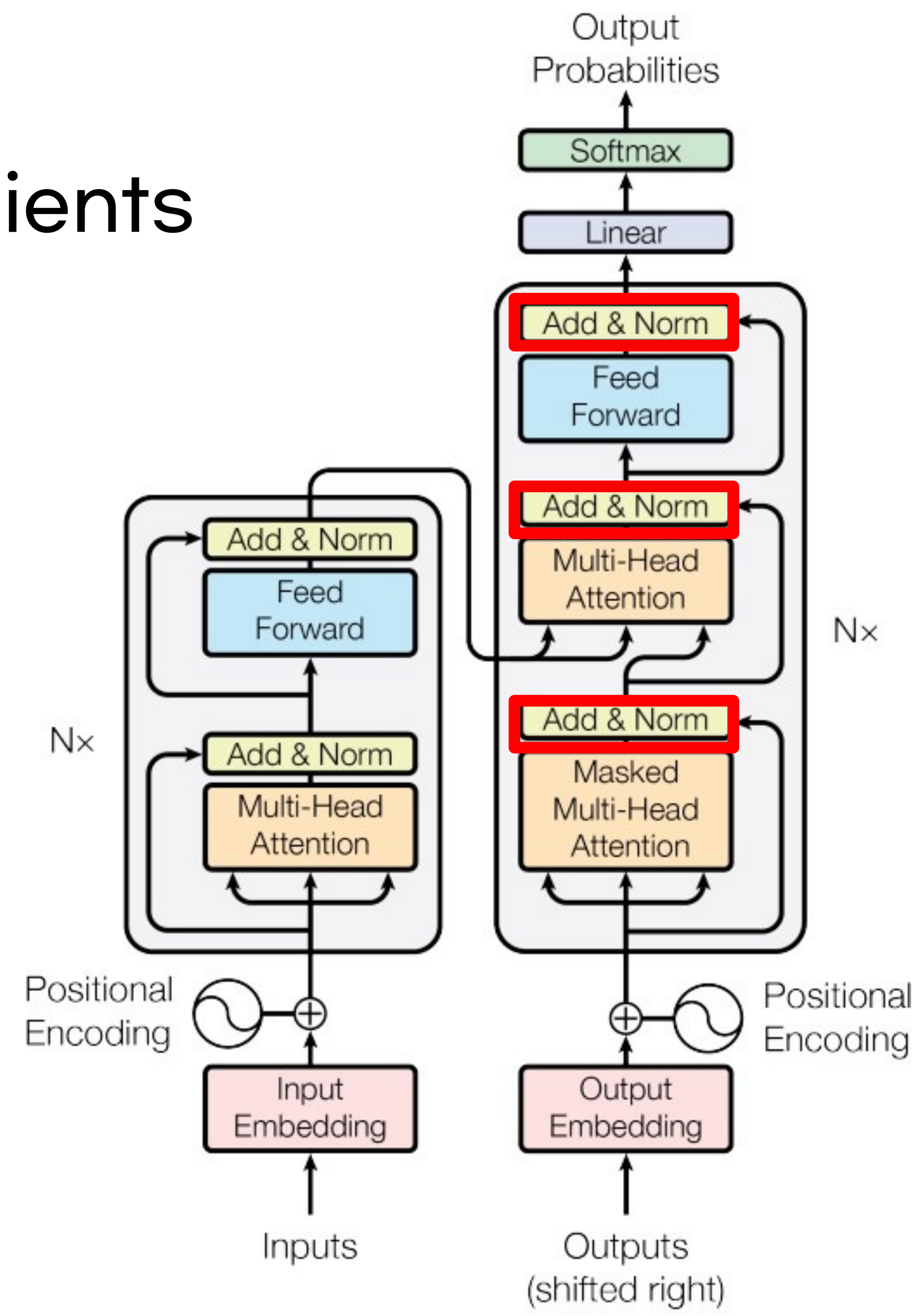


Figure 1: The Transformer - model architecture.

04 Linear & Softmax

Using a Linear Layer as a classifier and the Softmax Function

- Output as the word at the index with the highest probability value

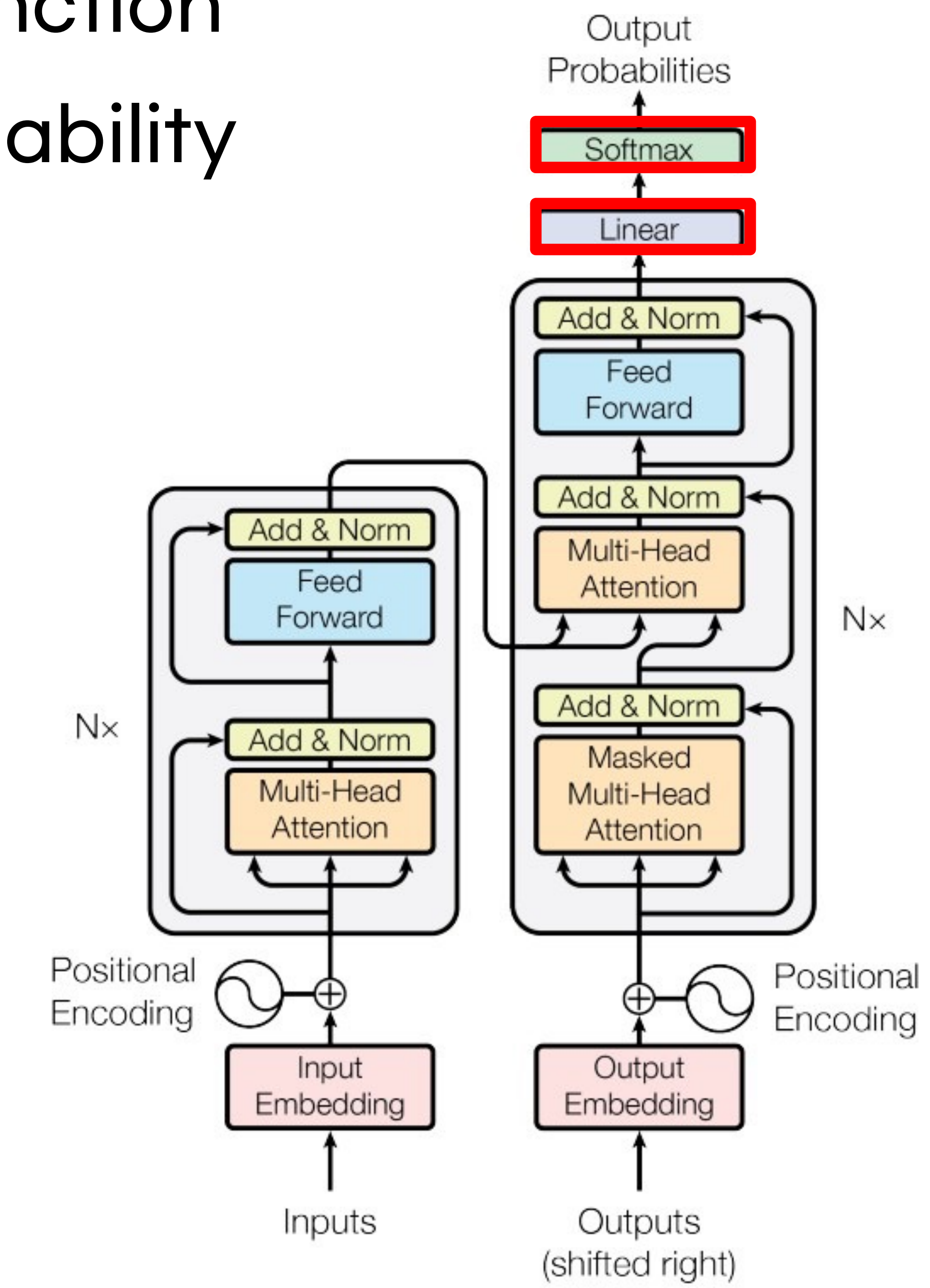


Figure 1: The Transformer - model architecture.

04 Classification head

- Like encoders, decoders are created by stacking layers of decoders.
- The final representation of the decoder has a classification head to predict the next word.

```
class ClassificationHead(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.encoder = TransformerEncoder(config)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.classifier = nn.Linear(config.hidden_size, config.num_labels)

    def forward(self, x):
        x = self.encoder(x)[:, -1, :] # 마지막 토큰의 최종 표현 선택
        x = self.dropout(x)
        x = self.classifier(x)
        return x
```

✓ 0.0s

참조 책

구글 BERT의 정석

참조 링크

- https://github.com/bentrevett/pytorch-seq2seq/blob/master/6%20-%20Attention%20is%20All%20You%20Need.ipynb
- https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0
  https://stackoverflow.com/questions/58127059/how-to-understand-masked-multi-head-attention-in-transformer/59713254#59713254?newreg=c60d6eca60764bd782a3b453a40ca880

Thank you