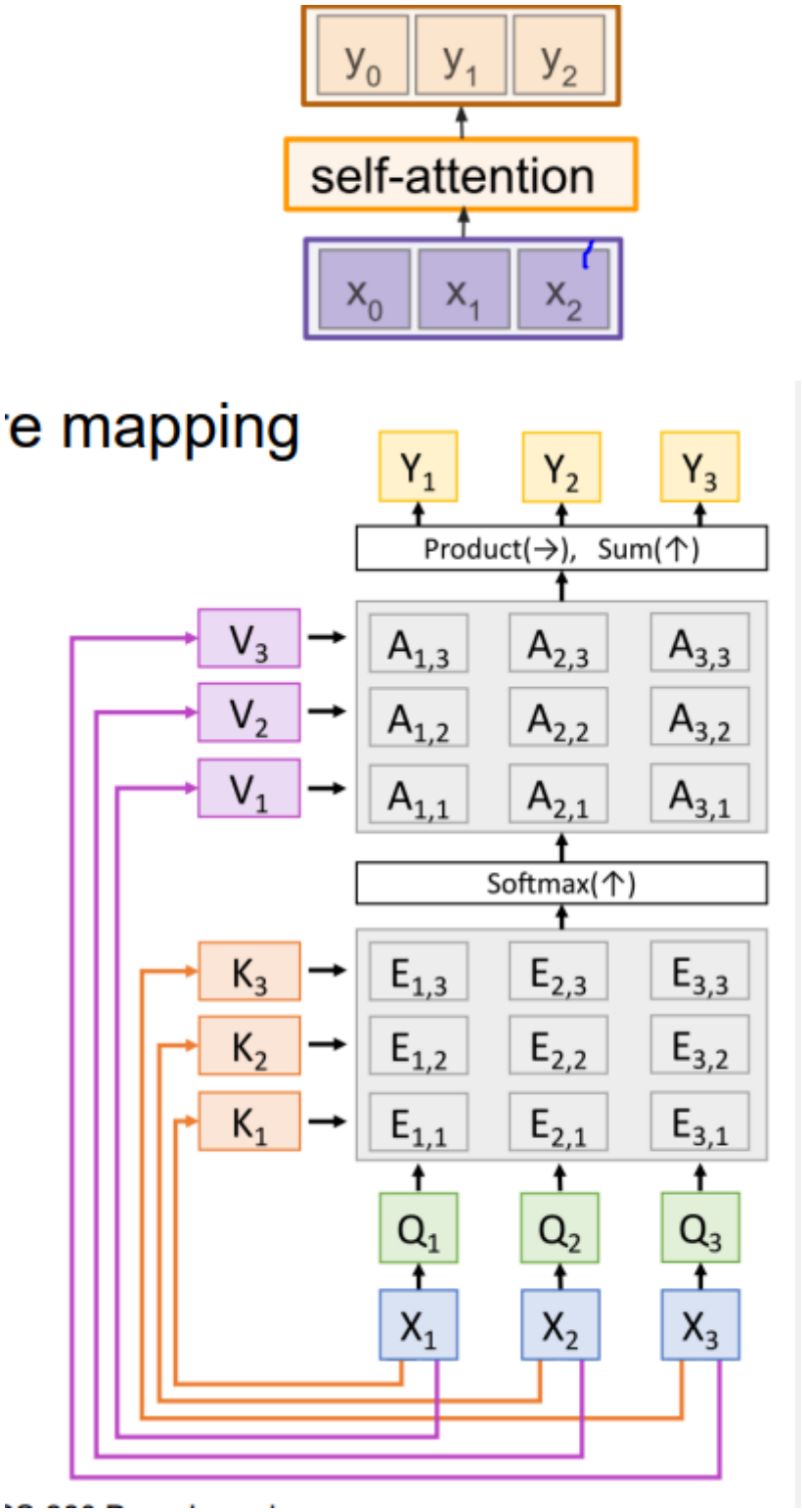
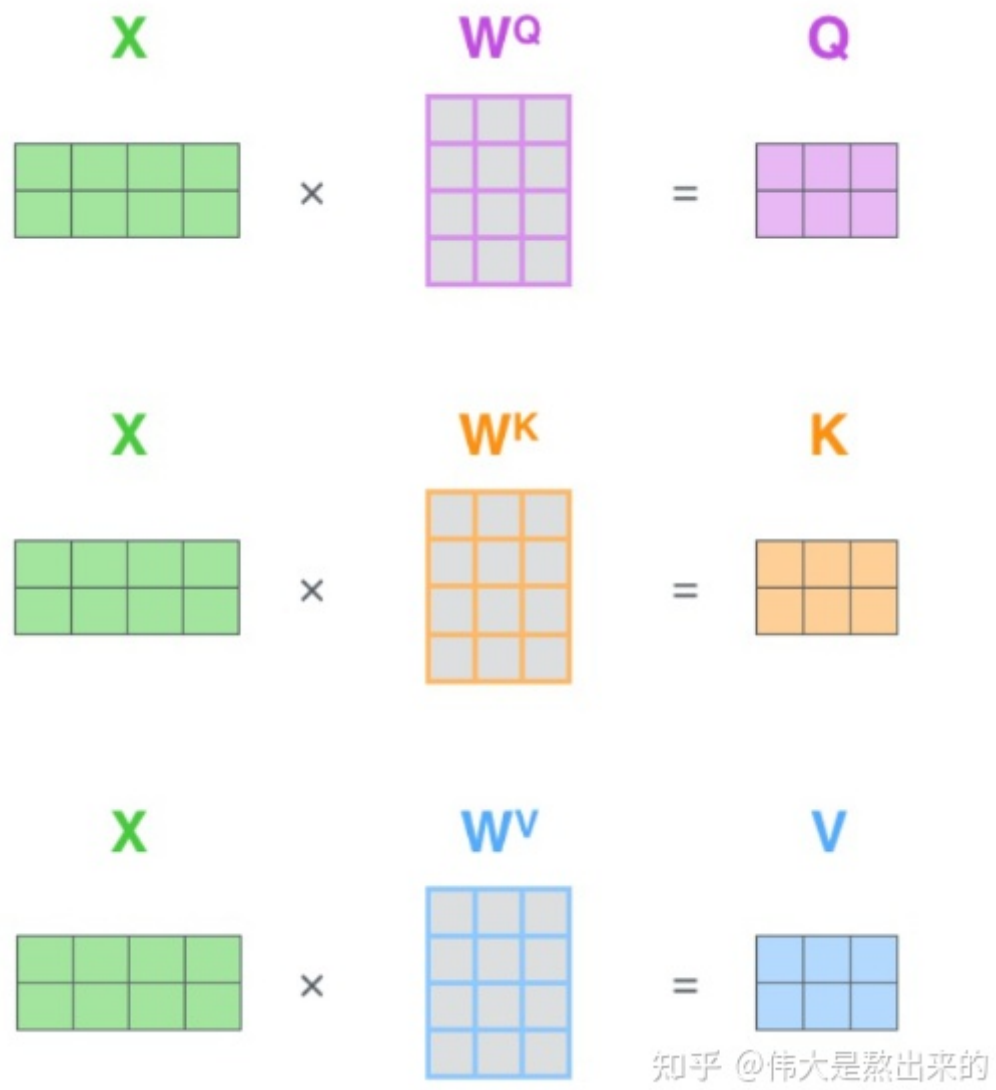
Transformer

Background Knowledge : Self-Attention

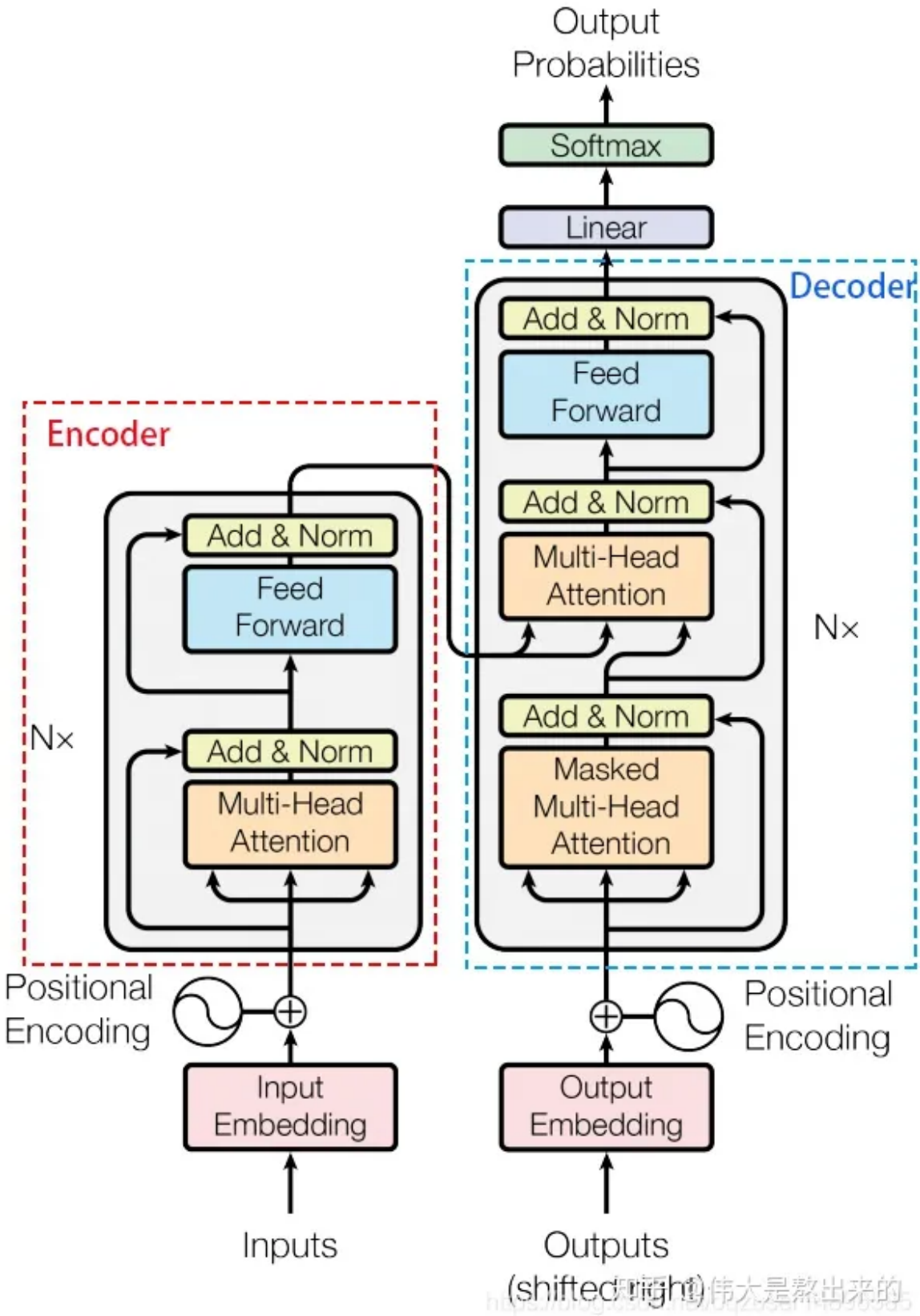


A brief description of what is self-attention, in fact, is to change the original attention as input K vector and Q vector to only need to input feature vector X, and through X directly obtain the relationship between K, Q, V, so to achieve the effect of only input X feature vector can get Y results.



Transformer Pipeline

Encoder has a two-layer structure, self-attention and feed-forward neural network. self-attention calculates the association of each word in a sentence with other words, which helps the model to better understand the contextual semantics, and with the introduction of Muti-Head attention, each head focuses on a different position of the sentence, enhancing the The expressive ability of the Attention mechanism to focus on the role between words within a sentence. Feedforward neural network introduces nonlinear transformation for encoder, which enhances the model fitting ability.
Decoder accepts output input while accepting encoder's input, helping the current node to get the content that needs to be focused on



Multi_Head Attention

In fact, in the Transformer model, the Muti-Head mechanism is used instead of the single self-attention we just explained

$$MutiHead(Q,K,V) = Concat(head_1, head_2, \dots, head_h)W^O$$
$$where, head_i = Attention(QW_i^Q, KW_i^k, VW_i^V)$$

$$W_i^Q \in R^{d_{model} \times d_k}, W_i^k \in R^{d_{model} \times d_k}, W_i^V \in R^{d_{model} \times d_v}, W_O^{\in} R^{hd_v \times d_{model}}$$

Therefore, we can find that **Attention maps query and key to the same high-dimensional space to compute similarity, while the corresponding multi-head attention maps query and key to different subspaces of the high-dimensional space to compute similarity.**

Position-wise Feed Forward

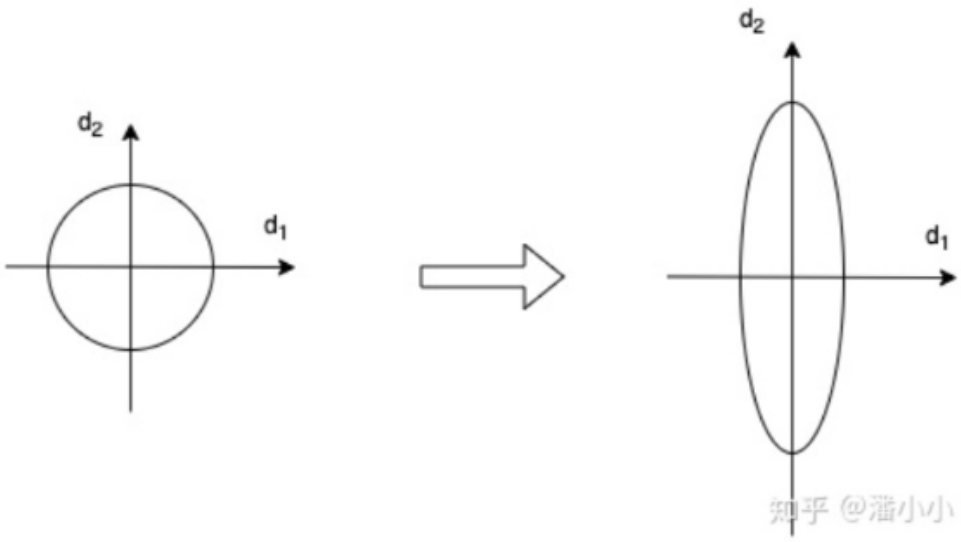
$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

The FFN contains 2 linear transformation layers, and the activation function in the middle is ReLu.In fact, the addition of FFN introduces nonlinearity (ReLu activation function), which transforms the space of attention output, thus increasing the expressiveness of the model. It is possible to remove the FFN from the model, but the effect is much worse.

Layer Normalization

In each block, the last occurrence is Layer Normalization, whose role is to normalize the optimization space and accelerate convergence.

$$LN(x_i) = \alpha \frac{x_i - \mu_i}{\sqrt{\sigma^2 + \xi}} + \beta$$



When we use gradient descent algorithm to do optimization, we may normalize the input data, but after the network layer effect, our data is not normalized anymore. Layer Normalization is used to ensure the stability of the data feature distribution and normalize the data to the region of the ReLU activation function, which can make the activation function work better.

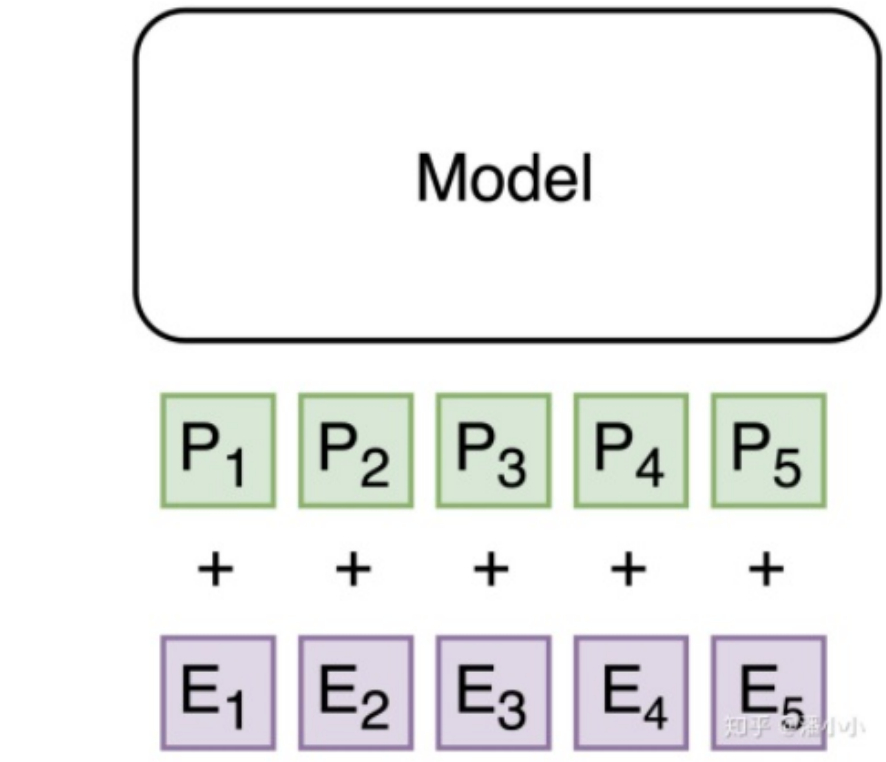
There are two methods of normalization, Batch Normalization and Layer Normalization, and the differences between them are not discussed in detail.

Positional Encoding

Positional information encoding is located after the embedding of encoder and decoder and before each block. Positional Encoding is a transformer-specific mechanism that compensates for the inability of the Attention mechanism to capture the token location information in the sequence.

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

The components of Positional Embedding are directly superimposed on Embedding, so that the positional information of each token and its semantic information (embedding) are fully integrated and passed to all subsequent sequence expressions that undergo complex transformations.



[a] 输入等于Positional Encoding + Embedding Output

The transformer feature allows complete equality between the vectors of the input encoder (no RNN recurrent structure), and the actual position of the token is uniquely bound to the positional information encoding. the introduction of Positional Encoding allows the model to take full advantage of the positional information of the token in the sequence.

The Positional Encoding (PE) used in the paper is a sine cosine function, where the smaller the position (pos), the longer the wavelength, and each position corresponds to a unique PE. Also the authors mention that the sine cosine function is chosen as PE because it allows the model to learn the relative position relations between tokens: since for any offset k , a linear representation of PE(pos+k) can be given by PE(pos)

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