Lecture 10

Tian Han

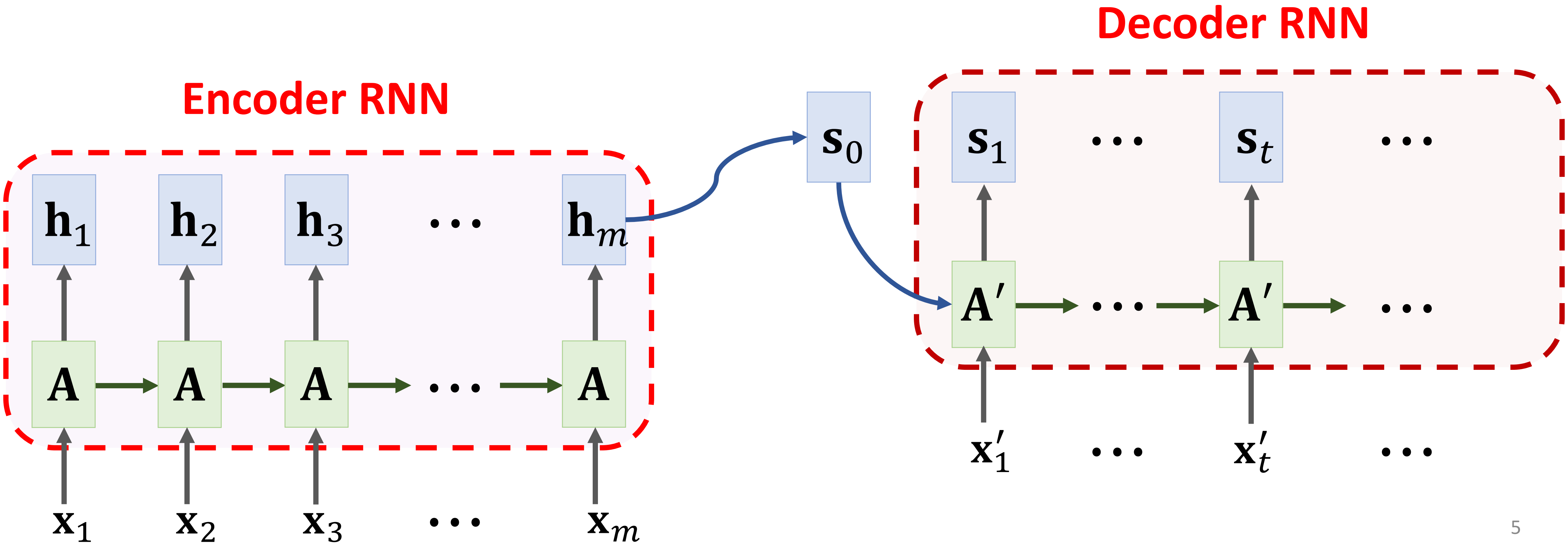
Outline

- Attention For RNN
- Self-attention
- Transformer basic (attention, self-attention layer)

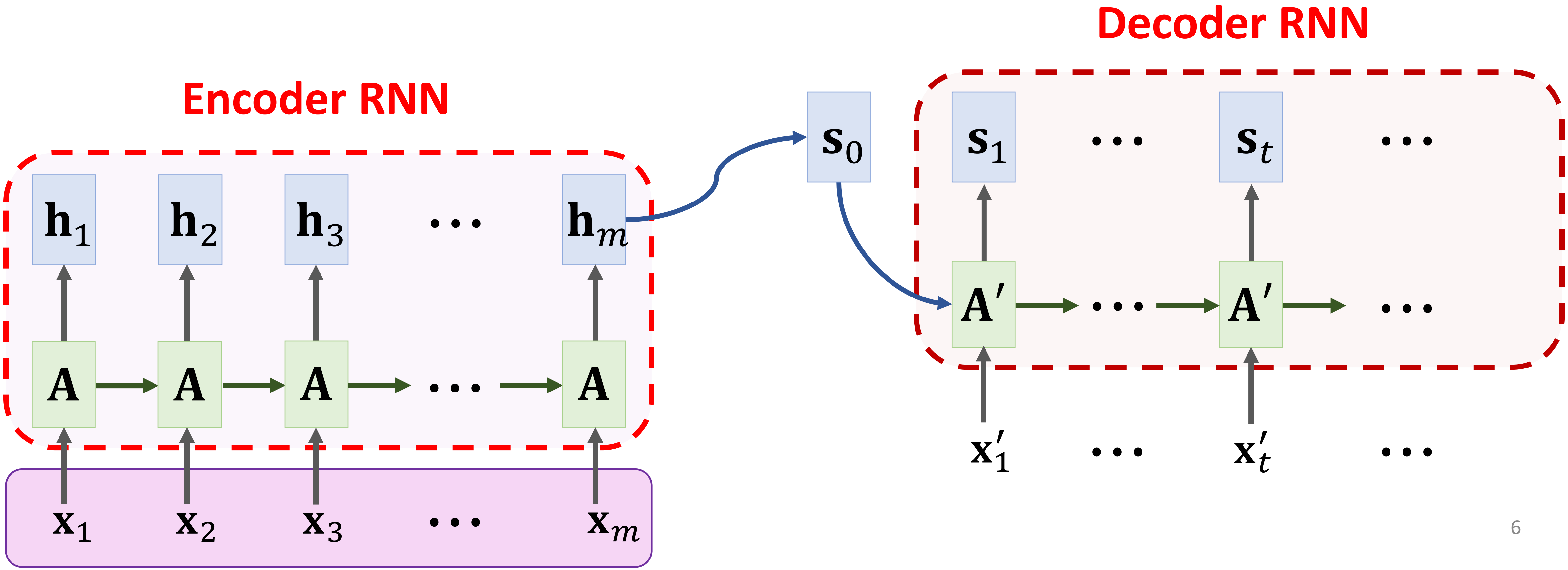
Attention for RNNs

Revisiting Seq2Seq Model

Seq2Seq Model

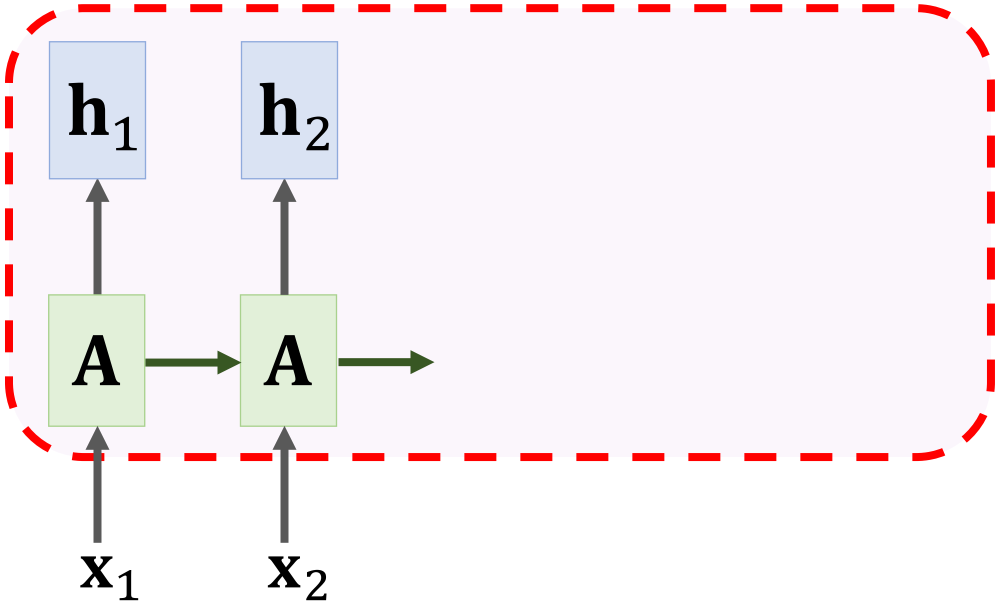


Seq2Seq Model

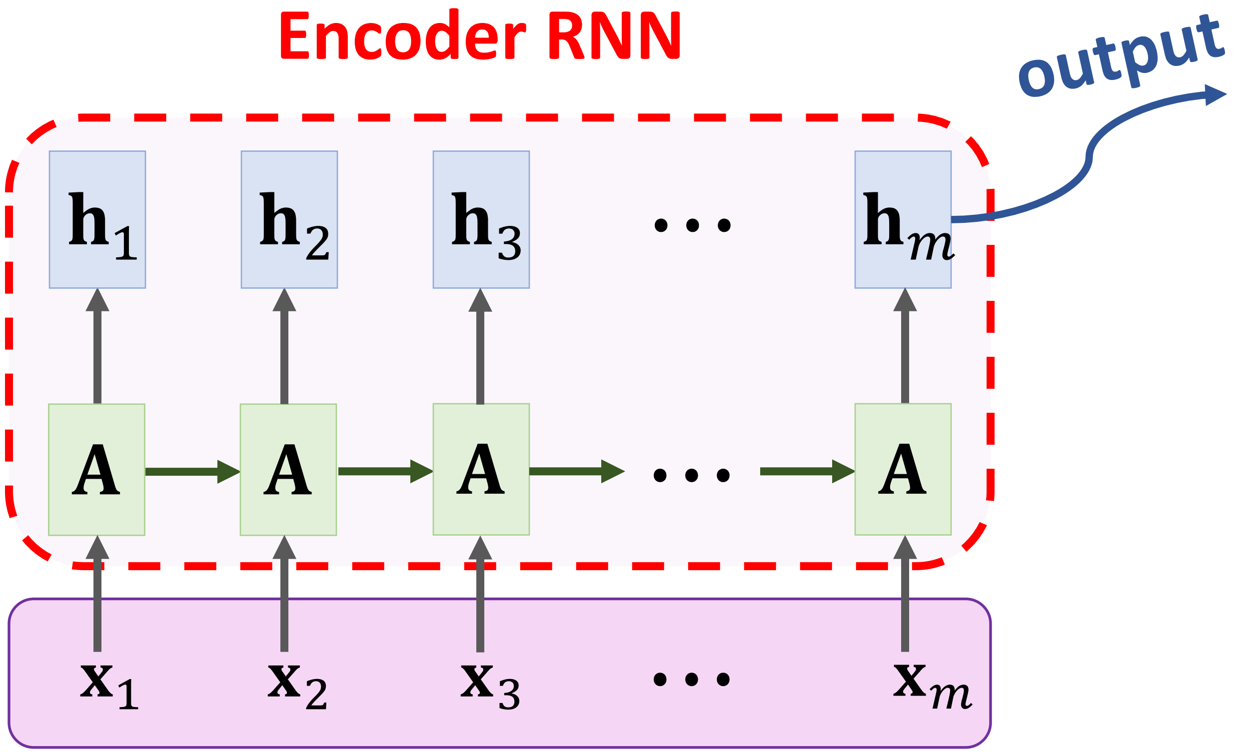


Seq2Seq Model

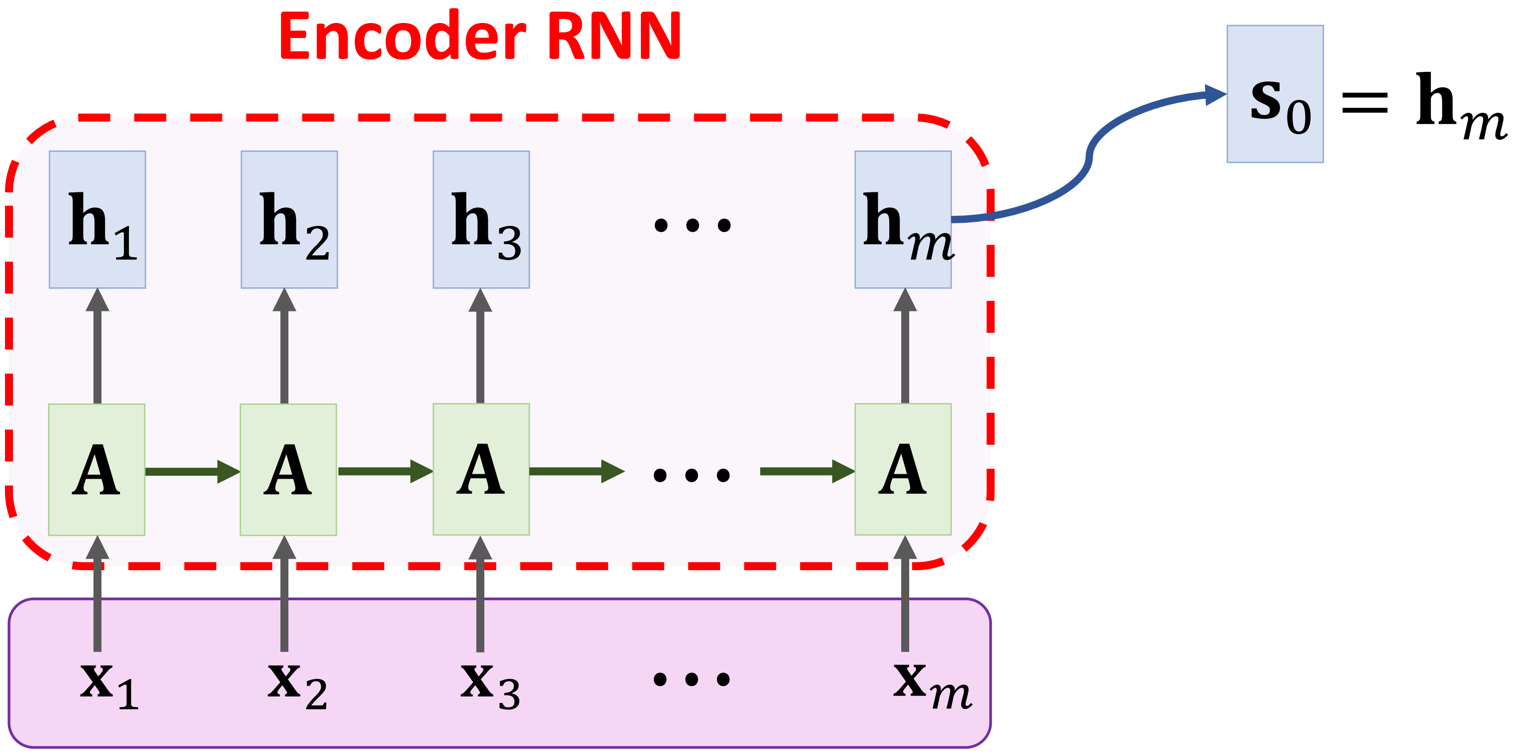
Encoder RNN



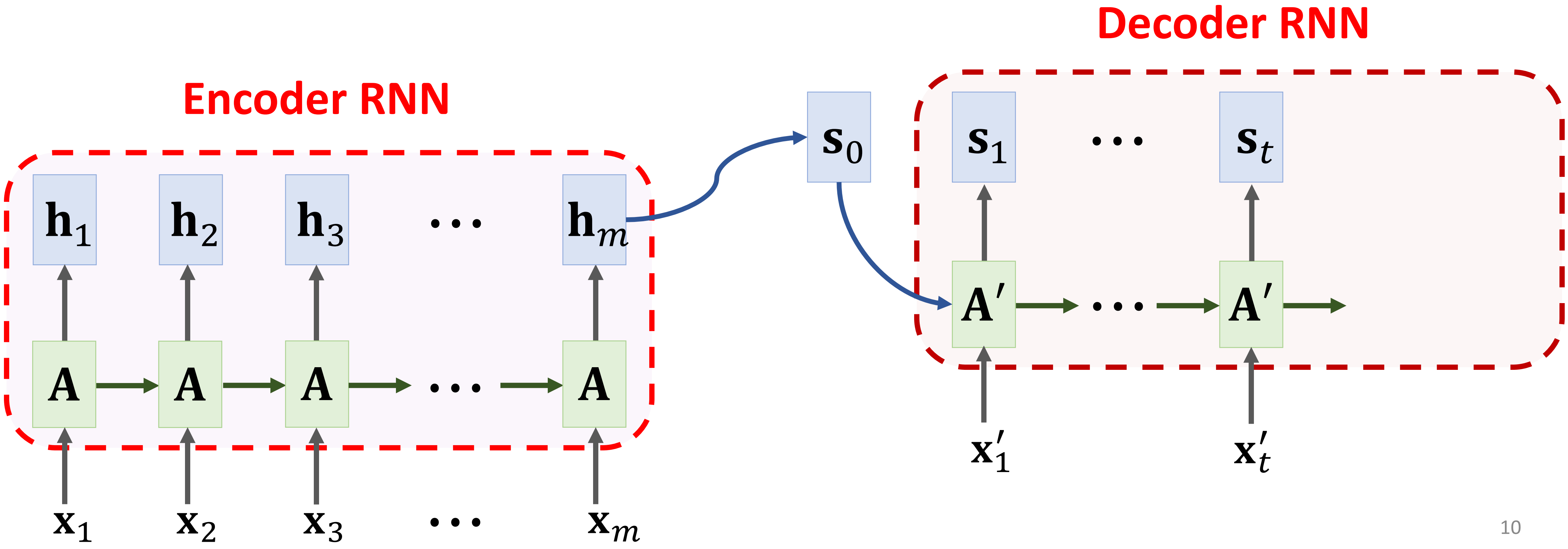
Seq2Seq Model



Seq2Seq Model

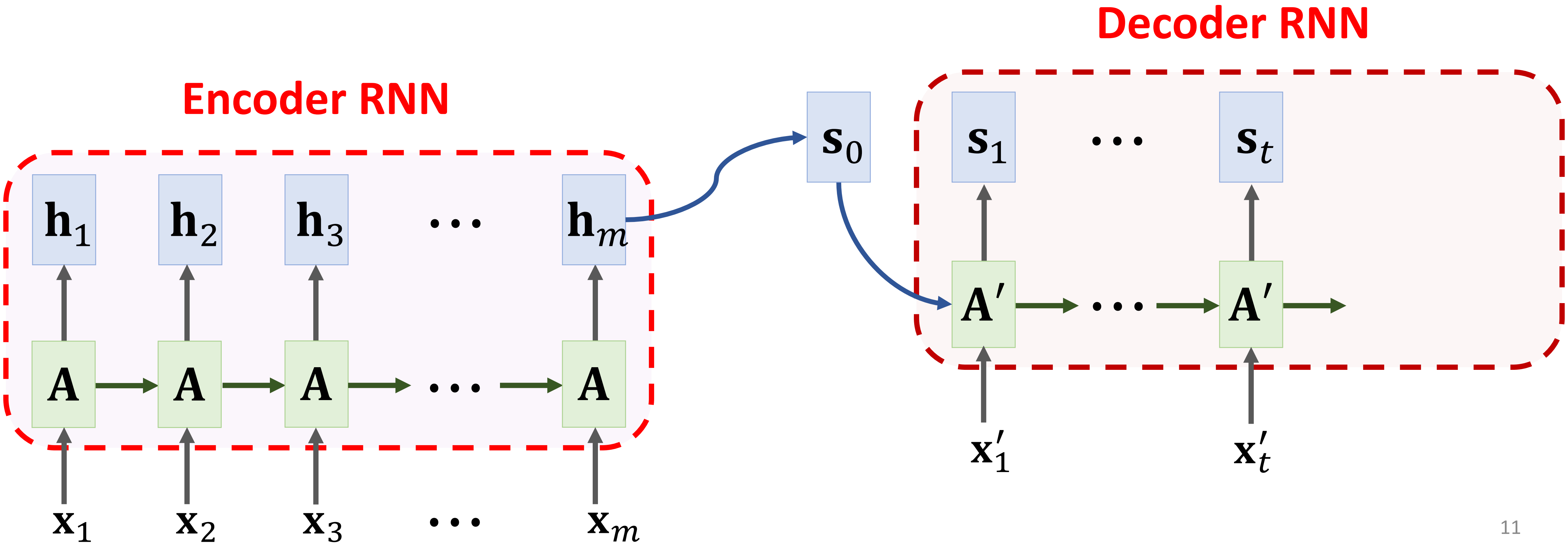


Seq2Seq Model



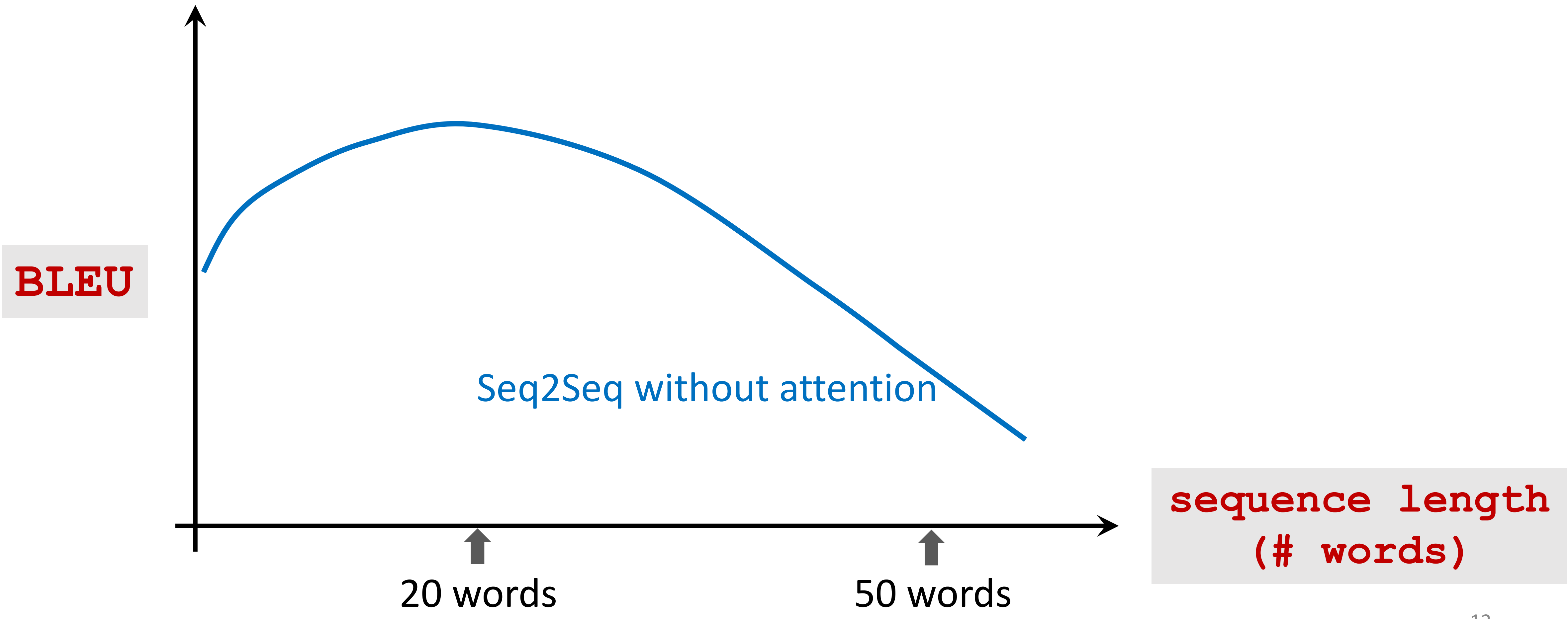
Seq2Seq Model

Shortcoming: The final state is incapable of remembering a **long** sequence.



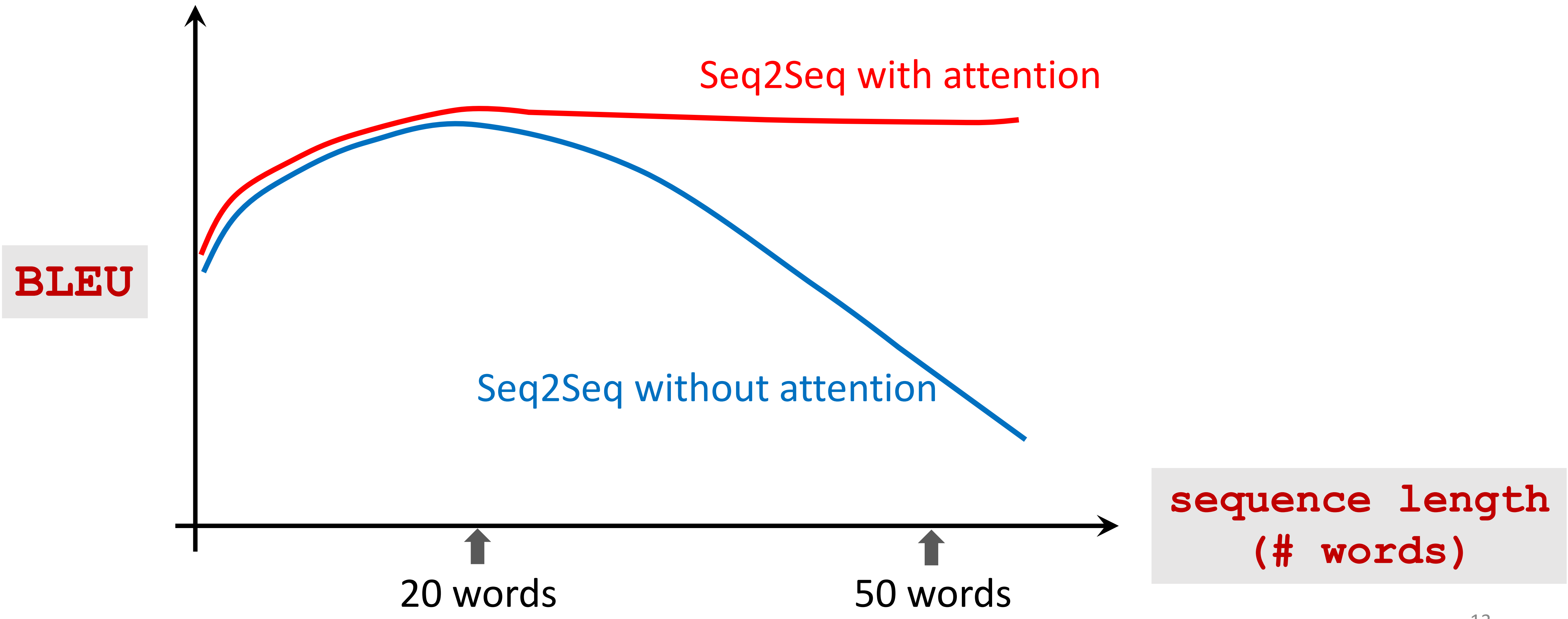
Seq2Seq Model

Shortcoming: The final state is incapable of remembering a **long** sequence.



Seq2Seq Model

Shortcoming: The final state is incapable of remembering a **long** sequence.



Attention for Seq2Seq Model

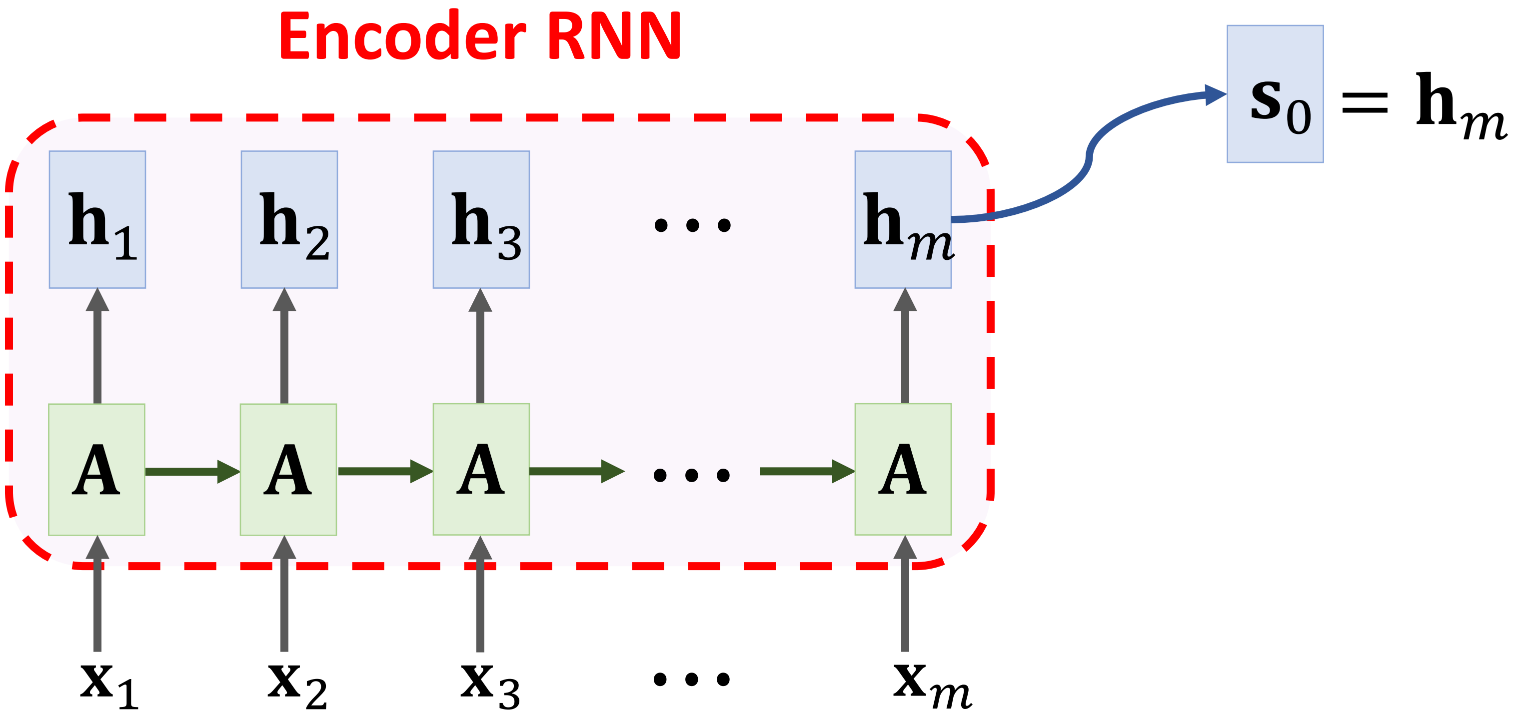
Reference

- Bahdanau, Cho, & Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

Seq2Seq Model with Attention

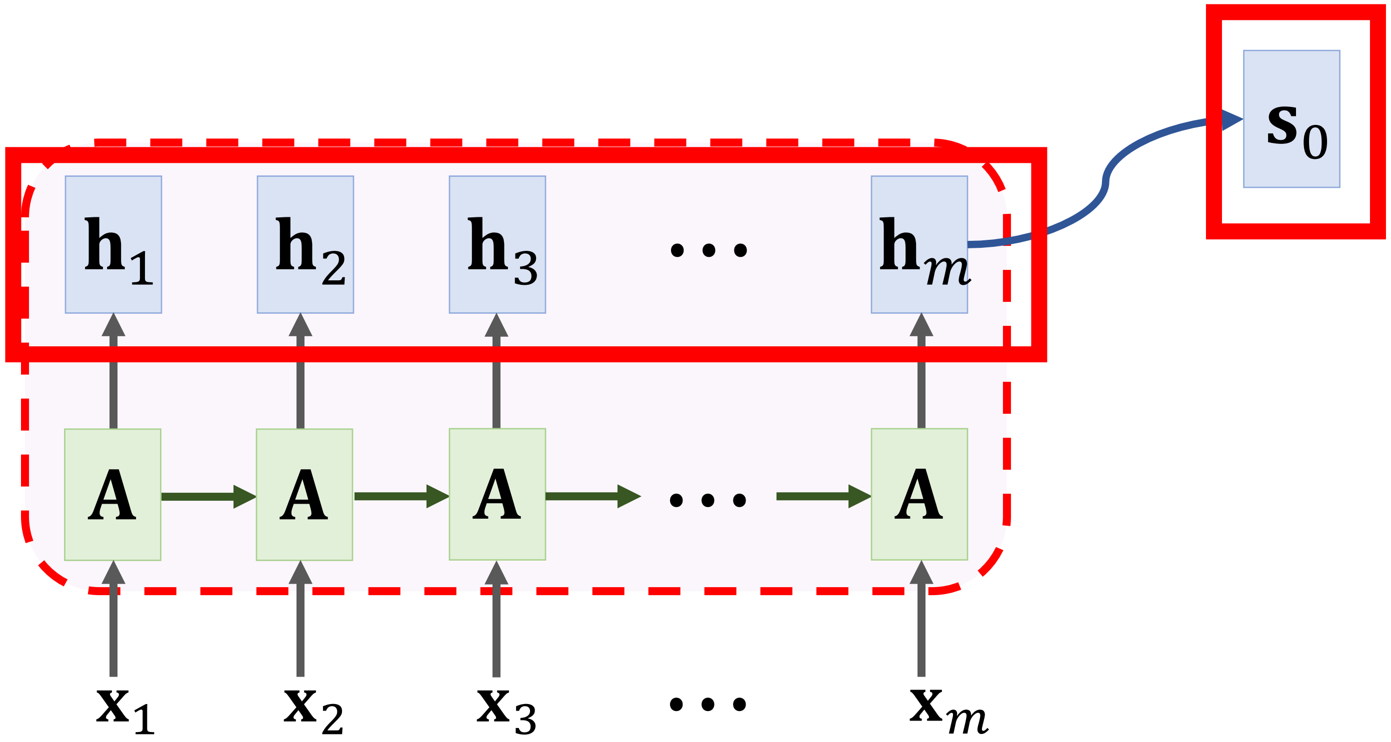
- Attention significantly improves RNN Seq2Seq model.
- With attention, RNN Seq2Seq model does not forget source input.
- With attention, the decoder knows where to focus.
- Downside: much more computation.

SimpleRNN + Attention



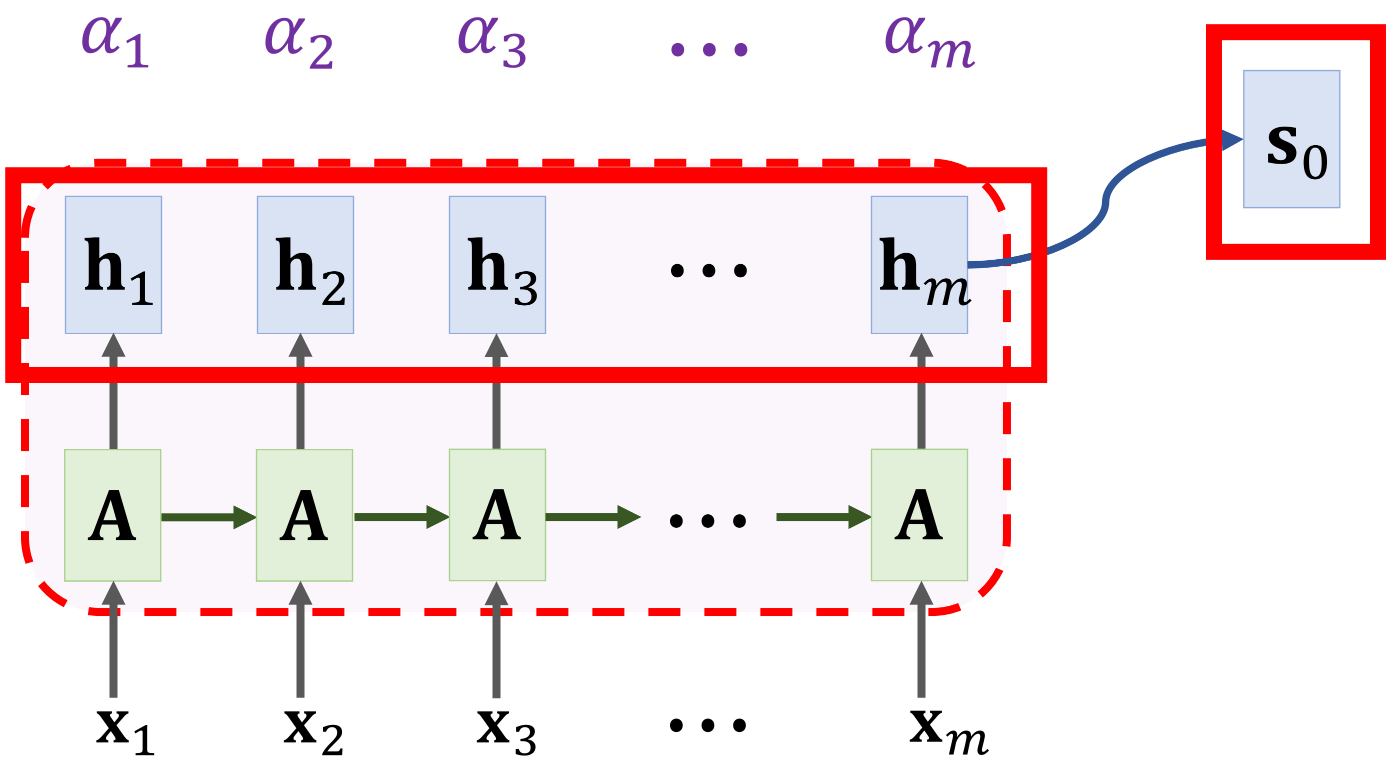
SimpleRNN + Attention

Weight: $\alpha_i = \text{align}(\mathbf{h}_i, \mathbf{s}_0)$.



SimpleRNN + Attention

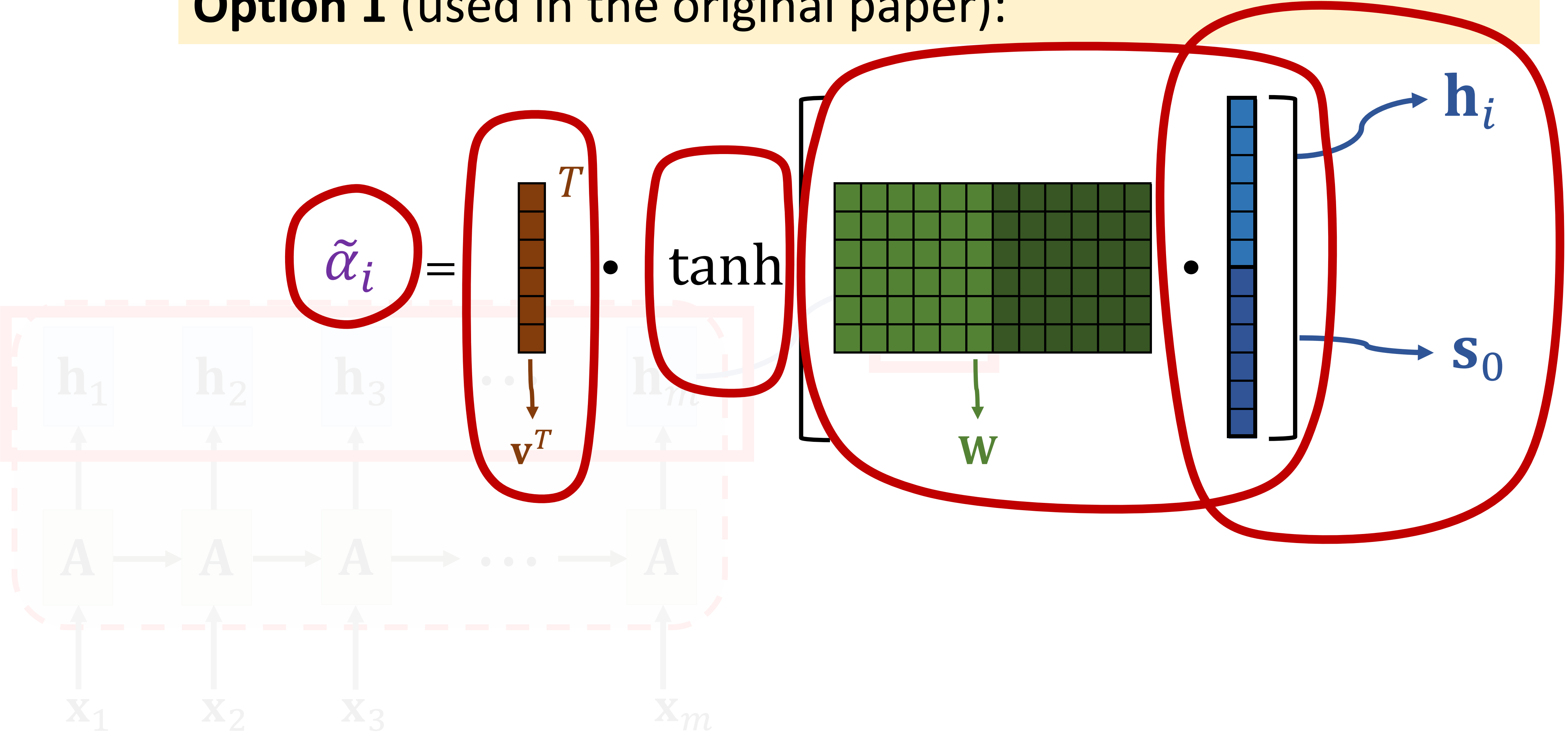
Weight: $\alpha_i = \text{align}(\mathbf{h}_i, \mathbf{s}_0)$.



SimpleRNN + Attention

Weight: $\alpha_i = \text{align}(\mathbf{h}_i, \mathbf{s}_0)$.

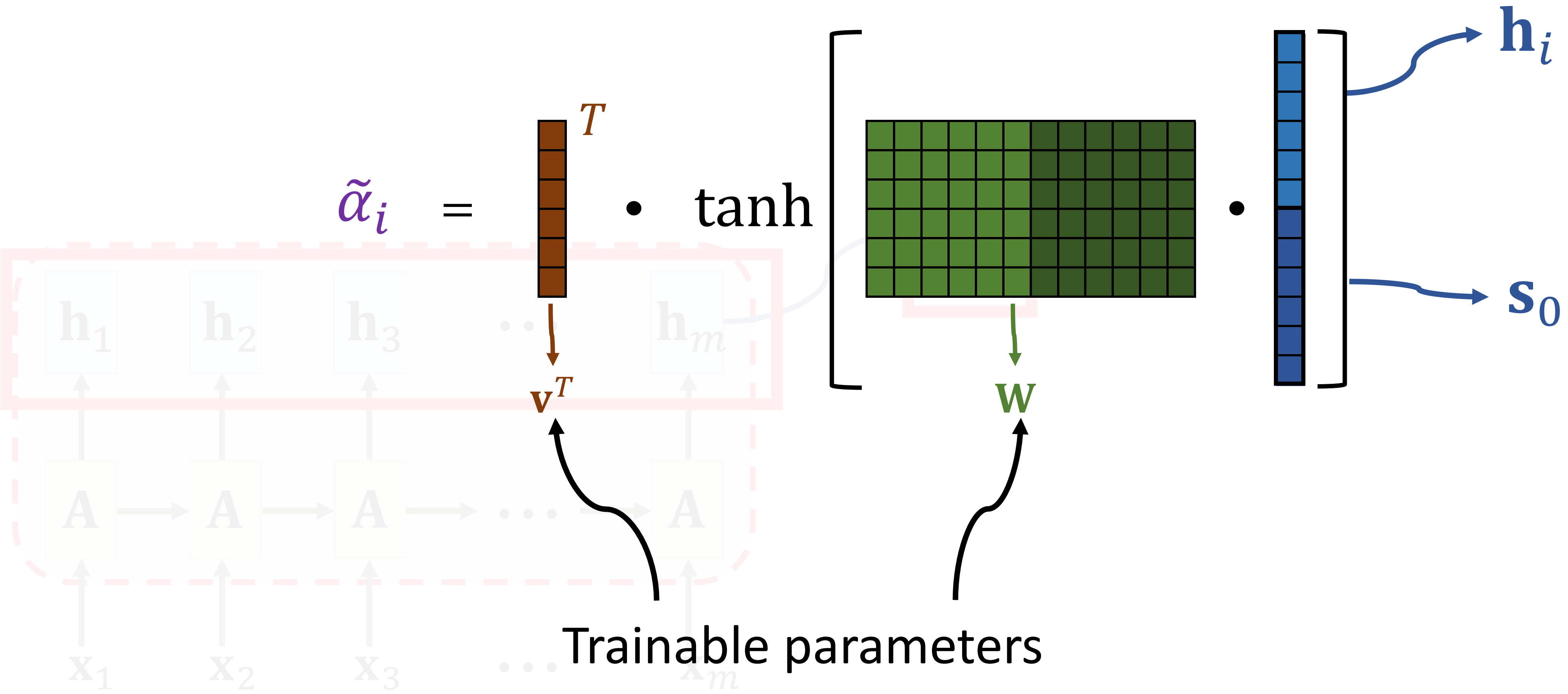
Option 1 (used in the original paper):



SimpleRNN + Attention

Weight: $\alpha_i = \text{align}(\mathbf{h}_i, \mathbf{s}_0)$.

Option 1 (used in the original paper):



SimpleRNN + Attention

$\boxed{\textbf{Weight:} \quad \alpha_i = \mathrm{align}(\mathbf{h}_i, \mathbf{s}_0).}$

Option 1 (used in the original paper):

$$\tilde{\alpha}_i = \mathbf{v}^T \cdot \tanh\left[\mathbf{W} \cdot \begin{bmatrix} \mathbf{h}_i \\ \mathbf{s}_0 \end{bmatrix} \right]$$

Then $\boxed{\textbf{normalize } \tilde{\alpha}_1, \cdots, \tilde{\alpha}_m}$ (so that they add up to 1):

$$\boxed{[\alpha_1, \cdots, \alpha_m]} = \mathrm{Softmax}\left([\tilde{\alpha}_1, \cdots, \tilde{\alpha}_m]\right).$$

SimpleRNN + Attention

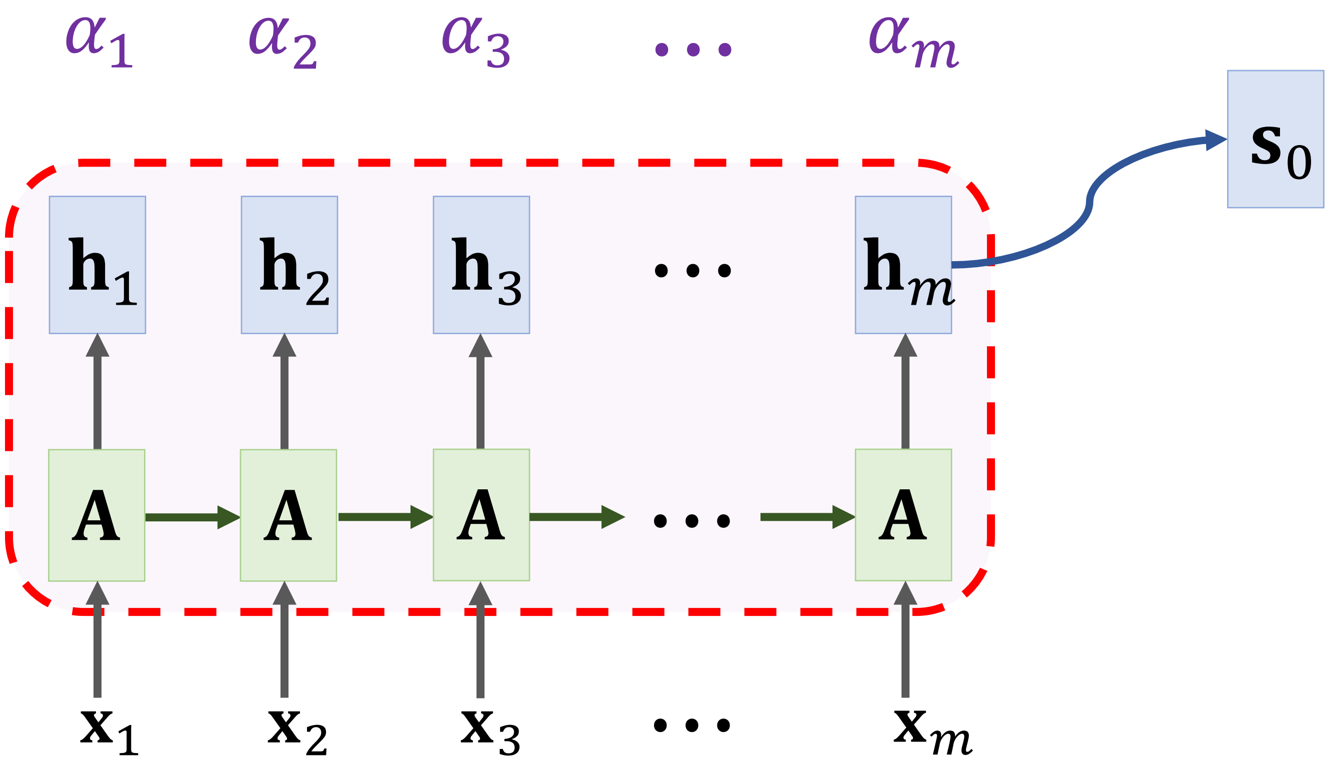
Weight: $\alpha_i = \text{align}(\mathbf{h}_i, \mathbf{s}_0)$.

Option 2 (more popular; the same to Transformer):

1. Linear maps:
   - $\mathbf{k}_i = \mathbf{W}_K \cdot \mathbf{h}_i$, for $i = 1$ to m.
   - $\mathbf{q}_0 = \mathbf{W}_Q \cdot \mathbf{s}_0$.
2. Inner product:
   - $\tilde{\alpha}_i = \mathbf{k}_i^T \mathbf{q}_0$, for $i = 1$ to m.
3. Normalization:
   - $[\alpha_1, \cdots, \alpha_m] = \text{Softmax}\left([\tilde{\alpha}_1, \cdots, \tilde{\alpha}_m]\right)$.

SimpleRNN + Attention

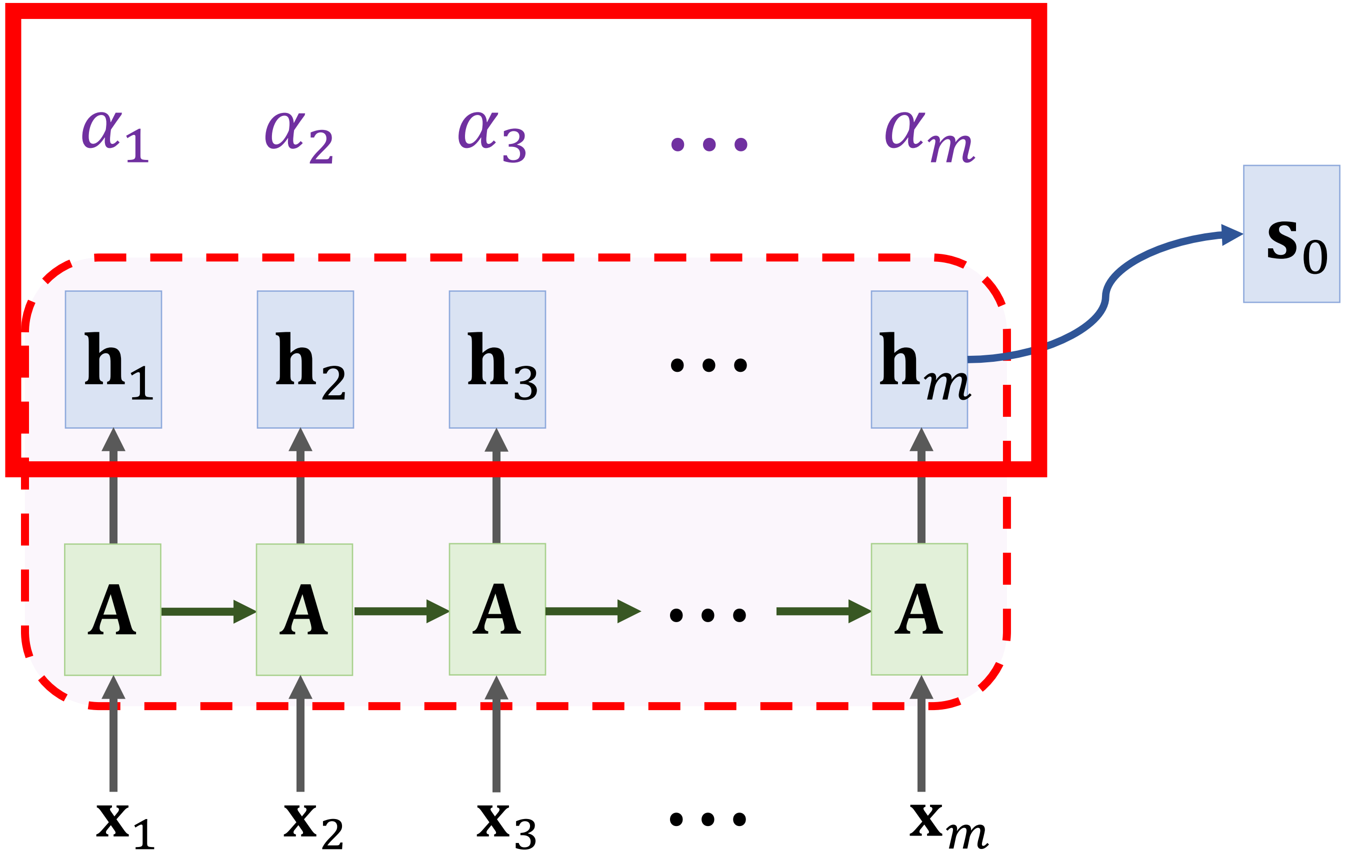
Weight: $\alpha_i = \text{align}(\mathbf{h}_i, \mathbf{s}_0)$.



SimpleRNN + Attention

Weight: $\alpha_i = \mathrm{align}(\mathbf{h}_i, \mathbf{s}_0).$

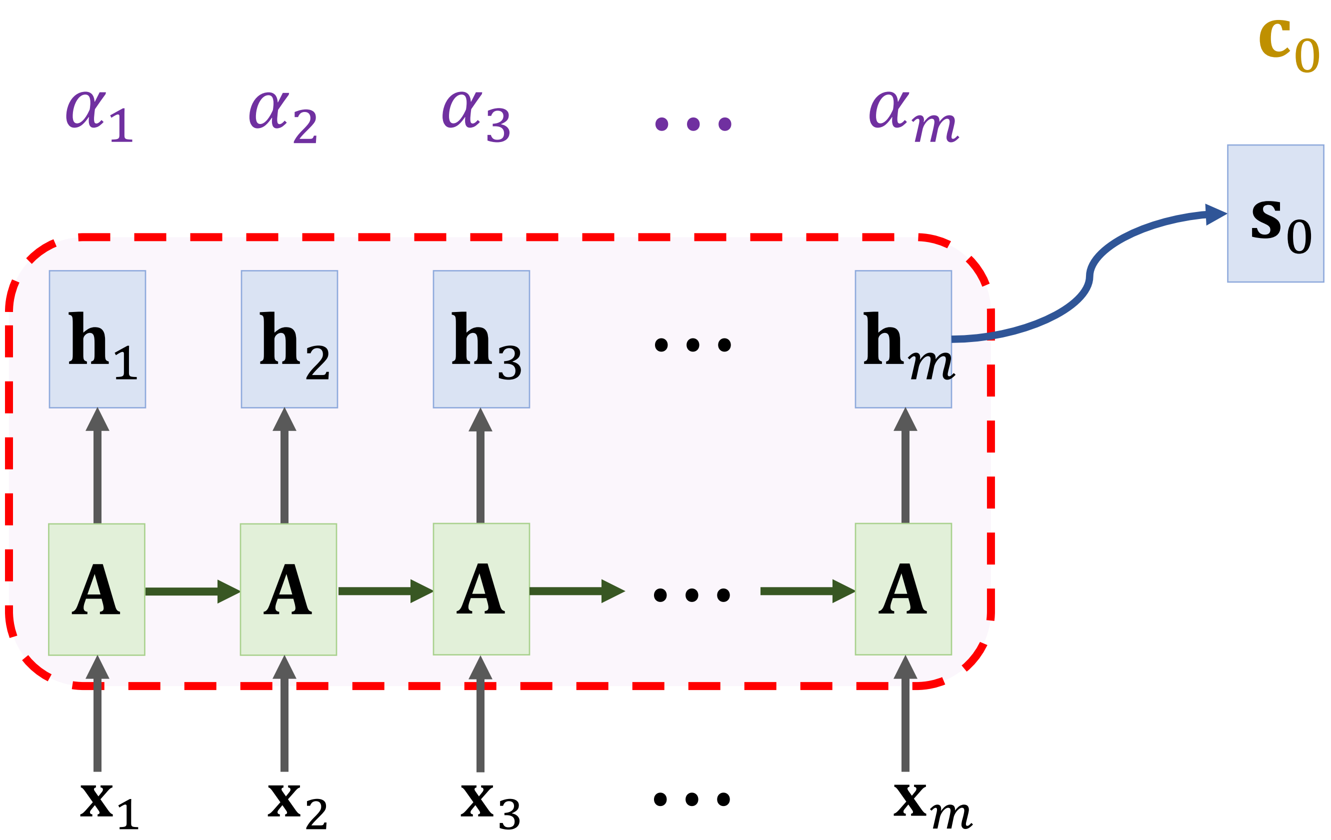
Context vector: $\mathbf{c}_0 = \alpha_1 \mathbf{h}_1 + \cdots + \alpha_m \mathbf{h}_m.$



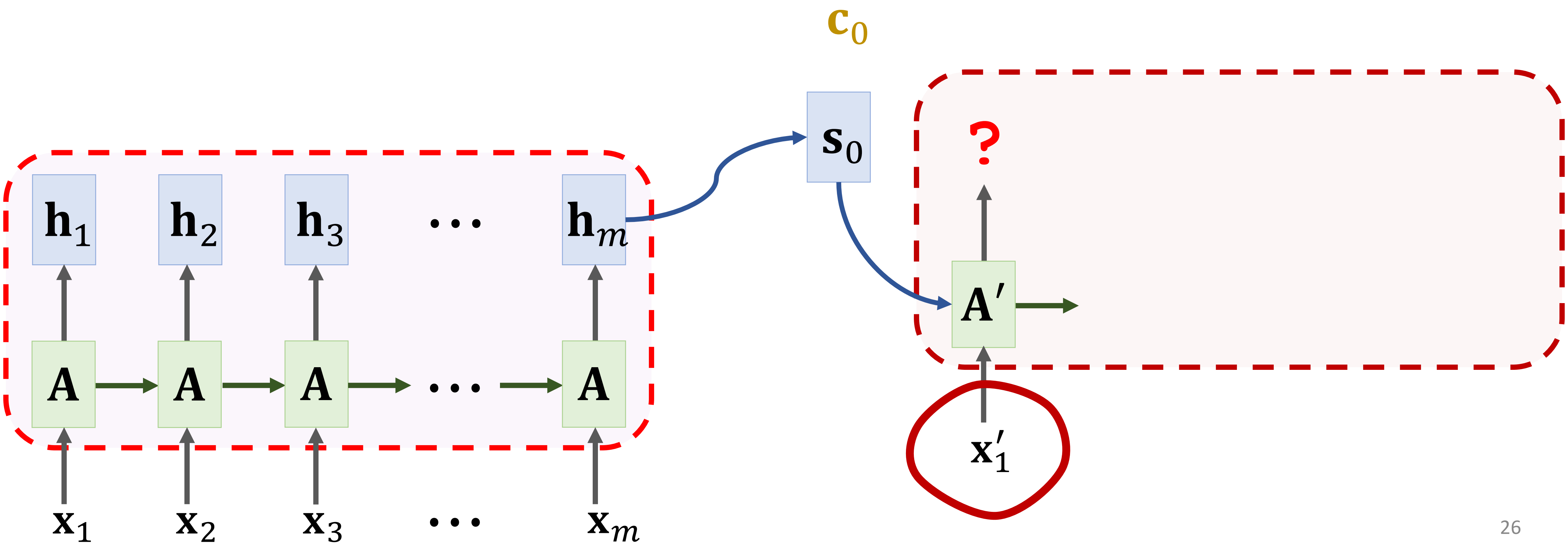
SimpleRNN + Attention

Weight: $\alpha_i = \text{align}(\mathbf{h}_i, \mathbf{s}_0)$.

Context vector: $\mathbf{c}_0 = \alpha_1 \mathbf{h}_1 + \cdots + \alpha_m \mathbf{h}_m$.



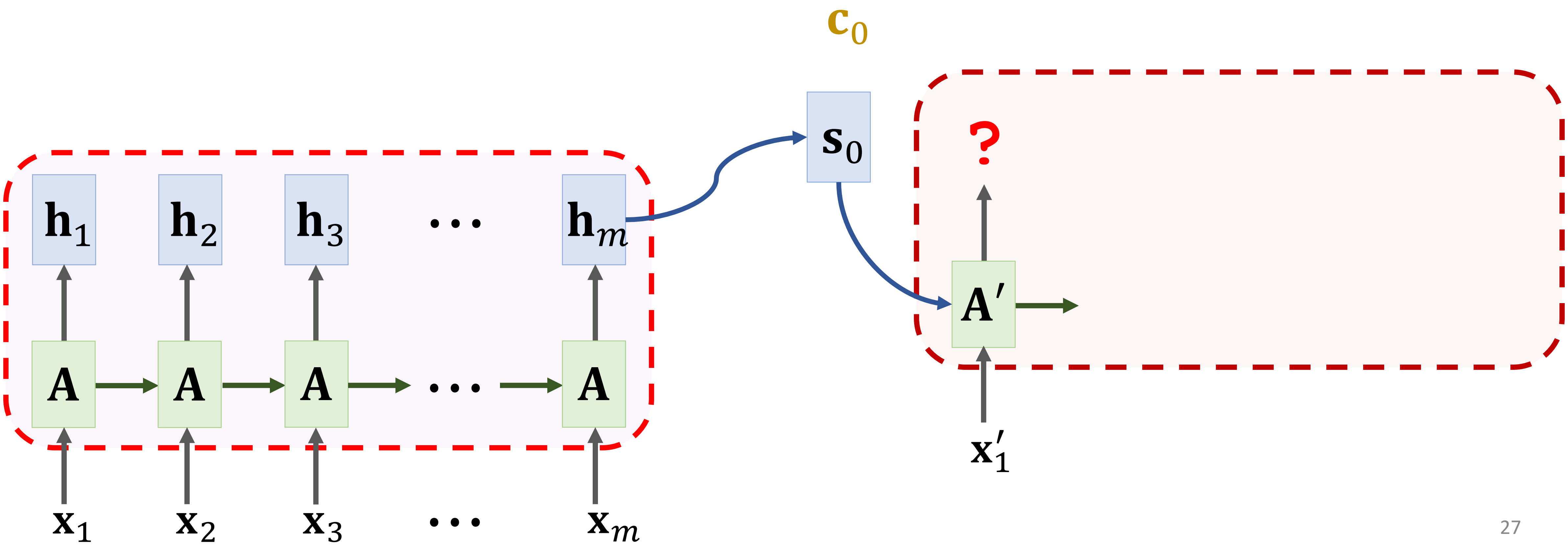
SimpleRNN + Attention



SimpleRNN

SimpleRNN:

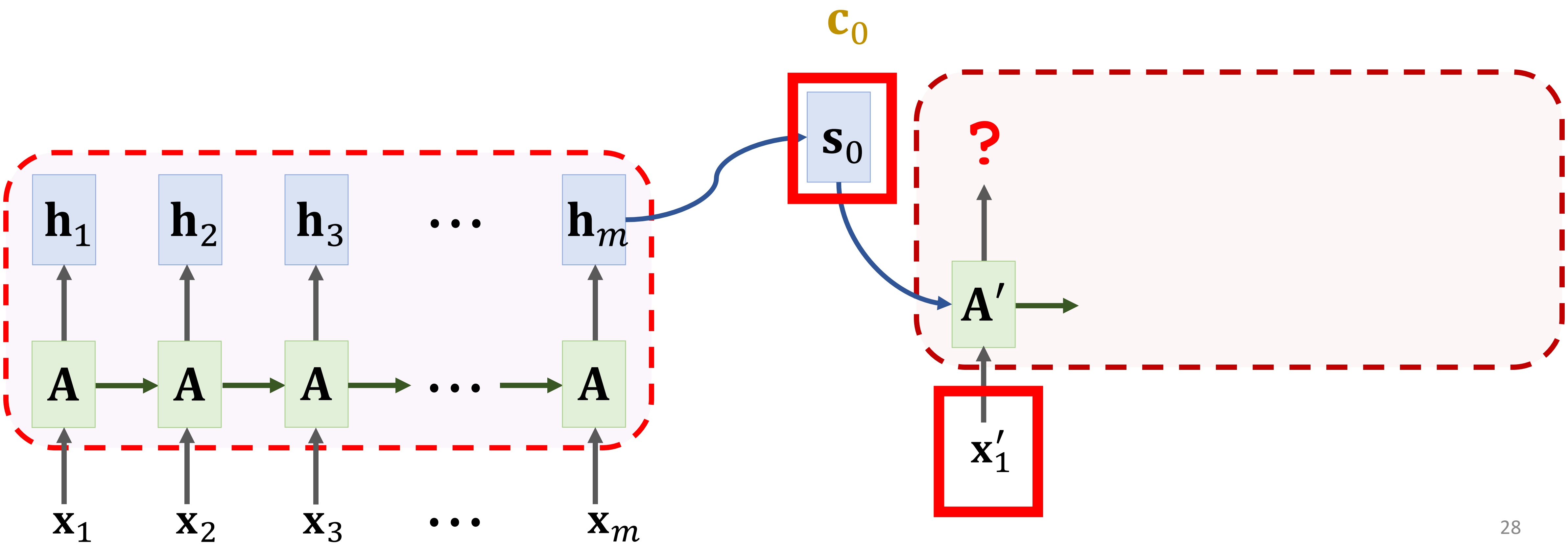
$$\boxed{\mathbf{s}_1} = \tanh\left(\mathbf{A}' \boxed{\begin{bmatrix}\mathbf{x}_1' \\ \mathbf{s}_0\end{bmatrix}} + \mathbf{b}\right)$$



SimpleRNN

SimpleRNN:

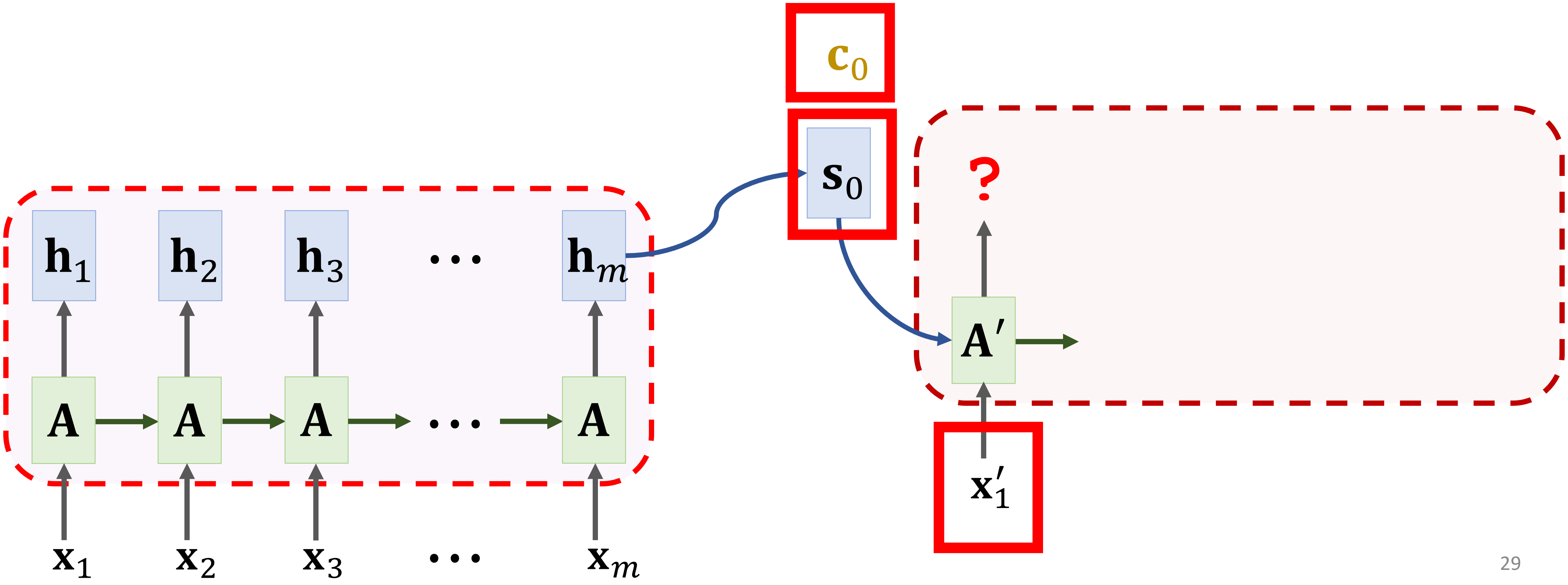
$$\mathbf{s}_1 = \tanh\left(\mathbf{A}' \cdot \begin{bmatrix} \mathbf{x}_1' \\ \mathbf{s}_0 \end{bmatrix} + \mathbf{b}\right)$$



SimpleRNN + Attention

SimpleRNN:

$$\mathbf{s}_1 = \tanh\left(\mathbf{A}' \cdot \begin{bmatrix} \mathbf{x}'_1 \\ \mathbf{s}_0 \end{bmatrix} + \mathbf{b}\right)$$



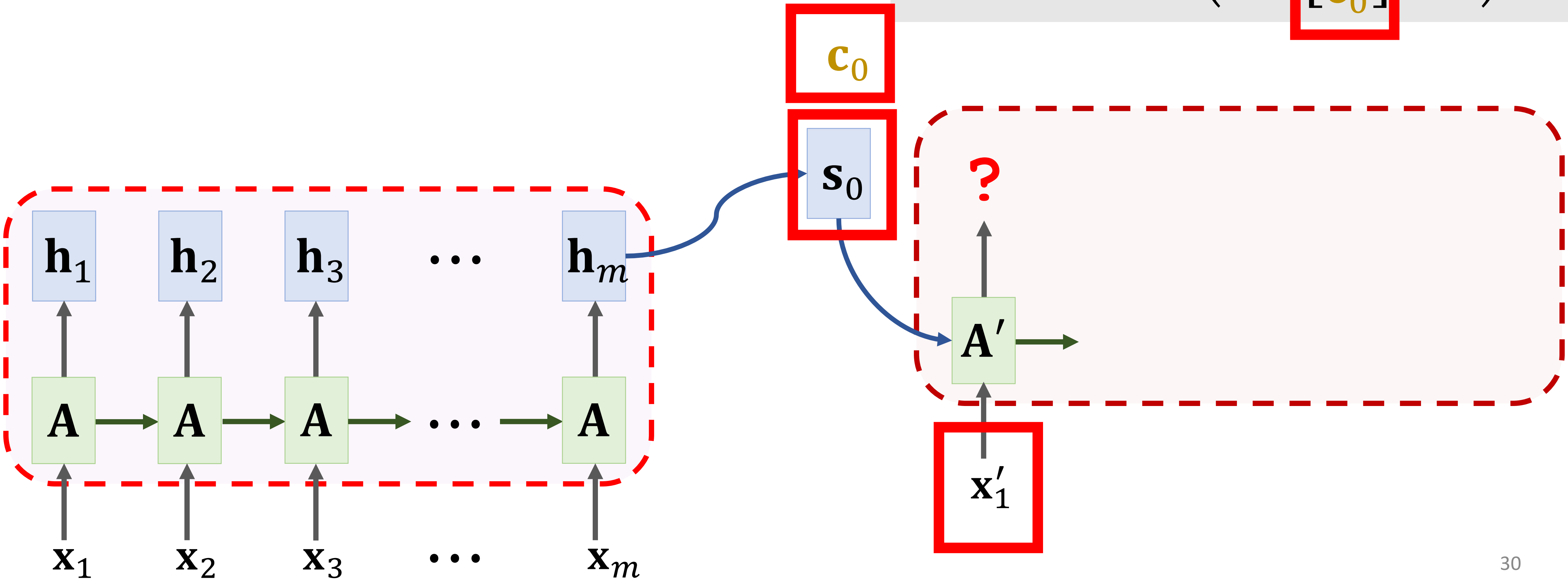
SimpleRNN + Attention

SimpleRNN:

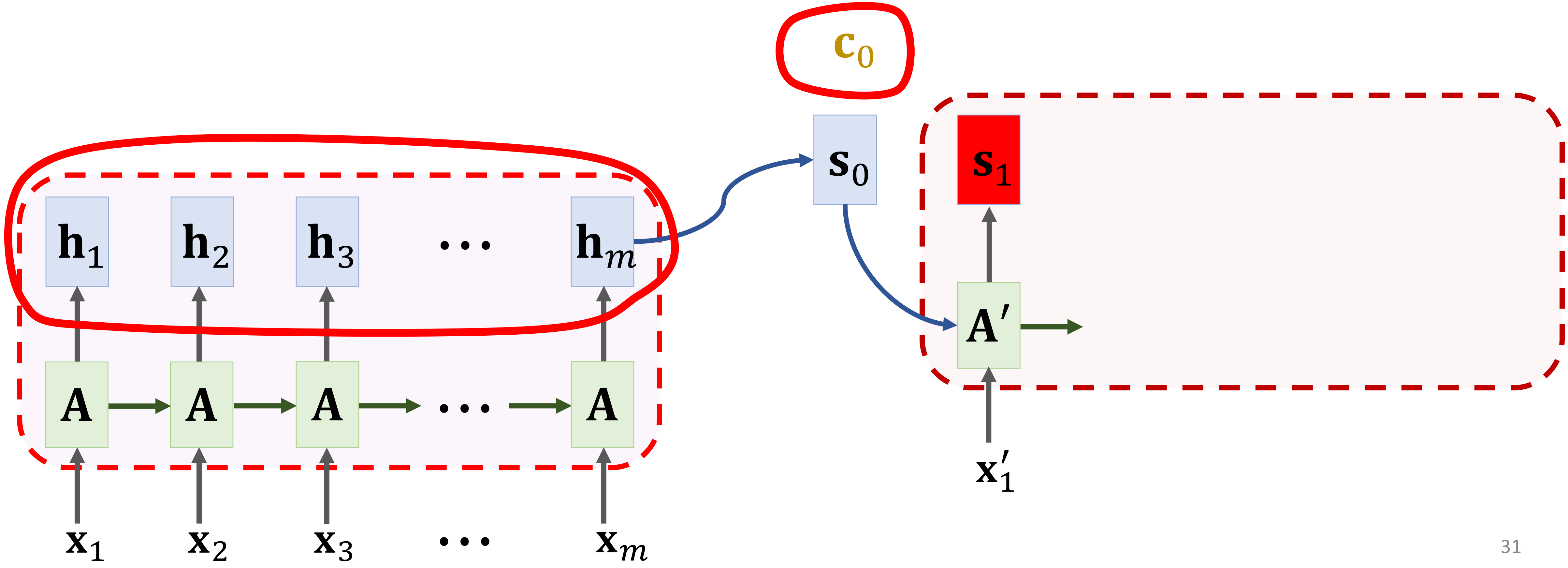
$$\mathbf{s}_1 = \tanh\left(\mathbf{A}' \cdot \begin{bmatrix} \mathbf{x}'_1 \\ \mathbf{s}_0 \end{bmatrix} + \mathbf{b}\right)$$

SimpleRNN + Attention:

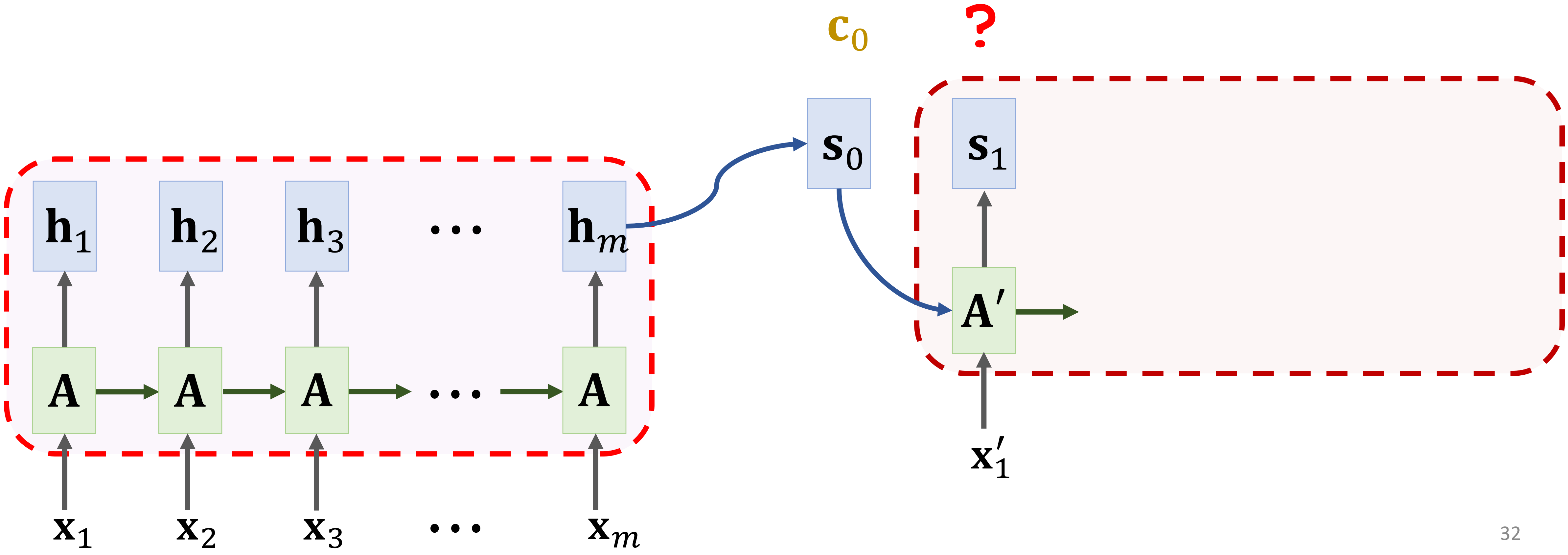
$$\boxed{\mathbf{s}_1} = \tanh\left(\mathbf{A}' \cdot \boxed{\begin{bmatrix} \mathbf{x}'_1 \\ \mathbf{s}_0 \\ \mathbf{c}_0 \end{bmatrix}} + \mathbf{b}\right)$$



SimpleRNN + Attention

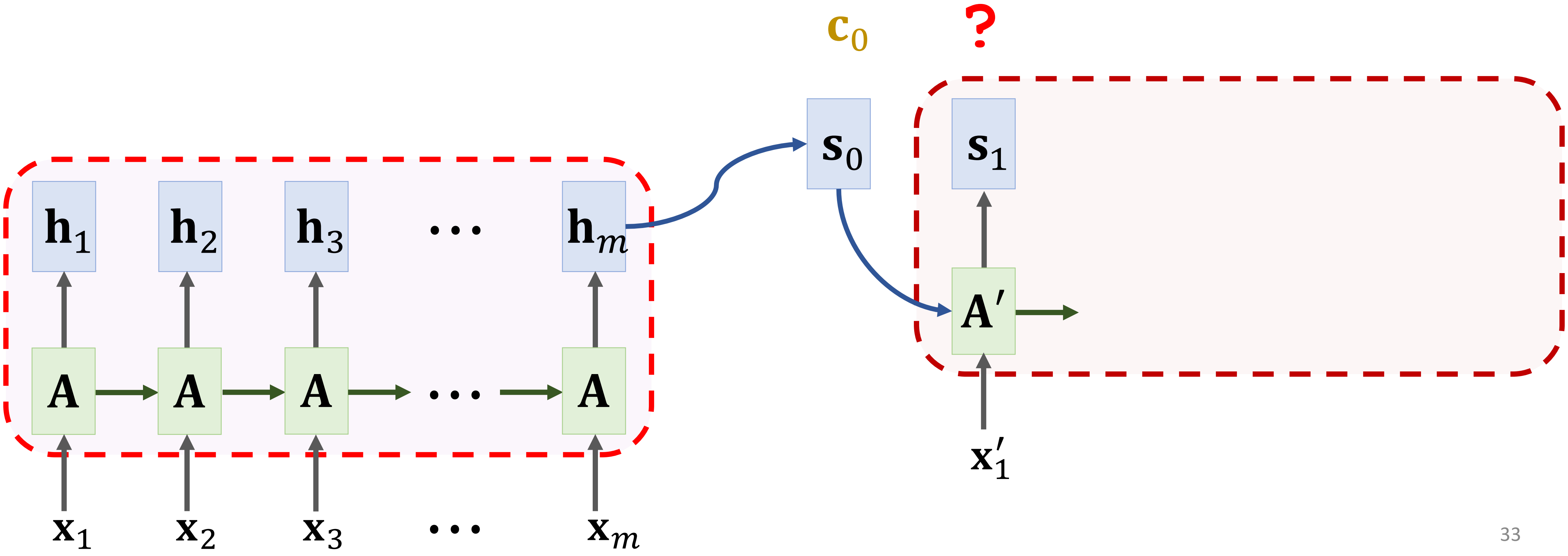


SimpleRNN + Attention



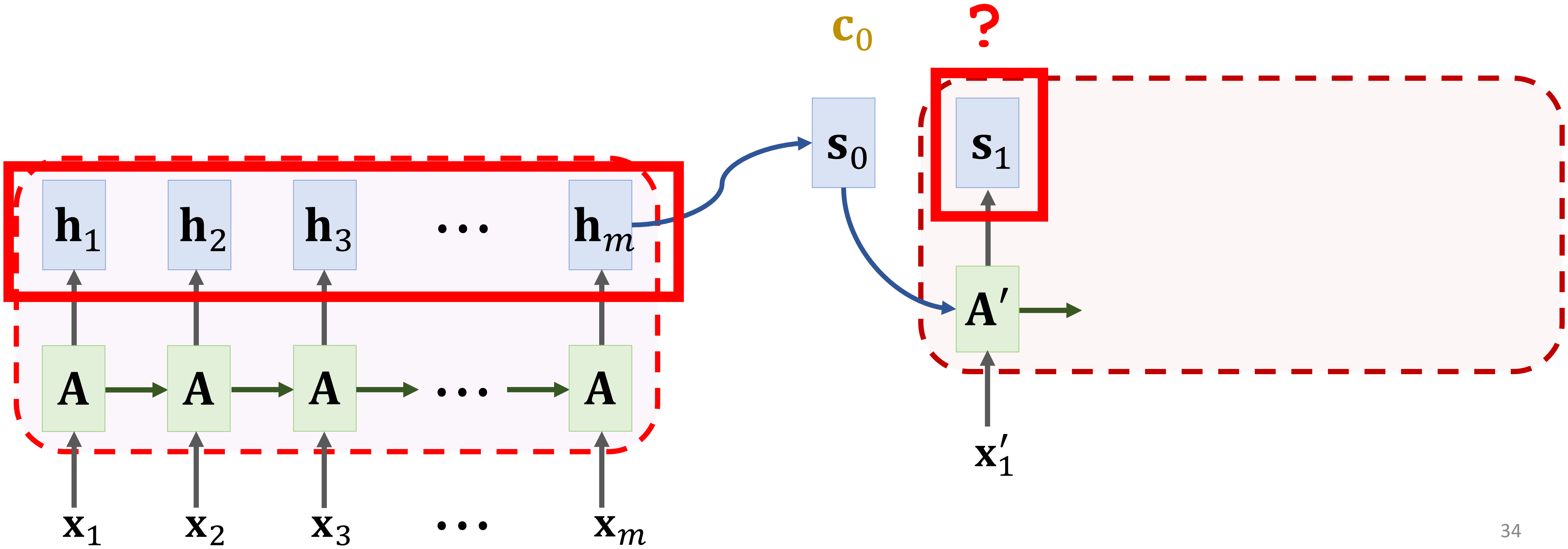
SimpleRNN + Attention

Weight: $\alpha_i = \text{align}(\mathbf{h}_i, \mathbf{s}_1)$.



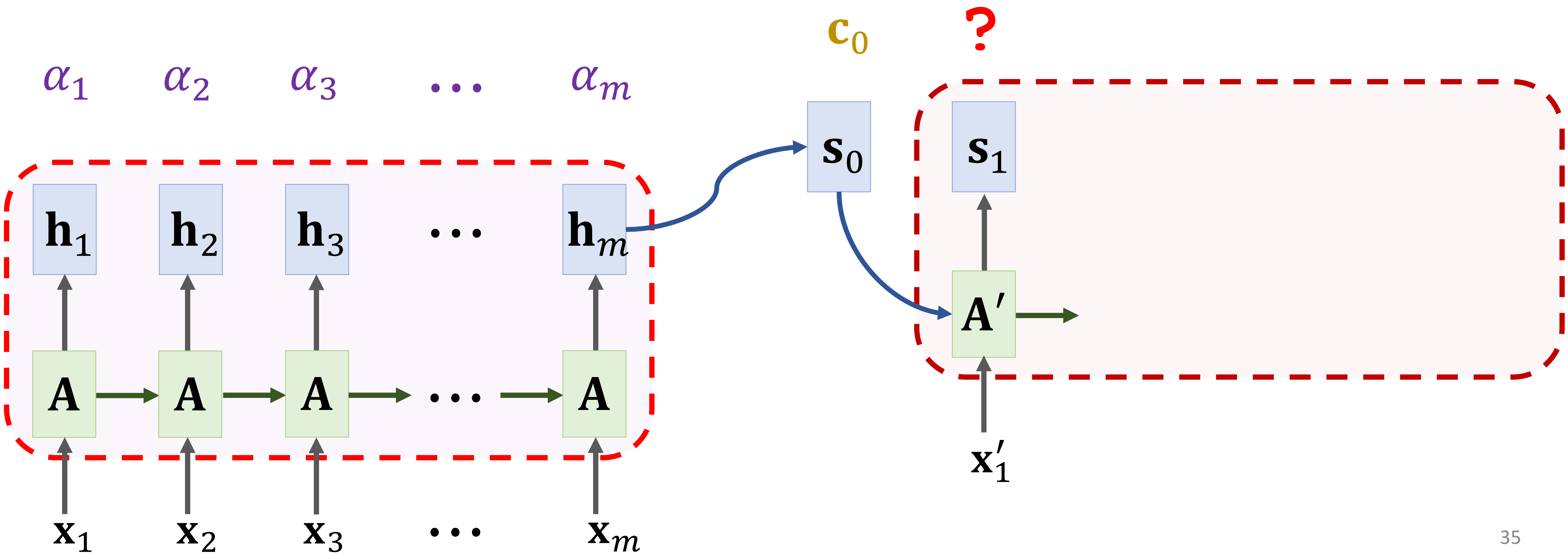
SimpleRNN + Attention

Weight: $\boxed{\alpha_i} = \text{align}(\mathbf{h}_i, \mathbf{s}_1)$.



SimpleRNN + Attention

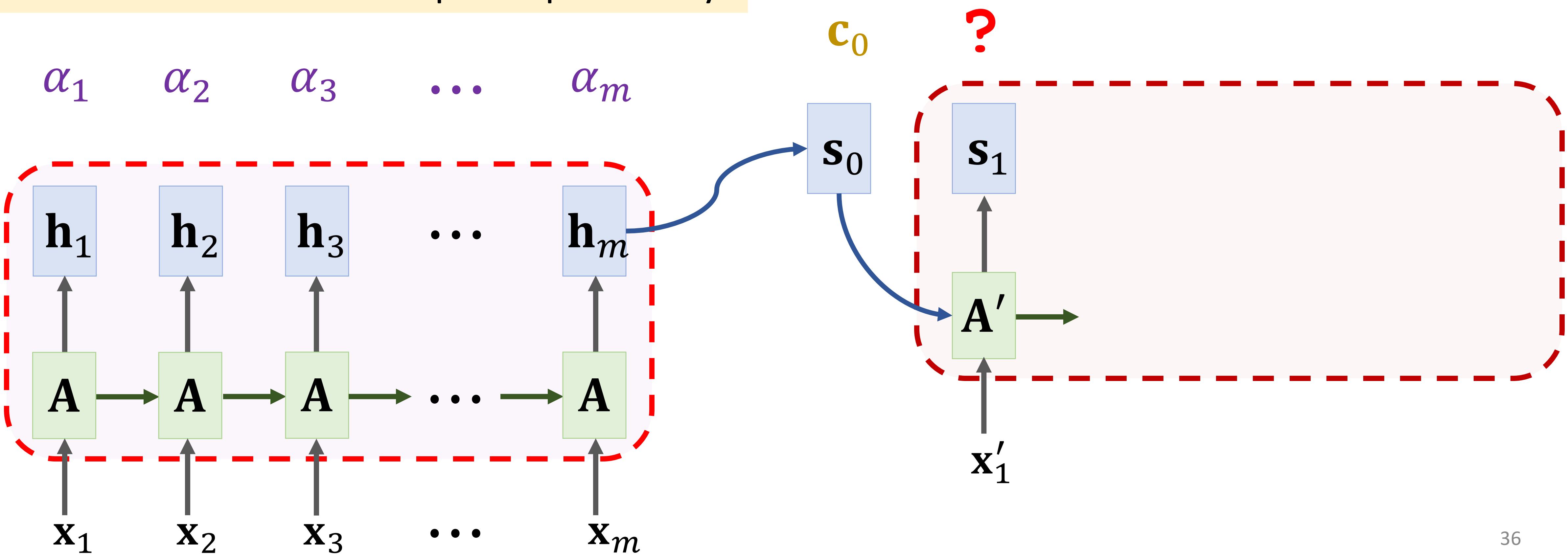
Weight: $\alpha_i = \text{align}(\mathbf{h}_i, \mathbf{s}_1)$.



SimpleRNN + Attention

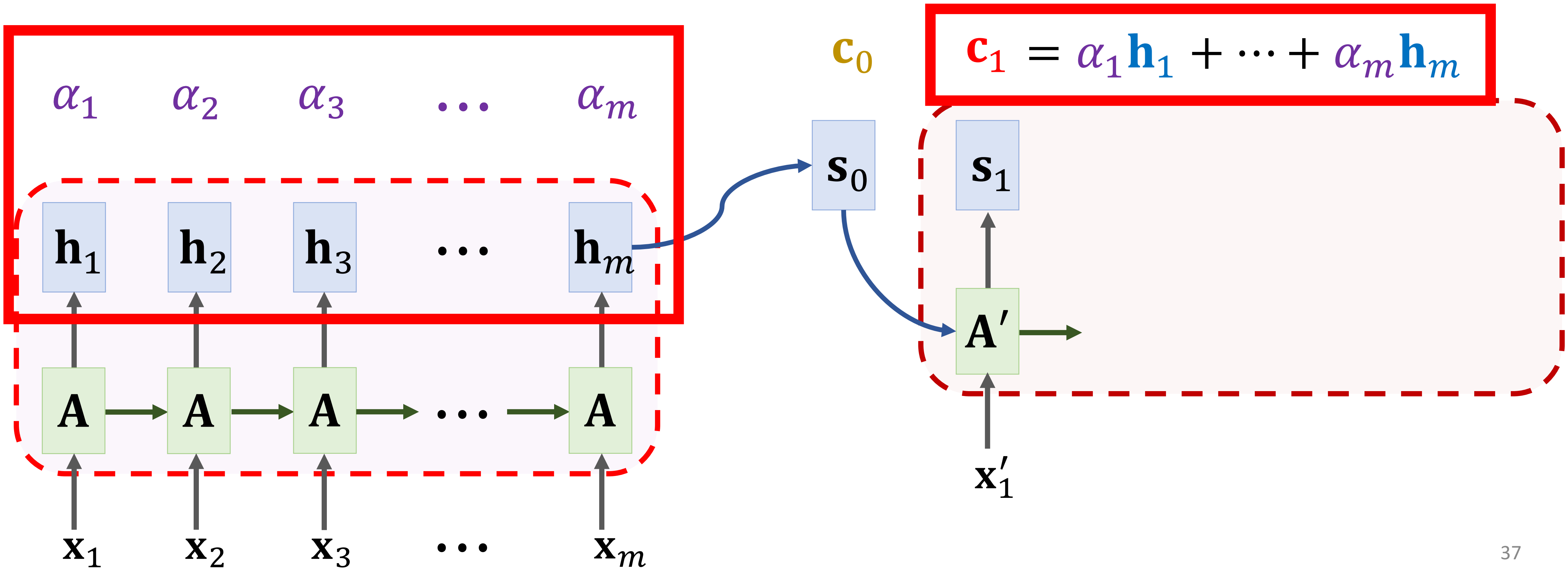
Weight: $\alpha_i = \text{align}(\mathbf{h}_i, \mathbf{s}_1)$.

Do not re-use the α's computed previously.

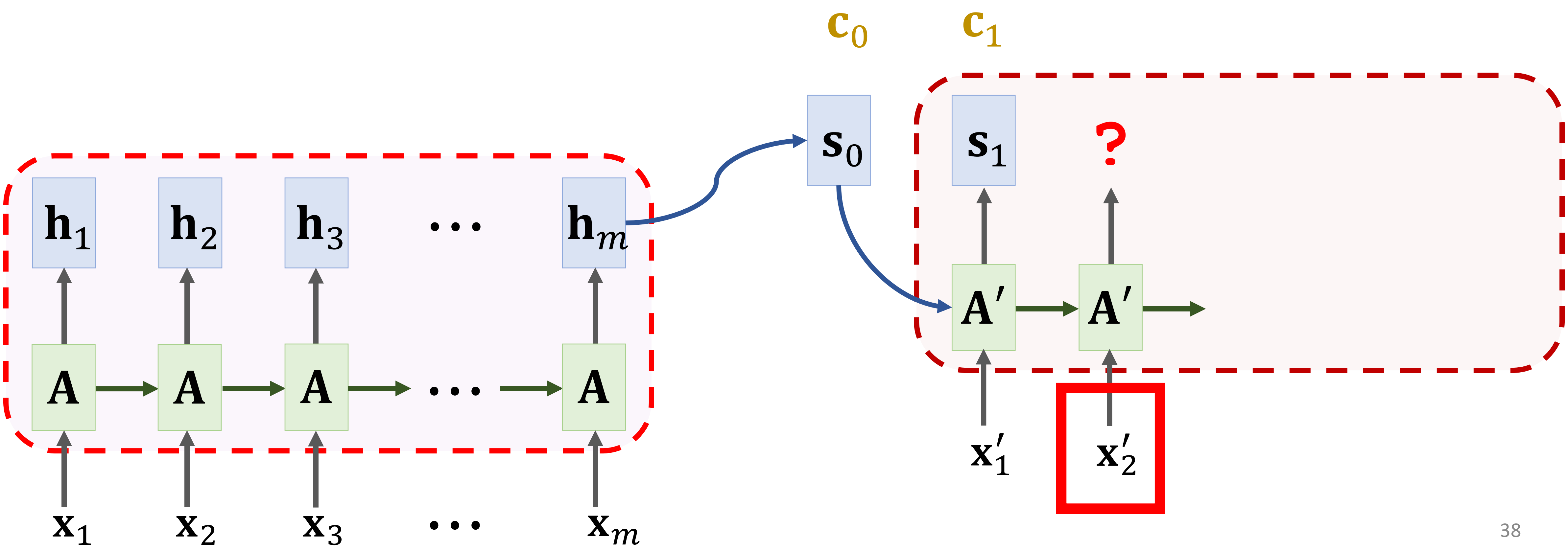


SimpleRNN + Attention

Weight: $\alpha_i = \mathrm{align}(\mathbf{h}_i, \mathbf{s}_1)$.

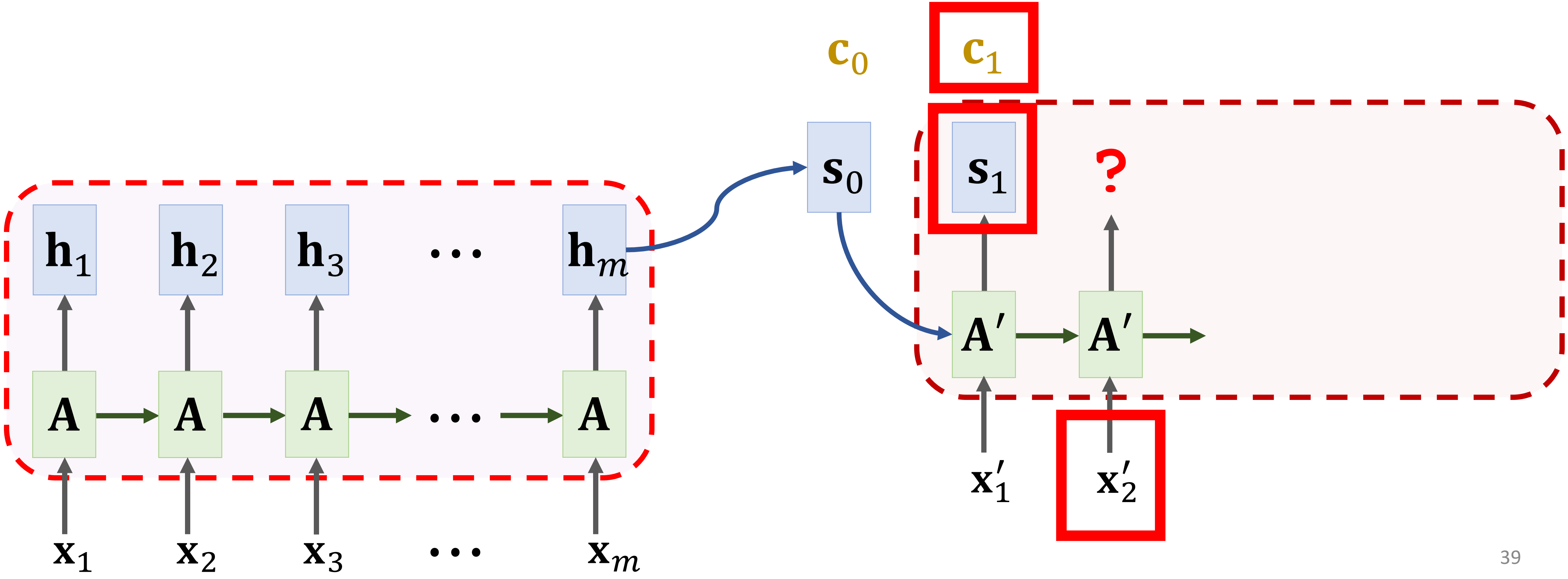


SimpleRNN + Attention

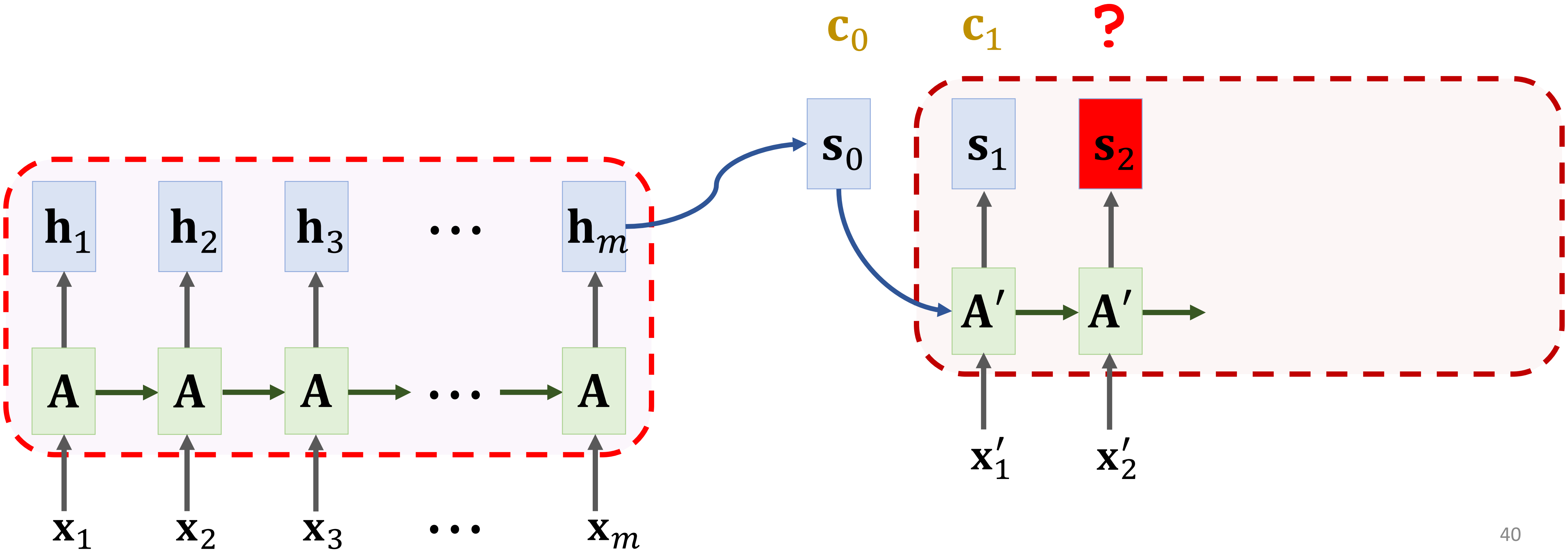


SimpleRNN + Attention

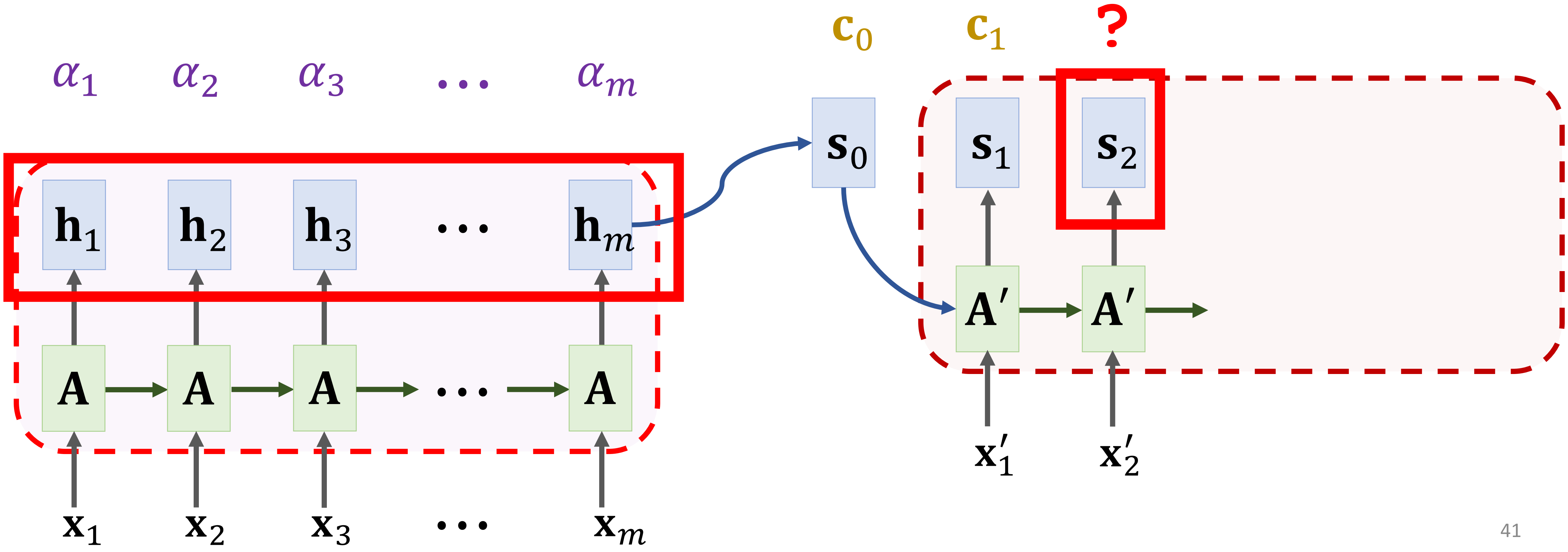
$$\mathbf{s}_2 = \tanh\left(\mathbf{A}' \cdot \begin{bmatrix} \mathbf{x}'_2 \\ \mathbf{s}_1 \\ \mathbf{c}_1 \end{bmatrix} + \mathbf{b}\right)$$



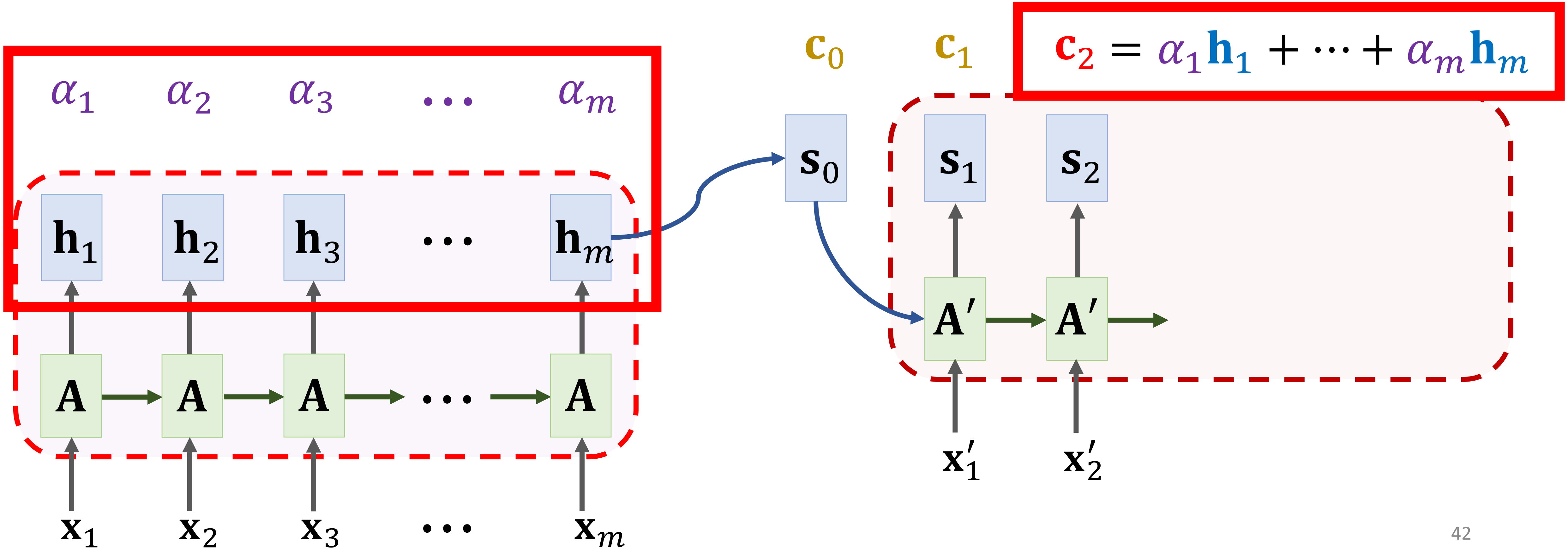
SimpleRNN + Attention



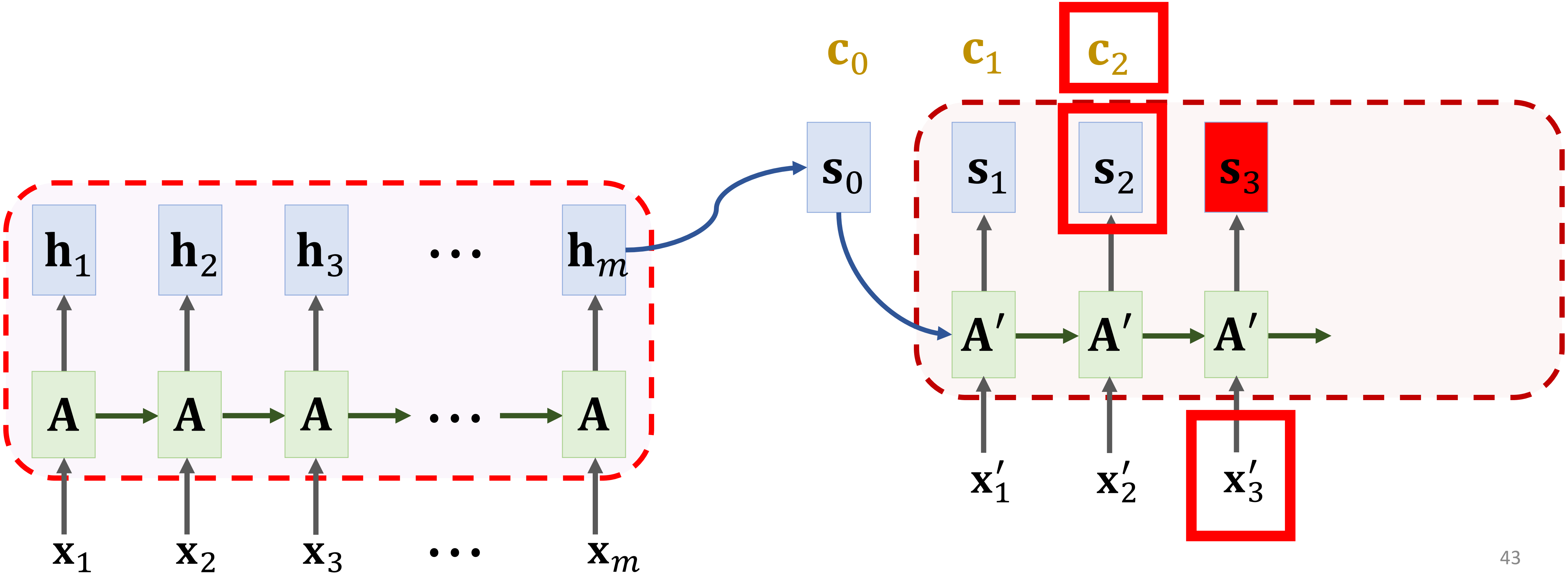
SimpleRNN + Attention



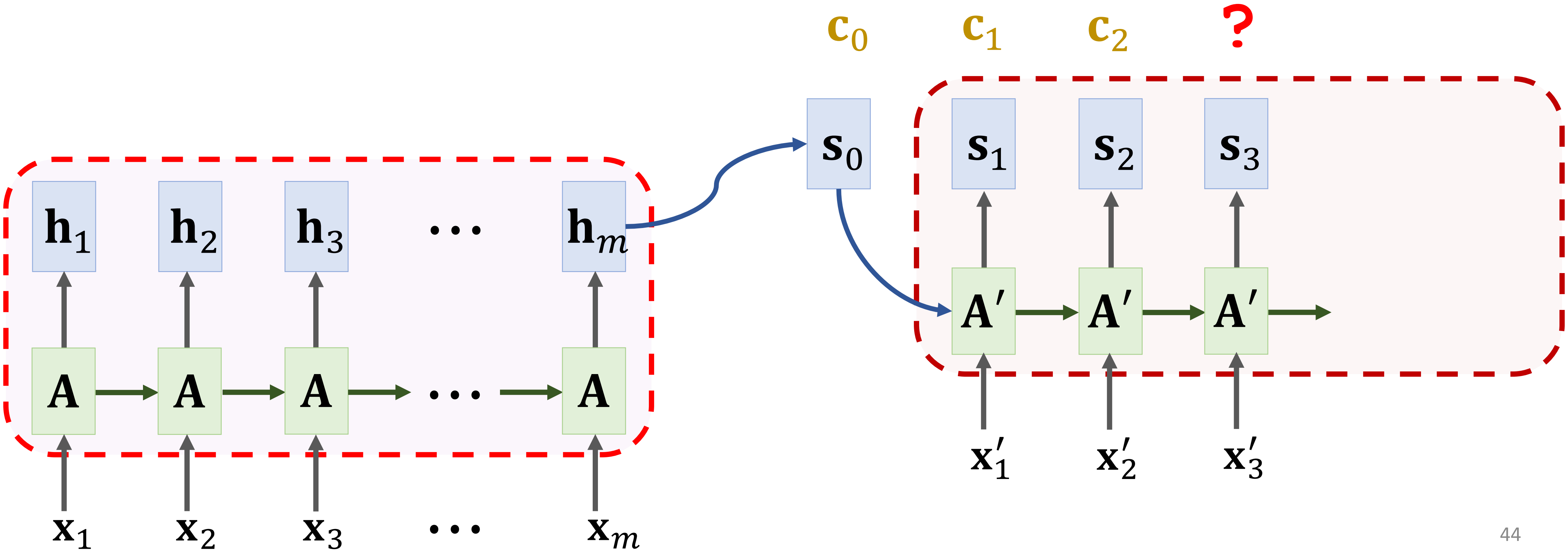
SimpleRNN + Attention



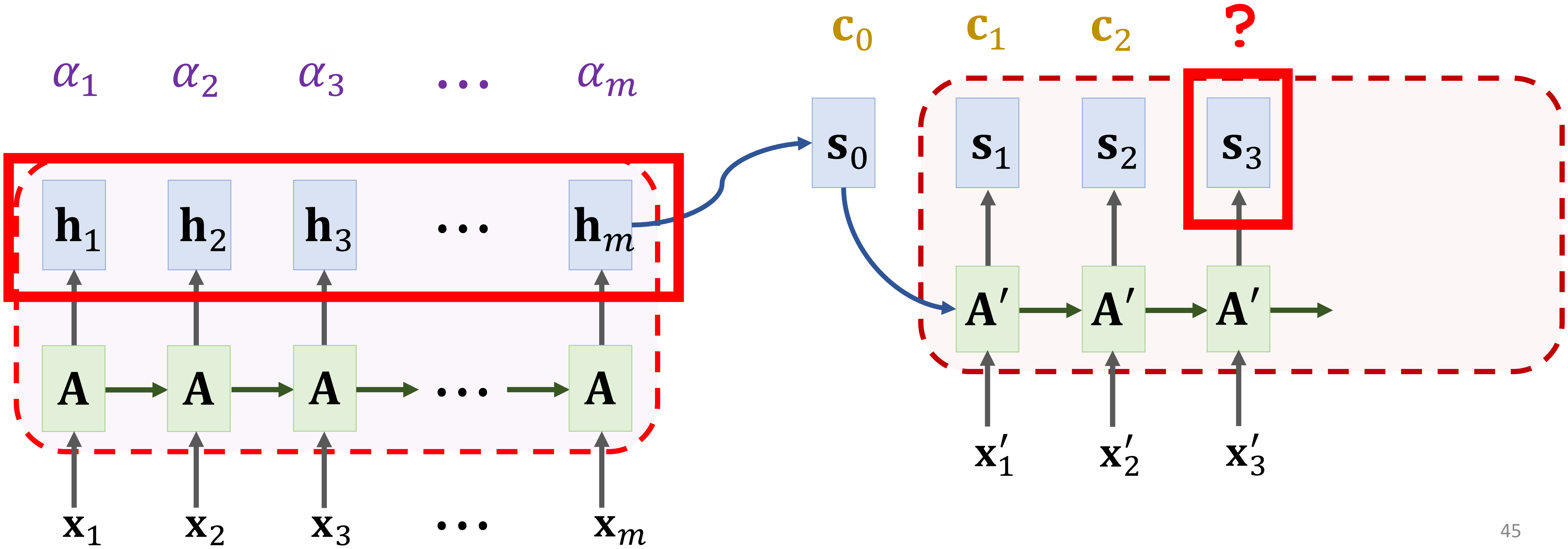
SimpleRNN + Attention



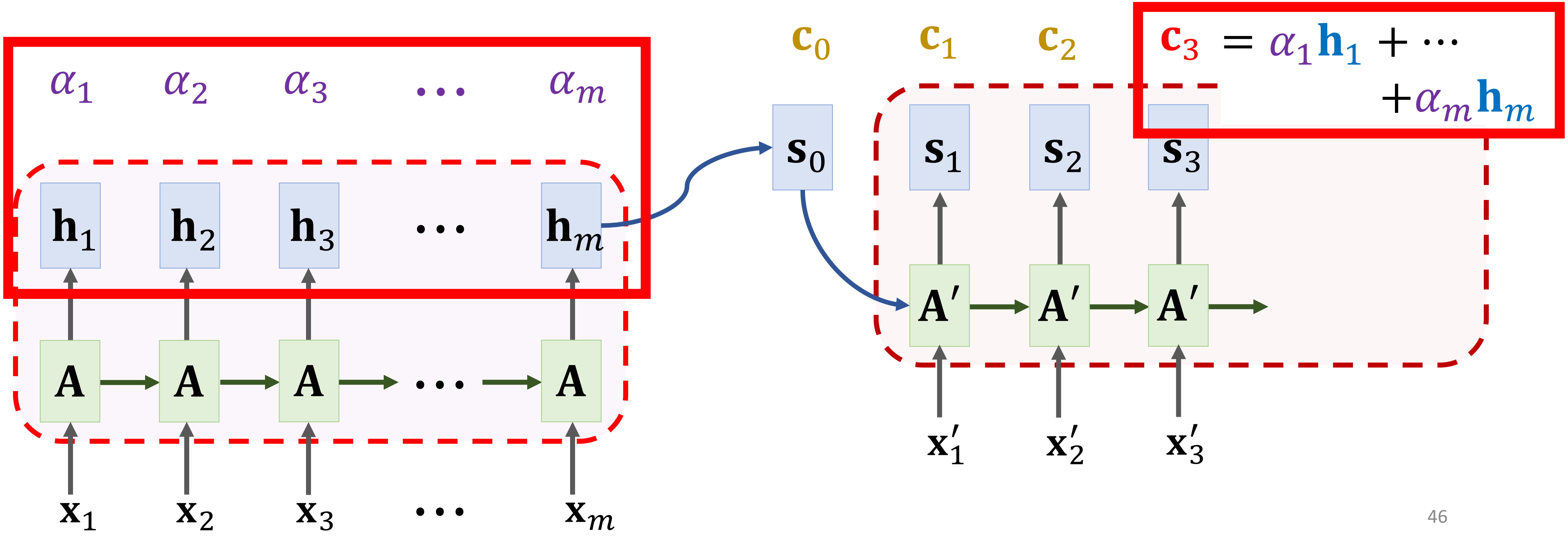
SimpleRNN + Attention



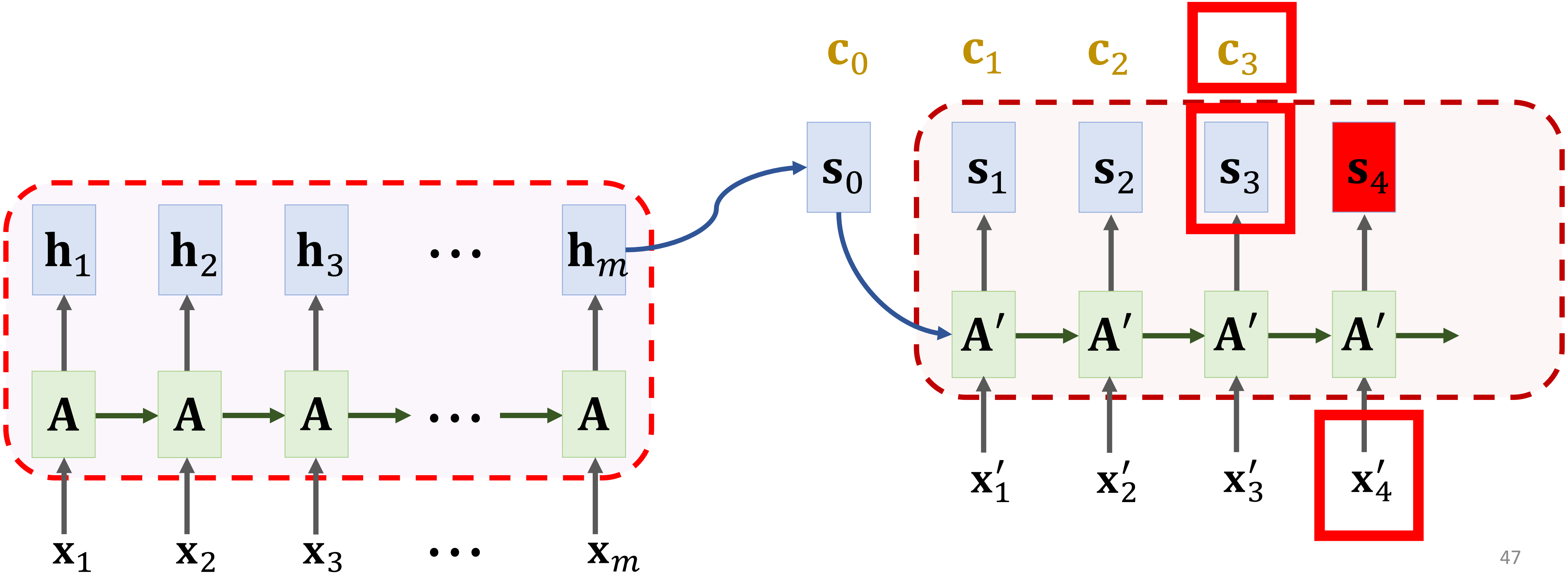
SimpleRNN + Attention



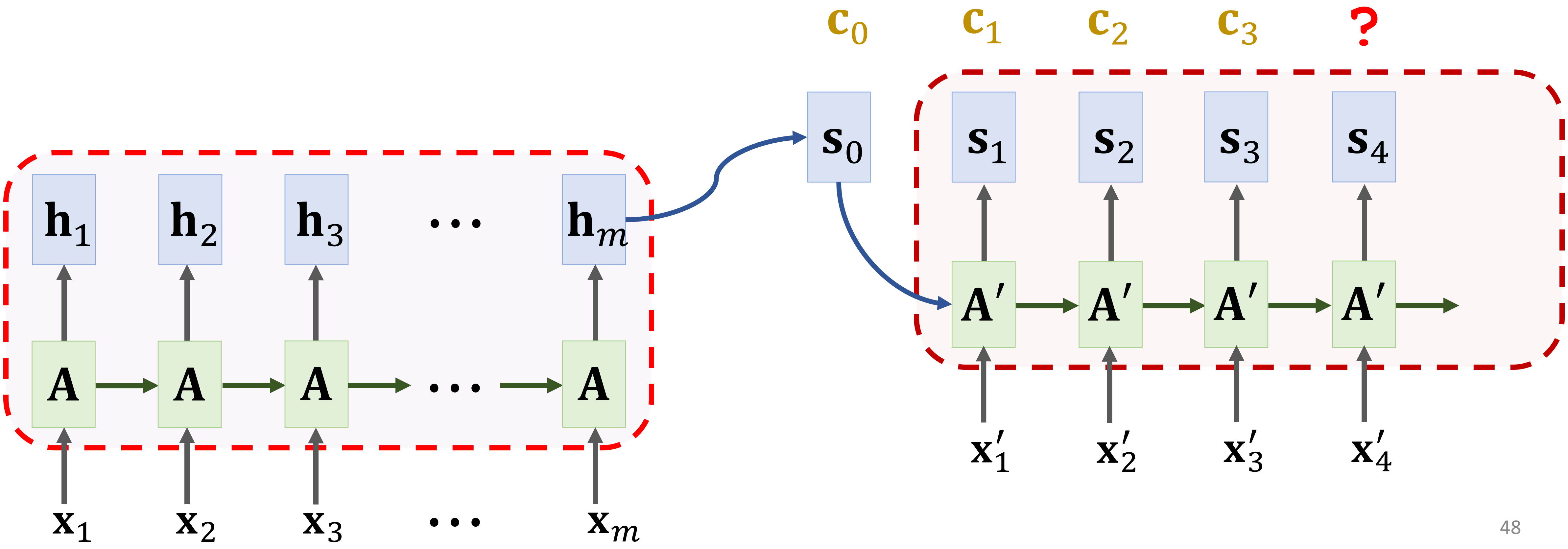
SimpleRNN + Attention



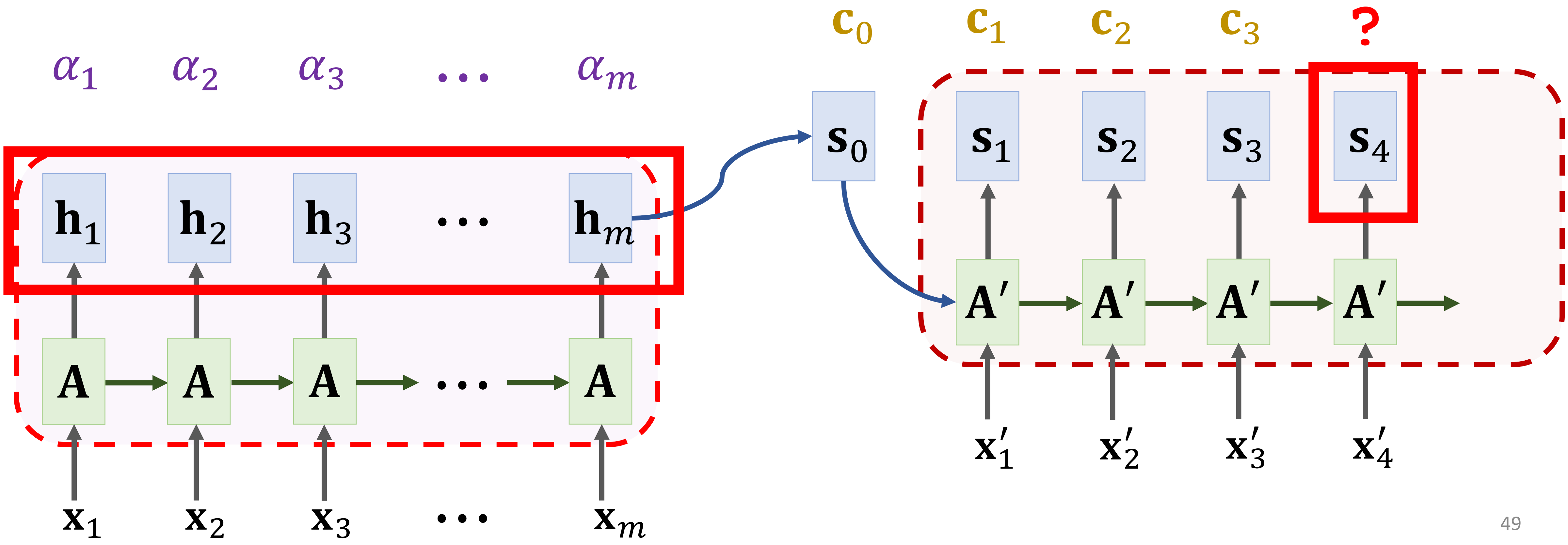
SimpleRNN + Attention



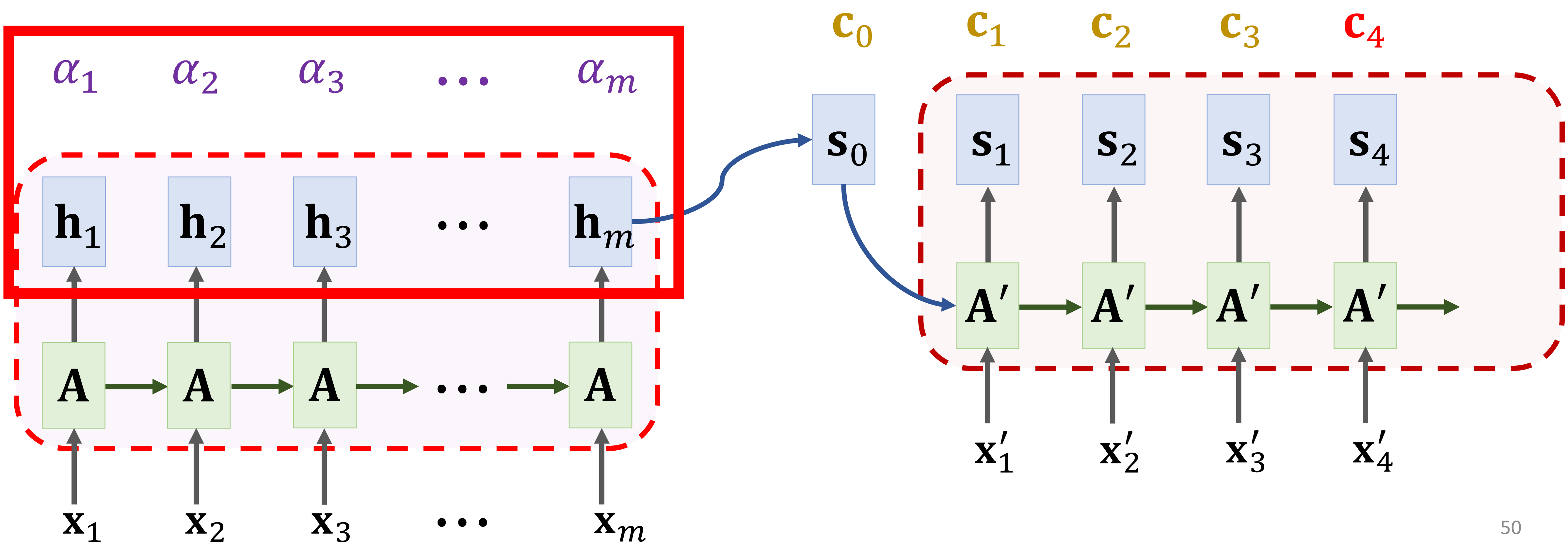
SimpleRNN + Attention



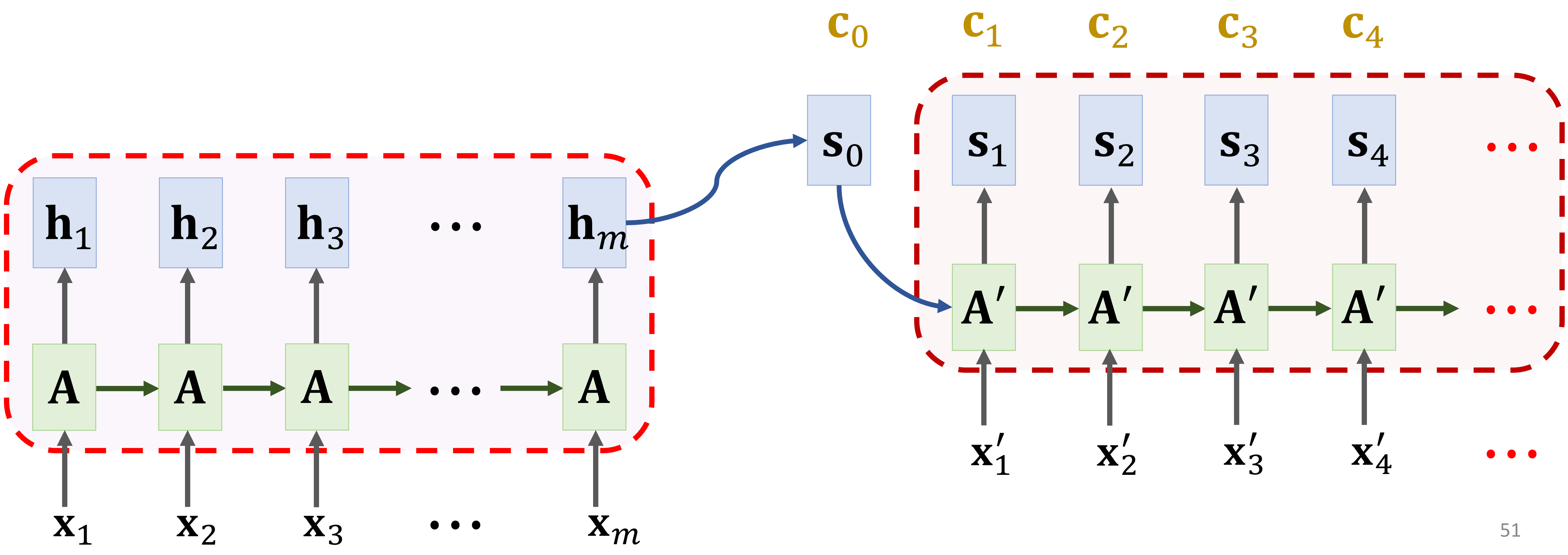
SimpleRNN + Attention



SimpleRNN + Attention

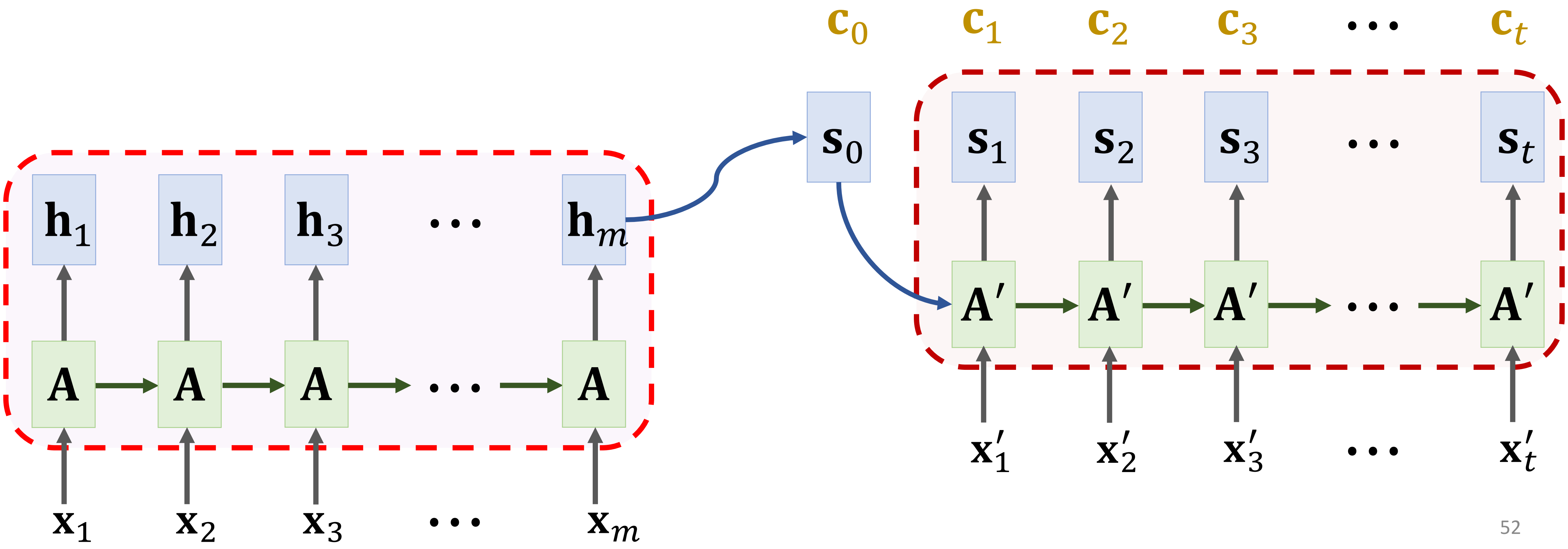


SimpleRNN + Attention



Time Complexity

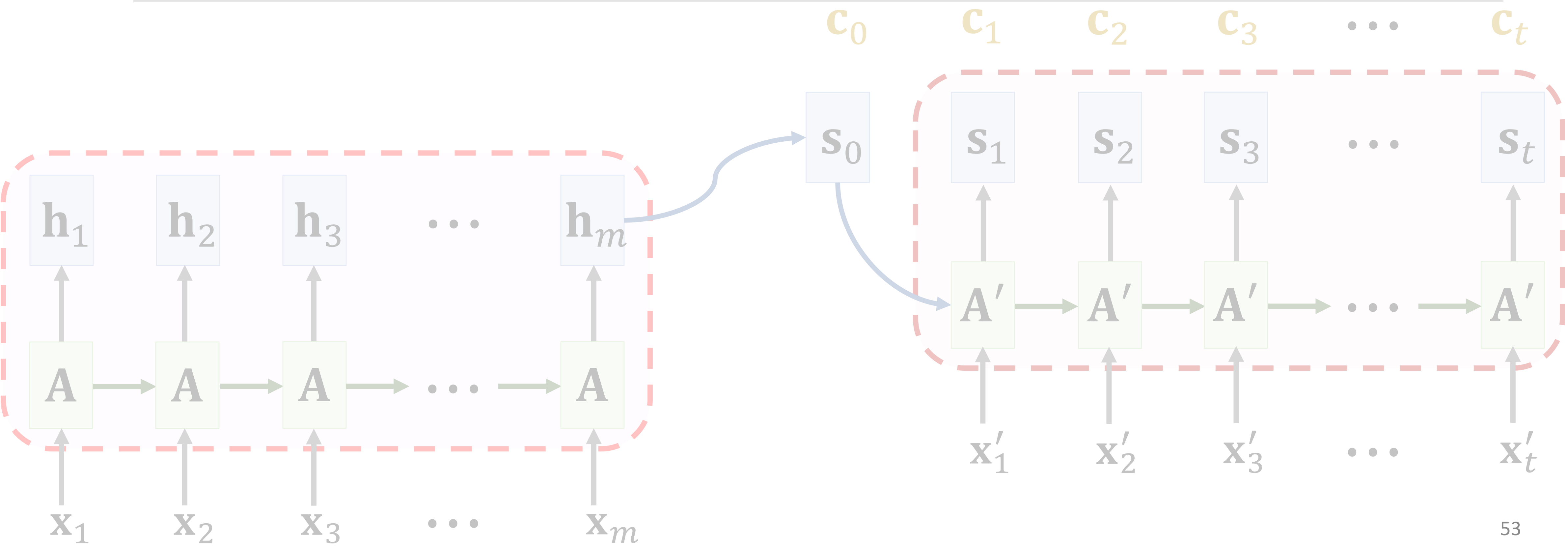
Question: What is the total number of α's we have computed?



Time Complexity

Question: What is the total number of α's we have computed?

- To compute one vector $\mathbf{c}_j$, we compute m weights: $\alpha_1, \cdots, \alpha_m$.
- The decode has t states, so there are a total of mt weights.

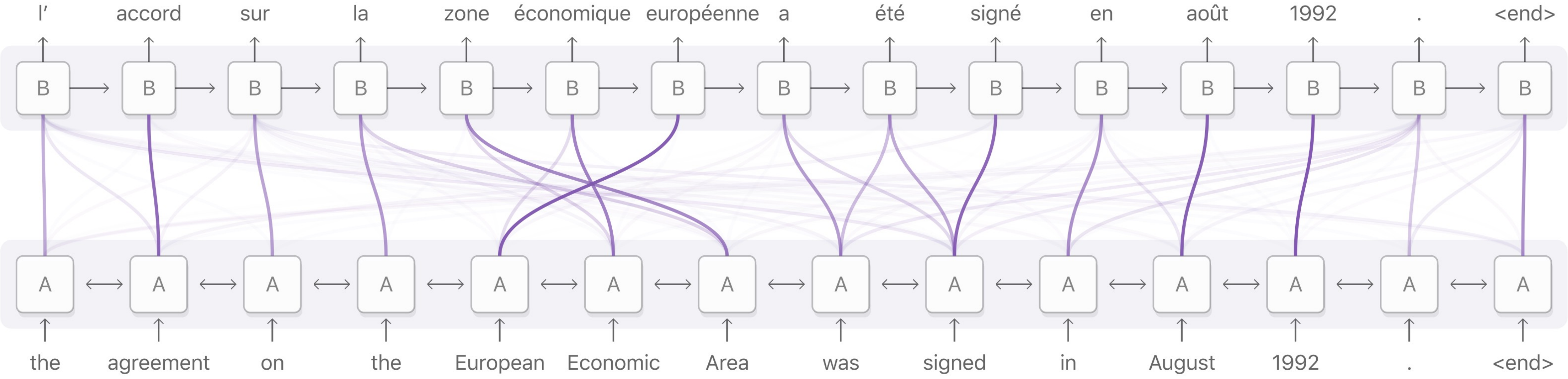


Comparisons

Without Attention	With Attention
Time complexity: $O(m+t)$	Time complexity: $O(mt)$

Attention: Weights Visualization

Decoder RNN (target language: French)

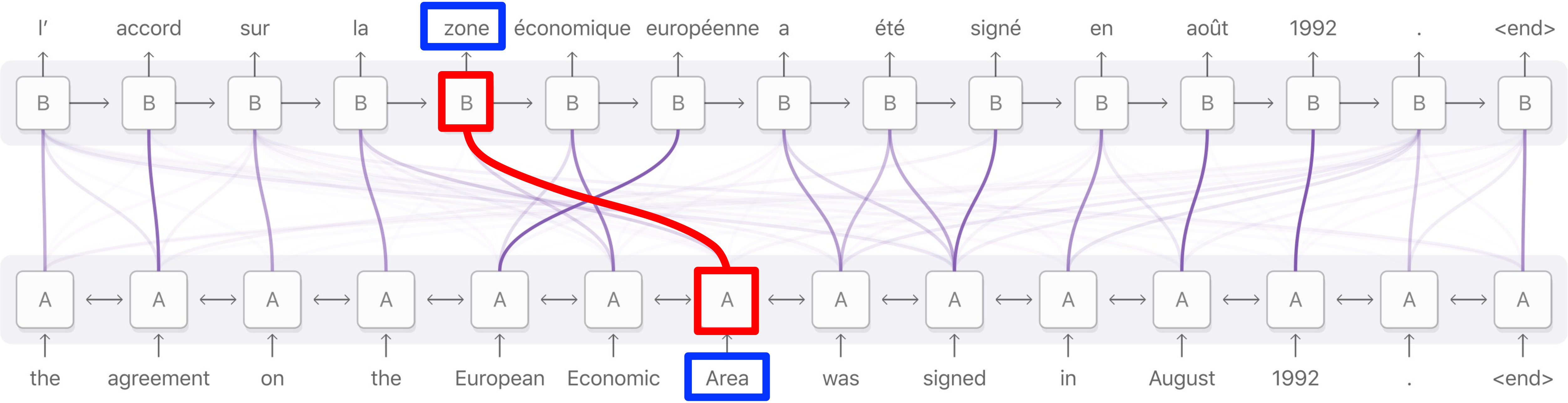


Encoder RNN (source language: English)

Figure is from https://distill.pub/2016/augmented-rnns/

Attention: Weights Visualization

Decoder RNN (target language: French)



Encoder RNN (source language: English)

Figure is from https://distill.pub/2016/augmented-rnns/

Summary

Summary

- Standard Seq2Seq model: the decoder looks at only its current state.

Summary

- Standard Seq2Seq model: the decoder looks at only its current state.
- Attention: decoder additionally looks at all the states of the encoder.

Summary

- Standard Seq2Seq model: the decoder looks at only its current state.
- Attention: decoder additionally looks at all the states of the encoder.
- Attention: decoder knows where to focus.

Summary

- Standard Seq2Seq model: the decoder looks at only its current state.
- Attention: decoder additionally looks at all the states of the encoder.
- Attention: decoder knows where to focus.

- Downside: higher time complexity.
  - m: source sequence length
  - t: target sequence length
  - Standard Seq2Seq: $O(m+t)$ time complexity
  - Seq2Seq + attention: $O(mt)$ time complexity

Self-Attention for RNNs

Self-Attention

- Self-Attention [2]: attention [1] beyond Seq2Seq models.
- The self-attention paper uses LSTM.
- To make teaching easy, I replace LSTM by SimpleRNN.

Original paper:

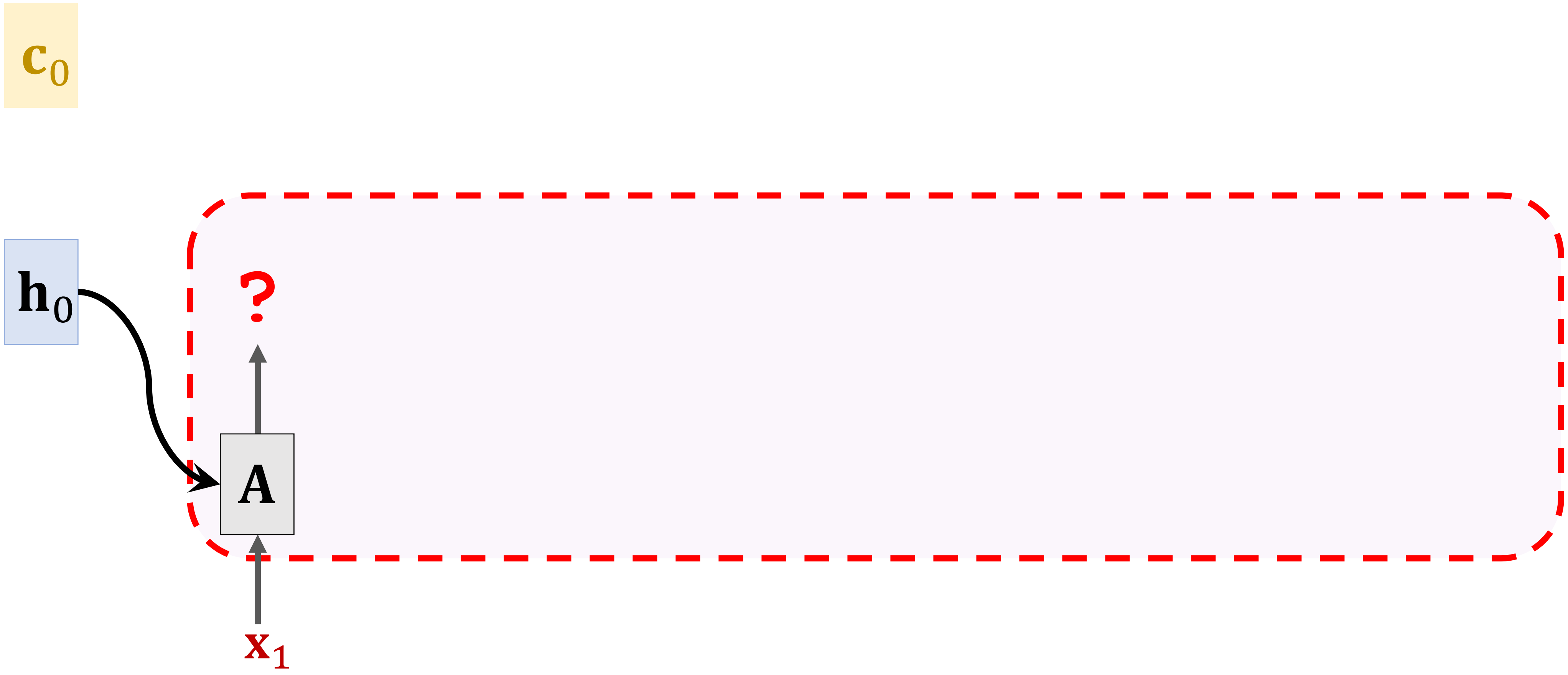
1. Bahdanau, Cho, & Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.
2. Cheng, Dong, & Lapata. Long Short-Term Memory-Networks for Machine Reading. In *EMNLP*, 2016.

SimpleRNN + Self-Attention

$\mathbf{c}_0 = \mathbf{0}$

$\mathbf{h}_0 = \mathbf{0}$

SimpleRNN + Self-Attention



SimpleRNN + Self-Attention

SimpleRNN:

$$\mathbf{h}_1 = \tanh\left(\mathbf{A} \cdot \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{h}_0 \end{bmatrix} + \mathbf{b}\right)$$

$\mathbf{c}_0$

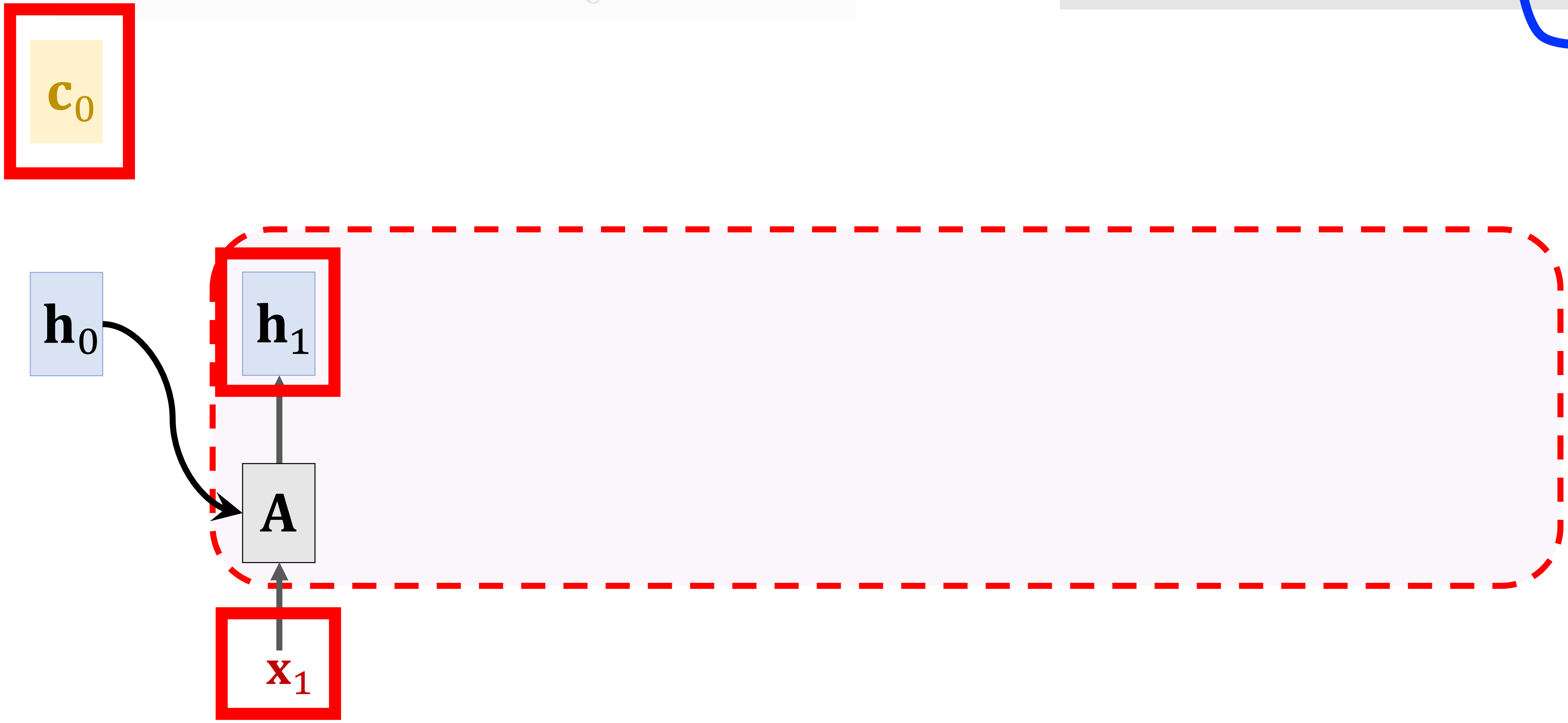
SimpleRNN + Self-Attention

SimpleRNN:

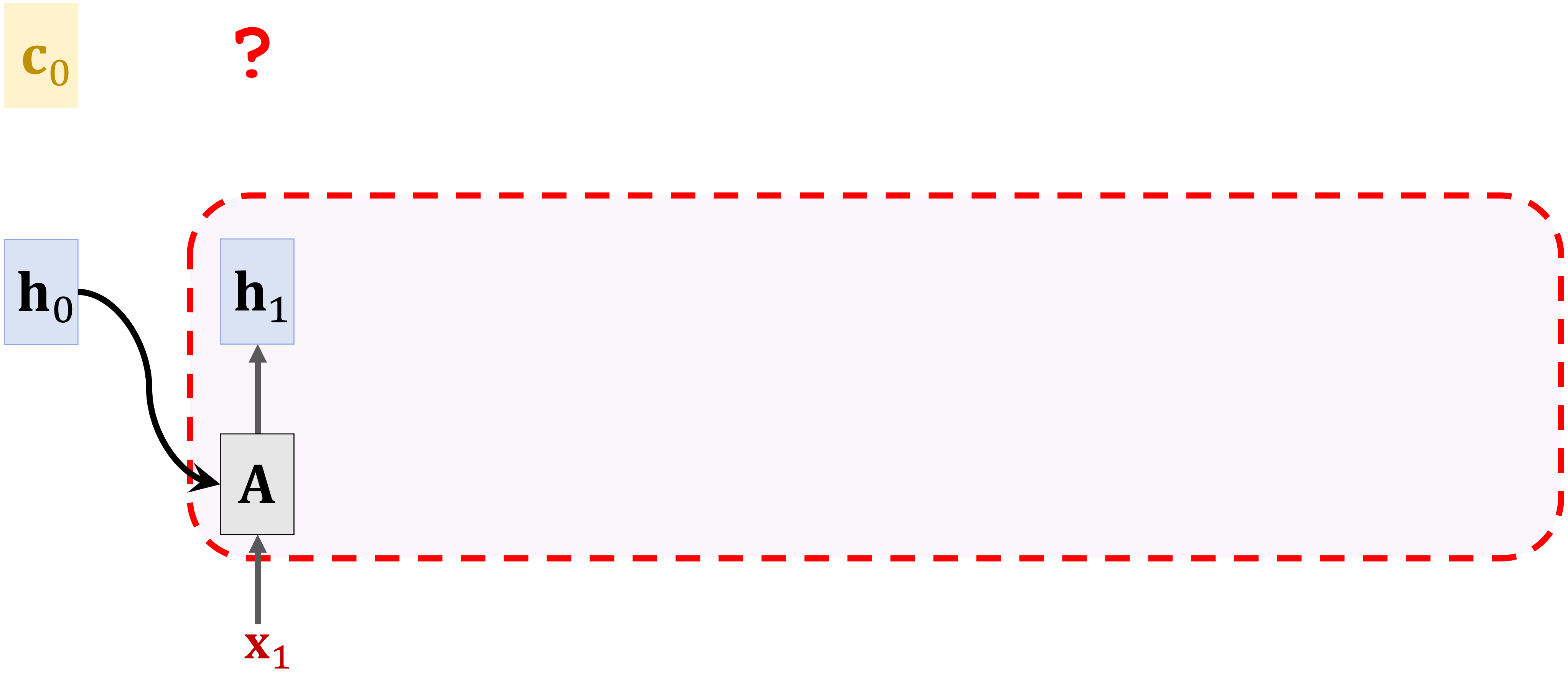
$$\mathbf{h}_1 = \tanh\left(\mathbf{A} \cdot \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{h}_0 \end{bmatrix} + \mathbf{b}\right)$$

SimpleRNN + Self-Attention:

$$\mathbf{h}_1 = \tanh\left(\mathbf{A} \cdot \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{c}_0 \end{bmatrix} + \mathbf{b}\right)$$



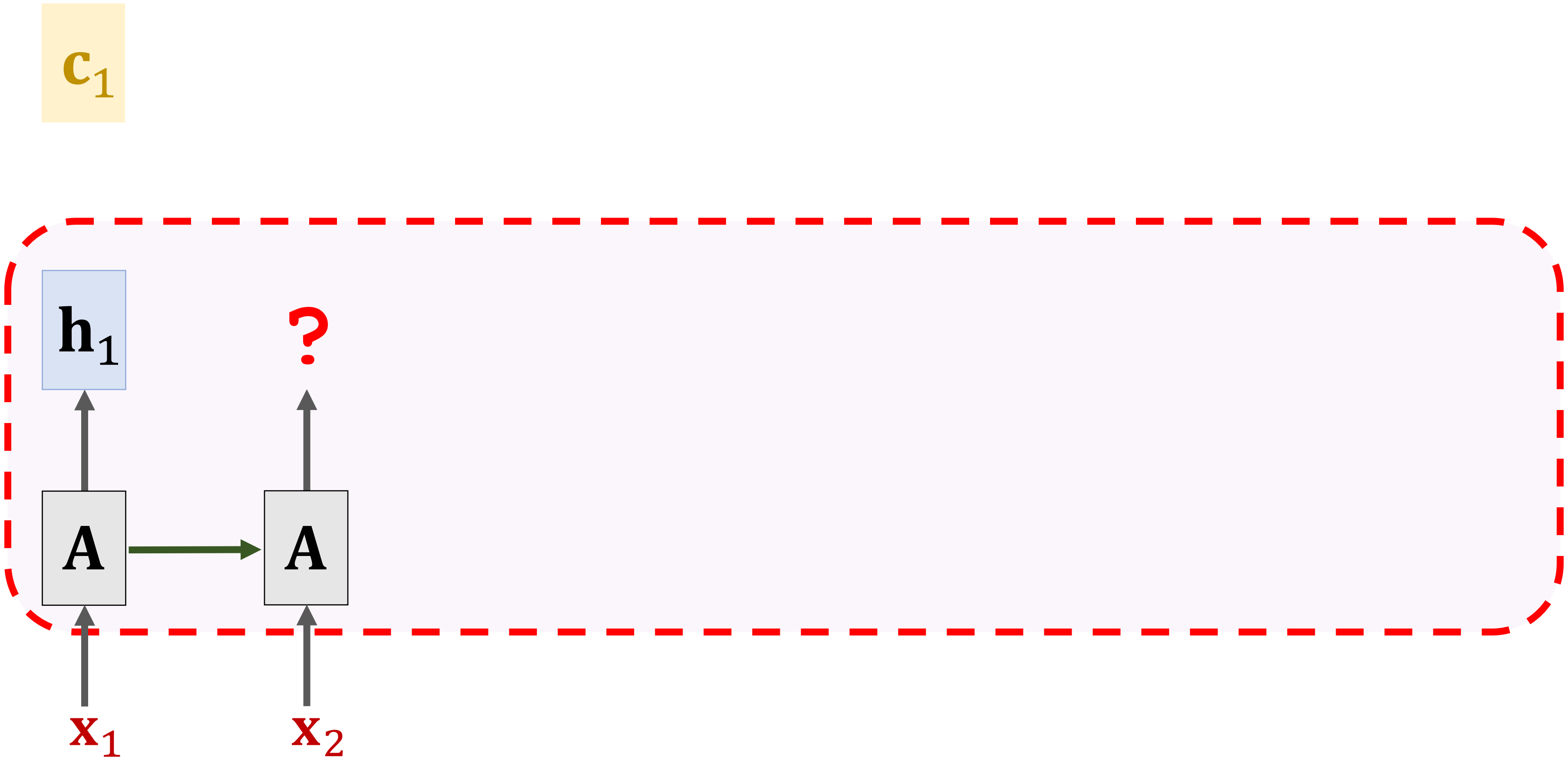
SimpleRNN + Self-Attention



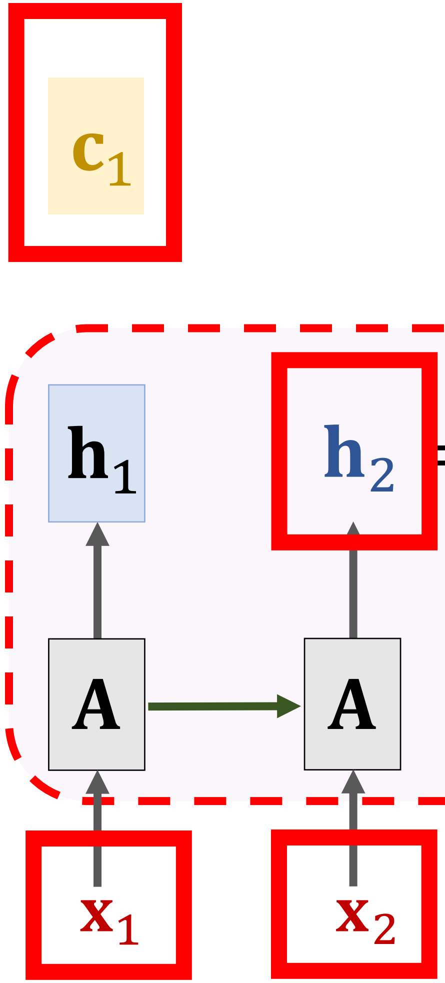
SimpleRNN + Self-Attention

$\mathbf{c}_0$ $\mathbf{c}_1 = \mathbf{h}_1$

SimpleRNN + Self-Attention

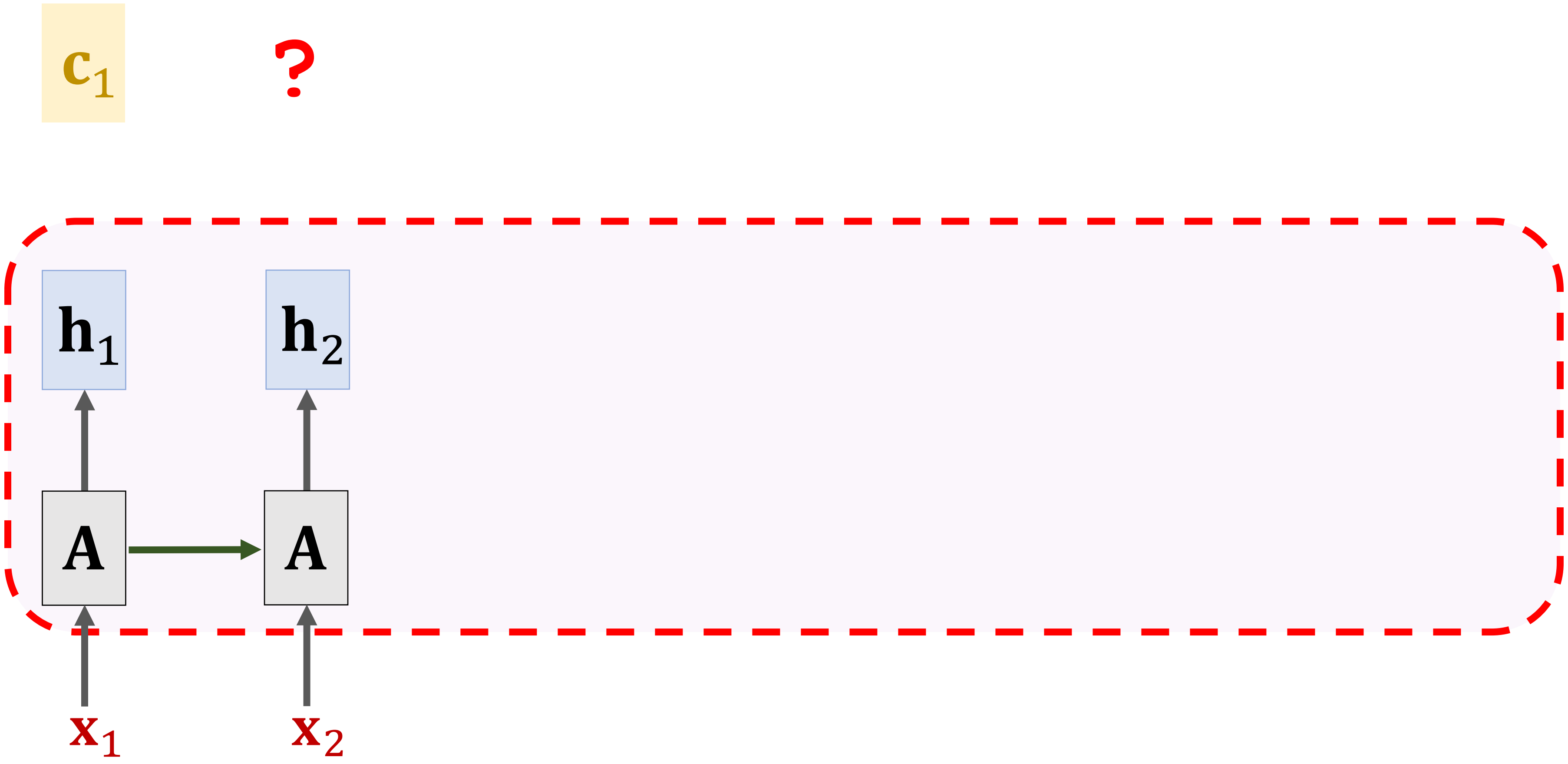


SimpleRNN + Self-Attention

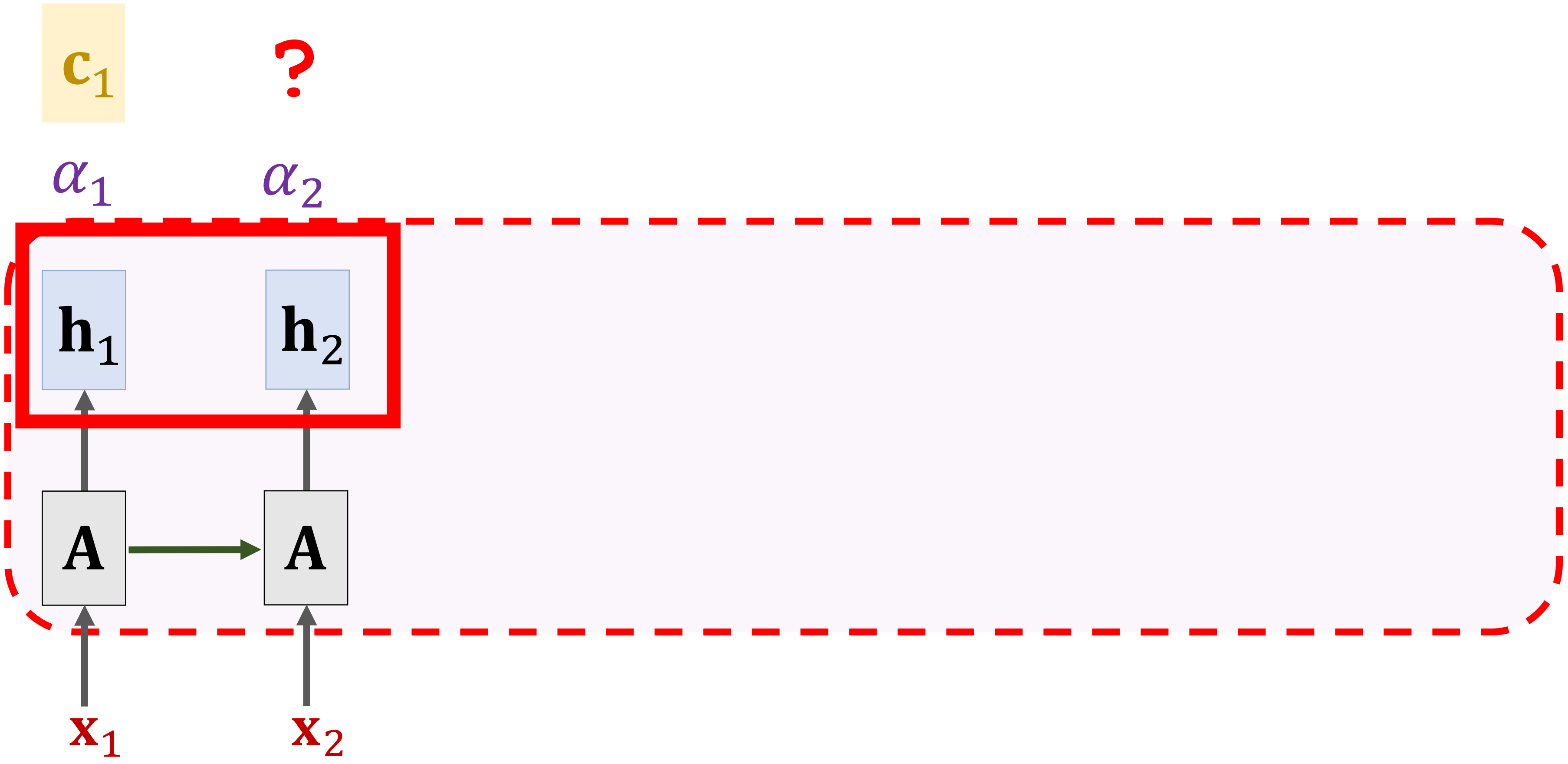


$$\mathbf{h}_2 = \tanh\left(\mathbf{A} \cdot \begin{bmatrix} \mathbf{x}_2 \\ \mathbf{c}_1 \end{bmatrix} + \mathbf{b}\right)$$

SimpleRNN + Self-Attention

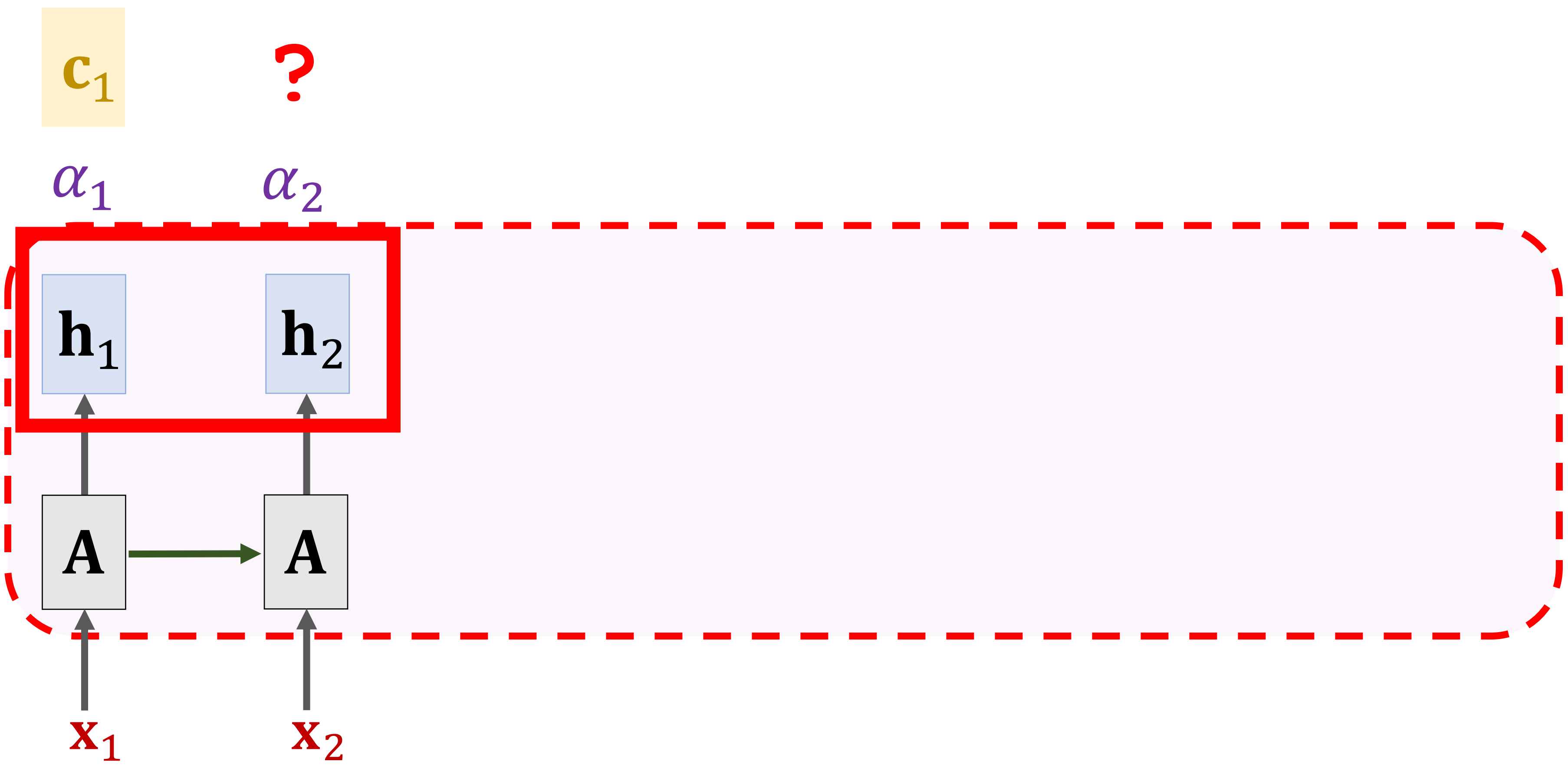


SimpleRNN + Self-Attention

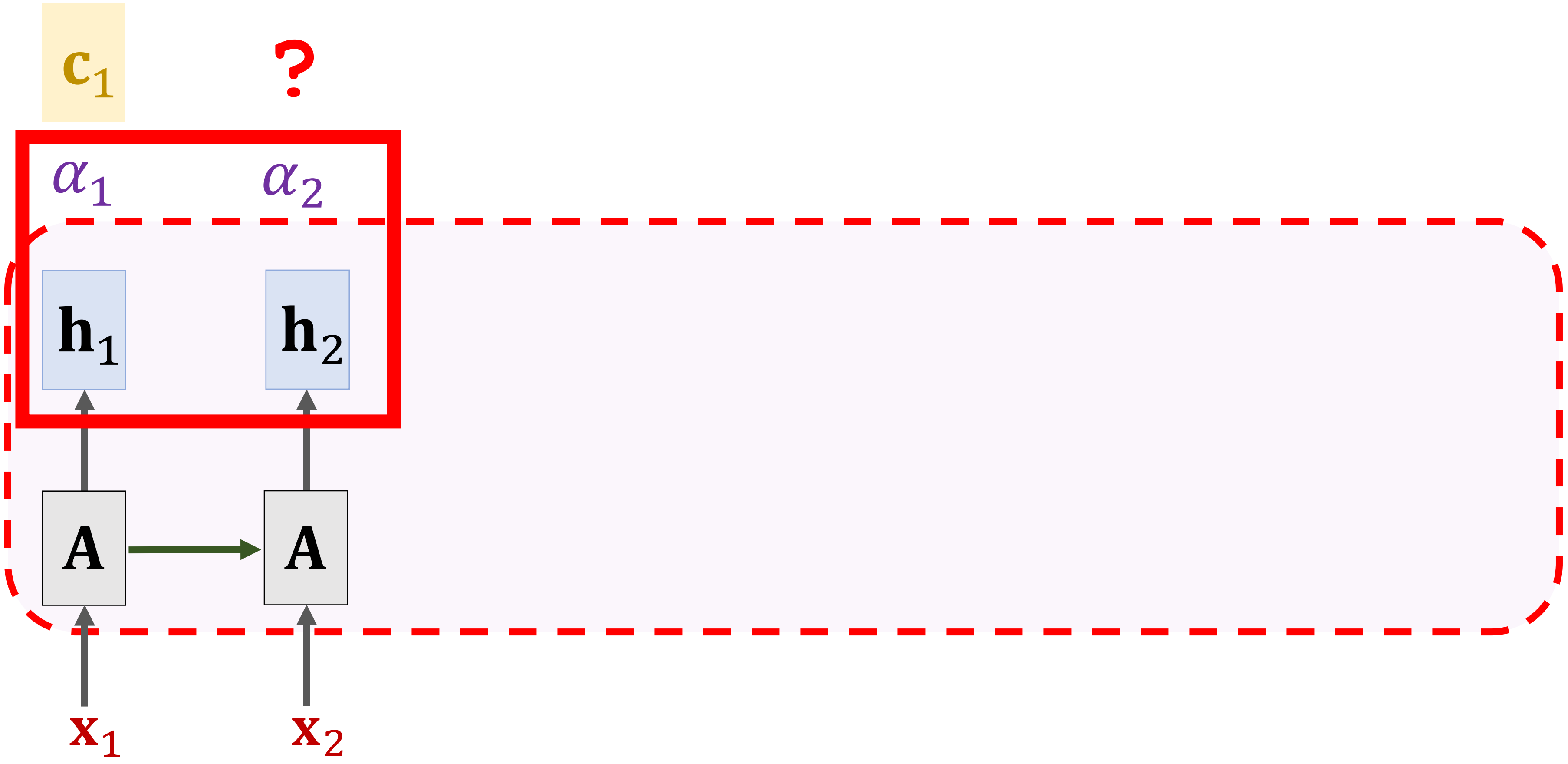


SimpleRNN + Self-Attention

Weights: $\alpha_i = \text{align}(\mathbf{h}_i, \mathbf{h}_2).$

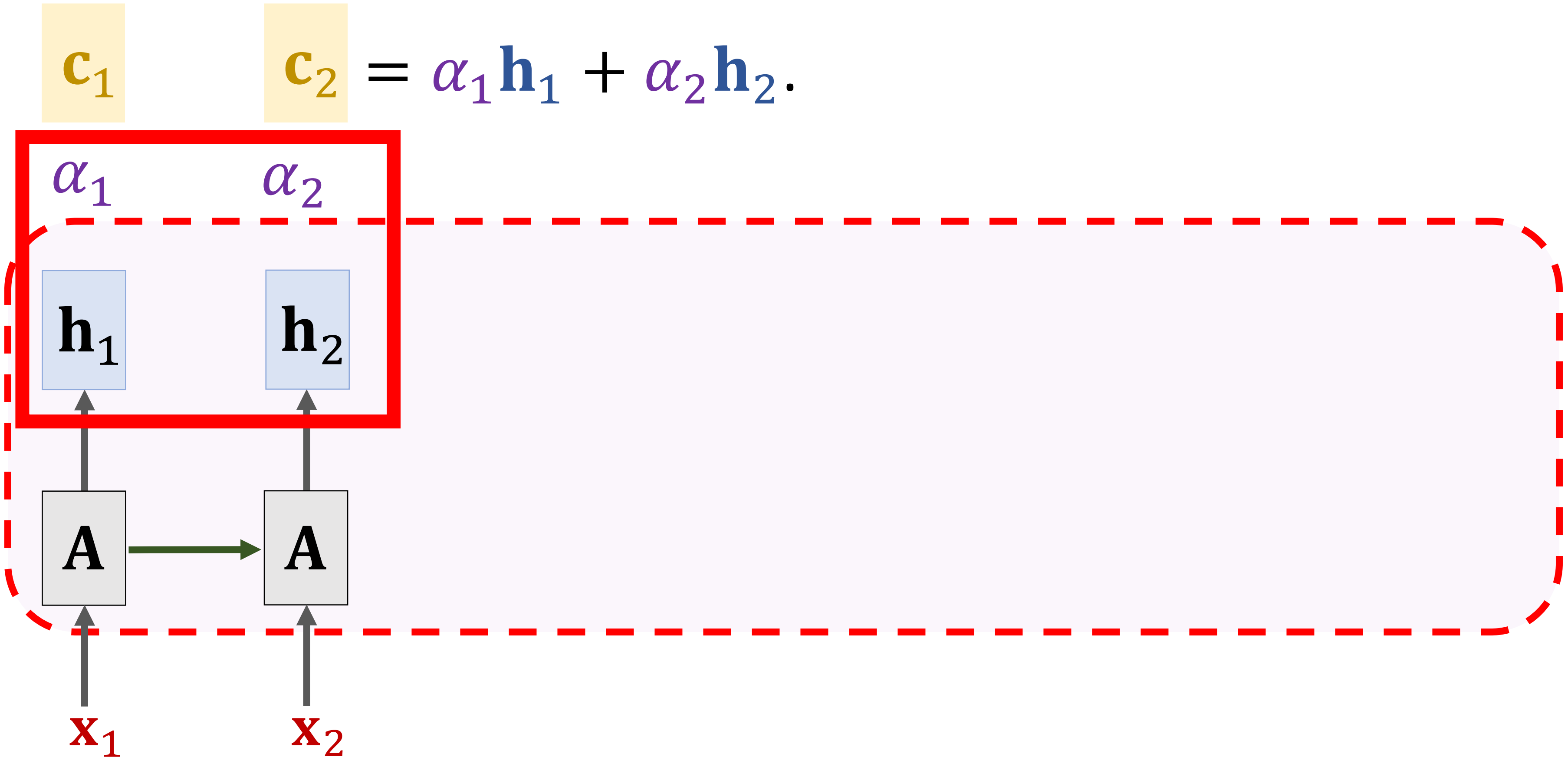


SimpleRNN + Self-Attention

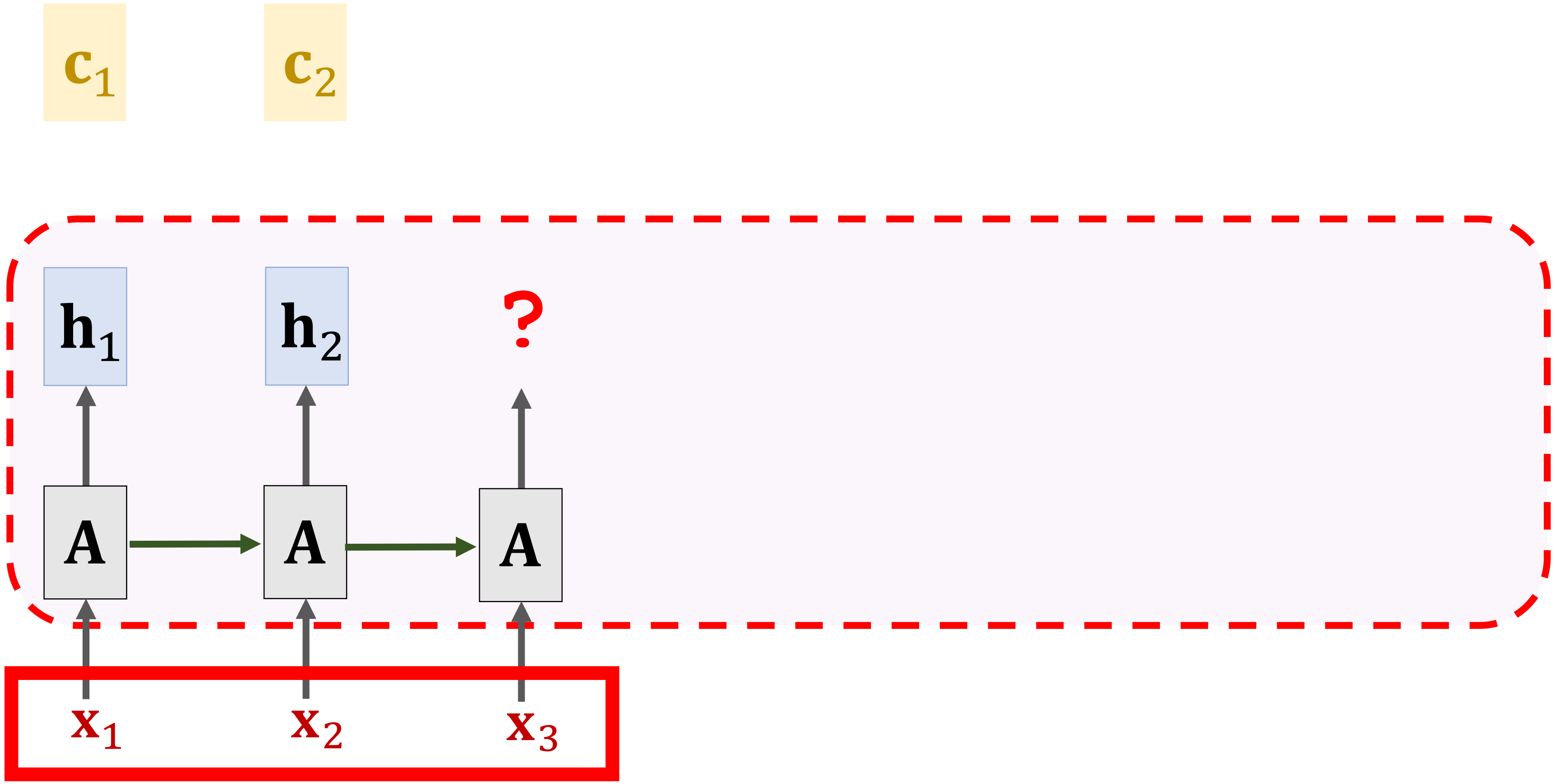


SimpleRNN + Self-Attention

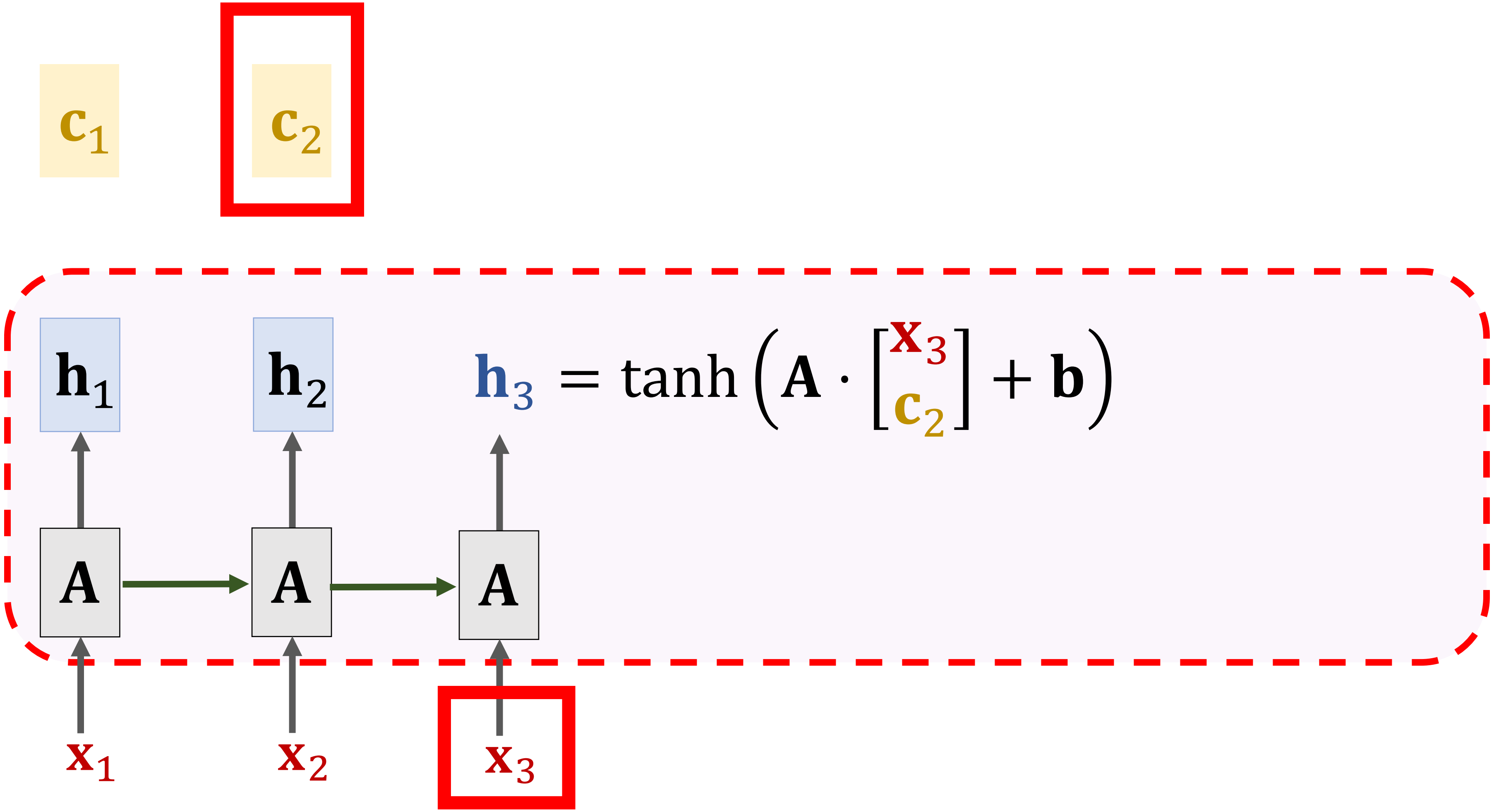
$\mathbf{c}_2 = \alpha_1 \mathbf{h}_1 + \alpha_2 \mathbf{h}_2.$



SimpleRNN + Self-Attention

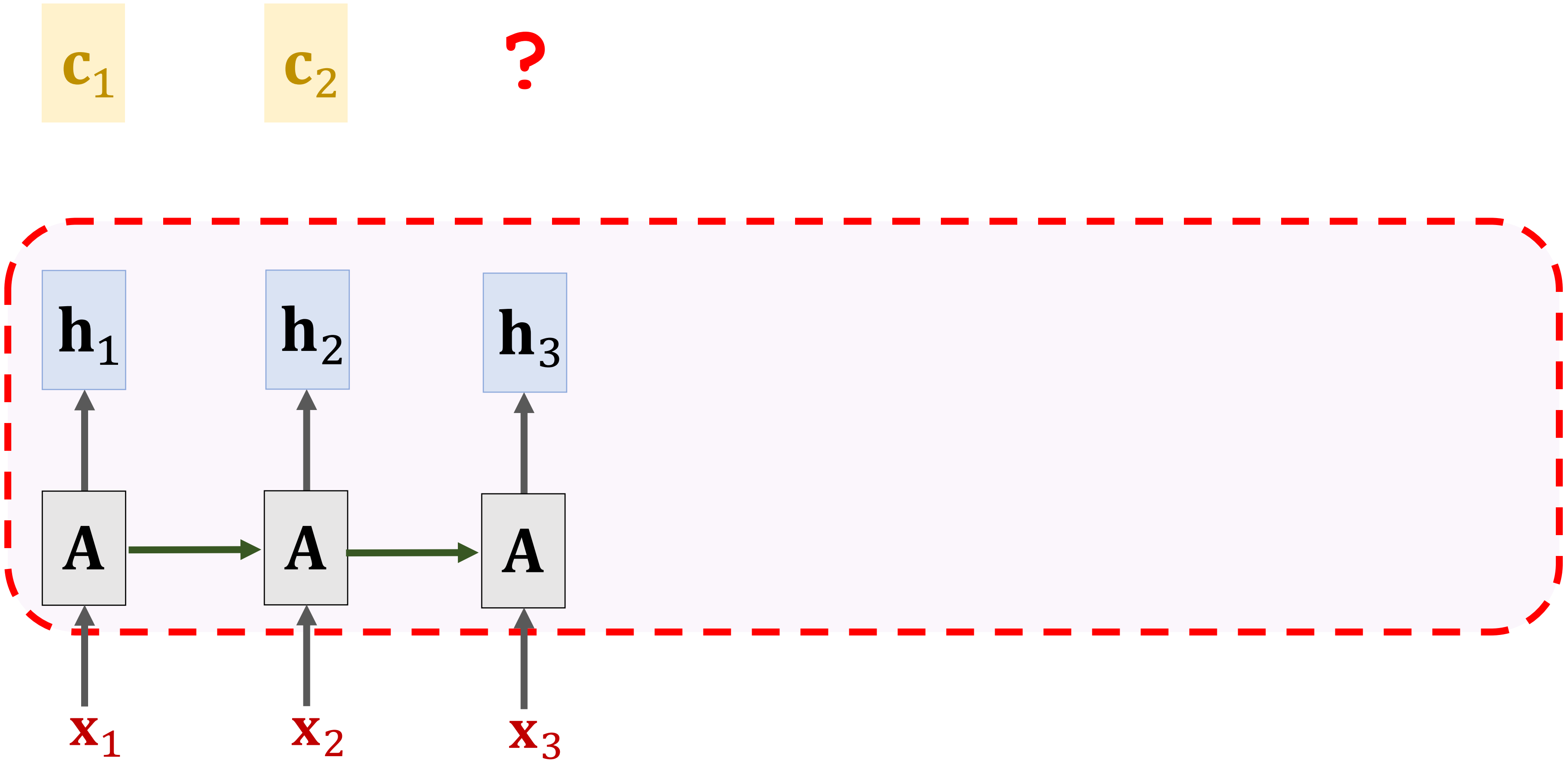


SimpleRNN + Self-Attention

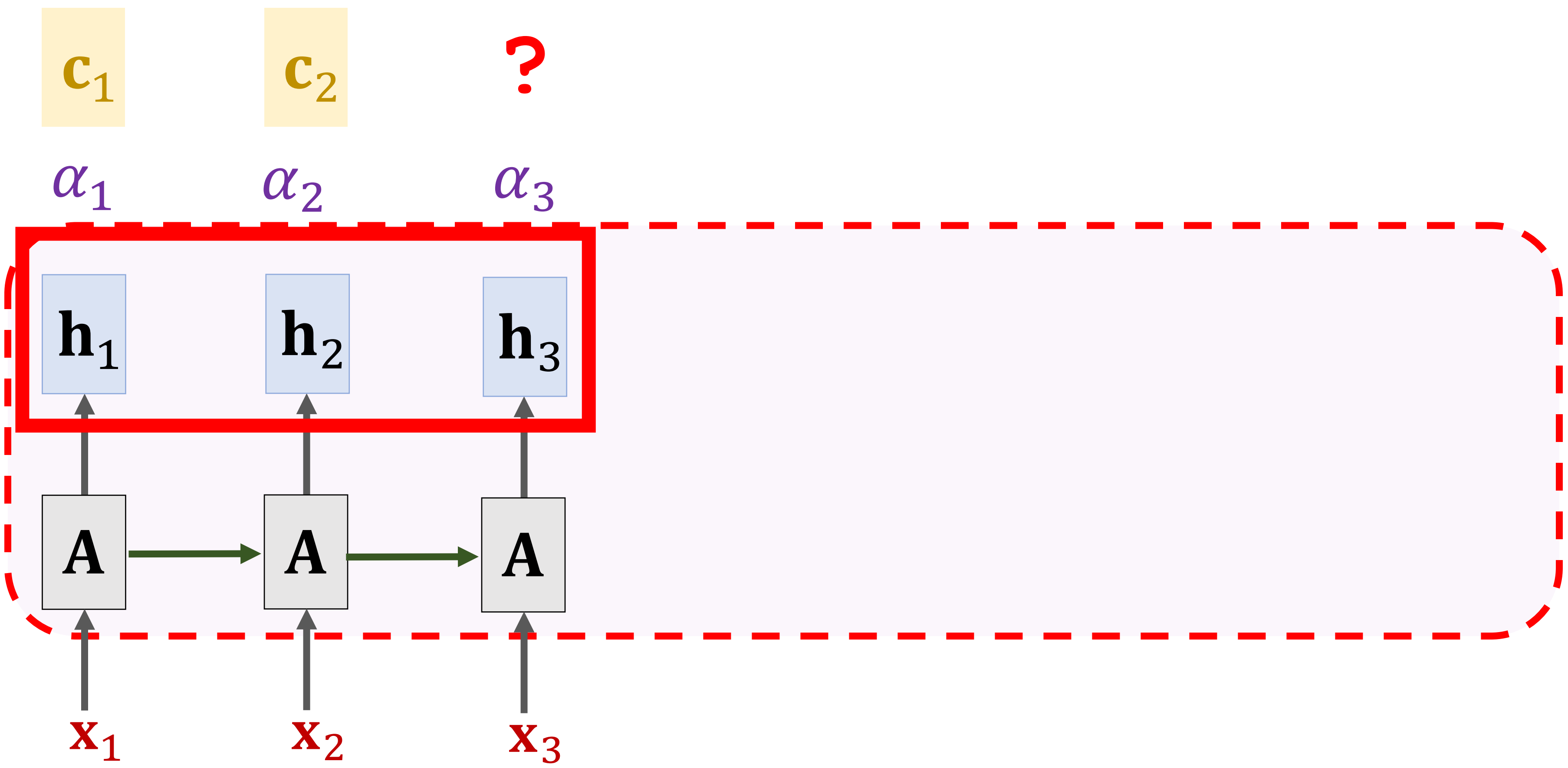


$$\mathbf{h}_3 = \tanh\left(\mathbf{A} \cdot \begin{bmatrix} \mathbf{x}_3 \\ \mathbf{c}_2 \end{bmatrix} + \mathbf{b}\right)$$

SimpleRNN + Self-Attention

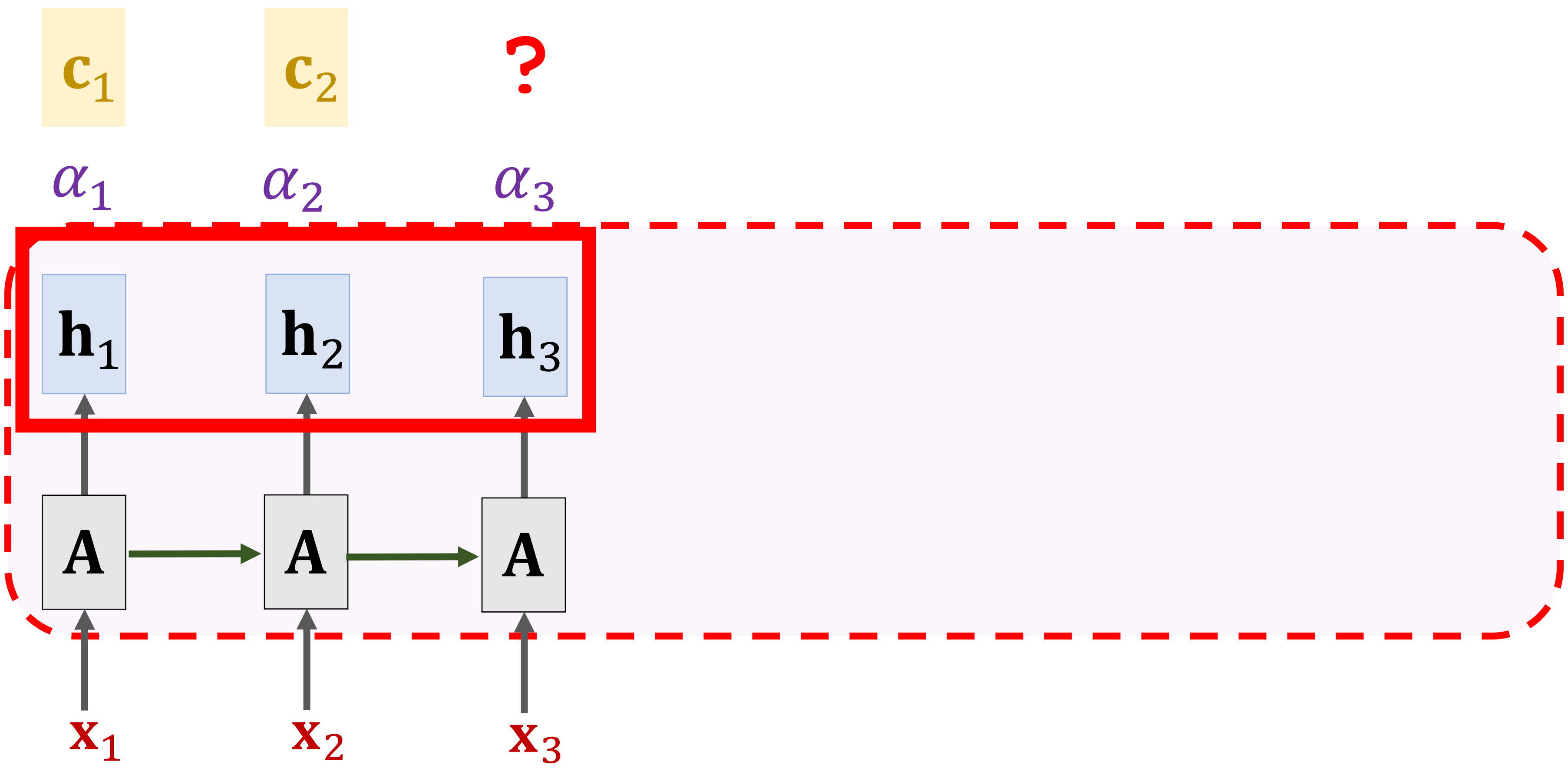


SimpleRNN + Self-Attention

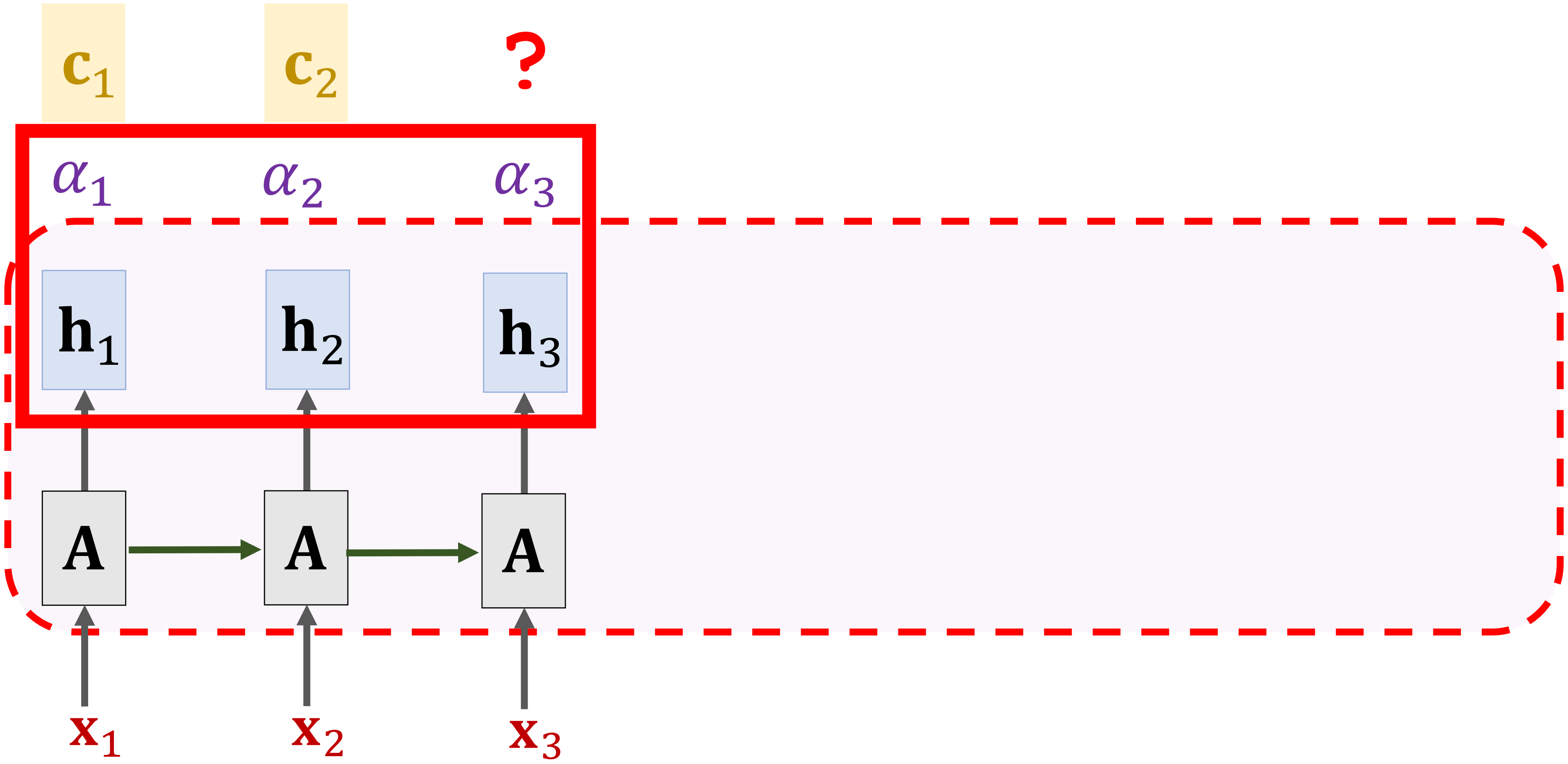


SimpleRNN + Self-Attention

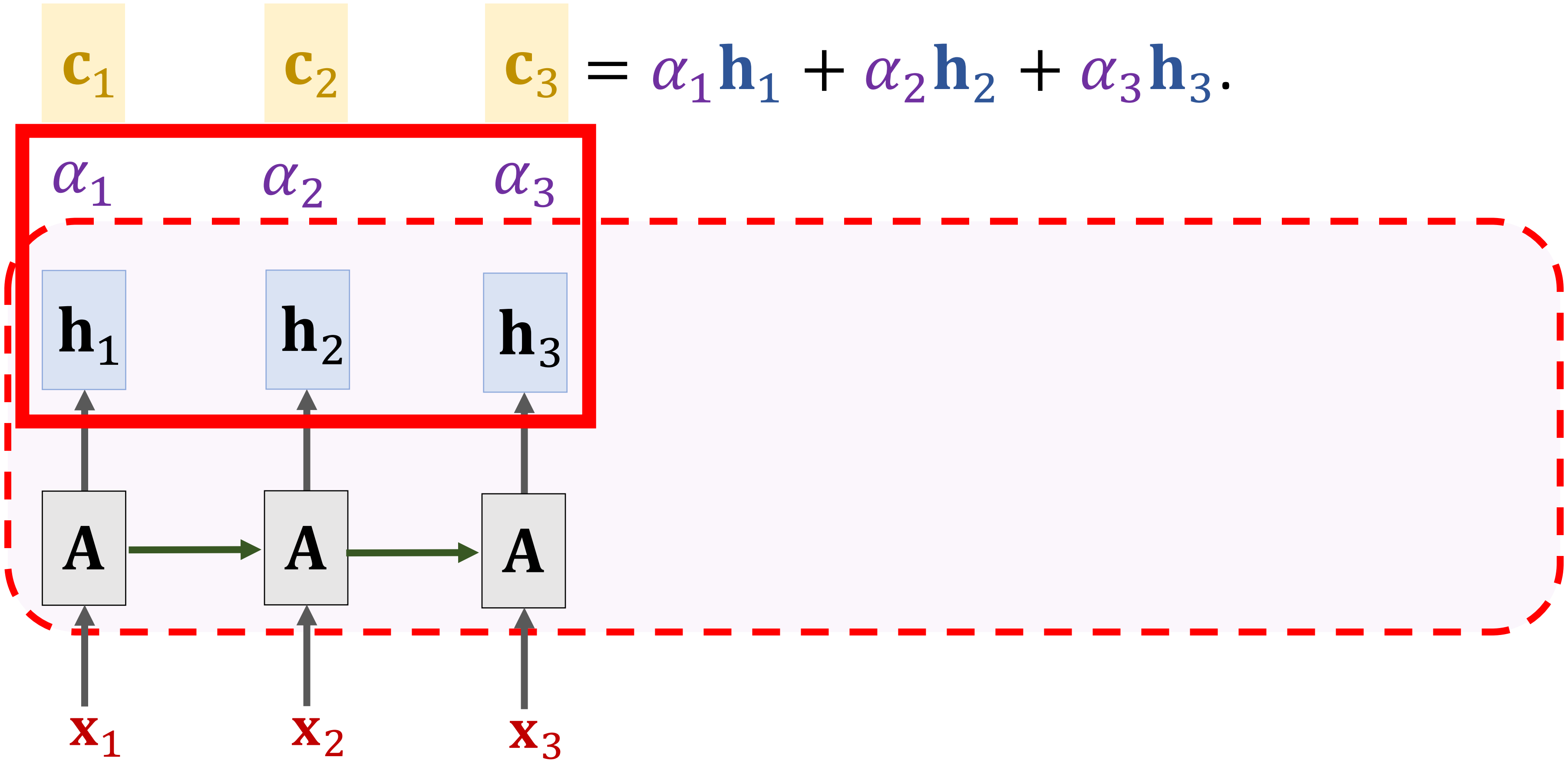
Weights: $\alpha_i = \text{align}(\mathbf{h}_i,\ \mathbf{h}_3).$



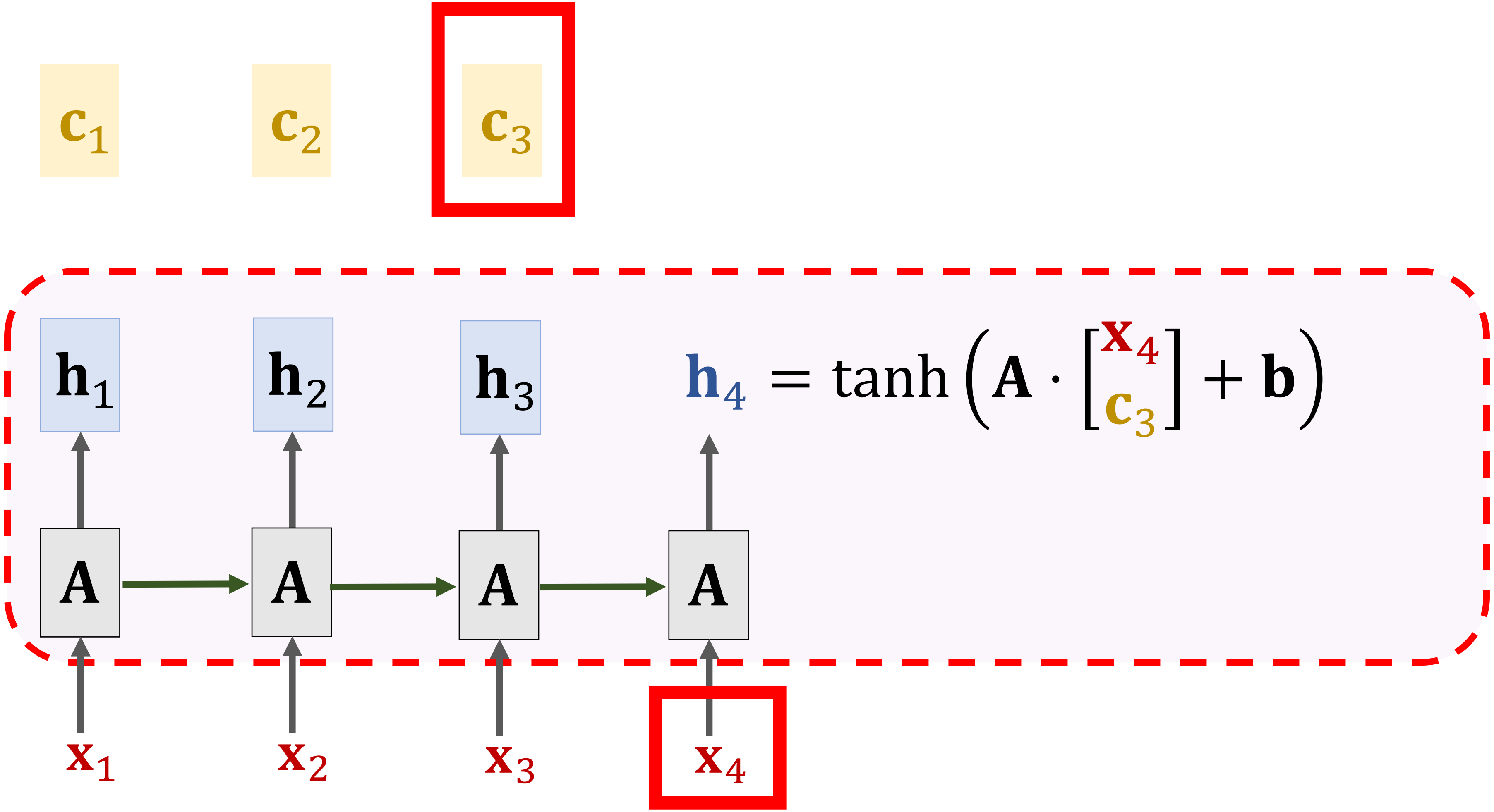
SimpleRNN + Self-Attention



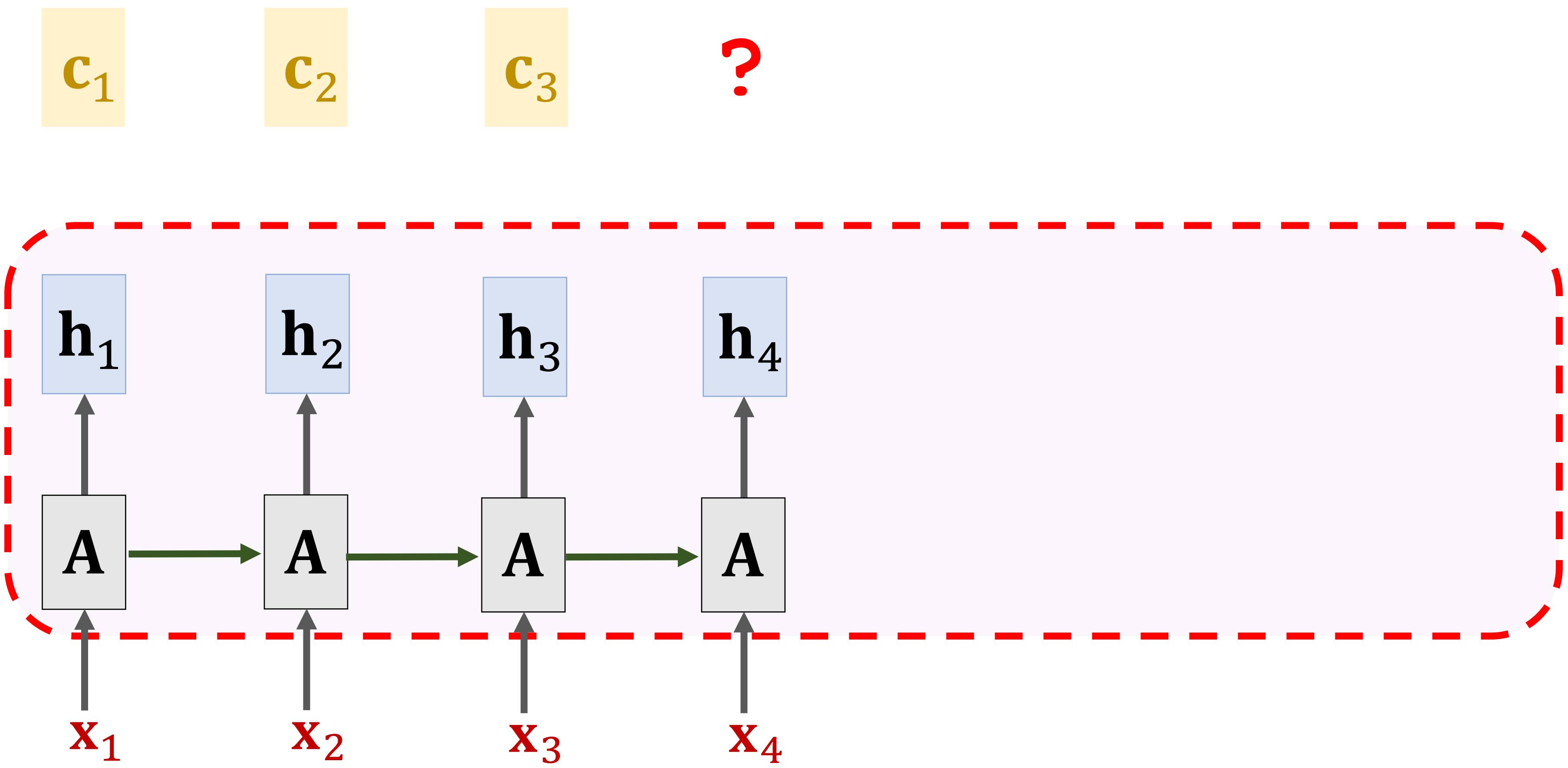
SimpleRNN + Self-Attention



SimpleRNN + Self-Attention

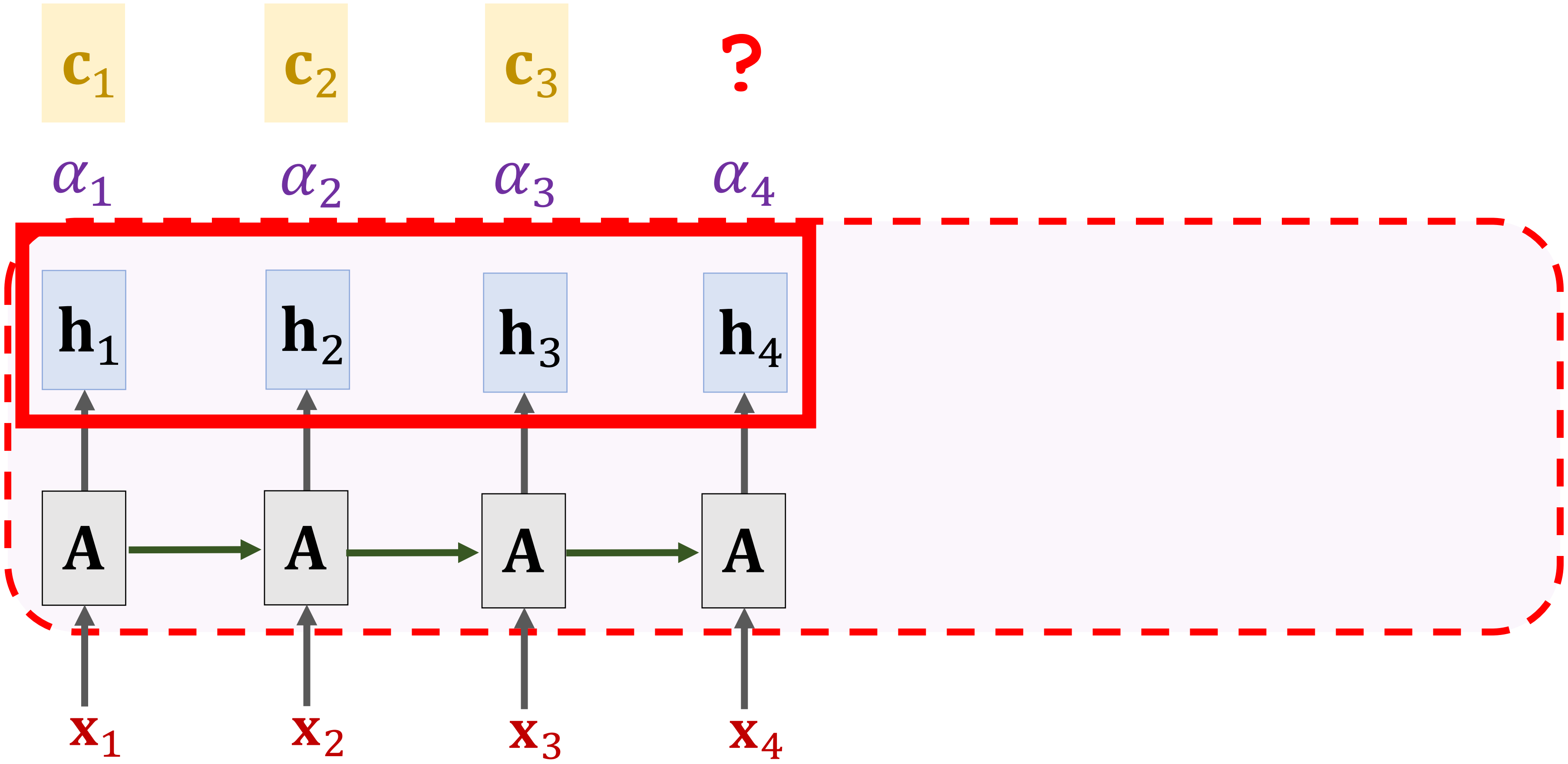


SimpleRNN + Self-Attention

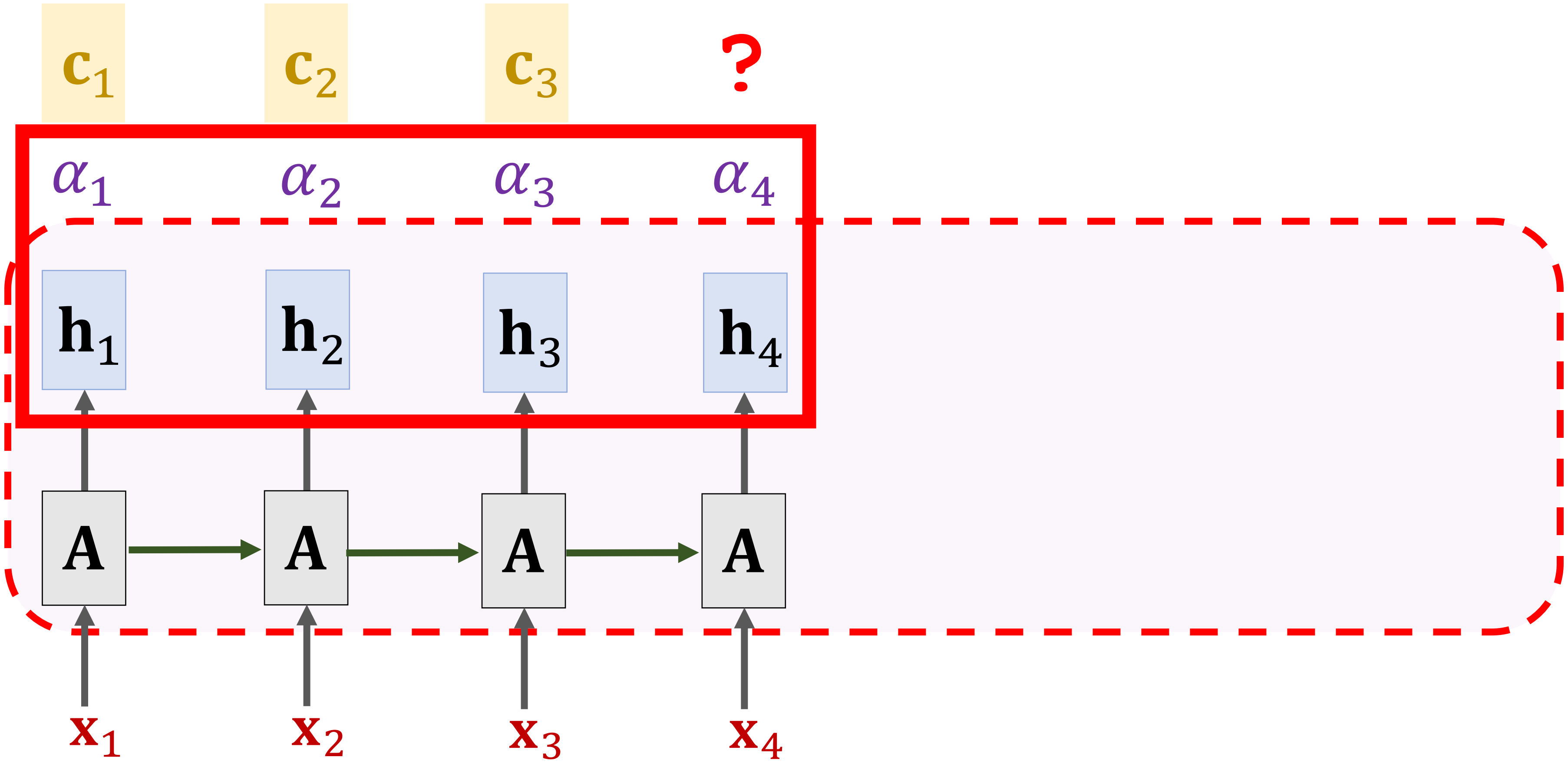


SimpleRNN + Self-Attention

Weights: $\alpha_i = \text{align}(\mathbf{h}_i, \ \mathbf{h}_4).$

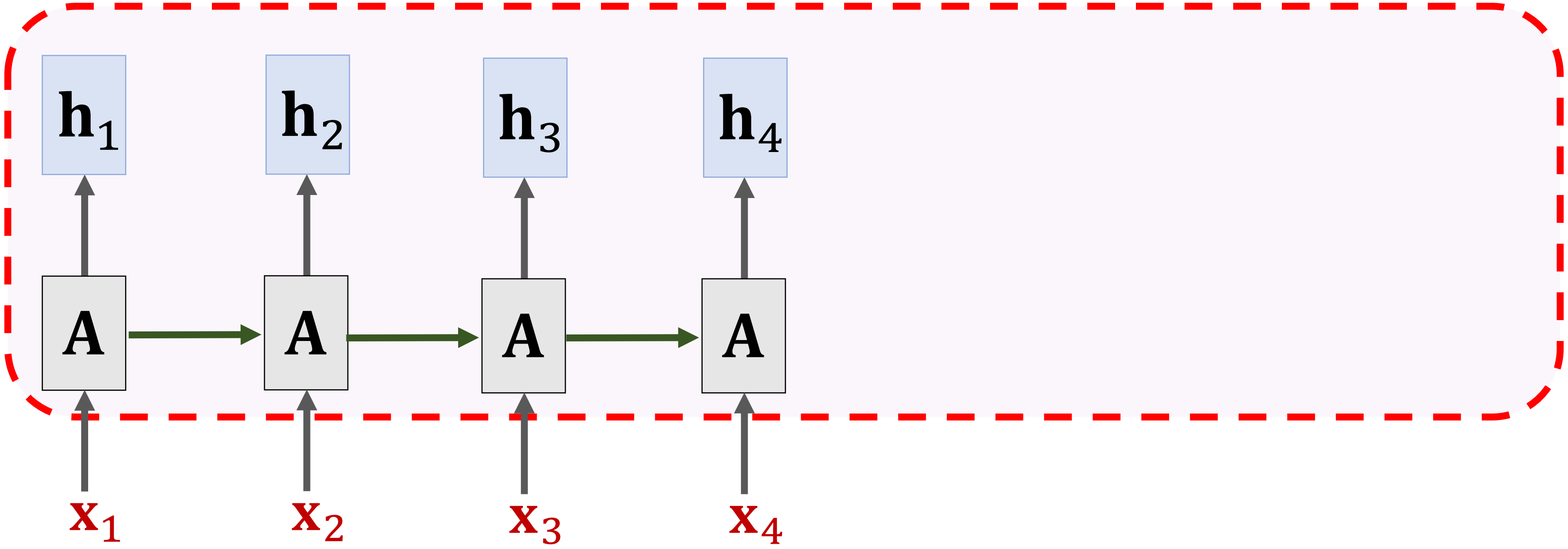


SimpleRNN + Self-Attention

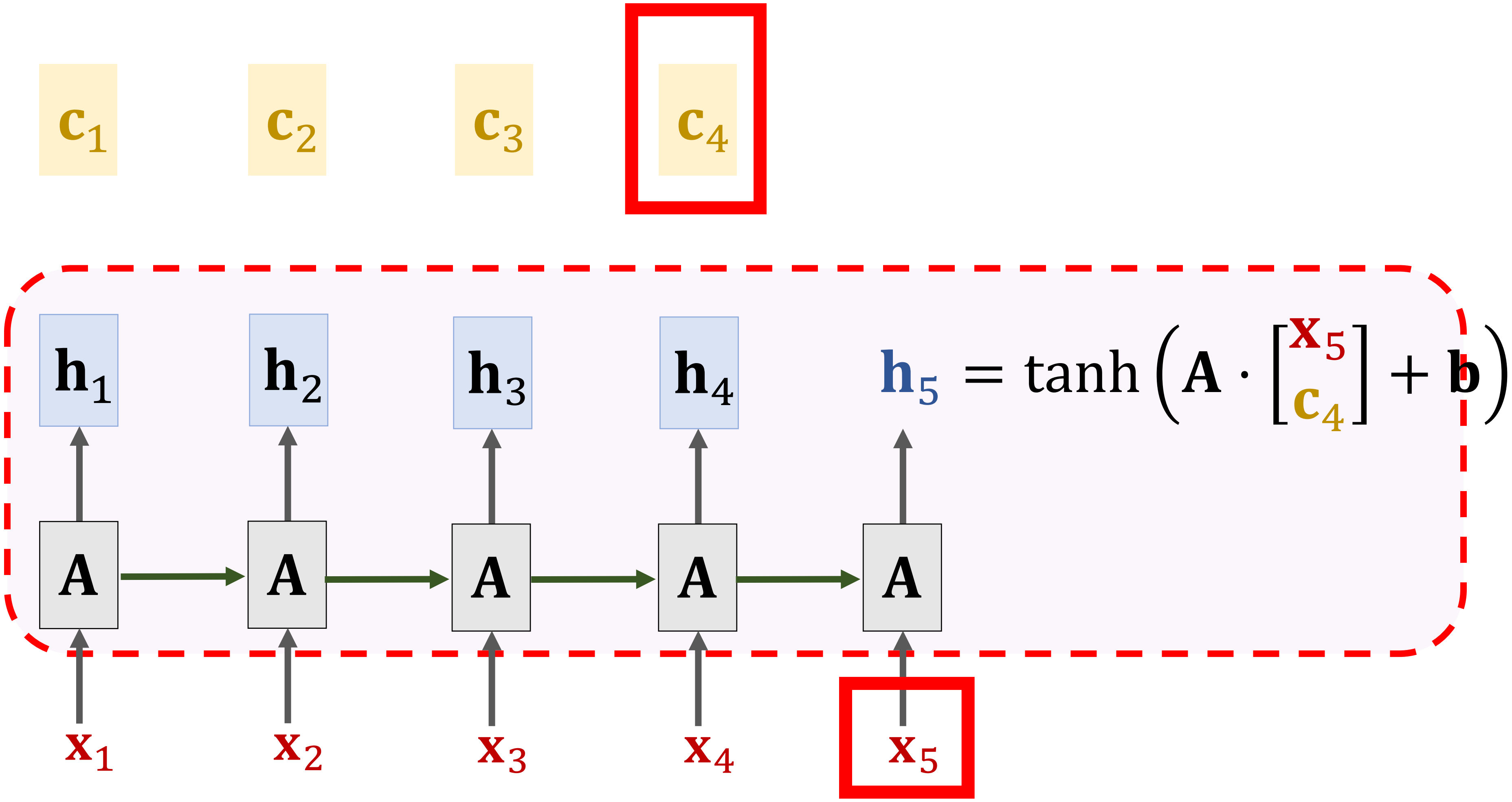


SimpleRNN + Self-Attention

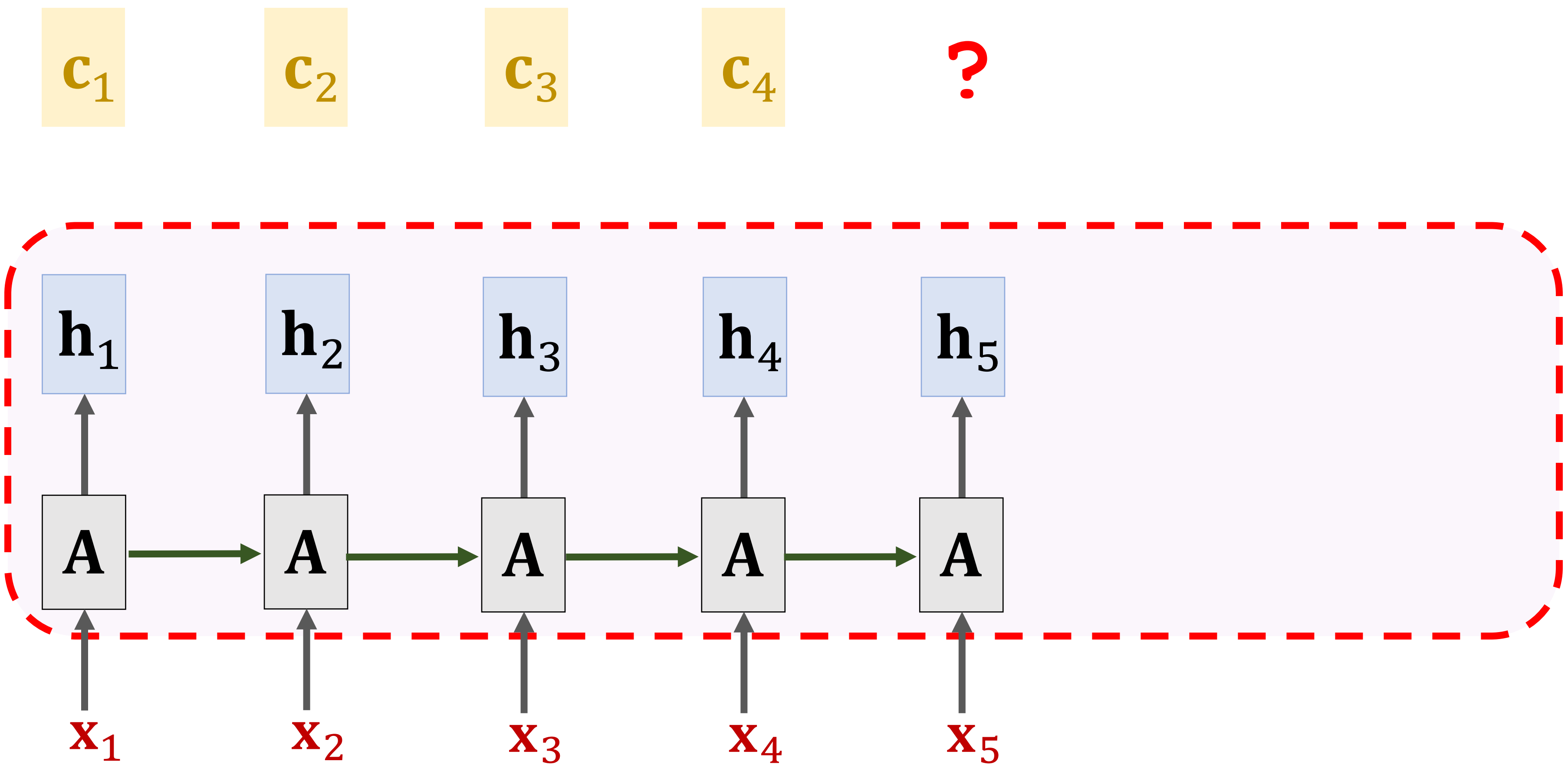
$\mathbf{c}_1$ $\mathbf{c}_2$ $\mathbf{c}_3$ $\mathbf{c}_4 = \alpha_1 \mathbf{h}_1 + \alpha_2 \mathbf{h}_2 + \alpha_3 \mathbf{h}_3 + \alpha_4 \mathbf{h}_4.$



SimpleRNN + Self-Attention

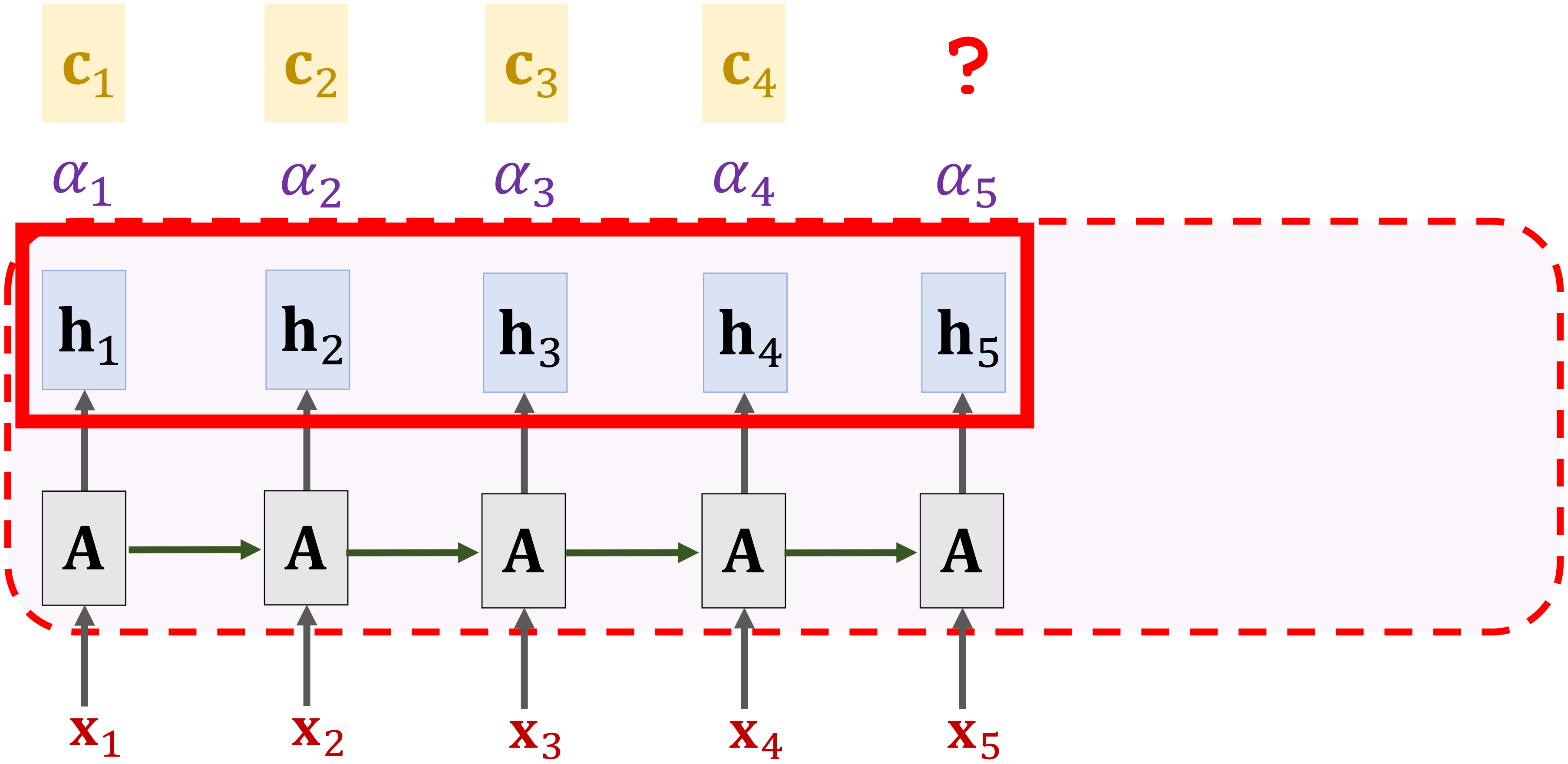


SimpleRNN + Self-Attention

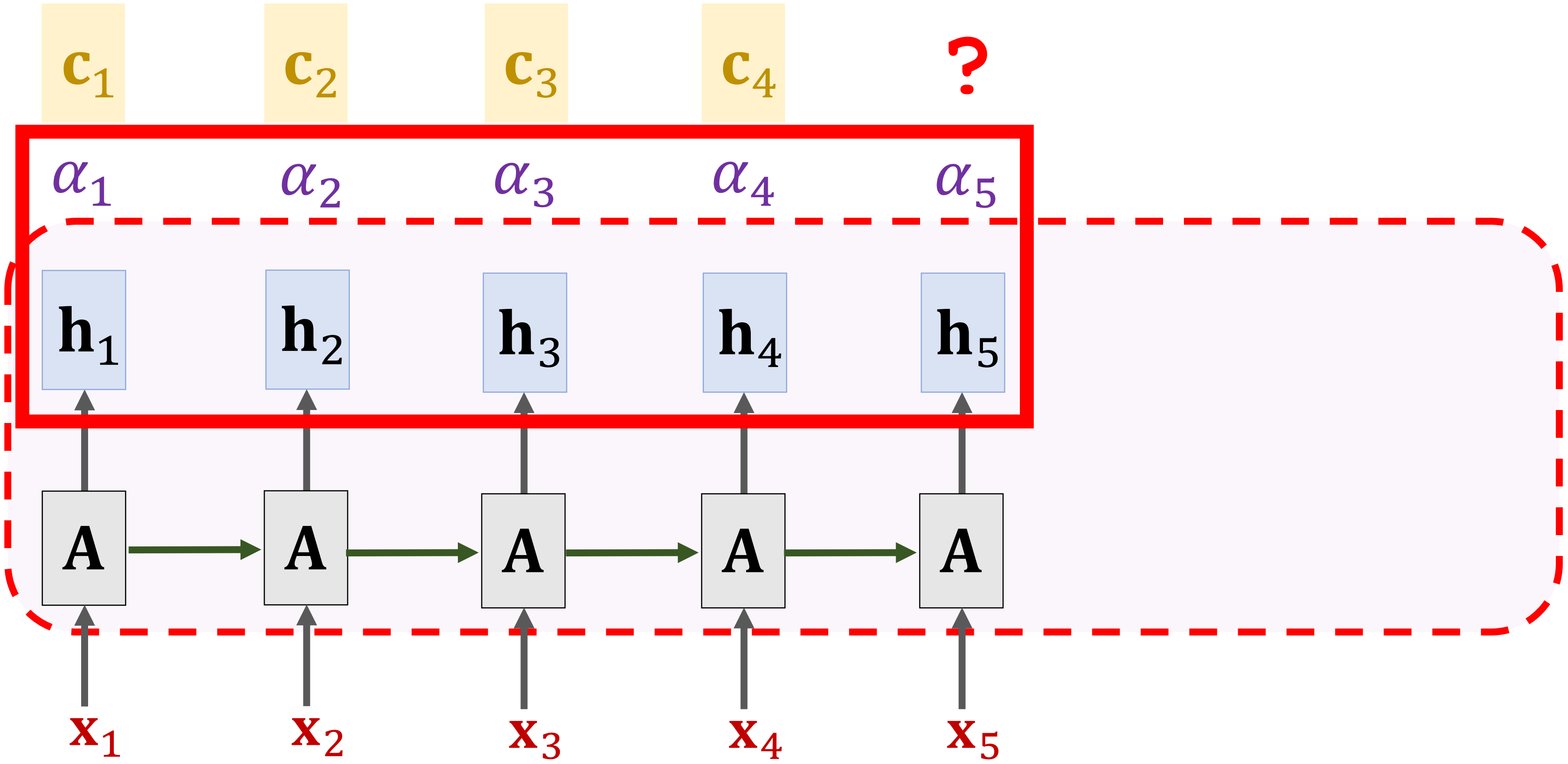


SimpleRNN + Self-Attention

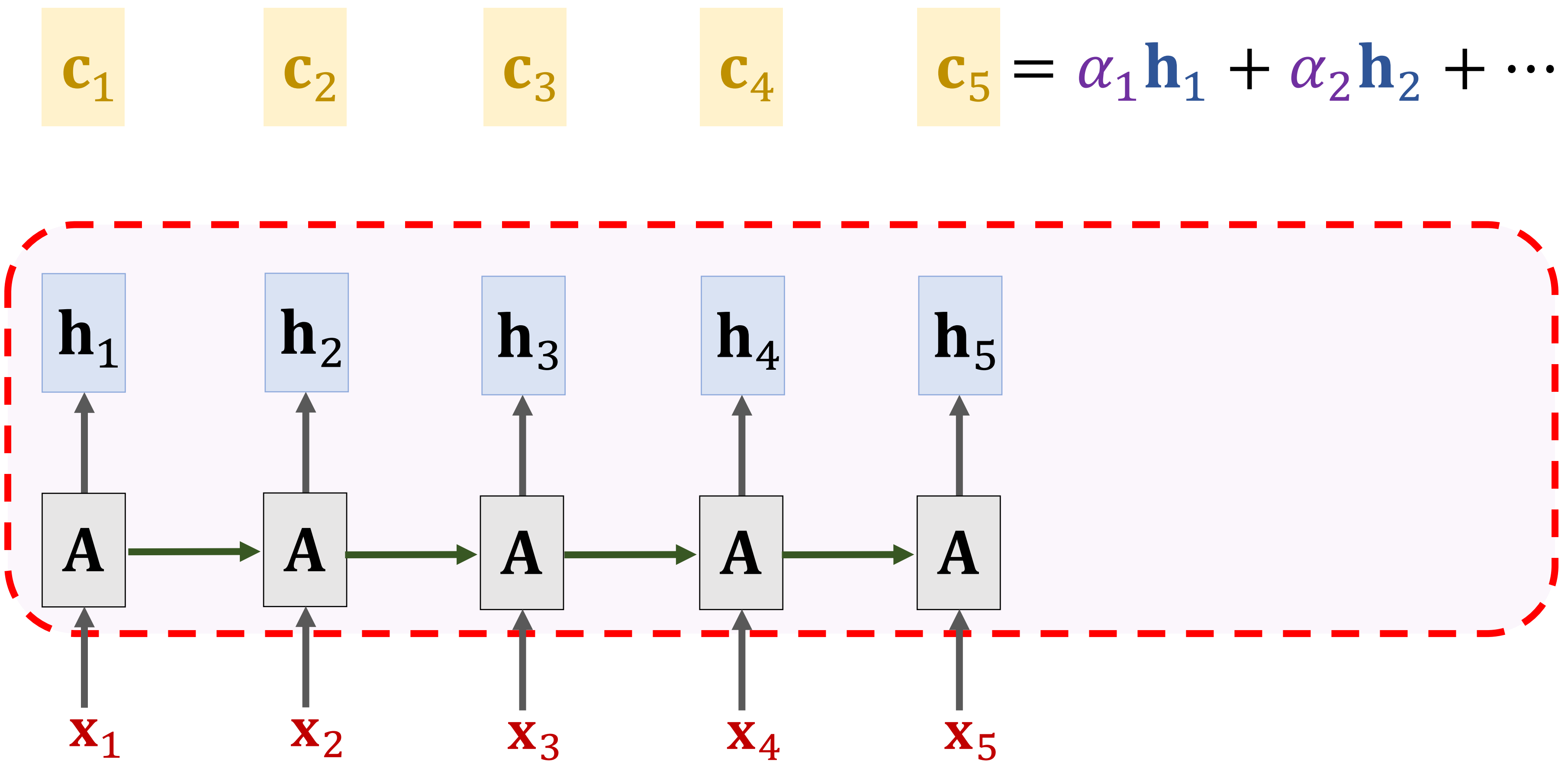
Weights: $\alpha_i = \text{align}(\mathbf{h}_i, \ \mathbf{h}_5).$



SimpleRNN + Self-Attention

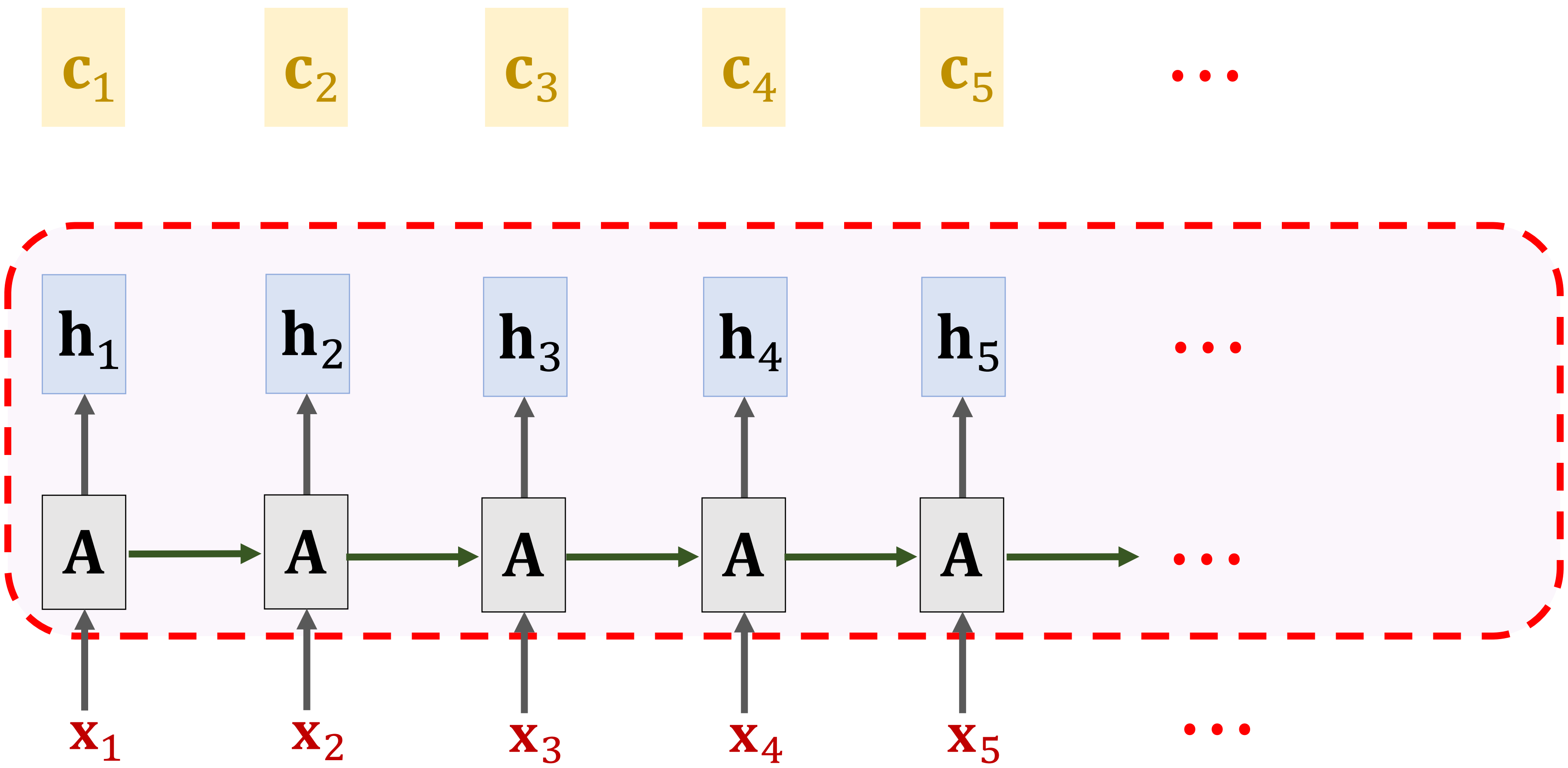


SimpleRNN + Self-Attention



$$\mathbf{c}_5 = \alpha_1 \mathbf{h}_1 + \alpha_2 \mathbf{h}_2 + \cdots + \alpha_5 \mathbf{h}_5.$$

SimpleRNN + Self-Attention



Summary

- With self-attention, RNN is less likely to forget.

Summary

- With self-attention, RNN is less likely to forget.
- Pay attention to the context relevant to the new input.

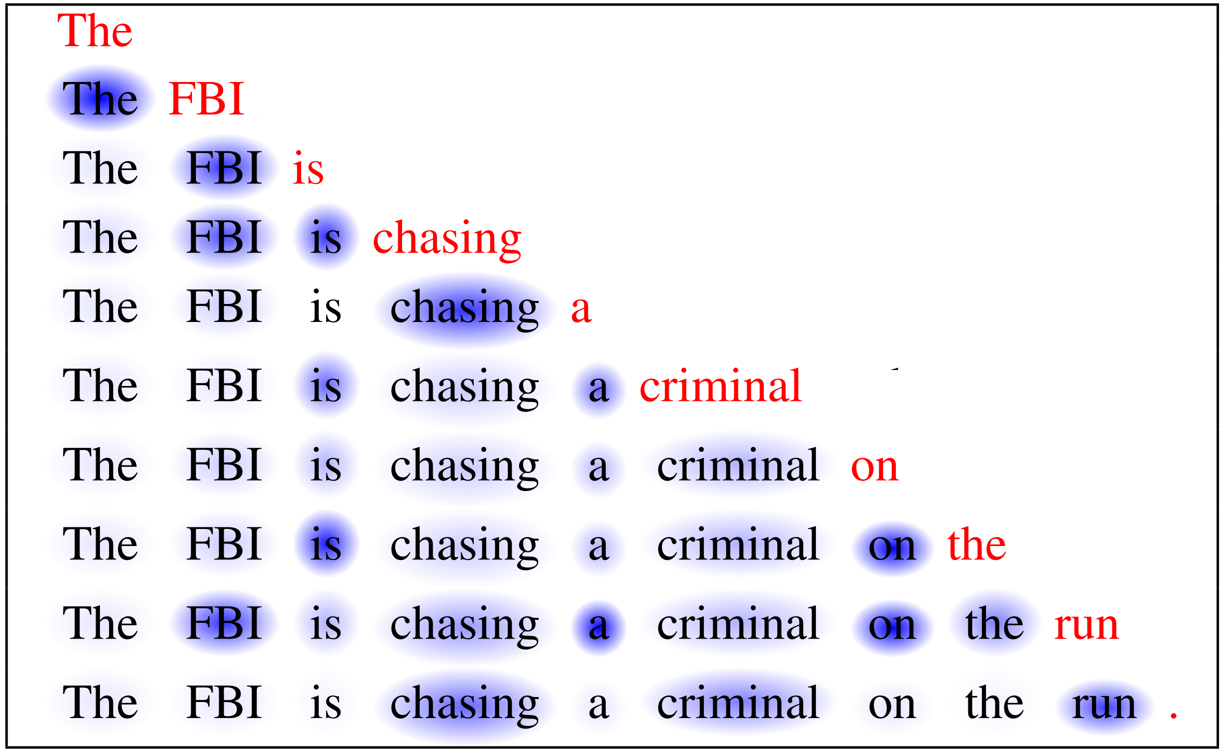


Figure is from the paper " Long Short-Term Memory-Networks for Machine Reading."

Transformer Model: Attention without RNN

Transformer Model

- **Original paper:** Vaswani et al. Attention Is All You Need. In *NIPS*, 2017.

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

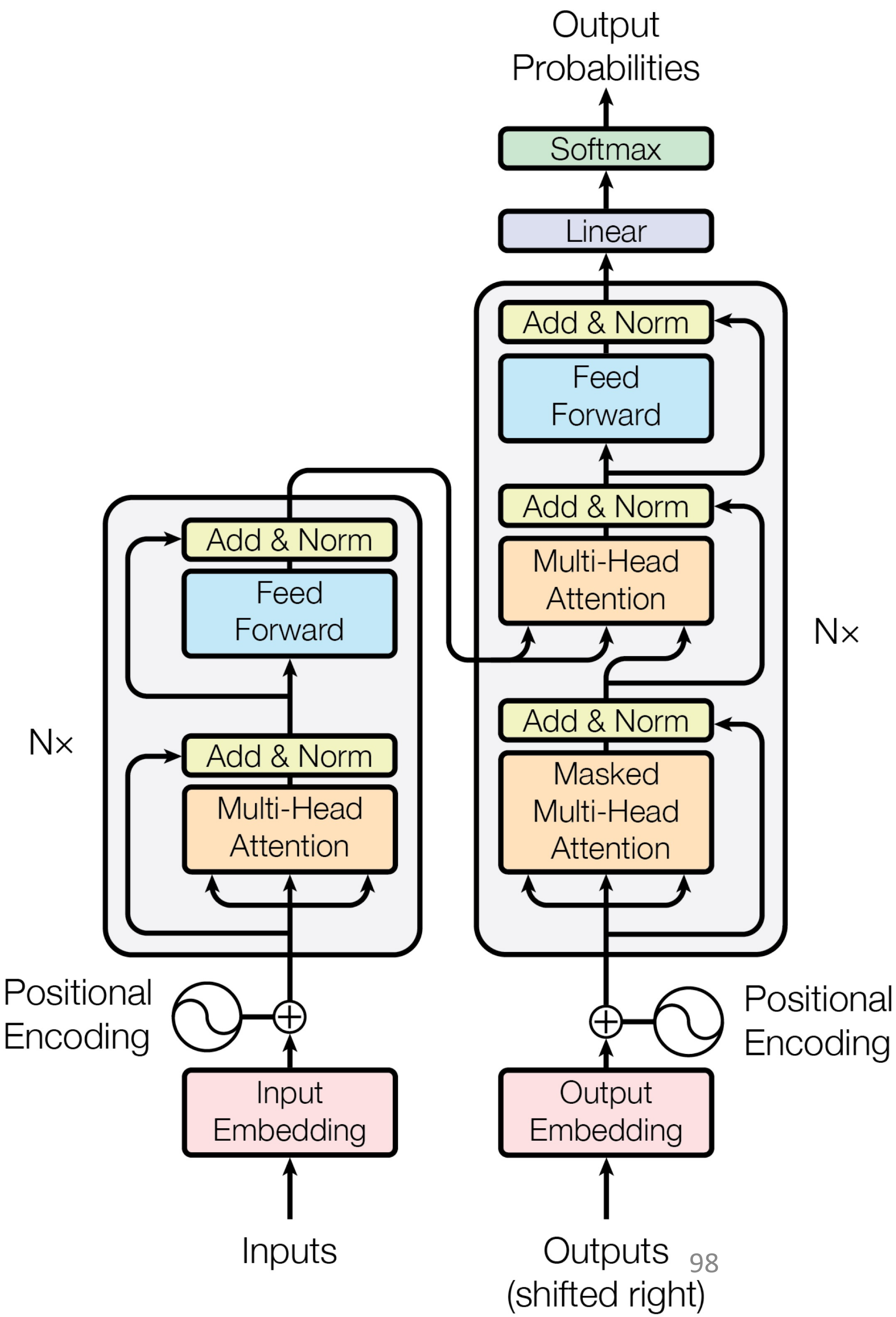
Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

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

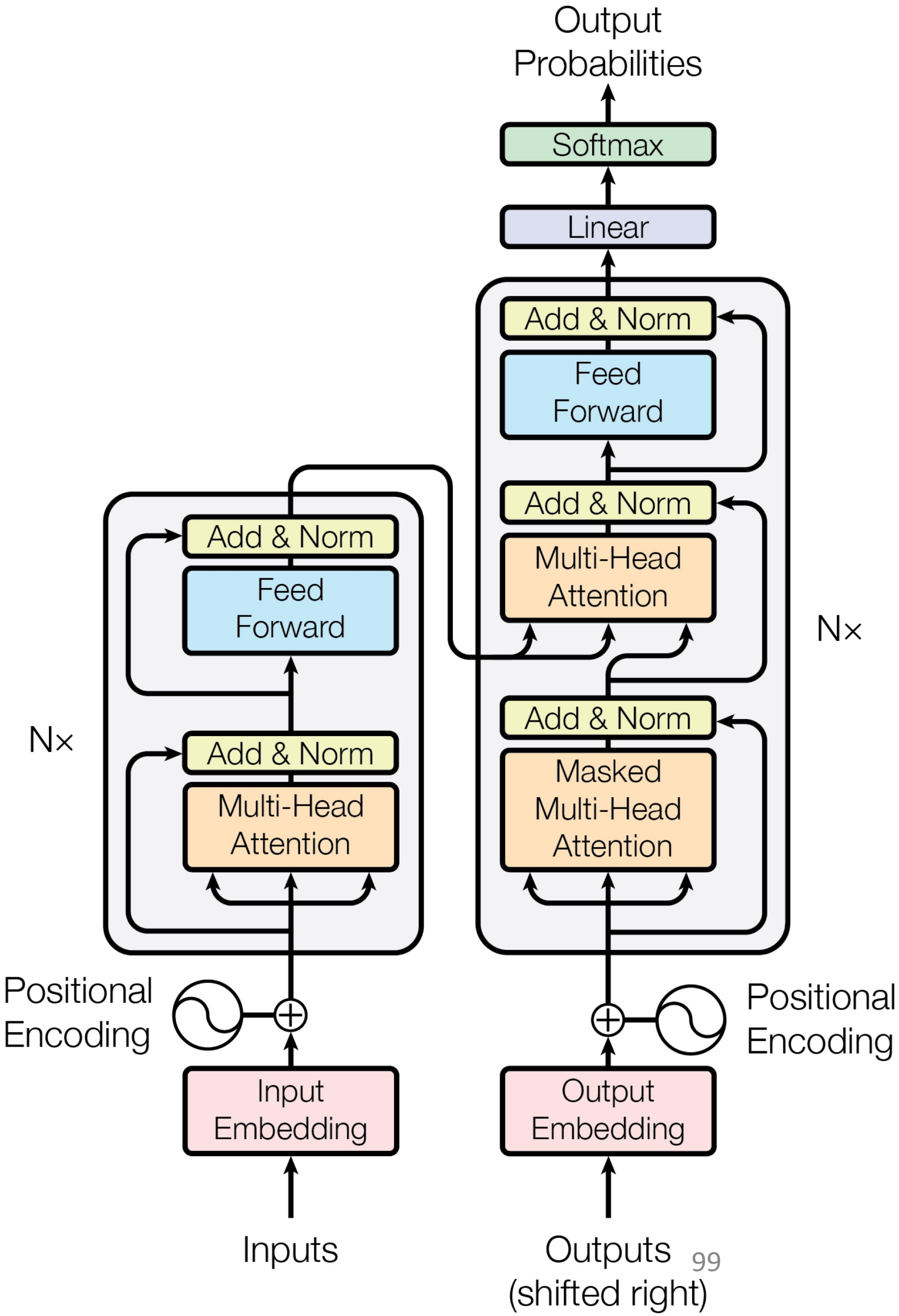
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com



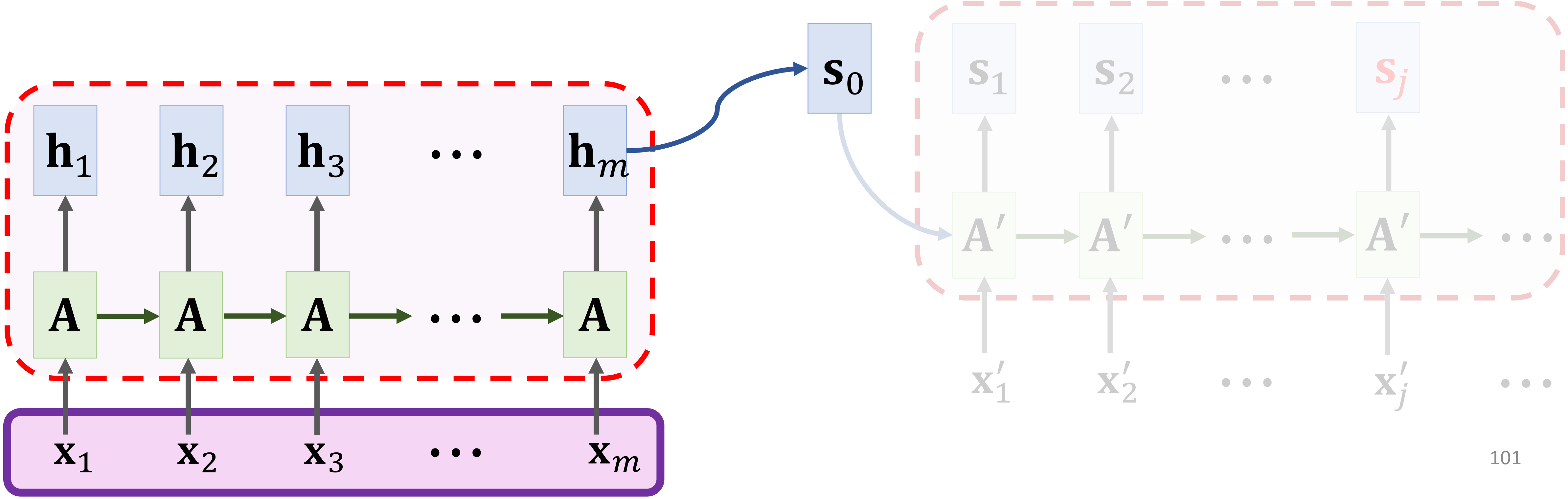
Transformer Model

- Transformer is a Seq2Seq model.
- Transformer is not RNN.
- Purely based on attention and dense layers.
- Higher accuracy than RNNs on large datasets.

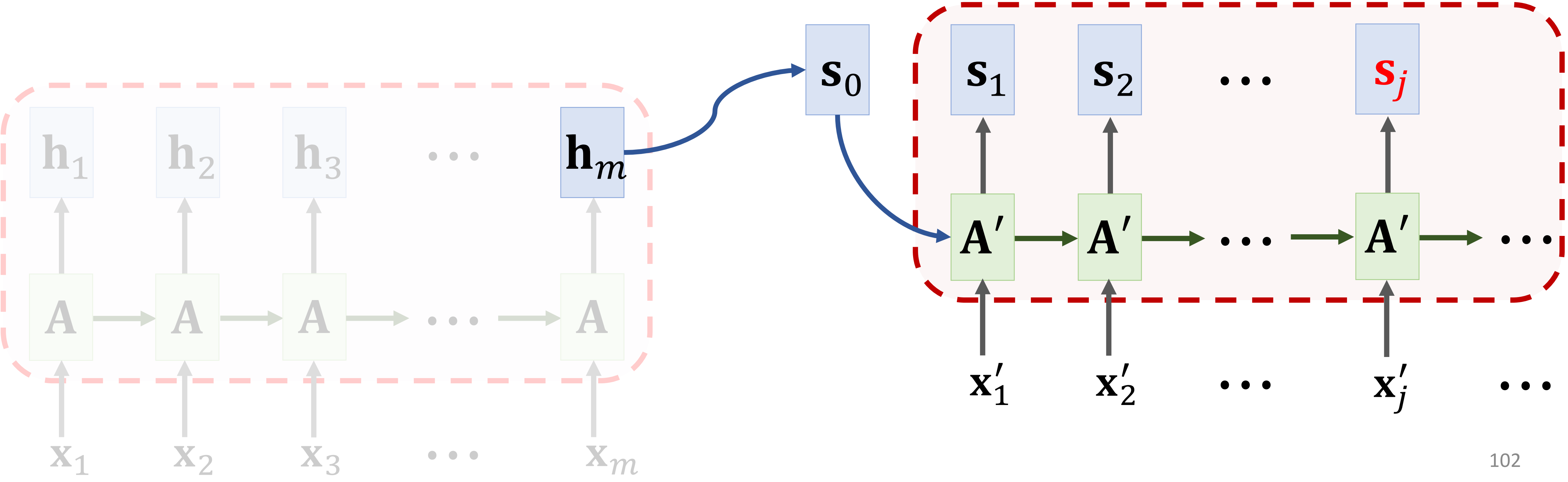


Revisiting Attention for RNN

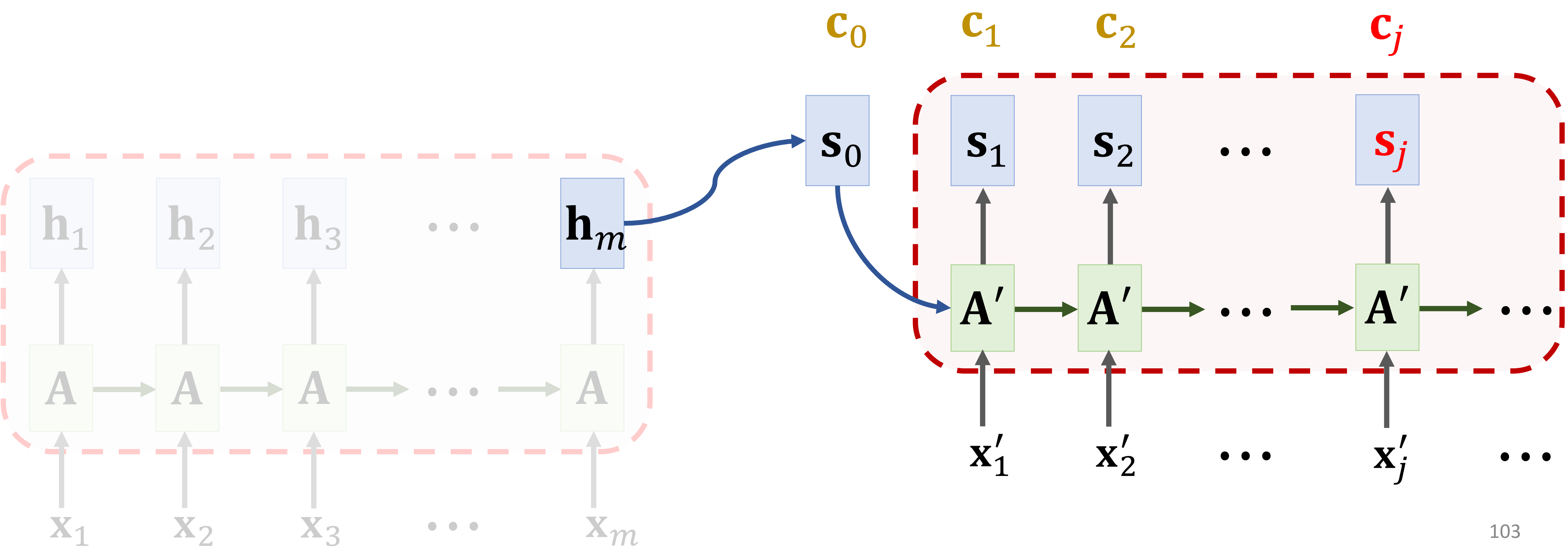
Attention for Seq2Seq Model



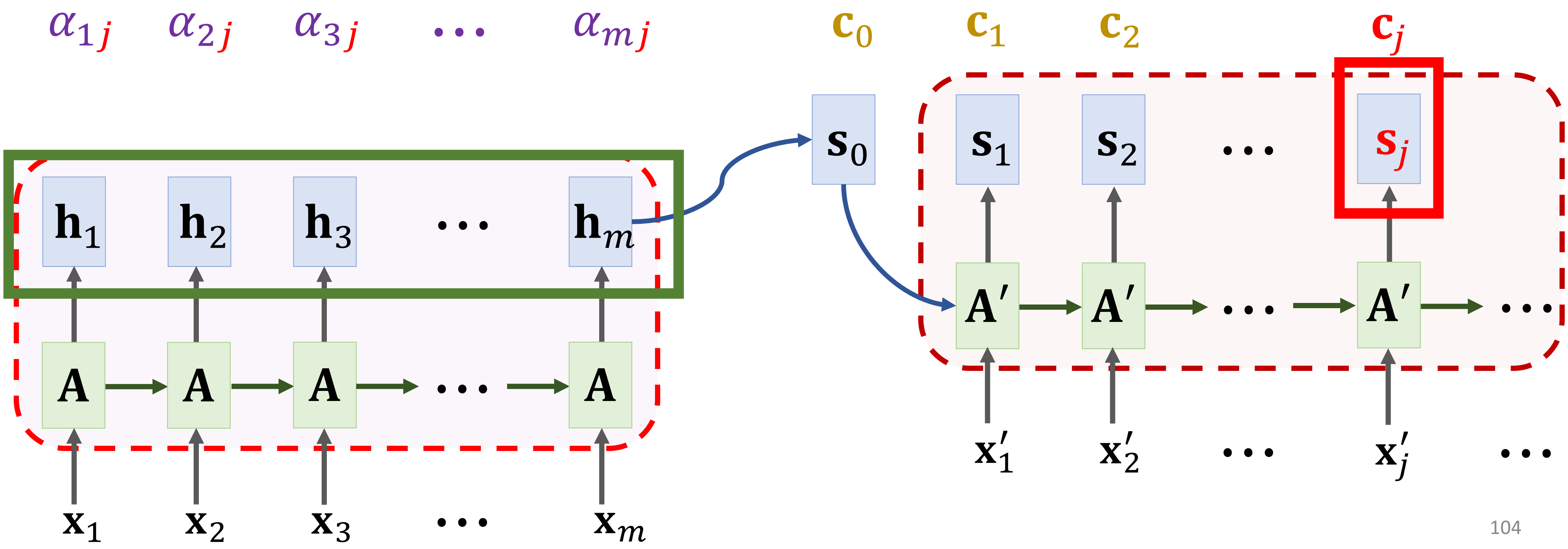
Attention for Seq2Seq Model



Attention for Seq2Seq Model

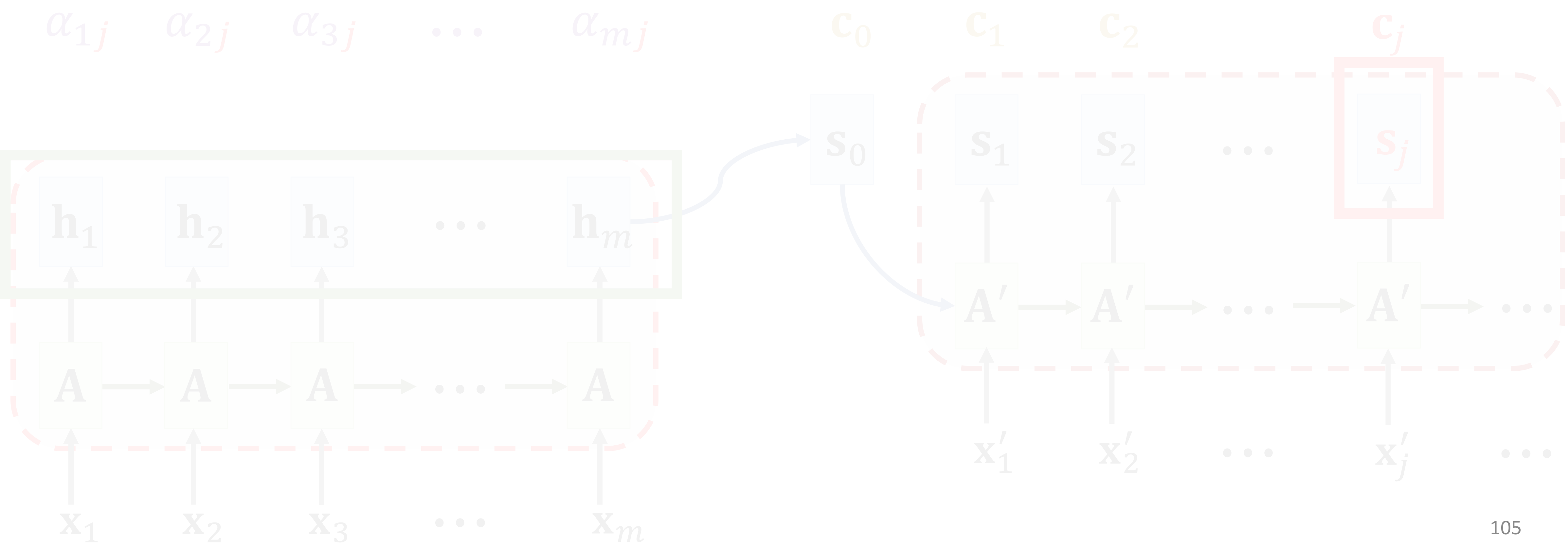


Attention for Seq2Seq Model



Attention for Seq2Seq Model

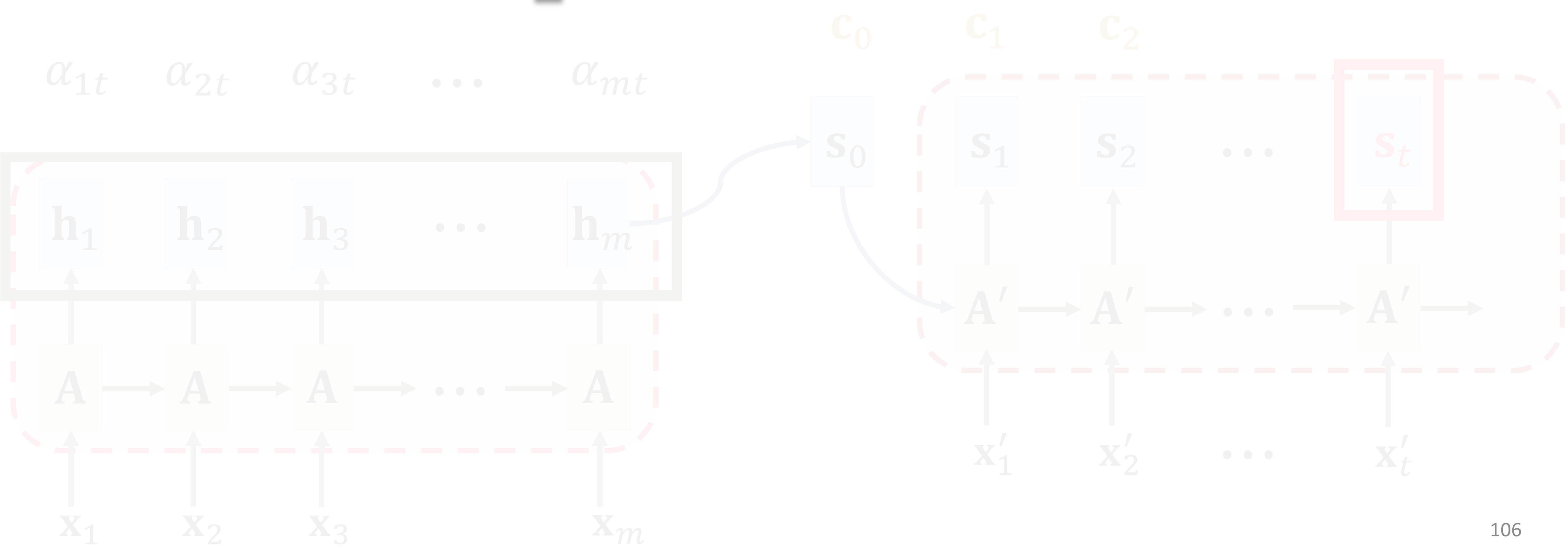
Weights: $\alpha_{ij} = \text{align}(\mathbf{h}_i, \mathbf{s}_j).$



Attention for Seq2Seq Model

Weights: $\alpha_{ij} = \text{align}\left(\mathbf{h}_i, \mathbf{s}_j\right).$

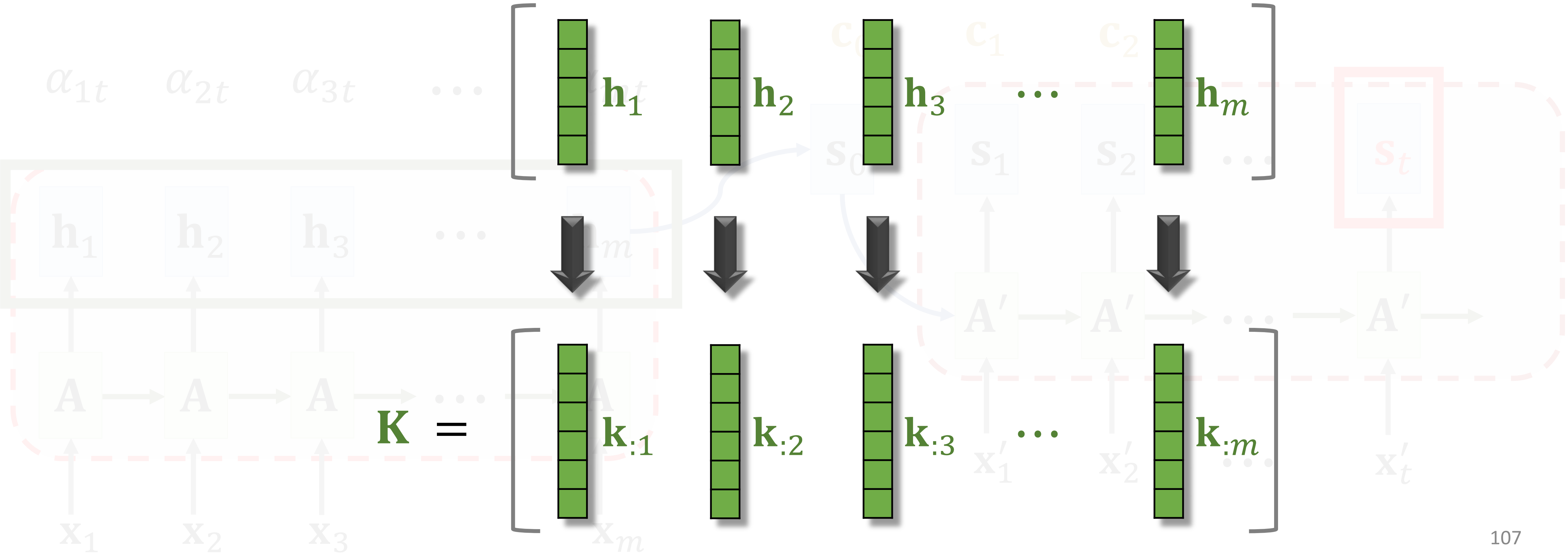
- Compute $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{h}_i.$



Attention for Seq2Seq Model

Weights: $\alpha_{ij} = \mathrm{align}(\mathbf{h}_i, \mathbf{s}_j)$.

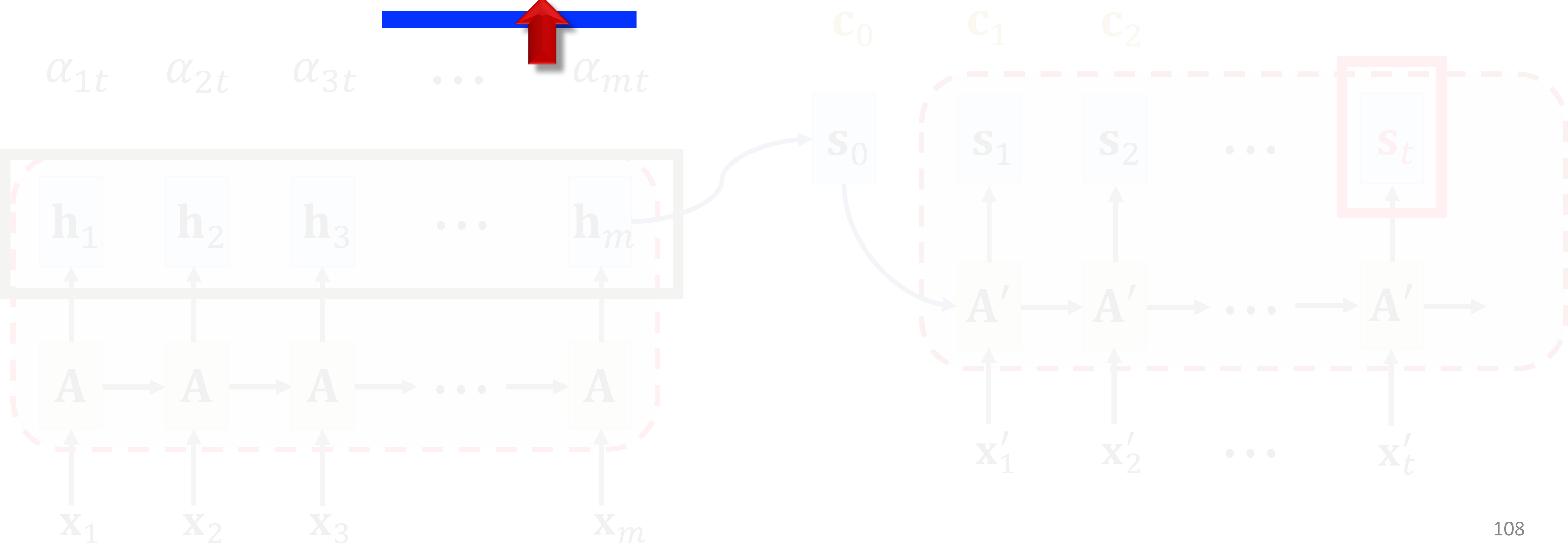
- Compute $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{h}_i$.



Attention for Seq2Seq Model

Weights: $\alpha_{ij} = \text{align}(\mathbf{h}_i, \mathbf{s}_j).$

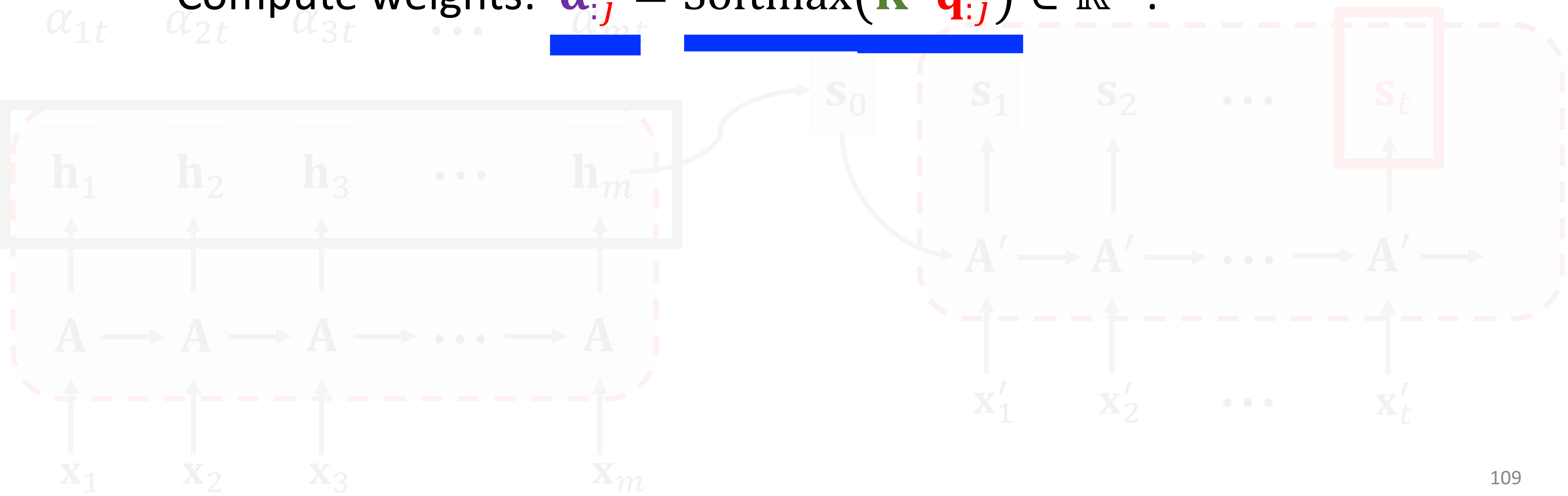
- Compute $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{h}_i.$
- Compute $\mathbf{q}_{:j} = \mathbf{W}_Q \mathbf{s}_j.$



Attention for Seq2Seq Model

Weights: $\alpha_{ij} = \mathrm{align}\left(\mathbf{h}_i, \mathbf{s}_j\right).$

- Compute $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{h}_i$.
- Compute $\mathbf{q}_{:j} = \mathbf{W}_Q \mathbf{s}_j$.
- Compute weights: $\boldsymbol{\alpha}_{:j} = \mathrm{Softmax}\left(\mathbf{K}^T \mathbf{q}_{:j}\right) \in \mathbb{R}^m$.

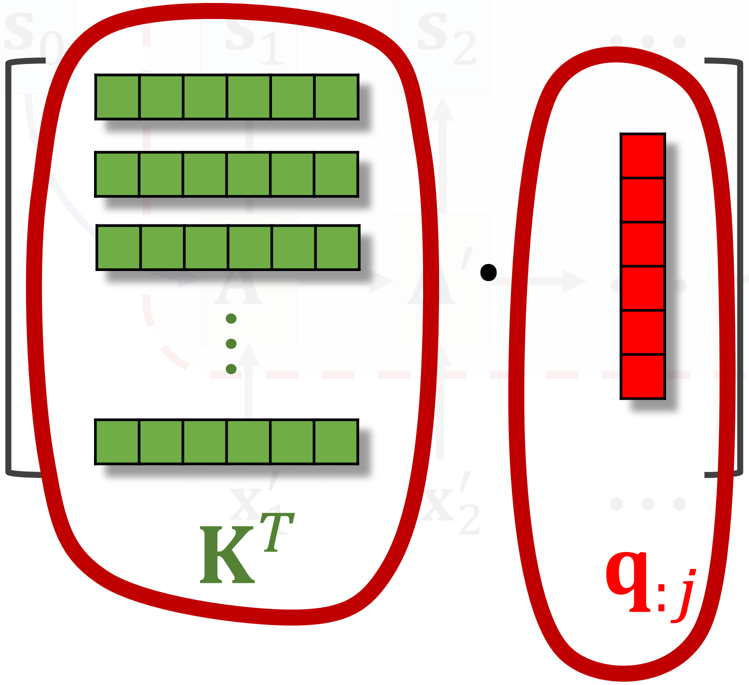


Attention for Seq2Seq Model

Weights: $\alpha_{ij} = \mathrm{align}\left(\mathbf{h}_i, \mathbf{s}_j\right).$

- Compute $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{h}_i.$
- Compute $\mathbf{q}_{:j} = \mathbf{W}_Q \mathbf{s}_j.$
- Compute weights: $\boldsymbol{\alpha}_{:j} = \mathrm{Softmax}\left(\mathbf{K}^T \mathbf{q}_{:j}\right) \in \mathbb{R}^m.$

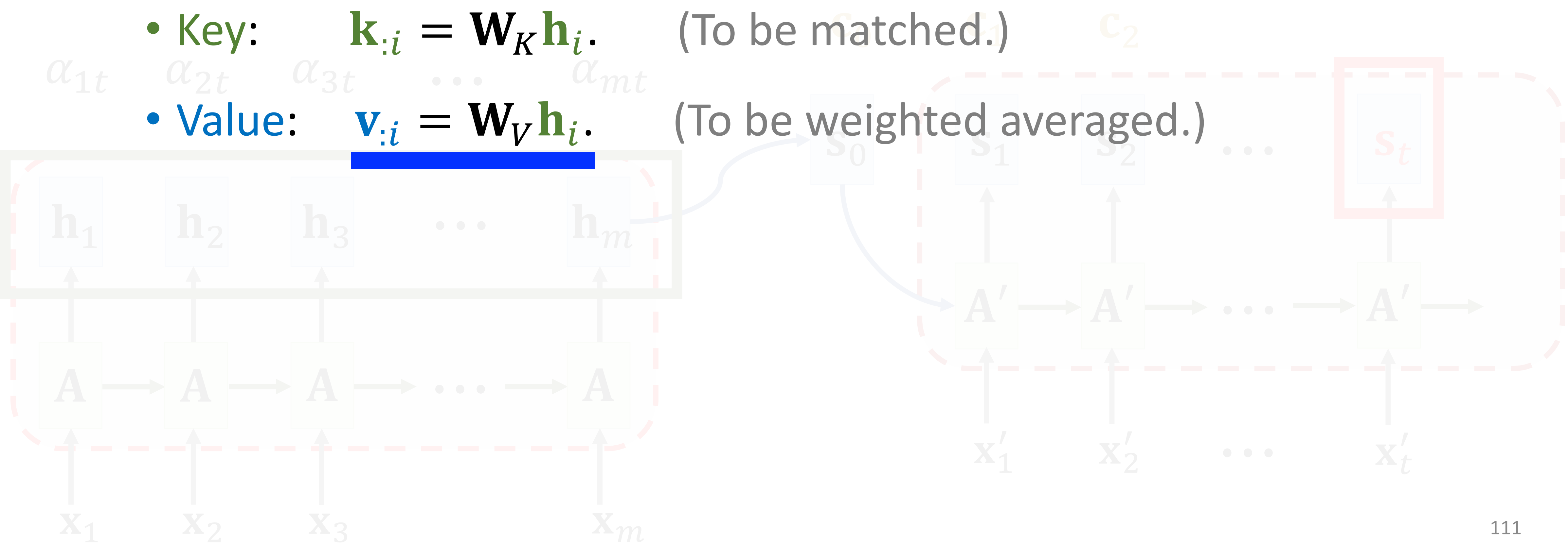
$$\boldsymbol{\alpha}_{:j} = \begin{bmatrix} \alpha_{1j} \\ \alpha_{2j} \\ \vdots \\ \alpha_{mj} \end{bmatrix} = \mathrm{Softmax}$$



Attention for Seq2Seq Model

Weights: $\alpha_{ij} = \mathrm{align}(\mathbf{h}_i, \mathbf{s}_j)$.

- Query: $\mathbf{q}_{:j} = \mathbf{W}_Q \mathbf{s}_j$. (To match others.)
- Key: $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{h}_i$. (To be matched.)
- Value: $\underline{\mathbf{v}_{:i} = \mathbf{W}_V \mathbf{h}_i}$. (To be weighted averaged.)



Attention for Seq2Seq Model

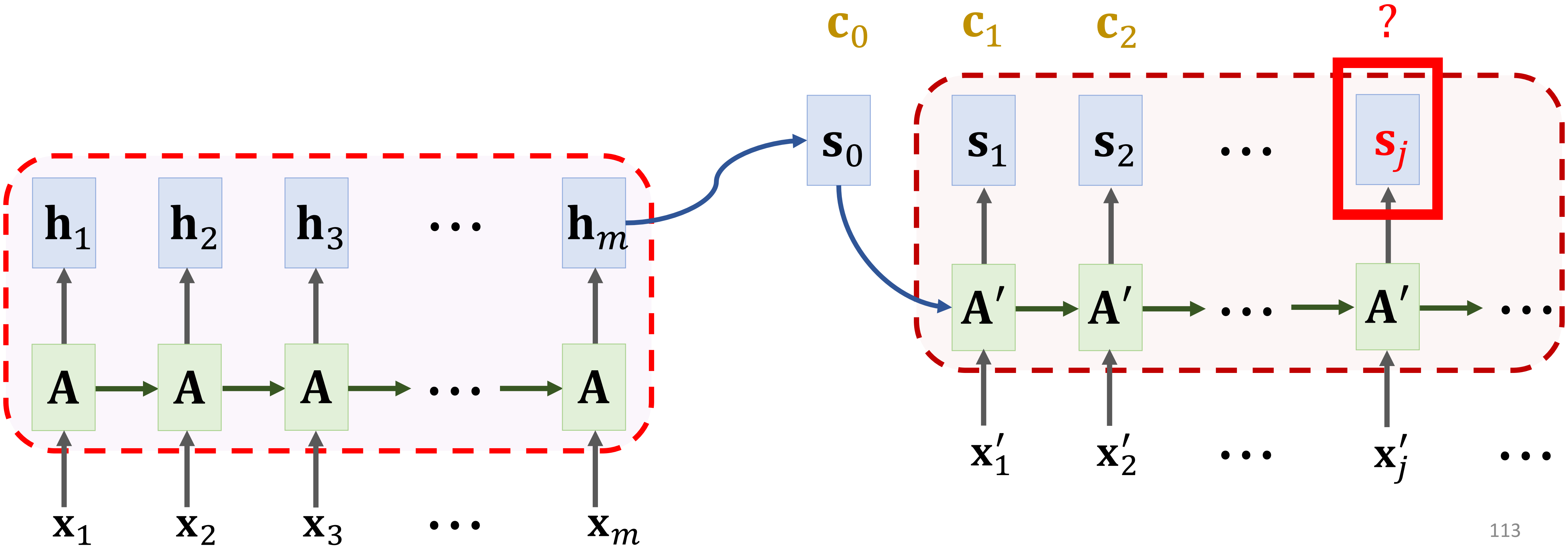
Weights: $\alpha_{ij} = \mathrm{align}(\mathbf{h}_i, \mathbf{s}_j)$.

- Query: $\mathbf{q}_{:j} = \mathbf{W}_Q \mathbf{s}_j$. (To match others.)
- Key: $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{h}_i$. (To be matched.)
- Value: $\mathbf{v}_{:i} = \mathbf{W}_V \mathbf{h}_i$. (To be weighted averaged.)

Context vector: $\mathbf{c}_j = \alpha_{1j}\mathbf{v}_{:1} + \cdots + \alpha_{mj}\mathbf{v}_{:m}$.

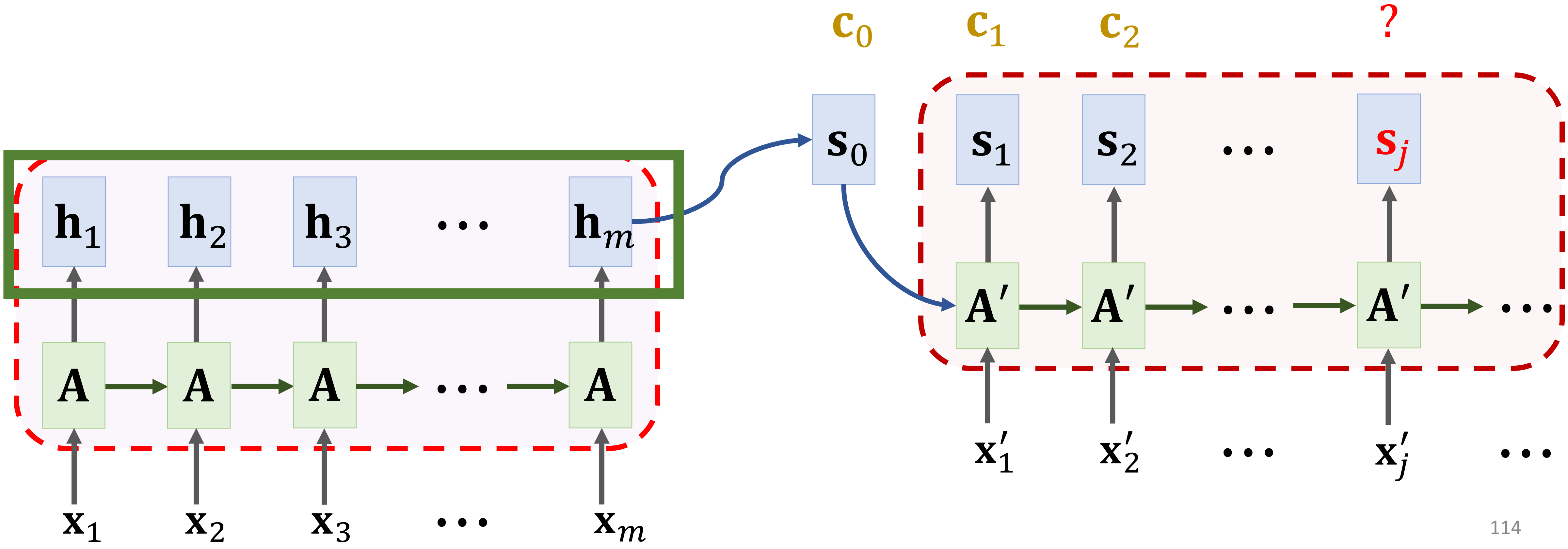
Attention for Seq2Seq Model

Query: $\mathbf{q}_{:j} = \mathbf{W}_Q \mathbf{s}_j$, Key: $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{h}_i$, Value: $\mathbf{v}_{:i} = \mathbf{W}_V \mathbf{h}_i$.



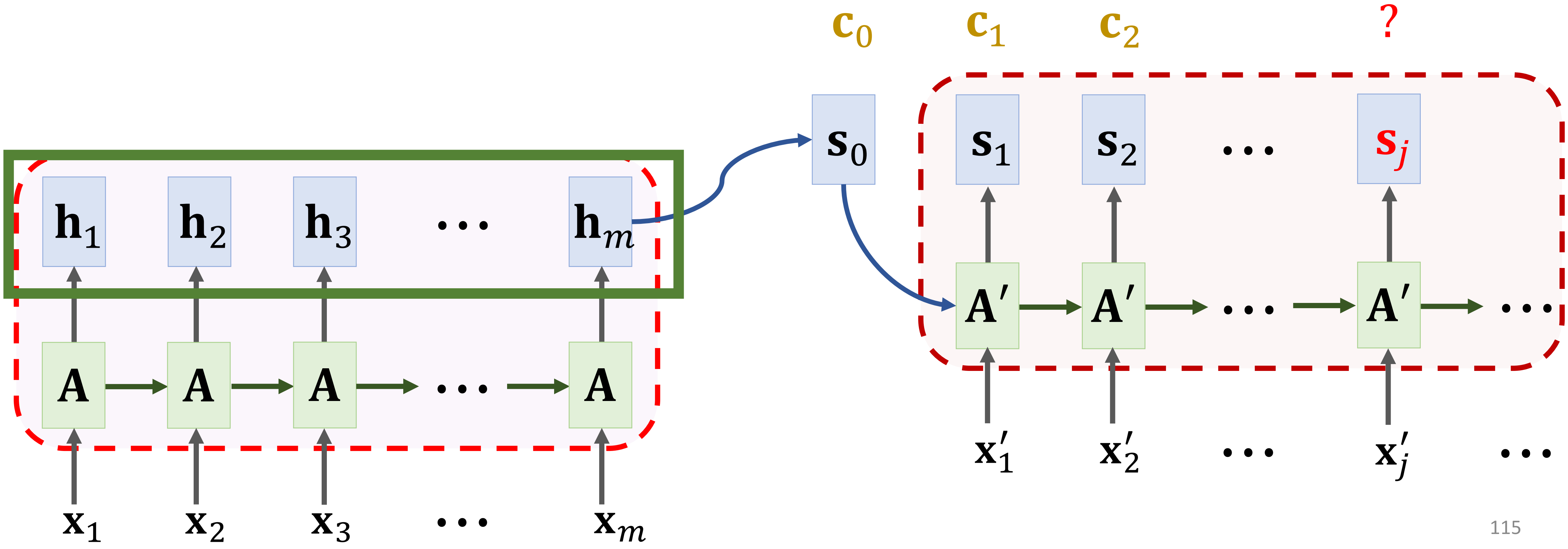
Attention for Seq2Seq Model

Query: $\mathbf{q}_{:j} = \mathbf{W}_Q \mathbf{s}_j$, Key: $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{h}_i$, Value: $\mathbf{v}_{:i} = \mathbf{W}_V \mathbf{h}_i$.



Attention for Seq2Seq Model

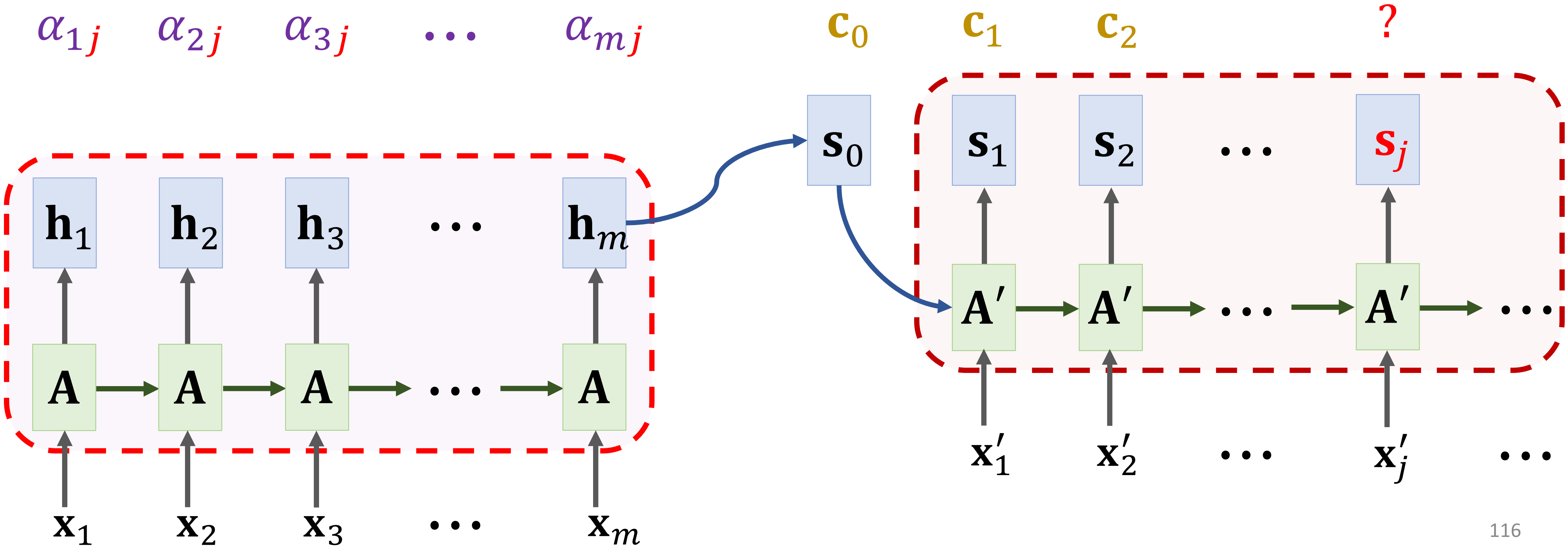
Query: $\mathbf{q}_{:j} = \mathbf{W}_Q \mathbf{s}_j$, Key: $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{h}_i$, Value: $\mathbf{v}_{:i} = \mathbf{W}_V \mathbf{h}_i$.



Attention for Seq2Seq Model

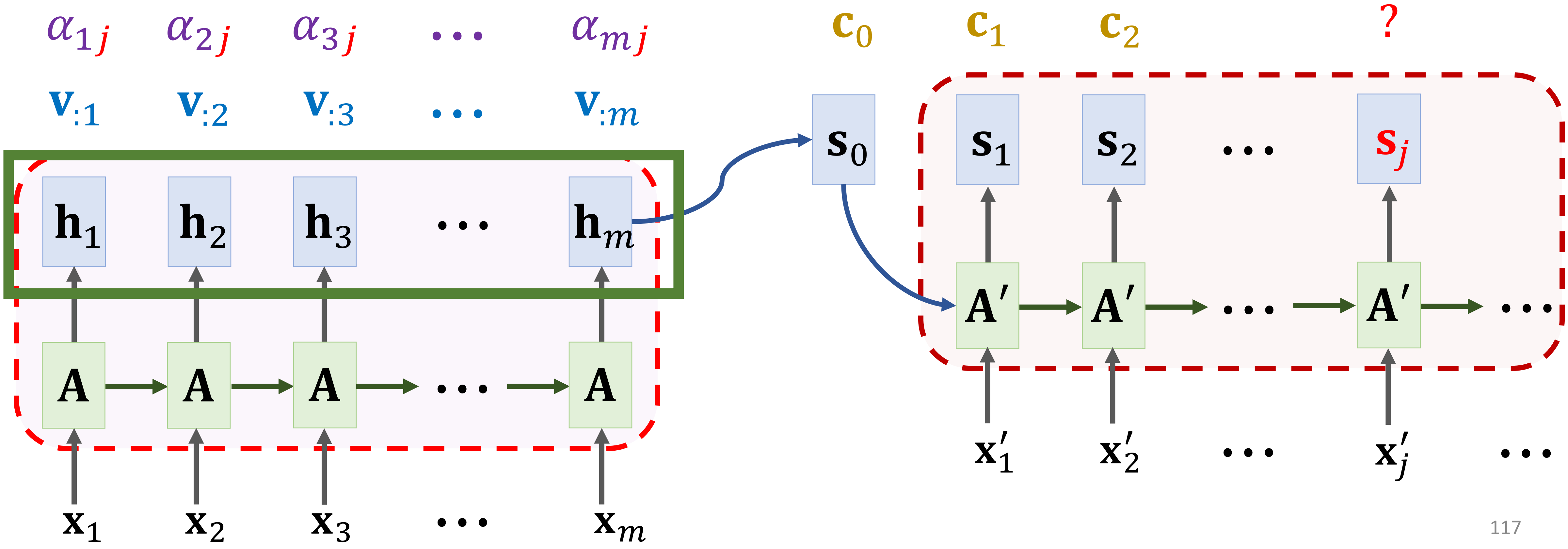
Query: $\mathbf{q}_{:j} = \mathbf{W}_Q \mathbf{s}_j$, Key: $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{h}_i$, Value: $\mathbf{v}_{:i} = \mathbf{W}_V \mathbf{h}_i$.

Weights: $\boldsymbol{\alpha}_{:j} = \mathrm{Softmax}\left(\mathbf{K}^T \mathbf{q}_{:j}\right) \in \mathbb{R}^m$.

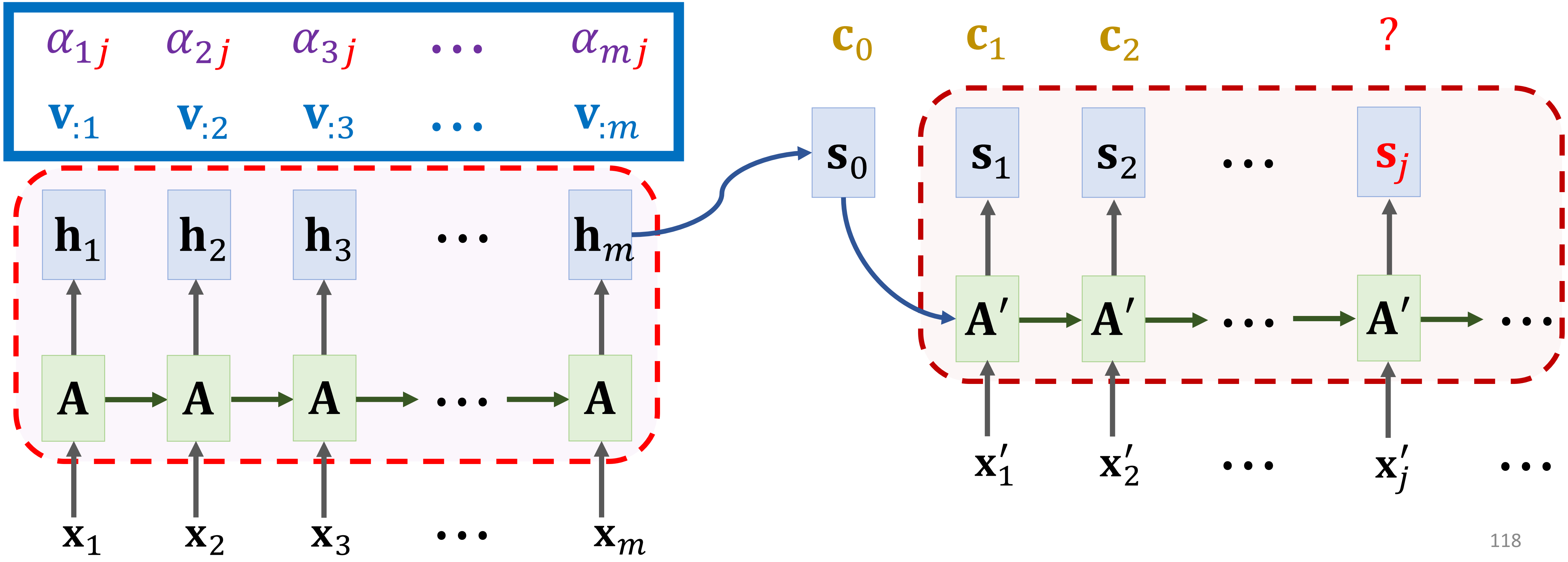


Attention for Seq2Seq Model

Query: $\mathbf{q}_{:j} = \mathbf{W}_Q \mathbf{s}_j$, Key: $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{h}_i$, Value: $\mathbf{v}_{:i} = \mathbf{W}_V \mathbf{h}_i$.

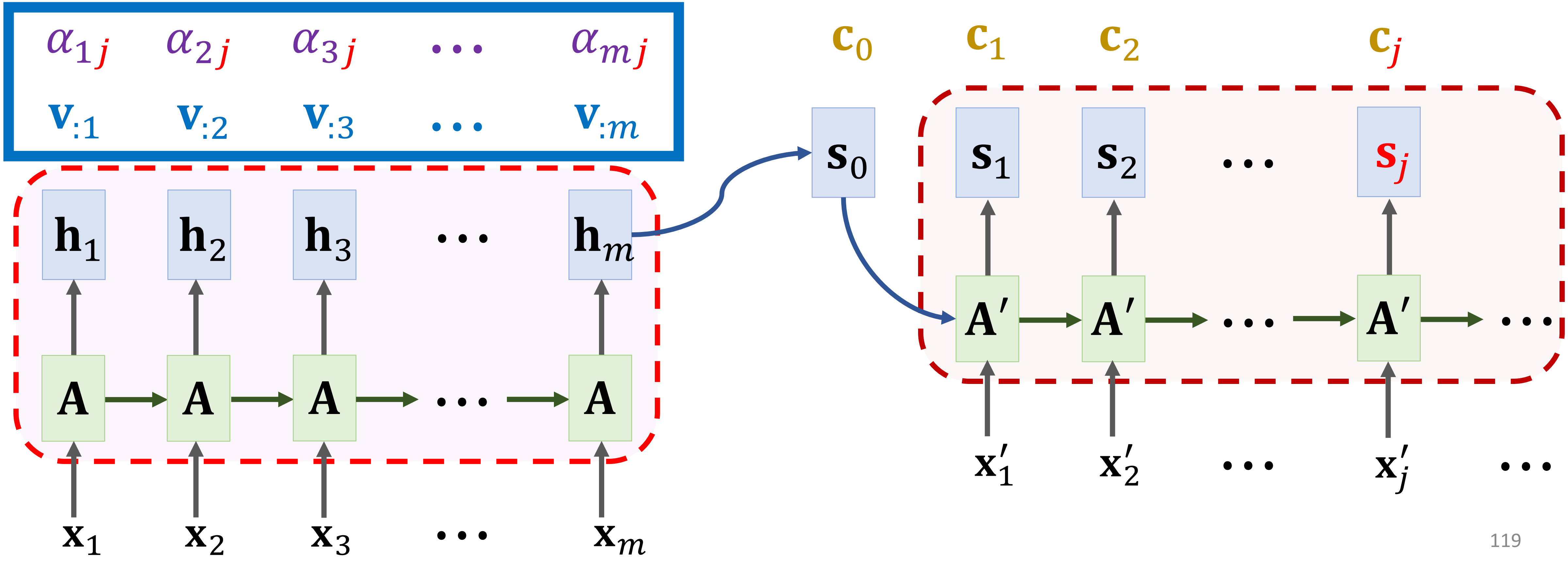


Attention for Seq2Seq Model



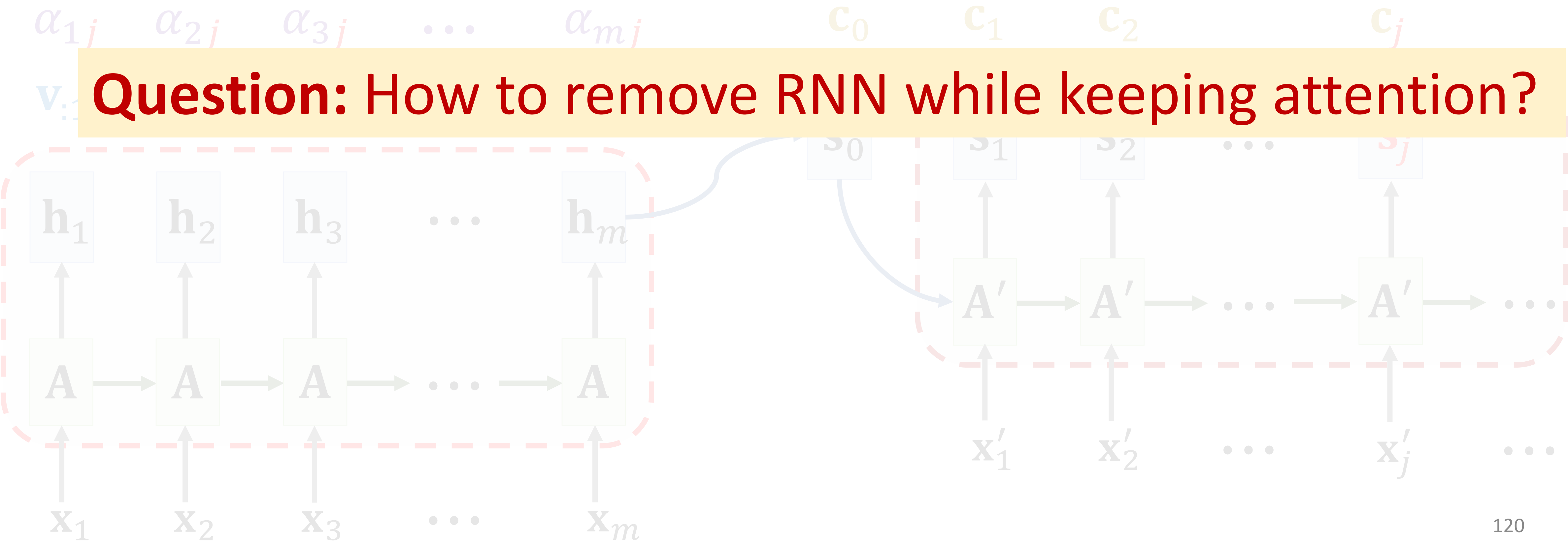
Attention for Seq2Seq Model

Context vector: $\mathbf{c}_j = \alpha_{1j}\mathbf{v}_{:1} + \cdots + \alpha_{mj}\mathbf{v}_{:m}.$



Attention for Seq2Seq Model

Context vector: $\mathbf{c}_j = \alpha_{1j}\mathbf{v}_{:1} + \cdots + \alpha_{mj}\mathbf{v}_{:m}.$



Question: How to remove RNN while keeping attention?

Attention without RNN

Attention Layer

- We study Seq2Seq model (encoder + decoder).
- Encoder's inputs are vectors $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m$.
- Decoder's inputs are vectors $\mathbf{x}'_1, \mathbf{x}'_2, \cdots, \mathbf{x}'_t$.

Encoder's inputs:

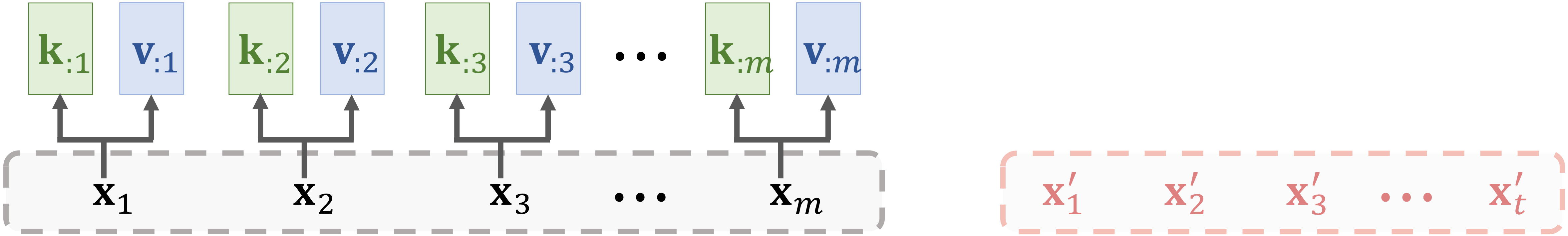
$\mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_3 \quad \cdots \quad \mathbf{x}_m$

Decoder's inputs:

$\mathbf{x}'_1 \quad \mathbf{x}'_2 \quad \mathbf{x}'_3 \quad \cdots \quad \mathbf{x}'_t$

Attention Layer

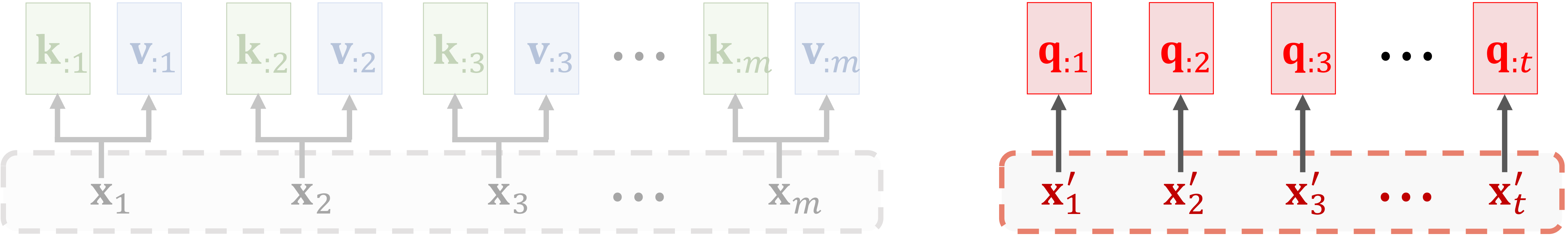
- Keys and values are based on encoder's inputs $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m$.
- Key: $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{x}_i$.
- Value: $\mathbf{v}_{:i} = \mathbf{W}_V \mathbf{x}_i$.



Attention Layer

- Keys and values are based on encoder's inputs $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m$.
- Key: $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{x}_i$.
- Value: $\mathbf{v}_{:i} = \mathbf{W}_V \mathbf{x}_i$.

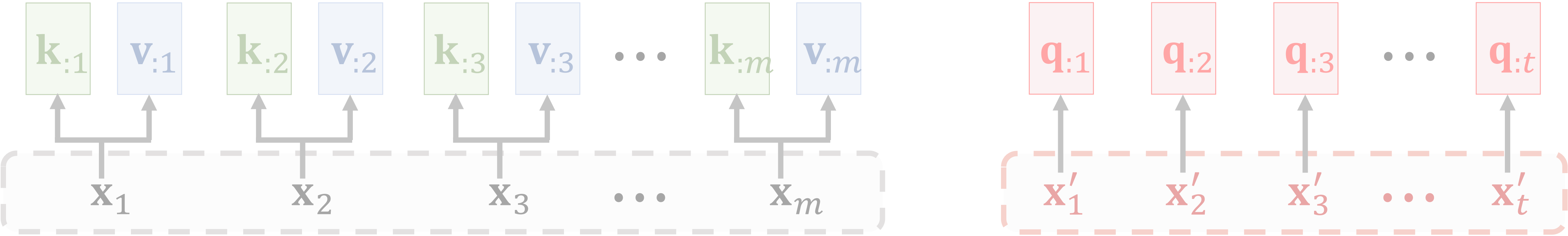
- Queries are based on decoder's inputs $\mathbf{x}'_1, \mathbf{x}'_2, \cdots, \mathbf{x}'_t$.
- Query: $\mathbf{q}_{:j} = \mathbf{W}_Q \mathbf{x}'_j$.



Attention Layer

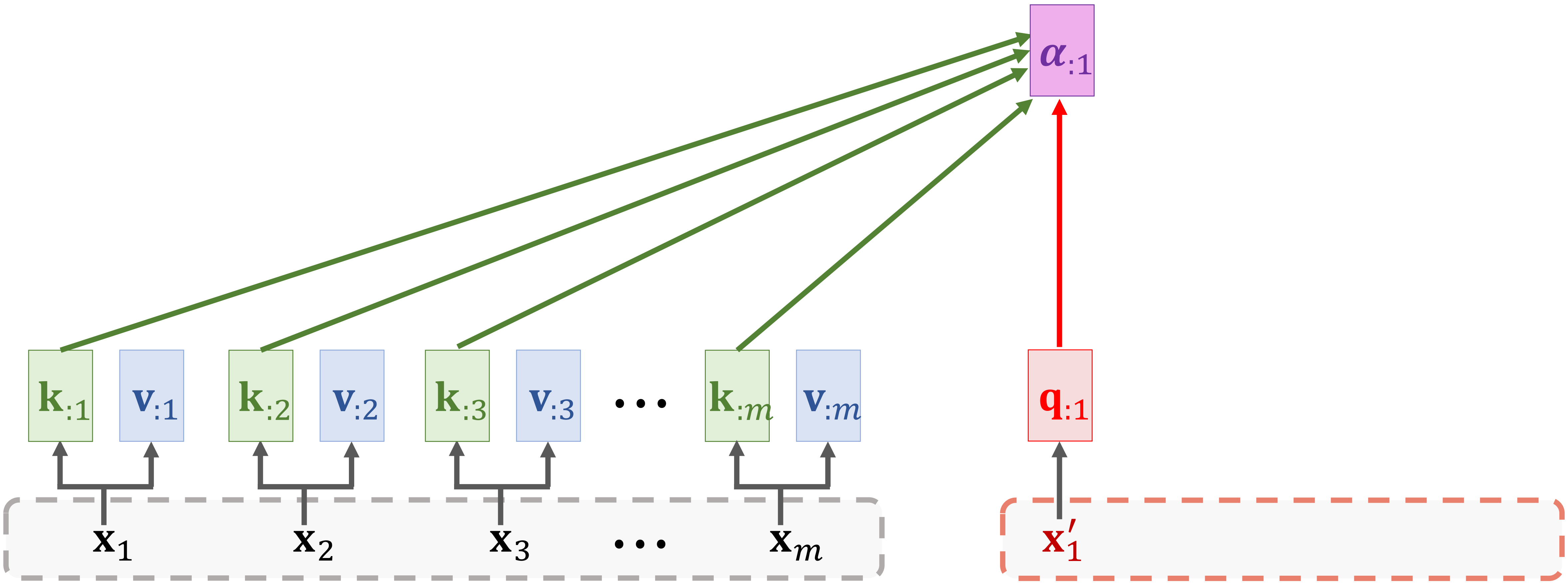
- Keys and values are based on encoder's inputs $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m$.
- Key: $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{x}_i$.
- Value: $\mathbf{v}_{:i} = \mathbf{W}_V \mathbf{x}_i$.

- Queries are based on decoder's inputs $\mathbf{x}'_1, \mathbf{x}'_2, \cdots, \mathbf{x}'_t$.
- Query: $\mathbf{q}_{:j} = \mathbf{W}_Q \mathbf{x}'_j$.



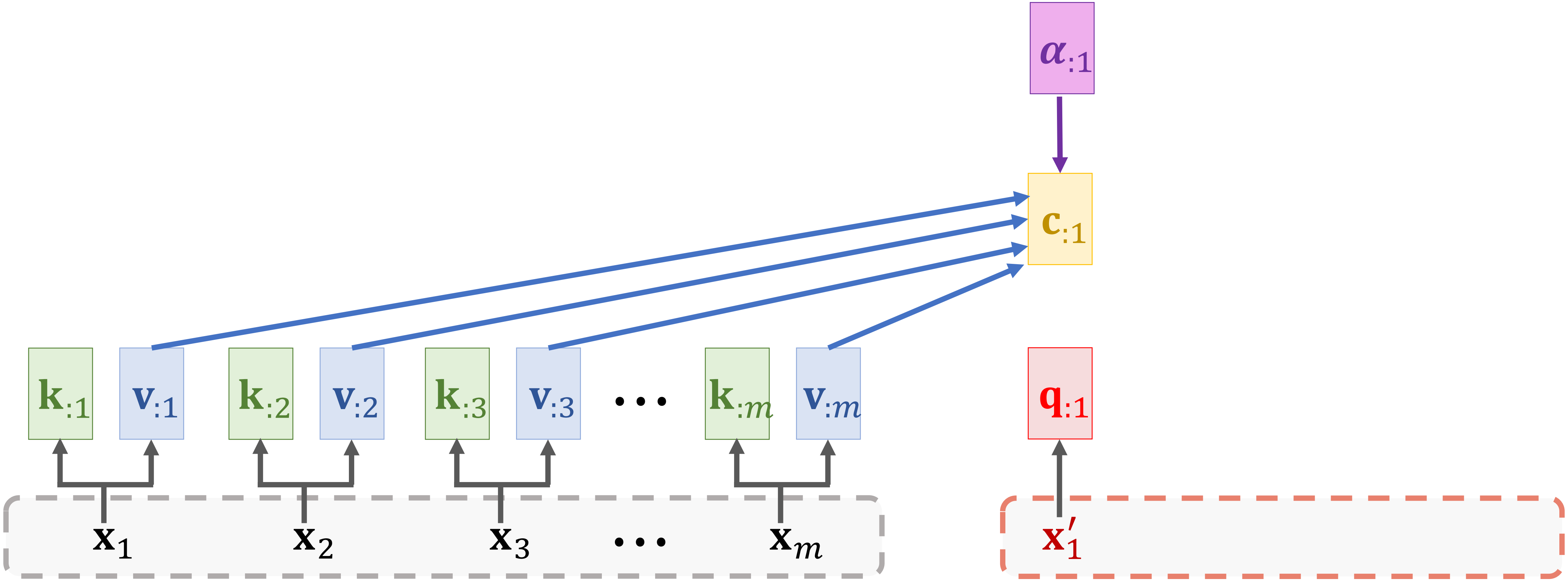
Attention Layer

- Compute weights: $\boldsymbol{\alpha}_{:1} = \mathrm{Softmax}(\mathbf{K}^T \mathbf{q}_{:1}) \in \mathbb{R}^m$.



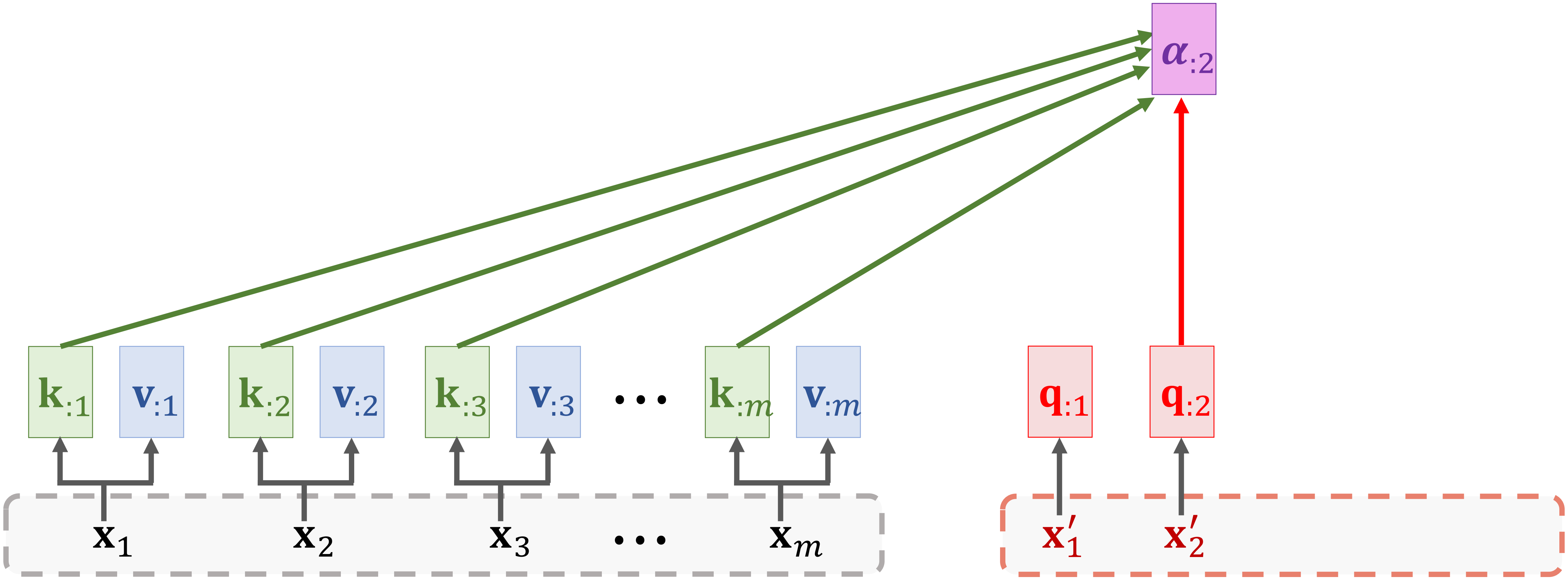
Attention Layer

- Compute context vector: $\mathbf{c}_{:1} = \alpha_{11}\mathbf{v}_{:1} + \cdots + \alpha_{m1}\mathbf{v}_{:m} = \underline{\mathbf{V}\boldsymbol{\alpha}_{:1}}$.



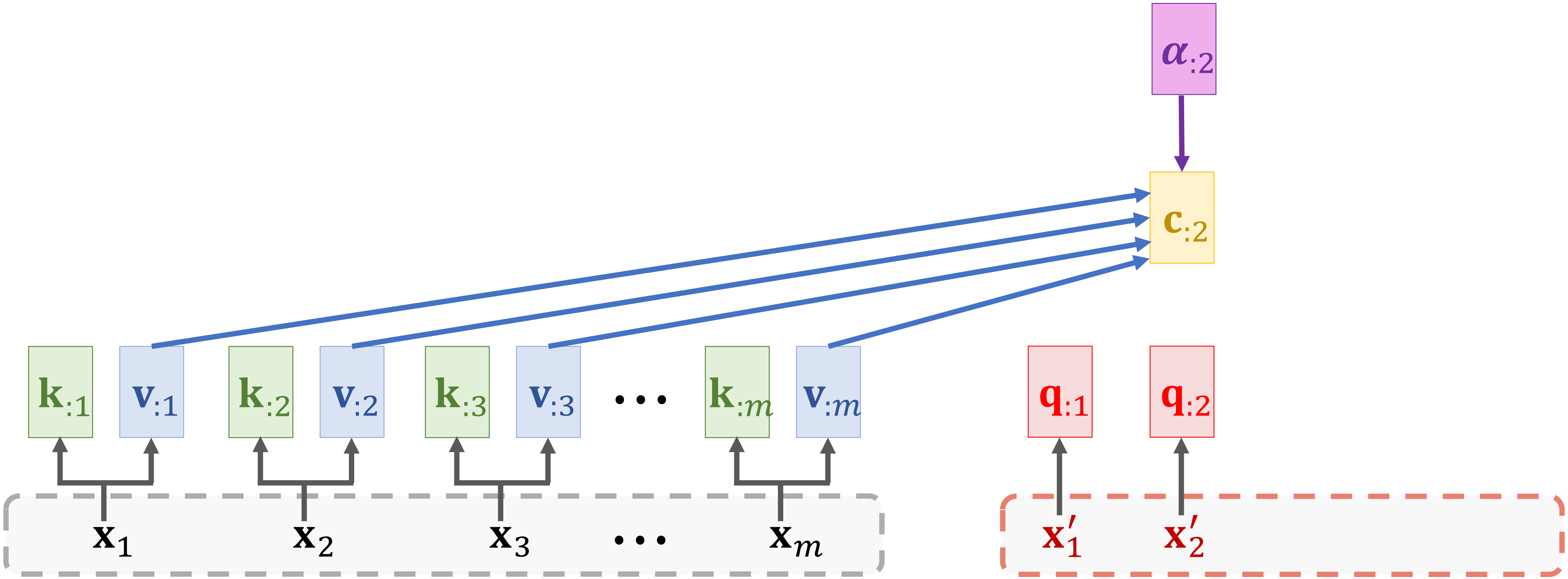
Attention Layer

- Compute weights: $\boldsymbol{\alpha}_{:2} = \mathrm{Softmax}(\mathbf{K}^T \mathbf{q}_{:2}) \in \mathbb{R}^m$.



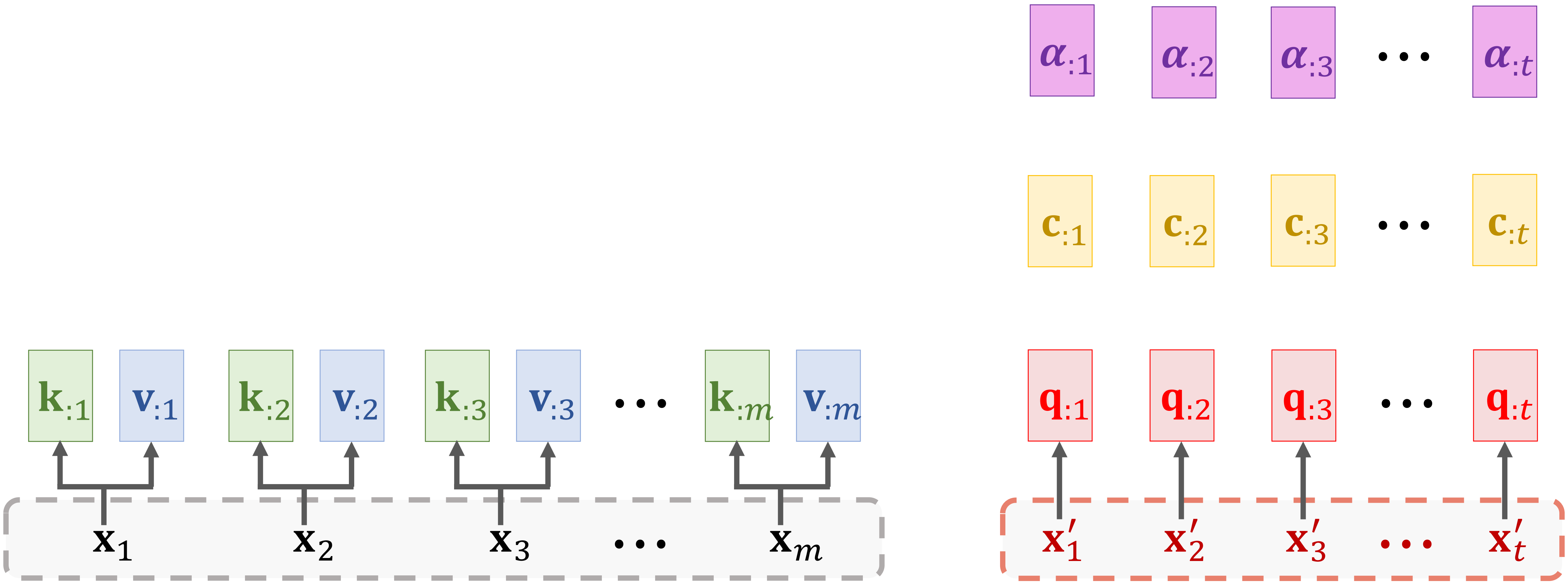
Attention Layer

- Compute context vector: $\mathbf{c}_{:2} = \alpha_{12}\mathbf{v}_{:1} + \cdots + \alpha_{m2}\mathbf{v}_{:m} = \mathbf{V}\boldsymbol{\alpha}_{:2}$.

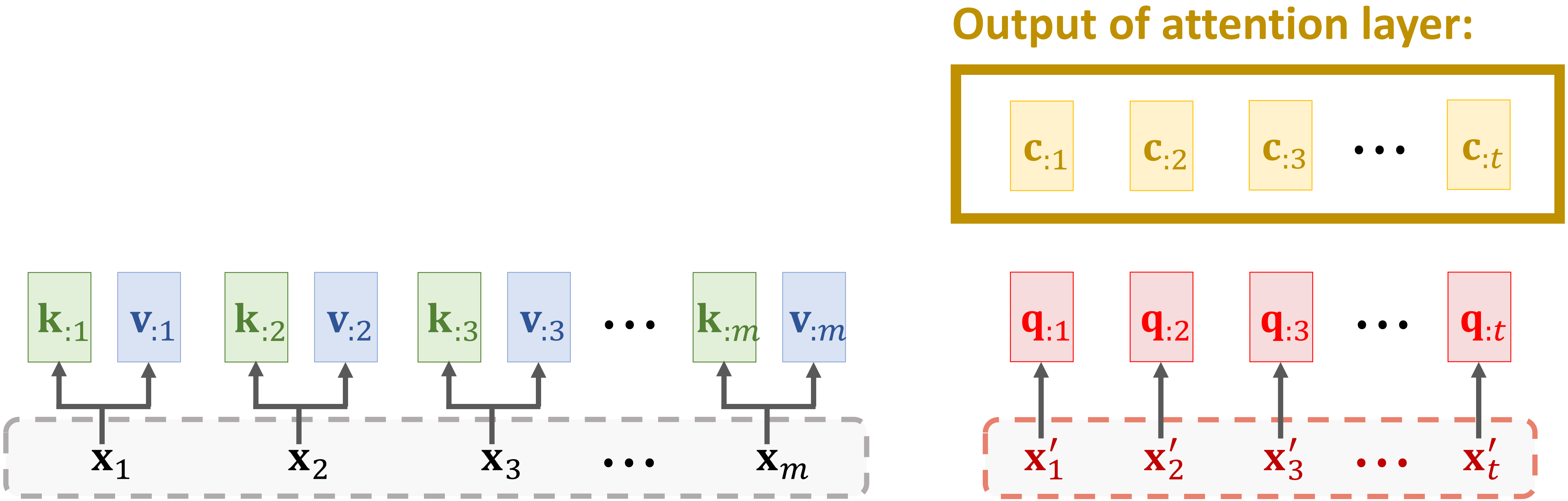


Attention Layer

- Compute context vector: $\mathbf{c}_{:j} = \alpha_{1j}\mathbf{v}_{:1} + \cdots + \alpha_{mj}\mathbf{v}_{:m} = \mathbf{V}\boldsymbol{\alpha}_{:j}$.



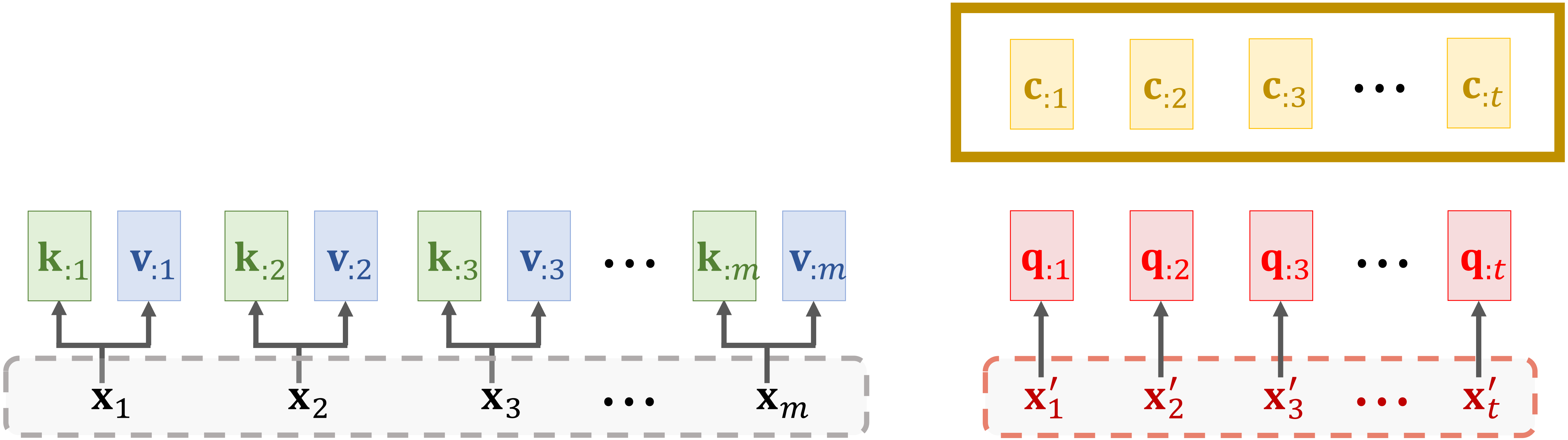
Attention Layer



Attention Layer

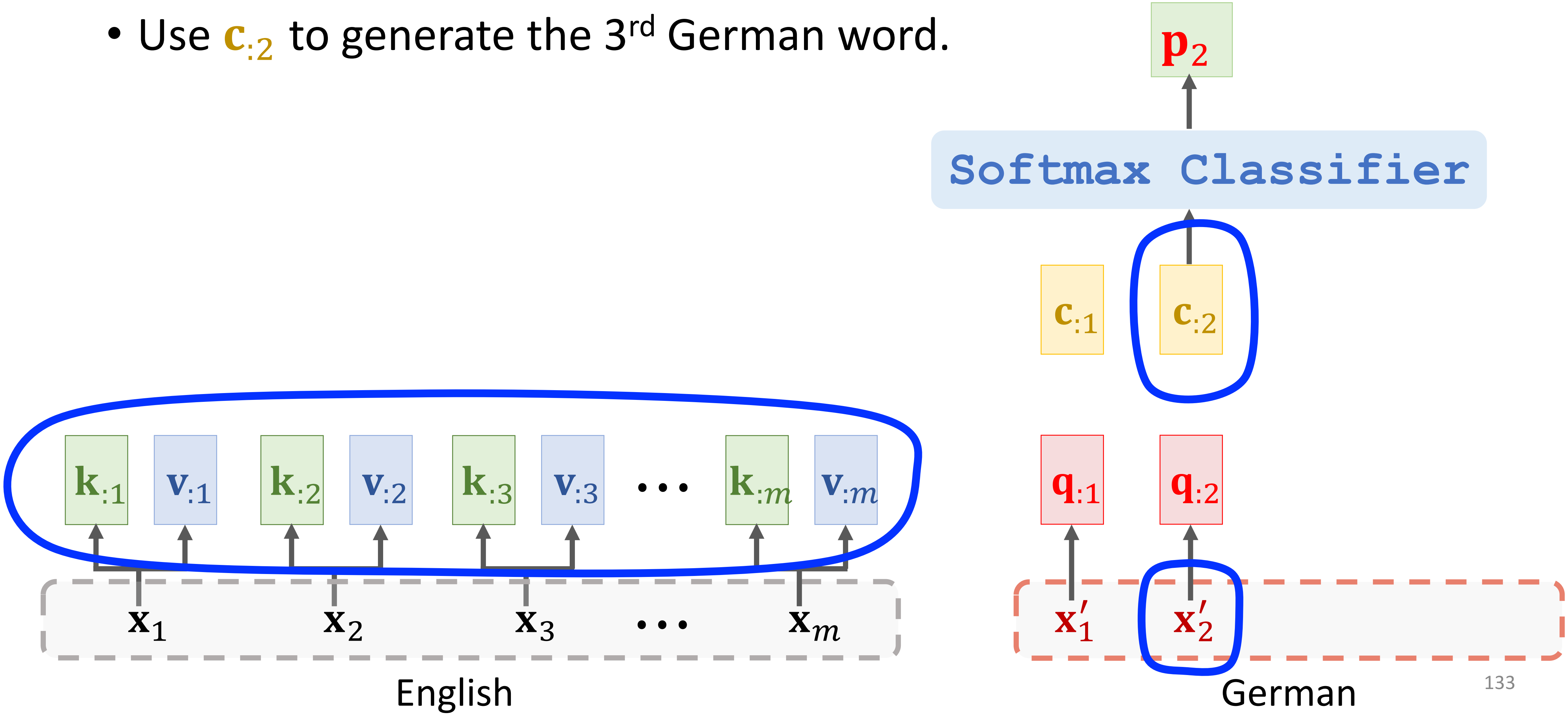
- Here, $\mathbf{c}_{:j} = \mathbf{V} \cdot \mathrm{Softmax}\left(\mathbf{K}^T \mathbf{q}_{:j}\right)$.
- Thus, $\mathbf{c}_{:j}$ is a function of $\mathbf{x}'_j$ and $[\mathbf{x}_1, \cdots, \mathbf{x}_m]$.

Output of attention layer:



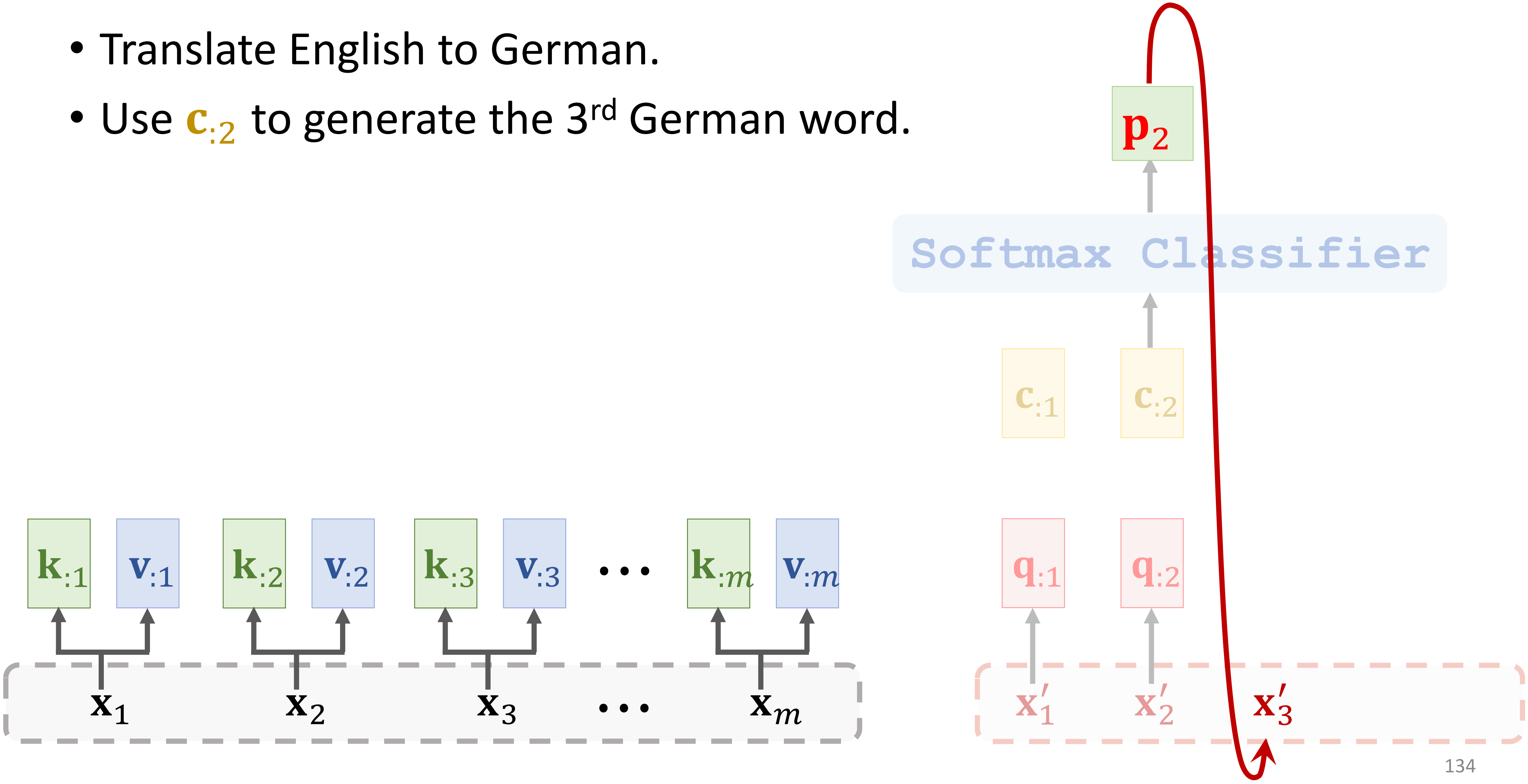
Attention Layer for Machine Translation

- Translate English to German.
- Use $\mathbf{c}_{:2}$ to generate the 3rd German word.



Attention Layer for Machine Translation

- Translate English to German.
- Use $\mathbf{c}_{:2}$ to generate the 3rd German word.

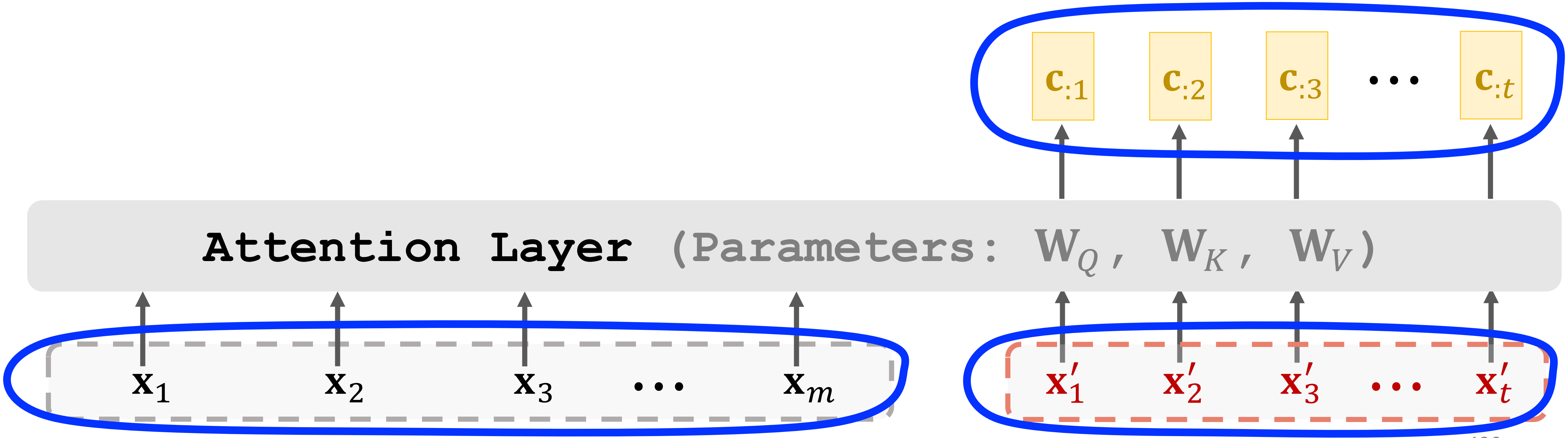


Attention Layer

- Attention layer: $[\mathbf{c}_{:1}, \mathbf{c}_{:2}, \cdots, \mathbf{c}_{:t}] = \text{Attn}(\mathbf{X}, \mathbf{X}')$.
  - Encoder's inputs: $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m]$.
  - Decoder's inputs: $\mathbf{X}' = [\mathbf{x}'_1, \mathbf{x}'_2, \cdots, \mathbf{x}'_t]$.
  - Parameters: $\mathbf{W}_Q$, $\mathbf{W}_K$, $\mathbf{W}_V$.

Attention Layer

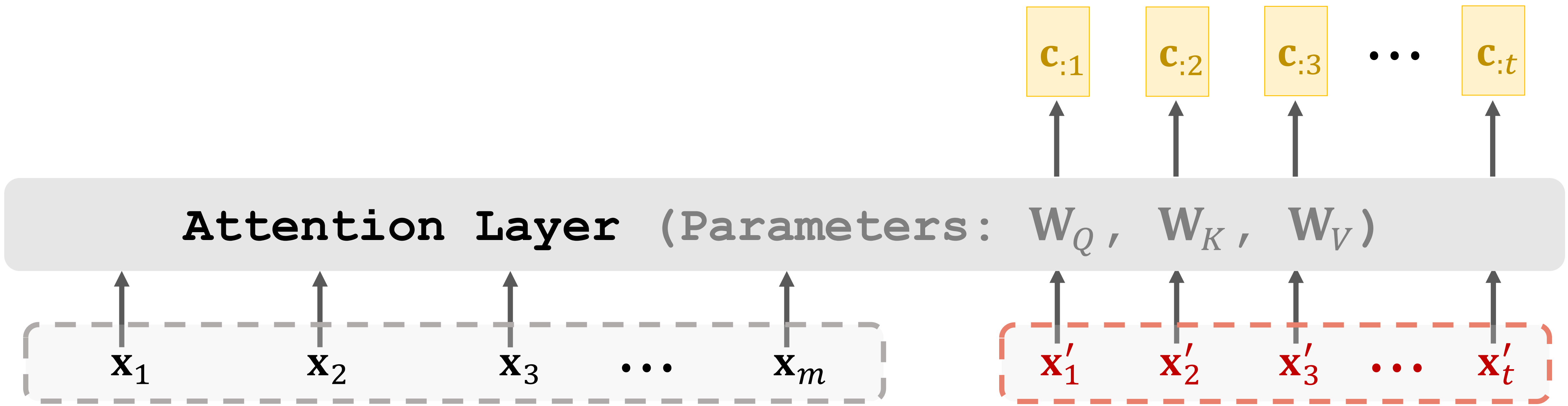
- Attention layer: $[\mathbf{c}_{:1}, \mathbf{c}_{:2}, \cdots, \mathbf{c}_{:t}] = \mathrm{Attn}(\mathbf{X}, \mathbf{X}')$.
  - Encoder's inputs: $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m]$.
  - Decoder's inputs: $\mathbf{X}' = [\mathbf{x}'_1, \mathbf{x}'_2, \cdots, \mathbf{x}'_t]$.
  - Parameters: $\mathbf{W}_Q$, $\mathbf{W}_K$, $\mathbf{W}_V$.



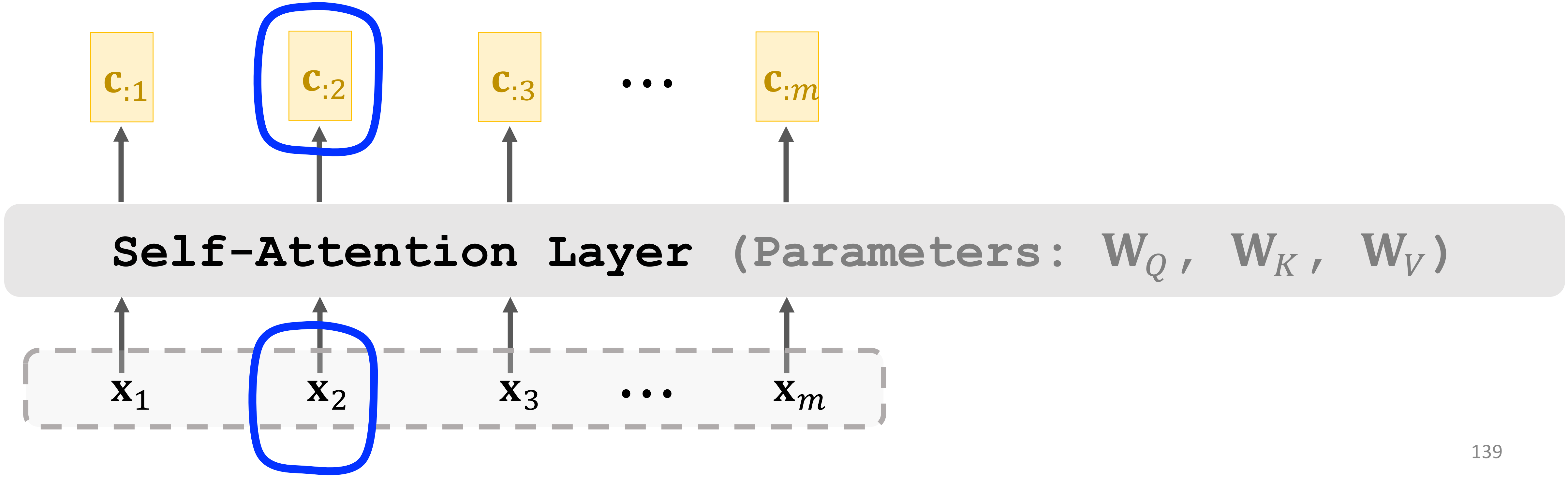
Self-Attention without RNN

Attention Layer

- Attention layer: $[\mathbf{c}_{:1}, \mathbf{c}_{:2}, \cdots, \mathbf{c}_{:t}] = \text{Attn}(\mathbf{X}, \mathbf{X}')$.

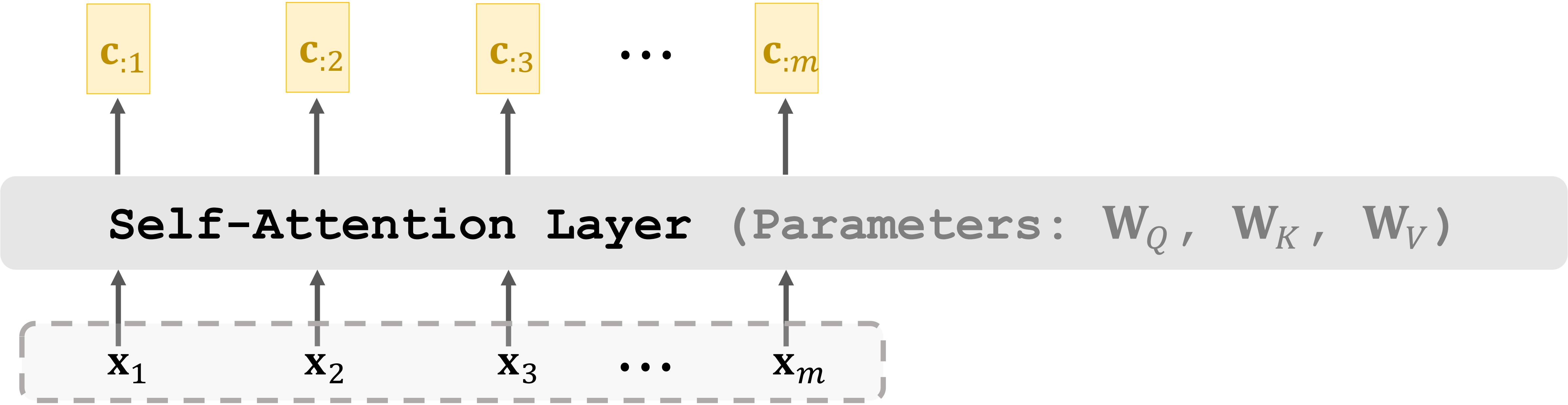


Self-Attention Layer



Self-Attention Layer

- Self-attention layer: $[\mathbf{c}_{:1}, \mathbf{c}_{:2}, \cdots, \mathbf{c}_{:m}] = \text{Attn}(\mathbf{X}, \mathbf{X})$.
  - Inputs: $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m]$.
  - Parameters: $\mathbf{W}_Q$, $\mathbf{W}_K$, $\mathbf{W}_V$.



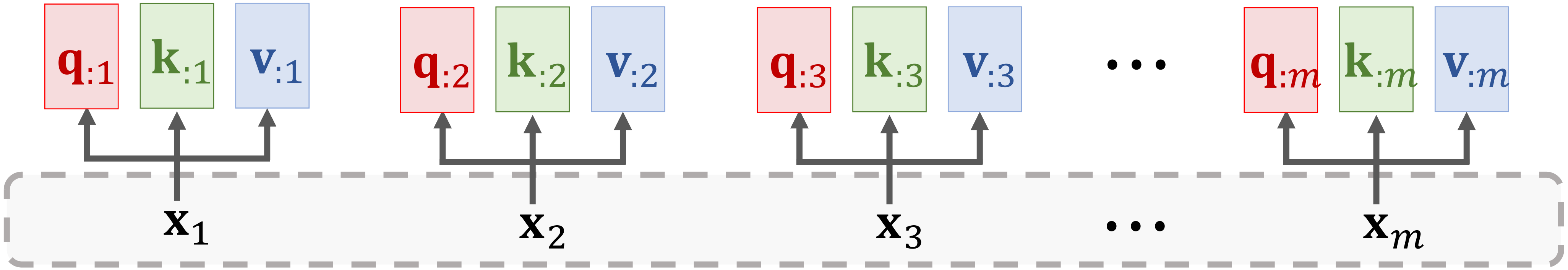
Self-Attention Layer

Inputs:

$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\cdots$ $\mathbf{x}_m$

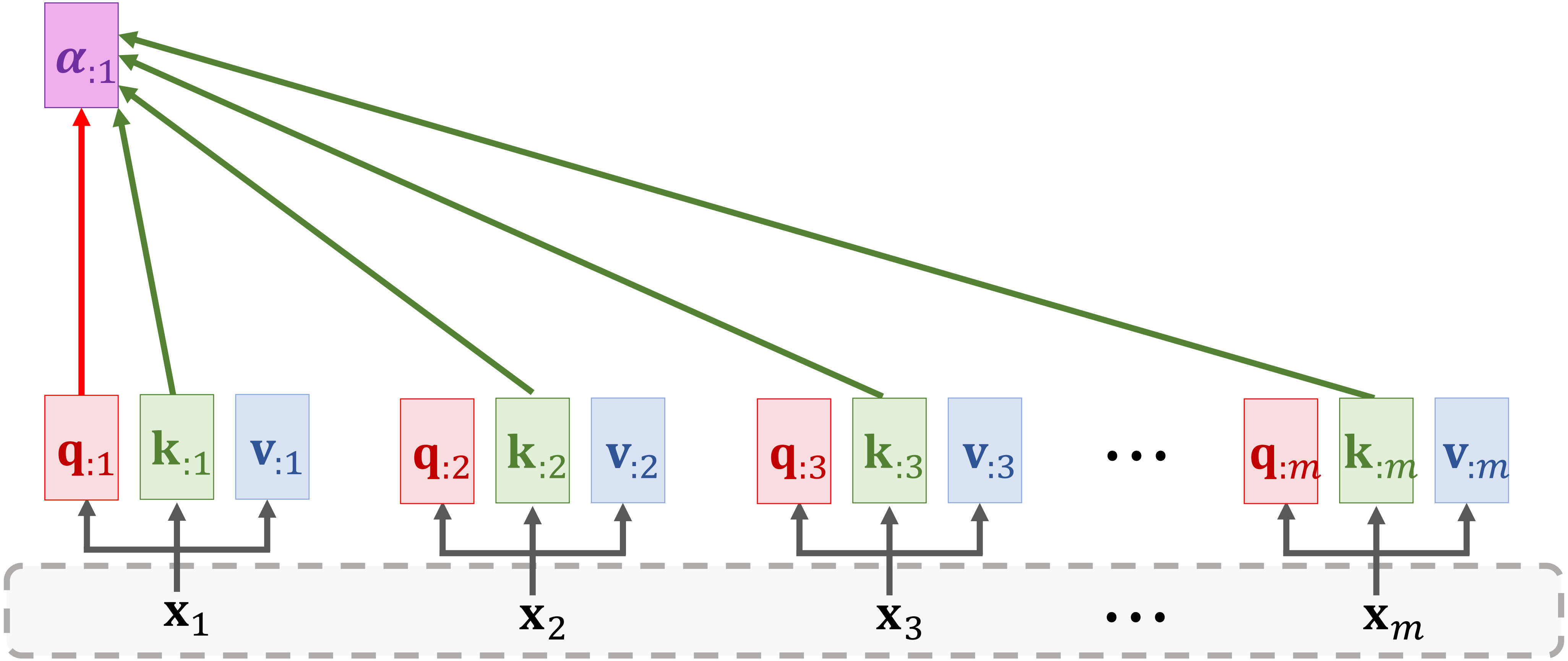
Self-Attention Layer

Query: $\mathbf{q}_{:i} = \mathbf{W}_Q \mathbf{x}_i$, Key: $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{x}_i$, Value: $\mathbf{v}_{:i} = \mathbf{W}_V \mathbf{x}_i$.



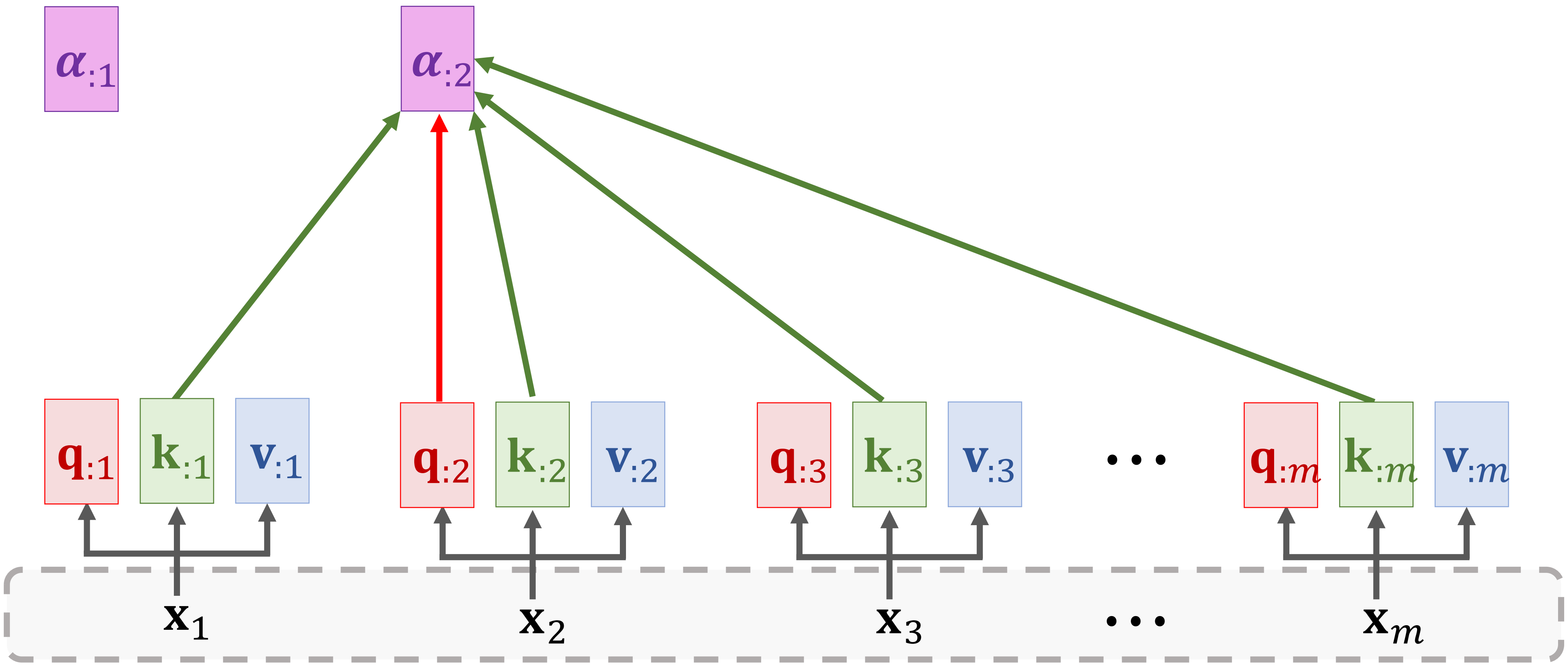
Self-Attention Layer

Weights: $\boldsymbol{\alpha}_{:j} = \mathrm{Softmax}\left(\mathbf{K}^T \mathbf{q}_{:j}\right) \in \mathbb{R}^m.$



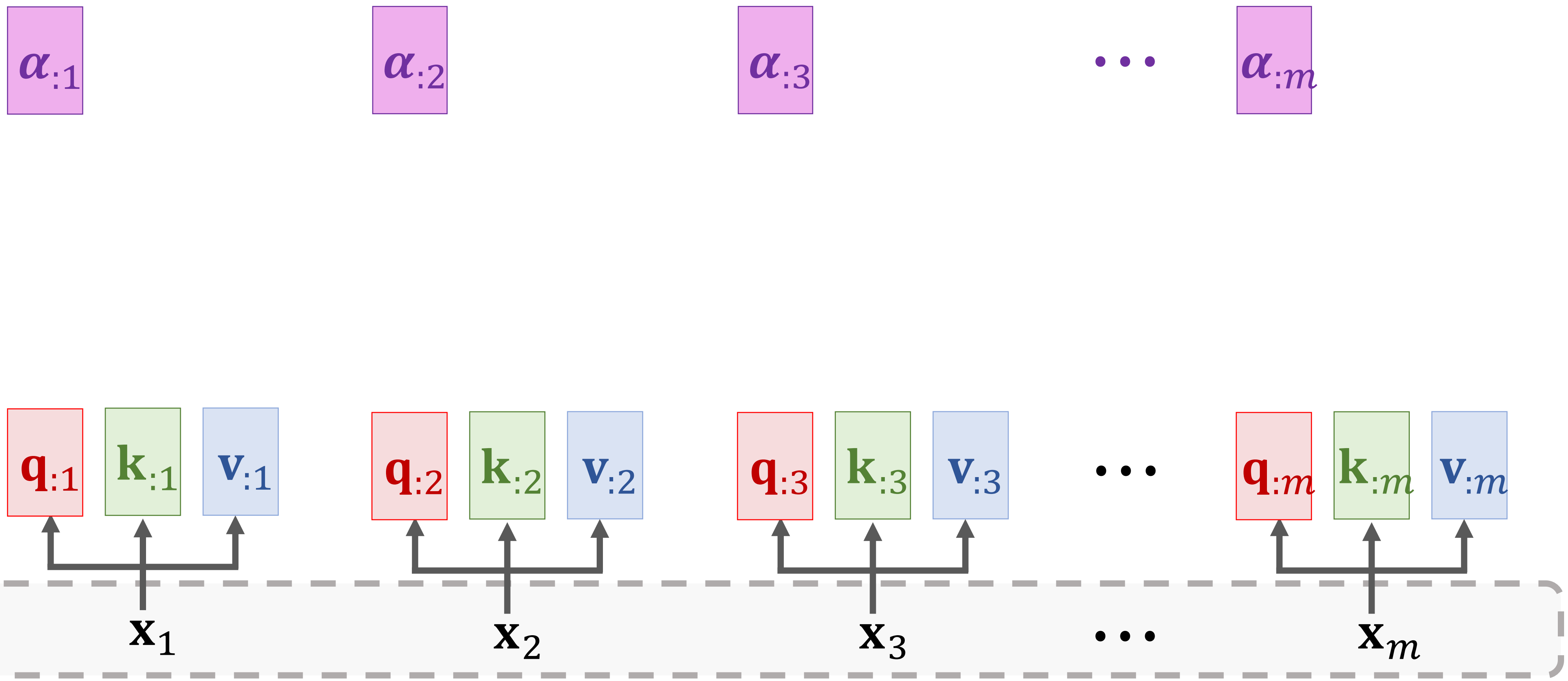
Self-Attention Layer

Weights: $\boldsymbol{\alpha}_{:j} = \mathrm{Softmax}\left(\mathbf{K}^T \mathbf{q}_{:j}\right) \in \mathbb{R}^m.$



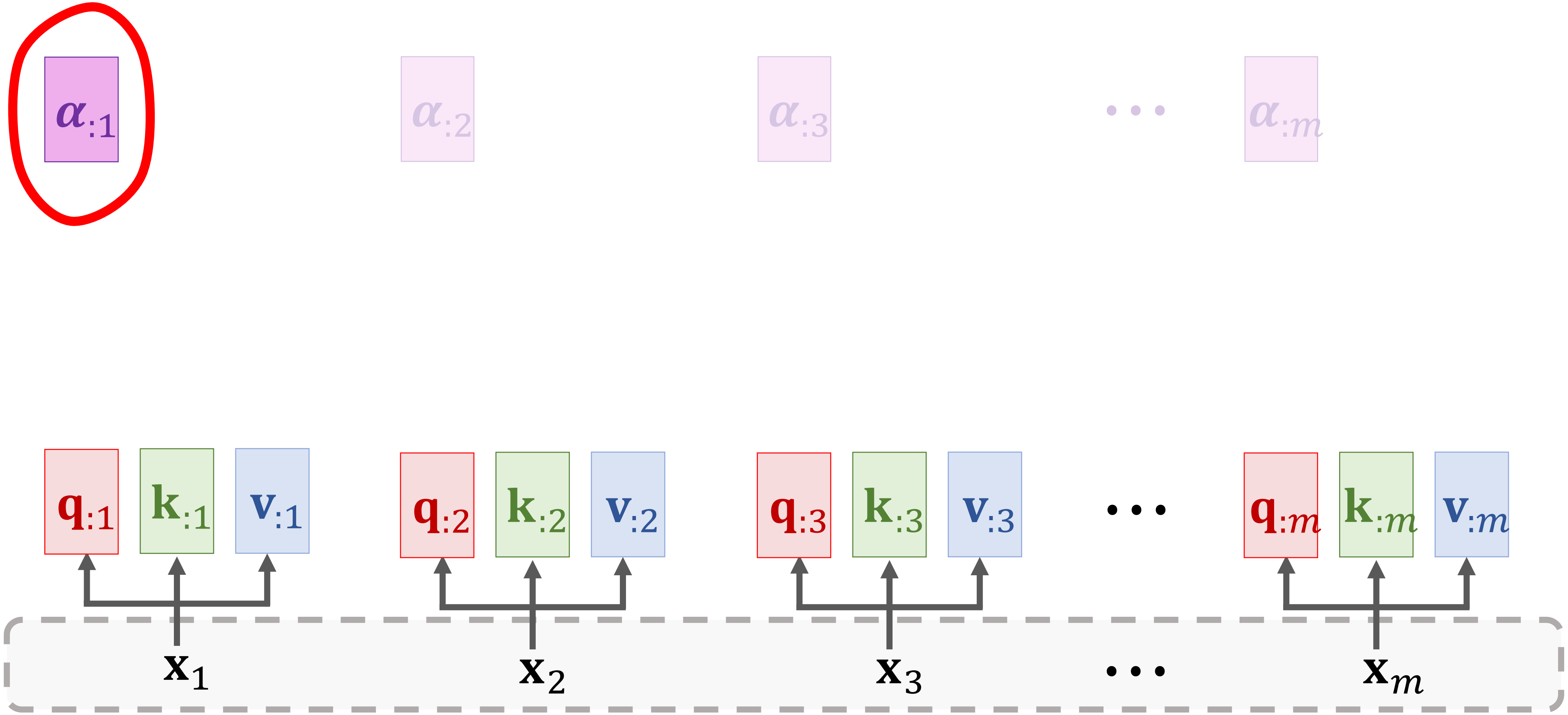
Self-Attention Layer

Weights: $\boldsymbol{\alpha}_{:j} = \mathrm{Softmax}\left(\mathbf{K}^T \mathbf{q}_{:j}\right) \in \mathbb{R}^m.$



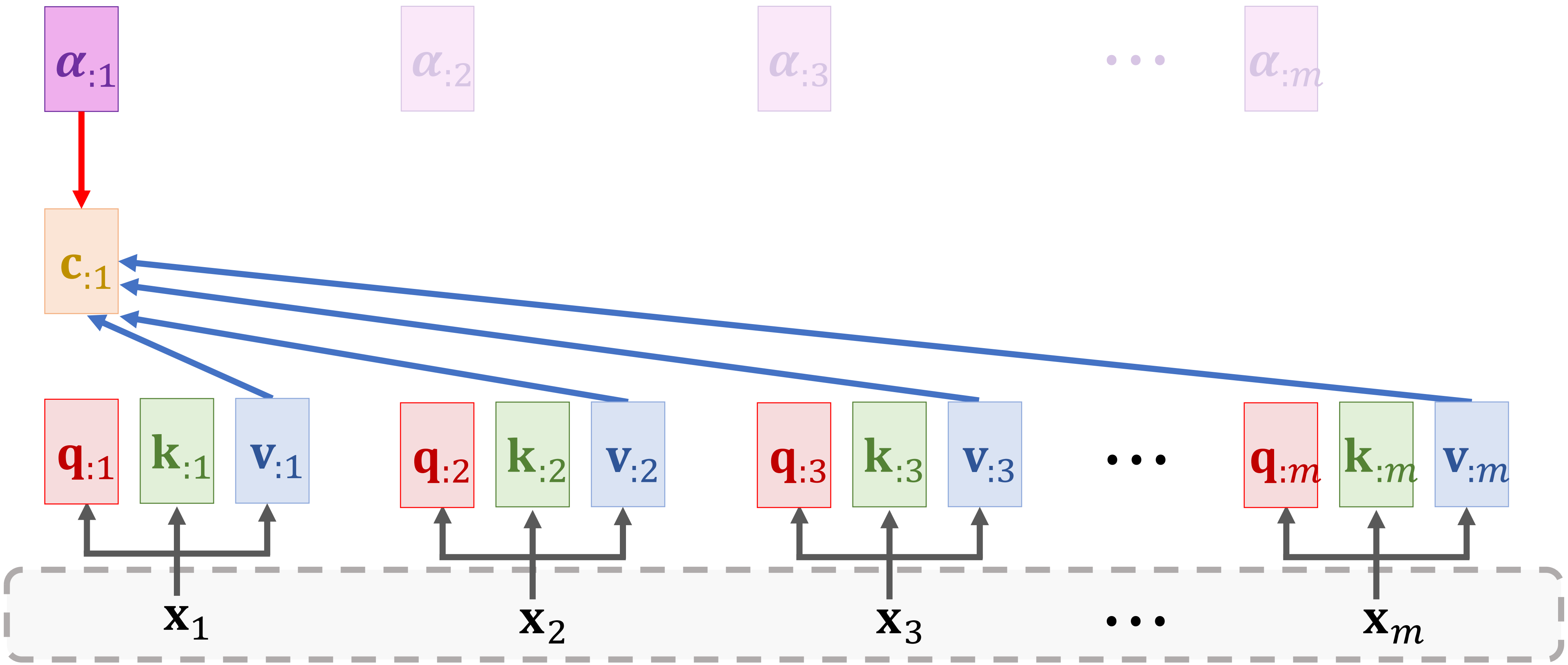
Self-Attention Layer

Context vector: $\mathbf{c}_{:1} = \alpha_{11}\mathbf{v}_{:1} + \cdots + \alpha_{m1}\mathbf{v}_{:m} = \mathbf{V}\boldsymbol{\alpha}_{:1}.$



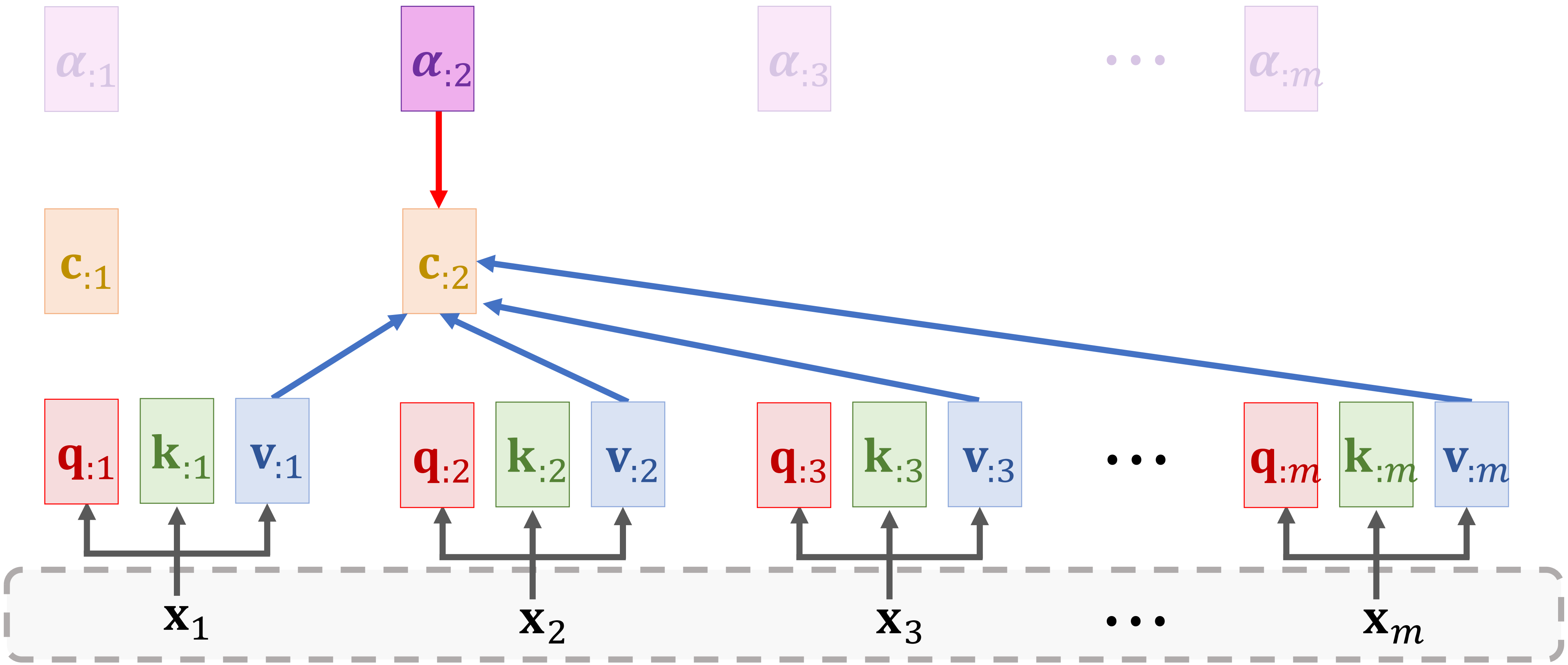
Self-Attention Layer

Context vector: $\mathbf{c}_{:1} = \alpha_{11}\mathbf{v}_{:1} + \cdots + \alpha_{m1}\mathbf{v}_{:m} = \mathbf{V}\boldsymbol{\alpha}_{:1}.$



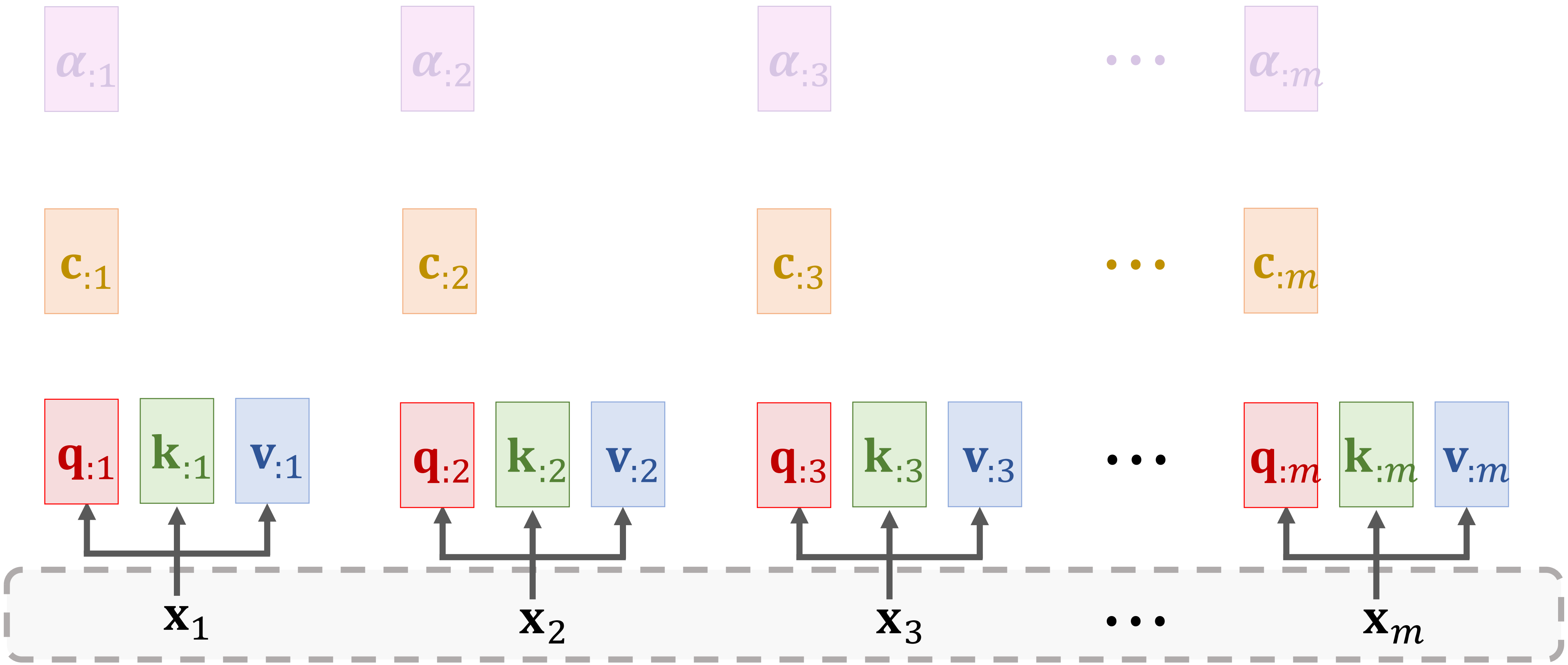
Self-Attention Layer

Context vector: $\mathbf{c}_{:2} = \alpha_{12}\mathbf{v}_{:1} + \cdots + \alpha_{m2}\mathbf{v}_{:m} = \mathbf{V}\boldsymbol{\alpha}_{:2}$.



Self-Attention Layer

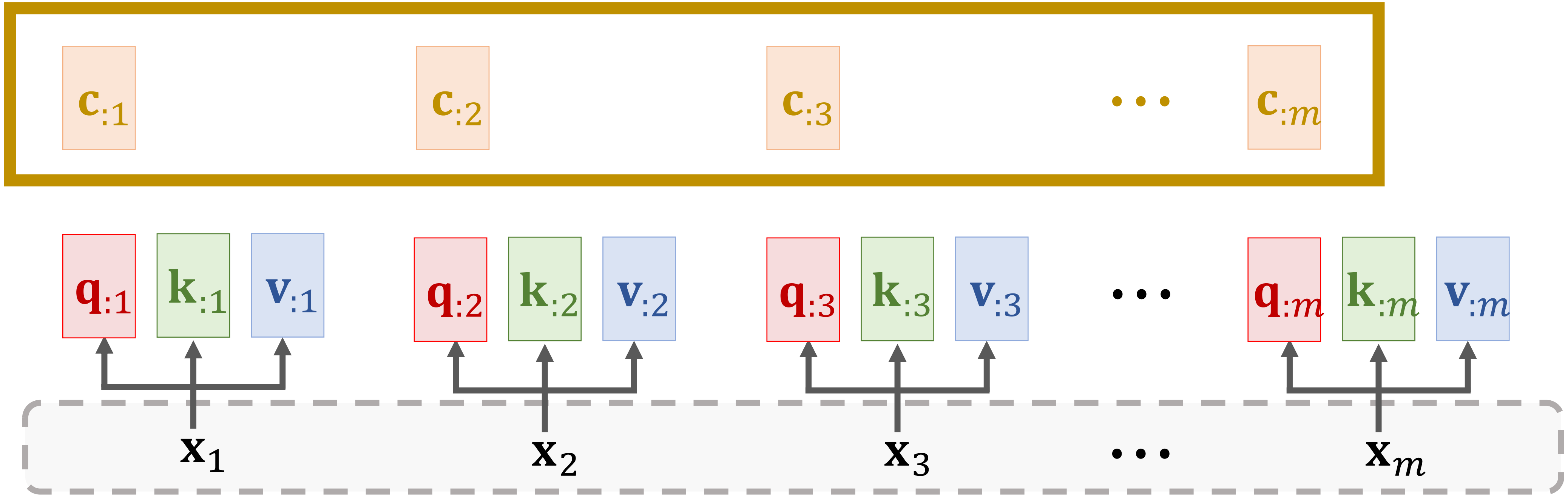
Context vector: $\mathbf{c}_{:j} = \alpha_{1j}\mathbf{v}_{:1} + \cdots + \alpha_{mj}\mathbf{v}_{:m} = \mathbf{V}\boldsymbol{\alpha}_{:j}.$



Self-Attention Layer

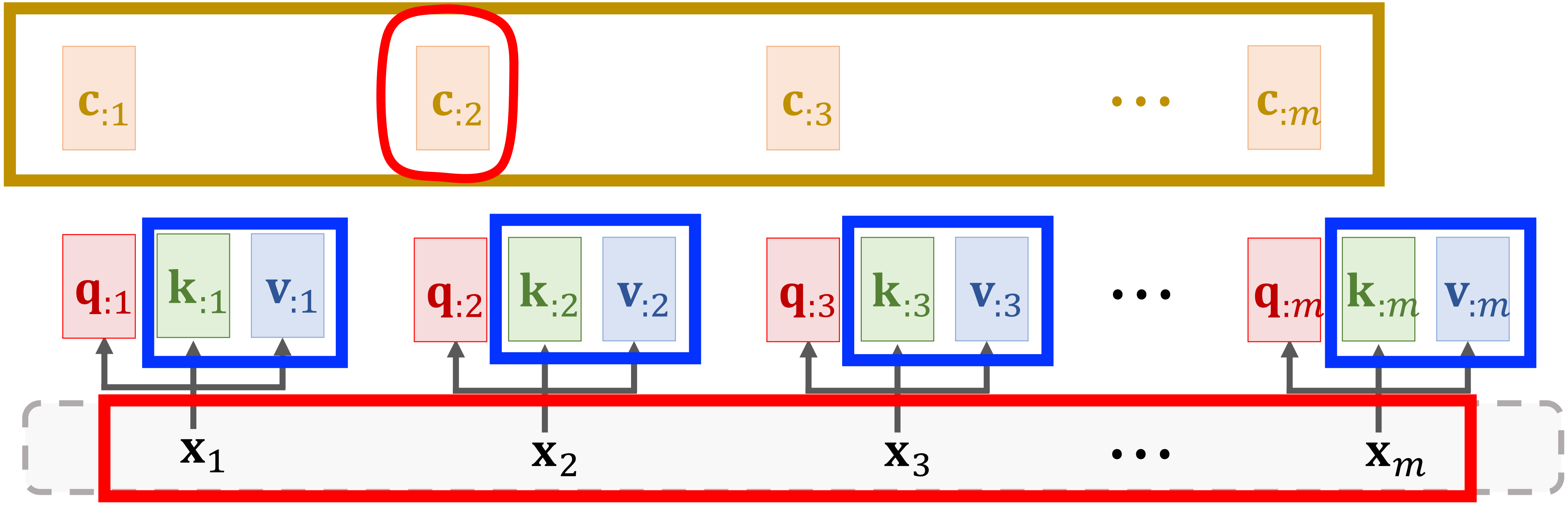
Context vector: $\mathbf{c}_{:j} = \mathbf{V} \cdot \underbrace{\text{Softmax}\left(\mathbf{K}^T \mathbf{q}_{:j}\right)}_{\boldsymbol{\alpha}_{:j}}$

Output of self-attention layer:



Self-Attention Layer

Output of self-attention layer:

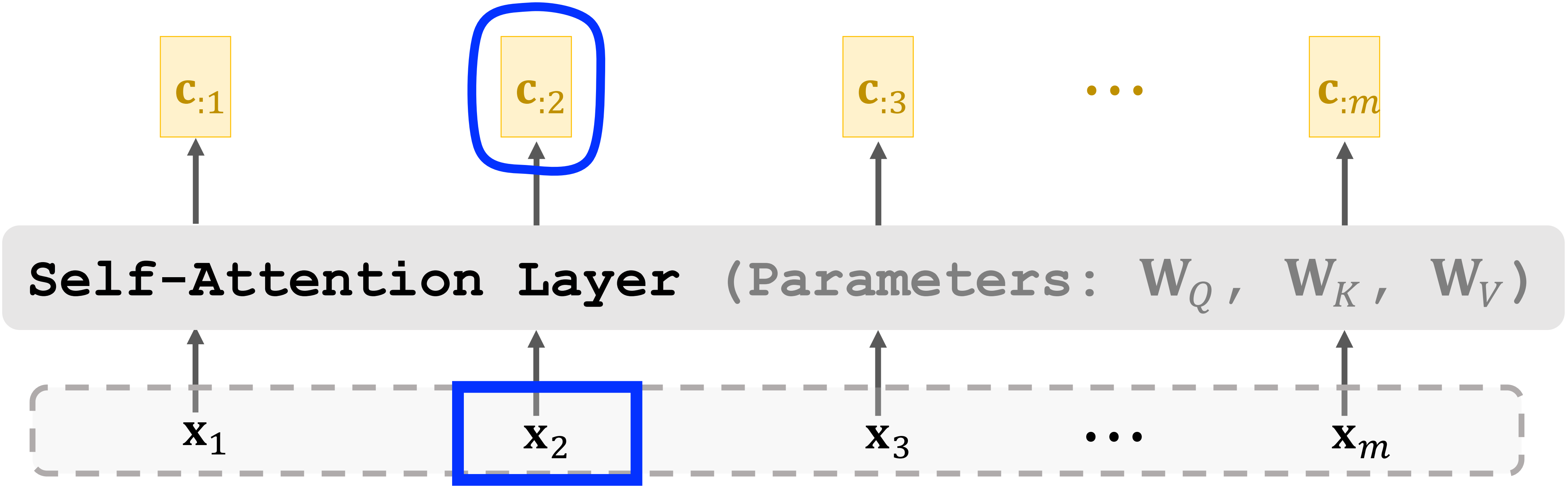


Self-Attention Layer

- Self-attention layer: $[\mathbf{c}_{:1}, \mathbf{c}_{:2}, \cdots, \mathbf{c}_{:m}] = \mathrm{Attn}(\mathbf{X}, \mathbf{X})$.
  - Inputs: $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m]$.
  - Parameters: $\mathbf{W}_Q$, $\mathbf{W}_K$, $\mathbf{W}_V$.

Self-Attention Layer

- Self-attention layer: $[\mathbf{c}_{:1}, \mathbf{c}_{:2}, \cdots, \mathbf{c}_{:m}] = \mathrm{Attn}(\mathbf{X}, \mathbf{X})$.
  - Inputs: $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m]$.
  - Parameters: $\mathbf{W}_Q$, $\mathbf{W}_K$, $\mathbf{W}_V$.



Summary

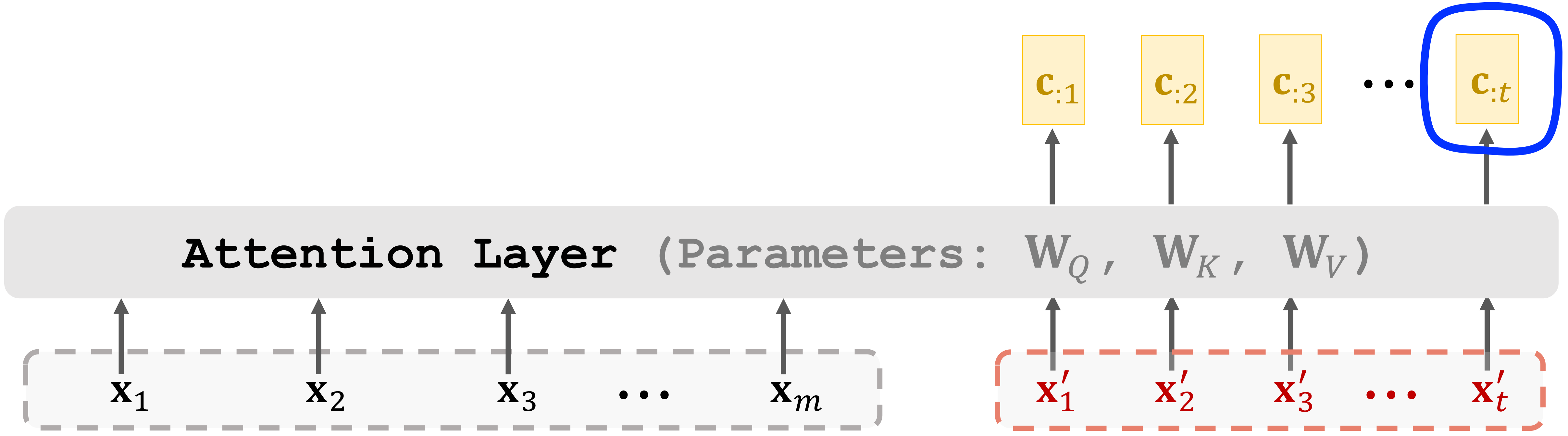
Summary

- Attention was originally developed for Seq2Seq RNN models [1].
- Self-attention: attention for all the RNN models (not necessarily Seq2Seq models [2].
- Attention can be used without RNN [3].
- We learned how to build attention layer and self-attention layer.

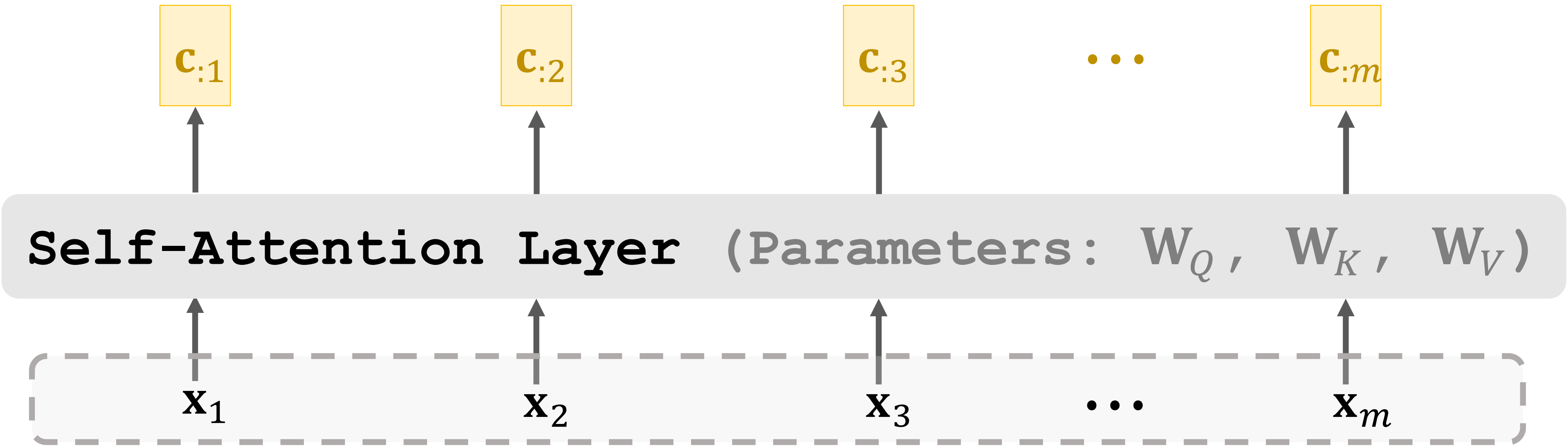
Reference:

1. Bahdanau, Cho, & Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.
2. Cheng, Dong, & Lapata. Long short-term memory-networks for machine reading. In *EMNLP*, 2016.
3. Vaswani et al. Attention is all you need. In *NIPS*, 2017.

Attention Layer



Self-Attention Layer



Thank You!