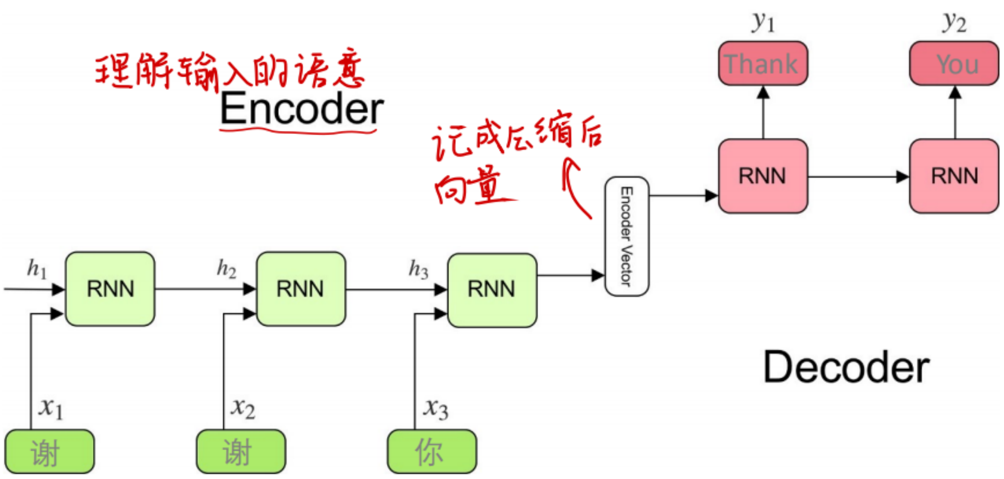
ch12: Natural Language Processing

RNN Encoder-Decoder

overview(P6)

- given $x = (x_1, \cdots, x_{T_x})$, generate $y = (y_1, \cdots, y_{T_y})$



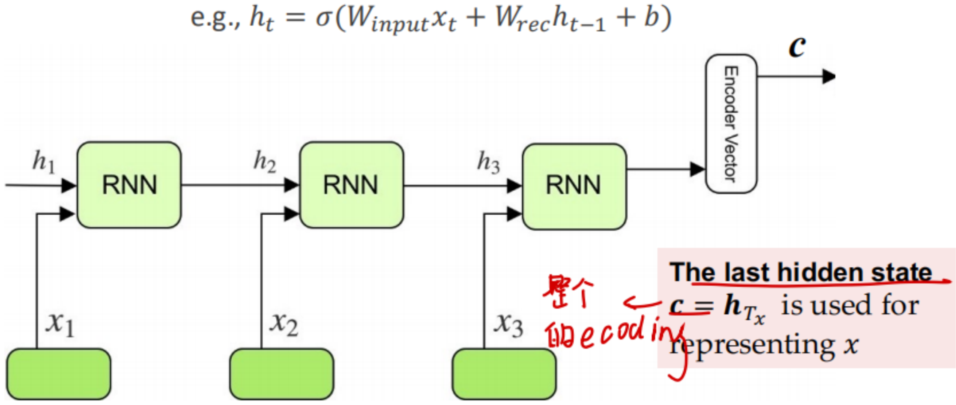
Encoder(P8)

- An RNN which learns a dense representation of a sequence.
- **Compresses** a sequence of tokens into a context vector c.

压缩

$$\boldsymbol{h}_t = \mathrm{RNN}_{in}(x_t, \boldsymbol{h}_{t-1})$$ 信息压缩

e.g., $h_t = \sigma(W_{input}x_t + W_{rec}h_{t-1} + b)$



Decoder(P9)

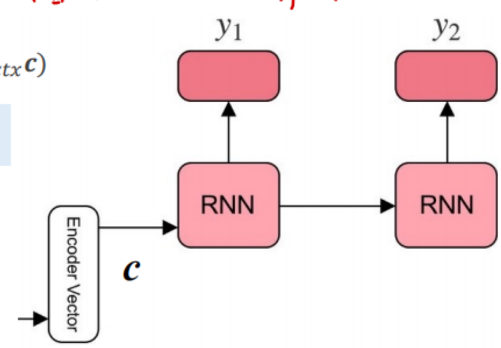
- An RNN which generates an sequence **conditioned on** the intermediate representation.
- Sequentially predicts the next token y_i given the context vector $\boldsymbol{c}$ and the hidden state of past-generated sequence. 逐个生成下一个token y

$$\boldsymbol{s}_i = \mathrm{RNN}_{out}(y_{i-1}, \boldsymbol{s}_{i-1}, \boldsymbol{c})$$ 也要看encoder的输入

e.g., $s_i = \sigma(W'_{input}y_{i-1} + W'_{rec}\boldsymbol{s}_{i-1} + W_{ctx}\boldsymbol{c})$

$$p(y_i \mid y_{<i}, \boldsymbol{c}) = \mathrm{softmax}(\mathrm{g}(\boldsymbol{s}_i))$$

e.g., $g = V^T(W_1\, y_{t-1} + W_2 s_t + W_3 c)$



Training(P10)

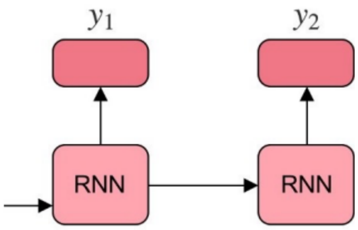
Data: $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$,

where $x^{(\ell)} = (x_1, \ldots x_{Tx})$ and $y^{(\ell)} = (y_1, \ldots y_{Ty})$.

Loss Function – minimize the **cross-entropy** loss:

$$L(\theta) = -\frac{1}{N}\sum_{\ell=1}^{N}\sum_{t=1}^{T_y} \log p_\theta(y_t^{(\ell)} | x^{(\ell)})$$

Optimization – gradient descend



Applications(P12-14)

- Translation
- Conversation
- Image Captioning
- **Example: Chatbot(Hierarchical RNN Encoder-Decoder)P14**

Attention

Sequence-to-Sequence with Attention(P18-19)

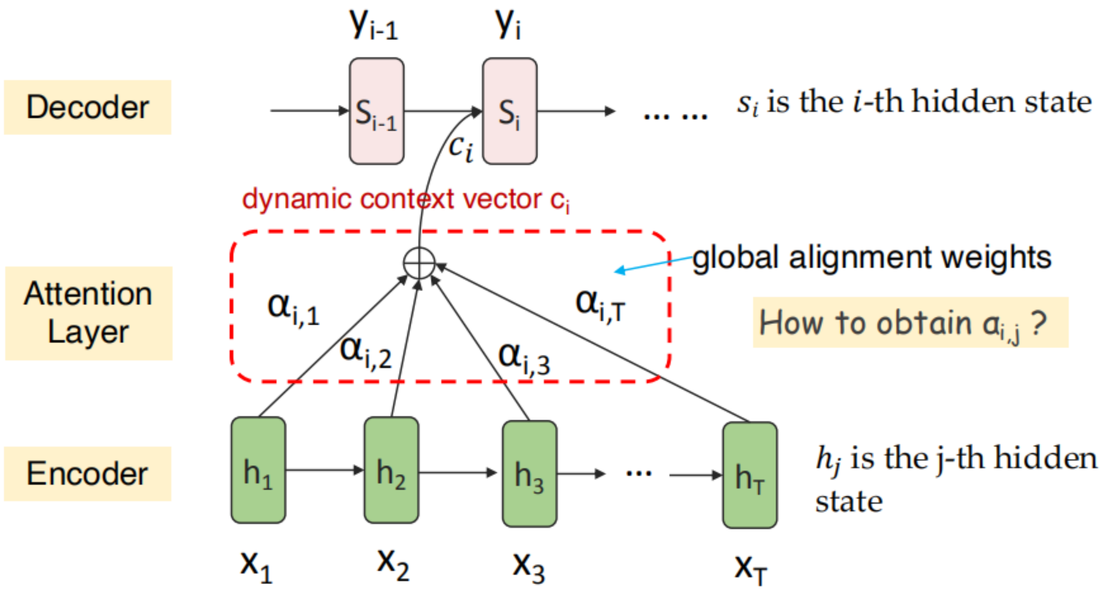
- When decoding each y_i in $y = (y_1, \cdots, y_{T_y})$, we use a **dynamic context vector** $\boldsymbol{c}_i$ which corresponds to a linear combination of different positions in x.

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

编码之后输入的信息

Without Attention		With Attention
• $p(y_i \mid y_{<i}, x) \propto g(y_i; y_{i-1}, s_i, c)$	VS	• $p(y_i \mid y_{<i}, x) \propto g(y_i; y_{i-1}, s_i, \boldsymbol{c_i})$
• RNN hidden state $s_i = \mathrm{RNN}_{\mathrm{out}}(s_{i-1}, y_{i-1}, c)$		• RNN hidden state $s_i = \mathrm{RNN}_{\mathrm{out}}(s_{i-1}, y_{i-1}, \boldsymbol{c_i})$

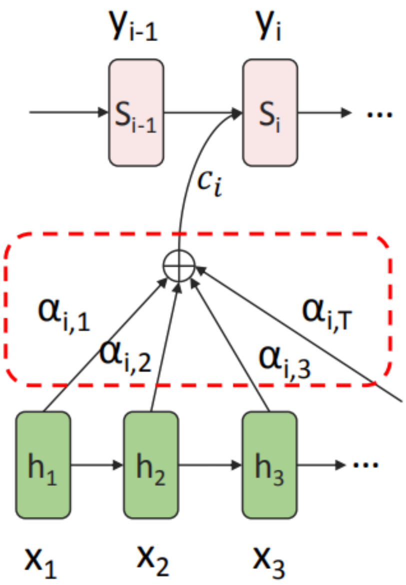
The decoder **dynamically** pays attention to different tokens in the source sentences during decoding.



AEention-based Model(P19)

$$\alpha_{ij} = \frac{\exp\left(a(s_{i-1}, h_j)\right)}{\sum_{k=1}^{T_x} \exp\left(a(s_{i-1}, h_k)\right)}$$

- scores how well the inputs around position j and the output at position i match.
- where $a(\cdot)$ denotes a **neural network**: e.g., $a(s_{i-1}, h_j) = v_a^T \tanh(W_a s_{i-1} + U_a h_j)$

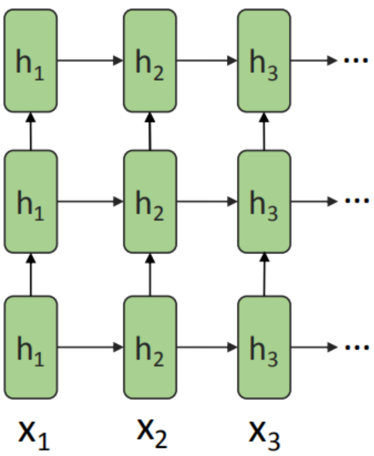


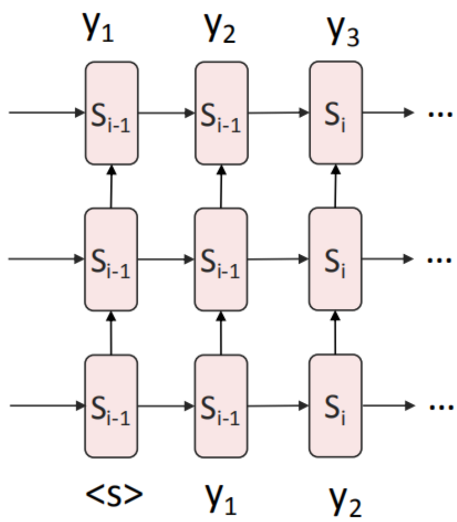
Transformer: Seq2seq model with “Self-attention”

Limitations of RNN（P22-23）

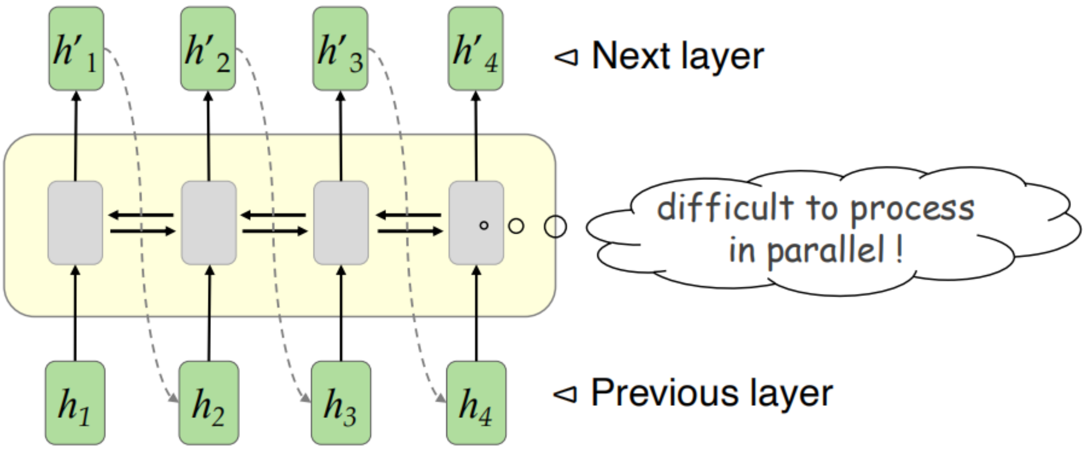
- Multilayer RNNs

问题：无法并行化，全部串行

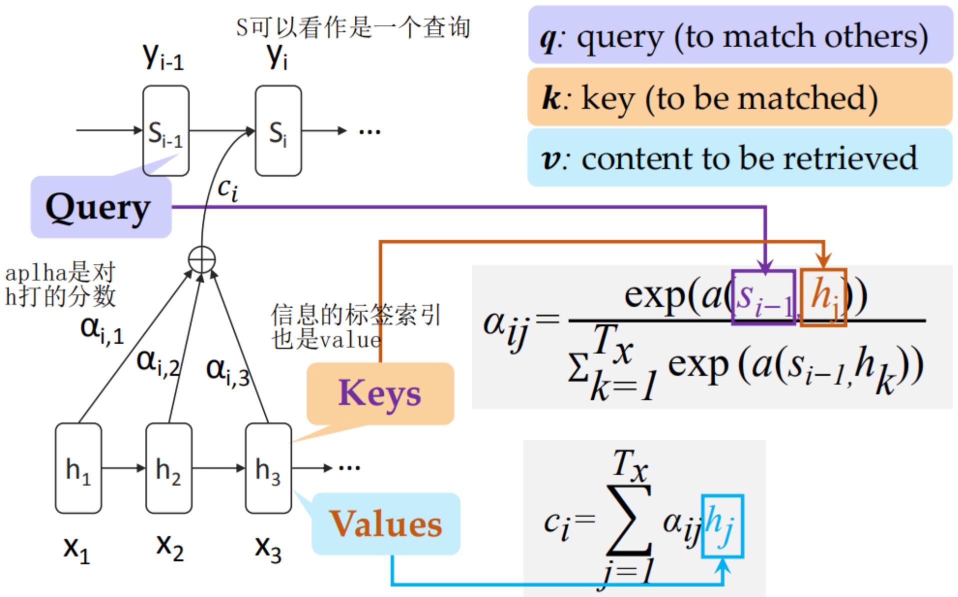




- **A problem for the RNN encoder/decoder:** the hidden state for one position is dependent on the computation of the preceding position.



Attention Revisited（P26）



Self-Attention: The Idea(P27-30)

> **self-attention: 在理解任何单词的语义时，同时查看其他所有的单词，并对语义进行总结**

- **Let each word pay attention to all other words.**
- Multiplying the query vector by each key vector produces a score for each value (technically: dot product followed by softmax)

- We multiply each value by its score and sum them up.

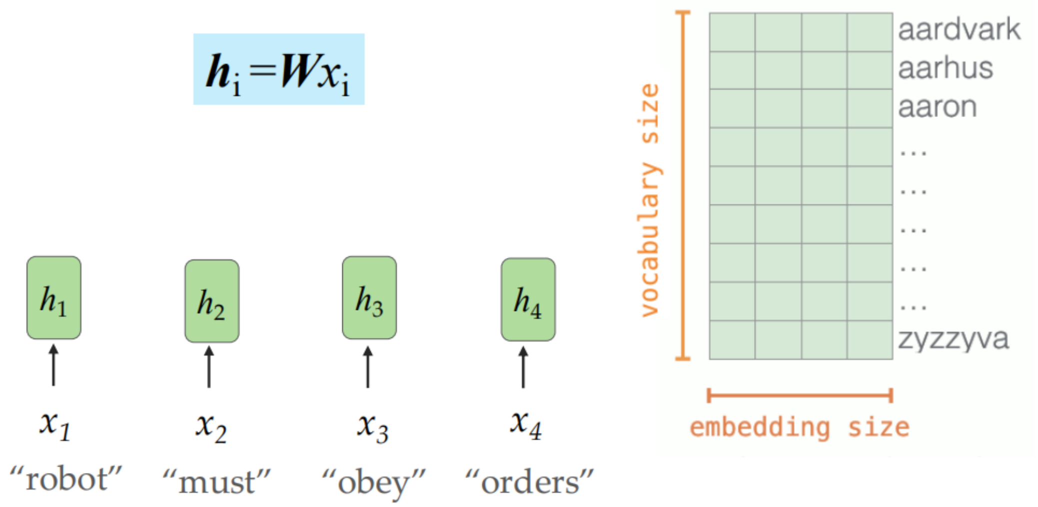
Word	Value vector	Score	Value X Score
<s>		0.001	
a		0.3	
robot		0.5	
must		0.002	
obey		0.001	
the		0.0003	
orders		0.005	
given		0.002	
it		0.19	
		Sum:	

做完内积后每个词都获得一个分数，表示和it的相关程度

- The outcome vector represents the new state (refreshed memory) for the query word.

Step 1: Token Embedding(P31)

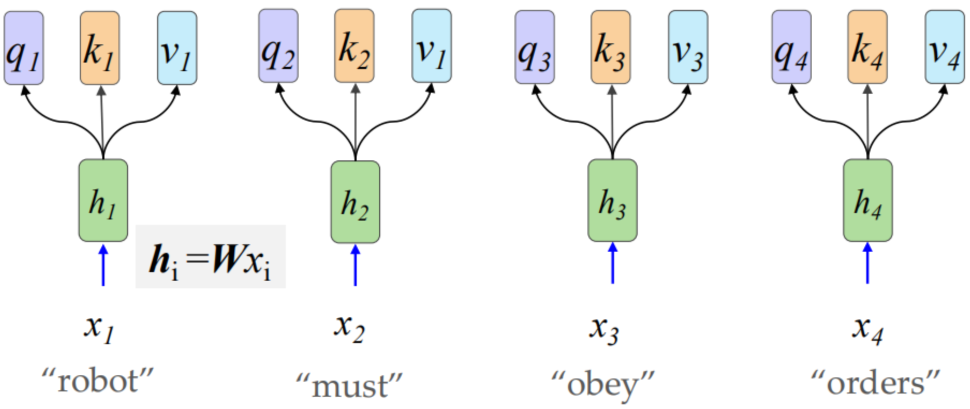
- Embedding tokens (integer id) into vectors (hidden states).



Step 2 : Q, K, V vectors(P32)

- Transform each hidden state into **query/key/value** vectors:

$q_i = W^q h_i$ $k_i = W^k h_i$ $v_i = W^v h_i$



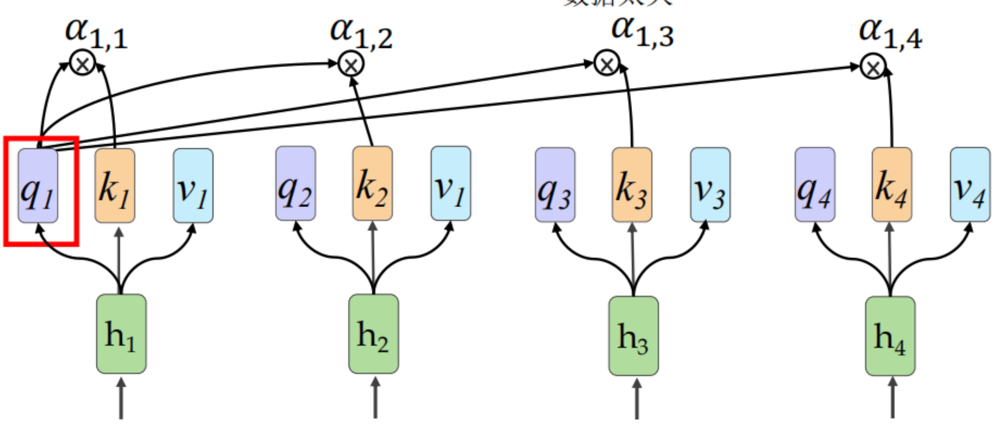
Step 3: Calculate Attention Scores(P33)

- Calculate an attention score for each <query, key> pair using **scaled dot-product**.

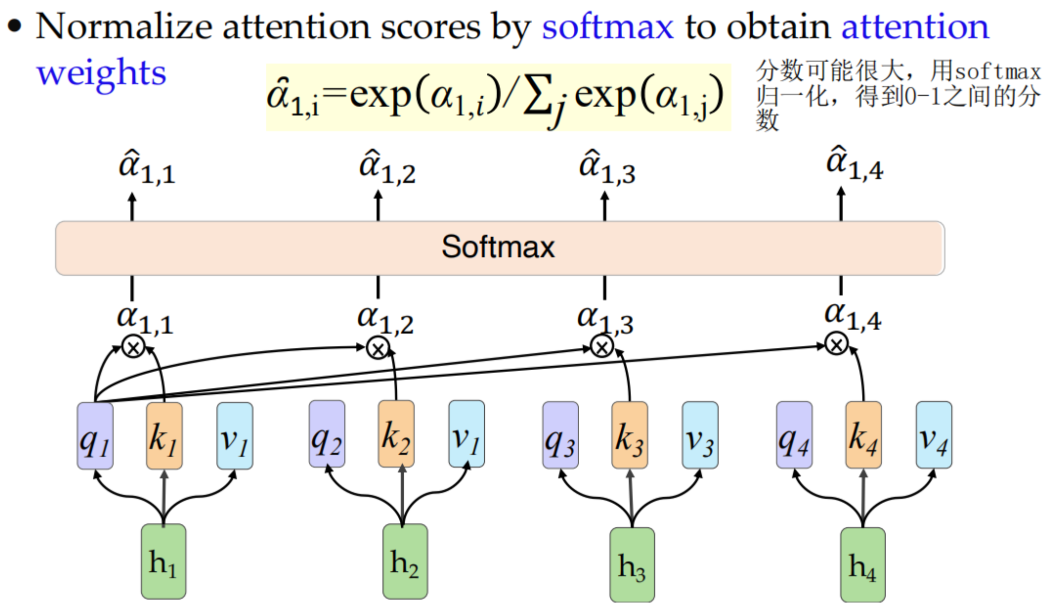
$$\alpha_{1,i} = \frac{q_i^T k_i}{\sqrt{d}}$$

where d is the dim of q and k

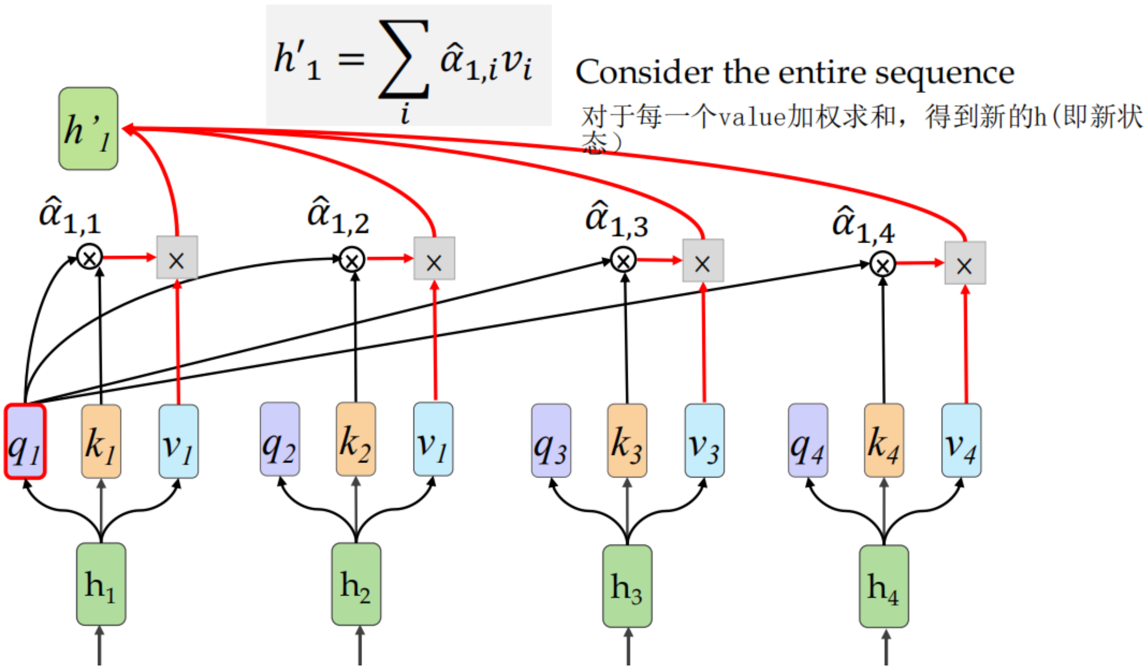
d在这里是数据的维度，在工程中会用到，防止数据太大



Step 4: Normalize Attention Scores(P34)

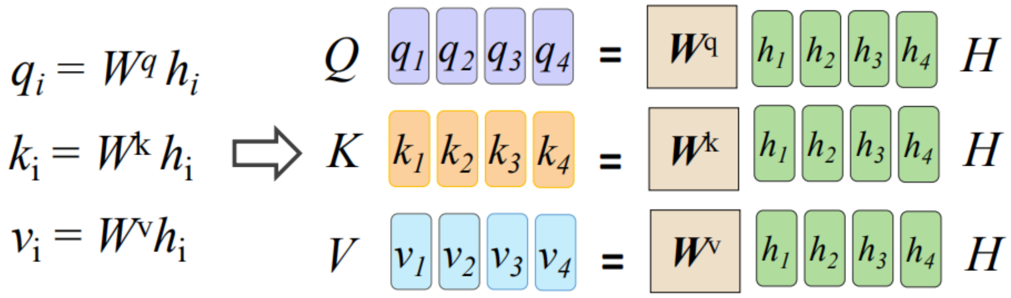


Step 5: Aggregate Values Based on Attention Weights(P35)

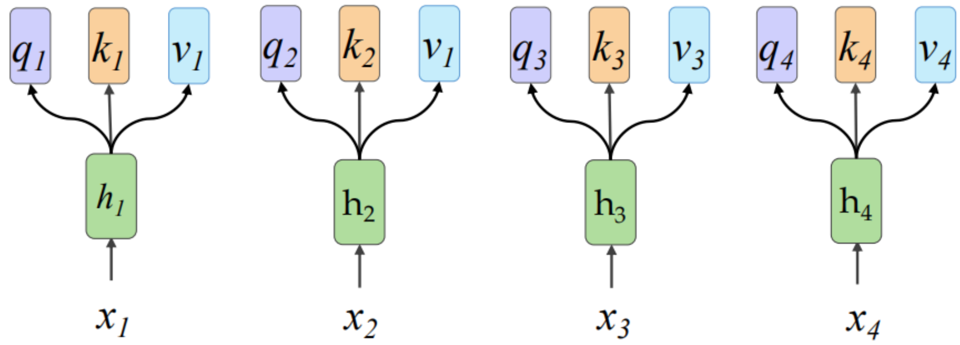


- h1,h2的更新其实是同步进行的，彼此之间互不影响

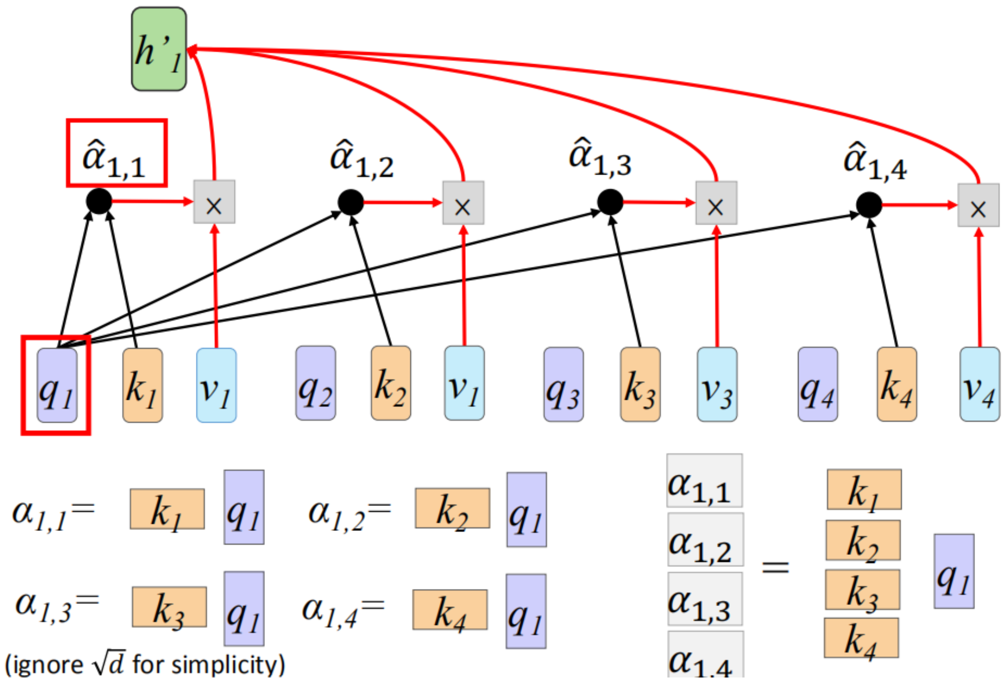
Self-Attention as Matrix Multiplication(P40-45)



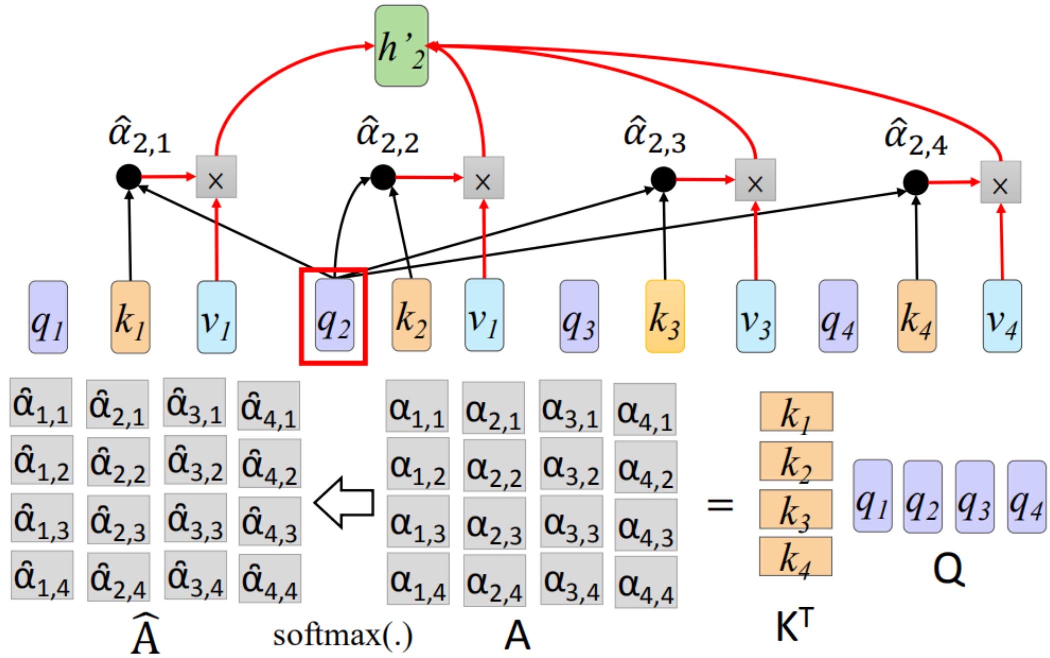
$q_i = W^q h_i$
$k_i = W^k h_i$
$v_i = W^v h_i$
Q
K
V
=
W^q
W^k
W^v
H

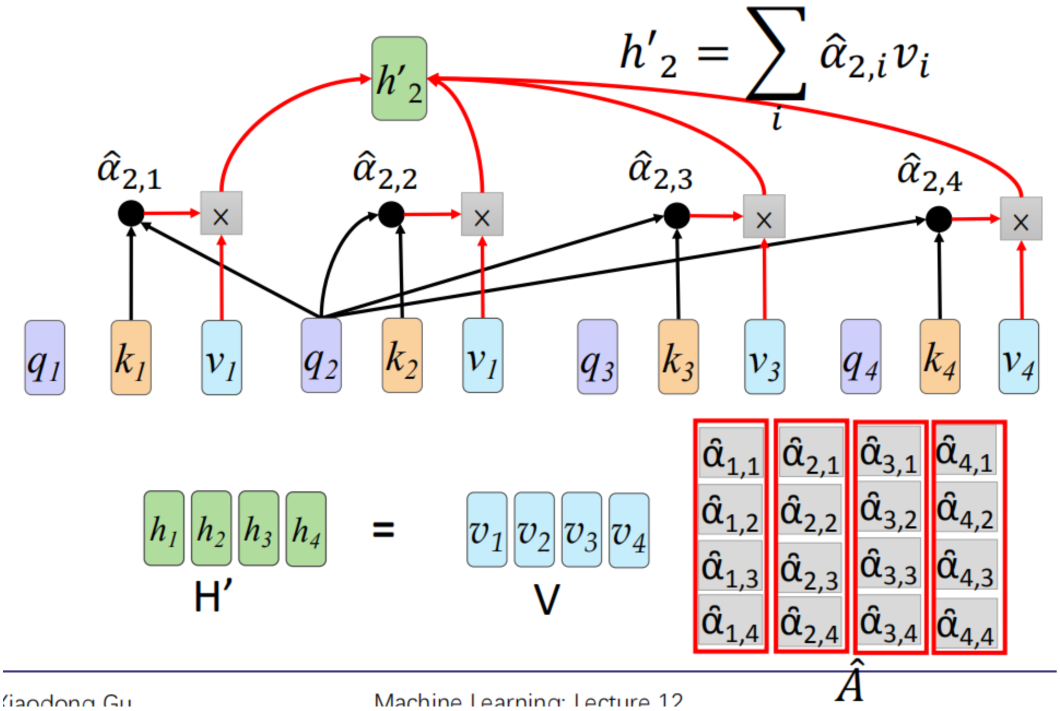


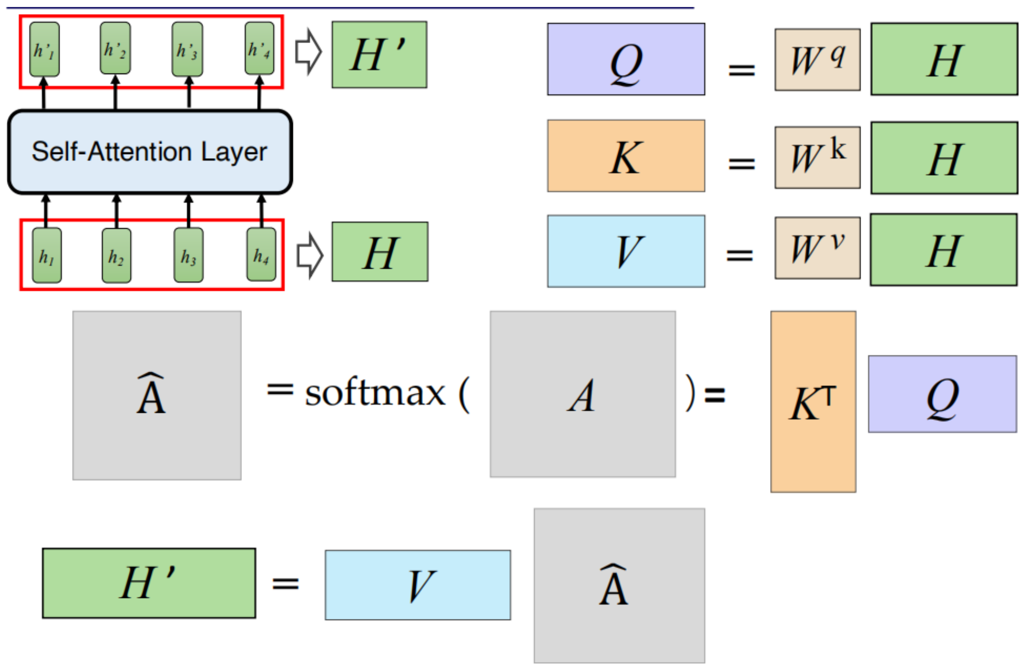
q_1 k_1 v_1
q_2 k_2 v_1
q_3 k_3 v_3
q_4 k_4 v_4
h_1
h_2
h_3
h_4
x_1
x_2
x_3
x_4



h'_1
$\hat{\alpha}_{1,1}$
$\hat{\alpha}_{1,2}$
$\hat{\alpha}_{1,3}$
$\hat{\alpha}_{1,4}$
$\alpha_{1,1}=$
$\alpha_{1,2}=$
$\alpha_{1,3}=$
$\alpha_{1,4}=$
(ignore $\sqrt{d}$ for simplicity)


softmax(.)
A
K^T
Q

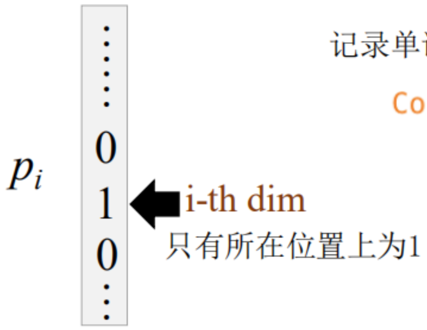

$h'_2 = \sum_i \hat{\alpha}_{2,i} v_i$
H'
=
V
Machine Learning: Lecture 12

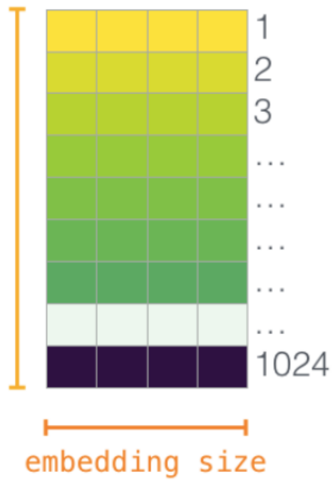


Self-Attention with Position Encoding(P48-49)

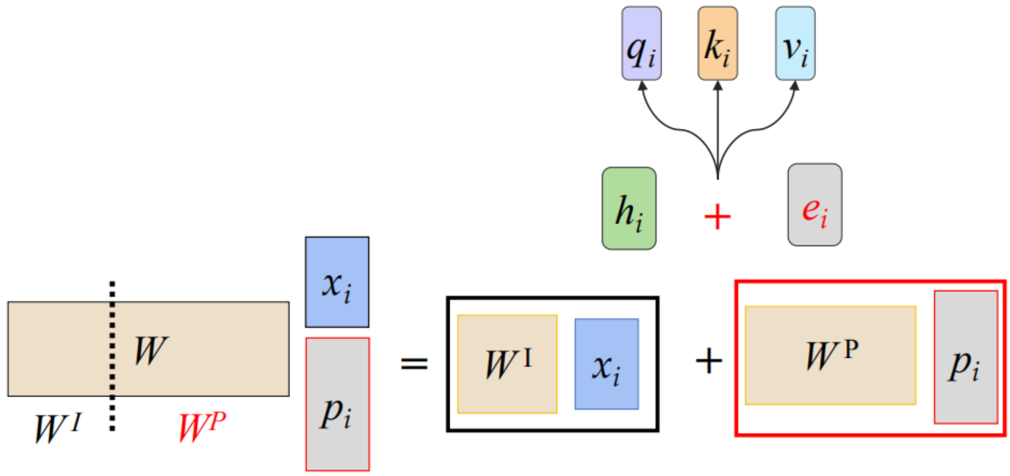
- Position Encoding: each position has a unique positional vector $\boldsymbol{e}_\mathrm{i}$ (typically learned from data)
- For each x_i we append a one-hot vector p_i indicating the position of the word in the sequence.

$$\boldsymbol{e}_i = \boldsymbol{W}^p\, p_i$$

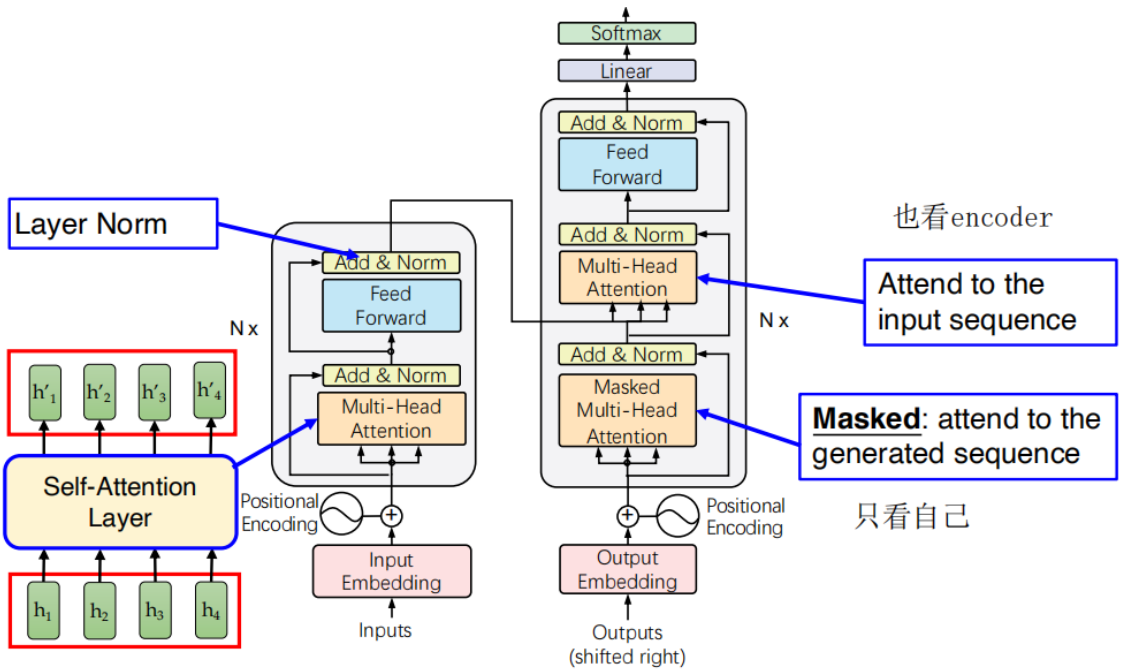




- The position encoding $\boldsymbol{e}_i$, combined with the word embedding h_i, is feed into the self-attention layer.



Transformer Architecture(P52)



Training(P64)

The Same as RNN Encoder-Decoder

Data: $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$,

where $x^{(\ell)} = (x_1, \ldots x_{Tx})$ and $y^{(\ell)} = (y_1, \ldots y_{Ty})$.

Loss Function – minimize the cross-entropy loss:

$$L(\theta) = -\frac{1}{N} \sum_{\ell=1}^{N} \sum_{t=1}^{T_y} \log p_\theta(y_t^{(\ell)} | x^{(\ell)})$$

Optimization – gradient descend

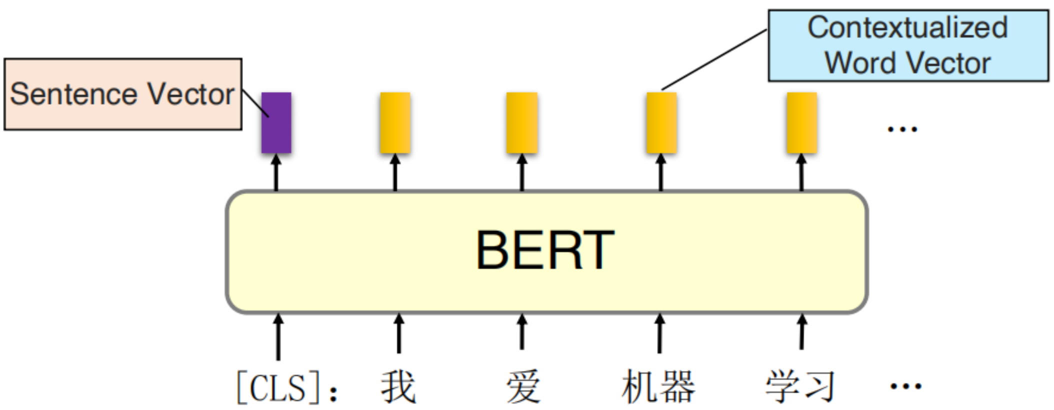
Pretrained Language Model

💡 **Pretrained Language Models def(P69):** 预训练模型，解决数据不够的问题，在公共的数据上先训练出一个模型（在大规模数据上面作无监督学习）

Bidirectional Encoder Representations from Transformers (BERT) overview(P72)

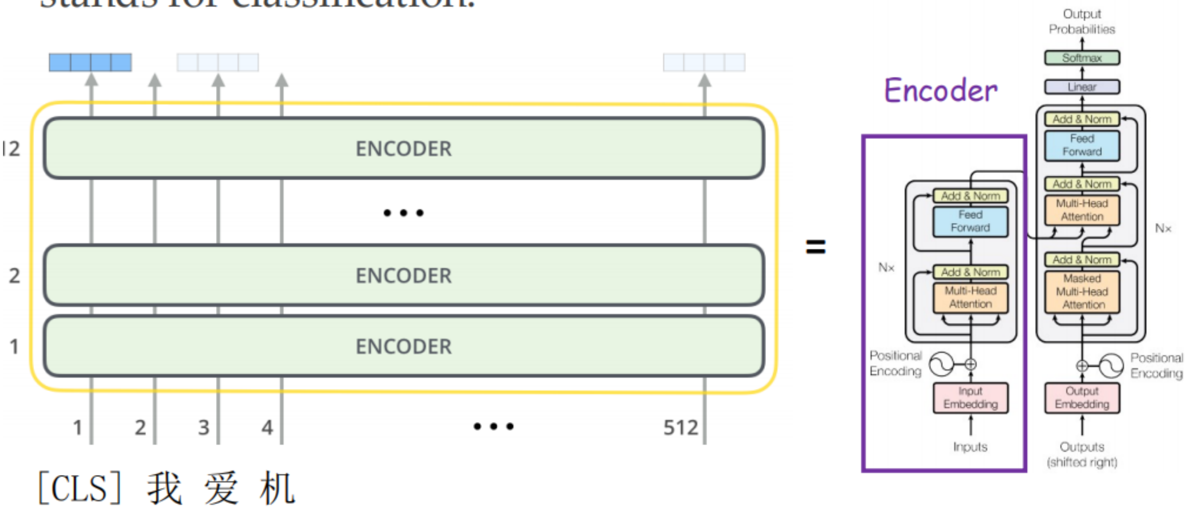
A Transformer Encoder that

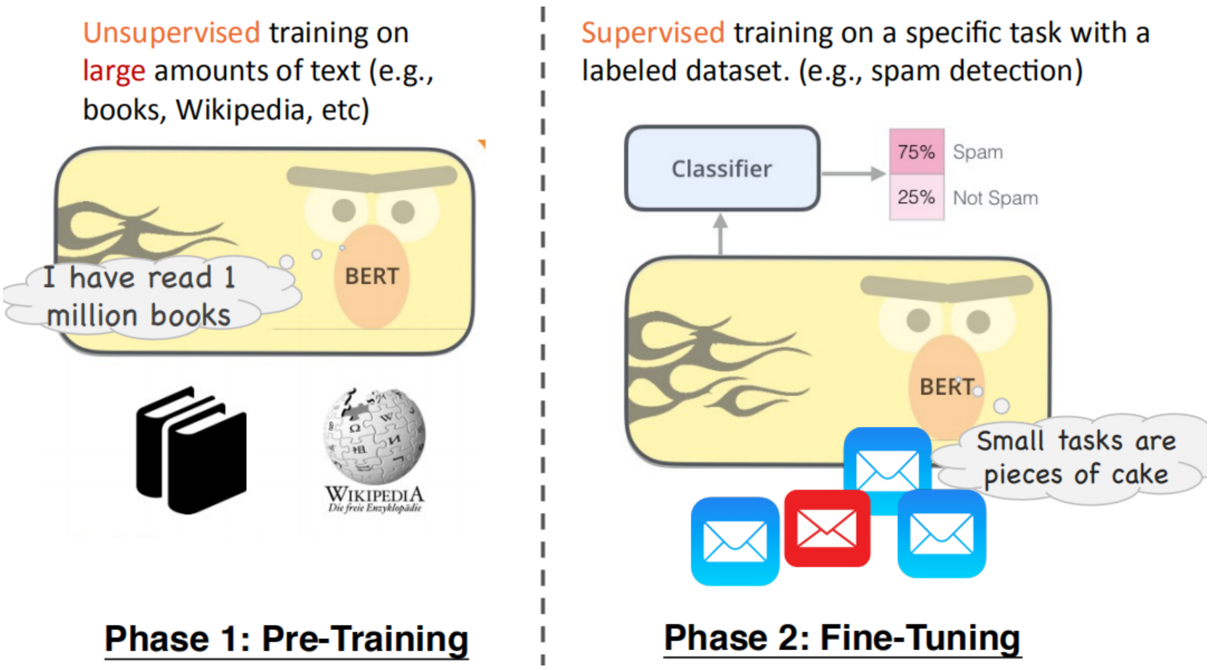
- allows for learning representations of words and sentences.
- pre-trained on large-scale text corpora and then（无监督）；
- fine-tuned on small task-specific datasets (e.g., classification, QA.)



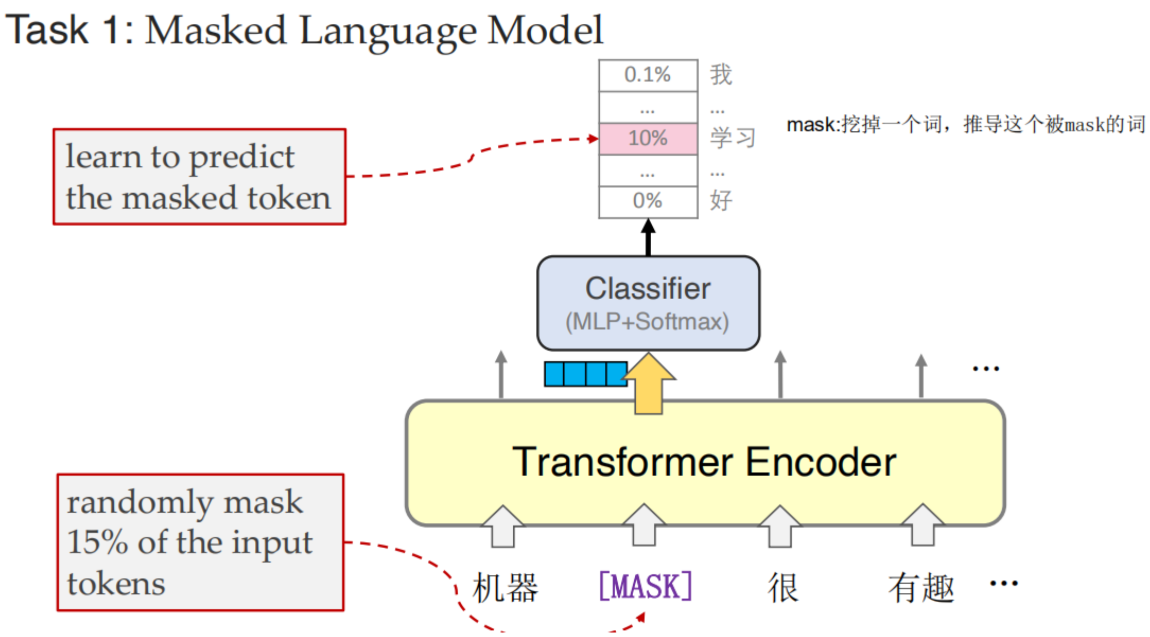
1. **Model Architecture**

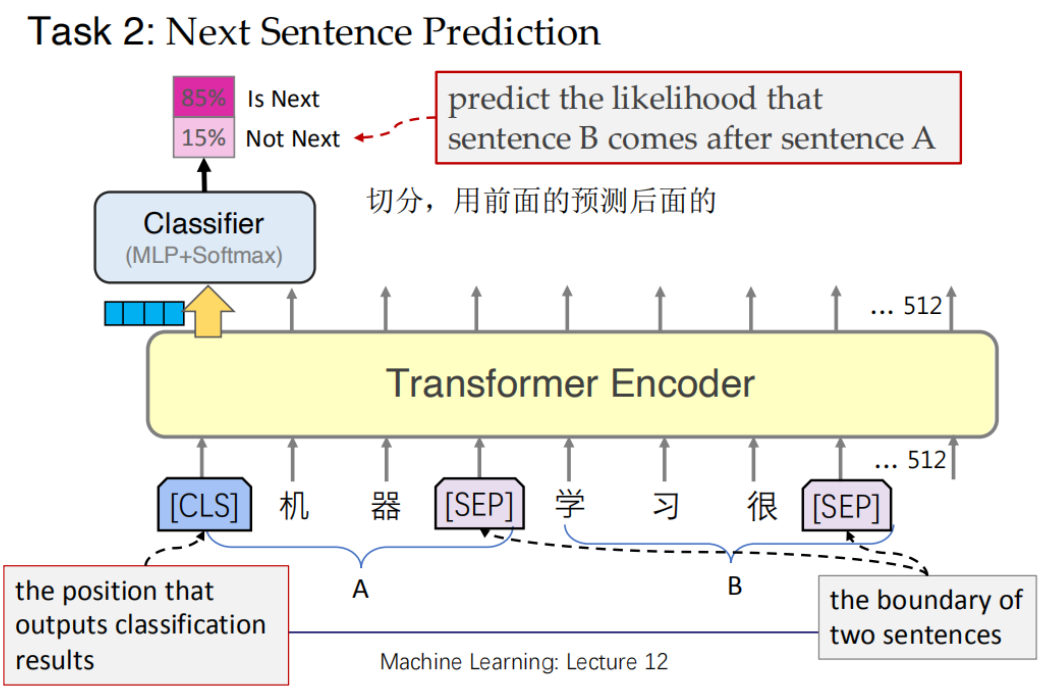
- Just like Transformer encoder, BERT takes a sequence of words as input which keep flowing up the stack.
- The first input token is always a special [CLS] token which stands for classification.





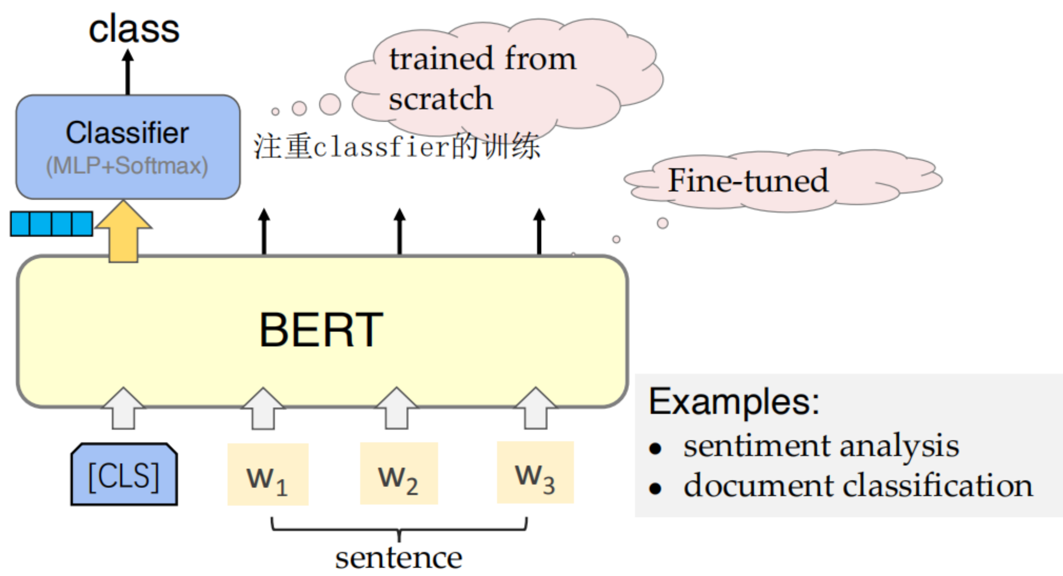
- **Pre-Training(P75-76)**



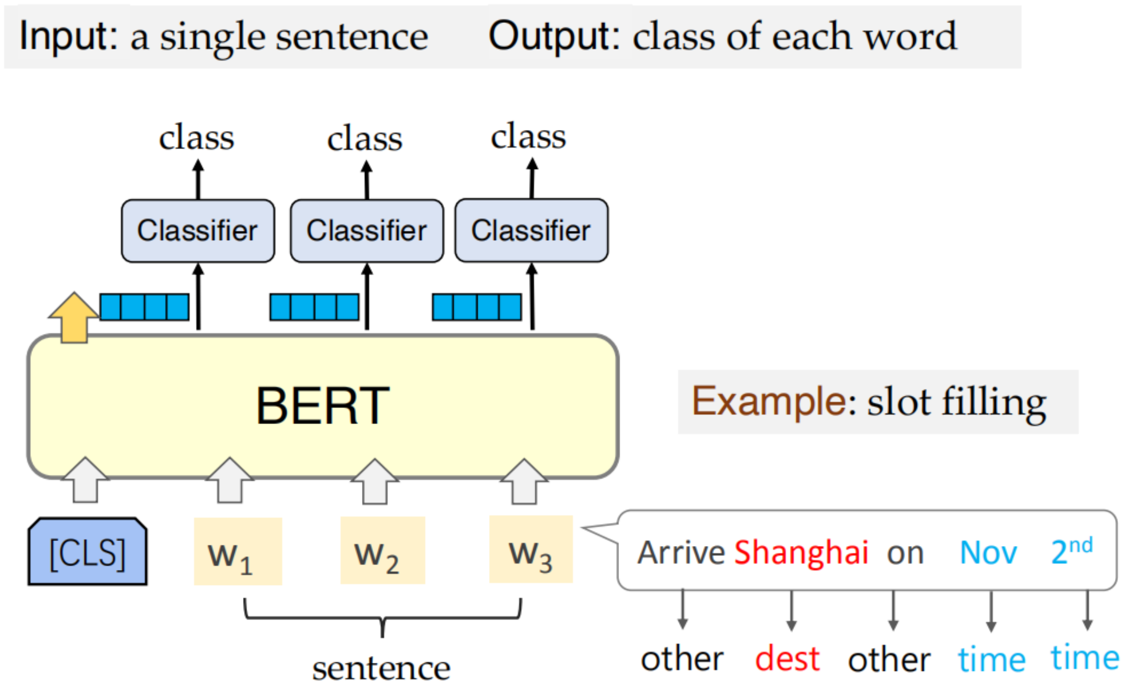


- **Finetuning(P77-79)**
    - **Sentence Classification**

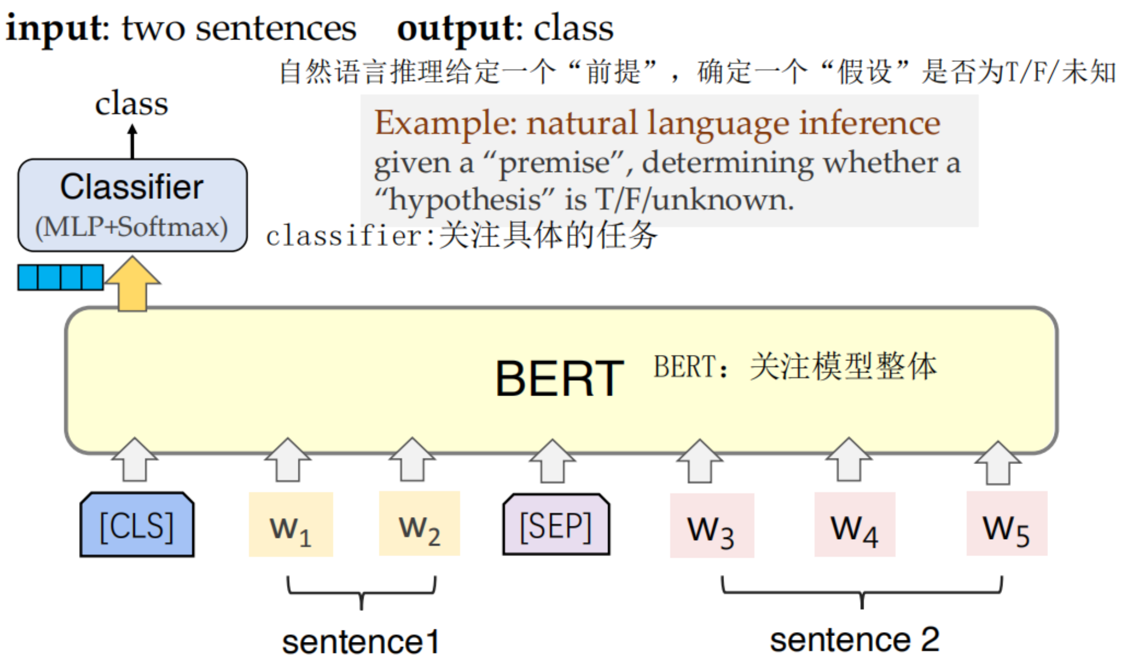
- **input**: a single sentence, **output**: class



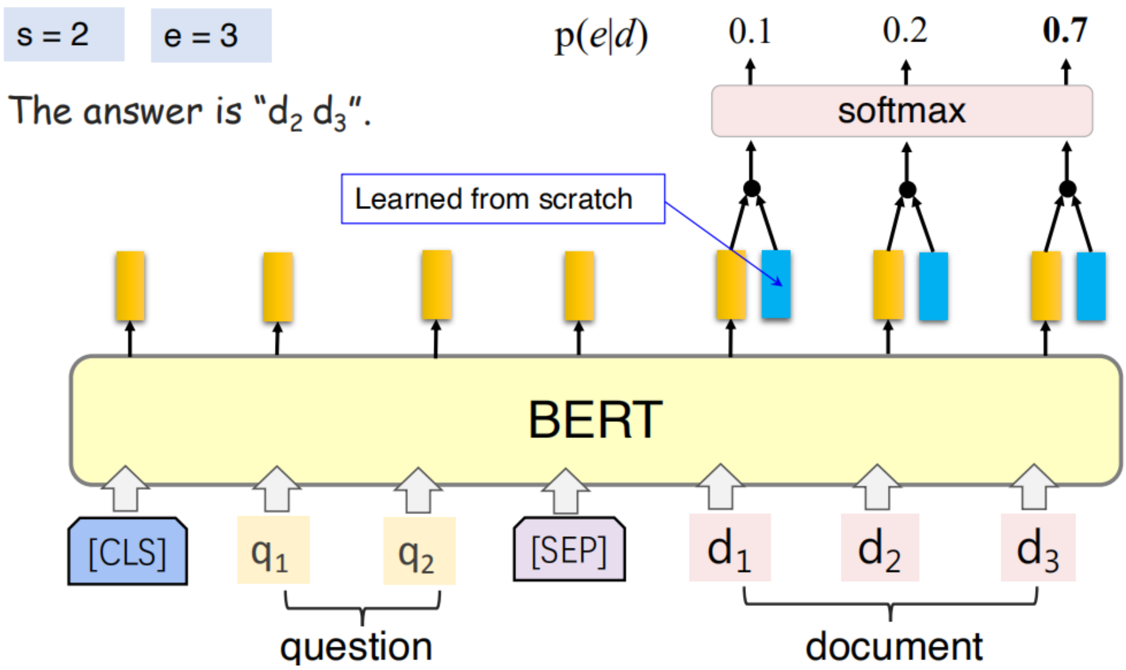
- **Word Tagging**



- **Classifying Sentence Pairs**



- QA (Reading Comprehension)



BART (Denoising Seq-to-Seq Pretraining)（P86）

- An **encoder-decoder** architecture 既用encoder也用decoder，来预训练
- Pre-training by reconstructing inputs that are **corrupted** by **5** methods: (token masking, token deletion, text infilling, sentence permutation, document rotation)
- More efficient for sequence-to-sequence tasks (e.g., generation, translation, comprehension)