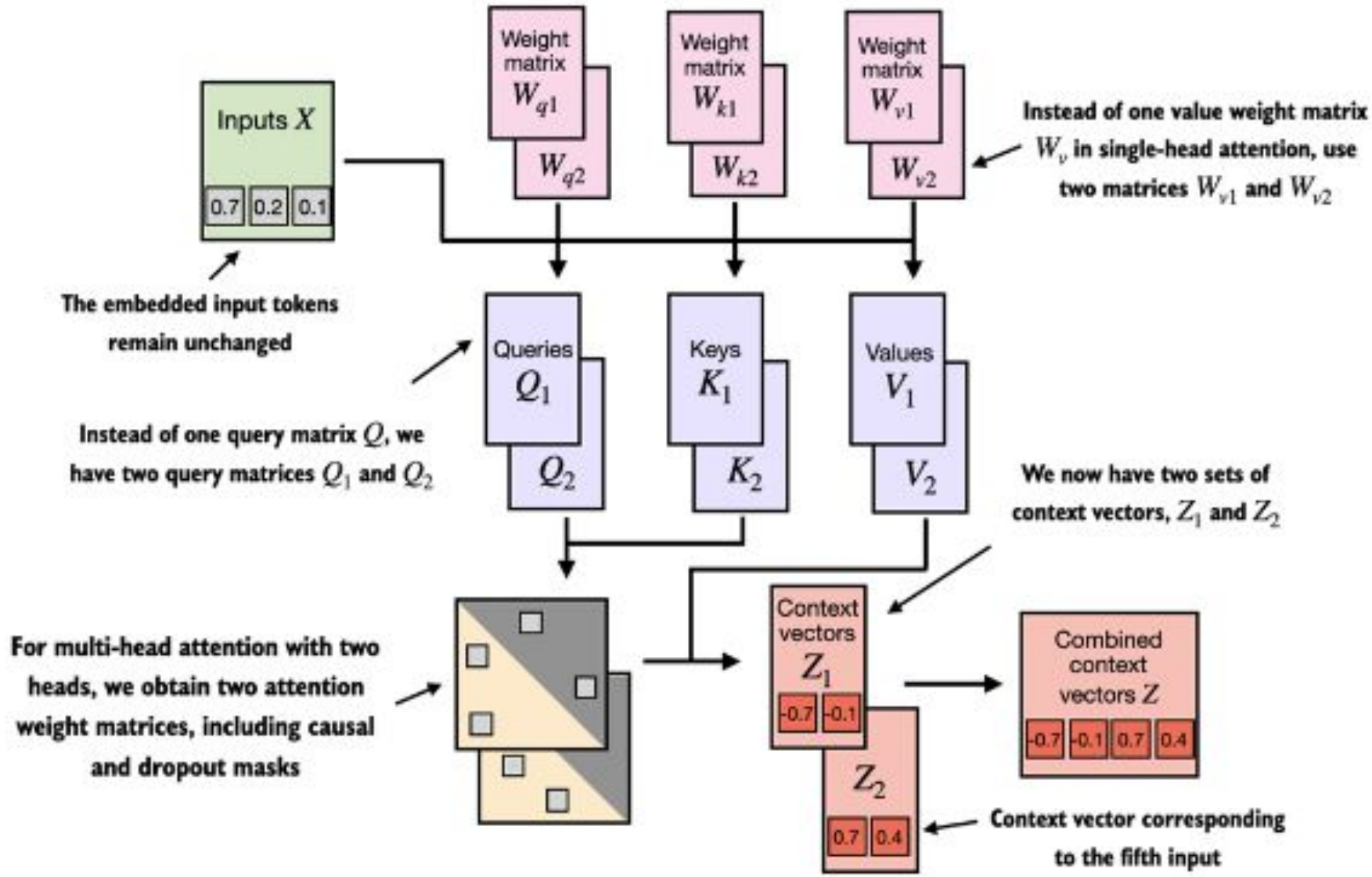
# Attention Mechanism Deep Dive

Naeemullah Khan
naeemullah.khan@kaust.edu.sa

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

LMH
Lady Margaret Hall

July 3, 2025


Inputs $X$
0.7
0.2
0.1
The embedded input tokens remain unchanged
Weight matrix $W_{q1}$
$W_{q2}$
Weight matrix $W_{k1}$
$W_{k2}$
Weight matrix $W_{v1}$
$W_{v2}$
Instead of one value weight matrix $W_v$ in single-head attention, use two matrices $W_{v1}$ and $W_{v2}$
Queries $Q_1$
$Q_2$
Keys $K_1$
$K_2$
Values $V_1$
$V_2$
Instead of one query matrix $Q$, we have two query matrices $Q_1$ and $Q_2$
We now have two sets of context vectors, $Z_1$ and $Z_2$
For multi-head attention with two heads, we obtain two attention weight matrices, including causal and dropout masks
Context vectors $Z_1$
-0.7
-0.1
$Z_2$
0.7
0.4
Combined context vectors $Z$
-0.7
-0.1
0.7
0.4
Context vector corresponding to the fifth input

# Table of Contents

# Motivation



- Sequence-to-sequence models struggle with **long-range dependencies**, making it difficult to capture context over lengthy inputs.
- Attention mechanisms address this by allowing the model to **selectively focus** on relevant parts of the input sequence during processing.
- They form the **backbone** of modern architectures such as **Transformer models** (e.g., BERT, GPT, ViT).
- Attention is crucial in various tasks including **translation, summarization, and image captioning**.

**Example:** In machine translation, rather than encoding an entire sentence into a single fixed-size vector, attention enables the model to dynamically focus on relevant source words when generating each target word.

# Learning Outcomes



After this session, you should be able to:

- Understand how **alignment functions** operate.
- Comprehend **QKV (Query, Key, Value)** mechanism in attention.
- Learn how attention is used in **decoding sequences**.
- Recognize the **strengths and limitations** of attention mechanisms.
- Get familiar with **modern attention research**.

# Alignment Function

- The alignment function is a crucial component of attention mechanisms, determining how much focus to place on different parts of the input sequence.
- It computes a score for each input token relative to the current output token being generated.
- The scores are then normalized (often using softmax) to create a probability distribution, indicating the relative importance of each input token.
- Common alignment functions include:
  - **Dot Product:** Computes the dot product between the query and key vectors.
  - **Scaled Dot Product:** Similar to dot product but scaled by the square root of the dimension of the key vectors to prevent large values that can saturate the softmax function.
  - **Additive Attention:** Combines the query and key vectors using a feed-forward neural network to compute the alignment scores.
- The choice of alignment function can significantly impact the model's performance, especially in tasks with long sequences or complex dependencies.

**Example:** In a translation task, the alignment function helps the model determine which words in the source language should be emphasized when generating each word in the target language. For instance, when translating "The cat sat on the mat," the model might focus more on "cat" when generating "gato" in Spanish, while still considering "sat" and "on" to maintain the context.

# Types of Alignment Functions



- **Dot Product:**

$$\text{score} = q^\top k$$

- **Scaled Dot Product:**

$$\text{score} = \frac{q^\top k}{\sqrt{d_k}}$$

(used in Transformers to prevent softmax saturation)

- **Additive (Bahdanau) Attention:**

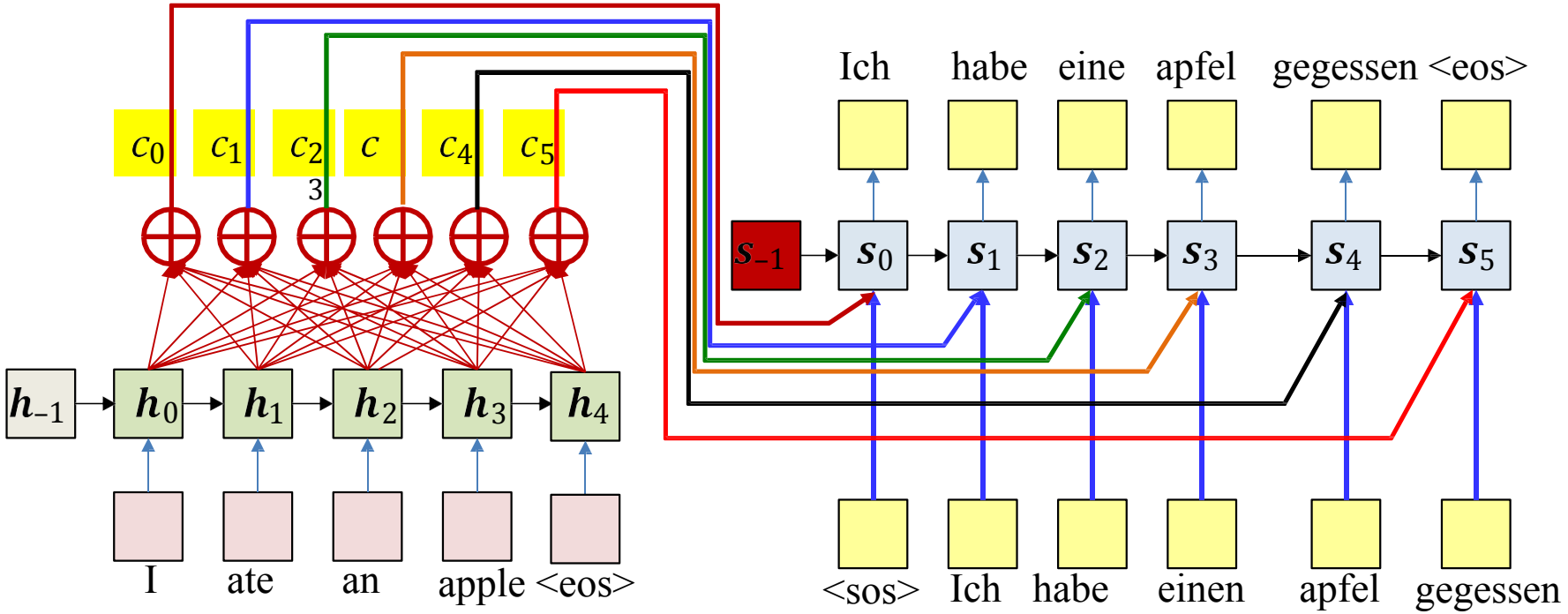
$$\text{score} = v^\top \tanh(W_1 h_i + W_2 s_t)$$

- **General:**

$$\text{score} = q^\top W k$$

**Each function balances efficiency vs expressiveness.**

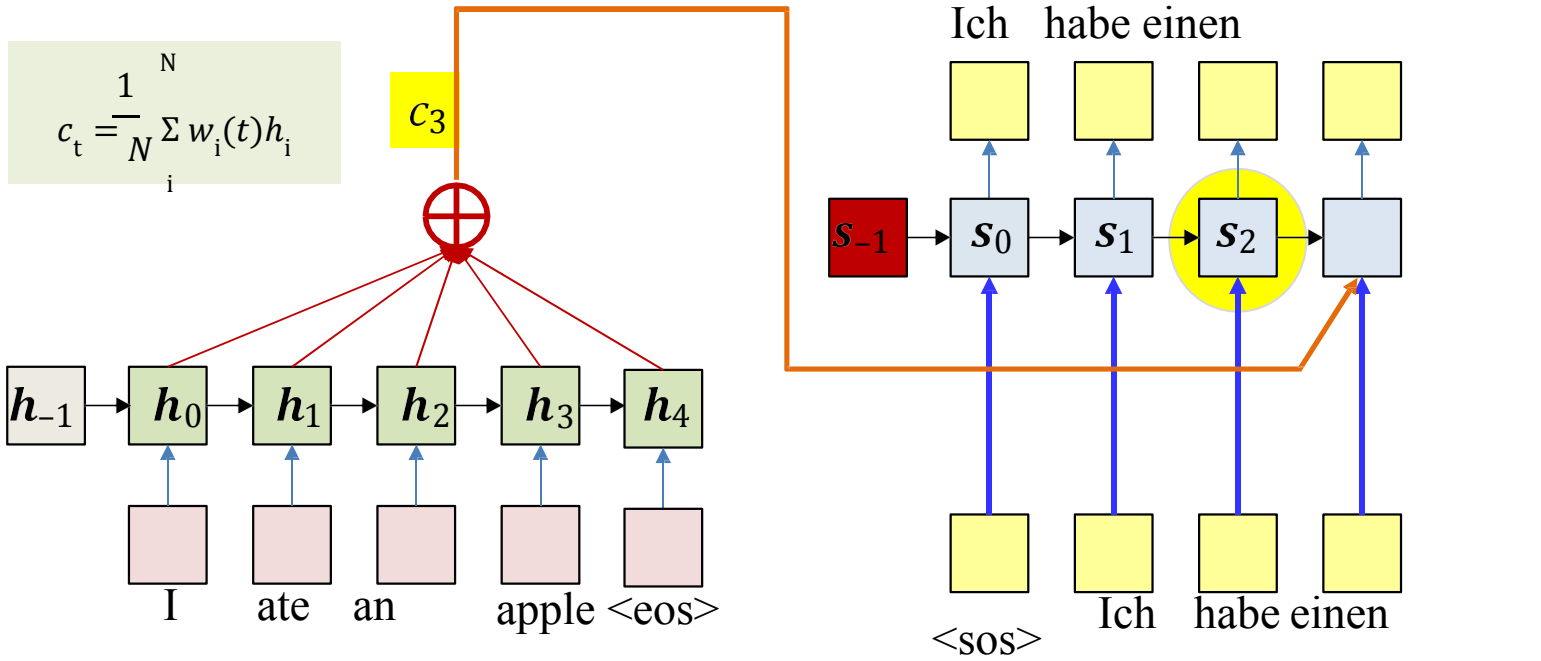
# Attention Models




$$c_t = \frac{1}{N}\sum_{i}^{N} w_i(t) h_i$$

- **Attention weights:** The weights $w_i(t)$ are dynamically computed as functions of decoder state
  - Expectation: if the model is well-trained, this will automatically "highlight" the correct input
- But how are these computed?

# Attention weights at time

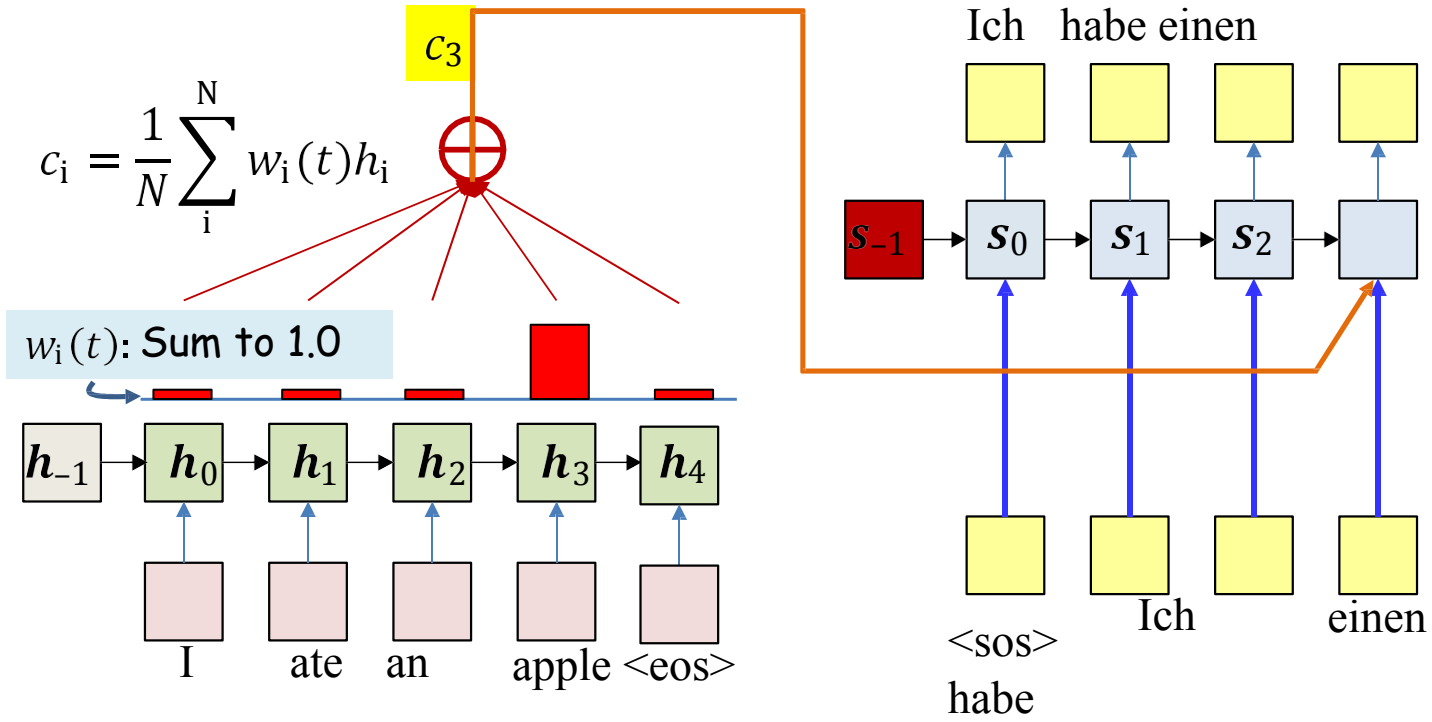



- The "attention" weights $w_i(t)$ at time $t$ must be computed from available information at time $t$

- The primary information is $s_{t-1}$ (the state at time time $t-1$)
  - Also, the input word at time $t$, but generally not used for simplicity

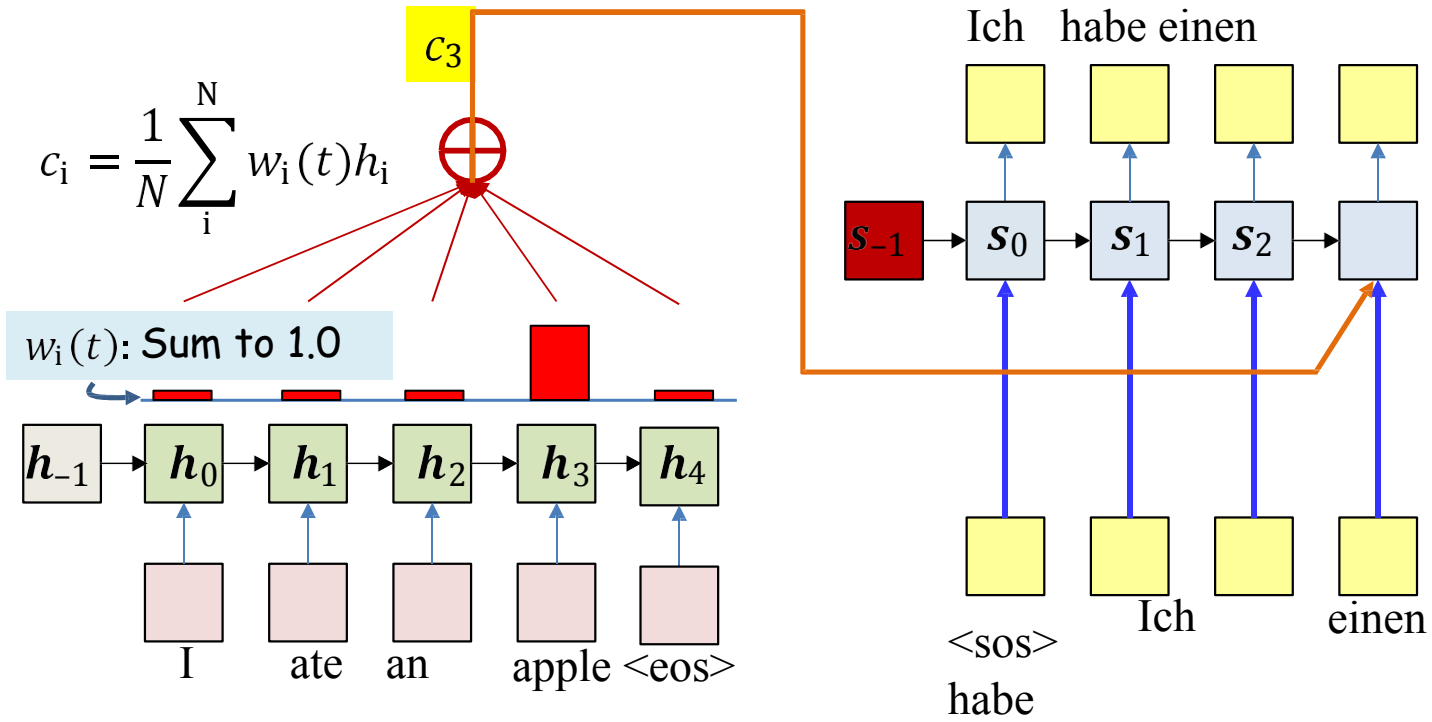
$$w_i(t) = a(\boldsymbol{h}_i, \boldsymbol{s}_{t-1})$$

# Requirement on attention weights




- The weights $w_i(t)$ must be positive and sum to 1.0
  - I.e. be a distribution
  - Ideally, they must be high for the most relevant inputs for the ith output and low elsewhere

# Requirement on attention weights




- The weights $w_i(t)$ must be positive and sum to 1.0
  - I.e. be a distribution
  - Ideally, they must be high for the most relevant inputs for the ith output and low elsewhere

- Solution: A two step weight computation
  - First compute *raw* weights (which could be +ve or –ve)
  - Then softmax them to convert them to a distribution

$$e_i(t) = g(h_t, s_{t-1})$$

$$w_i(t) = \frac{\exp(e_i(t))}{\sum_j \exp(e_j(t))}$$

# QUIZ



★ The attention framework computes a different “context” vector at each output step (T/F)

- ❖ True
- ❖ False

# QUIZ



★ The attention framework computes a different “context” vector at each output step (T/F)

- ❖ **True**
- ❖ False

# QUIZ



★ The context vector is chosen as the hidden (encoder) representation of the input word that is assigned the highest attention weight (T/F)

- ❖ True
- ❖ False

# QUIZ



★ The context vector is chosen as the hidden (encoder) representation of the input word that is assigned the highest attention weight (T/F)

- ❖ True
- ❖ **False**

# QUIZ



★ The attention weight to any input word is a function of the hidden encoder representation of the word and the most recent decoder state (T/F)

- ❖ True
- ❖ False

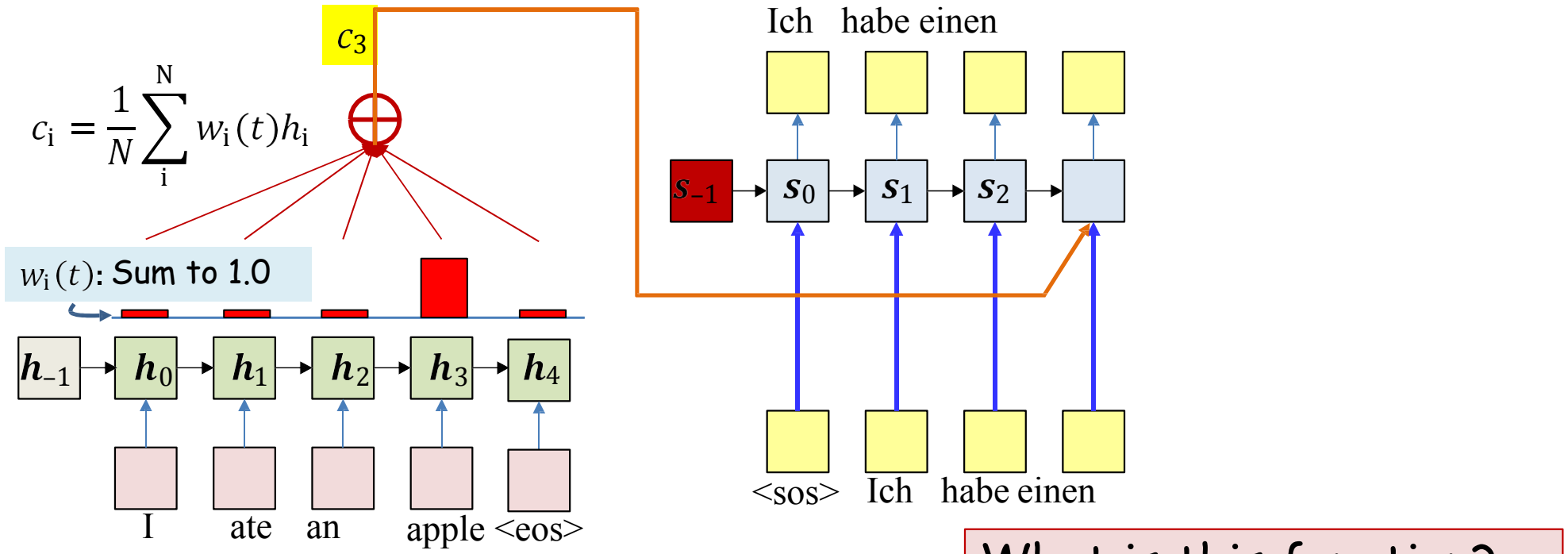
# QUIZ



★ The attention weight to any input word is a function of the hidden encoder representation of the word and the most recent decoder state (T/F)

- **True**
- False

# Requirement on attention weights




What is this function?

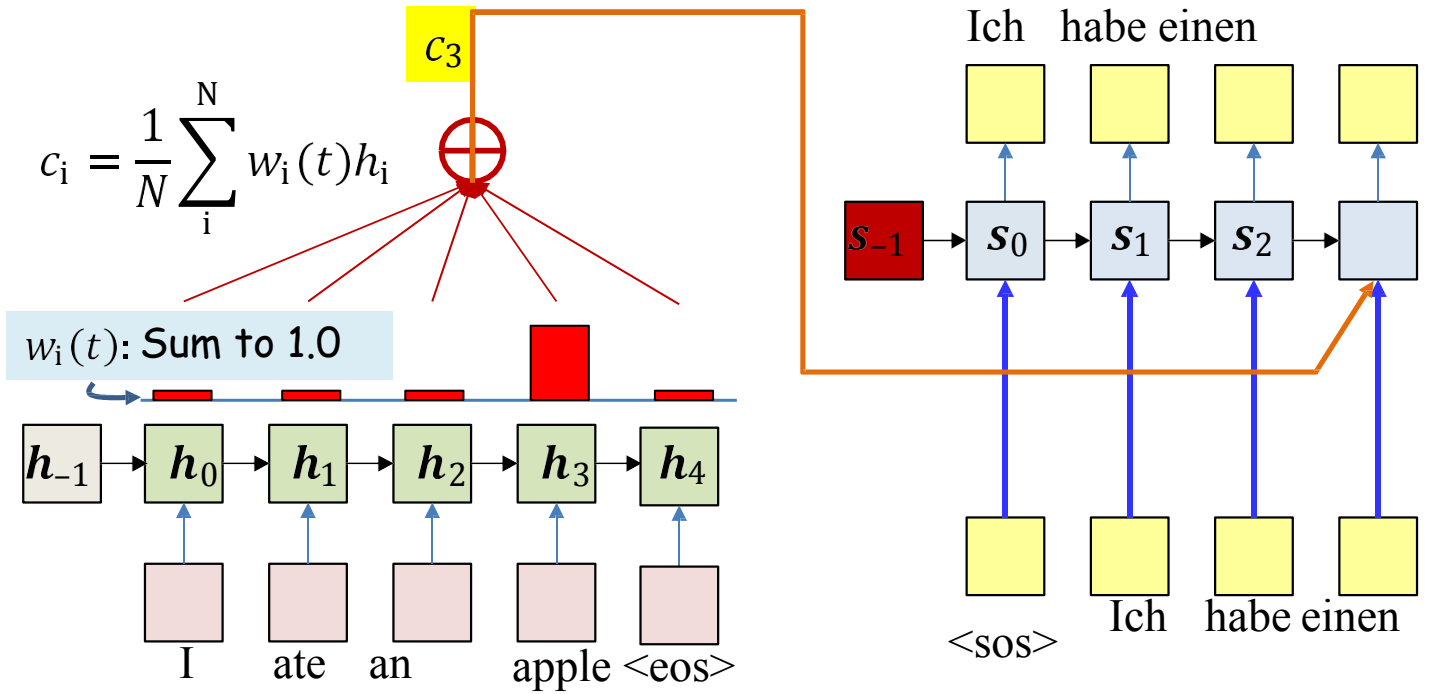
- The weights $w_i(t)$ must be positive and sum to 1.0
  - I.e. be a distribution
  - Ideally, they must be high for the most relevant inputs for the ith output and low elsewhere

- Solution: A two step weight computation
  - First compute *raw* weights (which could be +ve or –ve)
  - Then softmax them to convert them to a distribution

$$e_i(t) = g(\boldsymbol{h}_{t-1}, \boldsymbol{s}_{t-1})$$

$$w_i(t) = \frac{\exp(e_i(t))}{\sum_j \exp(e_j(t))}$$

# Attention weights




- Typical options for $g()$ (**variables in red must learned**)

$$g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}) = \boldsymbol{h}_i^T \boldsymbol{s}_{t-1}$$

$$g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}) = \boldsymbol{h}_i^T \boldsymbol{W}_g \boldsymbol{s}_{t-1}$$

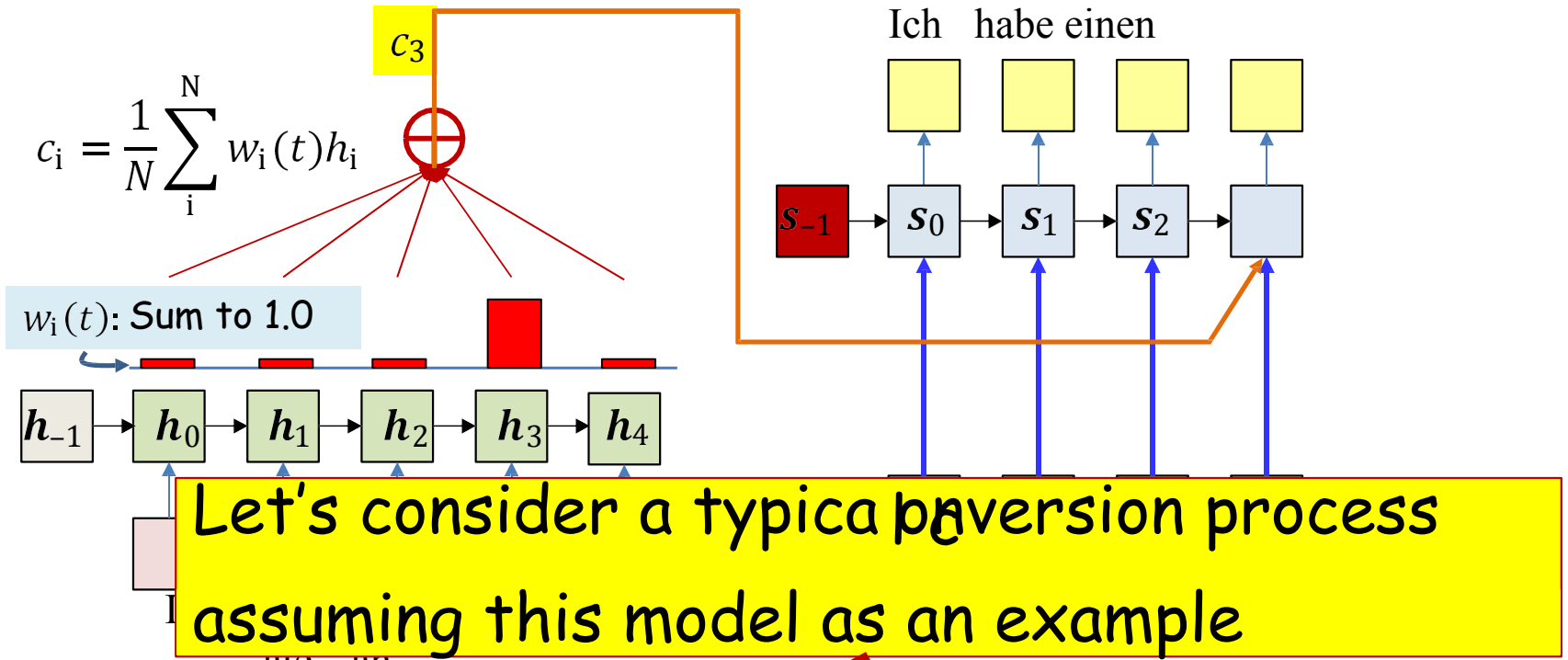
$$g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}) = \boldsymbol{v}_g^T tanh\left(\boldsymbol{W}_g \begin{bmatrix} \boldsymbol{h}_i \\ \boldsymbol{s}_{t-1} \end{bmatrix}\right)$$

$$g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}) = MLP([\boldsymbol{h}_i, \boldsymbol{s}_{t-1}])$$

$$e_i(t) = g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1})$$

$$w_i(t) = \frac{\exp(e_i(t))}{\sum_j \exp(e_j(t))}$$

# Attention weights




Let's consider a typical conversion process assuming this model as an example

- Typical options for $g()$ (**variables in red must learned**)

$$g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}) = \boldsymbol{h}_i^T \boldsymbol{s}_{t-1}$$

$$g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}) = \boldsymbol{h}_i^T \boldsymbol{W}_g \boldsymbol{s}_{t-1}$$

$$g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}) = \boldsymbol{v}_g^T \tanh\left(\boldsymbol{W}_g \begin{bmatrix} \boldsymbol{h}_i \\ \boldsymbol{s}_{t-1} \end{bmatrix}\right)$$

$$g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}) = MLP([\boldsymbol{h}_i, \boldsymbol{s}_{t-1}])$$

$$e_i(t) = g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1})$$

$$w_i(t) = \frac{\exp(e_i(t))}{\sum_j \exp(e_j(t))}$$

# Query, Key, Value (QKV)



- **Query:** What you want (the search)
- **Key:** Index of content (where to look)
- **Value:** Actual content (what you get)

## Intuition

Think of QKV like a search engine:

- **Query (Q):** Comes from the current decoder state (or token)
- **Key (K), Value (V):** Come from the encoder (or input sequence)

# QKV – Formulation



**Given:**

- Input sequence $\mathbf{X}$

**Linear projections:**

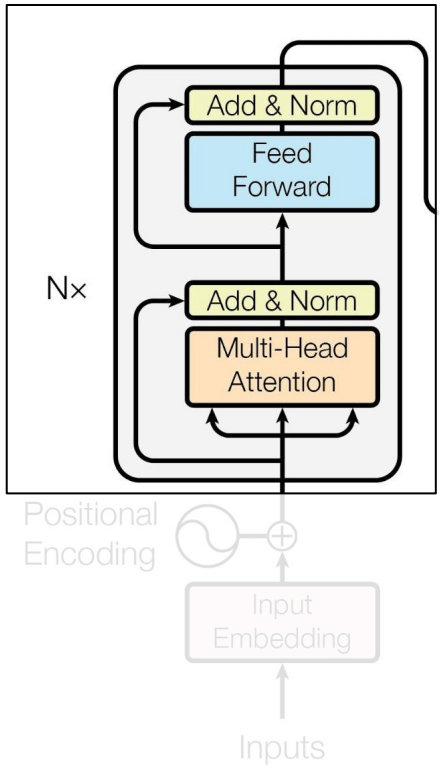
$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$$
$$\mathbf{K} = \mathbf{X}\mathbf{W}_K$$
$$\mathbf{V} = \mathbf{X}\mathbf{W}_V$$

**Attention:**

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

# Attention

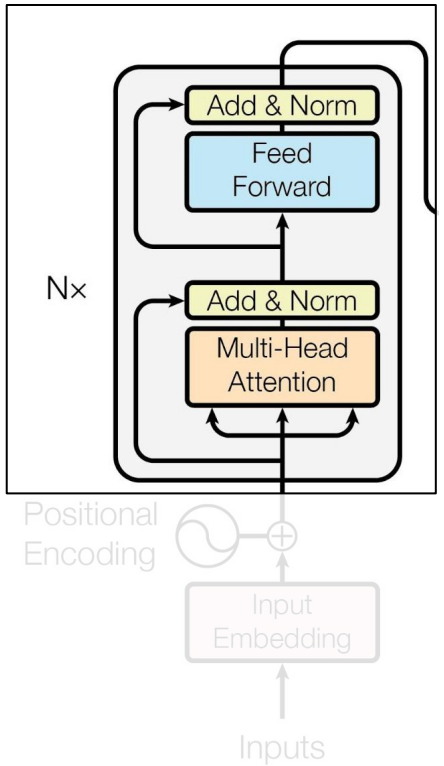
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



# Attention

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

- Query
- Key
- Value



# Query, Key & Value



**Database**

**{Key, Value store}**

```
{"order_100": {"items":"a1", "delivery_date":"a2", ...}},
{"order_101": {"items":"b1", "delivery_date":"b2", ...}},
{"order_102": {"items":"c1", "delivery_date":"c2", ...}},
{"order_103": {"items":"d1", "delivery_date":"d2", ...}},
{"order_104": {"items":"e1", "delivery_date":"e2", ...}},
{"order_105": {"items":"f1", "delivery_date":"f2", ...}},
{"order_106": {"items":"g1", "delivery_date":"g2", ...}},
{"order_107": {"items":"h1", "delivery_date":"h2", ...}},
{"order_108": {"items":"i1", "delivery_date":"i2", ...}},
{"order_109": {"items":"j1", "delivery_date":"j2", ...}},
{"order_110": {"items":"k1", "delivery_date":"k2", ...}}
```

# Query, Key & Value

{Query: “Order details of order_104”}

OR

{Query: “Order details of order_106”}

**Database**

**{Key, Value store}**


```
{"order_100": {"items":"a1", "delivery_date":"a2", ...}},
{"order_101": {"items":"b1", "delivery_date":"b2", ...}},
{"order_102": {"items":"c1", "delivery_date":"c2", ...}},
{"order_103": {"items":"d1", "delivery_date":"d2", ...}},
{"order_104": {"items":"e1", "delivery_date":"e2", ...}},
{"order_105": {"items":"f1", "delivery_date":"f2", ...}},
{"order_106": {"items":"g1", "delivery_date":"g2", ...}},
{"order_107": {"items":"h1", "delivery_date":"h2", ...}},
{"order_108": {"items":"i1", "delivery_date":"i2", ...}},
{"order_109": {"items":"j1", "delivery_date":"j2", ...}},
{"order_110": {"items":"k1", "delivery_date":"k2", ...}}
```

# Query, Key & Value

{Query: “Order details of order_104”}

OR

{Query: “Order details of order_106”}

{Key, Value store}


```
{"order_100": {"items":"a1", "delivery_date":"a2", ...}},
{"order_101": {"items":"b1", "delivery_date":"b2", ...}},
{"order_102": {"items":"c1", "delivery_date":"c2", ...}},
{"order_103": {"items":"d1", "delivery_date":"d2", ...}},
{"order_104": {"items":"e1", "delivery_date":"e2", ...}},
{"order_105": {"items":"f1", "delivery_date":"f2", ...}},
{"order_106": {"items":"g1", "delivery_date":"g2", ...}},
{"order_107": {"items":"h1", "delivery_date":"h2", ...}},
{"order_108": {"items":"i1", "delivery_date":"i2", ...}},
{"order_109": {"items":"j1", "delivery_date":"j2", ...}},
{"order_110": {"items":"k1", "delivery_date":"k2", ...}}
```

# Query, Key & Value

{Query: “Order details of order_104”}

OR

{Query: “Order details of order_106”}

{Key, Value store}

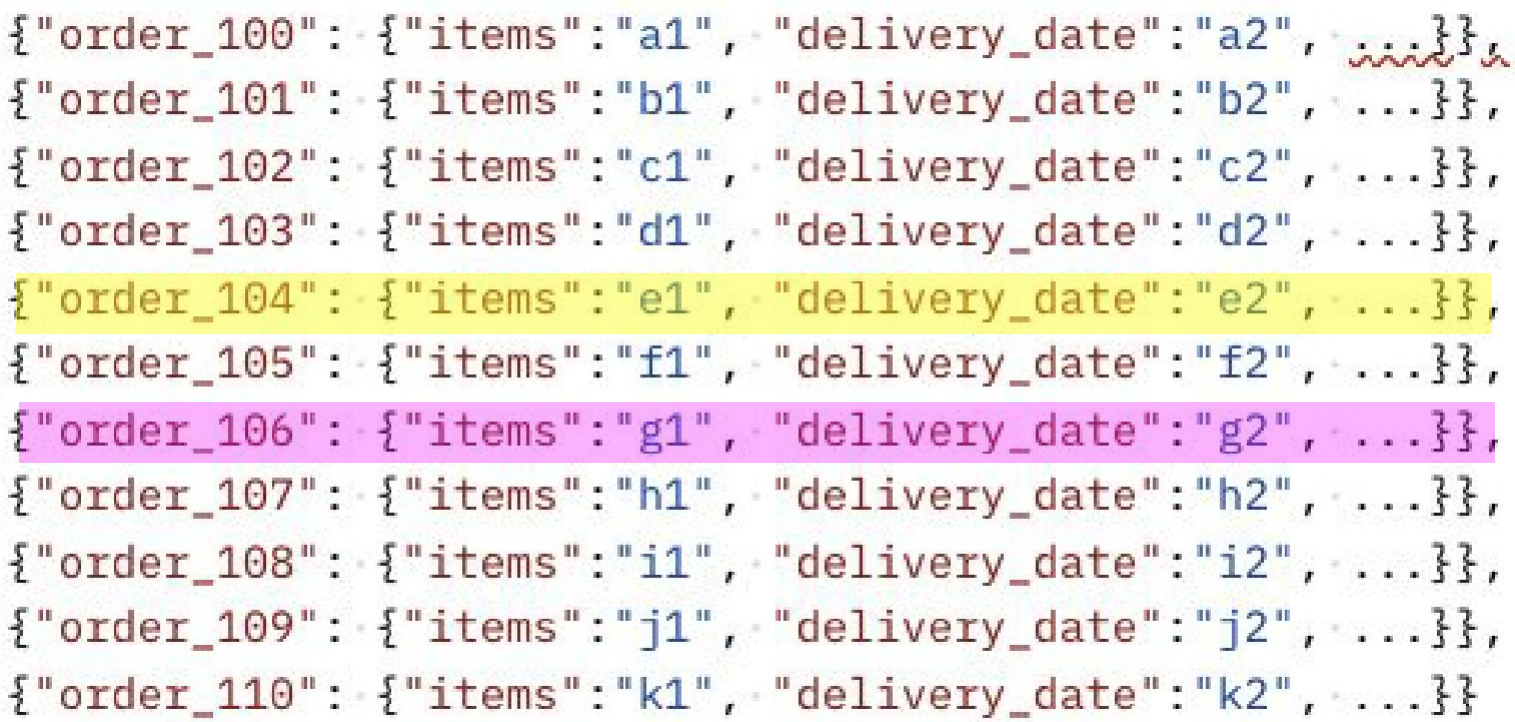
# Query, Key & Value

{Query: “Order details of order_104”}

OR

{Query: “Order details of order_106”}

{Key, Value store}



# Query, Key & Value



**Done at the same time !!**

{Query: “Order details of order_104”}

OR

{Query: “Order details of order_106”}

{Key, Value store}


```
{"order_100": {"items":"a1", "delivery_date":"a2", ...}},
{"order_101": {"items":"b1", "delivery_date":"b2", ...}},
{"order_102": {"items":"c1", "delivery_date":"c2", ...}},
{"order_103": {"items":"d1", "delivery_date":"d2", ...}},
{"order_104": {"items":"e1", "delivery_date":"e2", ...}},
{"order_105": {"items":"f1", "delivery_date":"f2", ...}},
{"order_106": {"items":"g1", "delivery_date":"g2", ...}},
{"order_107": {"items":"h1", "delivery_date":"h2", ...}},
{"order_108": {"items":"i1", "delivery_date":"i2", ...}},
{"order_109": {"items":"j1", "delivery_date":"j2", ...}},
{"order_110": {"items":"k1", "delivery_date":"k2", ...}}
```

# Query, Key & Value

{Query: “Order details of order_104”}

OR

{Query: “Order details of order_106”}

```
{"order_100": {"items":"a1", "delivery_date":"a2", ...}},
{"order_101": {"items":"b1", "delivery_date":"b2", ...}},
{"order_102": {"items":"c1", "delivery_date":"c2", ...}},
{"order_103": {"items":"d1", "delivery_date":"d2", ...}},
{"order_104": {"items":"e1", "delivery_date":"e2", ...}},
{"order_105": {"items":"f1", "delivery_date":"f2", ...}},
{"order_106": {"items":"g1", "delivery_date":"g2", ...}},
{"order_107": {"items":"h1", "delivery_date":"h2", ...}},
{"order_108": {"items":"i1", "delivery_date":"i2", ...}},
{"order_109": {"items":"j1", "delivery_date":"j2", ...}},
{"order_110": {"items":"k1", "delivery_date":"k2", ...}}
```

## Query

1. Search for info

## Key

1. Interacts directly with Queries
2. Distinguishes one object from another
3. Identify which object is the most relevant and by how much

## Value

1. Actual details of the object
2. More fine grained

# Decoding Sequences with Attention

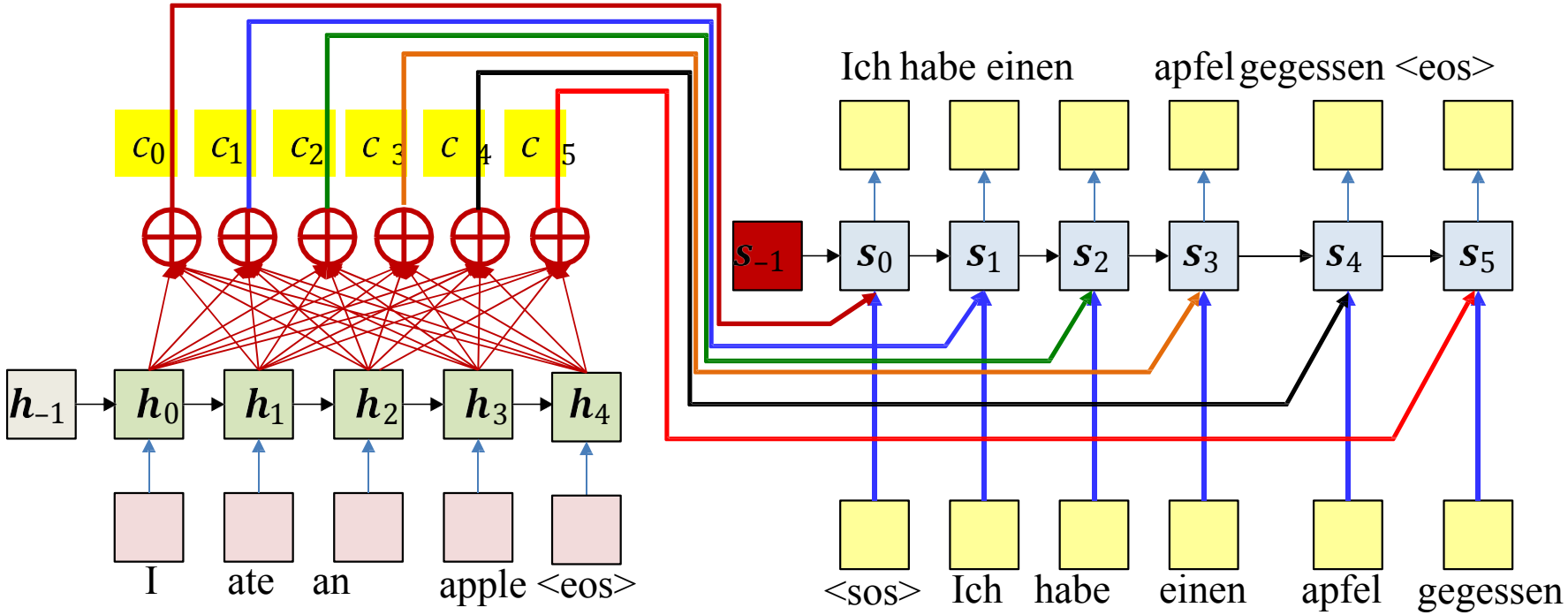


## Attention-Based Decoding – Overview

- The decoder uses attention to select relevant encoder hidden states at each decoding step.
- **Combines:**
  - Previous decoder state
  - Encoder hidden states
  - Attention context vector
- The output at time $t$:

$$y_t = \text{Decoder}(y_{t-1}, \text{context}_t)$$

# Attention-based decoding




$$e_i(t) = g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1})$$

$$w_i(t) = \frac{\exp(e_i(t))}{\sum_j \exp(e_j(t))}$$

$$c_t = \frac{1}{N}\sum_i^N w_i(t) h_i$$

# Decoding - Steps



1. **Compute query:** Generate a query vector from the current decoder hidden state.
2. **Align with encoder outputs:** Compare the query with encoder outputs (keys) to measure relevance.
3. **Get attention weights:** Apply softmax to the alignment scores to obtain attention weights.
4. **Compute context:** Calculate the context vector as the weighted sum of encoder outputs using the attention weights.
5. **Feed into decoder:** Use the context vector and previous outputs as input for the next decoding step.

# Attention

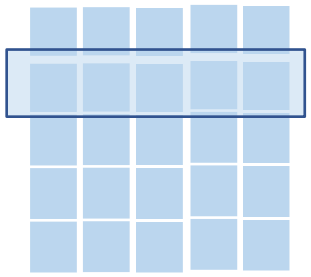
Query



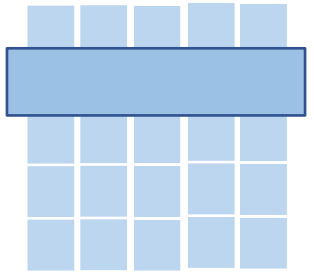
Key

Value

Store

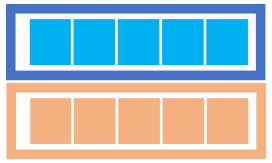


Key



Value

# Attention



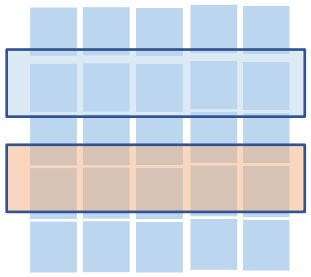
Query



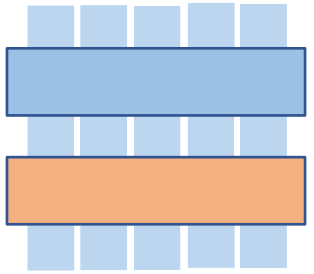
Key

Value

Store



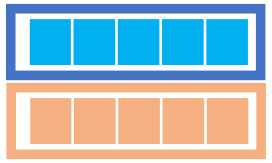
Key



Value

# Attention

**Done at the same time !!**



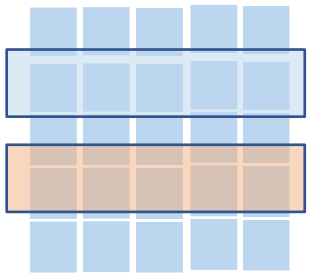
Query



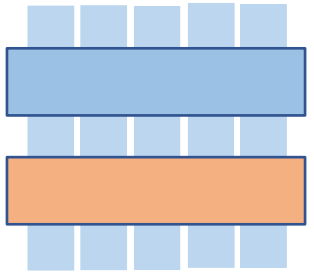
Key

Value

Store



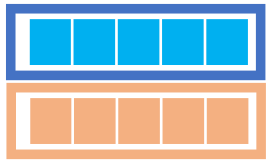
Key



Value

# Attention

Parallelizable !!!



Query

Q

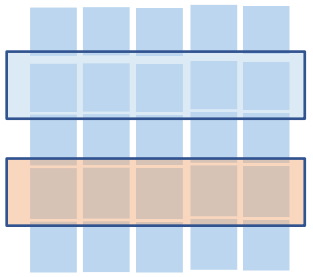


Key

Value

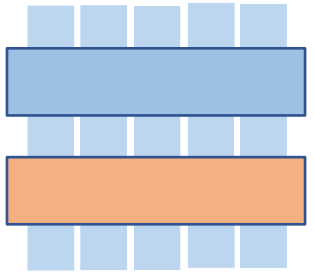
Store

$QK^T$



Key

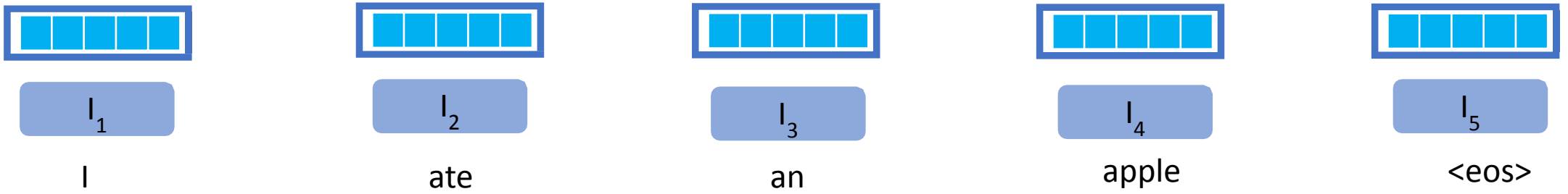
$$softmax(\frac{QK^T}{\sqrt{d}})$$



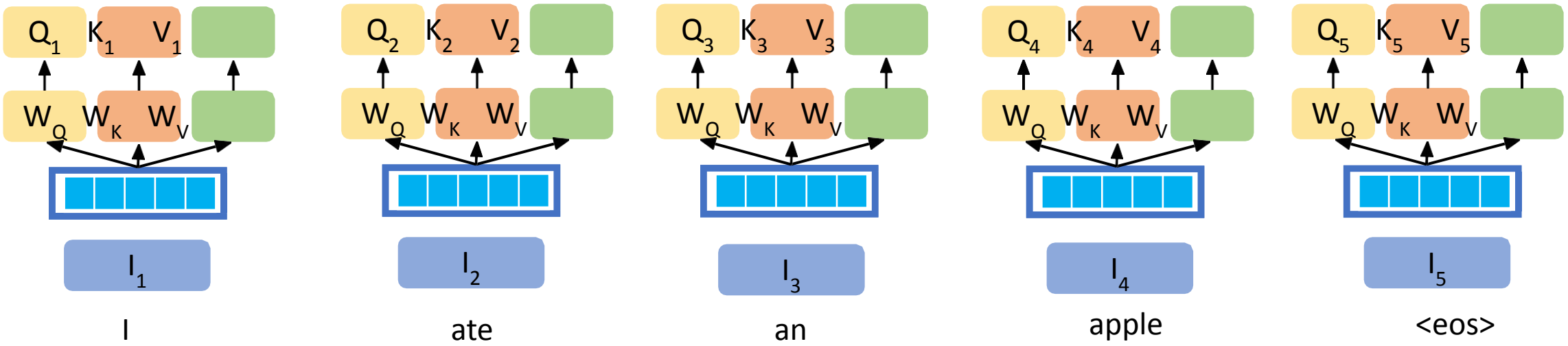
Value

$$softmax(\frac{QK^T}{\sqrt{d}})V$$

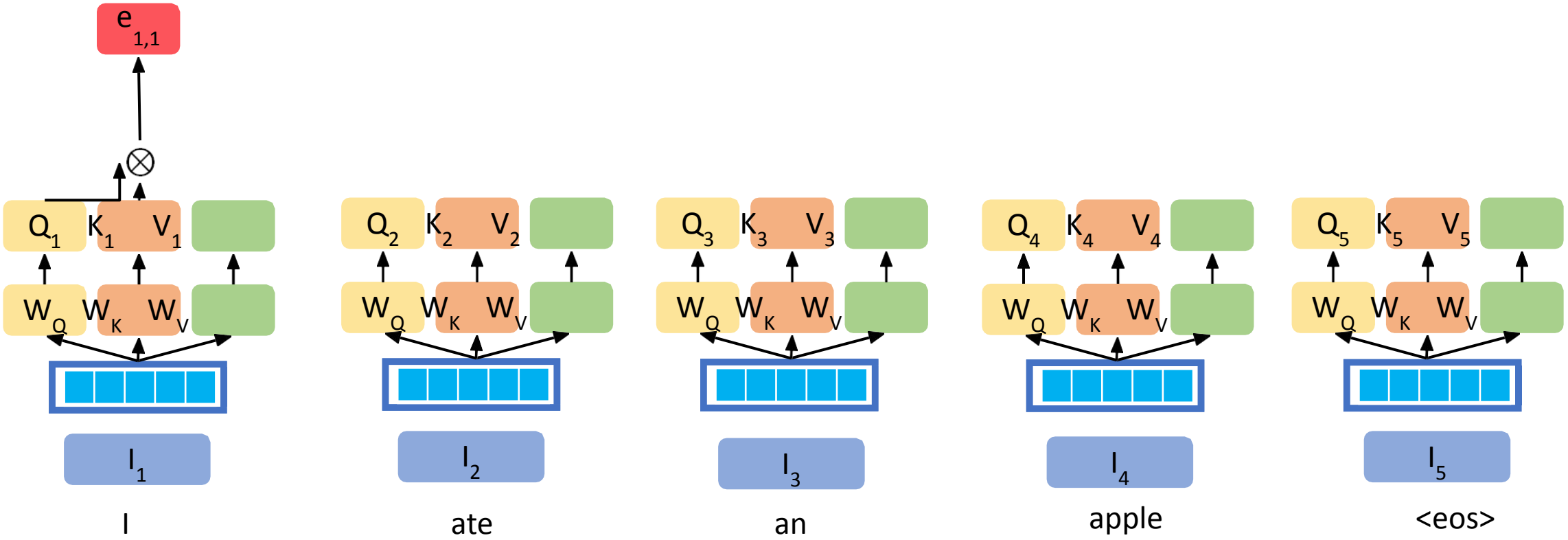
# Attention



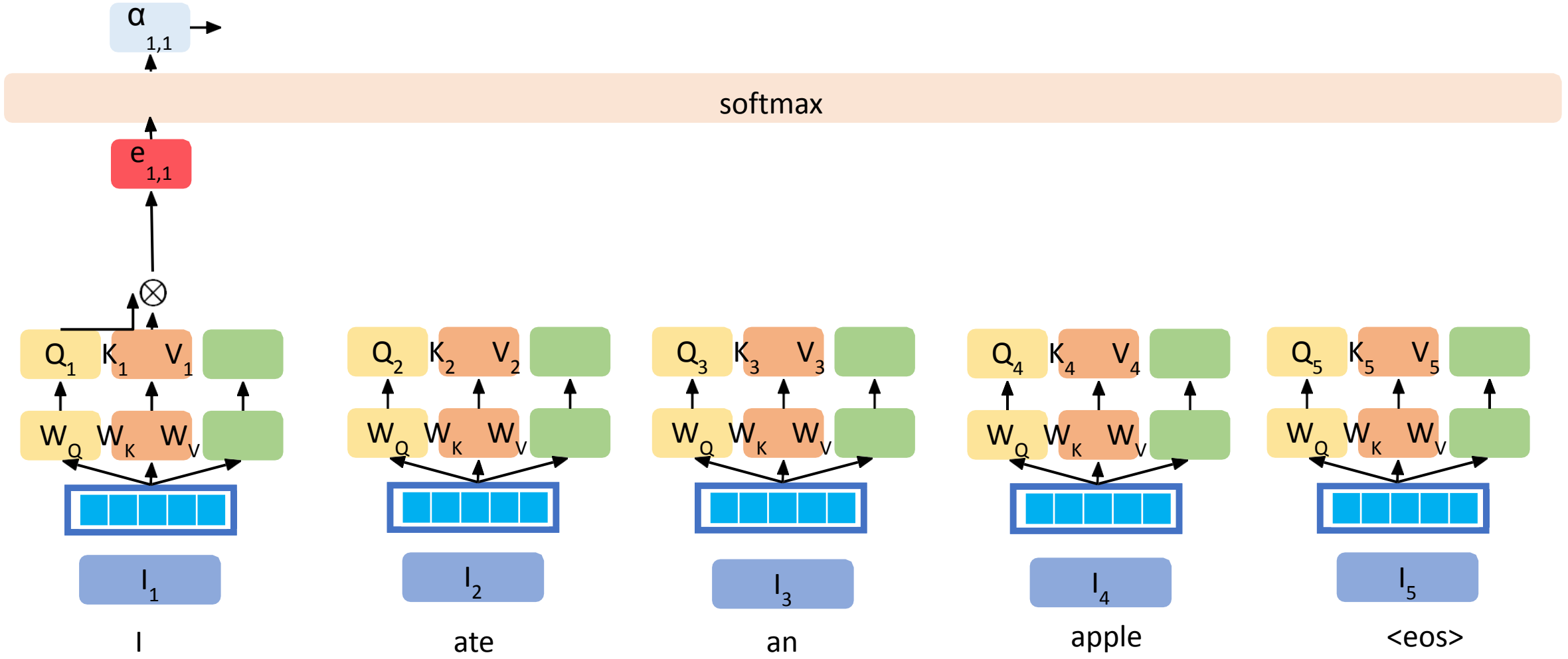
# Attention


Q1 K1 V1
WQ WK WV
I1
I
Q2 K2 V2
WQ WK WV
I2
ate
Q3 K3 V3
WQ WK WV
I3
an
Q4 K4 V4
WQ WK WV
I4
apple
Q5 K5 V5
WQ WK WV
I5
<eos>

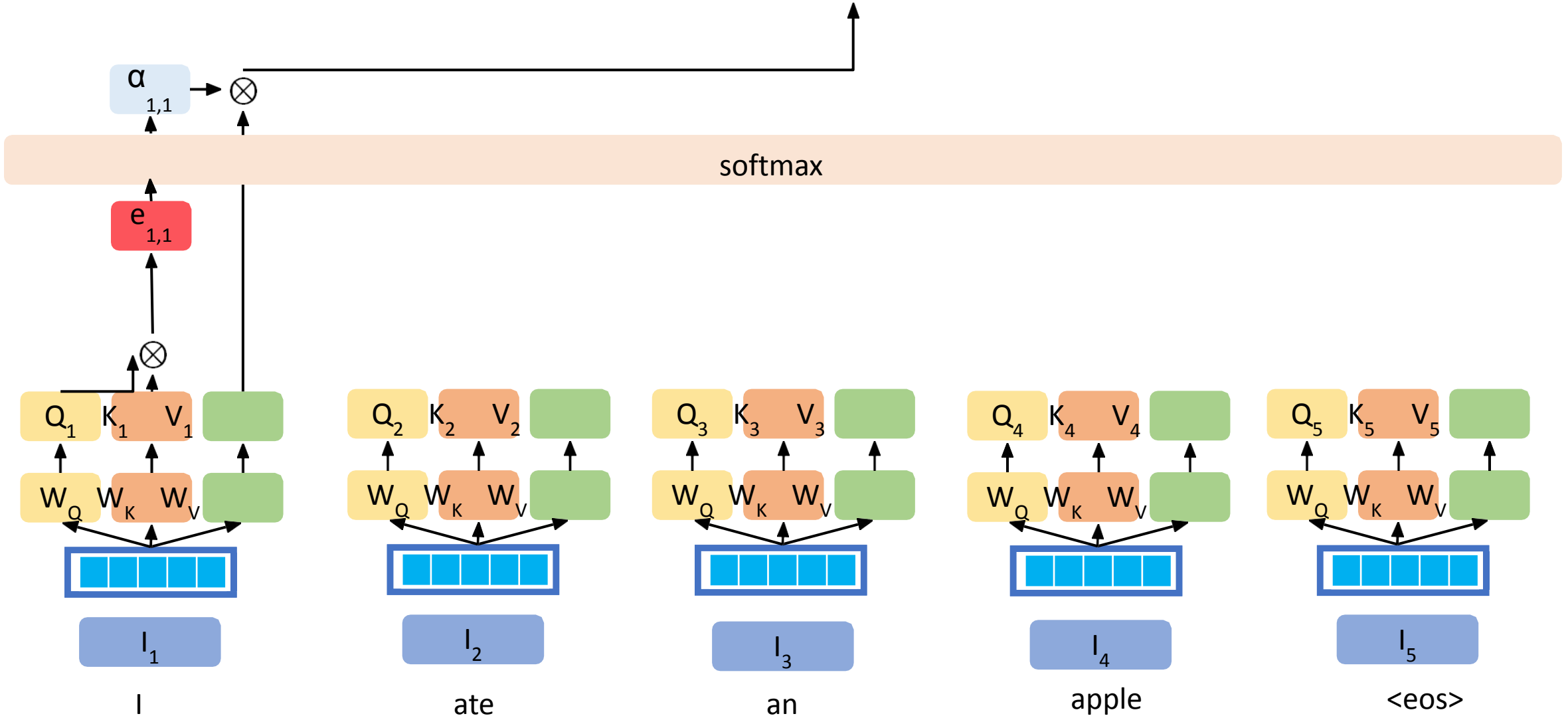
# Attention


e 1,1
Q1 K1 V1
WQ WK WV
I1
I
Q2 K2 V2
WQ WK WV
I2
ate
Q3 K3 V3
WQ WK WV
I3
an
Q4 K4 V4
WQ WK WV
I4
apple
Q5 K5 V5
WQ WK WV
I5
<eos>

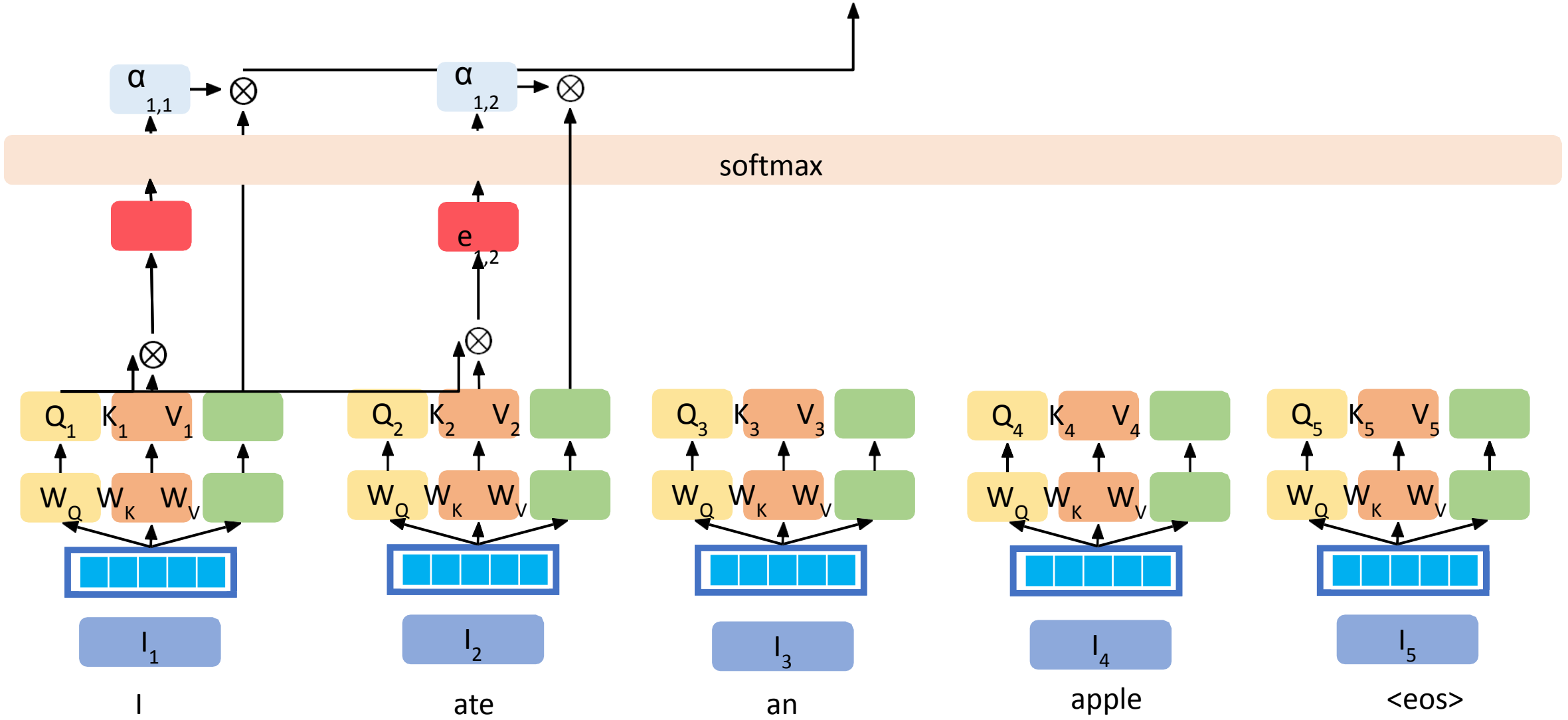
# Attention


α 1,1
softmax
e 1,1
Q1 K1 V1
WQ WK WV
I1
I
Q2 K2 V2
WQ WK WV
I2
ate
Q3 K3 V3
WQ WK WV
I3
an
Q4 K4 V4
WQ WK WV
I4
apple
Q5 K5 V5
WQ WK WV
I5
<eos>

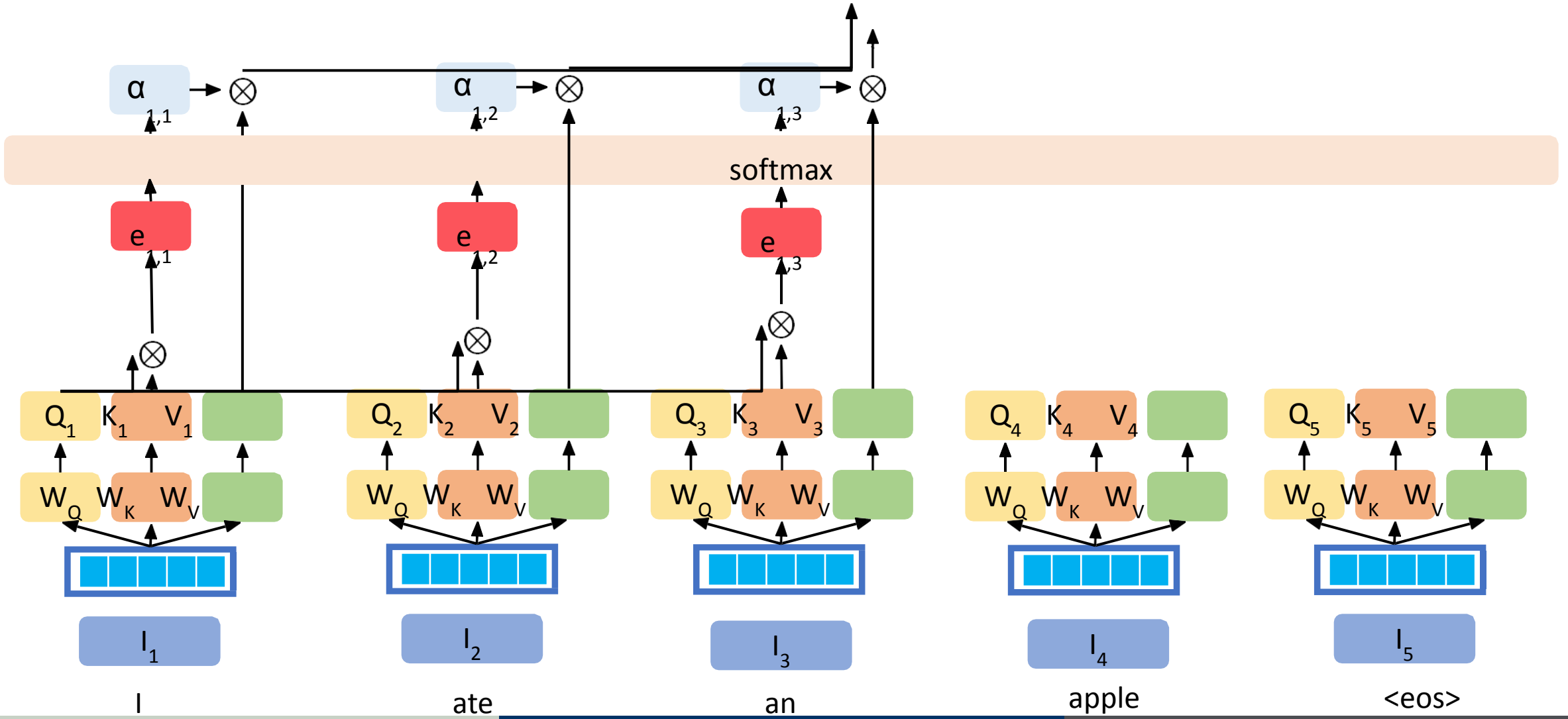
# Attention


α 1,1
softmax
e 1,1
Q1 K1 V1
WQ WK WV
I1
I
Q2 K2 V2
WQ WK WV
I2
ate
Q3 K3 V3
WQ WK WV
I3
an
Q4 K4 V4
WQ WK WV
I4
apple
Q5 K5 V5
WQ WK WV
I5
<eos>

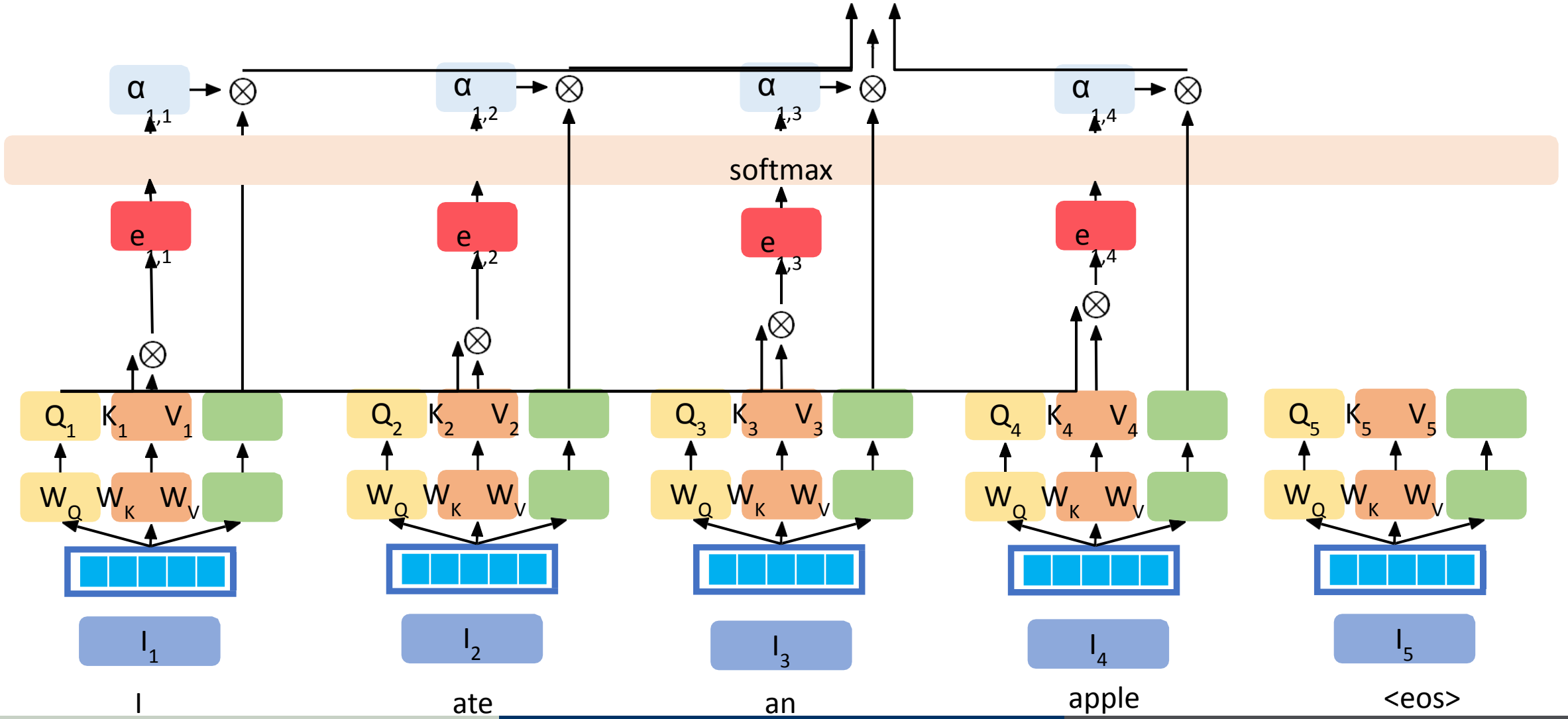
# Attention


α 1,1
α 1,2
softmax
e 1,2
Q1 K1 V1
Q2 K2 V2
Q3 K3 V3
Q4 K4 V4
Q5 K5 V5
WQ WK WV
I1
I2
I3
I4
I5
I
ate
an
apple
<eos>

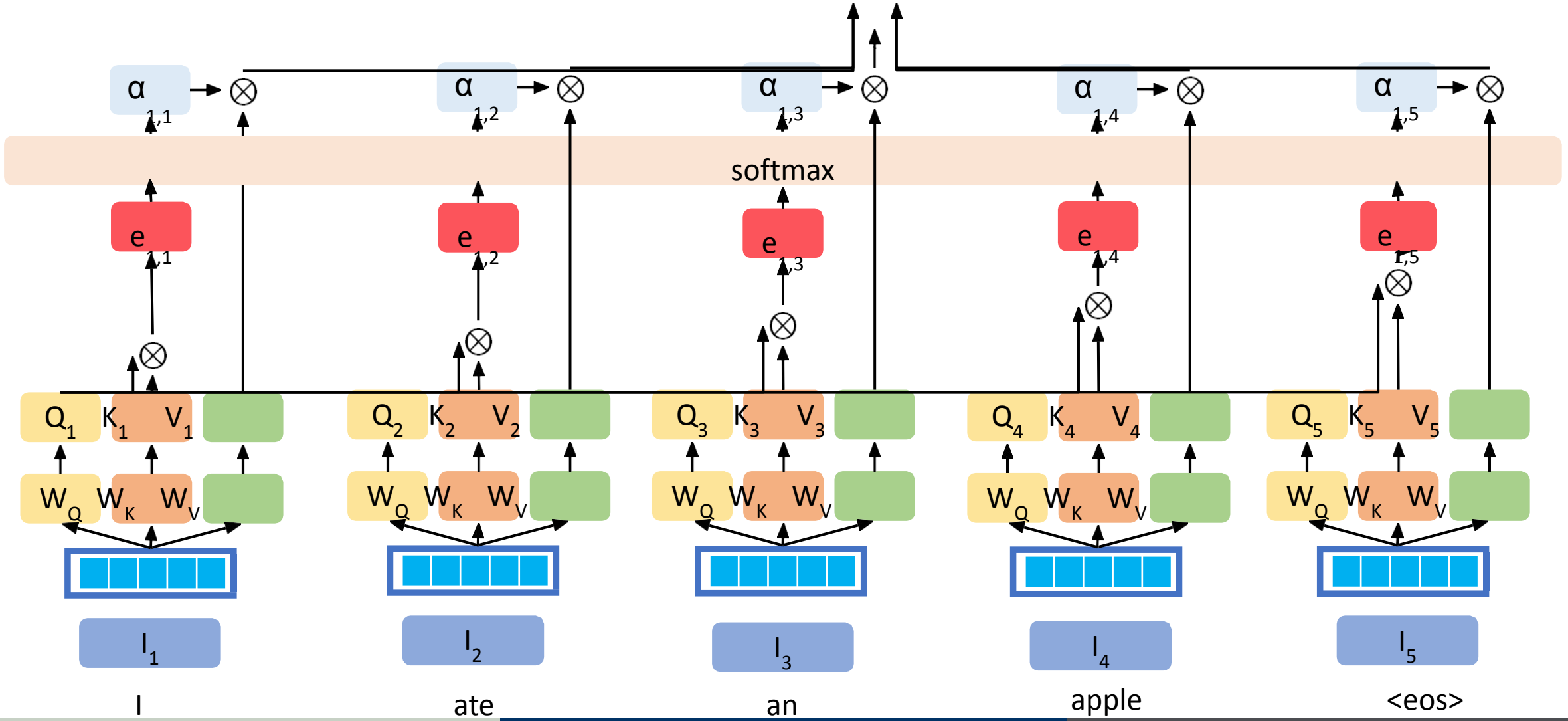
# Attention


α 1,1
α 1,2
α 1,3
softmax
e 1,1
e 1,2
e 1,3
Q1 K1 V1
Q2 K2 V2
Q3 K3 V3
Q4 K4 V4
Q5 K5 V5
WQ WK WV
I1
I2
I3
I4
I5
I
ate
an
apple
<eos>

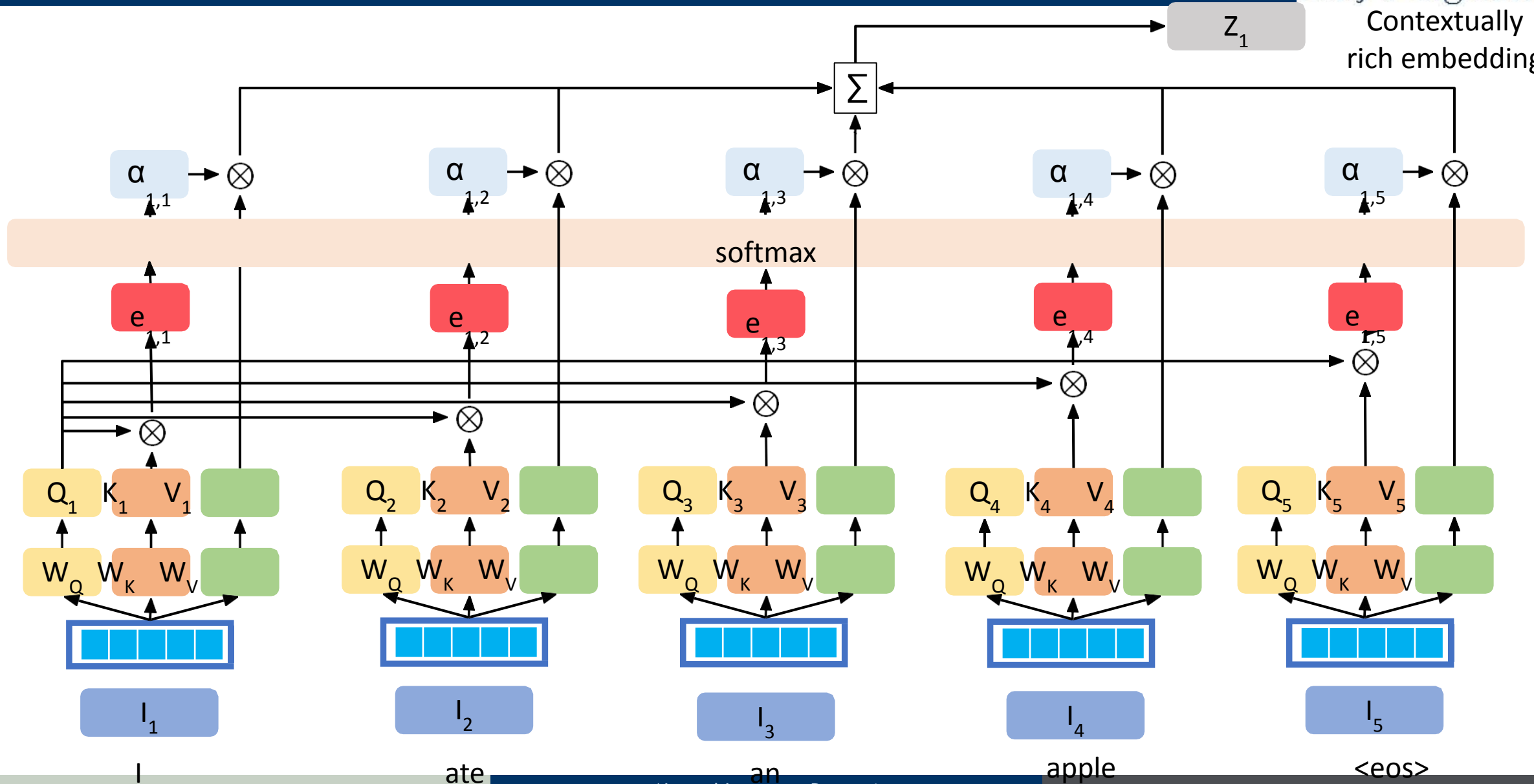
# Attention


α 1,1
α 1,2
α 1,3
α 1,4
softmax
e 1,1
e 1,2
e 1,3
e 1,4
Q1 K1 V1
Q2 K2 V2
Q3 K3 V3
Q4 K4 V4
Q5 K5 V5
WQ WK WV
I1
I2
I3
I4
I5
I
ate
an
apple
<eos>

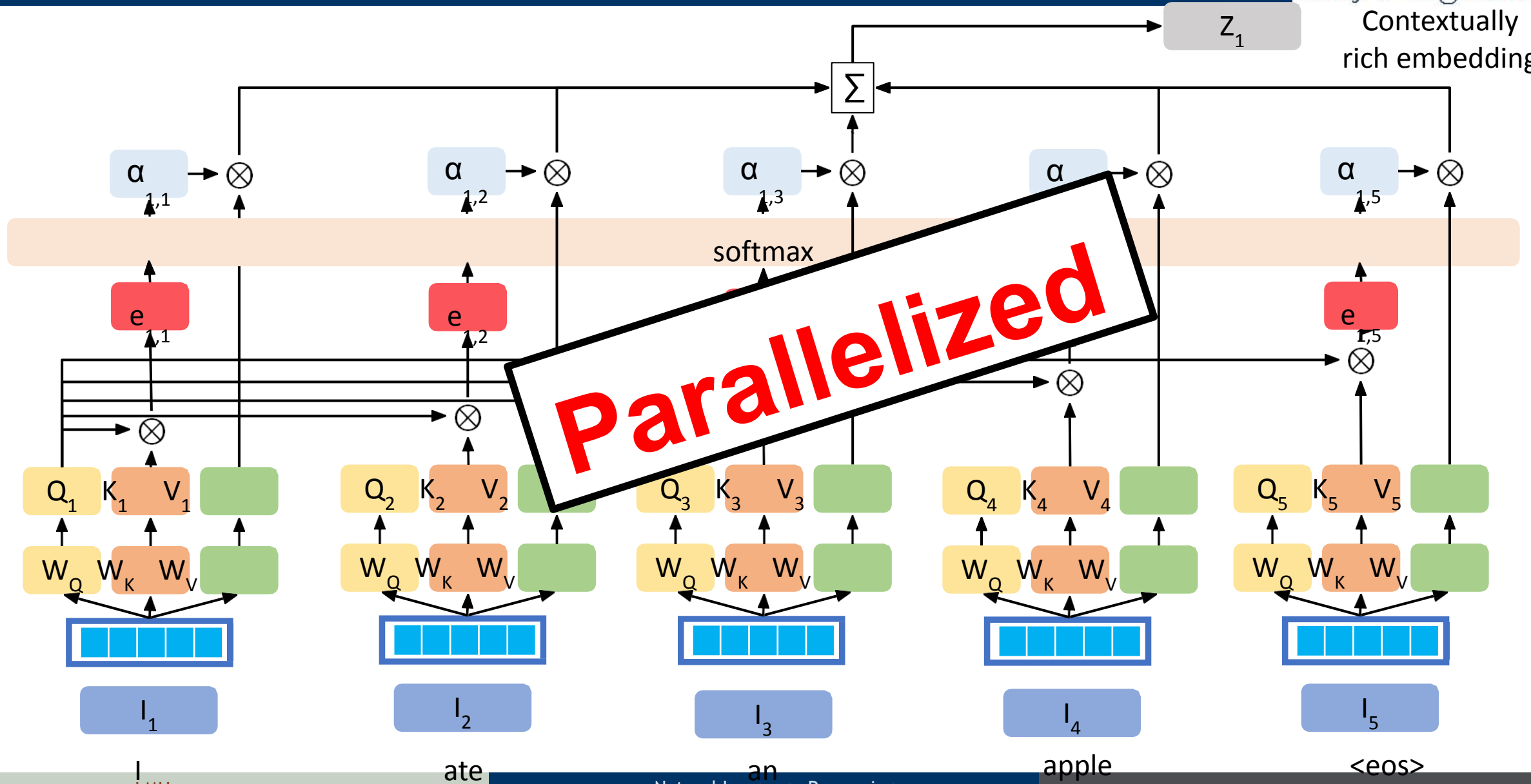
# Attention


α 1,1
α 1,2
α 1,3
α 1,4
α 1,5
softmax
e 1,1
e 1,2
e 1,3
e 1,4
e 1,5
Q1 K1 V1
Q2 K2 V2
Q3 K3 V3
Q4 K4 V4
Q5 K5 V5
WQ WK WV
I1
I2
I3
I4
I5
I
ate
an
apple
<eos>

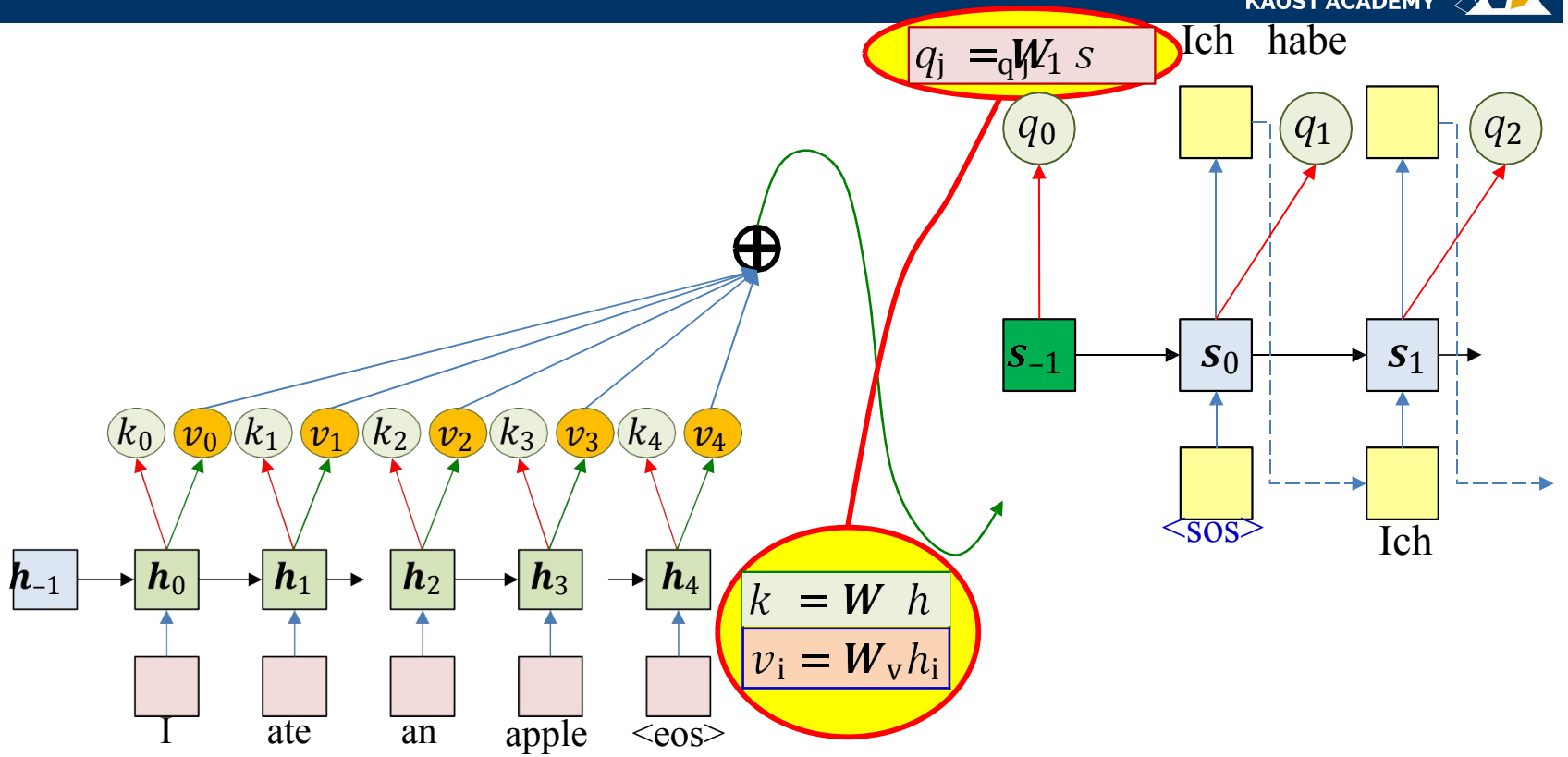
# Attention


Z1
Contextually rich embedding
Σ
α1,1
α1,2
α1,3
α1,4
α1,5
softmax
e1,1
e1,2
e1,3
e1,4
e1,5
Q1 K1 V1
Q2 K2 V2
Q3 K3 V3
Q4 K4 V4
Q5 K5 V5
WQ WK WV
I1
I2
I3
I4
I5
I
ate
an
apple
<eos>

# Attention

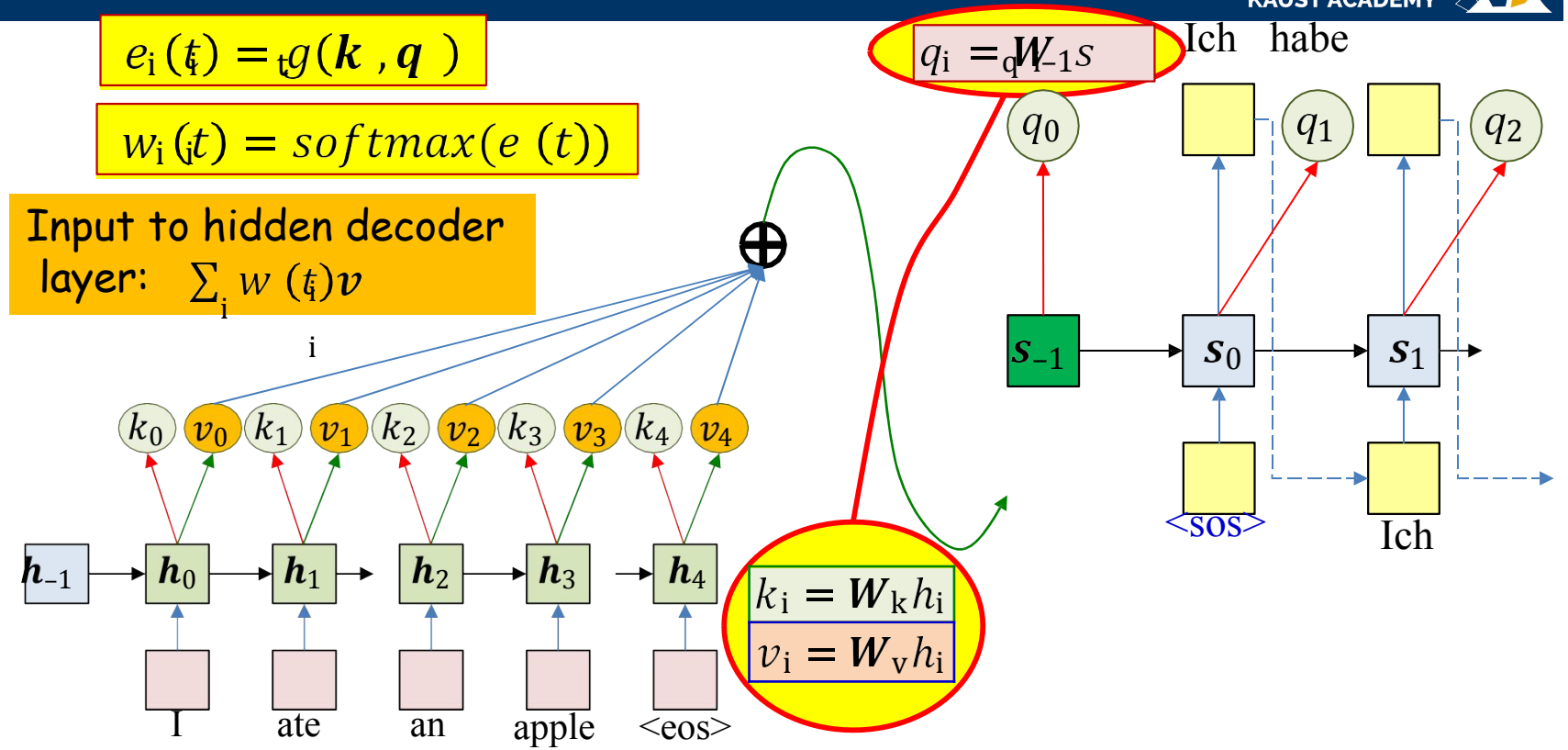

Contextually rich embedding
Z1
Σ
α1,1
α1,2
α1,3
α
α1,5
softmax
e1,1
e1,2
e1,5
Parallelized
Q1 K1 V1
Q2 K2 V2
Q3 K3 V3
Q4 K4 V4
Q5 K5 V5
WQ WK WV
I1
I2
I3
I4
I5
I
ate
an
apple
<eos>

# Modification: Query key value




- Encoder outputs an explicit “key” and “value” at each input time
  - Key is used to evaluate the importance of the input at that time, for a given output
- Decoder outputs an explicit “query” at each output time
  - Query is used to evaluate which inputs to pay attention to
- The weight is a function of key and query
- The actual context is a weighted sum of value

# Modification: Query key value



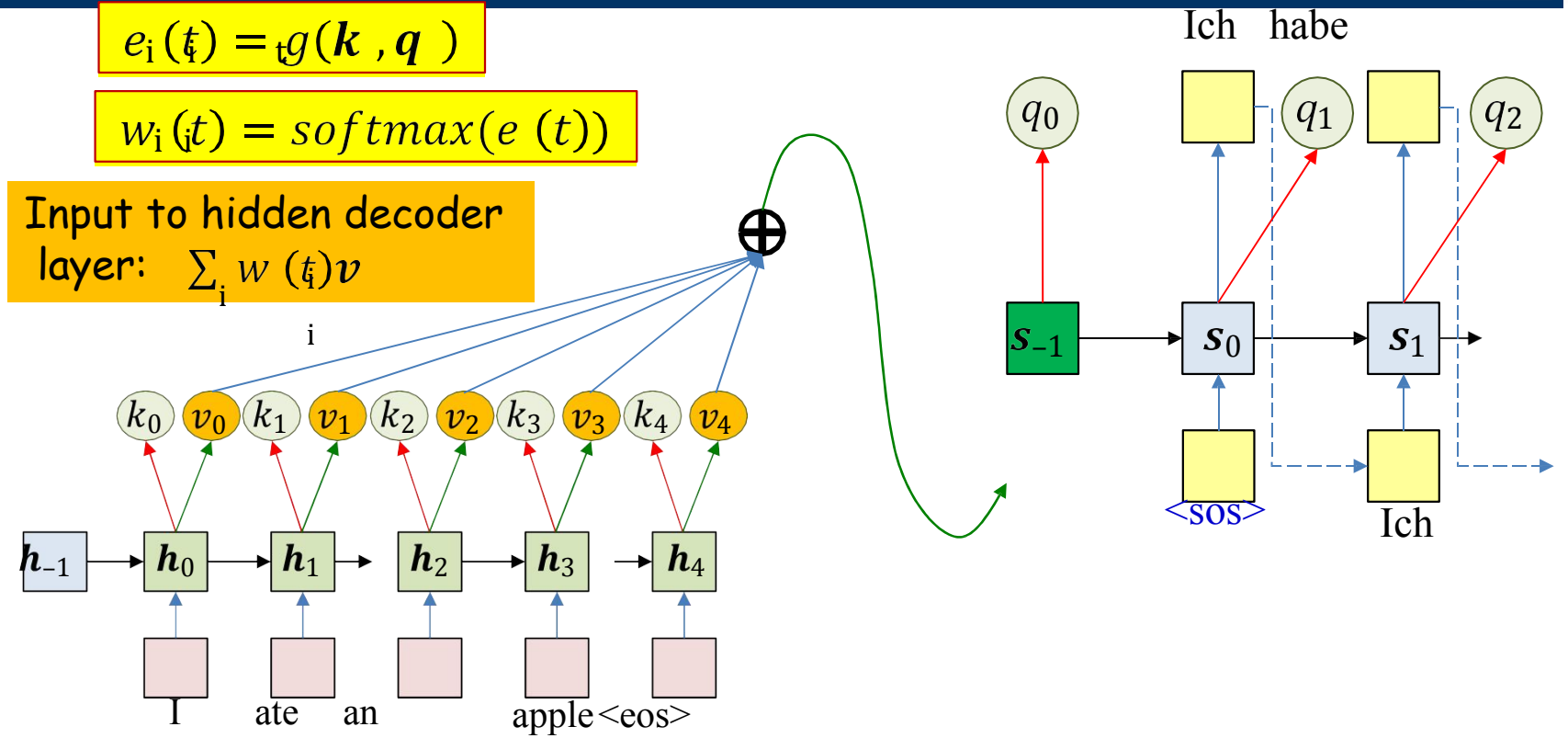

- Encoder outputs an explicit "key" and "value" at each input time
  - Key is used to evaluate the importance of the input at that time, for a given output
- Decoder outputs an explicit "query" at each output time
  - Query is used to evaluate which inputs to pay attention to
- The weight is a function of key and query
- The actual context is a weighted sum of value

# Modification: Query key value

$$e_i(t) = g(\boldsymbol{k}, \boldsymbol{q})$$

$$w_i(t) = softmax(e(t))$$

Input to hidden decoder layer: $\sum_i w_i(t) v$




Special case: $k_i = v_i = h_i$

$q_t = s_{t-1}$

- The weight is a function of key and query
- The actual context is a weighted sum of value

# Pseudocode



```
# Assuming encoded input
#   (K,V) = [k_enc[0]… k_enc[T]], [v_enc[0]… v_enc[T]]
# is available

t = -1
h_out[-1] = 0     # Initial Decoder hidden state
q[0] = 0          # Initial query

# Note: begins with a "start of sentence" symbol
#  <sos> and <eos> may be identical
Y_out[0] = <sos>
do
     t = t+1
     C  = compute_context_with_attention(q[t], K, V)
     y[t],h_out[t],q[t+1] = RNN_decode_step(h_out[t-1], y_out[t-1], C)
     y_out[t] = generate(y[t]) # Random, or greedy
until y_out[t] == <eos>
```

# Decoding - Example: Translation



**Example: English to French Translation**

Source Sentence (English)

*The cat sat on the mat.*

Target Sentence (French)

*Le chat s'est assis sur le tapis.*

- At each decoding step, the decoder generates a French word by:
  1. Using the current decoder hidden state.
  2. Attending to relevant English words via attention weights.
  3. Computing a context vector as a weighted sum of encoder outputs.
- For example, when generating "chat", the attention focuses on "cat".
- The process repeats for each word, dynamically shifting attention.

# Limitations of Attention Mechanisms



- **Quadratic Complexity:** Attention computation scales as $\mathcal{O}(n^2)$ with sequence length, making it expensive for long sequences.
- **Shallow Reasoning:** Standard attention mechanisms typically consider only one step of interaction, limiting deep reasoning capabilities.
- **Noisy Attention Maps:** Attention distributions can be diffuse or hard to interpret, making model decisions less transparent.
- **Limited Long-Term Memory:** Attention alone may struggle to capture dependencies over very long contexts.

# Future Directions in Attention Mechanisms



- **Efficient Attention Variants:**
  - *Linformer, Performer, Longformer, FlashAttention* – Reduce memory and computation for long sequences.
- **Structured Attention:**
  - *Sparsemax, Routing Attention* – Impose structure or sparsity for interpretability and efficiency.
- **Cross-modal Attention:**
  - *Vision + Language (e.g., Flamingo, Gato)* – Integrate information across different modalities.
- **Learnable Routing:**
  - *Mixture-of-Experts, Dynamic Attention* – Dynamically select computation paths for scalability.

# Summary

| Concept | Key Takeaway |
|---|---|
| Alignment | Scores relevance between states |
| QKV | Core mechanism to compute attention |
| Decoding | Uses attention to generate outputs |
| Limitation | Costly, hard to scale to long seqs |
| Future | Efficient, structured, multimodal attention |

# References



These slides have been adapted from:

- Younes Mourri & Lukasz Kaiser, Natural Language Processcing Specialization, DeepLearning.Ai
- Bhiksha Raj & Rita Singh, 11-785 Introduction to Deep Learning, CMU

# References


- Bahdanau, D., Cho, K., & Bengio, Y. (2015). *Neural Machine Translation by Jointly Learning to Align and Translate.*
- Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). *Attention Is All You Need.*
- Raffel, C., Shazeer, N., Roberts, A., et al. (2020). *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer.*
- Choromanski, K., Likhosherstov, V., Dohan, D., et al. (2021). *Rethinking Attention with Performers.*
- Tay, Y., Dehghani, M., Bahri, D., & Metzler, D. (2023). *Efficient Transformers: A Survey.*
- Google Research Slides: *The Annotated Transformer.*

## Credits

Dr. Prashant Aparajeya

Computer Vision Scientist — Director(AISimply Ltd)

p.aparajeya@aisimply.uk

This project benefited from external collaboration, and we acknowledge their contribution with gratitude.