ECE 884 Deep Learning

Lecture 20: Transformer Model

04/01/2021

Review of last lecture

- RNN Applications
  - Text generation (many to one)
  - Image Captioning (one to many)
  - Neural Machine Translation (many to many - Seq2Seq)
- Attention with RNN
- Self-attention with RNN

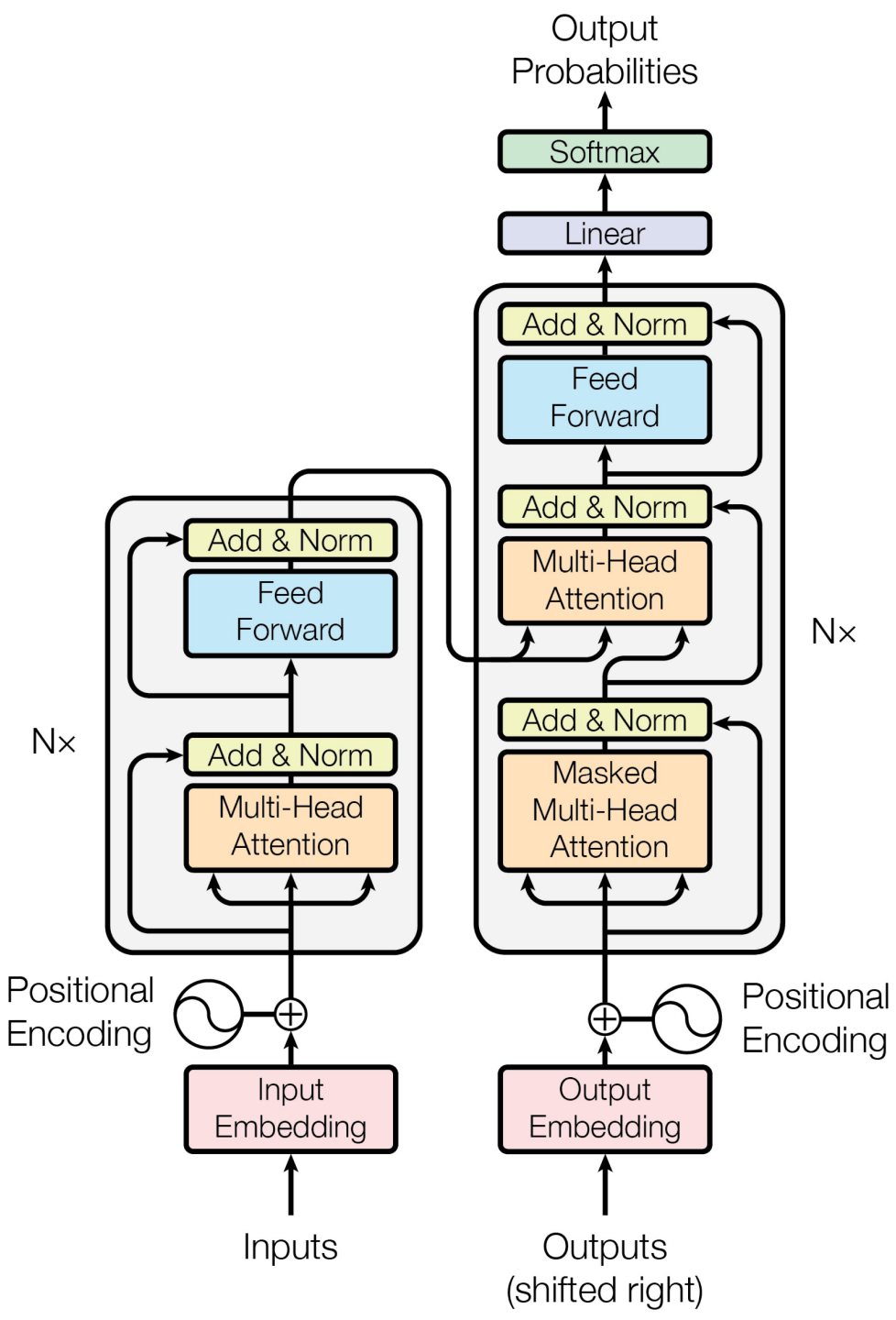
Today’s lecture

- Attention without RNN
- Self-Attention without RNN
- Transformer Model

Transformer Model

- Transformer is a Seq2Seq model.
- Transformer is not RNN.
- Purely based on attention and dense layers.

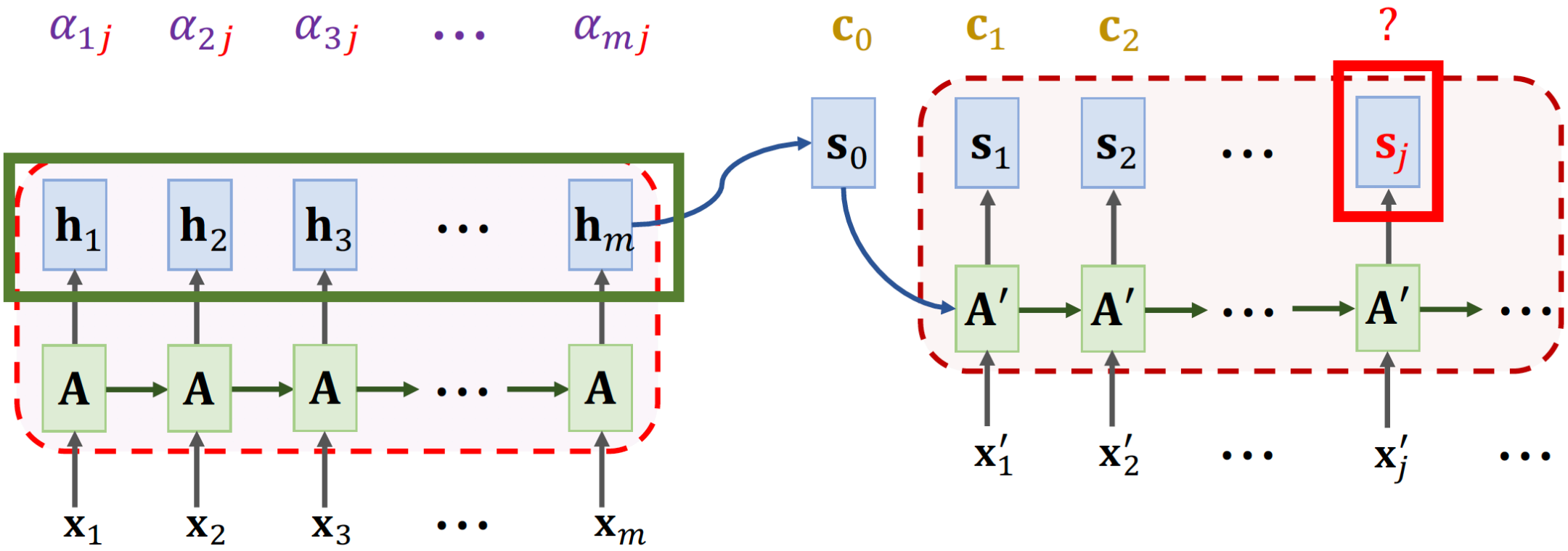
- Higher accuracy than RNNs on large datasets.



Revisiting Attention for RNN

Attention for Seq2Seq Model

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

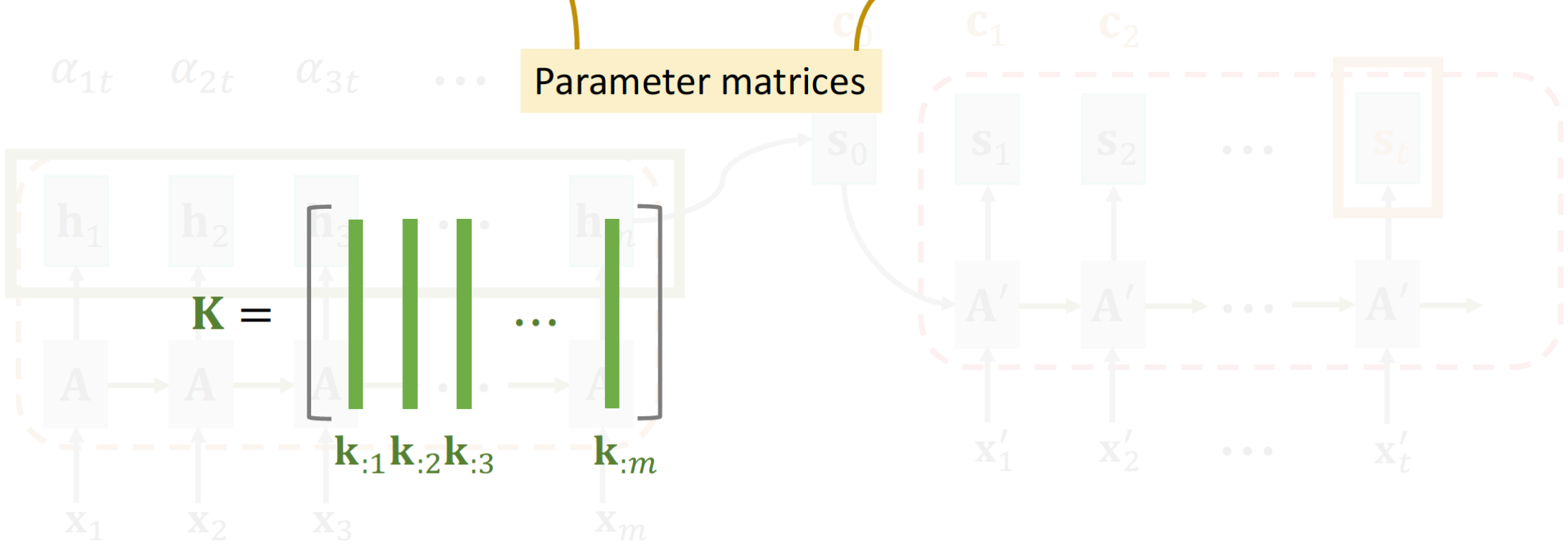


Attention for Seq2Seq Model

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

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

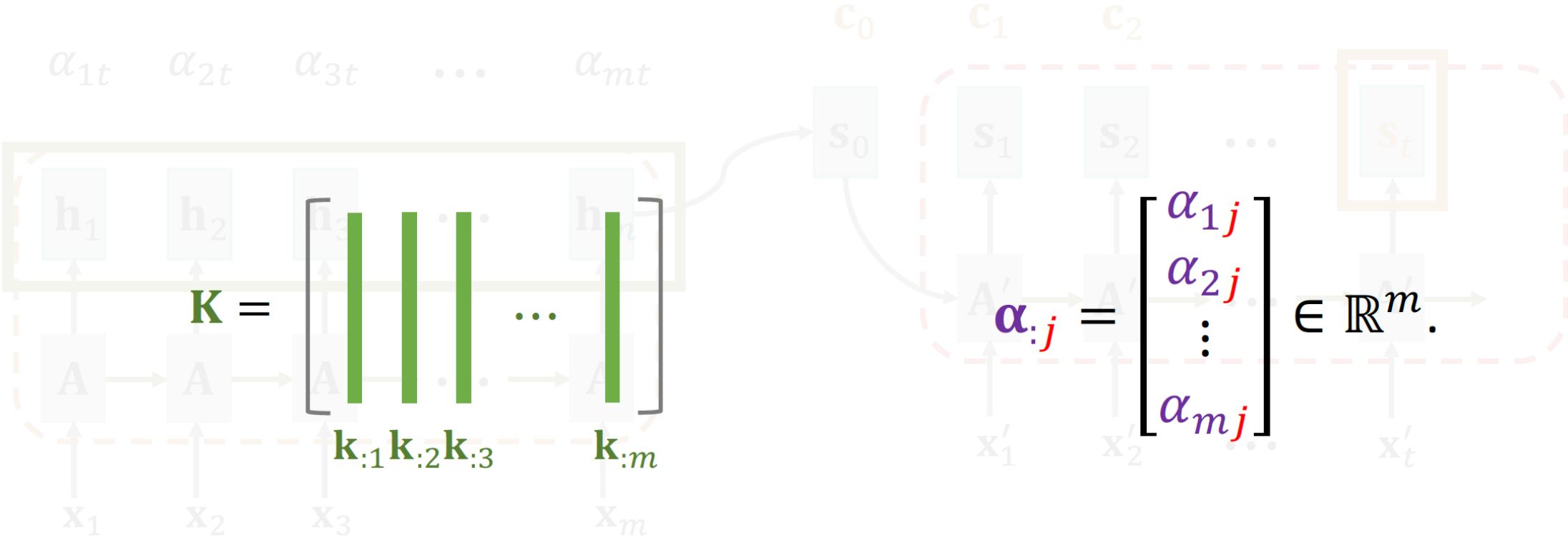
Parameter matrices



Attention for Seq2Seq Model

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

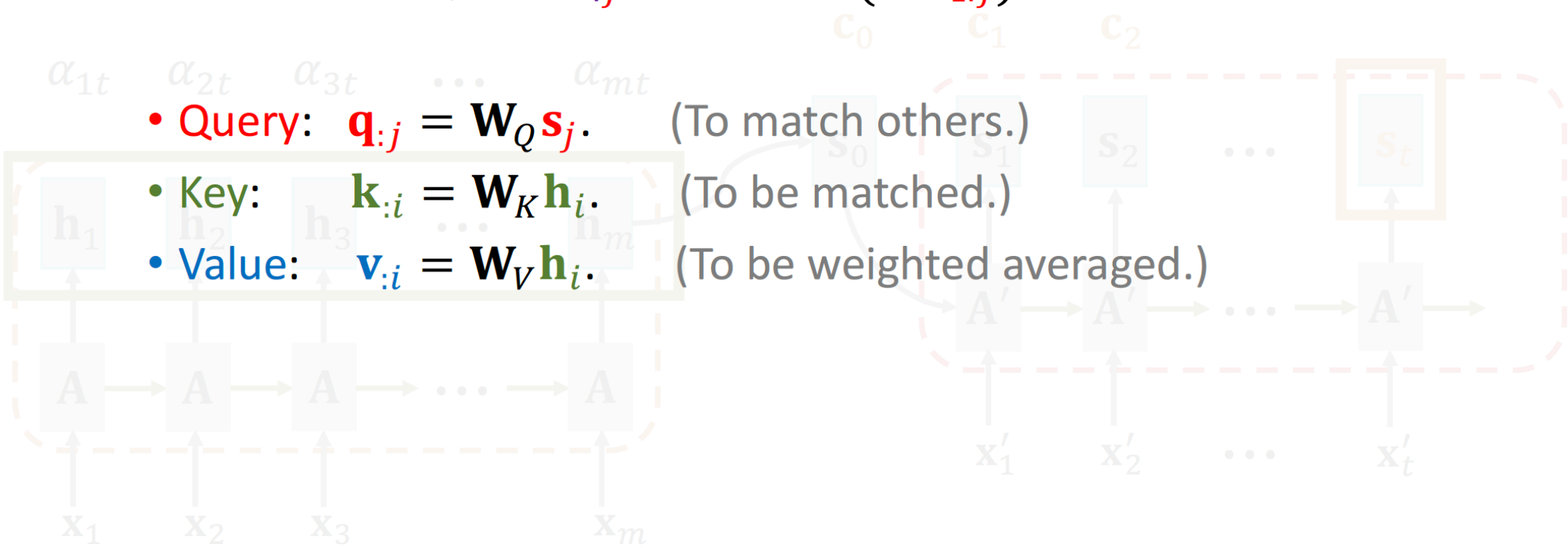
- Compute $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{h}_i$ and $\mathbf{q}_{:j} = \mathbf{W}_Q \mathbf{s}_j$.
- Compute weights: $\boldsymbol{\alpha}_{:j} = \text{Softmax}(\mathbf{K}^T \mathbf{q}_{:j}) \in \mathbb{R}^m$.



Attention for Seq2Seq Model

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

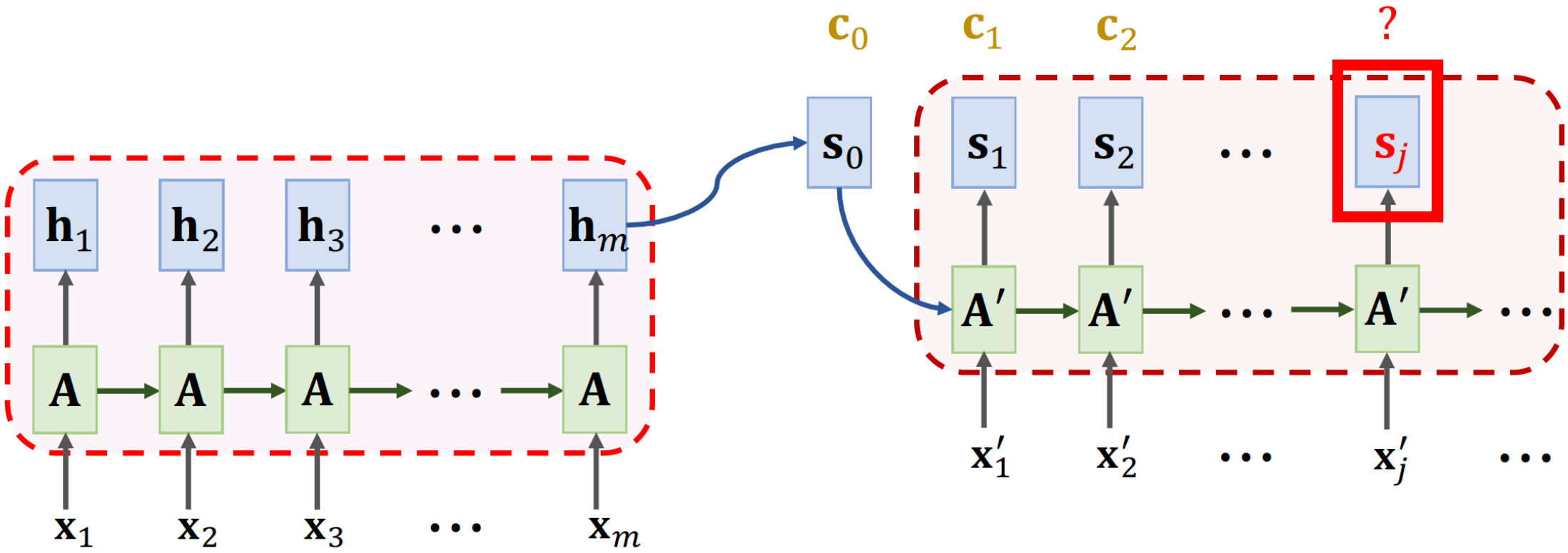
- Compute $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{h}_i$ and $\mathbf{q}_{:j} = \mathbf{W}_Q \mathbf{s}_j$.
- Compute weights: $\boldsymbol{\alpha}_{:j} = \text{Softmax}(\mathbf{K}^T \mathbf{q}_{:j}) \in \mathbb{R}^m$.



- 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.)

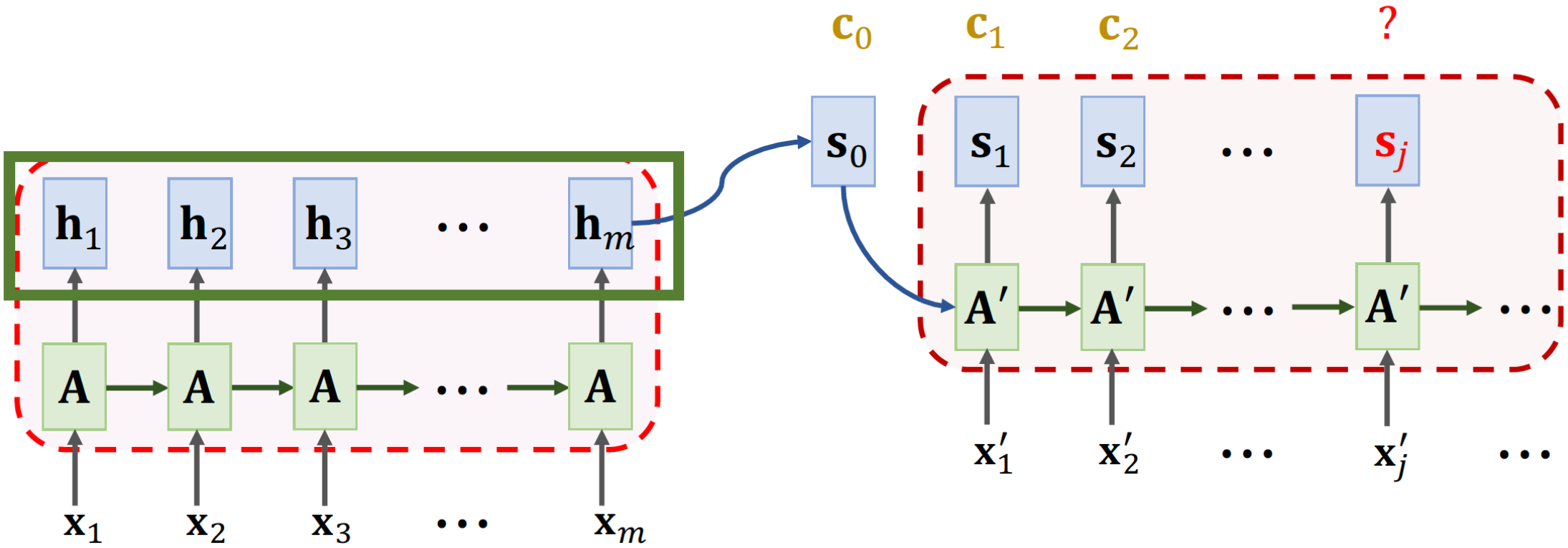
Attention for Seq2Seq Model

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

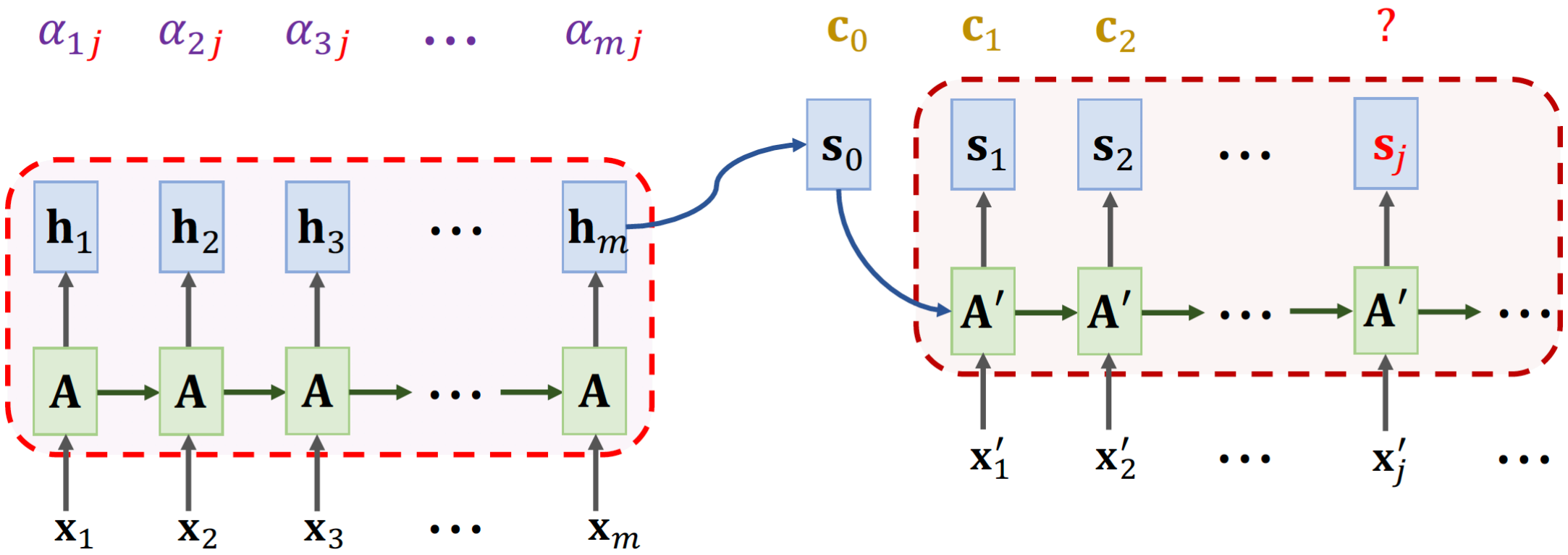
Query: $\mathbf{q}_{:j} = \mathbf{W}_Q \mathbf{s}_j$, Key: $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{h}_i$, Value: $\mathbf{v}_{:i} = \mathbf{W}_V \mathbf{h}_i$.



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: $\mathbf{v}_{:i} = \mathbf{W}_V \mathbf{h}_i$.

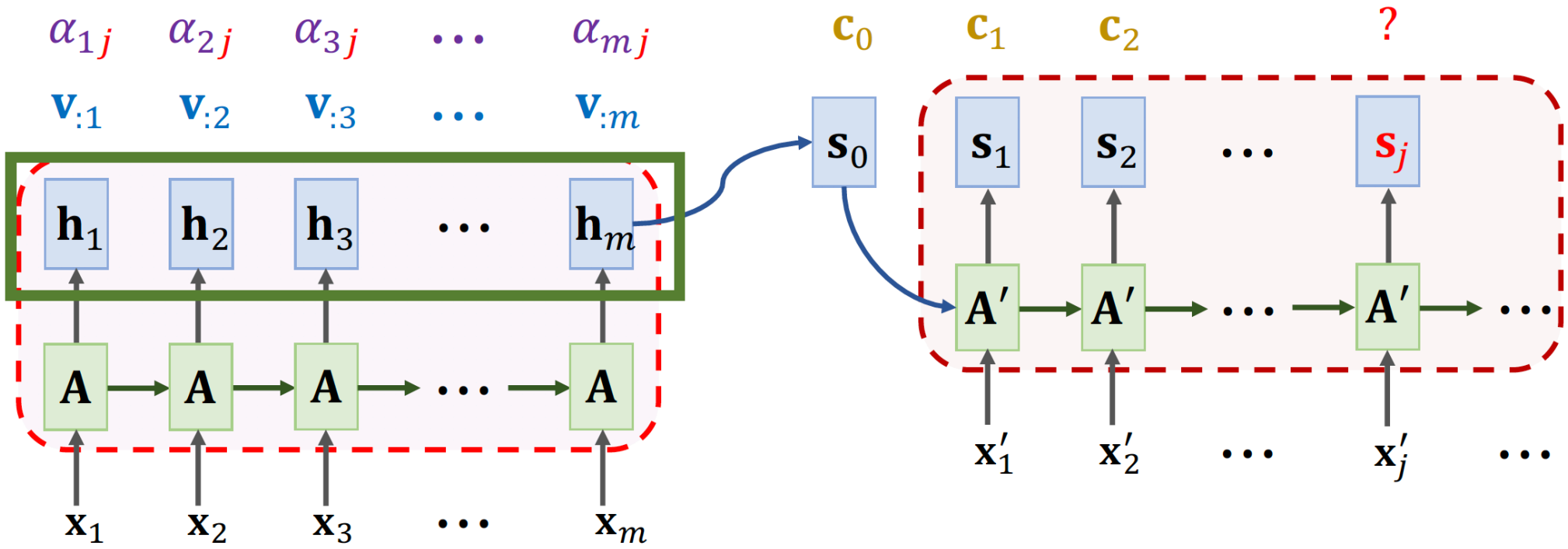
Weights: $\boldsymbol{\alpha}_{:j} = \text{Softmax}(\mathbf{K}^T \mathbf{q}_{:j}) \in \mathbb{R}^m$.



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: $\mathbf{v}_{:i} = \mathbf{W}_V \mathbf{h}_i$.

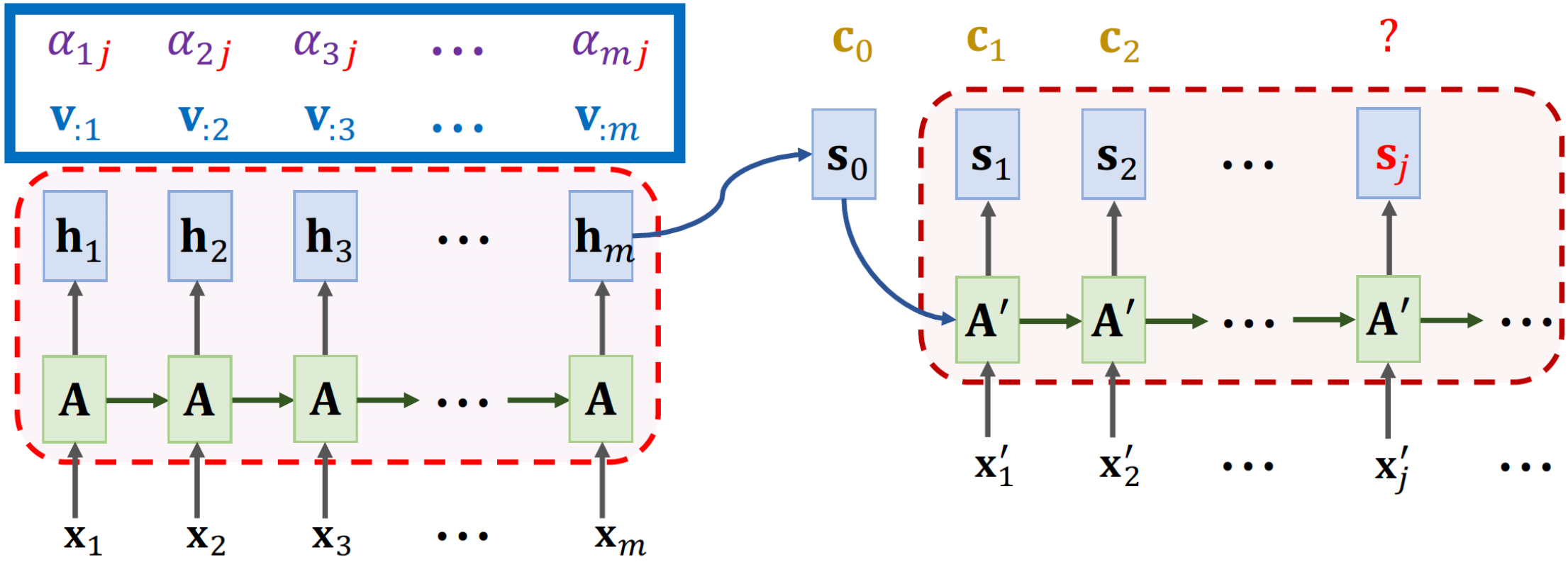
Weights: $\boldsymbol{\alpha}_{:j} = \mathrm{Softmax}\left(\mathbf{K}^T \mathbf{q}_{:j}\right) \in \mathbb{R}^m$.



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: $\mathbf{v}_{:i} = \mathbf{W}_V \mathbf{h}_i$.

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

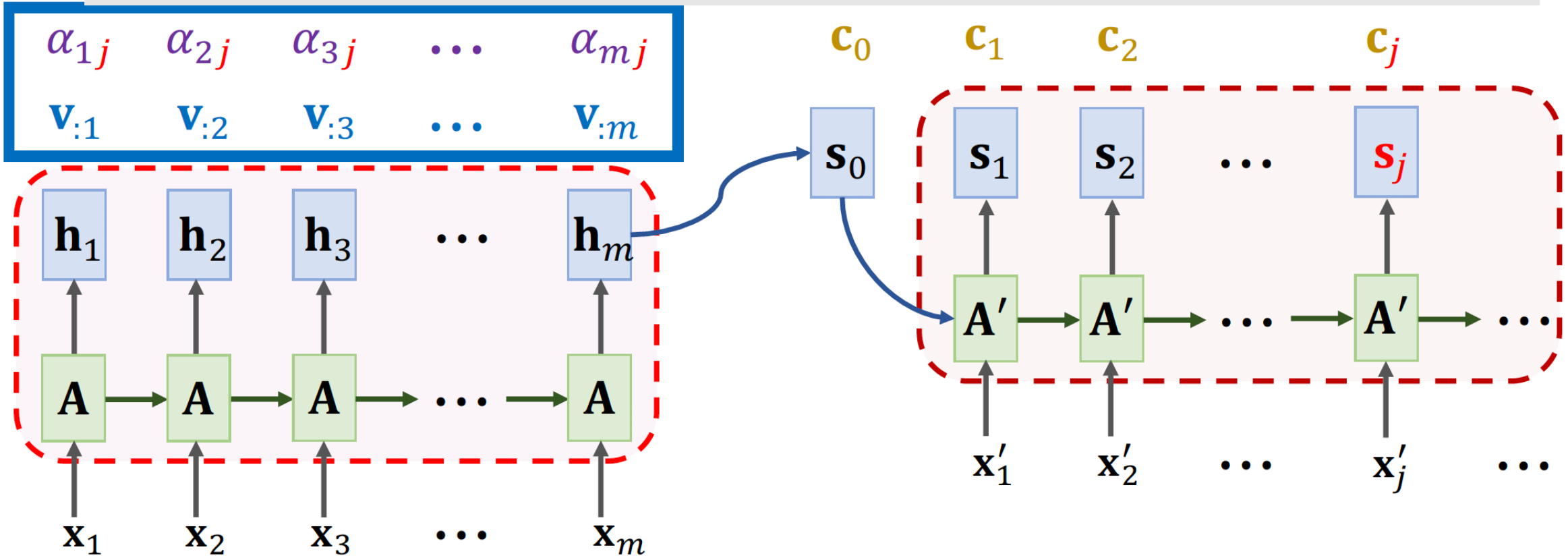


Attention for Seq2Seq Model

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

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

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: $\mathbf{v}_{:i} = \mathbf{W}_V \mathbf{h}_i$.

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

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:

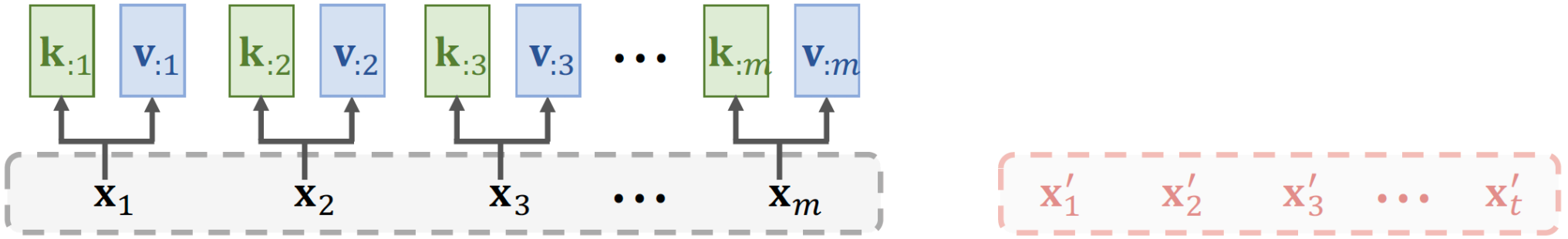
$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\cdots$ $\mathbf{x}_m$

Decoder's inputs:

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

Attention Layer

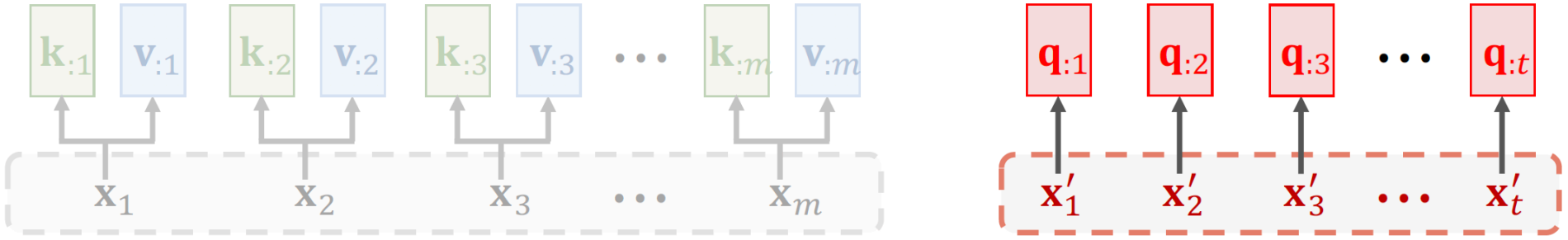
- Keys and values are based on encoder's inputs $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m$.
- Key: $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{x}_i$.
- Value: $\mathbf{v}_{:i} = \mathbf{W}_V \mathbf{x}_i$.



Attention Layer

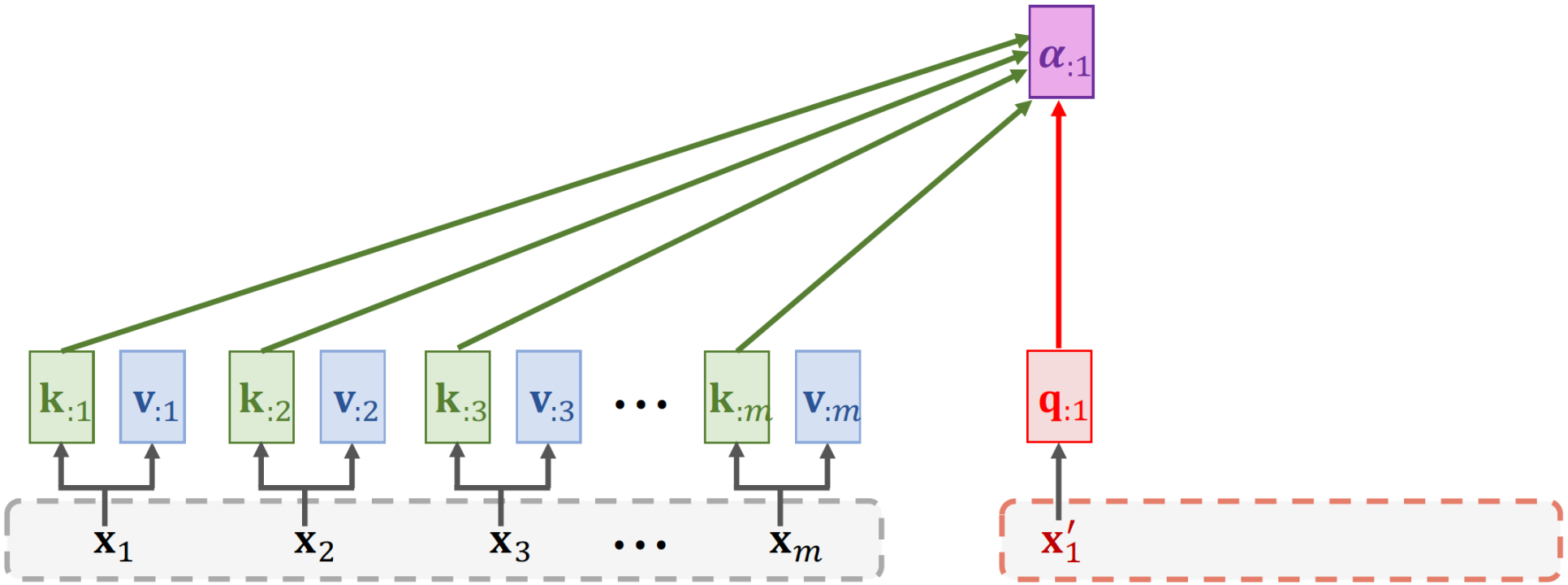
- Keys and values are based on encoder's inputs $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m$.
- Key: $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{x}_i$.
- Value: $\mathbf{v}_{:i} = \mathbf{W}_V \mathbf{x}_i$.

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



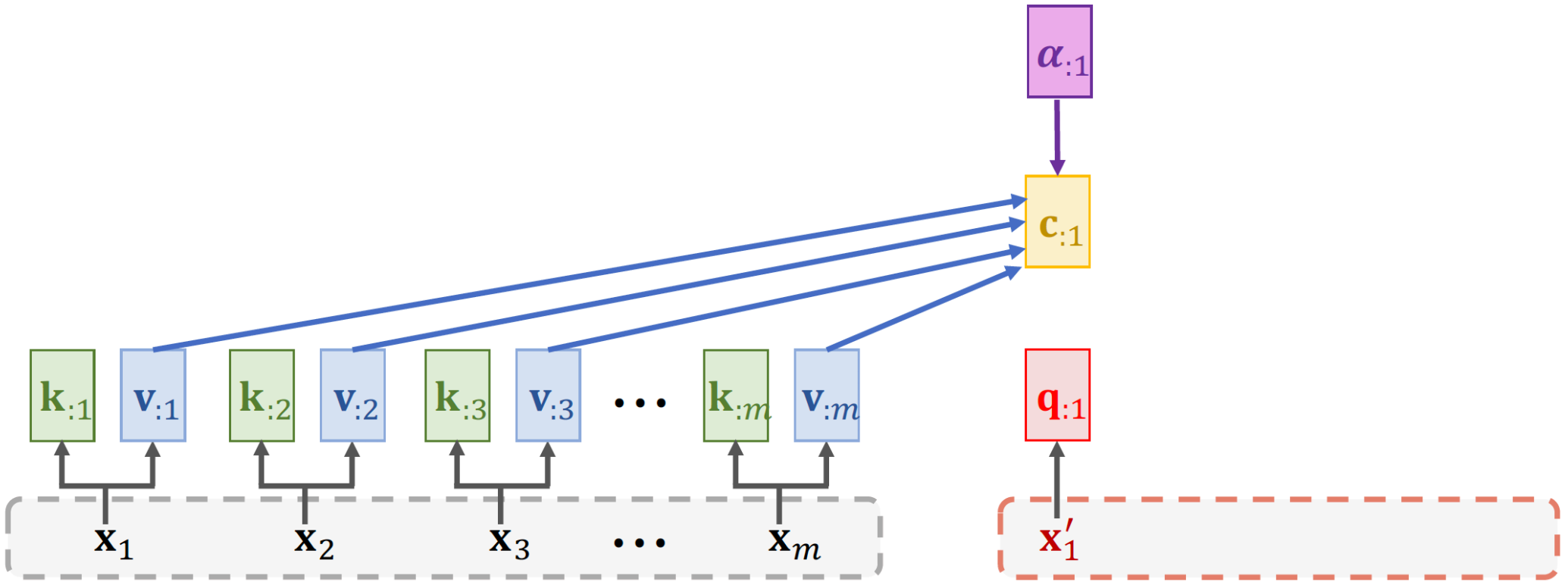
Attention Layer

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



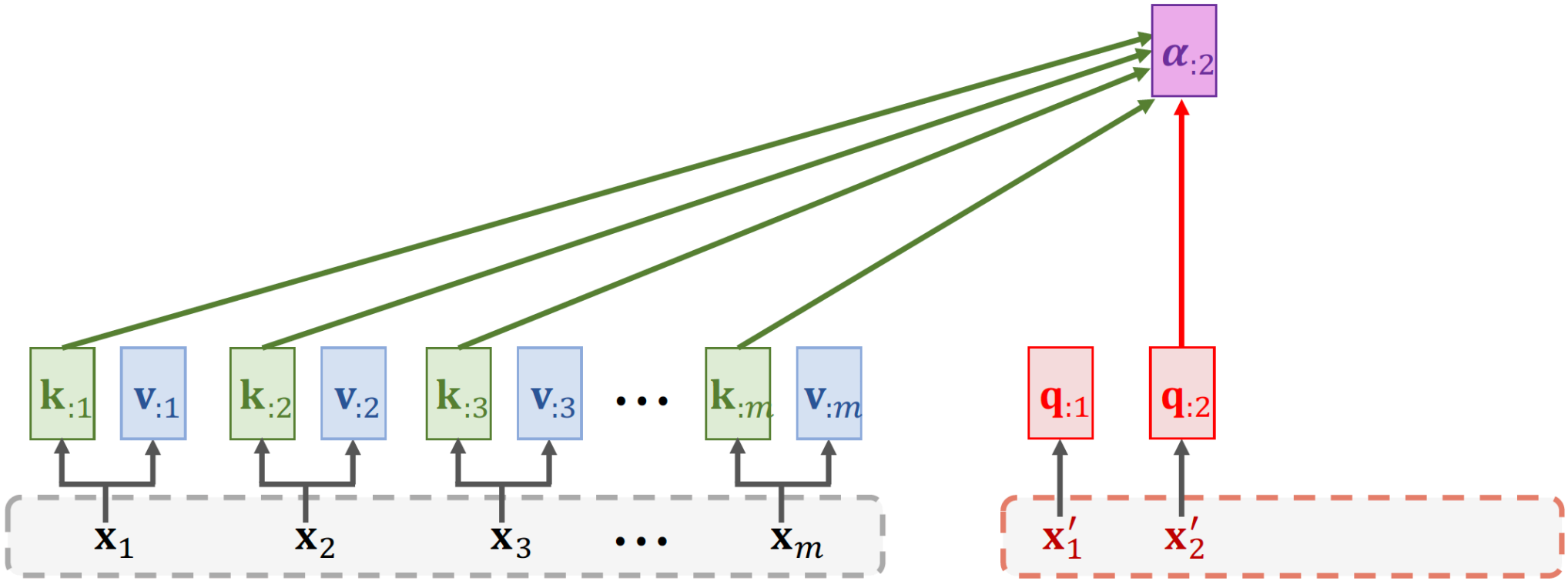
Attention Layer

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



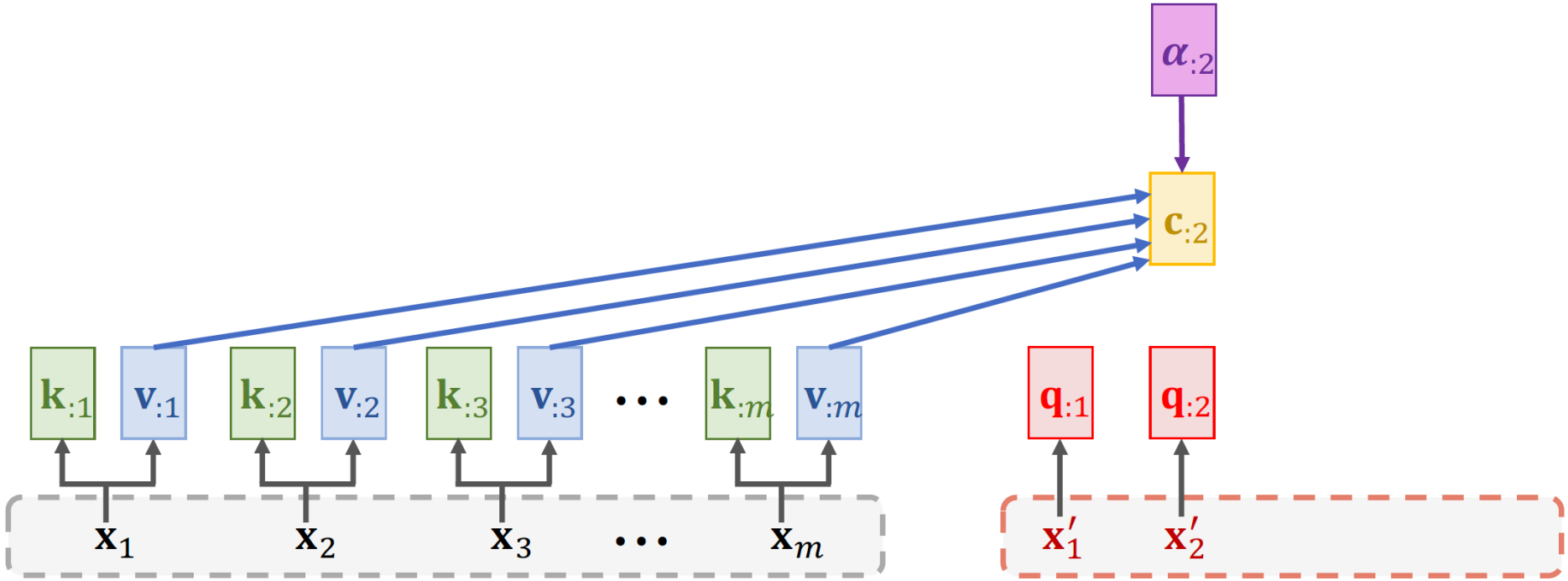
Attention Layer

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



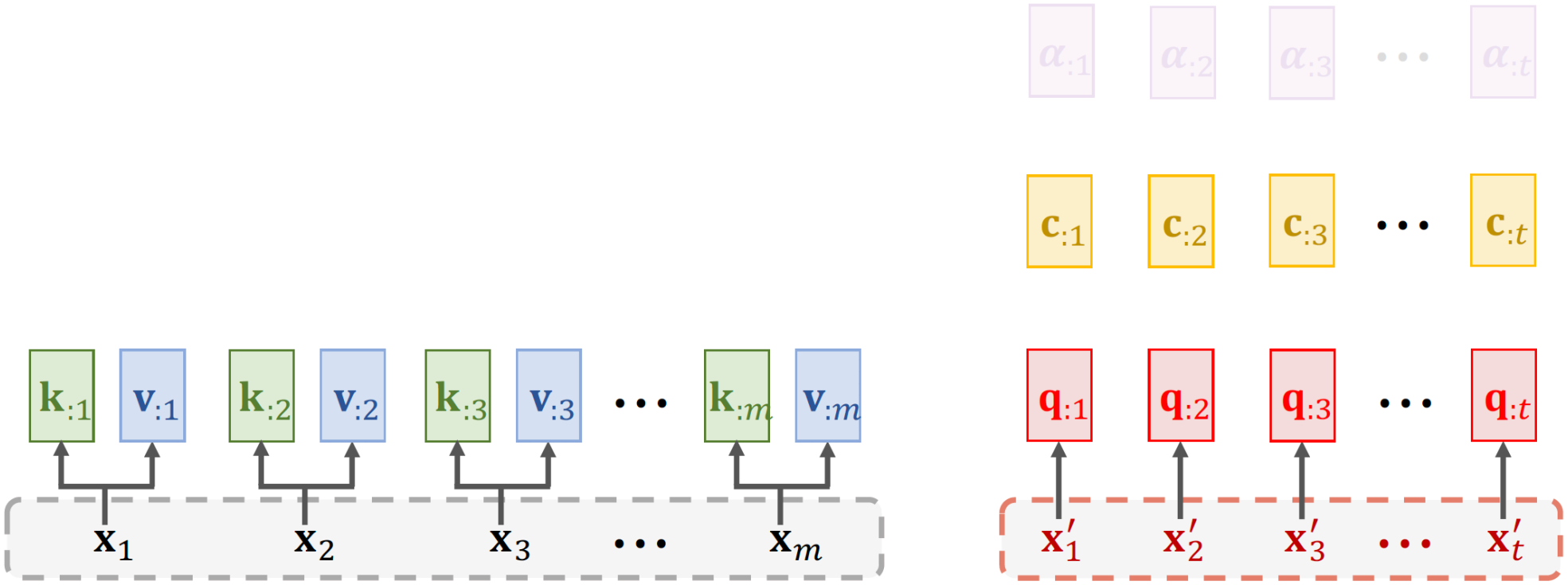
Attention Layer

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



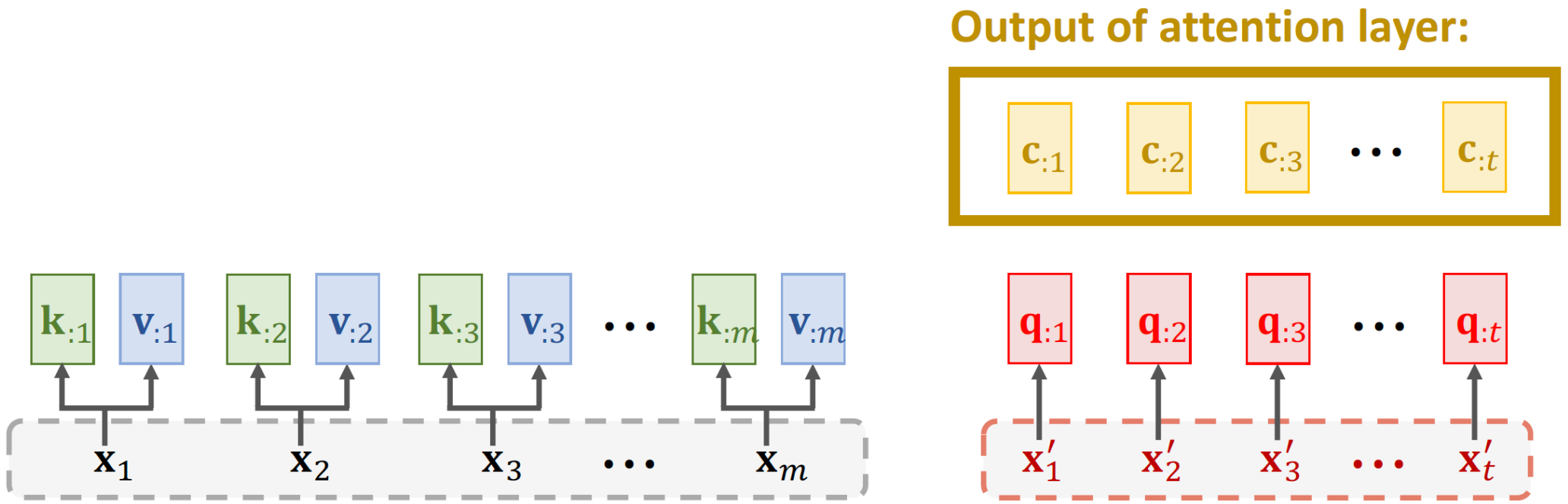
Attention Layer

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



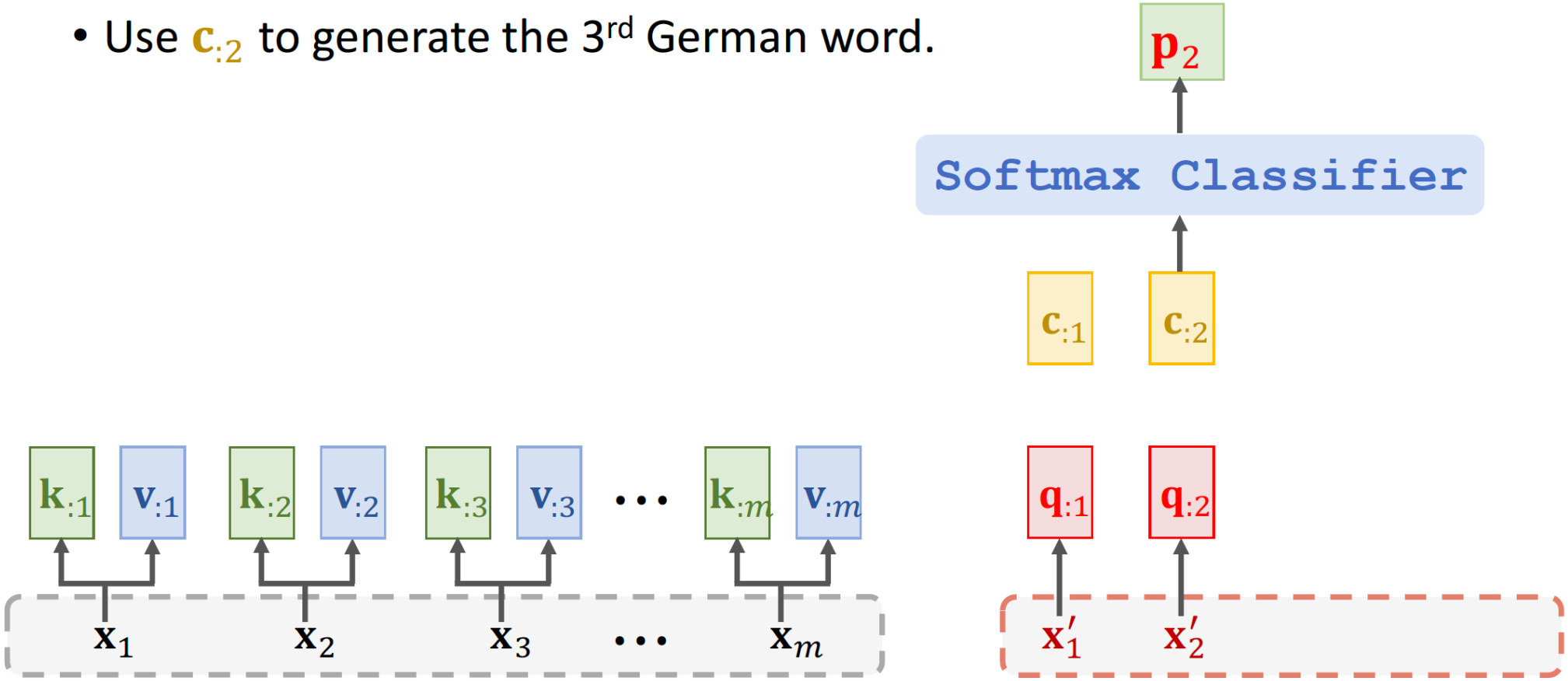
Attention Layer

- Output of attention layer: $\mathbf{C} = [\mathbf{c}_{:1}, \mathbf{c}_{:2}, \mathbf{c}_{:3}, \cdots, \mathbf{c}_{:t}]$.
- Here, $\mathbf{c}_{:j} = \mathbf{V} \cdot \mathrm{Softmax}(\mathbf{K}^T \mathbf{q}_{:j})$.
- Thus, $\mathbf{c}_{:j}$ is a function of $\mathbf{x}'_j$ and $[\mathbf{x}_1, \cdots, \mathbf{x}_m]$.



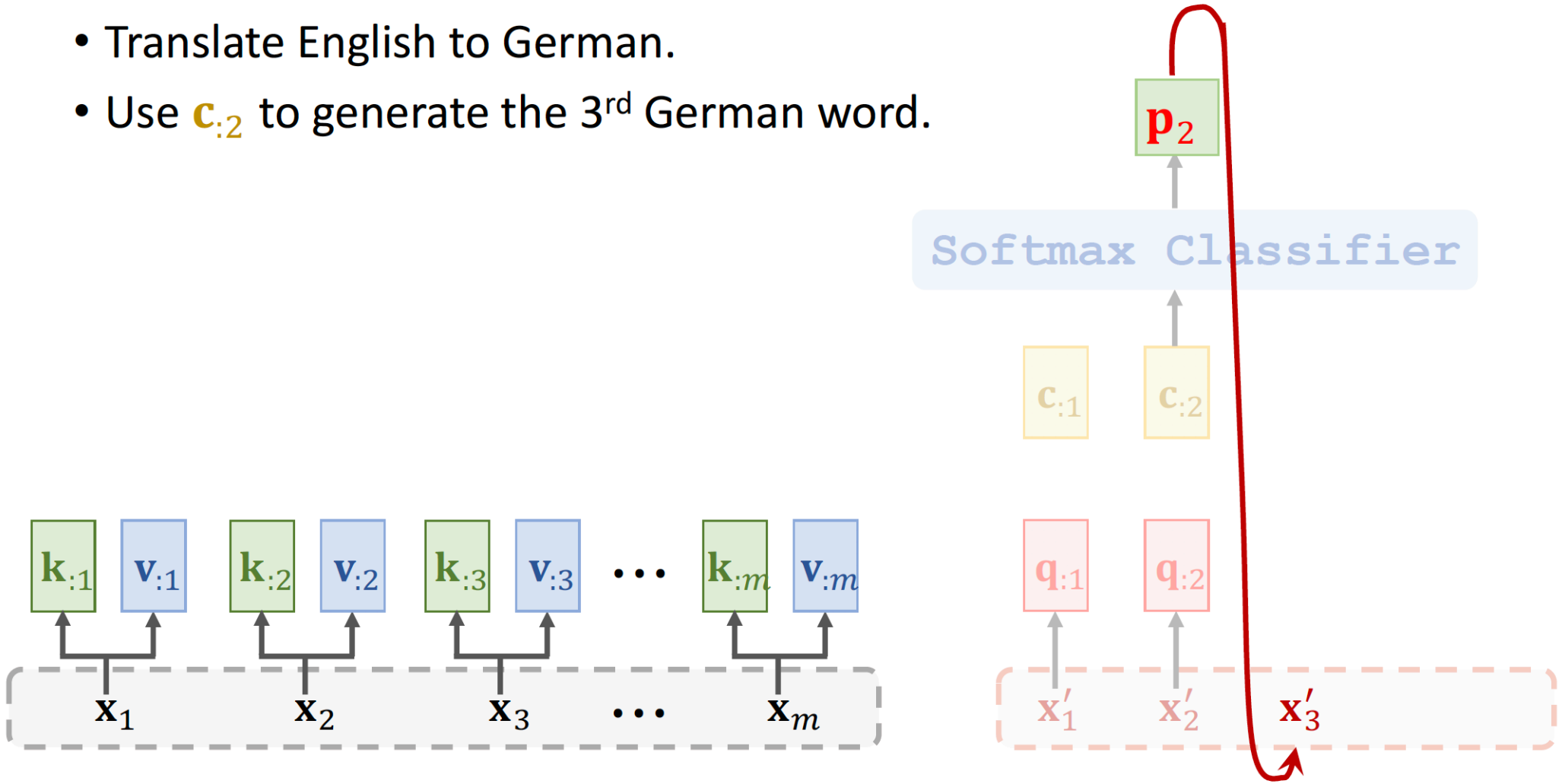
Attention Layer for Machine Translation

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



Attention Layer for Machine Translation

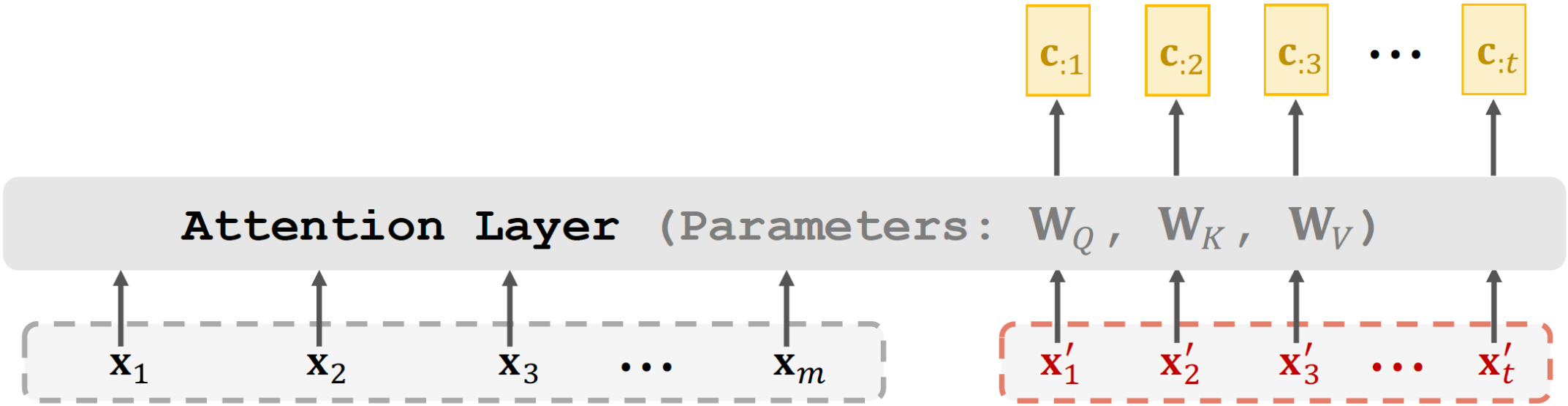
- Translate English to German.
- Use $\mathbf{c}_{:2}$ to generate the 3rd German word.



Attention Layer

- Attention layer: $\mathbf{C} = \mathrm{Attn}(\mathbf{X}, \mathbf{X}')$.
  - 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: $\mathbf{W}_Q$, $\mathbf{W}_K$, $\mathbf{W}_V$.

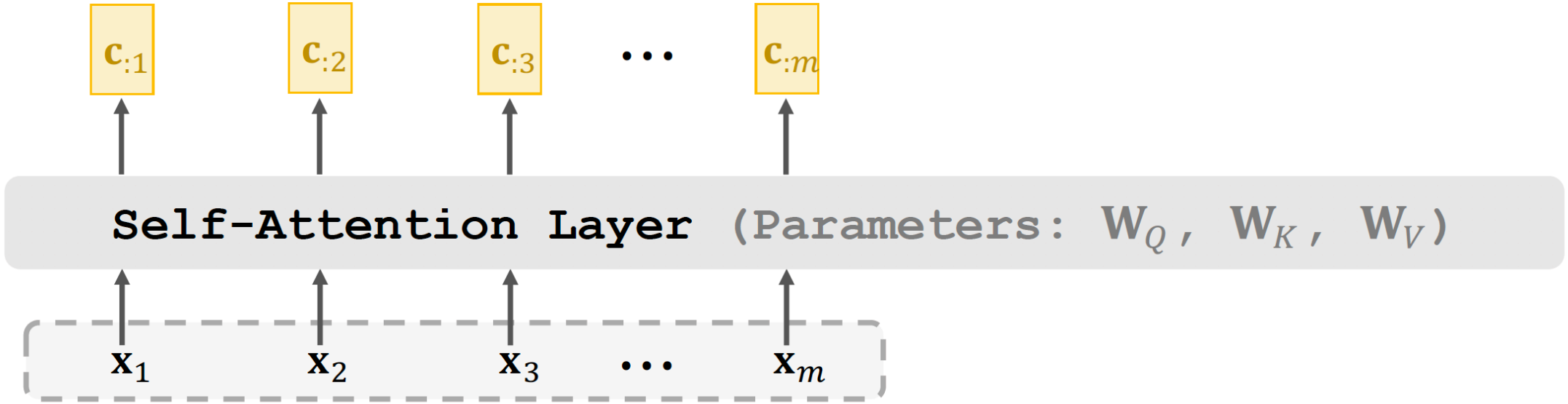
- **Query:** $\mathbf{q}_{:j} = \mathbf{W}_Q \mathbf{x}'_j$,
- **Key:** $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{x}_i$,
- **Value:** $\mathbf{v}_{:i} = \mathbf{W}_V \mathbf{x}_i$.
- **Output:** $\mathbf{c}_{:j} = \mathbf{V} \cdot \mathrm{Softmax}\left(\mathbf{K}^T \mathbf{q}_{:j}\right)$.



Self-Attention without RNN

Self-Attention Layer

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

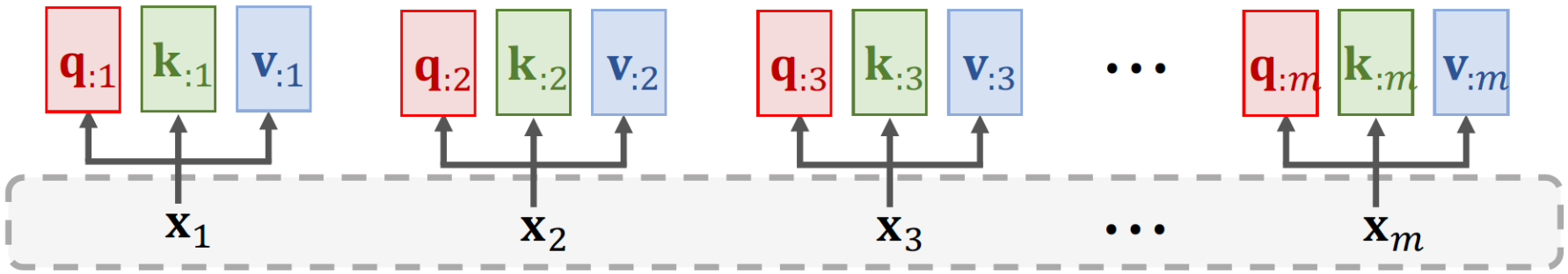


Self-Attention Layer

Inputs:

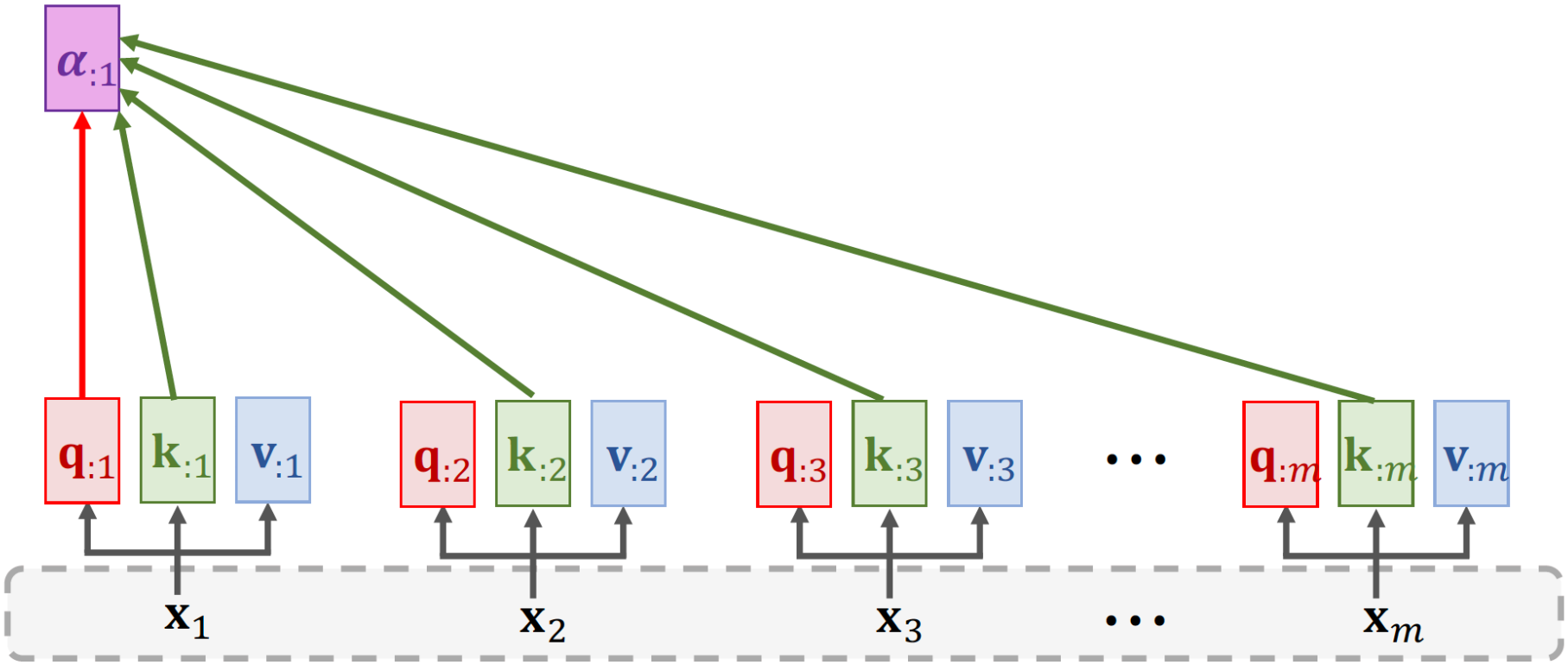
Self-Attention Layer

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



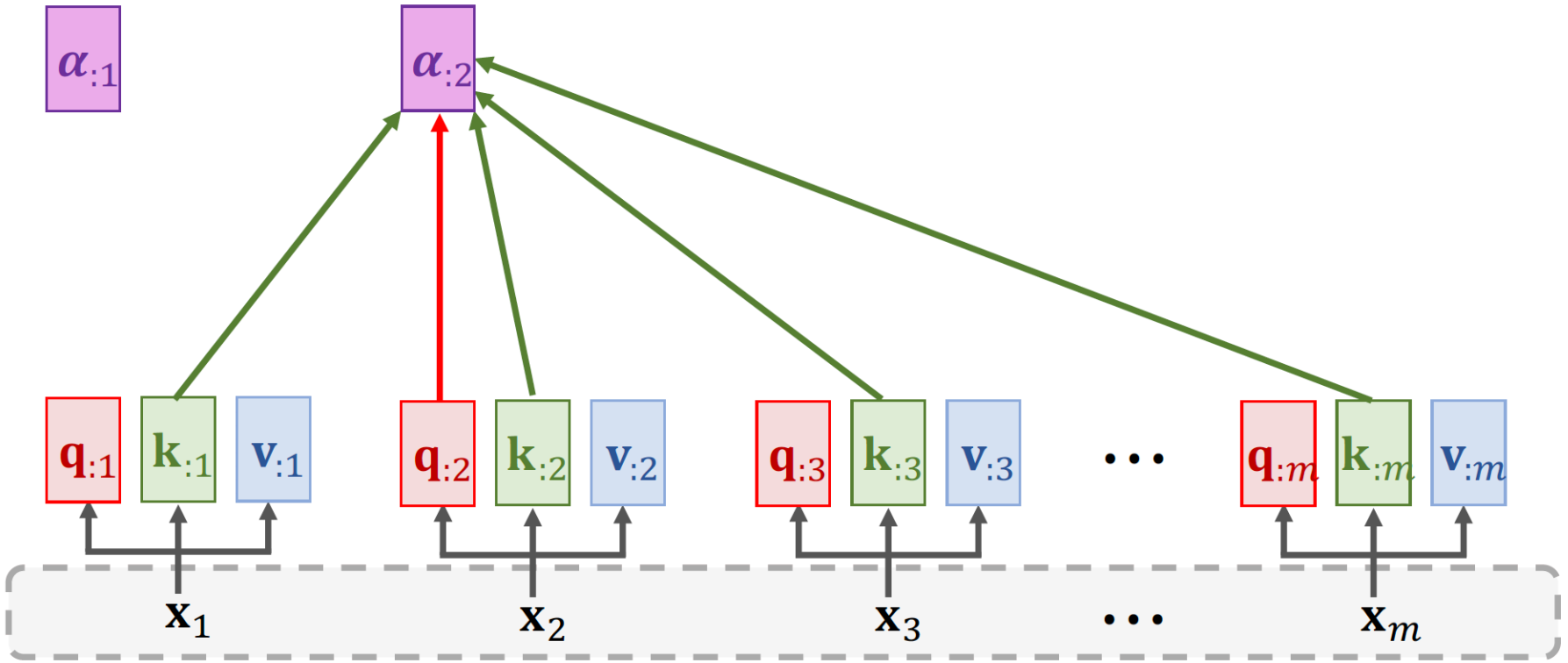
Self-Attention Layer

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



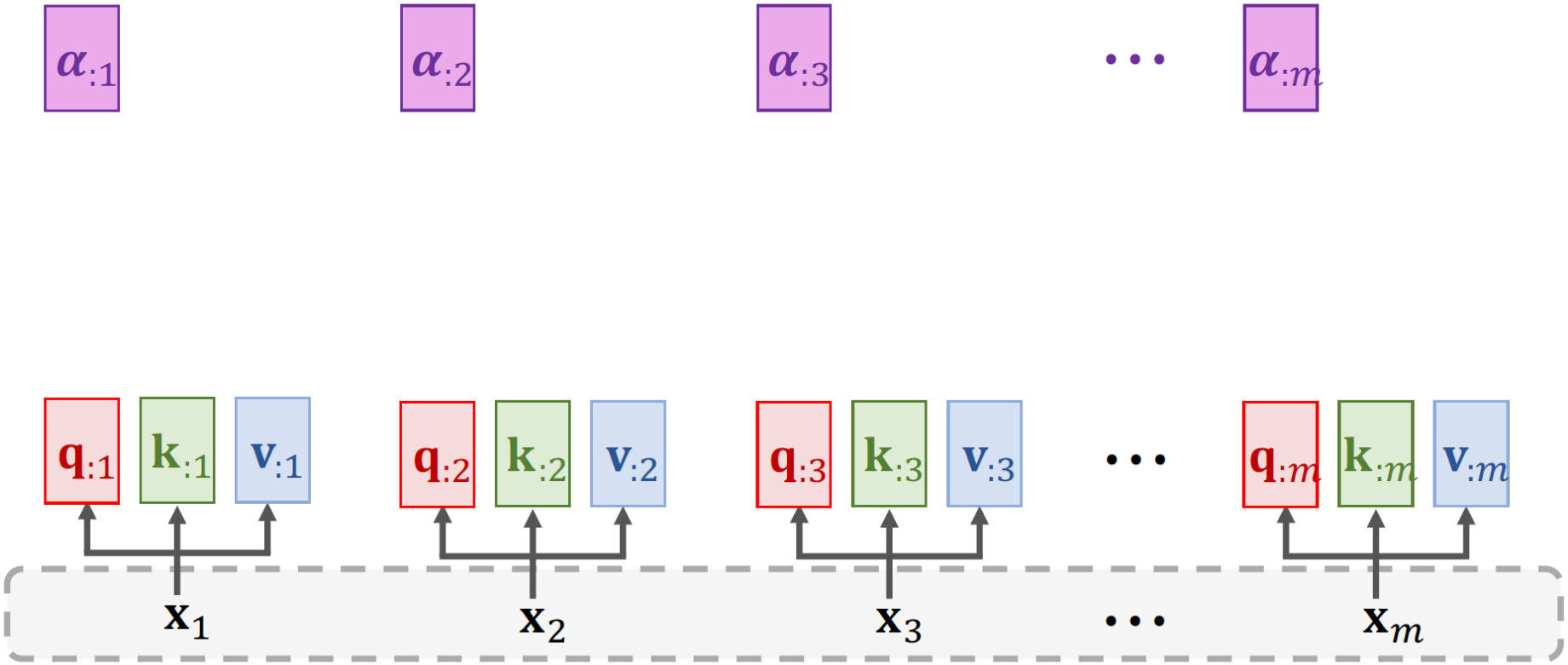
Self-Attention Layer

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



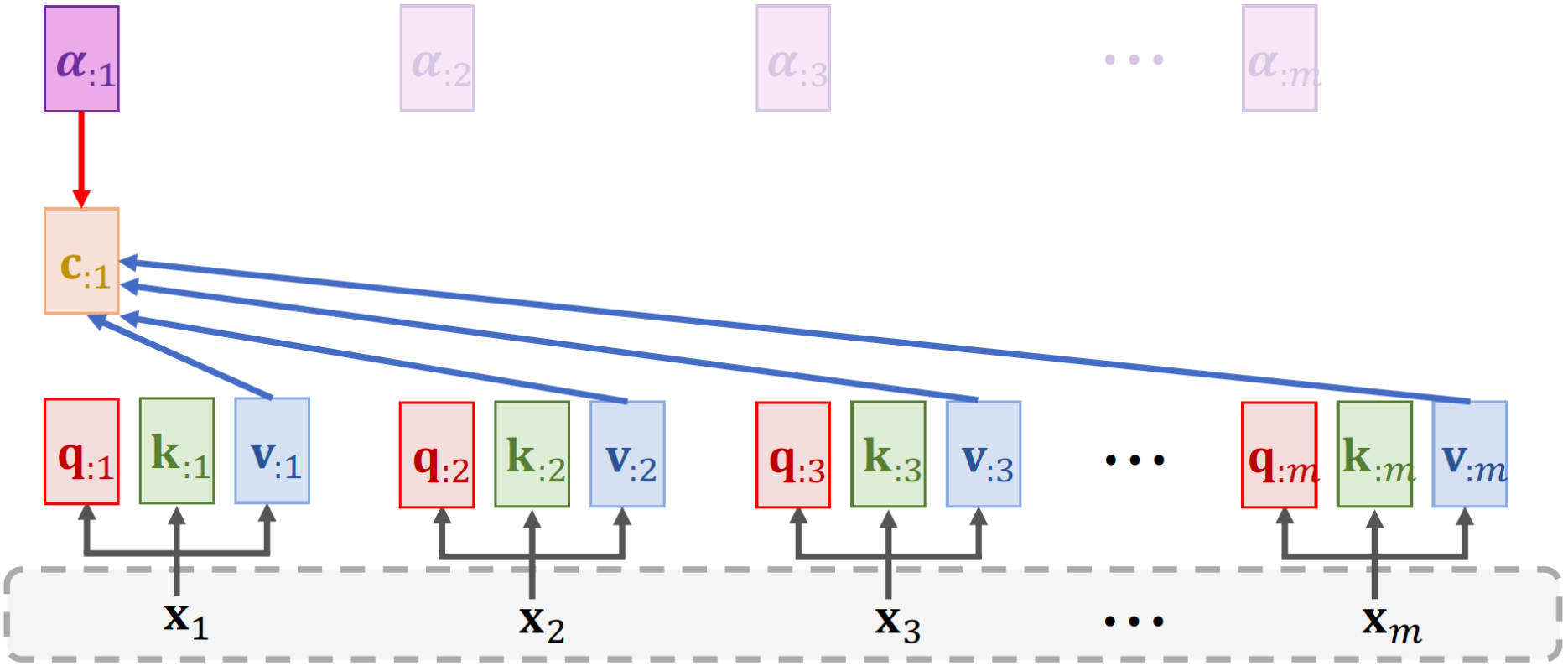
Self-Attention Layer

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



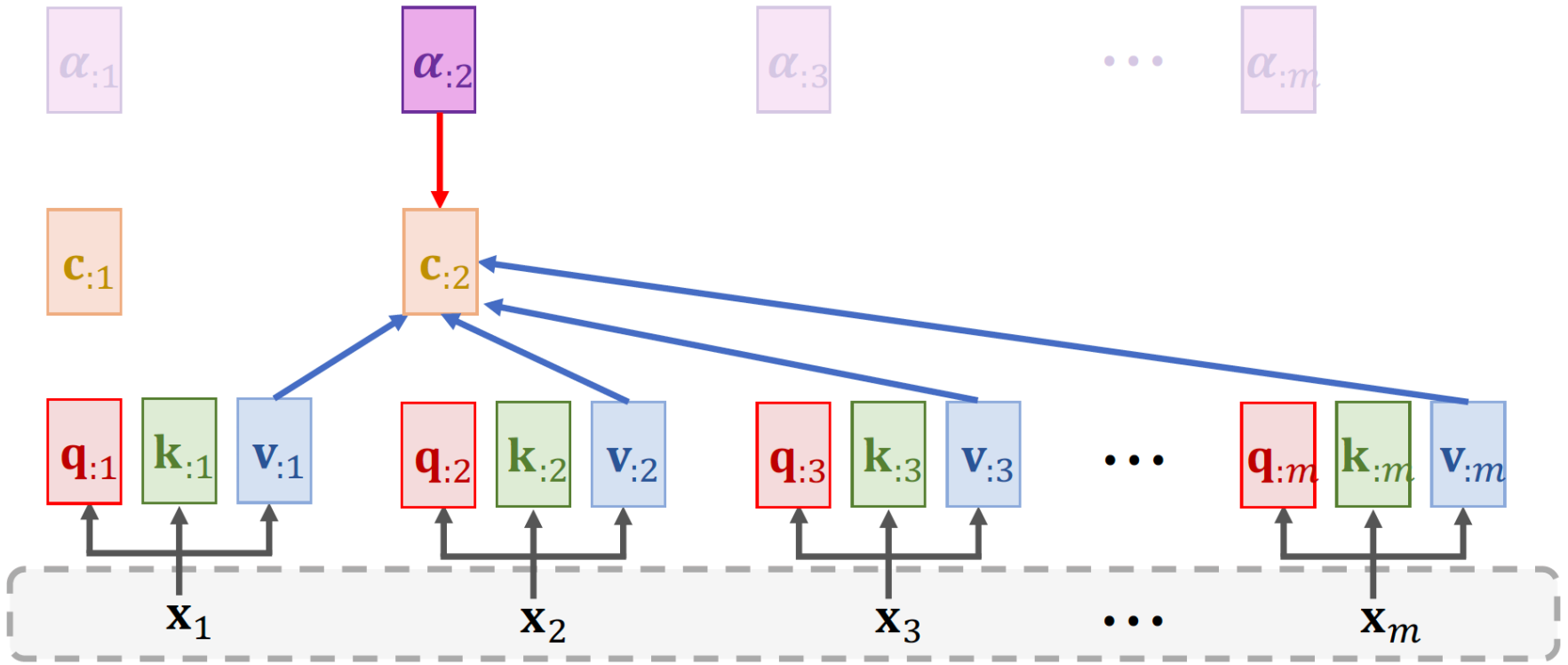
Self-Attention Layer

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



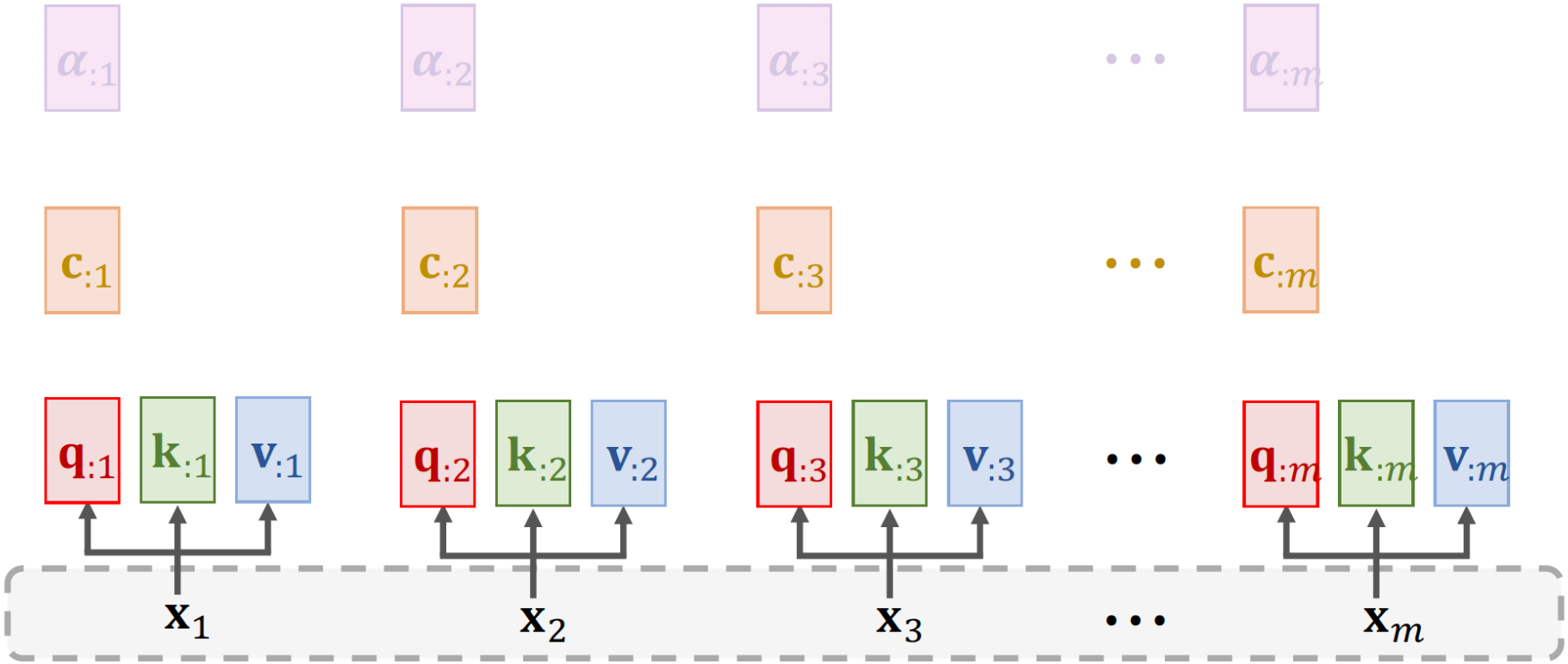
Self-Attention Layer

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



Self-Attention Layer

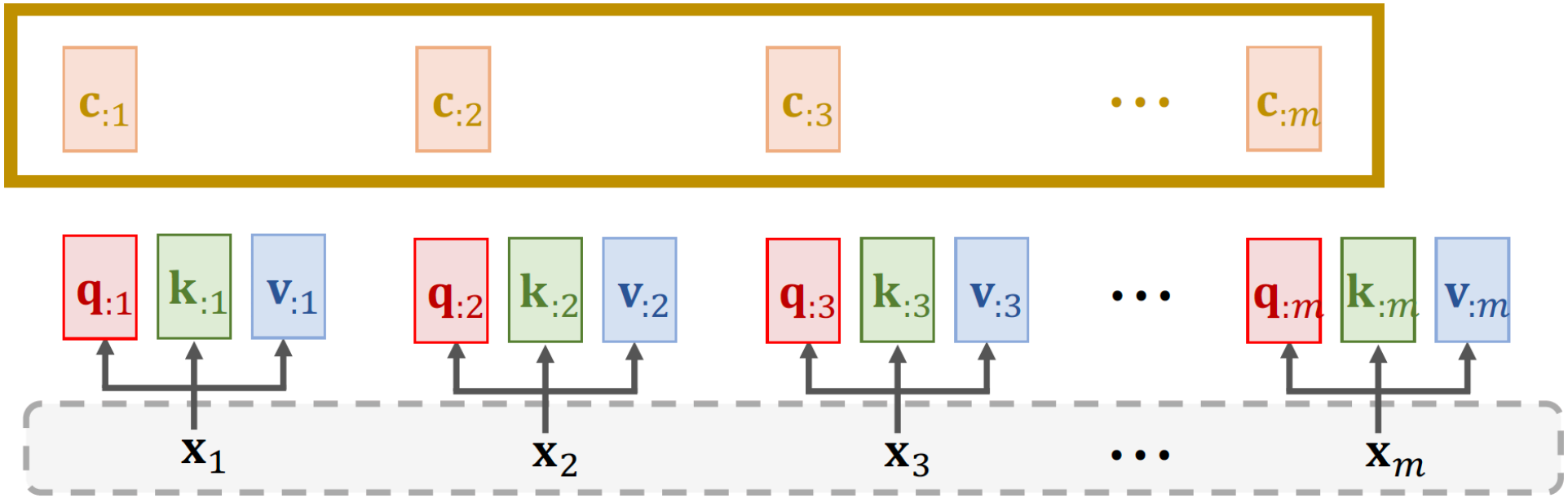
Context vector: $\mathbf{c}_{:j} = \alpha_{1j}\mathbf{v}_{:1} + \cdots + \alpha_{mj}\mathbf{v}_{:m} = \mathbf{V}\boldsymbol{\alpha}_{:j}.$



Self-Attention Layer

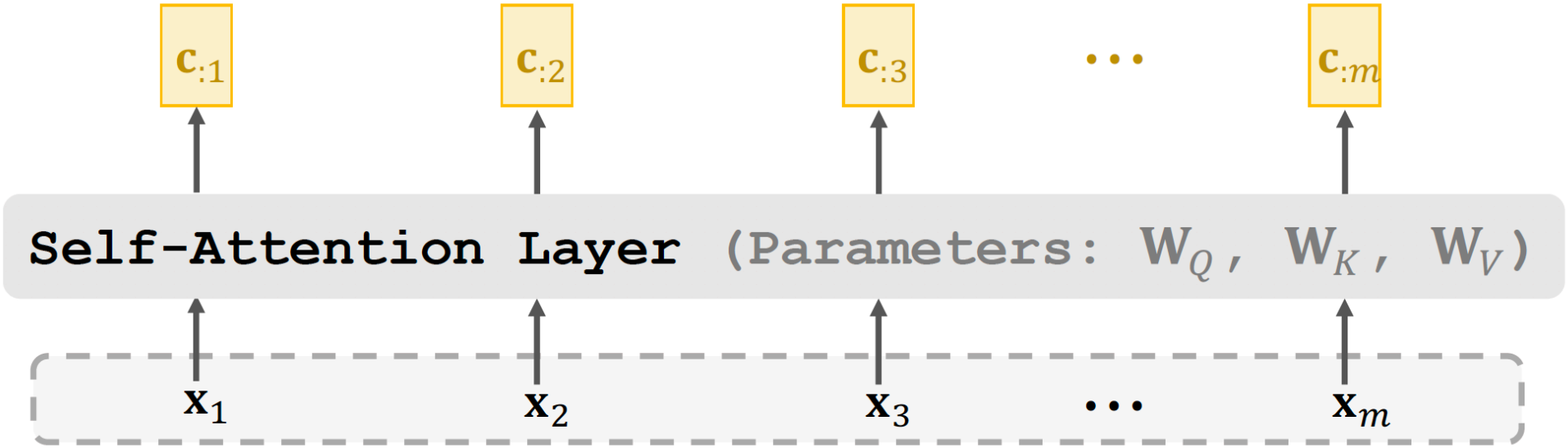
- Here, $\mathbf{c}_{:j} = \mathbf{V} \cdot \mathrm{Softmax}\left(\mathbf{K}^T \mathbf{q}_{:j}\right)$.
- Thus, $\mathbf{c}_{:j}$ is a function of all the m vectors $\mathbf{x}_1, \cdots, \mathbf{x}_m$.

Output of self-attention layer:



Self-Attention Layer

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



Summary

- 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.

Transformer Model

- Transformer is Seq2Seq model; it has an encoder and a decoder.
- Transformer model is not RNN.
- Transformer is based on multi-head attention and multi-head self-attention.

From Single-Head to Multi-Head

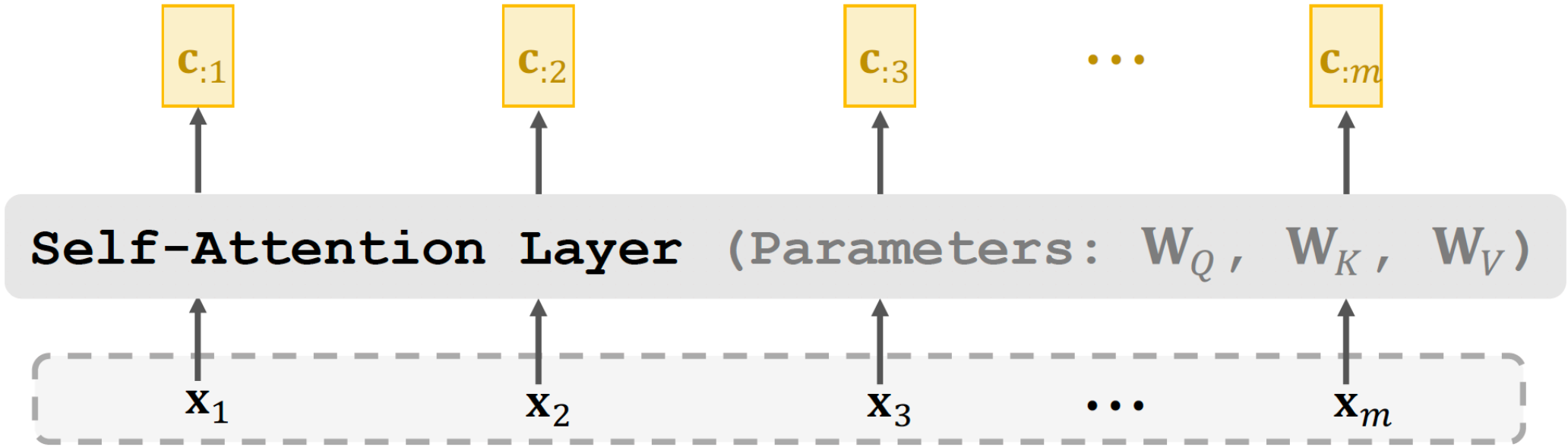
- Single-head self-attention can be combined to form a multi-head self-attention.

- Single-head attention can be combined to form a multi-head attention.

Multi-Head (Self-)Attention

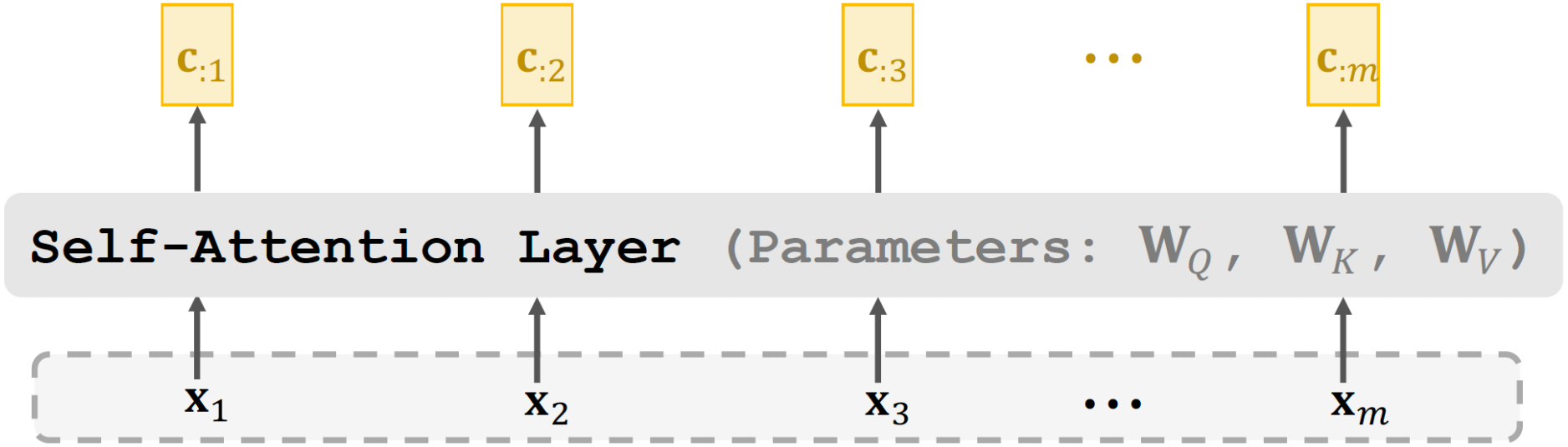
Single-Head Self-Attention

- Self-attention layer: $\mathbf{C} = \mathrm{Attn}(\mathbf{X}, \mathbf{X})$.
- This is called "single-head self-attention".



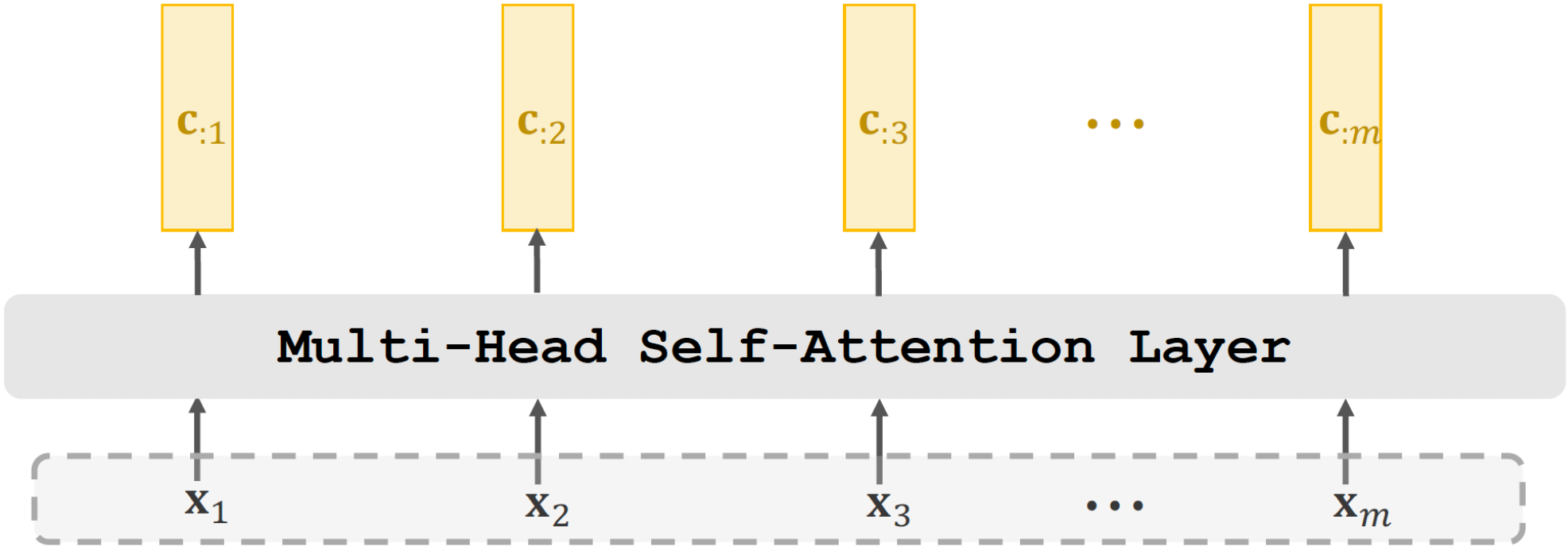
Multi-Head Self-Attention

- Using l single-head self-attentions (which do not share parameters.)
  - A single-head self-attention has 3 parameter matrices: $\mathbf{W}_Q$, $\mathbf{W}_K$, $\mathbf{W}_V$.
  - Totally $3l$ parameters matrices.



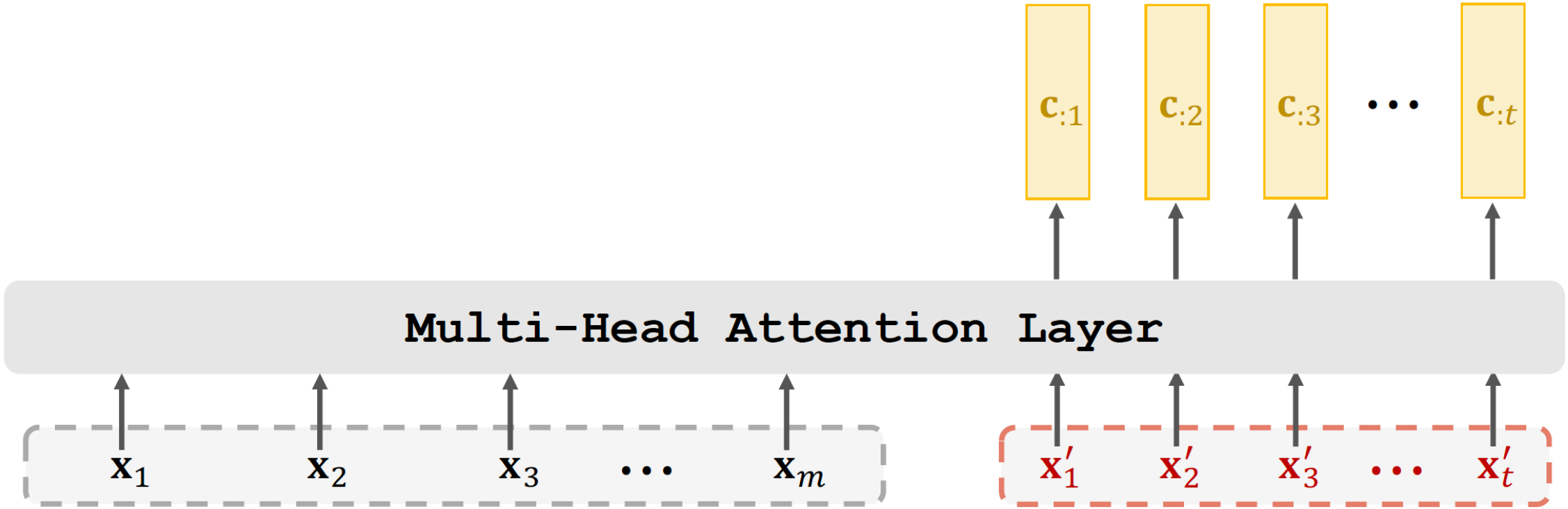
Multi-Head Self-Attention

- Using l single-head self-attentions (which do not share parameters.)
- Concatenating outputs of single-head self-attentions.
  - Suppose single-head self-attentions' outputs are $d \times m$ matrices.
  - Multi-head's output shape: $(ld) \times m$.



Multi-Head Attention

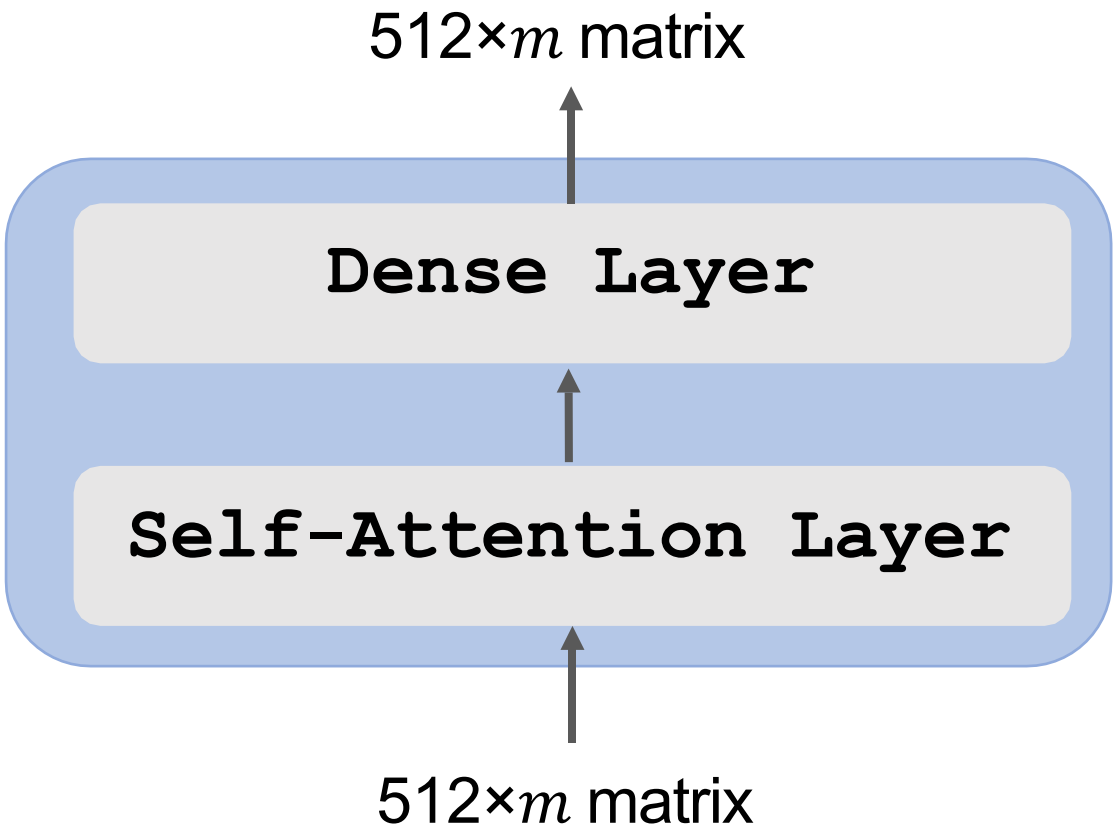
- Using l single-head attentions (which do not share parameters.)
- Concatenating single-head attentions' outputs.



Transformer's Encoder

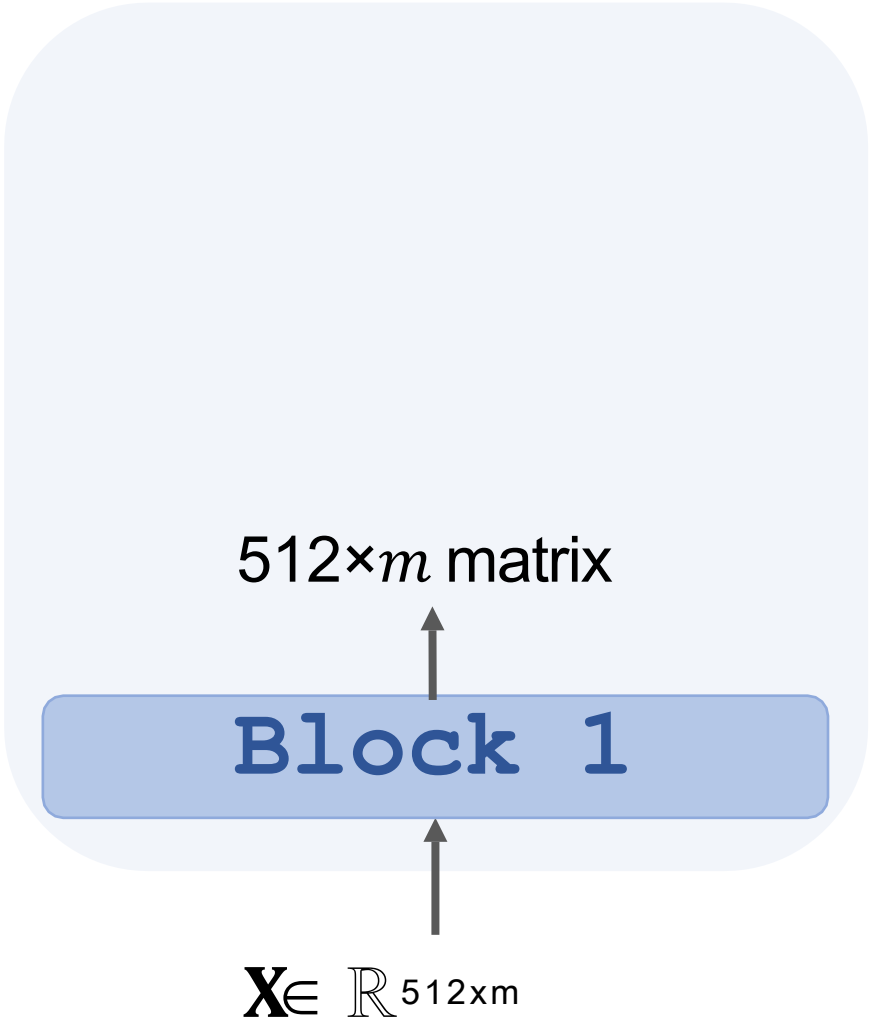
- Encoder is a stack of 6 blocks.
- 1 encoder block ≈ multi-head self-attention + dense.

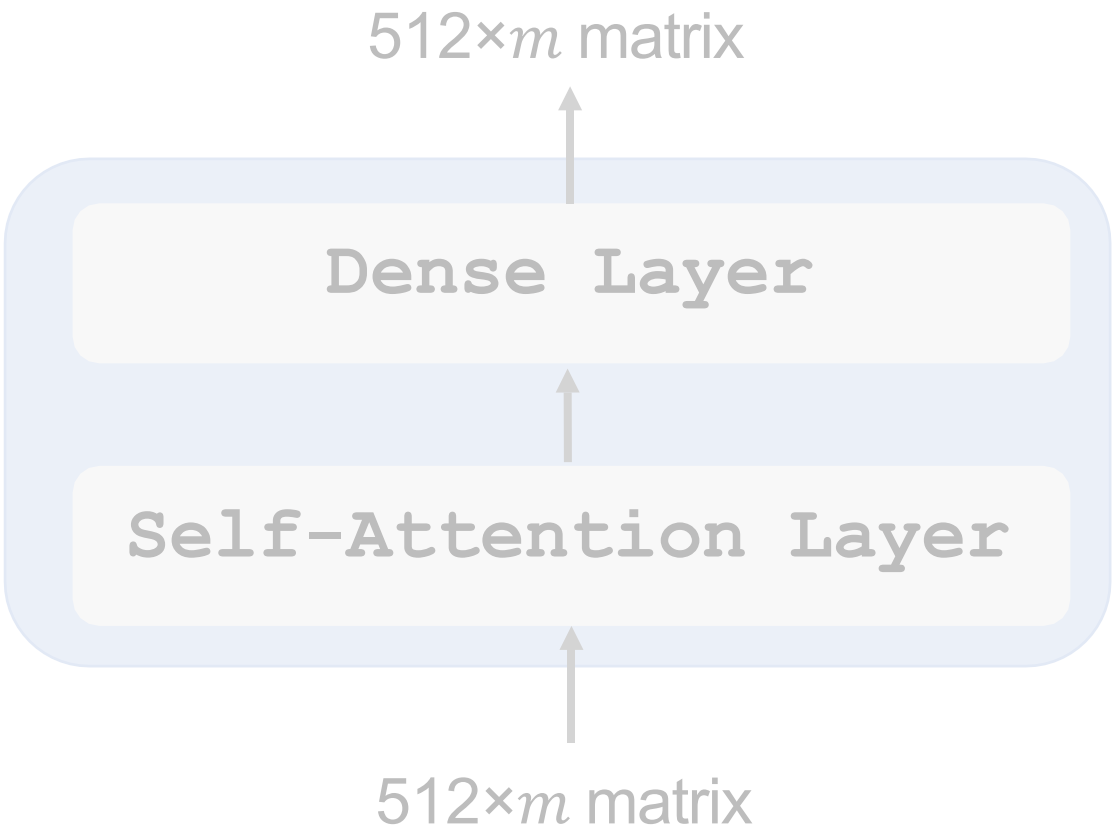
Transformer's Encoder



One Block of Encoder

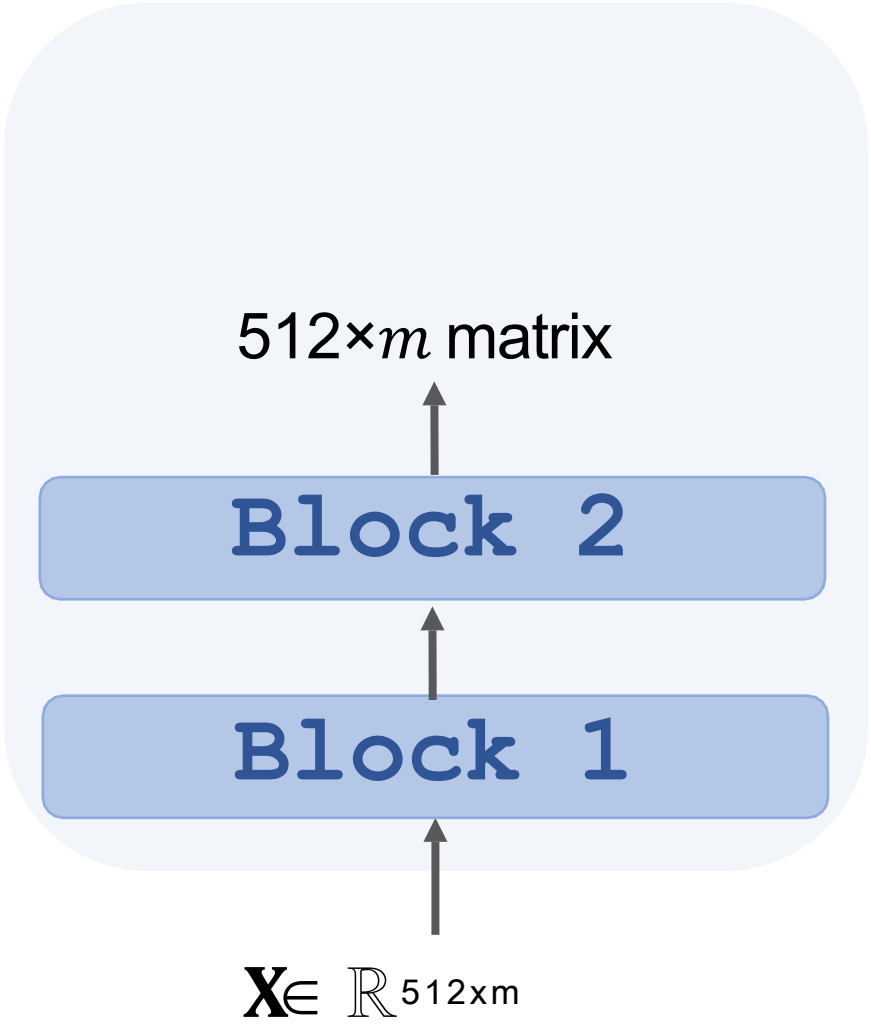
Transformer's Encoder

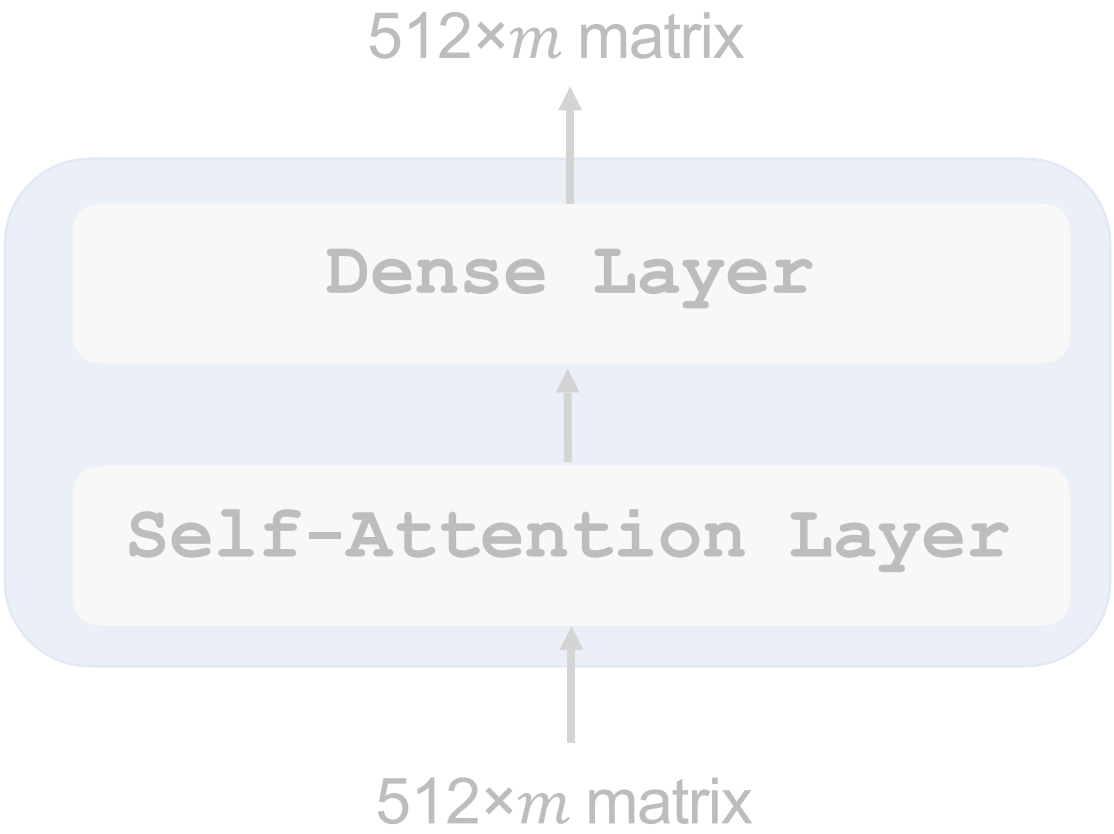




One Block of Encoder

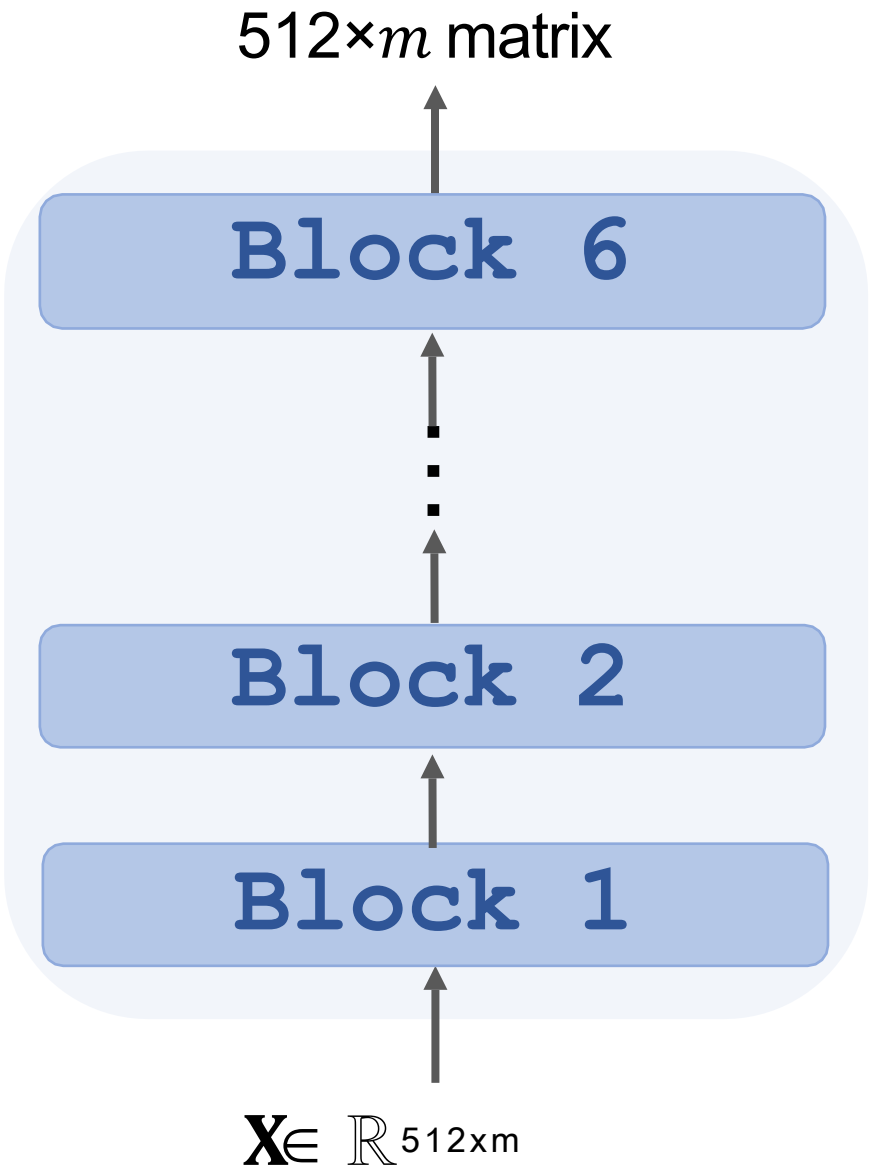
Transformer's Encoder

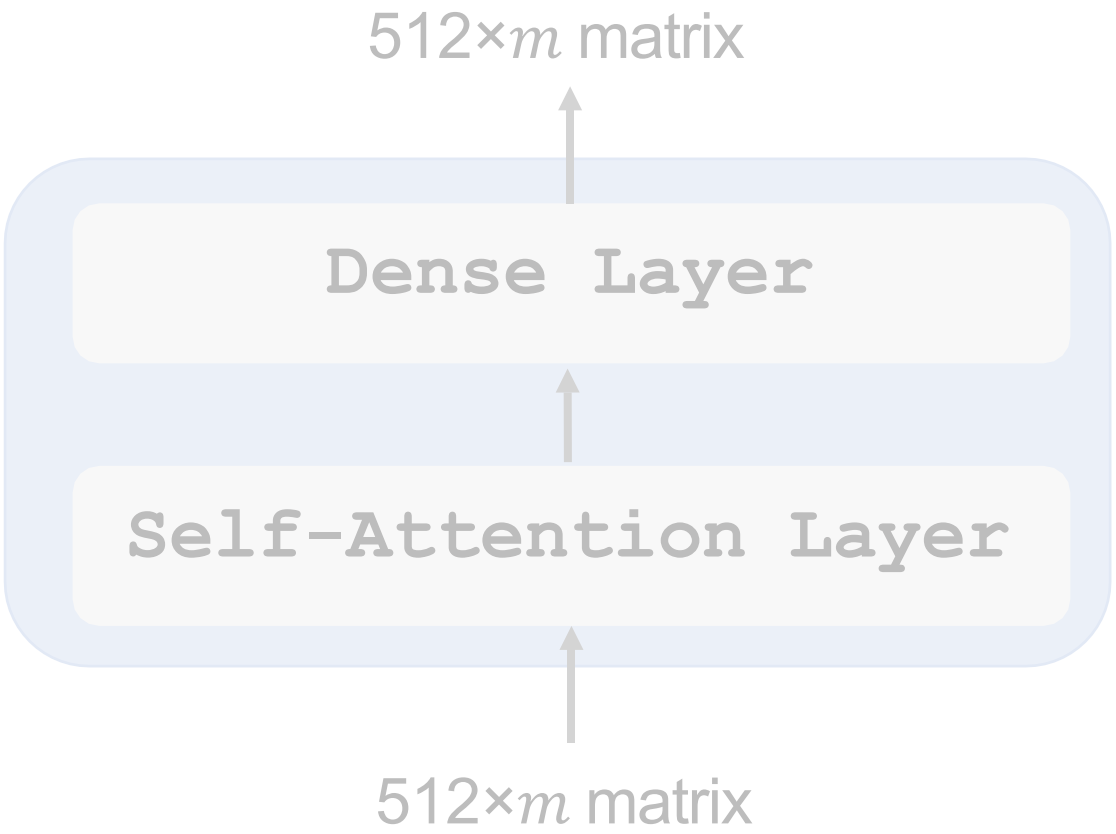




One Block of Encoder

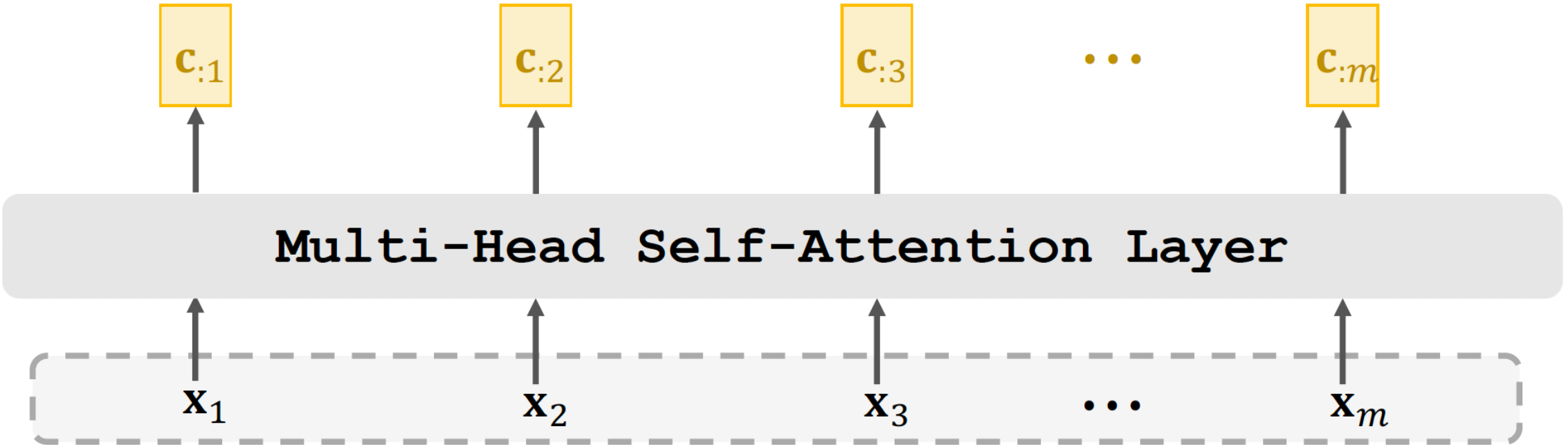
Transformer's Encoder



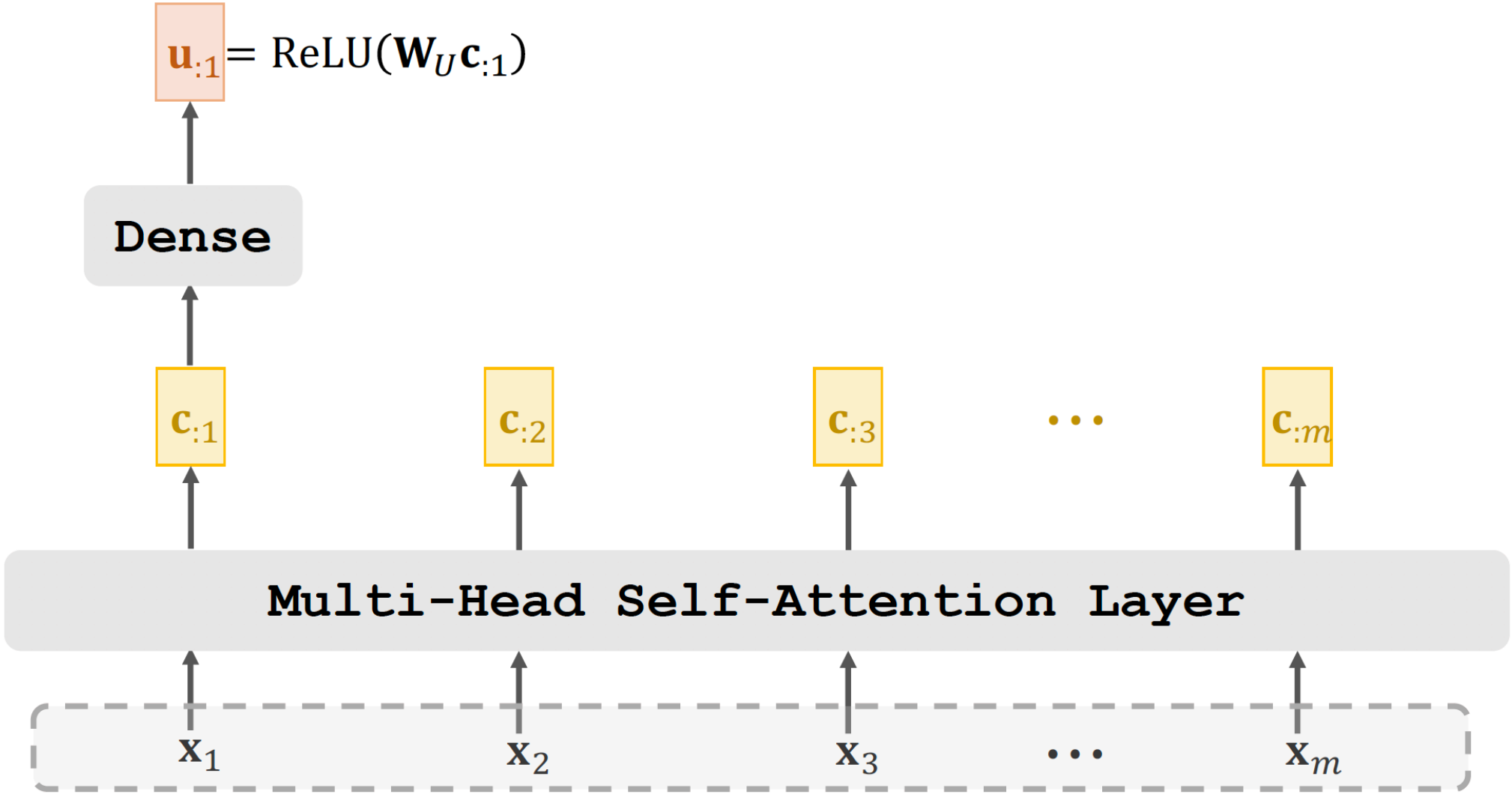


One Block of Encoder

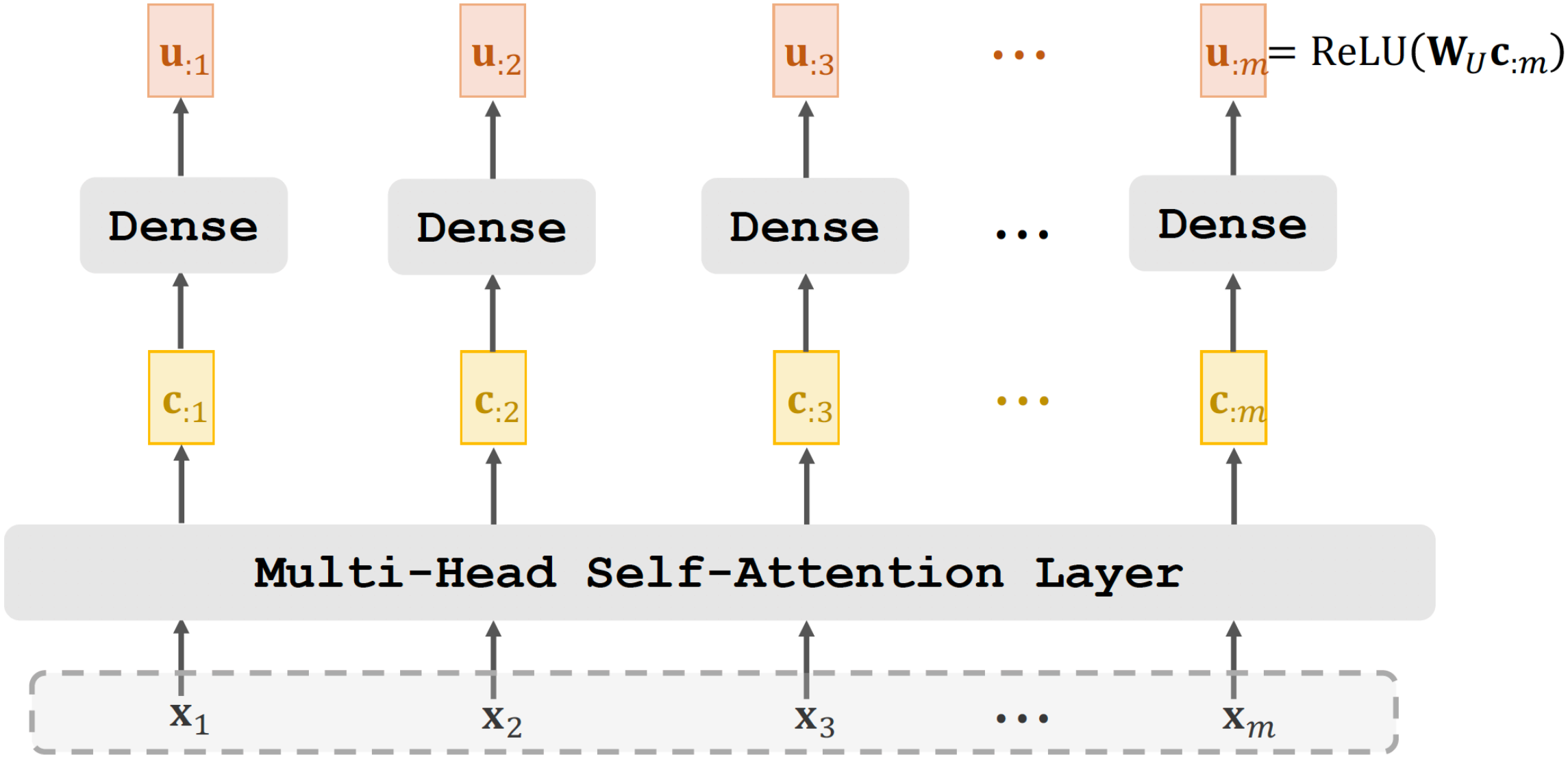
Self-Attention Layer



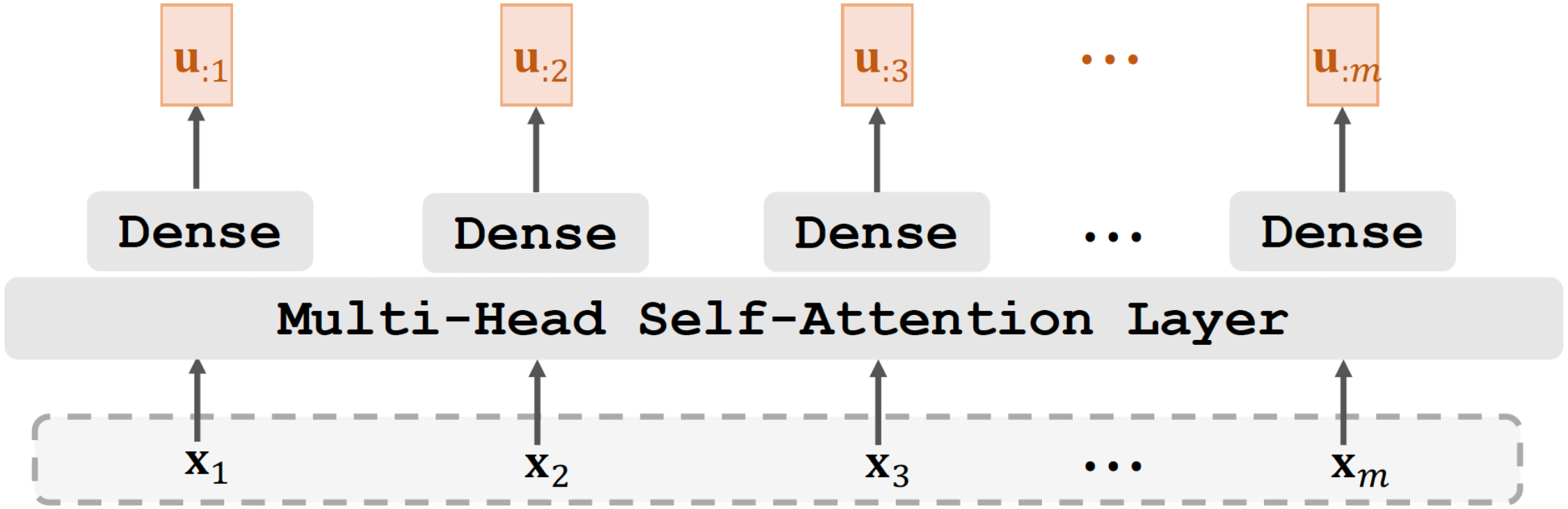
Self-Attention Layer + Dense Layer



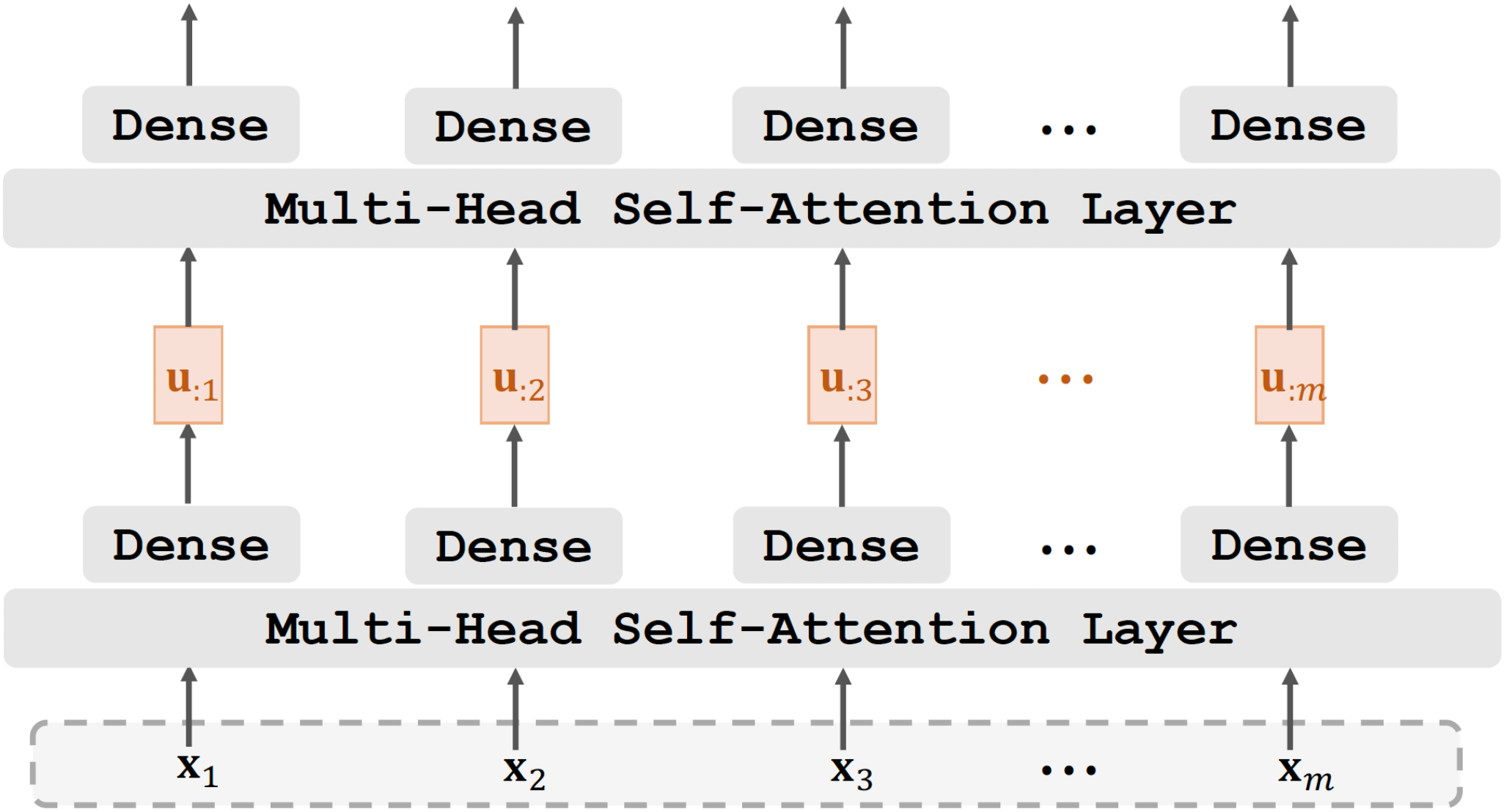
Self-Attention Layer + Dense Layer



Stacked Self-Attention Layers



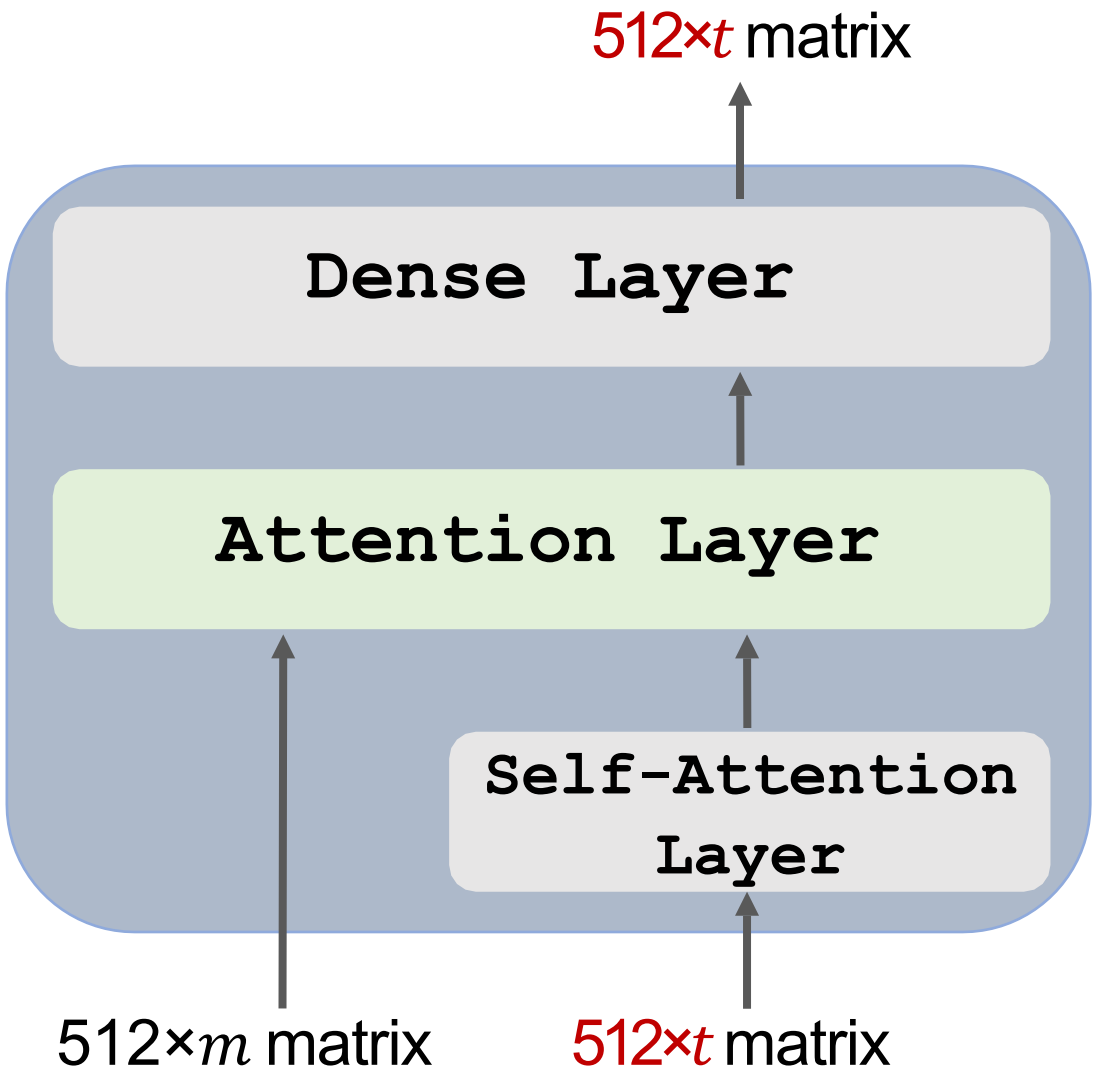
Stacked Self-Attention Layers

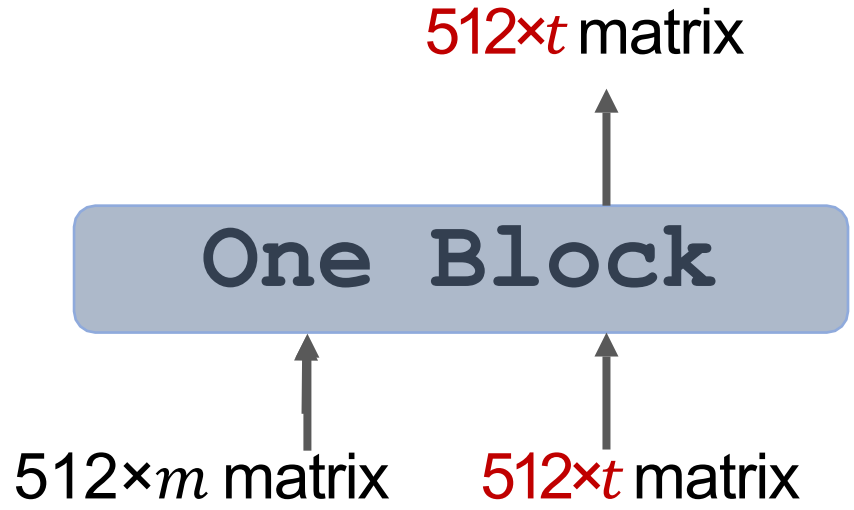


Transformer's Decoder

- Decoder network is a stack of 6 blocks.
- 1 decoder block ≈ multi-head self-attention + multi-head attention + dense.

Transformer's Decoder :One Block





Stacked Attentions

- Transformer is a 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:

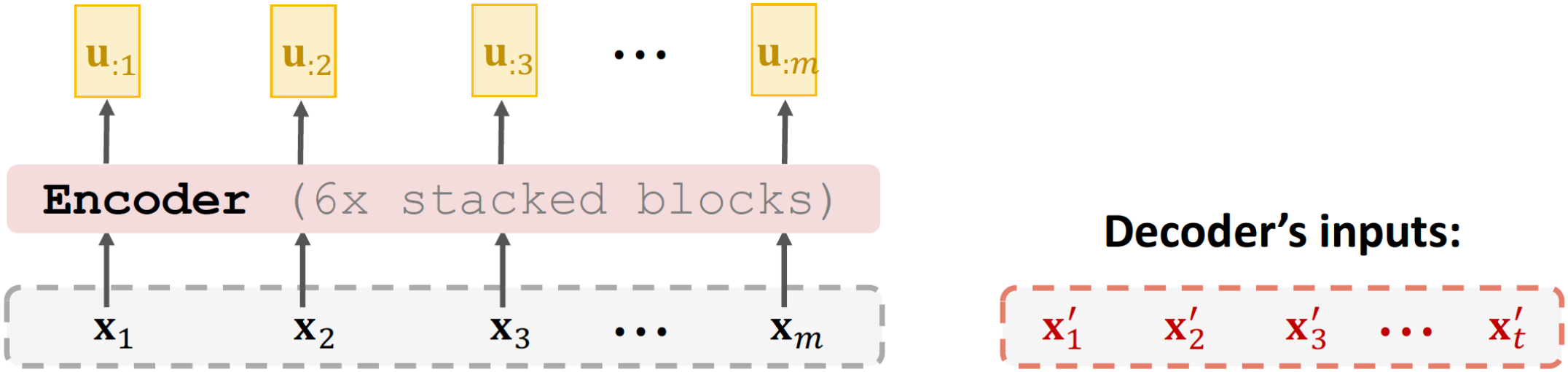
$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\cdots$ $\mathbf{x}_m$

Decoder's inputs:

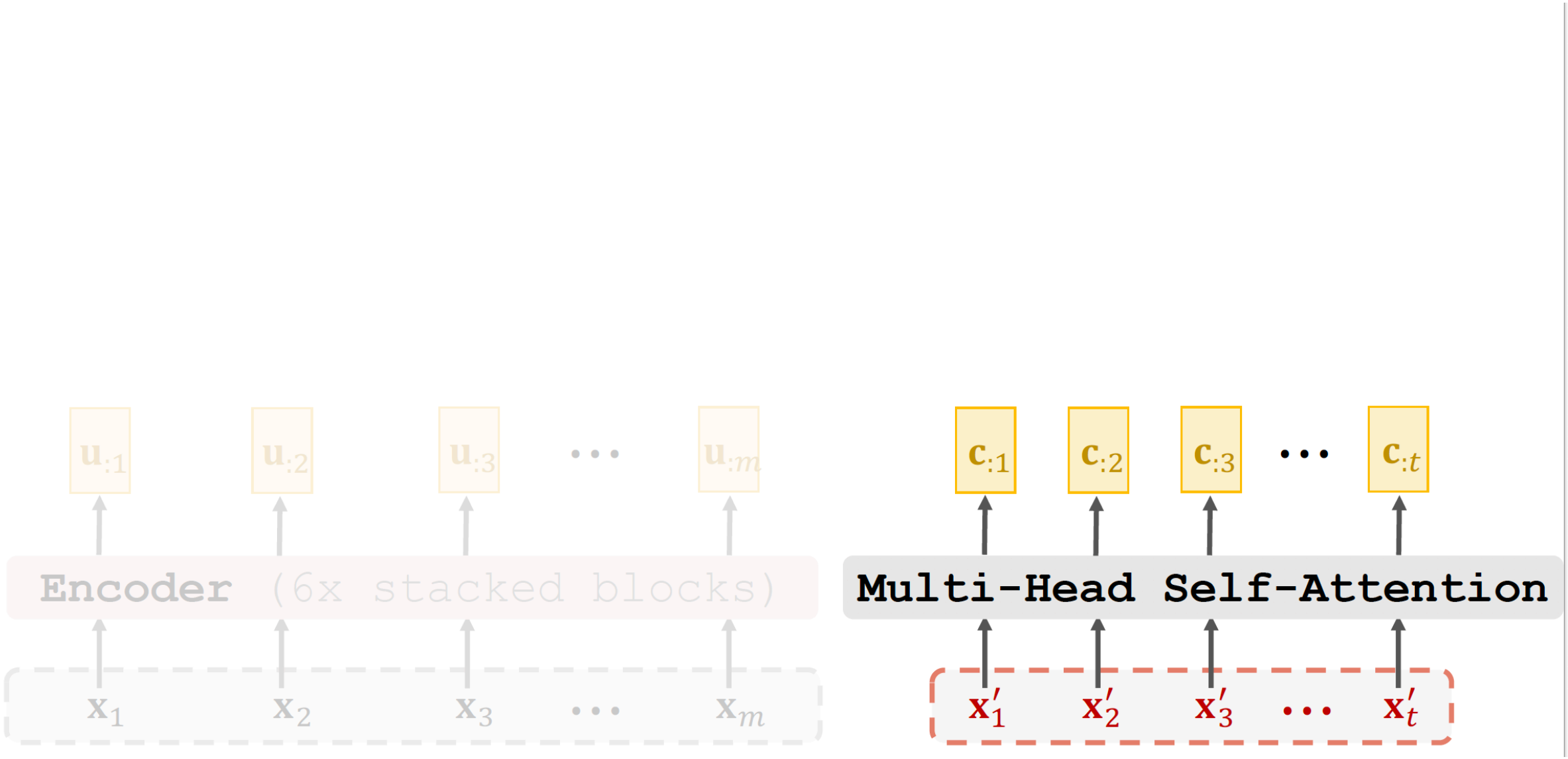
$\mathbf{x}'_1$ $\mathbf{x}'_2$ $\mathbf{x}'_3$ $\cdots$ $\mathbf{x}'_t$

Stacked Attentions

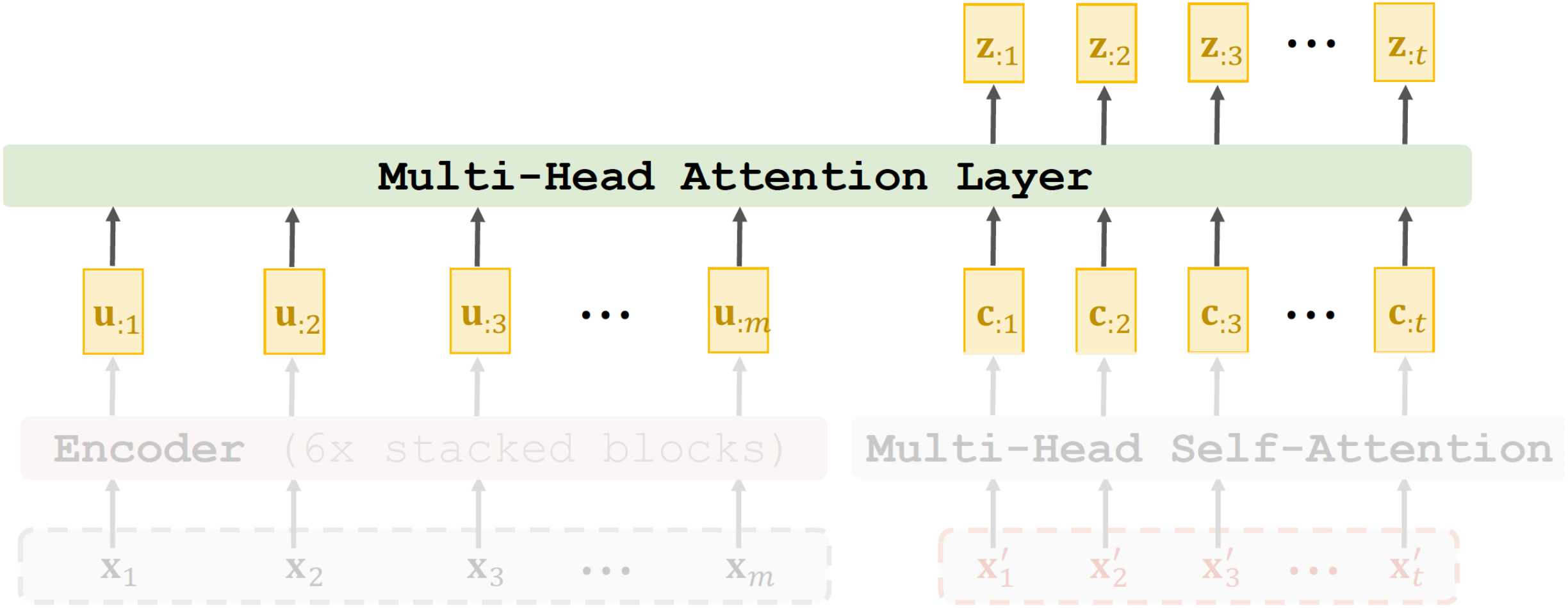
- Transformer's encoder contains 6 stacked blocks.
- 1 block ≈ 1 multi-head attention layer + 1 dense layer.



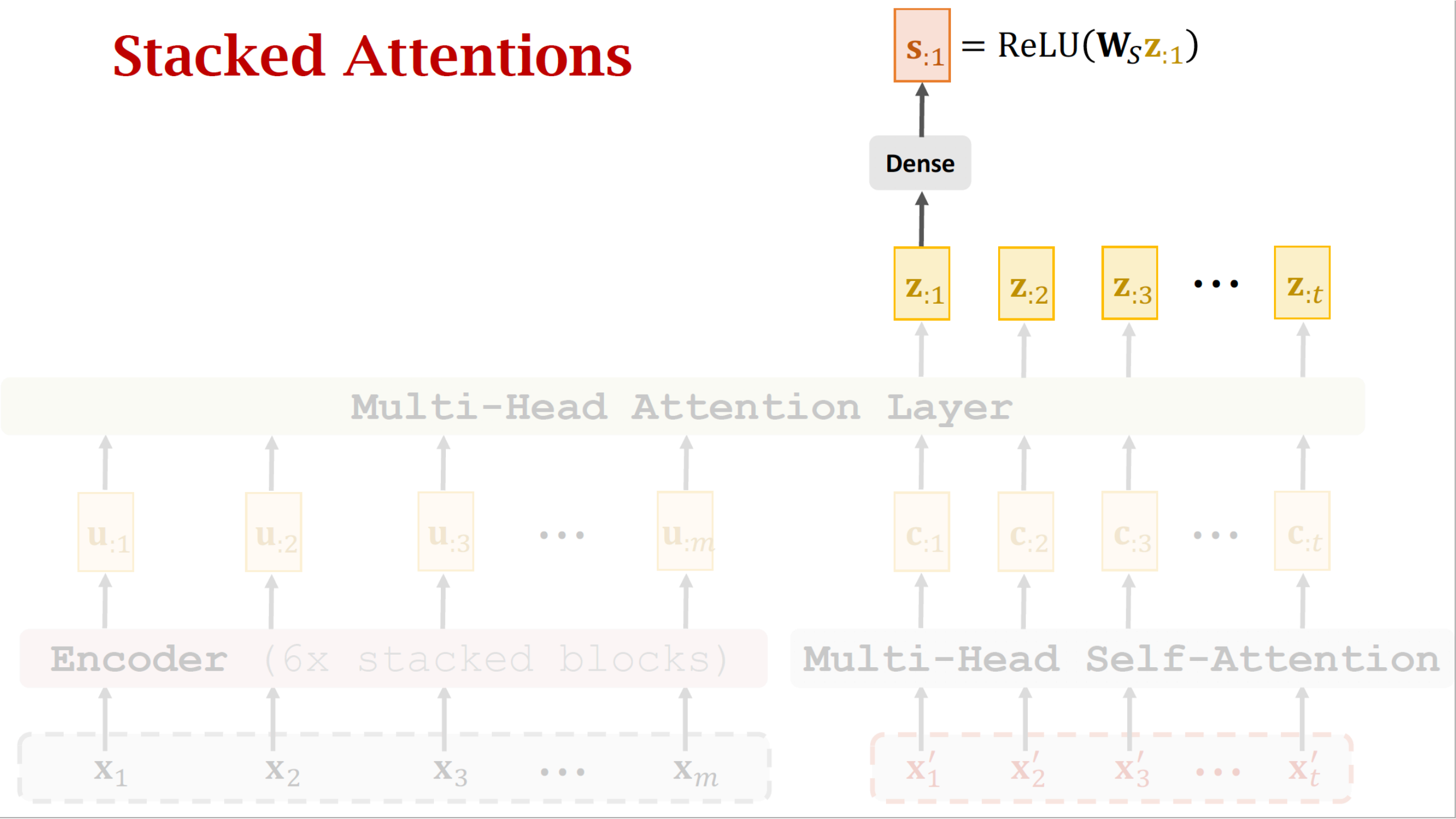
Stacked Attentions



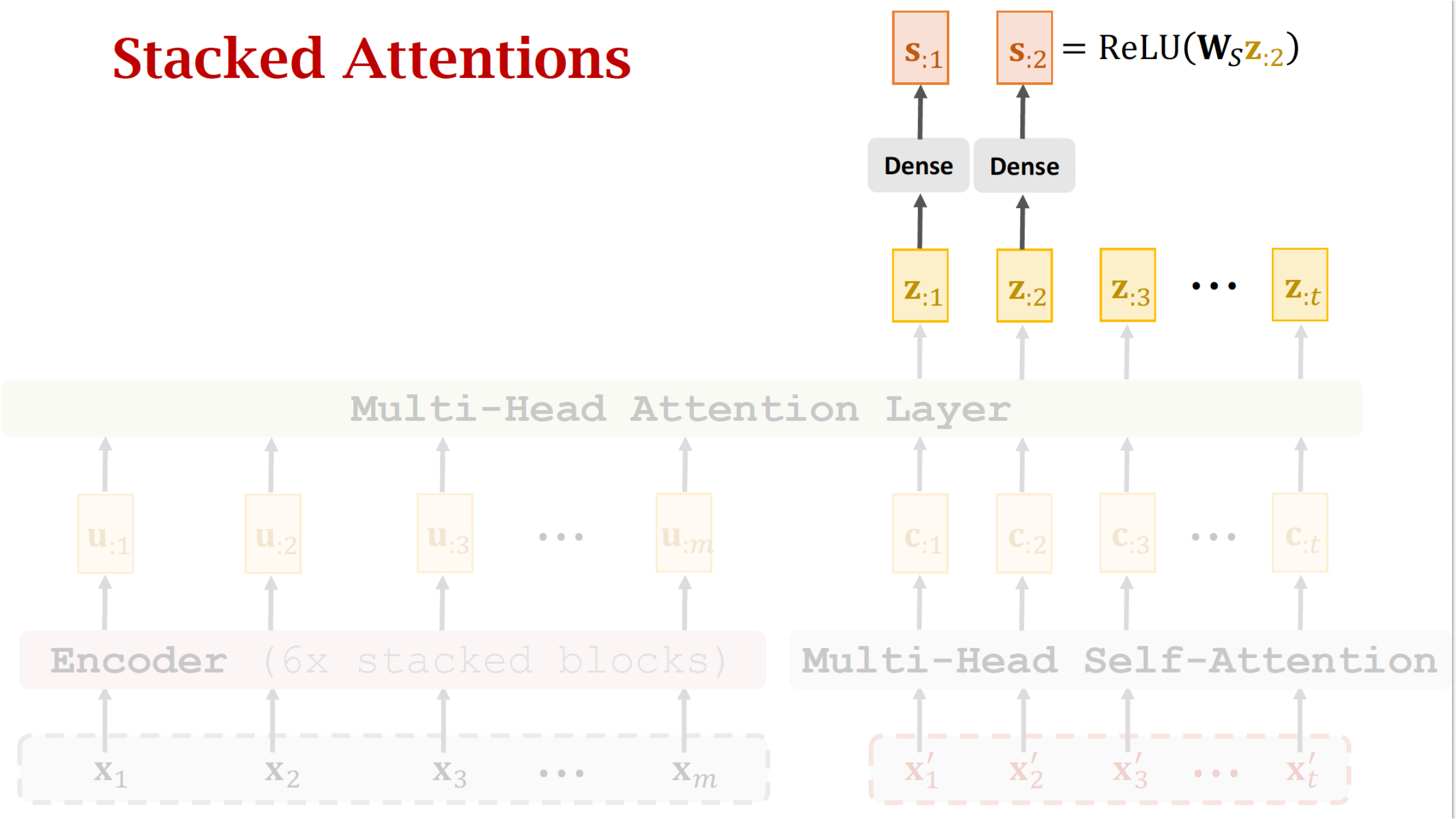
Stacked Attentions



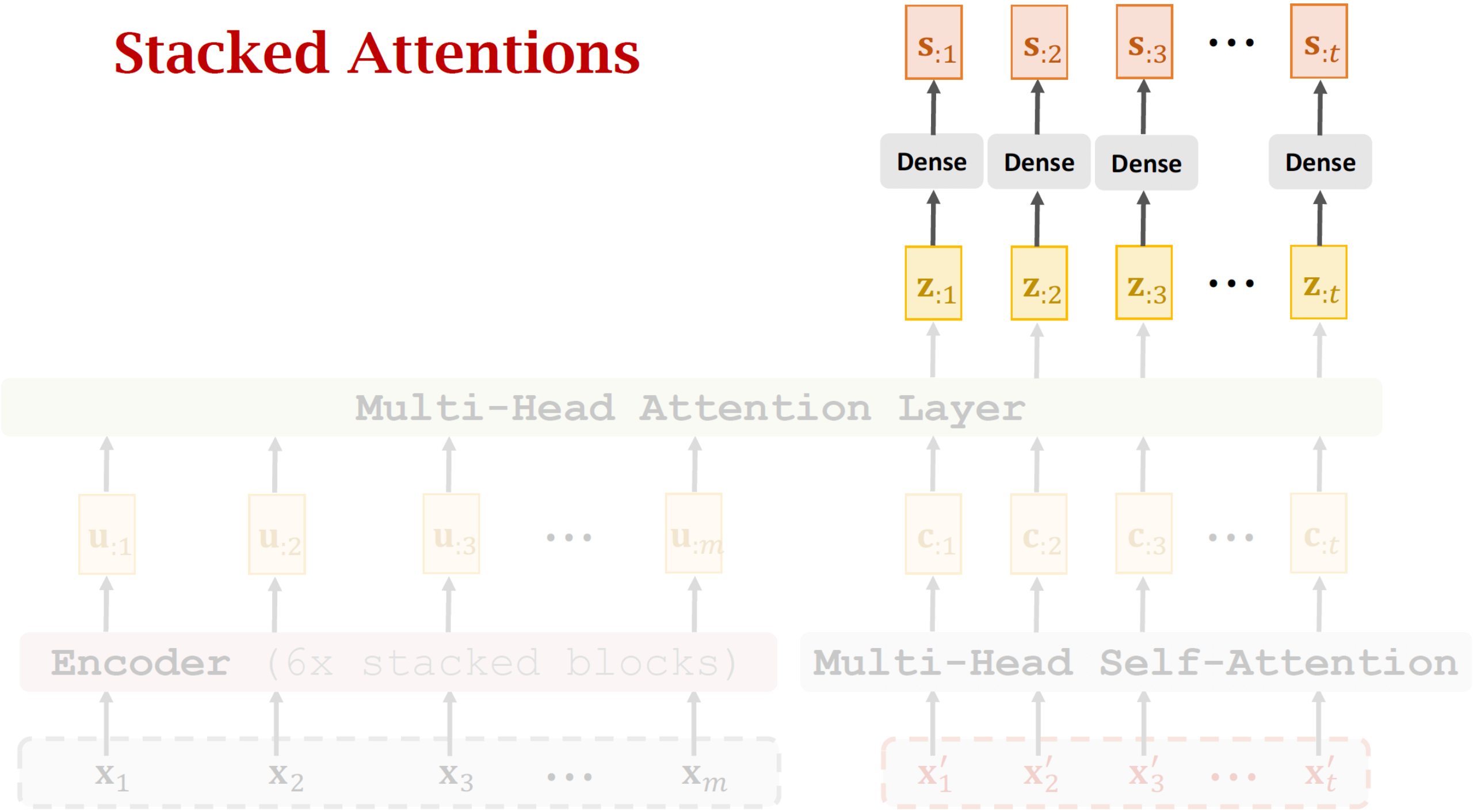
Stacked Attentions



Stacked Attentions

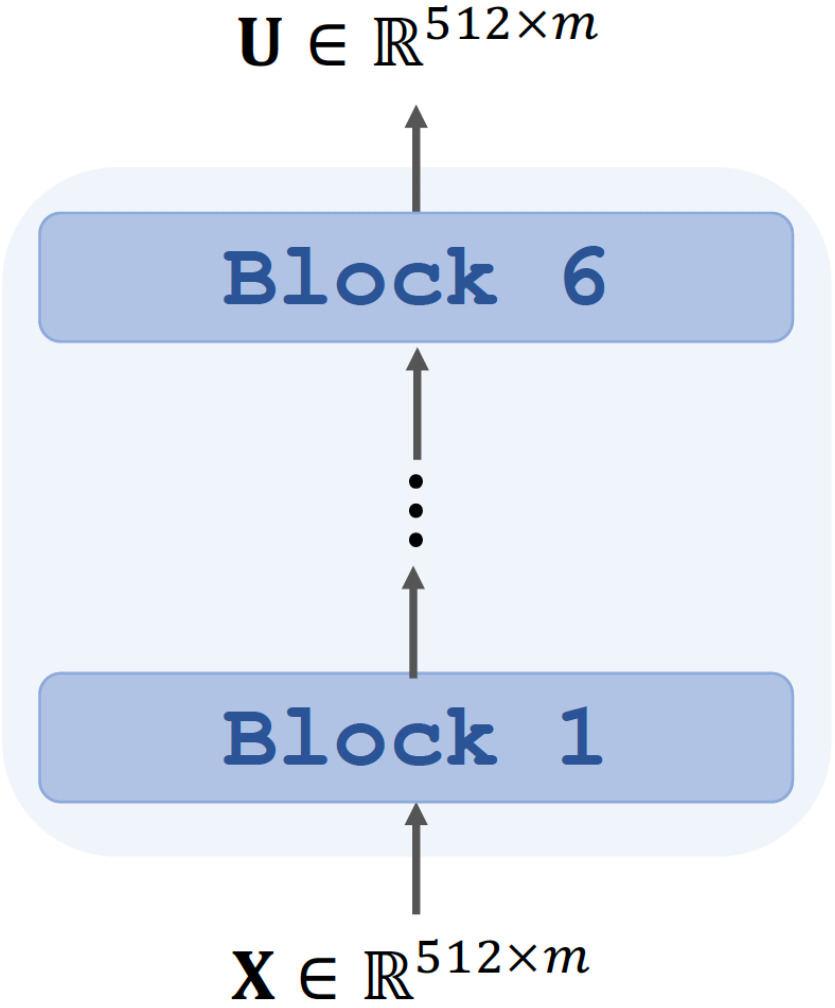


Stacked Attentions



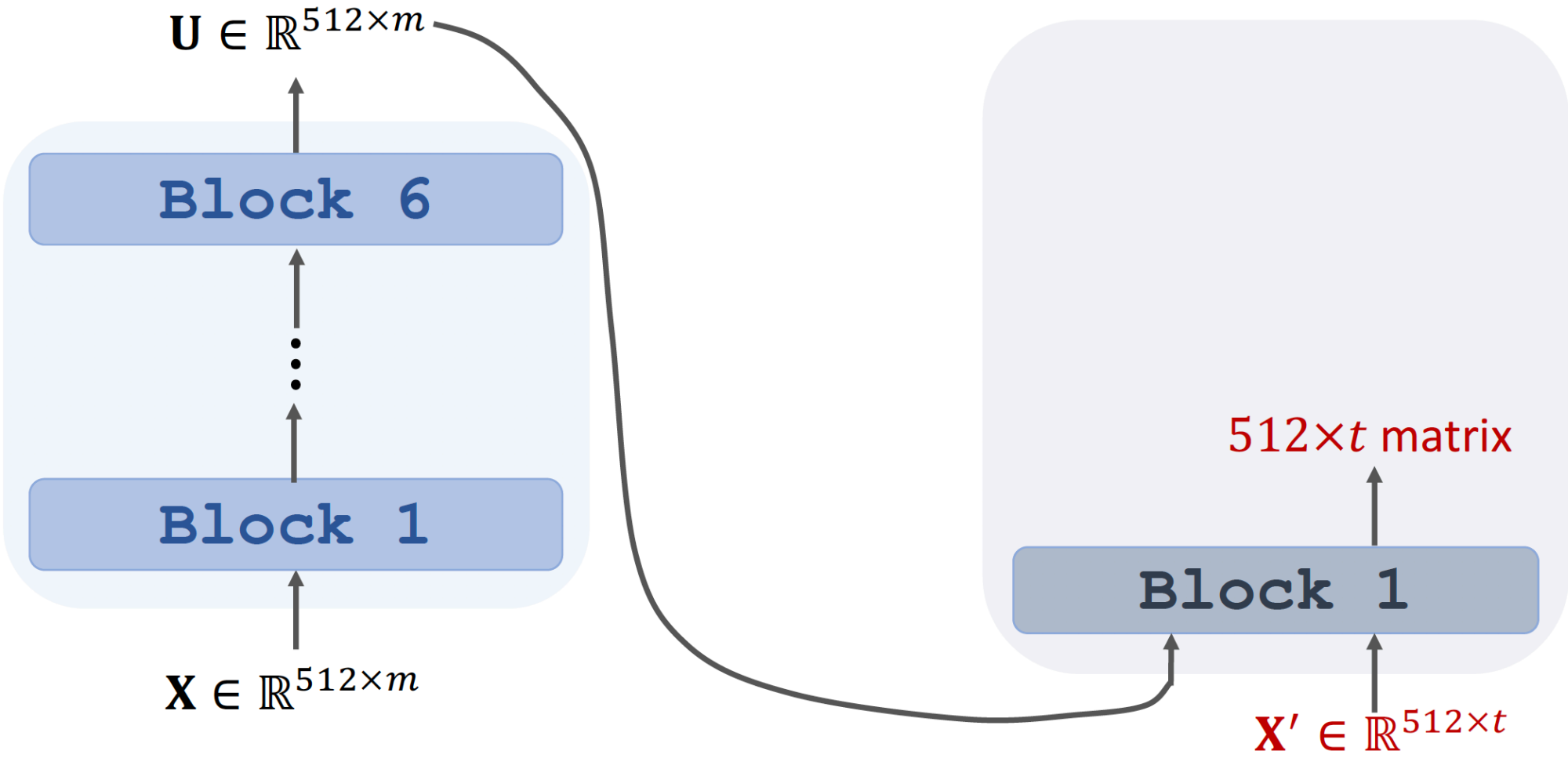
Put Everything Together

Transformer

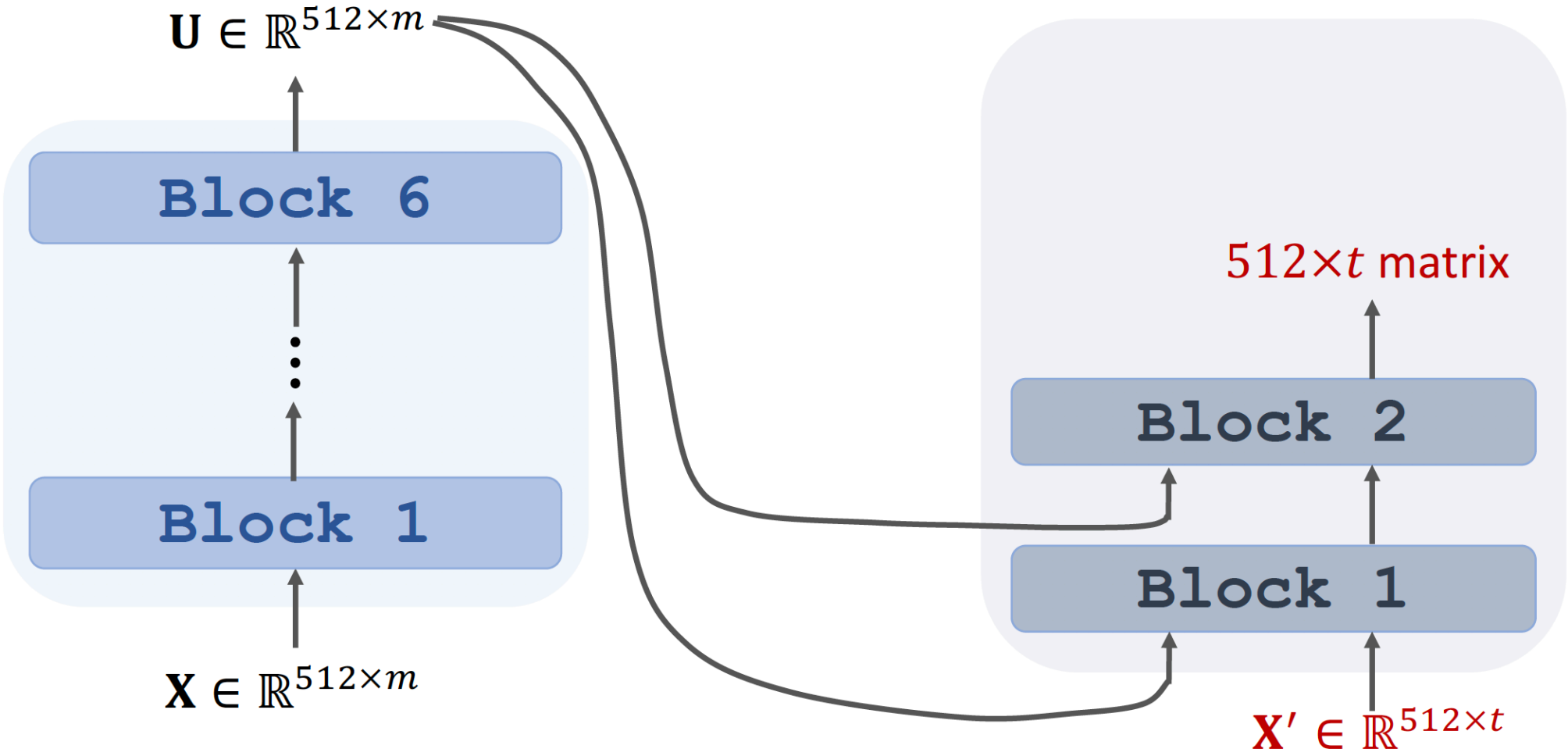


Encoder Network

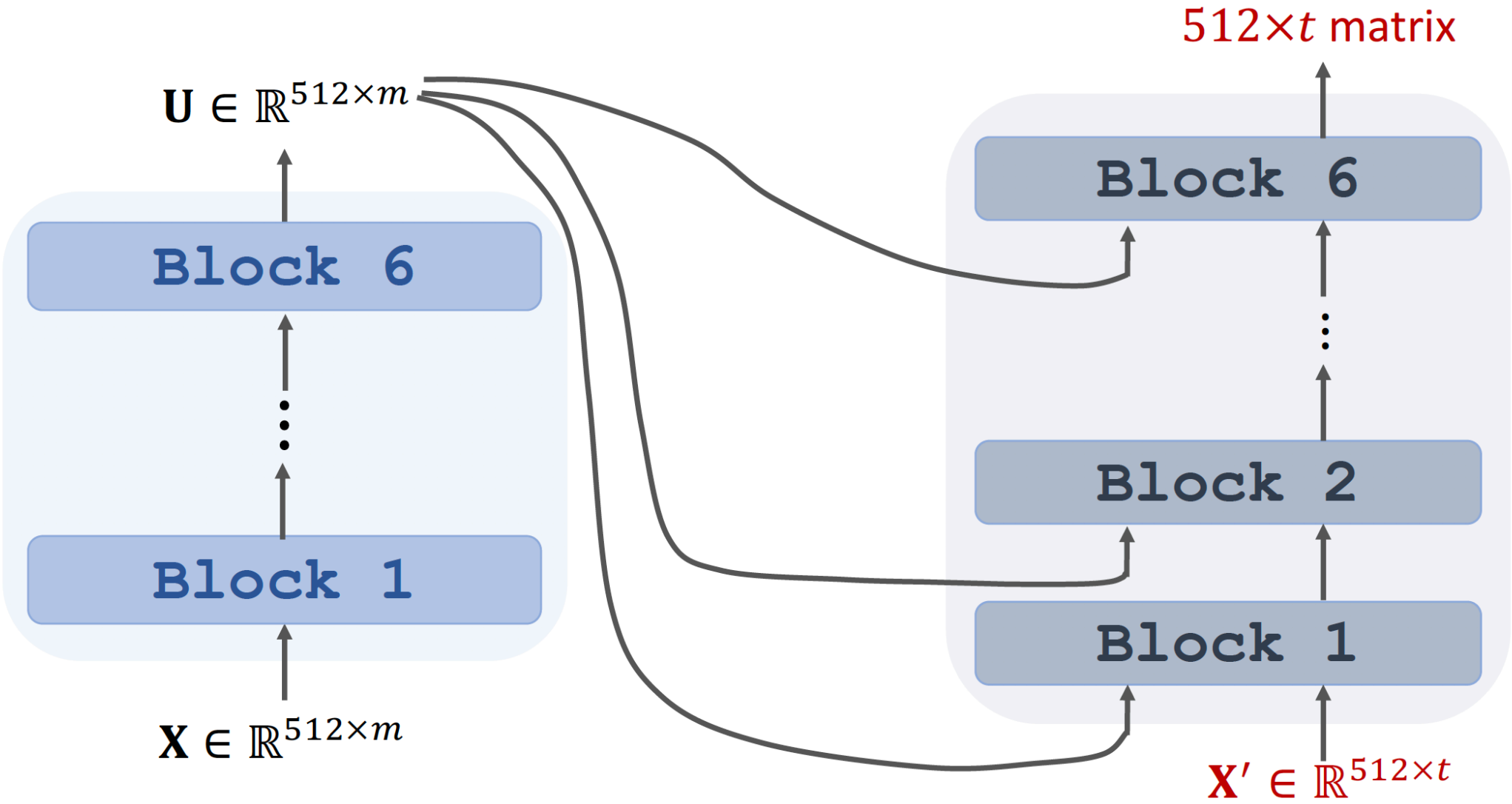
Transformer



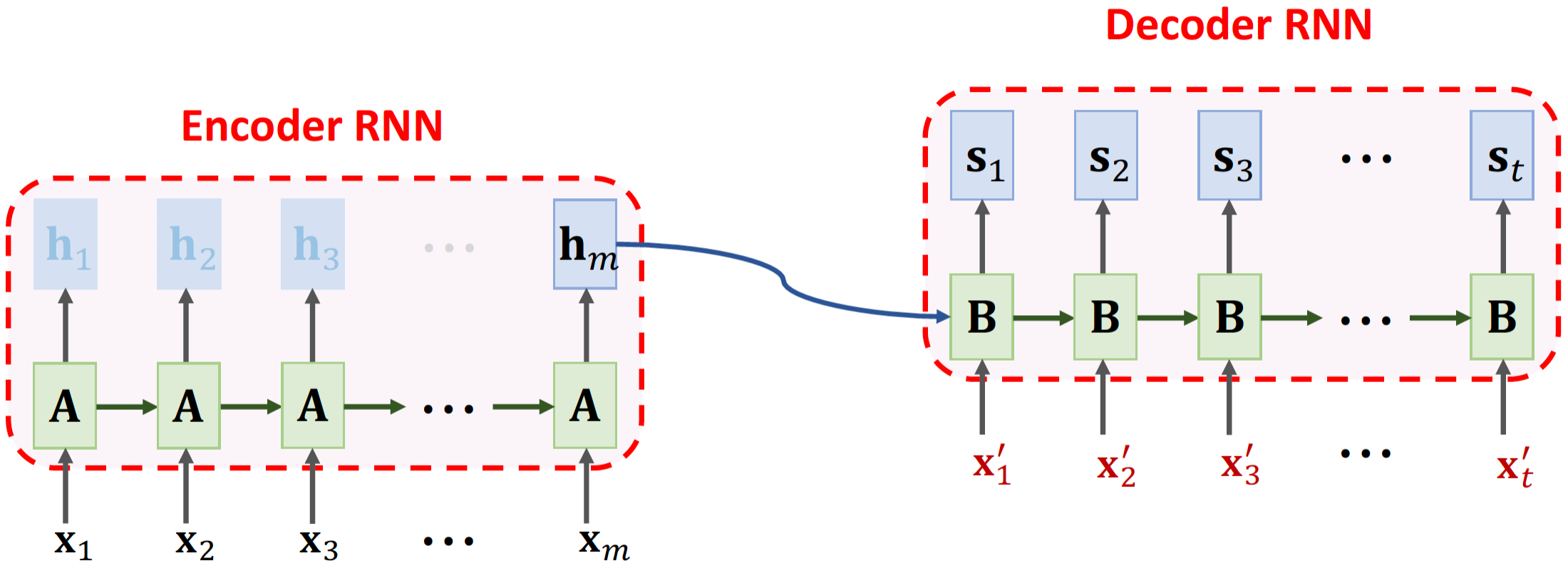
Transformer



Transformer



Comparison with RNN Seq2Seq Model



Any Question ?