
The Foundation of LLMs : Transformer Architecture Analyze

B989068 홍성준

HONGIK UNIVERSITY
Capstone Design(1)

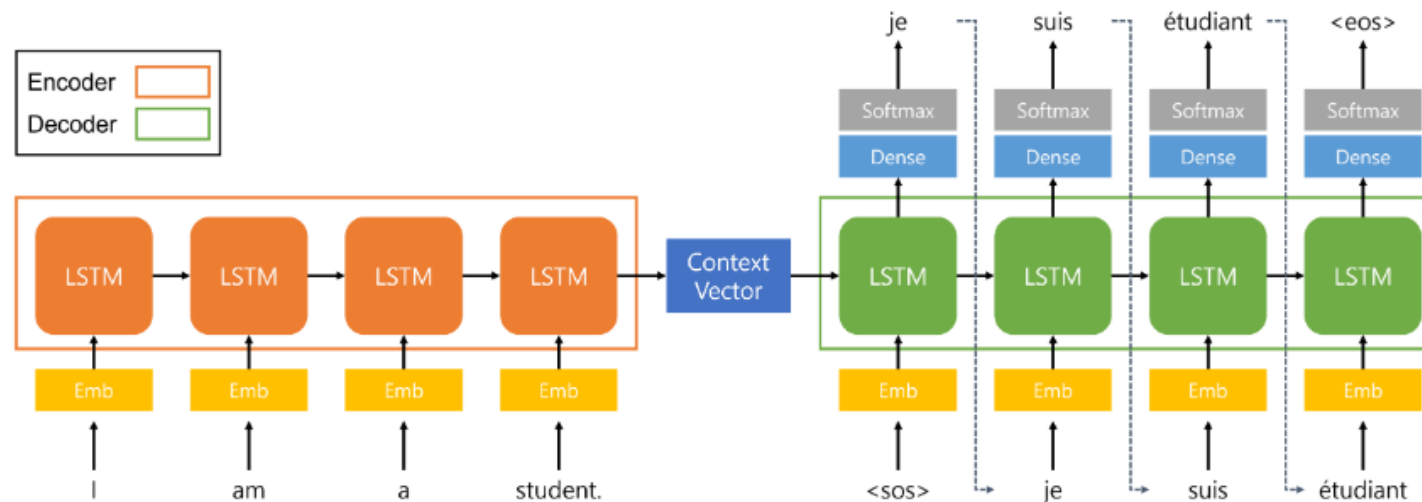
Contents

- Background
- Summary
- Transformer Architecture
 - Encoder-Decoder Architecture
 - Scaled Dot-Product Attention
 - Multi-Head Attention
 - Position-wise Feed-Forward Networks(FFN)
 - Embedding & Positional Encoding
- Transformer Example (Translation En-Ko)
- Conclusion

Background

Limitations of previous models

- Previous models struggle to learn long-term dependencies.
- This can lead to issues such as gradient vanishing or exploding.
- "Parallel computation is limited, so computational speed may be slow.



Transformer Architecture

Summary

- Encoder Block consists of two sub-layers.
 - Multi-Head(Self) Attention, Feed Forward
- Decoder Block consists of three sub-layers.
 - Masked Multi-Head(Self) Attention, Multi-Head Attention, Feed Forward
- To enhance the representational capacity of the model, the encoder and decoder are stacked with N blocks.
- Transformer model integrates Positional Encoding to effectively learn and utilize the relative position and sequence information of input sequences.

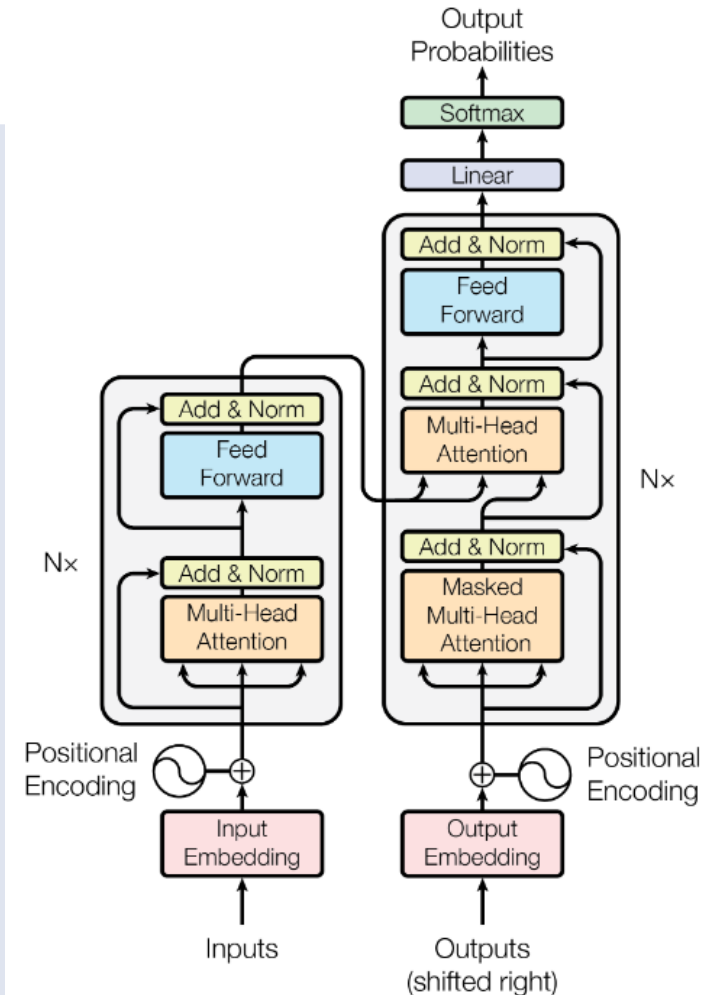


Figure 1: The Transformer - model architecture.

Transformer Architecture

Attention Mechanism

- The attention mechanism calculates the **similarity** between a given Query and Key and weights the Value based on this similarity to produce an output.
- **Query(Q)**: This is the vector used to calculate similarity. It represents the given input, typically a query or request in a specific task.
- **Key(K)**: This is the vector used as a **target** for calculating similarity. It is employed to compute the similarity with the given Query. It's commonly used as the feature vector of items to compare, especially in search tasks.
- **Value(V)**: This is the vector representing the **weighted sum** associated with the Key. It reflects the weights assigned based on the similarity between Query and Key. These weights are then combined with the Value to generate the final result.

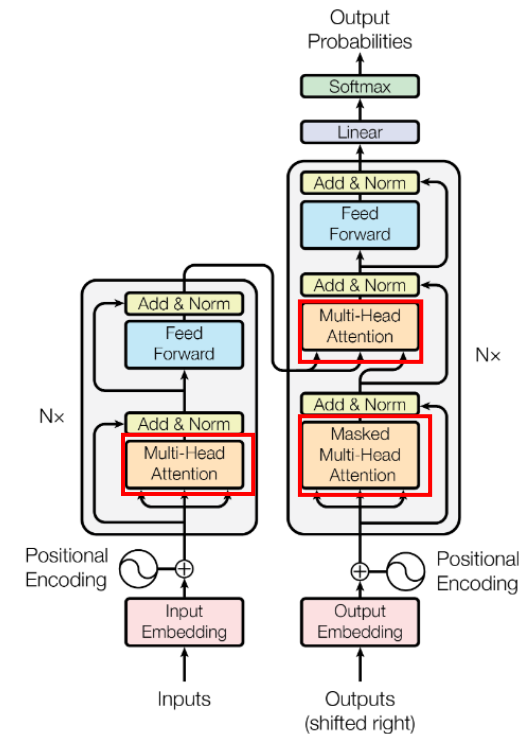


Figure 1: The Transformer - model architecture.

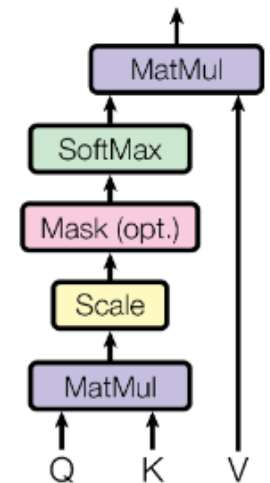
Transformer Architecture

Scaled Dot-Product Attention

- Generate Query (Q), Key (K), and Value (V) from the given input sequence by performing linear transformations on the input sequence.
- Obtain the **attention score** by taking the dot product of the Q and K vectors.
- To solve the issue of the softmax function being pushed into regions with extremely small gradients as dot product values increase, we scale these values by $1/\sqrt{d_k}$.
- Before applying softmax, we apply **masking**.
 - Decoder Masked Multi-Head(Self) Attention
- Applying **softmax** to obtain **Attention Weights**.
- Compute the matrix **multiplication of Attention Weight and V vector**

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

Scaled Dot-Product Attention



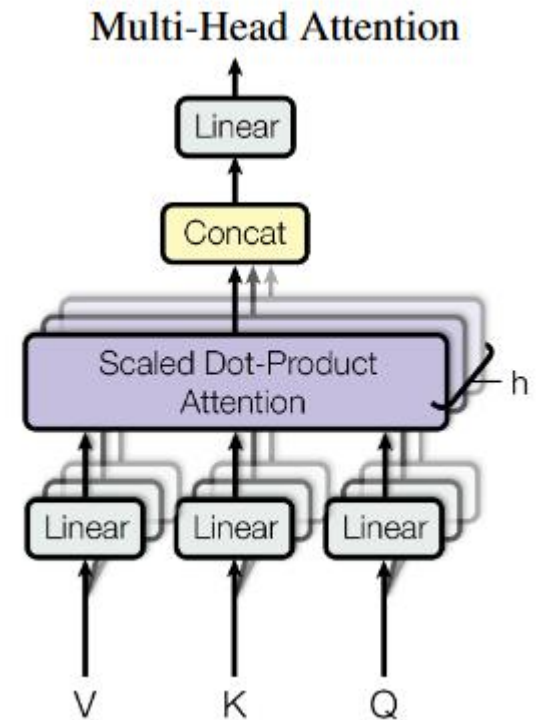
Transformer Architecture

Multi-Head Attention

- Q, K, and V are **independently** generated for **each head**.
- Each head performs the Scaled-Dot Product Attention process on different Q, K, V weight matrices.
- The attention outputs of each head are concatenated and then linearly projected into the d_{model} dimension through dot product with W^O .

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$



Transformer Architecture

Multi-Head Attention in Transformer

① Encoder Self-Attention:

- Keys, values, and queries all from previous encoder layer.
- Each encoder position attends to all positions in previous layer.

② Decoder Self-Attention:

- Allows each decoder position to attend to all positions up to and including itself.
- Leftward information flow in decoder prevented to maintain auto-regressive property.
- Achieved by masking out illegal connections in scaled dot-product attention.

③ Encoder-Decoder Attention:

- Queries from previous decoder layer.
- Memory keys and values from output of encoder.
- Enables decoder positions to attend over all input positions.

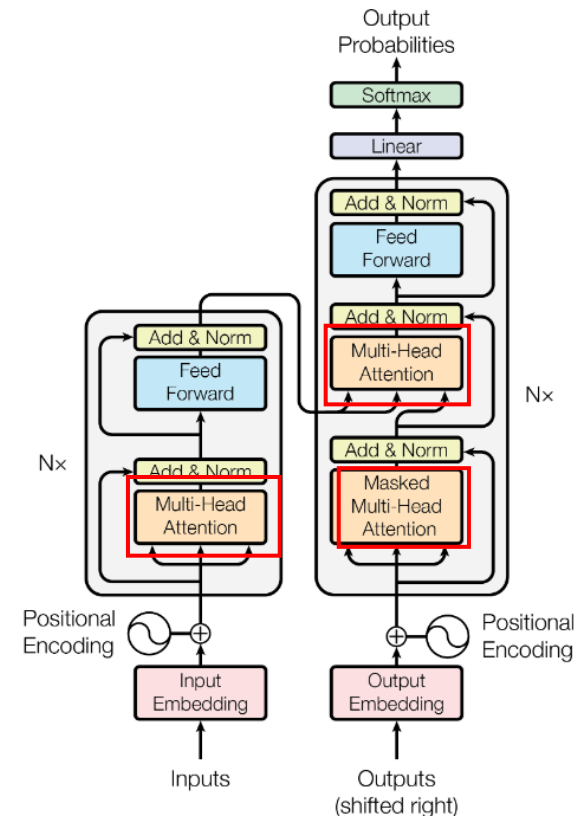


Figure 1: The Transformer - model architecture.

Transformer Architecture

Position-wise Feed-Forward Networks

- Each layer in encoder and decoder includes a fully connected feed-forward network.
- Consists of **two linear transformations with ReLU** activation in between.

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

- Linear transformations use different parameters from layer to layer.

- Position-wise Feed-Forward Networks

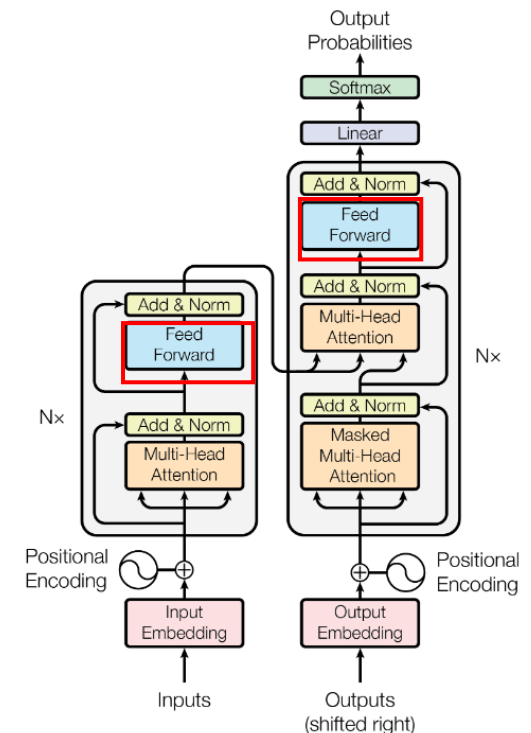
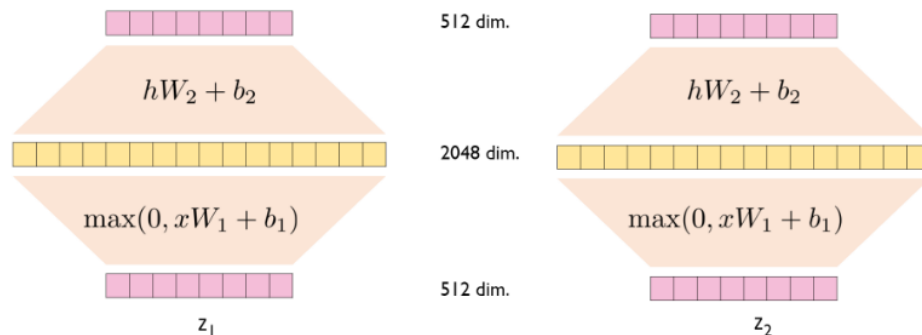


Figure 1: The Transformer - model architecture.

Transformer Architecture

Position-wise Feed-Forward Networks

- **Nonlinearity addition:** FFN takes the linear relationships modeled by Multi-Head Attention and transforms them into nonlinear relationships, allowing the model to learn more complex patterns and relationships.
- **Enhanced inter-layer representation:** FFN performs separate processing for the output of each Attention layer, enabling the encoder to extract and understand features of deeper data, not just the interaction of input data.
- **Increased model flexibility:** The combination of Attention mechanism and FFN provides greater flexibility to the model, crucial for handling various types of data and complex problems.

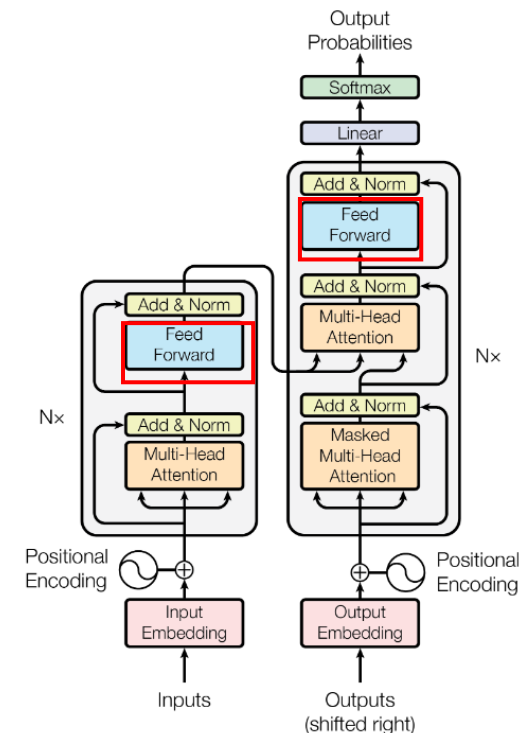


Figure 1: The Transformer - model architecture.

Transformer Architecture

Embedding Vectors

- An embedding vector refers to the representation of text as a fixed-size array composed of floating-point numbers.
- From text, tokens can be extracted using a tokenizer to generate embeddings.
- **Learned embeddings** to convert the input tokens and output tokens to vectors of dimension d_{model} .

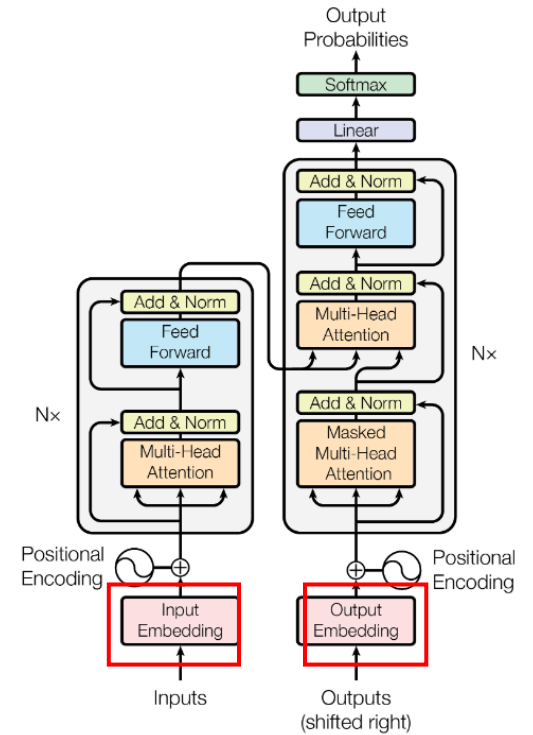


Figure 1: The Transformer - model architecture.

GPT-3.5 & GPT-4 GPT-3 (Legacy)

We study the topmost weight matrix of neural network language models.

Tokens	Characters
13	69

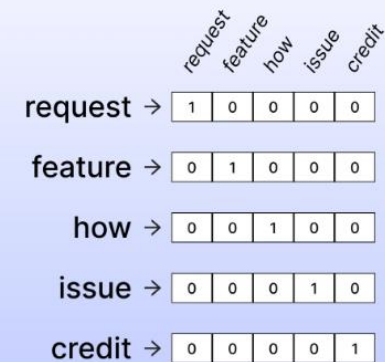
We study the topmost weight matrix of neural network language models.

Transformer Architecture

One-hot Encoding

- In the past, relatively simple methods were adopted to create embeddings.
- The process involves creating a vector of zeros with the same length as the total number of words in the vocabulary.
- Then, the vector is populated with 1 at the position corresponding to the index of the given word, resulting in the final embedding vector.

One-hot Encoding




	request	feature	how	issue	credit
request →	1	0	0	0	0
feature →	0	1	0	0	0
how →	0	0	1	0	0
issue →	0	0	0	1	0
credit →	0	0	0	0	1

Transformer Architecture

Learned Embedding

- The learned embeddings are trained in such a way that the embedding vectors of semantically similar words in context become close to each other, while those of dissimilar words become distant.
- This allows the model to effectively grasp the semantic relationships between words and adequately encode their meanings.
- Through this process, the model learns word embeddings by leveraging rich contextual information obtained from the training corpus.

Learned Embedding



embedding pairs

request →	0.007	-0.039	-0.002	-0.024
feature →	-0.014	-0.011	0.014	-0.017
how →	0.01	0.004	0.006	-0.037
issue →	-0.015	-0.035	-0.009	-0.01
credit →	-0.012	-0.027	-0.001	-0.05

Transformer Architecture

Positional Encoding

- Transformer model processes input data in **parallel**, which leads to the **loss of information** about the input sequence order.
- To address this issue, Positional Encoding vectors are added to the Embedding vectors and used as the input data for the first layer of the model.
- As a result, the positional information of input words can be represented.
- p : Variable representing the position of the word
- d_{model} : Embedding dimension (512 in the Transformer paper)
- i : Index of elements in the output positional embedding vector

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

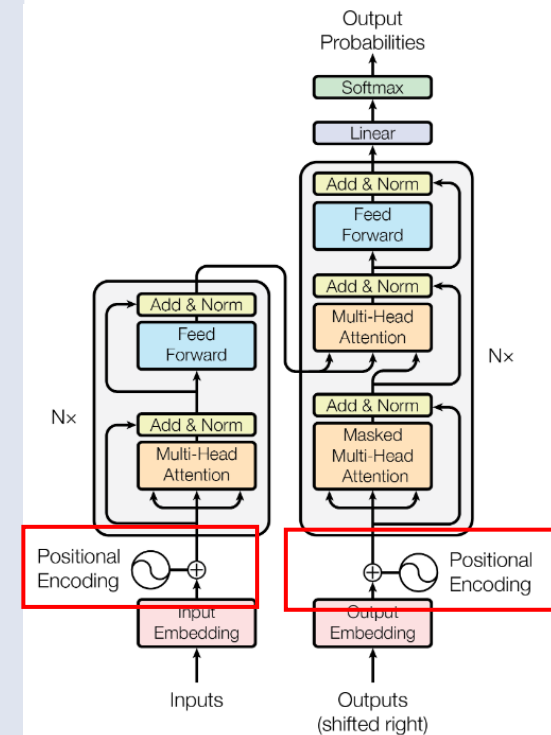


Figure 1: The Transformer - model architecture.

Transformer Architecture

Positional Encoding

- To represent positional information, each vector value of pos, which denotes the word's position, must be different.
- Therefore, to address the issue of repetitive position vector values, we alternate between applying **sine** and **cosine functions**.
- When i is an even number, including 0, we apply the sine function, and when i is odd, we apply the cosine function.

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

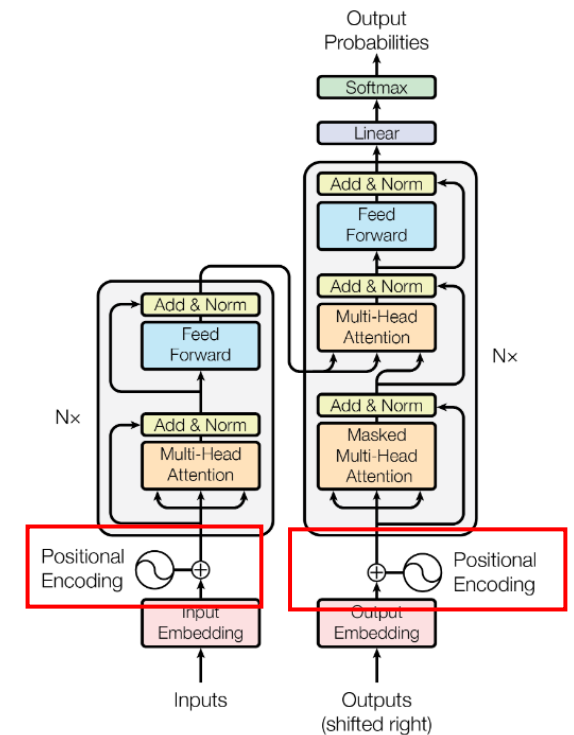


Figure 1: The Transformer - model architecture.

Translation Example

Example

- Describing an example of a Transformer model performing the task of translating the sentence
- "I am a student" -> "나는 학생이다".

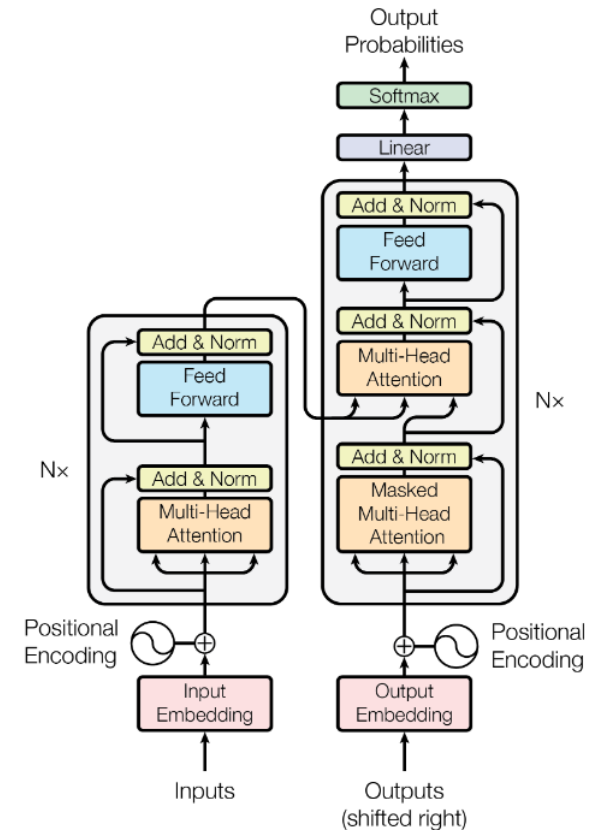
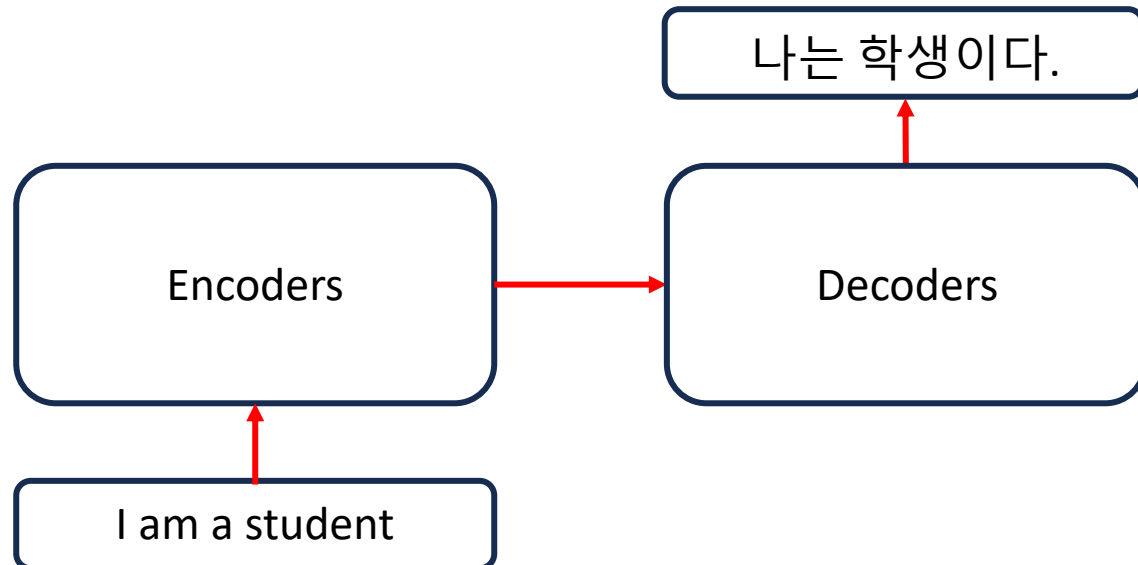


Figure 1: The Transformer - model architecture.

Translation Example

Input Embedding

- Similarly to other sequence transduction models, we use learned embeddings to convert the input tokens and output tokens to vectors of dimension d_{model} .
- In the example, the embedding vectors are described with d_{model} set to 4.

I				
am				
a				
student				

나				
는				
학생				
이다				

Translation Example

Positional Encoding

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

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

- Obtain the positional encoding vector using sinusoidal function and add it to the embedding vector.
- To prevent the positional encoding values from overshadowing the meaning of the embedding vector when added, we multiply those weights by $\sqrt{d_{model}}$.

$\sqrt{d_{model}}$	\times	I				
		am				
		a				
		student				

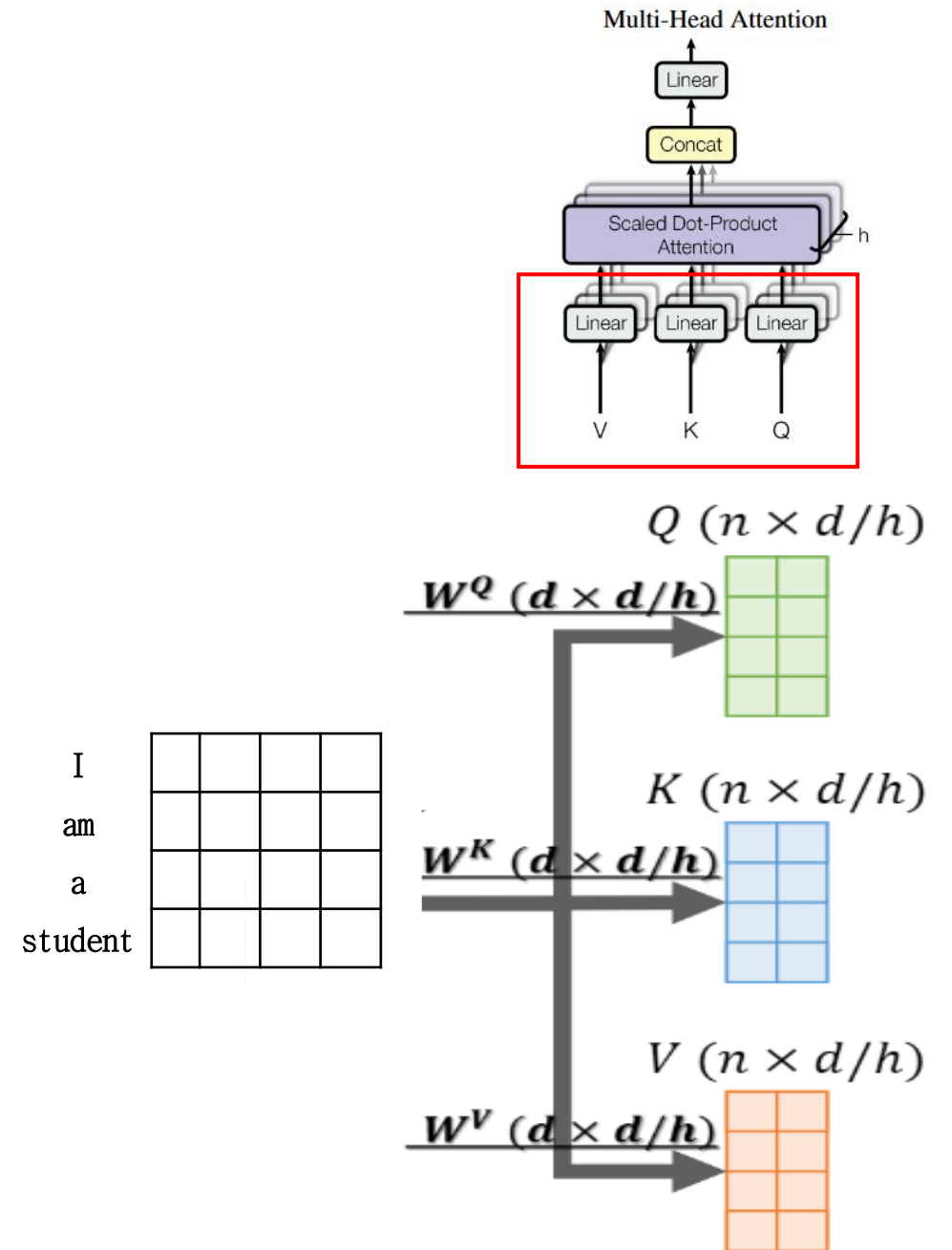
+

	i = 0	i = 0	i = 1	i = 1
p = 0	0	1	0	1
p = 1	0.91	-0.42	0.2	1
p = 2	0.14	-0.99	0.3	0.96
p = 3	-0.76	-0.65	0.39	0.92

Translation Example

Encoder : Multi-Head Attention

- The queries, keys, and values are linearly projected into dimensions d_q , d_k , and d_v , respectively, using learned different linear projections.
- Then, the attention function is performed in parallel on these projected versions of queries, keys, and values, resulting in d_v -dimensional output values.
- d_k and d_v are set to d_{model} / h .



Translation Example

Encoder : Multi-Head Attention

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

- To compute the dot product between the query and the key, we transpose the key vector and then perform the multiplication.
- This result represents the similarity between the query and the key.

The diagram illustrates the computation of the dot product between a query matrix and a transposed key matrix. On the left, the **Query₁** matrix (green) has rows for 'I', 'am', 'a', and 'student', and two columns. Next to it is the **Key₁^T** matrix (blue), which has columns for 'I', 'am', 'a', and 'student', and two rows. A large 'X' symbol indicates the dot product operation. To the right of the 'X' is an equals sign. On the far right is the resulting similarity matrix (yellow), which has rows for 'I', 'am', 'a', and 'student', and columns for 'I', 'am', 'a', and 'student'. The values in the matrix are: Row 'I': [5.5, 0.5, 0.2, 1.5]; Row 'am': [0.5, 4.4, 0.3, 0.6]; Row 'a': [0.2, 0.3, 4.5, 0.4]; Row 'student': [1.5, 0.6, 0.4, 5.5].

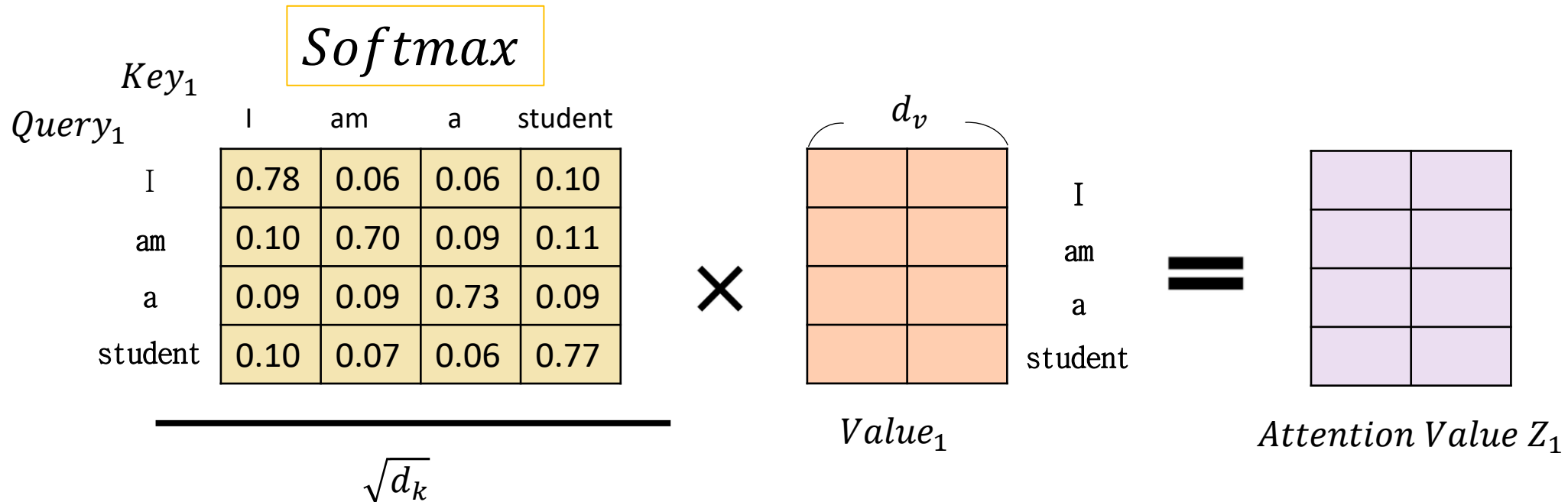
	I	am	a	student
I	5.5	0.5	0.2	1.5
am	0.5	4.4	0.3	0.6
a	0.2	0.3	4.5	0.4
student	1.5	0.6	0.4	5.5

Translation Example

Encoder : Multi-Head Attention

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

- After scaling by $1/\sqrt{d_k}$, we apply the softmax function to normalize the similarity scores with each key, obtaining the weights for each key.
- Then, multiplying by the value vector yields information where the importance of each value corresponding to the key is reflected.



Translation Example

Encoder : Multi-Head Attention

- The same process can be performed in parallel across different heads, and the results can be concatenated afterward to combine them into one.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

- Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions.

Attention Value Z_1

Attention Value Z_2

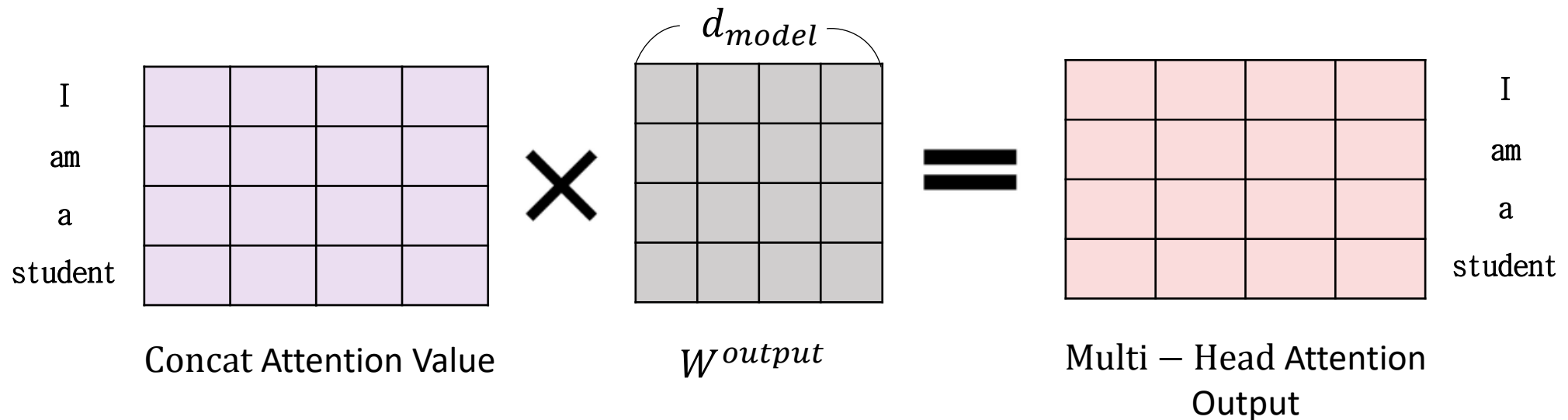
Concat Attention Value

I
am
a
student

Translation Example

Encoder : Multi-Head Attention

- To effectively utilize the results from different attention heads, a linear transformation is performed using the W_o weight matrix to restore the dimensions to their original state.
- This enables the model to efficiently combine information obtained from various aspects and generate the Attention Output.



Transformer Example

Residual Connection & Layer Normalization

- A Residual Connection, represented as $\text{output} = x + \text{sublayer}(x)$, solve the gradient vanishing problem that can occur as the layers deepen.
- The output of the residual connection is used as the input for layer normalization.

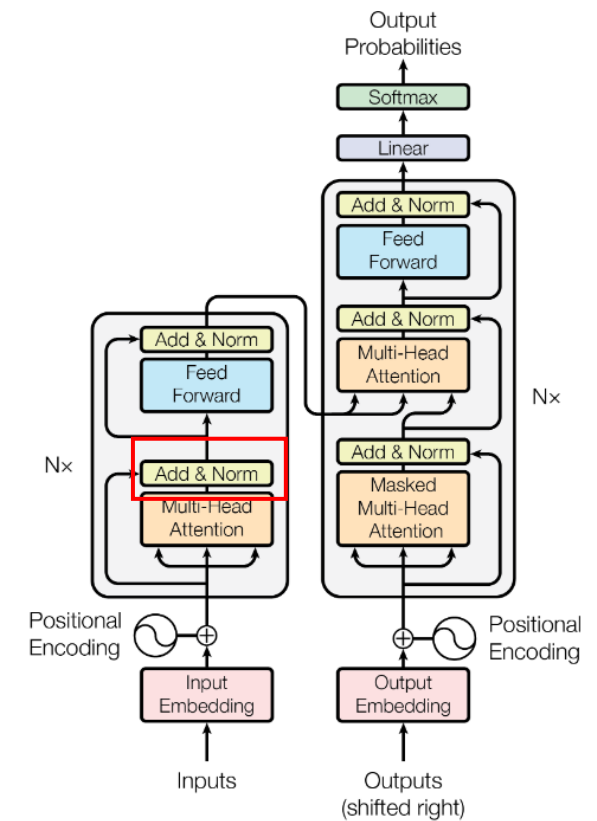
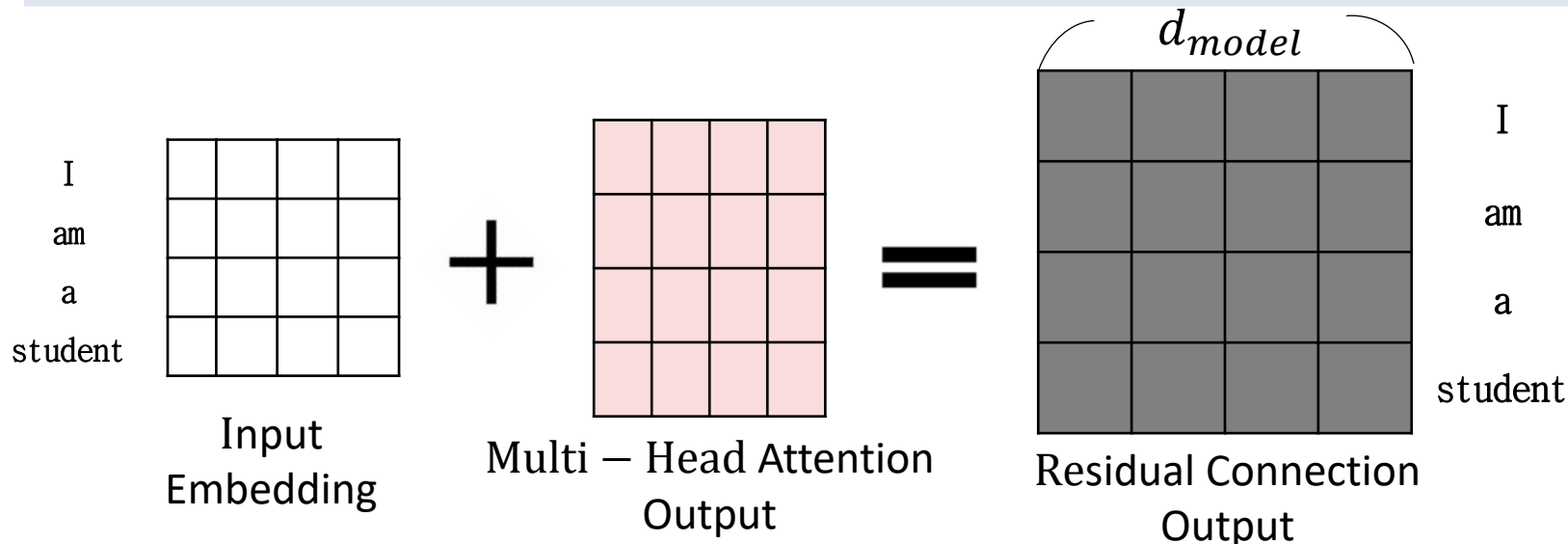
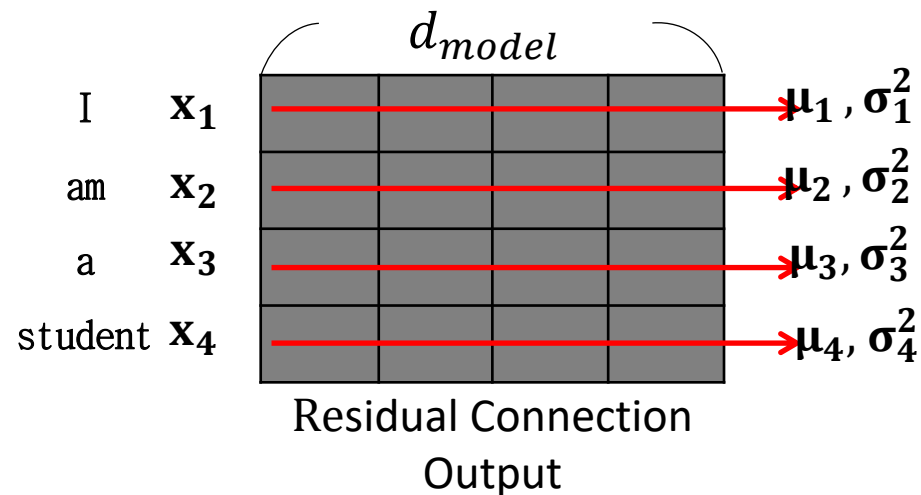


Figure 1: The Transformer - model architecture.

Transformer Example

Residual Connection & Layer Normalization

- The mean and variance for each feature dimension are computed from the layer's output. This calculation is performed for all samples (instances) in the layer.
- Using the calculated mean and variance, each feature dimension is normalized. Subtracting the mean from each dimension's value and then dividing by the variance, each dimension is standardized to have a mean of 0 and a variance of 1.
- Through these processes, Layer Normalization stabilizes the output of each layer and facilitates better learning performance.



$$\hat{x}_{i,k} = \frac{x_{i,k} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

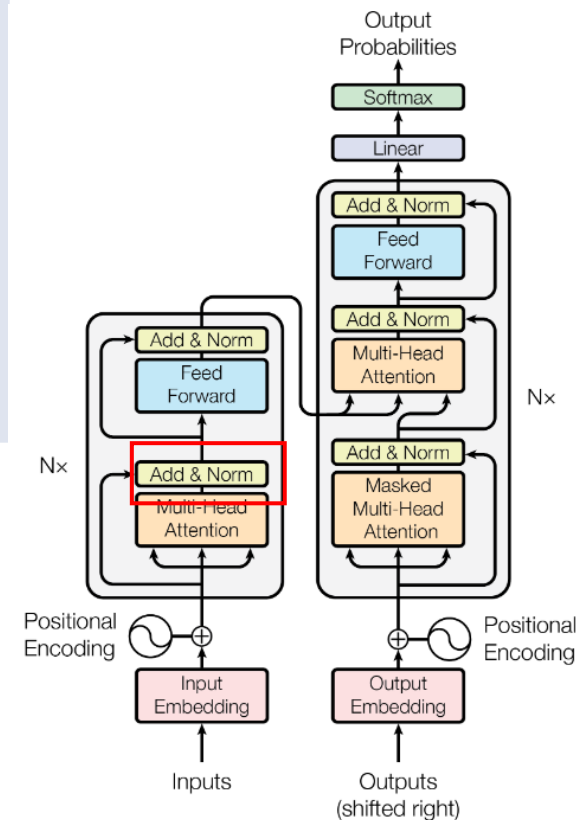


Figure 1: The Transformer - model architecture.

Transformer Example

Encoder : Feed Forward Layer

- The first layer expands the input, applies a non-linear transformation using the activation function (ReLU in this case), and then reduces back to the original dimension through the second layer.
- The ReLU activation function introduces non-linearity, aiding in learning non-linear patterns in the data.
- The weight parameters W_1 and W_2 are learned differently for each layer.

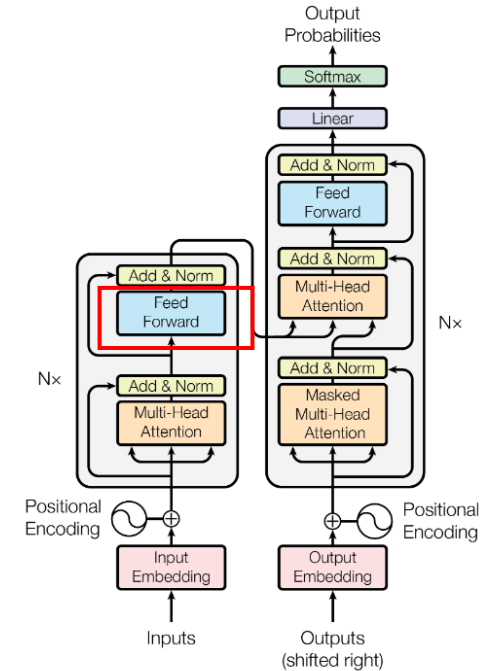
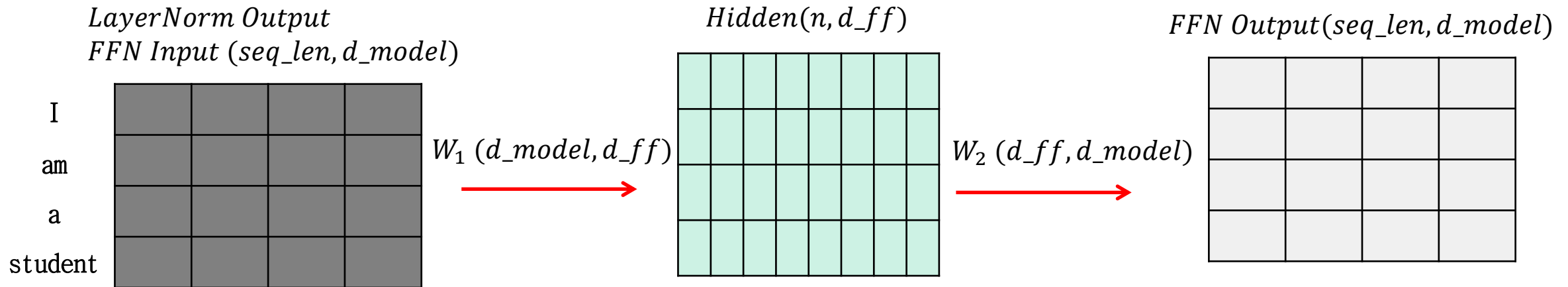


Figure 1: The Transformer - model architecture.



Transformer Example

Encoder Output

- The output of the FFN is used as the input to the next encoder layer after performing residual connection and layer normalization.
- The encoder consists of layers with identical structures, and the output of each layer is used as the input for the subsequent layer.
- The output of the last layer in the encoder is used as the input for the second sublayer of the decoder.

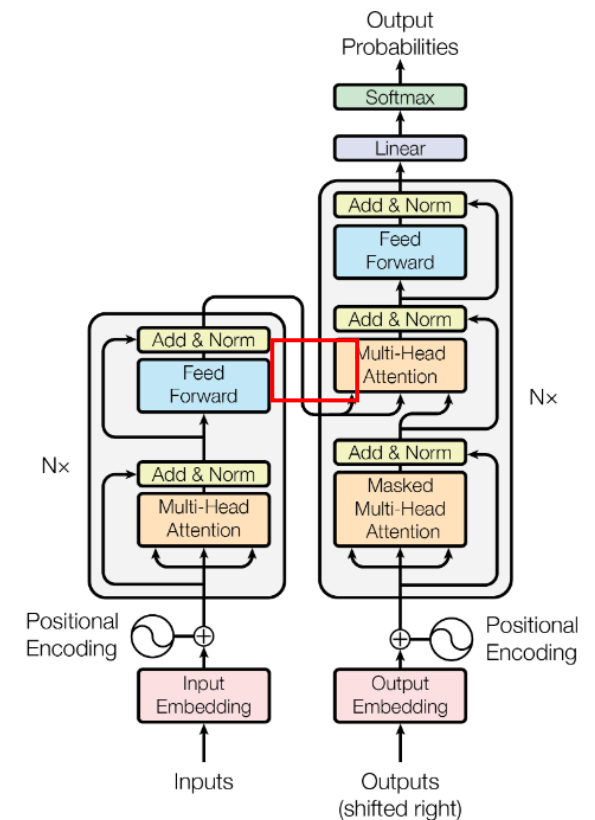
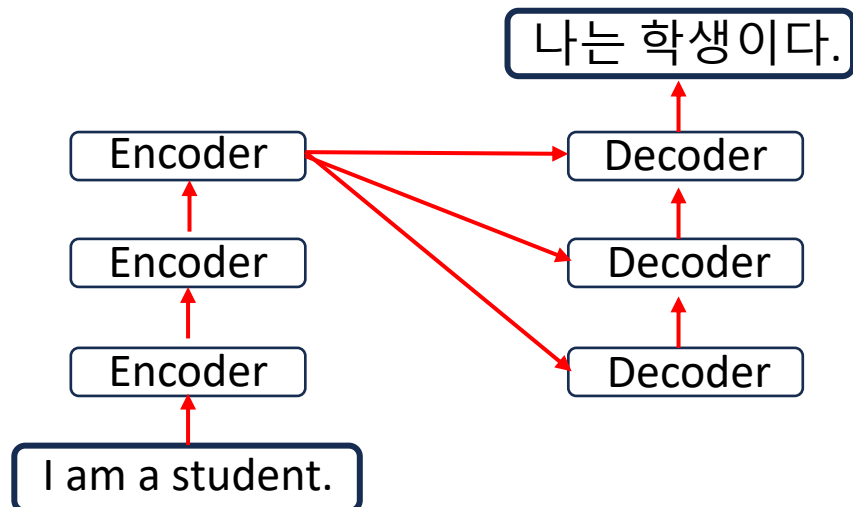


Figure 1: The Transformer - model architecture.

Translation Example

Decoder

- The decoder has a similar structure to the encoder, consisting of similar layers except for masked multi-head attention and encoder-decoder attention.
- This configuration allows the decoder to leverage the information transferred from the encoder to generate output sequences and predict the next token.
- The main role of the decoder is to perform sequence generation using the output of the encoder and the previous output of the decoder itself.

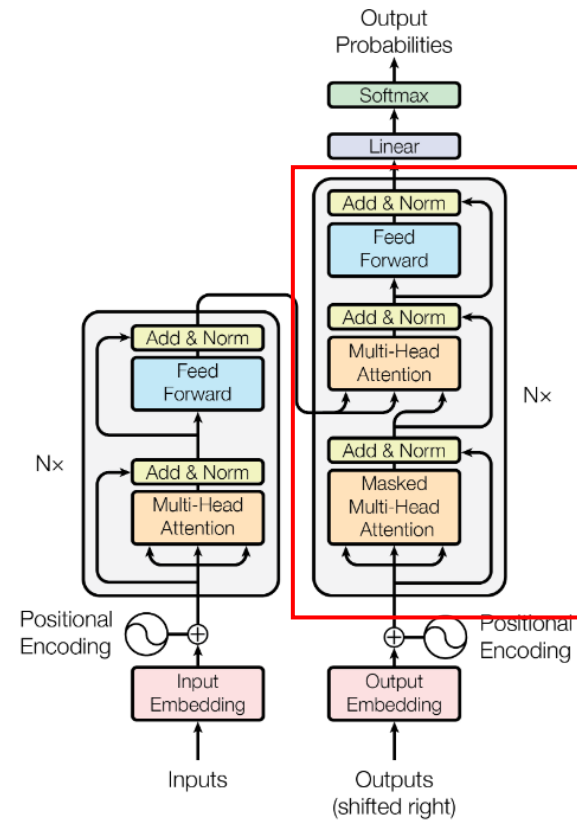


Figure 1: The Transformer - model architecture.

Translation Example

Embedding & Positional Encoding

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

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

- Obtain the positional encoding vector using sinusoidal function and add it to the embedding vector.
- To prevent the positional encoding values from overshadowing the meaning of the embedding vector when added, we multiply those weights by $\sqrt{d_{model}}$.

$\sqrt{d_{model}}$ ×	나				
	는				
	학생				
	이다				

+

	i = 0	i = 0	i = 1	i = 1
p = 0	0	1	0	1
p = 1	0.91	-0.42	0.2	1
p = 2	0.14	-0.99	0.3	0.96
p = 3	-0.76	-0.65	0.39	0.92

Translation Example

Decoder : Masked Multi-Head Attention

- To compute the dot product between the query and the key, we transpose the key vector and then perform the multiplication.
- This result represents the similarity between the query and the key.

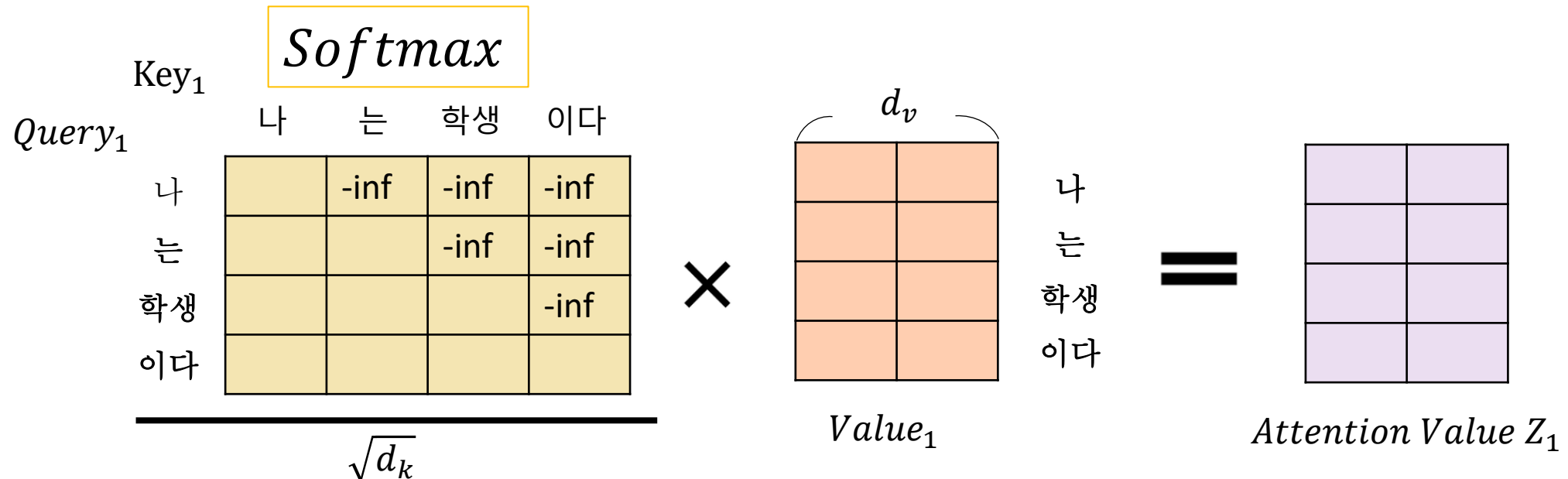
The diagram illustrates the computation of the dot product between a query vector and a transposed key vector. On the left, the query vector $Query_1$ is represented by a 4x2 grid of green cells, with the Korean text '나는 학생이다' (I am a student) written vertically to its left. This is multiplied (indicated by a large 'X') by the transposed key vector Key_1^T , which is a 2x4 grid of blue cells with the same text '나는 학생이다' written horizontally above it. An equals sign (=) follows, leading to the resulting similarity matrix. This matrix is a 4x4 grid of yellow cells. Above the matrix, the text 'Key₁' is written, and to the left, 'Query₁' is written. The Korean text '나는 학생이다' is also written vertically to the left of the matrix. The numerical values in the matrix are:

	나	는	학생	이다
나	5.5	0.5	1.5	0.2
는	0.5	4.4	0.3	0.6
학생	1.5	0.3	4.5	0.4
이다	0.2	0.6	0.4	5.5

Translation Example

Decoder : Masked Multi-Head Attention

- Before taking the softmax, masking is applied with -inf.
- This is done to prevent the decoder from referencing tokens beyond the position of the current predicted word when predicting the next word.
- It allows the model to rely solely on the current input without using future information for prediction.



Translation Example

Decoder : Masked Multi-Head Attention

- The following steps are performed in the same manner as the Multi-Head Attention in the Encoder.
- Similarly, after performing "Residual Connection" and "Layer Normalization," they are used as the input for the Encoder-Decoder Attention.

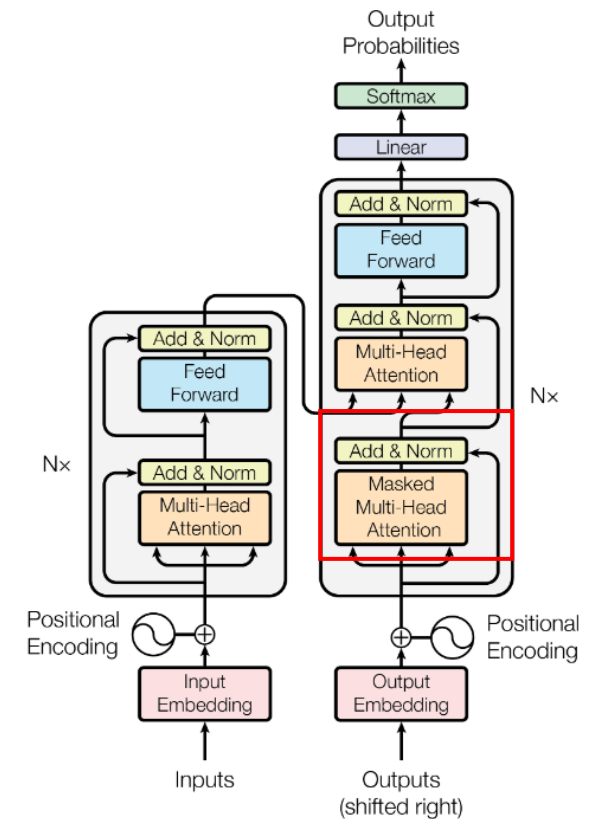


Figure 1: The Transformer - model architecture.

Translation Example

Decoder : Encoder-Decoder Attention

- While the form is Multi-Head Attention, it is not Self-Attention.
- More specifically, the query is constructed using itself, but the key and value utilize the output of the Encoder.

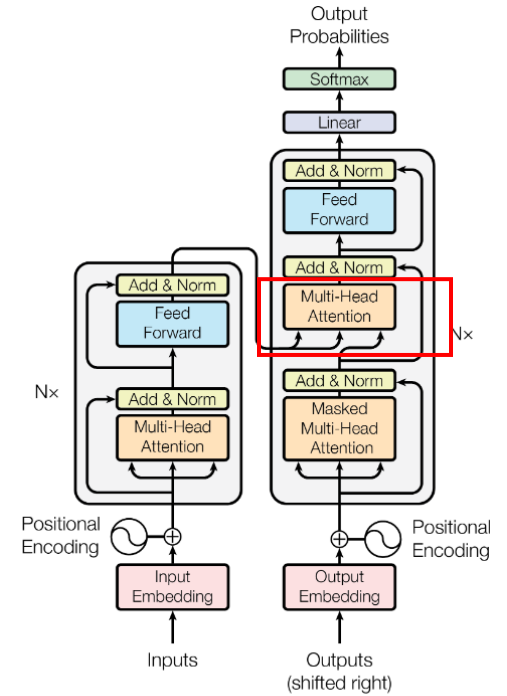


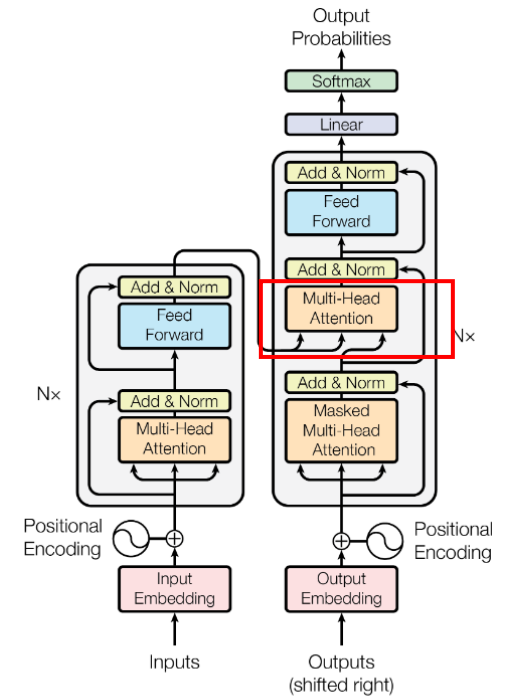
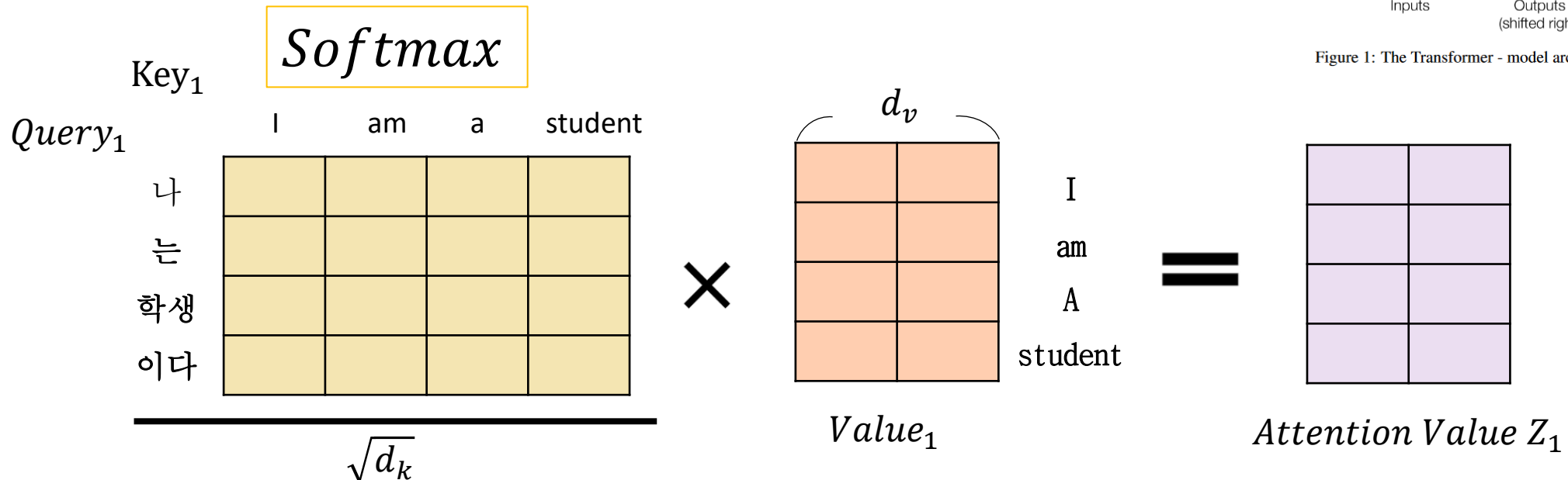
Figure 1: The Transformer - model architecture.

$$\begin{array}{c} \text{나} \\ \text{는} \\ \text{학생} \\ \text{이다} \end{array} \begin{array}{|c|c|} \hline \text{ } & \text{ } \\ \hline \text{ } & \text{ } \\ \hline \text{ } & \text{ } \\ \hline \text{ } & \text{ } \\ \hline \end{array} \times \begin{array}{c} \text{I} \quad \text{am} \quad \text{a} \quad \text{student} \\ \begin{array}{|c|c|c|c|} \hline \text{ } & \text{ } & \text{ } & \text{ } \\ \hline \text{ } & \text{ } & \text{ } & \text{ } \\ \hline \text{ } & \text{ } & \text{ } & \text{ } \\ \hline \end{array} \end{array} = \begin{array}{c} \text{Query}_1 \quad \text{Key}_1 \\ \text{나} \\ \text{는} \\ \text{학생} \\ \text{이다} \end{array} \begin{array}{c} \text{I} \quad \text{am} \quad \text{a} \quad \text{student} \\ \begin{array}{|c|c|c|c|} \hline 5.5 & 0.5 & 0.2 & 1.5 \\ \hline 0.5 & 4.4 & 0.3 & 0.6 \\ \hline 0.2 & 0.3 & 0.4 & 4.5 \\ \hline 1.5 & 0.6 & 2.2 & 2.3 \\ \hline \end{array} \end{array}$$

Translation Example

Decoder : Encoder-Decoder Attention

- This is because we predict the target sequence using information from the source sequence, hence the key and value rely on the Encoder's output.



Translation Example

Decoder : Encoder-Decoder Attention

- The following steps are performed in the same manner as the Multi-Head Attention.
- Similarly, after performing "Residual Connection" and "Layer Normalization," they are used as the input for the Feed Forward.
- Also, The Feed Forward Network (FFN) layer in the decoder performs a similar role to the FFN layer in the encoder.

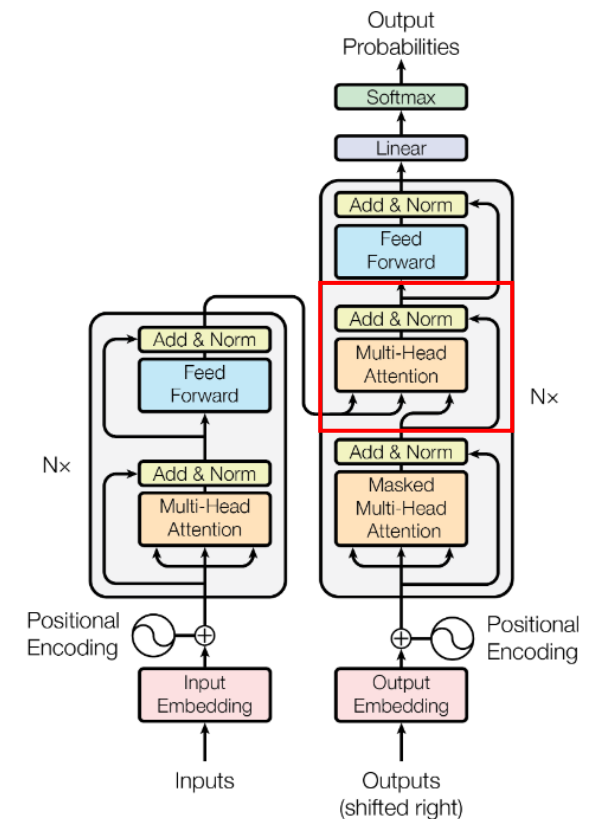


Figure 1: The Transformer - model architecture.

Translation Example

Output

- The linear layer is applied to the decoder's output to calculate logits.
- Then softmax is applied to the computed logits to obtain logit probabilities, with the highest probability word being the target word derived from the source.

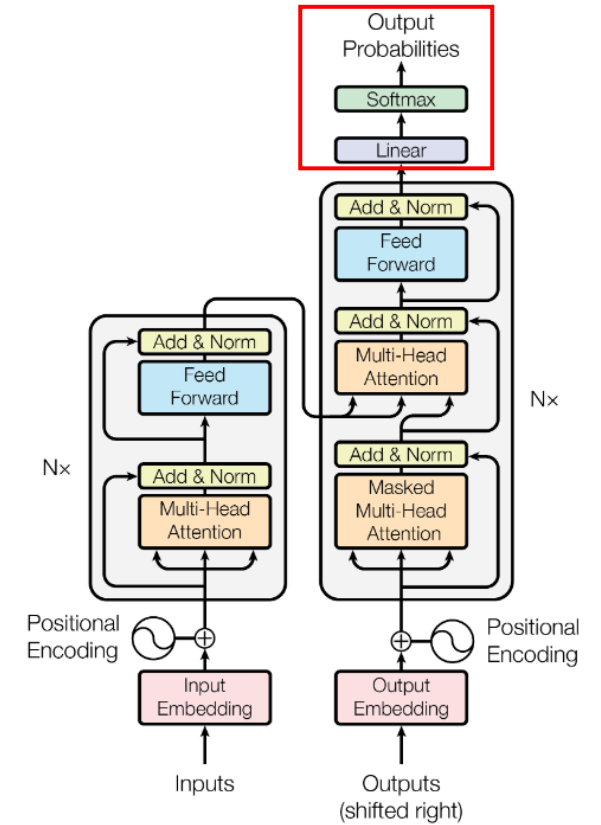
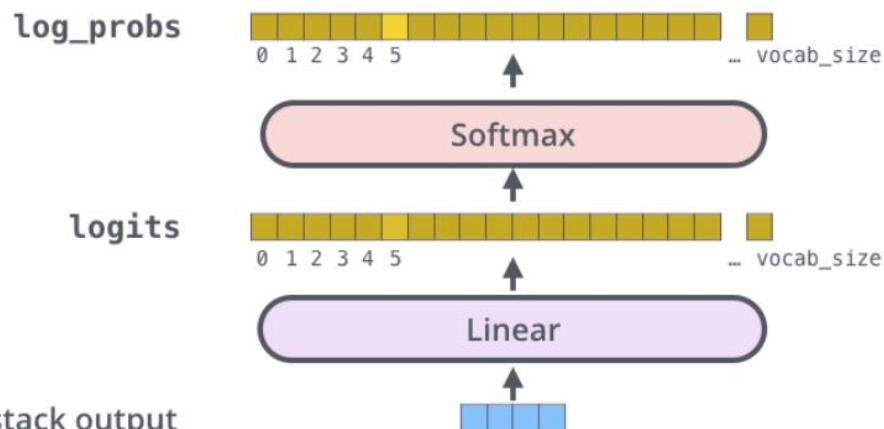


Figure 1: The Transformer - model architecture.

Conclusion

Conclusion

- Transformer, centered around the **self-attention** mechanism, easily learns **long-range dependencies** between input and output, enhancing learning speed through **parallel processing**.
- Transformer exhibits outstanding performance not only in translation but also in various natural language processing tasks such as summarization and question answering.
- Large-scale language models like OpenAI's **GPT** and Google's **BERT** are based on the Transformer architecture.
- The field of Transformer is **continuously evolving**, promising exciting research prospects in the future.

Thank You!

HONGIK UNIVERSITY