2023-2024学年第一学期本科生课程

《神经网络与深度学习》

第九节：自然语言处理基础

主讲人：戴金晟(副教授，博士生导师)

daijincheng@bupt.edu.cn

神经网络与深度学习课程组

北京郵電大學

Beijing University of Posts and Telecommunications

内容导览

- 词向量嵌入
- 循环神经网络(RNN)
- Seq2Seq问题与注意力机制
- 自注意力/多头注意力/交叉注意力
- Transformer
- ELMO/BERT/GPT

内容导览

- 词向量嵌入
- 循环神经网络(RNN)
- Seq2Seq问题与注意力机制
- 自注意力/多头注意力/交叉注意力
- Transformer
- ELMO/BERT/GPT

计算机如何如何读懂人类的文字?

1-of-N Encoding

apple = [1 0 0 0 0]

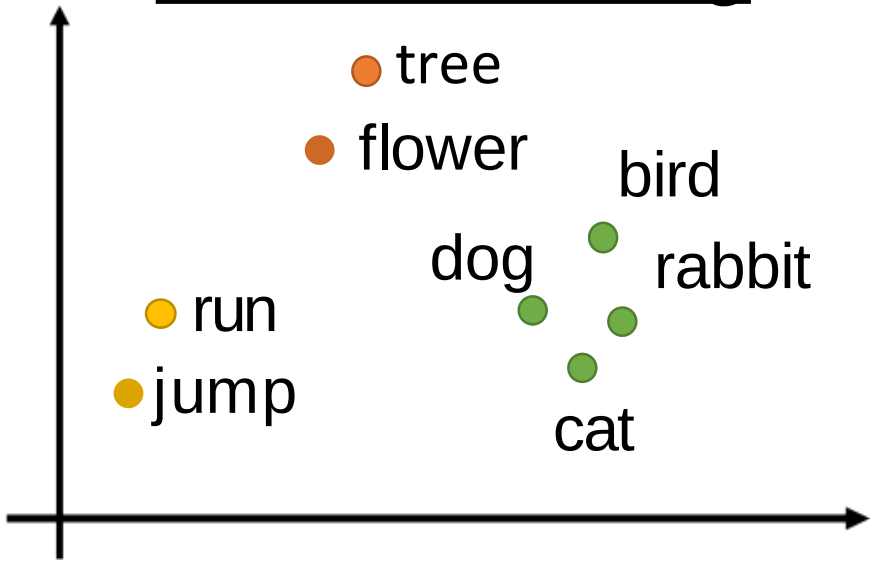
bag = [0 1 0 0 0]

cat = [0 0 1 0 0]

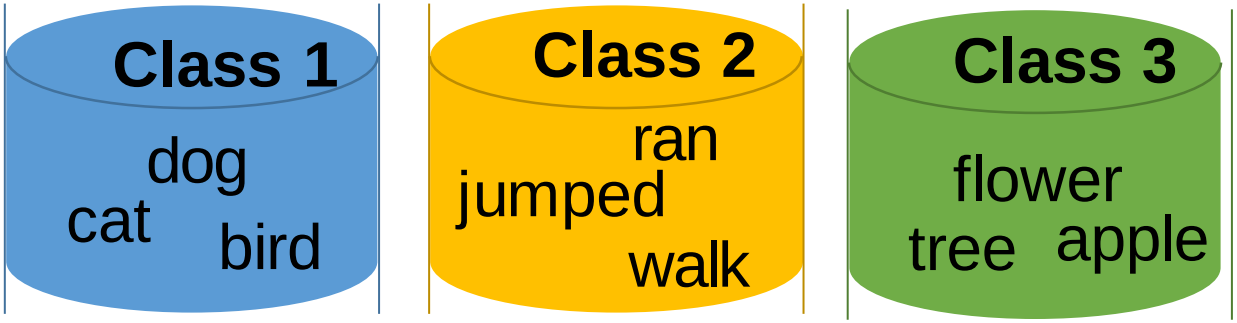
dog = [0 0 0 1 0]

elephant = [0 0 0 0 1]

Word Embedding

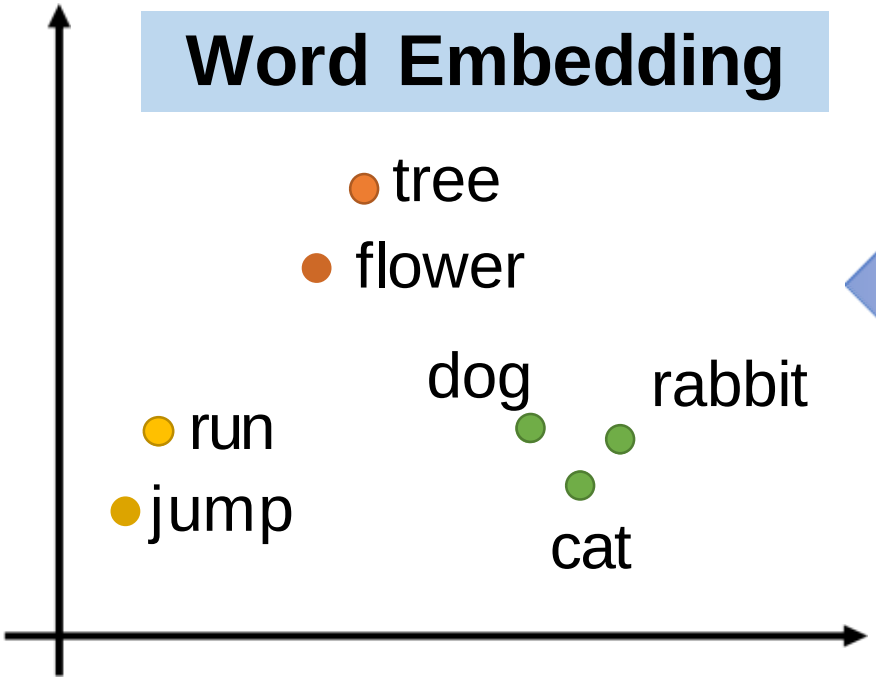


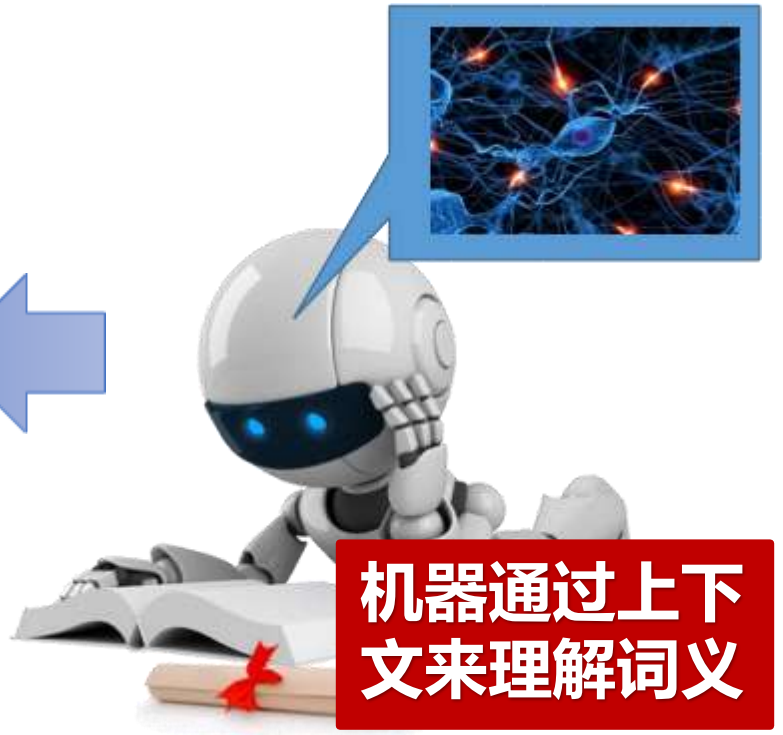
Word Class



词嵌入 (Word Embedding)

- 机器在没有监督的情况下通过阅读大量文档来学习单词的含义





北京属于中国

南京属于中国

“北京”和“南京”是非常相似的两个词

如何利用词的上下文?

- Count based
  - If two words w_i and w_j frequently co-occur, $V(w_i)$ and $V(w_j)$ would be close to each other.
  - E.g. Glove Vector: http://nlp.stanford.edu/projects/glove/

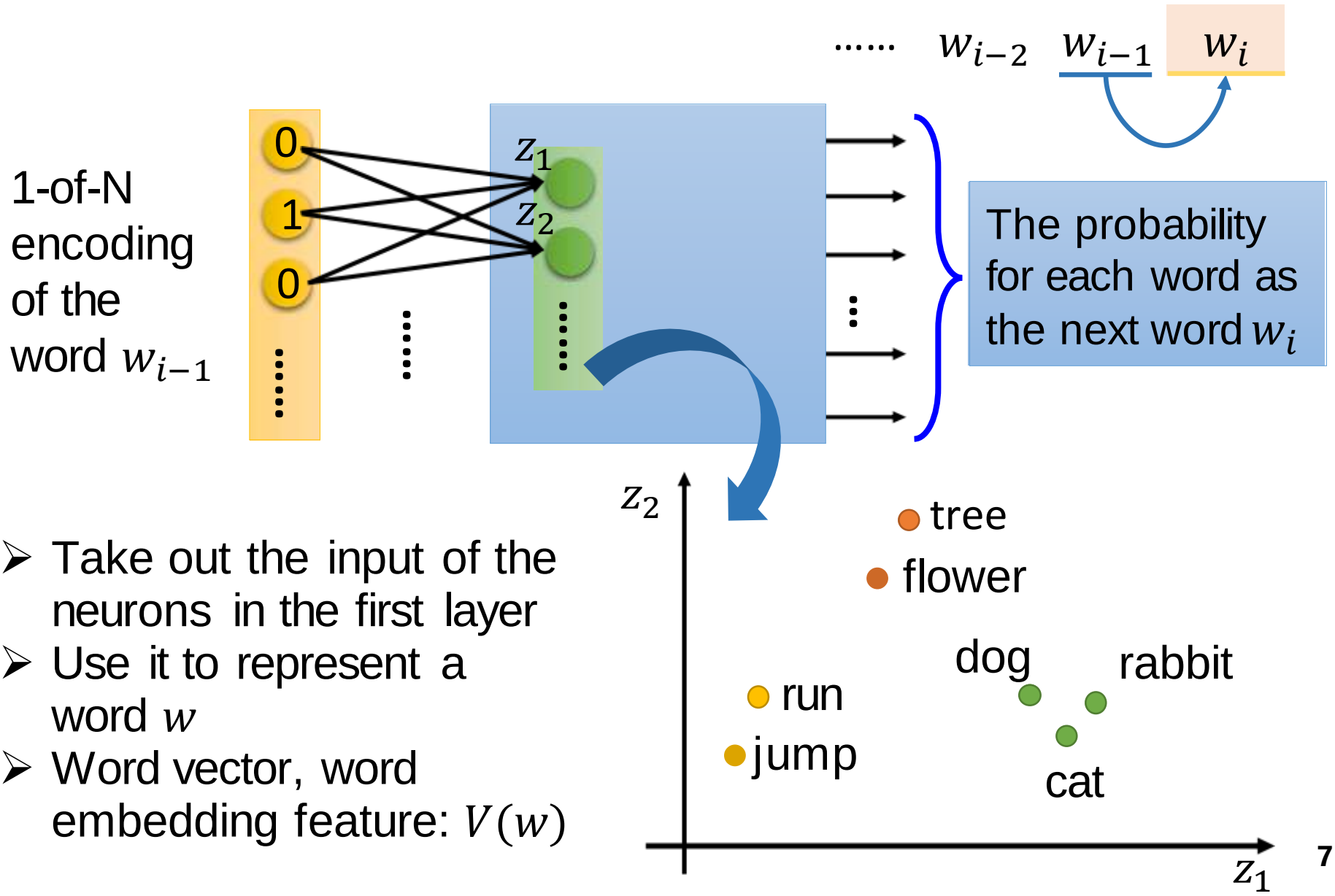
$V(w_i) \cdot V(w_j)$ ⟷ $N_{i,j}$

Inner product

Number of times w_i and w_j in the same document

- Prediction based

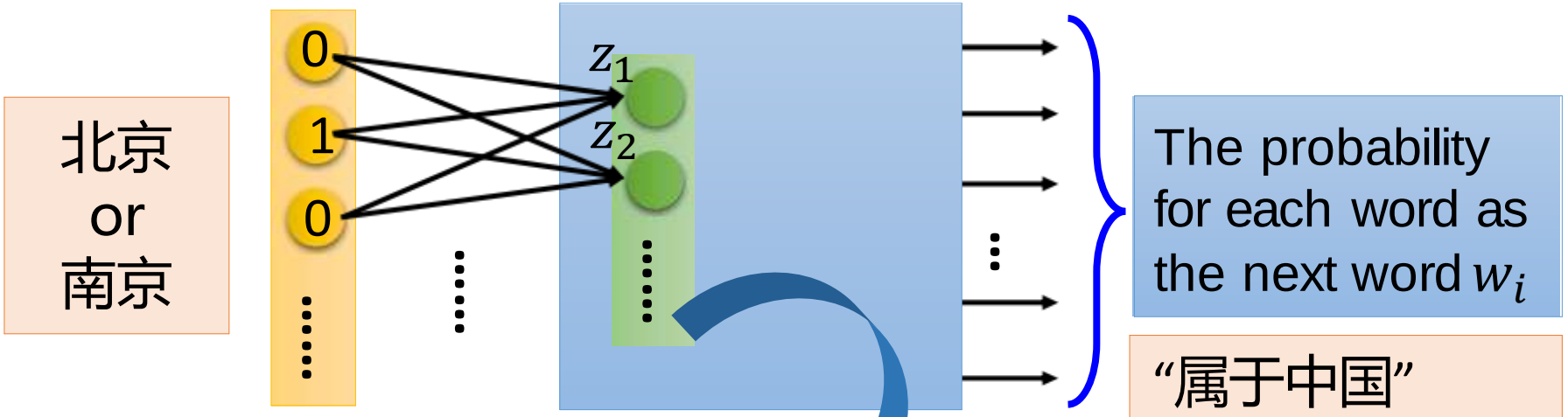
Prediction-based



- Take out the input of the neurons in the first layer
- Use it to represent a word w
- Word vector, word embedding feature: $V(w)$

Prediction-based

- You shall know a word by the company it keeps



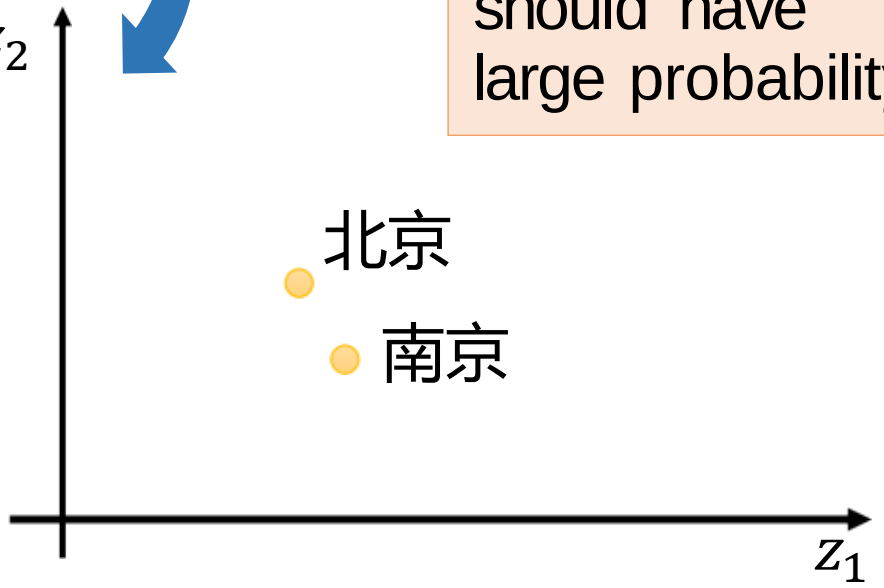
Training text:

…… 北京 属于中国 ……

w_{i-1} w_i

…… 南京 属于中国 ……

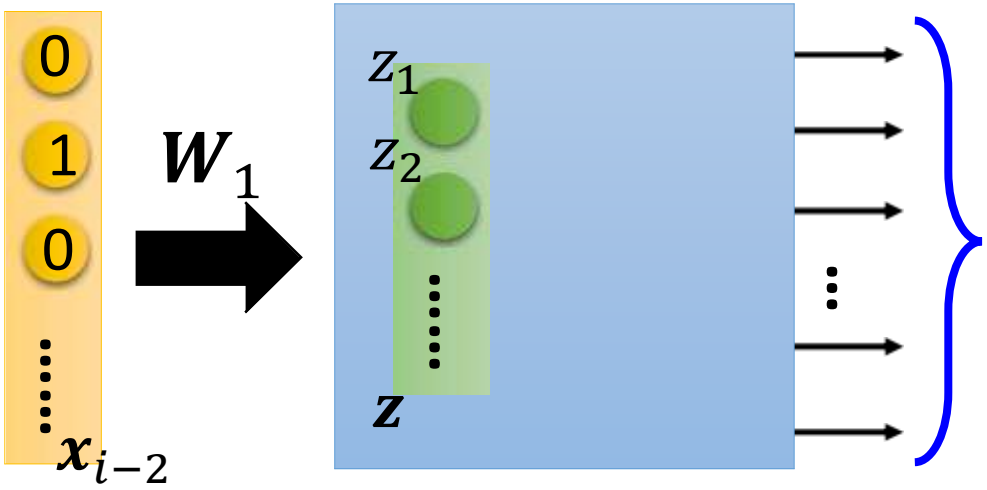
w_{i-1} w_i



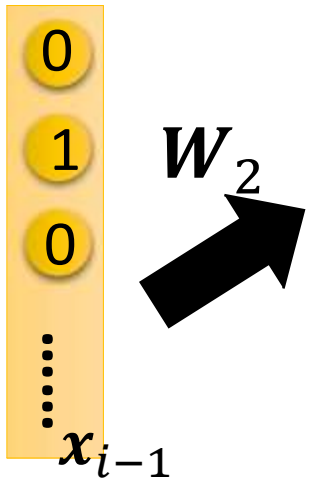
Prediction-based

- Sharing Parameters

1-of-N encoding of the word w_{i-2}



1-of-N encoding of the word w_{i-1}



The length of x_{i-1} and x_{i-2} are both .$|V|$

The length of z is $|Z|$.

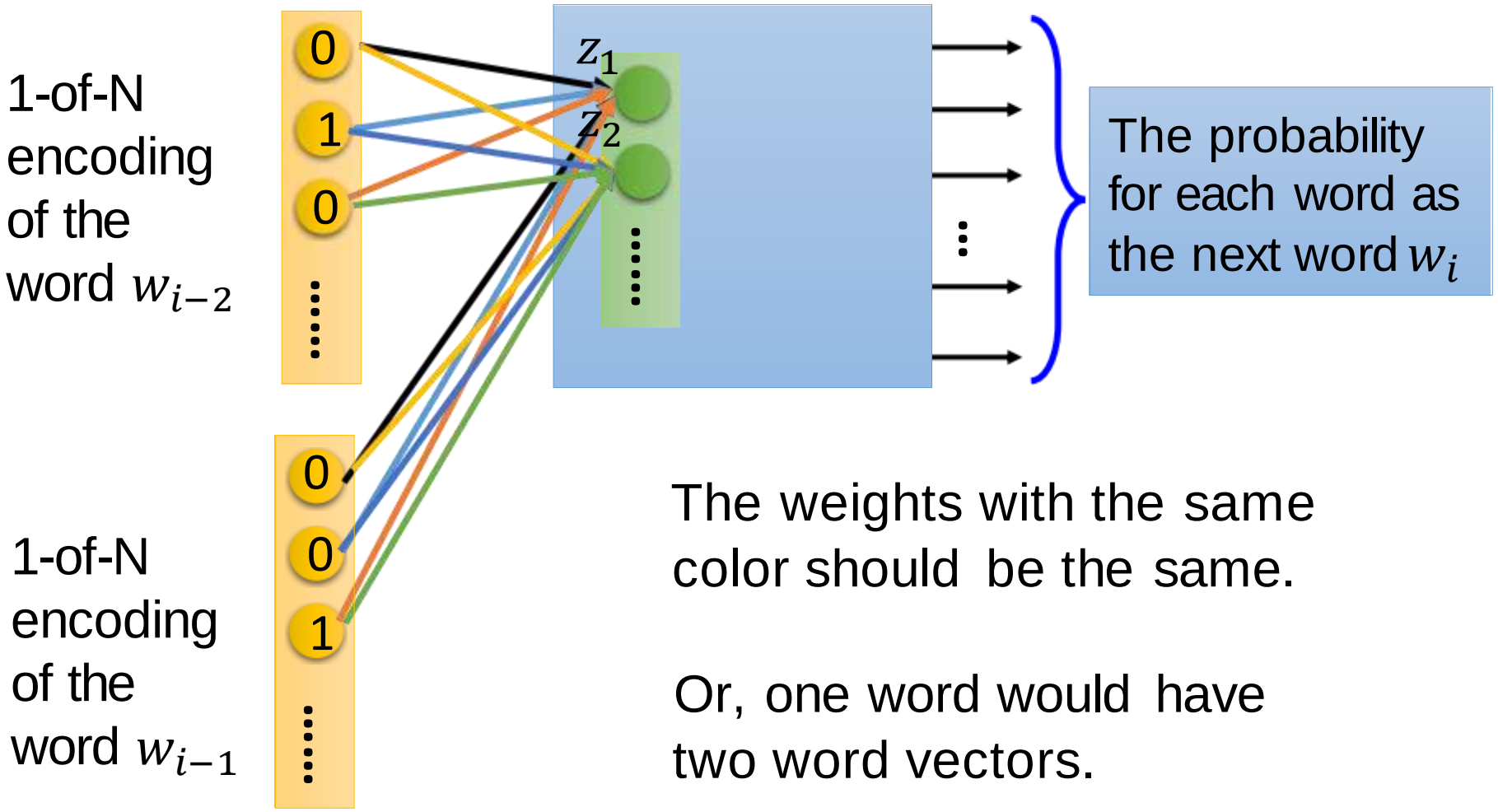
$$z = W_1 x_{i-2} + W_2 x_{i-1}$$

The weight matrix W_1 and W_2 are both $|Z| \times |V|$matrices.

$$W_1 = W_2 = W \quad z = W(x_{i-2} + x_{i-1})$$

Prediction-based

- Sharing Parameters

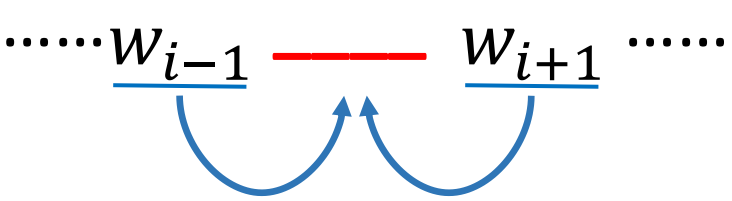


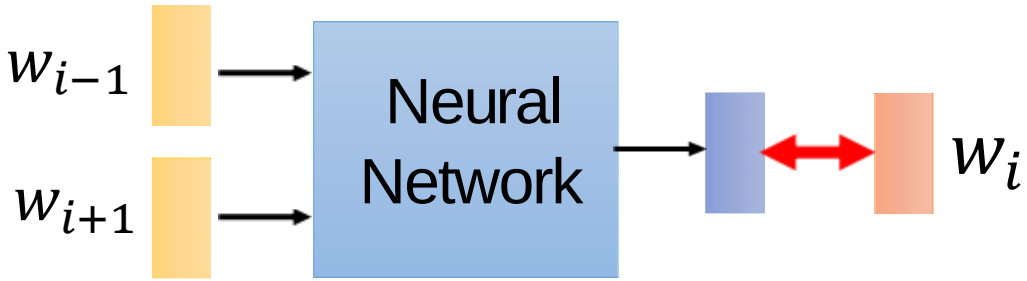
The weights with the same color should be the same.

Or, one word would have two word vectors.

Prediction-based

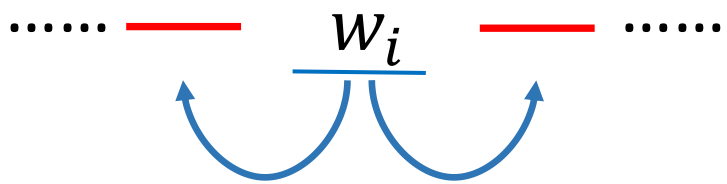
- Various Architectures
  - Continuous bag of word (CBOW) model

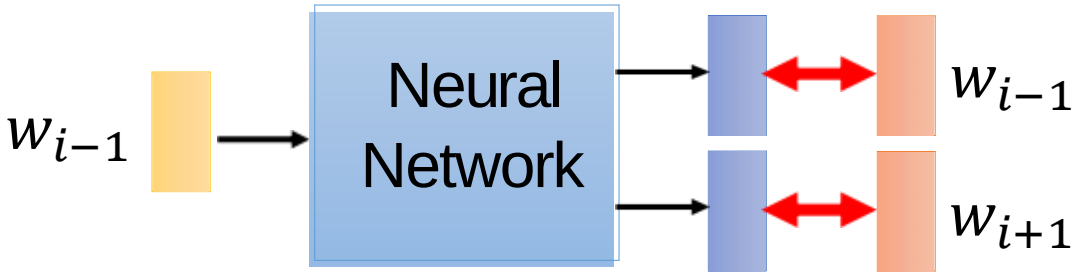




predicting the word given its context

  - Skip-gram





predicting the context given a word

问题导入：Slot Filling

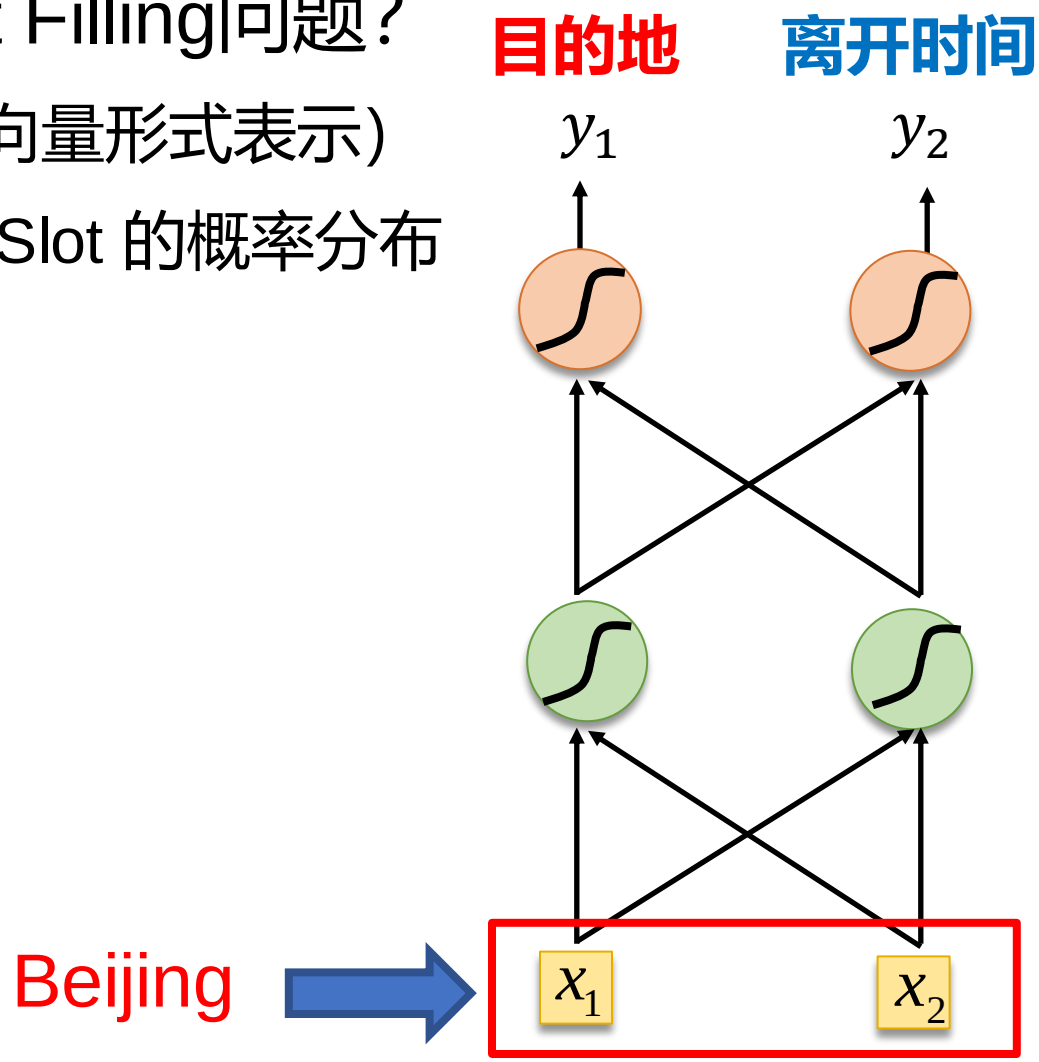
- 自然语言处理中的slot filling问题

内容导览

- 词向量嵌入
- 循环神经网络(RNN)
- Seq2Seq问题与注意力机制
- 自注意力/多头注意力/交叉注意力
- Transformer
- ELMO/BERT/GPT

问题导入：Slot Filling

- MLP 如何解决 Slot Filling问题?
  - 输入：一个词（以向量形式表示）
  - 输出：输入词属于 Slot 的概率分布



Slot Filling

arrive Beijing on November 2nd

other 目的地 other 时间 时间

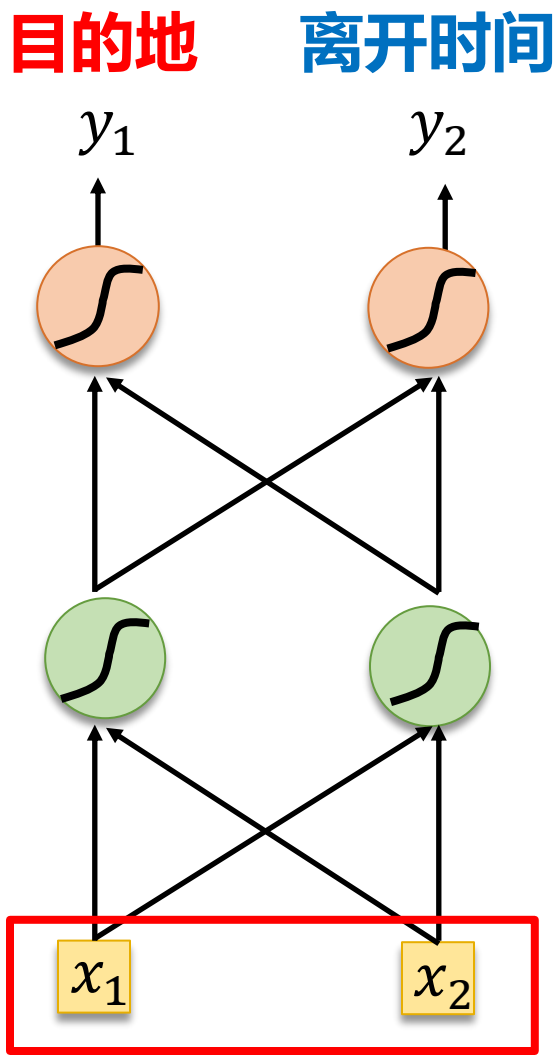
不同语境下词义是一致的吗

leave Beijing on November 2nd

出发地

神经网络需要记忆

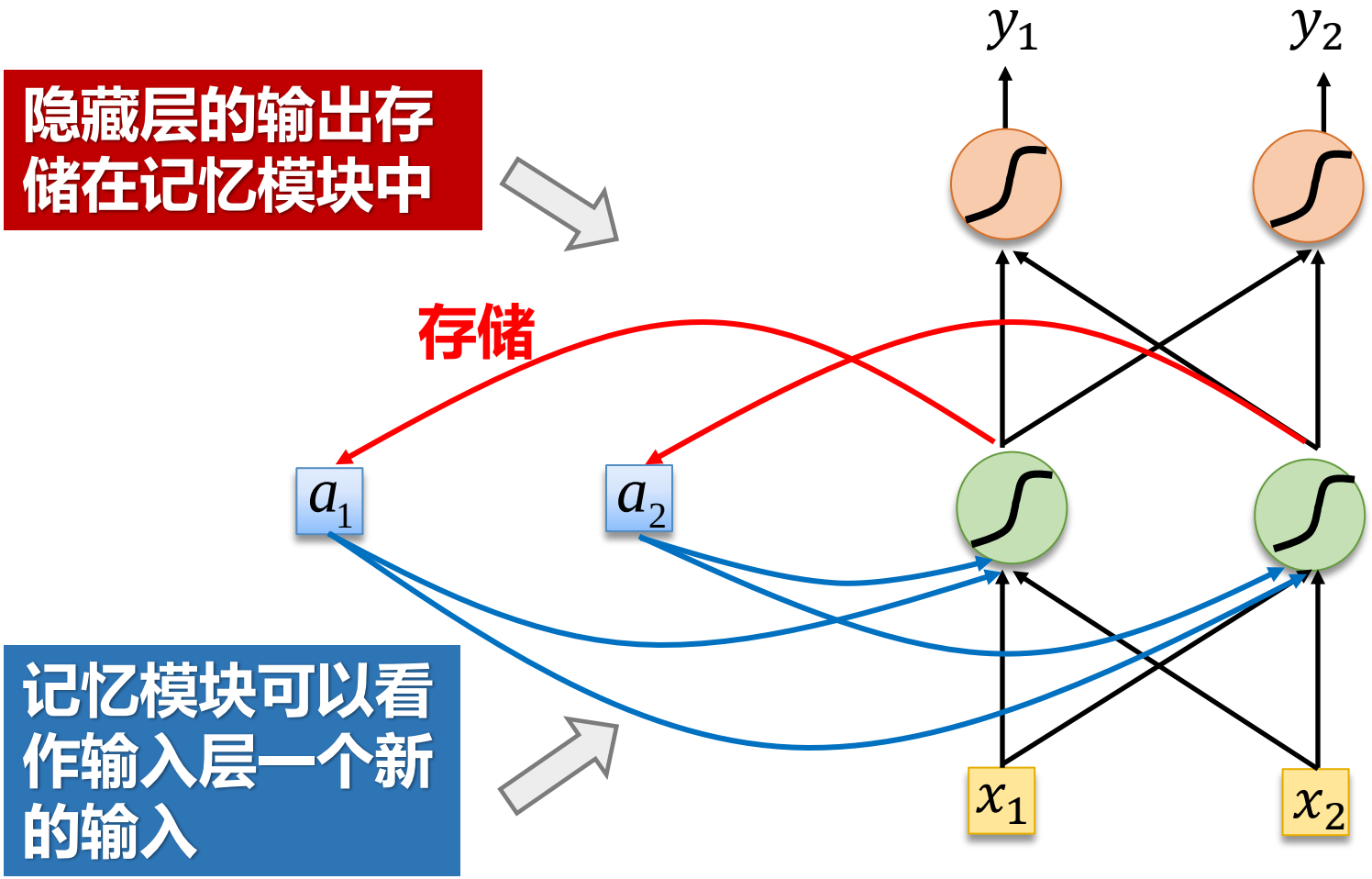
Beijing



循环神经网络(RNN)

口基本结构

- 每一次的输入都包含前面输入一定程度上的信息



循环神经网络(RNN)

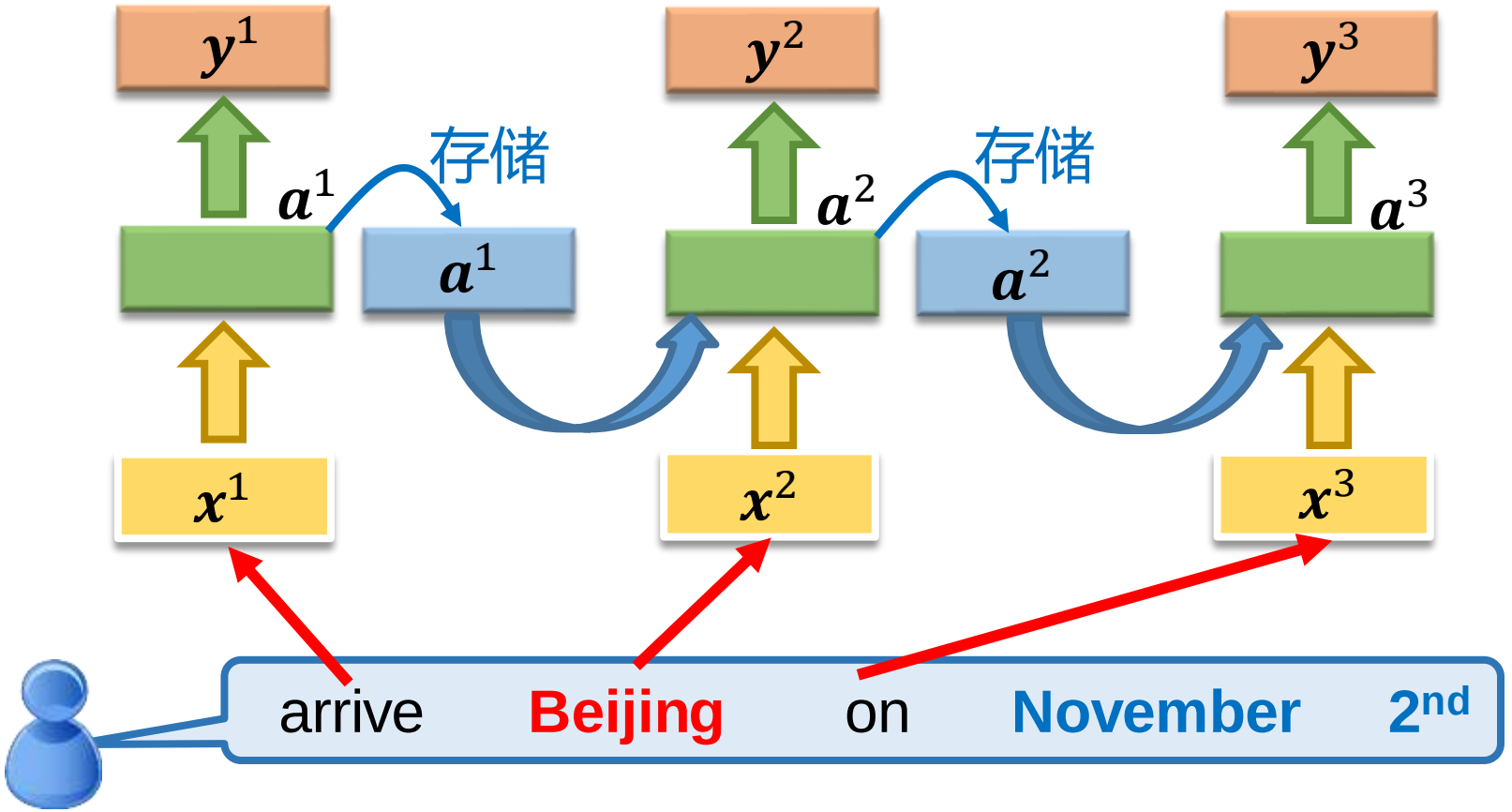
相同的网络一次又一次被使用

回到Slot问题：

每个slot中“arrive”的概率

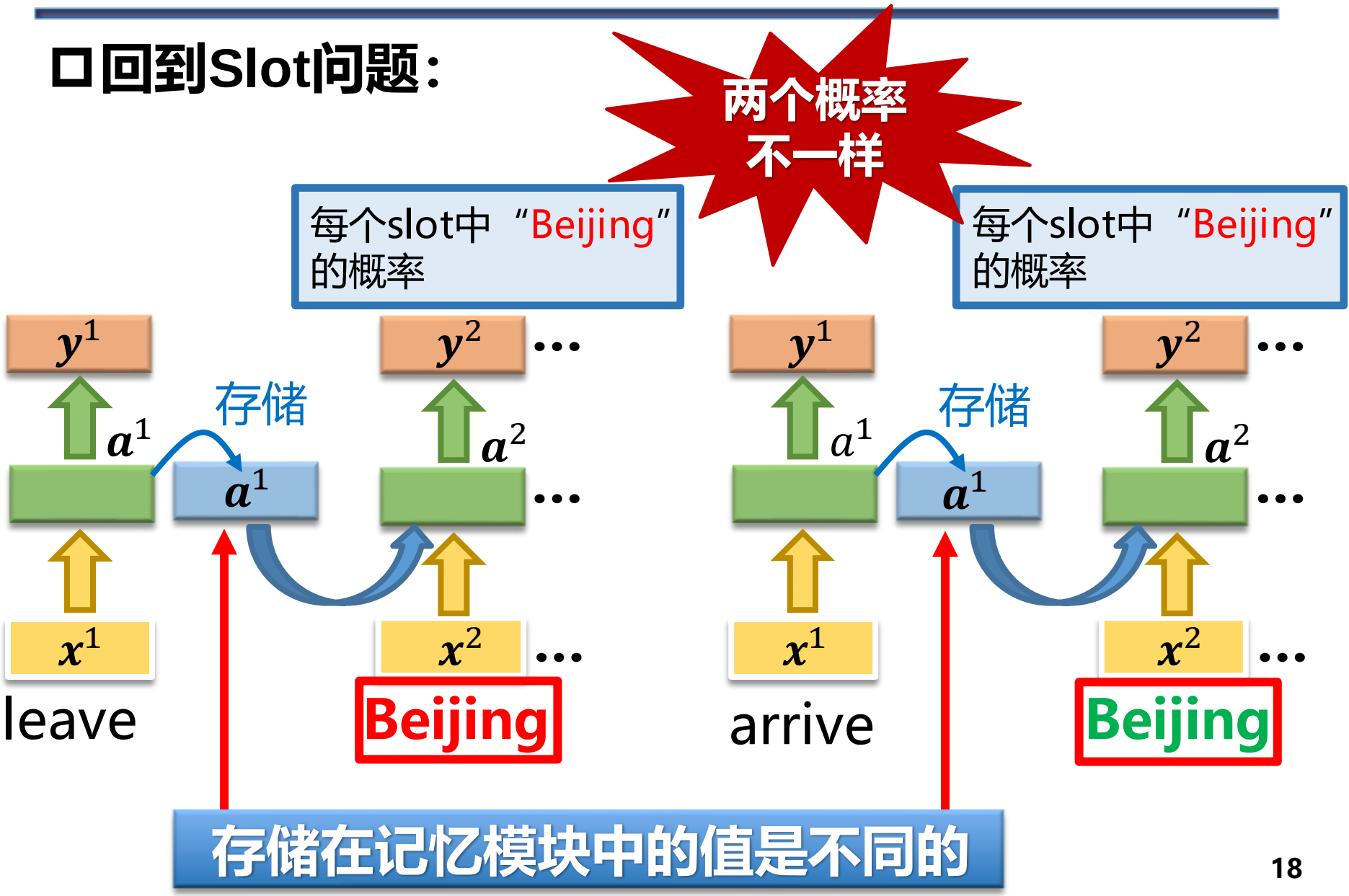
每个slot中“Beijing”的概率

每个slot中“on”的概率



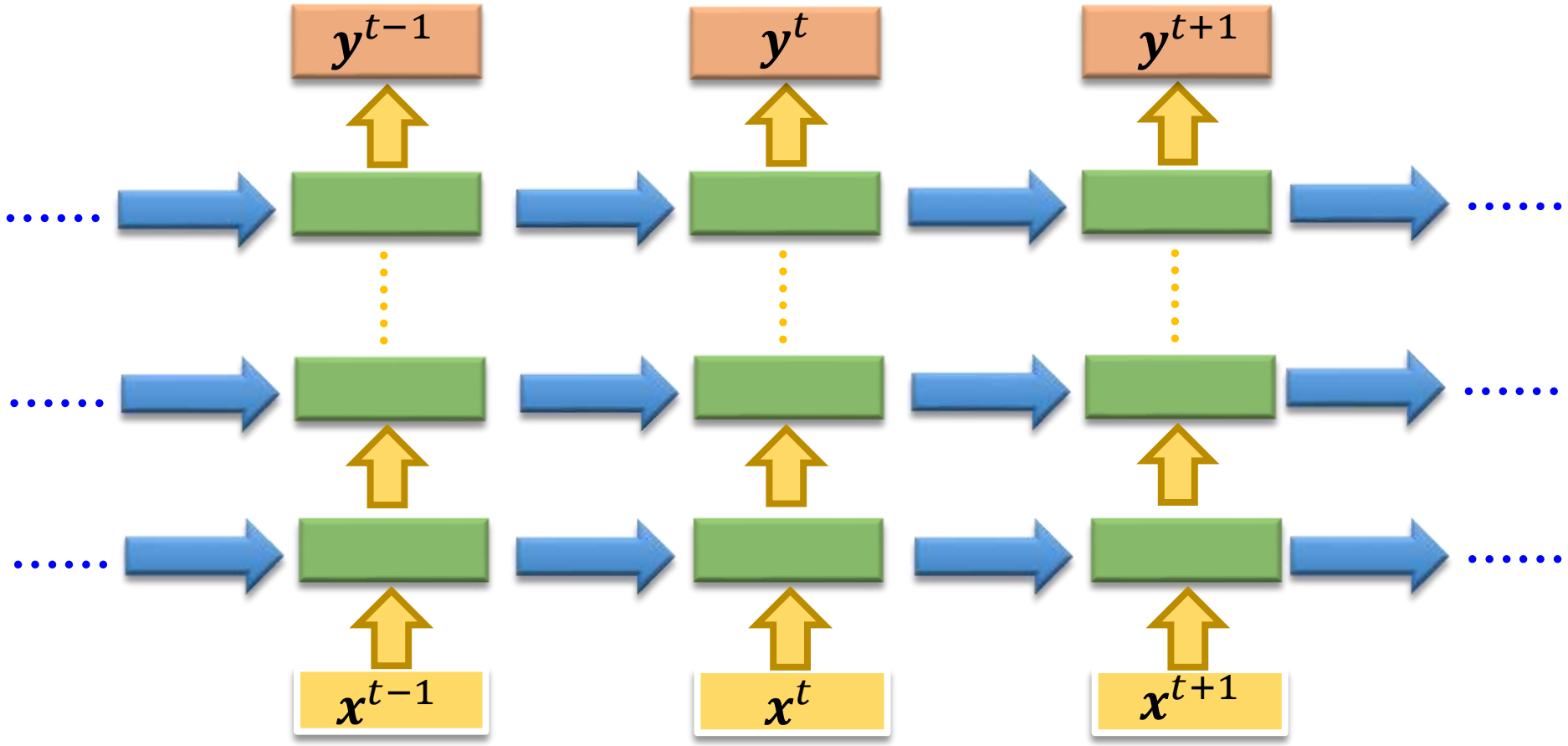
循环神经网络(RNN)

- 回到Slot问题：



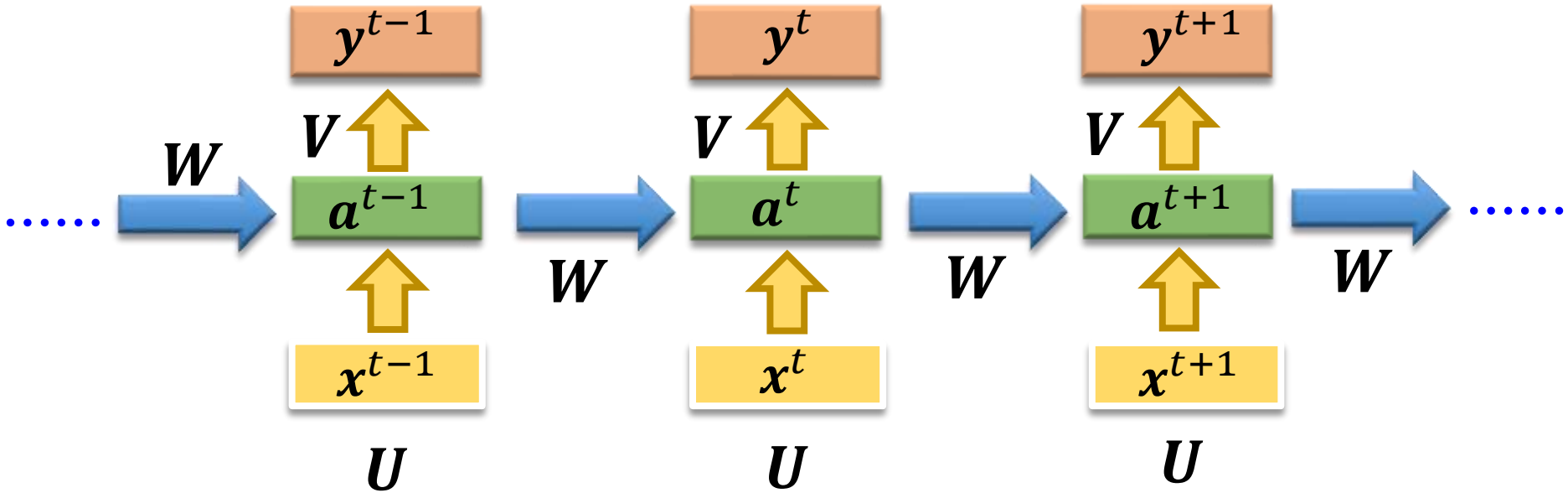
循环神经网络(RNN)

- 深度RNN



循环神经网络(RNN)

□数学表达



x：某时刻的输入(输入层)

a：隐藏状态(隐藏层)

y：某时刻的输出(输出层)

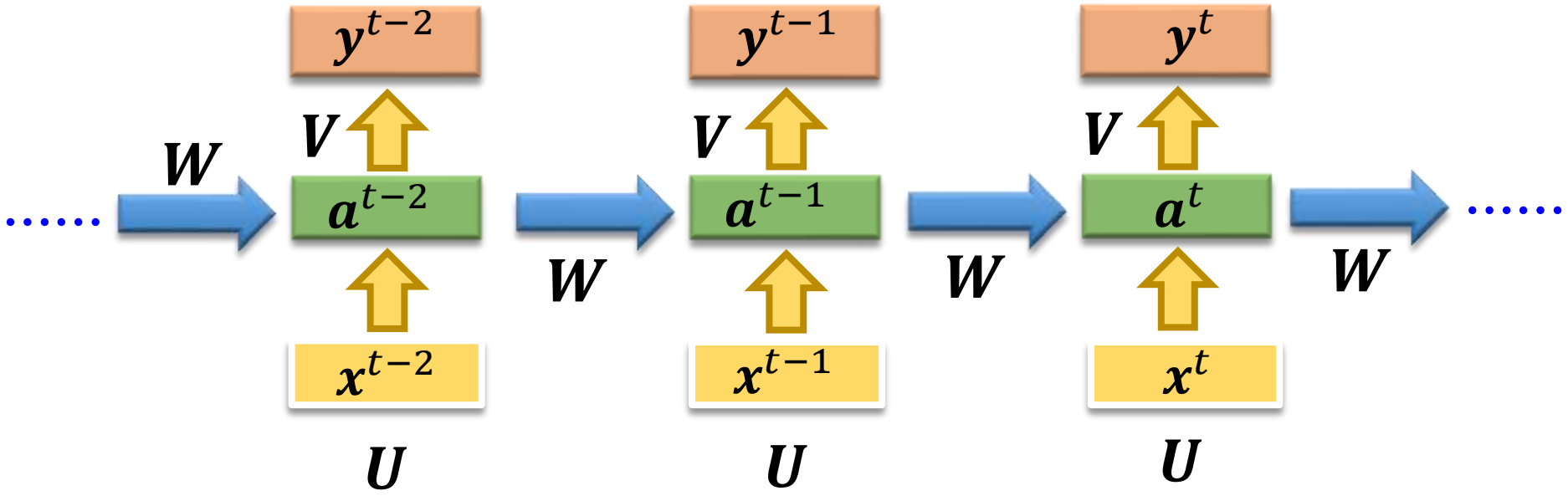
U：输入层到隐藏层的权重矩阵

V：隐藏层到输出层的权重矩阵

W：隐藏状态间的权重矩阵

循环神经网络(RNN)

RNN为什么有记忆?



$$y^t = Va^t$$

$$a^t = tanh(Ux^t + Wa^{t-1})$$

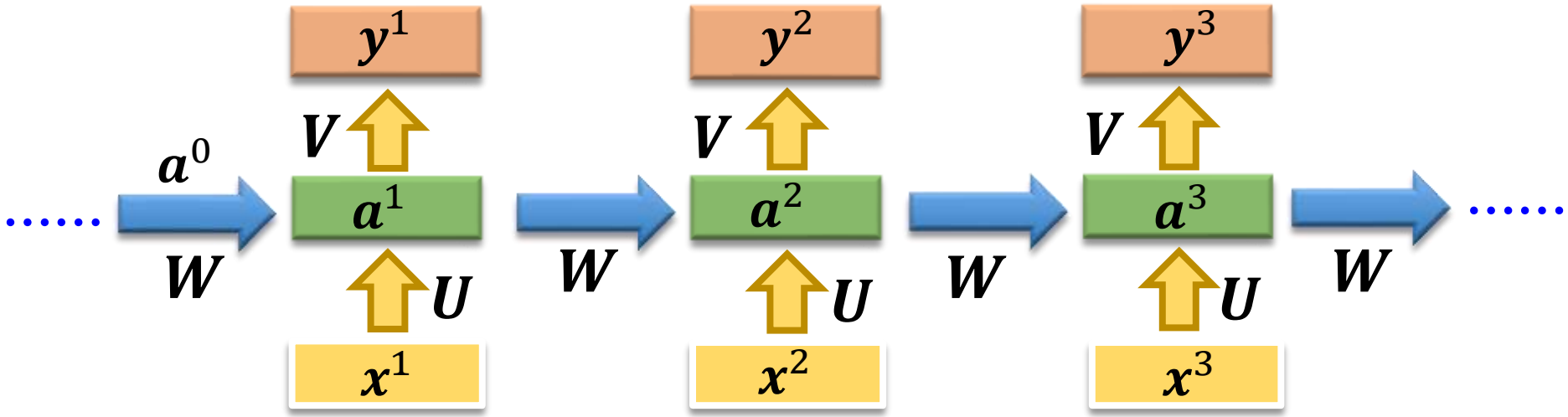
$$= tanh(Ux^t + W(tanh(Ux^{t-1} + Wa^{t-2}))$$

$$= tanh(Ux^t + W(tanh(Ux^{t-1} + W(tanh(Ux^{t-2} + Wa^{t-3}))$$

$$= \ldots$$

循环神经网络(RNN)

- 前向传播(一个简单的例子，没有激活函数)



$$a^1 = Ux_1 + Wa^0 + b_1 \qquad a^2 = Ux_2 + Wa^1 + b_1 \qquad a^3 = Ux_3 + Wa^1 + b_1$$

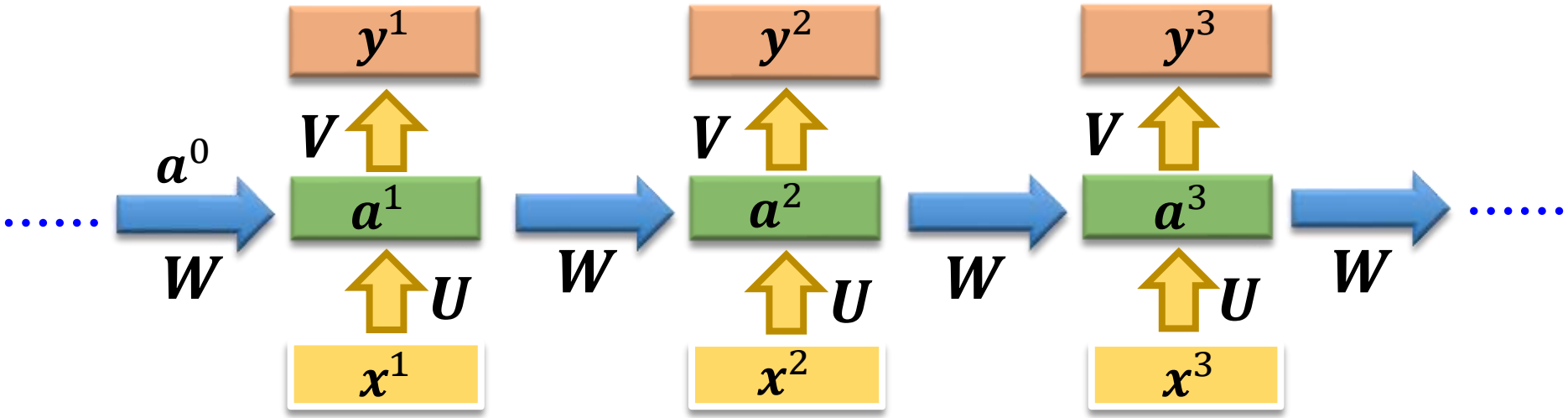
$$y^1 = Va^1 + b_2 \qquad y^2 = Va^2 + b_2 \qquad y^3 = Va^3 + b_2$$

损失函数 $L_3 = \frac{1}{2}(y^3 - x^3)^2 \qquad t = 3$

$$L = \sum_{t=0}^{T} L_t$$

循环神经网络(RNN)

- 反向传播(一个简单的例子，没有激活函数)



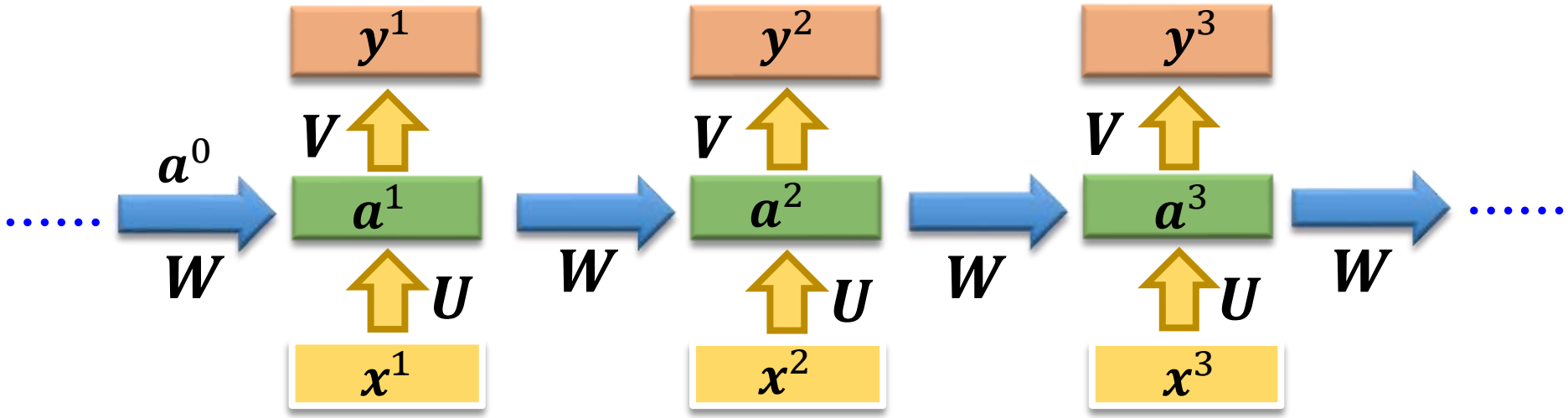
$$\frac{\mathrm{d}L_3}{\mathrm{d}V} = \frac{\mathrm{d}L_3}{\mathrm{d}y^3}\frac{\mathrm{d}y^3}{\mathrm{d}V}$$

$$\frac{\mathrm{d}L_3}{\mathrm{d}U} = \frac{\mathrm{d}L_3}{\mathrm{d}y^3}\frac{\mathrm{d}y^3}{\mathrm{d}a^3}\frac{\mathrm{d}a^3}{\mathrm{d}U} + \frac{\mathrm{d}L_3}{\mathrm{d}y^3}\frac{\mathrm{d}y^3}{\mathrm{d}a^3}\frac{\mathrm{d}a^3}{\mathrm{d}a^2}\frac{\mathrm{d}a^2}{\mathrm{d}U} + \frac{\mathrm{d}L_3}{\mathrm{d}y^3}\frac{\mathrm{d}y^3}{\mathrm{d}a^3}\frac{\mathrm{d}a^3}{\mathrm{d}a^2}\frac{\mathrm{d}a^2}{\mathrm{d}a^1}\frac{\mathrm{d}a^1}{\mathrm{d}U}$$

$$\frac{\mathrm{d}L_3}{\mathrm{d}W} = \frac{\mathrm{d}L_3}{\mathrm{d}y^3}\frac{\mathrm{d}y^3}{\mathrm{d}a^3}\frac{\mathrm{d}a^3}{\mathrm{d}W} + \frac{\mathrm{d}L_3}{\mathrm{d}y^3}\frac{\mathrm{d}y^3}{\mathrm{d}a^3}\frac{\mathrm{d}a^3}{\mathrm{d}a^2}\frac{\mathrm{d}a^2}{\mathrm{d}W} + \frac{\mathrm{d}L_3}{\mathrm{d}y^3}\frac{\mathrm{d}y^3}{\mathrm{d}a^3}\frac{\mathrm{d}a^3}{\mathrm{d}a^2}\frac{\mathrm{d}a^2}{\mathrm{d}a^1}\frac{\mathrm{d}a^1}{\mathrm{d}W}$$

循环神经网络(RNN)

- 反向传播(一个简单的例子)



扩展到任意时刻 t

$$\frac{\mathrm{d}L_t}{\mathrm{d}\boldsymbol{U}} = \sum_{k=1}^{t} \frac{\mathrm{d}L_t}{\mathrm{d}\boldsymbol{y}^t} \frac{\mathrm{d}\boldsymbol{y}^t}{\mathrm{d}\boldsymbol{a}^t} \left(\prod_{j=k+1}^{t} \frac{\mathrm{d}\boldsymbol{a}^j}{\mathrm{d}\boldsymbol{a}^{j-1}}\right) \frac{\mathrm{d}\boldsymbol{a}^k}{\mathrm{d}\boldsymbol{U}}$$

$$\frac{\mathrm{d}L_t}{\mathrm{d}\boldsymbol{W}} = \sum_{k=1}^{t} \frac{\mathrm{d}L_t}{\mathrm{d}\boldsymbol{y}^t} \frac{\mathrm{d}\boldsymbol{y}^t}{\mathrm{d}\boldsymbol{a}^t} \left(\prod_{j=k+1}^{t} \frac{\mathrm{d}\boldsymbol{a}^j}{\mathrm{d}\boldsymbol{a}^{j-1}}\right) \frac{\mathrm{d}\boldsymbol{a}^k}{\mathrm{d}\boldsymbol{W}}$$

加上激活函数

$$\boldsymbol{a}^j = \tanh(\boldsymbol{U}\boldsymbol{a}^j + \boldsymbol{W}\boldsymbol{a}^{j-1} + \boldsymbol{b}_1)$$

$$\prod_{j=k+1}^{t} \frac{\mathrm{d}\boldsymbol{a}^j}{\mathrm{d}\boldsymbol{a}^{j-1}} = \prod_{j=k+1}^{t} \tanh' \boldsymbol{W}$$

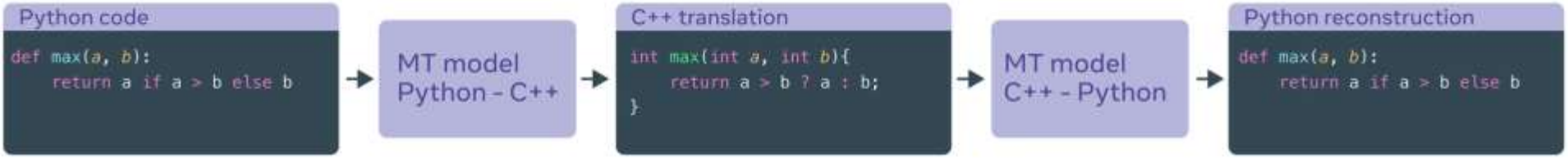
内容导览

- 词向量嵌入
- 循环神经网络(RNN)
- **Seq2Seq问题与注意力机制**
- 自注意力/多头注意力/交叉注意力
- Transformer
- ELMO/BERT/GPT

Seq2Seq问题

- The most popular sequence-to-sequence task is ***translation***. From now on, by machine translation we will mean any general sequence-to-sequence task, i.e. **translation between sequences of tokens of any nature**.

- Except for the popular machine translation between natural languages, you can **translate between programming languages**



https://ai.meta.com/blog/deep-learning-to-translate-between-programming-languages/

Seq2Seq的基本概念

Seq2Seq的基本概念

we have an input sequence $x_1, x_2, \ldots, x_m$ and an output sequence $y_1, y_2, \ldots, y_n$ (note that their **lengths can be different**), translation can be thought of as **finding the target sequence that is the most probable given the input**; formally, the target sequence that **maximizes the conditional probability**

$$p(\boldsymbol{y}|\boldsymbol{x}): \boldsymbol{y}^* = \underset{\boldsymbol{y}}{\operatorname{argmax}}\, p(\boldsymbol{y}|\boldsymbol{x})$$

- If you are bilingual and can translate between languages easily, you have an intuitive feeling of $p(\boldsymbol{y}|\boldsymbol{x})$.
- But in machine translation, we learn a function $p(\boldsymbol{y}|\boldsymbol{x}, \theta)$ with some parameters θ, and then **find its argmax for a given input**: $\boldsymbol{y}' = \underset{\boldsymbol{y}}{\operatorname{argmax}}\, p(\boldsymbol{y}|\boldsymbol{x}, \theta)$

Seq2Seq的基本概念

☐ Three questions of machine translation

Human Translation

$$\boldsymbol{y}^* = \underset{\boldsymbol{y}}{\operatorname{argmax}}\, p(\boldsymbol{y}|\boldsymbol{x})$$

The "probability" is intuitive and is given by human translators' expertise

Machine Translation

$$\boldsymbol{y}' = \underset{\boldsymbol{y}}{\operatorname{argmax}}\, p(\boldsymbol{y}|\boldsymbol{x}, \theta)$$

model parameters

Questions we need to answer

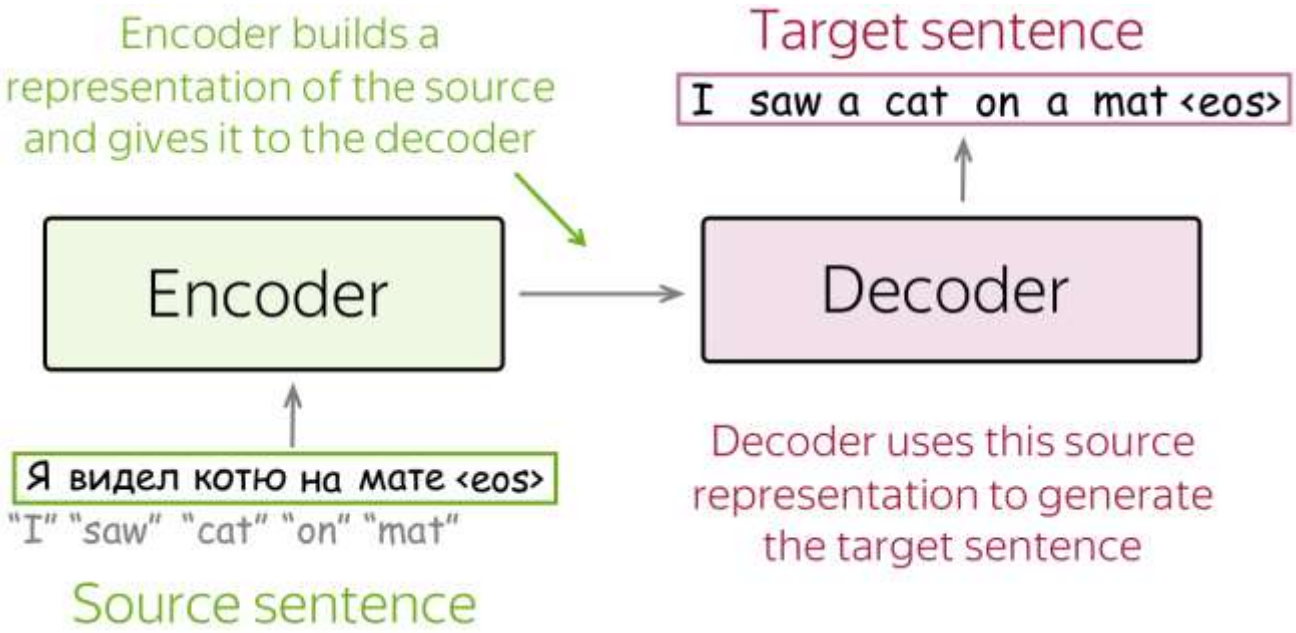
- modeling
- learning
- search

To define a machine translation system, we need to answer three questions:

- **modeling** - how does the model for $p(\boldsymbol{y}|\boldsymbol{x}, \theta)$ look like?
- **learning** - how to find the parameters θ?
- **inference** - how to find the best $\boldsymbol{y}$?

Encoder-Decoder Framework

- **Standard modeling paradigm for seq2seq tasks**



- **Encoder**: reads source sequence and produces its representation
- **Decoder**: uses source representation from the encoder to generate the target sequence

Conditional Language Models

☐ Seq2seq tasks can be modeled as CLM

While language models estimate the unconditional probability $p(\boldsymbol{y})$ of a sequence, **seq2seq models need to estimate the conditional probability** $p(\boldsymbol{y}|\boldsymbol{x})$ of a sequence $\boldsymbol{y}$ given a source $\boldsymbol{x}$.

Language Models

$$P(y_1, y_2, \dots, y_n) = \sum_{t=1}^{n} p(y_t | y_{<t})$$

Conditional Language Models

Condition on source $\boldsymbol{x}$

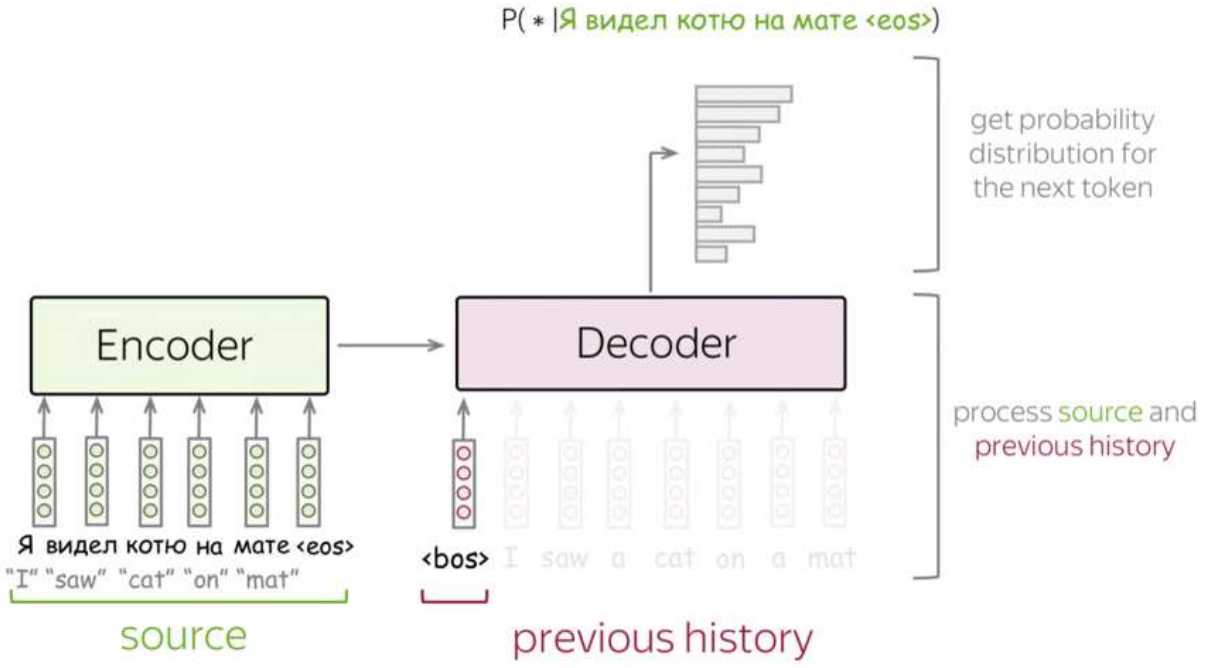
$$P(y_1, y_2, \dots, y_n | \boldsymbol{x}) = \sum_{t=1}^{n} p(y_t | y_{<t}, \boldsymbol{x})$$

Seq2seq tasks operate similarly to LMs, but additionally receive source information $\boldsymbol{x}$.

https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html#analysis_interpretability

Conditional Language Models

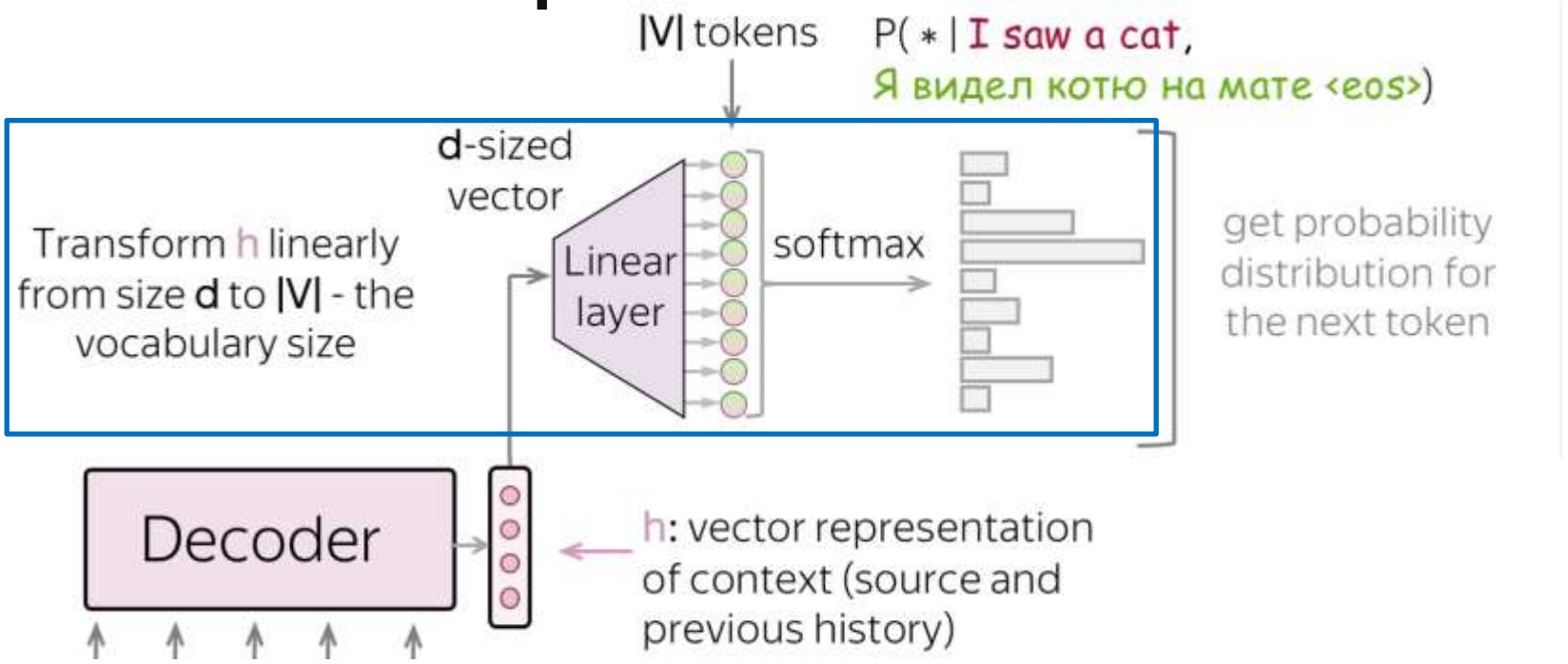
High-level pipeline



- Feed source and previously generated target words into a network
- Get vector representation of context (both source and previous target) from the networks decoder
- From this vector representation, predict a probability distribution for the next token

Conditional Language Models

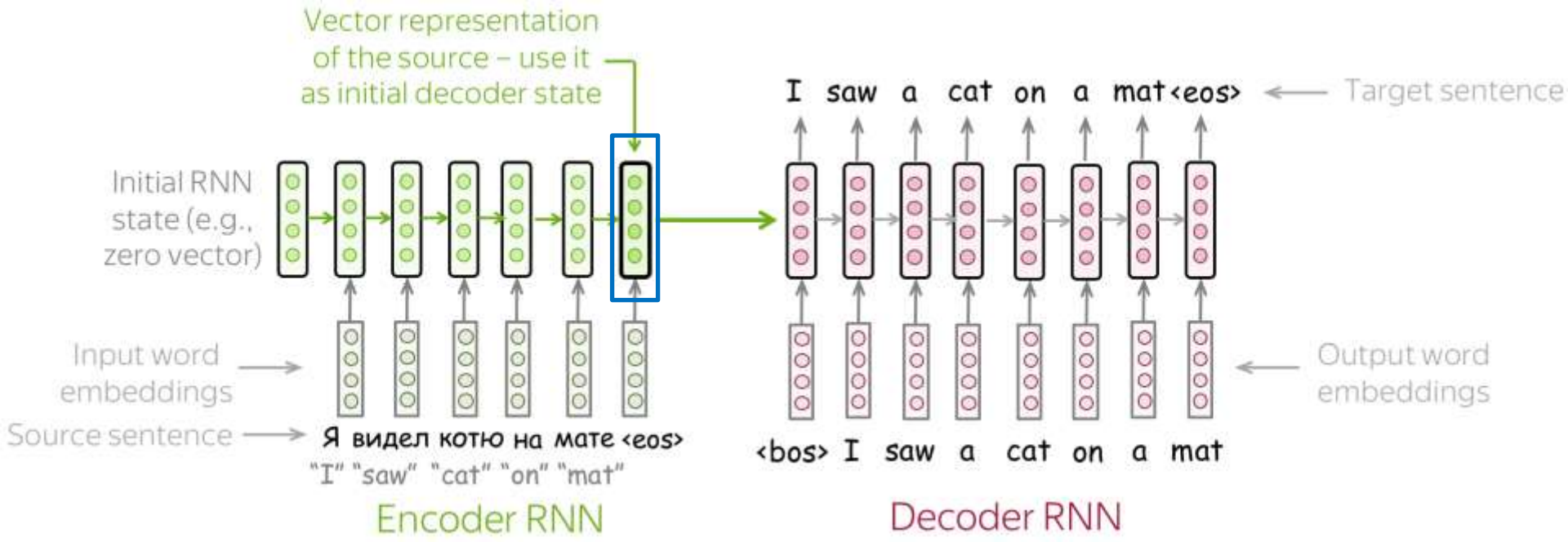
Classification part



- **Vector representation** of a text has some dimensionality d, but in the end, we need a vector of size $|V|$ (probabilities for $|V|$ tokens/classes).
- Once we have a $|V|$-sized vector, **all is left is to apply the softmax operation to convert the raw numbers into token probabilities**.

The Simplest Model

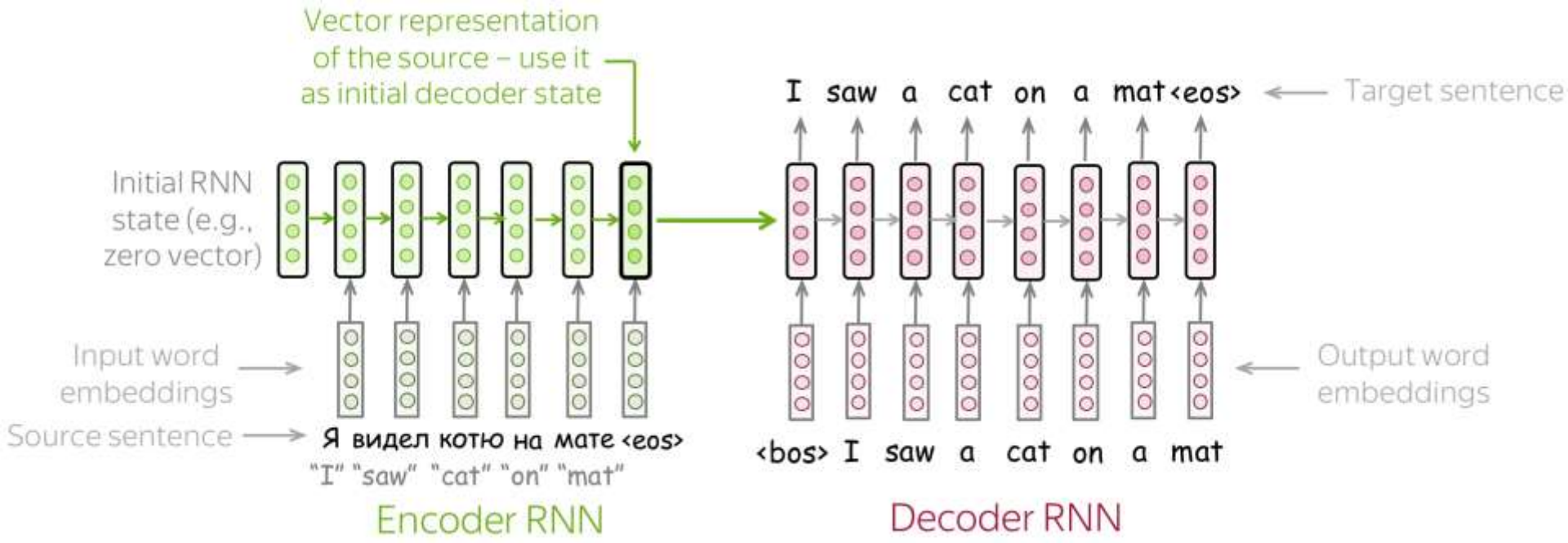
- **Two RNNs for Encoder and Decoder**



Encoder RNN reads the source sentence, and **the final state is used as the initial state of the decoder RNN**.

The Simplest Model

- **Two RNNs for Encoder and Decoder**



The hope is that the **final encoder state "encodes" all information about the source**, and the **decoder can generate the target sentence based on this vector**.

Model Training

The Cross-Entropy Loss

- Assume we have a training instance with the **source** $\boldsymbol{x} = (x_1, x_2, \ldots, x_m)$ and the **target** $\mathbf{y} = (y_1, y_2, \ldots, y_n)$.

- At the timestep t, a model predicts a **probability distribution** $p^{(t)} = p(* \,|y_1, \ldots, y_{t-1}, x_1, \ldots, x_m)$. The **target at this step** is $p^* = \text{one-hot}(y_t)$, i.e., we want a model to **assign probability 1 to the correct token**, y_t, and **zero to the rest**.

https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html#analysis_interpretability

Model Training

The Cross-Entropy Loss

- **Cross-entropy loss** for the target distribution p^* and the predicted distribution p

$$Loss(p^*, p) = -p^* \log(p) = -\sum_{i=1}^{|V|} p_i^* \log(p_i)$$

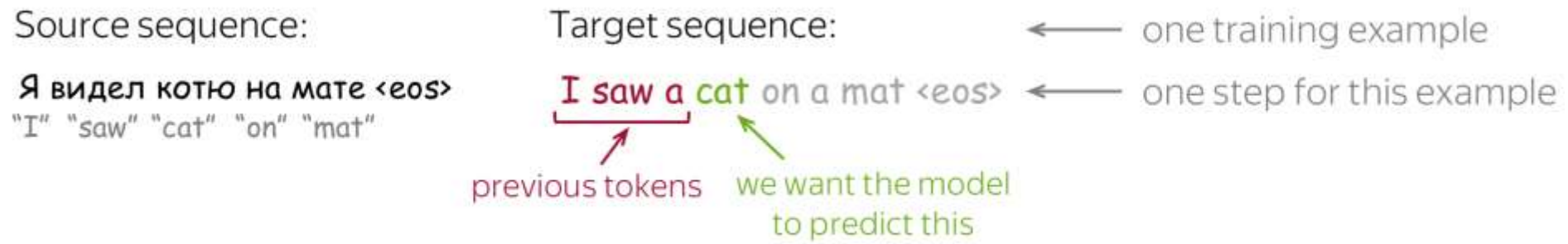
- Since **only one of p_i^* is non-zero** (for the correct token y_t), we will get

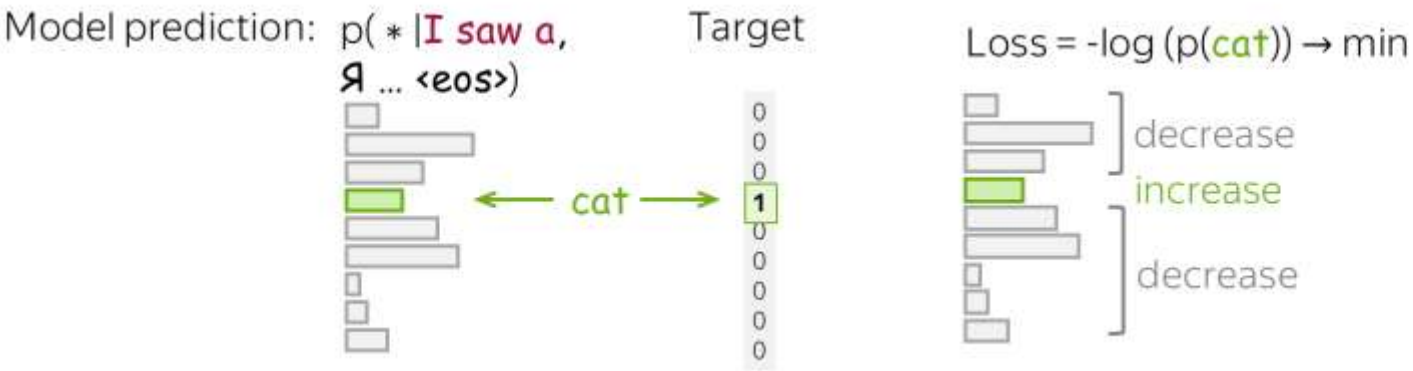
$$Loss(p^*, p) = -\log(y_t) = -\log(p(y_t | y_{<t}, \boldsymbol{x}))$$

Model Training

The Cross-Entropy Loss

- At each step, we **maximize the probability a model assigns to the correct token**.





Model Training

- The **Cross-Entropy Loss**
  - For the whole example, the loss will be

$$-\sum_{t=1}^{n} \log(p(y_t|y_{<t}, \boldsymbol{x}))$$

The illustration is for the RNN model, but the model can be different

Model Inference

Greedy Decoding and Beam Search

Let's think how to generate a translation using this model. We model the probability of a sentence as follows:

How to find the argmax?

$$\boldsymbol{y}' = \underset{\boldsymbol{y}}{\operatorname{argmax}}\, p(\boldsymbol{y}|\boldsymbol{x}) = \underset{\boldsymbol{y}}{\operatorname{argmax}} \prod_{t=1}^{n} p(y_t|y_{<t}, \boldsymbol{x})$$

- Note that **we can not find the exact solution**
  - The **total number of hypotheses** we need to check is $|V|^n$, which is not feasible in practice.
  - Therefore, we will **find an approximate solution**.

Greedy Decoding

At each step, pick the most probable token

- At each step, **generate a token with the highest probability**.
- This can be **a good baseline**, but this method is **inherently flawed**: the best token at the current step does not necessarily lead to the best sequence.

$$\underset{\boldsymbol{y}}{\operatorname{argmax}} \prod_{t=1}^{n} p(y_t | y_{<t}, \boldsymbol{x}) \neq \underset{y_t}{\operatorname{argmax}} \prod_{t=1}^{n} p(y_t | y_{<t}, \boldsymbol{x})$$

Beam Search

Keep track of several most probably hypotheses

- At each step, we will be continuing each of the current hypotheses and pick top-N of them.

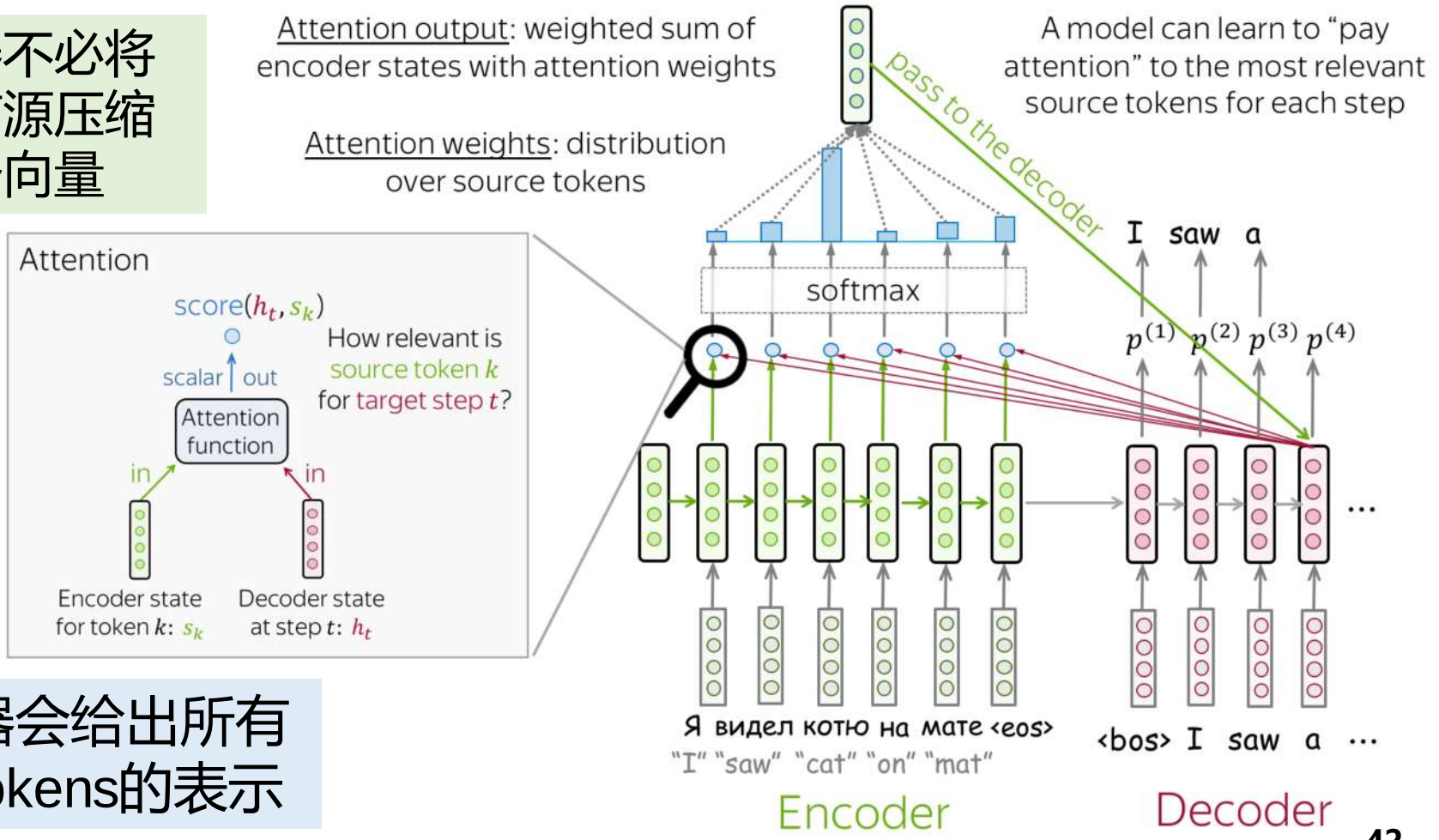
Start with the begin of sentence token or with an empty sequence

Usually, the **beam size is 4-10**. Increasing beam size is computationally inefficient and, what is more important, leads to worse quality.

什么是Attention?

- 注意力机制是神经网络的一部分。 在每个解码器步骤中，它决定哪些信源部分更重要。

编码器不必将整个信源压缩为单个向量

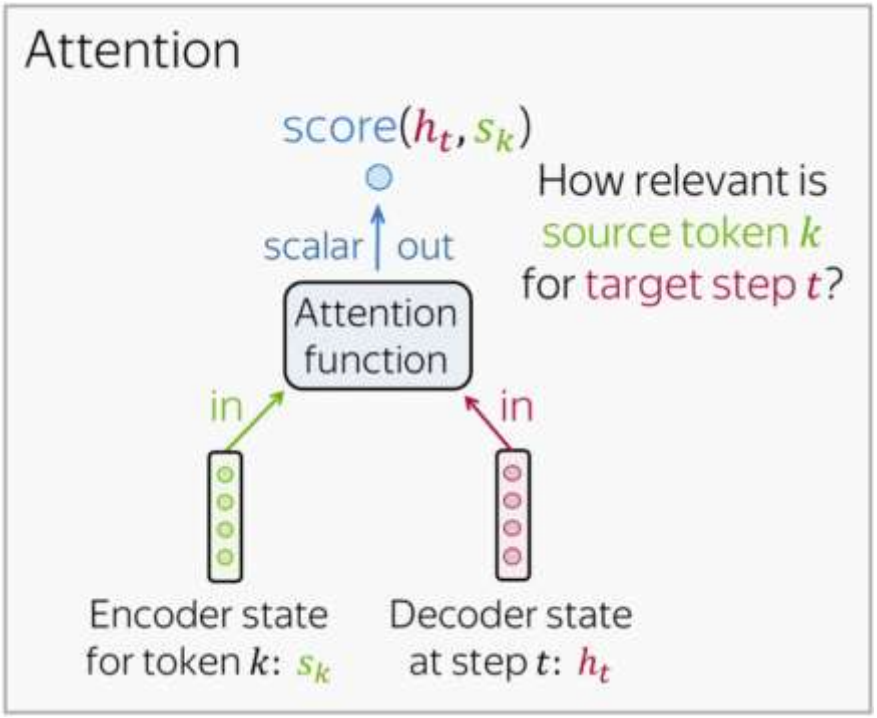


编码器会给出所有信源tokens的表示

什么是Attention?

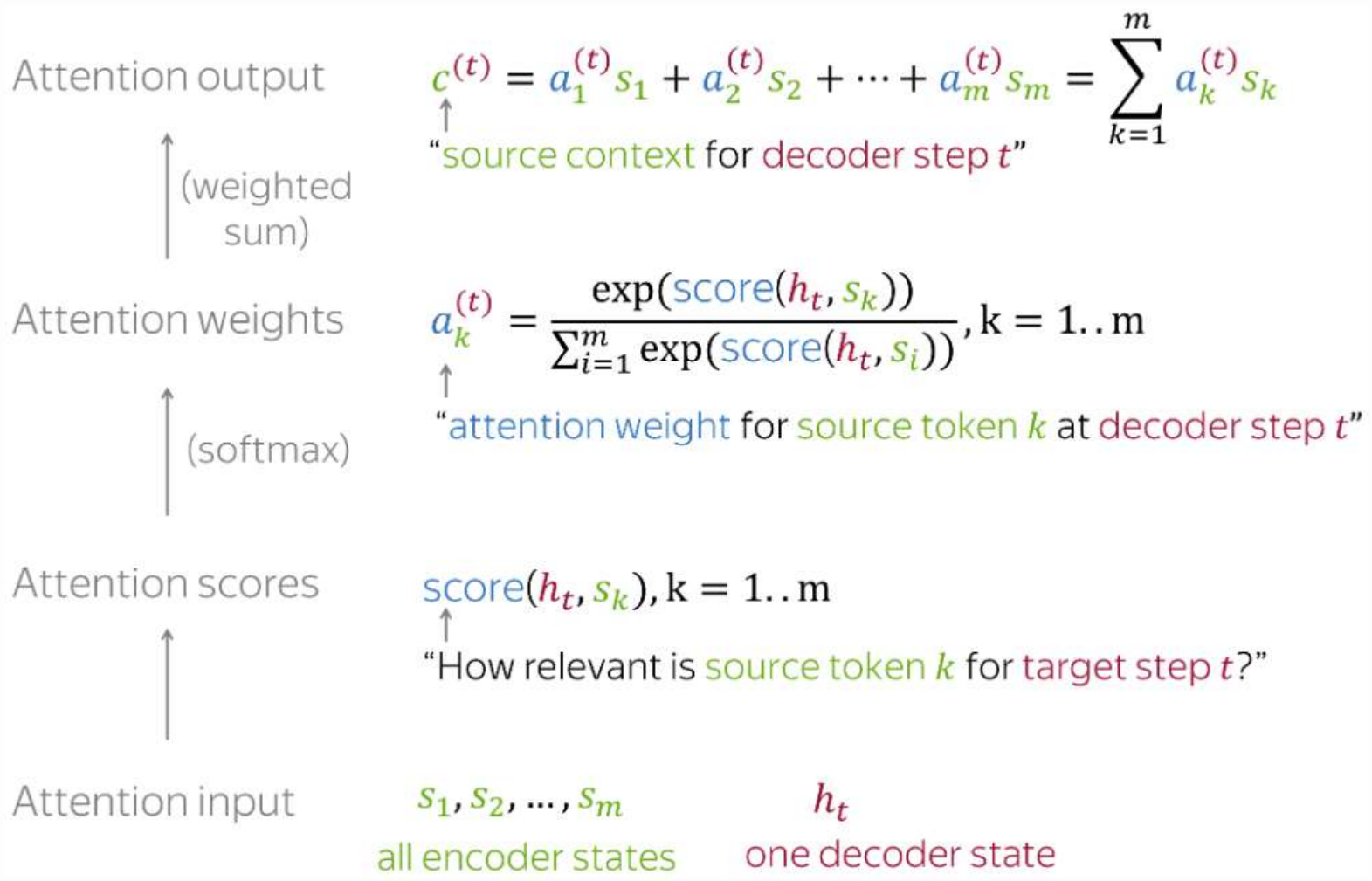
- 在解码器的每一步中，注意力机制会

- 收到**注意力输入**：一个解码器状态h_t和所有编码器状态$s_1, s_2, \ldots, s_m$
- 计算**注意力得分**：对于每一个编码器状态s_k，注意力机制计算它和解码器状态h_t的**相关性**。
- 计算**注意力权重**：一个**概率分布**，即在注意力得分上运用**softmax**。
- 计算**注意力输出**：带有注意力权重的编码器状态的加权和



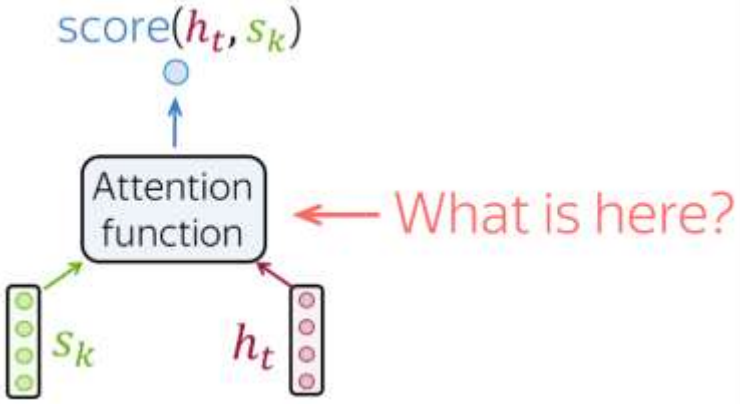
什么是Attention?

- 整体计算流程

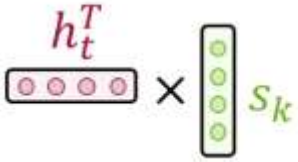


如何计算注意力得分(Attention Score)

- 一些经典并简单的方法
  - 本质上来说可以采用任何函数，即使它很复杂



Dot-product



$\text{score}(h_t, s_k) = h_t^T s_k$

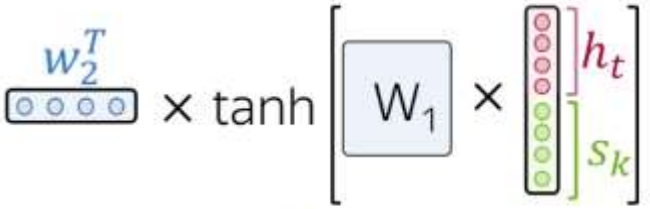
最简单的方法

Bilinear

$\text{score}(h_t, s_k) = h_t^T W s_k$

又名"Luong attention"

Multi-Layer Perceptron

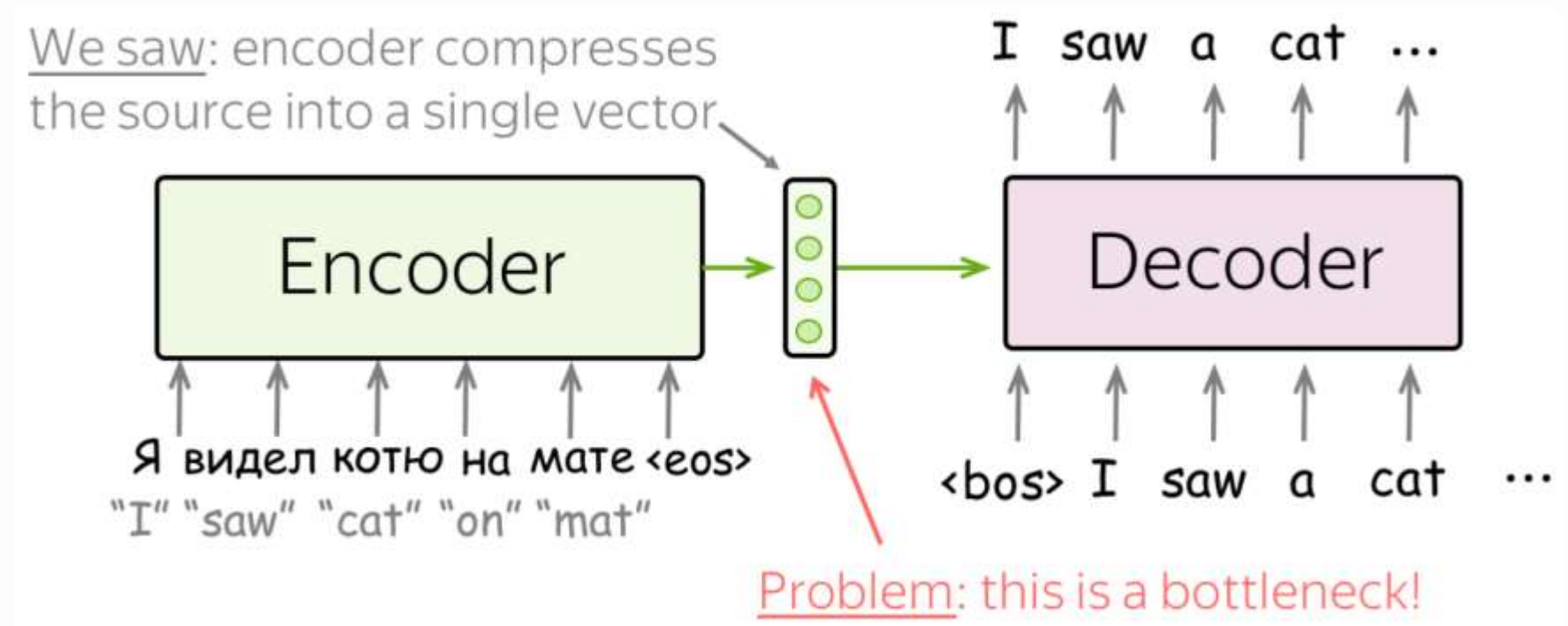


$\text{score}(h_t, s_k) = w_2^T \cdot \tanh(W_1[h_t, s_k])$

又名"Bahdanau attention"

为什么需要Attention机制

- **固定编码器表示**(Fixed Encoder Representation)的缺点
  - 对于编码器，很难压缩句子
  - 对于解码器，在解码的不同阶段，不同的信息也许是关联的



为什么需要Attention机制

- Attention机制使得模型**拥有极长的长期记忆**，可以“attend”或“focus”之前生成的所有token。
- 与RNN的对比
  - 循环神经网络(RNN)也能够查看之前的输入。 但**RNN的参考窗口较短**，因此当输入变长时，RNN无法访问序列中较早生成的词。
  - 对于门控循环单元 (GRU) 和长短期记忆 (LSTM) 网络来说，虽然它们有更长的RNN参考窗口，但从理论上讲，**注意力机制在给予足够的计算资源的情况下，有无限的参考窗口**，因此能够在生成文本时使用输入的整个上下文。

内容导览

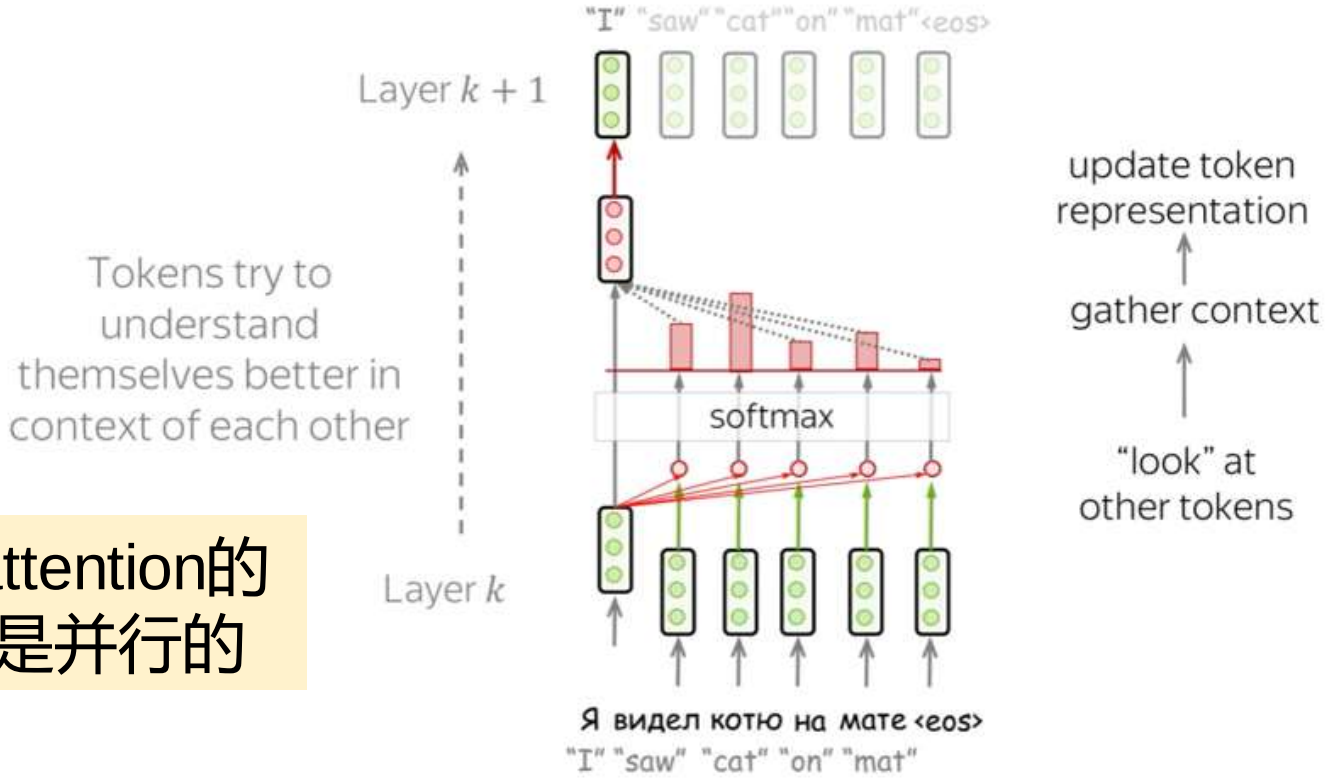
- 序列到序列(Seq2Seq)
- 循环神经网络(RNN)
- 注意力机制的基本原理
- **自注意力/多头注意力/交叉注意力**
- Transformer
- ELMO/BERT/GPT

Self-Attention

- Self-Attention与Attention的区别
  - Self-attention在**相同性质的表征**之间运行，例如，某个层中的所有编码器状态
- **Decoder-encoder** attention is looking
  - **from**: 当前解码器状态
  - **at:** 所有的编码器状态
- **Self-attention** is looking
  - **from**: 一组状态中的一个状态
  - **at**: 相同组中的其他状态

Self-Attention

- **Self-attention is the part of the model where tokens interact with each other**
  - Each token "looks" at other tokens in the sentence with an attention mechanism, **gathers context**, and **updates the previous representation of "self".**



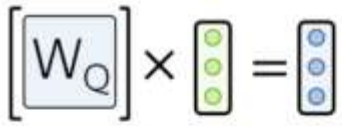
Self-attention的计算是并行的

Self-Attention的Query/Key/Value

- Formally, this intuition is implemented with a ***query-key-value attention***. Each input token in self-attention **receives three representations** corresponding to the roles it can play
  - **query - asking for information;**
  - **key - saying that it has some information;**
  - **value - giving the information.**
- The **query** is used when a token looks at others - *it's seeking the information to understand itself better*. The **key** is responding to a query's request: *it is used to compute attention weights*.
  The **value** is used to compute attention output: *it gives information to the tokens which "say" they need it* (i.e. assigned large weights to this token).

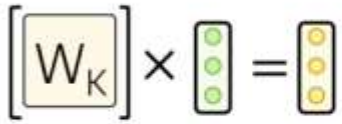
Self-Attention的Query/Key/Value

- Each vector receives three representations (“roles”)



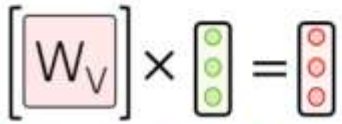
Query: vector **from** which the attention is looking

"Hey there, do you have this information?"



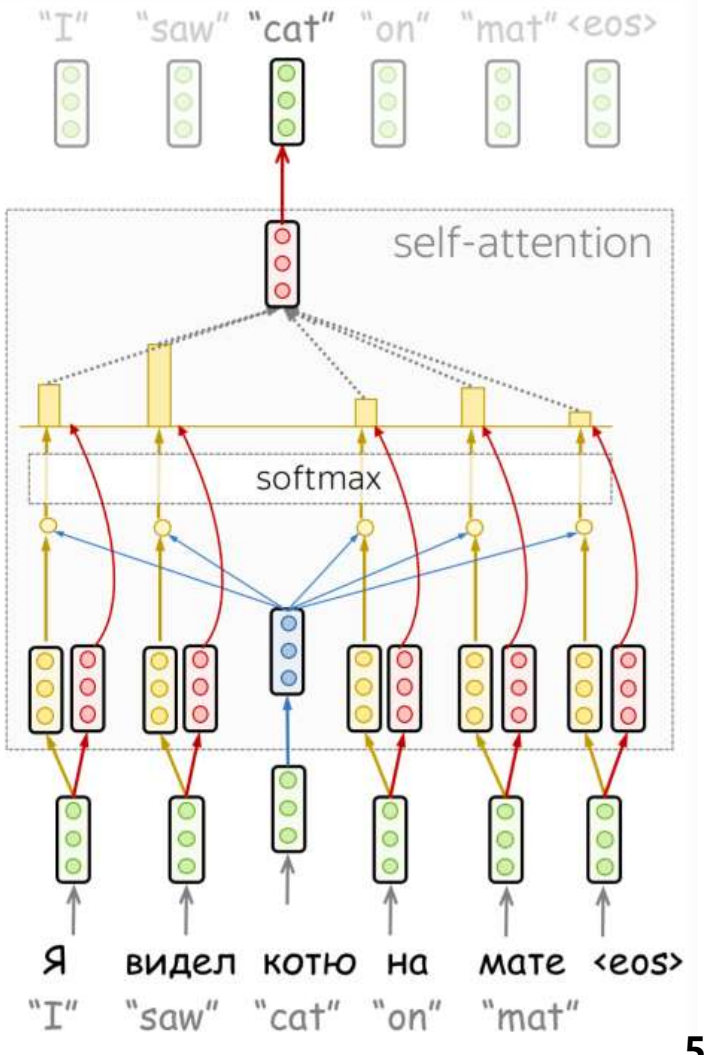
Key: vector **at** which the query looks to compute weights

"Hi, I have this information – give me a large weight!"

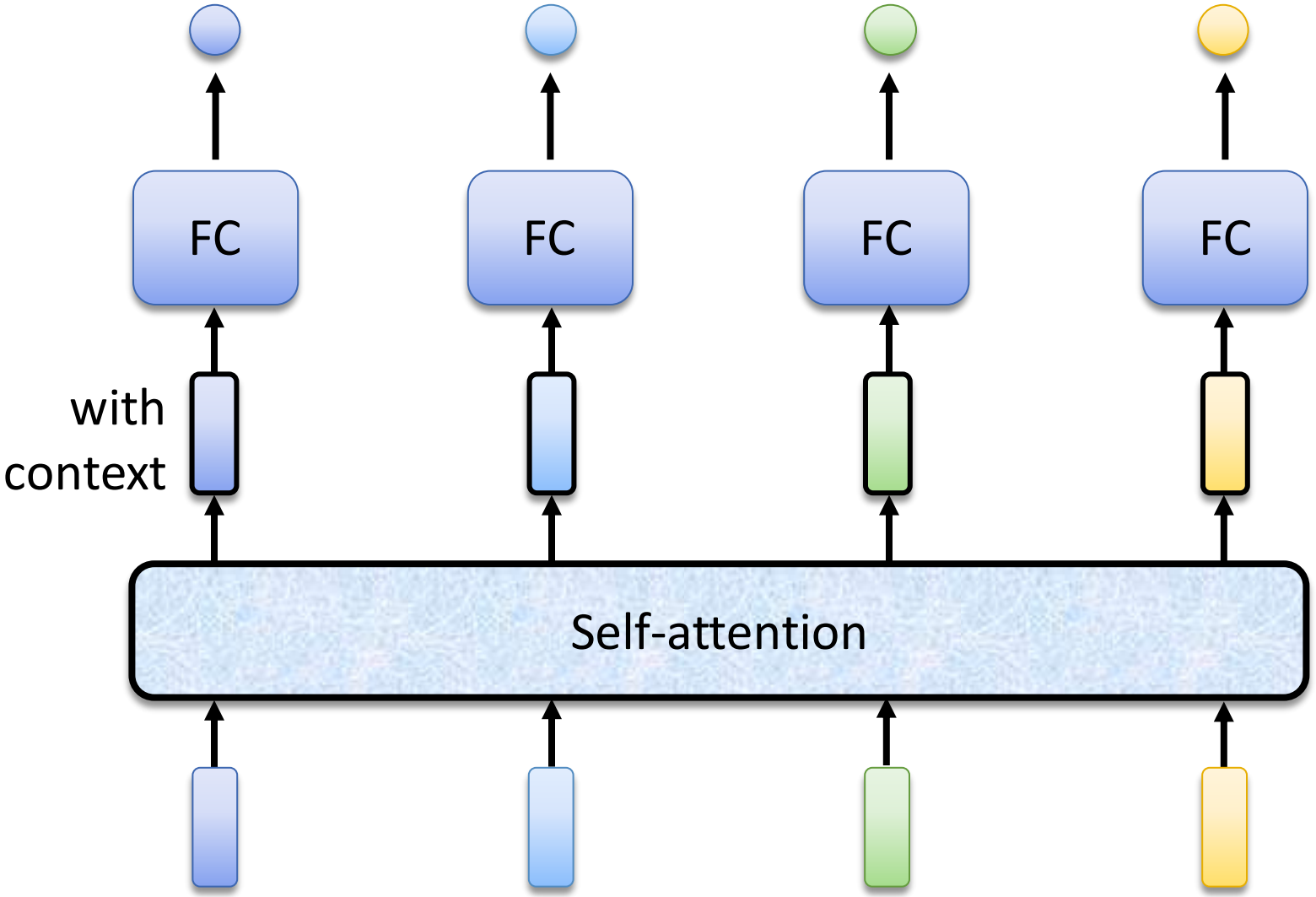


Value: their weighted sum is attention output

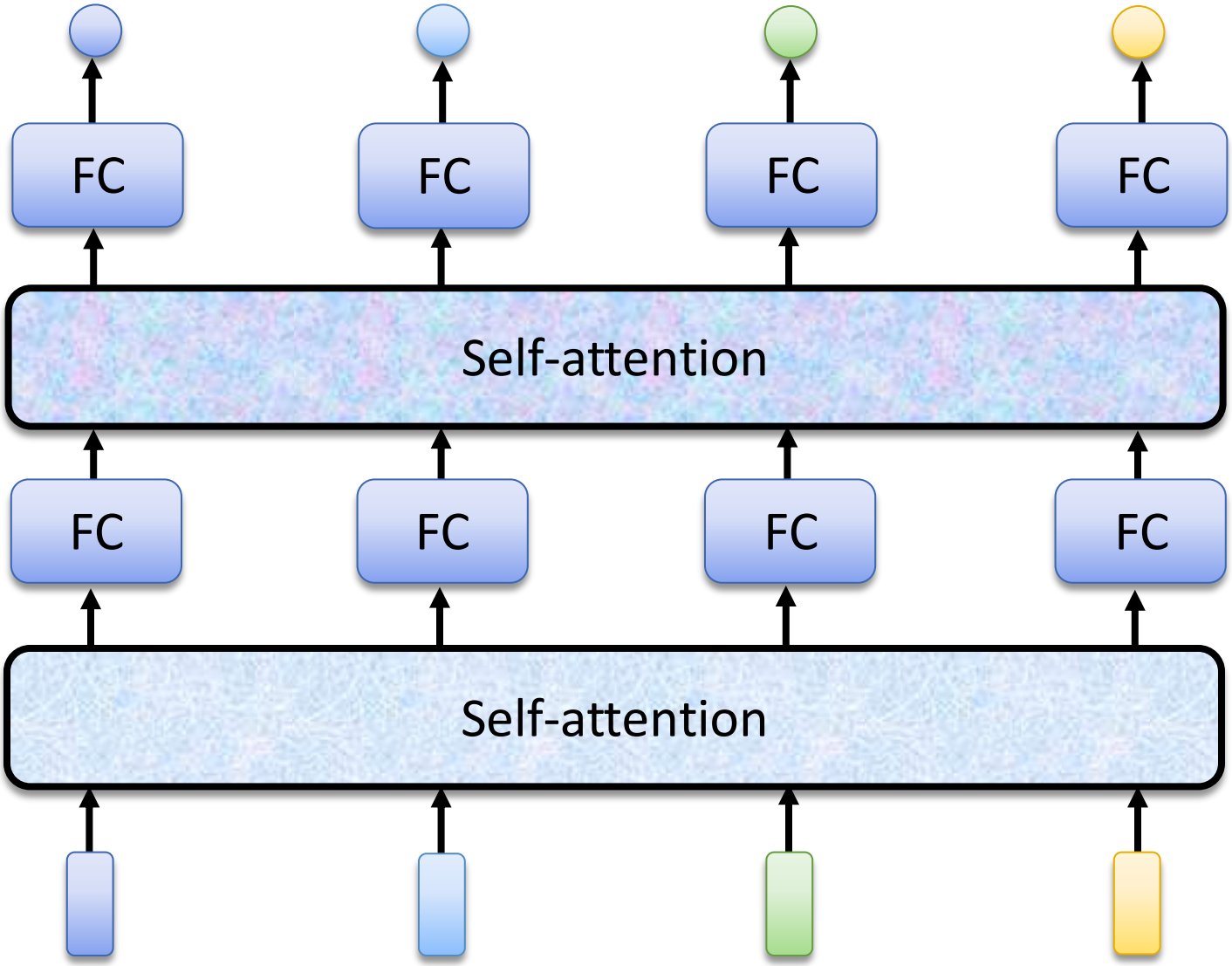
"Here's the information I have!"



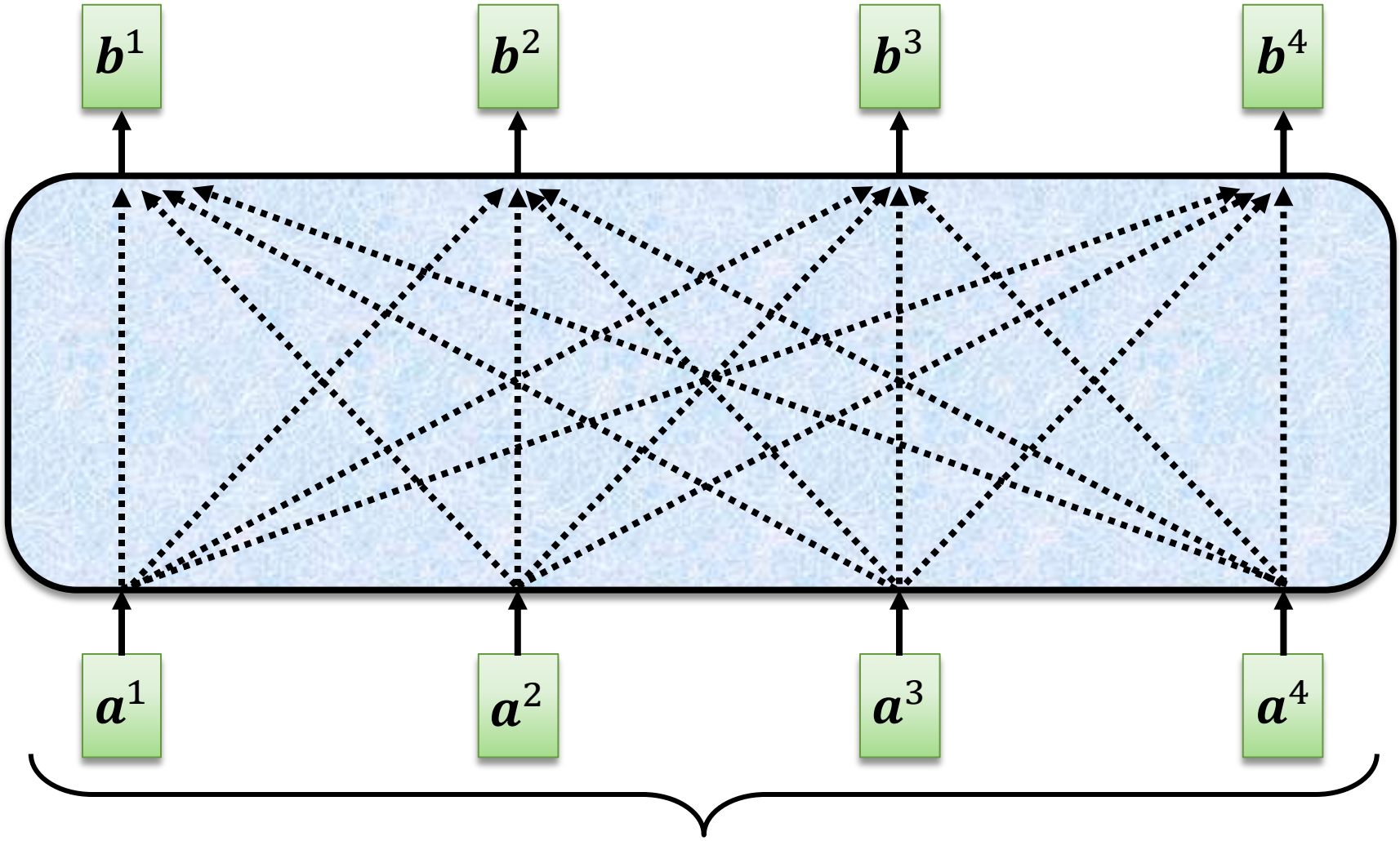
Self-Attention的实现



Self-Attention的实现

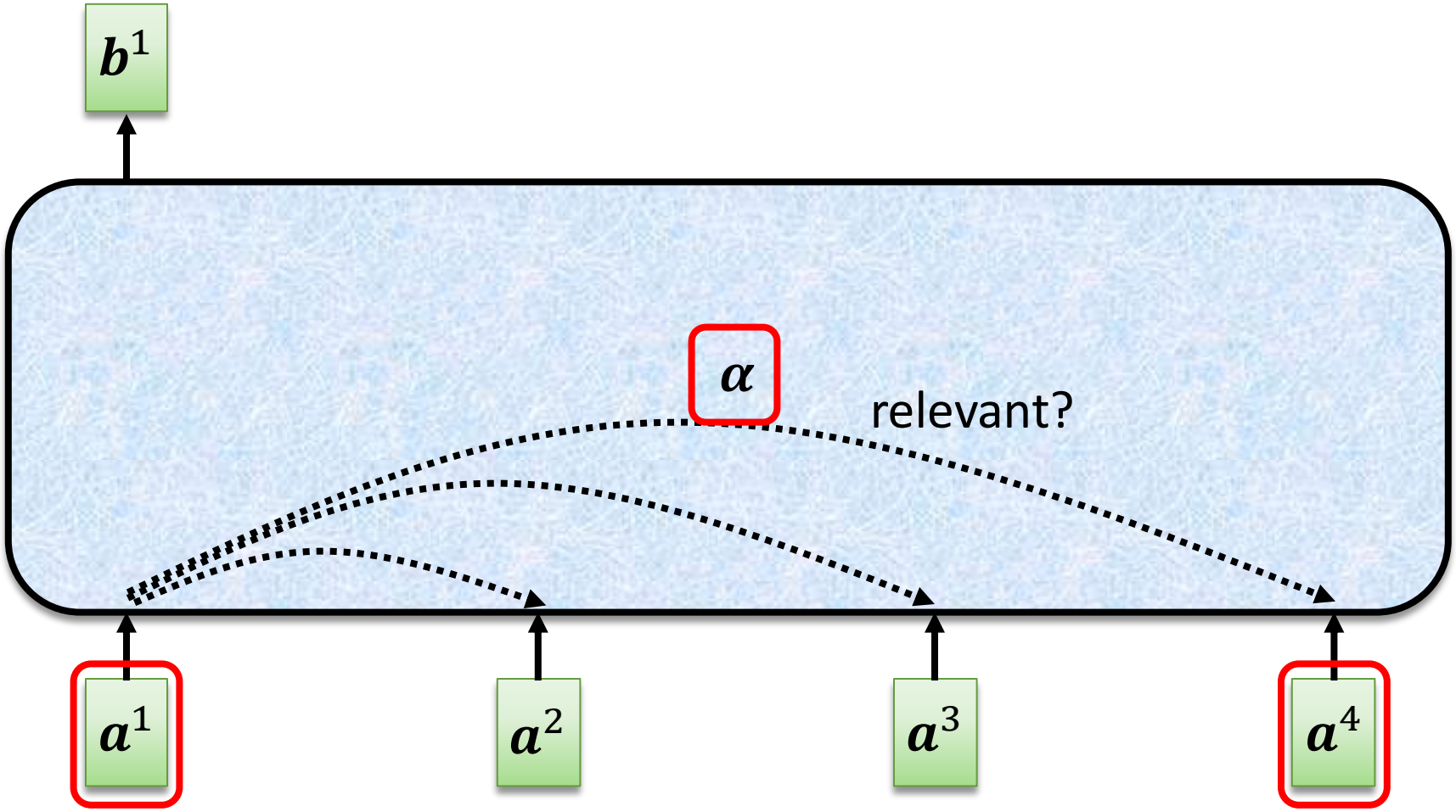


Self-Attention的实现



可以是**输入层**或**隐藏层**

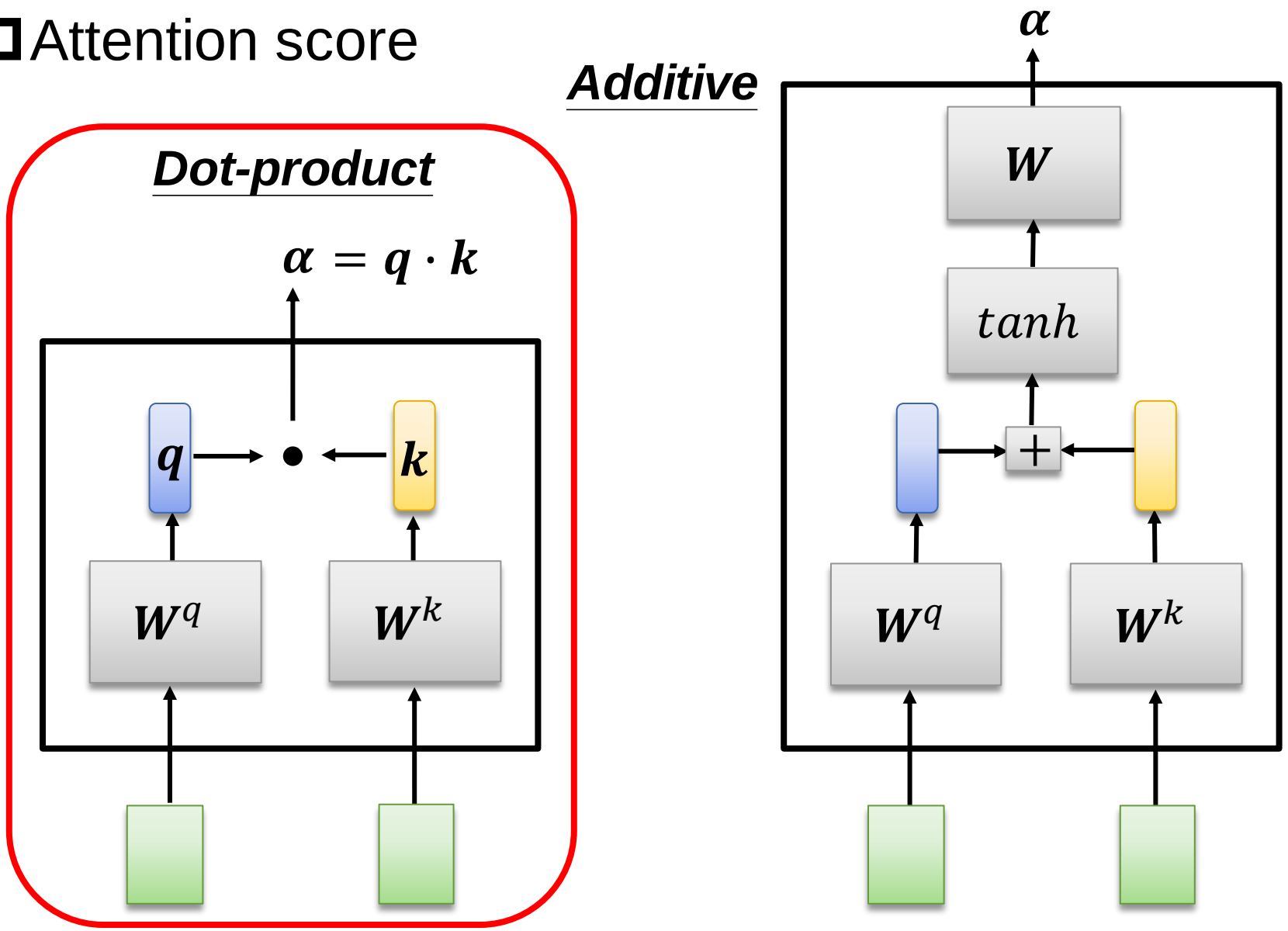
Self-Attention的实现



在一个序列中找到**相关向量(relevant vectors)**

Self-Attention的实现

- Attention score



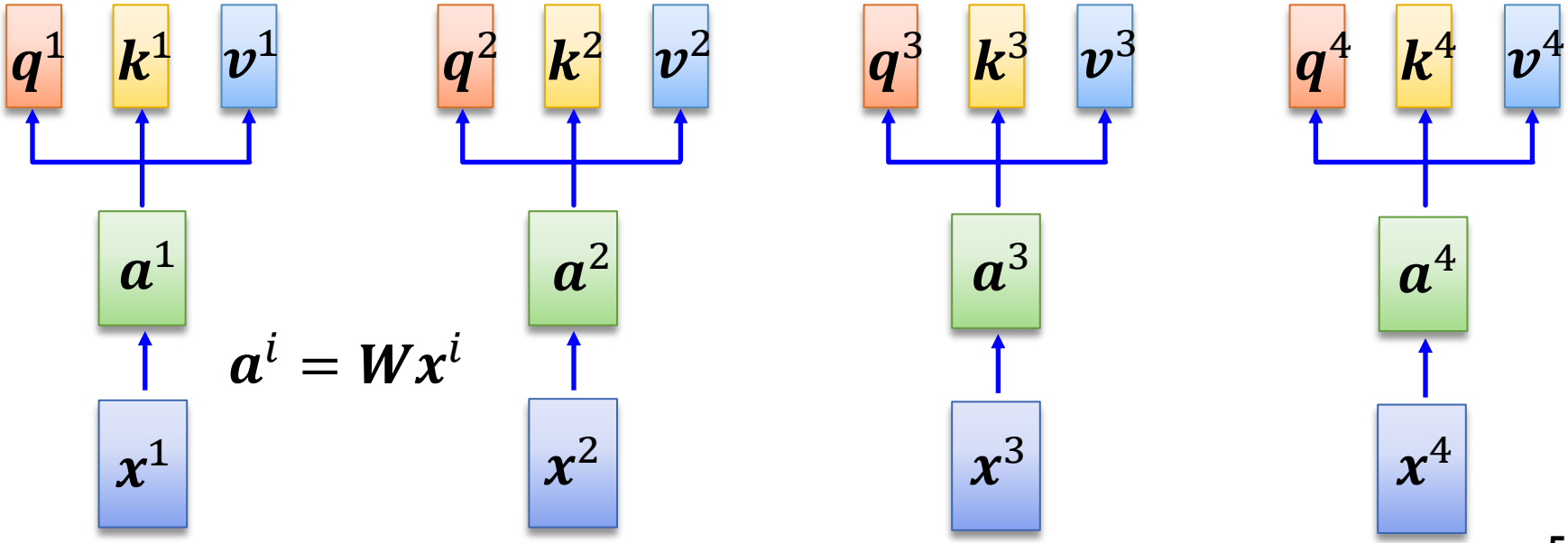
Self-Attention的实现

- Self-Attention 中 Query、Key和Value

$\boldsymbol{q}$: query (asking for information) $\quad \boldsymbol{q}^i = \boldsymbol{W}^q \boldsymbol{a}^i$

$\boldsymbol{k}$: key (saying that it has some information) $\quad \boldsymbol{k}^i = \boldsymbol{W}^k \boldsymbol{a}^i$

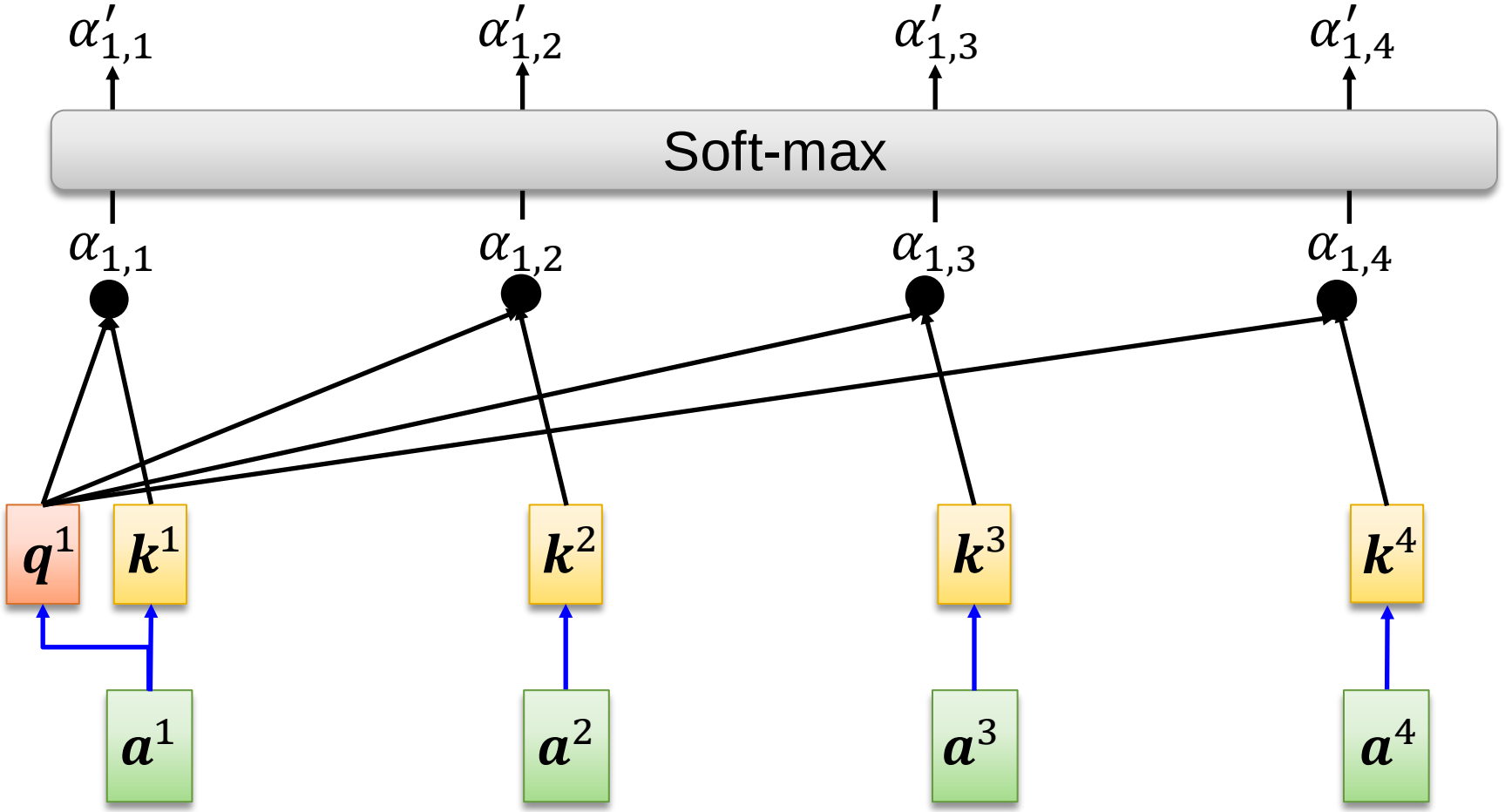
$\boldsymbol{v}$: value (giving the information) $\quad \boldsymbol{v}^i = \boldsymbol{W}^v \boldsymbol{a}^i$



Self-Attention的实现

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

- 计算attention weight



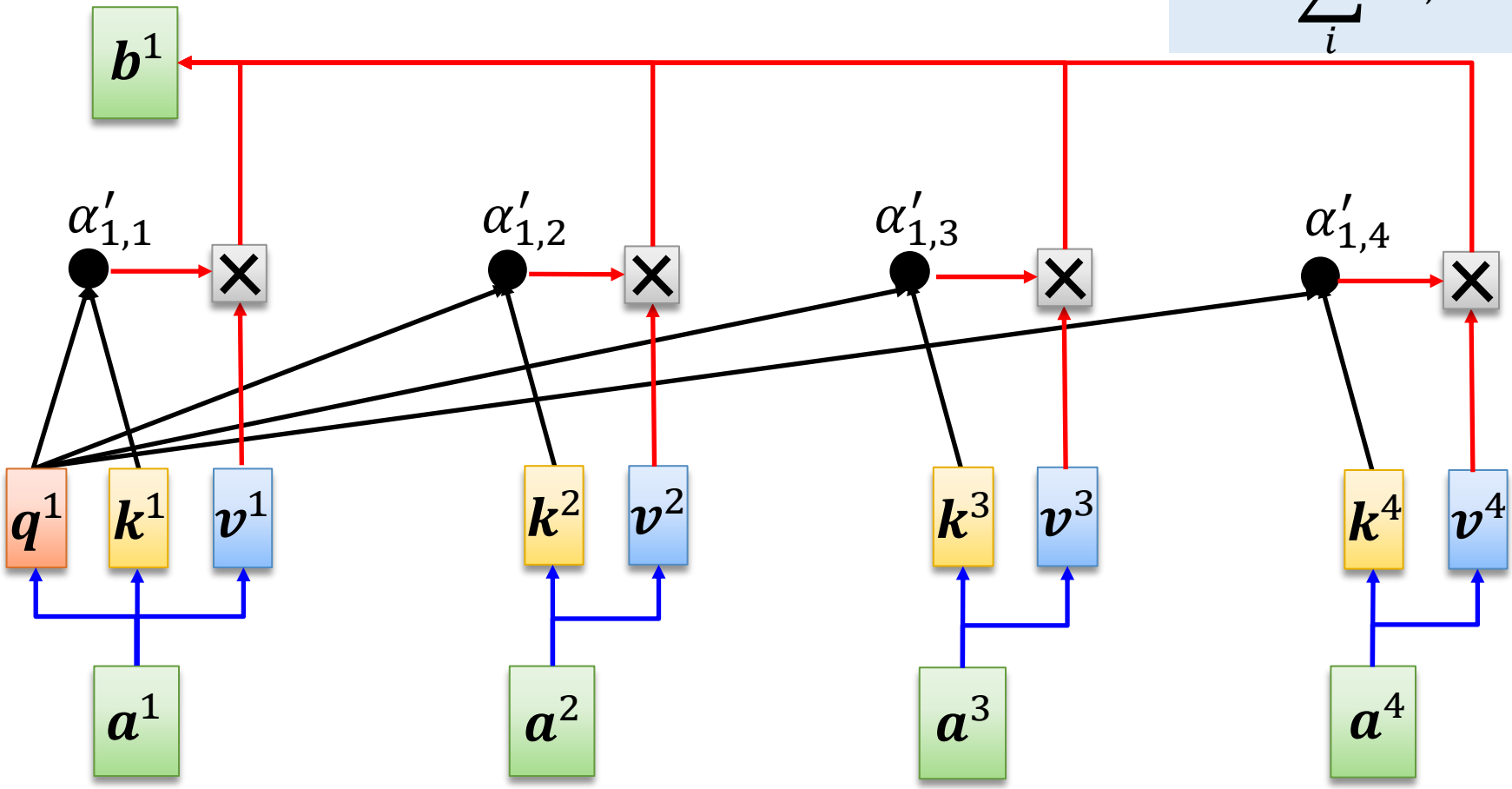
$q^1 = W^q a^1$ $\quad k^2 = W^k a^2$ $\quad k^3 = W^k a^3$ $\quad k^4 = W^k a^4$

$k^1 = W^k a^1$

Self-Attention的实现

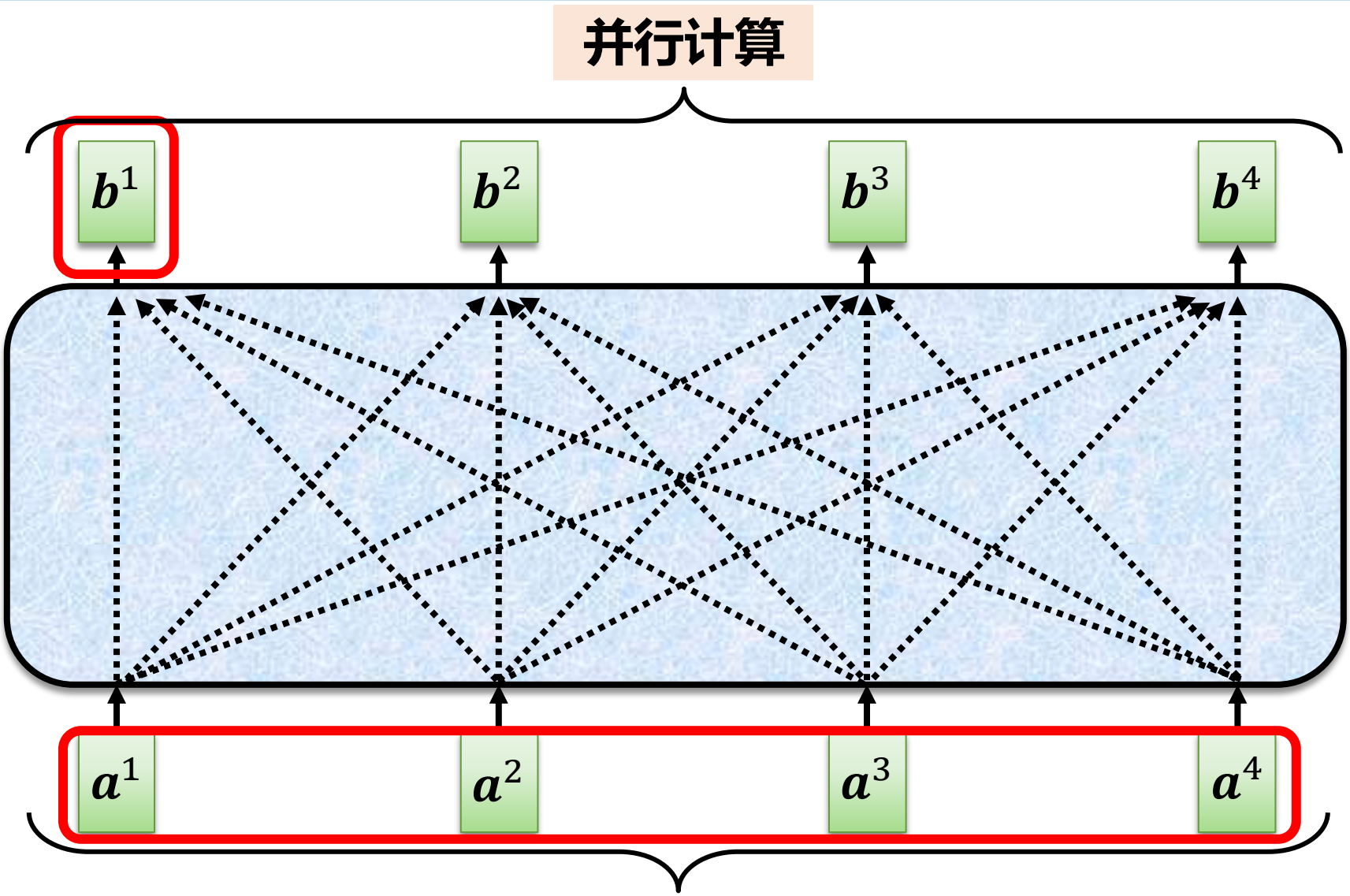
- 基于attention weight提取信息

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



$v^1 = W^v a^1$ $v^2 = W^v a^2$ $v^3 = W^v a^3$ $v^4 = W^v a^4$

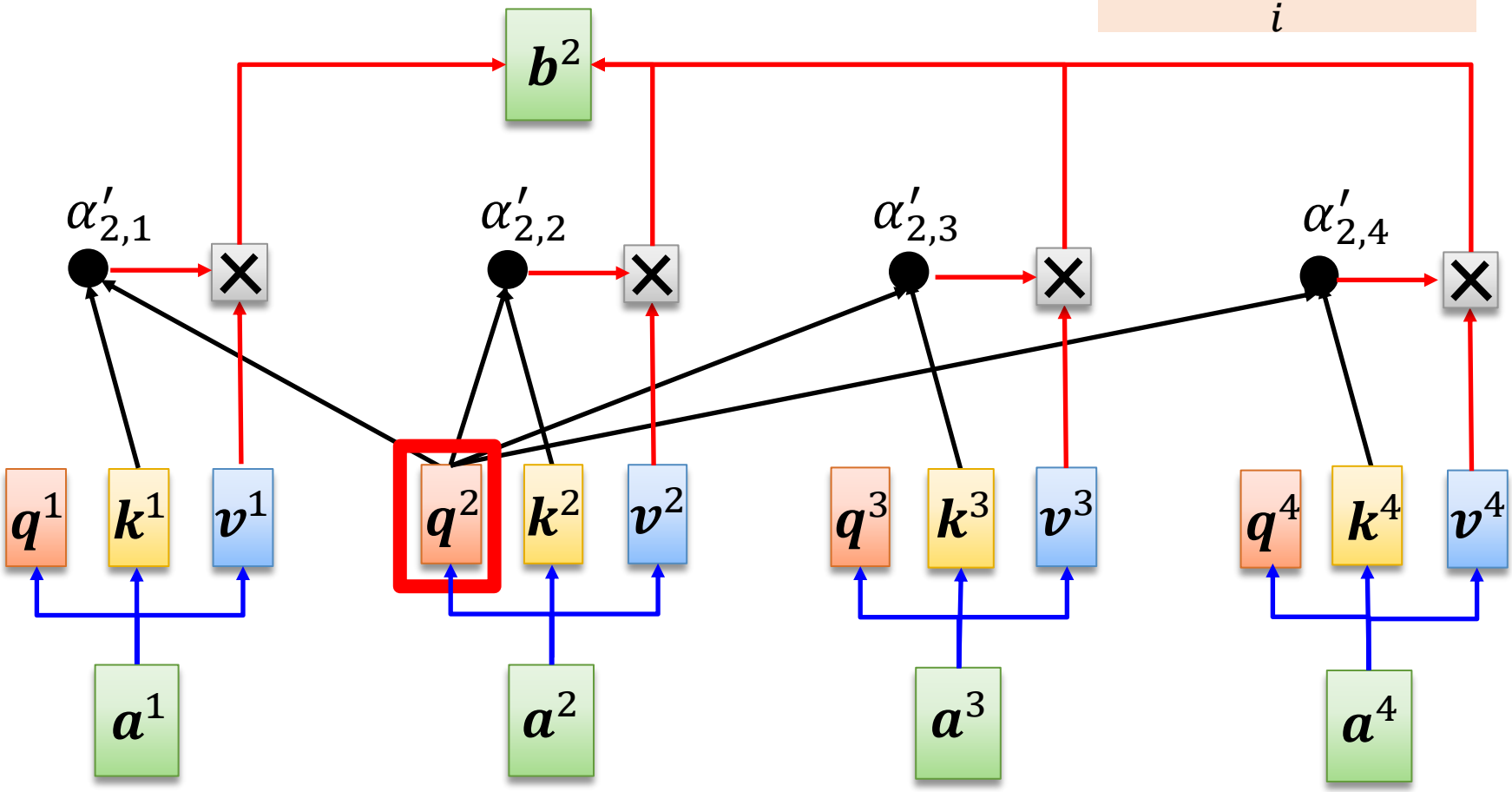
Self-Attention的实现



Self-Attention的实现

- 并行计算

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



Self-Attention的实现

- 并行计算

$$q^i = W^q a^i \qquad q^1 q^2 q^3 q^4 = W^q \, a^1 a^2 a^3 a^4$$

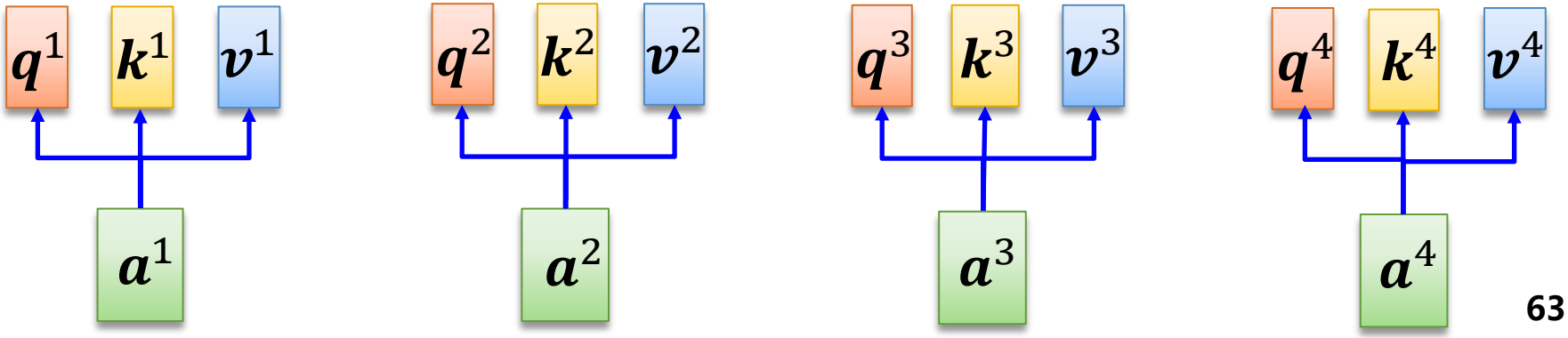
$$Q \qquad\qquad I$$

$$k^i = W^k a^i \qquad k^1 k^2 k^3 k^4 = W^k \, a^1 a^2 a^3 a^4$$

$$K \qquad\qquad I$$

$$v^i = W^v a^i \qquad v^1 v^2 v^3 v^4 = W^v \, a^1 a^2 a^3 a^4$$

$$V \qquad\qquad I$$



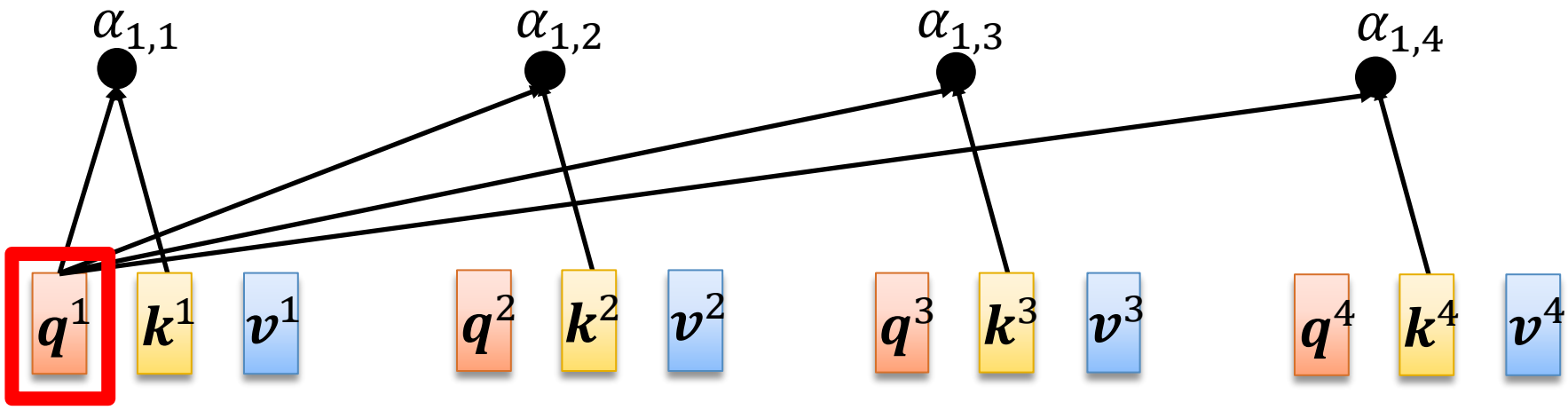
Self-Attention的实现

- 并行计算

$$\alpha_{1,1} = \boldsymbol{k}^1 \boldsymbol{q}^1 \quad \alpha_{1,2} = \boldsymbol{k}^2 \boldsymbol{q}^1$$

$$\alpha_{1,3} = \boldsymbol{k}^3 \boldsymbol{q}^1 \quad \alpha_{1,4} = \boldsymbol{k}^4 \boldsymbol{q}^1$$

$$\begin{matrix}\alpha_{1,1}\\ \alpha_{1,2}\\ \alpha_{1,3}\\ \alpha_{1,4}\end{matrix} = \begin{matrix}\boldsymbol{k}^1\\ \boldsymbol{k}^2\\ \boldsymbol{k}^3\\ \boldsymbol{k}^4\end{matrix} \boldsymbol{q}^1$$

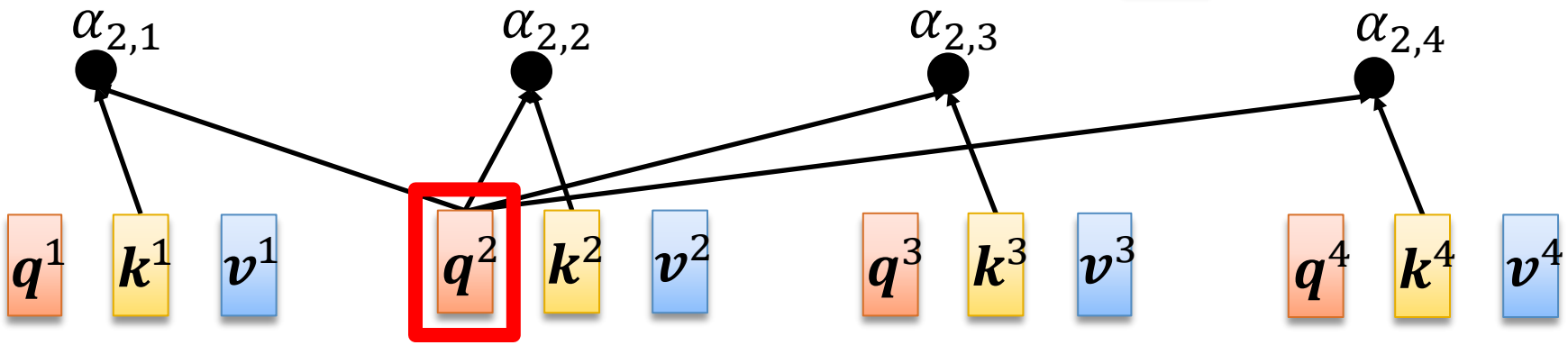


Self-Attention的实现

$$\alpha_{1,1} = k^1 q^1 \quad \alpha_{1,2} = k^2 q^1$$

$$\alpha_{1,3} = k^3 q^1 \quad \alpha_{1,4} = k^4 q^1$$

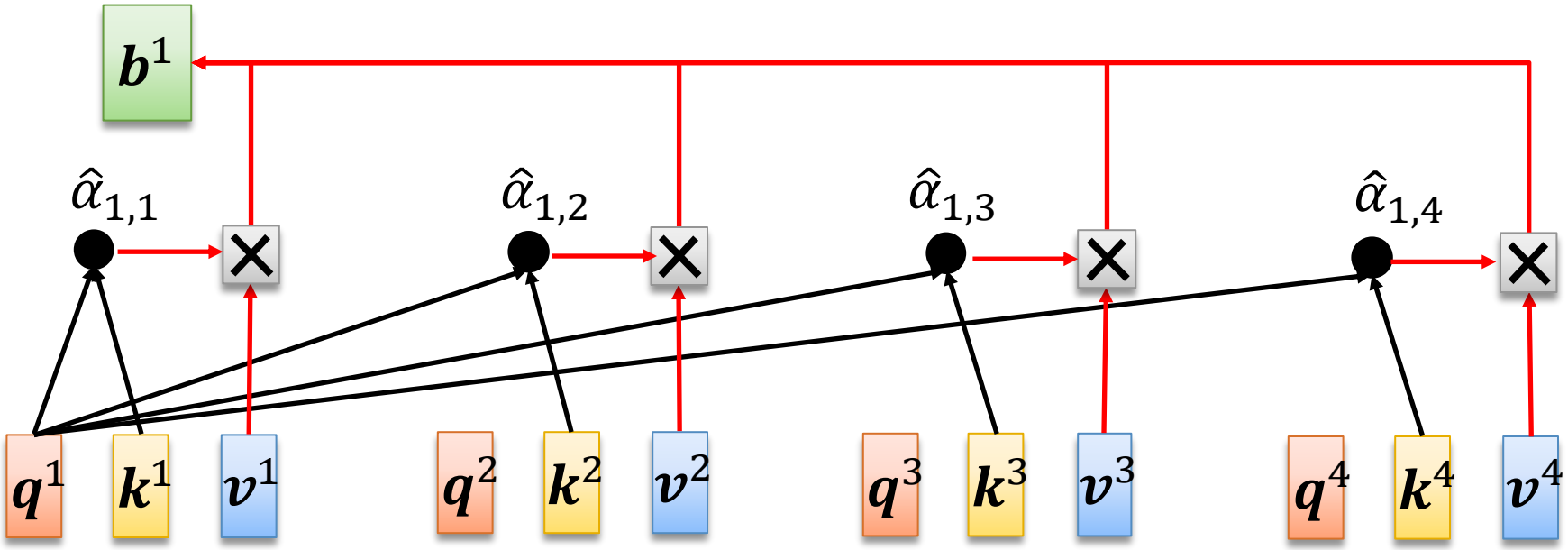
$$\begin{bmatrix}\alpha_{1,1}\\ \alpha_{1,2}\\ \alpha_{1,3}\\ \alpha_{1,4}\end{bmatrix} = \begin{bmatrix}k^1\\ k^2\\ k^3\\ k^4\end{bmatrix} q^1$$



$$\begin{bmatrix}\alpha'_{1,1} & \alpha'_{2,1} & \alpha'_{3,1} & \alpha'_{4,1}\\ \alpha'_{1,2} & \alpha'_{2,2} & \alpha'_{3,2} & \alpha'_{4,2}\\ \alpha'_{1,3} & \alpha'_{2,3} & \alpha'_{3,3} & \alpha'_{4,3}\\ \alpha'_{1,4} & \alpha'_{2,4} & \alpha'_{3,4} & \alpha'_{4,4}\end{bmatrix} = A' \leftarrow \begin{bmatrix}\alpha_{1,1} & \alpha_{2,1} & \alpha_{3,1} & \alpha_{4,1}\\ \alpha_{1,2} & \alpha_{2,2} & \alpha_{3,2} & \alpha_{4,2}\\ \alpha_{1,3} & \alpha_{2,3} & \alpha_{3,3} & \alpha_{4,3}\\ \alpha_{1,4} & \alpha_{2,4} & \alpha_{3,4} & \alpha_{4,4}\end{bmatrix} = A = \begin{bmatrix}k^1\\ k^2\\ k^3\\ k^4\end{bmatrix} = K^T \quad \begin{matrix}q^1 & q^2 & q^3\end{matrix} = Q$$

Self-Attention的实现

- 计算b^i

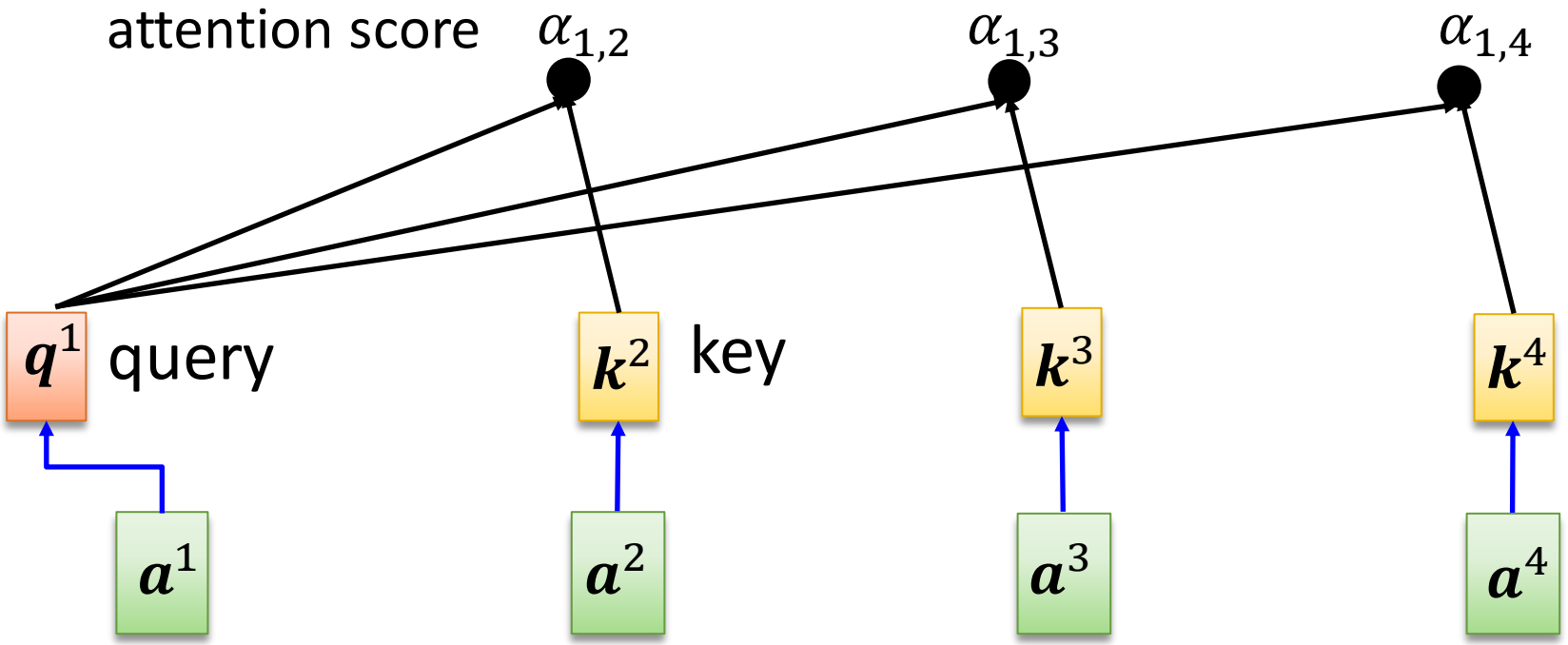


$$\underbrace{b^1\ b^2\ b^3\ b^4}_{O} = \underbrace{v^1\ v^2\ v^3\ v^4}_{V} \underbrace{\begin{bmatrix} \alpha'_{1,1} & \alpha'_{2,1} & \alpha'_{3,1} & \alpha'_{4,1} \\ \alpha'_{1,2} & \alpha'_{2,2} & \alpha'_{3,2} & \alpha'_{4,2} \\ \alpha'_{1,3} & \alpha'_{2,3} & \alpha'_{3,3} & \alpha'_{4,3} \\ \alpha'_{1,4} & \alpha'_{2,4} & \alpha'_{3,4} & \alpha'_{4,4} \end{bmatrix}}_{A'}$$

Self-Attention的实现

- 计算attention score

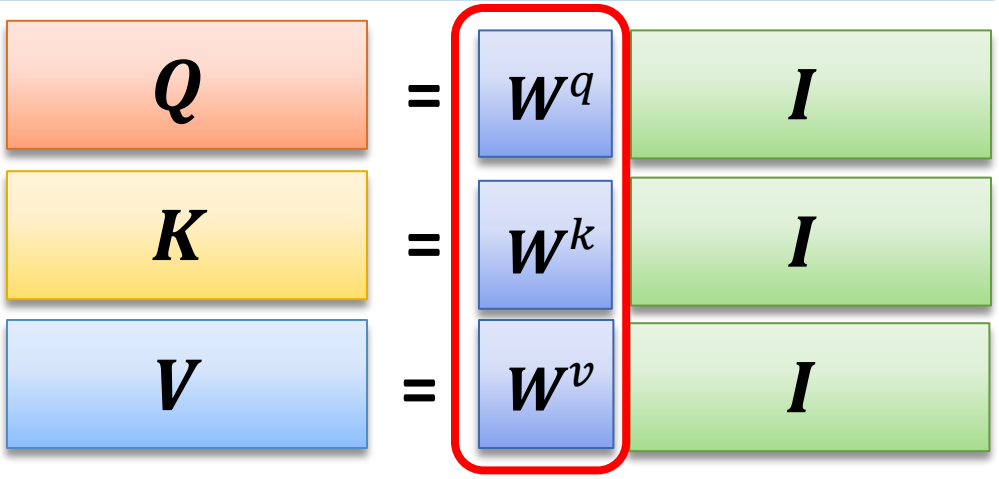
$$\alpha_{1,2} = \boldsymbol{q}^1 \cdot \boldsymbol{k}^2 \qquad \alpha_{1,3} = \boldsymbol{q}^1 \cdot \boldsymbol{k}^3 \qquad \alpha_{1,4} = \boldsymbol{q}^1 \cdot \boldsymbol{k}^4$$

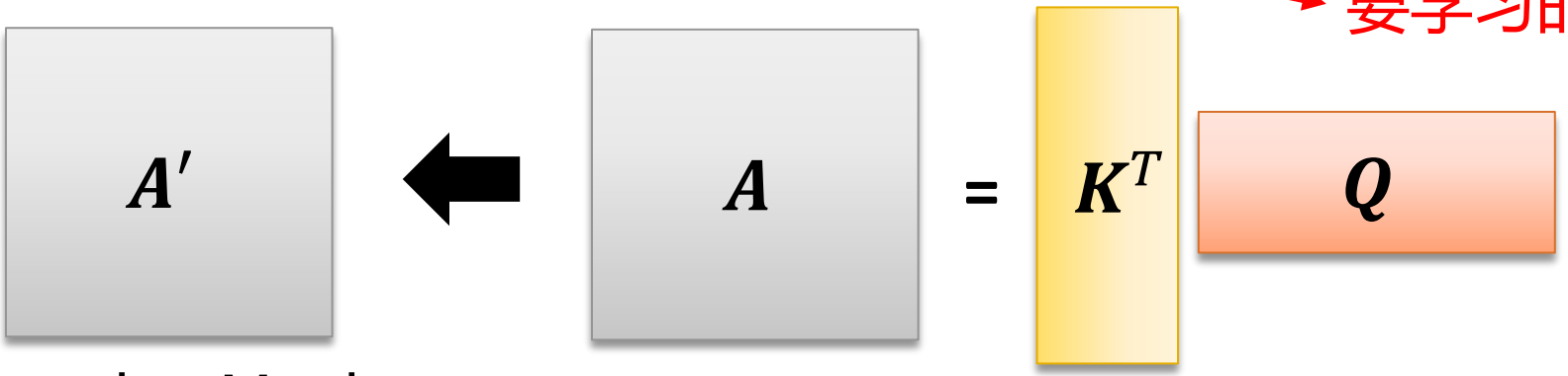


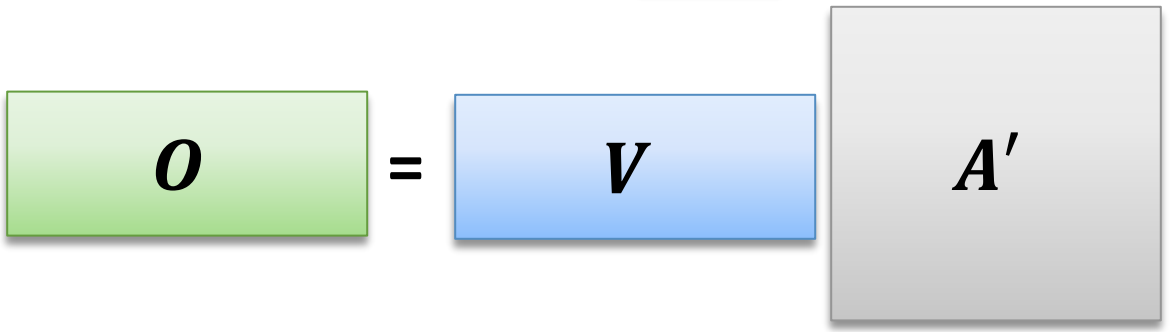
$$\boldsymbol{q}^1 = \boldsymbol{W}^q \boldsymbol{a}^1 \qquad \boldsymbol{k}^2 = \boldsymbol{W}^k \boldsymbol{a}^2 \qquad \boldsymbol{k}^3 = \boldsymbol{W}^k \boldsymbol{a}^3 \qquad \boldsymbol{k}^4 = \boldsymbol{W}^k \boldsymbol{a}^4$$

Self-Attention的实现

- 矩阵表示







实际运用中$\sqrt{d_k}$的作用

☐ Scale Attention Scores

$$Attention(q, k, v) = \overbrace{softmax\left(\frac{qk^T}{\sqrt{d_k}}\right)}^{\text{Attention weights}} v$$

from (q) to (k) vector dimensionality of K, V

$\sqrt{d_k}$必不可少!

- $\frac{1}{\sqrt{d_k}}$is used to **scale attention score** before normalizing it through the softmax function
- The scaling by d_k ensures that the **Euclidean length of the weight vectors will be approximately in the same magnitude**. This helps **prevent the attention weights from becoming too small or too large**, which could lead to numerical instability or affect the model's ability to converge during training.

https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html

选学：为什么除以$\sqrt{d_k}$如此重要

- 没有除以$\sqrt{d_k}$，容易出现Transformer**模型训练不收敛**的情况。但Google的T5**对模型初始化策略做出调整**，使得未除$\sqrt{d_k}$依旧可以正常收敛。
- 因此，除以$\sqrt{d_k}$与模型初始化有关
  - 一般情况下，都是**从指定均值和方差的随机分布中进行采样**来初始化模型。
  - 常用的随机分布有三个：正态分布(Normal)、均匀分布(Uniform)和截尾正态分布(Truncated Normal)。
  - **截尾正态分布**既指定均值μ和方差σ^2，也需要指定区间$[a,b]$，从$\mathcal{N}(\mu,\sigma^2)$中采样，如果采样结果在$[a,b]$中，则保留该结果，否则重复采样直到采样结果落到$[a,b]$中。

选学：除以$\sqrt{d_k}$与模型初始化相关

- 从二阶矩的角度来理解模型初始化方法
  - 一般情况下，**推导模型初始化方法的思想是尽量让输入输出具有相同的均值和方差**，通常会假设输入是均值为0、方差为1的随机向量，然后试图让输出的均值为0、方差为1。
  - 本质上，只需要一个衡量某个指标是否“消失”或者“爆炸”的指标，0均值、1方差是非必要的，可用**二阶(原点)矩**来代替，它可看成是L_2模长的变体，和方差的作用类似。

选学：从二阶矩的角度来理解模型初始化

Xavier初始化

考察无激活函数的全连接层(设输入和输出节点数分别m为和n)

$$y_j = b_j + \sum_i x_i w_{i,j}$$

简单起见，通常全零初始化偏置项b_j，并设$w_{i,j}$的均值$\mathbb{E}(w_{i,j})$也为0。计算二阶矩如下：

$$\mathbb{E}\left[y_j^2\right] = \mathbb{E}\left[\left(\sum_i x_i w_{i,j}\right)^2\right] = \mathbb{E}\left[\left(\sum_{i_1} x_{i_1} w_{i_1,j}\right)\left(\sum_{i_2} x_{i_2} w_{i_2,j}\right)\right]$$

$$= \mathbb{E}\left[\sum_{i_1,i_2}\left(x_{i_1}x_{i_2}\right)\left(w_{i_1,j}w_{i_2,j}\right)\right] = \sum_{i_1,i_2}\mathbb{E}\left[x_{i_1}x_{i_2}\right]\mathbb{E}\left[w_{i_1,j}w_{i_2,j}\right]$$

选学：从二阶矩的角度来理解模型初始化

□Xavier初始化

$$\mathbb{E}\left[y_j^2\right] = \sum_{i_1,i_2} \mathbb{E}\left[x_{i_1} x_{i_2}\right] \mathbb{E}\left[w_{i_1,j} w_{i_2,j}\right]$$

注意到$w_{i_1,j}$和$w_{i_2,j}$是独立同分布的，所以当$i_1 \neq i_2$时，$\mathbb{E}\left[w_{i_1,j} w_{i_2,j}\right] = \mathbb{E}\left[w_{i_1,j}\right] \mathbb{E}\left[w_{i_2,j}\right] = 0$。

当$i_1 = i_2 = i$时，假设输入的二阶矩为1，那么

$$\mathbb{E}\left[y_j^2\right] = \sum_{i} \mathbb{E}\left[x_i^2\right] \mathbb{E}\left[w_{i,j}^2\right] = m\mathbb{E}\left[w_{i,j}^2\right]$$

所以为使$\mathbb{E}\left[y_j^2\right] = 1$，则有$\mathbb{E}\left[w_{i,j}^2\right] = \frac{1}{m}$。综合均值为0的假设，Xavier初始化策略是“**从均值为0、方差为$\frac{1}{m}$的随机分布中独立重复采样**”。

选学：从二阶矩的角度来理解模型初始化

NTK参数化

- 用均值为0、方差为1的随机分布来初始化，但是**将输出结果除以$\sqrt{m}$**，即模型变为：

$$y_j = b_j + \frac{1}{\sqrt{m}} \sum_i x_i w_{i,j}$$

- 利用NTK参数化，可以将所有参数都用标准方差初始化，但依然保持二阶矩不变。
- **好处**：利用NTK参数化后，每个参数的量级大致都是相同的$\mathcal{O}(1)$级别，于是可以设置较大的学习率；能更平等地处理每一个参数，比较形象地了解到训练的更新幅度，以便更好地调整参数。

选学：为什么除以$\sqrt{d_k}$如此重要

- 对于两个d维向量$\boldsymbol{q}$和$\boldsymbol{k}$，假设它们都采样自”均值为0、方差为1”的分布，那么**q和k内积的二阶矩**是：

$$\mathbb{E}[(\boldsymbol{q}\cdot\boldsymbol{k})^2]=\mathbb{E}\left[\left(\prod_{i=1}^{\boldsymbol{d}}q_i\,k_i\right)^2\right]=\mathbb{E}\left[\left(\sum_i q_i k_i\right)\left(\sum_j q_j k_j\right)\right]$$

$$=\mathbb{E}\left[\sum_{i,j}(q_iq_j)(k_ik_j)\right]=\sum_{i,j}\mathbb{E}[q_iq_j]\mathbb{E}[k_ik_j]=\sum_i\mathbb{E}[q_i^2]\mathbb{E}[k_i^2]=d$$

内积的二阶矩为d，由于其均值是0，因此其方差也是d

可以大致认为内积之后、softmax之前的数值在$-3\sqrt{d}$到$3\sqrt{d}$的范围内。通常$d\geq 64$，所以$e^{3\sqrt{d}}$远大于$e^{-3\sqrt{d}}$，因此经过softmax之后，Attention的分布**非常接近one-hot**。

严重的梯度消失问题！

选学：为什么除以$\sqrt{d_k}$如此重要

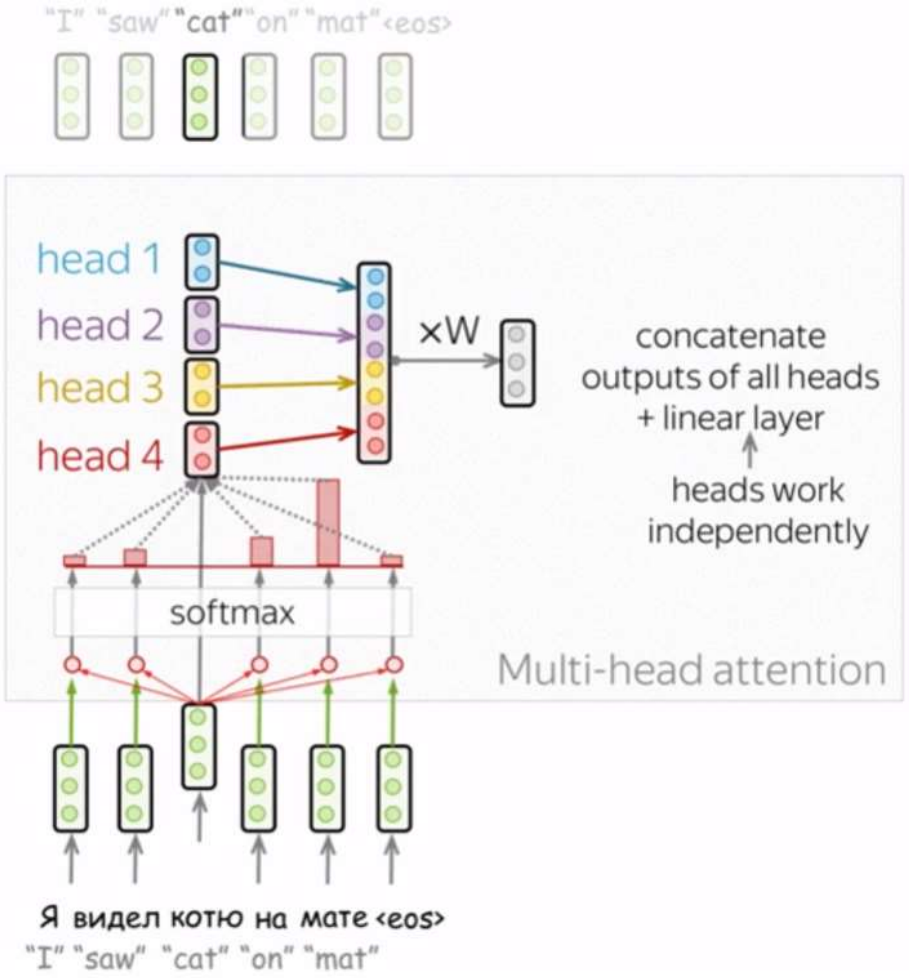
- 解决方法
  - 常规的Transformer中做法

    参照NTK参数化，在内积之后除以$\sqrt{d}$，使得$\boldsymbol{q}\cdot\boldsymbol{k}$的方差变为$\mathbf{1}$，对应的$e^3$和$e^{-3}$都不至于多大或过小，这样softmax之后不会变成one-hot导致梯度消失。
  - Google的T5中做法

    不除以$\sqrt{d}$，但在初始化$\boldsymbol{q}$和$\boldsymbol{k}$的全连接层的时候，其初始化方差要额外除以$\sqrt{d}$，这同样使得$\boldsymbol{q}\cdot\boldsymbol{k}$的初始化方差变为$\mathbf{1}$。

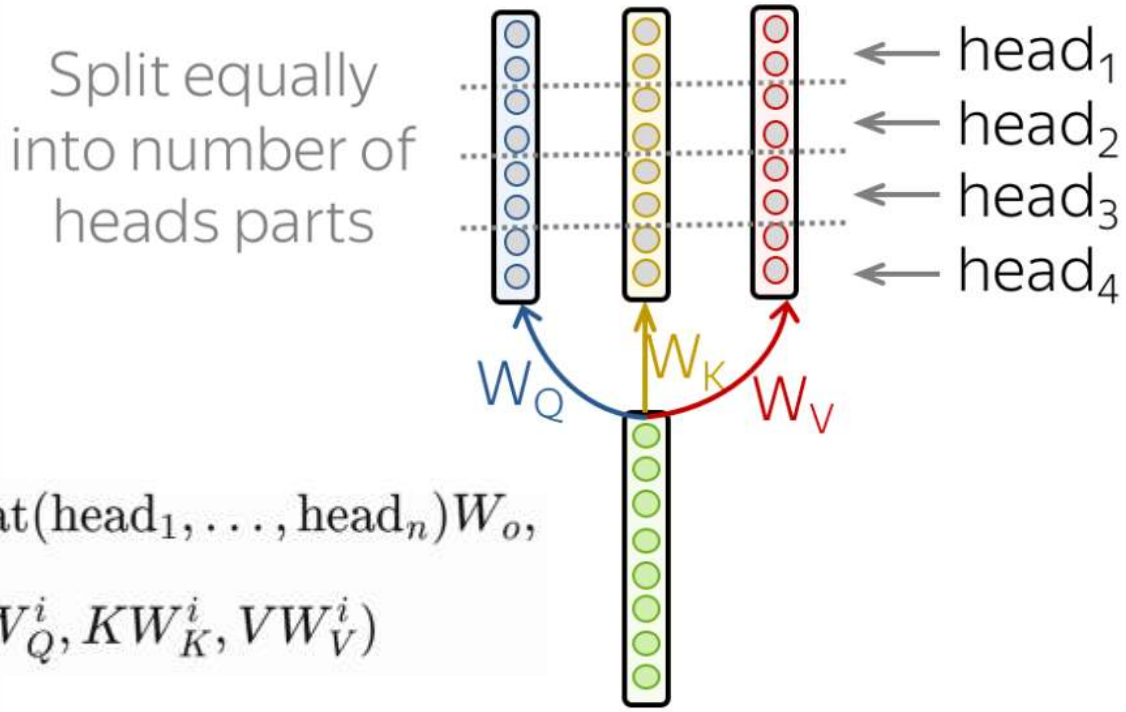
Multi-head Self-attention

- 独立专注于不同的事情
  - 通常情况下，理解单词在句子中的作用需要了解它与句子不同部分的关系
  - 每个词都是许多关系的一部分
  - 因此，我们必须让模型关注不同的事物
  - **有几个独立工作的“head”**



Multi-head Self-attention

- 本质上作为数个注意力机制来实现的，其结果被组合起来



$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_n)W_o,$$

$$\text{head}_i = \text{Attention}(QW_Q^i, KW_K^i, VW_V^i)$$

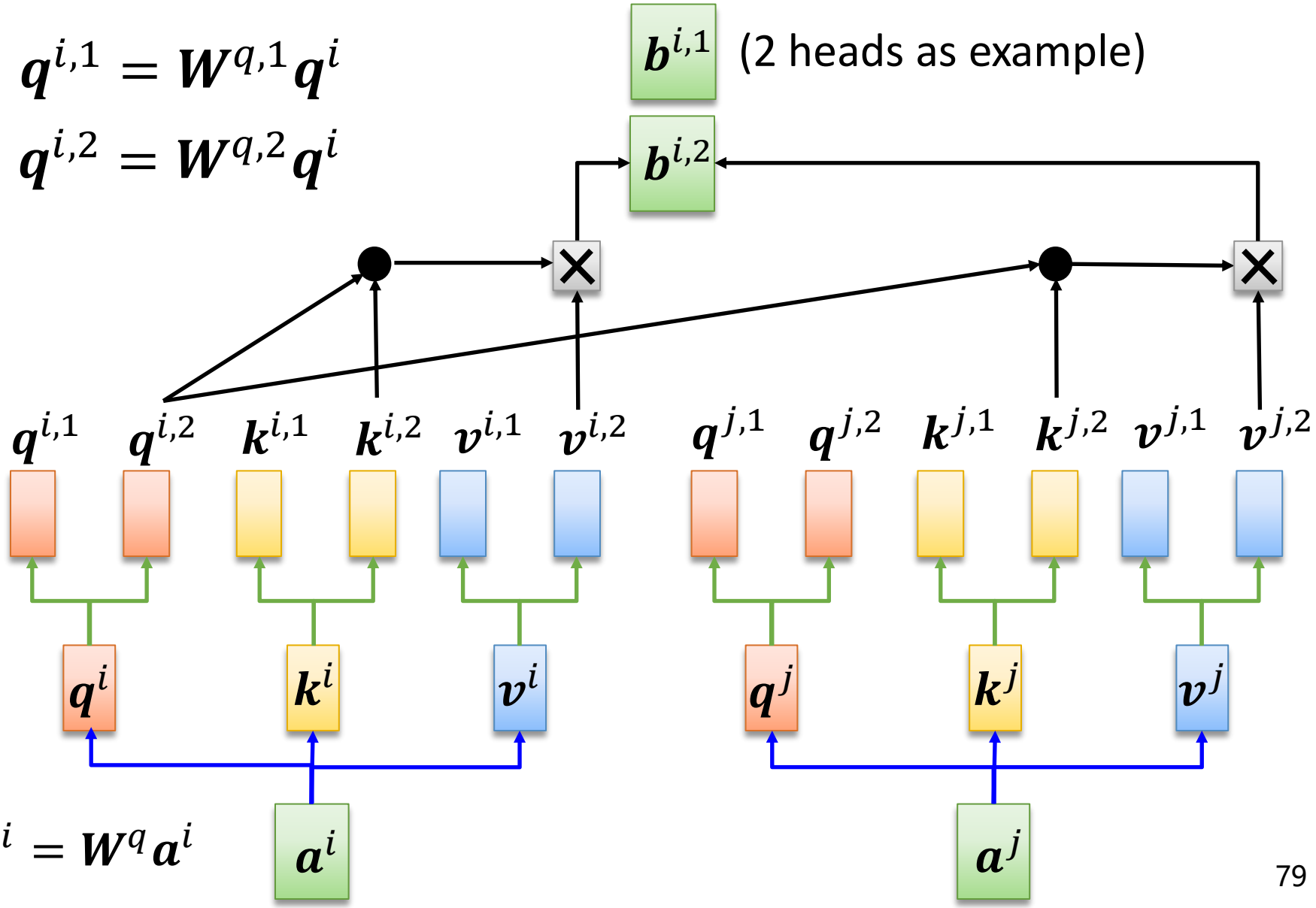
在实现中，只需将为单头注意力计算的query、key和value分成几个部分。 这样，具有一个或多个注意力头的模型具有相同的大小。

https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html#analysis_interpretability

Multi-head Self-attention

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

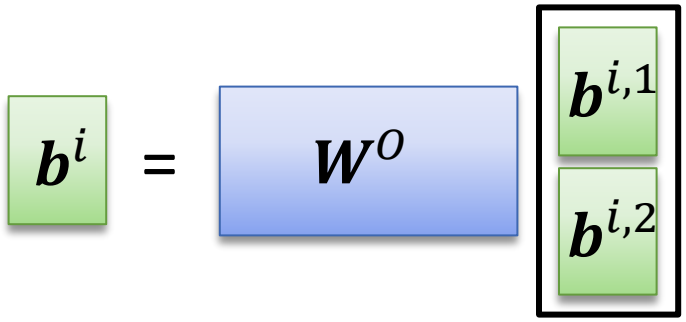
$$q^{i,2} = W^{q,2} q^i$$

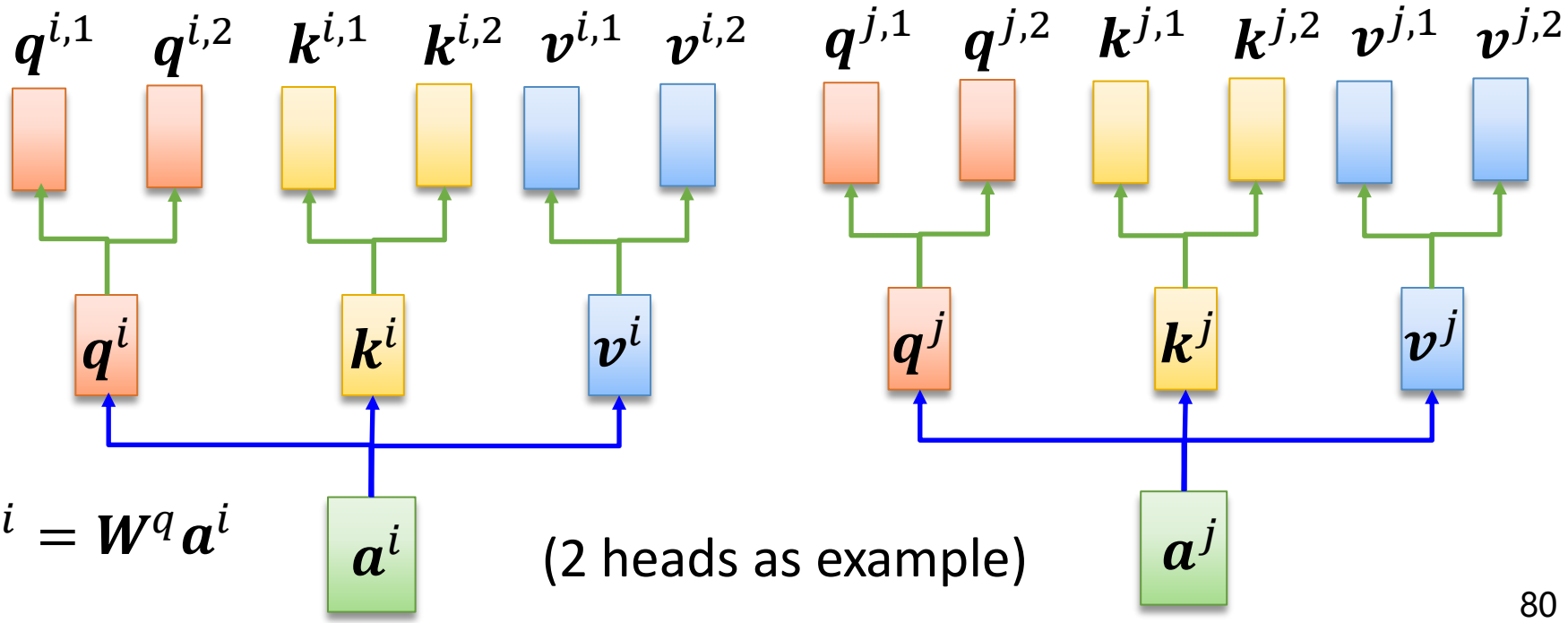


$$q^i = W^q a^i$$

Multi-head Self-attention

- 不同类型的相关性





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

(2 heads as example)

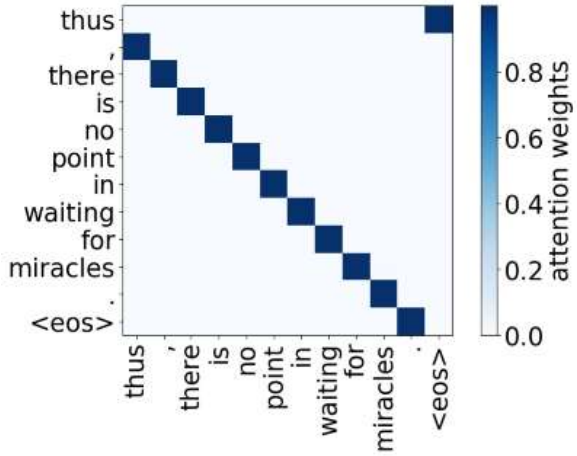
Multi-Head Self-Attention

- What are these heads doing?
  - 多头注意力是Transformer中引入的**inductive bias**
  - 只有**少数头部**对翻译很重要
  - 这些头扮演着可解释的“角色”。
- 重要的头部
  - Positional heads：**关注token的直接邻居**，并且模型有几个这样的头
  - Syntactic heads：学会跟踪句子中的一些**主要句法关系**(主语-动词，动词-宾语)
  - Rare tokens head：第一层最重要的头部关注句子中**最不常见的token(对于在不同的语言对上训练的模型)**

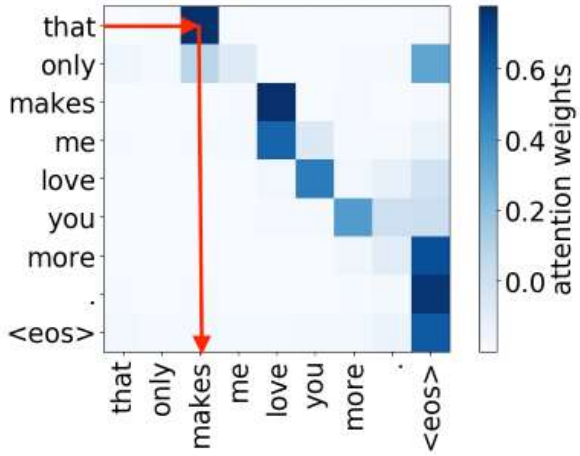
Multi-Head Self-Attention

- What are these heads doing?

Positional heads

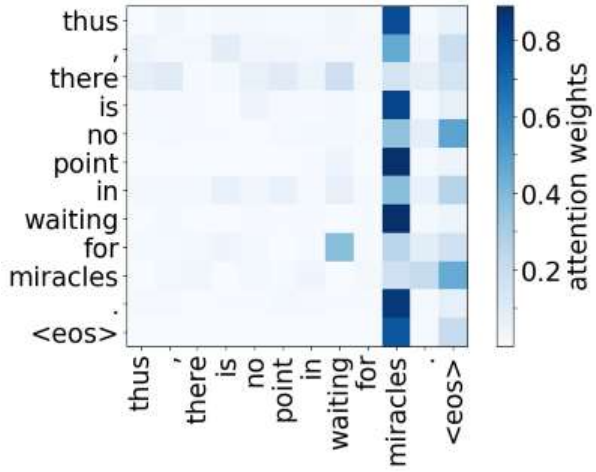


Syntactic heads



模型确实学会了跟踪单词之间的关系！

Rare tokens head



Rare tokens head很可能是过度拟合的迹象。 通过查看最不频繁的标记，模型试图抓住这些罕见的“线索”。

https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html#analysis_interpretability

内容导览

- 序列到序列(Seq2Seq)
- 循环神经网络(RNN)
- 注意力机制的基本原理
- 自注意力/多头注意力/交叉注意力
- Transformer
- ELMO/BERT/GPT

Transformer

□Transformer是神经网络架构的里程碑进展

Attention is all you need

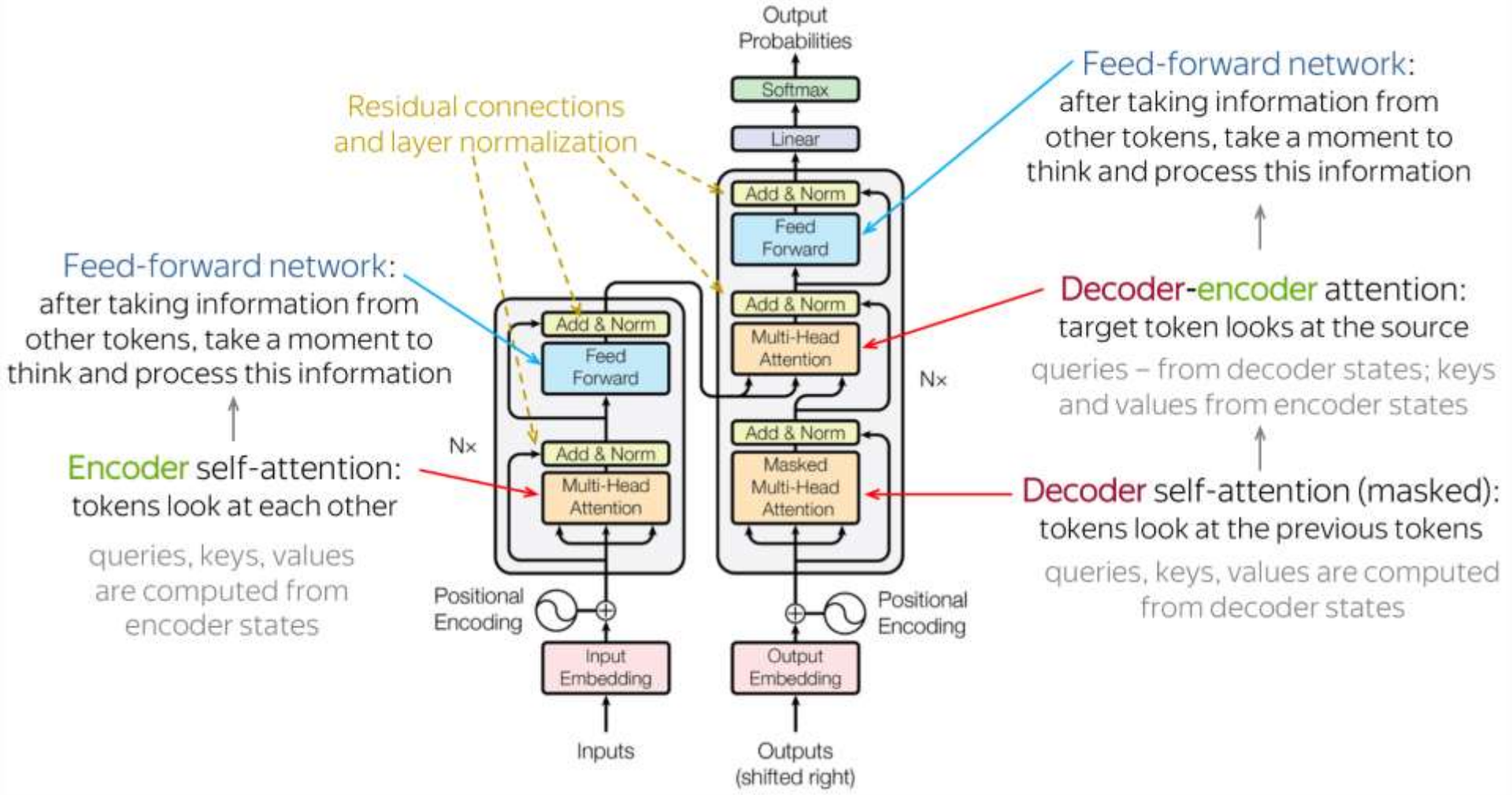
A Vaswani, N Shazeer, N Parmar… - Advances in neural …, 2017 - proceedings.neurips.cc

… to attend to **all** positions in the decoder up to and including that position. **We need** to prevent … **We** implement this inside of scaled dot-product **attention** by masking out (setting to −∞) …

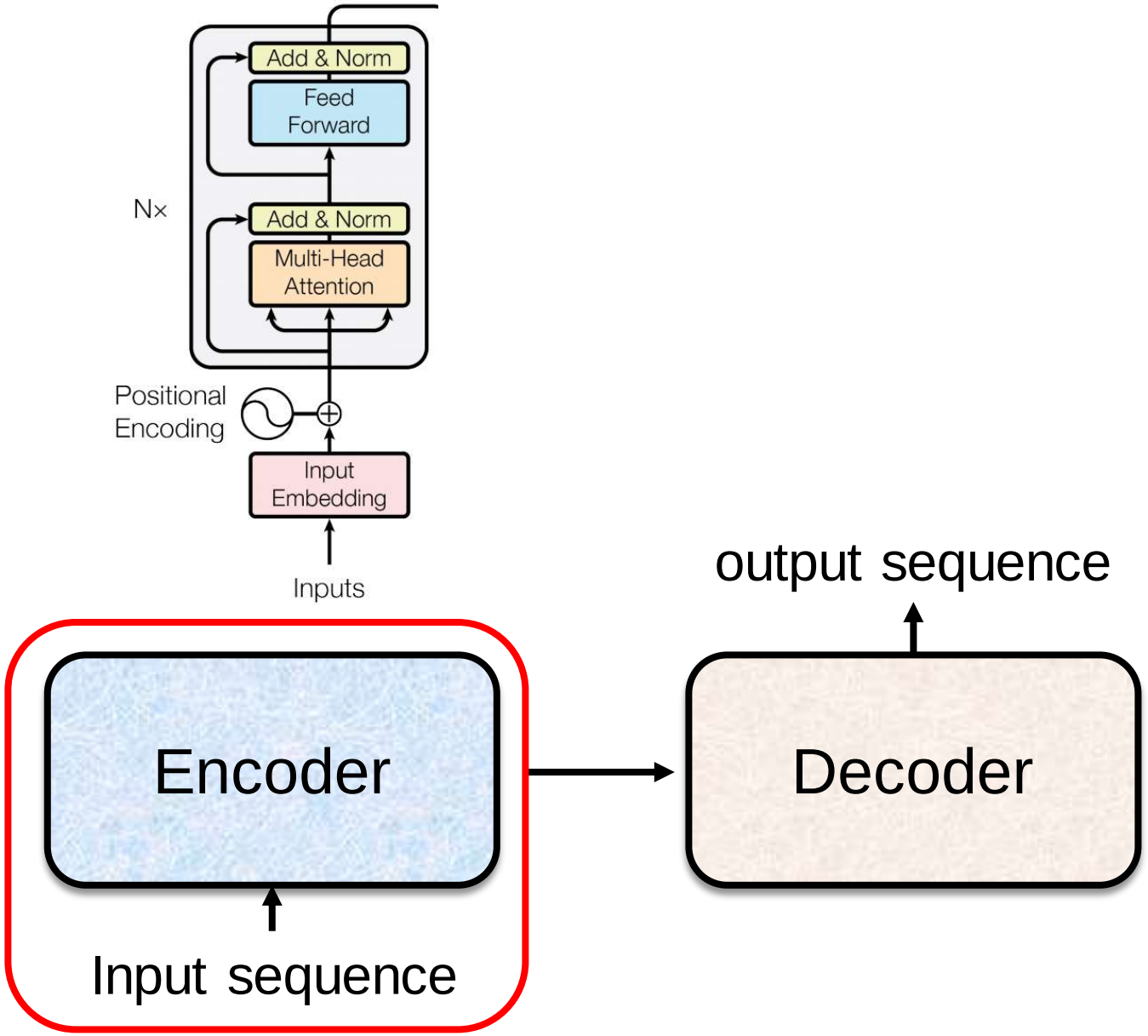
☆ Save 99 Cite Cited by 97503 Related articles All 62 versions »

Attention is all you need

Transformer架构

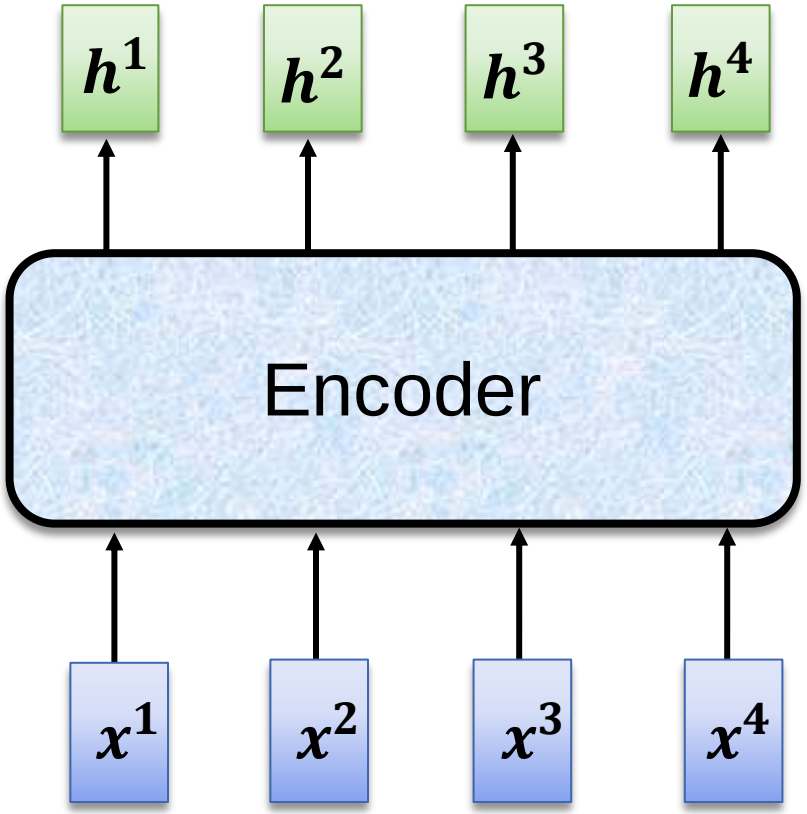


Transformer: Encoder

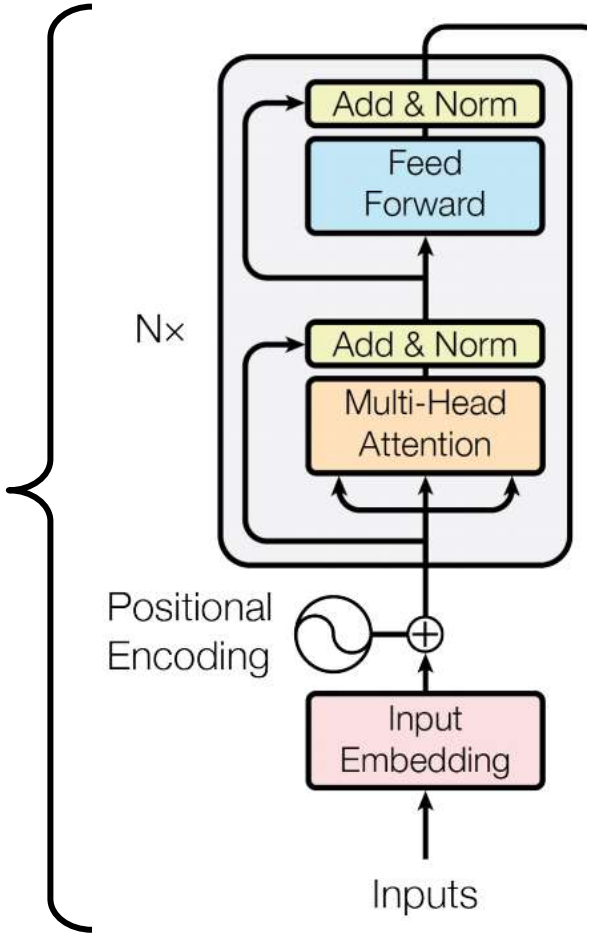


Transformer: Encoder

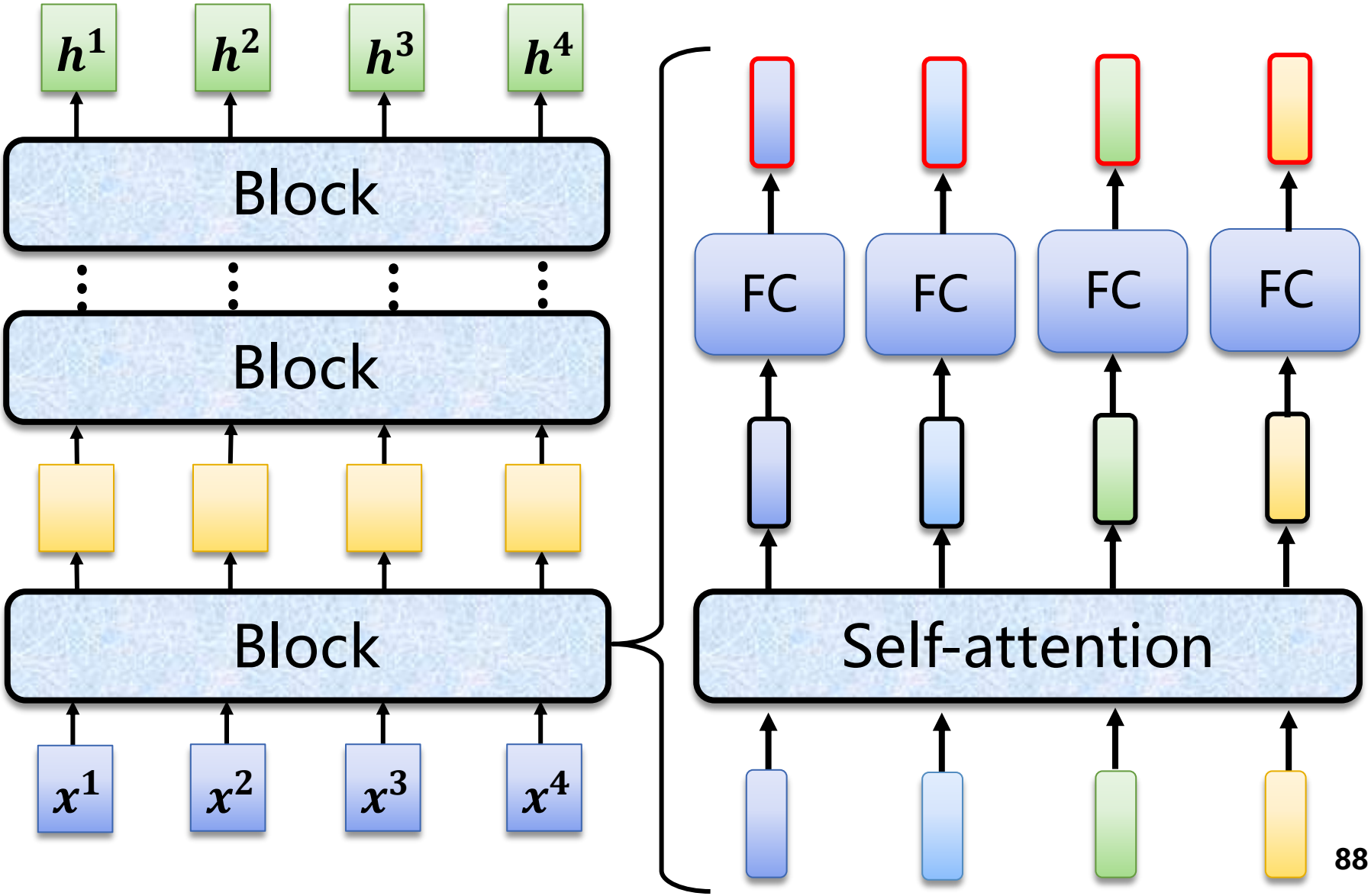
You can use **RNN** or **CNN**.



Transformer's Encoder

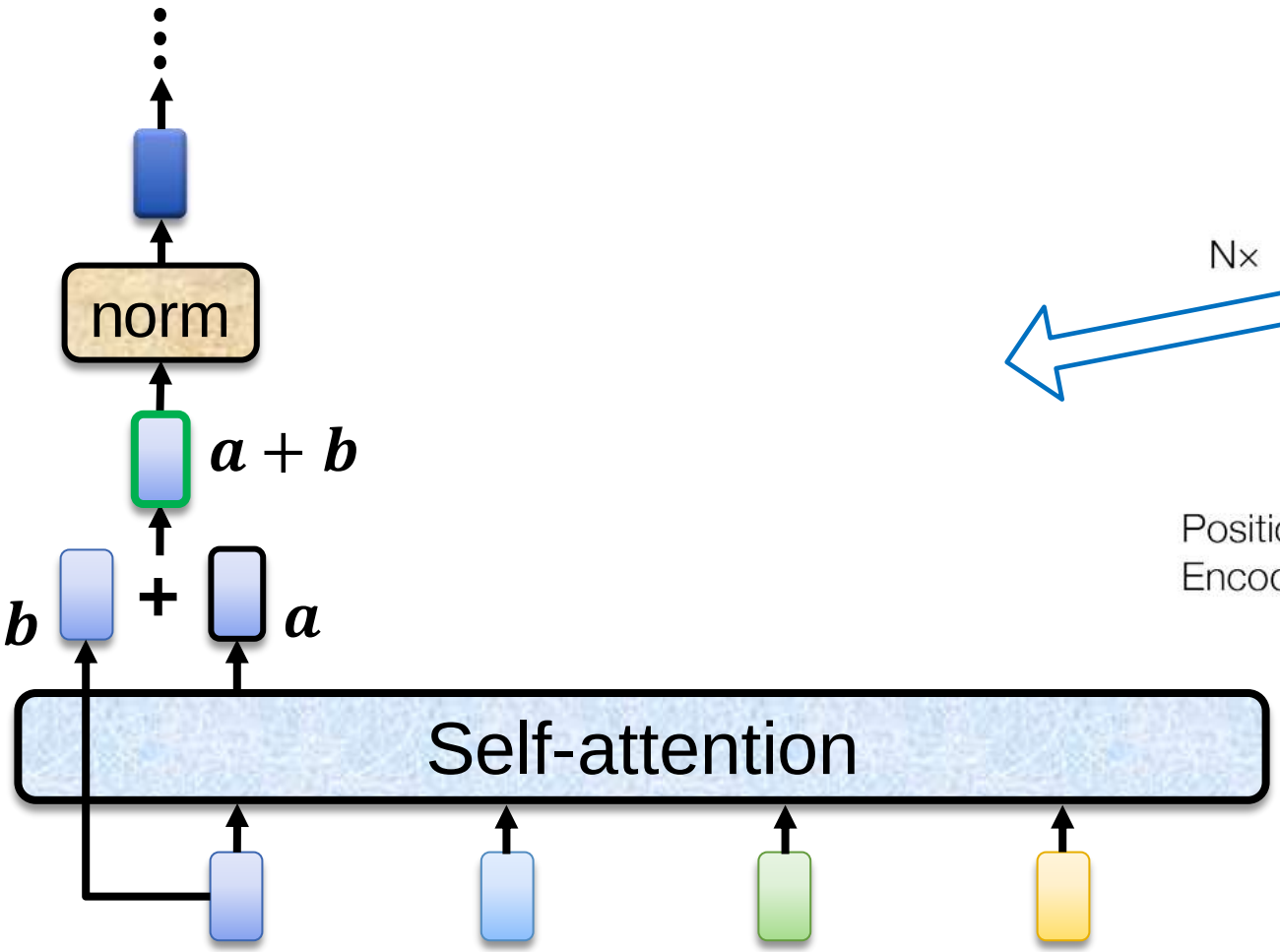


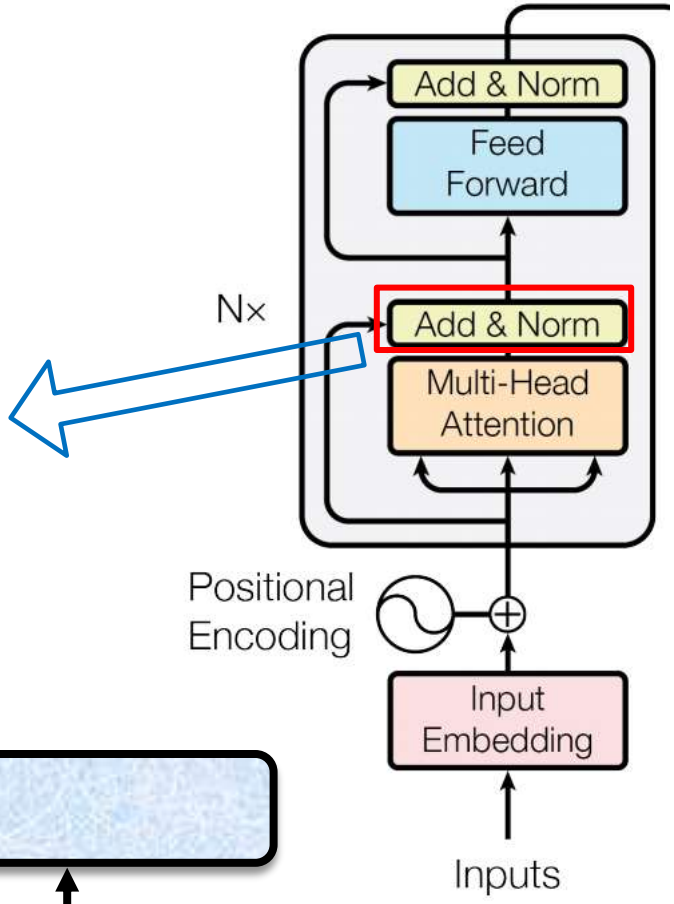
Transformer: Encoder



"Add & Norm"

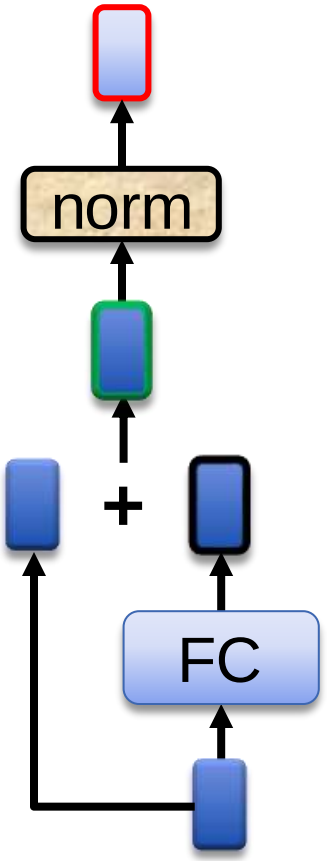
- "Add & Norm" layer denotes **Residual Connection** and **Layer Normalization**

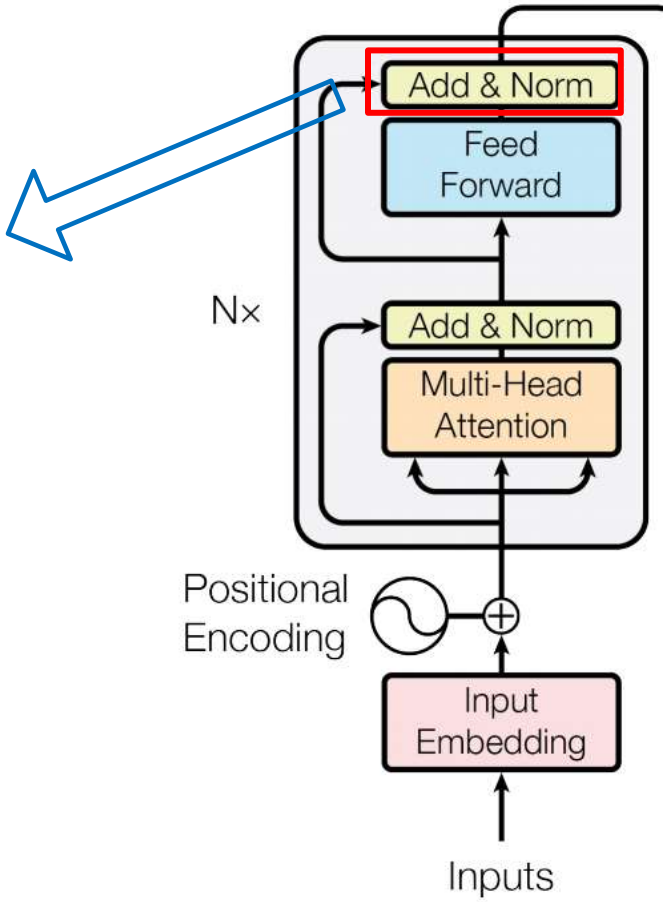




"Add & Norm"

- "Add & Norm" layer denotes **Residual Connection** and **Layer Normalization**

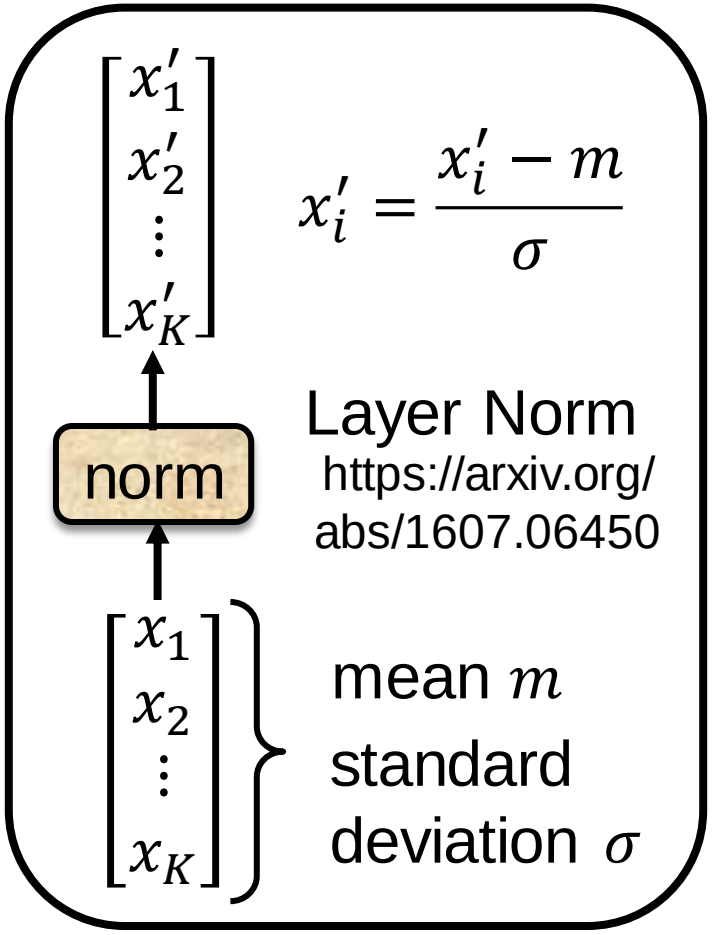




Layer Normalization

- The "Norm" part in the "Add & Norm" layer denotes **Layer Normalization**

- It **independently normalizes vector representation of each example in batch** - this is done to control "flow" to the next layer.
- It **improves convergence stability** and sometimes **even quality**.

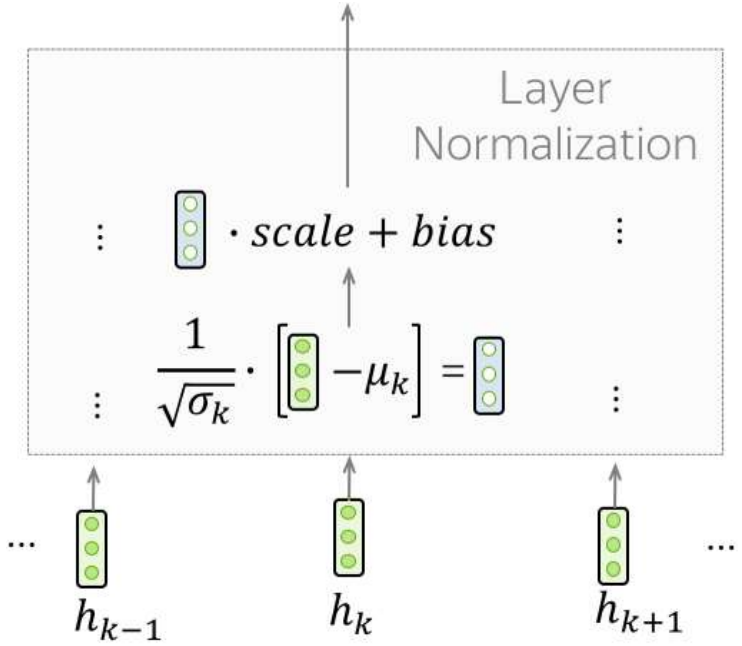


https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

Layer Normalization

- The "Norm" part in the "Add & Norm" layer denotes Layer Normalization

- In the Transformer, you have to **normalize vector representation of each token**.
- Additionally, **here LayerNorm has trainable parameters**, $scale$ and $bias$, which are **used after normalization to rescale layer's outputs** (or the next layer's inputs).
- Note that μ_k **and** σ_k **are evaluated for each example**, but $scale$ and $bias$ are the same - these are layer parameters.

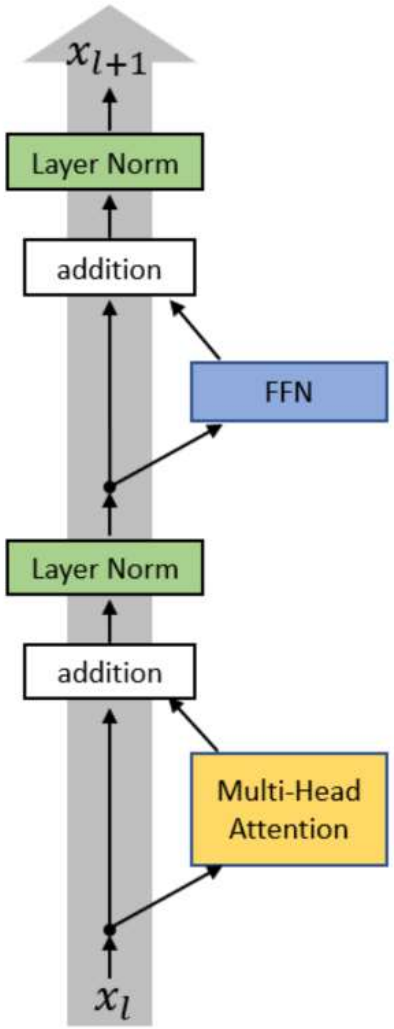


https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

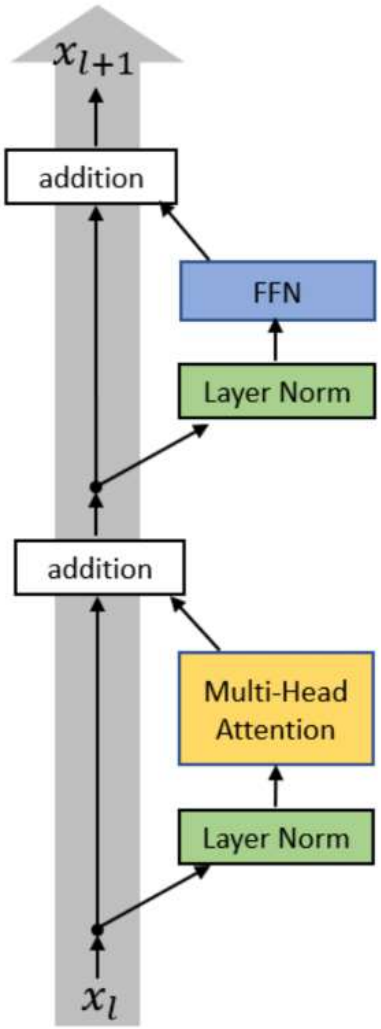
Layer Normalization

- The **location of the LayerNorm** remains a hotly **debated subject**

Post_LN Transformer layer



(a)



(b)

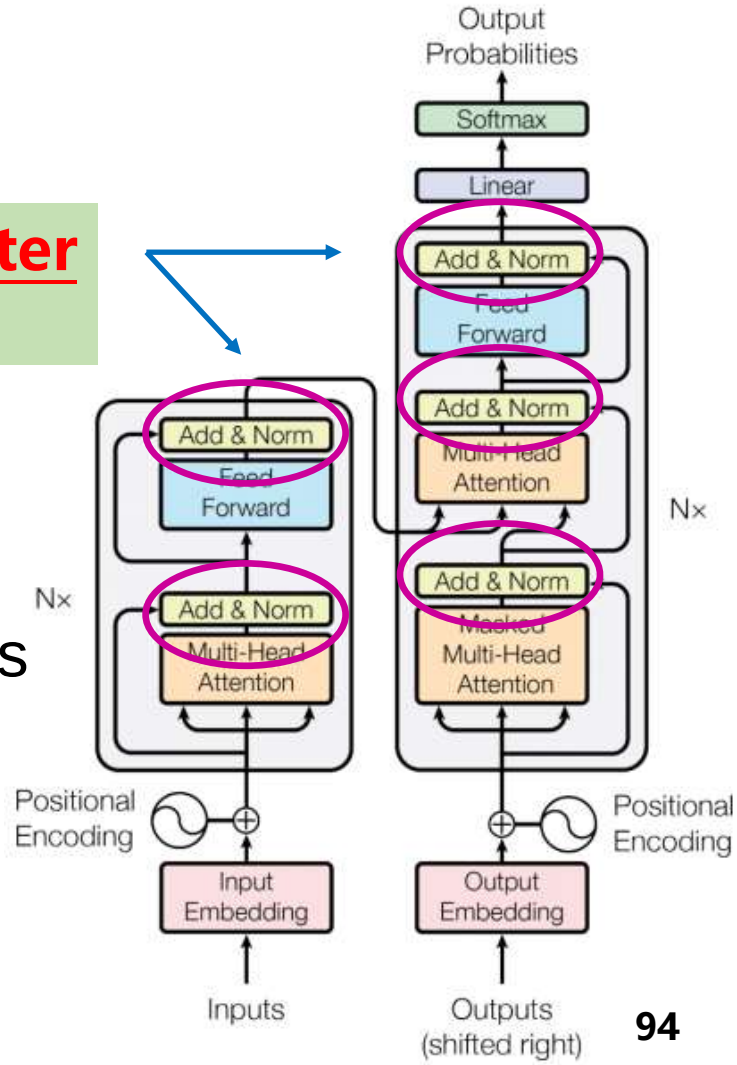
Pre_LN Transformer layer

Layer Normalization

- Post-LN Transformer

"Attention Is All You Need"
https://arxiv.org/abs/1706.03762

LayerNorms are **after** attention and **after** fully connected layers



- **It places the layer normalization between the residual blocks**: the expected gradients of the parameters near the output layer are large.
- It **doesn't match the official code implementation** accompanying the original transformer paper.

Layer Normalization

残差连接

对于残差$x + F(x)$，如果x的方差为σ_1^2而$F(x)$的方差为σ_2^2，并且假设二者相互独立，那么残差的方差即为$\sigma_1^2 + \sigma_2^2$，因此**残差的存在会进一步放大方差**，所以我们要采取相应的策略缩小此方差。

- 较为朴素的方案就是Post-LN，直接在残差后面运用Layer Normalization：$x_{t+1} = LN(x_t + F_t(x_t))$
- 但这种做法虽然稳定了前向传播的方差，但事实上已经**严重削弱了残差的恒等分支**，所以反而**失去了残差“易于训练”的优点**，通常要warmup并设置足够小的学习率才能使它收敛。

Layer Normalization

- Post-LN对残差的影响

假设初始状态下x 和$F(x)$的方差均为1，那么$x + F(x)$的方差就是2，而LN负责将方差降为1，这说明初始阶段的Post-LN相当于

$$x_{t+1} = \frac{x_t + F_t(x_t)}{\sqrt{2}}$$

递归下去，可以得到

$$x_l = \frac{x_{l-1}}{\sqrt{2}} + \frac{F_{l-1}(x_{l-1})}{\sqrt{2}}$$

$$= \frac{x_{l-2}}{2} + \frac{F_{l-2}(x_{l-2})}{2} + \frac{F_{l-1}(x_{l-1})}{\sqrt{2}}$$

$$= \cdots$$

$$= \frac{x_0}{2^{l/2}} + \frac{F_0(x_0)}{2^{l/2}} + \frac{F_1(x_1)}{2^{(l-1)/2}} + \frac{F_2(x_2)}{2^{(l-2)/2}} + \cdots + \frac{F_{l-1}(x_{l-1})}{2^{1/2}}$$

让梯度可以直接回传的“绿色通道”被严重削弱，越靠近前面的通道反而权重越小

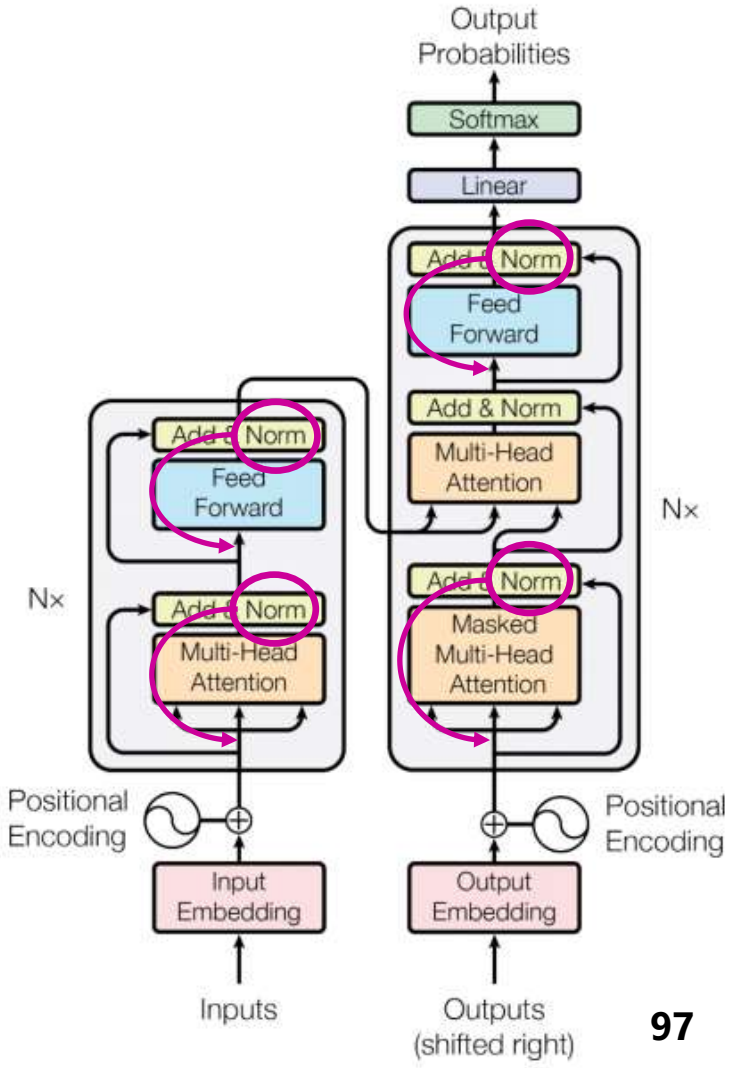
残差“名存实亡”！

Layer Normalization

- Pre-LN Transformer

"On Layer Normalization in the Transformer Architecture"
https://arxiv.org/abs/2002.04745

- **Better gradients** if LayerNorm is placed inside residual connection, **before** the attention and fully connected layers.
- Many architectures adopted this in practice, but **it can result in representation collapse**.



Layer Normalization

- Pre-LN对残差的影响

“要用的时候才去归一化”，其形式为

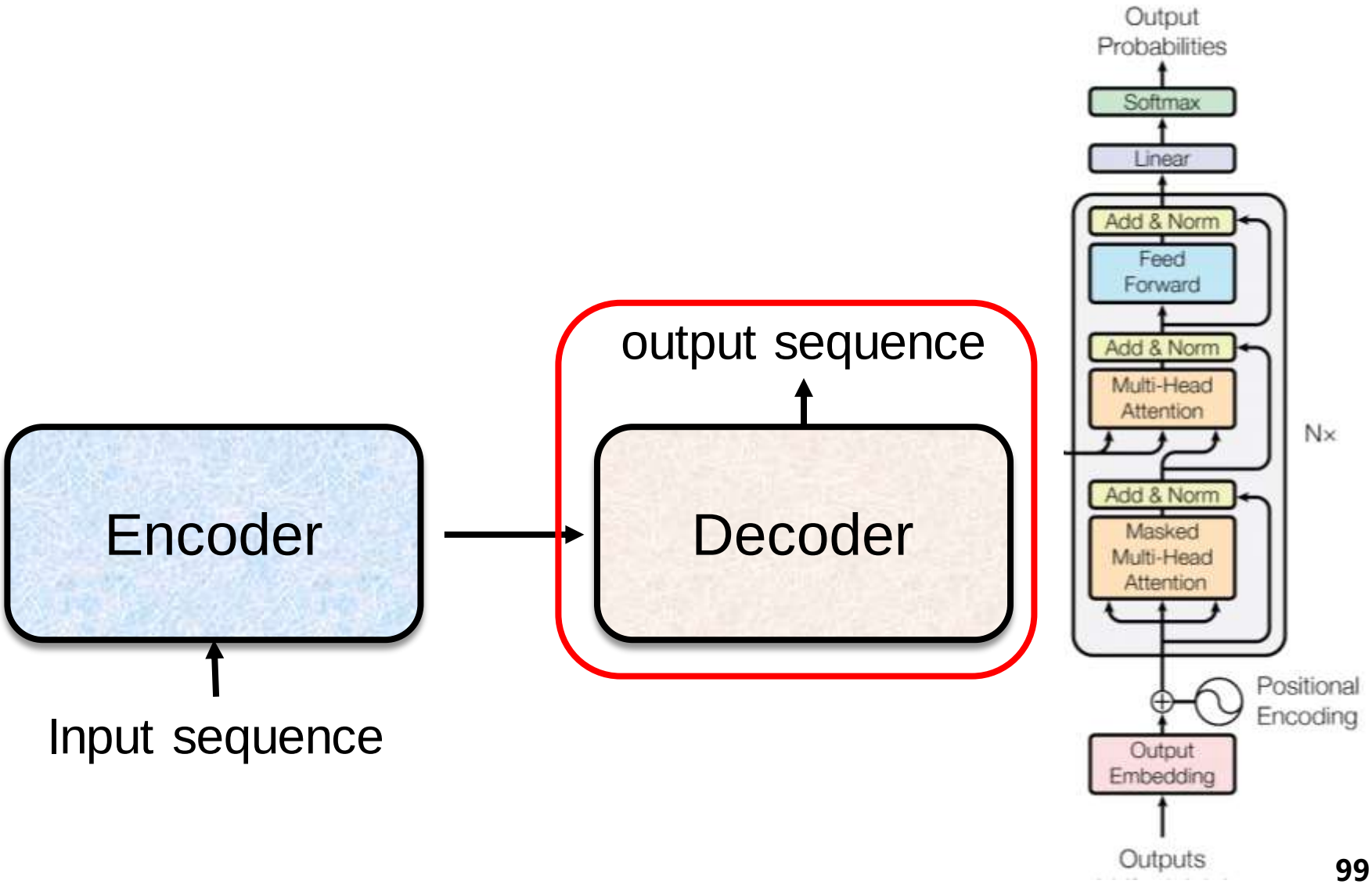
$$x_{t+1} = x_t + F_t(LN(x_t))$$

类似的迭代展开之后可以认为初始阶段有

$$x_l = x_0 + F_0(x_0) + F_1\left(x_1/\sqrt{2}\right) + F_2\left(x_2/\sqrt{3}\right) + \cdots + F_{l-1}(x_{l-1}/\sqrt{l})$$

- **每一条残差通道权重都是相等的**，残差的作用会比Post-LN更加明显，所以它也更好优化。
- 但**最后x_l的方差将会很大**，所以在预测层之前x_l也要进行LN。

Transformer: Decoder



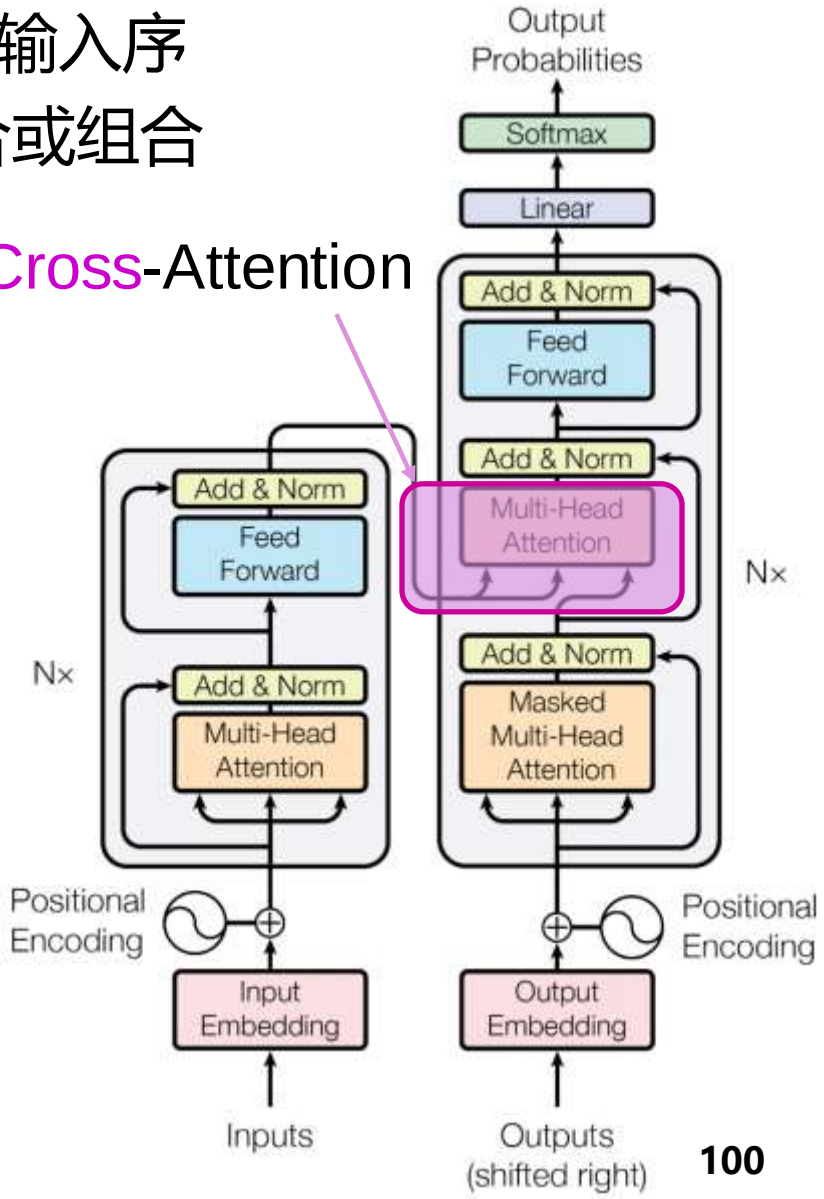
Cross-Attention

- 在自注意力中，我们使用相同的输入序列。 在交叉注意力中，我们混合或组合两个不同的输入序列。

在原始 Transformer 架构的情况下，分别是左侧编码器模块返回的序列和右侧解码器部分正在处理的输入序列。

猜测下：紫色框中输入与输出部分交汇的是query、key和value中的哪两个?

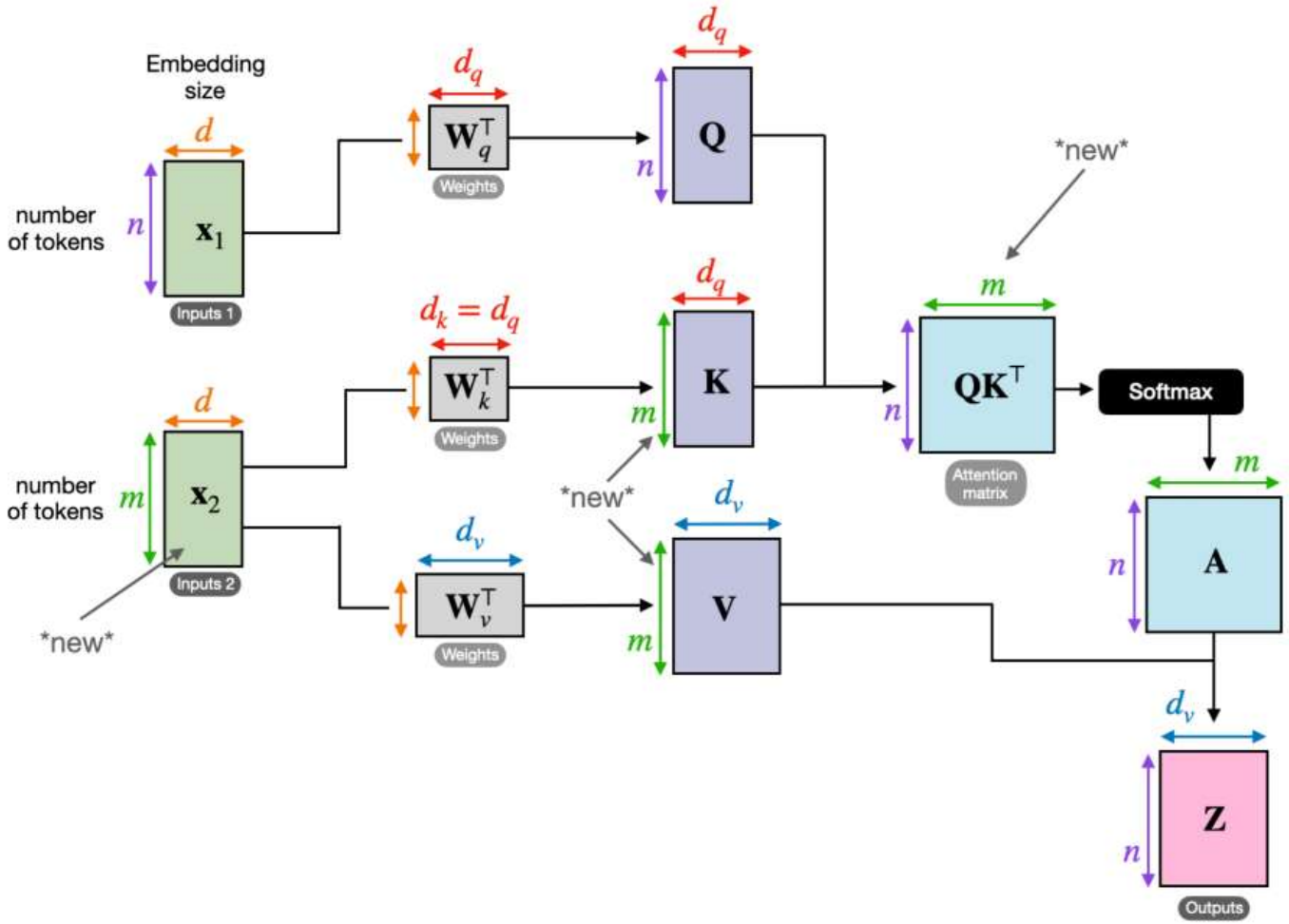
正确答案： **key和value**!



Cross-Attention

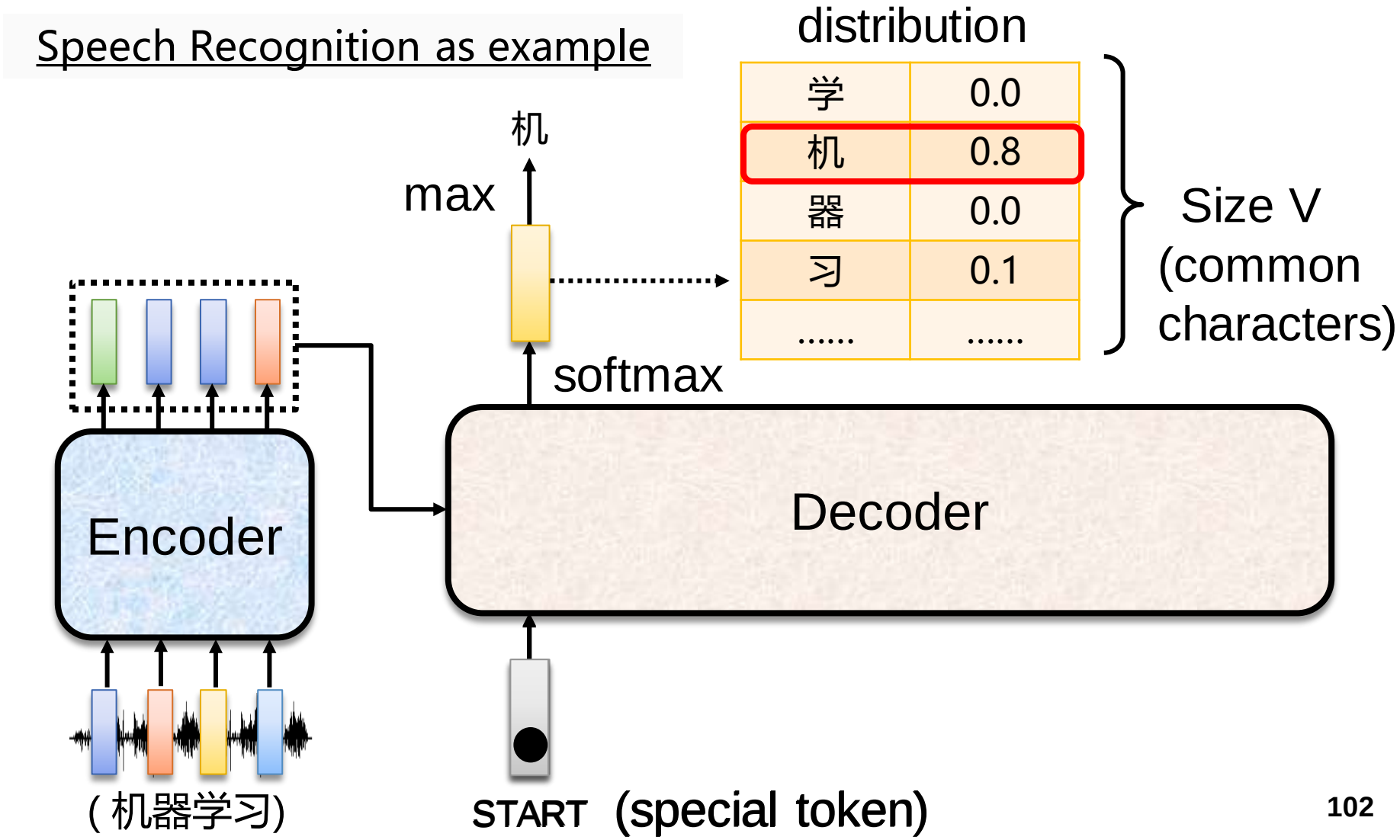
- 概念与流程

$\mathbf{x}_1 = \mathbf{x}_2$时即为self-attention



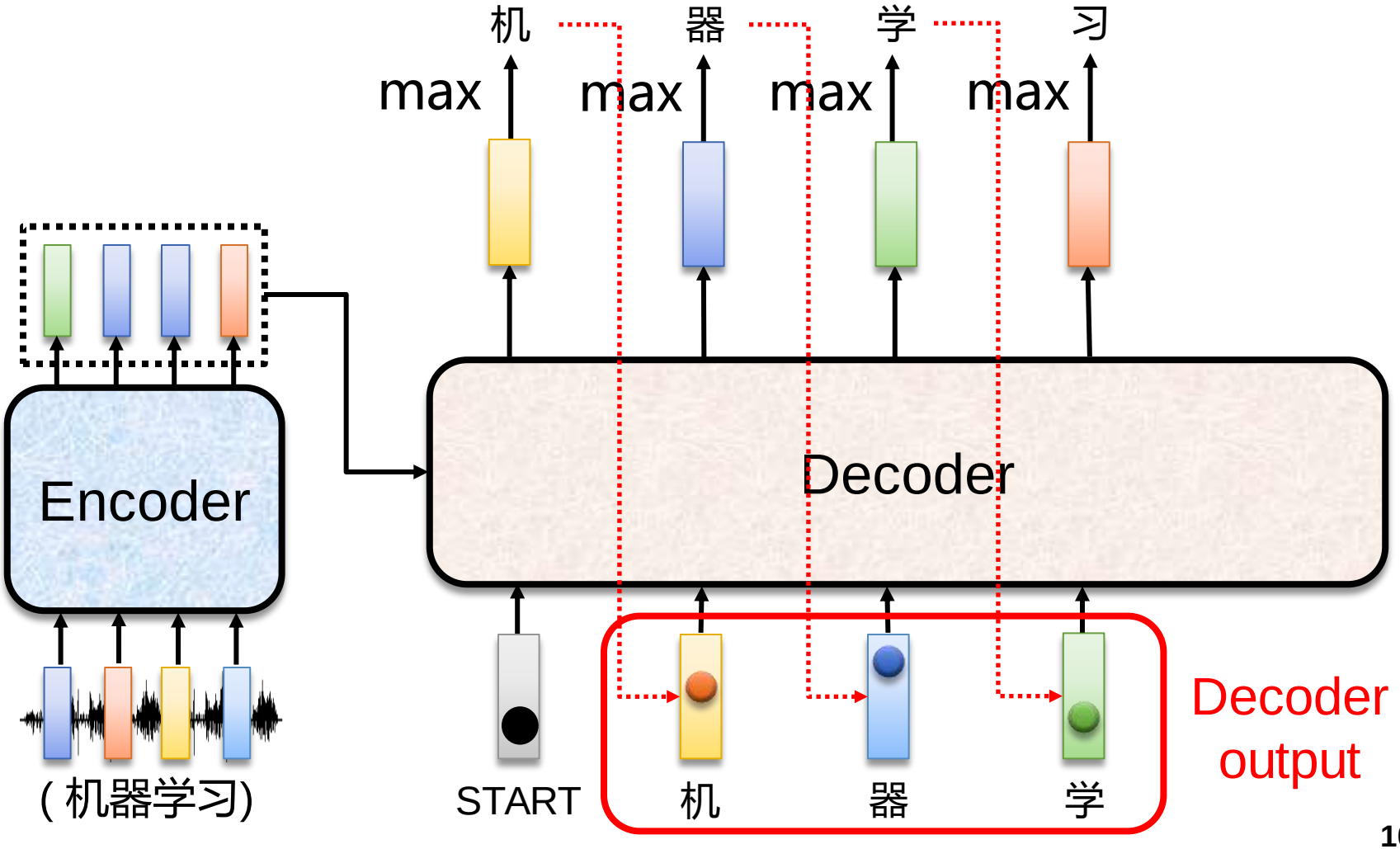
Decoder：Autoregressive Part

- 解码时不仅依靠编码器输出还依靠已解码部分

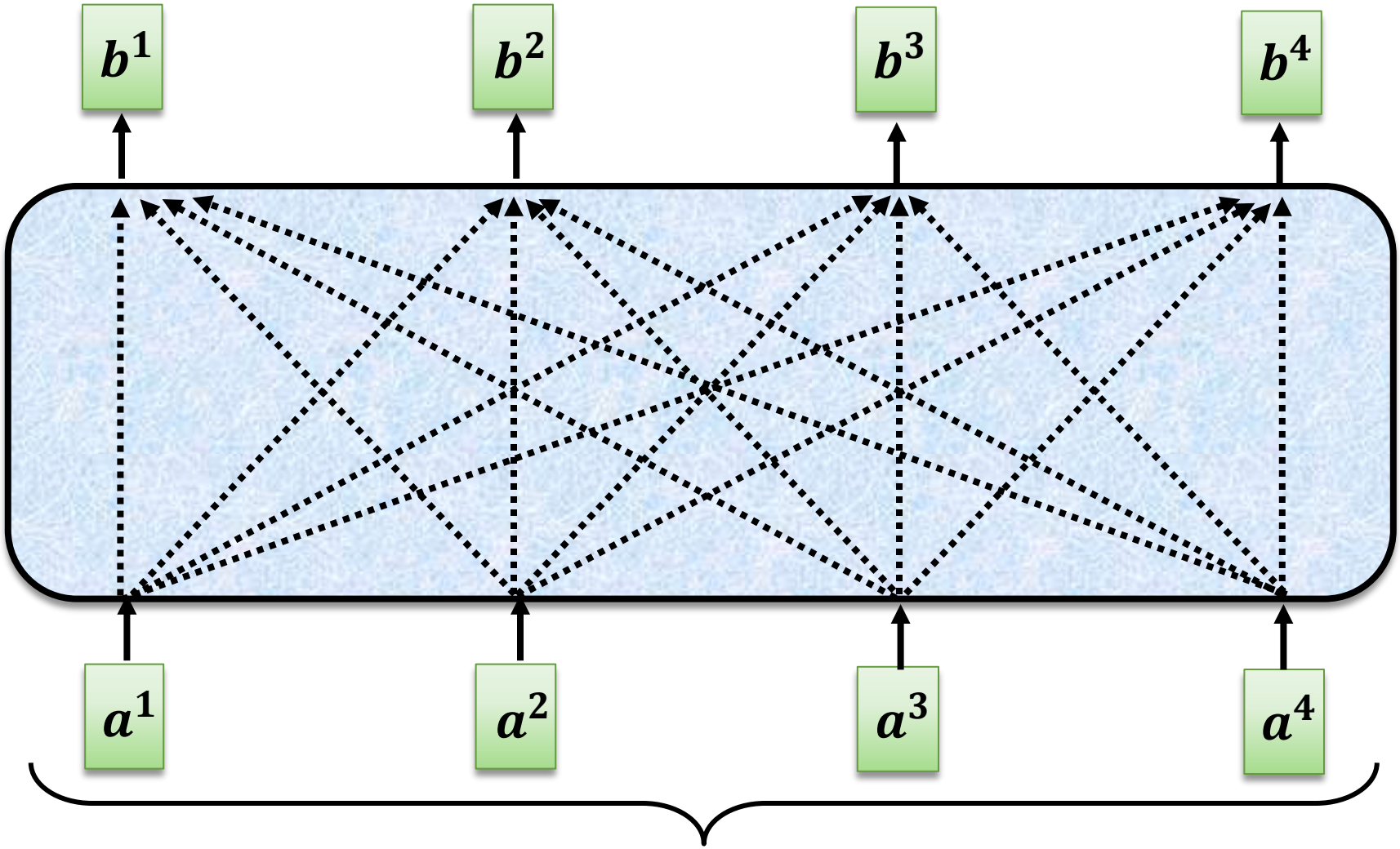


Decoder：Autoregressive Part

- 解码时不仅依靠编码器输出还依靠已解码部分

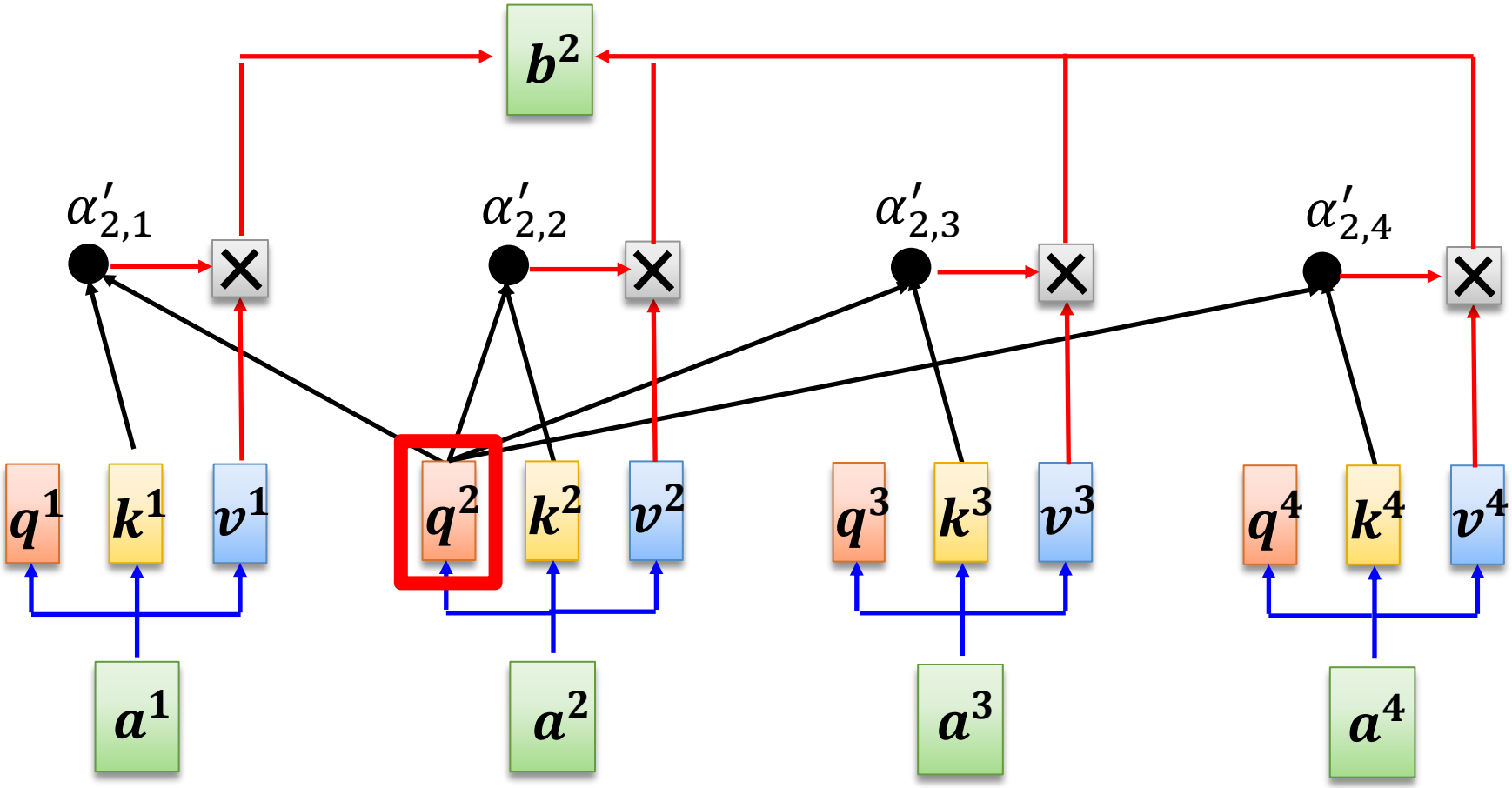


Decoder: Masked Self-attention



Can be either **input** or **a hidden layer**

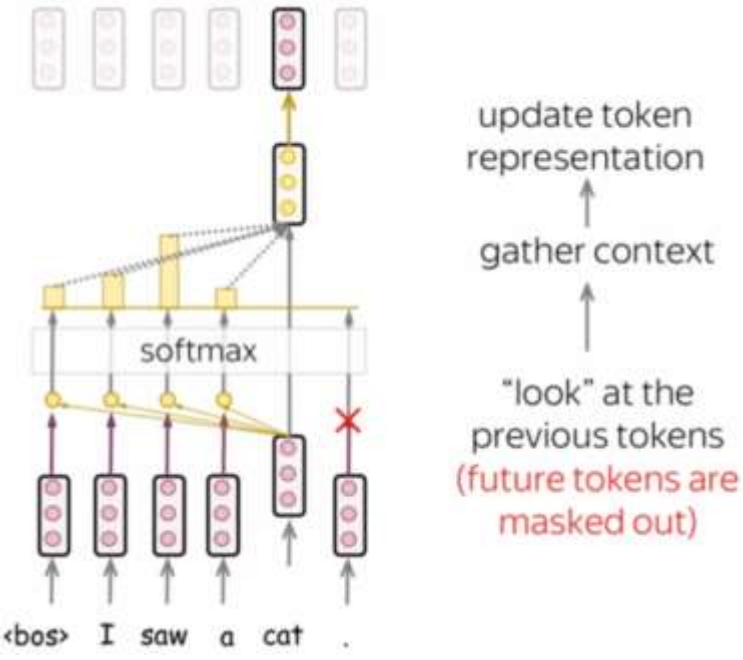
Decoder: Masked Self-attention



Why masked? Consider how does decoder work

Decoder： Masked Self-attention

- 解码器“不要向前看”
  - 在解码器中，我们一次生成一个token。在生成过程中，我们不知道将来会生成哪些token。
  - 在训练中，我们将整个目标句子提供给解码器，**没有mask，标记将“看到未来”**。
  - 这样做可以**提高计算效率**：Transformer 没有递归，因此可以立即处理所有令牌。

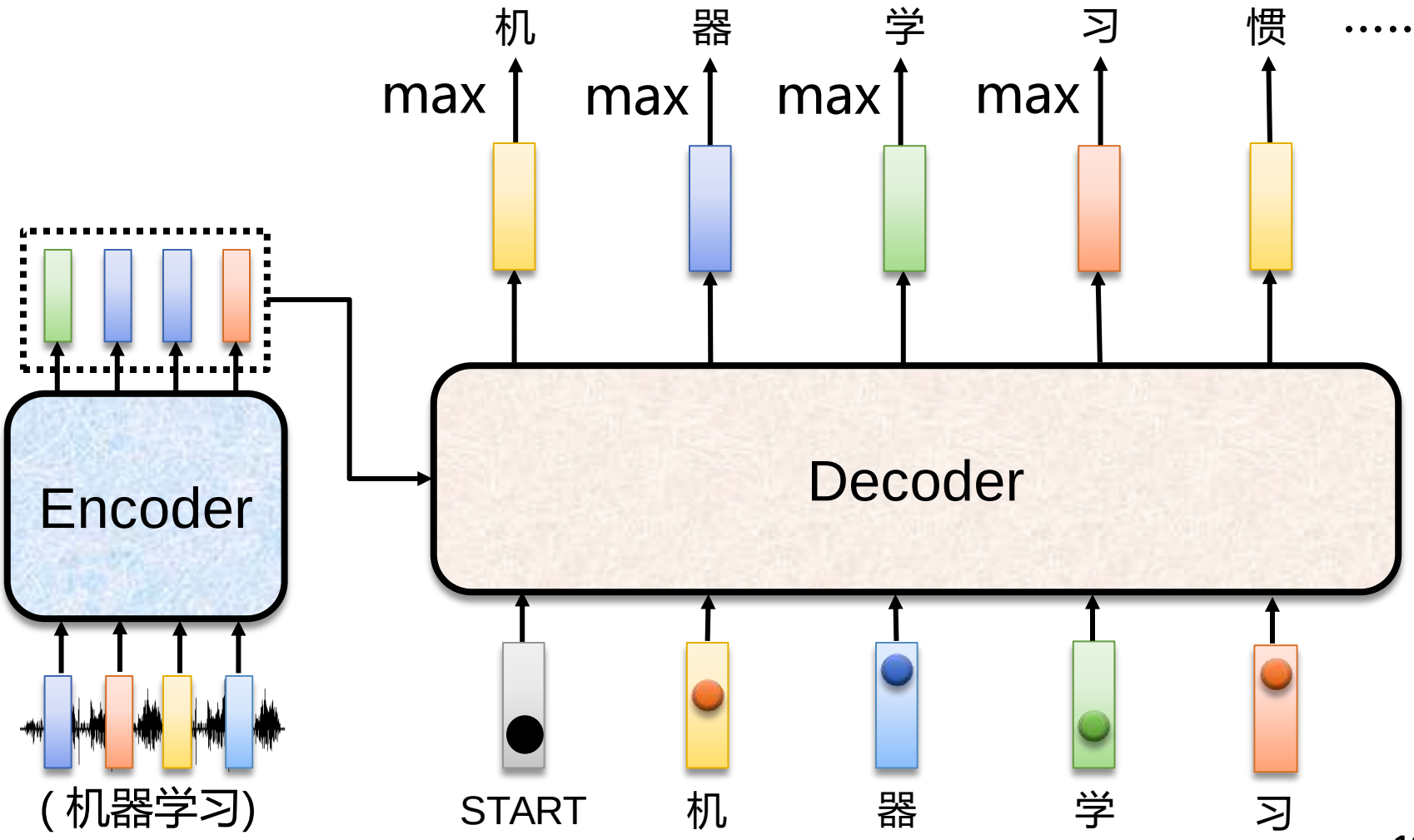


对于循环模型，一个训练步骤需要O(len(source) + len(target))步骤，但对于 Transformer 来说，它是O(1)，即常数。

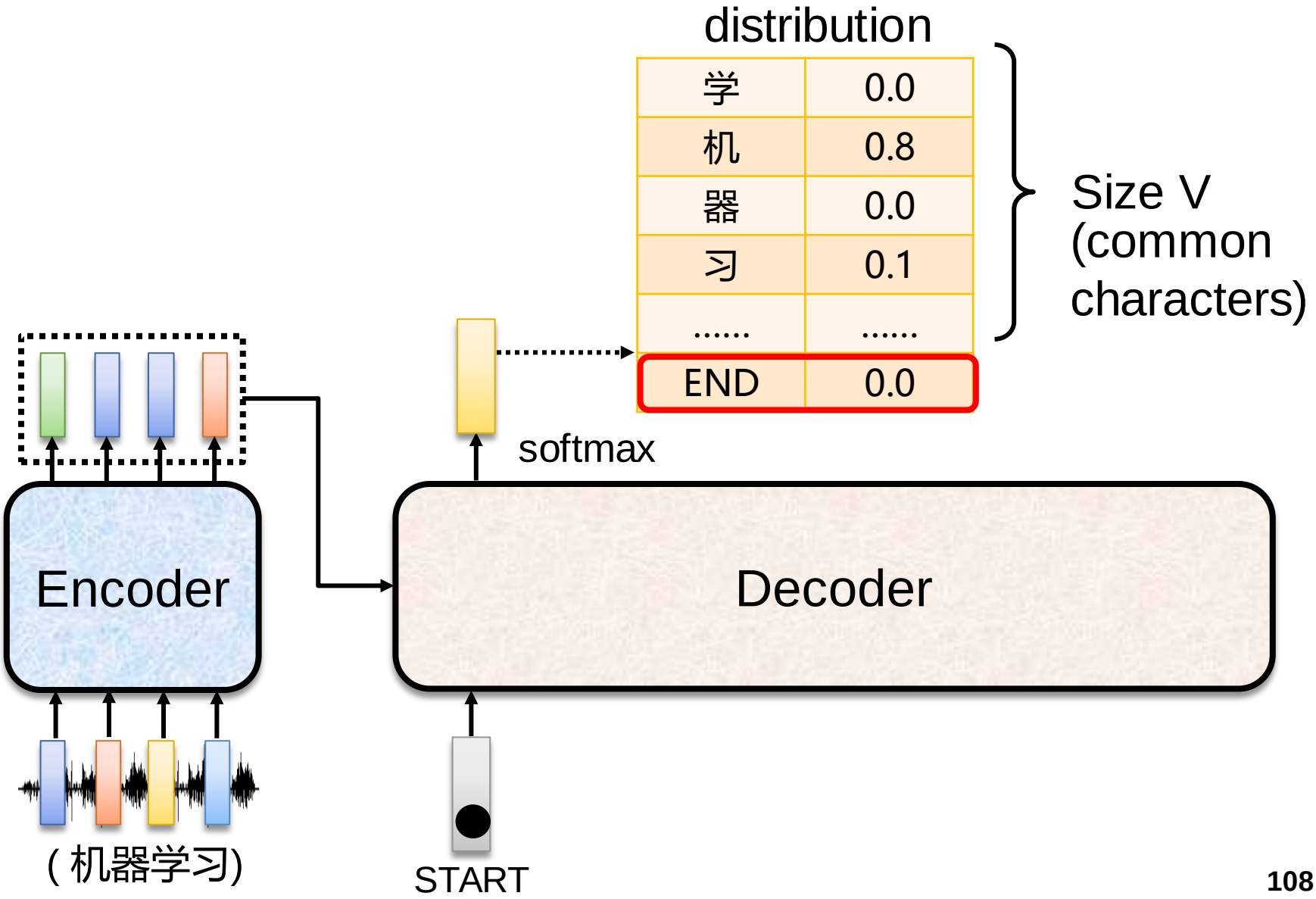
Decoder: Adding "Stop Token"

- We do not know the correct output length.

Never stop!

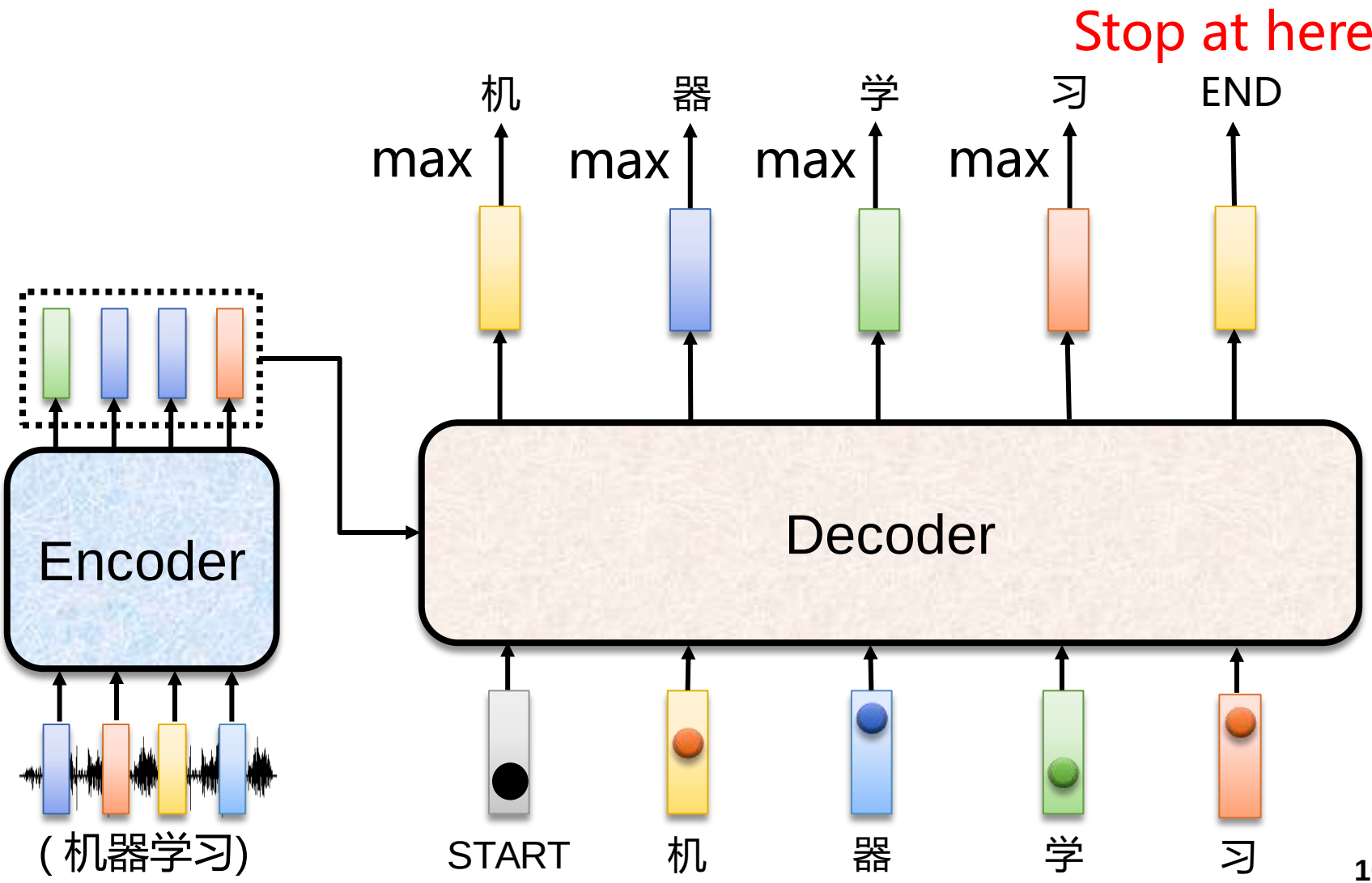


Decoder: Adding "Stop Token"



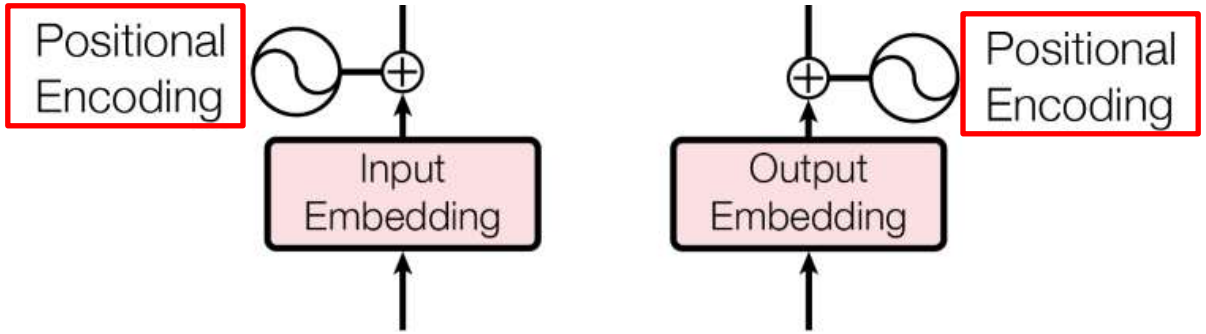
Decoder: Adding "Stop Token"

- We do not know the correct output length.



Positional Encoding

- 原因：**self-attention中没有位置信息**
  - Transformers 一次性获取所有输入的词嵌入，尽管这有助于运行地更快，但会丢失与词序相关的信息。



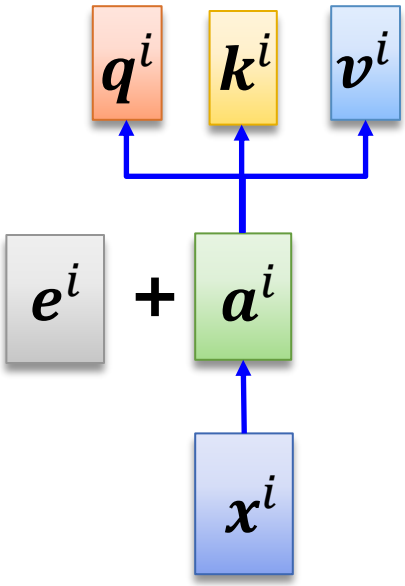
- 做法：利用**波频率来捕获位置信息**

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

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

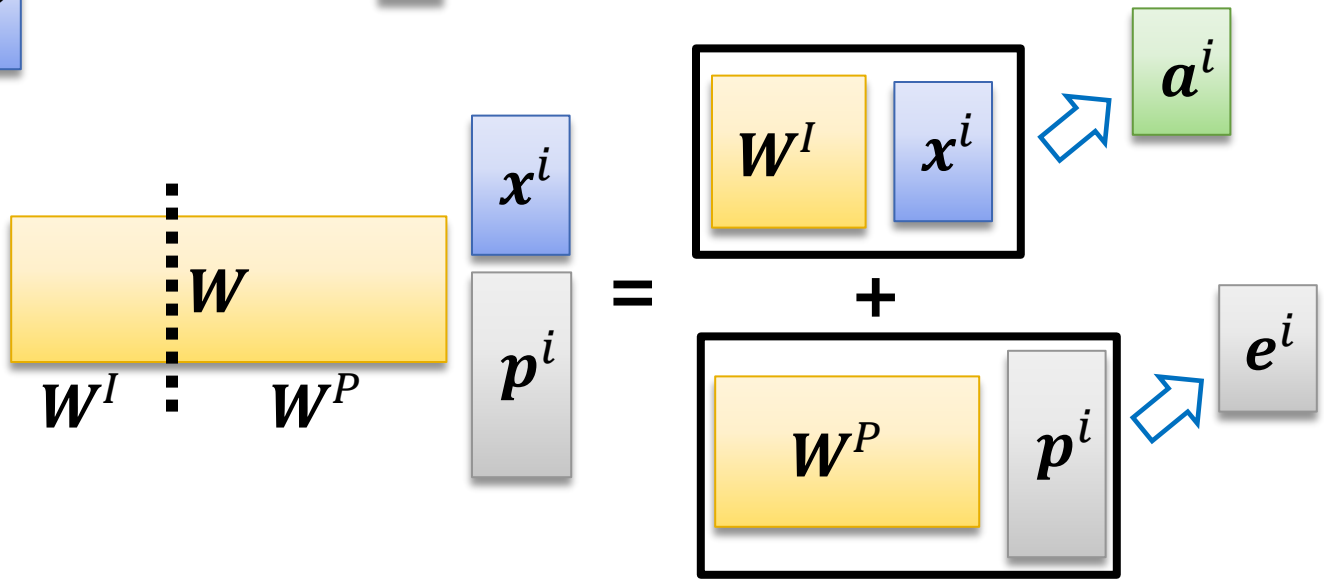
Positional Encoding

- 矩阵表示



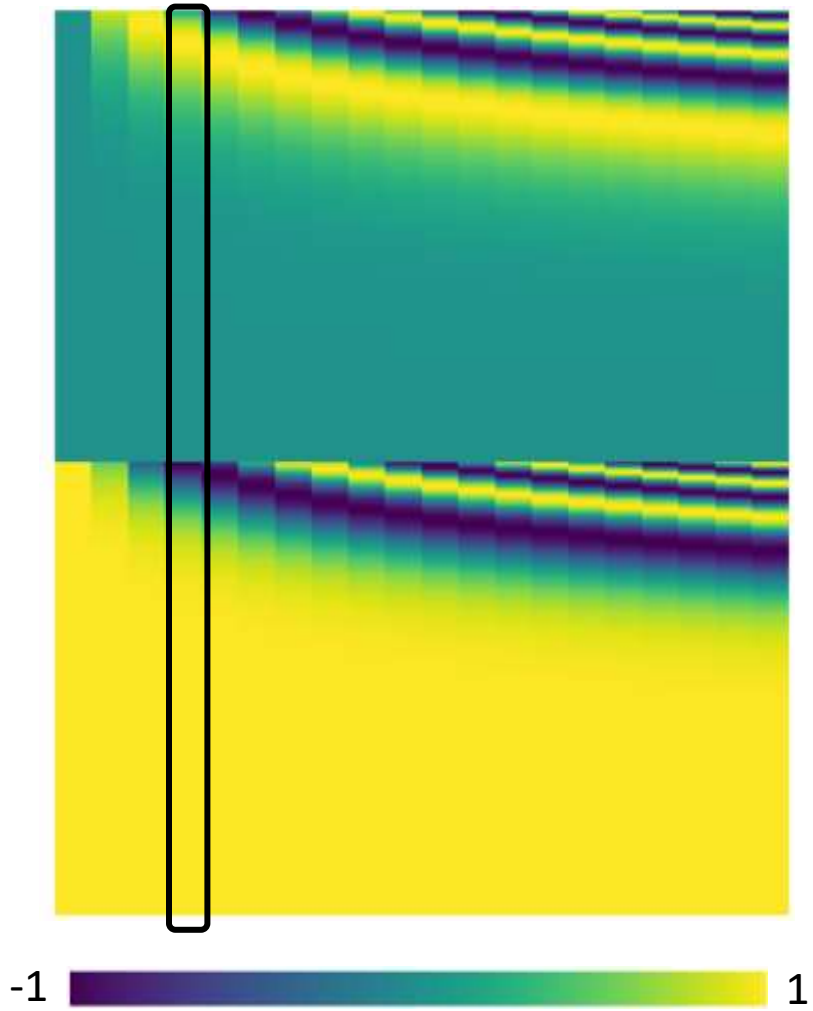
$$p^i = \begin{bmatrix} \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix} \leftarrow \text{第}i\text{个维度}$$

每个位置都有一个唯一的位置向量e^i，即每个x^i都增添一个one-hot向量p^i



Positional Encoding

- 每一列都表示一个位置向量e^i

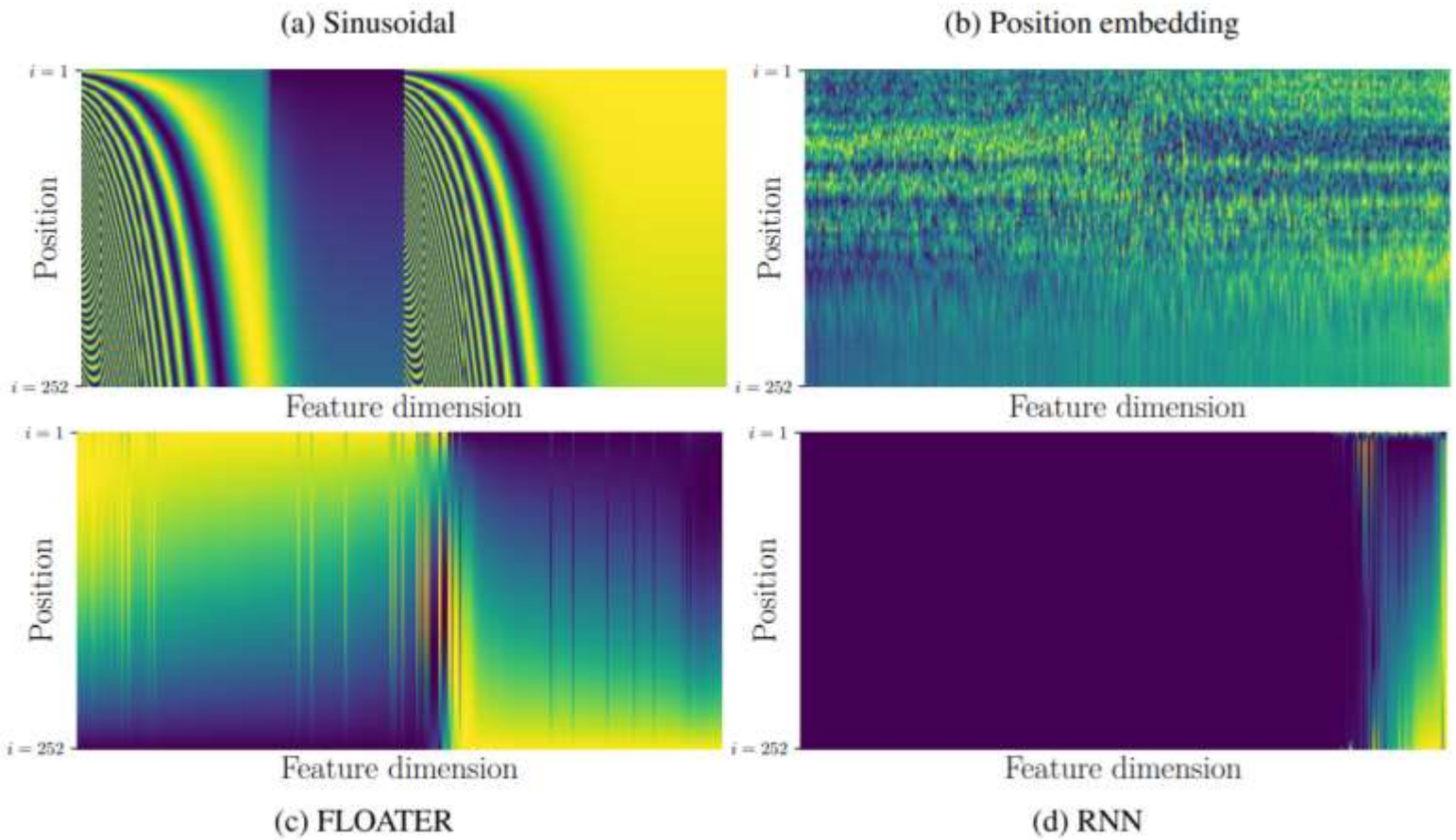


Positional Encoding

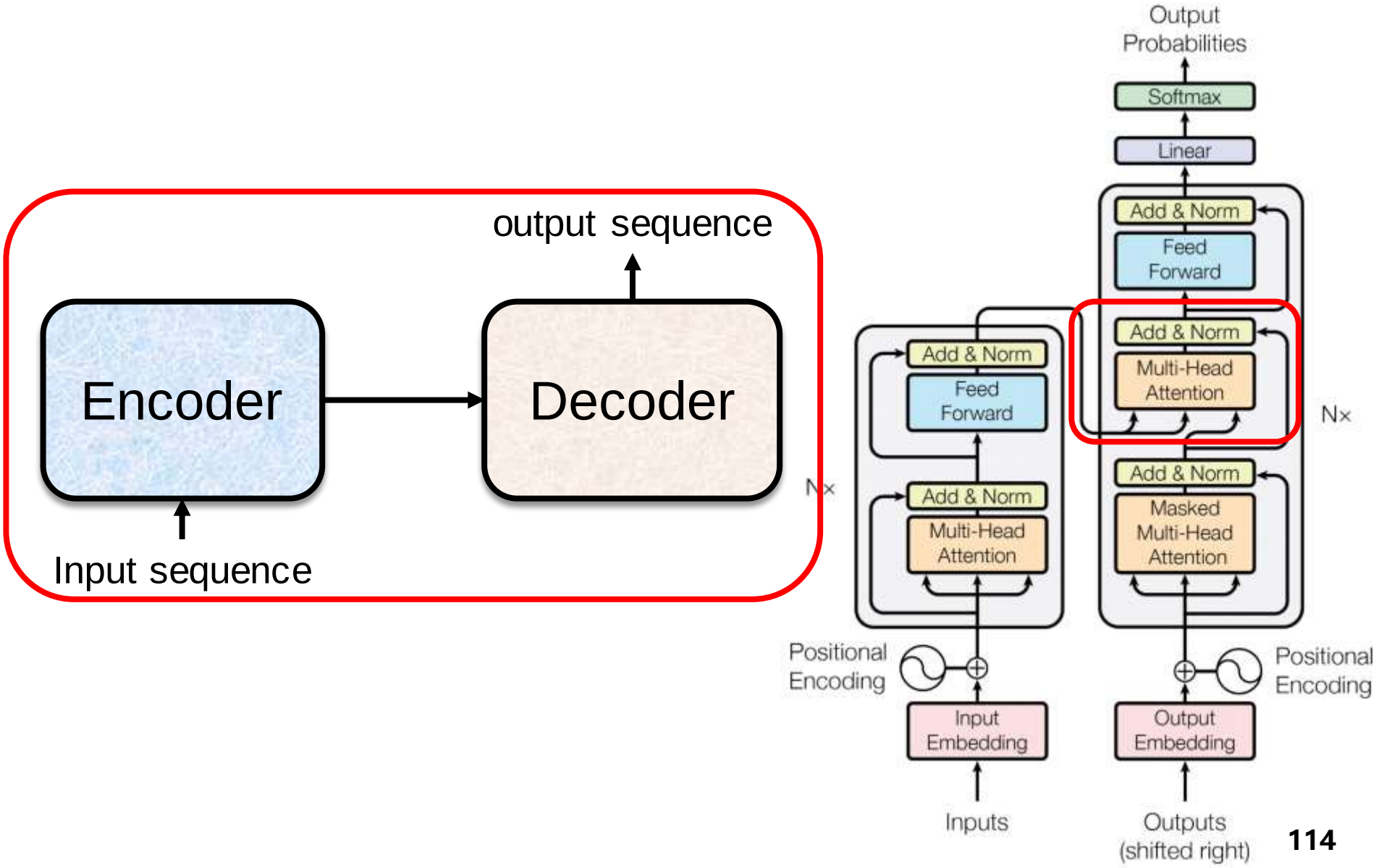
https://arxiv.org/abs/2003.09229

Table 1. Comparing position representation methods

Methods	Inductive	Data-Driven	Parameter Efficient
Sinusoidal (Vaswani et al., 2017)	✓	✗	✓
Embedding (Devlin et al., 2018)	✗	✓	✗
Relative (Shaw et al., 2018)	✗	✓	✓
This paper	✓	✓	✓



Transformer: Encoder-Decoder



Transformer: Encoder-Decoder

- Routing source representations

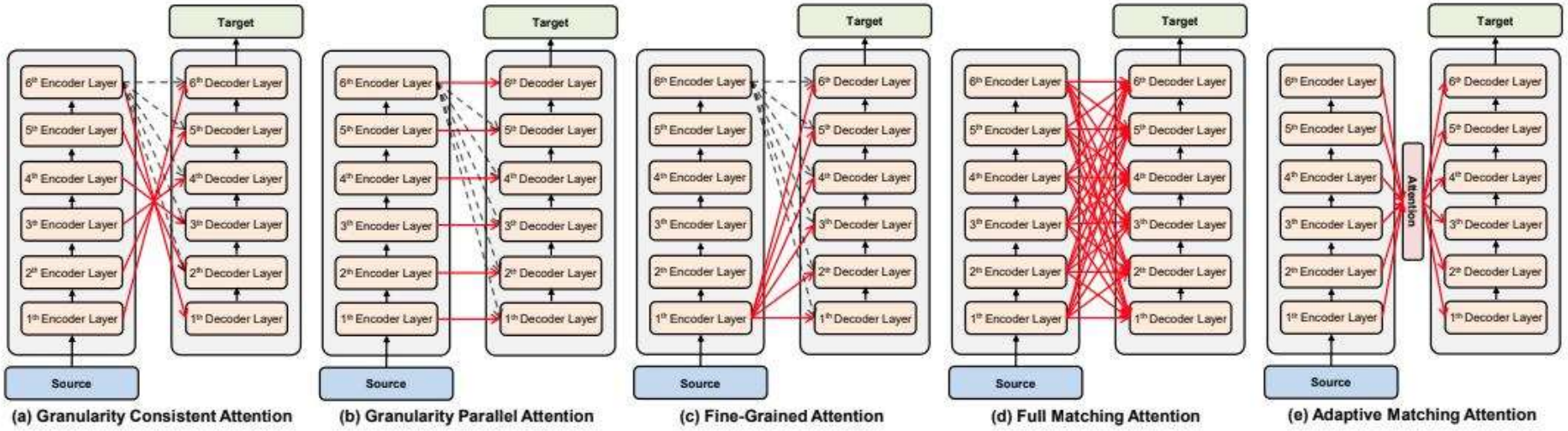
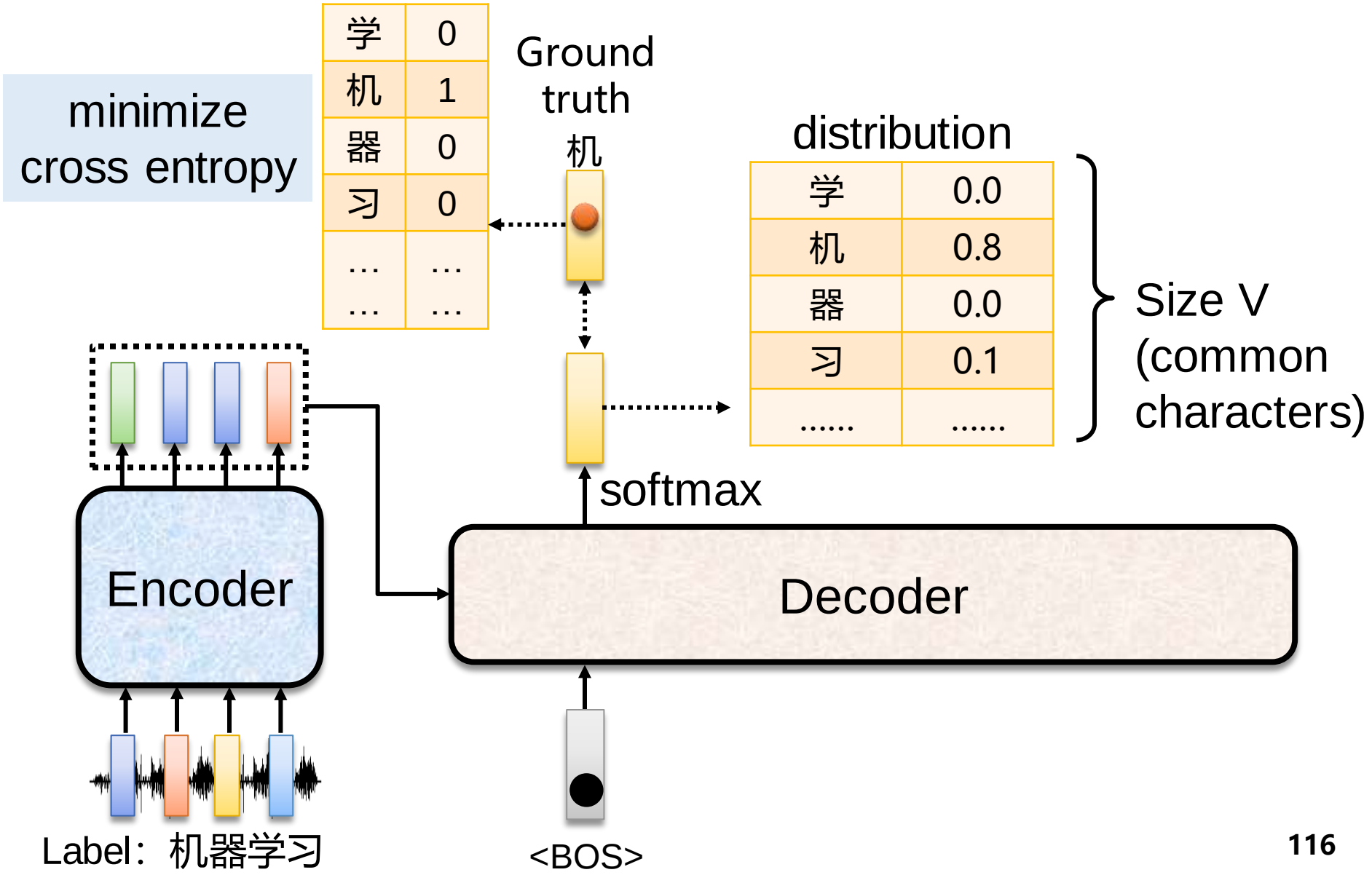
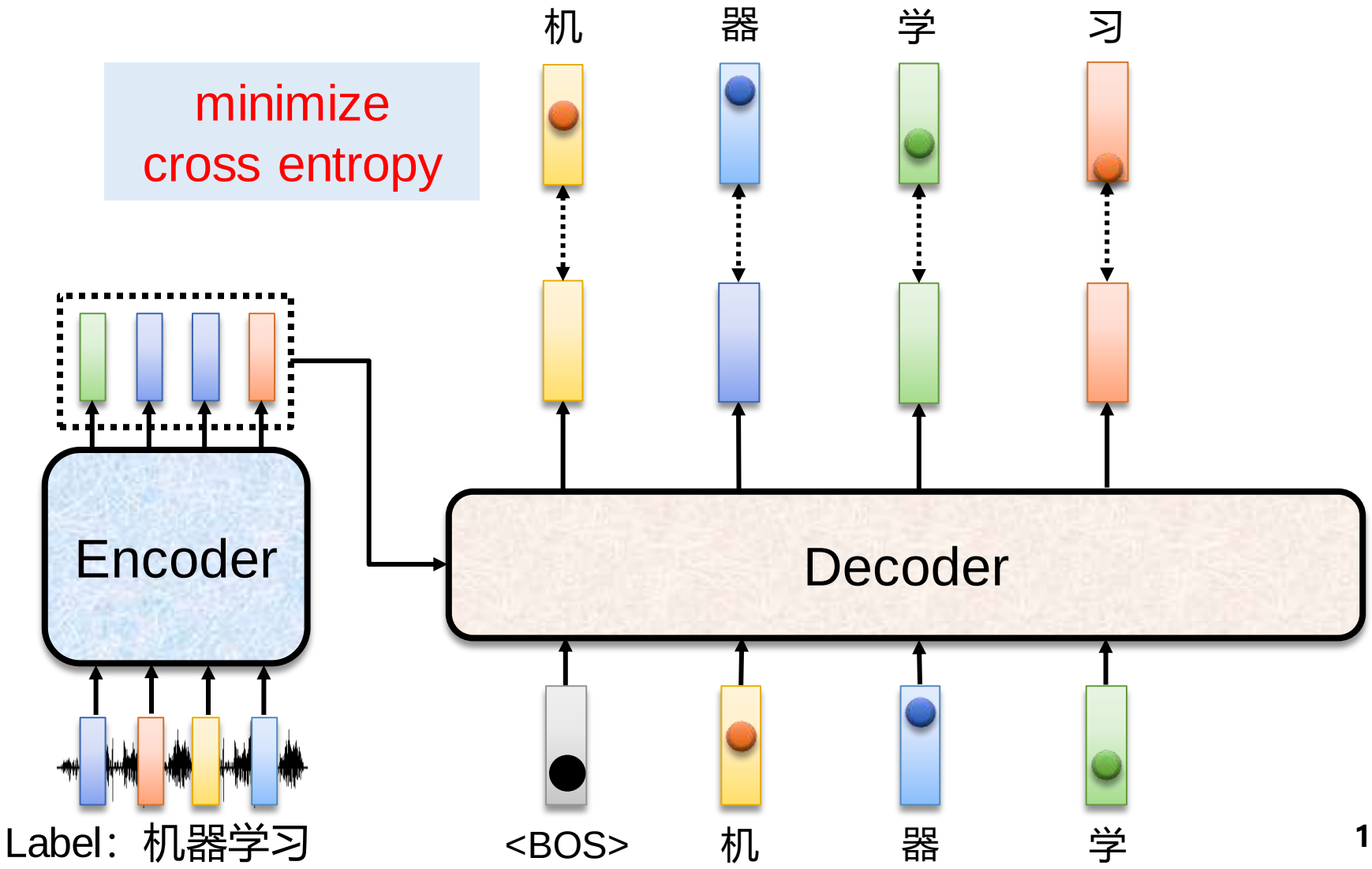


Figure 2: We present the proposal on Transformer with various strategies for routing the source representations: (a) Granularity Consistent Attention; (b) Granularity Parallel Attention; (c) Fine-Grained Attention; (d) Full Matching Attention; (e) Adaptive Matching Attention. The dashed lines represent the original attention to the last encoder layer and we omit them in (e) for clarity.

Transformer: Training



Transformer: Training



Transformer工作流程动画

- The illustration from the **Google AI blog post** introducing Transformer

请同学讲述其含义

内容导览

- 序列到序列(Seq2Seq)
- 循环神经网络(RNN)
- 注意力机制的基本原理
- 自注意力/多头注意力/交叉注意力
- Transformer
- ELMO/BERT/GPT

ELMO

1-of-N Encoding

apple = [1 0 0 0 0]

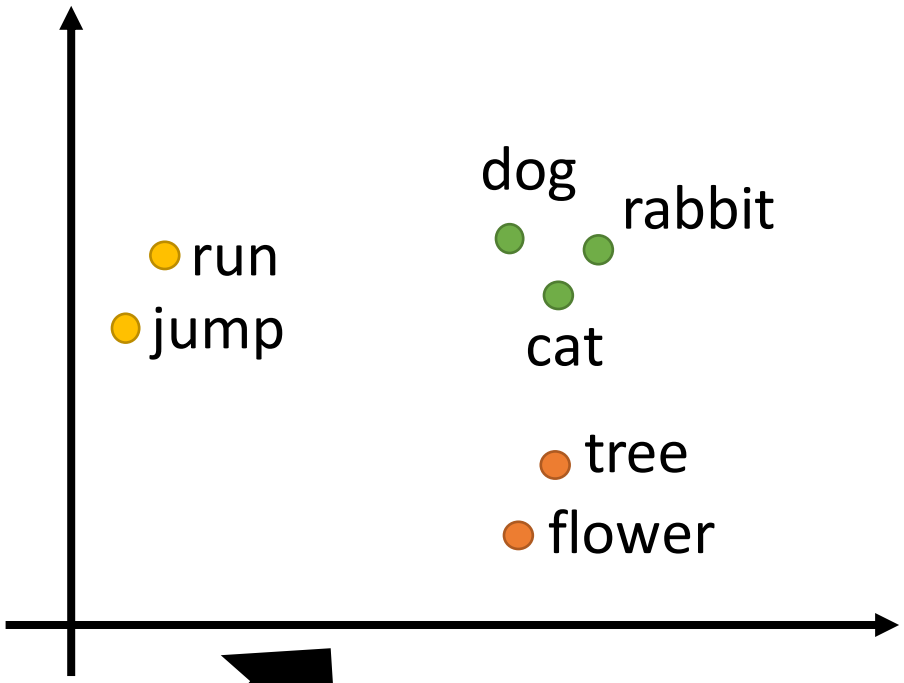
bag = [0 1 0 0 0]

cat = [0 0 1 0 0]

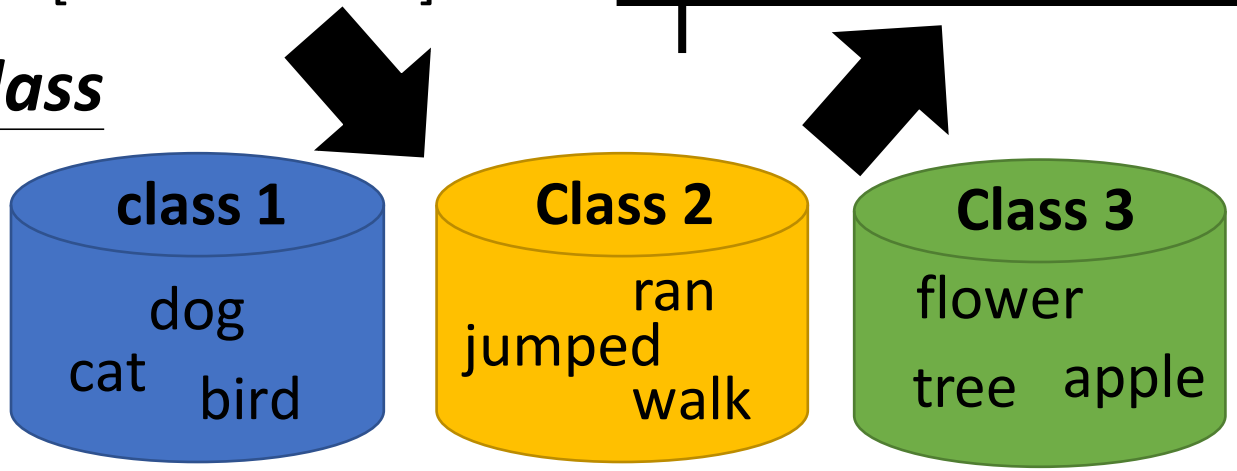
dog = [0 0 0 1 0]

elephant = [0 0 0 0 1]

Word Embedding



Word Class



ELMO

- 同一个单次有多种语义

Have you paid that money to the bank yet ?

It is safest to deposit your money in the bank .

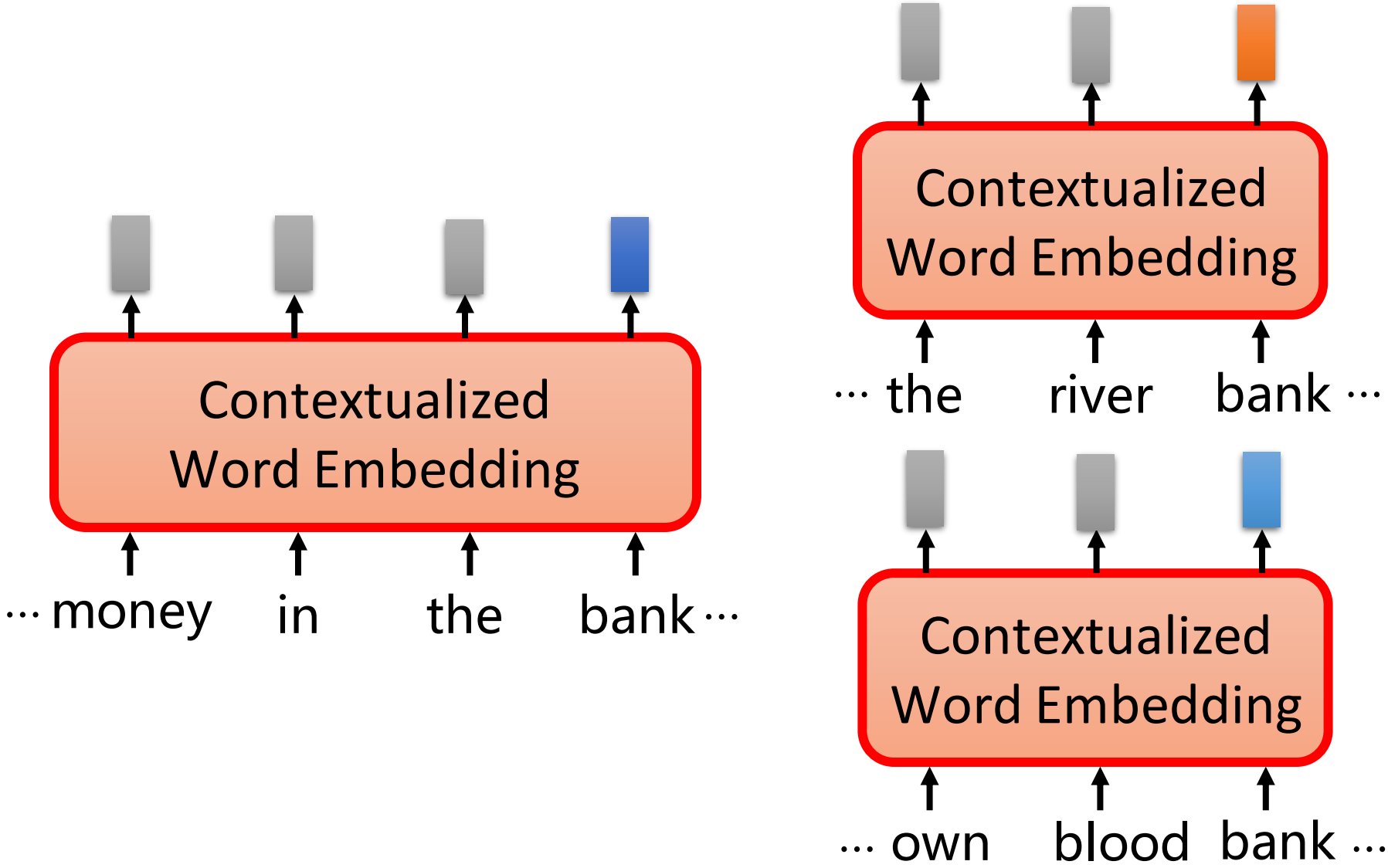
The victim was found lying dead on the river bank .

They stood on the river bank to fish.

The hospital has its own blood bank.

The third sense or not?

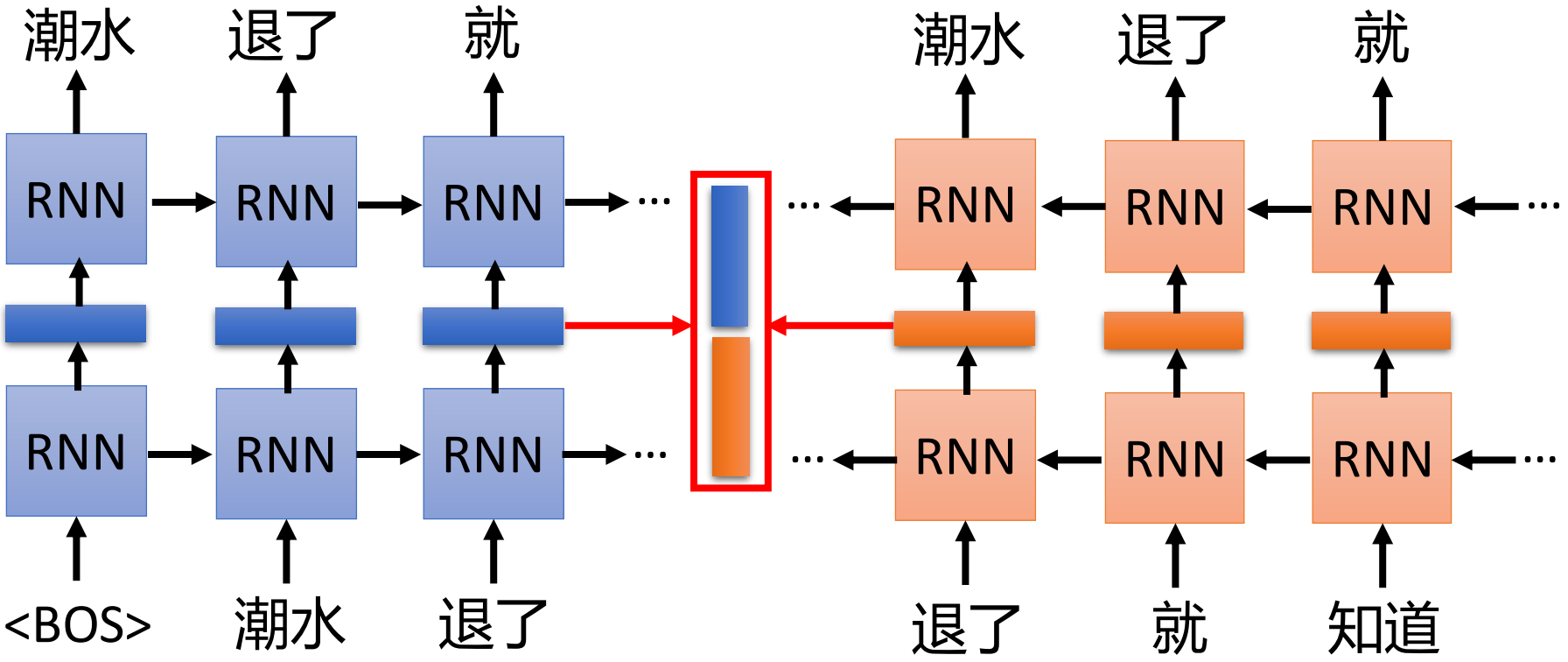
Contextualized Word Embedding



Embeddings from Language Model (ELMO)

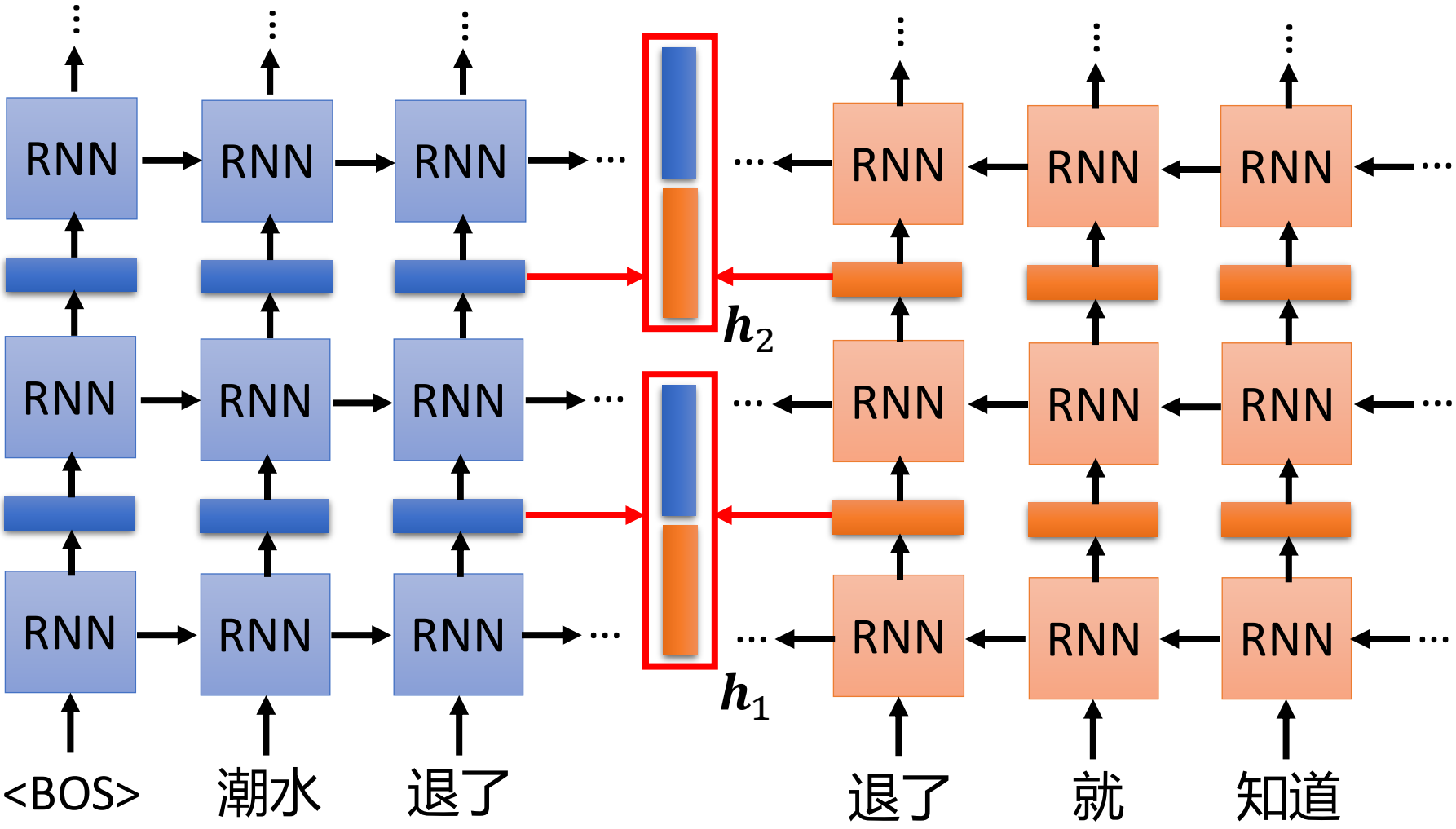
- RNN-based language models (trained from lots of sentences)

e.g. given “潮水 退了 就 知道 xxxx”

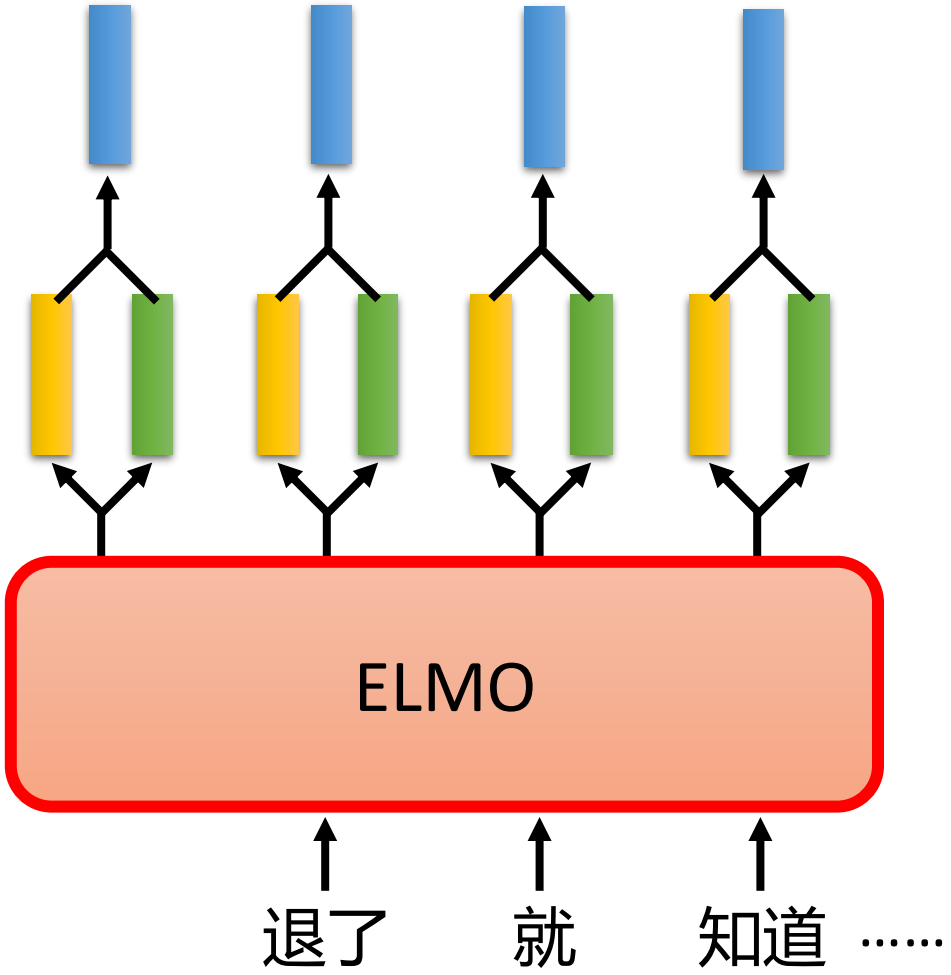


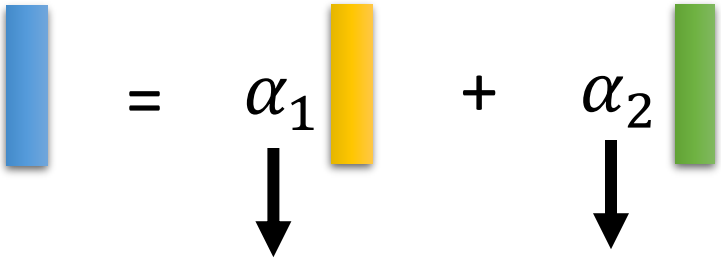
Embeddings from Language Model (ELMO)

Each layer in deep RNN can generate a latent representation.



Embeddings from Language Model (ELMO)

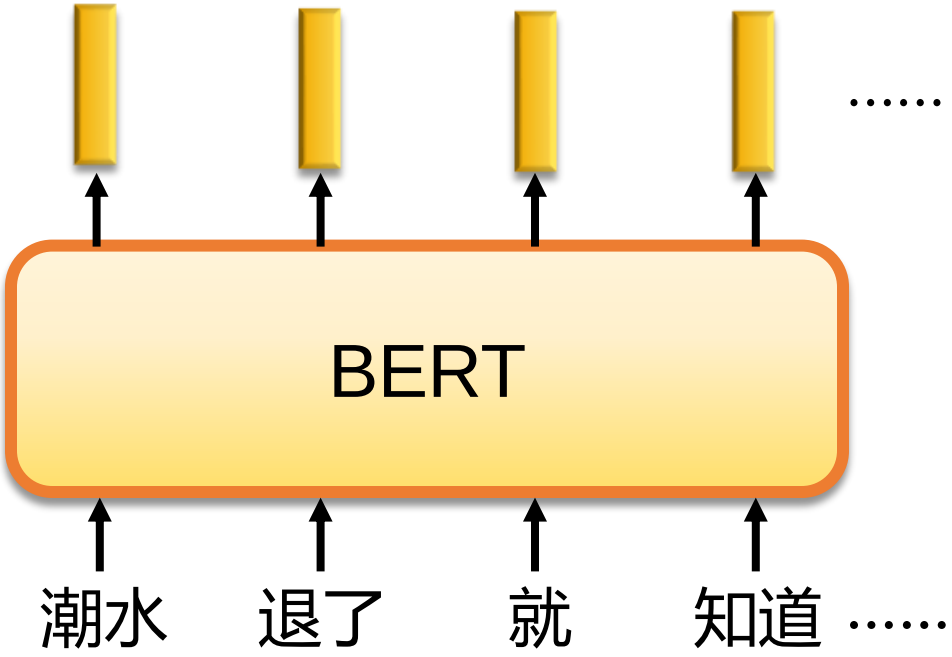


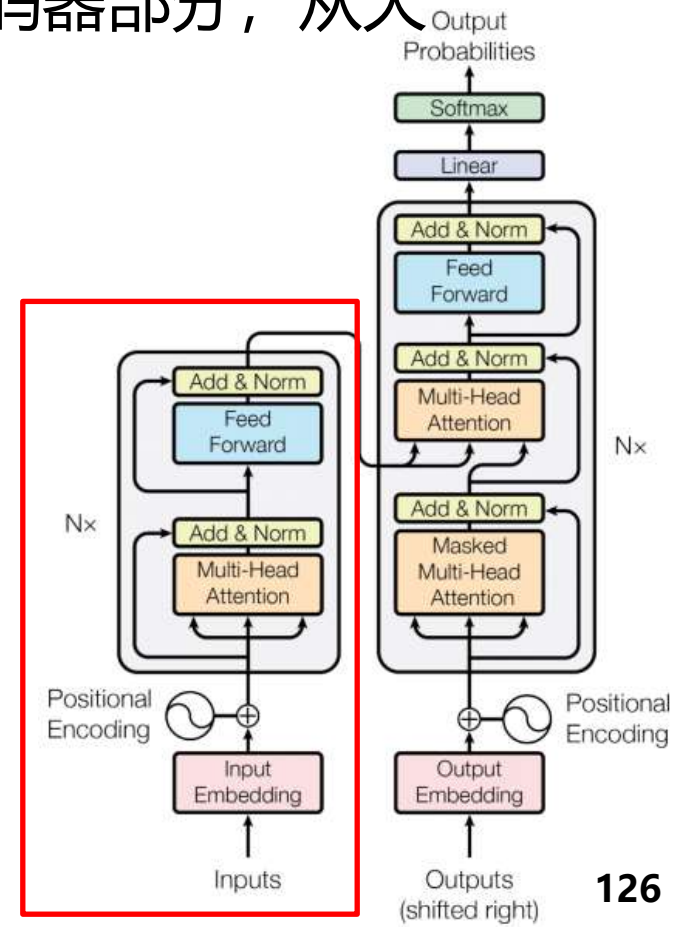


Learned with the down stream tasks

BERT

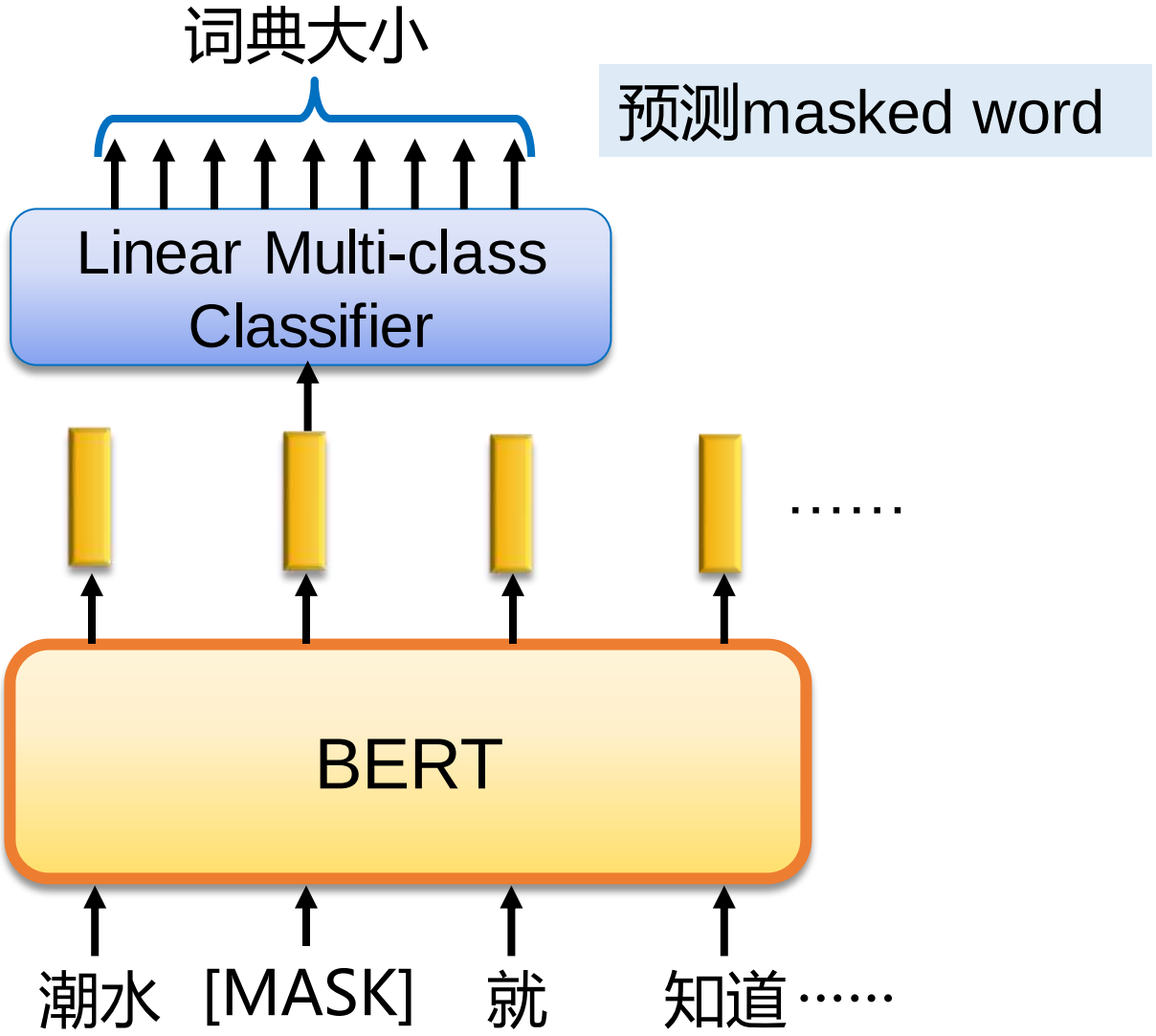
- **B**idirectional **E**ncoder **R**epresentations from **T**ransformers
  - BERT可以看作Transformer的编码器部分，从大量没有标注的文本中学习





BERT的训练

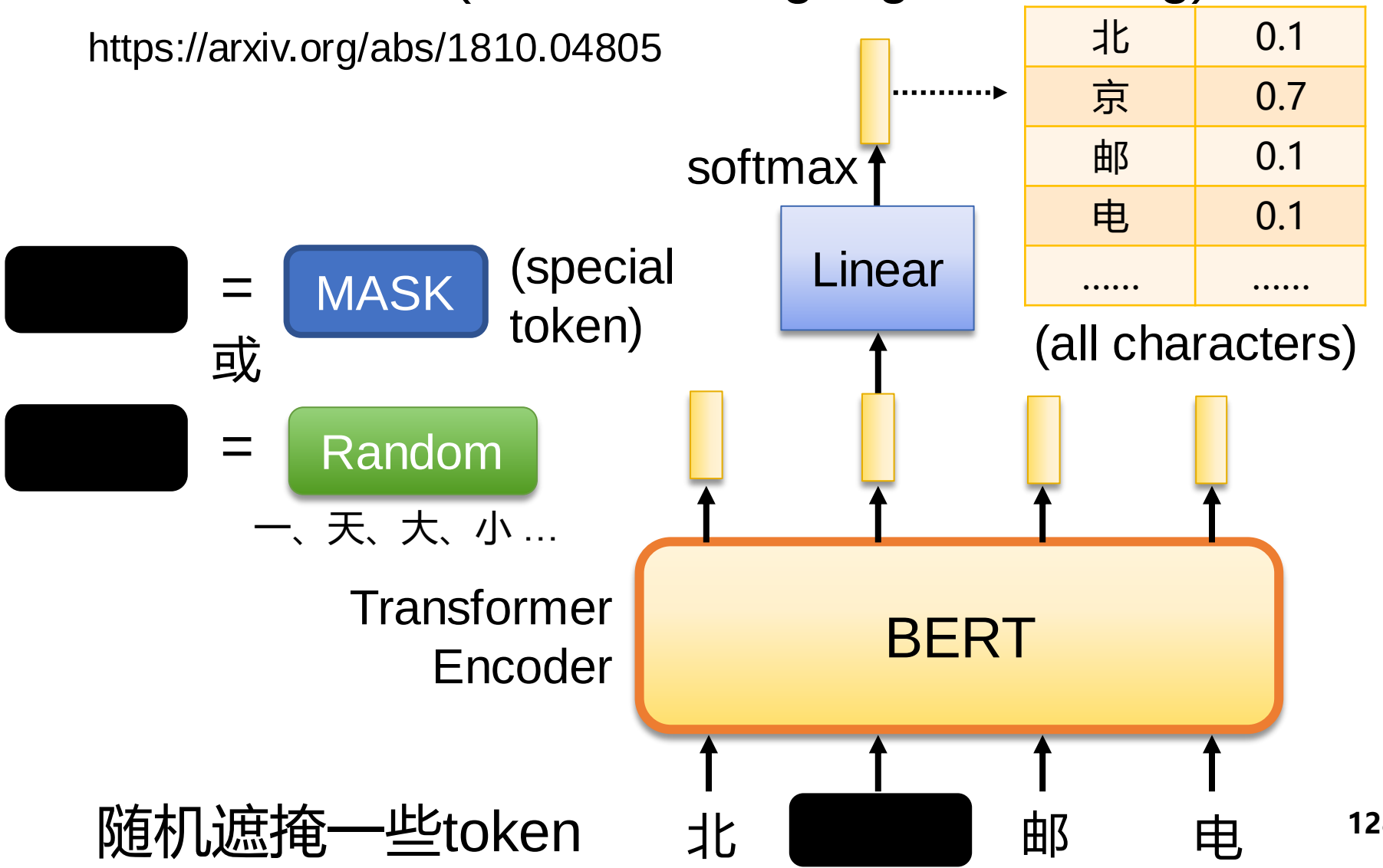
- 方法1：MLM(Masked Language Modeling)



BERT的训练

- 方法1：MLM(Masked Language Modeling)

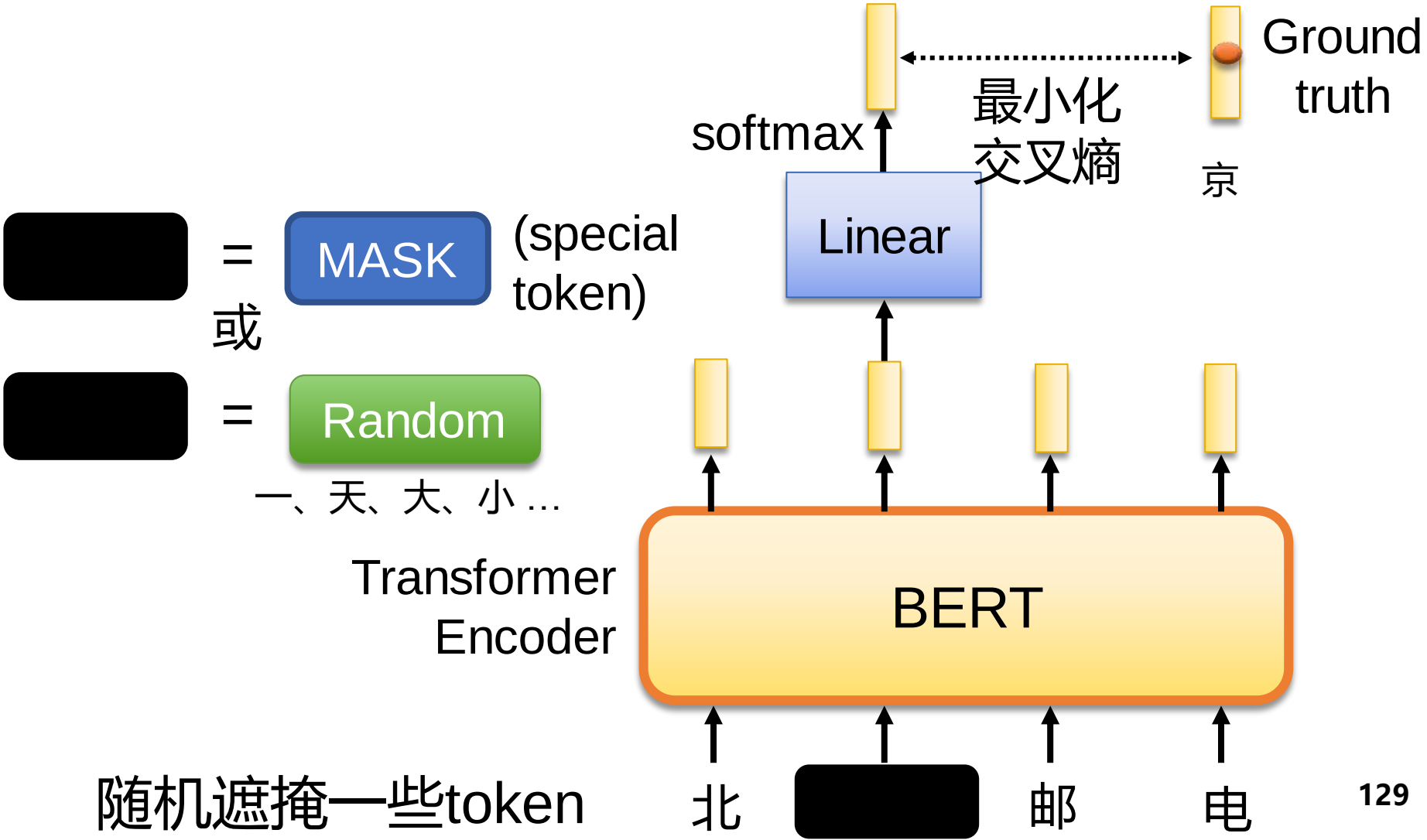
https://arxiv.org/abs/1810.04805



随机遮掩一些token

BERT的训练

- 方法1：MLM(Masked Language Modeling)



随机遮掩一些token

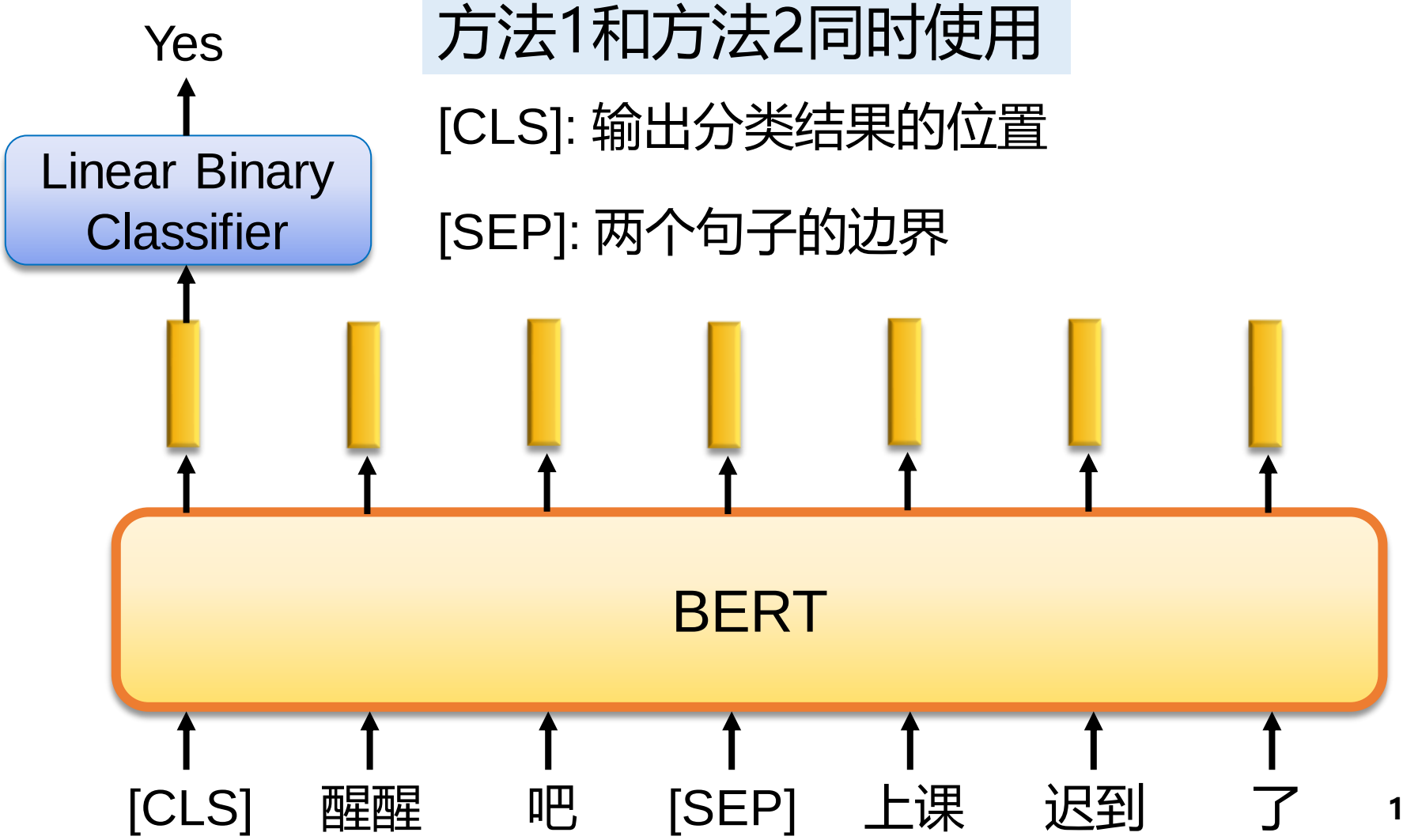
BERT的训练

- 方法2：Next Sentence Prediction

方法1和方法2同时使用

[CLS]: 输出分类结果的位置

[SEP]: 两个句子的边界



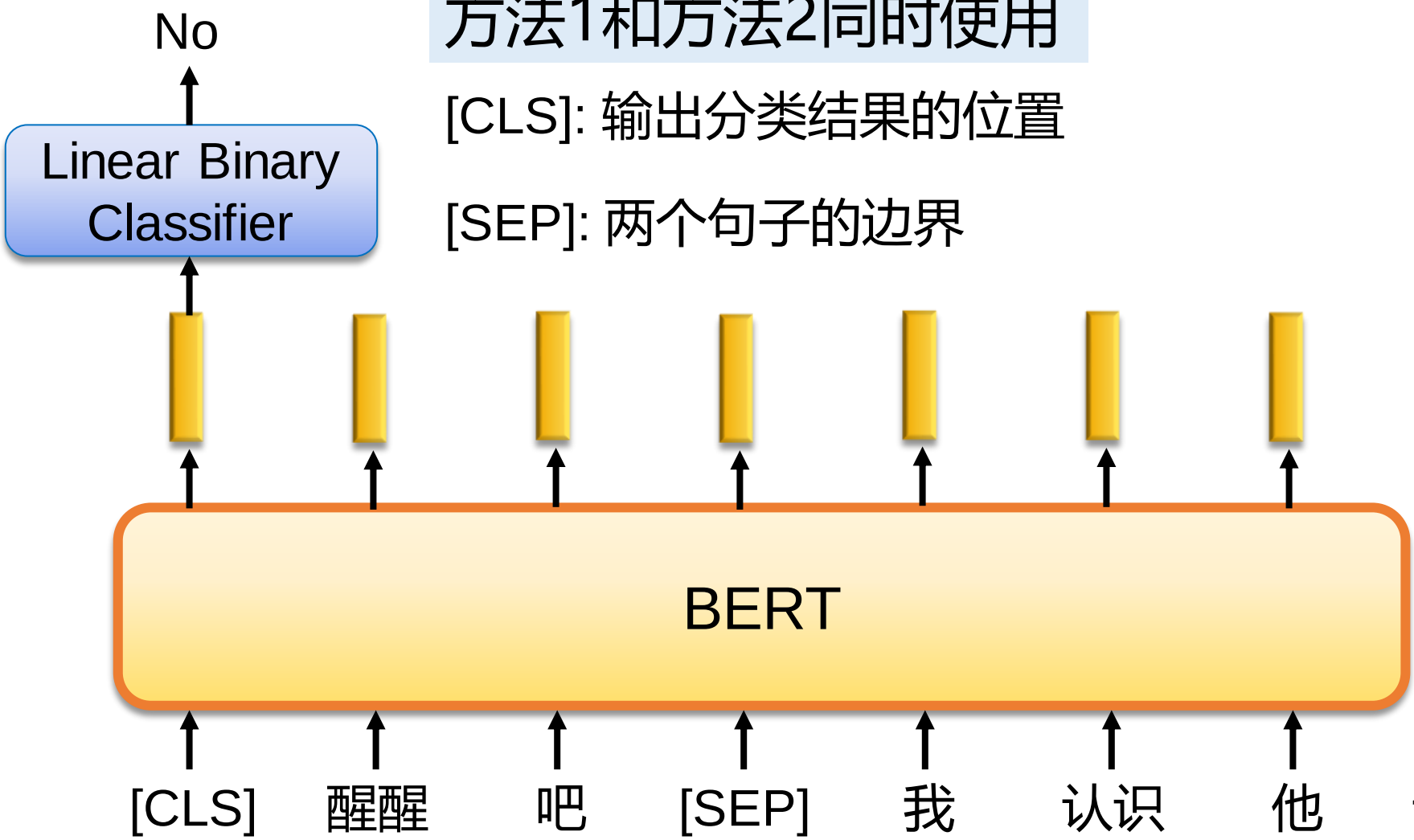
BERT的训练

- 方法2：Next Sentence Prediction

方法1和方法2同时使用

[CLS]: 输出分类结果的位置

[SEP]: 两个句子的边界



如何使用BERT

自监督学习

预训练

BERT

- Masked token prediction
- Next sentence prediction

微调

Model for Task 1

Model for Task 2

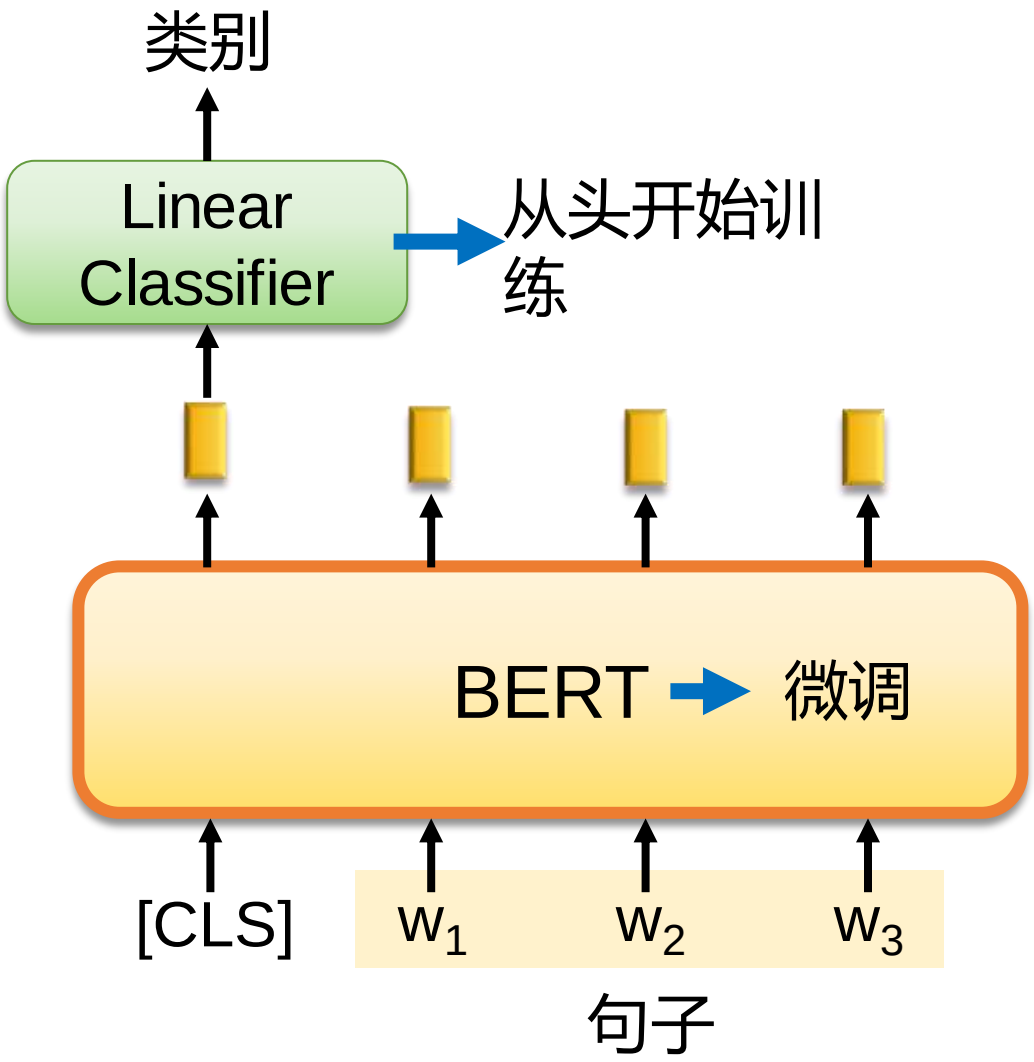
Model for Task 3

下游任务

- 我们关注的任务
- 有一些带标签的数据

如何使用BERT

- 情形1

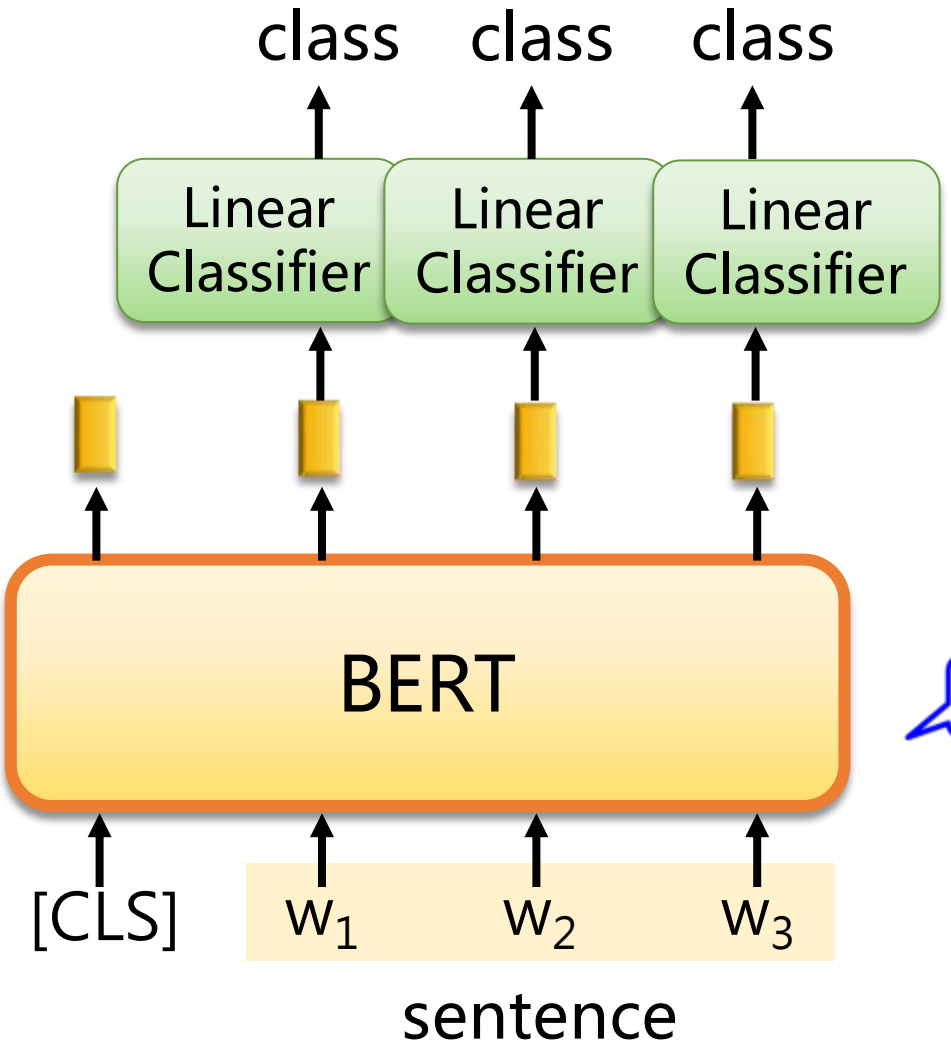


输入：单个句子
输出：类别

示例:
情感分析、 文本分类

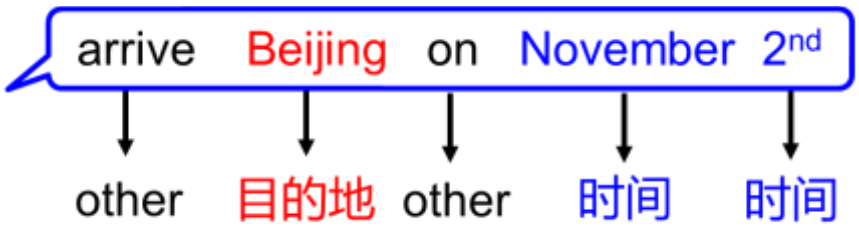
如何使用BERT

- 情形2



输入：单个句子
输出：每个词的类别

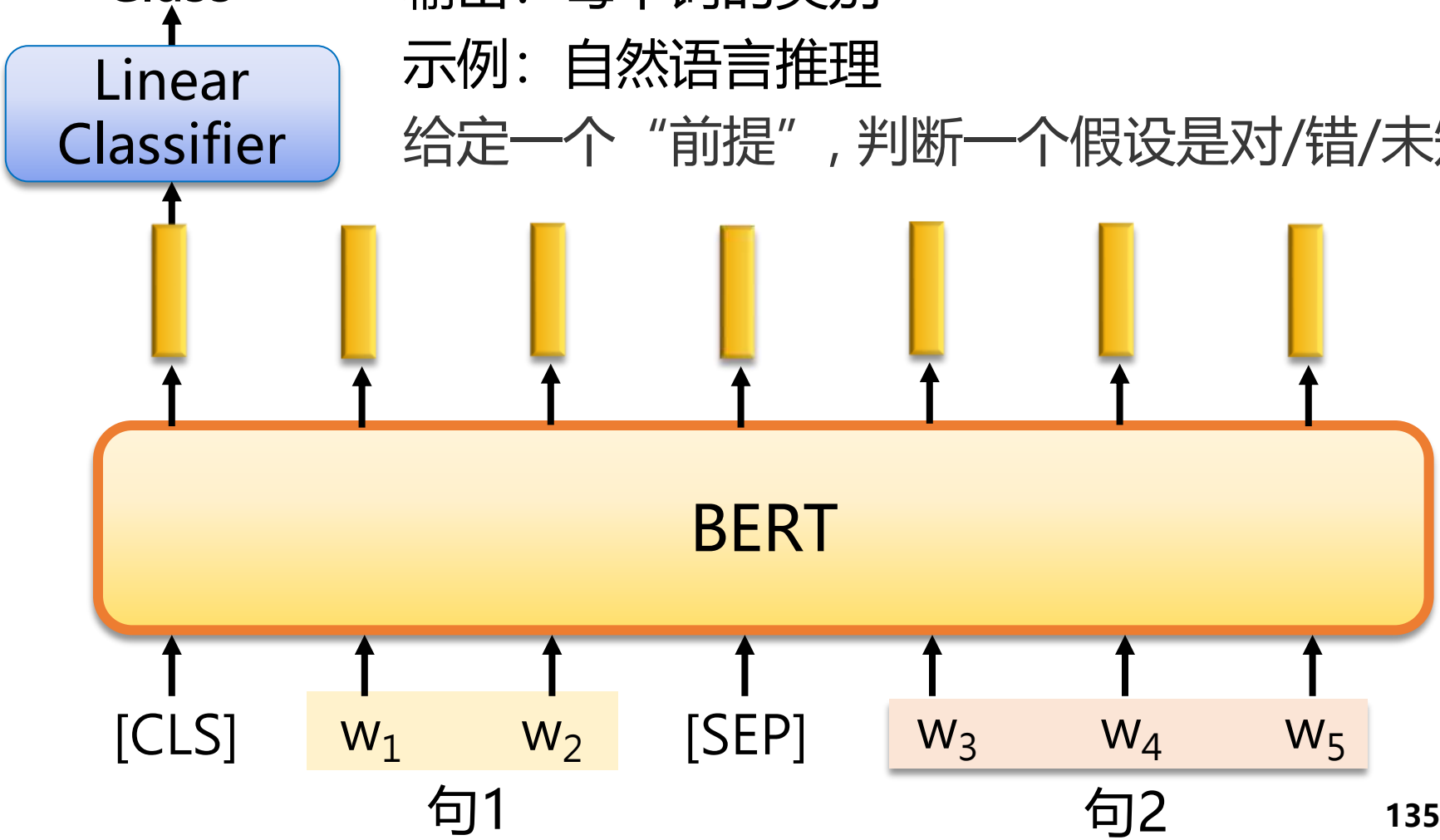
示例：Slot filling



如何使用BERT

- 情形3

输入：单个句子
输出：每个词的类别
示例：自然语言推理
给定一个“前提”，判断一个假设是对/错/未知



如何使用BERT

- 情形4：基于提取的问答(Extraction-based Question Answering)

文本 $D = \{d_1, d_2, \cdots, d_N\}$

询问 $Q = \{q_1, q_2, \cdots, q_N\}$

输出：两个整数(s, e)

回答 $A = \{q_s, \cdots, q_e\}$

In meteorology, precipitation is any product of the condensation of [17] spheric water vapor that falls under gravity. The main forms of precipitation include drizzle, rain, sleet, snow, graupel and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals within a cloud. Short, intense periods of rain [77] atte [79] cations are called "showers".

What causes precipitation to fall?
gravity

$s = 17, e = 17$

What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?
graupel

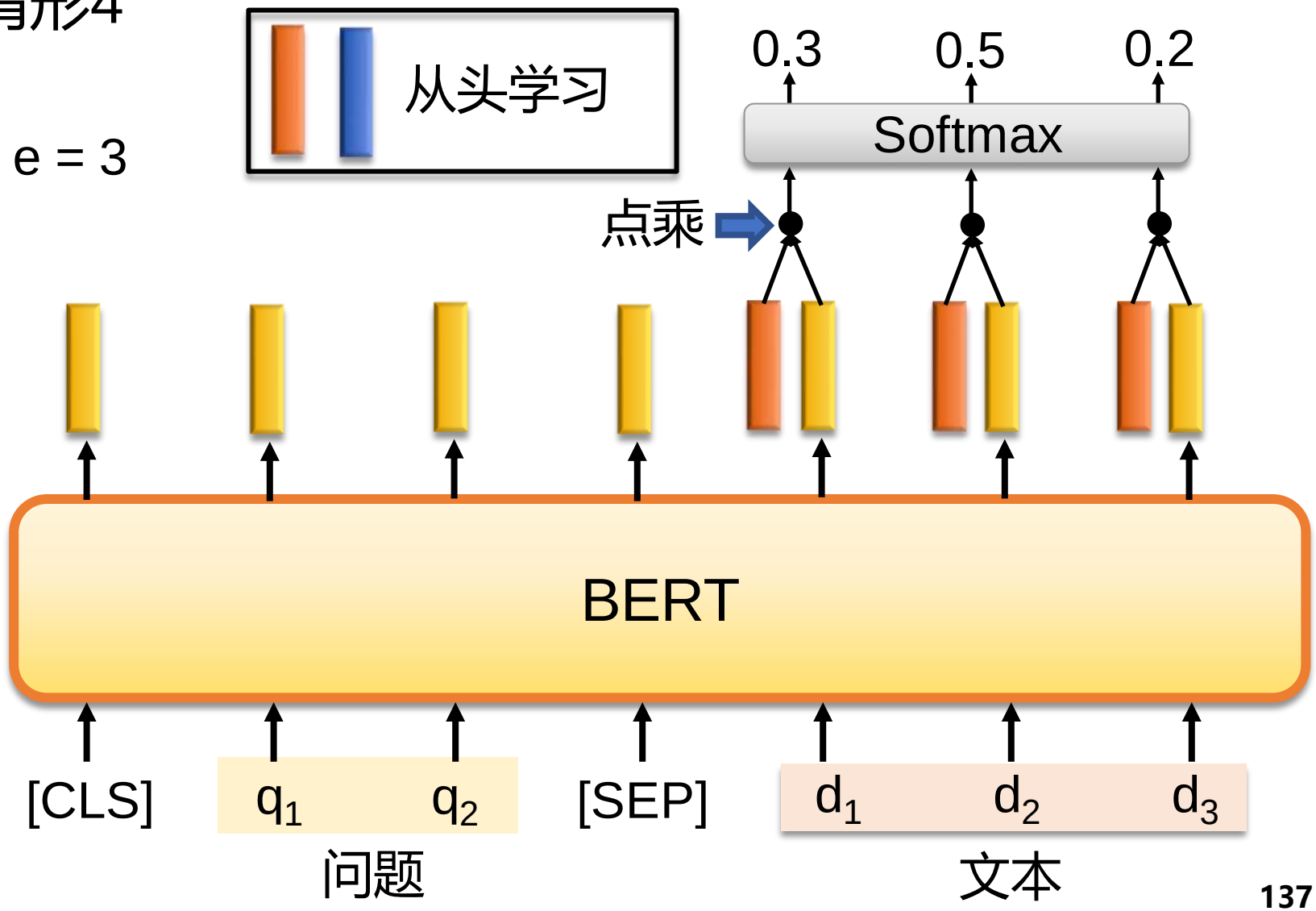
Where do water droplets collide with ice crystals to form precipitation?
within a cloud

$s = 77, e = 79$

如何使用BERT

- 情形4

s = 2, e = 3

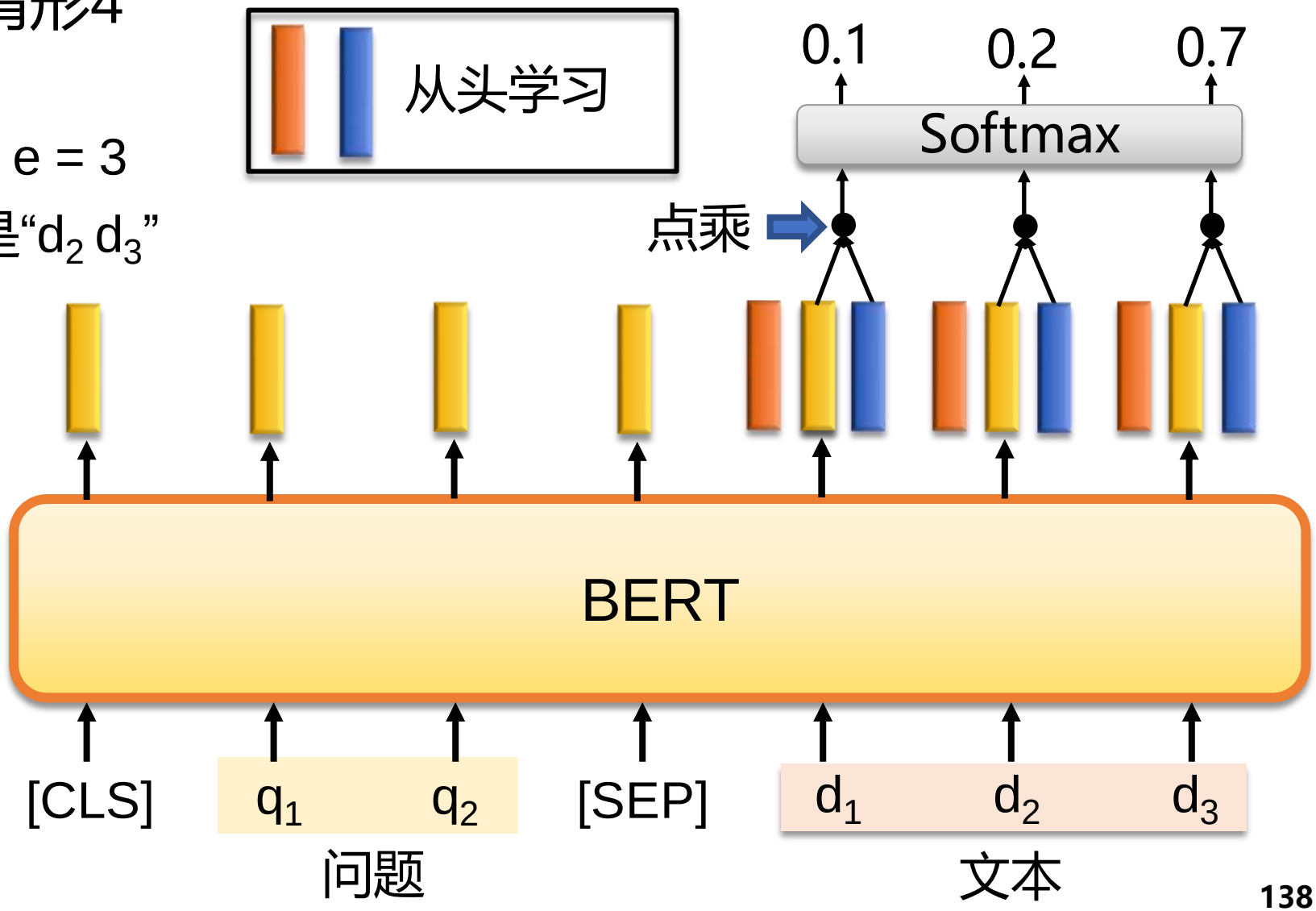


如何使用BERT

- 情形4

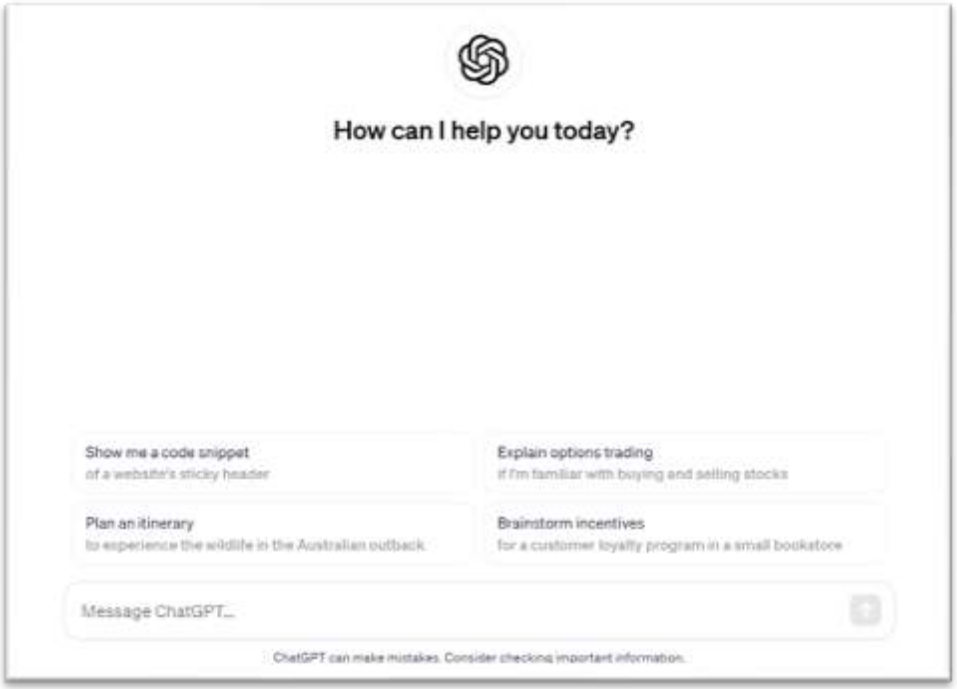
s = 2, e = 3

答案是“$d_2\ d_3$”

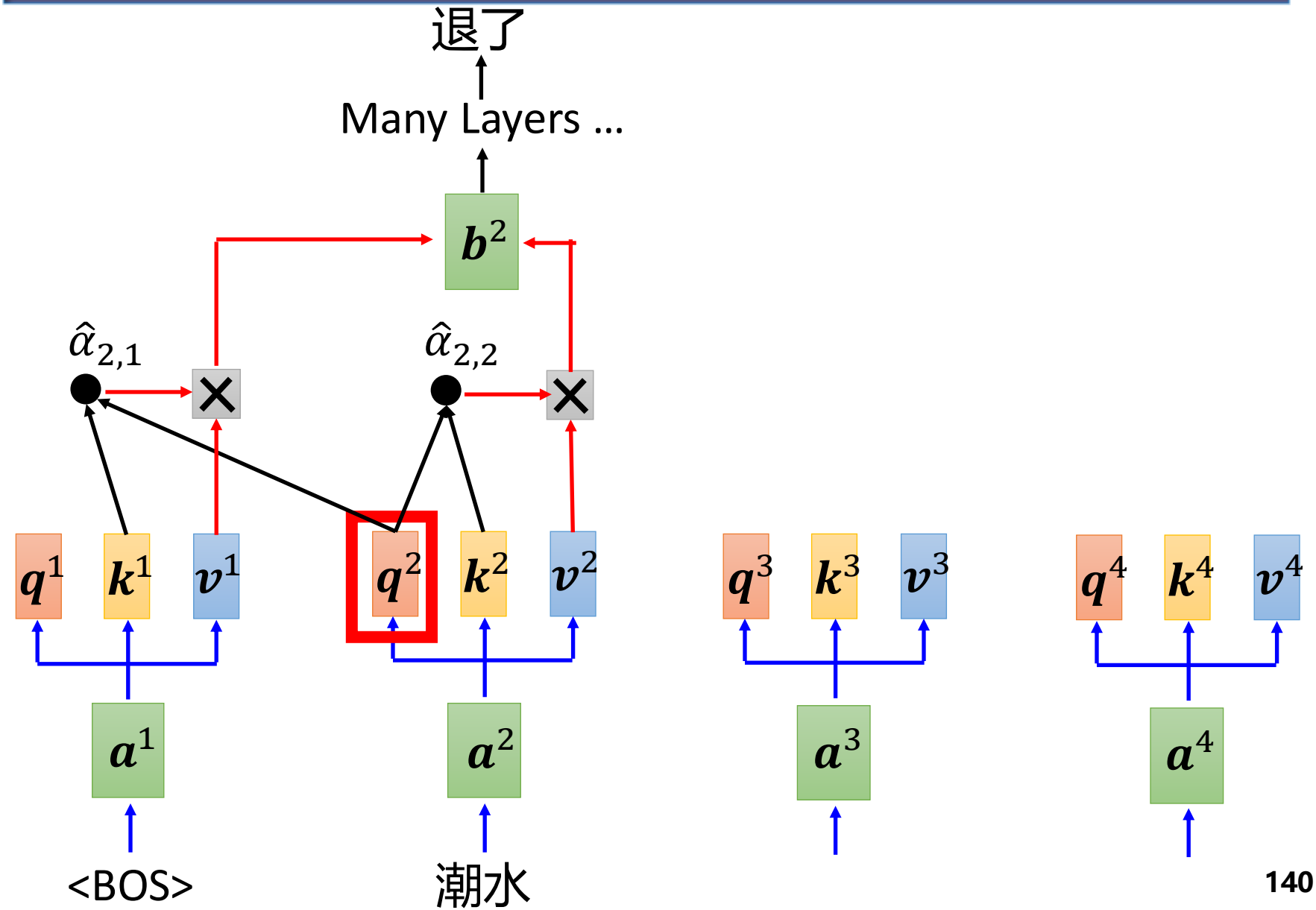


GPT(Generative Pre-trained Transformer)

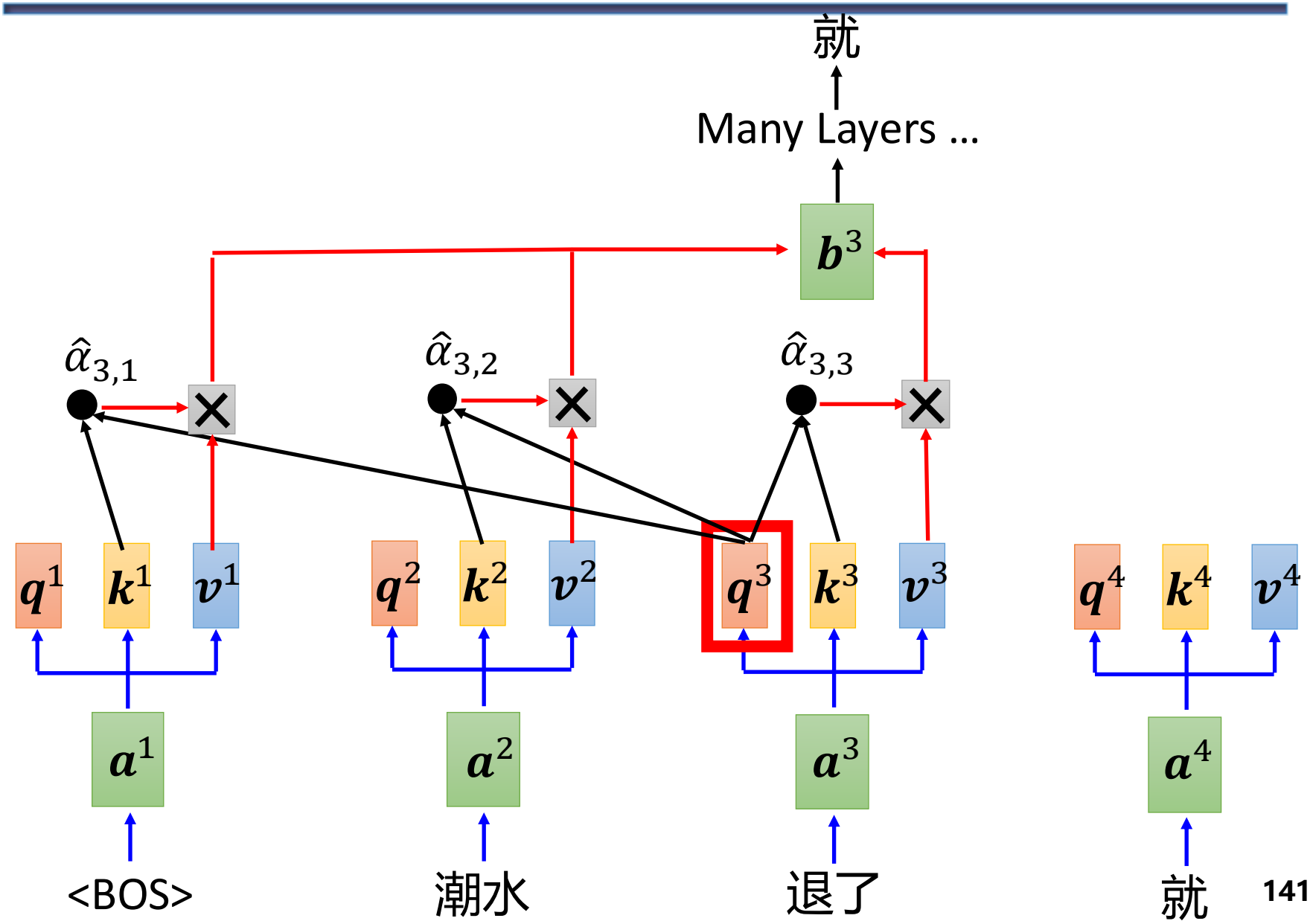
- OpenAI 提出了仅使用解码器堆栈的 GPT-1-4 模型，使其成为从左到右的自回归模型。
  - GPT是一系列使用 Transformer 架构的神经网络模型
  - GPT 模型使应用程序能够创建类似人类的文本和内容（图像、音乐等），并以对话方式回答问题。



GPT


退了
Many Layers …
b^2
$\hat{\alpha}_{2,1}$
$\hat{\alpha}_{2,2}$
q^1 k^1 v^1
q^2 k^2 v^2
q^3 k^3 v^3
q^4 k^4 v^4
a^1
a^2
a^3
a^4
<BOS>
潮水

GPT


就
Many Layers …
b^3
$\hat{\alpha}_{3,1}$
$\hat{\alpha}_{3,2}$
$\hat{\alpha}_{3,3}$
q^1 k^1 v^1
q^2 k^2 v^2
q^3 k^3 v^3
q^4 k^4 v^4
a^1 a^2 a^3 a^4
<BOS>
潮水
退了
就

GPT-1

- 2018, 117 million 参数
  - 没有表现出应急能力，并且严重依赖于单个下游任务的微调。

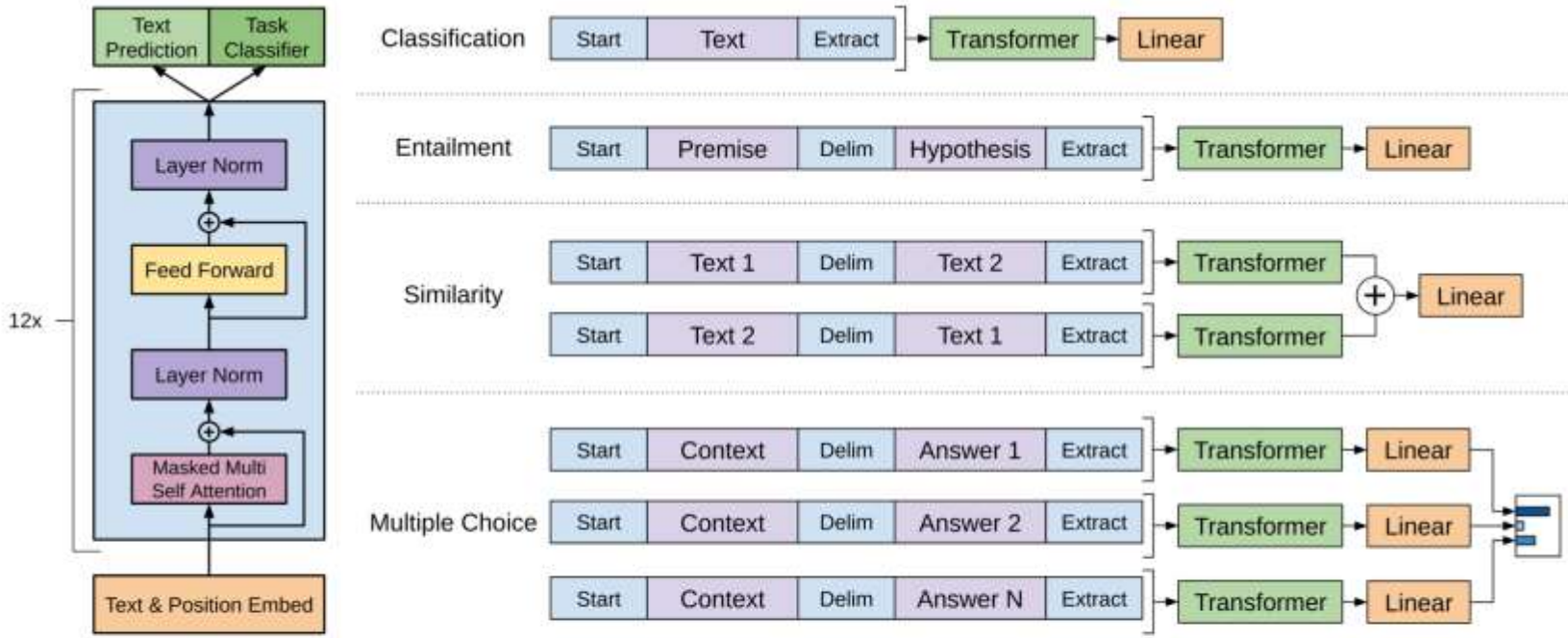
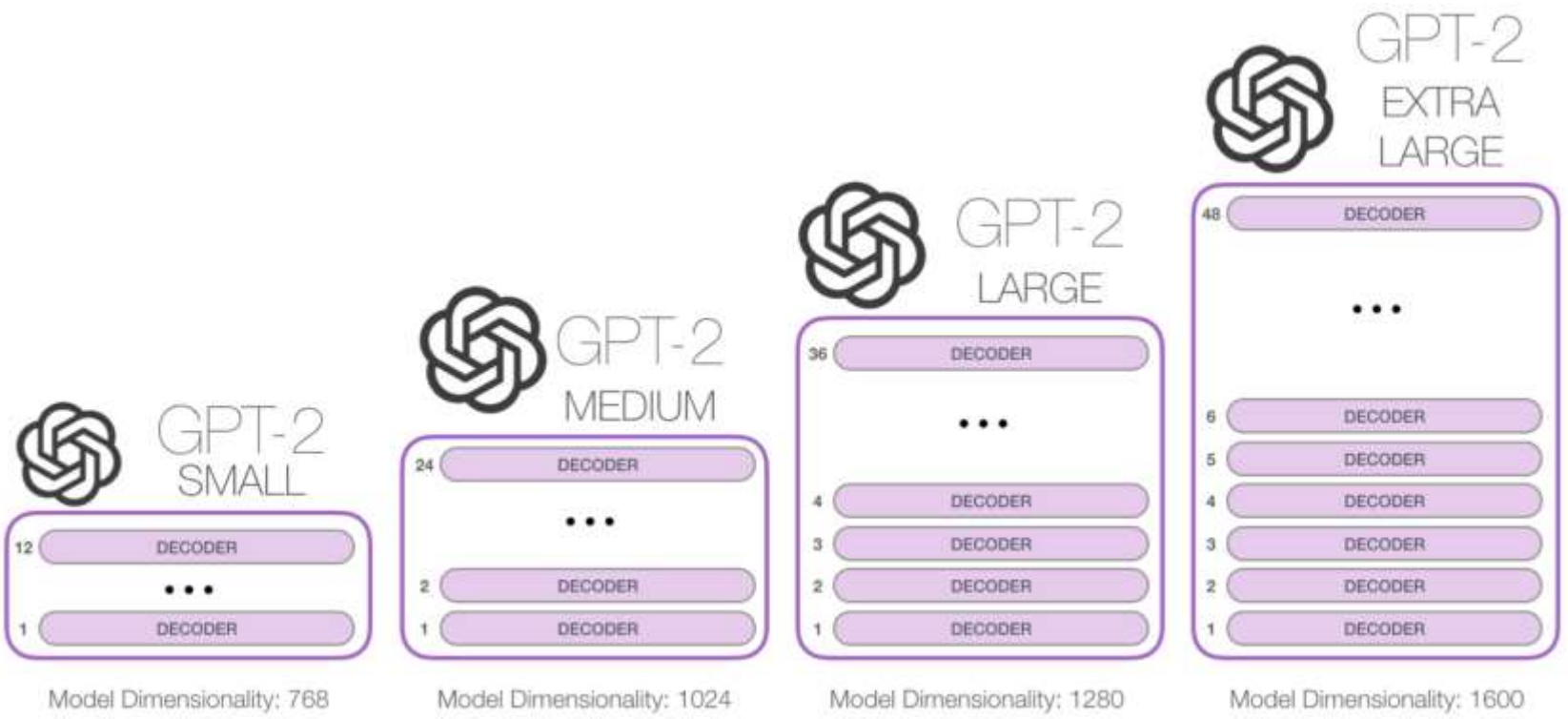


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

GPT-2

- 2019, 1.5 billion 参数
  - GPT-2引入了一些任务的上下文学习现象，并在 GPT-1 中使用的原始 spacy 分词器之上使用字节级编码 (BLE) 改进了其分词器。



GPT-3

- 2019, 175 billion 参数
  - 表现出强大的情境学习能力，包括零样本和少样本学习能力

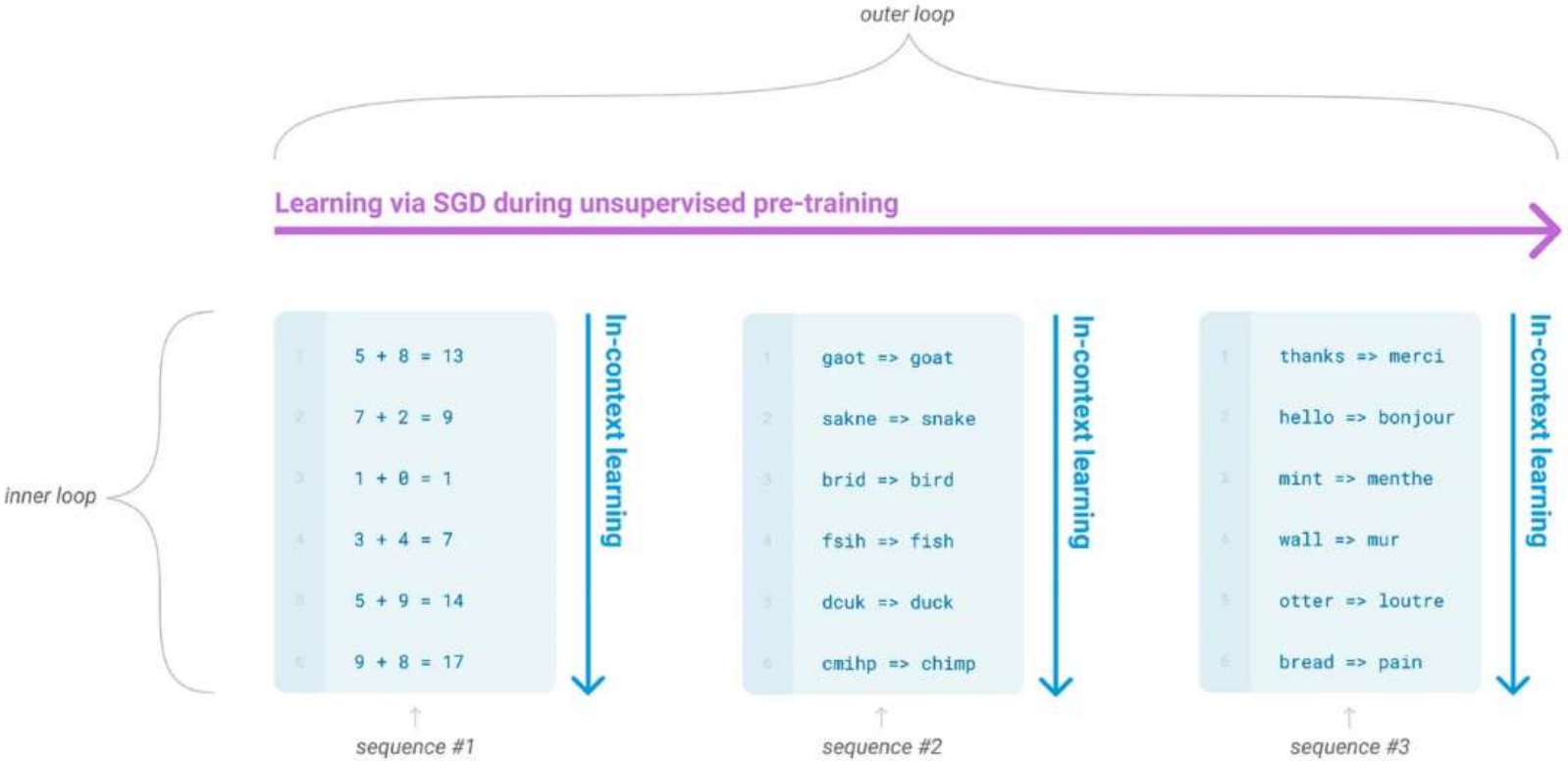


Fig. 7: GPT-3's In-context learning with unsupervised pre-training only. Without further fine-tuning on specific tasks, GPT-3 learns (left) simple arithemetic operations; (middle) word typo auto-correction; (right) multi-language word translation. The patterns are learnt from the pre-training context dataset only. (Image Source: Brown et. al., 2020)

GPT-4

◘最新的GPT-4 Turbo支持个人定制GPT应用

https://openai.com/blog/introducing-gpts

GPT-3 是如何训练的?

- GPT-3 在半监督模式下训练
  - 首先，机器学习工程师向深度学习模型提供未标记的训练数据。GPT-3 会理解这些句子，将其分解，然后将它们重构成新的句子。在无监督训练中，GPT-3 试图自行生成准确、真实的结果。
  - 然后，机器学习工程师对有监督训练的结果进行微调，这个过程被称为人工反馈强化学习 (RLHF)。