NEURAL NETWORKS AND DEEP LEARNING

Transformer

http://speech.ee.ntu.edu.tw/~tlkagk/courses_ML19.html

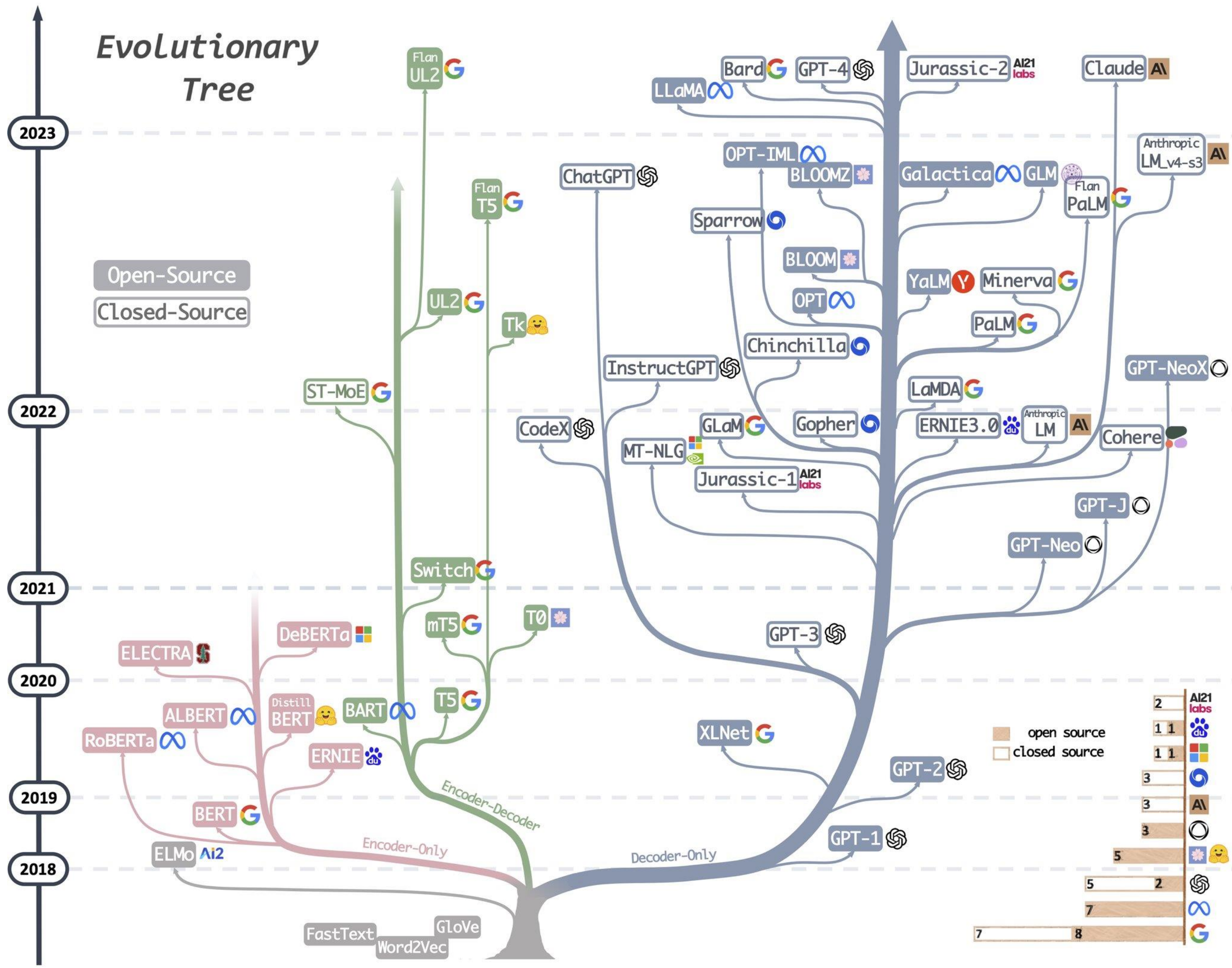
李宏毅Transformer教程：
http://speech.ee.ntu.edu.tw/~tlkagk/courses_ML19.html

Transformer Blog:
https://slds-lmu.github.io/seminar_nlp_ss20/attention-and-self-attention-for-nlp.html

https://lilianweng.github.io/posts/2018-06-24-attention/

http://jalammar.github.io/illustrated-transformer/

LLM in NLP



Evolutionary Tree
2023
2022
2021
2020
2019
2018
Open-Source
Closed-Source
FastText
Word2Vec
GloVe
ELMo
Ai2
Encoder-Only
Encoder-Decoder
Decoder-Only
BERT
RoBERTa
ALBERT
ELECTRA
DeBERTa
Distill BERT
ERNIE
BART
T5
mT5
T0
Switch
ST-MoE
UL2
Tk
Flan T5
Flan UL2
GPT-1
GPT-2
XLNet
GPT-3
GPT-Neo
GPT-J
Cohere
Anthropic LM
ERNIE3.0
Jurassic-1
MT-NLG
GLaM
Gopher
LaMDA
CodeX
InstructGPT
Chinchilla
GPT-NeoX
PaLM
OPT
YaLM
Minerva
BLOOM
Sparrow
BLOOMZ
OPT-IML
Galactica
GLM
Flan PaLM
Anthropic LM_v4-s3
ChatGPT
LLaMA
Bard
GPT-4
Jurassic-2
Claude
AI21 labs
open source
closed source
2
1 1
1 1
3
3
3
5
5 2
7
7 8

LLM in Time-series

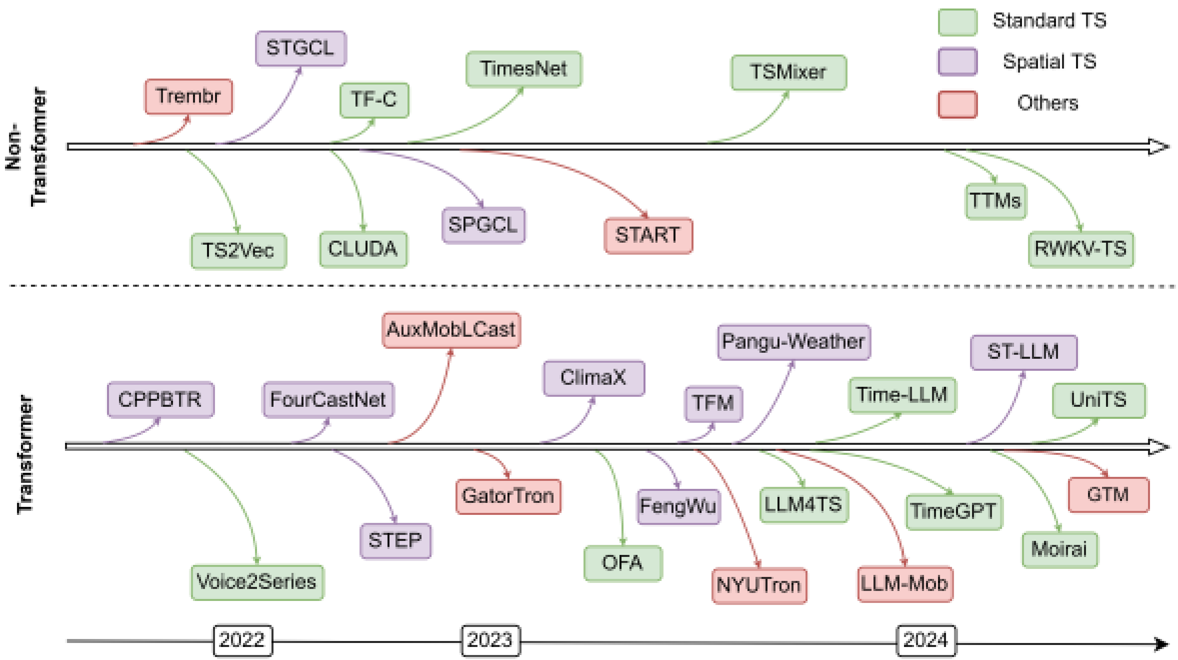


Figure 1: Roadmaps of representative TSFMs.

https://arxiv.org/pdf/2403.14735

Transformer in Time-series

- Vanilla Transformer, NeurIPS 2017
- Informer, AAAI 2021
- Autoformer, NeurIPS 2021
- FEDformer, ICML 2022

… (Transformer variants)

- Dlinear, AAAI 2023

… (non-Transformer model)

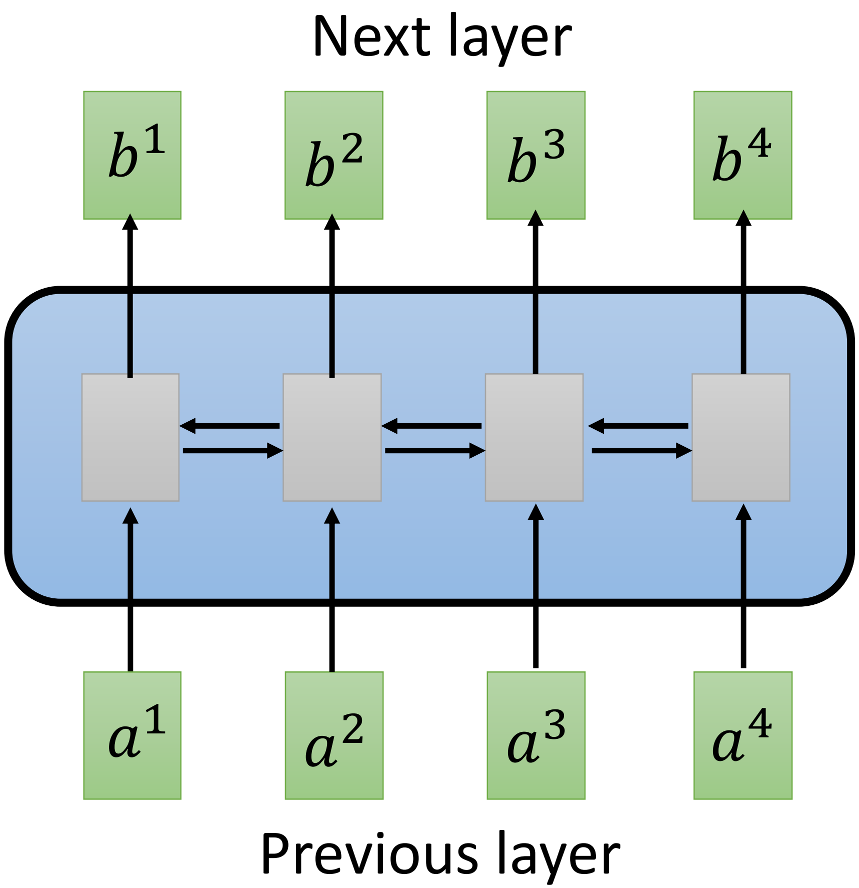
- PatchTST, ICLR 2023

… (Patching based Transformer)

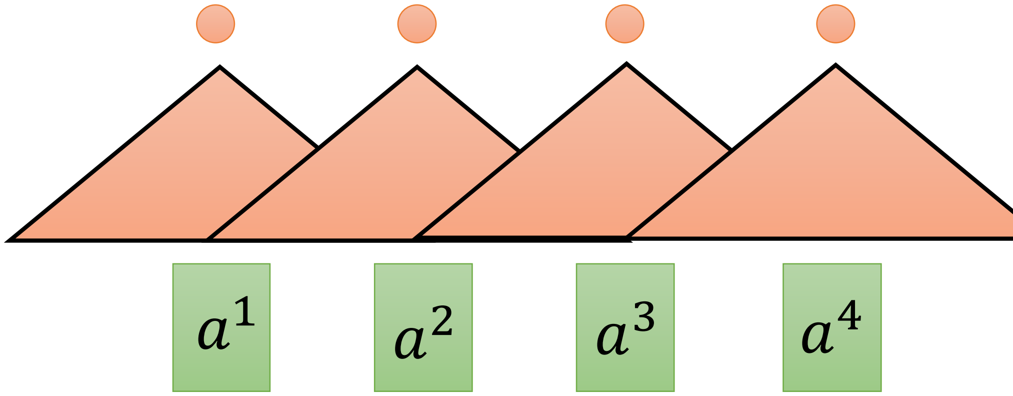
- Time-series Foundation model

… (TSFM)

Sequence

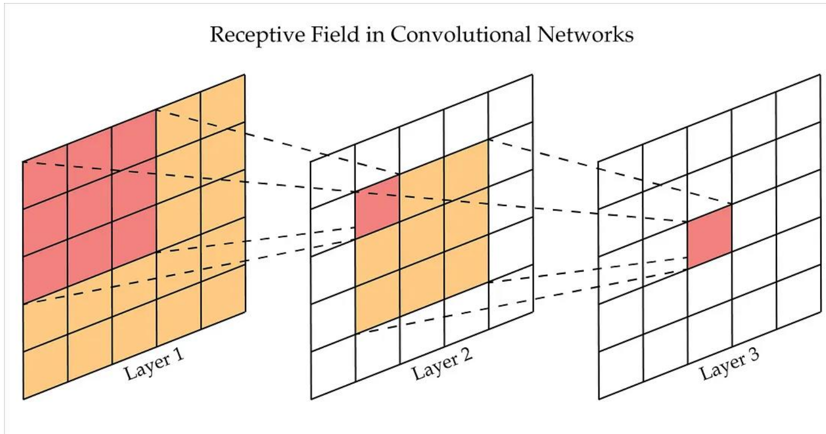


Hard to parallel !

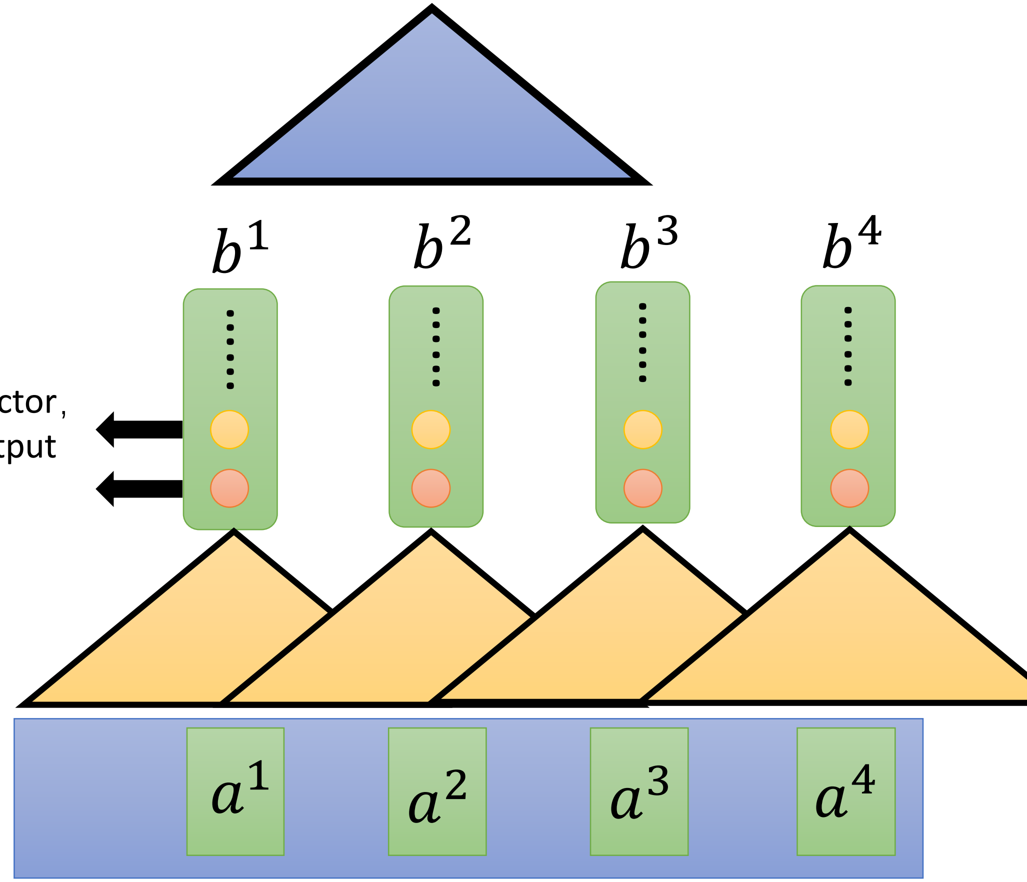


Using CNN to replace RNN

CNN can parallel !



Filters in higher layer can consider longer sequence



不同卷积核得到的hidden vector，有多少个卷积核对应output hidden vector有多少channel

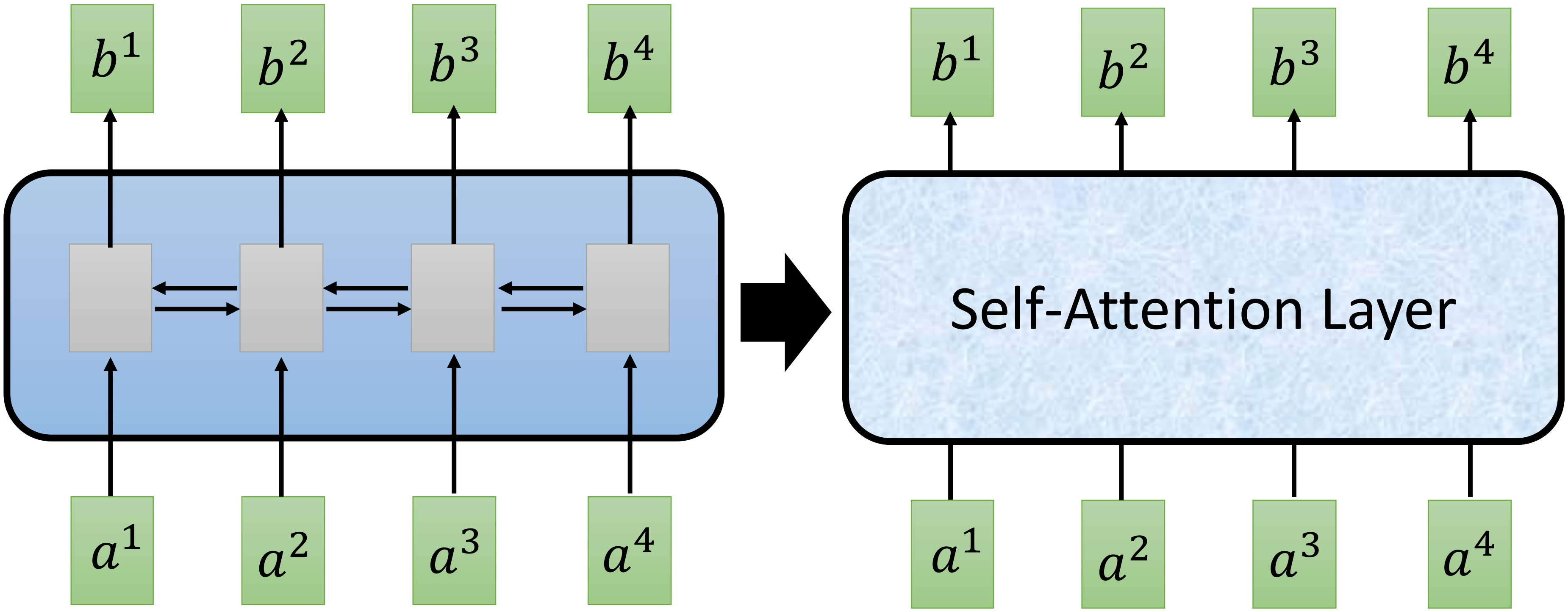
虽然higher layer filters能够捕捉longer sequence，但lower layer filters始终只能捕捉shorter（局部）sequence

Using CNN to replace RNN
(CNN can parallel)

Self-Attention

b^i is obtained based on the whole input sequence. 与双向RNN相同

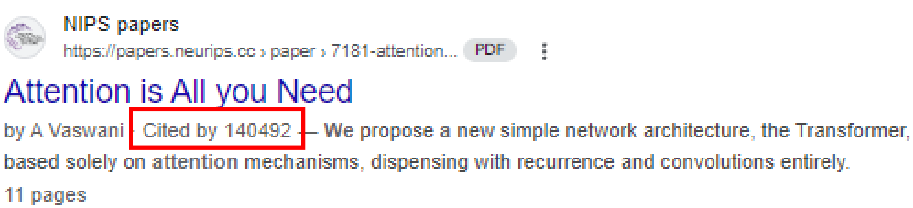
b^1, b^2, b^3, b^4 can be parallelly computed. 与RNN不同

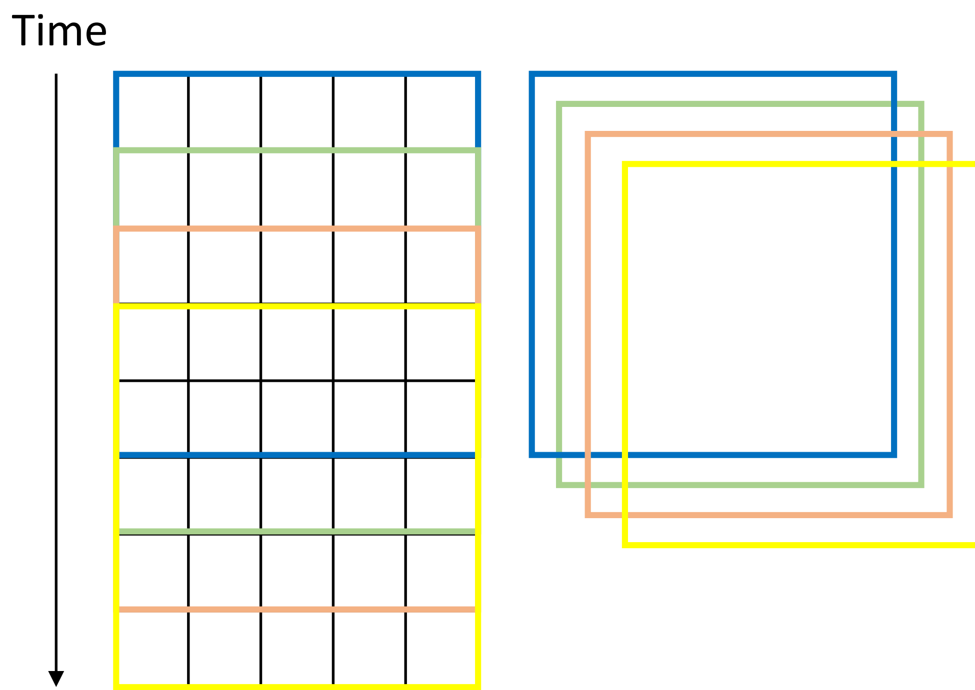


Anyway, you can try to replace any thing that has been done by RNN with self-attention.

Self-attention

https://arxiv.org/abs/1706.03762


NIPS papers
https://papers.neurips.cc › paper › 7181-attention... PDF
Attention is All you Need
by A Vaswani · Cited by 140492 — We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely.
11 pages



4x5x5

- 4: batch size
- 5: sequence length
- 5: feature dimension

对于time-series数据而言， attention一般作用在sequence维度（第二个维度）上

Self-attention

https://arxiv.org/abs/1706.03762

q: query (to match others)

$$q^i = W^q a^i$$

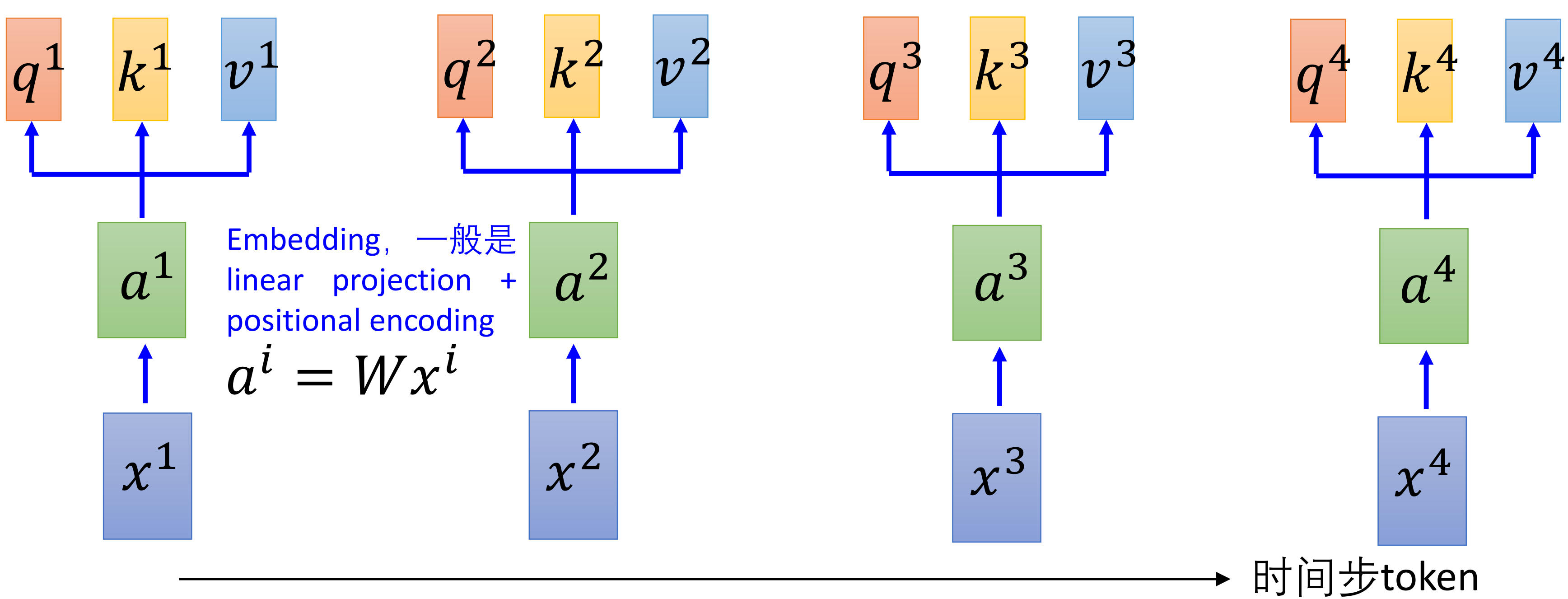
不同时间步embedding对应的W^q, W^k, W^v是同一个，换句话说qkv的projection parameter matrix在不同时间步上是shared

k: key (to be matched)

$$k^i = W^k a^i$$

v: information to be extracted

$$v^i = W^v a^i$$



q: query (to match others)

$$q^i = W^q a^i$$

k: key (to be matched)

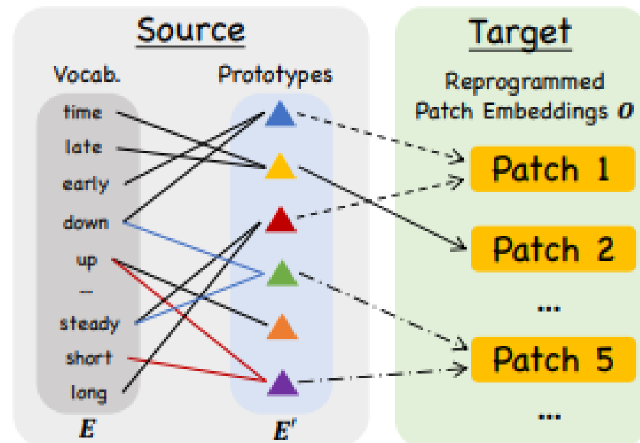
$$k^i = W^k a^i$$

v: information to be extracted

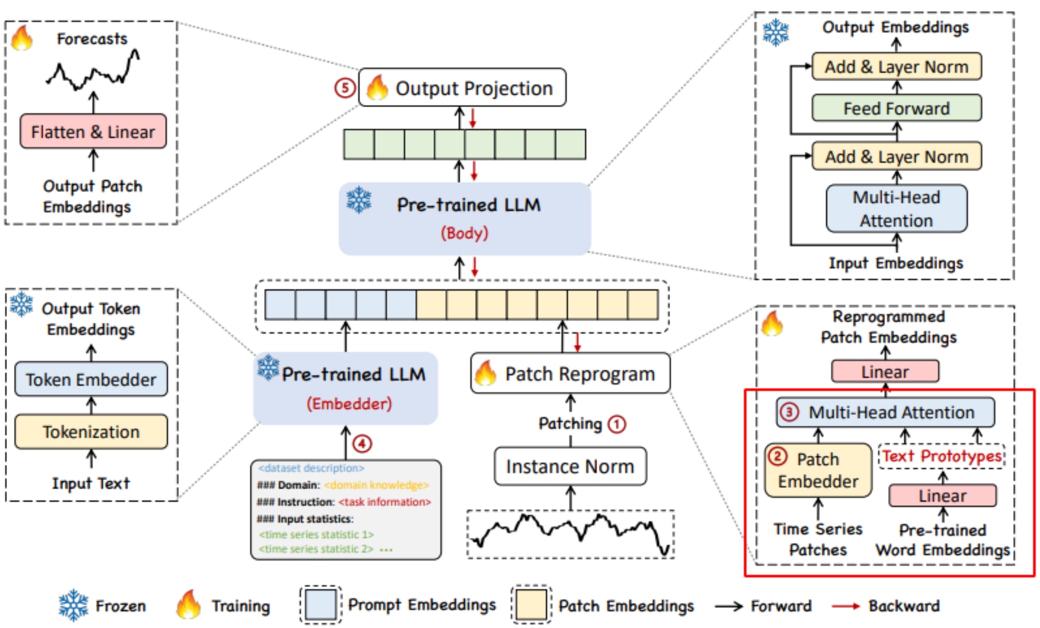
$$v^i = W^v a^i$$

Time-LLM: Time Series Forecasting by Reprogramming Large Language Models, ICLR 2024

Paper: https://openreview.net/pdf?id=Unb5CVPtae
Blog: https://zhuanlan.zhihu.com/p/676256783



(a) Patch Reprogramming



Pretrained LLM (embedder)是在**文本语料上预训练的**，具有文本modality上的知识，但现在我们需要的是time-series modality上做forecasting任务，因此需要**以time-series data (embedding) 作为query去match text embedding** (text embedding是被matched)，然后再**从text embedding里面extract信息**，相当于将text中和当前time-series embedding相似度高的部分（例如up, short)提取出来

q: query (to match others)

$$q^i = W^q a^i$$

k: key (to be matched)

$$k^i = W^k a^i$$

v: information to be extracted

$$v^i = W^v a^i$$

刚才介绍的是cross-attention，涉及两个data modality之间的信息交互，self-attention就是自己和自己算attention，qkv来自于同一个data modality

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^\top}{\sqrt{d_k}})V$$

Self-attention

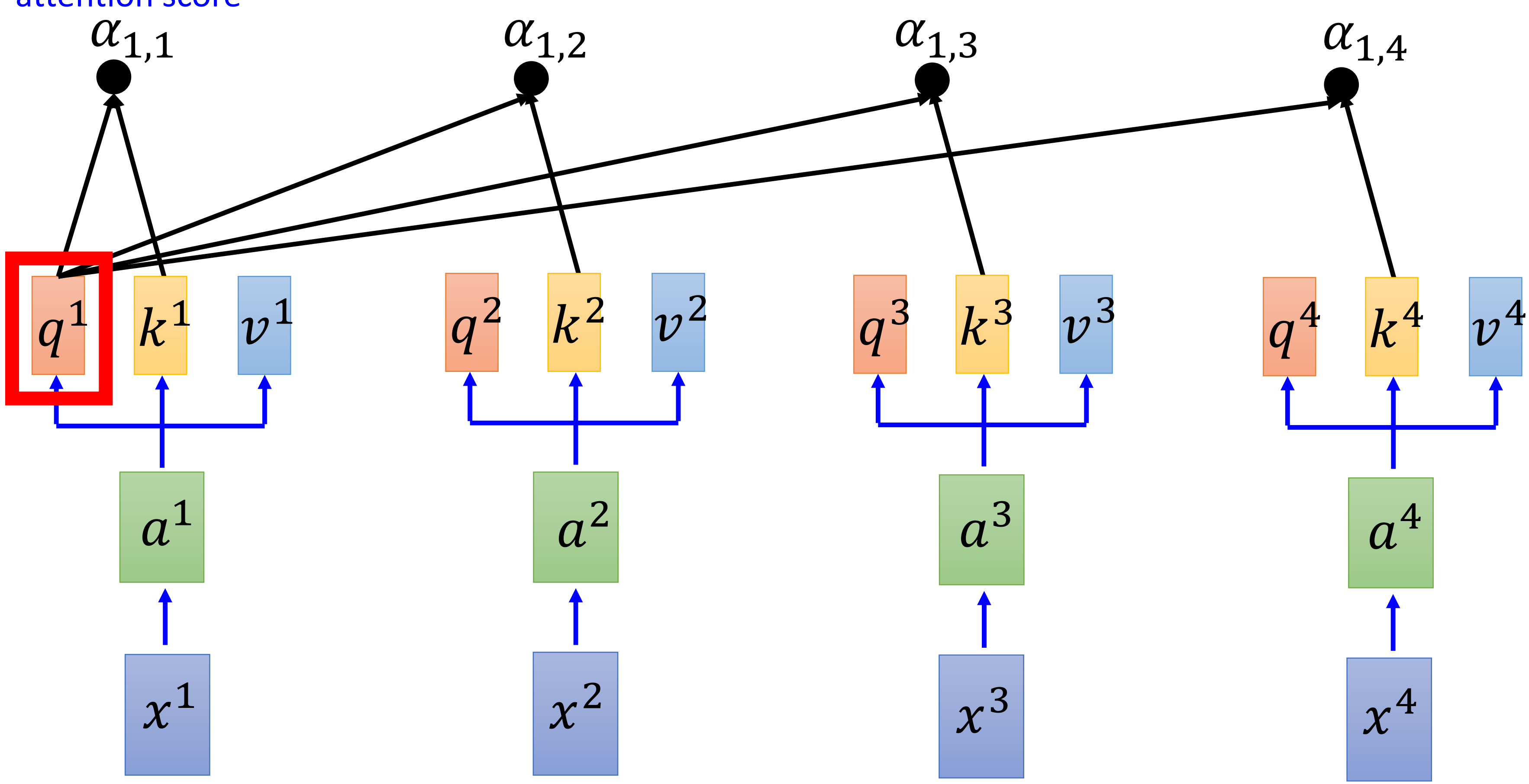
qk之间做dot product（element-wise相乘相加），会随着元素个数，即维度d的增大而使得variance增大，因此用根号d做scaled

拿每個 query q 去對每個 key k 做 attention

d is the dim of q and k

Scaled Dot-Product Attention: $\alpha_{1,i} = \underbrace{q^1 \cdot k^i}_{\text{dot product}} / \sqrt{d}$

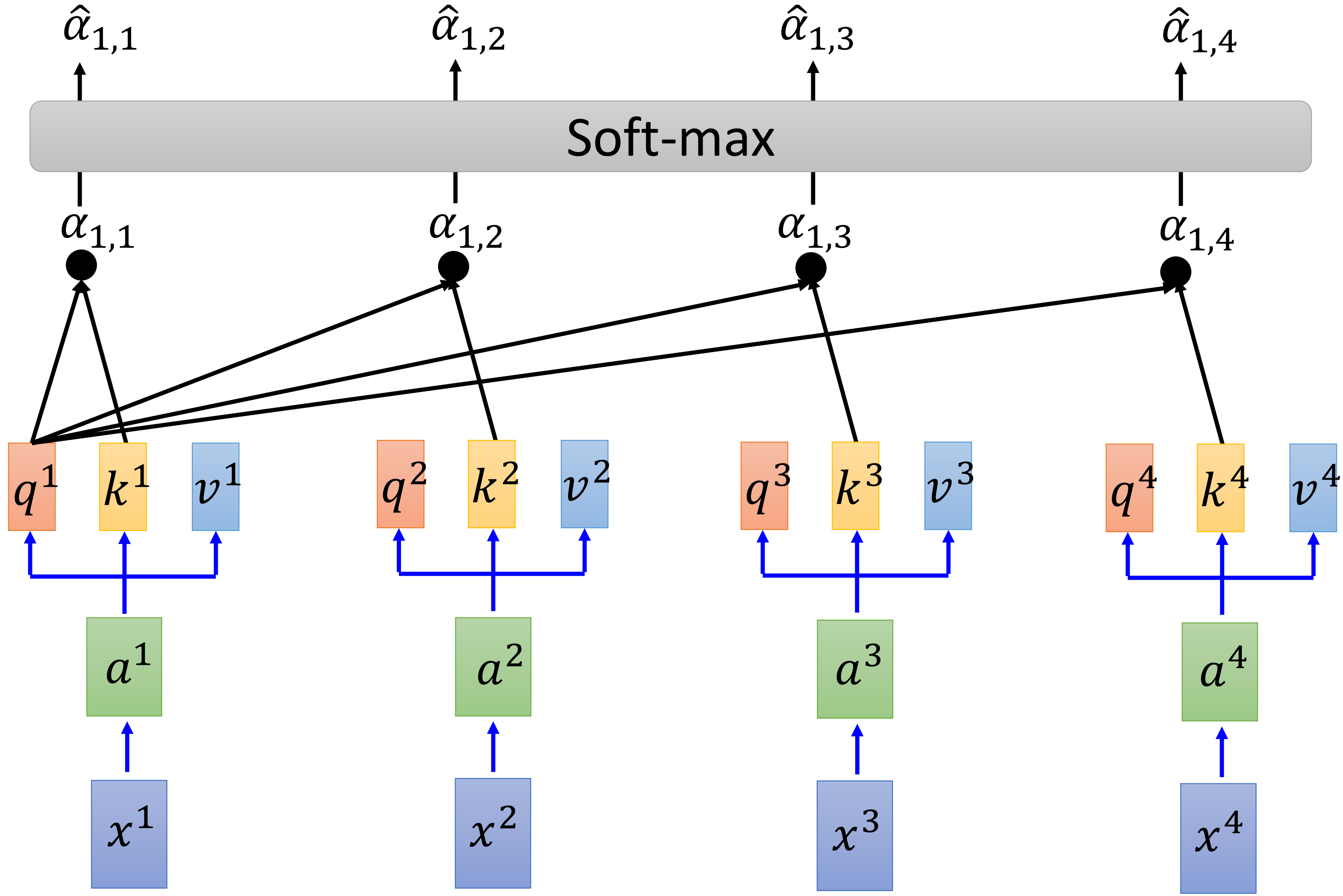
q^1和k^1之间的 attention score



Softmax相当于把attention score归一化

Self-attention

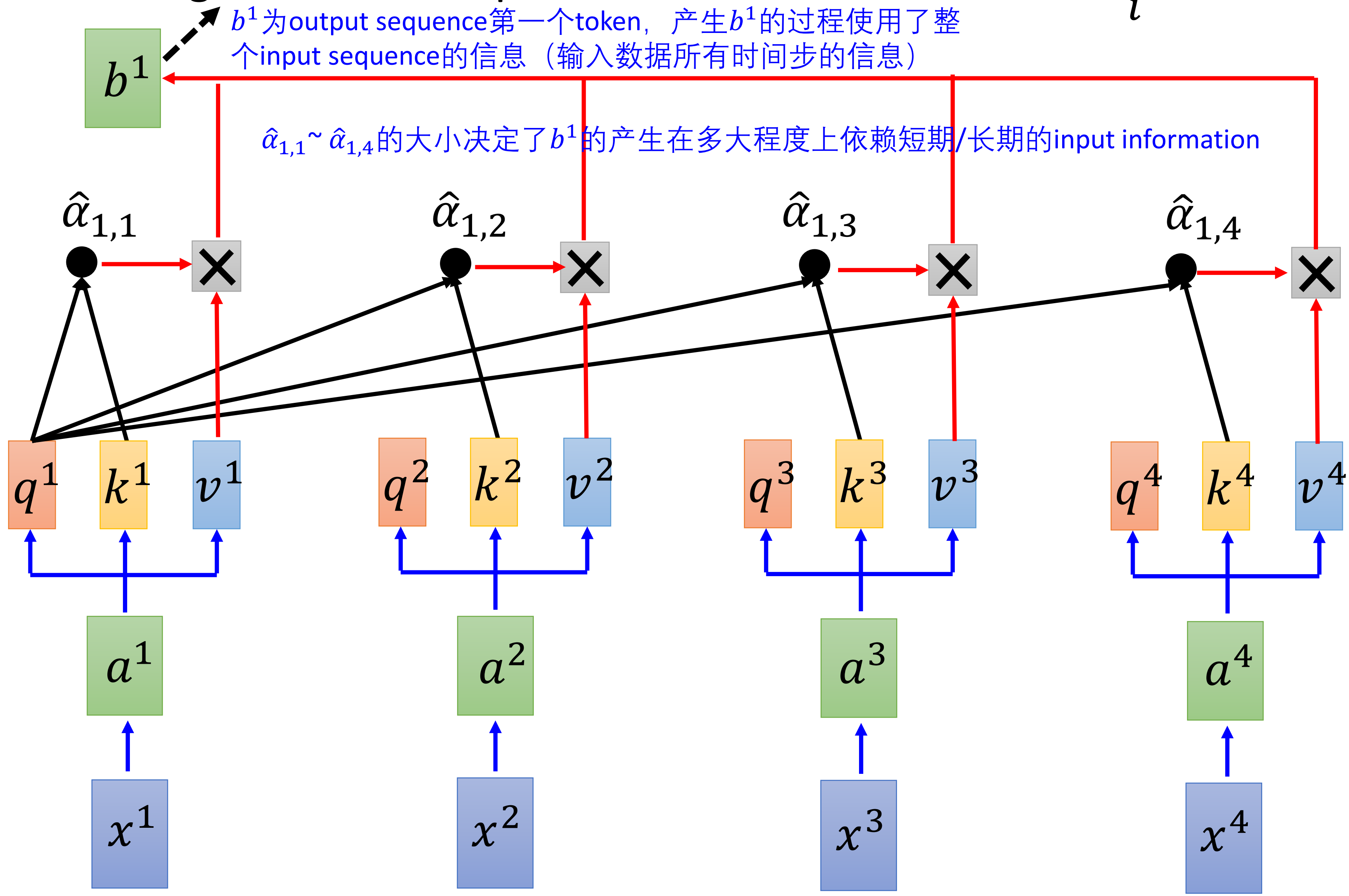
$$\hat{\alpha}_{1,i} = exp(\alpha_{1,i})/\sum_j exp(\alpha_{1,j})$$



Self-attention

Considering the whole sequence

$$b^1 = \sum_i \hat{\alpha}_{1,i} v^i$$

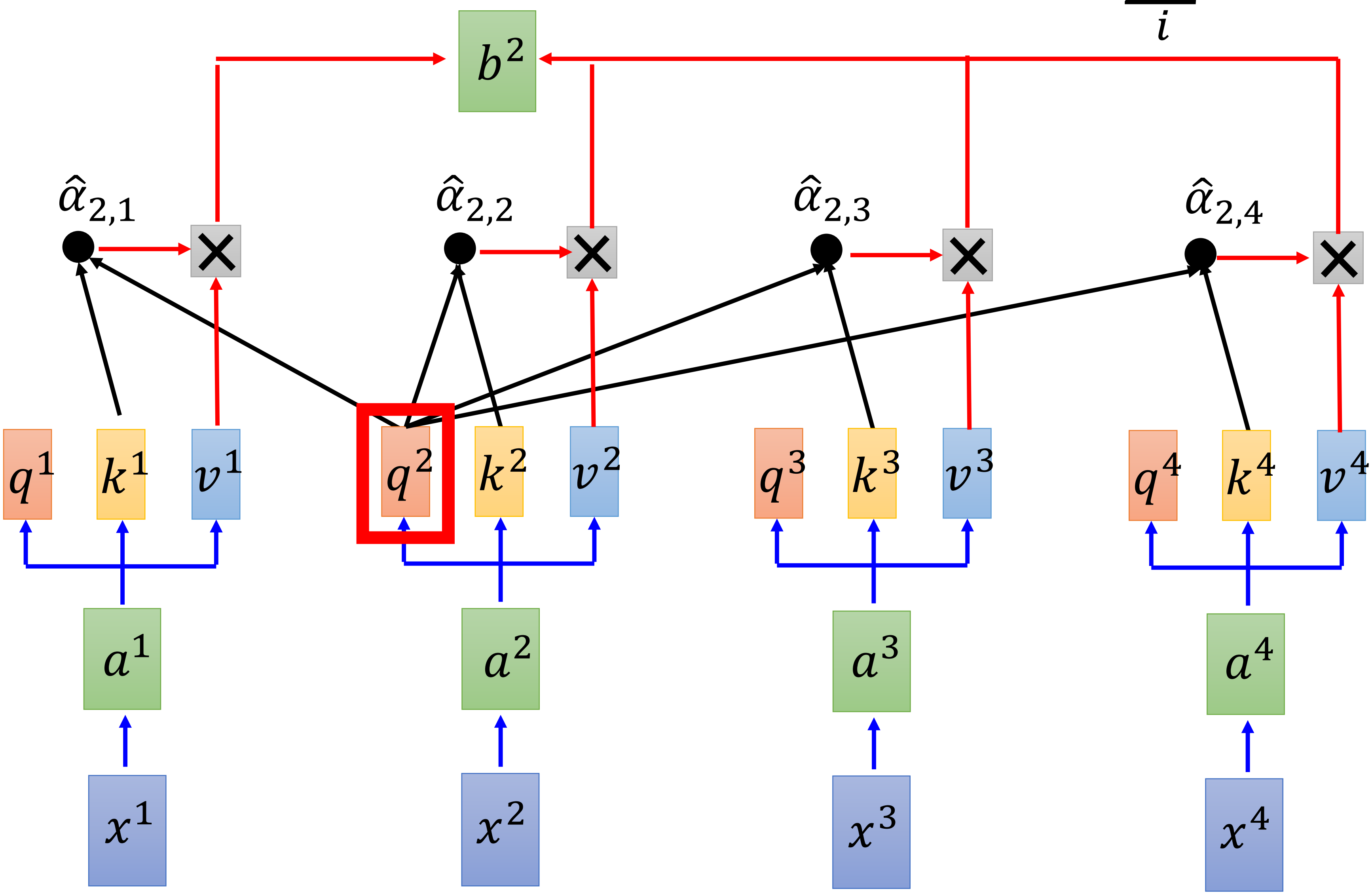


Self-attention

b^1, b^2 ...的运算是parallel的， b^2并不需要像RNN那样等b^1计算完成

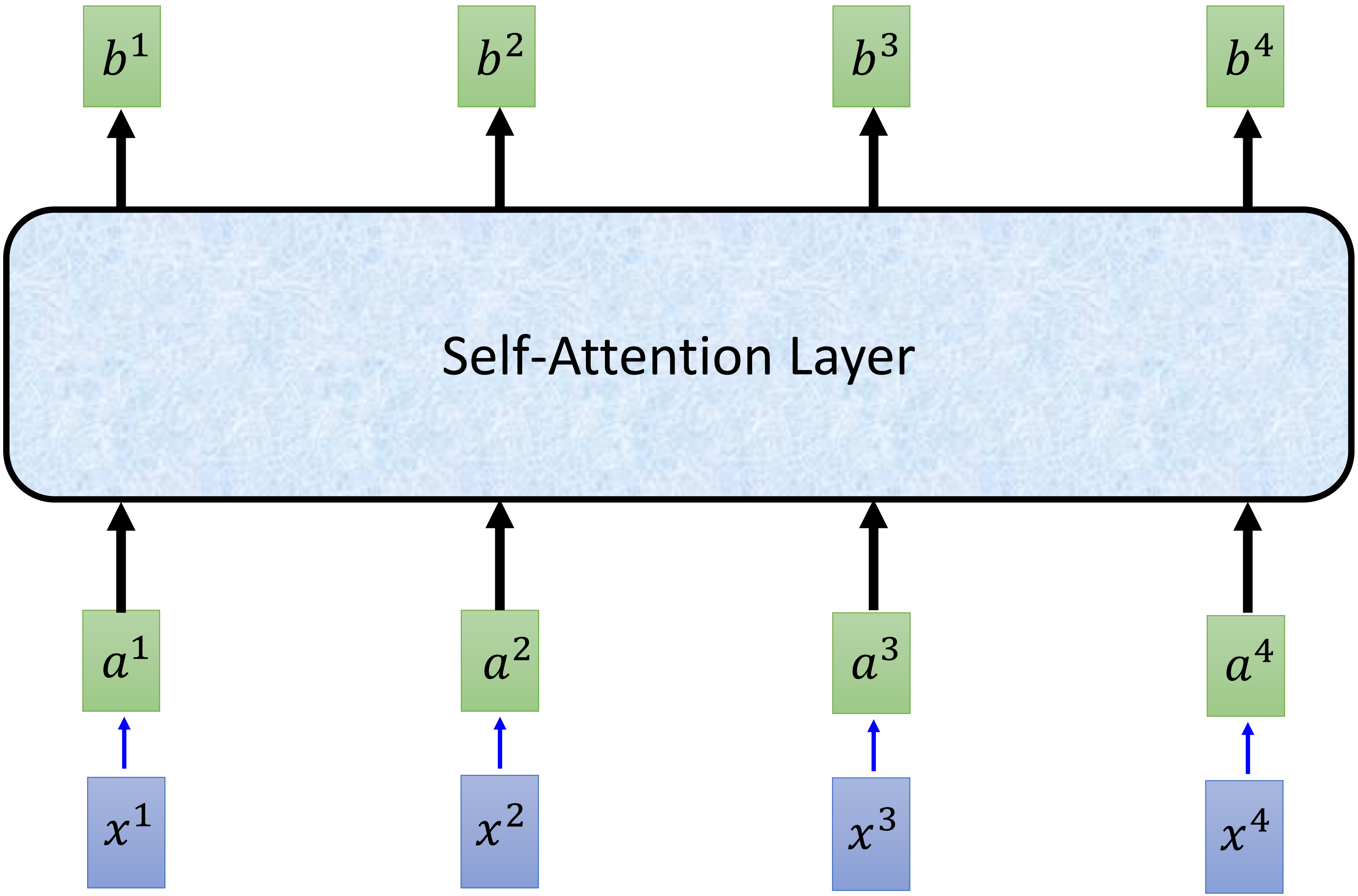
拿每個 query q 去對每個 key k 做 attention

$$b^2 = \sum_i \hat{\alpha}_{2,i} v^i$$

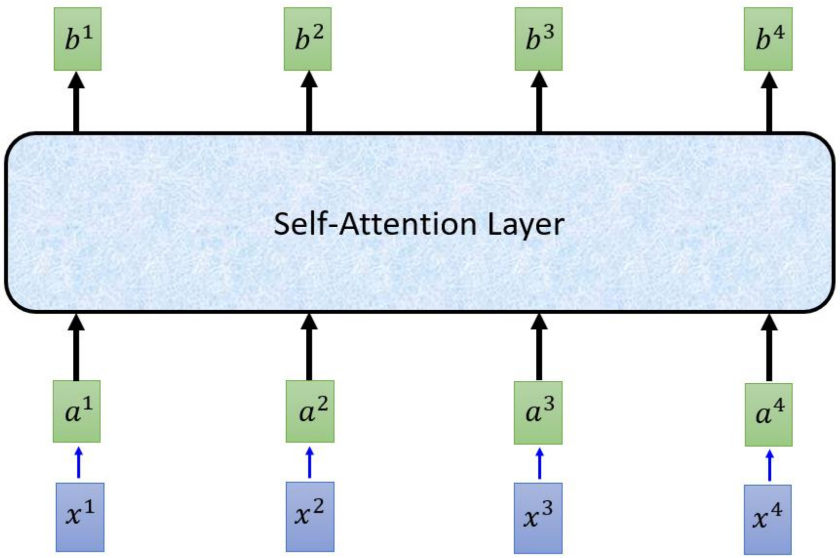


Self-attention

b^1, b^2, b^3, b^4 can be parallelly computed.



Self-attention



Attention 本质上在计算token之间的相似度（与cosine similarity的计算类似），然后用归一化的相似度去加权得到output token

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

$$\text{cosine similarity} = S_C(A,B) := \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2}\cdot\sqrt{\sum\limits_{i=1}^{n} B_i^2}},$$

Self-attention

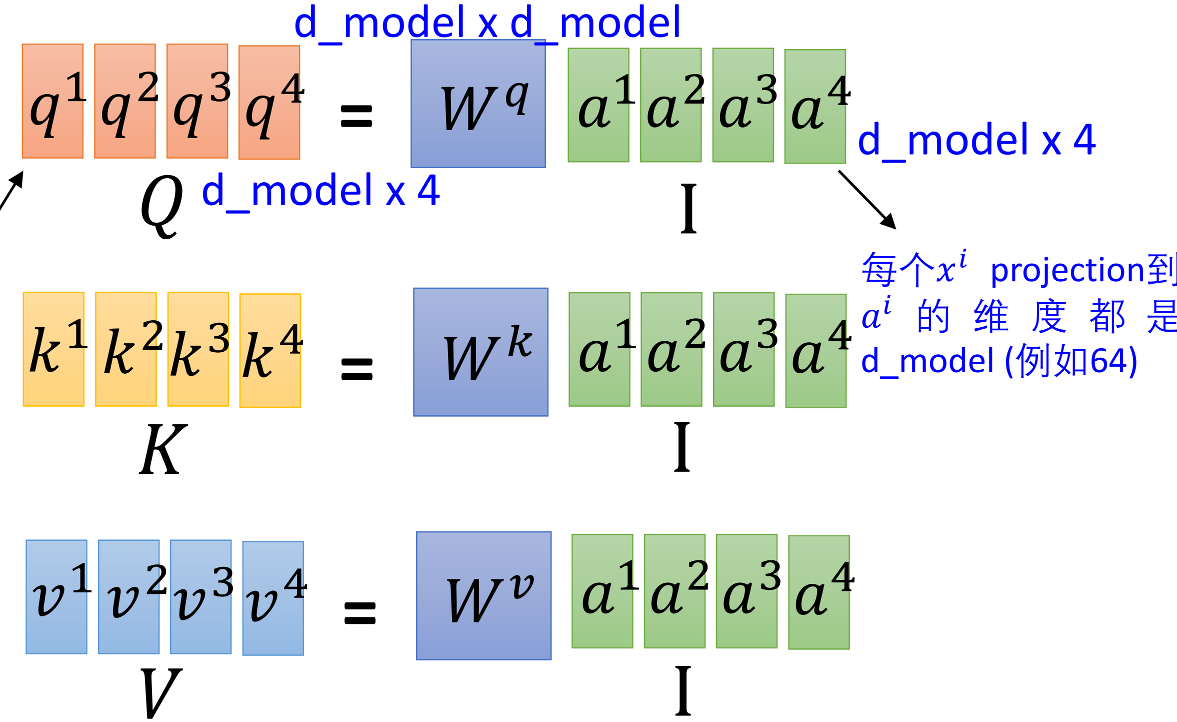
用矩阵的思维来理解 self-attention

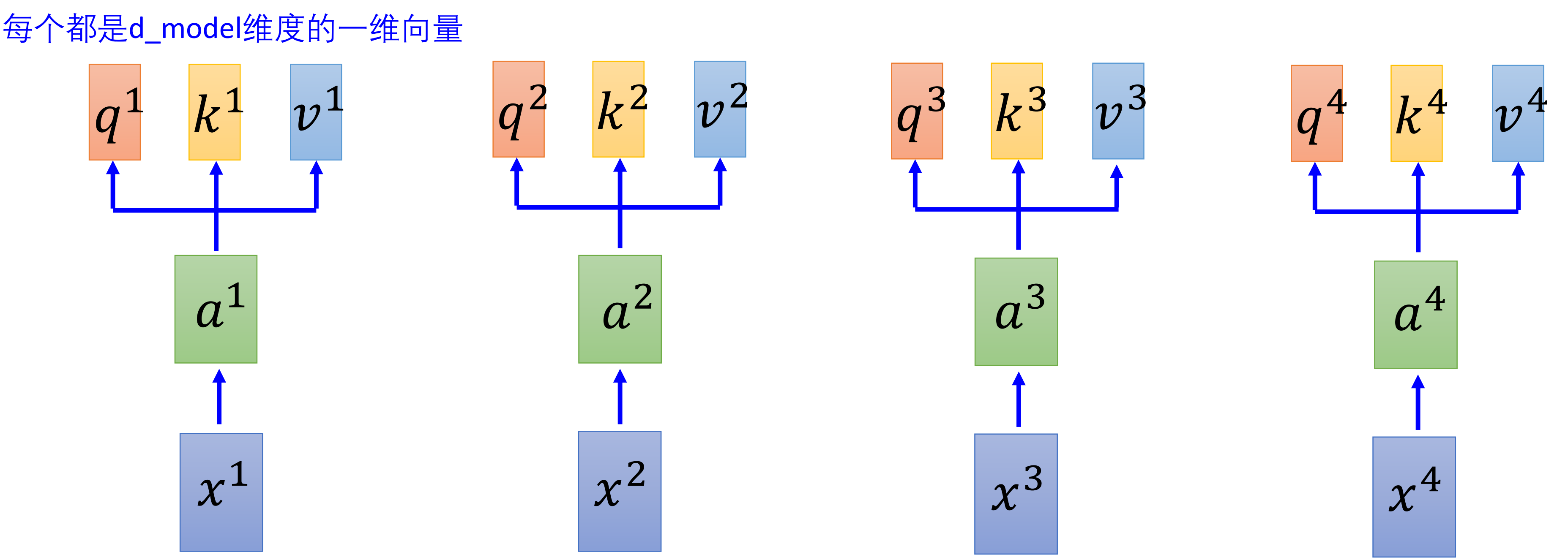
$$q^i = W^q a^i$$

$$k^i = W^k a^i$$

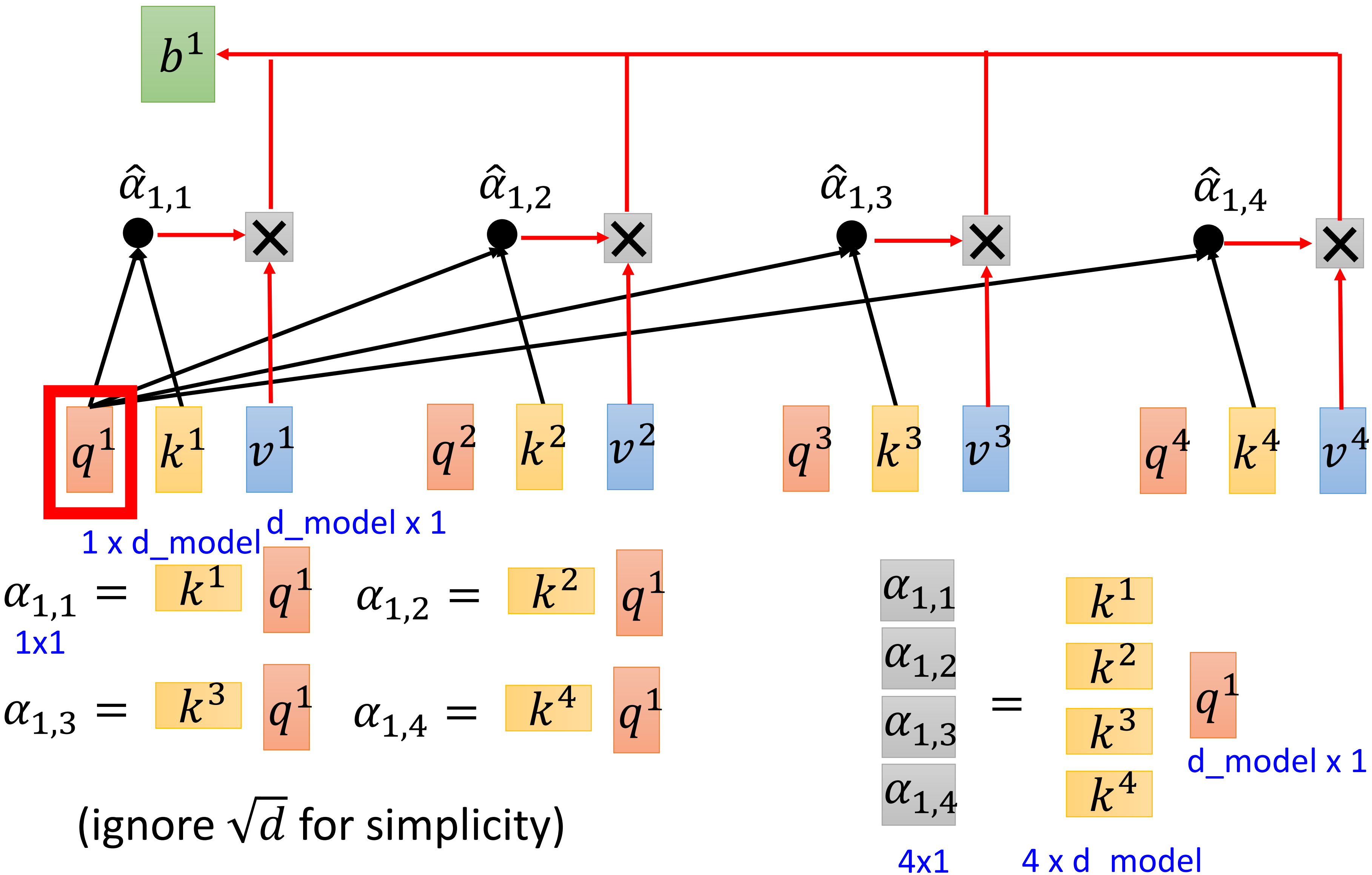
$$v^i = W^v a^i$$

拼起来





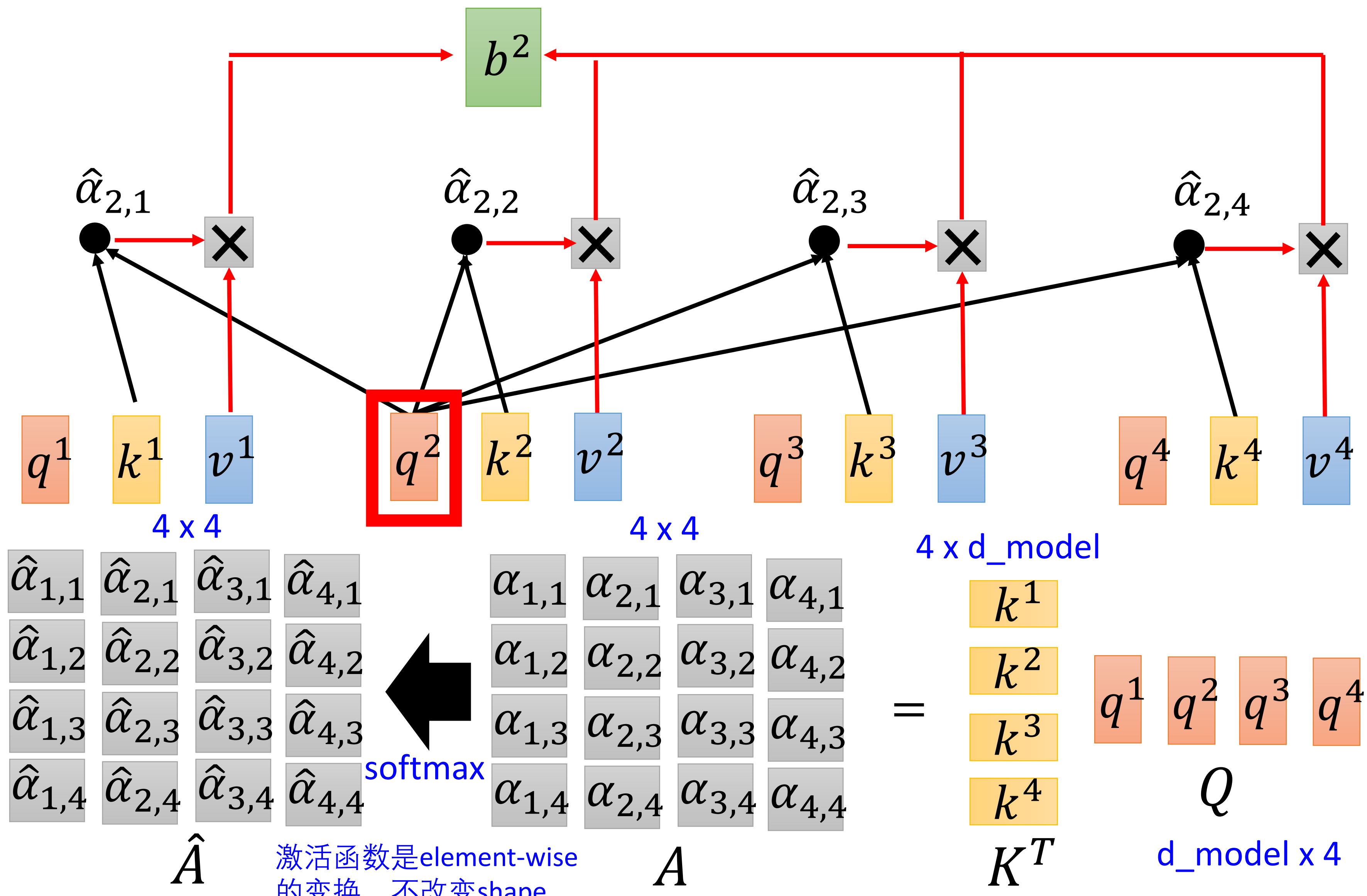
Self-attention



Self-attention

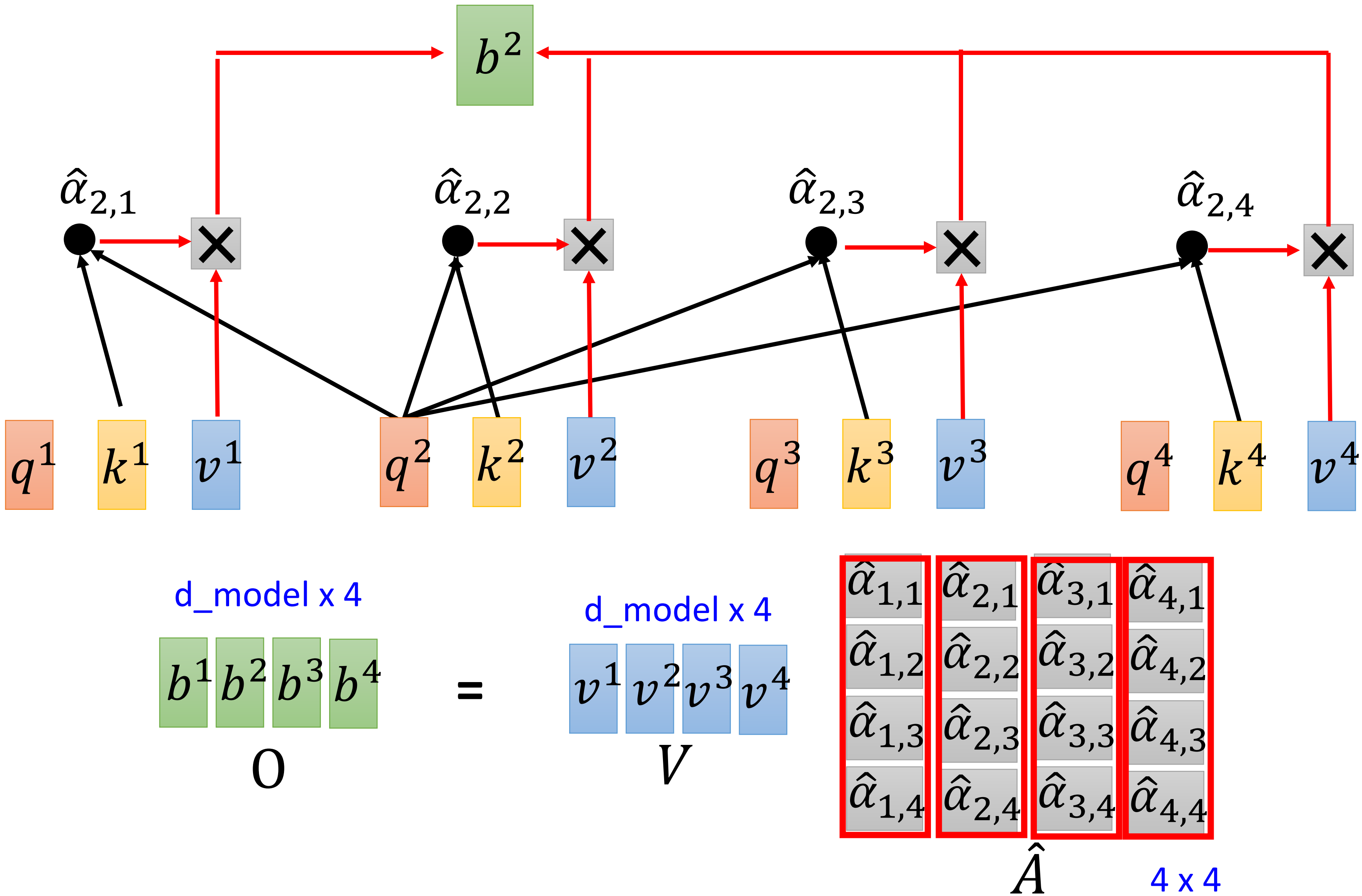
重复之前的操作

$$b^2 = \sum_i \hat{\alpha}_{2,i} v^i$$

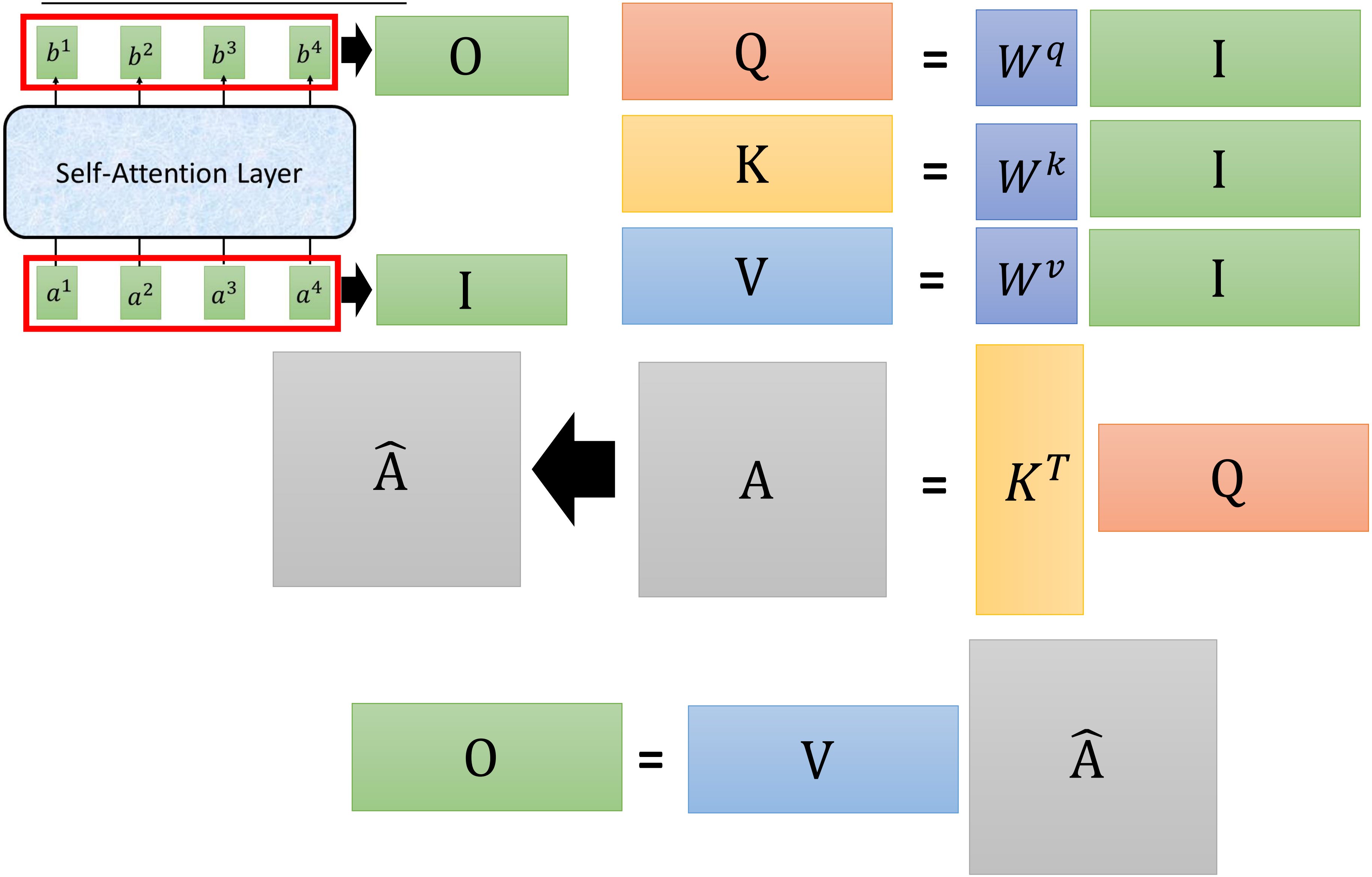


Self-attention

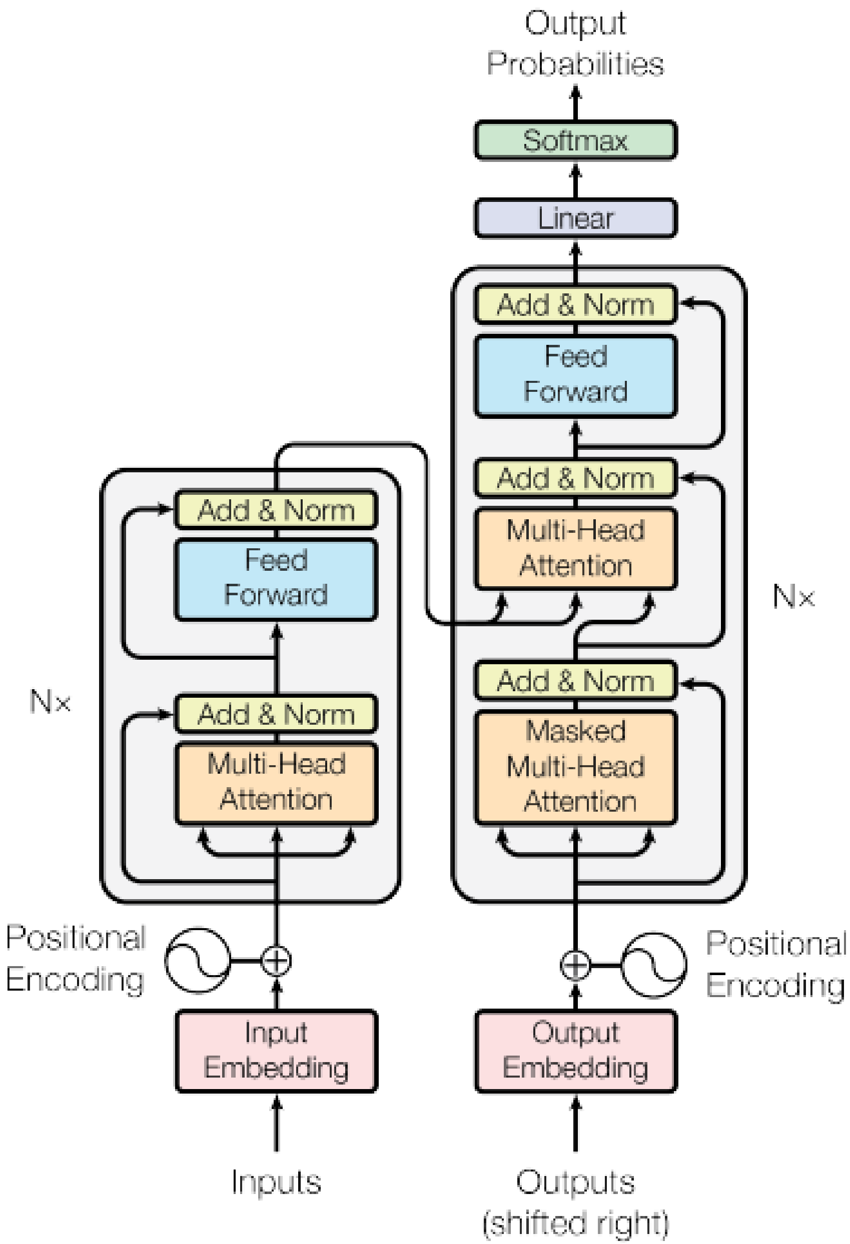
$$b^2 = \sum_i \hat{\alpha}_{2,i} v^i$$



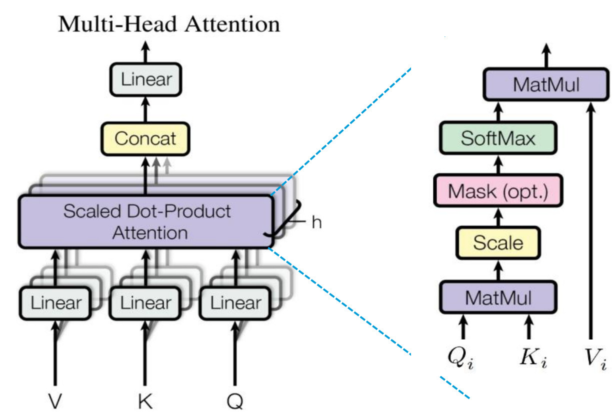
Self-attention



反正就是一堆矩陣乘法，用 GPU 可以加速



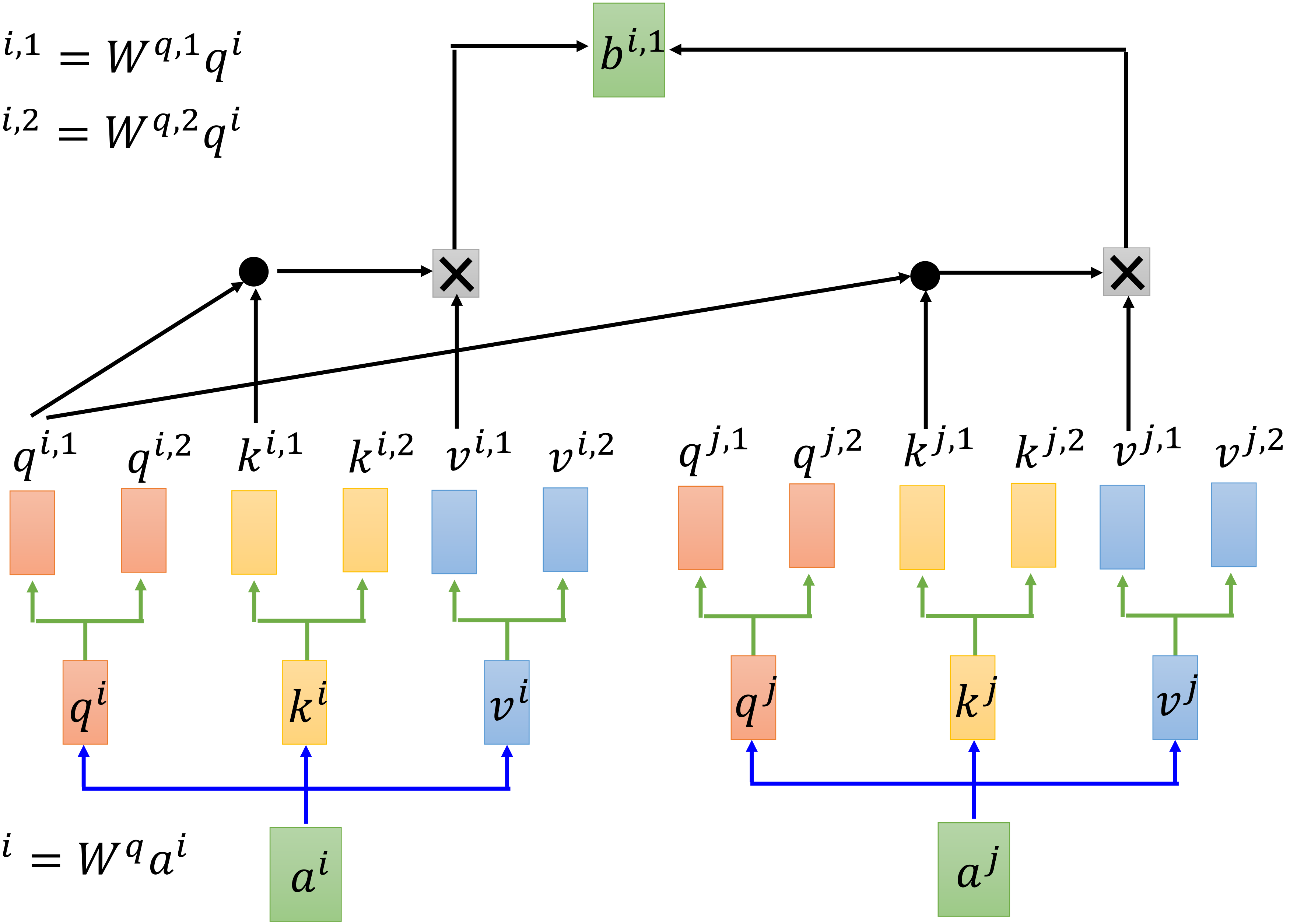
- Multi-head attention
- Masked multi-head attention
- Attention中的permutation invariant问题
- Transformer整体结构解读
- Transformer/Attention变体



Multi-head Self-attention (2 heads as example)

$$q^{i,1} = W^{q,1}q^i$$

$$q^{i,2} = W^{q,2}q^i$$

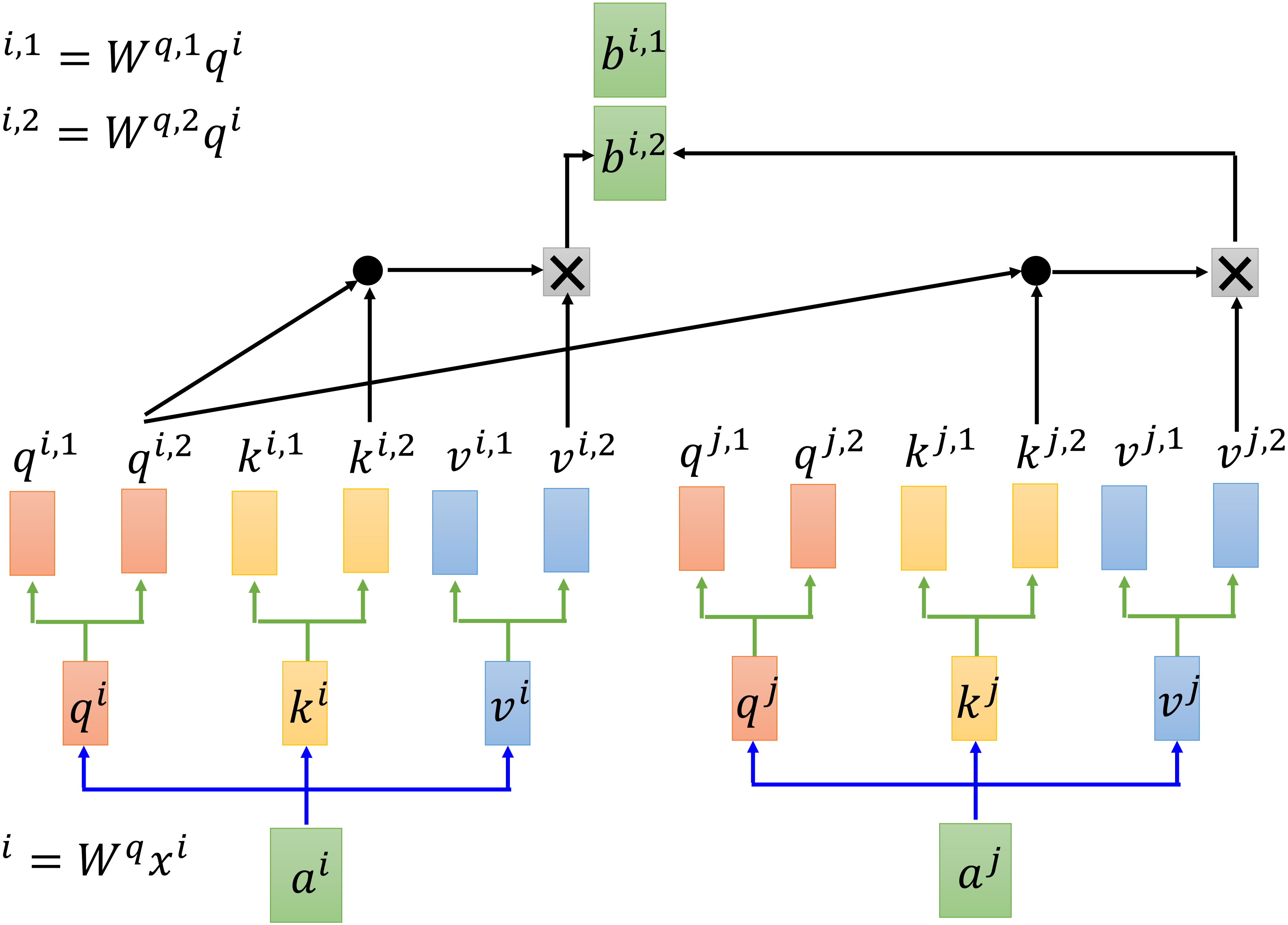


$$q^i = W^q a^i$$

Multi-head Self-attention (2 heads as example)

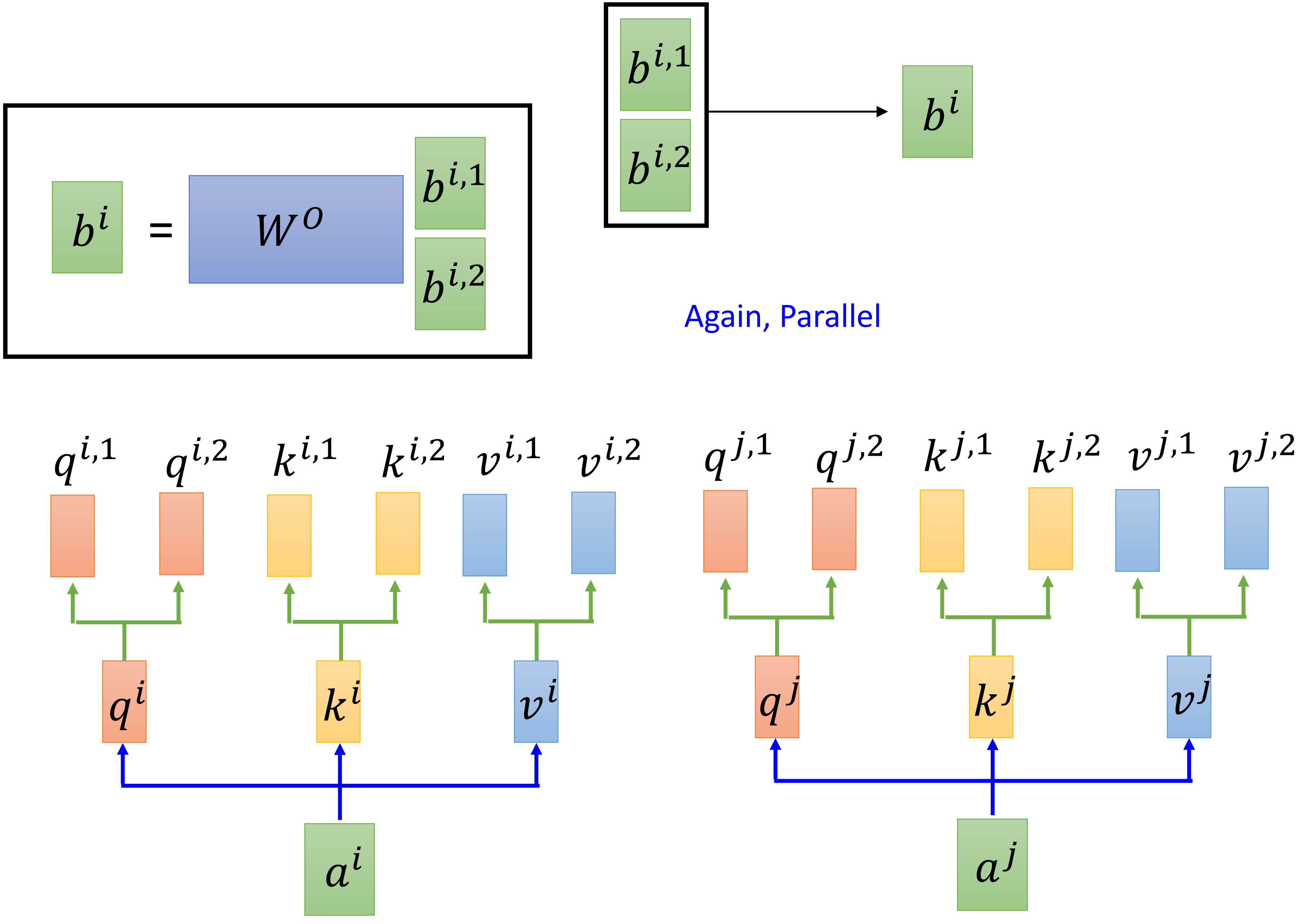
$$q^{i,1} = W^{q,1} q^i$$

$$q^{i,2} = W^{q,2} q^i$$



$$q^i = W^q x^i$$

Multi-head Self-attention (2 heads as example)



Multi-head Self-attention (2 heads as example)

```python
# multi-head attention layer
class AttentionLayer(nn.Module):
    def __init__(self, configs,
                 attention: nn.Module):
        super(AttentionLayer, self).__init__()
        d_model = configs['model']['d_model']
        n_heads = configs['model']['n_heads']
        seq_len = configs['model']['seq_len']
        self.configs = configs

        d_queries = d_model // n_heads
        d_keys = d_model // n_heads
        d_values = d_model // n_heads

        self.attention = attention
        self.query_projection = nn.Linear(d_model, d_queries * n_heads)
        self.key_projection = nn.Linear(d_model, d_keys * n_heads)
        self.value_projection = nn.Linear(d_model, d_values * n_heads)
        self.out_projection = nn.Linear(d_values * n_heads, d_model)
        self.n_heads = n_heads

    def forward(self, q, k, v, attn_mask=None):
        B, L, _ = q.size()
        _, S, _ = k.size()
        H = self.n_heads

        q = self.query_projection(q).view(B, H, L, -1)
        k = self.key_projection(k).view(B, H, S, -1)
        v = self.value_projection(v).view(B, H, S, -1)

        out, attn = self.attention(q, k, v)
        out = out.view(B, L, -1) # concat multi-head attention calculated results

        return out, attn
```

可以嵌套多种attention

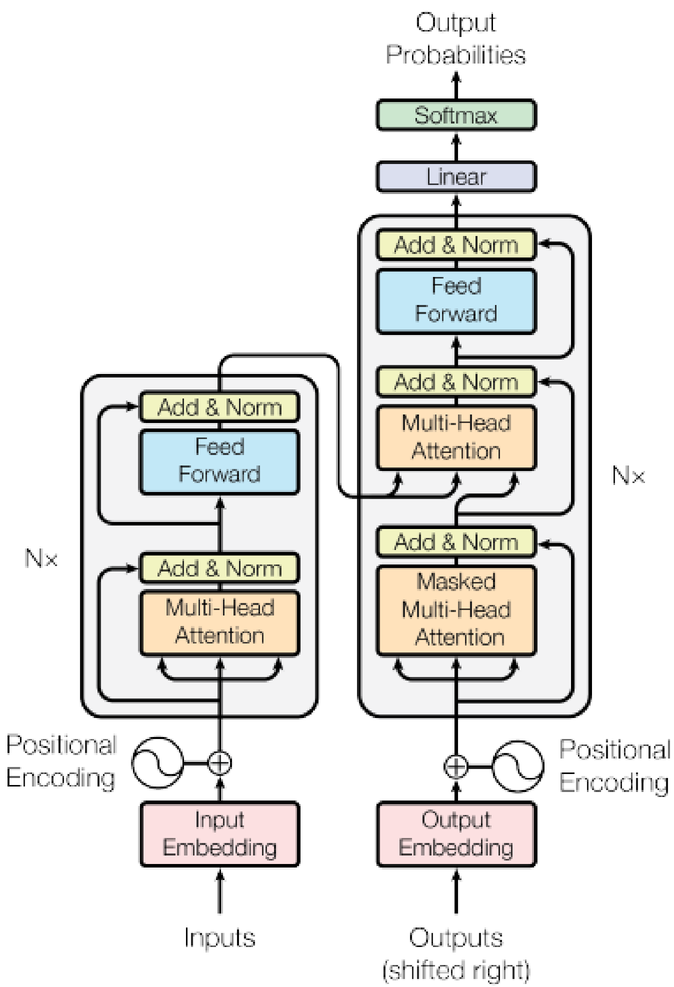
input_size 维 的 输 入 先 通 过 input embedding (linear projection) + positional encoding 到 d_model 维， 然 后 通 过 n_heads等分

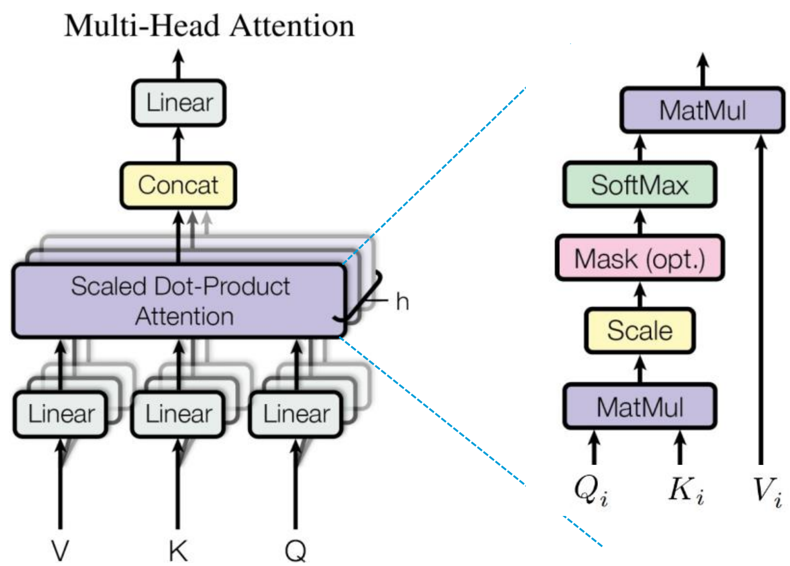
This multi-dimensionality allows the attention mechanism to jointly attend to different information from different representation at different positions

从矩阵角度来看multi-head只是reshape了一下，把原来的特征空间等分维多个特征子空间，然后再不同特征子空间的角度，关注序列中的不同信息。

即在每个特征子空间上做self-attention，每个特征子空间关注到的序列信息不同

Masked Multi-head Self-attention

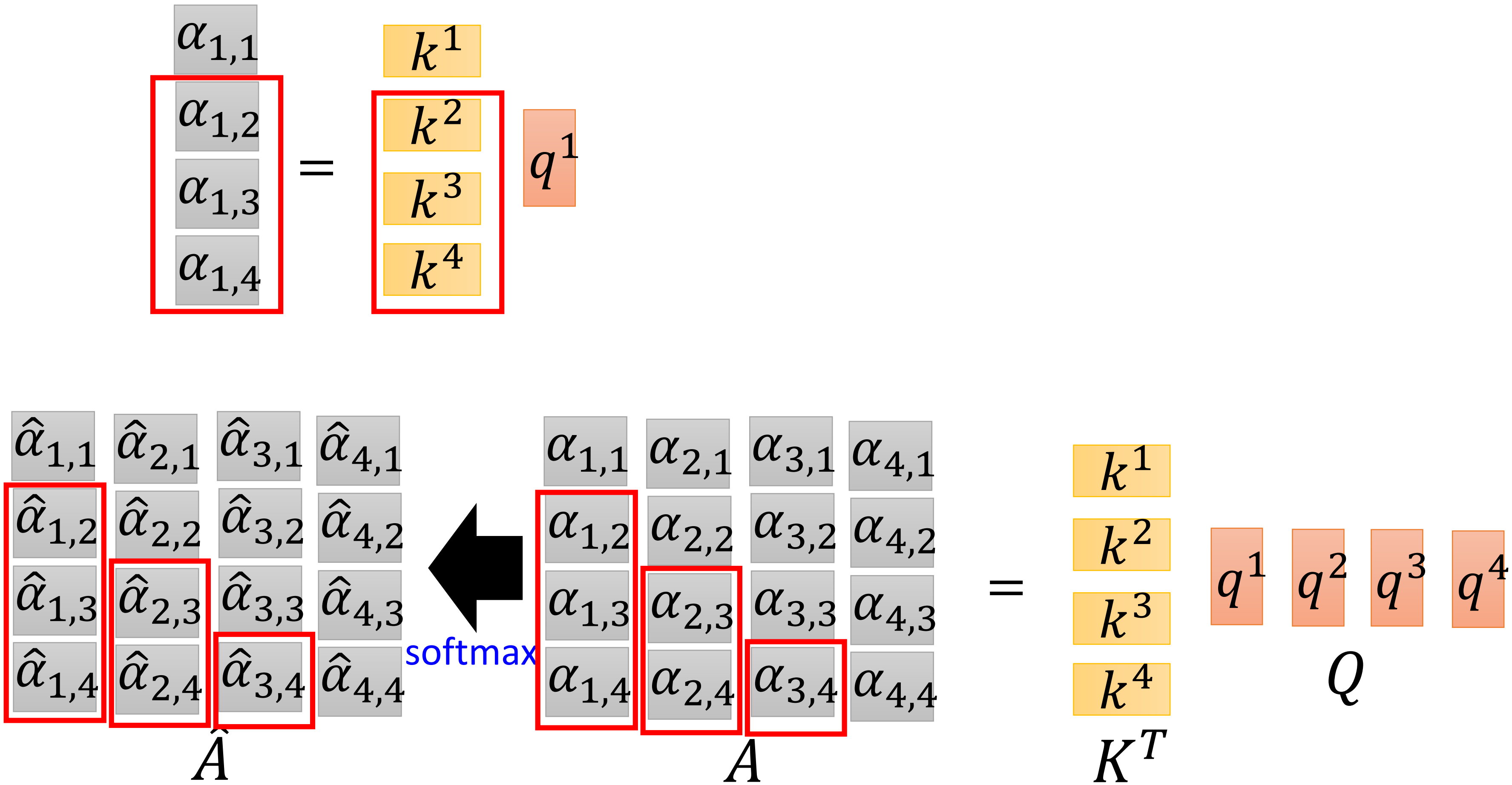




NLP：确保模型在生成每个词时，只能关注到当前和之前的词，而不能关注到未来的词；

Time-series：确保模型只能关注到当前时间步及之前时间步的信息，而不用未来信息作弊

Masked Multi-head Self-attention



应该让主对角线下方的
attention score变成0

Masked Multi-head Self-attention

```python
class MaskGenerator():
    def __init__(self, batch_size, seq_length, device):
        # broadcast on the attention heads dimension
        mask_shape = [batch_size, 1, seq_length, seq_length]
        with torch.no_grad():
            self.mask = torch.triu(torch.ones(mask_shape, dtype=torch.bool), diagonal=1).to(device)

    @property
    def mask(self):
        return self.mask
```

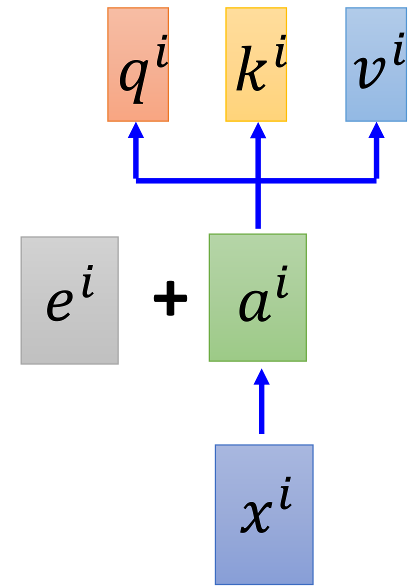
Mask那些位于主对角线下方的元素

```python
# self-attention
class FullAttention(nn.Module):
    def __init__(self, configs):
        super(FullAttention, self).__init__()
        self.mask = configs['model']['mask']
        self.dropout = nn.Dropout(p=configs['model']['dropout'])

    def forward(self, q, k, v, attn_mask=None):
        # qkv的输入维度分别是batch size, attention heads, sequence length, embedding dim
        # b, h, d分别代表batch size, attention heads个数, 以及embedding维度
        # l和s分别是query和key的sequence length, 只是因为输出矩阵bhss是不合法的, 需要取两个变量
        # QK^T, lxdxdxs=lxs, 再考虑batch size和attention heads为bxhxlxs的张量
        attn_scores = torch.einsum("bhld,bhsd->bhls", q, k)
        if self.mask:
            attn_mask = MaskGenerator(batch_size=q.size(0), seq_length=q.size(2), device=q.device)
            attn_scores.masked_fill_(attn_mask.mask, -np.inf)

        # 只考虑最后两个维度的话lxs, 即softmax计算每个query对于所有key的attention scores
        attn_scores = self.dropout(torch.softmax(1. / sqrt(q.size(-1)) * attn_scores, dim=-1))
        weighted_values = torch.einsum("bhls,bhsd->bhld", attn_scores, v)

        # .contiguous()确保张量的数据在内存中是连续的,以提高后续操作的效率
        # The .contiguous() ensures the memory of the tensor is stored contiguously
        # which helps avoid potential issues during processing.
        return attn_scores, weighted_values.contiguous()
```

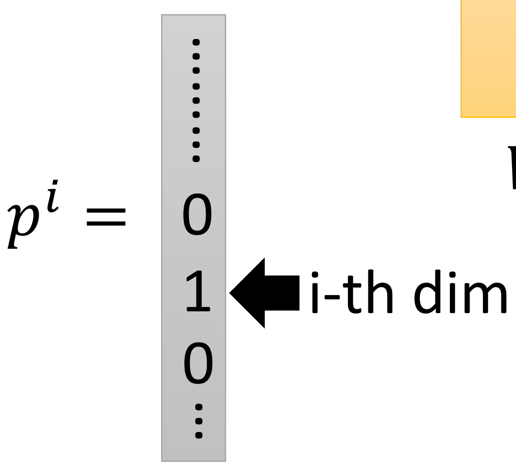
$$\hat{\alpha}_{1,i} = exp(\alpha_{1,i})/\sum_j exp(\alpha_{1,j})$$

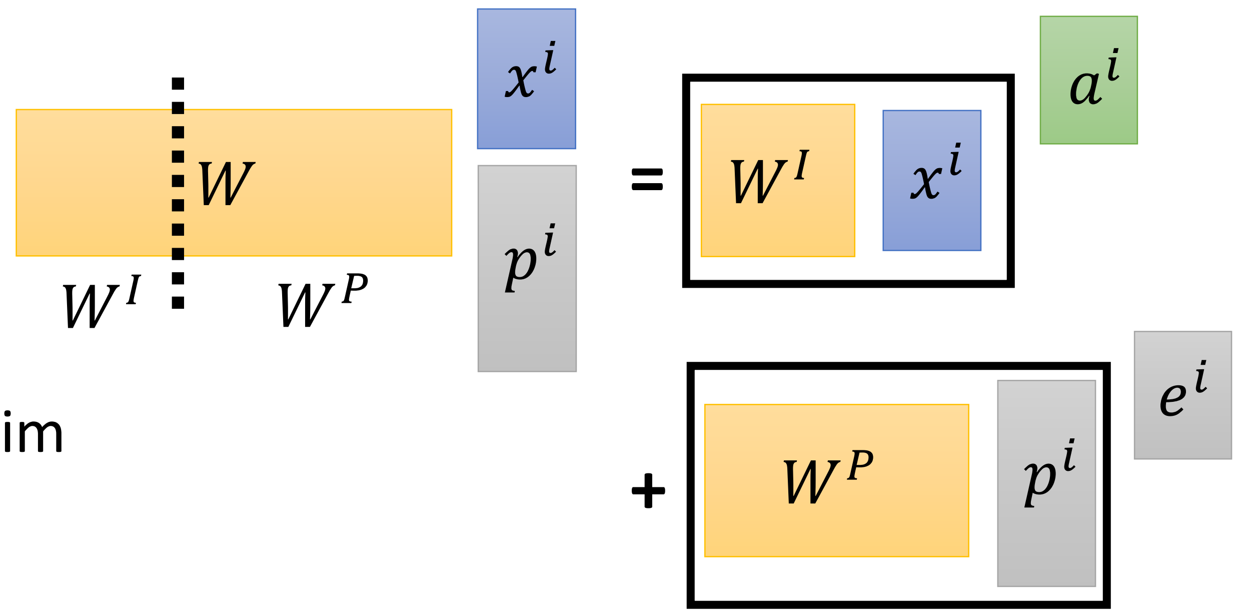
让那些元素值变成-inf，这样过softmax之后probability就变成了0

Positional Encoding



- **No position information** in self-attention.
- Original paper: each position has a unique positional vector e^i (not learned from data)
- In other words: each x^i appends a one-hot vector p^i

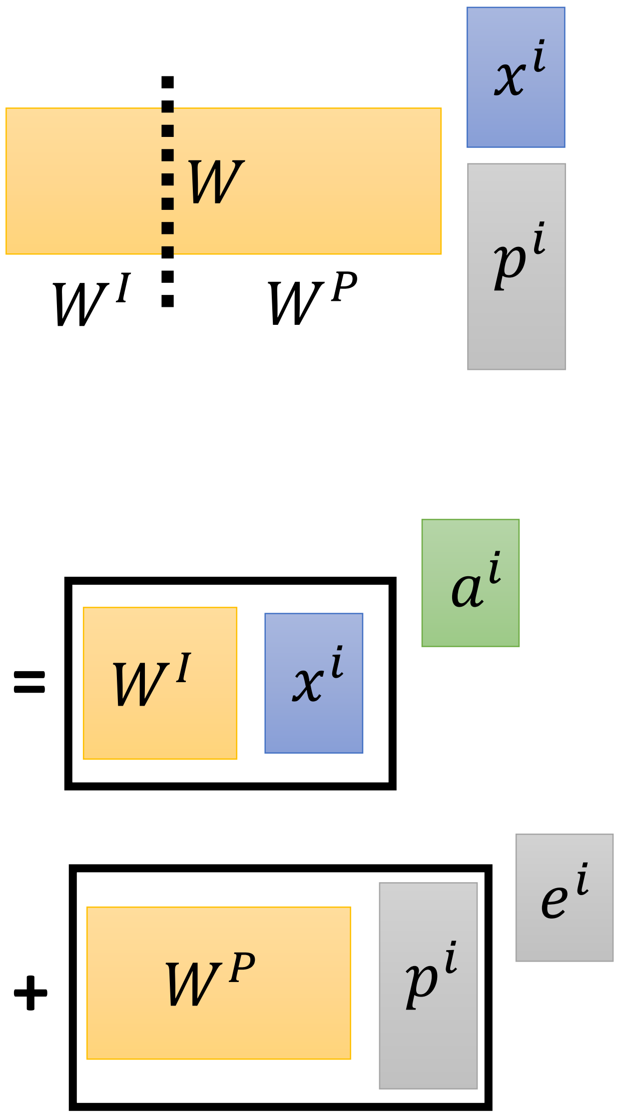


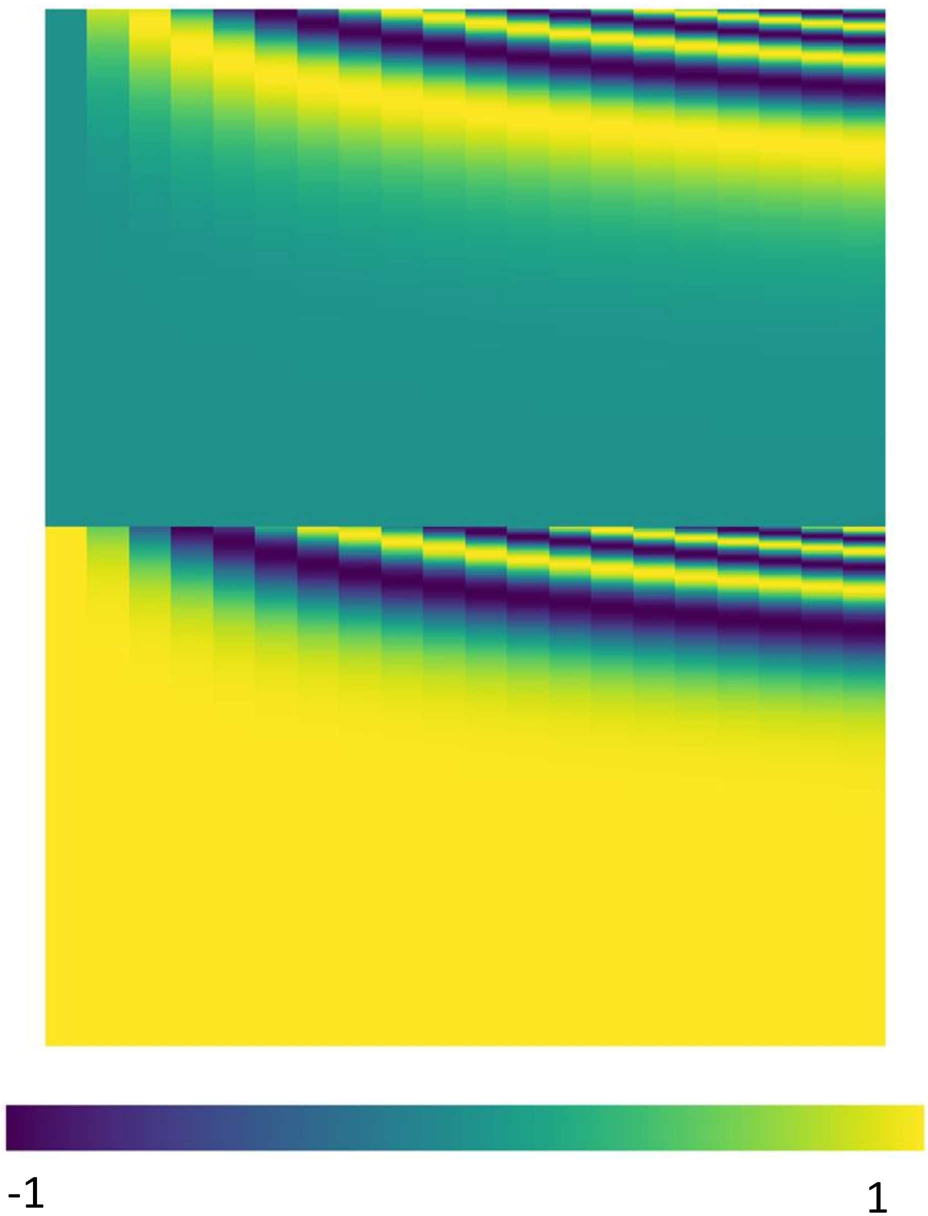


Attention is all you need: https://arxiv.org/pdf/1706.03762

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

```
class PositionalEmbedding(nn.Module):
    def __init__(self, d_model, max_len=5000):
        super(PositionalEmbedding, self).__init__()
        # Compute the positional encodings once in log space.
        pe = torch.zeros(max_len, d_model).float()
        pe.require_grad = False

        position = torch.arange(0, max_len).float().unsqueeze(1)
        div_term = (torch.arange(0, d_model, 2).float()
                    * -(math.log(10000.0) / d_model)).exp()

        pe[:, 0::2] = torch.sin(position * div_term)  2i
        pe[:, 1::2] = torch.cos(position * div_term)  2i+1

        pe = pe.unsqueeze(0)
        self.register_buffer('pe', pe)

    def forward(self, x):
        return self.pe[:, :x.size(1)]
```

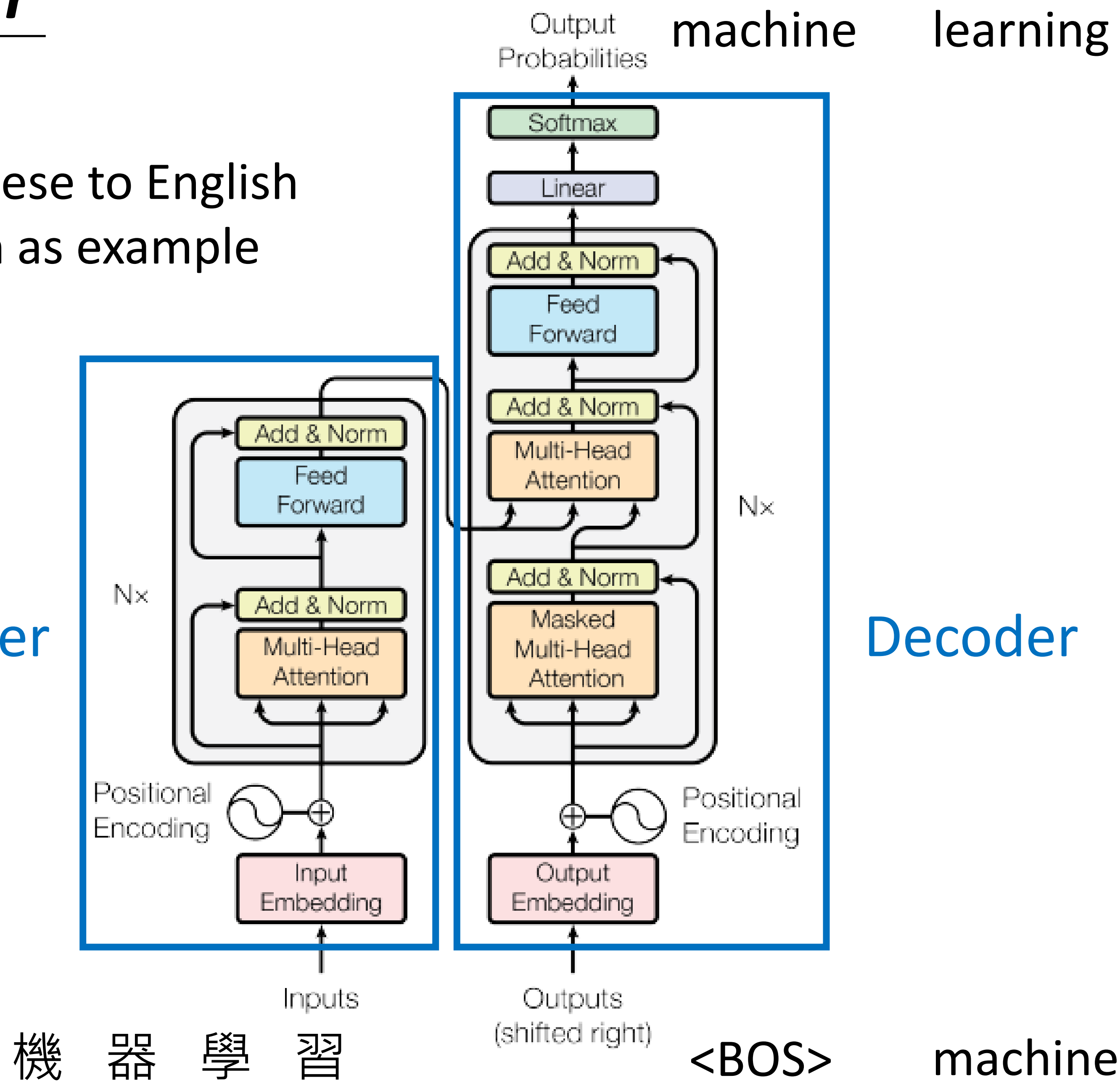




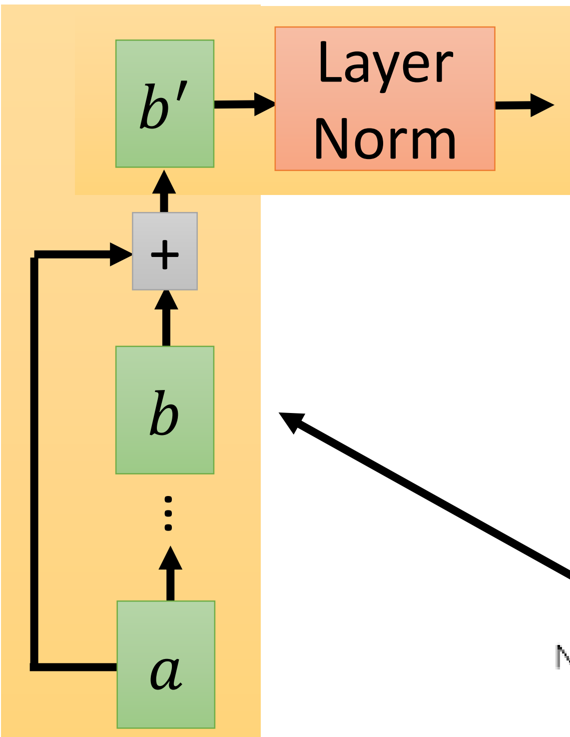
source of image: http://jalammar.github.io/illustrated-transformer/

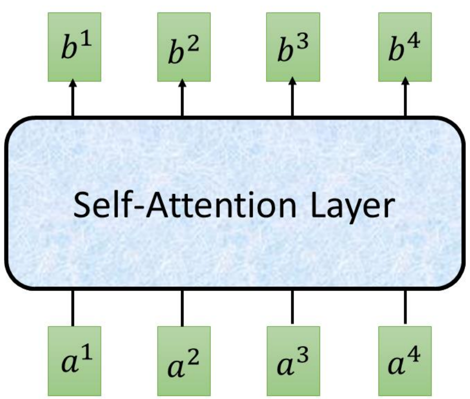
Transformer

Using Chinese to English translation as example



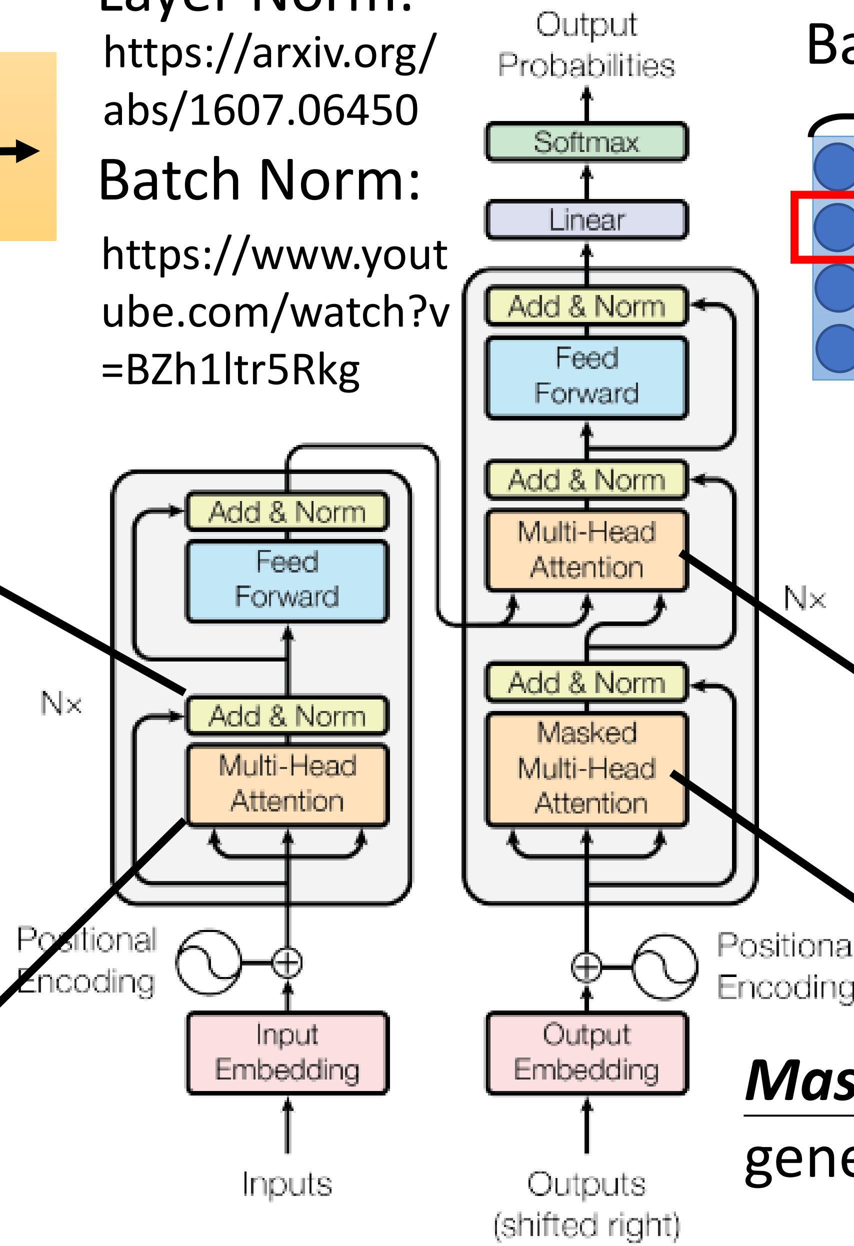
Transformer

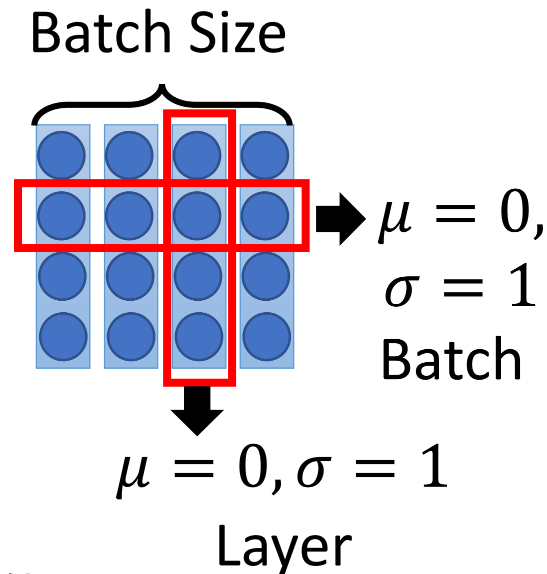




Layer Norm: https://arxiv.org/abs/1607.06450

Batch Norm: https://www.youtube.com/watch?v=BZh1ltr5Rkg





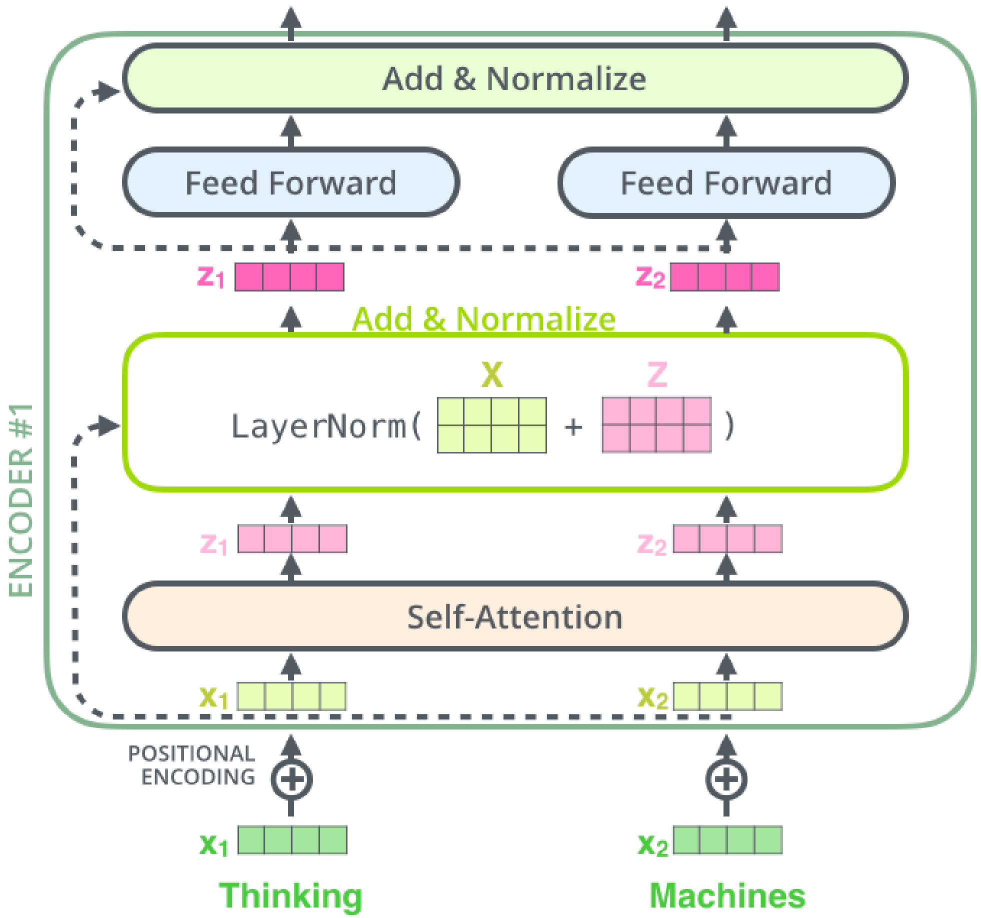
attend on the input sequence

Masked: attend on the generated sequence

Transformer

http://jalammar.github.io/illustrated-transformer/

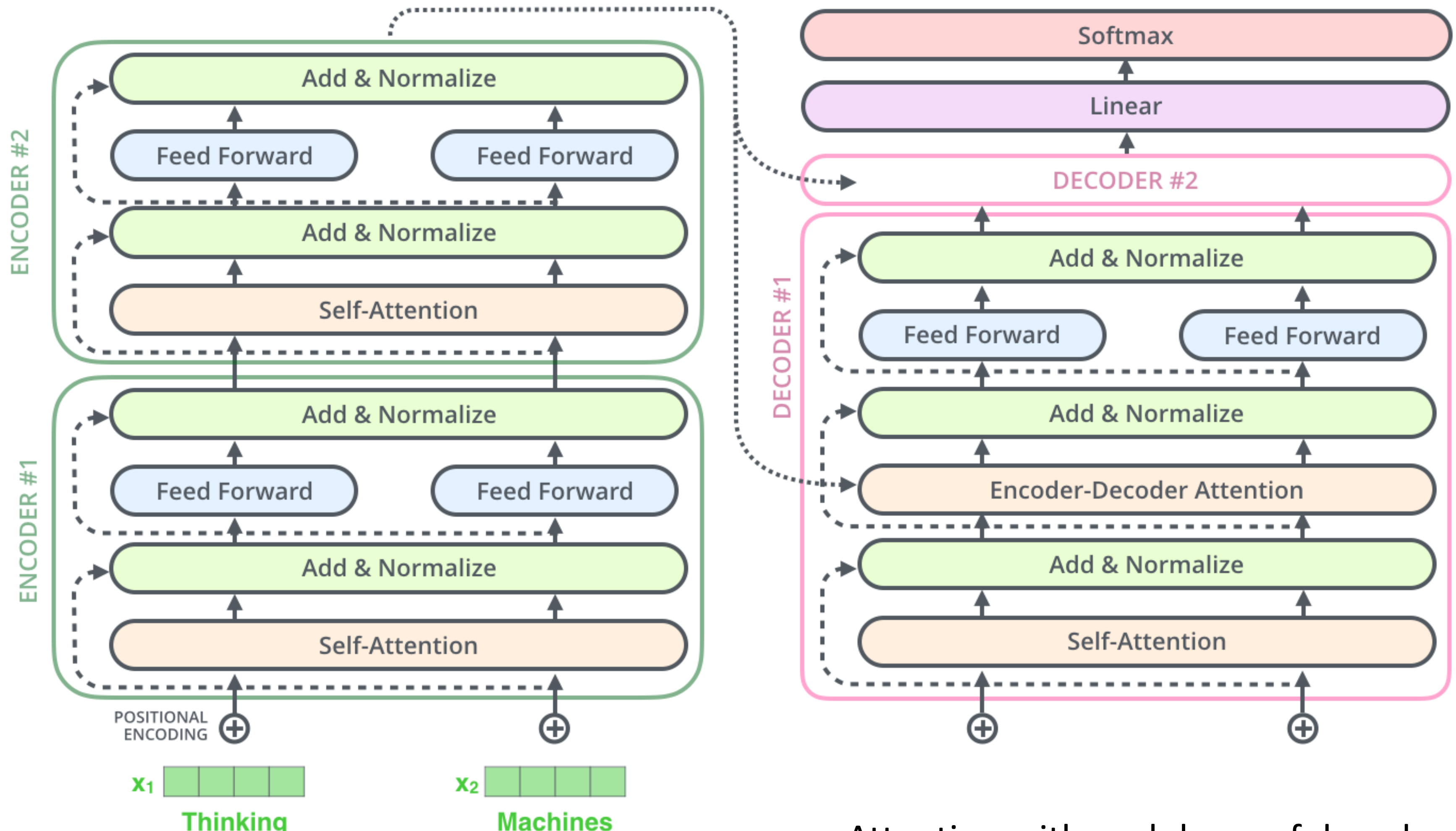
Encoder part



Transformer

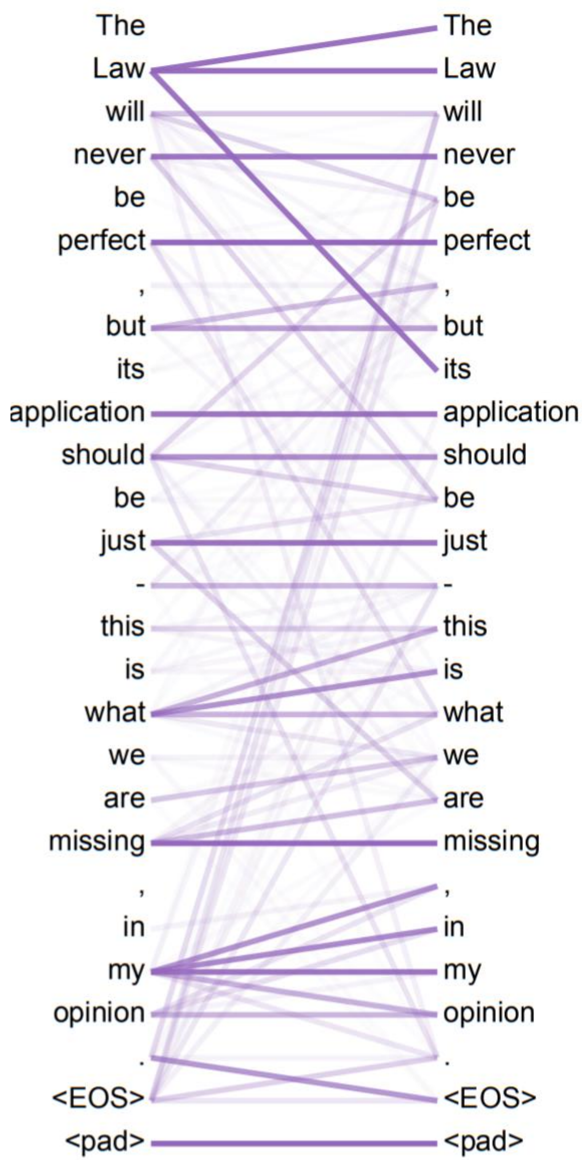
Encoder可以有很多层

Encoder-Decoder



Attention with each layer of decoder

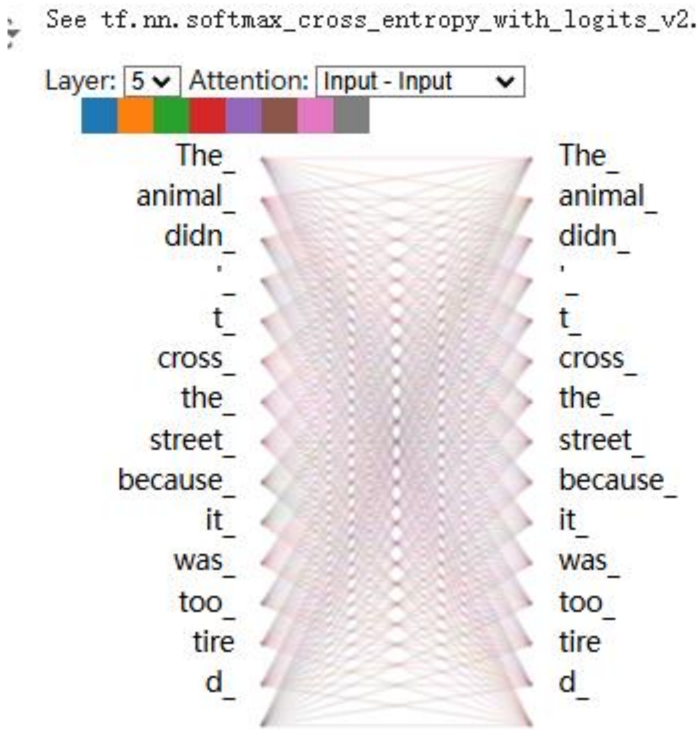
Attention Visualization

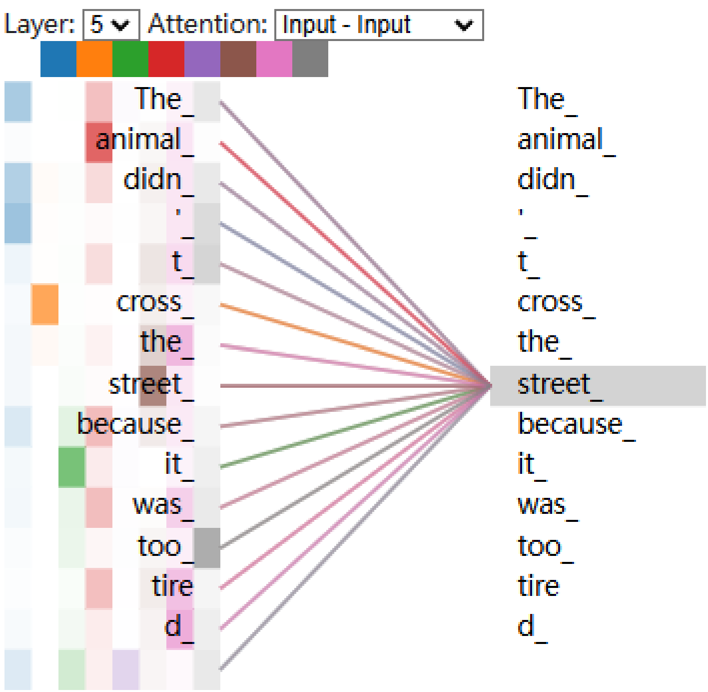


https://arxiv.org/abs/1706.03762

Attention Visualization

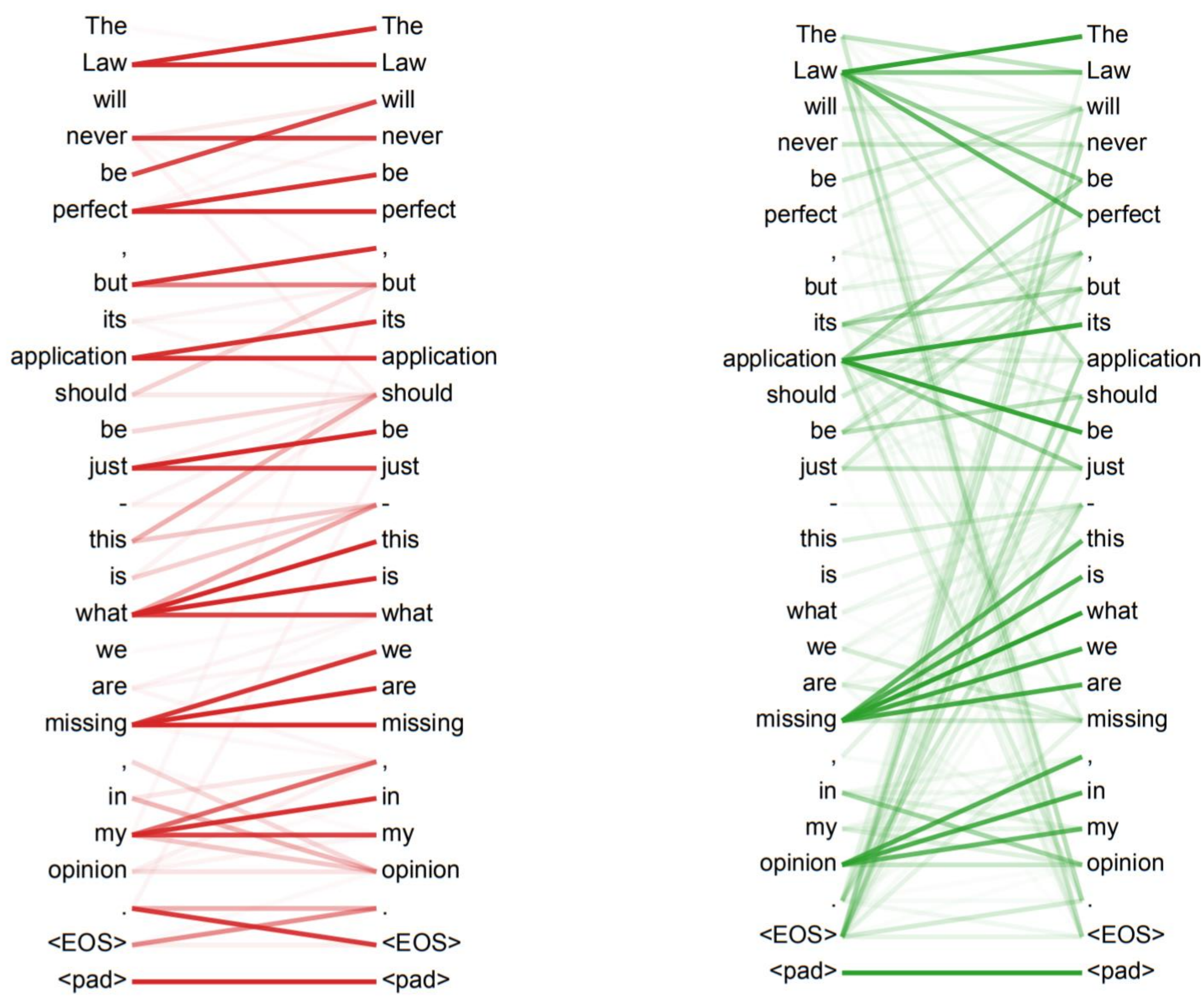
https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb#scrollTo=OJKU36QAfqOC





Multi-head Attention

每个attention head学到的知识不同



https://arxiv.org/pdf/2202.07125

Transformers in Time Series: A Survey

Qingsong Wen[1], Tian Zhou[2], Chaoli Zhang[2], Weiqi Chen[2], Ziqing Ma[2], Junchi Yan[3], Liang Sun[1]
[1]DAMO Academy, Alibaba Group, Bellevue, USA
[2]DAMO Academy, Alibaba Group, Hangzhou, China
[3]Department of CSE, MoE Key Lab of Artificial Intelligence, Shanghai Jiao Tong University
{qingsong.wen, tian.zt, chaoli.zcl, jarvus.cwq, maziqing.mzq, liang.sun}@alibaba-inc.com,
yanjunchi@sjtu.edu.cn