Lecture 09
Transformer based Object Detection

Lin ZHANG, PhD
School of Software Engineering
Tongji University
Fall 2024

Outline

- Transformer in NLP
- Multi-head Attention
- Vision transform (ViT)
- Swin-Transformer
- DETR
- RT-DETR
- Practice of RT-DETR

Transformer in NLP

- Transformer was first proposed by Google Brain in the domain of NLP[1]
  - Transformer encoder is used to transform a set of tokens (vectors) $\mathcal{I}$ into another set of tokens $\mathcal{O}$, where each element in $\mathcal{O}$ can catch the **global information** of $\mathcal{I}$; usually the number of tokens in $\mathcal{I}$ and $\mathcal{O}$ are the same
  - This architecture has since become the foundation for many state-of-the-art NLP models, including BERT and GPT (Generative Pre-trained Transformer)
  - It is now also widely applied in computer vision
- Transformer encoder is composed of basic blocks, including **self-attention**, **positional encoding**, MLP, residual connection and **Layer-norm**
  - A set of vectors is transformed into another different set of vectors; that is why such a structure is called "transformer"

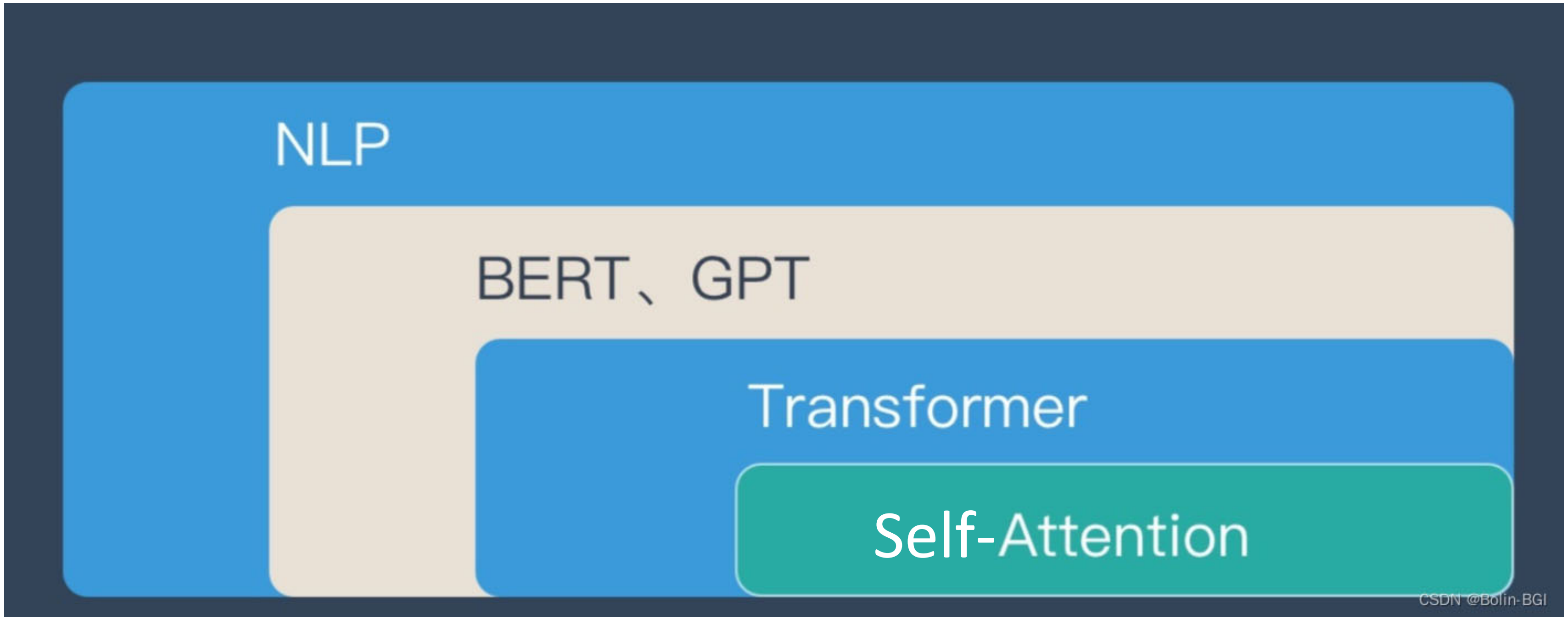
[1] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need[C]//Proc. Adv. Neural Inf. Process. Syst., 2017: 6000-6010. (**Cited by 125704**, Jun. 21, 2024)

Transformer in NLP

Ashish Vaswani (born in 1986) is a computer scientist. He was a co-founder of Adept AI Labs and a former staff research scientist at Google Brain. Vaswani completed his engineering in Computer Science from BIT Mesra (印度贝拉理工学院,梅斯拉) in 2002. In 2004, he moved to the US to pursue higher studies at University of Southern California. He did his PhD at the University of Southern California.

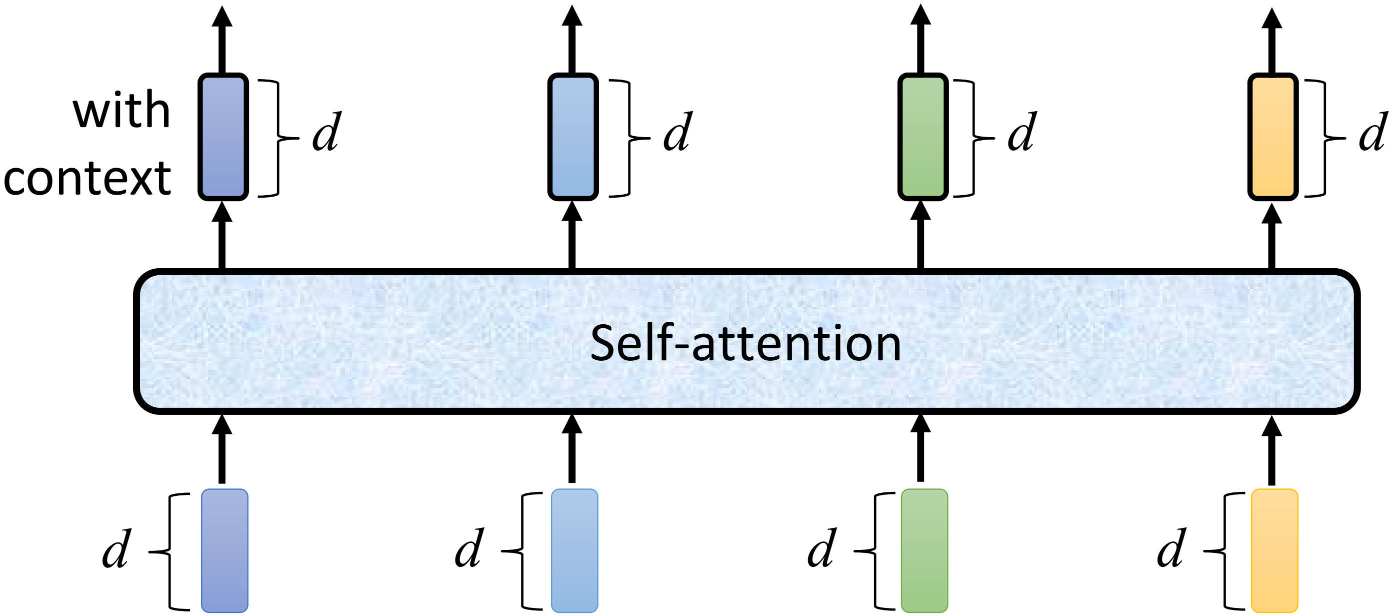
Transformer in NLP

The relationship among several terms, NLP, BERT, GPT, Transformer and self-attention



Self-attention

- It is the core component in Transformer encoder
- Key characteristics of self-attention
  - Input N vectors with dimension d, output N vectors with dimension d
  - Each output vector can capture the **context information of all the input vectors**



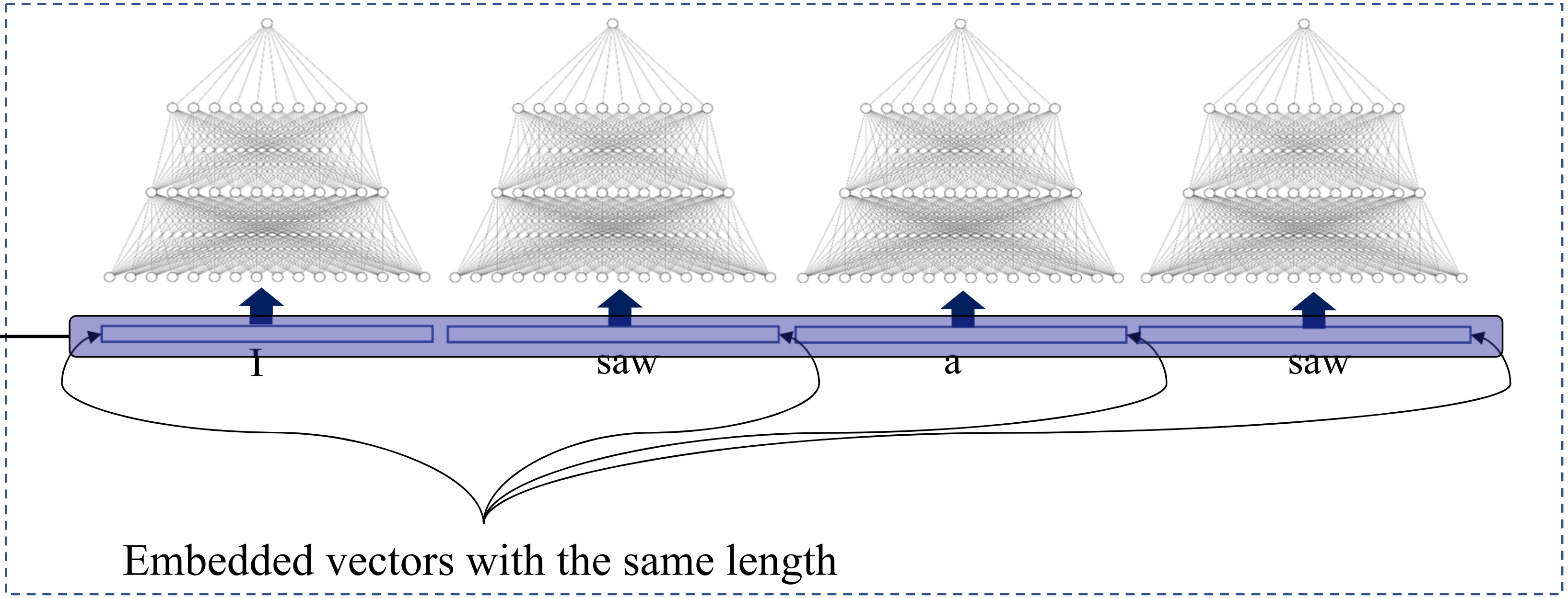
Why do we need SA?

Let's see an example

Self-attention

Ex: word tagging for an input sentence "I saw a saw"

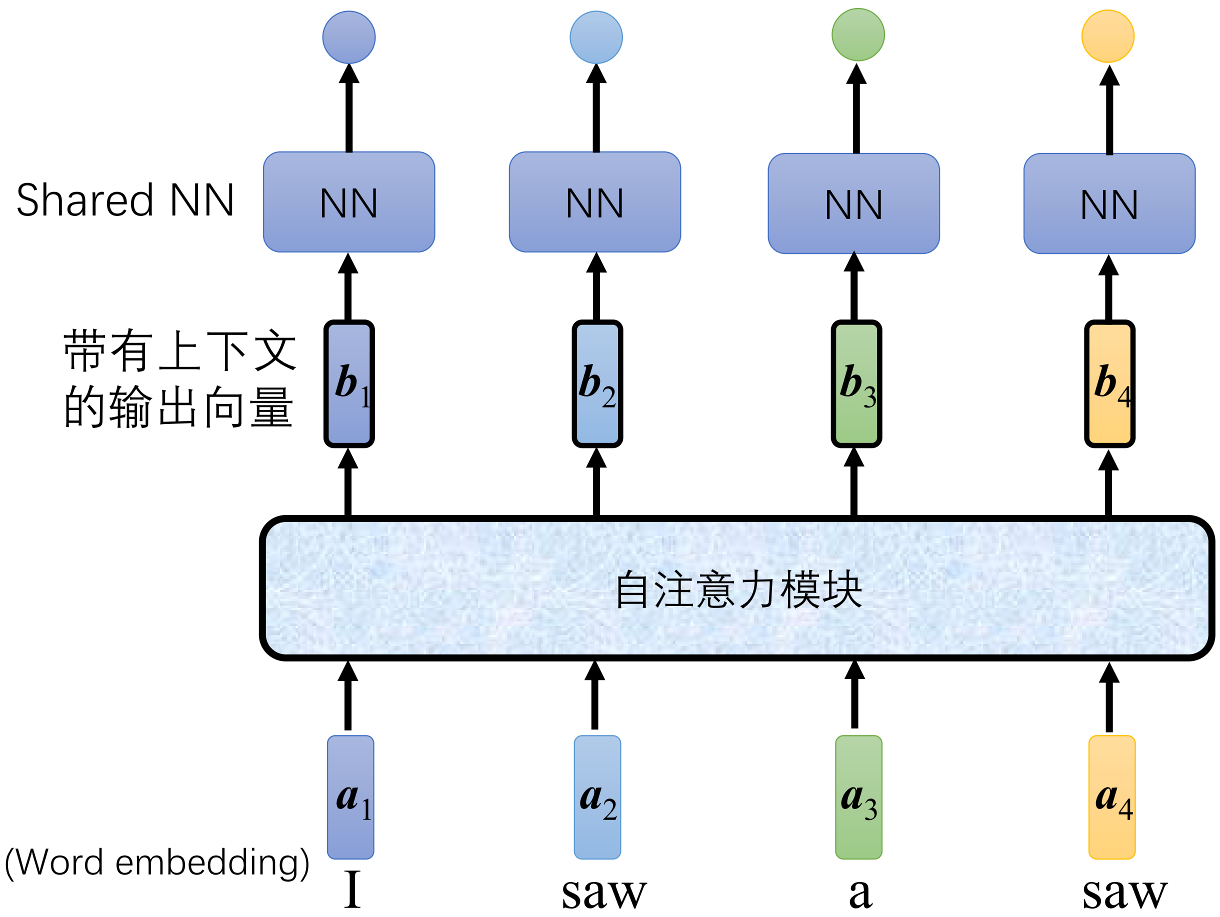
Naïve solution: deal with each word independently using a network



Is it a good solution? No. Tagging for a word depends on the word itself and also its context

The embedded vectors need to be processed to make them hold the global (contextual) information → **Self-attention**

Self-attention



- $\boldsymbol{b}_1 \sim \boldsymbol{b}_4$ are computed from $\boldsymbol{a}_1 \sim \boldsymbol{a}_4$ and can be computed in parallel
- Let's see how to compute $\boldsymbol{b}_1$ in detail; $\boldsymbol{b}_2 \sim \boldsymbol{b}_4$ can be obtained similarly

Output vectors $\boldsymbol{b}_1 \sim \boldsymbol{b}_4 \in \mathbb{R}^{d\times 1}$

Input vectors $\boldsymbol{a}_1 \sim \boldsymbol{a}_4 \in \mathbb{R}^{d\times 1}$

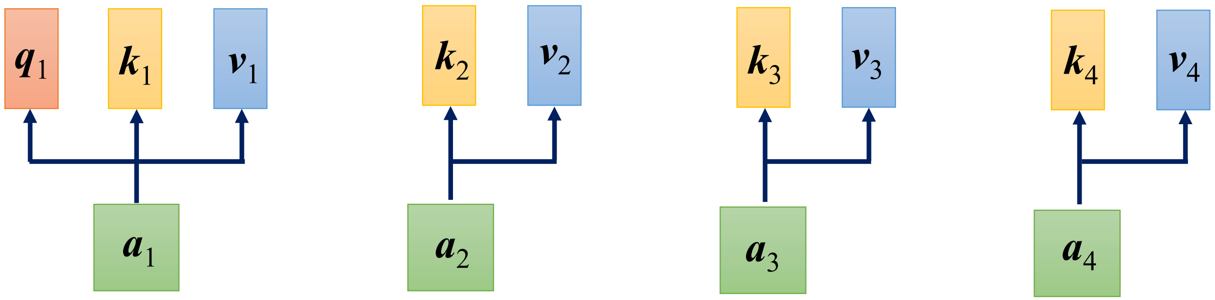
Self-attention

- ✓ $\boldsymbol{W}_q \in \mathbb{R}^{d\times d}, \boldsymbol{W}_k \in \mathbb{R}^{d\times d}, \boldsymbol{W}_v \in \mathbb{R}^{d\times d}$ are the matrices that need **to be learned by training**
- ✓ $\boldsymbol{q}_1$ is the **embedded query** vector
- ✓ $\boldsymbol{k}_1$, $\boldsymbol{k}_2$, $\boldsymbol{k}_3$, and $\boldsymbol{k}_4$ are the **embedded key** vectors
- ✓ $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, $\boldsymbol{v}_3$, and $\boldsymbol{v}_4$ are the **embedded value** vectors

$$\boldsymbol{q}_1 \in \mathbb{R}^{d\times 1}$$

$$\boldsymbol{k}_1, \boldsymbol{k}_2, \boldsymbol{k}_3, \boldsymbol{k}_4 \in \mathbb{R}^{d\times 1}$$

$$\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3, \boldsymbol{v}_4 \in \mathbb{R}^{d\times 1}$$



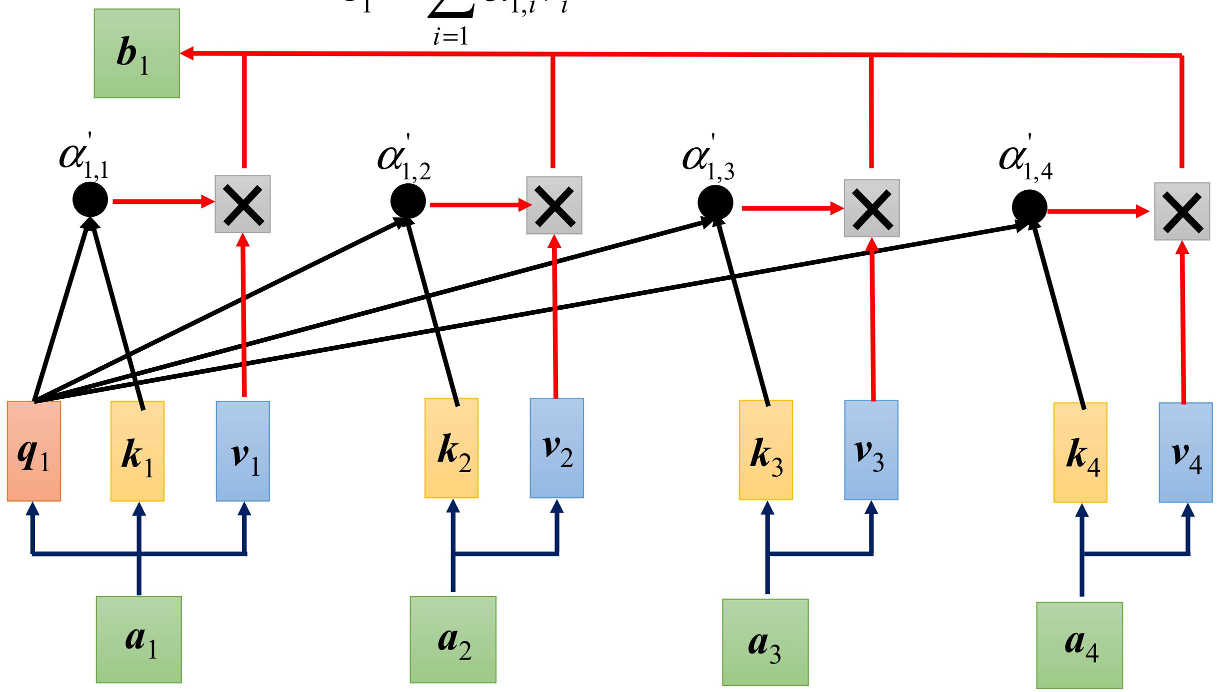
$$\boldsymbol{q}_1 = \boldsymbol{W}_q \boldsymbol{a}_1$$

$$\boldsymbol{k}_1 = \boldsymbol{W}_k \boldsymbol{a}_1 \quad \boldsymbol{k}_2 = \boldsymbol{W}_k \boldsymbol{a}_2 \quad \boldsymbol{k}_3 = \boldsymbol{W}_k \boldsymbol{a}_3 \quad \boldsymbol{k}_4 = \boldsymbol{W}_k \boldsymbol{a}_4$$

$$\boldsymbol{v}_1 = \boldsymbol{W}_v \boldsymbol{a}_1 \quad \boldsymbol{v}_2 = \boldsymbol{W}_v \boldsymbol{a}_2 \quad \boldsymbol{v}_3 = \boldsymbol{W}_v \boldsymbol{a}_3 \quad \boldsymbol{v}_4 = \boldsymbol{W}_v \boldsymbol{a}_4$$

Self-attention

$$b_1 = \sum_{i=1}^{4} \alpha'_{1,i} v_i$$



$q_1 = W_q a_1$

$k_1 = W_k a_1$	$k_2 = W_k a_2$	$k_3 = W_k a_3$	$k_4 = W_k a_4$
$v_1 = W_v a_1$	$v_2 = W_v a_2$	$v_3 = W_v a_3$	$v_4 = W_v a_4$

$$\alpha_{1,1} = k_1 \cdot q_1$$
$$\alpha_{1,2} = k_2 \cdot q_1$$
$$\alpha_{1,3} = k_3 \cdot q_1$$
$$\alpha_{1,4} = k_4 \cdot q_1$$

softmax $\longrightarrow$ $$\alpha'_{1,i} = \frac{\exp\left(\alpha_{1,i} / \sqrt{d}\right)}{\sum_{j=1}^{4} \exp\left(\alpha_{1,j} / \sqrt{d}\right)}$$

$$\alpha_1 \triangleq \begin{pmatrix} \alpha_{1,1} \\ \alpha_{1,2} \\ \alpha_{1,3} \\ \alpha_{1,4} \end{pmatrix}$$

attention scores

softmax $\longrightarrow$

$$\alpha'_1 \triangleq \begin{pmatrix} \alpha'_{1,1} \\ \alpha'_{1,2} \\ \alpha'_{1,3} \\ \alpha'_{1,4} \end{pmatrix}$$

normalized attention scores

b_1 is the average of $\{v_i\}$ weighted by α'_1,

$$b_1 = \sum_{i=1}^{4} \alpha'_{1,i} v_i = [v_1 \; v_2 \; v_3 \; v_4] \alpha'_1$$

$b_2 \sim b_4$ can be computed in the same way in parallel

Self-attention

$\boldsymbol{q}_1=\boldsymbol{W}_q\boldsymbol{a}_1$ $\boldsymbol{q}_2=\boldsymbol{W}_q\boldsymbol{a}_2$ $\boldsymbol{q}_3=\boldsymbol{W}_q\boldsymbol{a}_3$ $\boldsymbol{q}_4=\boldsymbol{W}_q\boldsymbol{a}_4$

$\boldsymbol{k}_1=\boldsymbol{W}_k\boldsymbol{a}_1$ $\boldsymbol{k}_2=\boldsymbol{W}_k\boldsymbol{a}_2$ $\boldsymbol{k}_3=\boldsymbol{W}_k\boldsymbol{a}_3$ $\boldsymbol{k}_4=\boldsymbol{W}_k\boldsymbol{a}_4$

$\boldsymbol{v}_1=\boldsymbol{W}_v\boldsymbol{a}_1$ $\boldsymbol{v}_2=\boldsymbol{W}_v\boldsymbol{a}_2$ $\boldsymbol{v}_3=\boldsymbol{W}_v\boldsymbol{a}_3$ $\boldsymbol{v}_4=\boldsymbol{W}_v\boldsymbol{a}_4$

$$\boldsymbol{\alpha}_2 \triangleq \begin{pmatrix} \alpha_{2,1}=\boldsymbol{k}_1\cdot\boldsymbol{q}_2 \\ \alpha_{2,2}=\boldsymbol{k}_2\cdot\boldsymbol{q}_2 \\ \alpha_{2,3}=\boldsymbol{k}_3\cdot\boldsymbol{q}_2 \\ \alpha_{2,4}=\boldsymbol{k}_4\cdot\boldsymbol{q}_2 \end{pmatrix} \xrightarrow{\text{softmax}} \boldsymbol{\alpha}_2' \triangleq \begin{pmatrix} \alpha_{2,1}' \\ \alpha_{2,2}' \\ \alpha_{2,3}' \\ \alpha_{2,4}' \end{pmatrix} \Rightarrow \boldsymbol{b}_2=\sum_{i=1}^{4}\alpha_{2,i}'\boldsymbol{v}_i=[\boldsymbol{v}_1\ \boldsymbol{v}_2\ \boldsymbol{v}_3\ \boldsymbol{v}_4]\boldsymbol{\alpha}_2'$$

$$\boldsymbol{\alpha}_3 \triangleq \begin{pmatrix} \alpha_{3,1}=\boldsymbol{k}_1\cdot\boldsymbol{q}_3 \\ \alpha_{3,2}=\boldsymbol{k}_2\cdot\boldsymbol{q}_3 \\ \alpha_{3,3}=\boldsymbol{k}_3\cdot\boldsymbol{q}_3 \\ \alpha_{3,4}=\boldsymbol{k}_4\cdot\boldsymbol{q}_3 \end{pmatrix} \xrightarrow{\text{softmax}} \boldsymbol{\alpha}_3' \triangleq \begin{pmatrix} \alpha_{3,1}' \\ \alpha_{3,2}' \\ \alpha_{3,3}' \\ \alpha_{3,4}' \end{pmatrix} \Rightarrow \boldsymbol{b}_3=\sum_{i=1}^{4}\alpha_{3,i}'\boldsymbol{v}_i=[\boldsymbol{v}_1\ \boldsymbol{v}_2\ \boldsymbol{v}_3\ \boldsymbol{v}_4]\boldsymbol{\alpha}_3'$$

$$\boldsymbol{\alpha}_4 \triangleq \begin{pmatrix} \alpha_{4,1}=\boldsymbol{k}_1\cdot\boldsymbol{q}_4 \\ \alpha_{4,2}=\boldsymbol{k}_2\cdot\boldsymbol{q}_4 \\ \alpha_{4,3}=\boldsymbol{k}_3\cdot\boldsymbol{q}_4 \\ \alpha_{4,4}=\boldsymbol{k}_4\cdot\boldsymbol{q}_4 \end{pmatrix} \xrightarrow{\text{softmax}} \boldsymbol{\alpha}_4' \triangleq \begin{pmatrix} \alpha_{4,1}' \\ \alpha_{4,2}' \\ \alpha_{4,3}' \\ \alpha_{4,4}' \end{pmatrix} \Rightarrow \boldsymbol{b}_4=\sum_{i=1}^{4}\alpha_{4,i}'\boldsymbol{v}_i=[\boldsymbol{v}_1\ \boldsymbol{v}_2\ \boldsymbol{v}_3\ \boldsymbol{v}_4]\boldsymbol{\alpha}_4'$$

The whole process can be represented in matrix form

Self-attention

Let $I_{d\times4} \triangleq [a_1\ a_2\ a_3\ a_4]$, $Q_{d\times4} \triangleq [q_1\ q_2\ q_3\ q_4]$, $K_{d\times4} \triangleq [k_1\ k_2\ k_3\ k_4]$, $V_{d\times4} \triangleq [v_1\ v_2\ v_3\ v_4]$, $A_{4\times4} \triangleq [\alpha_1\ \alpha_2\ \alpha_3\ \alpha_4]$, $A'_{4\times4} = [\alpha'_1\ \alpha'_2\ \alpha'_3\ \alpha'_4]$ and $O_{d\times4} \triangleq [b_1\ b_2\ b_3\ b_4]$

We have,

$$Q_{d\times4} = [q_1\ q_2\ q_3\ q_4] = [W_q a_1\ W_q a_2\ W_q a_3\ W_q a_4] = W_q I_{d\times4}$$

$$K_{d\times4} = [k_1\ k_2\ k_3\ k_4] = [W_k a_1\ W_k a_2\ W_k a_3\ W_k a_4] = W_k I_{d\times4}$$

$$V_{d\times4} = [v_1\ v_2\ v_3\ v_4] = [W_v a_1\ W_v a_2\ W_v a_3\ W_v a_4] = W_v I_{d\times4}$$

$$\alpha_1 = \begin{pmatrix} \alpha_{1,1} = k_1 \cdot q_1 \\ \alpha_{1,2} = k_2 \cdot q_1 \\ \alpha_{1,3} = k_3 \cdot q_1 \\ \alpha_{1,4} = k_4 \cdot q_1 \end{pmatrix} = \begin{pmatrix} k_1^T q_1 \\ k_2^T q_1 \\ k_3^T q_1 \\ k_4^T q_1 \end{pmatrix} = K^T q_1 \qquad \alpha_2 = K^T q_2 \qquad \alpha_3 = K^T q_3 \qquad \alpha_4 = K^T q_4$$

softmax is applied to each column vector

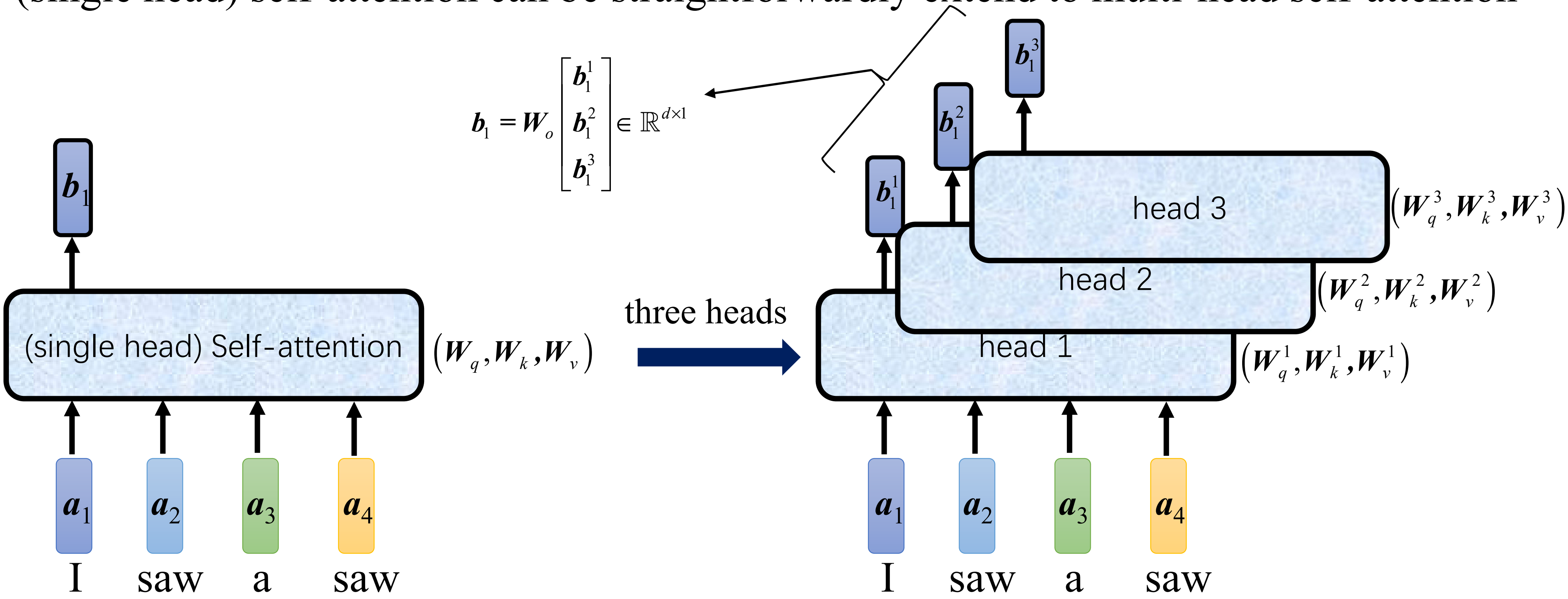
$$A_{4\times4} = [\alpha_1\ \alpha_2\ \alpha_3\ \alpha_4] = [K^T q_1\ K^T q_2\ K^T q_3\ K^T q_4] = K^T[q_1\ q_2\ q_3\ q_4] = K^T Q$$

$$A'_{4\times4} = [\alpha'_1\ \alpha'_2\ \alpha'_3\ \alpha'_4] = \left[softmax\left(\frac{\alpha_1}{\sqrt{d}}\right) softmax\left(\frac{\alpha_2}{\sqrt{d}}\right) softmax\left(\frac{\alpha_3}{\sqrt{d}}\right) softmax\left(\frac{\alpha_4}{\sqrt{d}}\right) \right] = softmax\left(\frac{A}{\sqrt{d}}\right) = softmax\left(\frac{K^T Q}{\sqrt{d}}\right)$$

$$O = [b_1\ b_2\ b_3\ b_4] = [V_{d\times4}\alpha'_1\ V_{d\times4}\alpha'_2\ V_{d\times4}\alpha'_3\ V_{d\times4}\alpha'_4] = V_{d\times4}[\alpha'_1\ \alpha'_2\ \alpha'_3\ \alpha'_4] = V_{d\times4} A'_{4\times4}$$

Multi-head Self-attention (MSA)

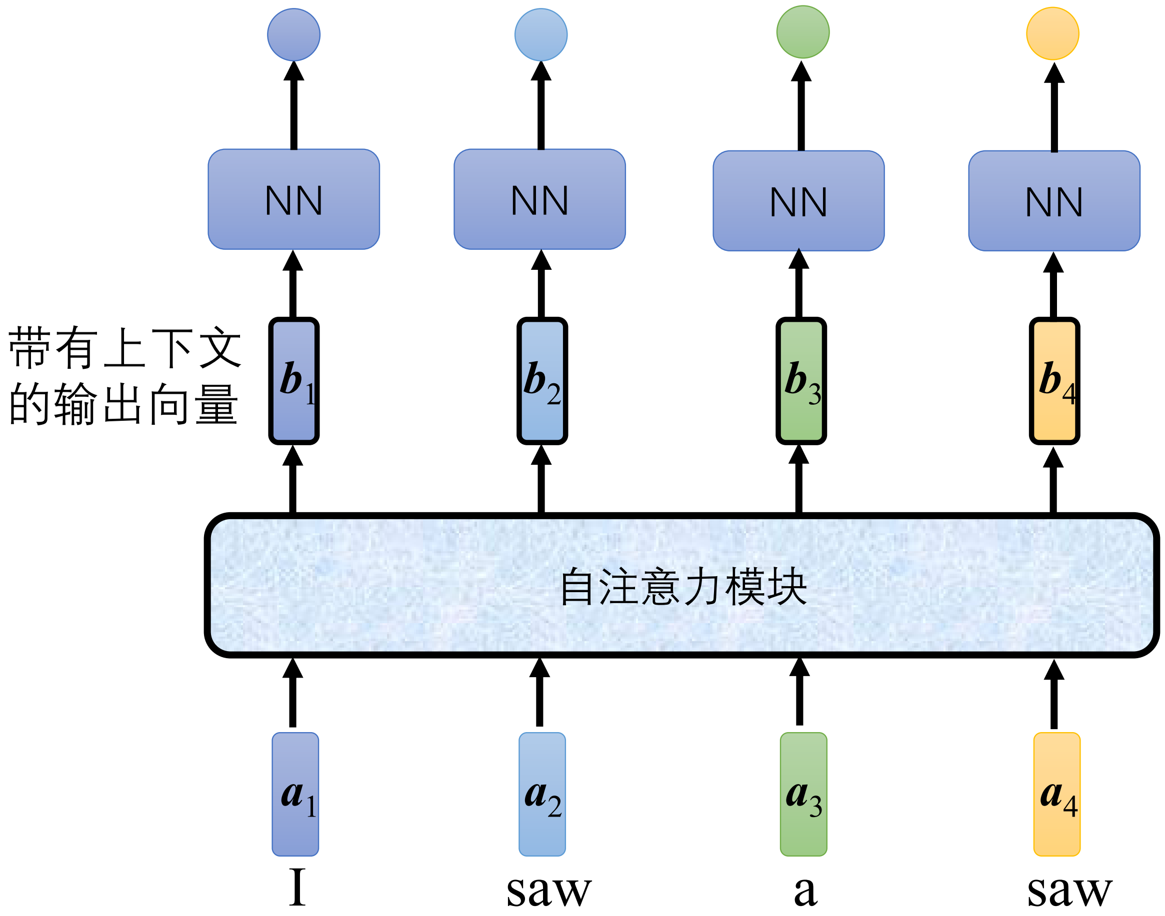
(single head) self-attention can be straightforwardly extend to multi-head self-attention



$W_q^i \in \mathbb{R}^{\frac{d}{3}\times d}$, $W_k^i \in \mathbb{R}^{\frac{d}{3}\times d}$, $W_v^i \in \mathbb{R}^{\frac{d}{3}\times d}$, $b_1^i \in \mathbb{R}^{\frac{d}{3}\times 1}$, $W_o \in \mathbb{R}^{d\times d}$

Positional encoding

- Can the self-attention really solve our word-tagging problem?



✓ Using the self-attention, actually $\boldsymbol{b}_2$ and $\boldsymbol{b}_4$ are the same!

✓ Accordingly, the two "saw"s will be predicted to have the same tagging

What do we miss?

The position information of the vectors in the input sequence

That the two "saw"s have different tagging largely owes to the fact that they have different positions in the sequence

We need to modify the input vectors by embedding their positional information

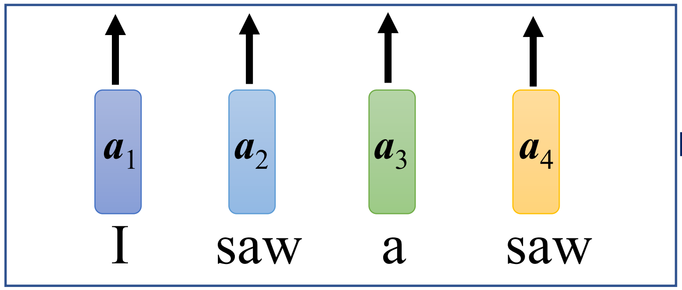
Positional encoding

For each input vector $\boldsymbol{a}_t \in \mathbb{R}^{d\times 1}$, construct a positional encoding vector $\boldsymbol{pe}_t \triangleq \left\{ pe_t^{(i)} \right\}_{i=0}^{d-1}$

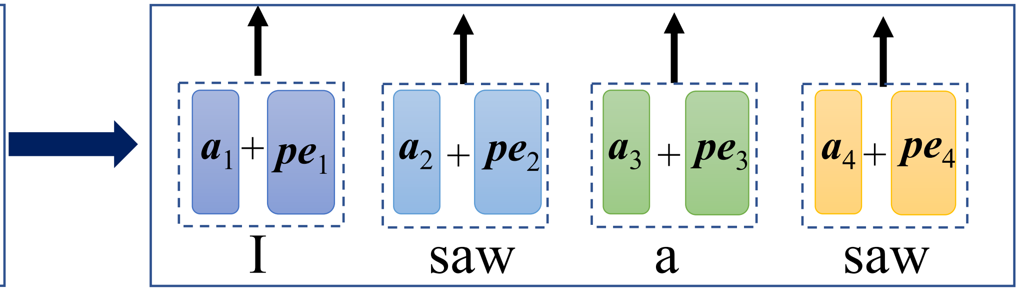
$$pe_t^{(i)} = \begin{cases} \sin(w_k t), & if\ i = 2k \\ \cos(w_k t), & if\ i = 2k+1 \end{cases}$$

where t is the position of this input vector in the sequence, $w_k = \frac{1}{10000^{2k/d}}$, k=0, 1, 2,…, d/2-1

Positional encoding means that we modify $\boldsymbol{a}_t$ as $\boldsymbol{a}_t + \boldsymbol{pe}_t$



without positional encoding



with positional encoding

Layer-norm VS batch-norm

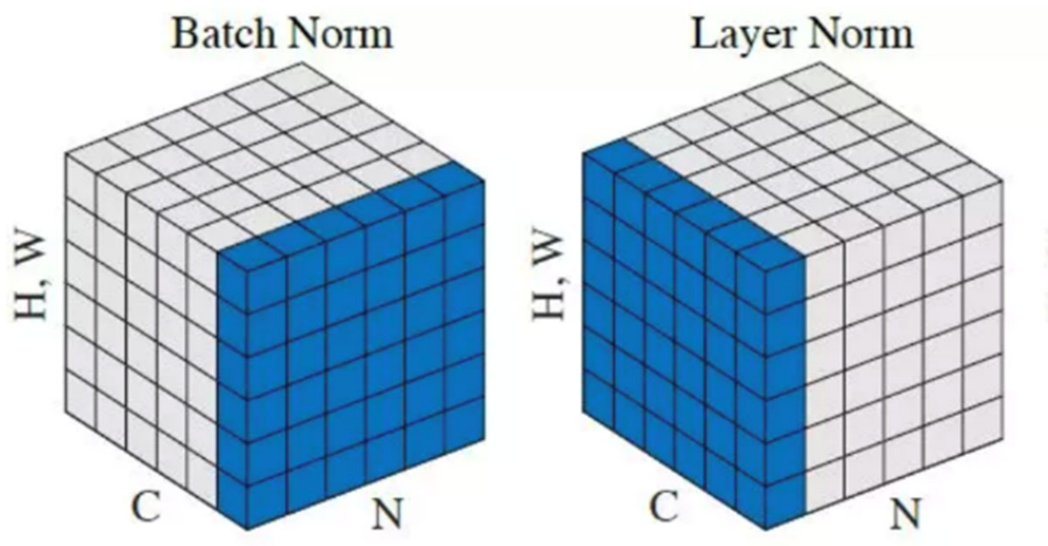
- Batch-norm and layer-norm are two strategies for training neural networks faster and more stably
- In Yolov3, we have met batch-norm; in transformer-related structures, in most cases, they use layer-norm
- Batch-norm VS layer-norm
  - **Batch-norm normalizes each feature (channel) independently across the samples in a mini-batch**
  - **Layer-norm normalizes each sample in the mini-batch independently across all features (channels)**
  - As batch-norm is dependent on batch size, it's not effective for small batch sizes

Layer-norm VS batch-norm

They use the same updating formular, but adopt different ways to compute statistics (μ, σ^2),

$$y_i \leftarrow \gamma \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}} + \beta$$

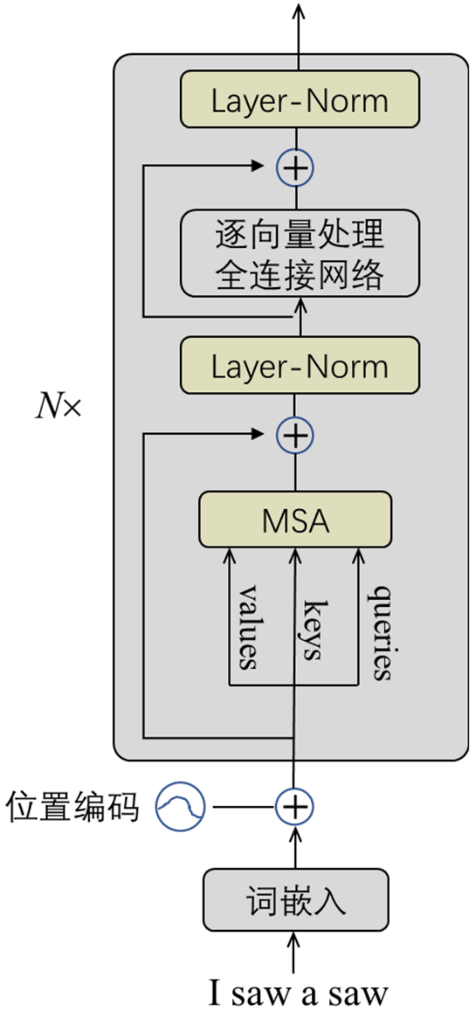
where γ and β are learnable parameters; for batch-norm, each neuron (channel) has a (γ, β) pair while for layer-norm each layer has a (γ, β) pair



- ✓ C means the feature channels, $H\times W$ defines the instances of a feature by a sample, and N means the number of samples in a mini-batch
- ✓ The normalization is applied to the blue part

For our case, C=d, $H\times W$=4, N=1

Transformer encoder



- ✓ The inputs are embedded in a sequence of vectors with the same length
- ✓ Then, the embedded vectors are modified with positional encoding
- ✓ Transformer encoder is composed of *N* blocks with the same structures
- ✓ Each block processes the input vectors by a multi-head self-attention layer, a residual connection layer, a layer-norm operation, a vector-wise fully connected MLP, another residual connection layer, and another layer-norm
- ✓ If the dimension of the input is $d \times N$, the output of a transformer encoder will also be $d \times N$

Outline

- Transformer in NLP
- Multi-head Attention
- Vision transform (ViT)
- Swin-Transformer
- DETR
- RT-DETR
- Practice of RT-DETR

Multi-head Attention

Self-attention

Let $\boldsymbol{I}_{d\times 4} \triangleq [\boldsymbol{a}_1\ \boldsymbol{a}_2\ \boldsymbol{a}_3\ \boldsymbol{a}_4]$, $\boldsymbol{Q}_{d\times 4} \triangleq [\boldsymbol{q}_1\ \boldsymbol{q}_2\ \boldsymbol{q}_3\ \boldsymbol{q}_4]$, $\boldsymbol{K}_{d\times 4} \triangleq [\boldsymbol{k}_1\ \boldsymbol{k}_2\ \boldsymbol{k}_3\ \boldsymbol{k}_4]$, $\boldsymbol{V}_{d\times 4} \triangleq [\boldsymbol{v}_1\ \boldsymbol{v}_2\ \boldsymbol{v}_3\ \boldsymbol{v}_4]$, $\boldsymbol{A}_{4\times 4} \triangleq [\boldsymbol{\alpha}_1\ \boldsymbol{\alpha}_2\ \boldsymbol{\alpha}_3\ \boldsymbol{\alpha}_4]$, $\boldsymbol{A}'_{4\times 4} = [\boldsymbol{\alpha}'_1\ \boldsymbol{\alpha}'_2\ \boldsymbol{\alpha}'_3\ \boldsymbol{\alpha}'_4]$ and $\boldsymbol{O}_{d\times 4} \triangleq [\boldsymbol{b}_1\ \boldsymbol{b}_2\ \boldsymbol{b}_3\ \boldsymbol{b}_4]$

We have,

$$\boldsymbol{Q}_{d\times 4} = [\boldsymbol{q}_1\ \boldsymbol{q}_2\ \boldsymbol{q}_3\ \boldsymbol{q}_4] = [\boldsymbol{W}_q\boldsymbol{a}_1\ \boldsymbol{W}_q\boldsymbol{a}_2\ \boldsymbol{W}_q\boldsymbol{a}_3\ \boldsymbol{W}_q\boldsymbol{a}_4] = \boldsymbol{W}_q\boldsymbol{I}_{d\times 4}$$ → **query sequence**

$$\boldsymbol{K}_{d\times 4} = [\boldsymbol{k}_1\ \boldsymbol{k}_2\ \boldsymbol{k}_3\ \boldsymbol{k}_4] = [\boldsymbol{W}_k\boldsymbol{a}_1\ \boldsymbol{W}_k\boldsymbol{a}_2\ \boldsymbol{W}_k\boldsymbol{a}_3\ \boldsymbol{W}_k\boldsymbol{a}_4] = \boldsymbol{W}_k\boldsymbol{I}_{d\times 4}$$ → **key sequence**

$$\boldsymbol{V}_{d\times 4} = [\boldsymbol{v}_1\ \boldsymbol{v}_2\ \boldsymbol{v}_3\ \boldsymbol{v}_4] = [\boldsymbol{W}_v\boldsymbol{a}_1\ \boldsymbol{W}_v\boldsymbol{a}_2\ \boldsymbol{W}_v\boldsymbol{a}_3\ \boldsymbol{W}_v\boldsymbol{a}_4] = \boldsymbol{W}_v\boldsymbol{I}_{d\times 4}$$ → **value sequence**

In self-attention, the query sequence, the key sequence and the value sequence are actually identical; that is why it is called **self**-attention.

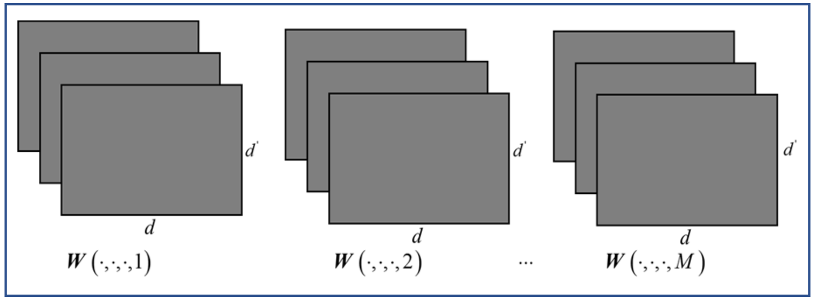
If the key sequence and the value sequence are the same while the query sequence is different, the self-attention changes to **attention**. In other words, self-attention is a special case of attention

Multi-head Attention

- Multi-head attention is an extension to the multi-head self-attention; their computation frameworks are the same, except that **in multi-head attention, the query sequence is different from the key and value sequences**
- Multi-head attention can be used in **transformer decoders**

A multi-head attention module with M heads can be expressed as a function mh-attn,

$$\text{mh-attn}\left(\underbrace{\boldsymbol{I}_q}_{d\times N_q}, \underbrace{\boldsymbol{I}_{kv}}_{d\times N_{kv}}, \underbrace{\boldsymbol{W}}_{d'\times d\times 3\times M}, \underbrace{\boldsymbol{W}_o}_{d\times d}\right) :\mapsto \underbrace{\boldsymbol{O}}_{d\times N_q}$$



$\boldsymbol{I}_q \in \mathbb{R}^{d\times N_q}$ is the query sequence; $\boldsymbol{I}_{kv} \in \mathbb{R}^{d\times N_{kv}}$ is the key/value sequence

$\boldsymbol{W} \in \mathbb{R}^{d'\times d\times 3\times M}$ is the weight tensor, where $d' = \frac{d}{M}$ is the dimension of the embedded vectors in each single-head

$\boldsymbol{W}_o \in \mathbb{R}^{d\times d}$ linearly maps the concatenation of the outputs of single-heads to a space of dimension d

$\boldsymbol{O} \in \mathbb{R}^{d\times N_q}$ is the final output

Remember: each query generates an output vector

Multi-head Attention

Multi-head attention

$$\text{mh-attn}\left(\underbrace{\boldsymbol{I}_q}_{d\times N_q}, \underbrace{\boldsymbol{I}_{kv}}_{d\times N_{kv}}, \underbrace{\boldsymbol{W}}_{d'\times d\times 3\times M}, \underbrace{\boldsymbol{W}_o}_{d\times d}\right) :\mapsto \underbrace{\boldsymbol{O}}_{d\times N_q}$$

Similar as multi-head self-attention, the final output of a MHA module is generated by

- ✓ Concatenating the outputs of all the single-heads
- ✓ then perform a linear mapping using $\boldsymbol{W}_o$

$$\boldsymbol{O}'_{d\times N_q} = \left[\text{attn}\left(\boldsymbol{I}_q, \boldsymbol{I}_{kv}, \boldsymbol{W}_1\right); \text{attn}\left(\boldsymbol{I}_q, \boldsymbol{I}_{kv}, \boldsymbol{W}_2\right); \cdots; \text{attn}\left(\boldsymbol{I}_q, \boldsymbol{I}_{kv}, \boldsymbol{W}_M\right)\right]$$

$$\boldsymbol{O}_{d\times N_q} = \boldsymbol{W}_o \boldsymbol{O}'_{d\times N_q}$$

where $\text{attn}\left(\boldsymbol{I}_q, \boldsymbol{I}_{kv}, \boldsymbol{W}_h\right) \in \mathbb{R}^{d'\times N_q}$ is the output of the h^{th} single-head and $\boldsymbol{W}_h \in \mathbb{R}^{d'\times d\times 3}$ is the h^{th} "slice" of the tensor $\boldsymbol{W}$; [;] denotes the channel-wise concatenation

Single-head

$$\text{attn}\left(\boldsymbol{I}_q, \boldsymbol{I}_{kv}, \boldsymbol{W}'\right) \in \mathbb{R}^{d'\times N_q}$$

where $\boldsymbol{W}' = \left[\boldsymbol{W}_1'; \boldsymbol{W}_2'; \boldsymbol{W}_3'\right] \in \mathbb{R}^{d'\times d\times 3}$ is the weight tensor for this head; $\boldsymbol{W}_1' \in \mathbb{R}^{d'\times d}$, $\boldsymbol{W}_2' \in \mathbb{R}^{d'\times d}$, $\boldsymbol{W}_3' \in \mathbb{R}^{d'\times d}$ are used to compute embedded queries, keys, and values, respectively

Positional encoding

$$\boldsymbol{Q}_{d'\times N_q} = \boldsymbol{W}_1'\left(\boldsymbol{I}_q + \boldsymbol{P}_q\right), \boldsymbol{K}_{d'\times N_{kv}} = \boldsymbol{W}_2'\left(\boldsymbol{I}_{kv} + \boldsymbol{P}_{kv}\right), \boldsymbol{V}_{d'\times N_{kv}} = \boldsymbol{W}_3'\boldsymbol{I}_{kv}$$

Compute the normalized attention scores between $\boldsymbol{q}_i$ and $\boldsymbol{K}$,

$$\boldsymbol{\alpha}_i \triangleq \left(\alpha_{i,1}, \alpha_{i,2}, \ldots, \alpha_{i,N_{kv}}\right)^T, \quad \alpha_{i,j} = \frac{\exp\left(\boldsymbol{k}_j^T \boldsymbol{q}_i / \sqrt{d'}\right)}{\sum_{k=1}^{N_{kv}} \exp\left(\boldsymbol{k}_k^T \boldsymbol{q}_i / \sqrt{d'}\right)}$$

The i-th output vector of this head is,

$$\text{attn}_i\left(\boldsymbol{I}_q, \boldsymbol{I}_{kv}, \boldsymbol{W}'\right) = \sum_{j=1}^{N_{kv}} \alpha_{ij} \boldsymbol{v}_j = \boldsymbol{V}\boldsymbol{\alpha}_i \in \mathbb{R}^{d'\times 1}$$

Outline

- Transformer in NLP
- Multi-head Attention
- Vision transform (ViT)
- Swin-Transformer
- DETR
- RT-DETR
- Practice of RT-DETR

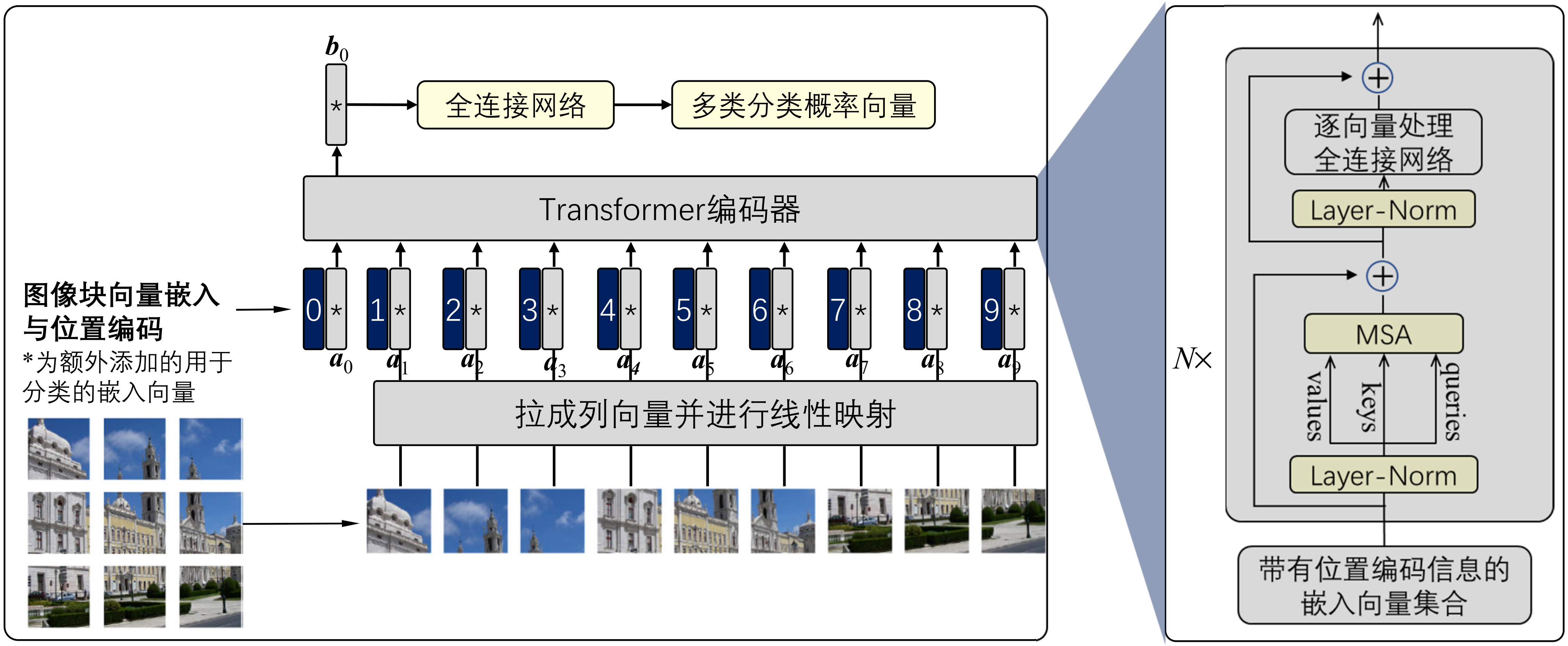
Vision Transformer (ViT)

- Inspired by the great success of transformer in the field of NLP, researchers in CV have started to adopt transformers in CV tasks
- The first image classification framework totally based on transformer is ViT (Vision Transformer) proposed also by researchers of Google in 2021

Alexey Dosovitskiy received the M.Sc. and Ph.D. degrees in mathematics (functional analysis) from Moscow State University, Moscow, Russia, in 2009 and 2012, respectively.,He is currently a Research Scientist with the Intelligent Systems Laboratory, Intel, Munich, Germany. From 2013 to 2016, he was a Postdoctoral Researcher, with Prof. T. Brox, with the Computer Vision Group, University of Freiburg, Breisgau, Germany, working on various topics in deep learning, including self-supervised learning, image generation with neural networks, motion, and 3-D structure estimation.

[1] DOSOVITSKIY A, BEYER L, KOLESNIKOV A, et al. An image is worth 16x16 words: Transformers for image recognition at scale[C]//Proc. Int'l. Conf. Learning Representations, 2021. (**Cited by 38008**, Jun. 25, 2024)

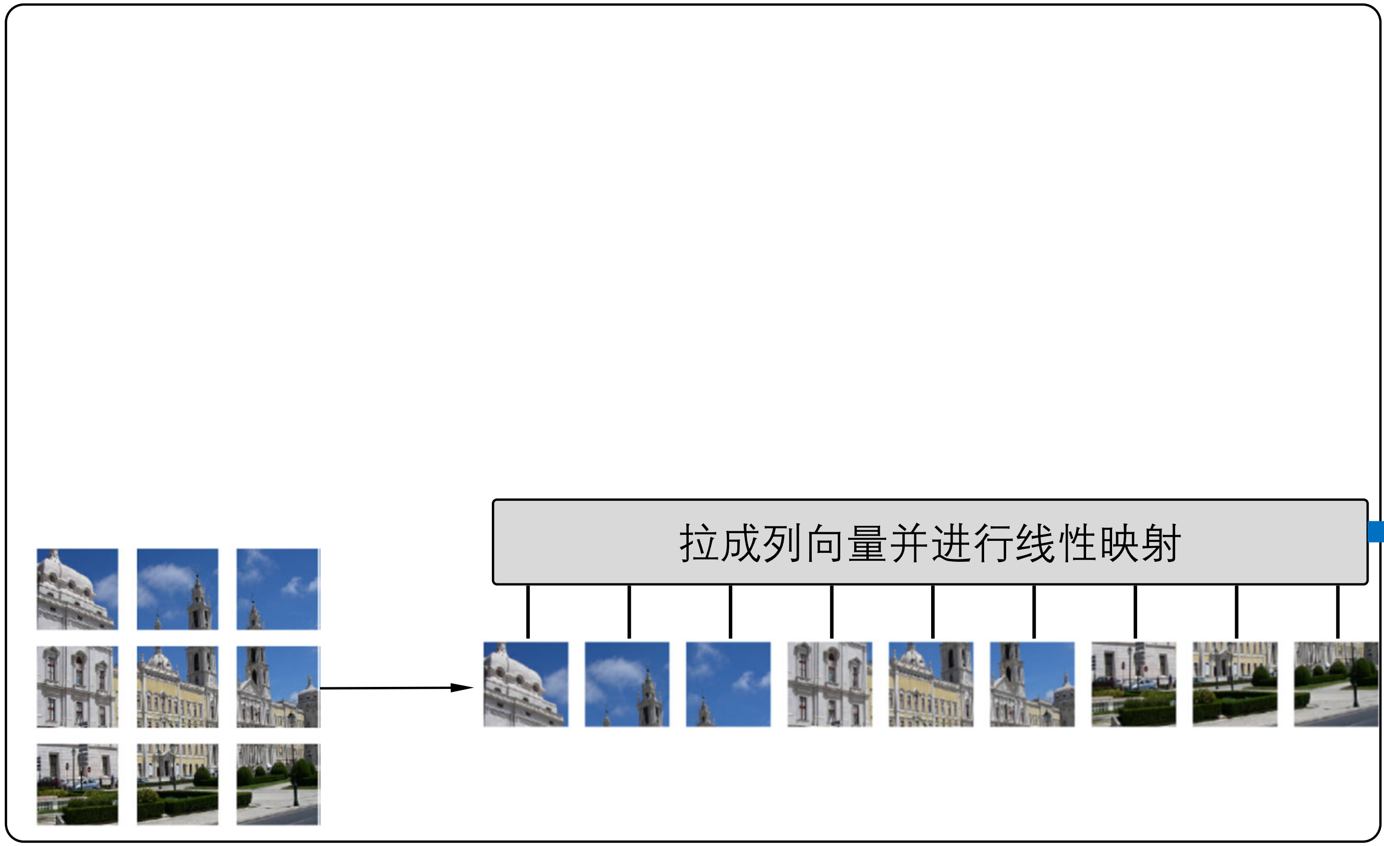
Vision Transformer (ViT)

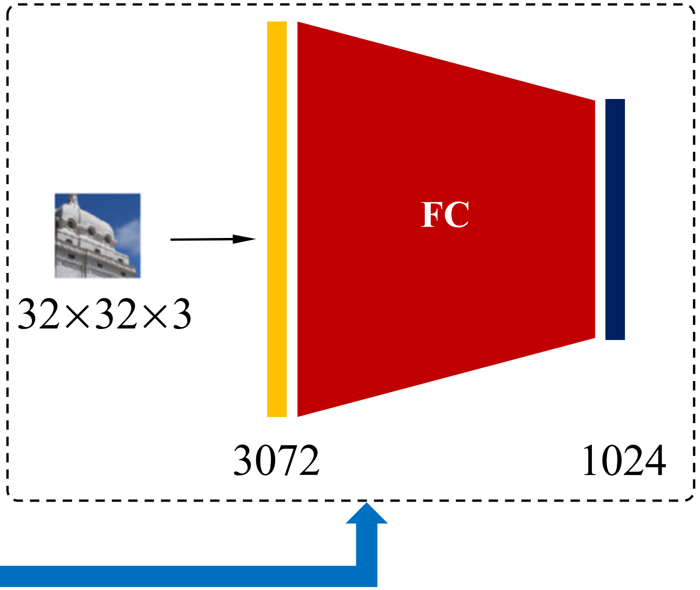


视觉Transformer

Transformer编码器

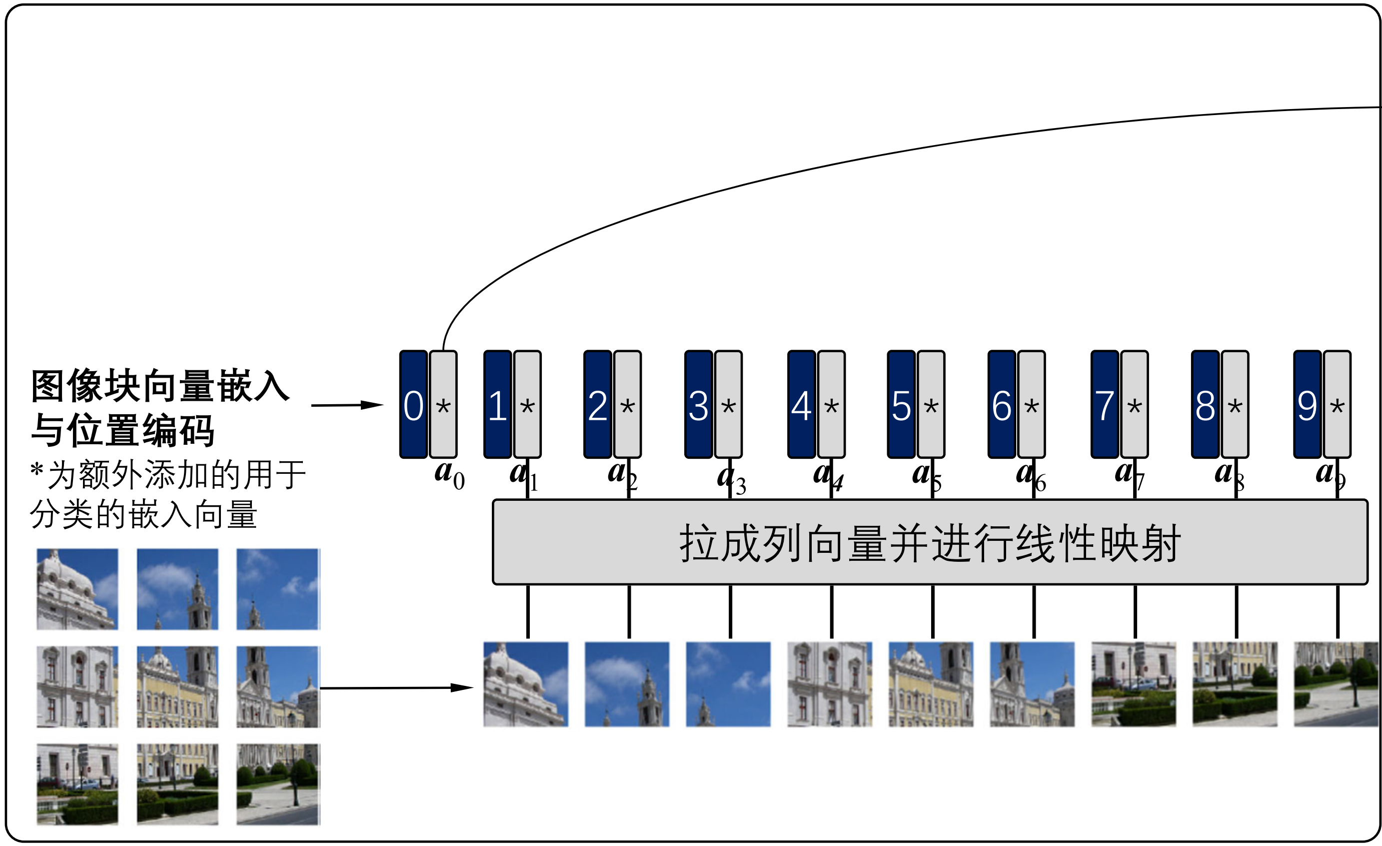
Vision Transformer (ViT)





视觉Transformer

Vision Transformer (ViT)



视觉Transformer

✓ A learnable vector $\boldsymbol{a}_0$ is added here and is expected to hold the global information of the image for classification; $\boldsymbol{a}_0$ can be randomly initialized

With $\boldsymbol{a}_0$, the input vectors can be seen as a matrix,

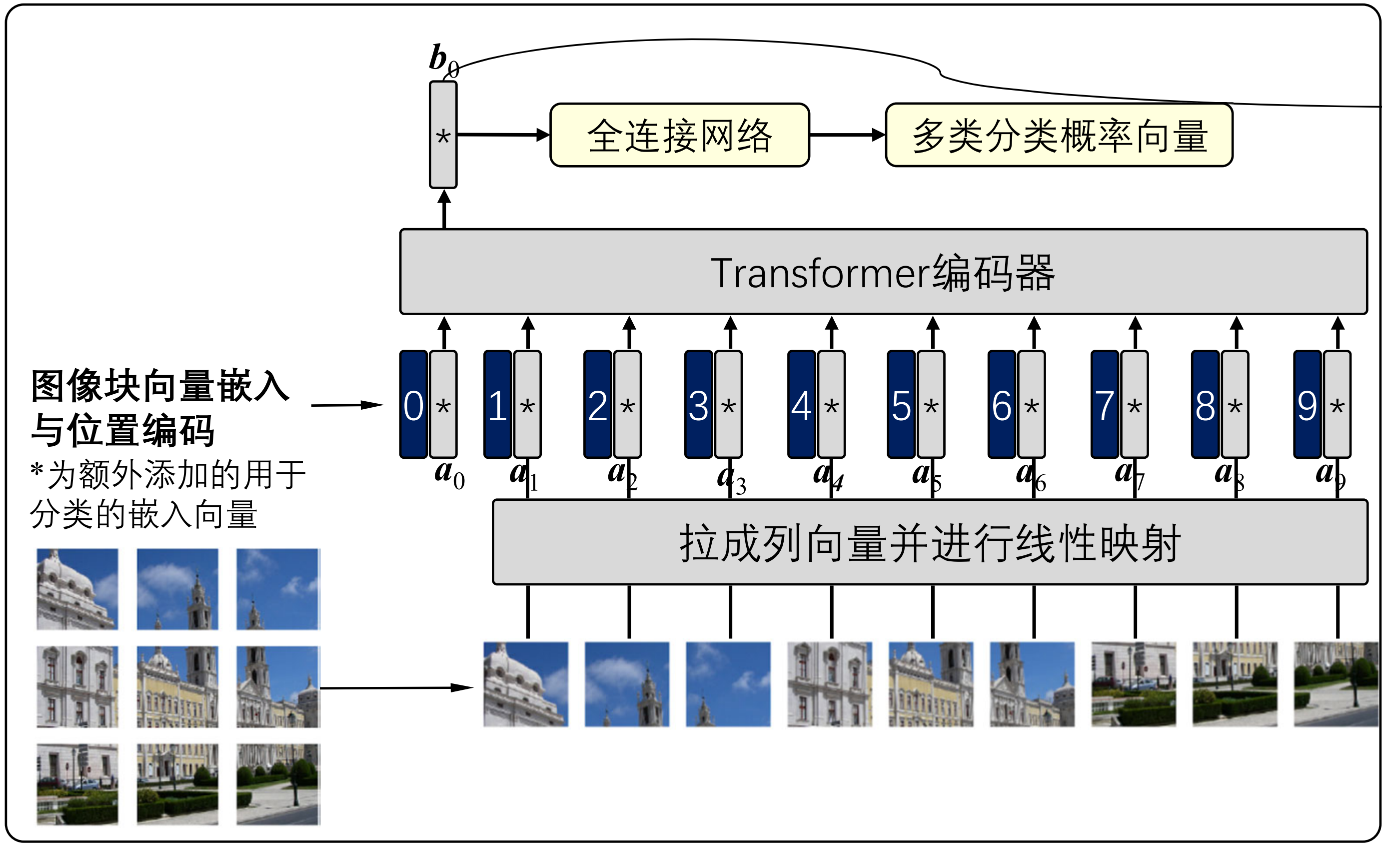
$$\boldsymbol{A} \triangleq [\boldsymbol{a}_0\ \boldsymbol{a}_1\ \boldsymbol{a}_2 \dots \boldsymbol{a}_9] \in \mathbb{R}^{1024\times10}$$

For positional encoding, ViT uses a learnable matrix $\boldsymbol{PE} \in \mathbb{R}^{1024\times10}$

Then, the input vectors with positional encoding form the matrix

$$\boldsymbol{A} + \boldsymbol{PE} \in \mathbb{R}^{1024\times10}$$

Vision Transformer (ViT)



视觉Transformer

✓ b_0 corresponds to a_0; as it is the output of a transformer encoder, it can hold the global information of all the block vectors and can be trained to perform the classification task

Outline

- Transformer in NLP
- Multi-head Attention
- Vision transform (ViT)
- Swin-Transformer
- DETR
- RT-DETR
- Practice of RT-DETR

Swin-Transformer

- ViT is a cool idea and demonstrates that transformer has a good potential in CV; however, it has too many parameters and thus is computationally expensive
- Swin-Transformer[1] proposes several universal and straightforward ideas to improve ViT
  - To reduce computational cost: **self-attention is restricted to a local window**
  - To make the outputs of the self-attention still capture the contextual information: **windows used to compute self-attention are shifted and the self-attention is repeated**; the shifted window partitioning approach introduces connections between neighboring non-overlapping windows in the previous layer
  - To get multi-resolution feature maps (as Yolov3 and Yolov8): **a patch merging mechanism is proposed**; the obtained hierarchical representation is very similar to multi-resolution CNN feature maps

[1] LIU Z, LIN Y, CAO Y, et al. Swin transformer: Hierarchical vision transformer using shifted windows[C]//Proc. IEEE Int'l. Conf. Computer Vision, 2021: 9992-10002. (**Cited by 19016**, July 4, 2024)

Swin-Transformer

Marr Prize

Swin Transformer: Hierarchical Vision Transformer using Shifted Windows

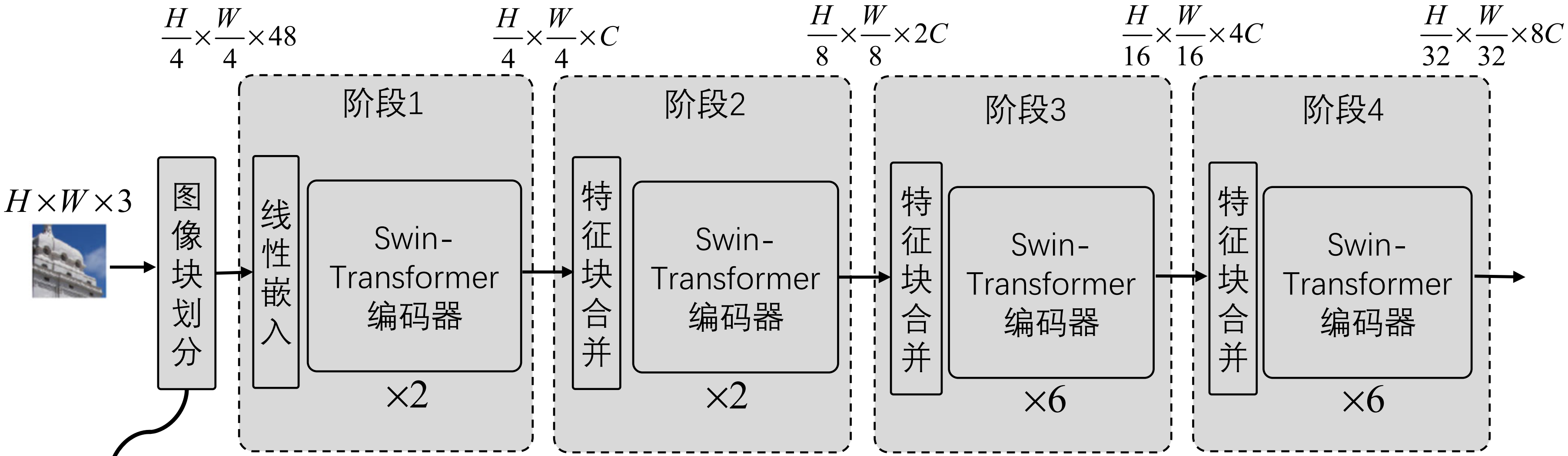
Ze Liu (USTC), Yutong Lin (Xi'an Jiaotong University),
Yue Cao (Microsoft Research), Han Hu (Microsoft Research Asia),
Yixuan Wei (Tsinghua University), Zheng Zhang (MSRA, Huazhong University of Science and Technolog), Stephen Lin (Microsoft Research),
Baining Guo (MSR Asia)

Session 8 (A/B)

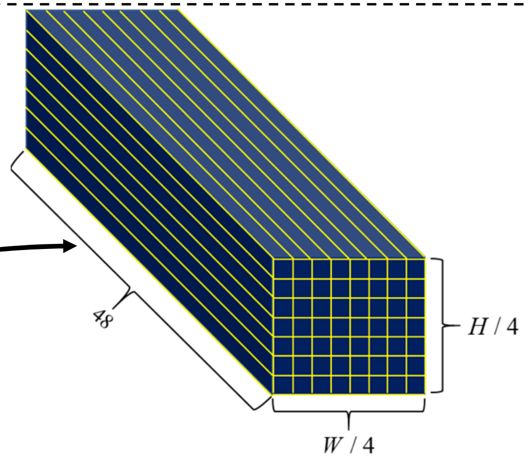
2021 ICCV OCTOBER 11-17 VIRTUAL

Ze Liu is currently a final-year joint Ph.D. candidate at the University of Science and Technology of China (USTC) and Microsoft Research Asia (MSRA)

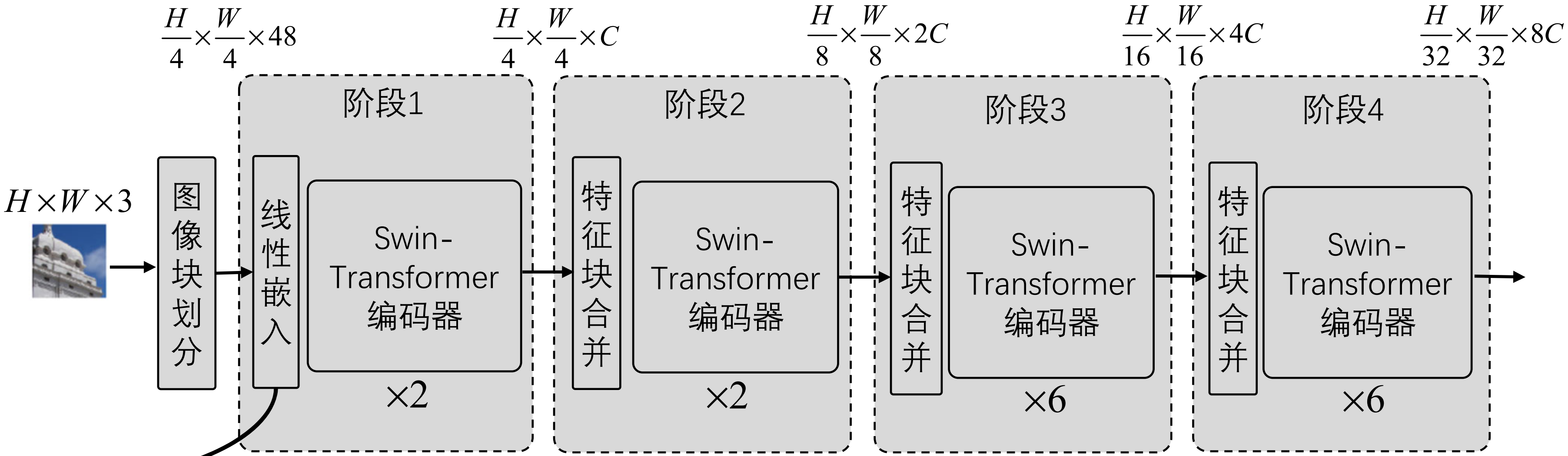
Swin-Transformer



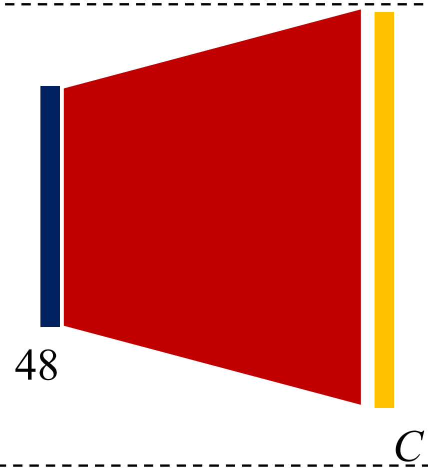
- ✓ Each patch is of the size 4×4×3, and is stacked into a 48-*d* vector
- ✓ After the patch partition, the input to the "stage 1" is a feature map $V_0 \in \mathbb{R}^{\frac{H}{4}\times\frac{W}{4}\times48}$



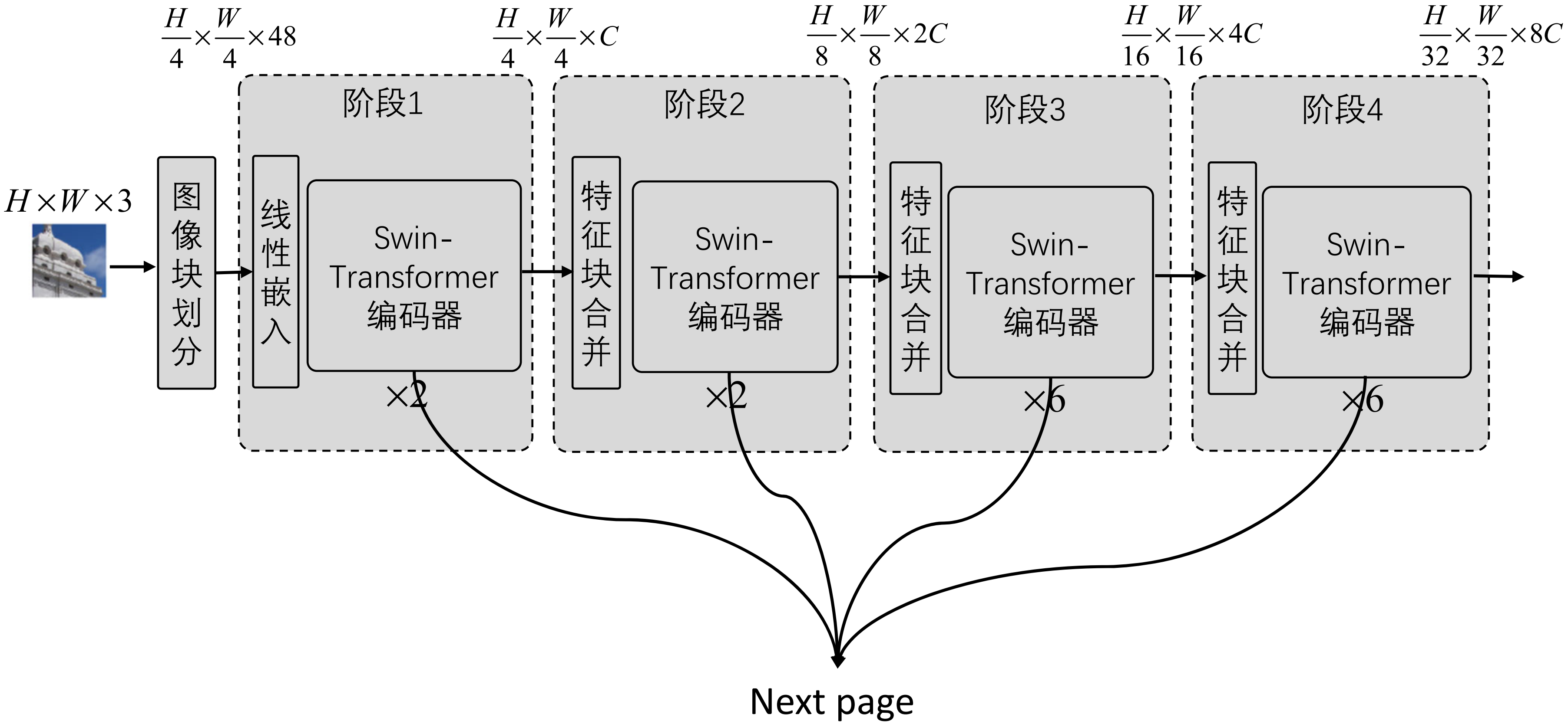
Swin-Transformer



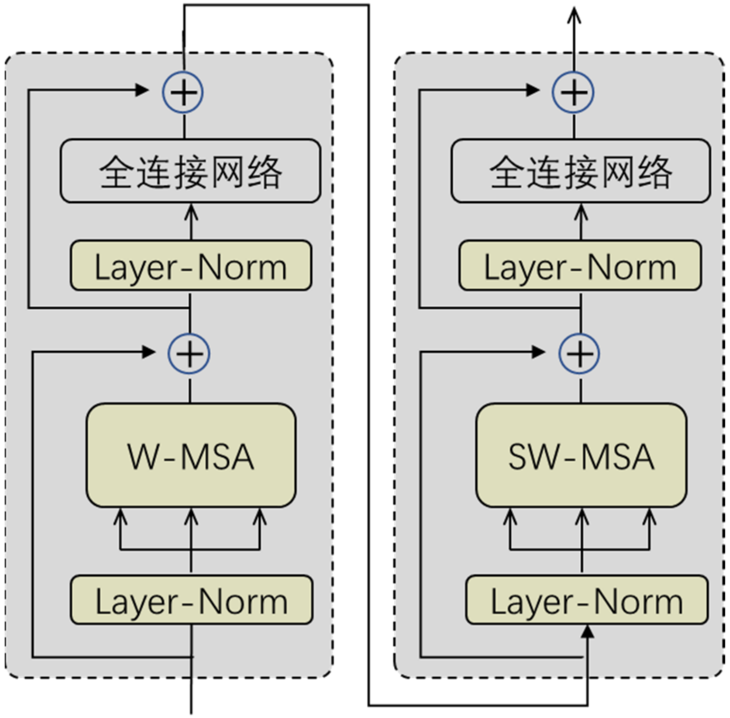
- ✓ Using a fully-connect network to map each 48-d vector in V_0 to a C-d vector
- ✓ After linear embedding, the feature map becomes $V_0^1 \in \mathbb{R}^{\frac{H}{4}\times\frac{W}{4}\times C}$
- ✓ Since the swin-transformer encoder does not change the dimension, the output dimension of stage 1 also is $\frac{H}{4}\times\frac{W}{4}\times C$



Swin-Transformer

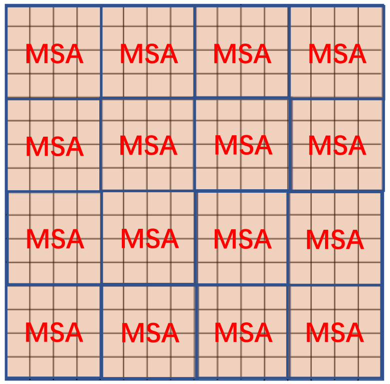


Swin-Transformer

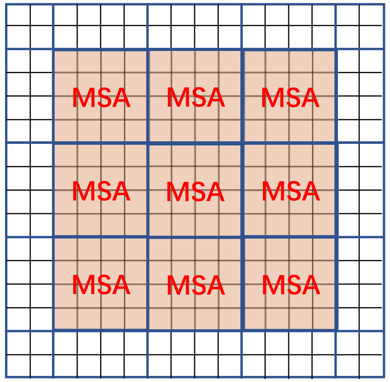


Swin-Transformer Encoders

- ✓ Two swin-transformer encoders are used in pair
- ✓ With respect to structures, they are similar to the transformer encoder of ViT; the only difference is that the swin-transformer encoders compute multi-head self-attention in local windows
- ✓ The windows used in SW-MSA are shifted from windows in W-MSA; the window-shifting mechanism can supplement connections across non-overlapping windows



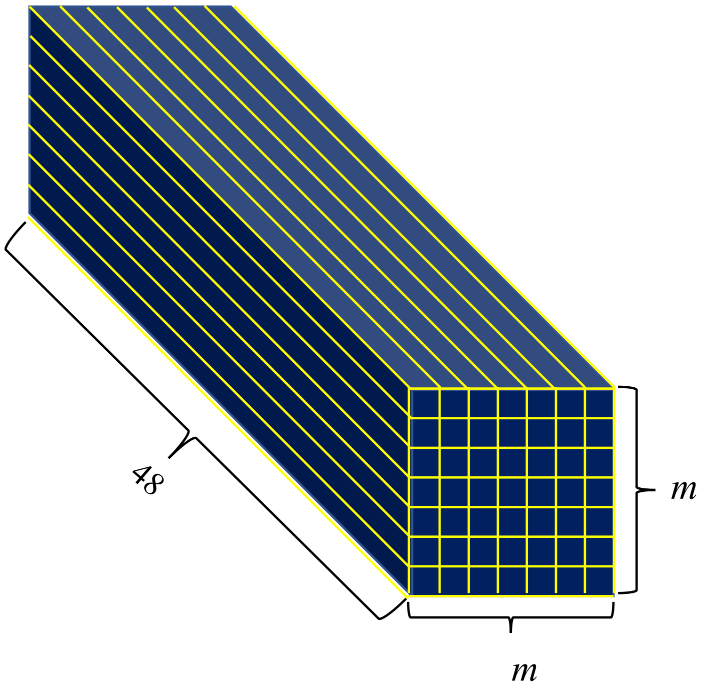
Window-partition in W-MSA



Window-partition in SW-MSA

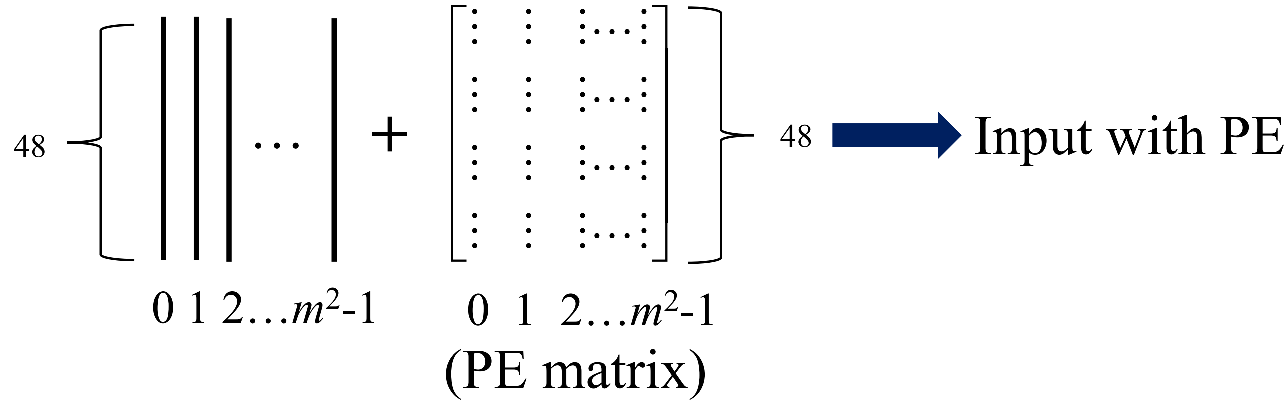
Note: Each point actually is a vector

Swin-Transformer



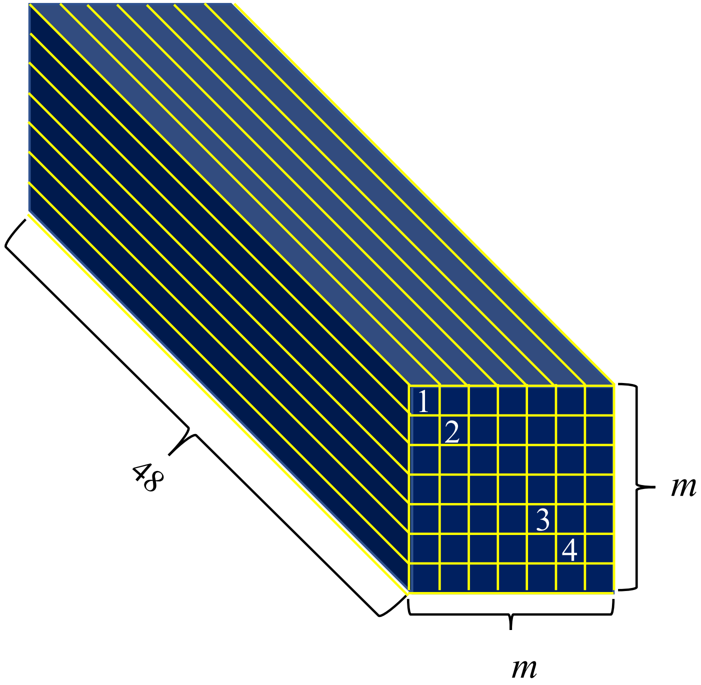
How to perform positional encoding?

Using a common positional encoding strategy,



- ✓ PE matrix can be pre-fixed (such as sinusoid functions) or can be learned as in the case of ViT
- ✓ Such a positional encoding strategy can be considered as a “1-D” strategy since the vectors are arranged in a line
- ✓ **The “2D” positional relationships of the vectors cannot be well embedded in such a way** → **Relative Position Bias**

Swin-Transformer



- Relative position bias can encode the relative position relationship between two vectors in a 2D array

  E.g., the RPB of "2" to "1" is the same as the RPB of "4" to "3"

- Positional encodings are added to the input vectors; differently, RPBs are added to attention scores

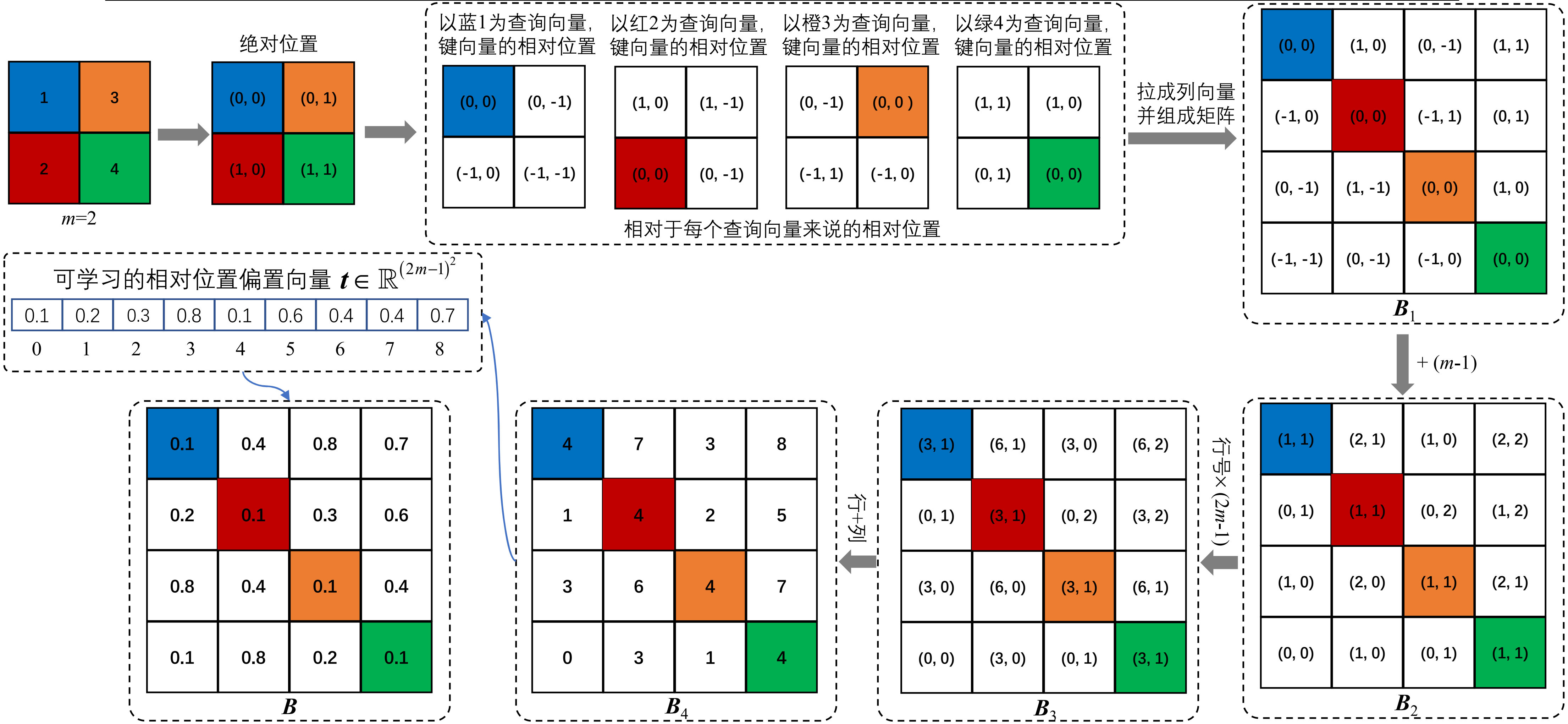
  Embedded queries and keys: $\boldsymbol{Q} \in \mathbb{R}^{d \times m^2}, \quad \boldsymbol{K} \in \mathbb{R}^{d \times m^2}$

  Attention scores,

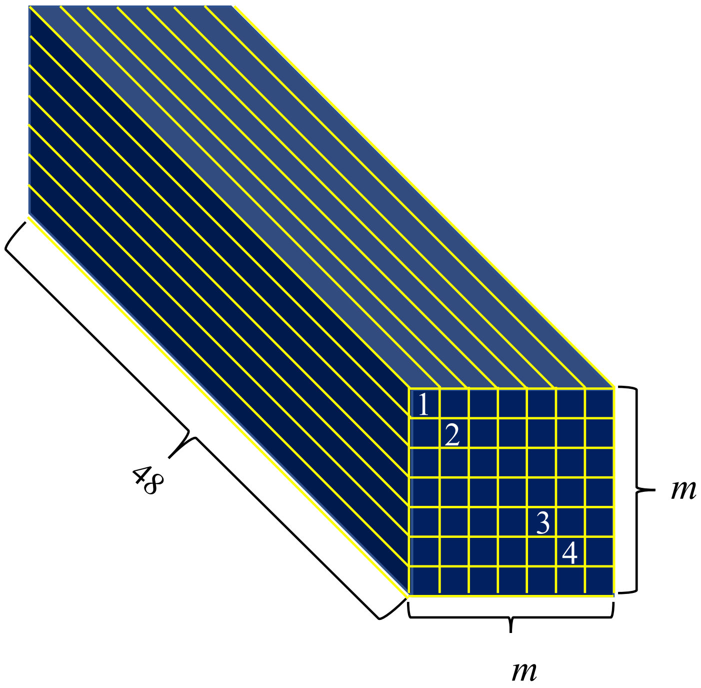
$$\boldsymbol{A}_{m^2 \times m^2} = \boldsymbol{K}^T \boldsymbol{Q} = \begin{bmatrix} \alpha_{1,1} & \alpha_{2,1} & \cdots & \alpha_{m^2,1} \\ \alpha_{1,2} & \alpha_{2,2} & \cdots & \alpha_{m^2,2} \\ \vdots & \vdots & & \vdots \\ \alpha_{1,m^2} & \alpha_{2,m^2} & \cdots & \alpha_{m^2,m^2} \end{bmatrix}$$

It implies that the RPB matrix $\boldsymbol{B}$ should be of the dimension $m^2 \times m^2$, and $\boldsymbol{B}(i, j)$ reflects the relative position of the query $\boldsymbol{q}_i$ and the key $\boldsymbol{k}_j$

Swin-Transformer



Swin-Transformer



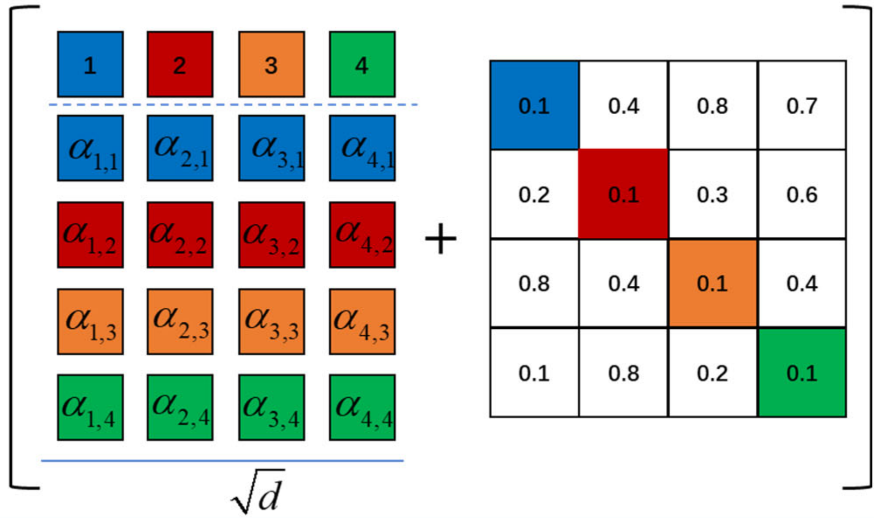
- Relative position bias can encode the relative position relationship between two vectors in a 2D array

  E.g., the RPB of "2" to "1" is the same as the RPB of "4" to "3"

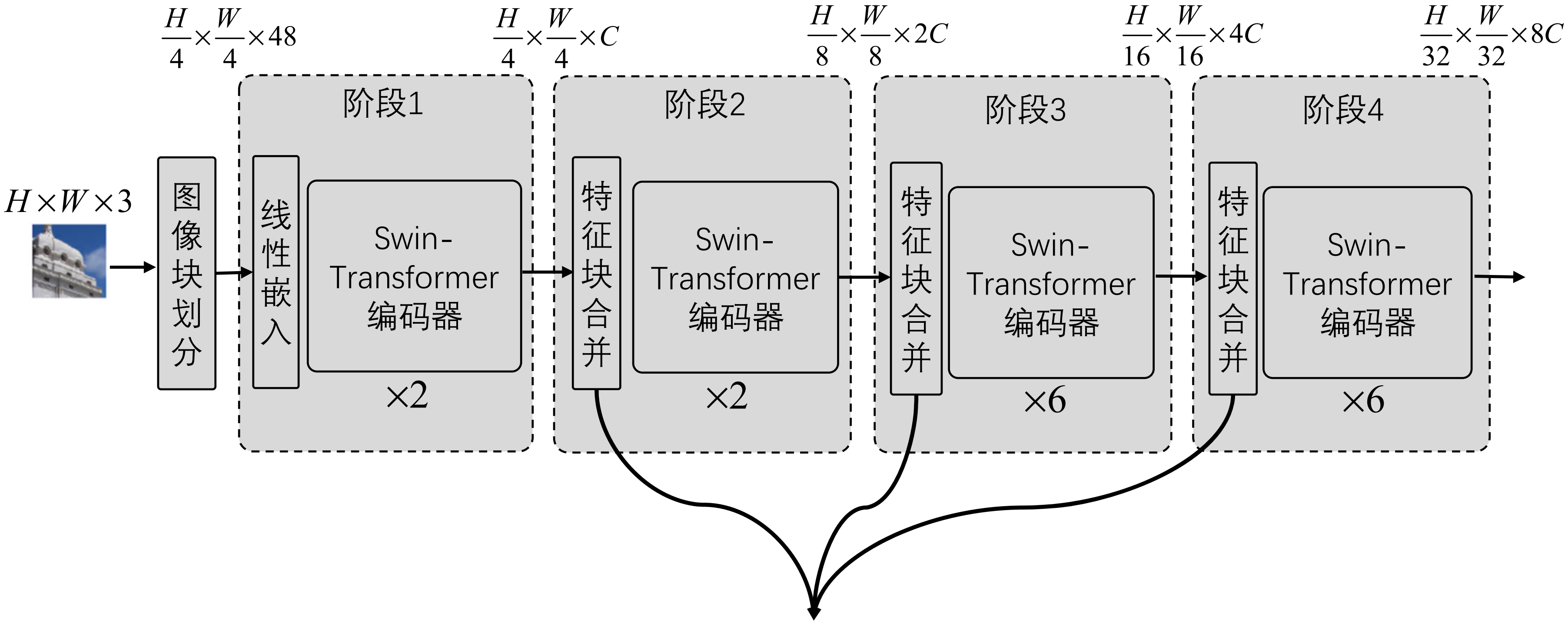
- Positional encodings are added to the input vectors; differently, RPBs are added to attention scores

$$A'_{m^2 \times m^2} = softmax\left(\frac{K^T Q}{\sqrt{d}} + B\right)$$

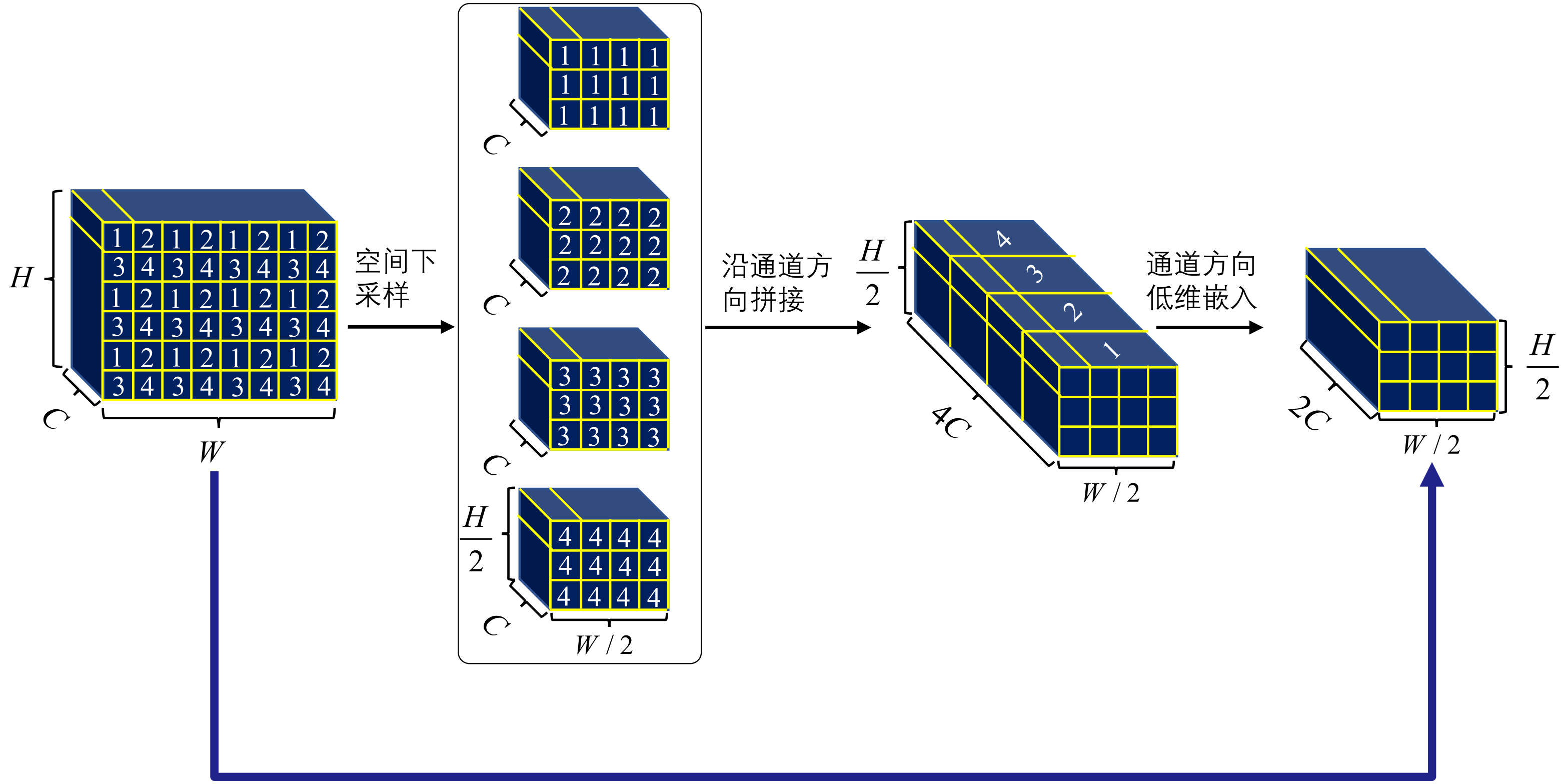
$= softmax$



Swin-Transformer



Swin-Transformer



- ✓ Half the spatial resolution
- ✓ Double the channels

Swin-Transformer

- Play with Swin-Transformer
  - Swin-Transformer is a backbone network architecture
  - It outputs 4 feature maps with four different spatial resolutions; these four feature maps can be used similar as CNN feature maps; thus with different task heads, it can be used for object detection, object classification, semantic segmentation, etc.
  - The authors provide pre-trained Swin-Transformer models at four different scales

	downsp. rate (output size)	Swin-T	Swin-S	Swin-B	Swin-L
stage 1	4× (56×56)	concat 4×4, 96-d, LN	concat 4×4, 96-d, LN	concat 4×4, 128-d, LN	concat 4×4, 192-d, LN
		[win. sz. 7×7, dim 96, head 3] × 2	[win. sz. 7×7, dim 96, head 3] × 2	[win. sz. 7×7, dim 128, head 4] × 2	[win. sz. 7×7, dim 192, head 6] × 2
stage 2	8× (28×28)	concat 2×2, 192-d , LN	concat 2×2, 192-d , LN	concat 2×2, 256-d , LN	concat 2×2, 384-d , LN
		[win. sz. 7×7, dim 192, head 6] × 2	[win. sz. 7×7, dim 192, head 6] × 2	[win. sz. 7×7, dim 256, head 8] × 2	[win. sz. 7×7, dim 384, head 12] × 2
stage 3	16× (14×14)	concat 2×2, 384-d , LN	concat 2×2, 384-d , LN	concat 2×2, 512-d , LN	concat 2×2, 768-d , LN
		[win. sz. 7×7, dim 384, head 12] × 6	[win. sz. 7×7, dim 384, head 12] × 18	[win. sz. 7×7, dim 512, head 16] × 18	[win. sz. 7×7, dim 768, head 24] × 18
stage 4	32× (7×7)	concat 2×2, 768-d , LN	concat 2×2, 768-d , LN	concat 2×2, 1024-d , LN	concat 2×2, 1536-d , LN
		[win. sz. 7×7, dim 768, head 24] × 2	[win. sz. 7×7, dim 768, head 24] × 2	[win. sz. 7×7, dim 1024, head 32] × 2	[win. sz. 7×7, dim 1536, head 48] × 2

Outline

- Transformer in NLP
- Multi-head Attention
- Vision transform (ViT)
- Swin-Transformer
- DETR
- RT-DETR
- Practice of RT-DETR

DETR


Q+A