Machine Learning

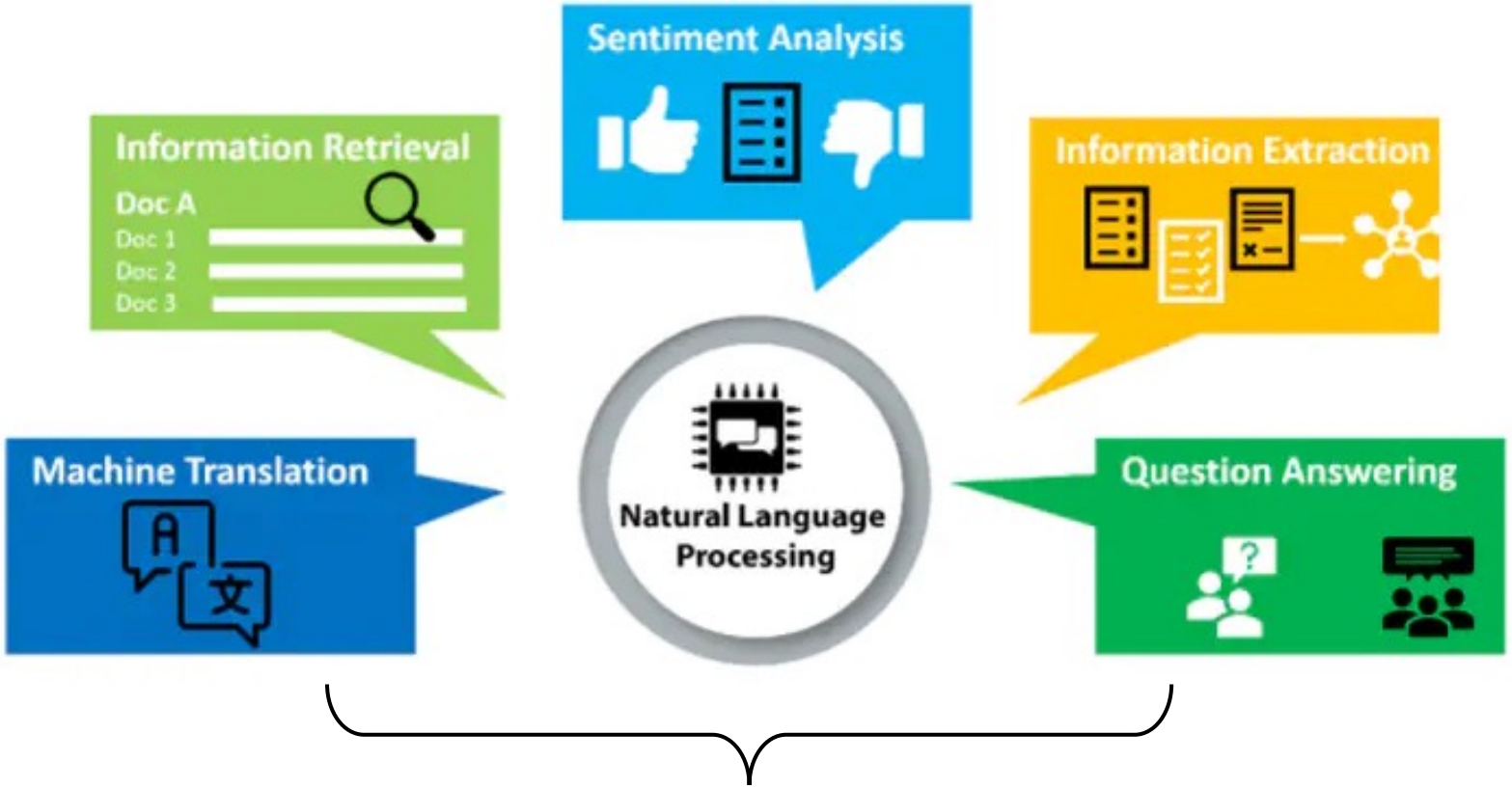
Lecture 12: Deep Language Modeling

Fall 2023

Instructor: Xiaodong Gu

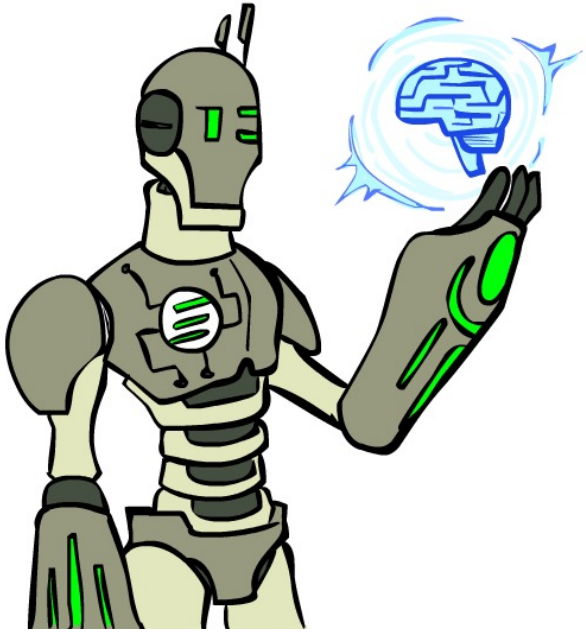
Natural Language Processing (NLP)

Language Understanding and Generation



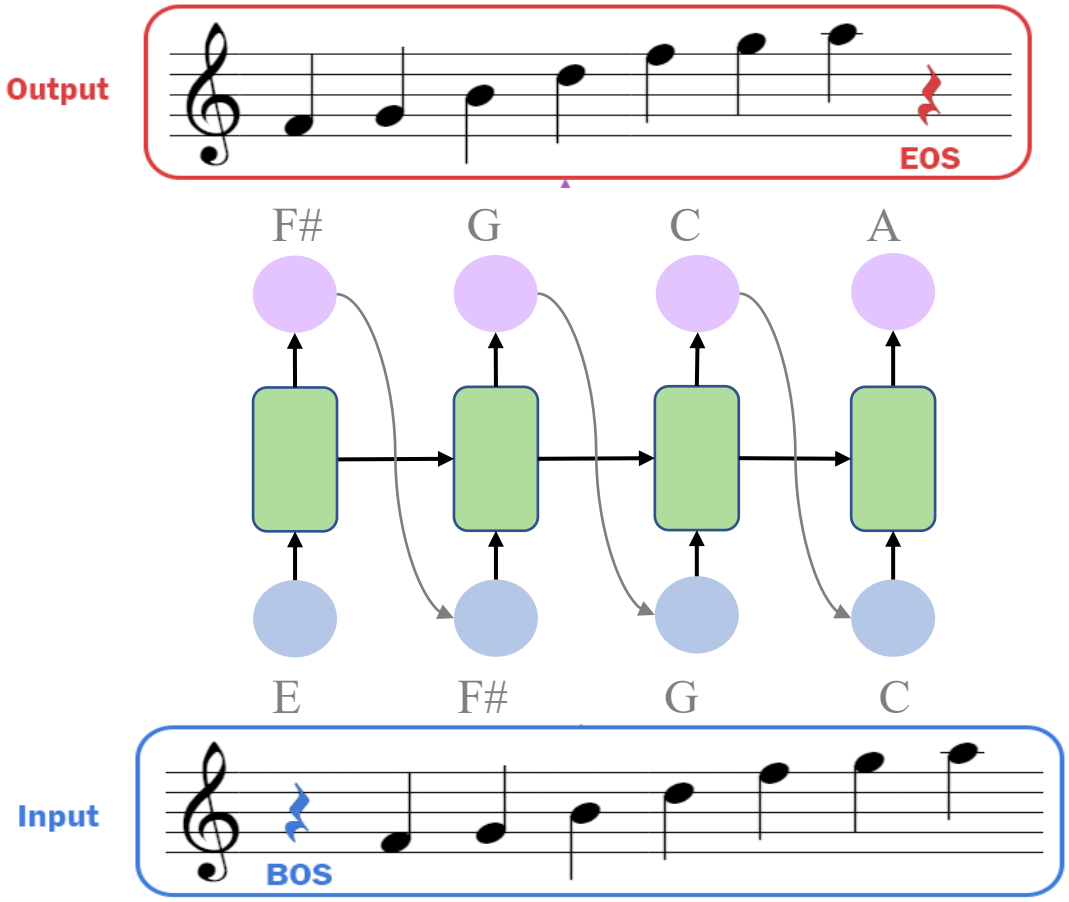
Today

- RNN Encoder-Decoder
- Attention
- Transformer
- Pretrained Language Models



Sequence-to-Sequence Learning

Recall: Sequence Generation Using RNNs



Conditional Sequence Generation

- generate a (**target**) sequence given a (**source**) sequence.

ASR

Generator

How are you

Translation

机器学习

Generator

Machine Learning

Dialogue

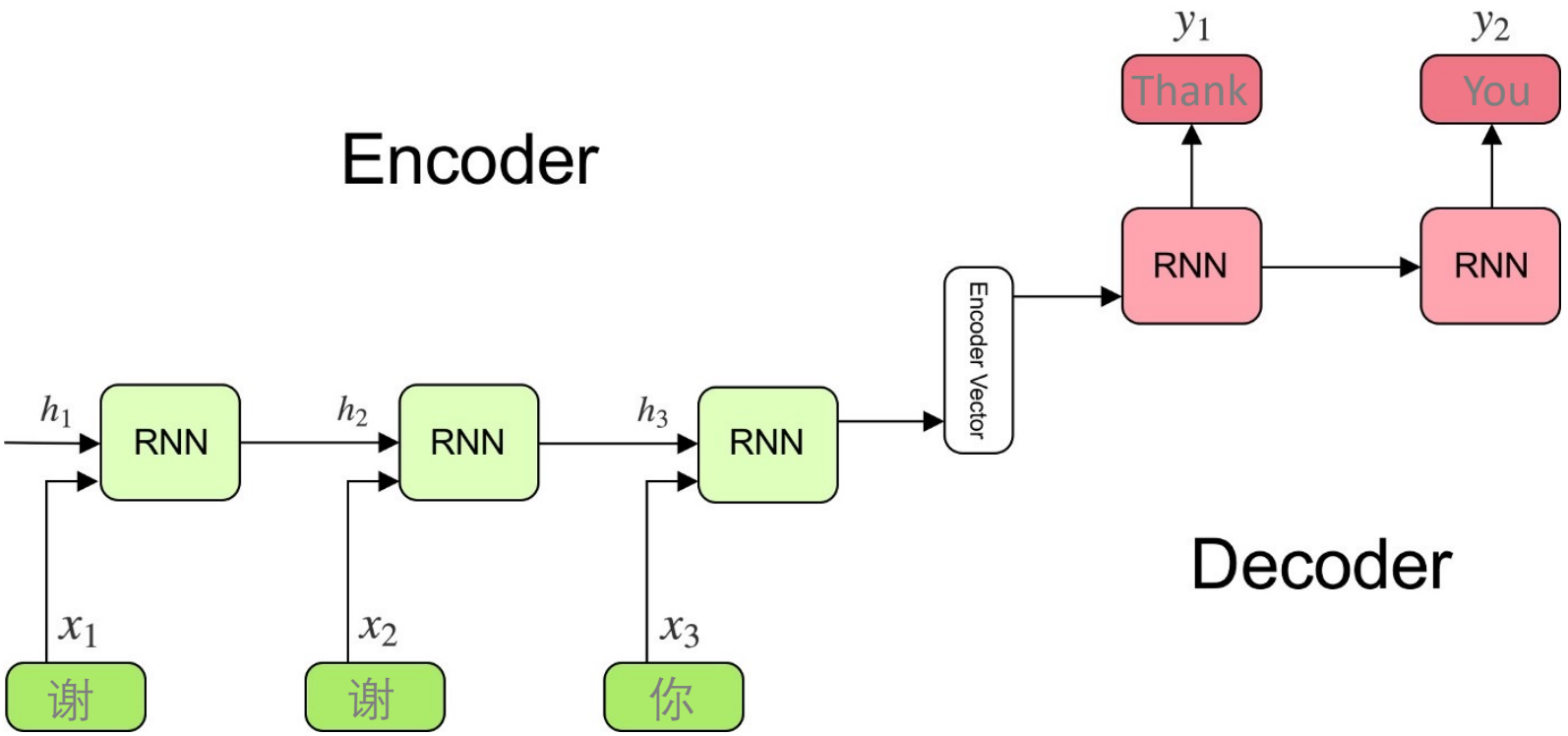
How are you

Generator

I am fine

RNN Encoder-Decoder (Cho 2014)

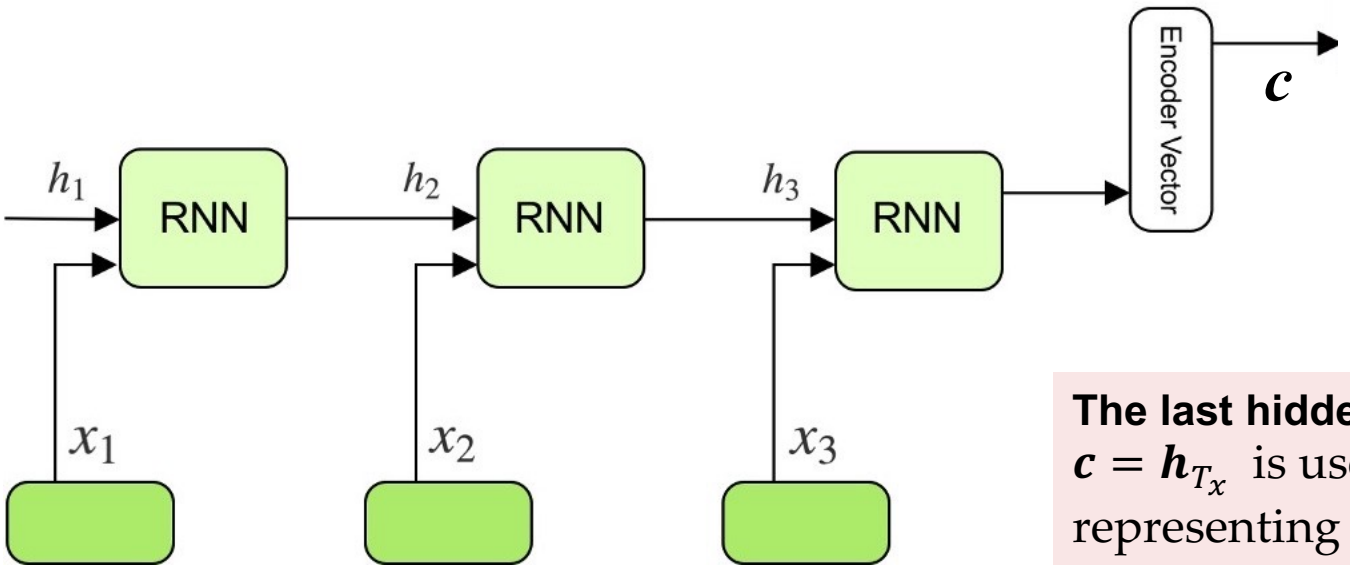
- given $x = (x_1, \cdots, x_{T_x})$, generate $y = (y_1, \cdots, y_{T_y})$



Encoder

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

$$\boldsymbol{h}_t = \text{RNN}_{\text{in}}(x_t, \boldsymbol{h}_{t\text{-}1})$$



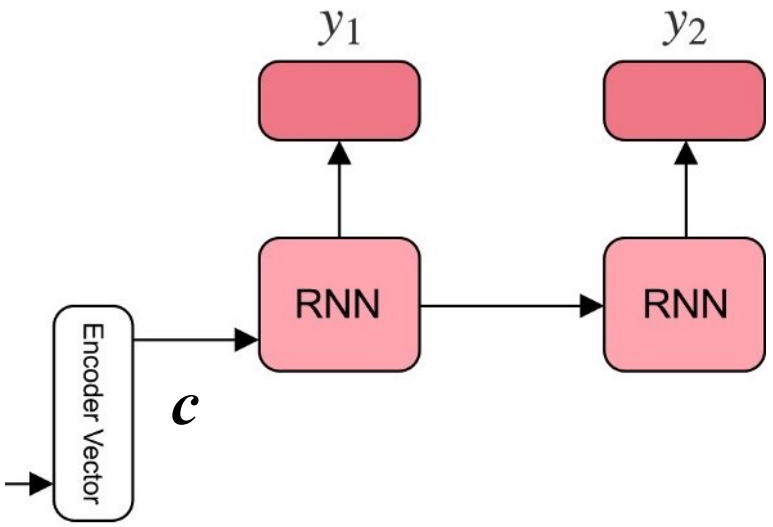
The last hidden state $\boldsymbol{c} = \boldsymbol{h}_{T_x}$ is used for representing x

Decoder

- 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.

$$\boldsymbol{s}_i = \text{RNN}_{\text{out}}(y_{i-1}, \boldsymbol{s}_{i-1}, \boldsymbol{c})$$

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



Training

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

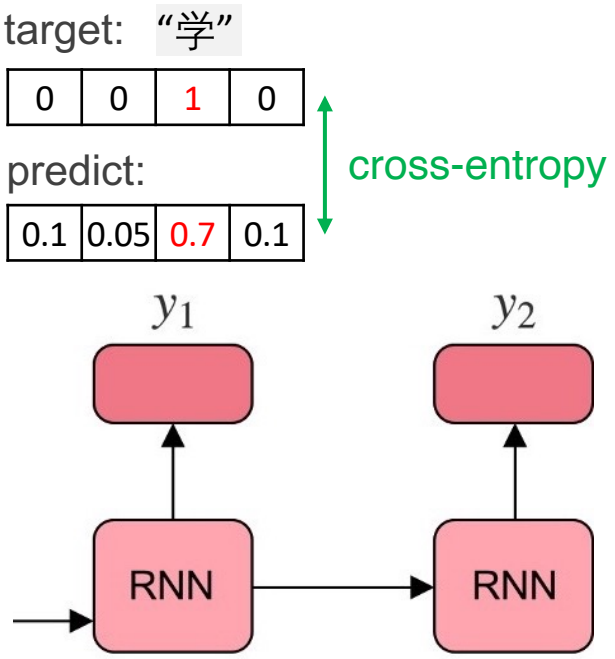
where $x^{(\ell)} = (x_1, \ldots x_{Tx})$ and $y^{(\ell)} = (y_1, \ldots y_{Ty})$.

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

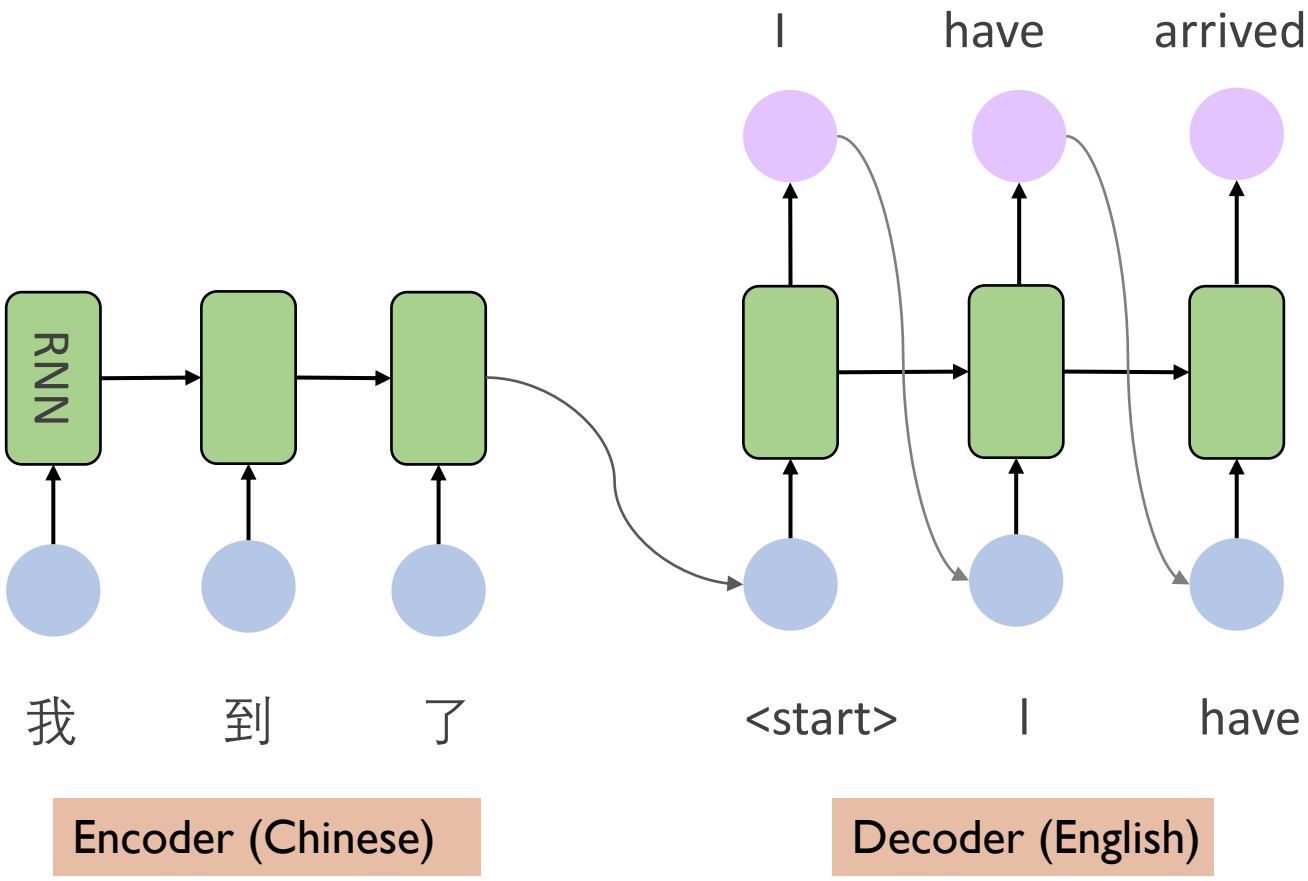
$$l(\theta|x, y) = -\sum_{t=1}^{T} \log p_\theta(y_t|y_1, \ldots, y_{t-1}, x)$$

$$L(\theta|D) = -\frac{1}{N}\sum_{l=1}^{N} l(\theta|x^{(l)}, y^{(l)})$$

Optimization – gradient descend



Example: Machine Translation



TIME for Some Coding

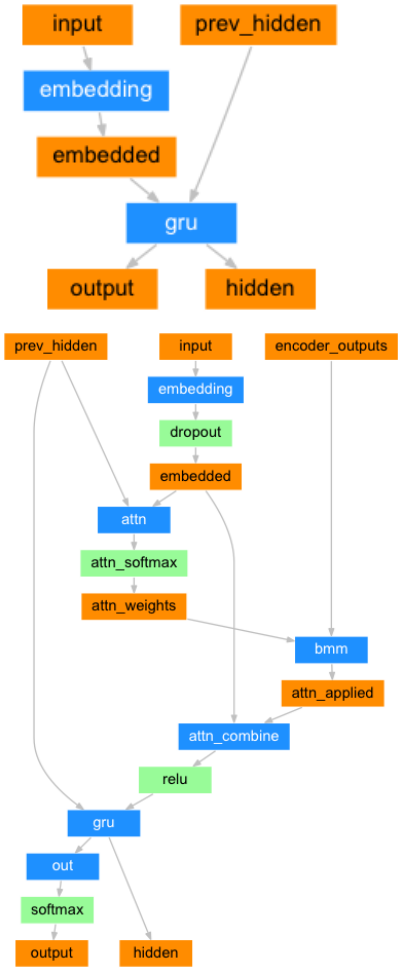
Tutorial: machine translation

https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

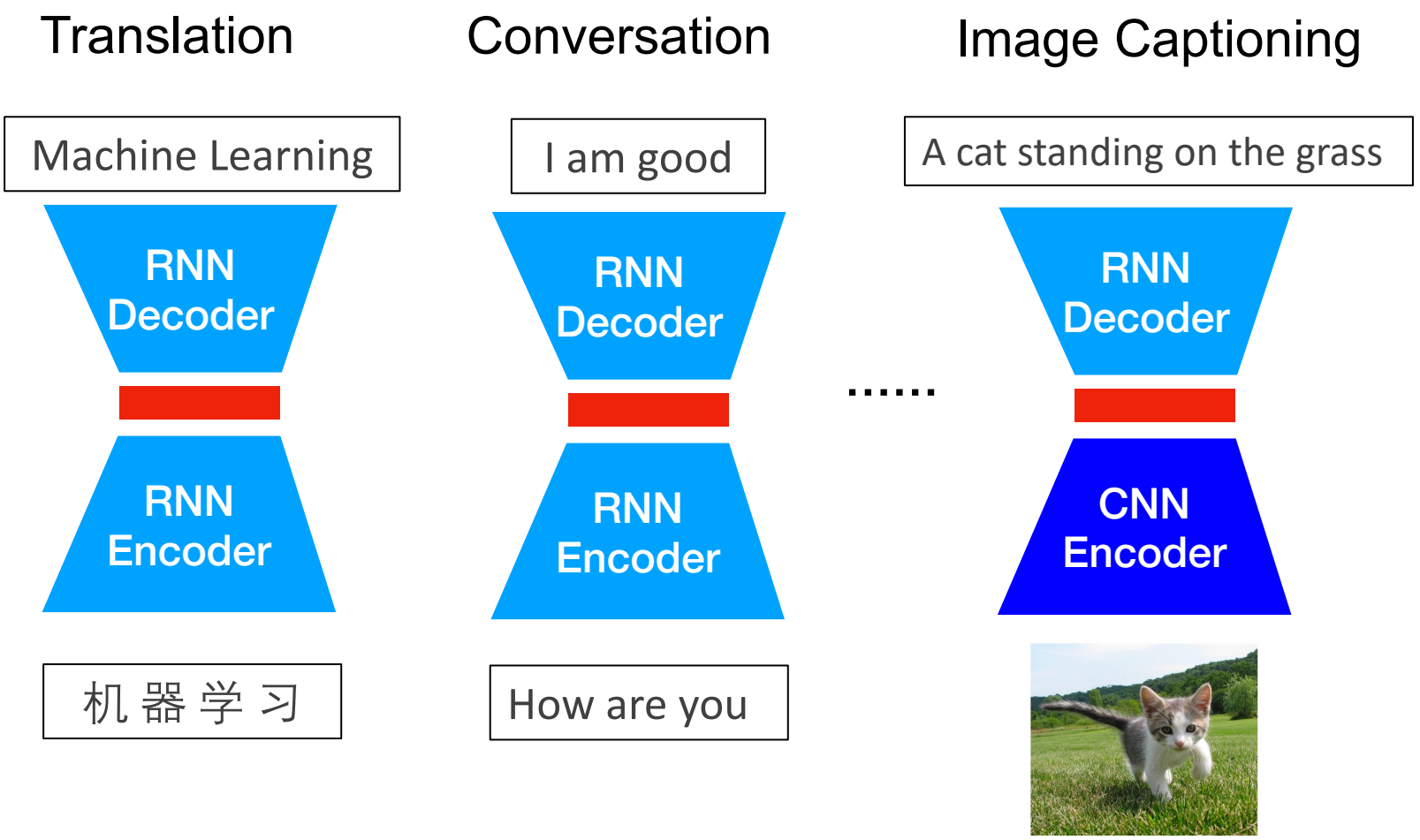
Tutorial: Machine Translation

```
class Encoder(nn.Module):
    def __init__(self, input_size, hidden_size):
        self.embedding = nn.Embedding(input_size, hidden_size)
        self.rnn = nn.LSTM(hidden_size, hidden_size)
    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output, hidden = self.rnn(embedded, hidden)
        return output, hidden
```

```
class Decoder(nn.Module):
    def __init__(self, hidden_size, vocab_size):
        self.embedding = nn.Embedding(output_size, hidden_size)
        self.rnn = nn.LSTM(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, vocab_size)
    def forward(self, input, c, hidden):
        input = self.embedding(input).view(1, 1, -1)
        output, hidden = self.rnn([input,c], hidden)
        output = self.out(output[0]))
        return output, hidden
```

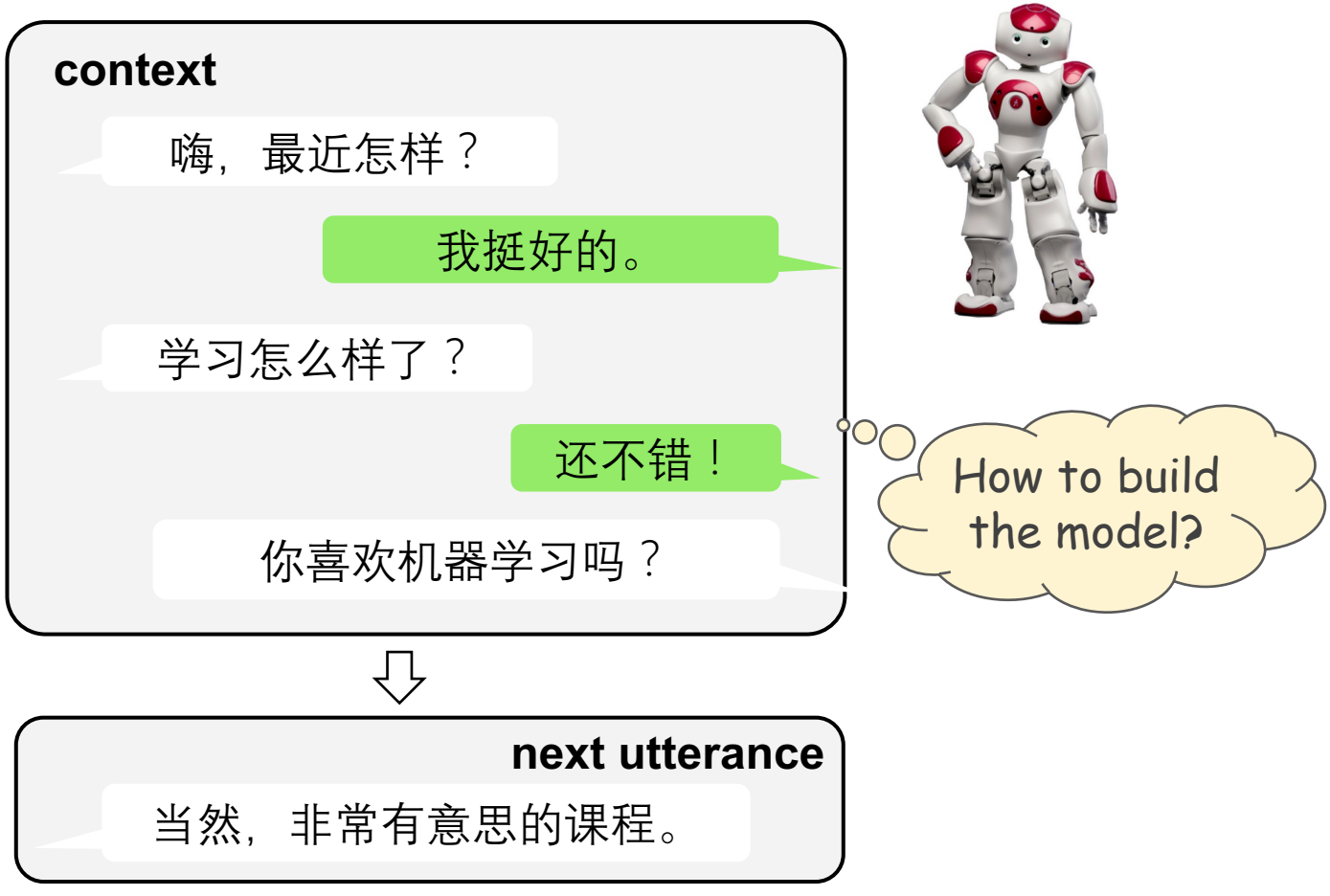


Applications



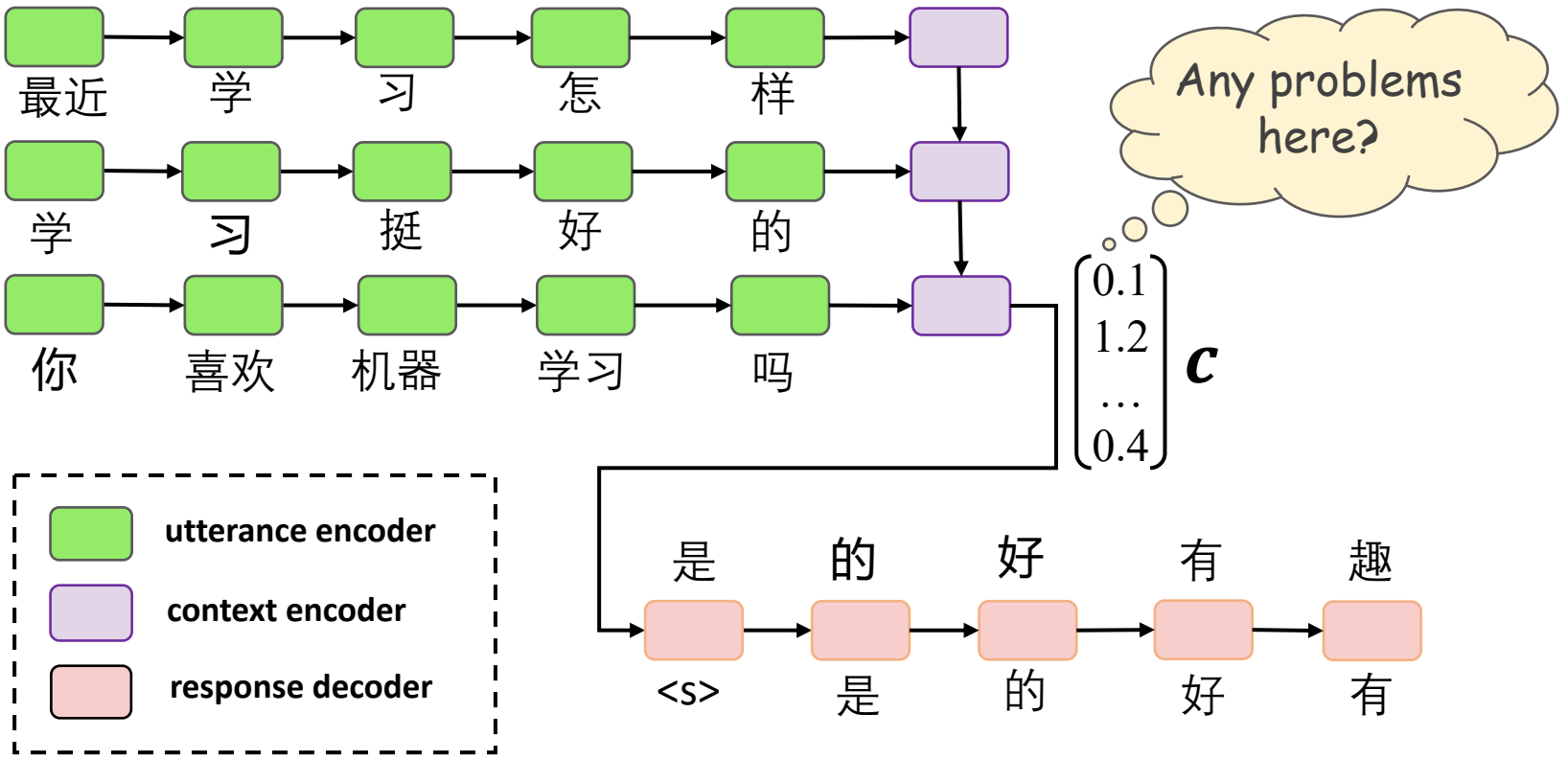
Example: Chatbot

- Open-Domain Dialog Generation



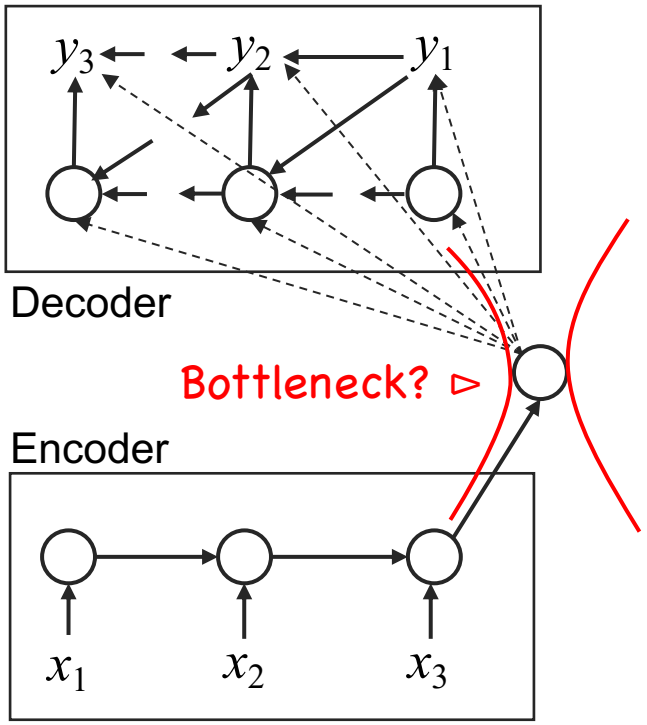
Example: Chatbots

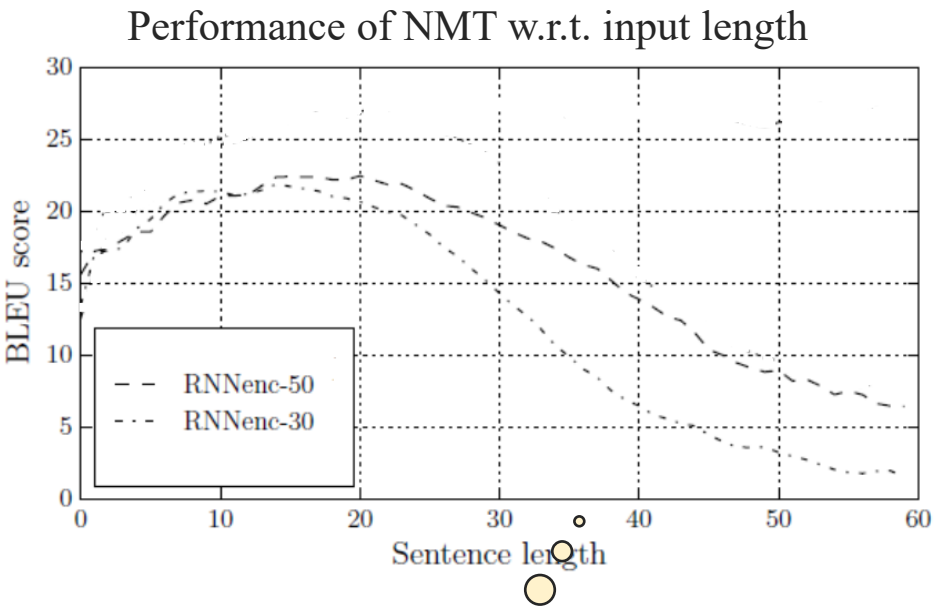
- **Hierarchical** RNN Encoder-Decoder



Intermediate Representation: One Size Fits All?

Limitation: the entire sequence is compressed into **one** vector, regardless of the importance of each position.



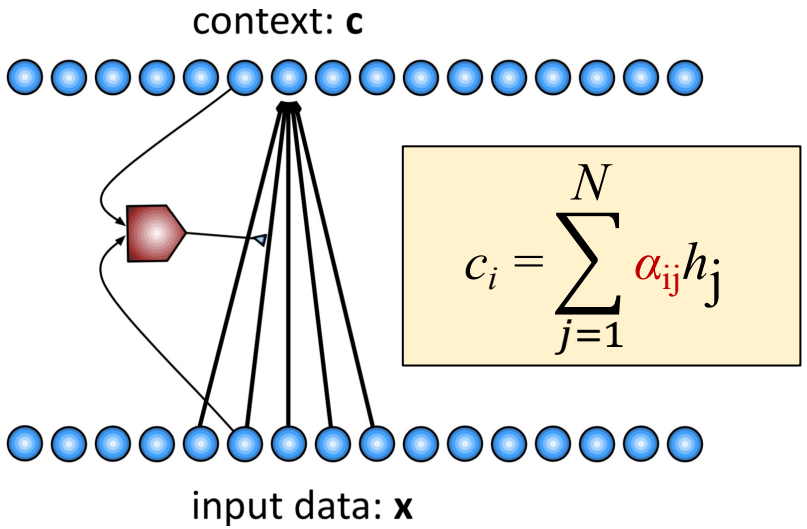


Performance decreases as the sequence length grows.

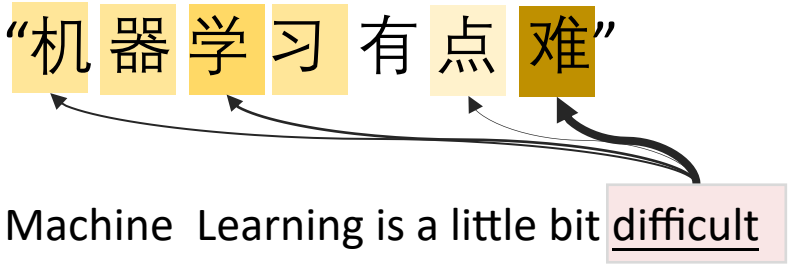
How to Mitigate "the Curse of Length"?

Idea: use dynamic context vectors

- When decoding each y_i in $y = (y_1, \cdots, y_T)$, use a **dynamic context vector** c_i which is a linear combination of hidden states at different positions of x.



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

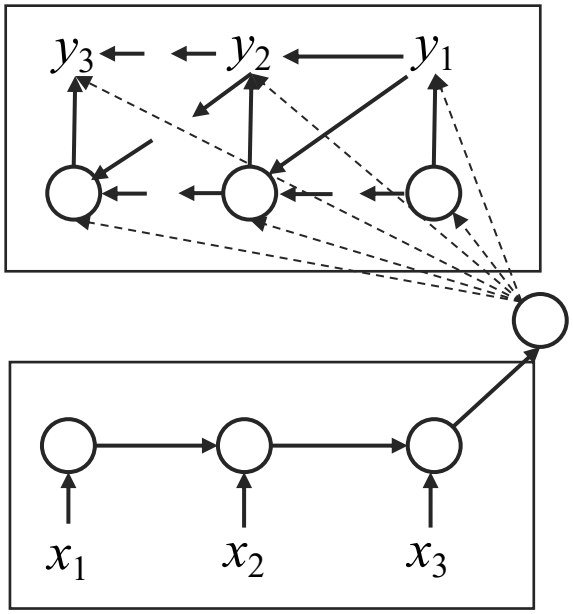


Sequence-to-Sequence with Attention

No Attention

decoding based on a single, fixed encoding vector

$h_i = \text{RNN}_{\text{out}}(h_{i-1}, y_{i-1}, \boldsymbol{c})$

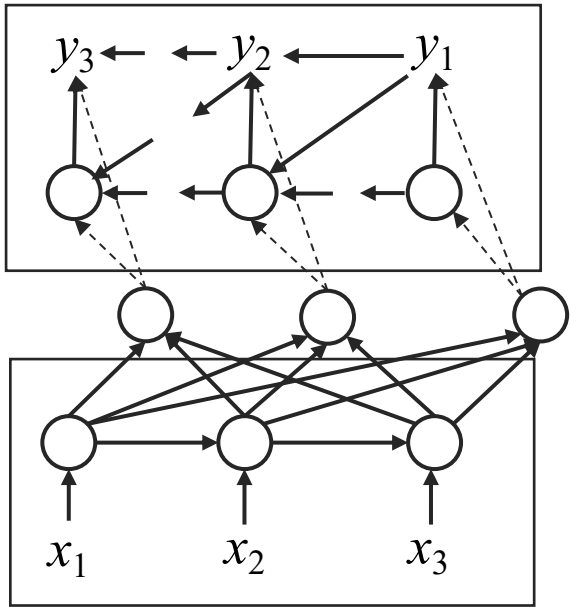


VS

With Attention

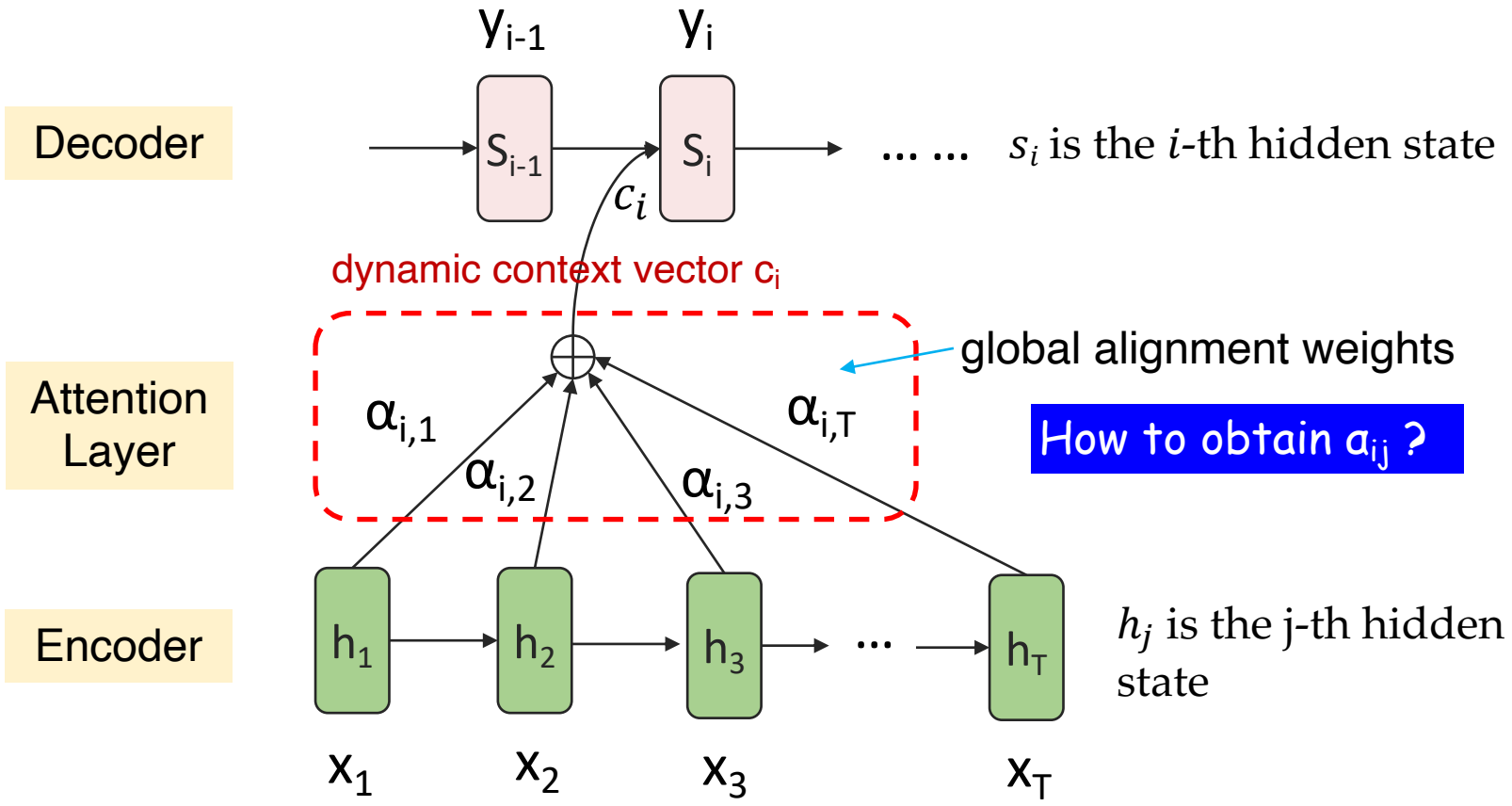
select what to encode in a contextual manner.

$h_i = \text{RNN}_{\text{out}}(h_{i-1}, y_{i-1}, \boldsymbol{c_i})$



Sequence-to-Sequence with Attention

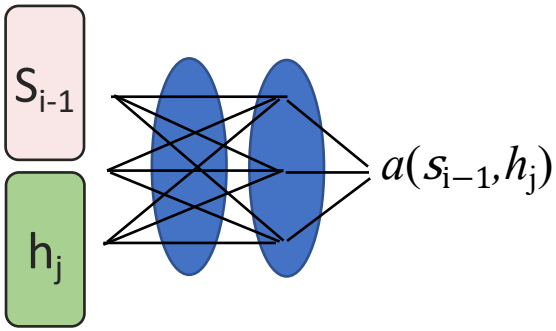
The decoder **dynamically** pays attention to different tokens in the source sentences during decoding.

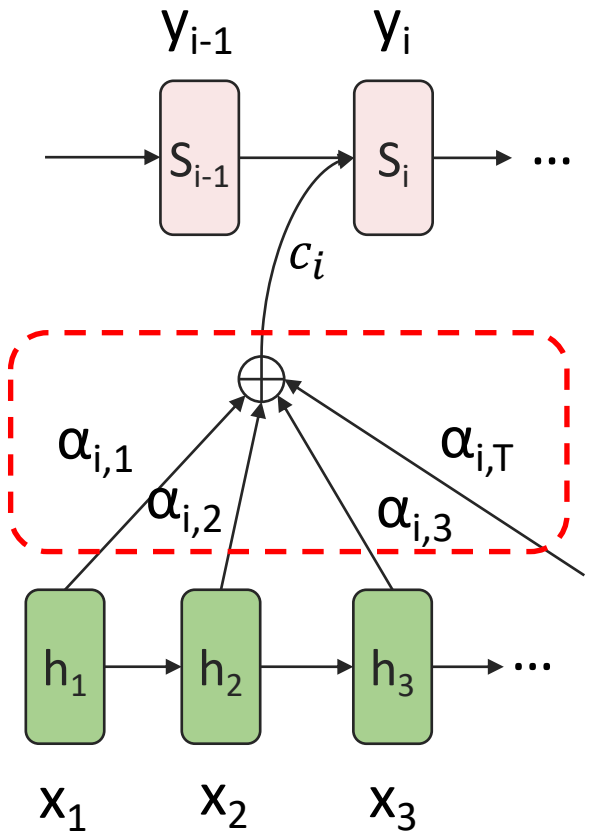


Attention-based Model

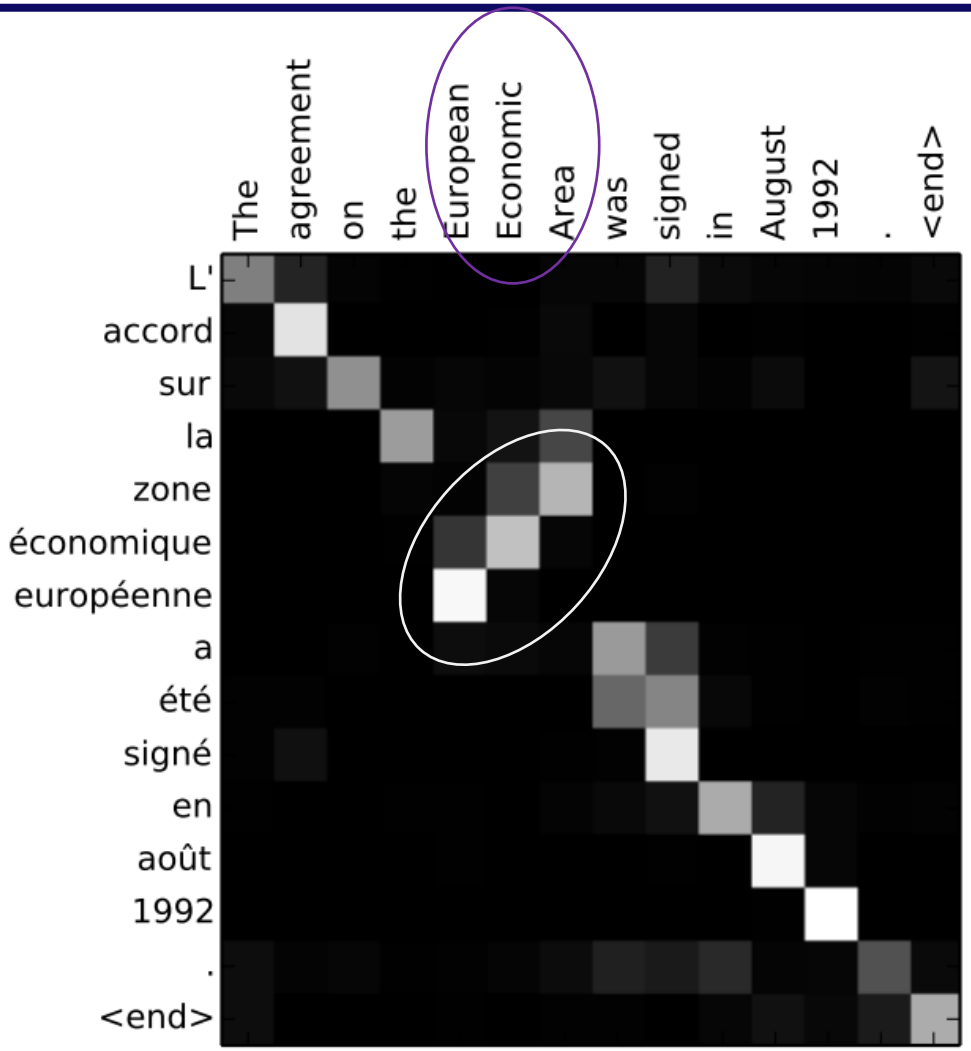
$$\alpha_{ij} = \frac{\exp\left(a(s_{i-1}, h_j)\right)}{\sum_{k=1}^{N} \exp\left(a(s_{i-1}, h_k)\right)}$$

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





Visualizing Attention



real α_{ij}'s in machine translation

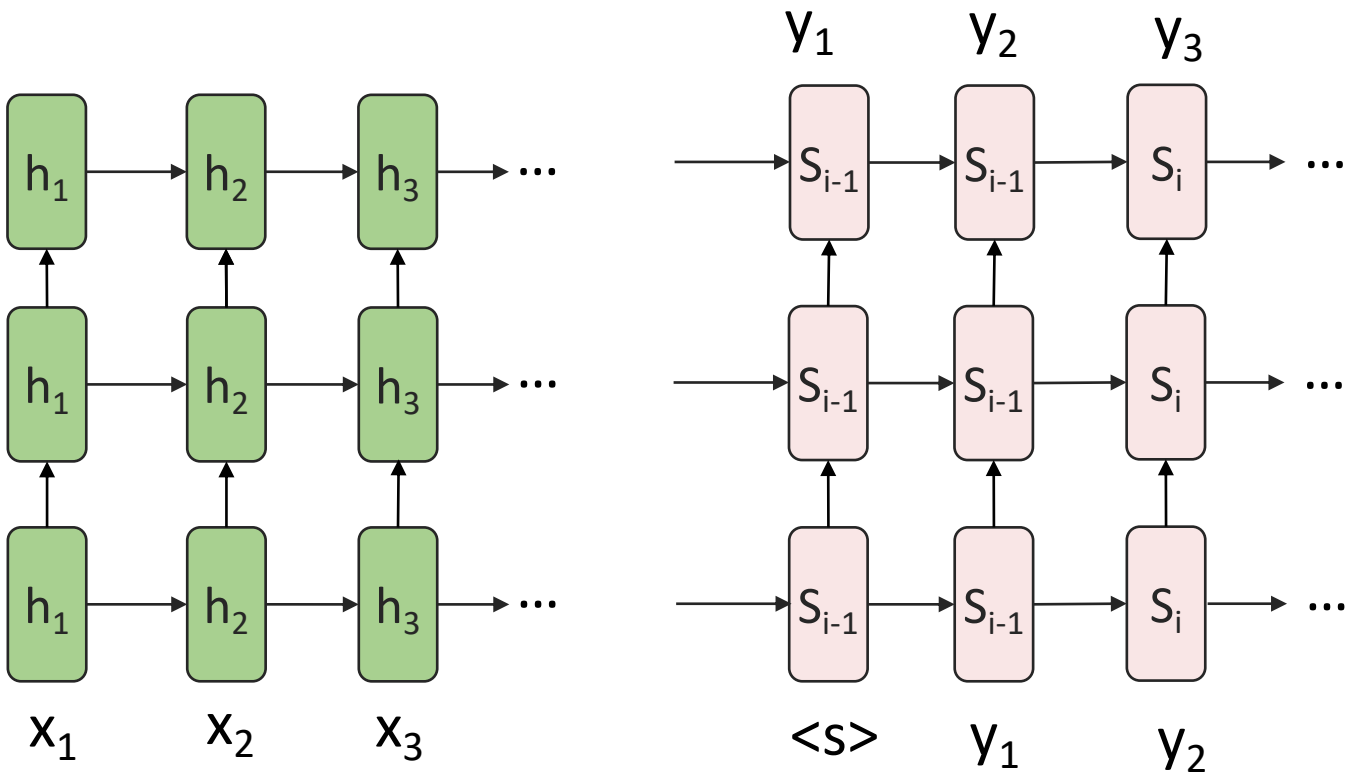
You can see how the model paid attention correctly when outputting "European Economic Area". In French, the **order of these words is reversed** ("européenne économique zone") as compared to English. Every other word in the sentence is in similar order.

Transformer

Seq2seq model with "Self-attention"

Limitations of RNN

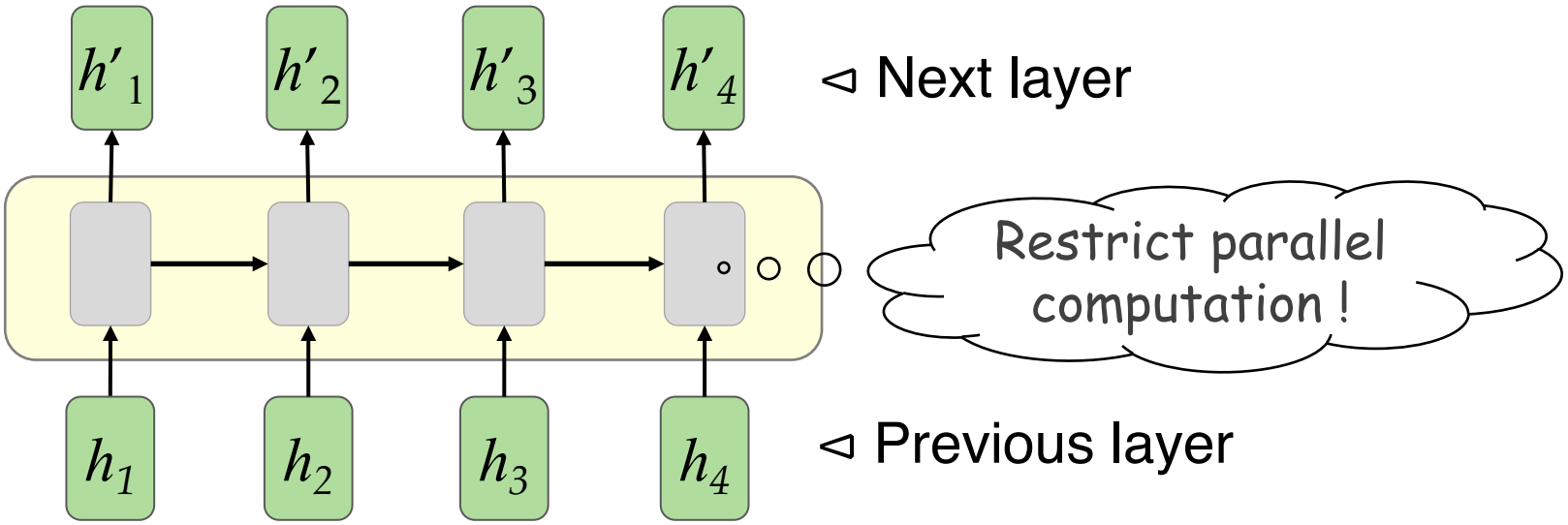
- Multilayer RNNs



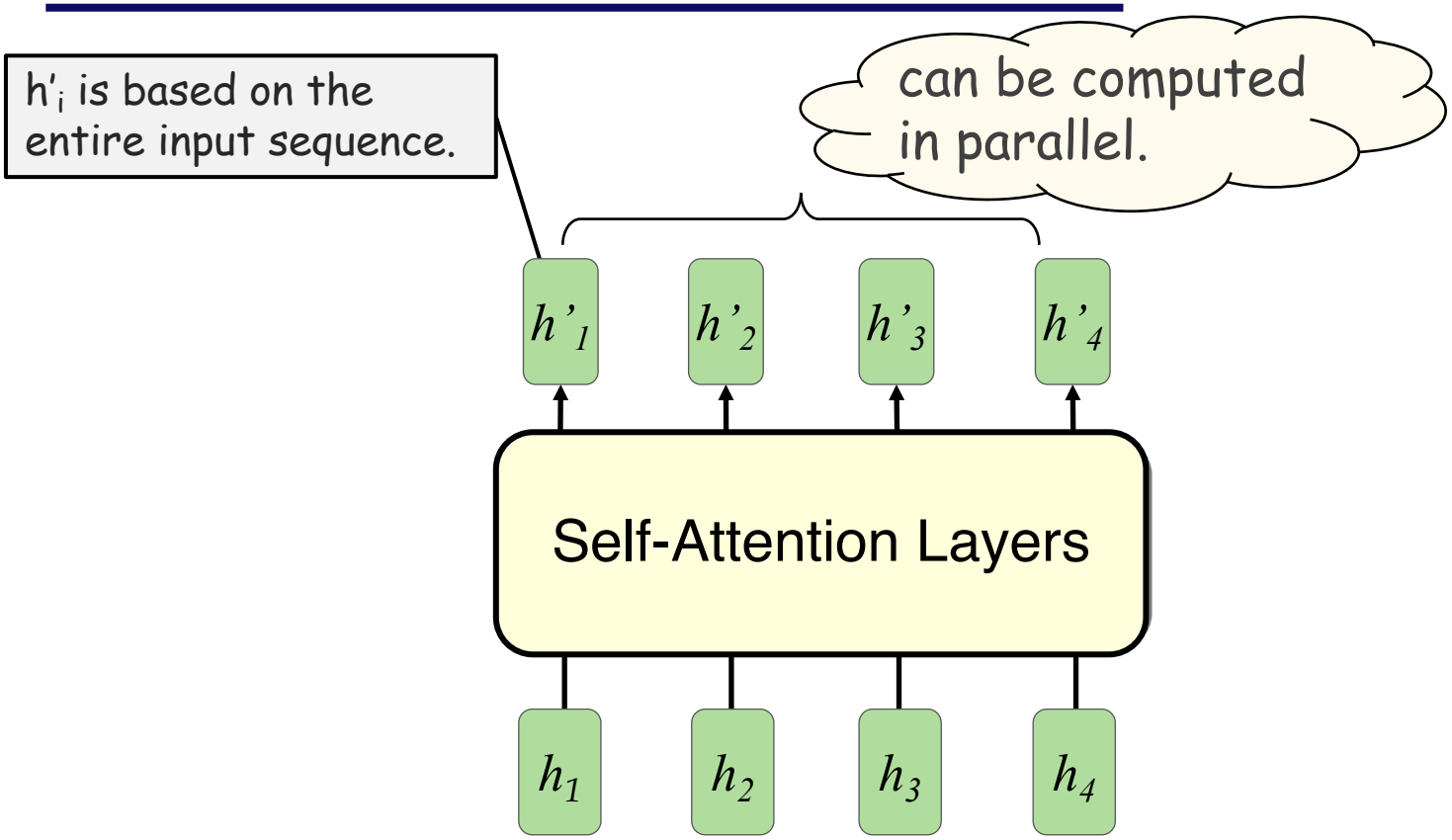
Q: How many *for* loops ?

Limitations of RNN

- **Problem :** the hidden state for one position is dependent on the computation of the preceding position.



From RNN to Self-Attention



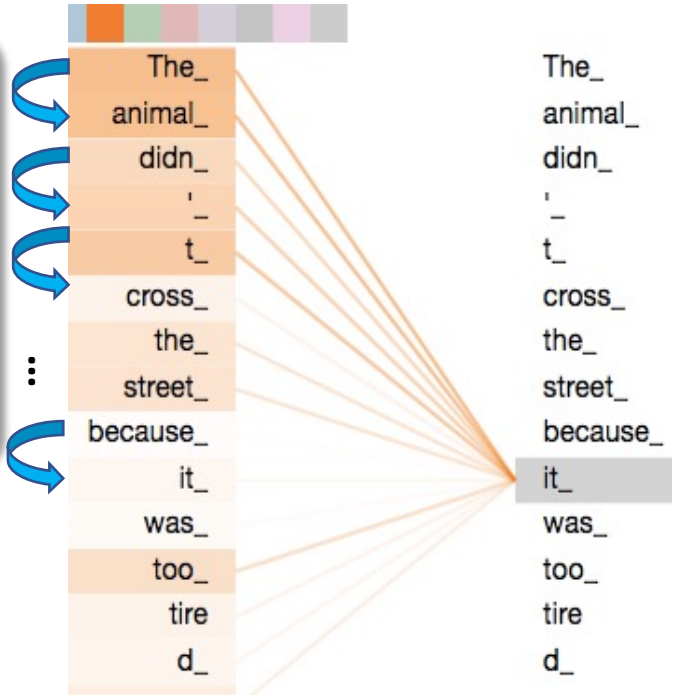
Farewell RNN: We can replace anything that can be processed by an RNN with a self-attention network.

From RNN to Self-Attention

What does "**it**" in this sentence refer to?

RNN

maintain a hidden state to sequentially incorporate previous words with the current one it's processing.

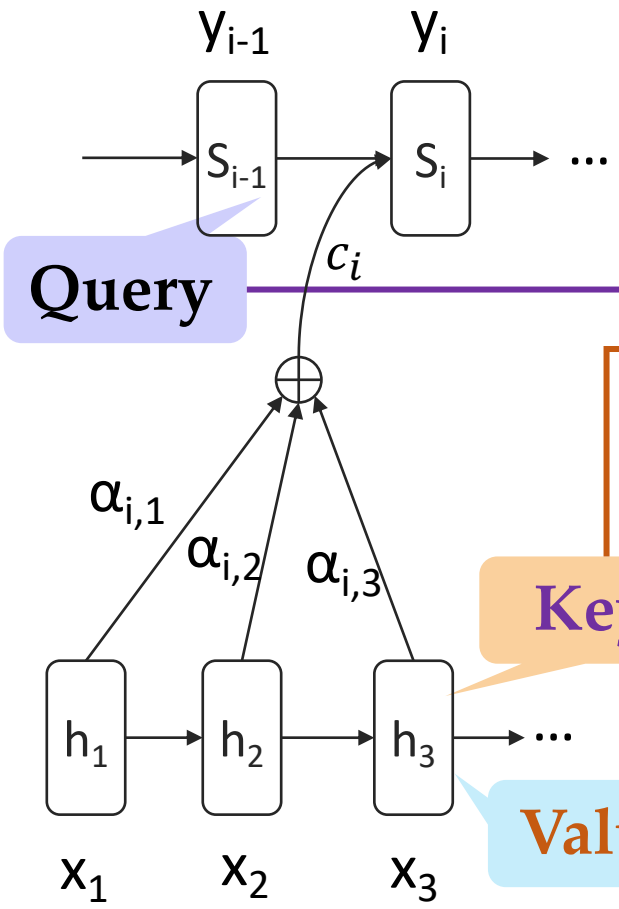


Self-Attention

bake the "understanding" of other relevant words into the one we're currently processing simultaneously.

The animal didn't cross the street because **it** was too tiered.

Attention Revisited



q: query (to match others)

k: key (to be matched)

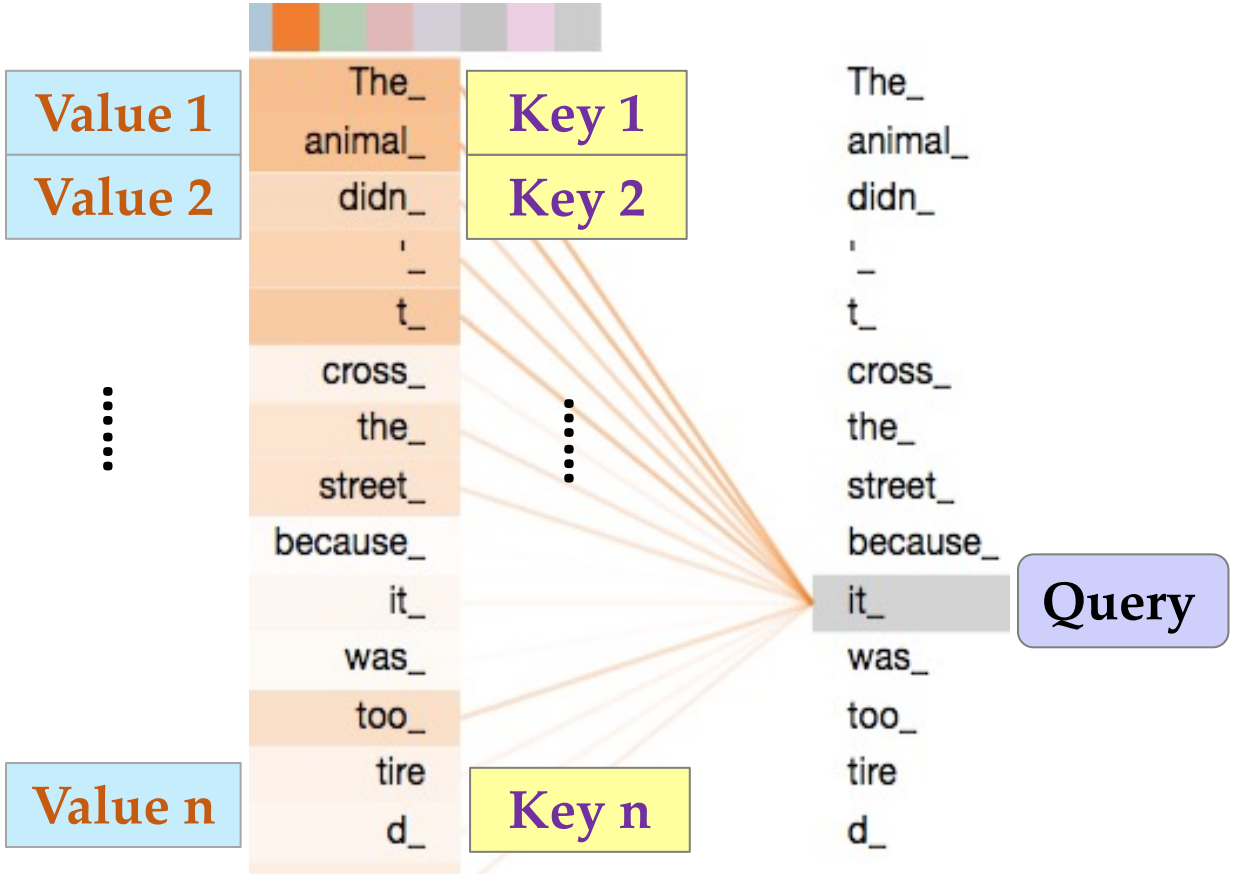
v: content to be retrieved

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

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

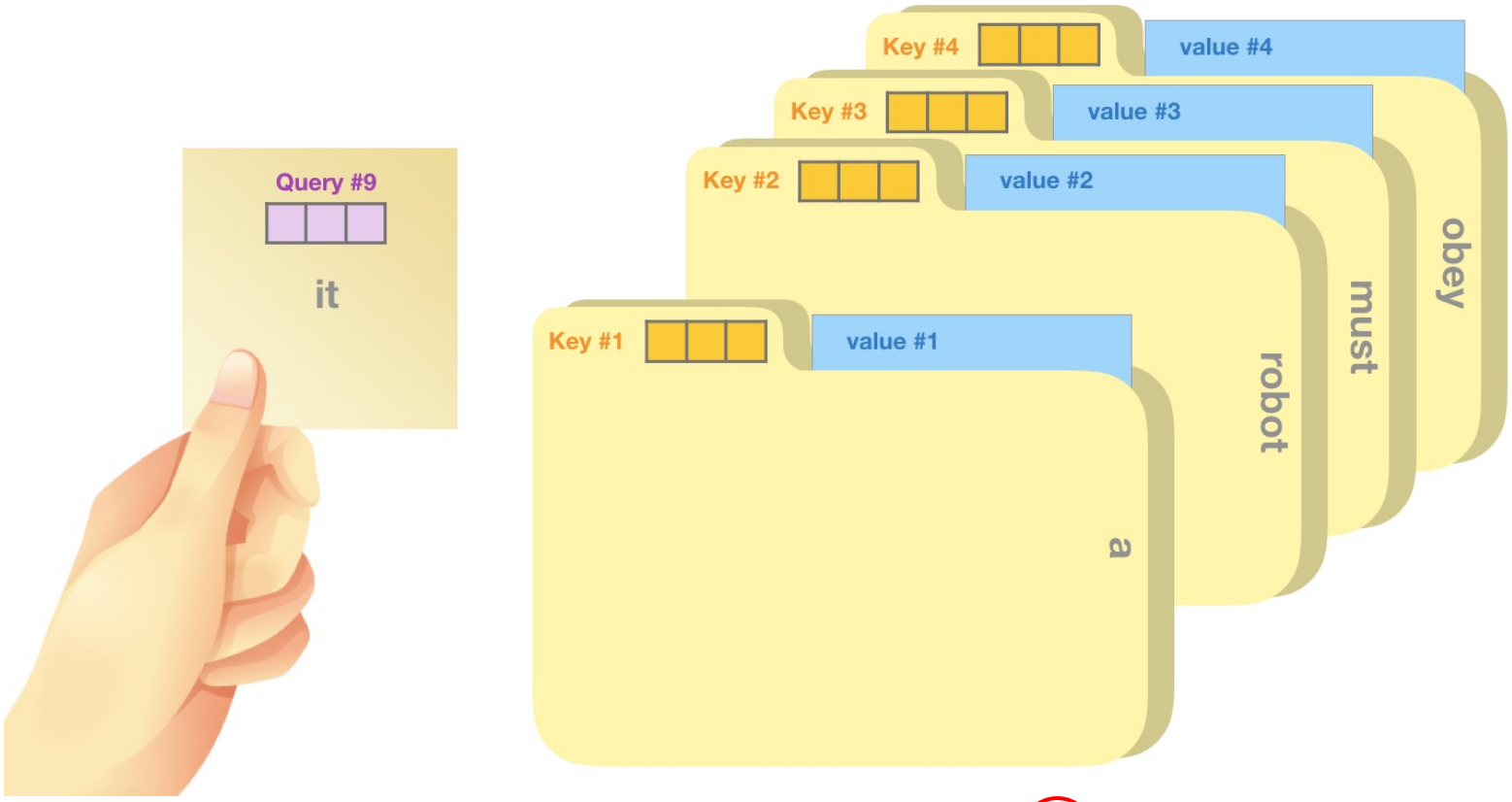
Self-Attention: The Idea

- Let each word pay attention to all other words.



Self-Attention: The Idea

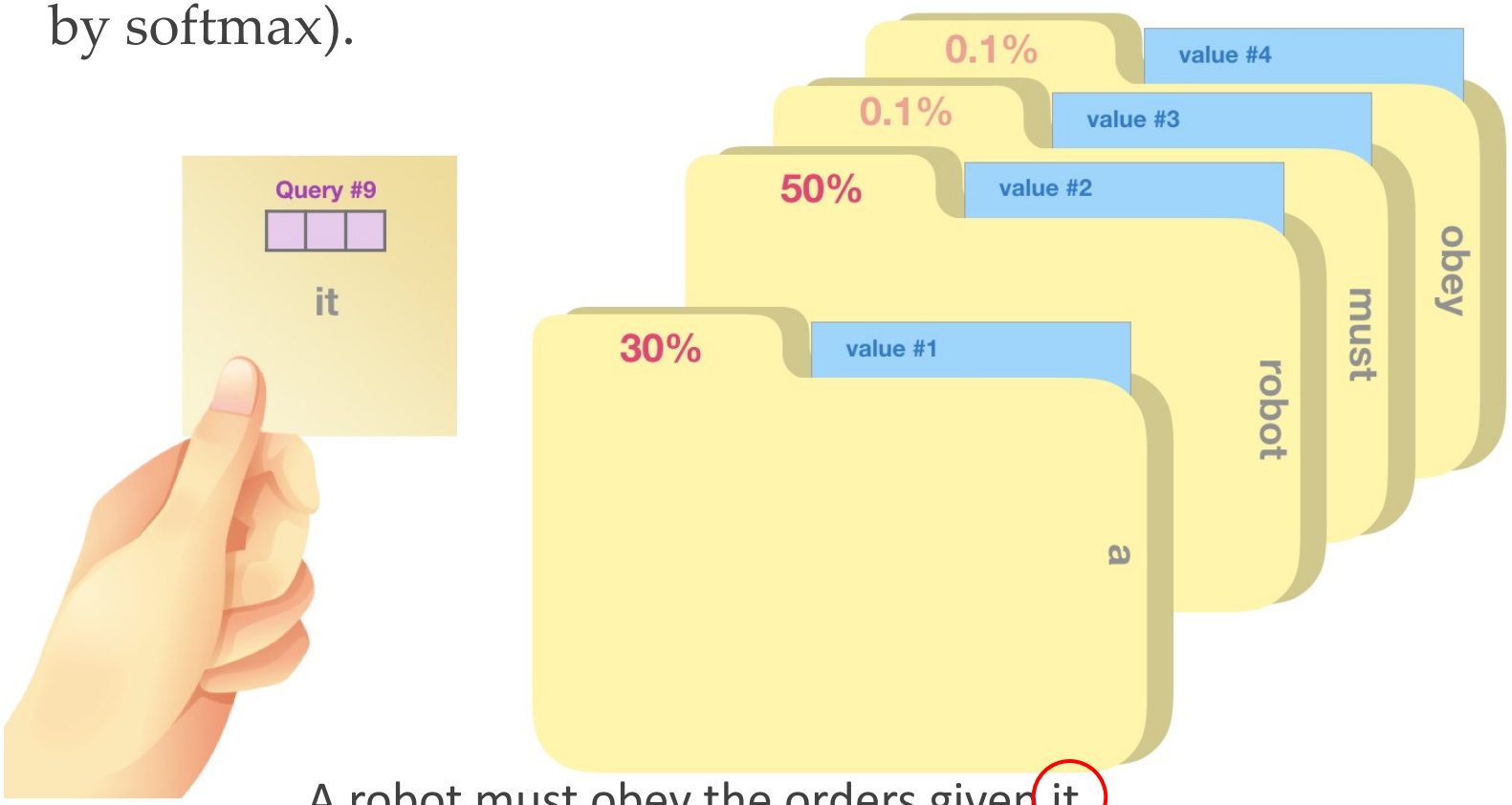
- Let each word pay attention to all other words.



A robot must obey the orders given it.

Self-Attention: the Idea

- Multiplying the query vector by each key vector produces a score for each value (technically: dot product followed by softmax).



Self-Attention: the Key Idea

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

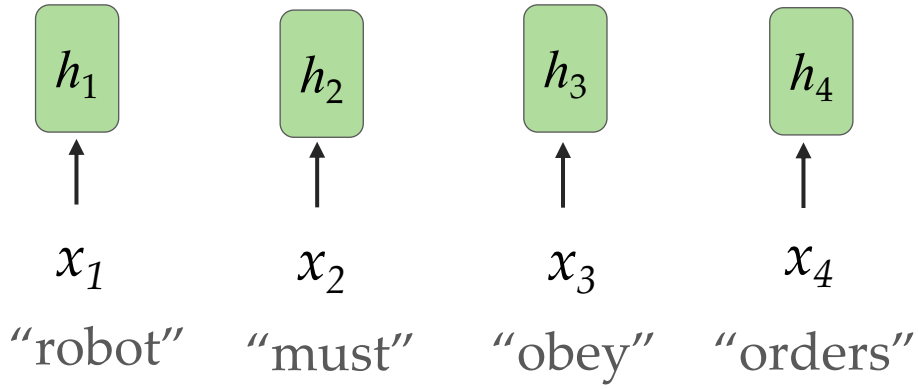
Word	Value vector	Score	Value X Score
<s>		0.001	
a		0.3	
robot		0.5	
must		0.002	
obey		0.001	
the		0.0003	
orders		0.005	
given		0.002	
it		0.19	
		Sum:	

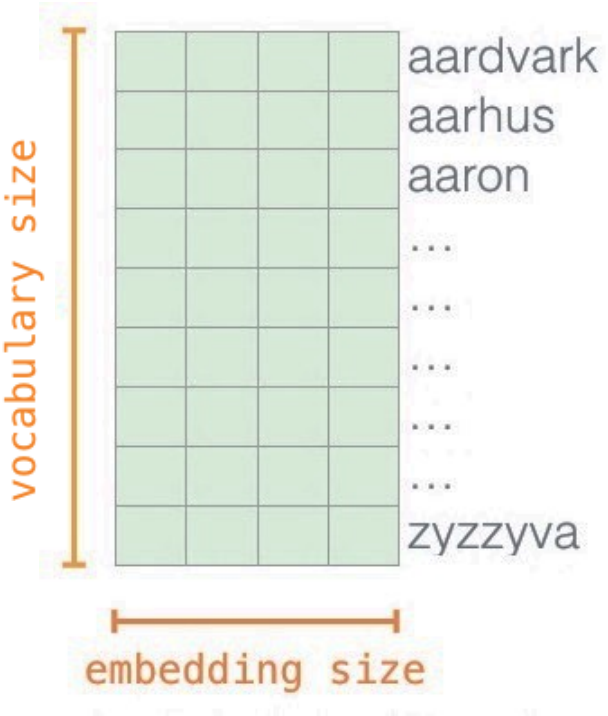
- The outcome vector represents the new state (refreshed memory) for the query word.

Step 1: Token Embedding

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

$$\boldsymbol{h}_i = \boldsymbol{W} x_i$$

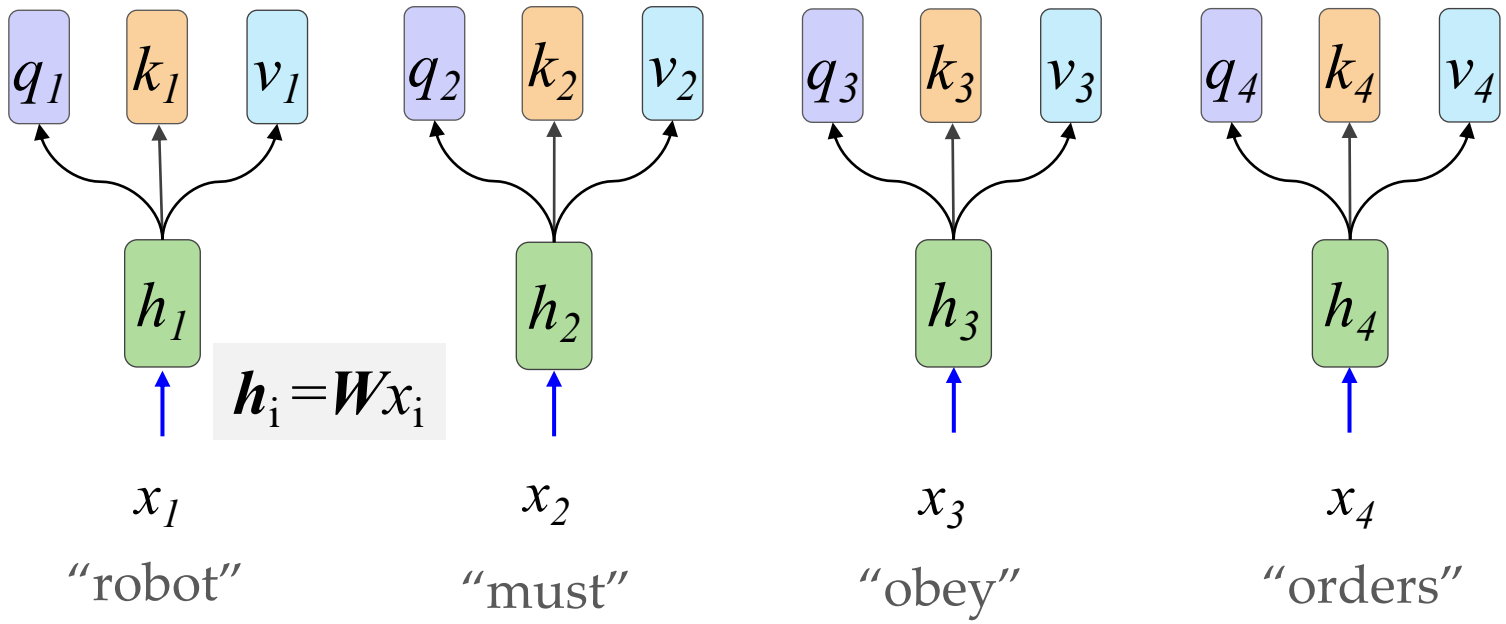




Step 2 : Q, K, V vectors

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

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

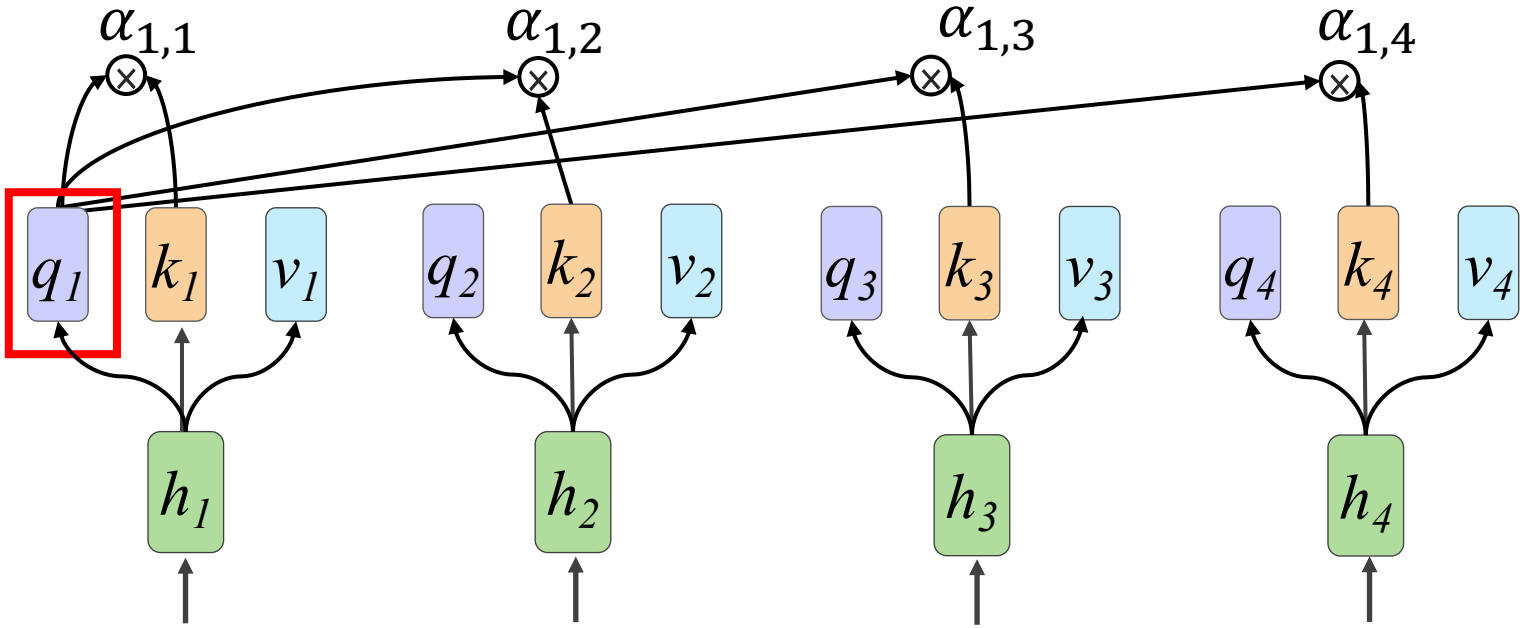


Step 3: Calculate Attention Scores

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

$$\alpha_{1,i} = \frac{q_1^T k_i}{\sqrt{d}}$$

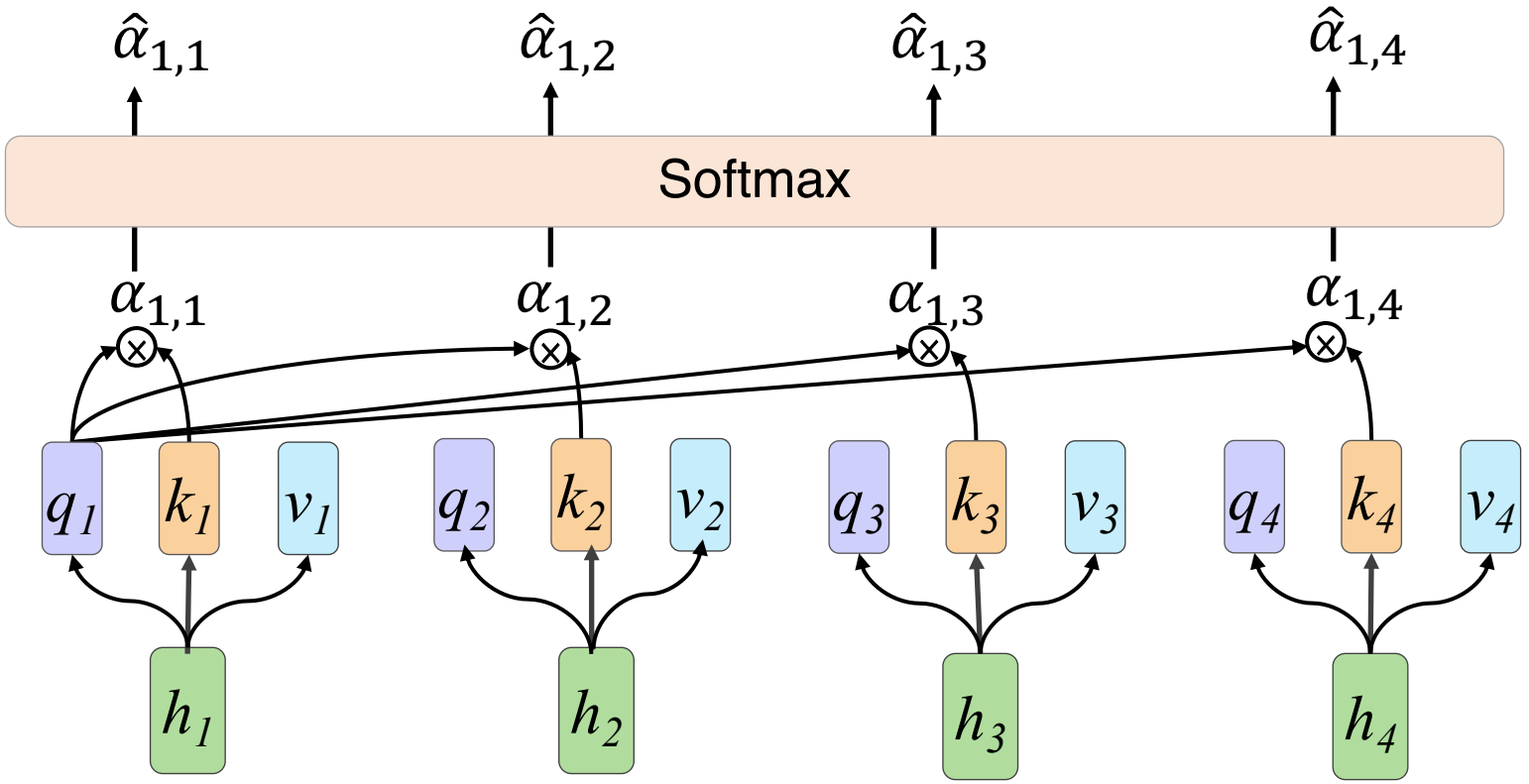
where d is the dim of q and k



Step 4: Normalize Attention Scores

- Normalize attention scores by softmax to obtain attention weights

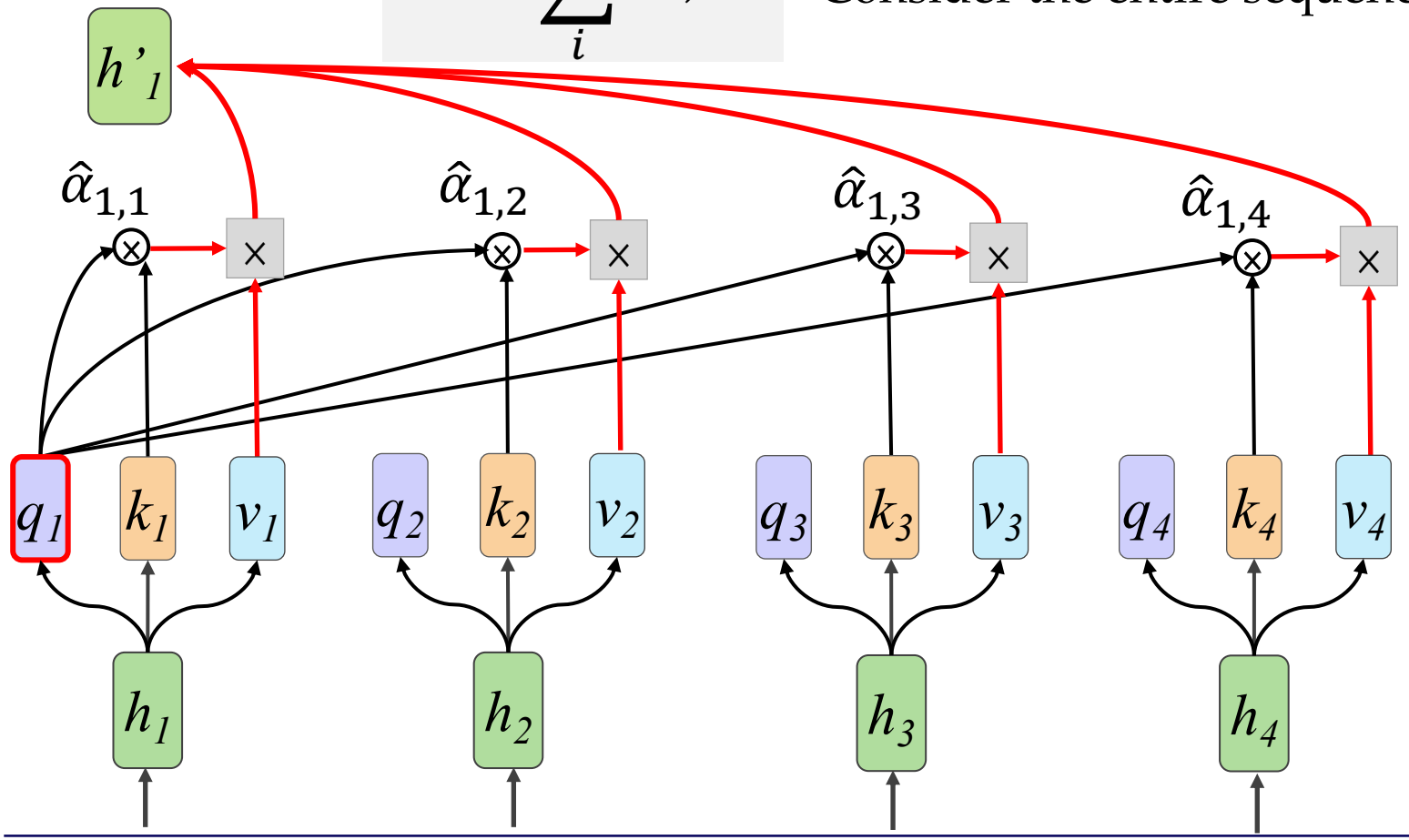
$$\hat{\alpha}_{1,i}=\exp(\alpha_{1,i})/\sum_j \exp(\alpha_{1,j})$$



Step 5: Aggregate Values Based on Attention Weights

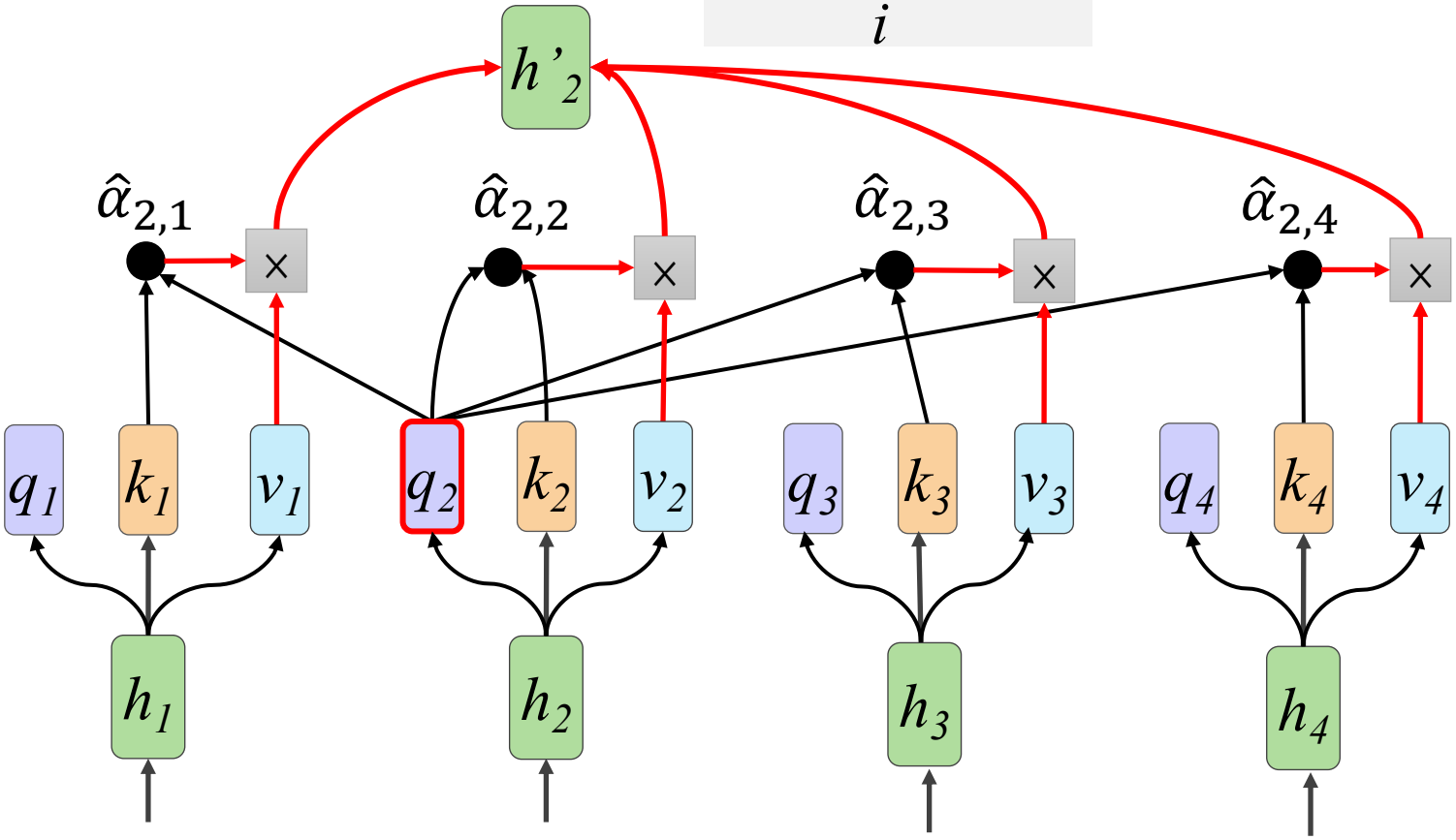
$$h'_1 = \sum_i \hat{\alpha}_{1,i} v_i$$

Consider the entire sequence

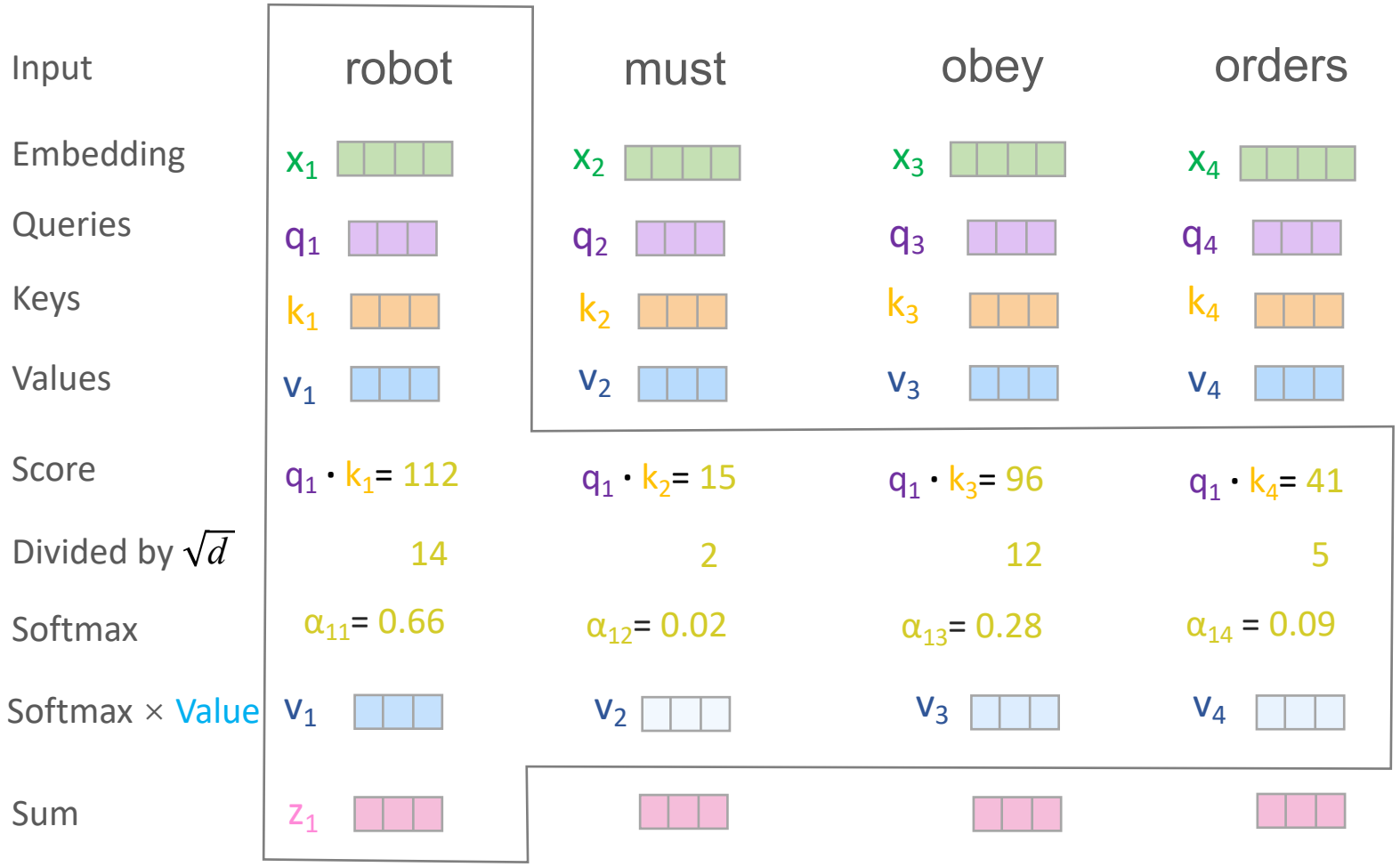


Step 5: Aggregate Values Based on Attention Weights

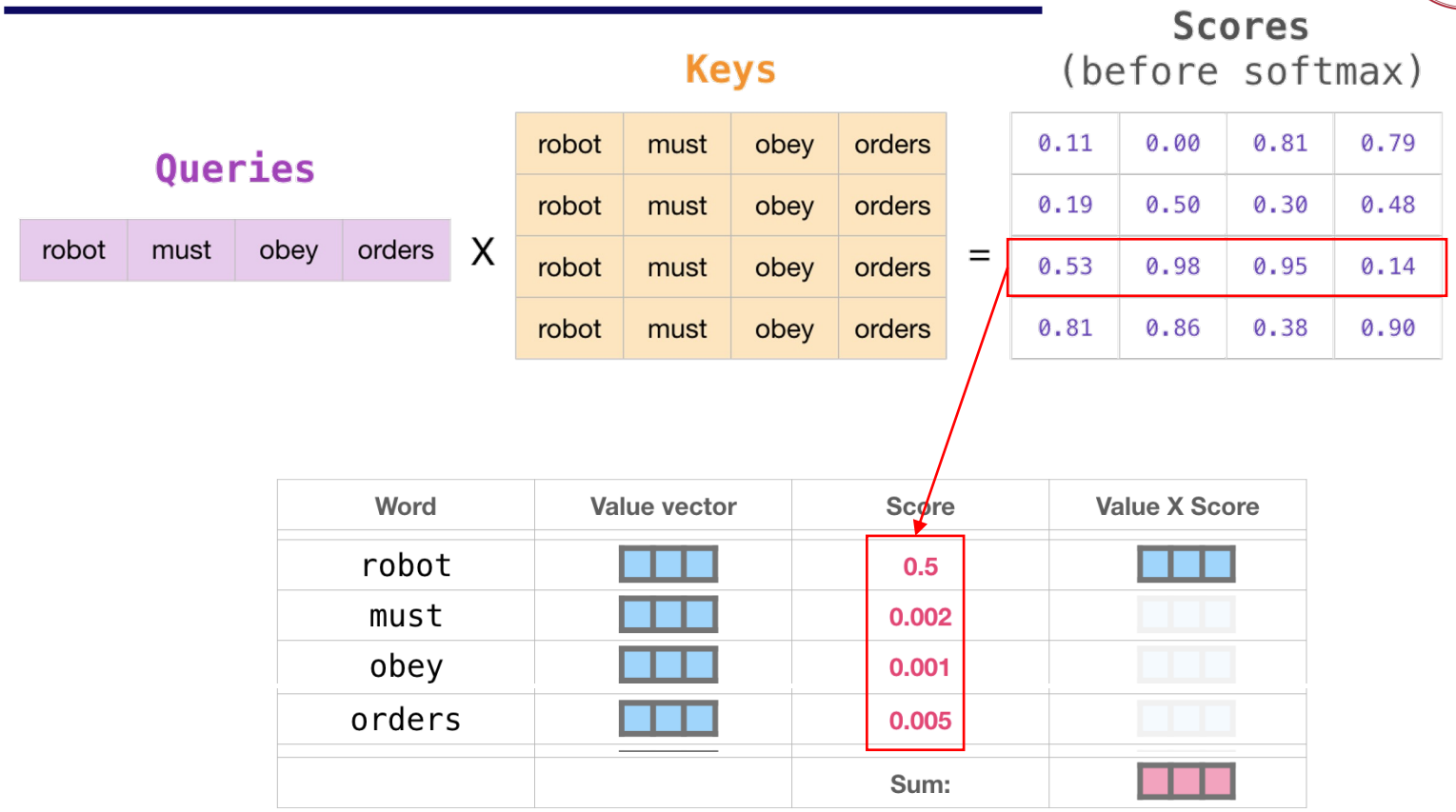
$$h'_2 = \sum_i \hat{\alpha}_{2,i} v_i$$



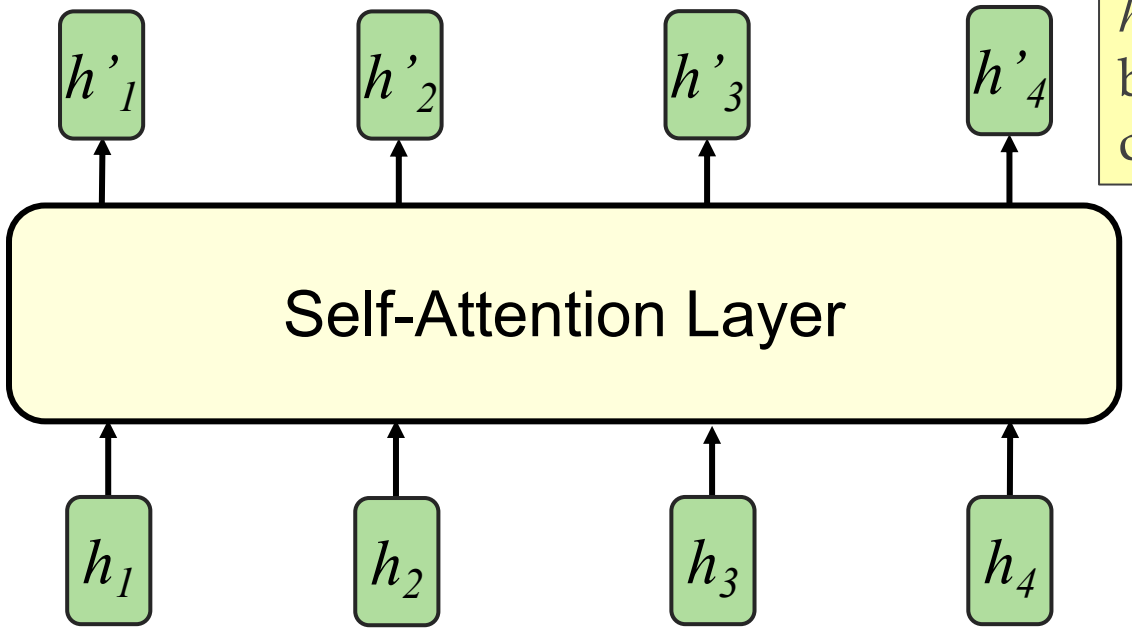
Example


Input
robot
must
obey
orders
Embedding
x_1
x_2
x_3
x_4
Queries
q_1
q_2
q_3
q_4
Keys
k_1
k_2
k_3
k_4
Values
v_1
v_2
v_3
v_4
Score
$q_1 \cdot k_1 = 112$
$q_1 \cdot k_2 = 15$
$q_1 \cdot k_3 = 96$
$q_1 \cdot k_4 = 41$
Divided by $\sqrt{d}$
14
2
12
5
Softmax
$\alpha_{11} = 0.66$
$\alpha_{12} = 0.02$
$\alpha_{13} = 0.28$
$\alpha_{14} = 0.09$
Softmax × Value
v_1
v_2
v_3
v_4
Sum
z_1

Example


Queries
robot must obey orders
X
Keys
robot must obey orders
robot must obey orders
robot must obey orders
robot must obey orders
=
Scores
(before softmax)
0.11 0.00 0.81 0.79
0.19 0.50 0.30 0.48
0.53 0.98 0.95 0.14
0.81 0.86 0.38 0.90
Word
Value vector
Score
Value X Score
robot
0.5
must
0.002
obey
0.001
orders
0.005
Sum:

Matrix Calculation?

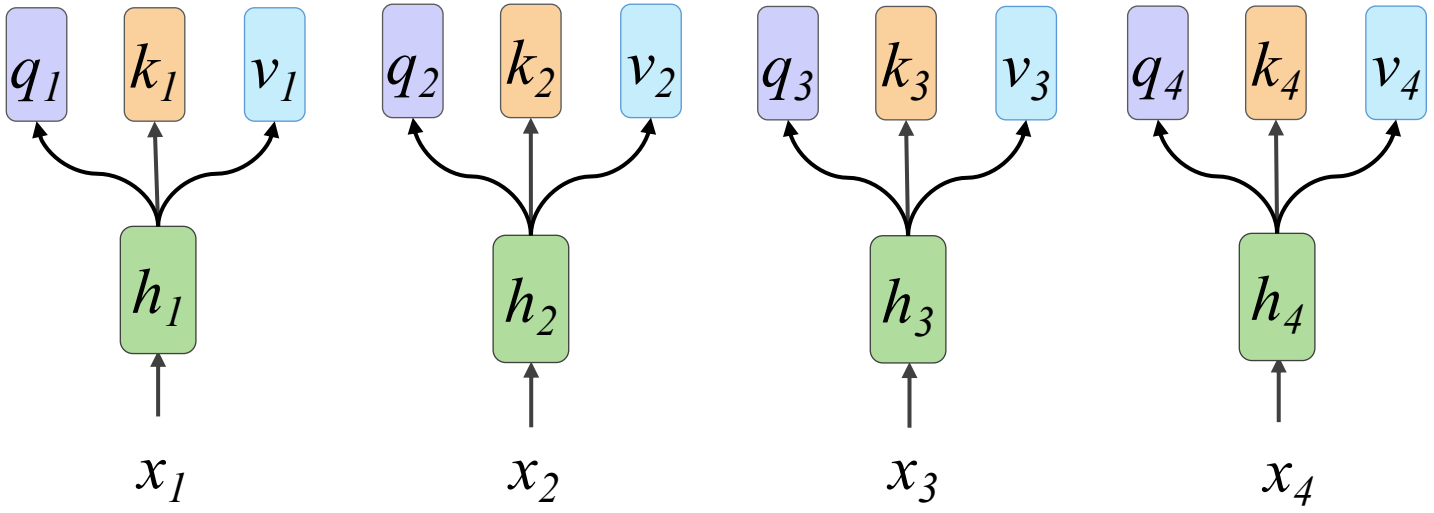


h'_1, h'_2, h'_3, h'_4 can be parallelly computed.

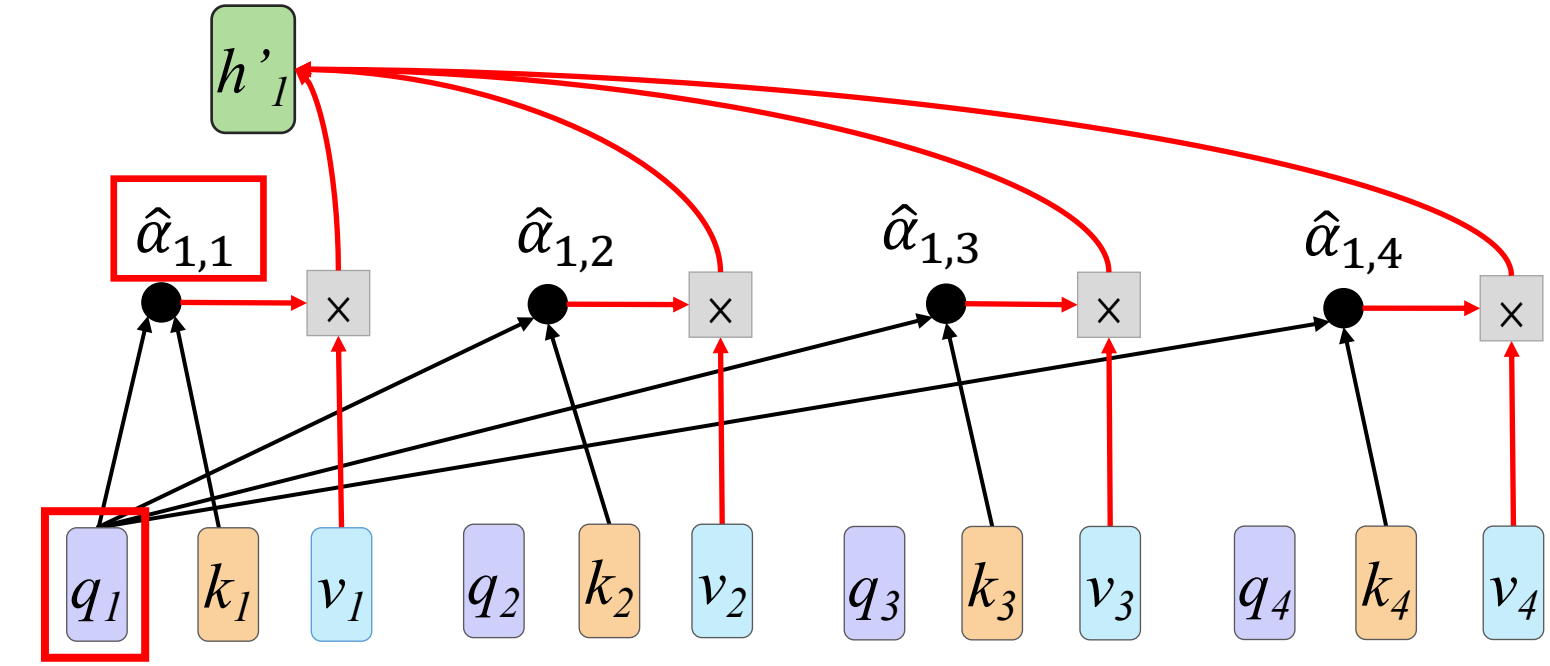
Self-Attention as Matrix Multiplication

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

$$Q \; [q_1\, q_2\, q_3\, q_4] = W^q \; [h_1\, h_2\, h_3\, h_4] \; H$$
$$K \; [k_1\, k_2\, k_3\, k_4] = W^k \; [h_1\, h_2\, h_3\, h_4] \; H$$
$$V \; [v_1\, v_2\, v_3\, v_4] = W^v \; [h_1\, h_2\, h_3\, h_4] \; H$$



Self-Attention as Matrix Multiplication



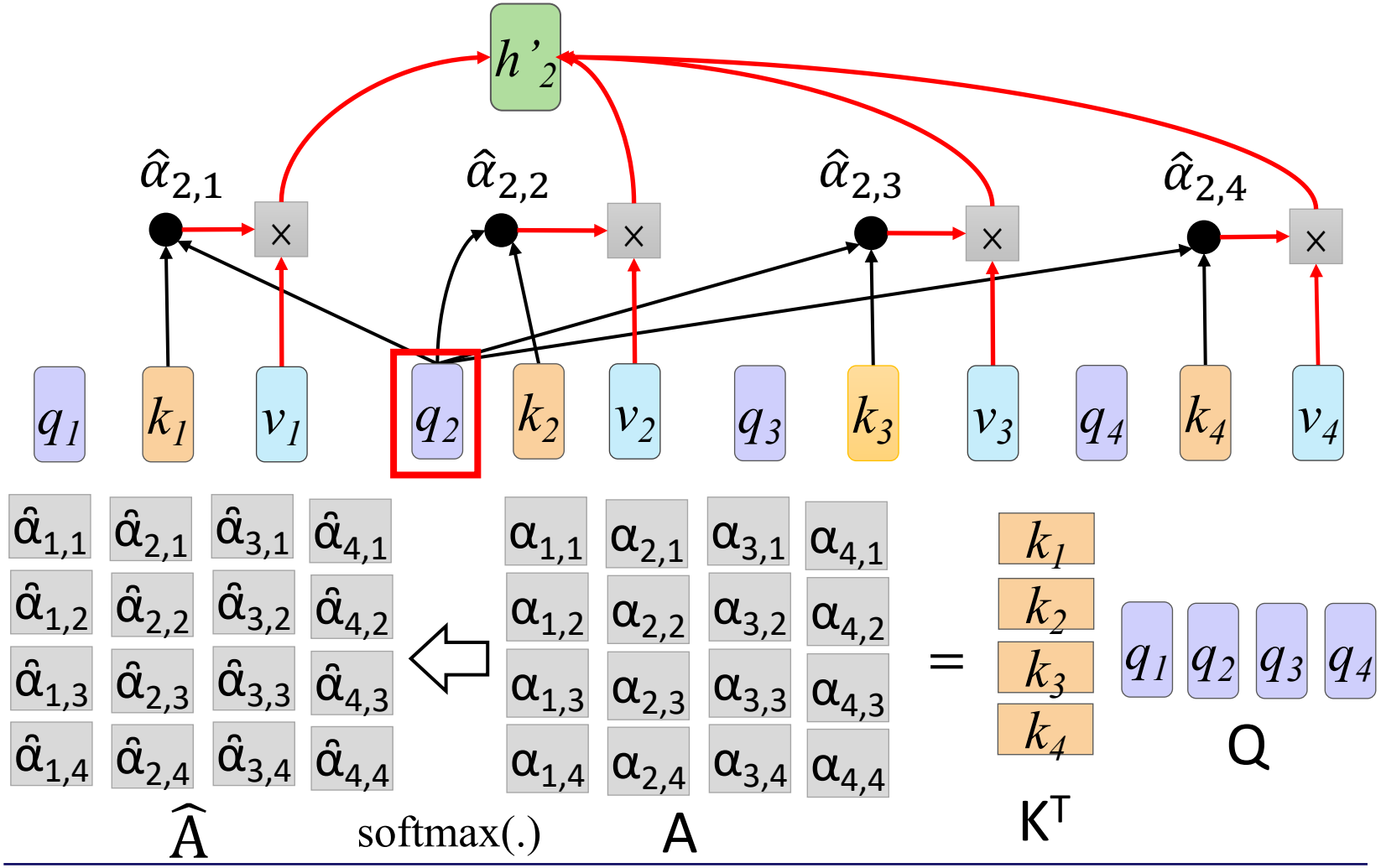
$\alpha_{1,1} = k_1 q_1$ $\alpha_{1,2} = k_2 q_1$

$\alpha_{1,3} = k_3 q_1$ $\alpha_{1,4} = k_4 q_1$

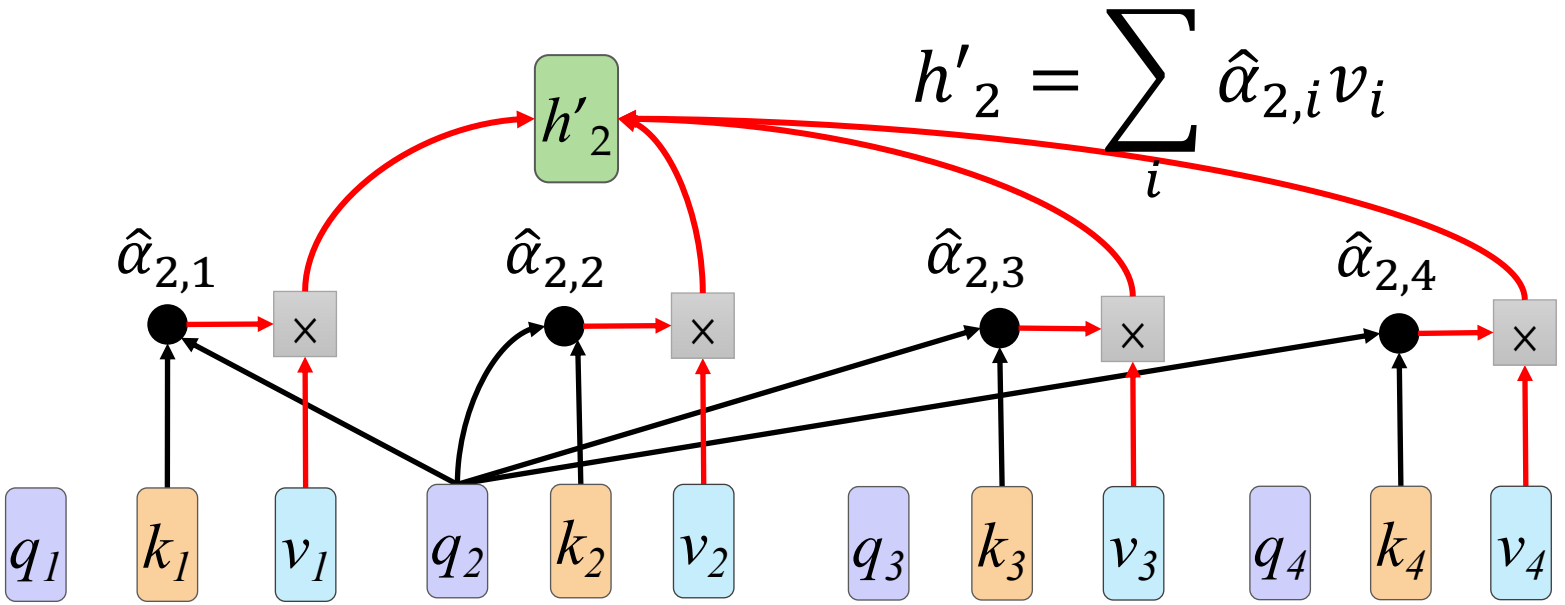
(ignore $\sqrt{d}$ for simplicity)

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

Self-Attention as Matrix Multiplication



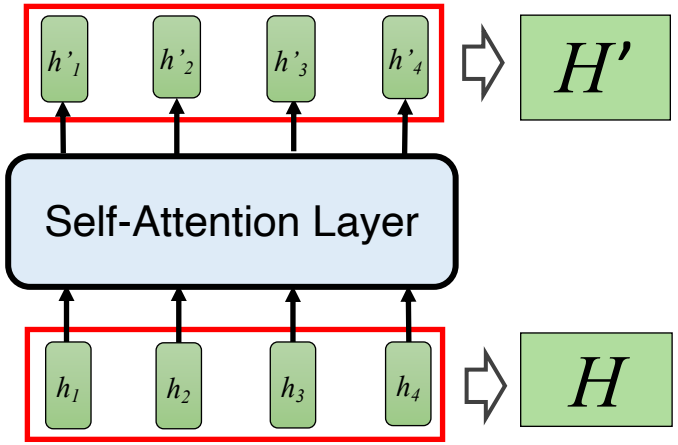
Self-Attention as Matrix Multiplication



$$\begin{bmatrix} h'_1 & h'_2 & h'_3 & h'_4 \end{bmatrix} = \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix} \begin{bmatrix} \hat{\alpha}_{1,1} & \hat{\alpha}_{2,1} & \hat{\alpha}_{3,1} & \hat{\alpha}_{4,1} \\ \hat{\alpha}_{1,2} & \hat{\alpha}_{2,2} & \hat{\alpha}_{3,2} & \hat{\alpha}_{4,2} \\ \hat{\alpha}_{1,3} & \hat{\alpha}_{2,3} & \hat{\alpha}_{3,3} & \hat{\alpha}_{4,3} \\ \hat{\alpha}_{1,4} & \hat{\alpha}_{2,4} & \hat{\alpha}_{3,4} & \hat{\alpha}_{4,4} \end{bmatrix}$$

H' = V $\hat{A}$

Self-Attention as Matrix Multiplication



$$Q = W^q H$$

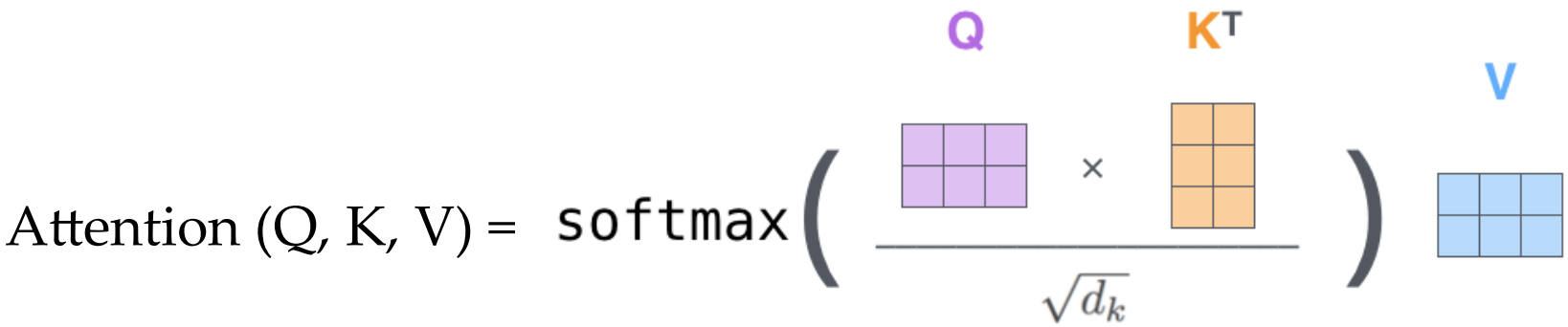
$$K = W^k H$$

$$V = W^v H$$

$$\hat{A} = \text{softmax}(A) = K^T Q$$

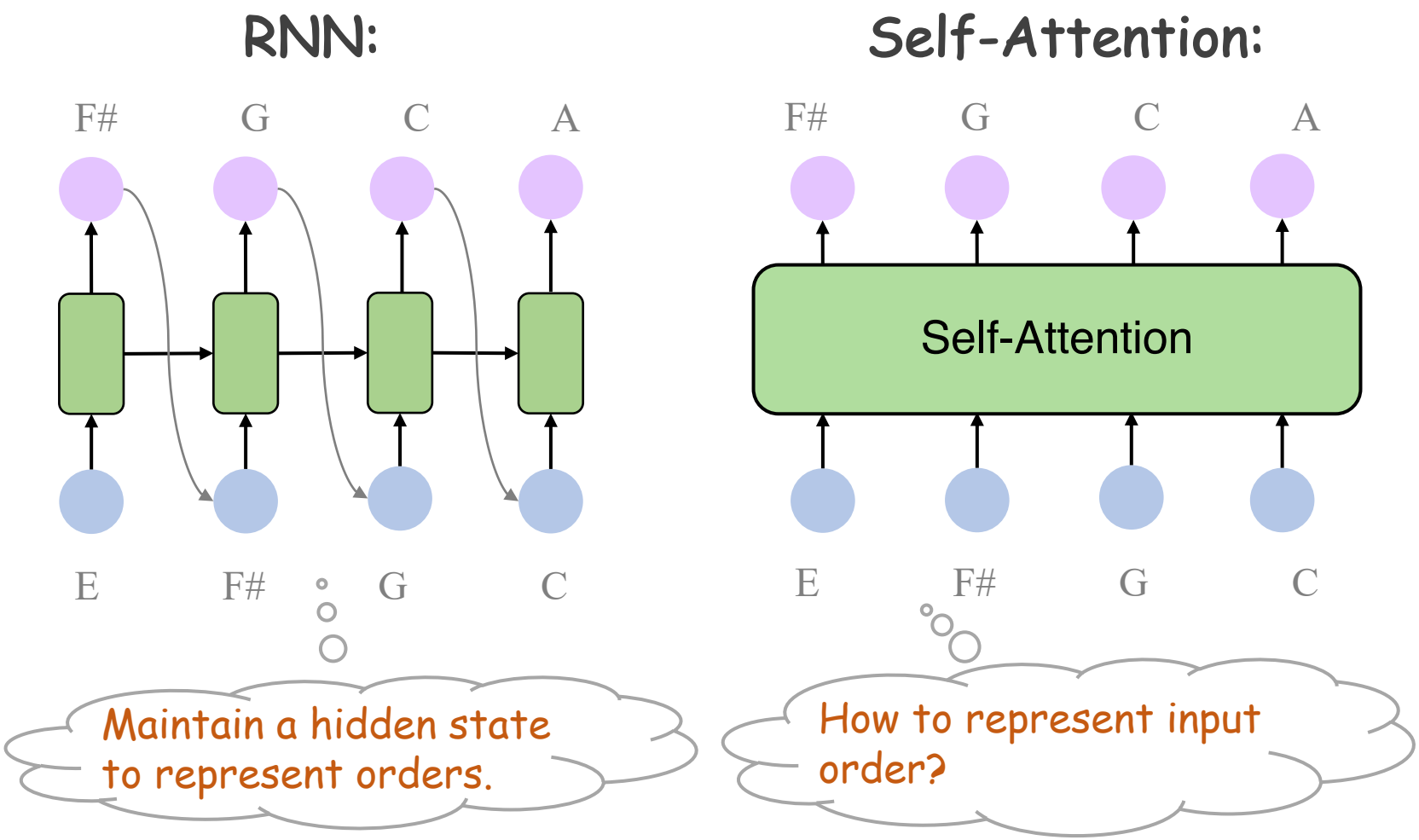
$$H' = V \hat{A}$$

Self-Attention as Matrix Multiplication



矩陣乘法，可用 GPU 加速

Representing the Order?

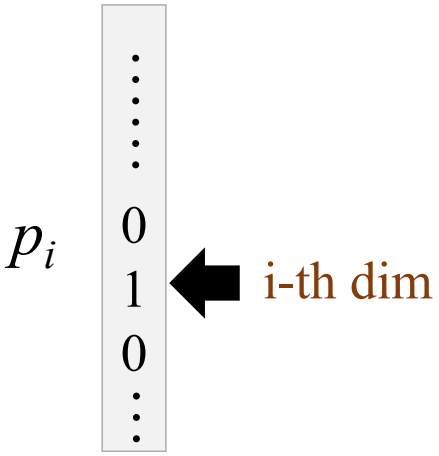


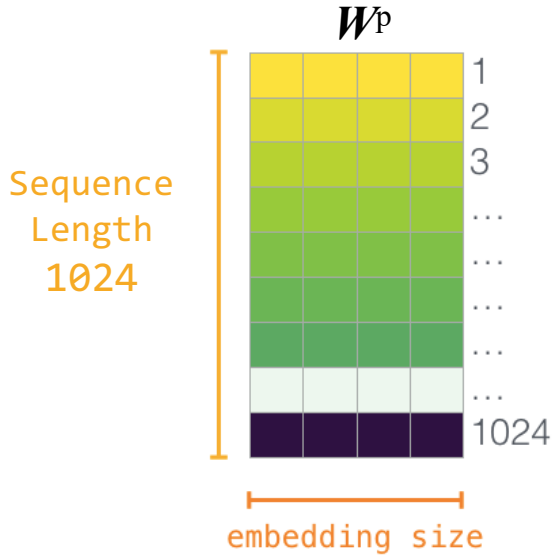
Representing the Order?

- Position Encoding: each position has a unique positional vector $\boldsymbol{e}_i$ (typically learned from data)

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

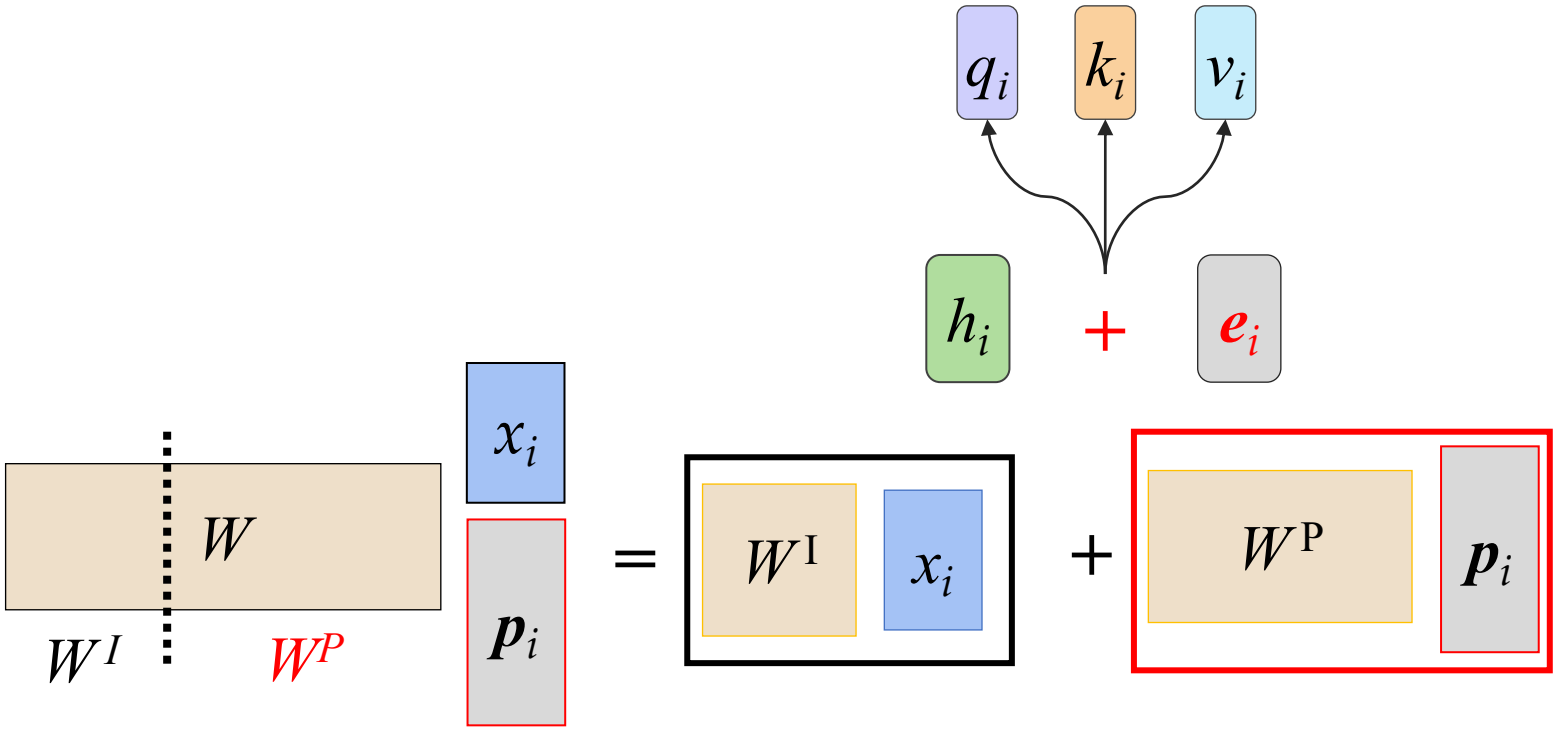
where $\boldsymbol{p}_i$ is a **one-hot** vector indicating the position of x_i in the sequence, $\boldsymbol{W}^p$ is the embedding matrix.



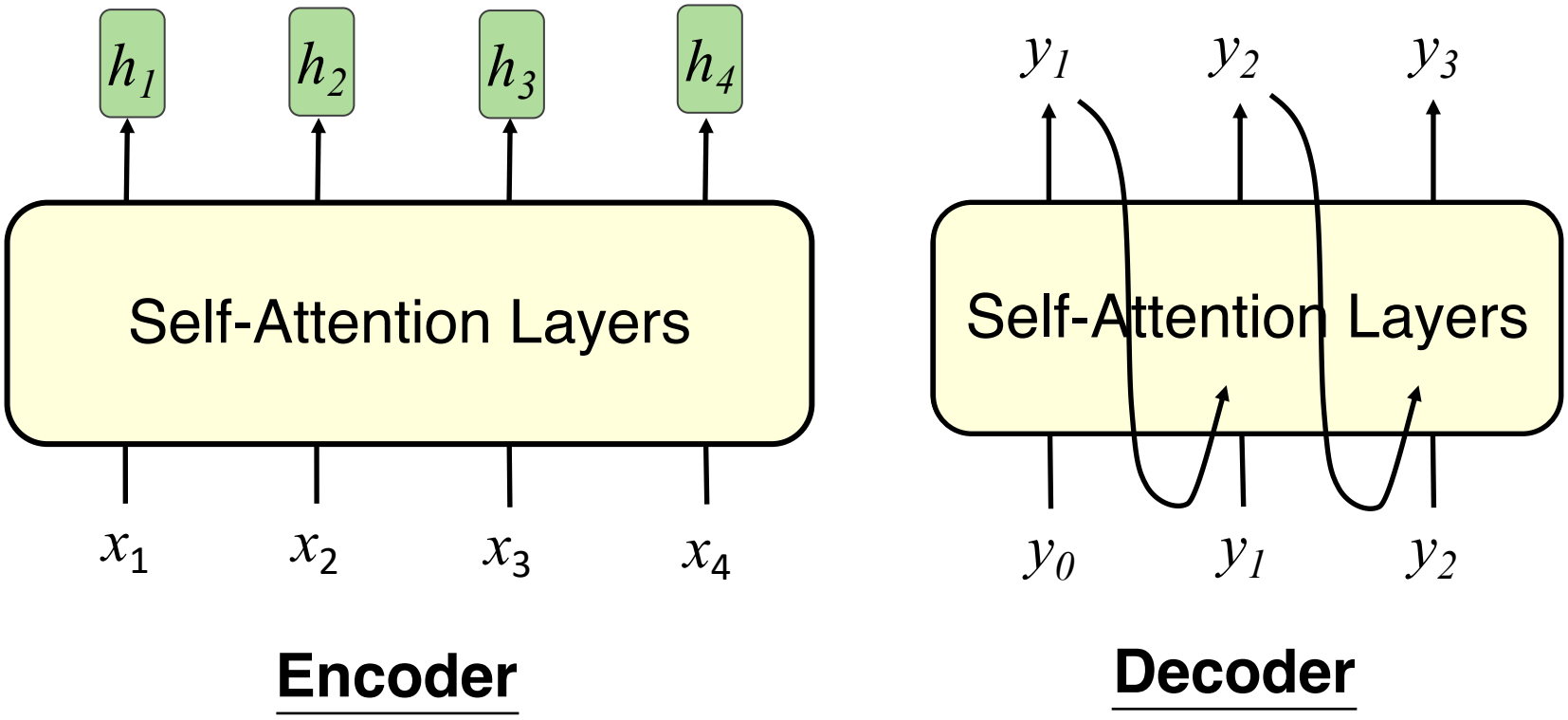


Self-Attention with Position Encoding

- The position encoding $\boldsymbol{e}_i$ is added to the word embedding h_i, as a input to the self-attention layer.

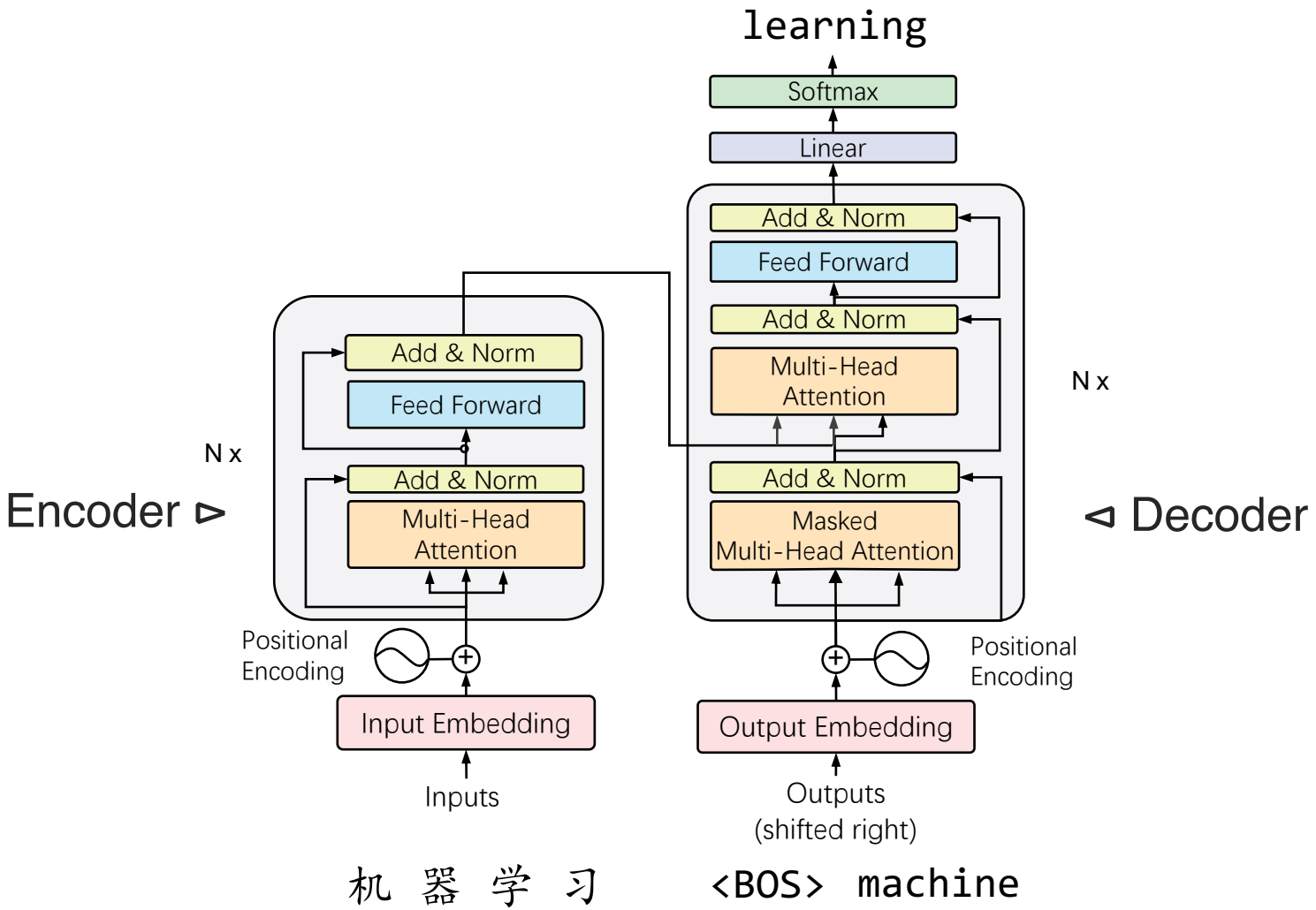


Seq2seq ~~with~~Using Self-Attention

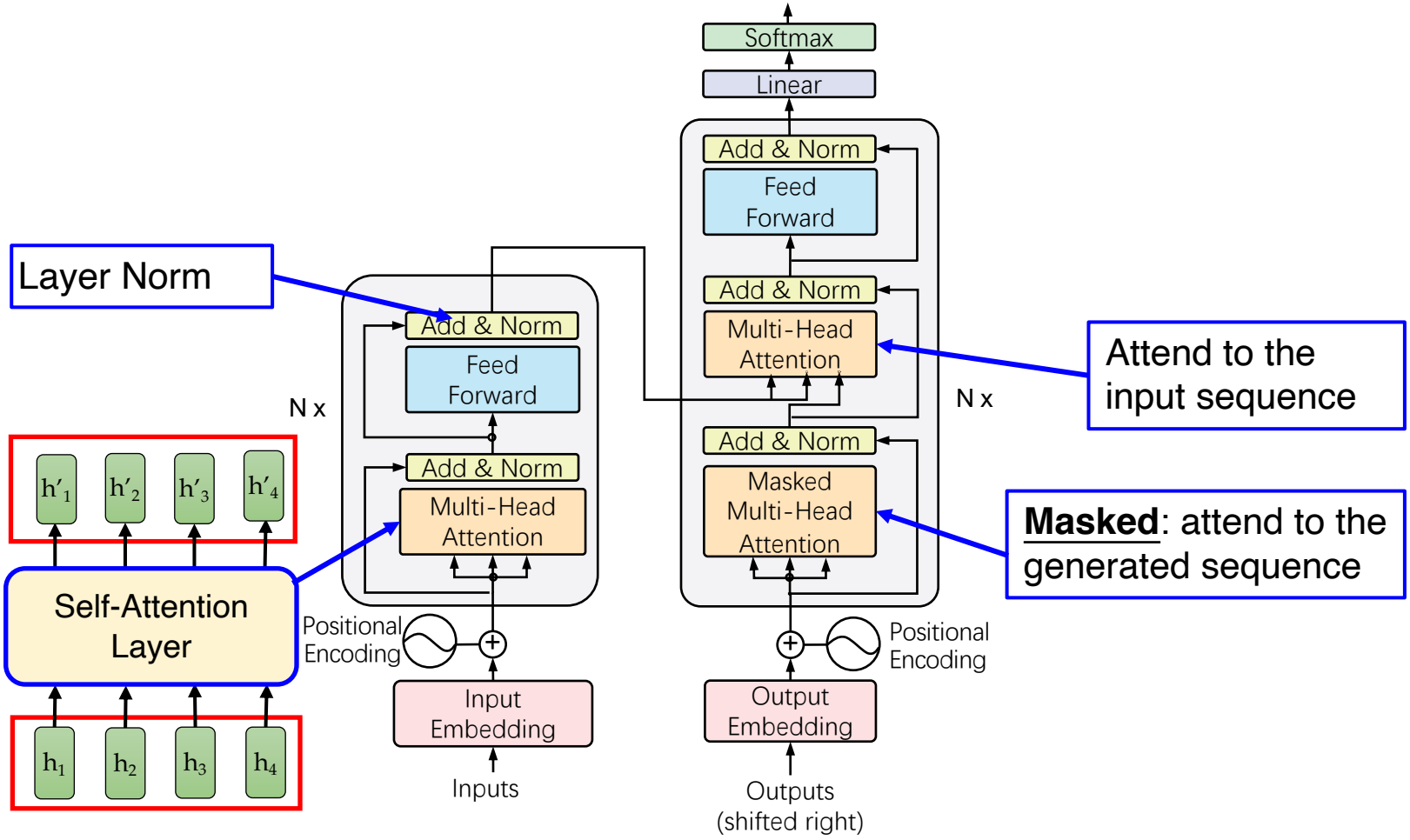


Ashish Vaswani, et al. "Attention Is All You Need". ICLR 2017

Transformer

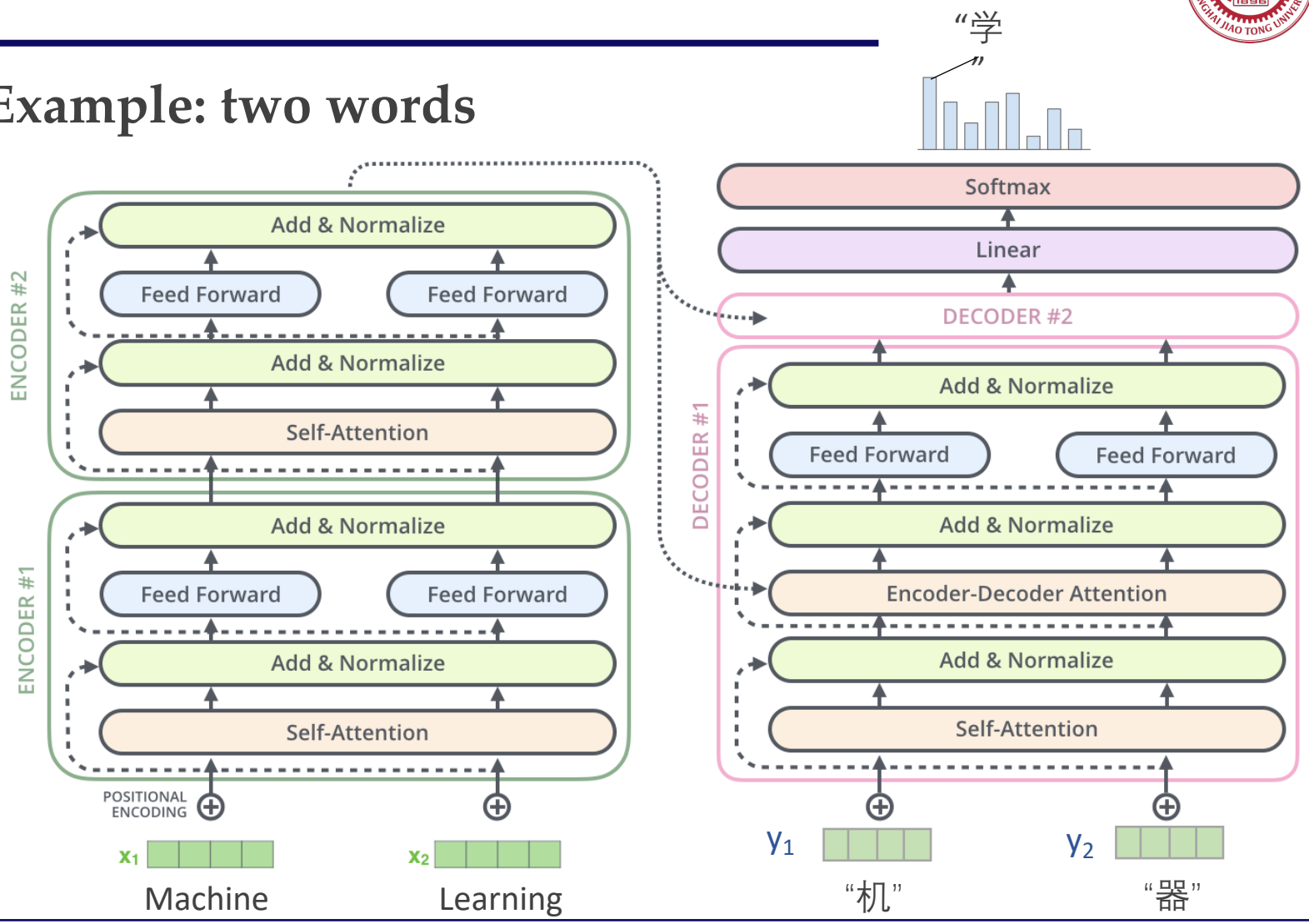


Architecture



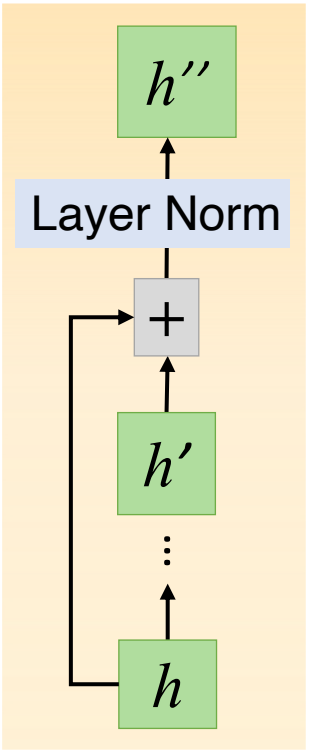
Architecture

Example: two words

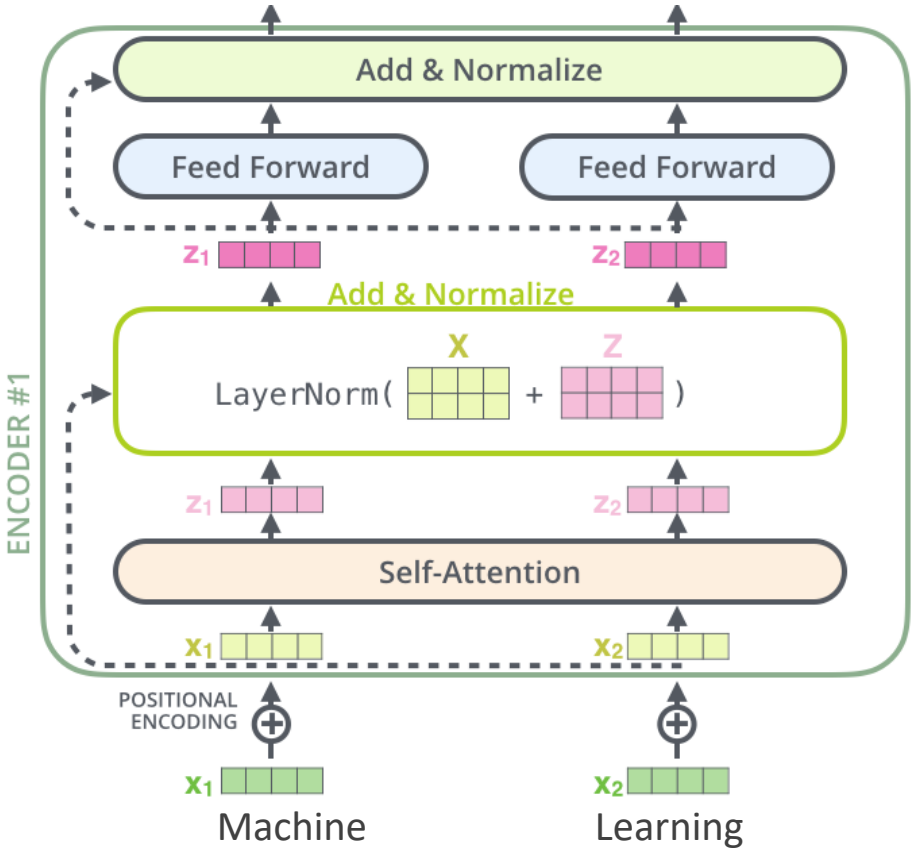


The Residuals (拓展内容不作要求)

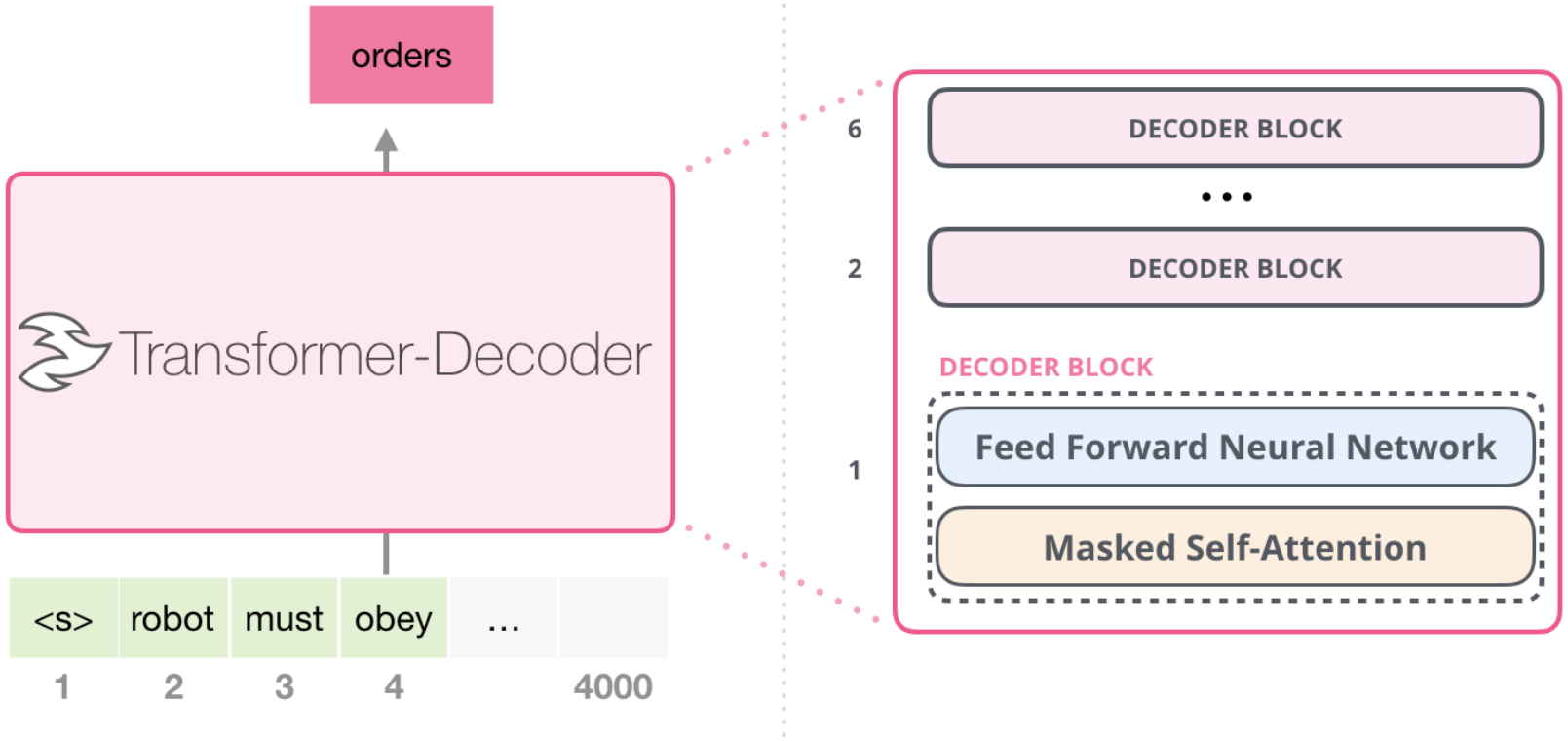
- Each sub-layer (self-attention, ffnn) has a **residual connection** around it, and is followed by a **layer-normalization** step.

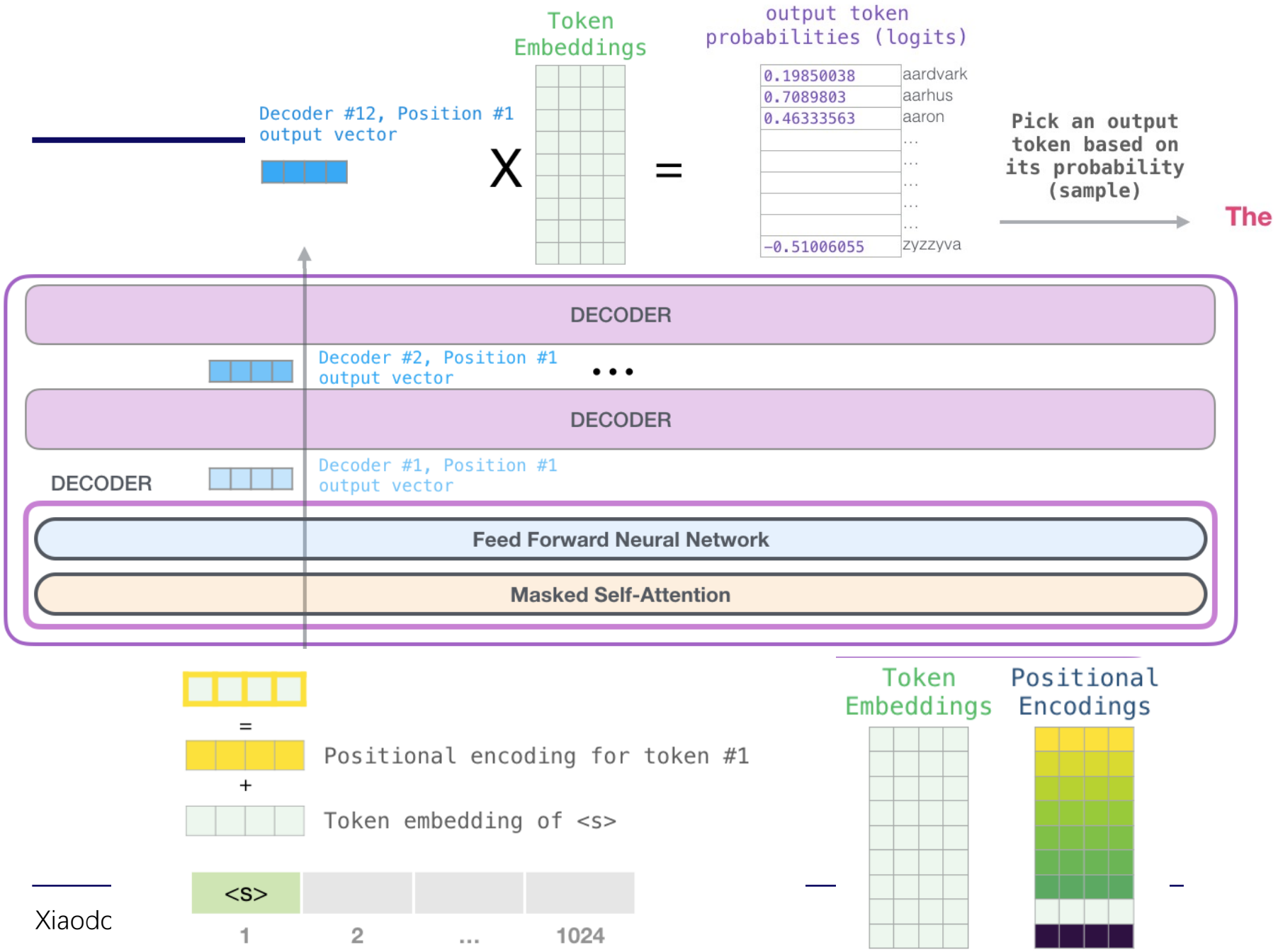


https://arxiv.org/abs/1607.06450



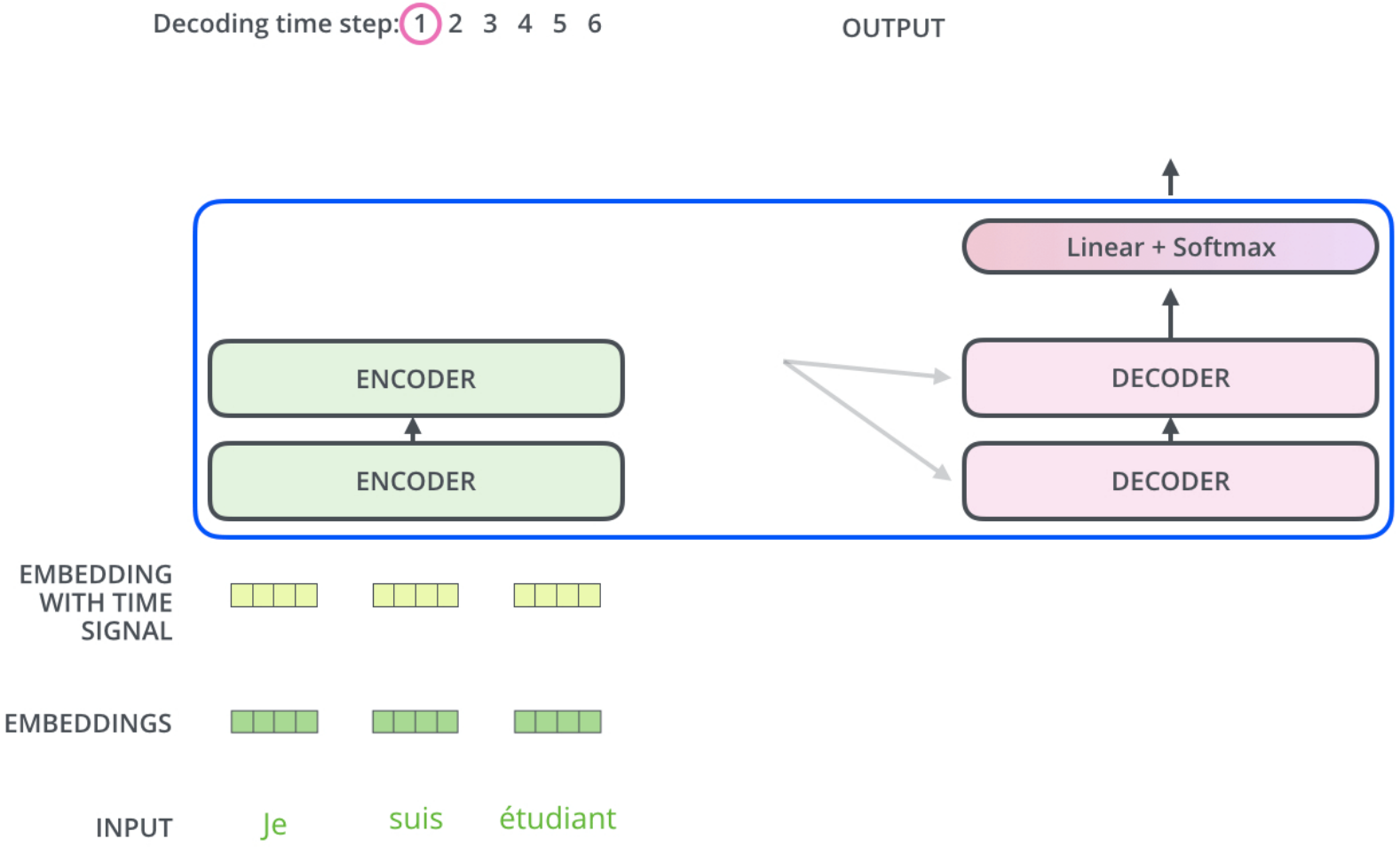
The Decoder Side



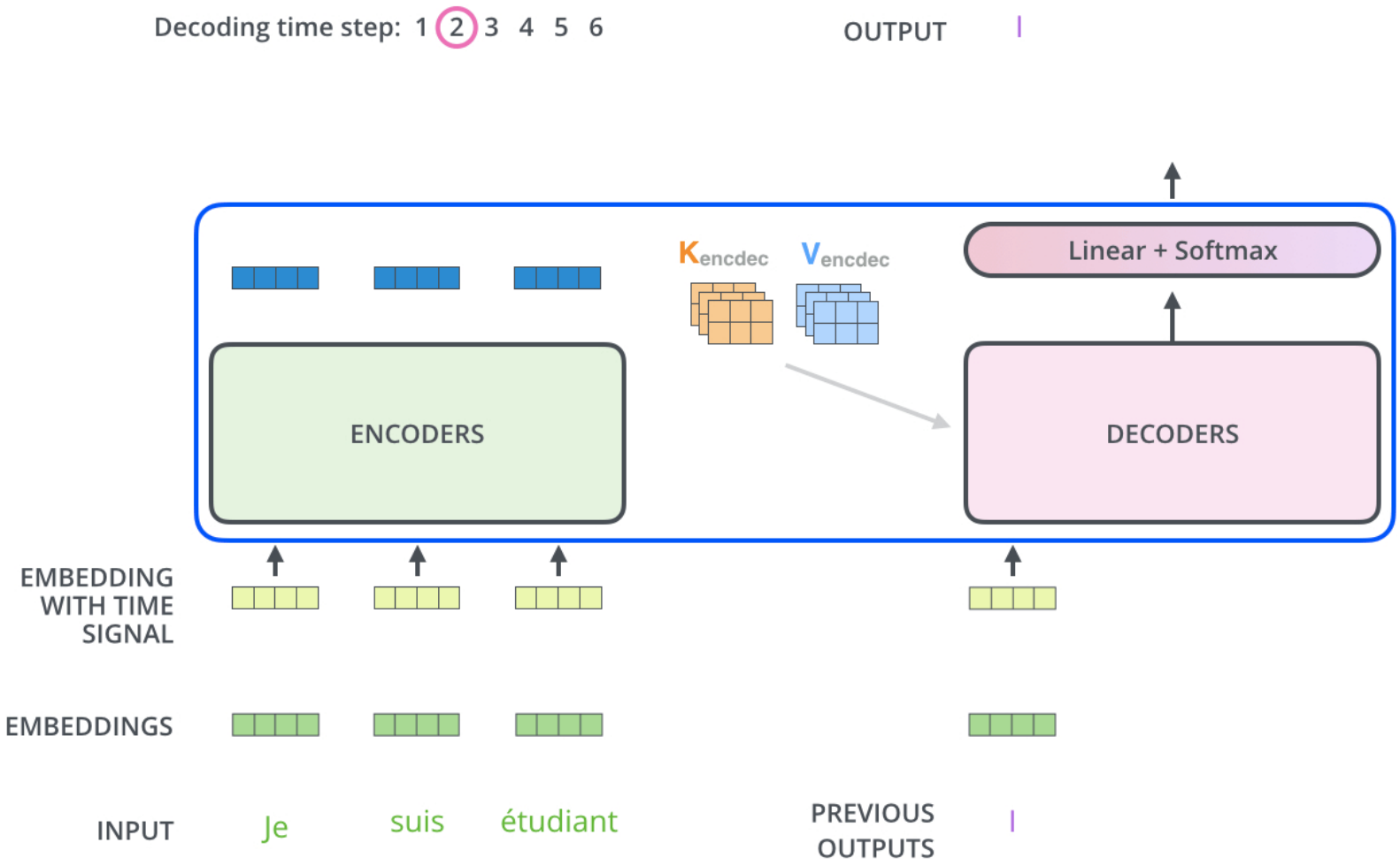

Token Embeddings
output token probabilities (logits)
Decoder #12, Position #1 output vector
X
=
0.19850038 aardvark
0.7089803 aarhus
0.46333563 aaron
-0.51006055 zyzzyva
Pick an output token based on its probability (sample)
The
DECODER
Decoder #2, Position #1 output vector
DECODER
Decoder #1, Position #1 output vector
DECODER
Feed Forward Neural Network
Masked Self-Attention
=
Positional encoding for token #1
+
Token embedding of <s>
<s>
1
2
...
1024
Token Embeddings
Positional Encodings

Xiaodc

The Decoder Side

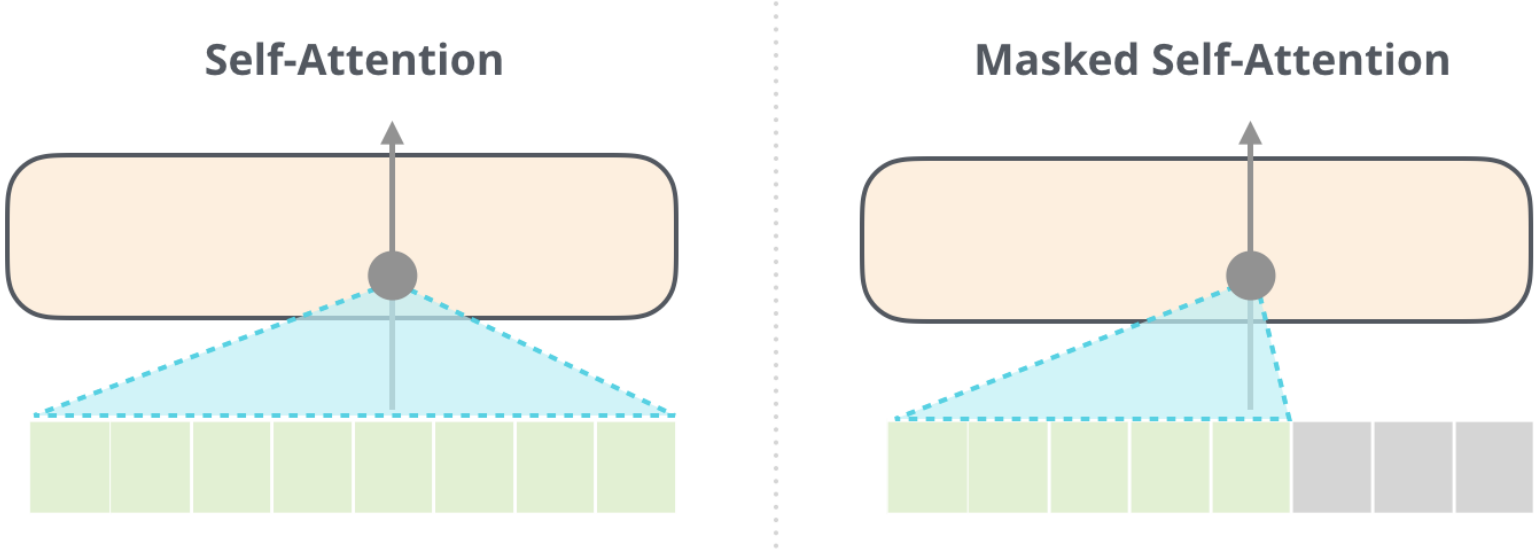


The Decoder Side

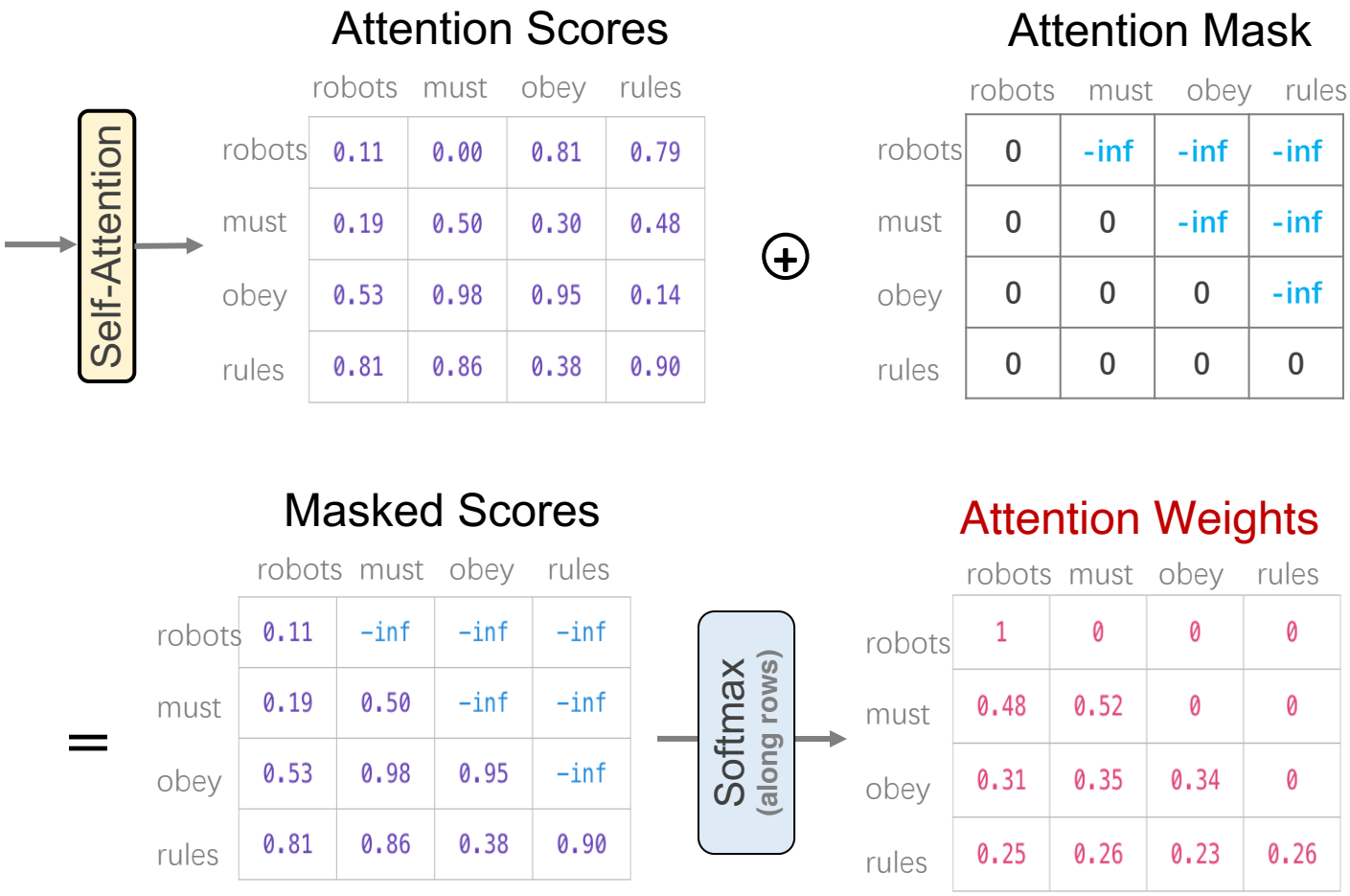


Attention Masks

- an L×L matrix that is applied to the attention scores
- avoid paying attention to specific tokens



Attention Masks



Attention Masks

Self-Attention Mask

- for both encoder and decoder
- to ignore padding tokens

Cross-Attention Mask

- for decoder only
- to ignore padding tokens in the source sequence

Causal Mask

- for decoder only
- only pay attention to generated words

	robots	must	obey	orders	<pad>
robots	0	0	0	0	-inf
must	0	0	0	0	-inf
obey	0	0	0	0	-inf
orders	0	0	0	0	-inf
<pad>	0	0	0	0	-inf

	robots	must	obey	orders
robots	0	-inf	-inf	-inf
must	0	0	-inf	-inf
obey	0	0	0	-inf
rules	0	0	0	0

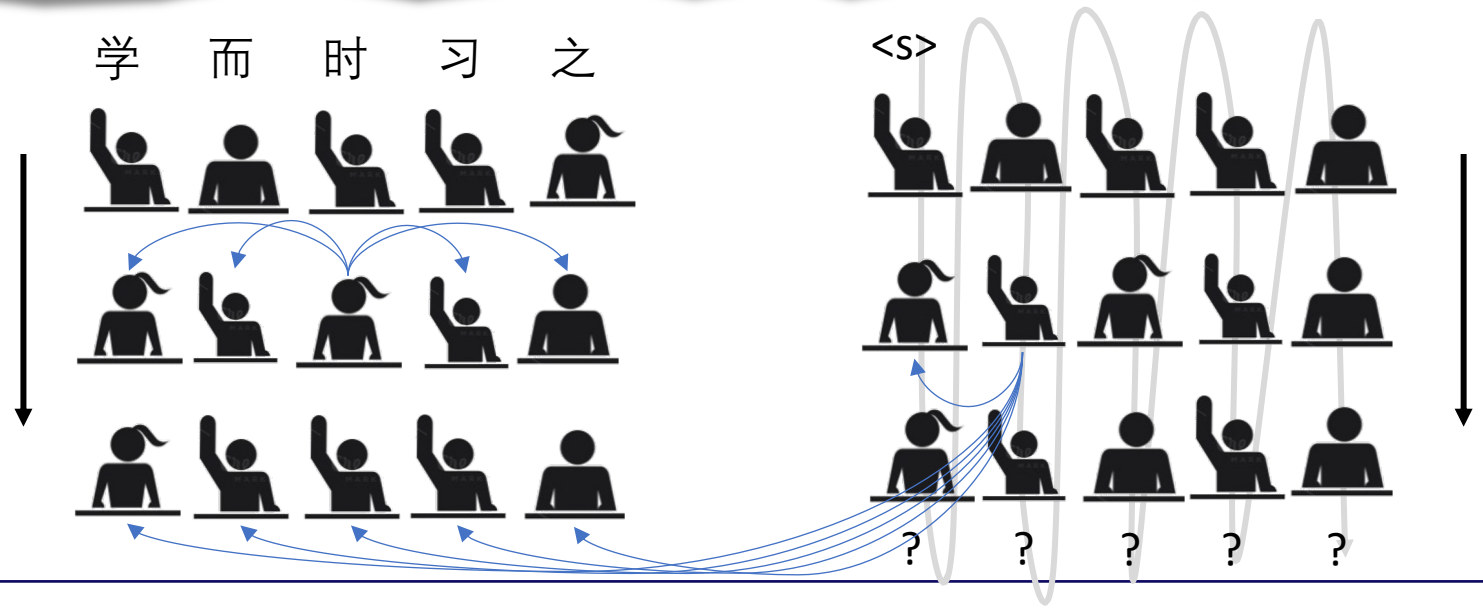
A Live Demo

https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html


GAMES
TIME

Game Time – 模拟Transformer翻译

游戏规则：两组学生分别扮演编码器和解码器，老师给编码器第一排学生一个句子(有初始词向量)。每一排的每个学生模拟自注意力机制将该排与自己最相关的词的向量进行平均(大致心算)，传给后一排。解码器最后一排输出翻译结果。



Training

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:

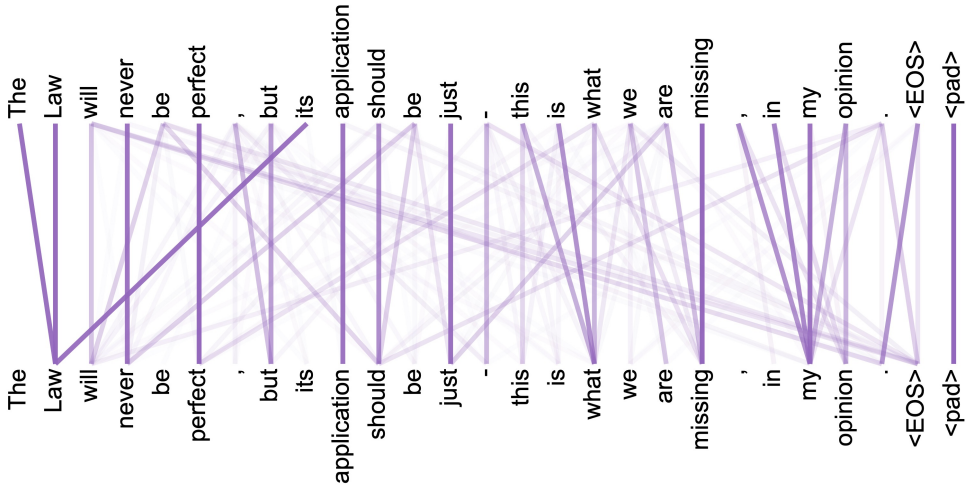
$$l(\theta|x, y) = -\sum_{t=1}^{T} \log p_\theta(y_t|y_1, \ldots, y_{t-1}, x)$$

$$L(\theta|D) = -\frac{1}{N}\sum_{l=1}^{N} l(\theta|x^{(l)}, y^{(l)})$$

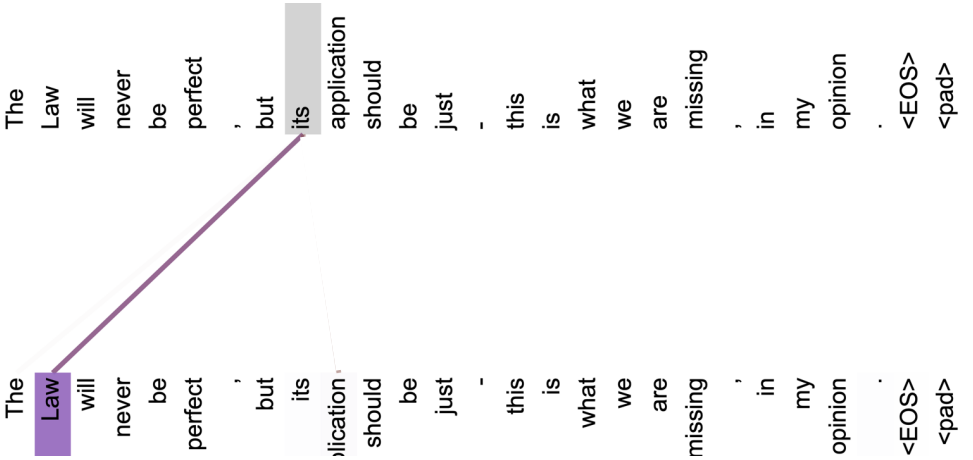
Optimization – gradient descend

Visualizing Attention

Full attentions in layer 5 of 6 for head 5.



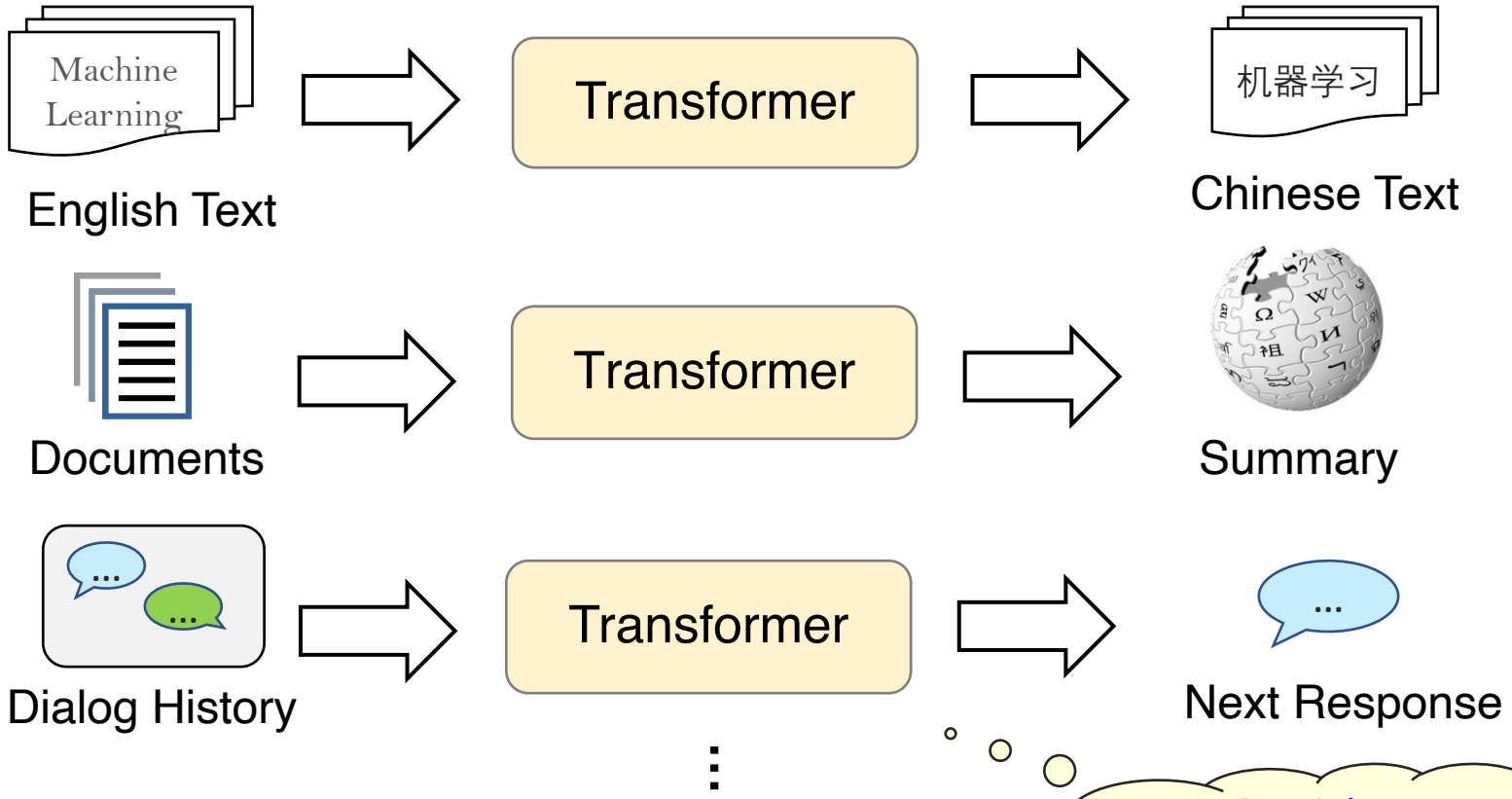
Isolated attentions from just the word 'its' for attention heads 5. Note that the attentions are very sharp for this word.



https://arxiv.org/abs/1706.03762

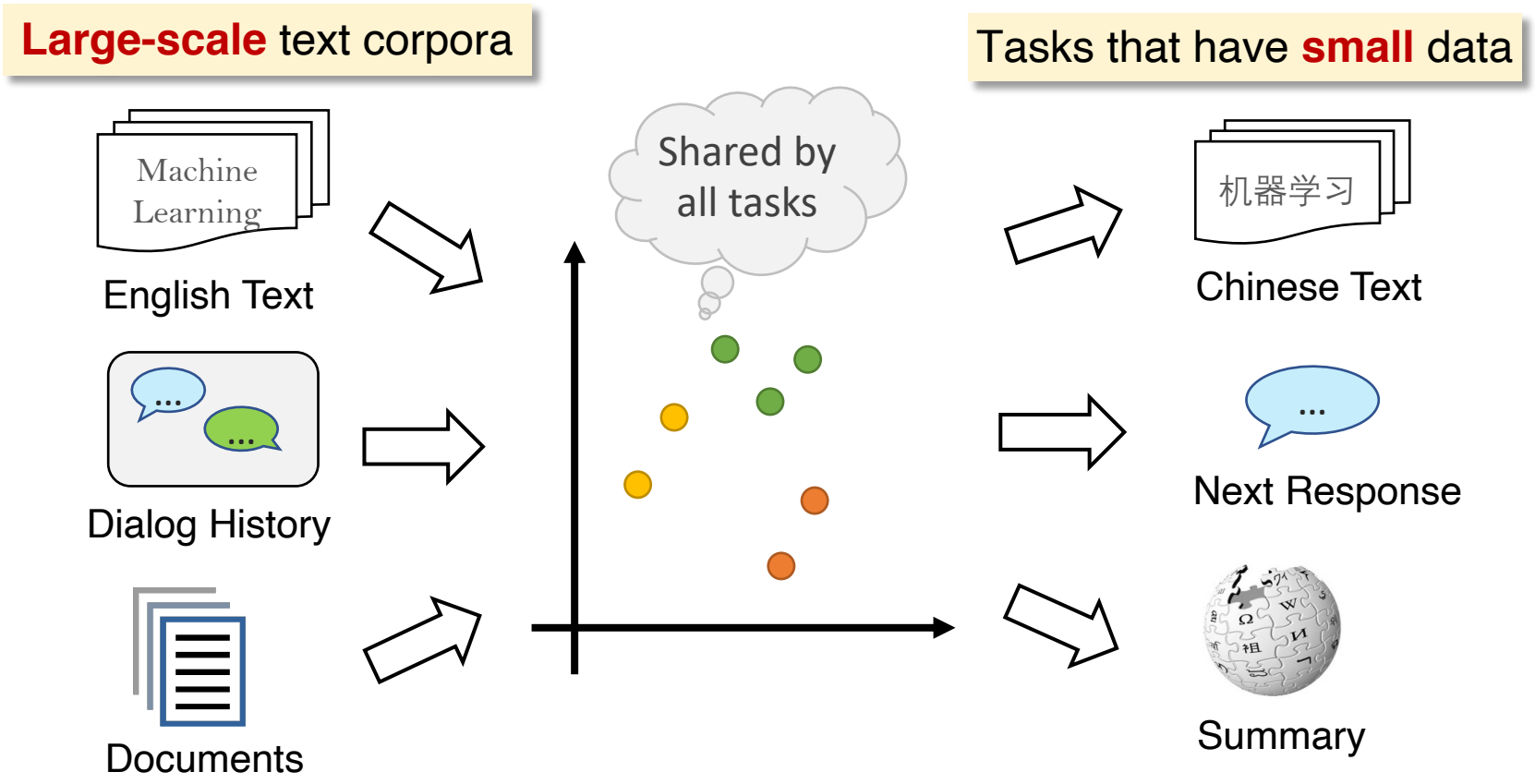
Applications

Whenever we need sequence-to-sequence learning, we can always use Transformer.



Thinking…

Can we learn a unified vector space for all texts?

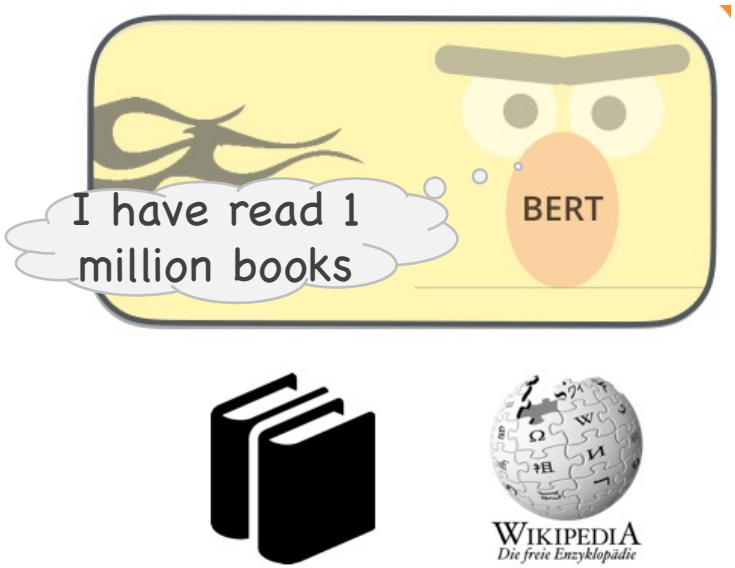


Pretrained Language Models

Learn common representations from large-scale data and then adapt the model to specific tasks.

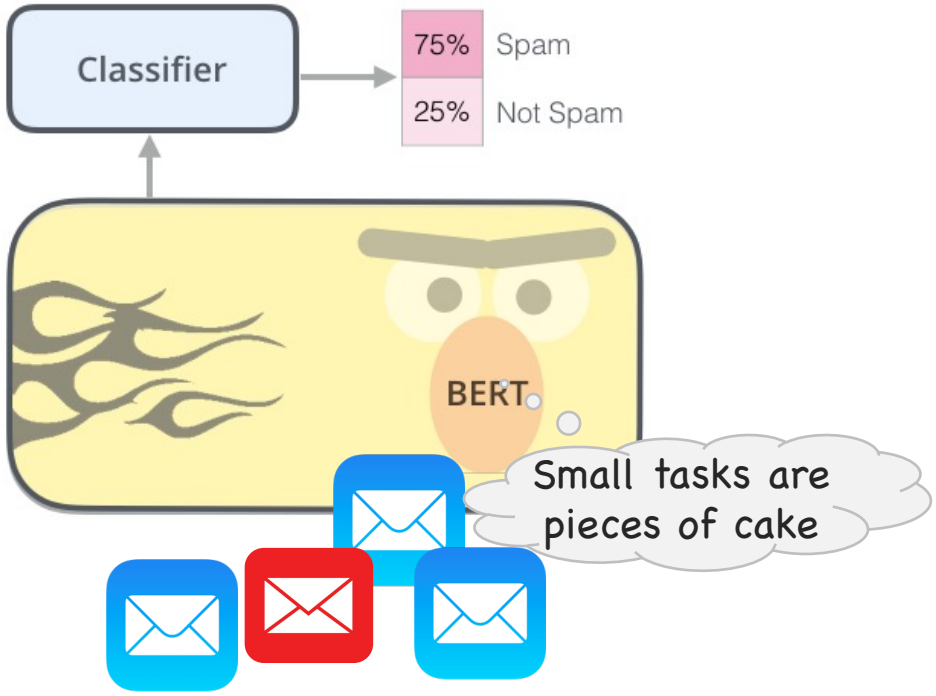
Pre-trained Language Models

Unsupervised training on large amounts of text (e.g., books, Wikipedia, etc)



Phase 1: Pre-Training

Supervised training on a specific task with a labeled dataset. (e.g., spam detection)

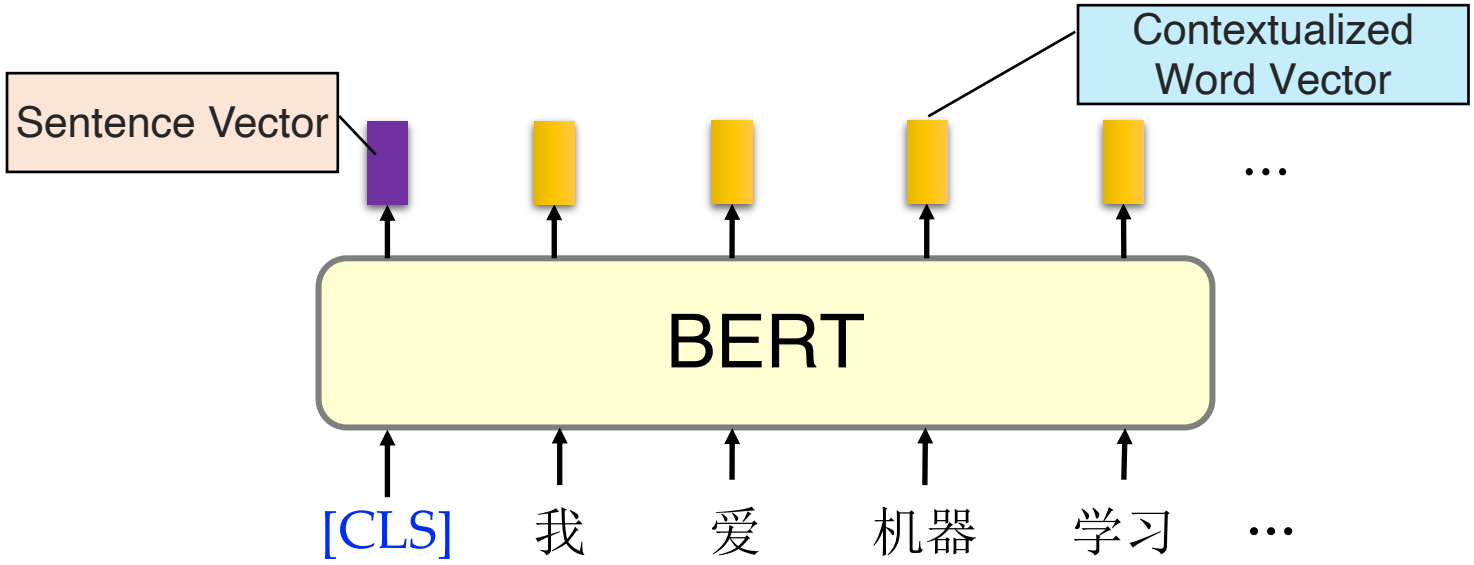


Phase 2: Fine-Tuning

BERT – Bidirectional Encoder Representations from Transformers

A Transformer Encoder that

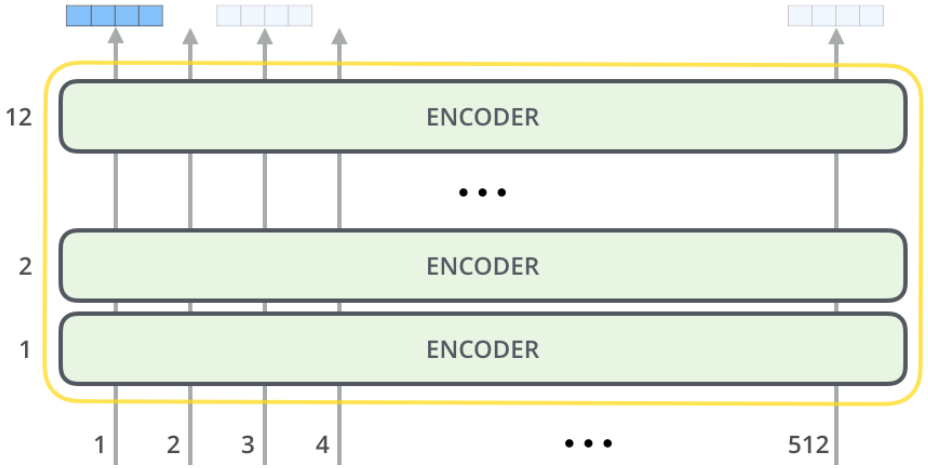
- allows for learning representations of words and sentences.
- pre-trained on large-scale text corpora and then;
- fine-tuned on small task-specific datasets (e.g., classification, QA.)



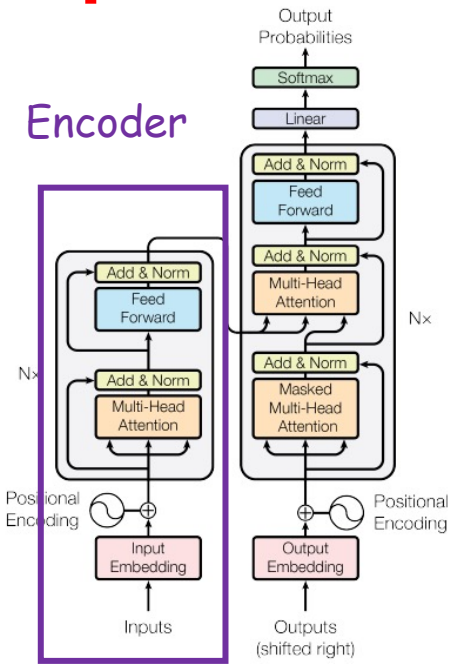
Model Architecture

BERT = 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 the classification.

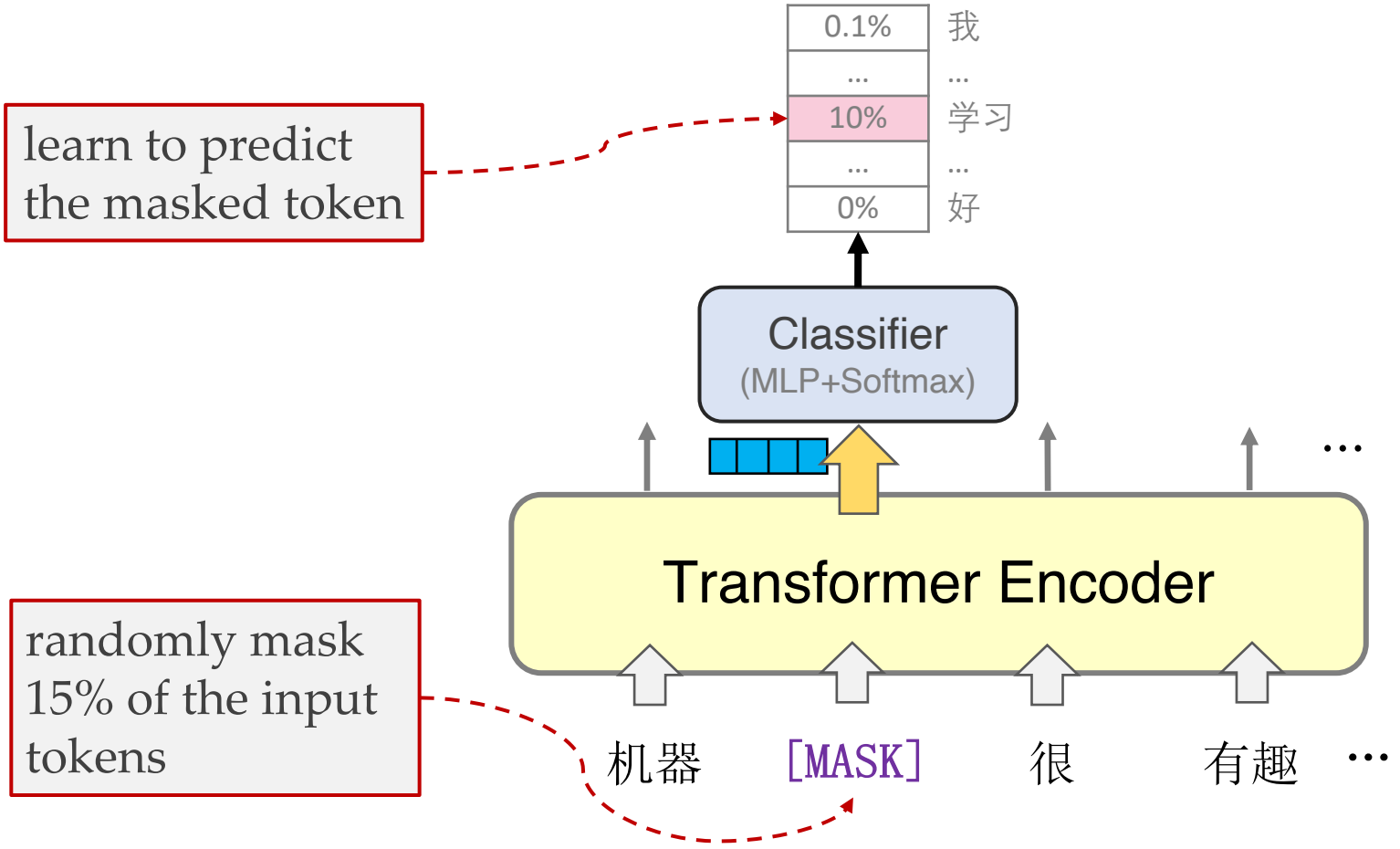


=



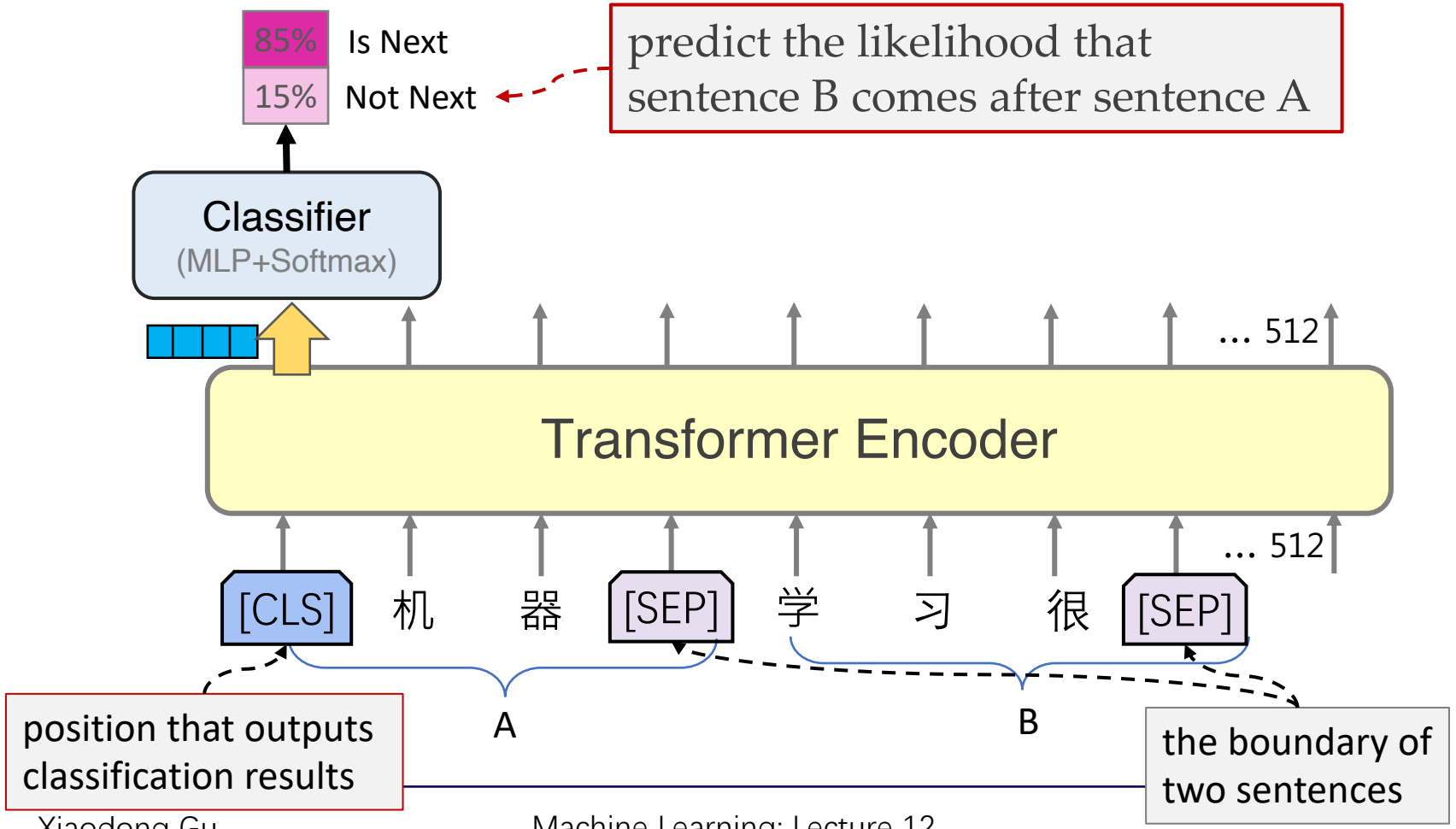
Pre-Training

Task 1: Masked Language Model (MLM)



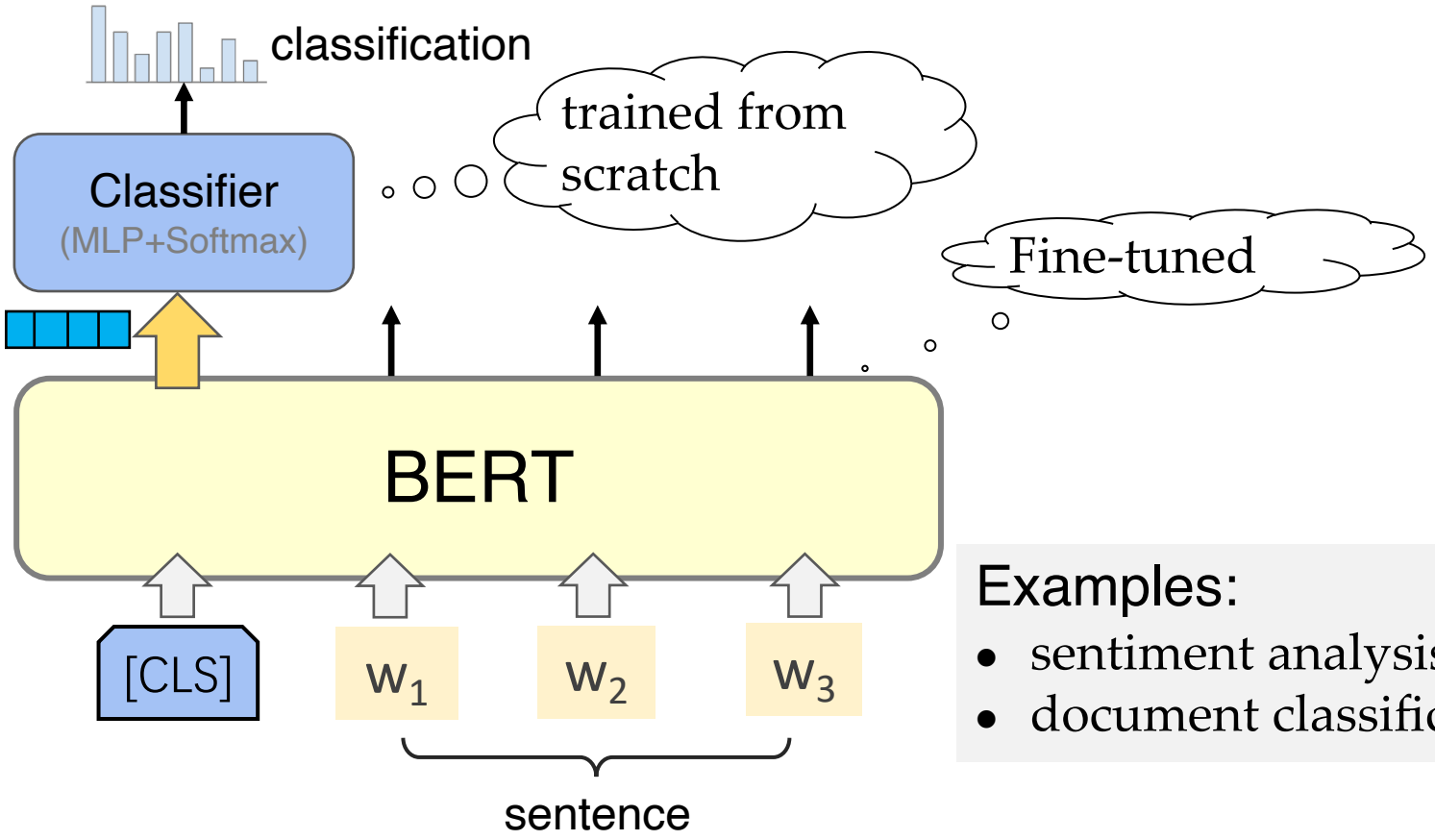
Pre-Training

Task 2: Next Sentence Prediction (NSP)



Fine-tuning: Sentence Classification

- **Input**: a single sentence, **Output**: sentence class

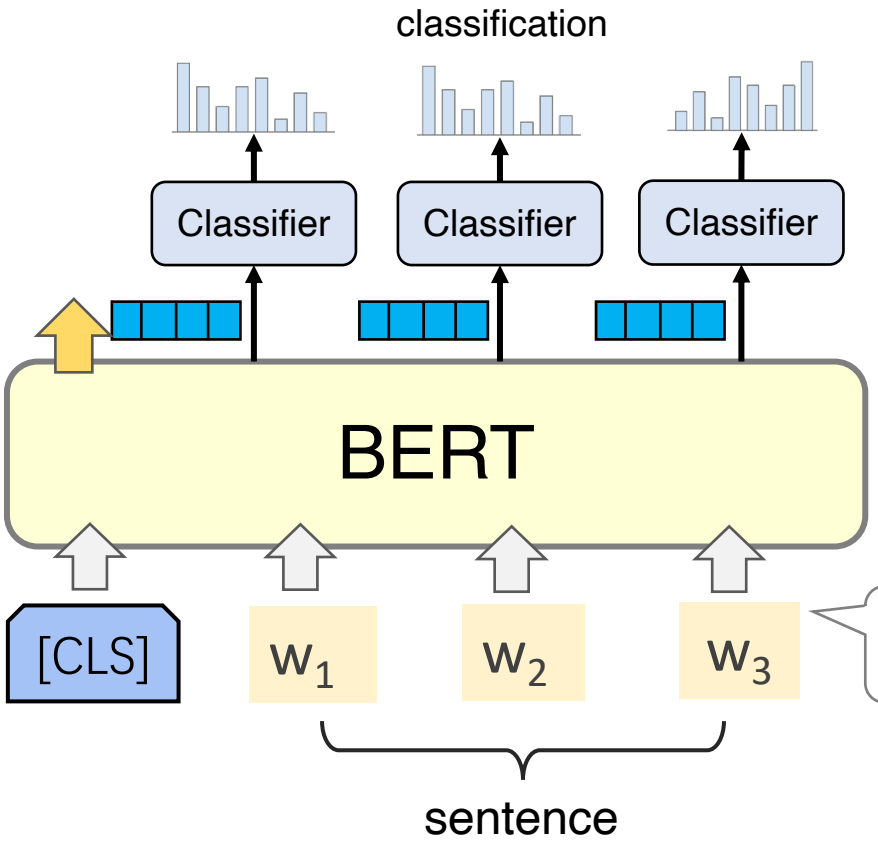


Examples:

- sentiment analysis
- document classification

Finetuning – Word Tagging

Input: a single sentence Output: classification of each word



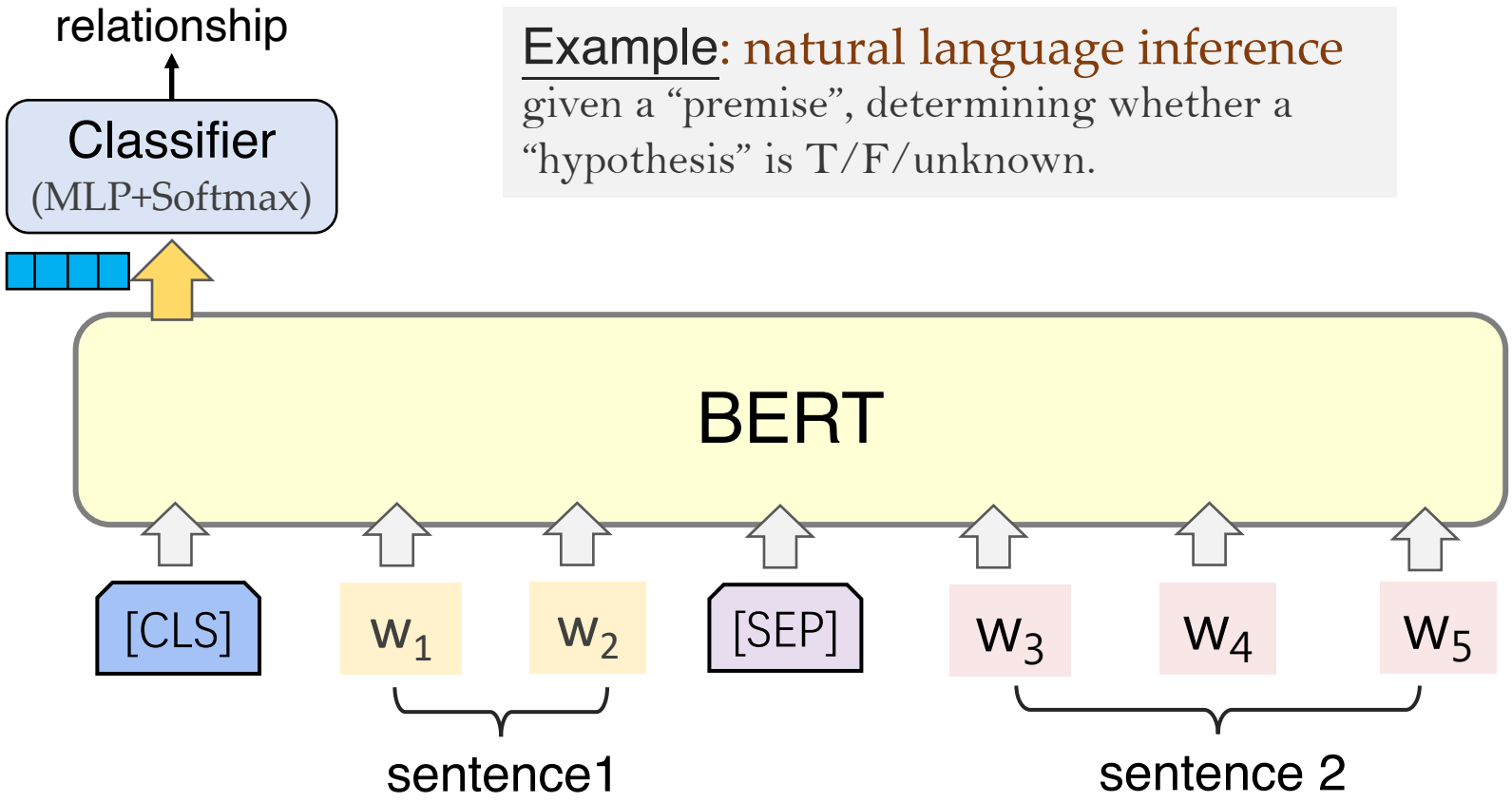
Example: slot filling

other dest other time time

Arrive Shanghai on Nov 2nd

Finetuning – Classifying Sentence Pairs

input: two sentences **output**: relationship



Example: natural language inference
given a "premise", determining whether a "hypothesis" is T/F/unknown.

Finetuning – QA (Reading Comprehension)

Task: for a given question, find the answer in a paragraph.

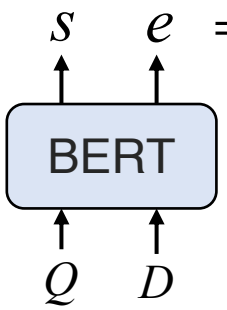
In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under **gravity**. The main forms of precipitation 17 include drizzle, rain, sleet, snow, **graupel** and hail ... Precipitation forms as smaller droplets coalesce via collision with other rain drops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

Q: what causes precipitations to fall?

A: gravity (s=17, e=17)

Q: where do water droplets collide with ice crystals to form precipitations?

A: within a cloud (s=77, e=79)

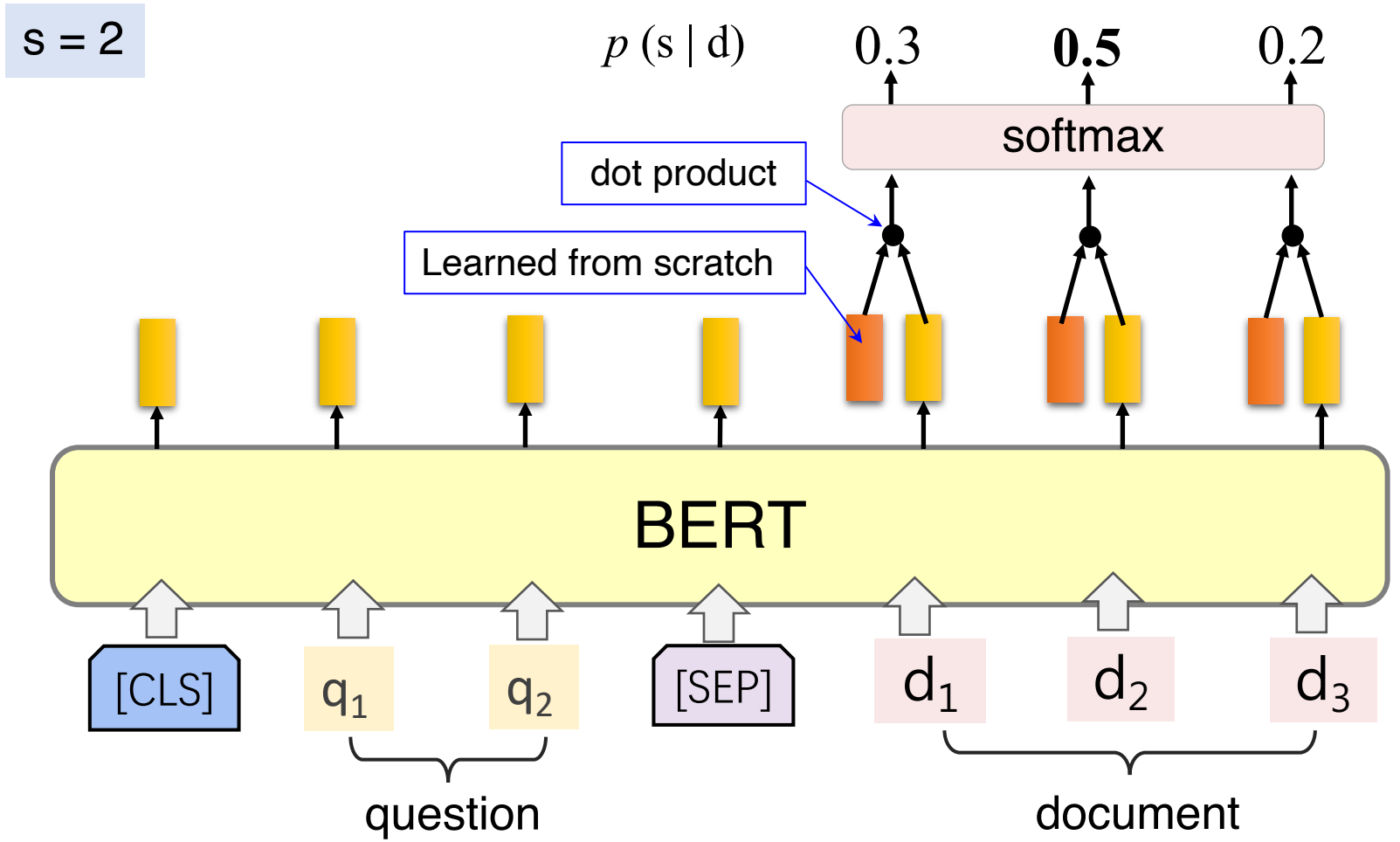


$\Rightarrow$ answer A = $\{d_s, \ldots, d_e\}$

Input:
a document D=$\{d_1, \ldots, d_N\}$
a question Q=$\{q_1, \ldots, q_N\}$

Output: two integers (s, e) which indicate the start and the end positions of the answer in the document.

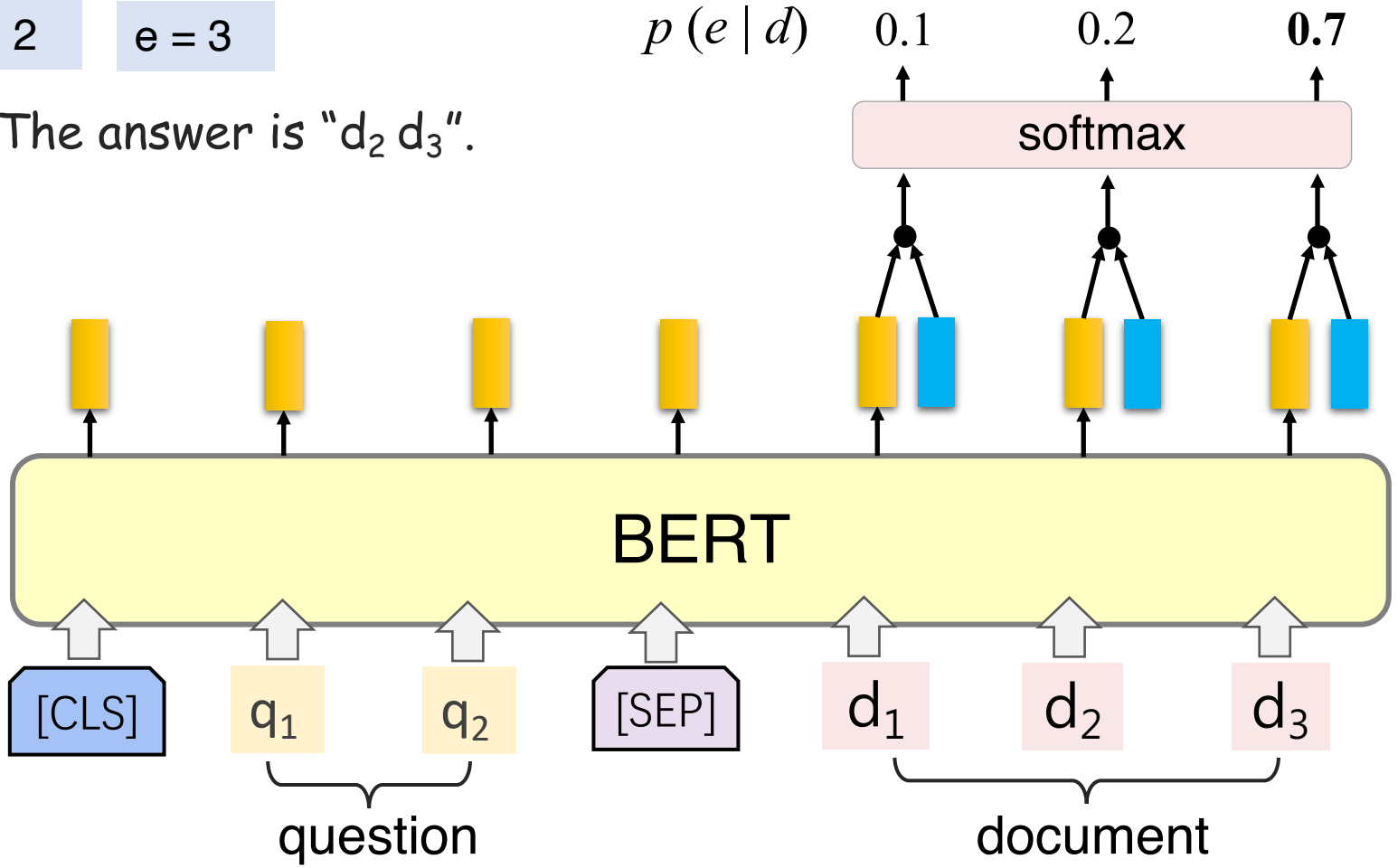
Finetuning – QA (Reading Comprehension)



Finetuning – QA (Reading Comprehension)

s = 2 e = 3

⇒ The answer is "d_2 d_3".



BERT屠榜

SQuAD 2.0

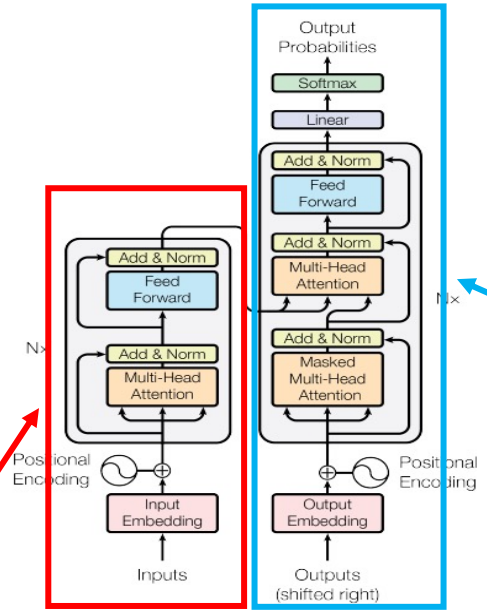
Rank	Model	EM	F1
	Human Performance *Stanford University* (Rajpurkar & Jia et al. '18)	86.831	89.452
1 Mar 20, 2019	BERT + DAE + AoA (ensemble) *Joint Laboratory of HIT and iFLYTEK Research*	**87.147**	**89.474**
2 Mar 15, 2019	BERT + ConvLSTM + MTL + Verifier (ensemble) *Layer 6 AI*	86.730	89.286
3 Mar 05, 2019	BERT + N-Gram Masking + Synthetic Self-Training (ensemble) *Google AI Language* https://github.com/google-research/bert	86.673	89.147
4 May 21, 2019	XLNet (single model) *XLNet Team*	86.346	89.133
5 Apr 13, 2019	SemBERT(ensemble) *Shanghai Jiao Tong University*	86.166	88.886

https://paperswithcode.com/sota/question-answering-on-squad20

Generative Pre-Trained Transformer (GPT)

- A **pretrained language model** based on Transformer decoder.

BERT (340M)



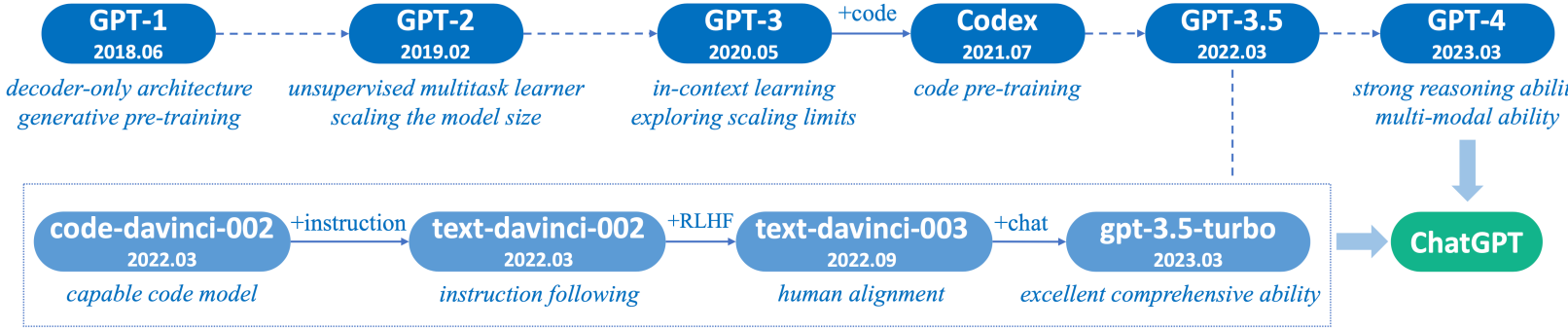
GPT-2 (1542M)

[OpenAI 2019]

GPT-3 (175B)

[OpenAI 2020]

GPT Evolution



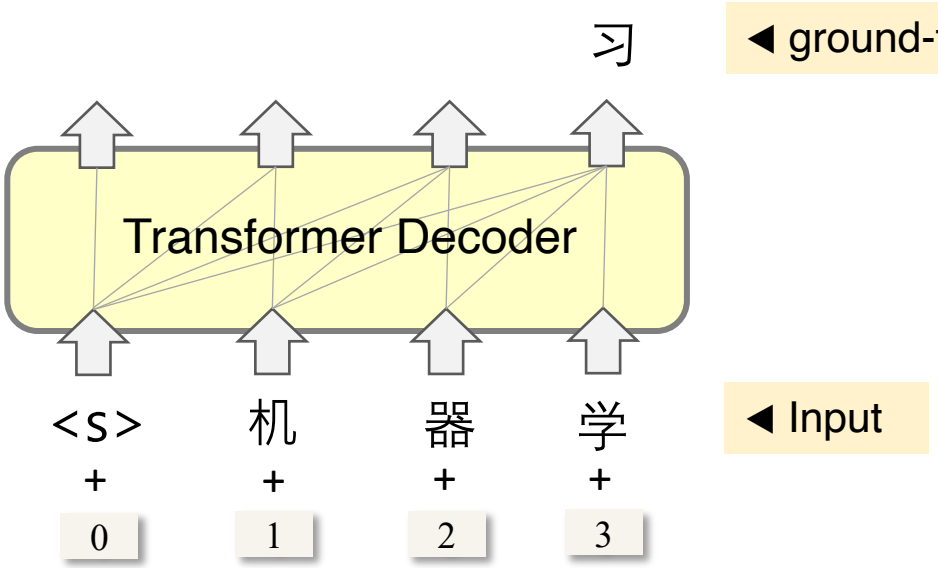
Simulated exams	GPT-4 estimated percentile	GPT-4 (no vision) estimated percentile	GPT-3.5 estimated percentile
Uniform Bar Exam (MBE+MEE+MPT)[1]	298 / 400 ~90th	298 / 400 ~90th	213 / 400 ~10th
LSAT	163 ~88th	161 ~83rd	149 ~40th
SAT Evidence-Based Reading & Writing	710 / 800 ~93rd	710 / 800 ~93rd	670 / 800 ~87th
SAT Math	700 / 800 ~89th	690 / 800 ~89th	590 / 800 ~70th
Graduate Record Examination (GRE) Quantitative	163 / 170 ~80th	157 / 170 ~62nd	147 / 170 ~25th
Graduate Record Examination (GRE) Verbal	169 / 170 ~99th	165 / 170 ~96th	154 / 170 ~63rd
Graduate Record Examination (GRE) Writing	4 / 6 ~54th	4 / 6 ~54th	4 / 6 ~54th
USABO Semifinal Exam 2020	87 / 150 99th - 100th	87 / 150 99th - 100th	43 / 150 31st - 33rd
USNCO Local Section Exam 2022	36 / 60	38 / 60	24 / 60
Medical Knowledge Self-Assessment Program	75 %	75 %	53 %
Codeforces Rating	392 below 5th	392 below 5th	260 below 5th

A Survey of Large Language Models https://arxiv.org/abs/2303.18223

Pre-Training GPT

Simply Language Model:

— Given the preceding n-1 tokens, predict the n-th token.

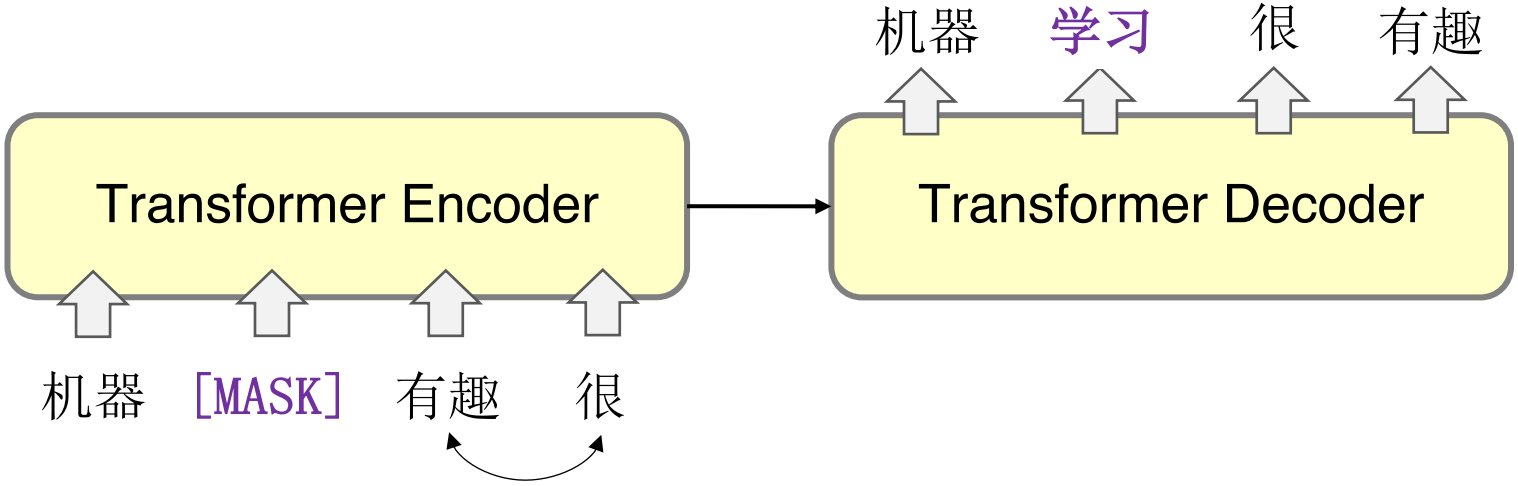


Training objective:

$$l(\theta) = -\sum_{t=1}^{T} \log p_\theta\left(x_t | x_1, \ldots, x_{t-1}\right)$$

BART (Denoising Seq-to-Seq Pretraining)

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

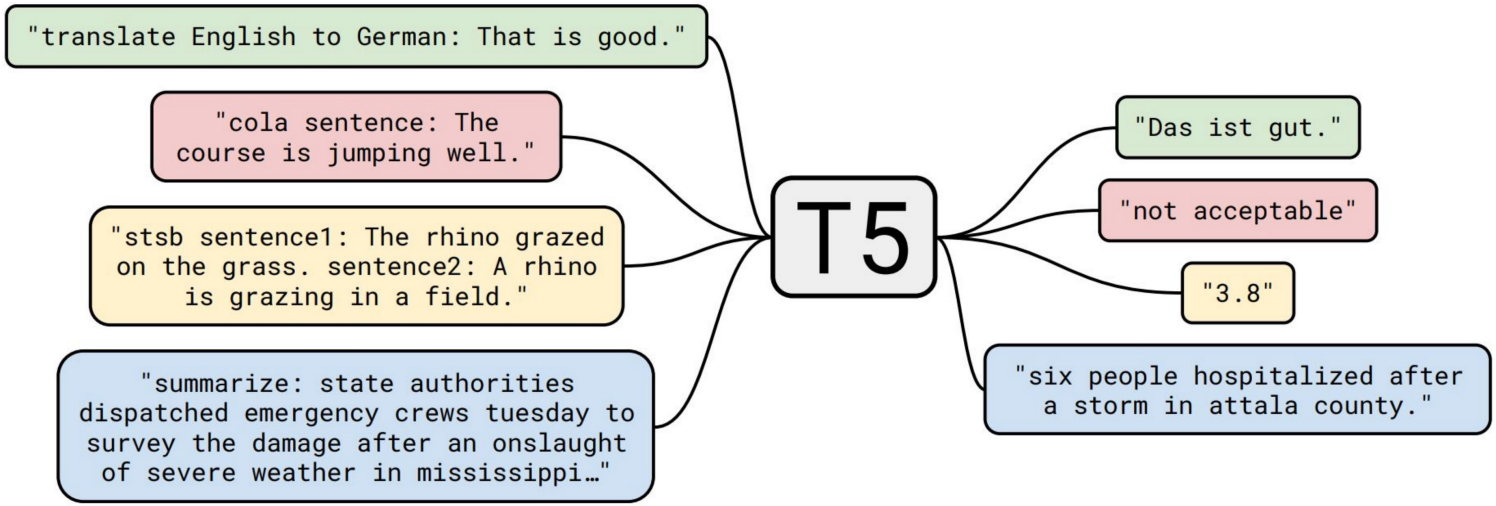


Mike Lewis, et al. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. ACL 2020.

T5: Text-to-Text Transfer Transformer

Idea: Every task, one format!

"[Task-specific prefix]: [Input text]" → "[output text]"



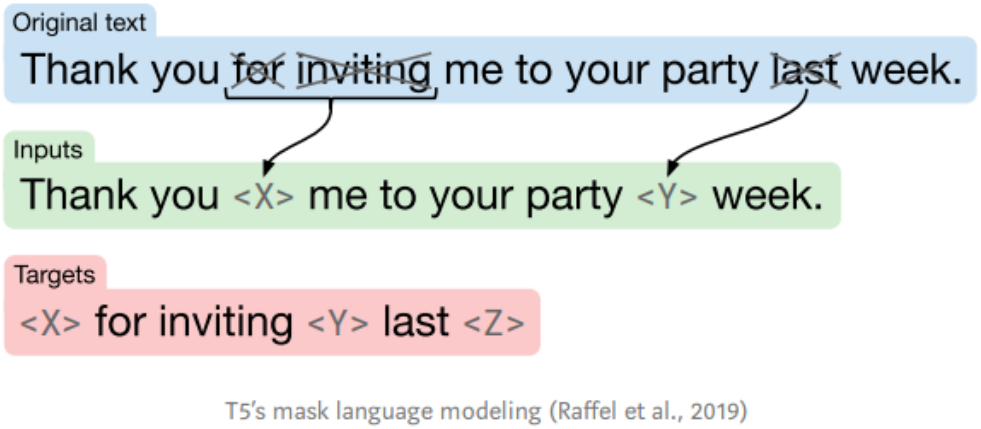
Treat every text processing problem as a "text-to-text" problem

T5: Text-to-Text Transfer Transformer

Architecture: Transformer Encoder-Decoder

Data: 750GB web-extracted text

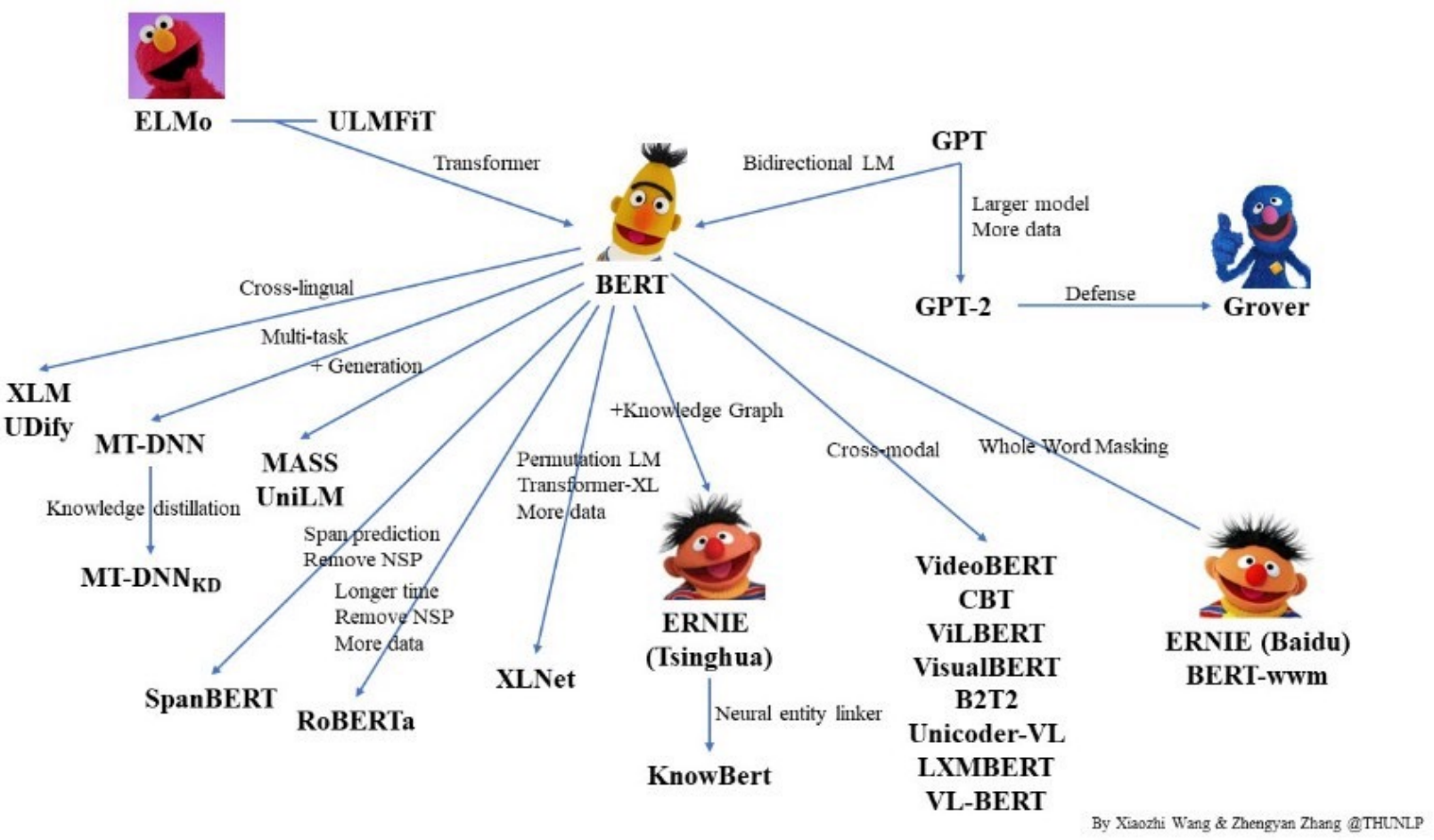
Pretraining: Fill-in-the-Blank Denoising



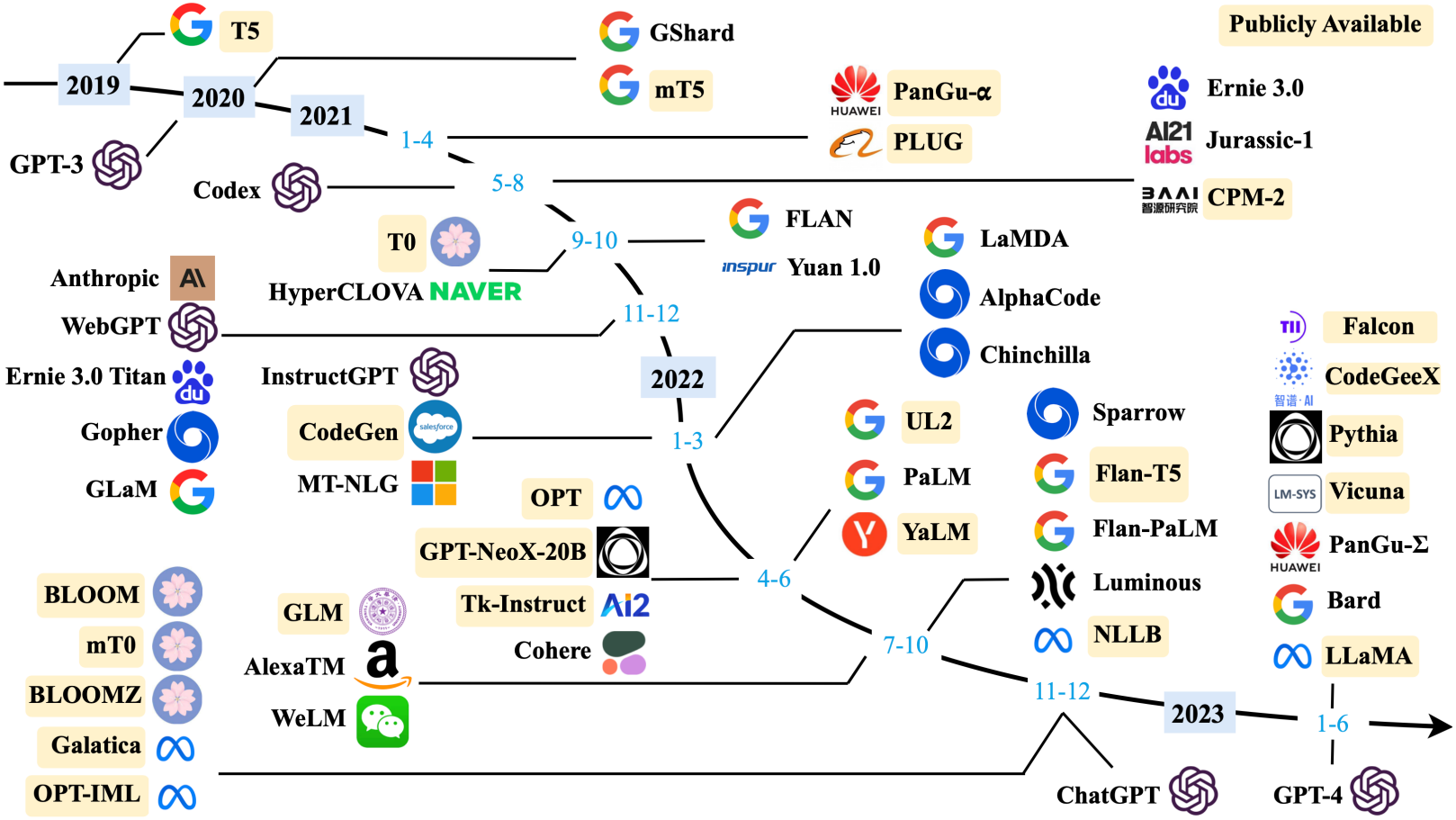
T5's mask language modeling (Raffel et al., 2019)

Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer.2020. https://arxiv.org/abs/1910.10683

The PLM Family (till 2020)



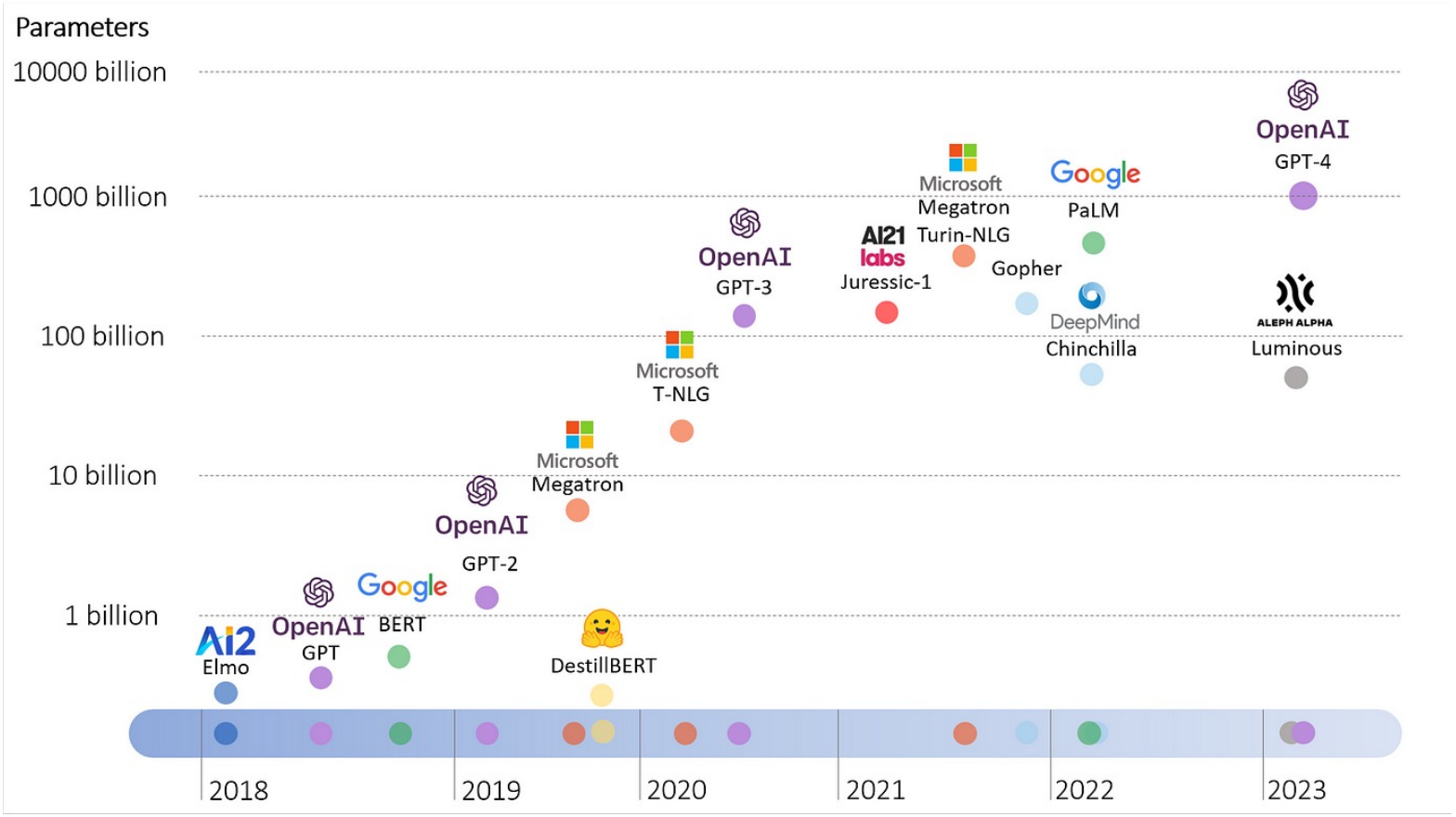
The LLM Family (2019-)



A Survey of Large Language Models https://arxiv.org/abs/2303.18223

Make Them Bigger

Trend of sizes of state-of-the-art NLP models over time on a logarithmic scale



TIME for Coding

Tutorial: Fine-tune gpt2 to generate stories

https://github.com/huggingface/transformers/tree/main/src/transformers/models

https://www.kaggle.com/code/emily2008/fine-tune-gpt-2-to-generate-stories/notebook

What's Next?

Convolutional Neural Networks

The world of computer vision!

- A new type of neural networks that can understand images
- Computer vision tasks: classification, detection, segmentation, ..