南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Advanced Natural Language Processing

Lecture 7: Transformer

陈冠华 CHEN Guanhua

Department of Statistics and Data Science

Content

- Transformer modules
- Code example

Transformer: Is Attention All We Need?

- Proposed by Google brain in 2017
- Nearly applied in every state-of-the-art NLP model today
  - Even CV models and protein/music/audio models

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

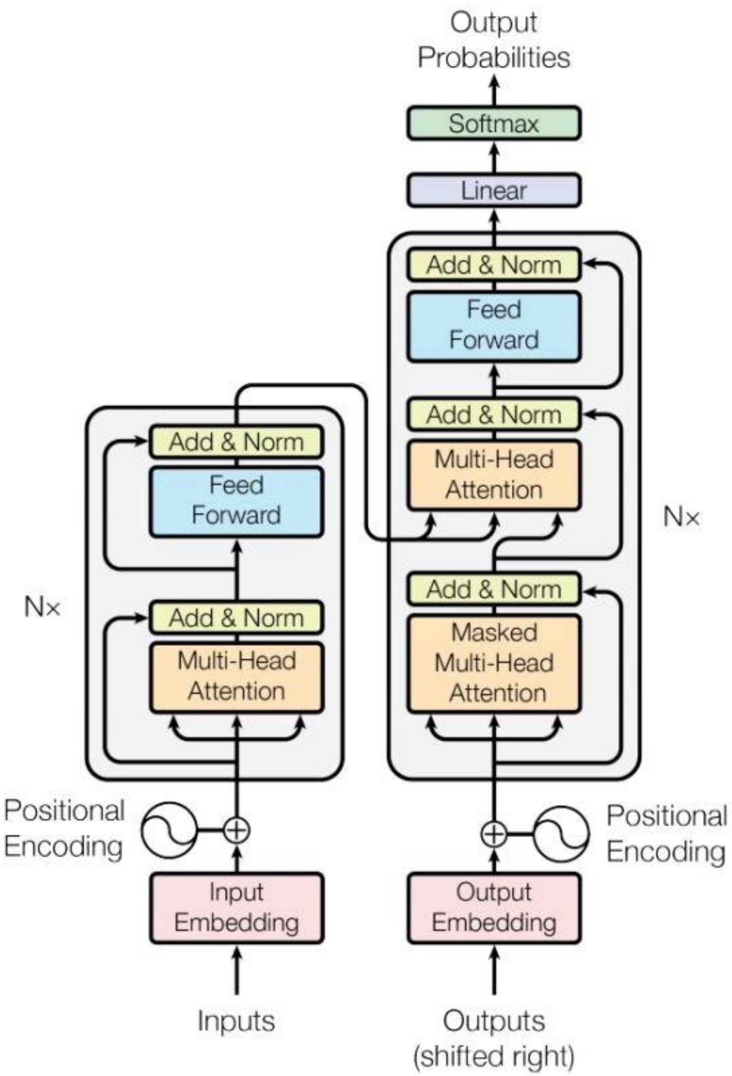
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Courtesy of Paramount Pictures

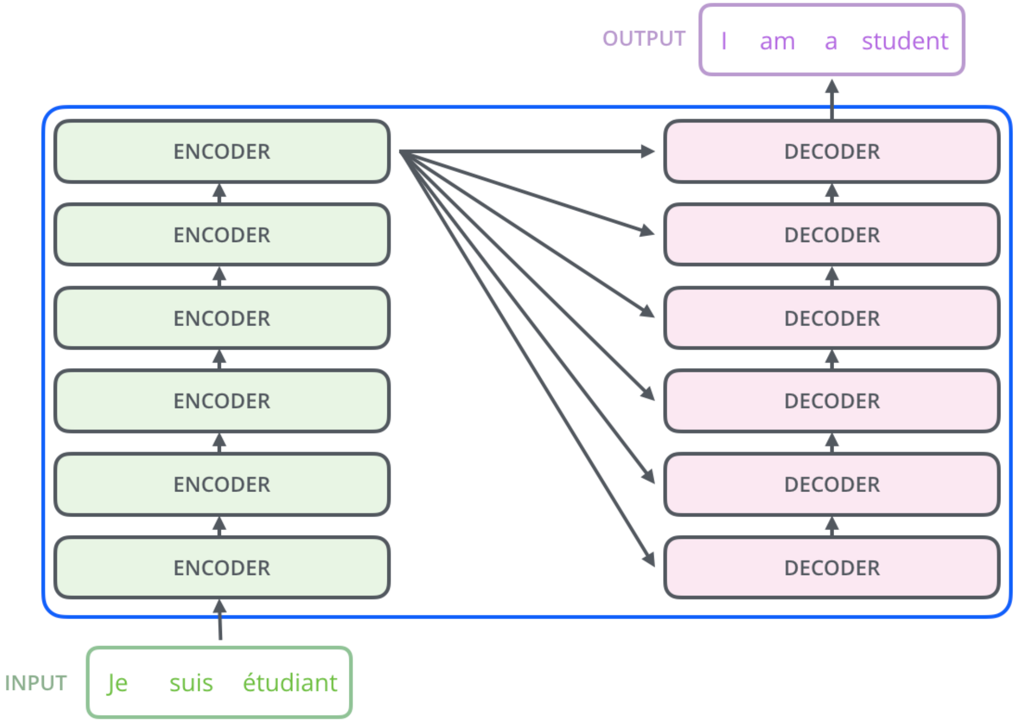
Transformer

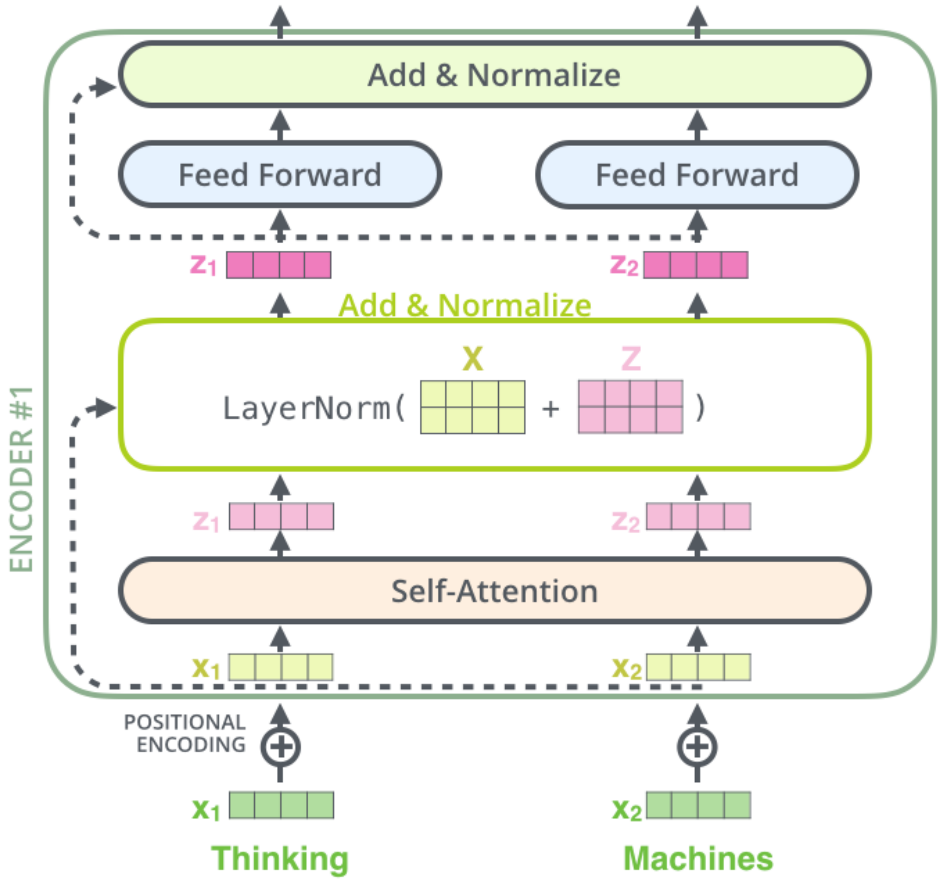
南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY



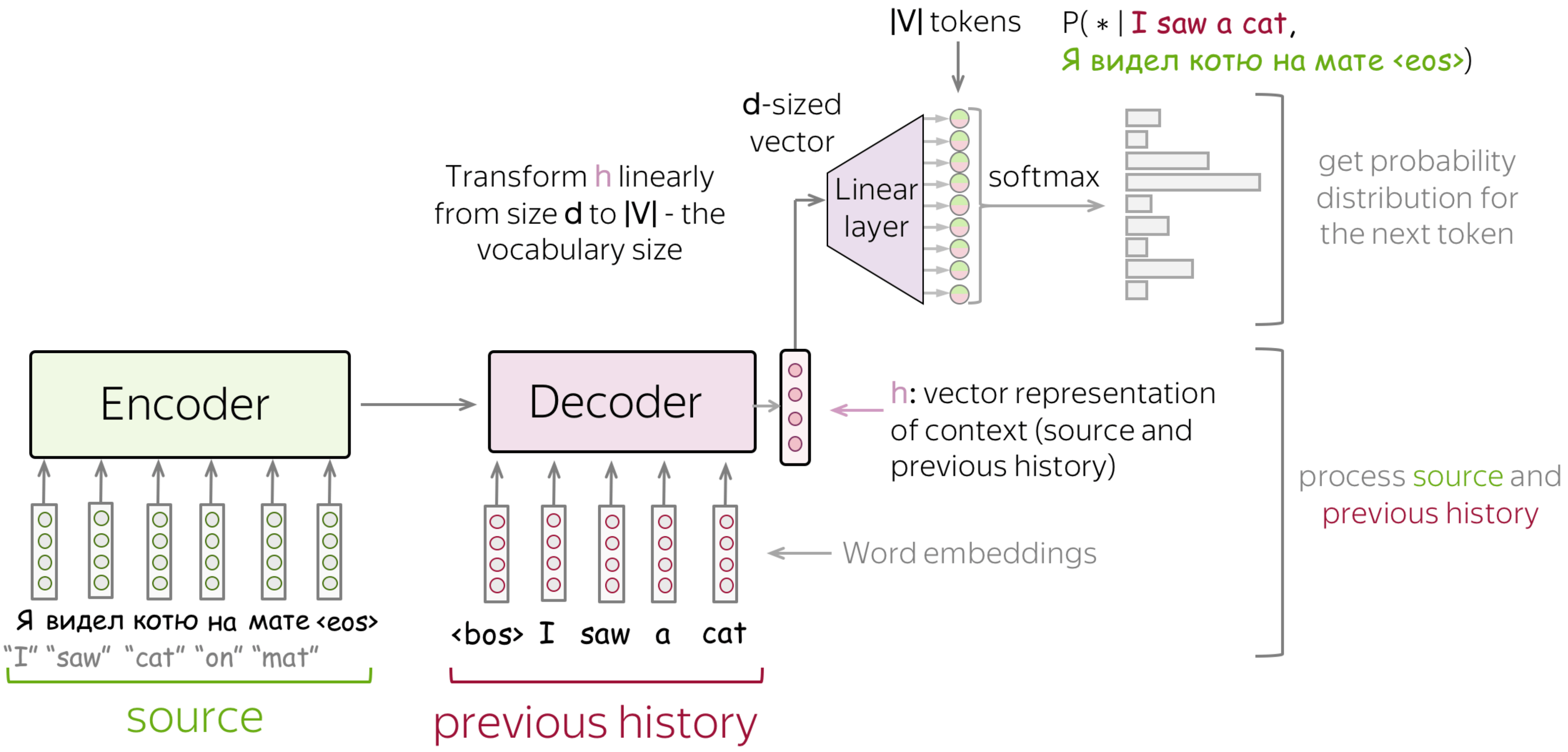
- Purely rely on attention mechanism
  - No CNN or RNN
- Stacked encoder and decoder layers
- Self-attention and cross-attention
- Feed forward layers
- Layer normalization
- Residual connection

Transformer



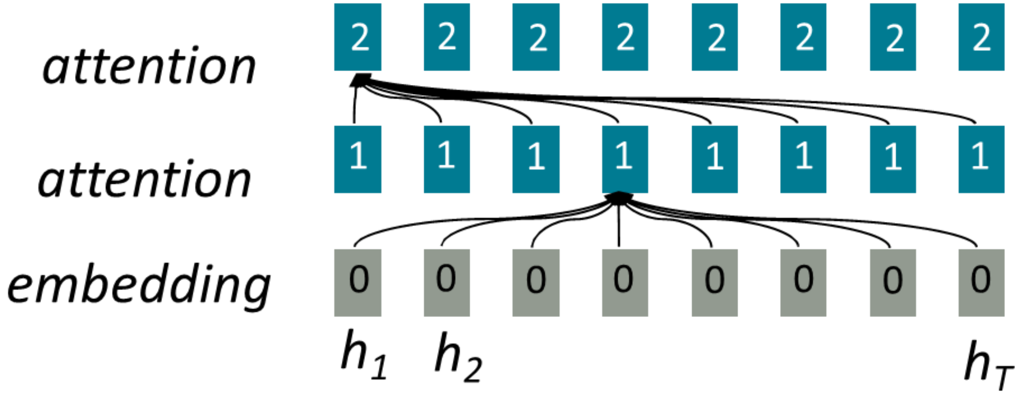


Transformer



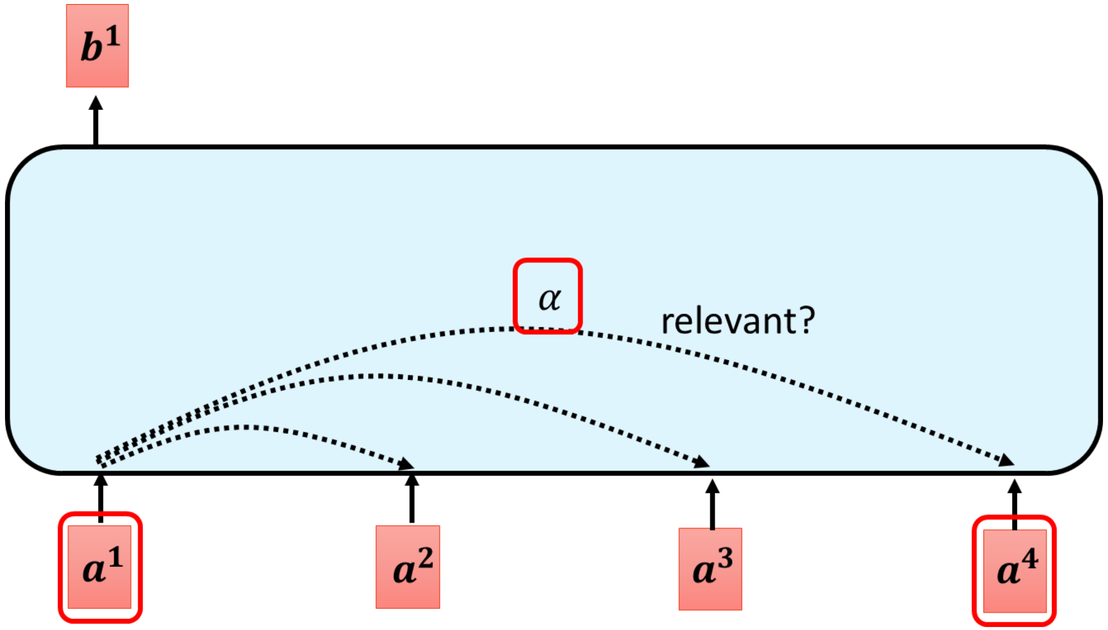
Self-Attention

- Attention treats each word's representation as a query to access and incorporate information from a set of values.
- Self-attention is encoder-encoder (or decoder-decoder) attention where each word attends to each other word within the input (or output).

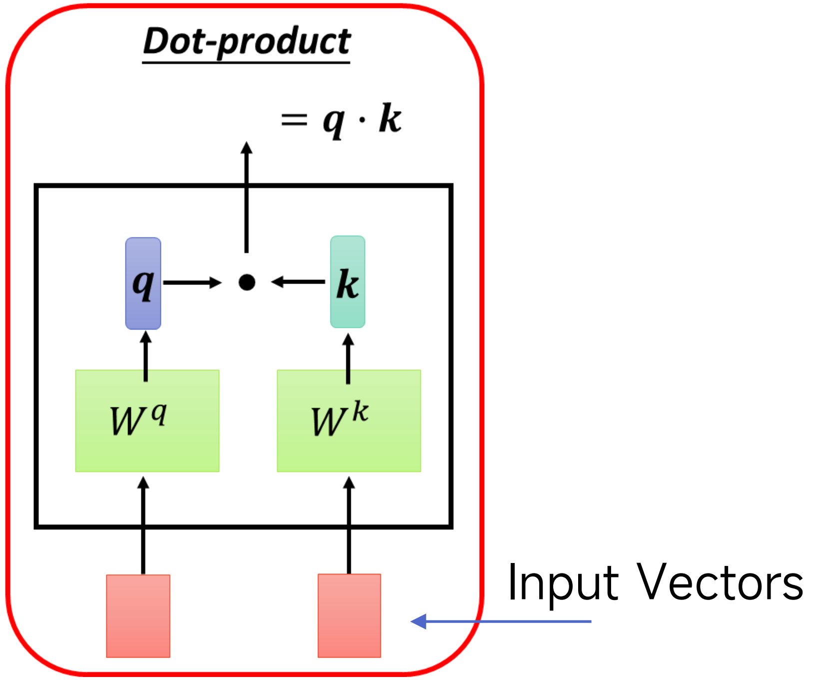


All words attend to all words in previous layer; most arrows here are omitted

SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY



- query - asking for information;
- key - saying that it has some information;
- value - giving the information.
- (key-value) storage

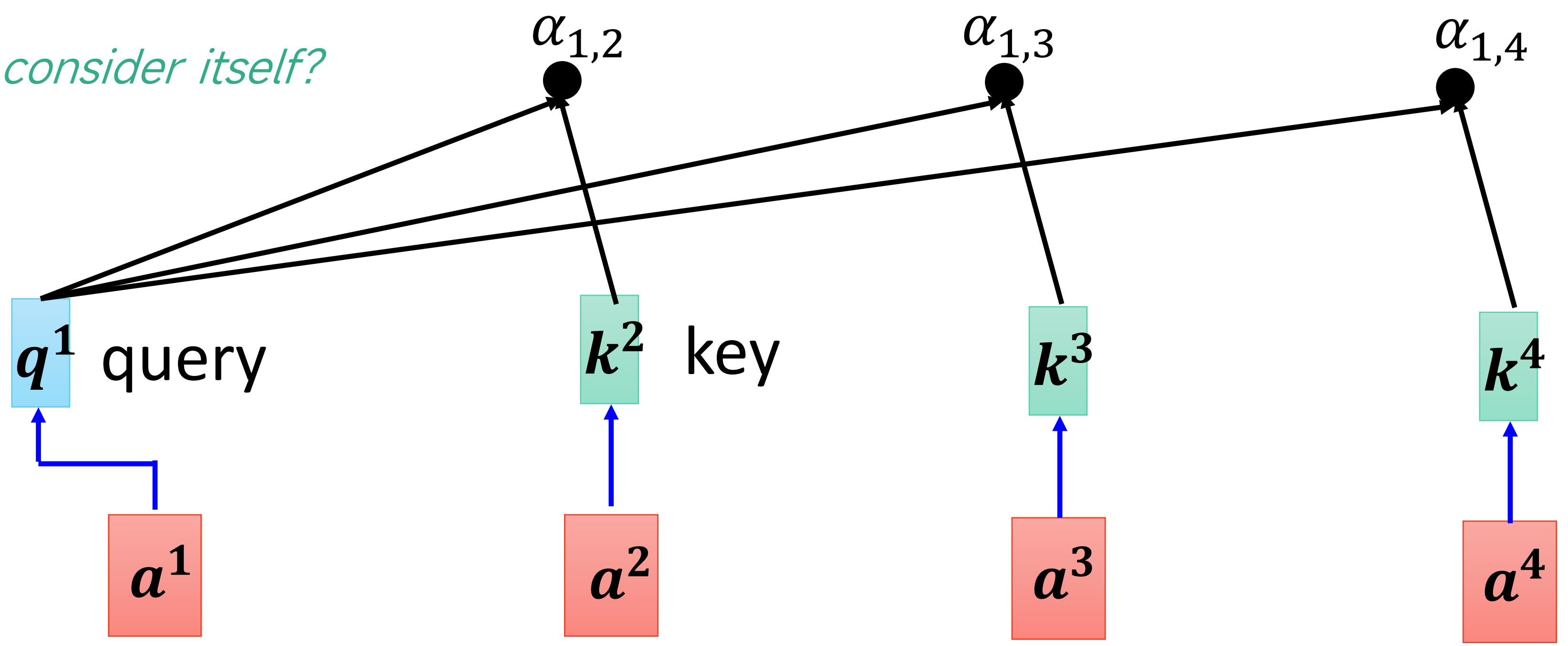


Find the relevant vectors in a sequence

Self-Attention

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

Shall we consider itself?

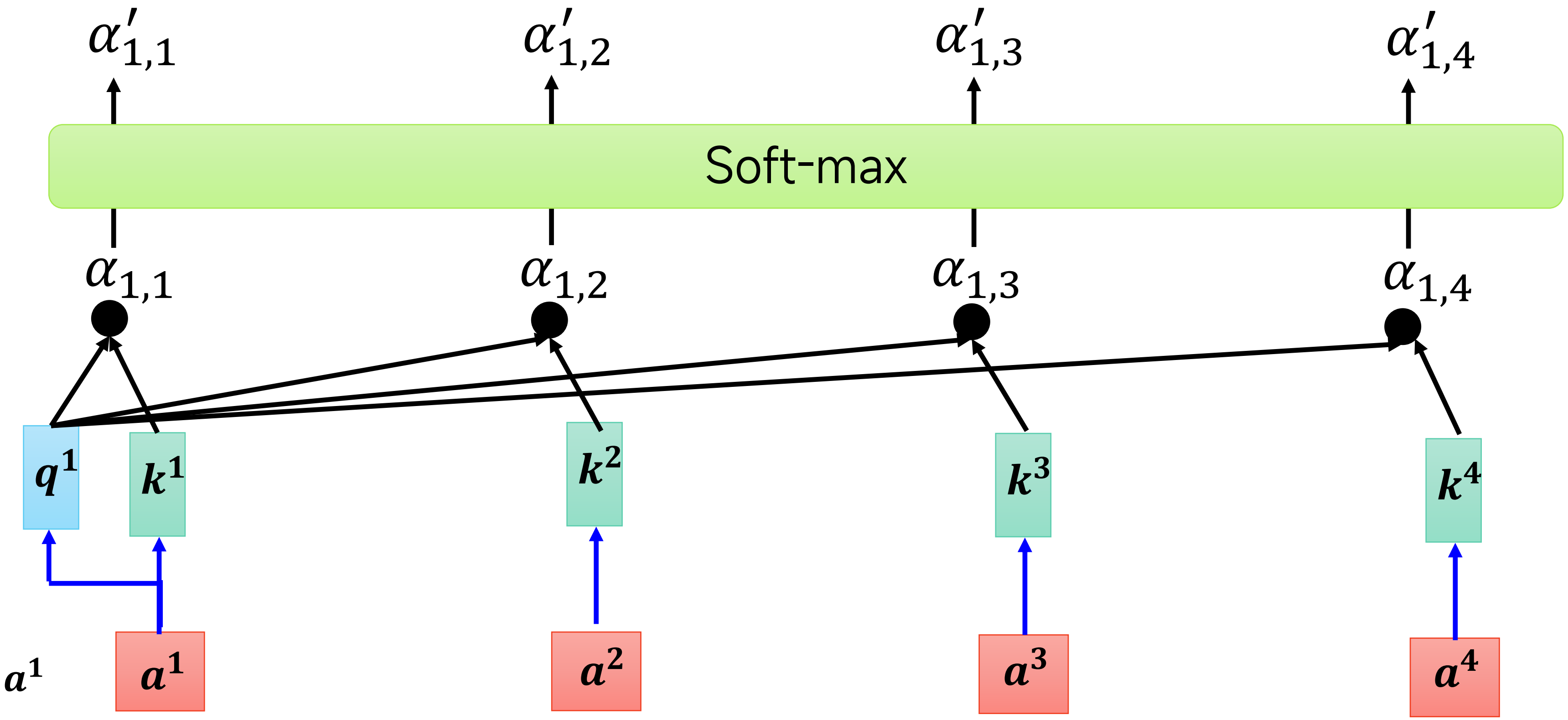


$\boldsymbol{q}^1 = W^q \boldsymbol{a}^1$ $\boldsymbol{k}^2 = W^k \boldsymbol{a}^2$ $\boldsymbol{k}^3 = W^k \boldsymbol{a}^3$ $\boldsymbol{k}^4 = W^k \boldsymbol{a}^4$

Self-Attention

attention score

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



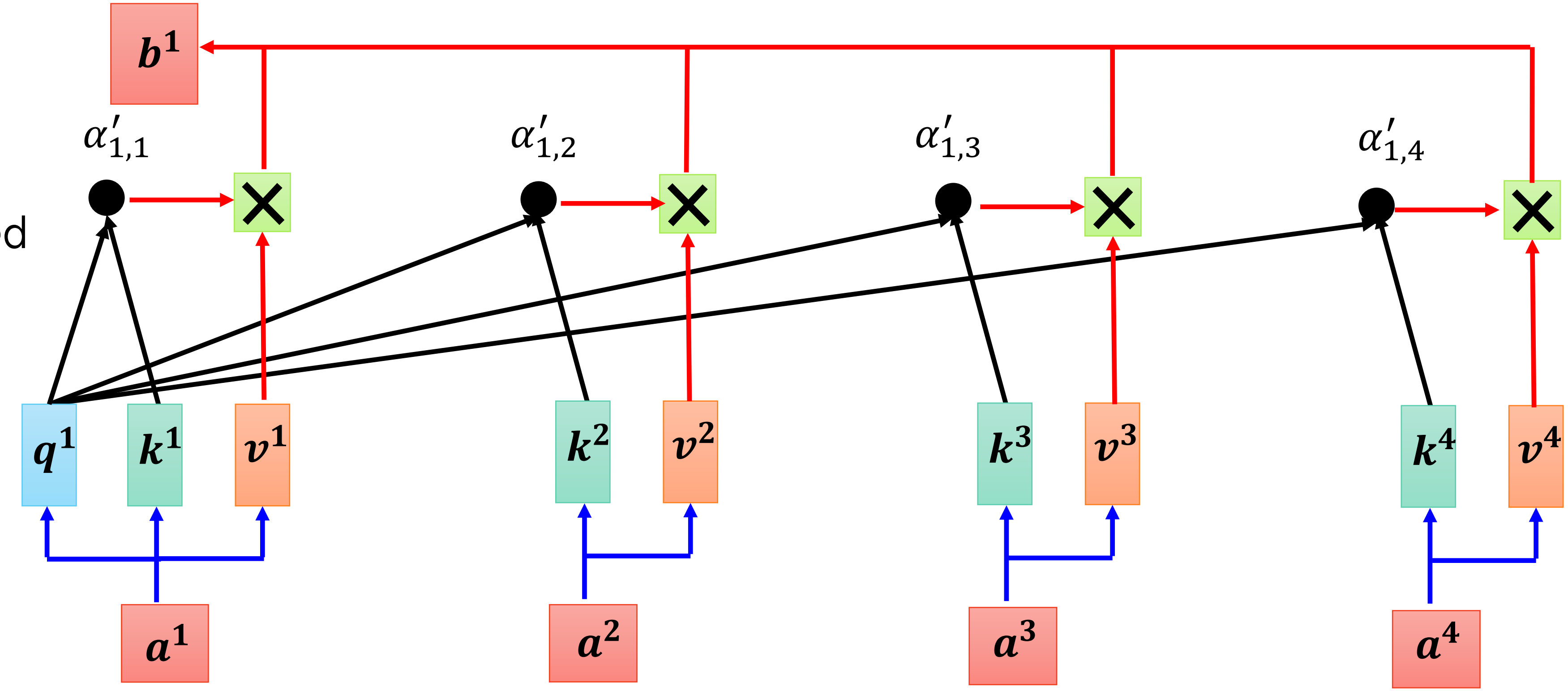
$\boldsymbol{q}^1 = W^q \boldsymbol{a}^1$

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

Self-Attention

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

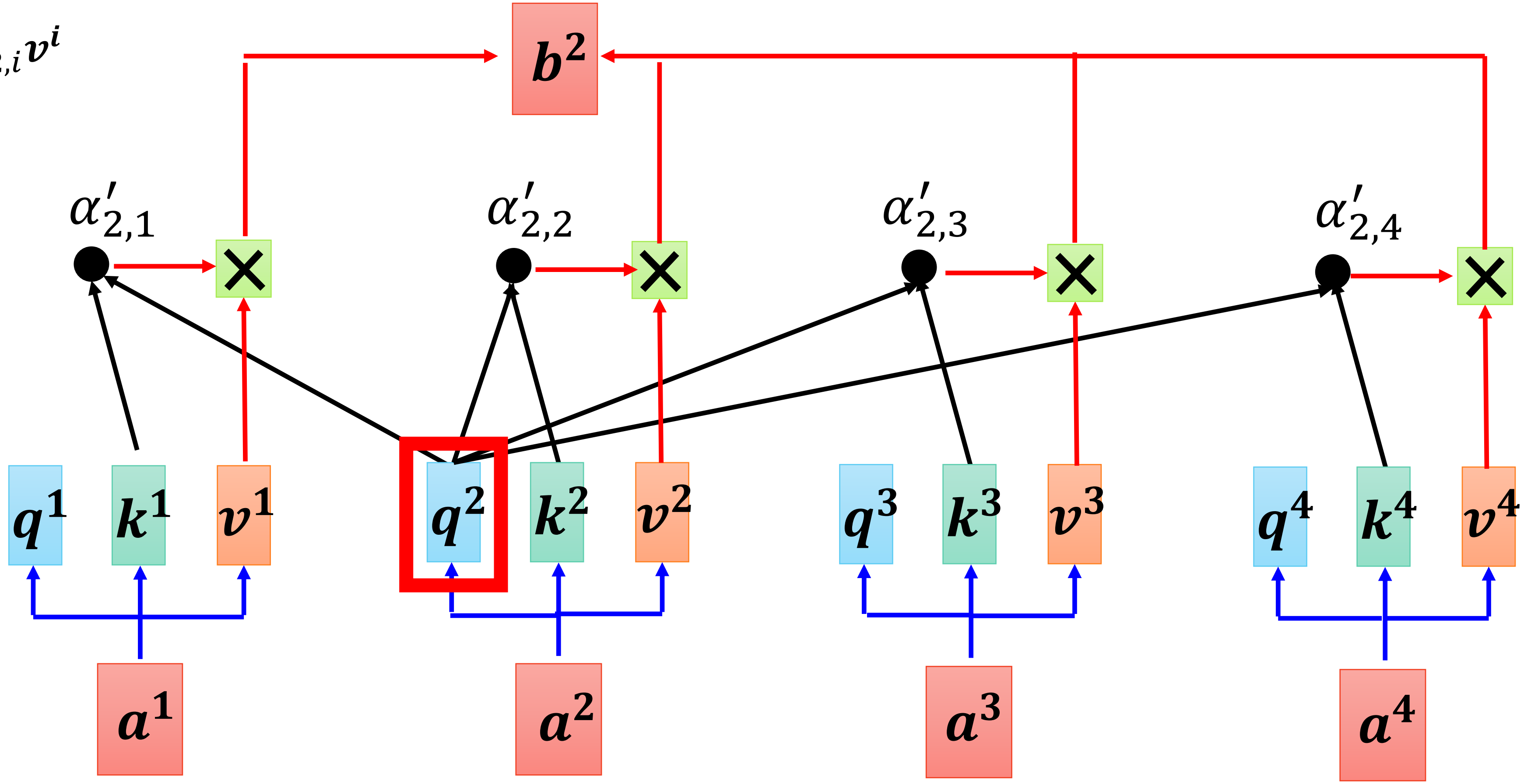
- Extract information based on attention scores
- Weighted sum of input vectors



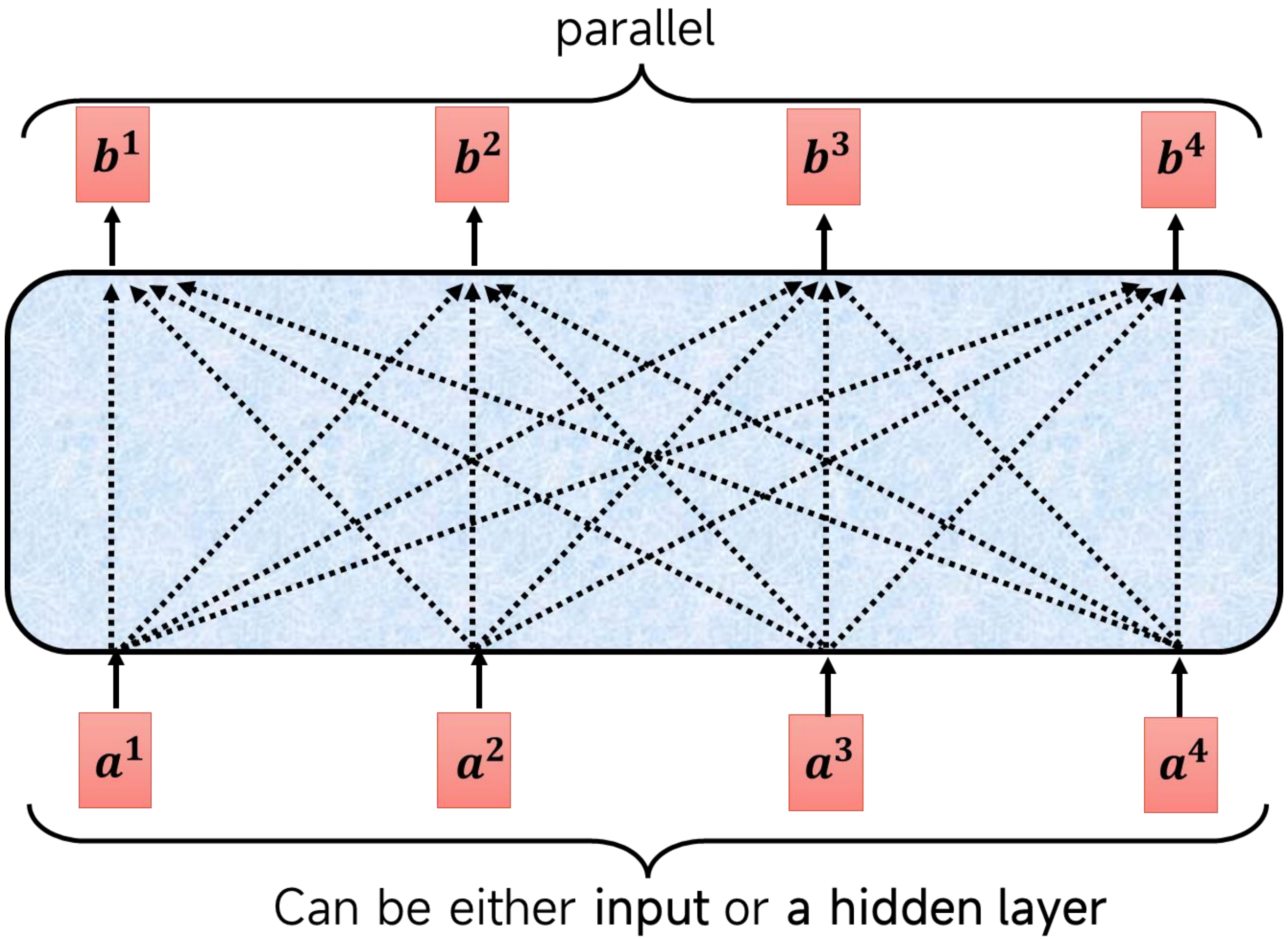
$\boldsymbol{v}^1 = W^v \boldsymbol{a}^1$ $\boldsymbol{v}^2 = W^v \boldsymbol{a}^2$ $\boldsymbol{v}^3 = W^v \boldsymbol{a}^3$ $\boldsymbol{v}^4 = W^v \boldsymbol{a}^4$

Self-Attention

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



Self-Attention

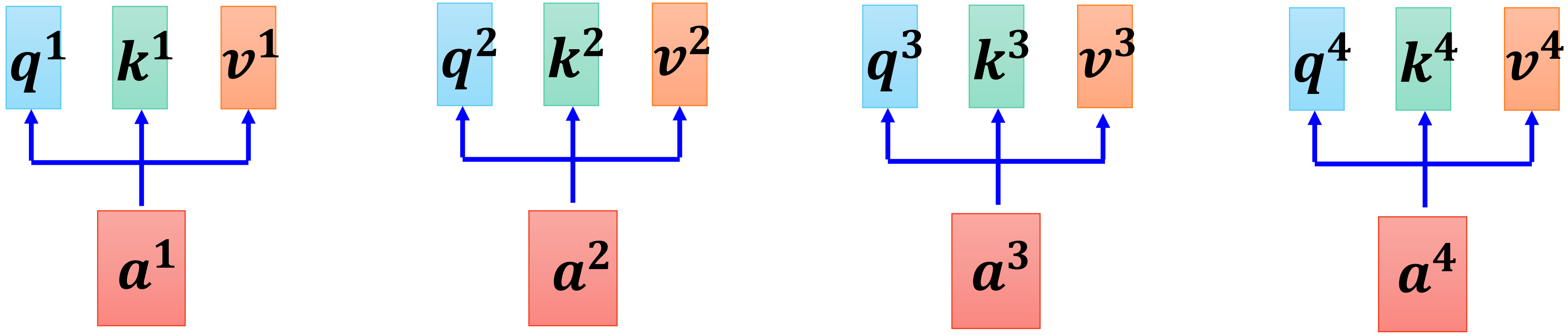


Self-Attention

$$q^i = W^q a^i \qquad \underbrace{q^1\, q^2\, q^3\, q^4}_{Q} = W^q\, \underbrace{a^1\, a^2\, a^3\, a^4}_{I}$$

$$k^i = W^k a^i \qquad \underbrace{k^1\, k^2\, k^3\, k^4}_{K} = W^k\, \underbrace{a^1\, a^2\, a^3\, a^4}_{I}$$

$$v^i = W^v a^i \qquad \underbrace{v^1\, v^2\, v^3\, v^4}_{V} = W^v\, \underbrace{a^1\, a^2\, a^3\, a^4}_{I}$$

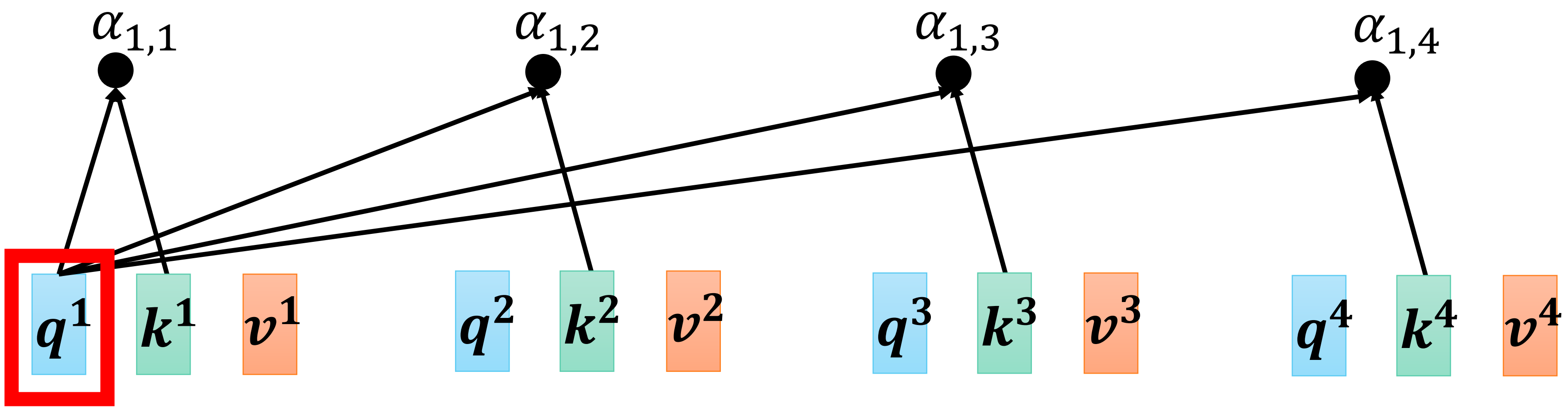


Self-Attention

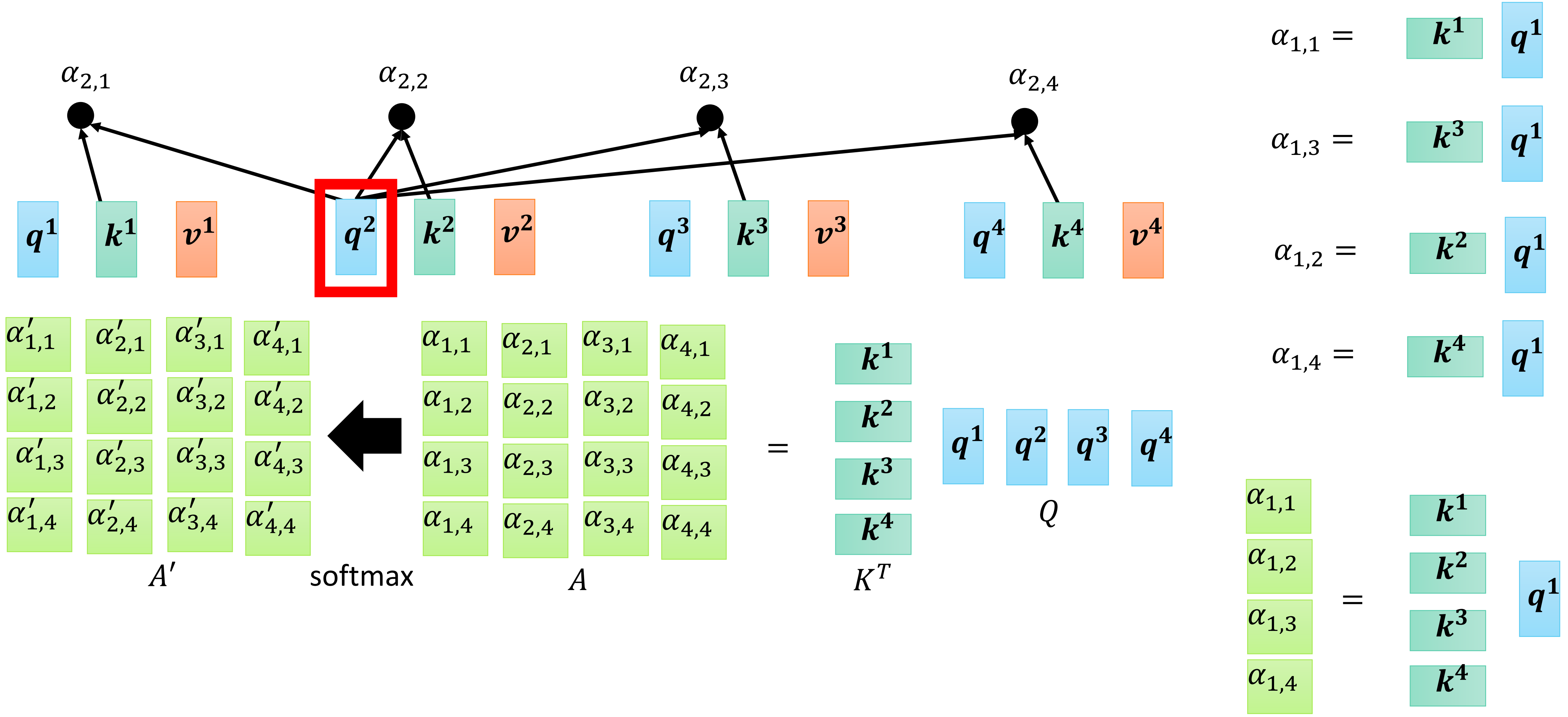
$$\alpha_{1,1} = \boldsymbol{k}^1 \; \boldsymbol{q}^1 \qquad \alpha_{1,2} = \boldsymbol{k}^2 \; \boldsymbol{q}^1$$

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

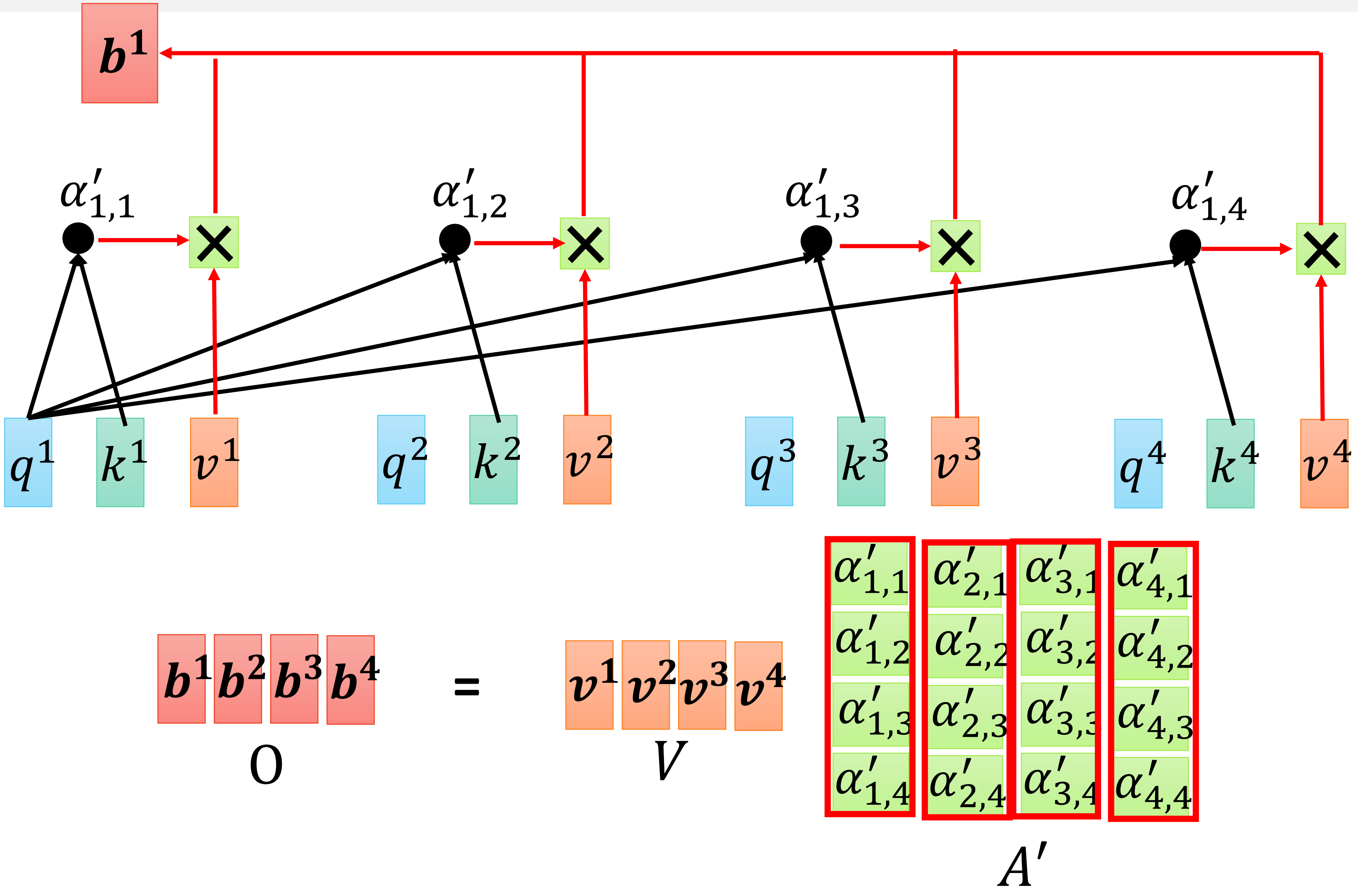
$$\begin{bmatrix} \alpha_{1,1} \\ \alpha_{1,2} \\ \alpha_{1,3} \\ \alpha_{1,4} \end{bmatrix} = \begin{bmatrix} \boldsymbol{k}^1 \\ \boldsymbol{k}^2 \\ \boldsymbol{k}^3 \\ \boldsymbol{k}^4 \end{bmatrix} \boldsymbol{q}^1$$



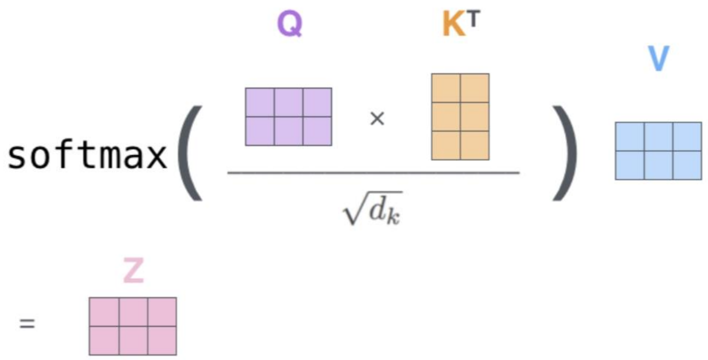
Self-Attention



Self-Attention



Self-Attention



$$A' \leftarrow A = K^T Q$$

Attention Matrix

$$Q = \boxed{W^q} I$$
$$K = \boxed{W^k} I$$
$$V = \boxed{W^v} I$$

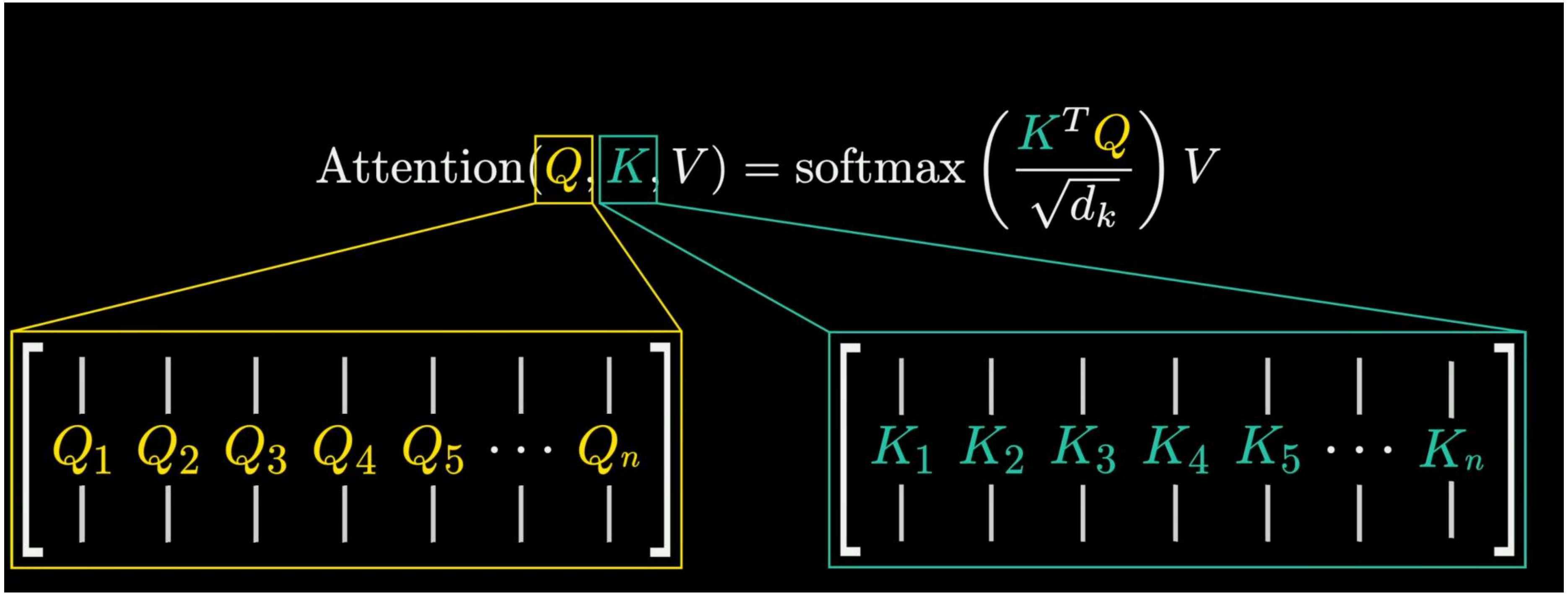
Parameters to be learned

$$W^q, W^k, W^v$$

$$a^i \rightarrow b^i$$

$$O = V A'$$

Self-Attention



[Attention in transformers, step-by-step | Deep Learning Chapter 6]

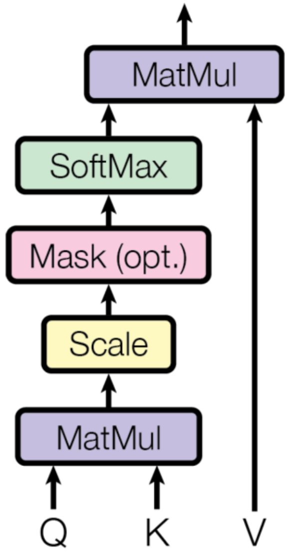
Self-Attention

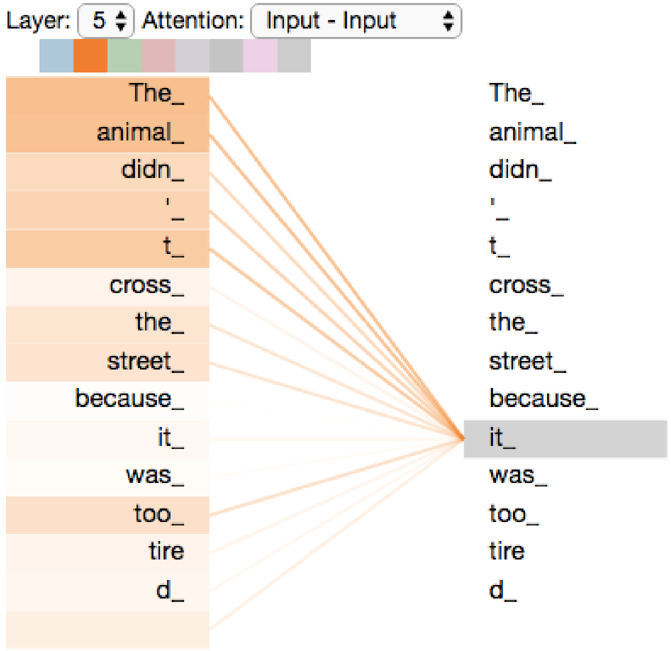
- The input hidden states $X = [x_1, x_2, \dots, x_S]^T$
  - For position i, hidden state is x_i
- The key and value input are X
  - $K = W^K X, V = W^V X$
- For position i, the query input is x_i
  - $Q_i = W^Q x_i$
- The self-attention output is calculated as

$$y_i = \text{softmax}(\frac{K^T Q_i}{\sqrt{d_k}})V$$

Put the weighted sum in a matrix form,

$$y = \text{softmax}\left(\frac{K^T Q}{\sqrt{d_k}}\right)V$$





Self-Attention

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- To illustrate why the dot products get large, assume that the components of q and k are independent random variables with mean 0 and variance 1. Then their dot product,

$$q \cdot k = \sum_{i=1}^{d_k} q_i k_i$$

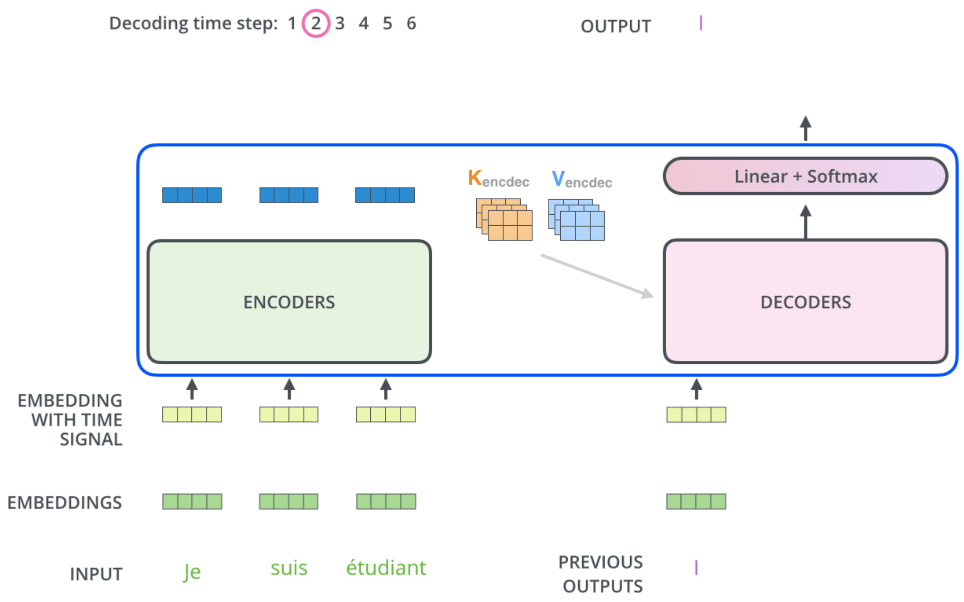
has mean 0 and variance d_k.

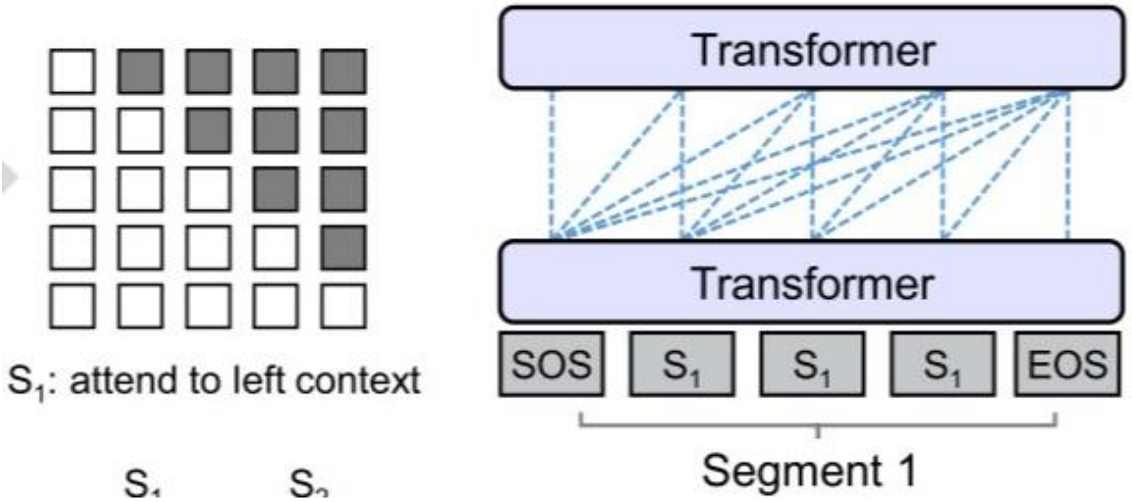
[4]To illustrate why the dot products get large, assume that the components of q and k are independent random variables with mean 0 and variance 1. Then their dot product, $q \cdot k = \sum_{i=1}^{d_k} q_i k_i$, has mean 0 and variance d_k.

[从熵不变性看Attention的Scale操作 - 科学空间]

Masked Self-Attention

- Many models generate texts in an auto-regressive manner, from left to right

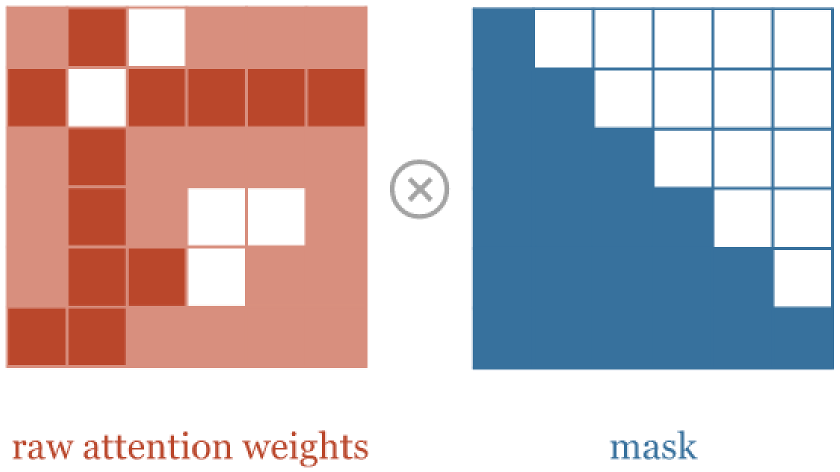




Masked Self-Attention

Efficient implementation: compute attention as we normally do, mask out attention to future words by setting attention scores to $-\infty$

$$\alpha = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})$$




```
dot = torch.bmm(queries, keys.transpose(1, 2))

indices = torch.triu_indices(t, t, offset=1)
dot[:, indices[0], indices[1]] = float('-inf')

dot = F.softmax(dot, dim=2)
```

Missing $\sqrt{d_k}$

http://peterbloem.nl/blog/transformers

Multi-head Self-Attention

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Linearly project the queries, keys and values h times with different, learned linear projections

$$\text{MultiHead}(X,X,X) = \text{Concat}(\text{head}_1, \dots, \text{head}_\text{h})W^O$$

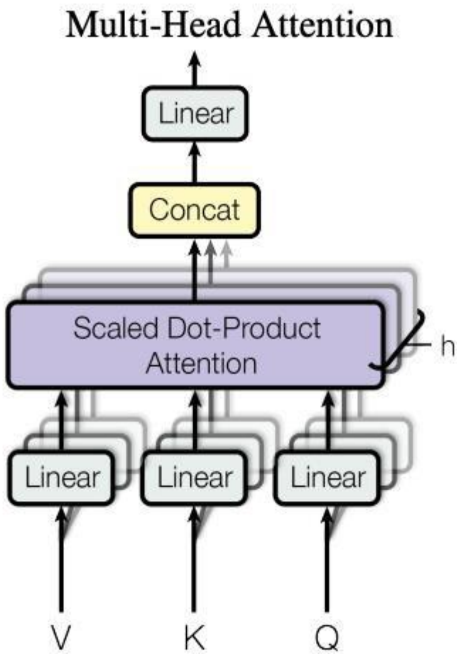
- Where,

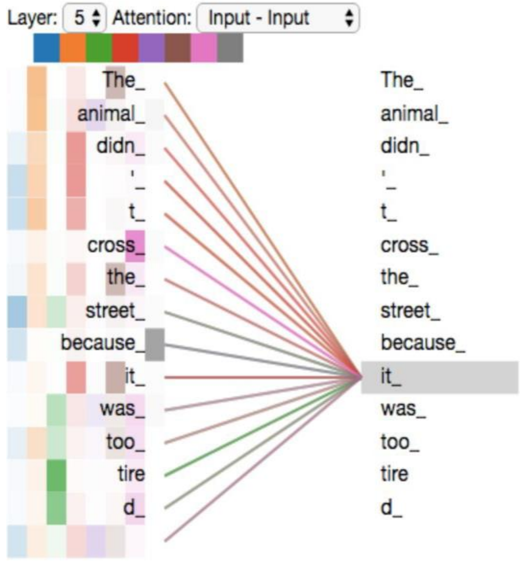
$$\text{head}_\text{i} = \text{Attention}\left(XW_i^Q, XW_i^K, XW_i^V\right)$$

$$= \text{softmax}\left(\left(X\mathrm{W}_\text{i}^\text{Q}\right)\left(X\mathrm{W}_\text{i}^\text{K}\right)^T\right)\left(X\mathrm{W}_\text{i}^\text{V}\right)\mathrm{W}_\text{i}^\text{O}$$

$$W_i^K \in R^{d_m \times d_k}, d_k = \frac{d_m}{h}$$

- It expands the model's ability to focus on different positions
- It gives the attention layer multiple representation subspaces





Multi-head Self-Attention

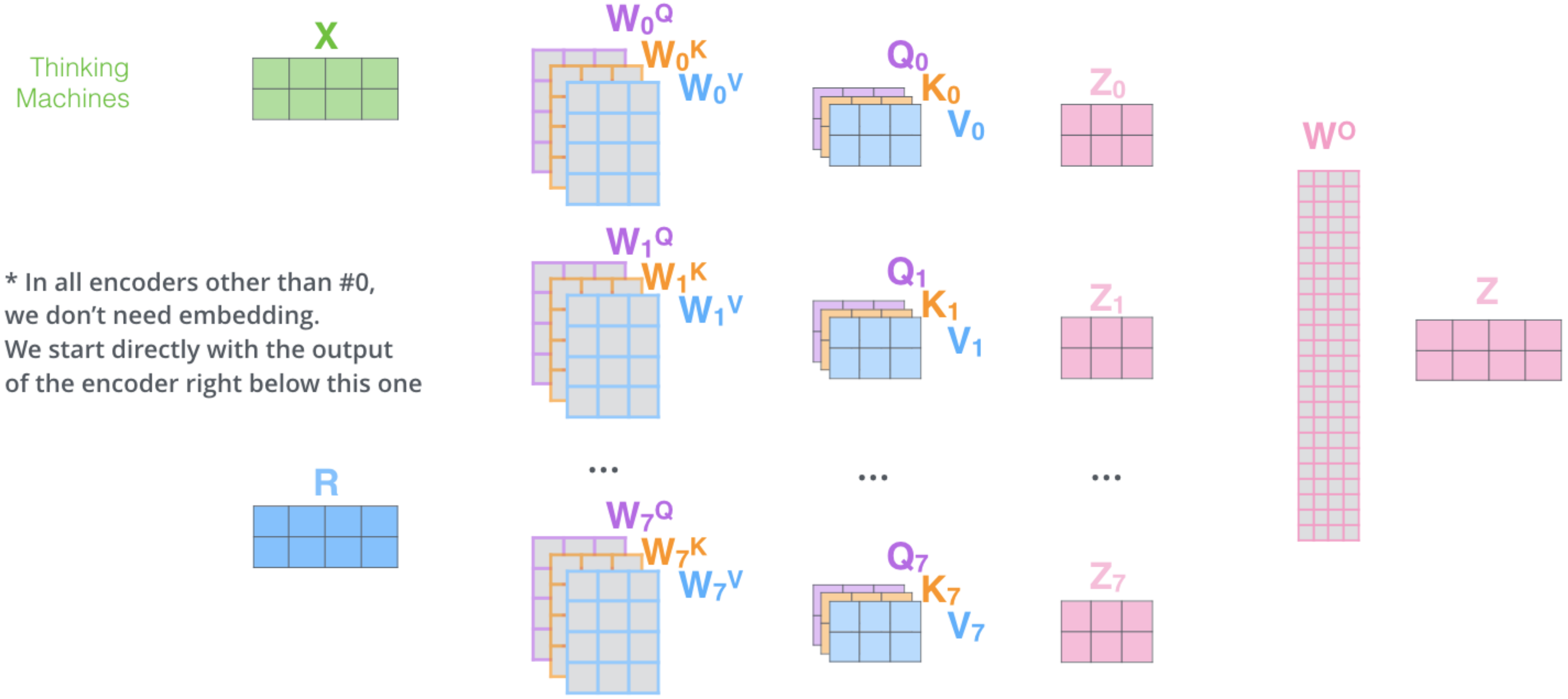
1) This is our input sentence*

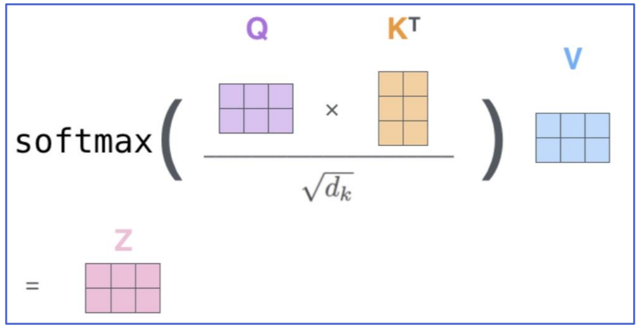
2) We embed each word*

3) Split into 8 heads. We multiply X or R with weight matrices

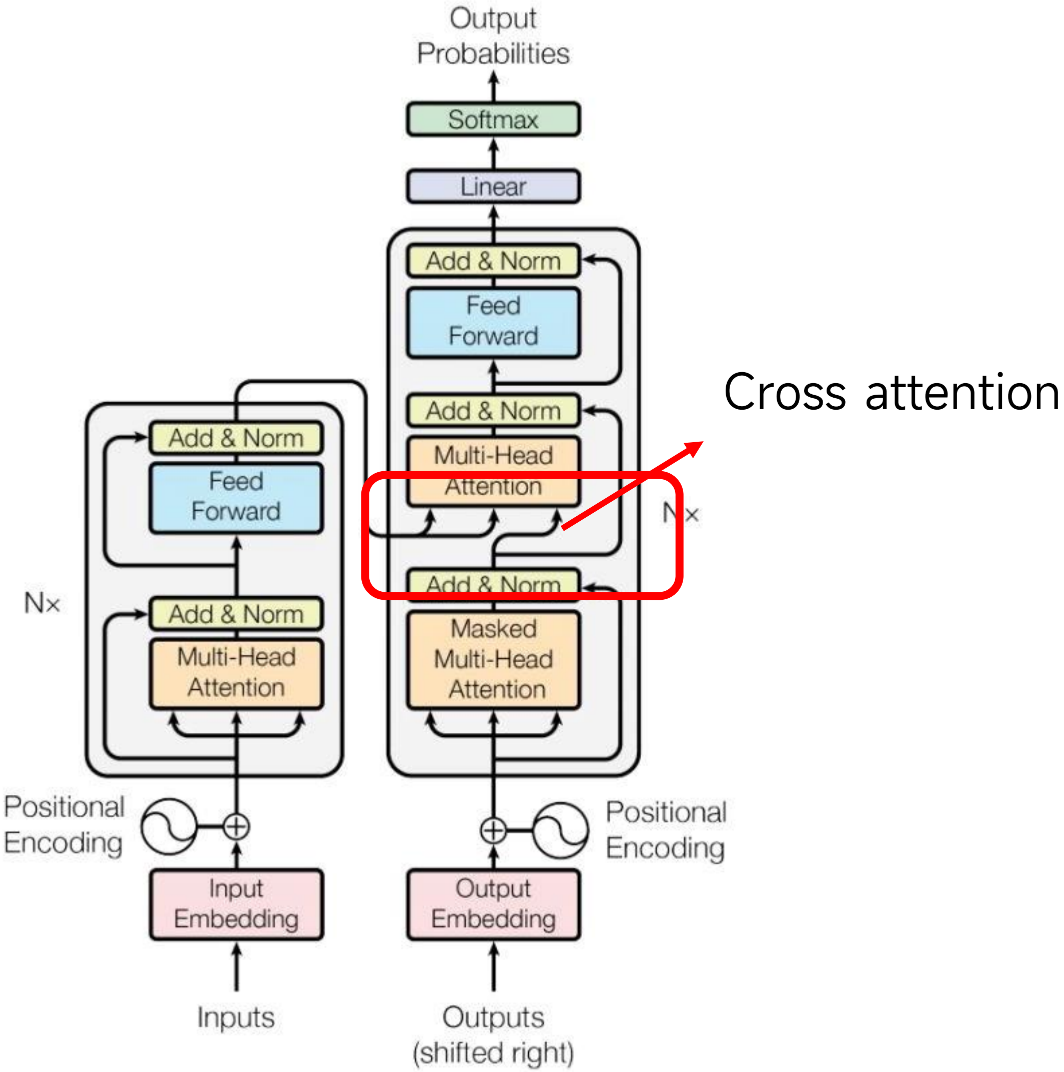
4) Calculate attention using the resulting Q/K/V matrices

5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer





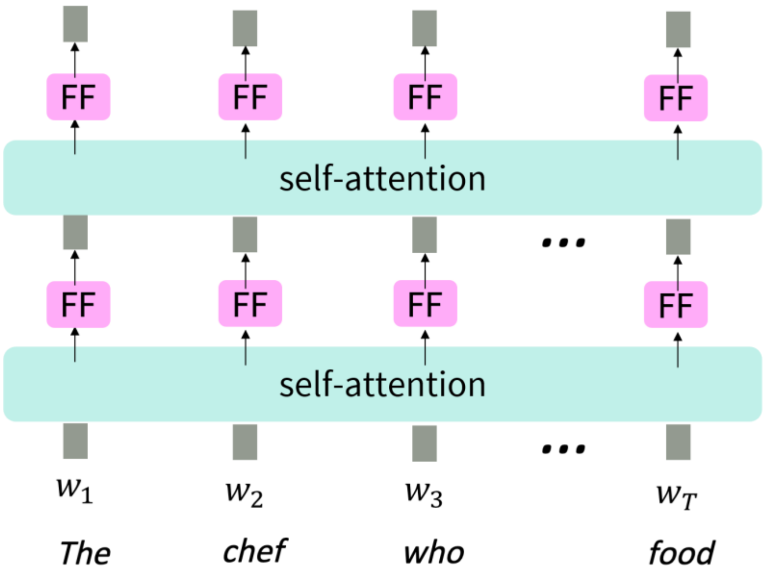
Cross-Attention



- The key/value vectors are from encoder output
- The query vectors are from last block in the decoder

Feed Forward Layer

- Apply a feedforward layer to the output of attention, providing non-linear activation (and additional expressive power)
- Feature expansion and compression ($4 \cdot d_m$)
- Store knowledge and act as a memory
- Consists of two linear layers [link]



$$\mathrm{FFN}(\mathbf{x}_i) = \mathrm{ReLU}(\mathbf{x}_i \mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$$

$$\mathbf{W}_1 \in \mathbb{R}^{d \times d_{ff}}, \mathbf{b}_1 \in \mathbb{R}^{d_{ff}}$$

$$\mathbf{W}_2 \in \mathbb{R}^{d_{ff} \times d}, \mathbf{b}_2 \in \mathbb{R}^{d}$$

In practice, they use $d_{ff} = 4d$

[2012.14913] Transformer Feed-Forward Layers Are Key-Value Memories

Layer Normalization

- Estimates the normalization statistics from the inputs within a hidden layer
- Reduce uninformative variation by normalizing to zero mean and standard deviation of one within each layer
- All the hidden units in a layer share the same trainable parameter γ, β

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} \gamma + \beta$$

- Each token share the same normalization term $\mathrm{E}[x]$, $\mathrm{Var}[x]$

```python
# features: (bsz, max_len, hidden_dim)

class LayerNorm(nn.Module):
    def __init__(self, features, eps=1e-6):
        super(LayerNorm, self).__init__()
        self.a_2 = nn.Parameter(torch.ones(features))
        self.b_2 = nn.Parameter(torch.zeros(features))
        self.eps = eps

    def forward(self, x):
        mean = x.mean(-1, keepdim=True) # mean: [bsz,
max_len, 1]
        std = x.std(-1, keepdim=True) # std: [bsz,
max_len, 1]
        return self.a_2 * (x - mean) / (std + self.eps)
+ self.b_2
```

Layer Normalization

- Why layer normalization for Transformer?
  - Stabilize training by reducing the network sensitivity to the scale of input features, leading to faster convergence and improved performance
  - Address internal covariate shift problem, a phenomenon where the distribution of inputs to a given layer changes during training as the model parameters are updated
  - Enhance gradient flow and address challenges like exploding or vanishing gradients

[Understanding Layer Normalization - by Daniel Kleine]

Layer Normalization

- Layer normalization is [different] in the context of CNN and Transformer

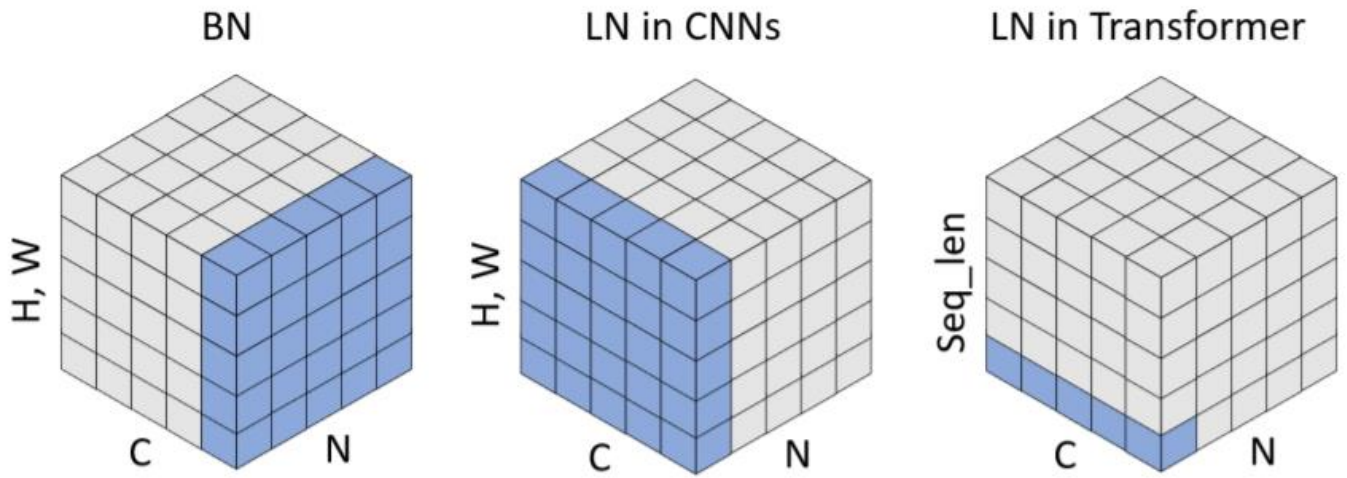
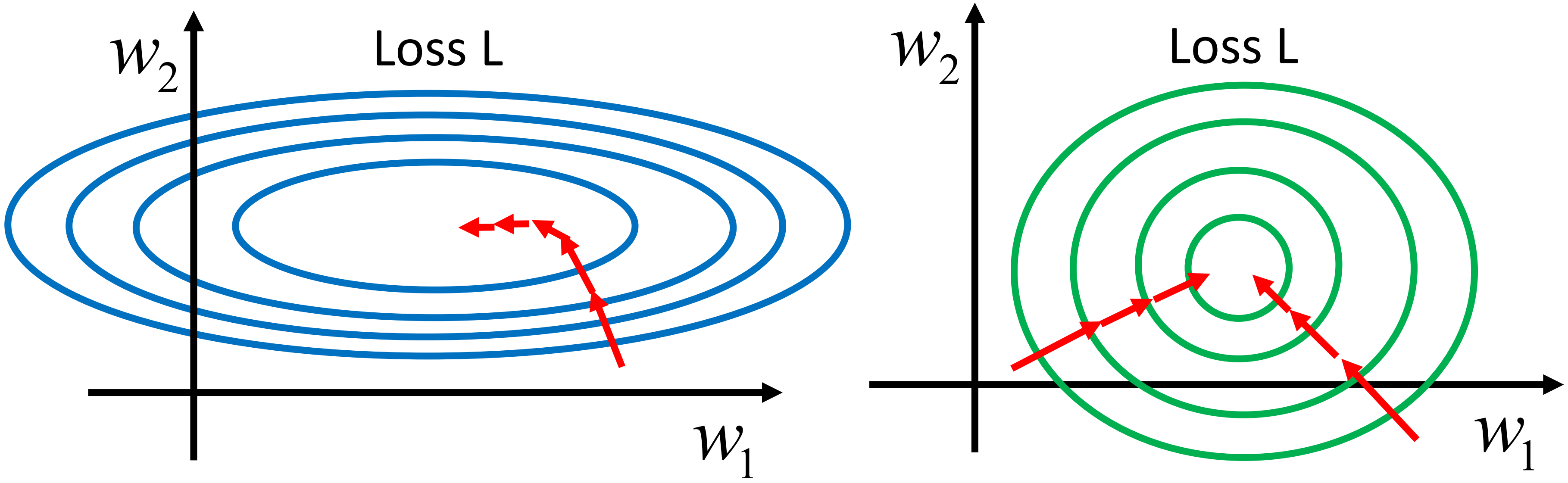


Figure 1: An illustration of **normalization methods**. Each subplot represents a feature map tensor, with B as the batch axis, C as the channel axis, and (H, W) or Seq_len as the spatial axes. The elements in blue are normalized by the same mean and variance, computed by aggregating the values of these elements.

Layer Normalization

- Improves convergence stability and sometimes even quality.



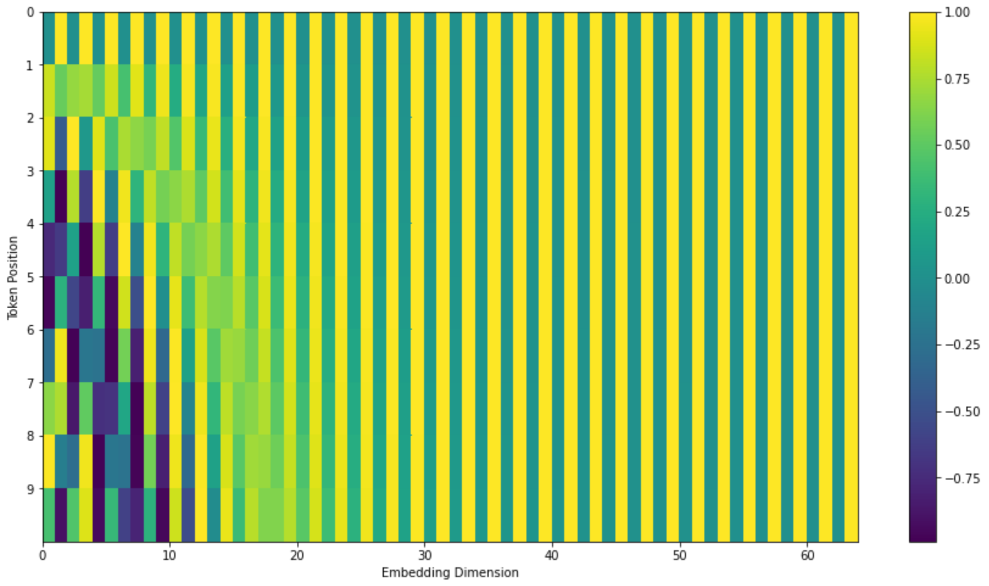
Positional Embedding

- Help the model determine the position of each word, or the distance between different words in the sequence

$$\mathrm{PE}(\mathrm{pos}, 2i) = \sin\left(\frac{\mathrm{pos}}{10000^{\frac{2i}{d_m}}}\right)$$

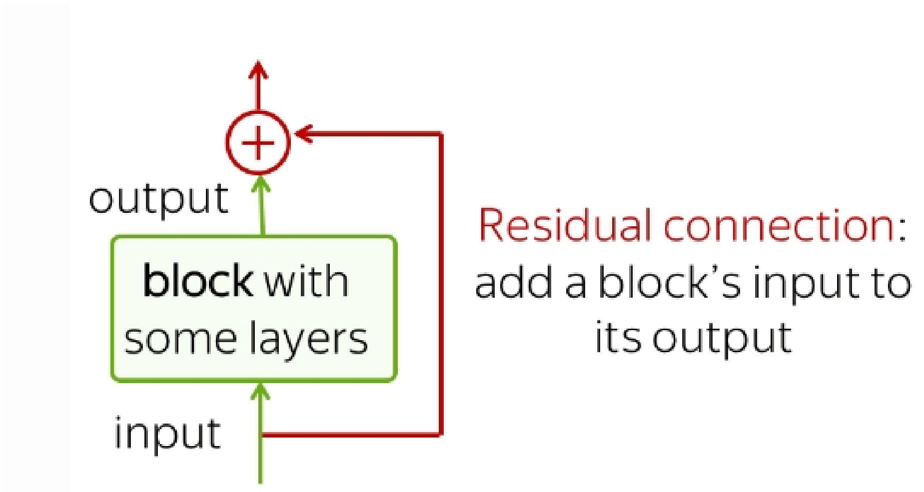
$$\mathrm{PE}(\mathrm{pos}, 2i+1) = \cos\left(\frac{\mathrm{pos}}{10000^{\frac{2i}{d_m}}}\right)$$

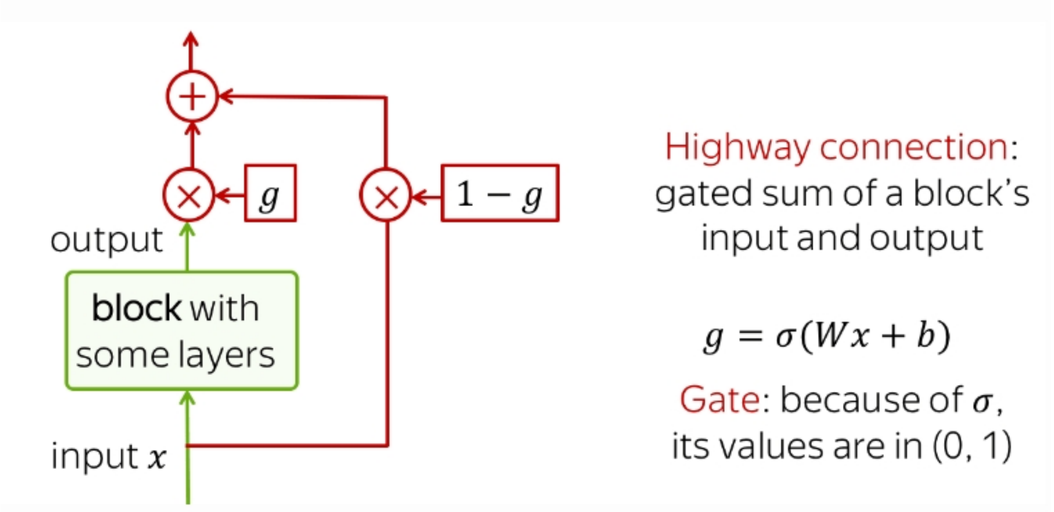
- pos is the position, i is the dimension, d_m is the model hidden state dimension
- Relative position embedding [ALiBi, RoPE]
  - In future class



Residual Connection

- Residual connection $x^l = \text{block}(x^{l-1}) + x^{l-1}$
- Used after each attention and FFN block
- Mitigate the gradient vanishing problem
- Ease the gradient flow through a network and allow stacking a lot of layers.





[1505.00387] Highway Networks (arxiv.org)

Dropout

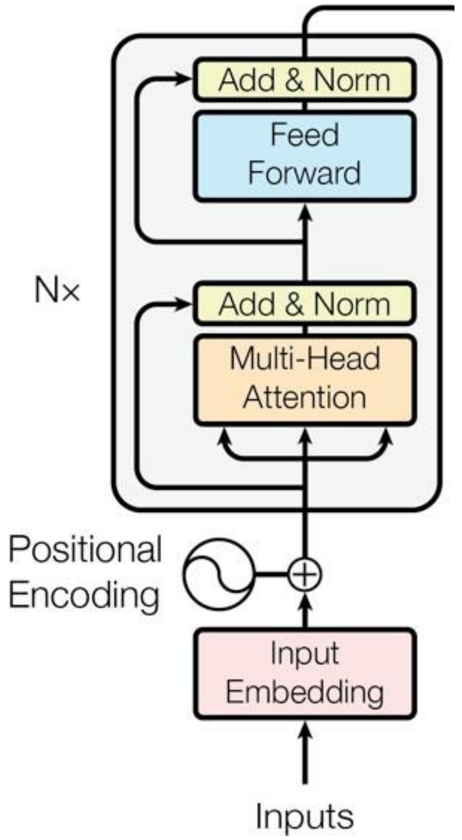
- During training, randomly zeroes some of the elements of the input tensor with probability p using samples from a Bernoulli distribution
- Compare model.train(), vs. model.eval(), vs. torch.no_grads()

```python
def forward(self, hidden_states: torch.Tensor, input_tensor: torch.Tensor)
    hidden_states = self.dense(hidden_states)
    hidden_states = self.dropout(hidden_states)
    hidden_states = self.LayerNorm(hidden_states + input_tensor)
    return hidden_states
```

[Illustrated Guide to Transformers Neural Network]

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929-1958.

Transformer Encoder



From the bottom to the top:

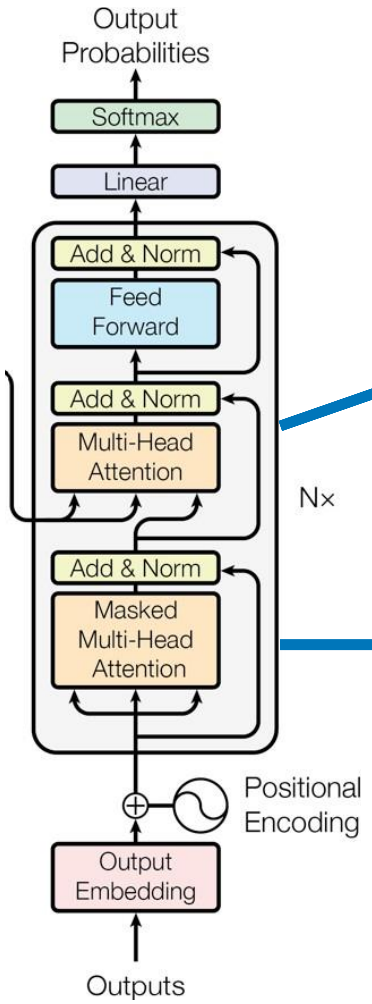
- Input embedding
- Positional encoding
- A stack of Transformer encoder layers

Transformer encoder is a stack of *N* layers, which consists of two sub-layers:

- Multi-head attention layer
- Feed-forward layer

$$\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^{d_1} \longrightarrow \mathbf{h}_1, \ldots, \mathbf{h}_n \in \mathbb{R}^{d_2}$$

Transformer Decoder



Cross-attention between source and target sequence

Self-attention within target sequence

From the bottom to the top:
- Output embedding
- Positional encoding
- A stack of Transformer decoder layers
- Linear + softmax

Transformer decoder is a stack of *N* layers, which consists of three sub-layers:
- Masked multi-head attention
- Multi-head cross-attention
- Feed-forward layer
- (W/ Add & Norm between sub-layers)

Count the Model Parameters

SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Suppose hidden state h, layer number L, head number n_h,
- Total model parameters = embedding params. + layer params. × layer numbers
- Embedding layer
  - Token embedding $|V| \cdot d$
  - Position embedding (learned or not): $l_c \cdot d$
    - l_c: Max. sentence length

Use code:

```
sum(p.numel() for p in model.parameters())
```

Count the Model Parameters

- Suppose hidden state h, encoder/decoder layer number L, head number n_h, then head hidden state $h_d = \frac{h}{n_h}$
  - Self-attention / cross-attention
    - For all heads, W_q, W_k, W_v, W_o each has $d \cdot n_h \cdot h_d = d^2$ parameters
  - Feedforward sublayer
    - Two matrices of $d \cdot (d \cdot 4)$ has $8d^2$ params. in total
  - Layer Norm $2 \cdot (d + d) = 4d$ (each encoder layer has two LNs)
- For each encoder layer, $4d^2 + 8d^2 + 4d = 12d^2 + 4d$ params.
- For each decoder layer, $4d^2 + 4d^2 + 8d^2 + 6d = 16d^2 + 6d$ params.
- Total parameters count is $(|V| + l_c)d + (28d^2 + 10d) \cdot L$.
  - When tie the input and output embeddings

Transformer Deep Dive: Parameter Counting

Count the Model Parameters

Example: suppose a model has 61M parameters

- If all weights are stored with 32-bit numbers, total storage will be about
  - 61M × 4 Bytes (32 bits) = 232.7 MB (244×10^6 Bytes)
- If all weights are stored with 8-bit numbers, total storage will be about
  - 61M × 1 Byte (8 bits) = 58.2 MB

- Bit (比特/位) vs. bytes (字节), 1B = 8b
- MB vs. MiB → 1MiB = 1024KB
- Usually, MB = MiB
- WLAN 1000M → 1000 Mbps (megabits per second)
- Hard drive 1MB = 1000 KB

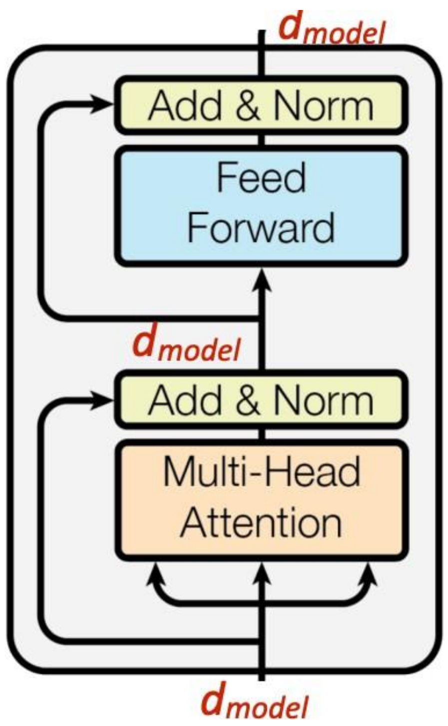
Transformer Architecture Specifications

	N	d_{model}	d_{ff}	h	d_k	d_v
base	6	512	2048	8	64	64

- From Vaswani et al.

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}
GPT-3 Small	125M	12	768	12	64
GPT-3 Medium	350M	24	1024	16	64
GPT-3 Large	760M	24	1536	16	96
GPT-3 XL	1.3B	24	2048	24	128
GPT-3 2.7B	2.7B	32	2560	32	80
GPT-3 6.7B	6.7B	32	4096	32	128
GPT-3 13B	13.0B	40	5140	40	128
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128

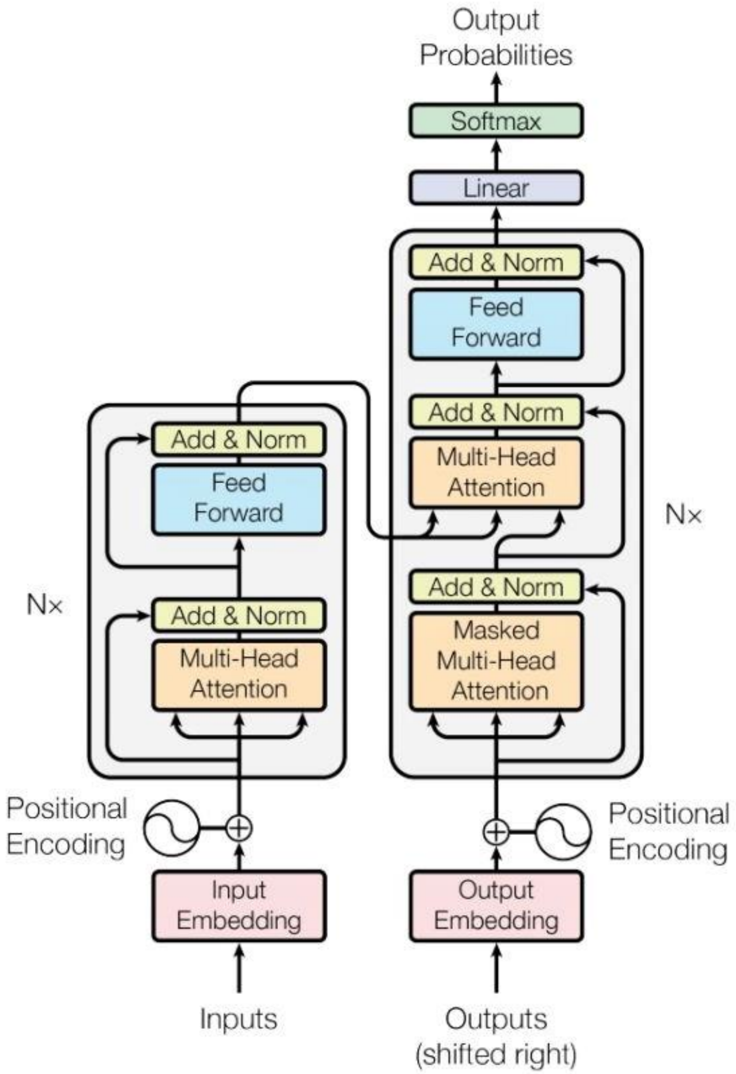
- From GPT-3; d_{head} is our d_k



Further Reading

- [The Illustrated Transformer]
- The Annotated Transformer
- Transformer论文逐段精读 –李沐
- Transformer models: an introduction and catalog
- Formal Algorithms for Transformers [arxiv]
- Transformers from scratch | peterbloem.nl
- LLM Visualization
- Transformer Explainer: LLM Transformer Model Visually Explained

- huggingface/transformers
- karpathy/nanoGPT



Configure the Coding Environment

- Windows, Linux, Mac OS
- Environment management
    - Conda [Documentation]
    - uv [Documentation]
- Command line tool
    - For windows, use git
- Coding IDE
    - VSCode, PyCharm
    - Remote SSH development
    - Plugins (e.g., github Copilot, Jupyter Notebook)

```
conda create -n sta python=3.11.*  # create a conda env named 'sta'
conda activate sta  # activate the sta env and enter in
pip install torch transformers pdbpp  # when you are in the 'sta'
env, install python packages using pip
conda deactivate sta  # deactivate and exit from the env
```

配置python开发环境-参考1　参考2

Coding Tools

- Linux and shell scripts
  - User SSH to connect a remote server

```
mkdir -p ~/data
cd ~/data
wget -c https://dl.fbaipublicfiles.com/flores101/dataset/flores101_dataset.tar.gz -O mydata.tar.gz
tar -zxf mydata.tar.gz

for fname in flores101_dataset/dev/*; do
    lang=$(basename $fname '.dev')
    echo $lang
    mkdir -p results/$lang
    head -n 10 flores101_dataset/dev/$lang.dev > results/${lang}/${lang}-copy.txt
done
```

Coding Tools

Schedule

- **1/13/20**: Course overview + the shell
- **1/14/20**: Shell Tools and Scripting
- **1/15/20**: Editors (Vim)
- **1/16/20**: Data Wrangling
- **1/21/20**: Command-line Environment
- **1/22/20**: Version Control (Git)
- **1/23/20**: Debugging and Profiling
- **1/27/20**: Metaprogramming
- **1/28/20**: Security and Cryptography
- **1/29/20**: Potpourri
- **1/30/20**: Q&A

The Missing Semester of Your CS Education

- Python debug
  - Read the error message
  - Read the official documentation
  - Read the code/package examples
  - Read tutorials
  - Narrow down the bug by toy samples
  - Learn the code style [Google Python Style Guide]
- Play with the python code

Example of LLM Training

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

```python
from datasets import load_dataset
from trl import SFTConfig, SFTTrainer
import torch


# Set device
device = "cuda" if torch.cuda.is_available() else "cpu"


# Load dataset
dataset = load_dataset("HuggingFaceTB/smoltalk", "all")


# Configure model and tokenizer
model_name = "HuggingFaceTB/SmolLM2-135M"
model =
AutoModelForCausalLM.from_pretrained(pretrained_model_name_o
r_path=model_name).to(
    device
)
tokenizer =
AutoTokenizer.from_pretrained(pretrained_model_name_or_path=
model_name)
# Setup chat template
model, tokenizer = setup_chat_format(model=model,
tokenizer=tokenizer)
```

```python
# Configure trainer
training_args = SFTConfig(
    output_dir="./sft_output",
    max_steps=1000,
    per_device_train_batch_size=4,
    learning_rate=5e-5,
    logging_steps=10,
    save_steps=100,
    eval_strategy="steps",
    eval_steps=50,
)


# Initialize trainer
trainer = SFTTrainer(
    model=model,
    args=training_args,
    train_dataset=dataset["train"],
    eval_dataset=dataset["test"],
    processing_class=tokenizer,
)


# Start training
trainer.train()
```

[Supervised Fine-Tuning - Hugging Face LLM Course]

More Examples

- karpathy/nanoGPT.
    - Config/
    - Model.py
    - Train.py
    - Sample.py
    - Bench.py

```python
# training loop
X, Y = get_batch('train') # fetch the very first batch
t0 = time.time()
local_iter_num = 0 # number of iterations in the lifetime of this process
raw_model = model.module if ddp else model # unwrap DDP container if needed
running_mfu = -1.0
while True:

    # determine and set the learning rate for this iteration
    lr = get_lr(iter_num) if decay_lr else learning_rate
    for param_group in optimizer.param_groups:
        param_group['lr'] = lr

    # evaluate the loss on train/val sets and write checkpoints
    if iter_num % eval_interval == 0 and master_process:
        losses = estimate_loss()
        print(f"step {iter_num}: train loss {losses['train']:.4f}, val loss {losses['val']:.4f}")
```

Further Reading

- Python-For-Beginners Book_
- Pytorch tutorial
- [★] CS224N-pytorch tutorial
- [★] Deep Learning Tuning Playbook
- Bash-handbook github repo
- Bash scripting cheatsheet

- AutoDL帮助文档
- 文档中心 · 魔搭社区

Decoding Algorithm

- Generate target sentences

$$Y = \underset{(y_1, y_2, \ldots, y_L)}{\operatorname{argmax}} \prod_{i=1}^{L} p(y_i | y_{<i}, X; \theta)$$

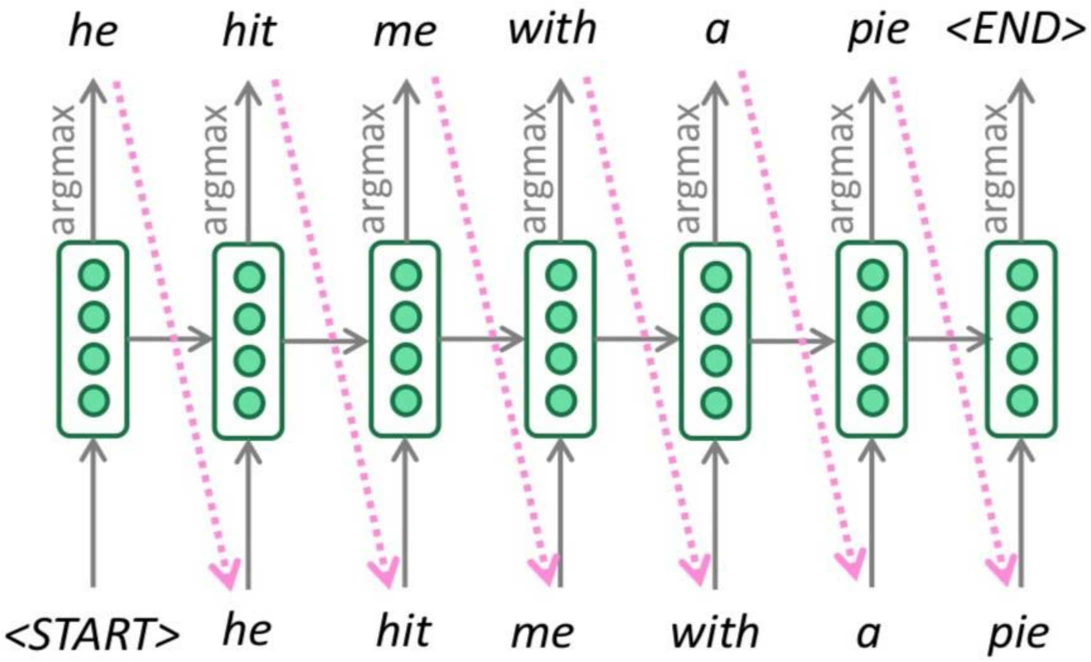
- Exhaustive search is very expensive

Greedy search	Beam search	Top-k sampling	Top-p sampling

Greedy Search

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

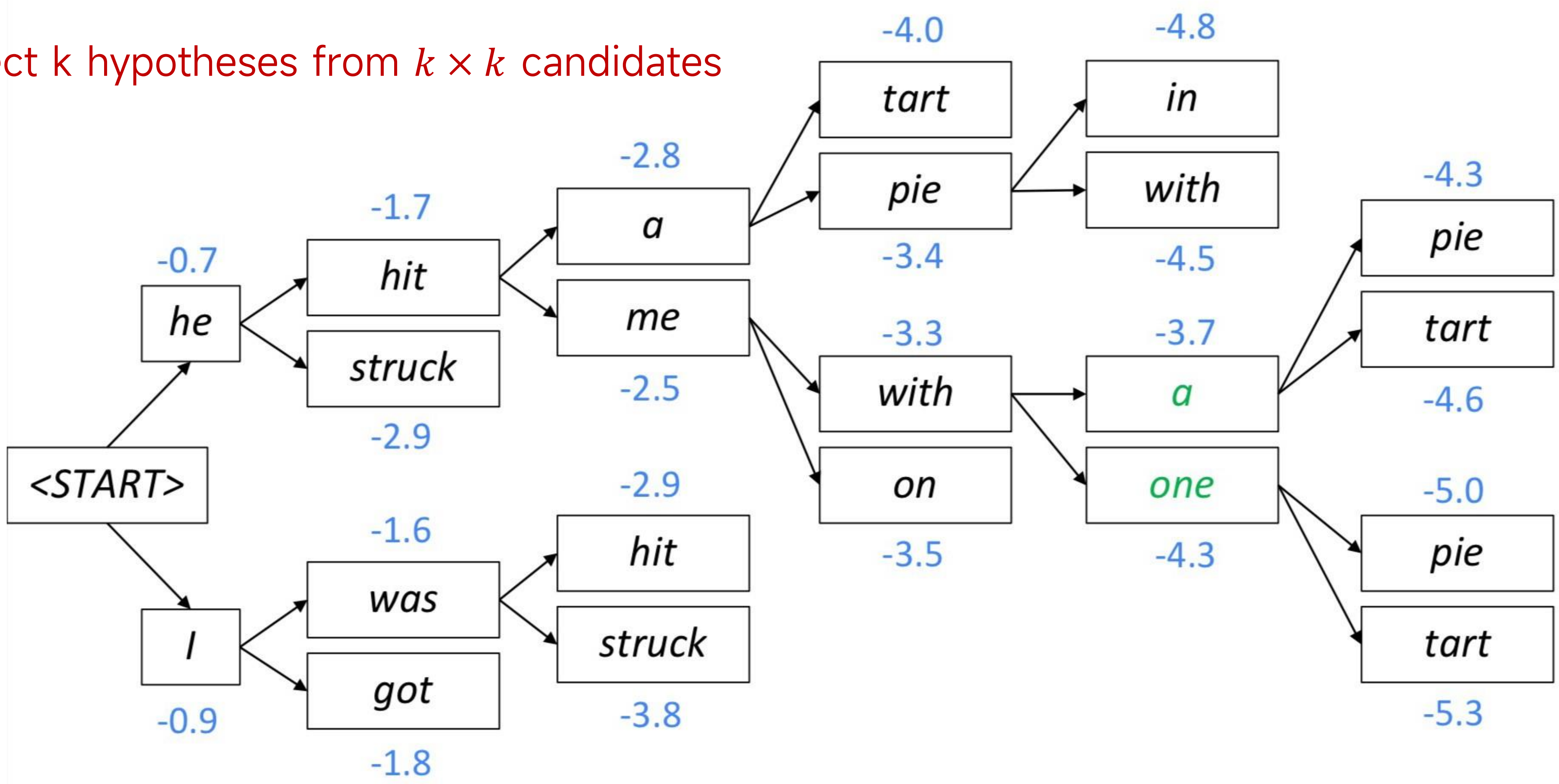
- Compute argmax at every step of decoder to generate word y_i
- Problems:
    - Will often generate the "easy" words first
    - Will prefer multiple common words to one rare word
    - May return a poor sentence that has low probabilities of words at the end.



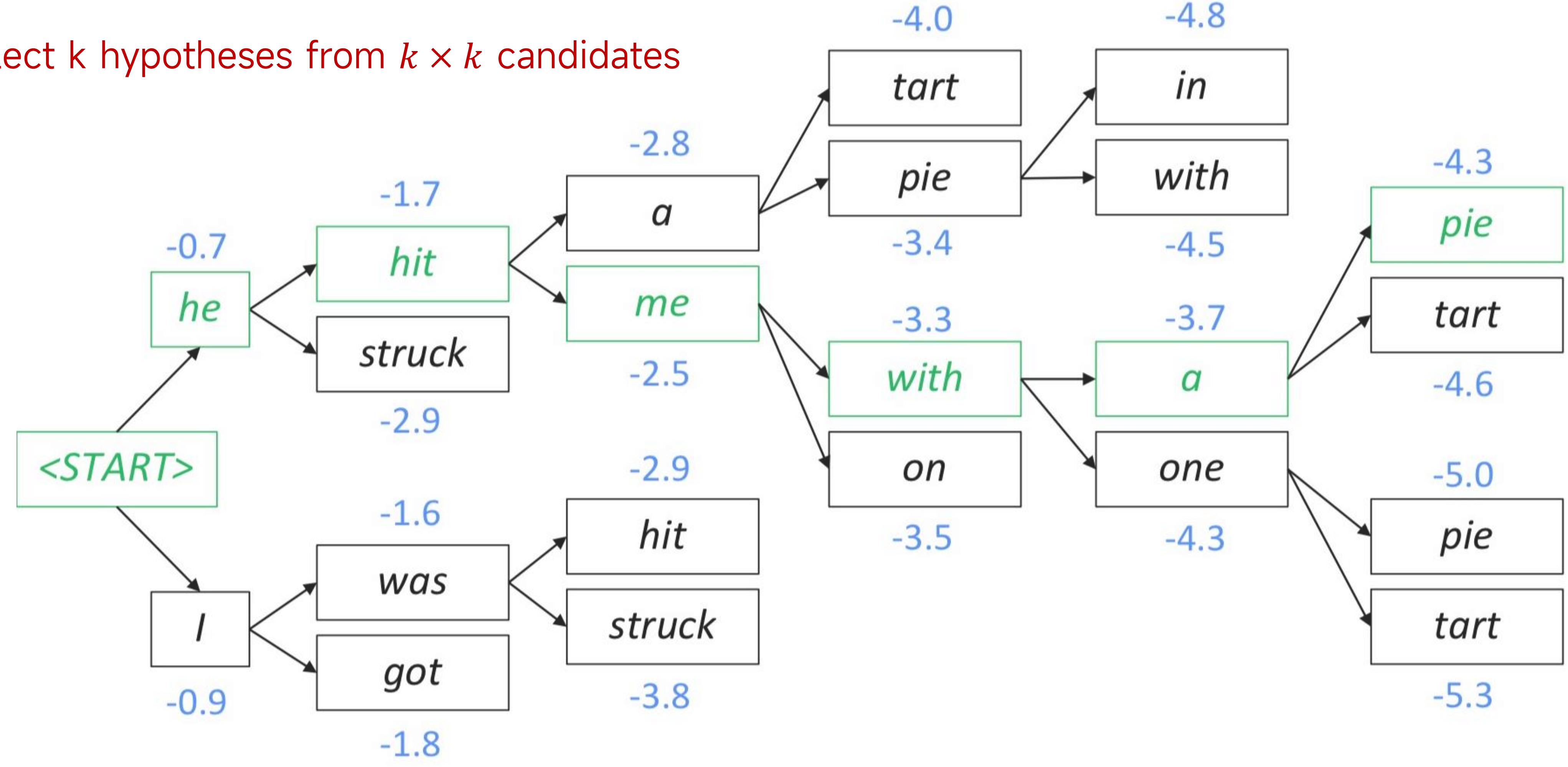
Beam Search

Beam size = k = 2. Blue numbers = $\text{score}(y_1,\ldots,y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i|y_1,\ldots,y_{i-1},x)$

Select k hypotheses from $k \times k$ candidates



Select k hypotheses from $k \times k$ candidates



Beam Search

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

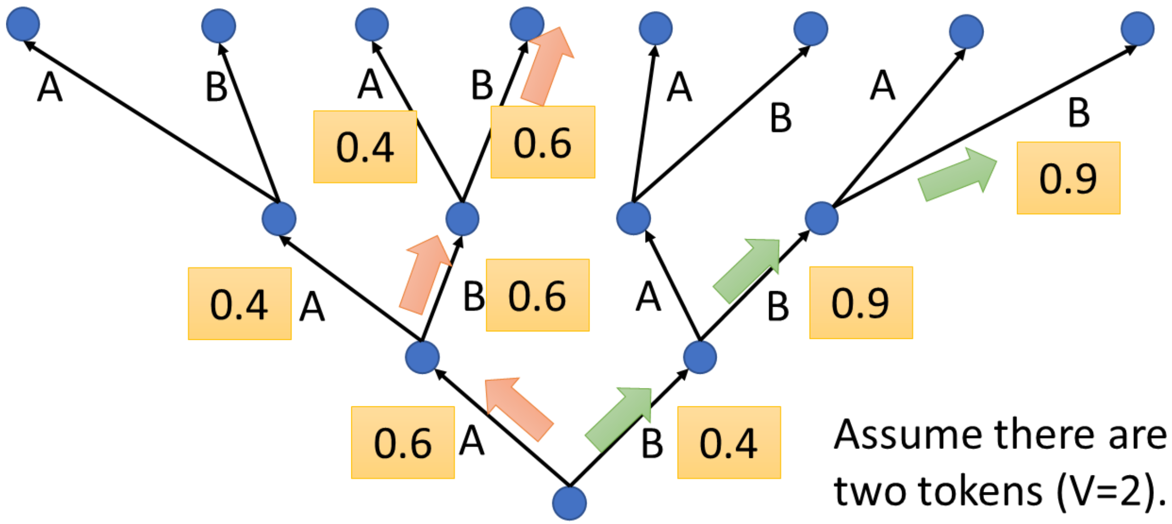
- Different hypotheses may produce <eos> token at different time steps
  - When a hypothesis produces <eos>, save it along with its score, stop expanding it and place it aside
  - Keep expanding the remaining best hypotheses
- Continue beam search until:
  - All k hypotheses produce <eos>, or
  - Hit max decoding length limit T
- Select top hypotheses using the normalized likelihood score

$$S_{norm} = \frac{1}{T}\sum_{i=1}^{t} \log p(y_i | y_{<i}, X; \theta)$$

  - Otherwise, shorter hypotheses have higher scores

Beam Search

- [Code for beam search]



Assume there are only two tokens (V=2).

The red path is greedy decoding.

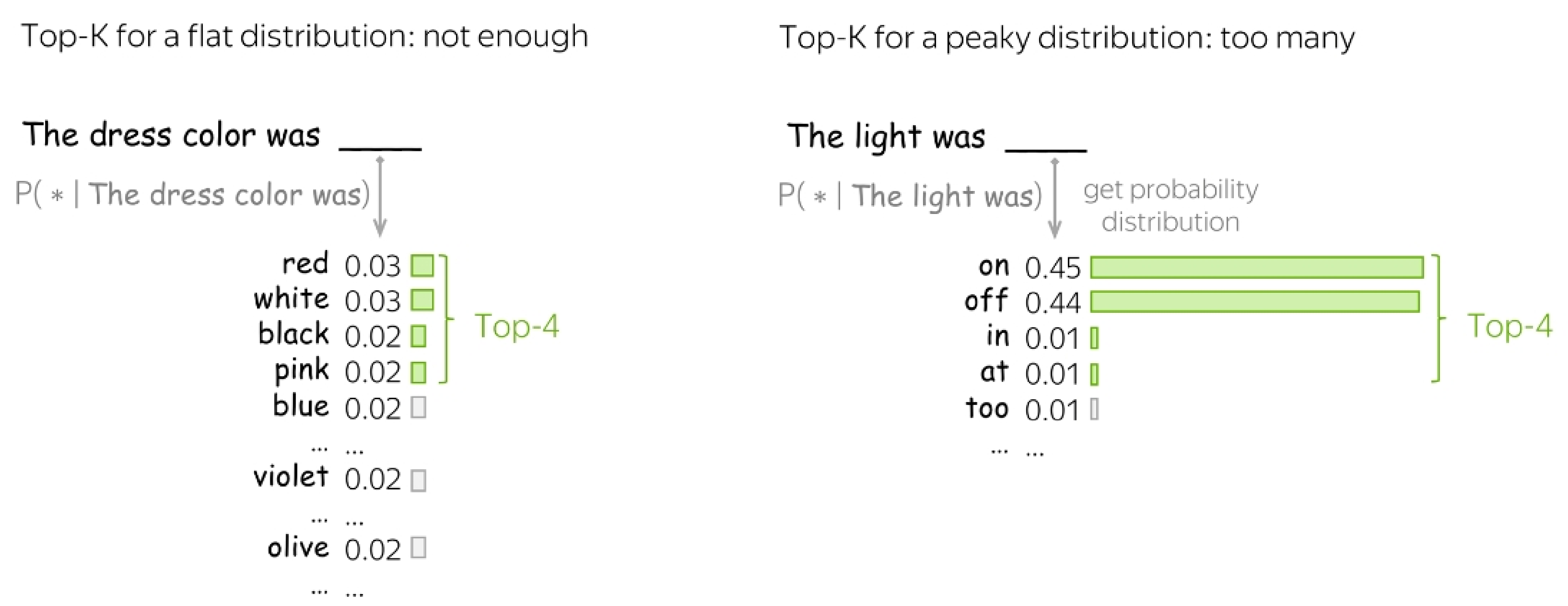
The green path is the best one.

[束搜索 — 动手学深度学习]

Top-k Sampling

- Always sample from top-K most likely tokens
  - Instead of selecting with scores in beam search
- When k=3, first select the top-3 tokens that have highest probabilities
- Renormalize the token distribution and sample the next token accordingly

```
# set top_k to 50
sample_output = model.generate(
    **model_inputs,
    max_new_tokens=40,
    do_sample=True,
    top_k=50
)
```



Top-k Sampling

- Renormalize the token distribution and sample the next token accordingly

```
# sample
probs = nn.functional.softmax(next_token_scores, dim=-1)
next_tokens = torch.multinomial(probs, num_samples=1).squeeze(1)
```

Code for sampling-based decoding algorithm, and top-K logits wrapper

Code of decoding algorithms in Transformers

Nucleus sampling

- Select the smallest number of top tokens such that their cumulative probability is at least p

We propose a new stochastic decoding method: Nucleus Sampling. The key idea is to use the shape of the probability distribution to determine the set of tokens to be sampled from. Given a distribution $P(x|x_{1:i-1})$, we define its top-p vocabulary $V^{(p)} \subset V$ as the smallest set such that

$$\sum_{x \in V^{(p)}} P(x|x_{1:i-1}) \geq p. \tag{2}$$

Top-p Sampling

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- The number of tokens we sample from is dynamic and depends on the properties of the distribution
- Renormalize the token distribution and sample the next token accordingly

```
sample_output = model.generate(
    **model_inputs,
    max_new_tokens=40,
    do_sample=True,
    top_p=0.92,
    top_k=0
)
```

Code for Top-p logits wrapper

Top-p and Top-k sampling code

How to generate text: using different decoding methods for language generation with Transformers

Temperature-based Sampling

- Define logits as $l = \{l_i\}_{i \in |V|}$, we use softmax function to get the probabilities

$$p(v_k) = \frac{e^{l_k/\tau}}{\sum_j e^{l_j/\tau}}$$

- Here, τ is the decoding temperature
  - A low temperature results a more deterministic response, keeps the model focused on the most likely response
  - A high temperature makes the model explore a wider range of responses, resulting in a more creative response

Code for temperature-based sampling

The Effect of Sampling Temperature on Problem Solving in Large Language Models - ACL Anthology

Thank you